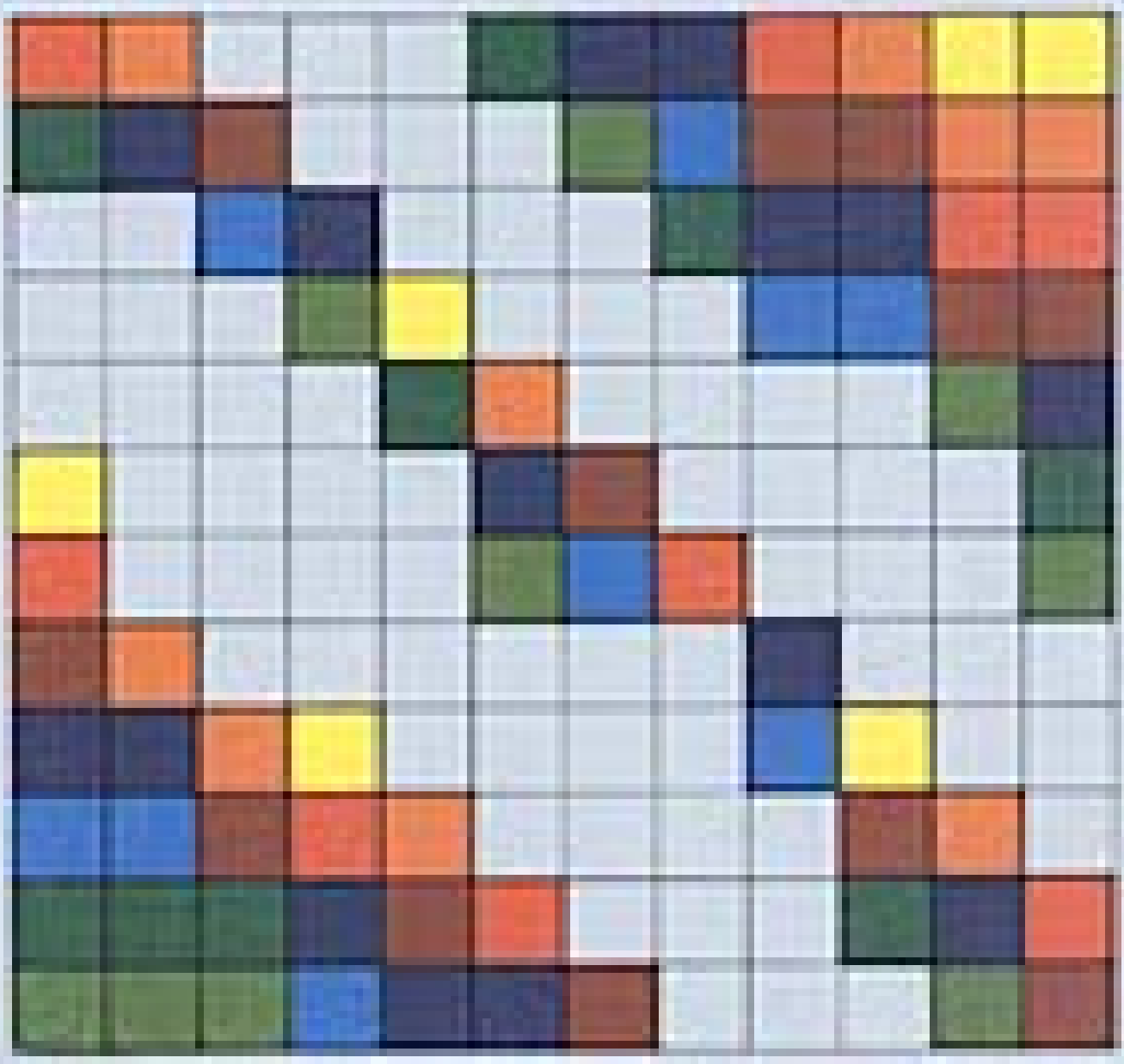
cláudio l.lucchesi
imre simon istvan simon
janos simon
tomasz kowaltowski

aspectos teóricos da computação



projeto

aspectos teóricos da computação

Aspectos teóricos da computação / Cláudio L. Lucchesi. . . [et al.]. – Rio de Janeiro: Instituto de Matemática Pura e Aplicada, 1979. 292 pp. (Projeto Euclides)

Bibliografia

1. Computação. 2. Grafos. 3. Autômatos. I. Lucchesi, Cláudio L., 1945– . II. Série.

CDD-519.4

cláudio l. lucchesi
imre simon
istvan simon
janos simon
tomasz
kowaltowski

aspectos teóricos da computação

impa
Instituto de Matemática Pura e Aplicada · CNPq

Impresso no Brasil / Printed in Brazil

Capa: Gian Calvi Criação Visual Ltda
Ladeira Ari Barroso 40, Leme, Rio de Janeiro, RJ

Projeto Euclides — Coordenado por Elon Lages Lima

Títulos já publicados

1. Curso de Análise, vol. 1, Elon Lages Lima
2. Medida e Integração, Pedro Jesus Fernández
3. Aplicações da Topologia à Análise, Chaim Samuel Hönig
4. Espaços Métricos, Elon Lages Lima
5. Análise de Fourier e Equações Diferenciais Parciais, Djairo Guedes de Figueiredo
6. Introdução aos Sistemas Dinâmicos, Jacob Palis Junior e Welington C. de Melo
7. Introdução à Álgebra, Adilson Gonçalves
8. Aspectos Teóricos da Computação, Cláudio L. Lucchesi, Imre Simon, Istvan Simon, Janos Simon e Tomasz Kowaltowski
9. Teoria Geométrica das Folheações, Alcides Lins Neto e Cesar Camacho
10. Geometria Riemanniana, Manfredo P. do Carmo
11. Lições de Equações Diferenciais Ordinárias, Jorge Sotomayor

Composição e arte:
AM Produções Gráficas Ltda.

Impresso por:
Gráfica Editora Hamburg Ltda.
Rua Apeninos, 294 - São Paulo - Brasil

Distribuido por:
Livros Técnicos e Científicos Editora S.A.
Avenida Venezuela, 163
20.220 - Rio de Janeiro, RJ - Brasil

Às nossas esposas,
Marlei, Gabriella, Karen, Betty e Doris

e aos nossos filhos,
Andrea, Claudio e Liliana,, Sandor e Alicia

CONTEÚDO

PREFÁCIO

O advento dos computadores eletrônicos na década de 1940 deu um ímpeto extraordinário à Computação, estimulando o interesse pelo estudo de seus fundamentos teóricos. O objetivo do presente texto é expor uma seleção dos tópicos abordados em Teoria da Computação, chegando a ilustrar alguns aspectos da pesquisa realizada nesta área.

Este volume consiste de quatro partes: Programação de Computadores, Complexidade de Algoritmos, Teoria dos Grafos e Teoria dos Autômatos Finitos, cada uma das quais representa uma área relativamente independente das demais. Embora cada uma das várias partes tenha suas características próprias, o conceito de algoritmo ocupa lugar de destaque em todas elas.

Na parte sobre Programação de Computadores discutimos o conceito de algoritmo, e a sua representação por meio de programas. Mostramos também como o princípio de indução matemática é utilizado nas várias fases da atividade de programação de computadores.

Na parte sobre Complexidade de Algoritmos, estes são examinados do ponto de vista de sua eficiência. O objetivo principal aqui é obter informações sobre a quantidade de recursos envolvidos na solução de problemas por meio de algoritmos.

Na parte sobre Teoria dos Grafos apresentamos alguns resultados básicos nesta área. Examinamos também vários problemas, chegando a sintetizar algoritmos para a sua solução.

Na parte sobre Teoria dos Autômatos Finitos adotamos um ponto de vista algébrico. Autômatos finitos podem ser encarados como representações de algoritmos que usam um espaço de trabalho limitado por uma constante. Como um subproduto da teoria, obtemos algoritmos para decidir se conjuntos dados possuem certas propriedades. Convém realçar que a priori, isto é, antes de desenvolver a teoria, nem sequer a existência de tais algoritmos pode ser garantida.

Nenhuma das quatro partes do livro tem profundidade suficiente para ser considerado um texto completo sobre o seu assunto. Mesmo assim, acreditamos que o livro possa ser útil tanto para estudantes

de matemática como de computação. Para os matemáticos, o texto pode servir como uma introdução à Teoria da Computação, permitindo inclusive uma compreensão de alguns problemas abertos desta área relativamente recente. Por outro lado, o livro reflete os conhecimentos de Teoria da Computação com que, na nossa opinião, todo estudante sério de computação deve estar familiarizado.

O livro contém aproximadamente 400 exercícios de diversos graus de dificuldade. Queremos realçar a importância de resolver um bom número destes, tanto para fixar as idéias expostas no texto, como para possibilitar que o estudante adquira uma perspectiva mais ampla sobre os assuntos tratados.

No final de cada capítulo as Notas Bibliográficas traçam um breve histórico do assunto tratado e indicam os autores dos resultados daquele capítulo. São sugeridas também leituras suplementares, indicando, quando for o caso, os textos principais nas respectivas áreas.

As quatro partes do livro são identificadas pelas letras A, B, C e D; em cada parte, os capítulos são identificados por números romanos; em cada capítulo as seções, teoremas, proposições, lemas, exemplos, figuras e exercícios são numerados com algarismos decimais. Quanto a referências no meio do texto, o Teorema 2 do Capítulo V da Parte C é citado como Teorema 2 naquele capítulo, como Teorema V.2 nos demais capítulos da Parte C e como Teorema C.V.2 nas demais partes do livro.

O texto resultou da colaboração de cinco autores e, inevitavelmente, apresenta diferentes estilos nos vários capítulos.

O presente volume é uma versão revista e aumentada de um texto escrito para o 11.º Colóquio Brasileiro de Matemática. Gostaríamos de registrar aqui os nossos agradecimentos pela oportunidade a nós concedida tanto pela Comissão Organizadora daquele Colóquio, como pela Comissão Editorial do Projeto Euclides, bem como o nosso reconhecimento pelo incentivo constante que recebemos do Professor Chaim Samuel Hönig durante todas as fases da realização deste trabalho.

São Paulo, janeiro de 1979.

Os autores

PARTE A

PROGRAMAÇÃO DE COMPUTADORES

CAPÍTULO I

PROGRAMAÇÃO DE COMPUTADORES E INDUÇÃO MATEMÁTICA

1. Introdução

A programação de computadores é uma atividade que pode ser vista de várias maneiras. Por um lado, ela é praticada hoje em dia em escala industrial, e os problemas encontrados na confecção de grandes sistemas de programação – como sistemas operacionais ou de reservas de passagens aéreas – são comparáveis aos problemas encontrados em grandes projetos de engenharia. Por outro lado, a programação ainda é uma atividade quase que artesanal e a metodologia para desenvolvimento de programas de grande porte, e que sejam confiáveis e eficientes, é muito rudimentar. Este estado de coisas deve-se, em parte, ao fato da programação de computadores ser uma atividade muito recente, e que sofre transformações rápidas resultantes da expansão do seu campo de aplicação, e dos progressos tecnológicos na construção dos próprios computadores. Uma boa parte da pesquisa nesta área está dedicada à identificação dos princípios básicos que podem ser aplicados à programação, e do seu uso sistemático. Este tipo de pesquisa teve uma influência muito grande sobre projetos de novas linguagens de programação, e sobre técnicas de programação utilizadas.

O objetivo deste capítulo é apresentar alguns dos princípios em que se baseia a programação de computadores. Em primeiro lugar, é discutido na Seção 2 o conceito de procedimento efetivo que deve corresponder à noção intuitiva daquilo que pode ser calculado por meios mecânicos. Na Seção 3 apresentamos o conceito de linguagem de programação e o de programa, que é uma representação formal de procedimento. Na Seção 4, discutimos a relação entre os mecanismos de repetição existentes nos programas e o princípio de indução matemática. Finalmente, na Seção 5, apresentamos vários exemplos que ilustram o uso do princípio de indução para desenvolver, analisar e demonstrar propriedades de programas.

2. Procedimentos e algoritmos

Um *procedimento* é uma seqüência finita de instruções que podem ser executadas por um agente computacional, seja ele humano ou não. Este conceito corresponde, portanto, às noções intuitivas de "receita", "roteiro", "método", etc. Um exemplo clássico de procedimento é o chamado "Algoritmo de Euclides" que especifica como calcular o máximo divisor comum (*mdc*) de dois inteiros positivos m e n (o significado da palavra "algoritmo" será explicado mais adiante — por enquanto ele faz parte do nome do procedimento):

Passo 1: Adote como valores iniciais de x e y os valores m e n, respectivamente.

Passo 2: Adote como valor de r o resto da divisão do valor de x pelo valor de y.

Passo 3: Adote como o novo valor de x o valor de y, e como novo valor de y o valor de r.

Passo 4: Se o valor de r é nulo, então o valor de x é o *mdc* procurado, e o cálculo termina; caso contrário volte a executar as instruções do procedimento a partir do passo 2.

Este exemplo ilustra algumas propriedades que vamos exigir de um procedimento:

(i) A descrição do procedimento deve ser finita. No exemplo citado utilizamos uma seqüência finita de palavras e símbolos para descrever o procedimento.

(ii) Todo procedimento parte de um certo número de dados pertencentes a conjuntos especificados de objetos (como m e n que são inteiros positivos), e espera-se que produza um certo número de resultados (como o valor final de x) que mantêm uma relação específica com os dados.

(iii) Supõe-se que exista um agente computacional — humano, mecânico, eletrônico, etc. — que execute as instruções do procedimento. Este agente deverá ter uma maneira de guardar e recuperar informações durante a execução do procedimento. No exemplo anterior, estas informações seriam constituídas pelos valores de x, y e r, bem como a indicação do próximo passo a ser executado.

(iv) Cada instrução especificada pelo procedimento deve estar bem definida, não deixando dúvidas quanto ao seu resultado. Assim, no exemplo acima, supõe-se que dados os valores inteiros positivos de x e y, o agente computacional sabe calcular o resto da divisão de x

por y. A mesma instrução não estaria bem definida se x e y pudessem ter valores inteiros quaisquer: quais são os restos da divisão de 10 por 0 ou de -20 por -7? Evidentemente poderíamos estender de maneira conveniente a definição do resto da divisão para inteiros negativos, e neste caso o resultado para -20 e -7 estaria bem definido.

(v) As instruções do procedimento devem ser efetivas, isto é, devem ser tão simples que poderiam ser executadas, em princípio, por uma pessoa usando lápis e papel, num espaço de tempo finito. As instruções do Algoritmo de Euclides satisfazem este critério quando os valores envolvidos são inteiros positivos. Entretanto, elas deixariam de ser efetivas se os valores de x e de y pudessem ser números reais quaisquer em representação decimal, possivelmente de comprimento infinito. Um outro exemplo de instrução não efetiva seria: "adote para s o valor zero se existir um inteiro k maior do que 2, e inteiros positivos x, y e z tais que $x^k + y^k = z^k$". Para poder executar esta instrução, teríamos que saber se o chamado Último Teorema de Fermat, que afirma que tais inteiros não existem, é verdadeiro ou não. Este problema está em aberto desde que foi proposto no século XVII.

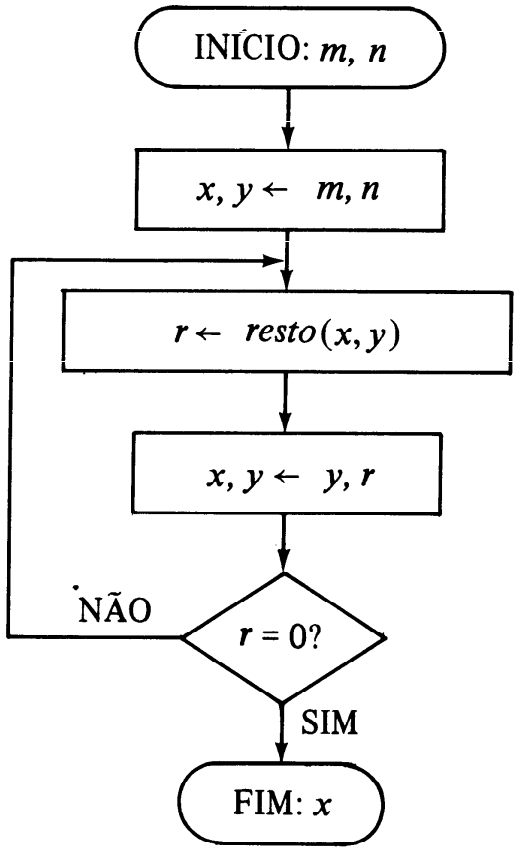


Figura 1

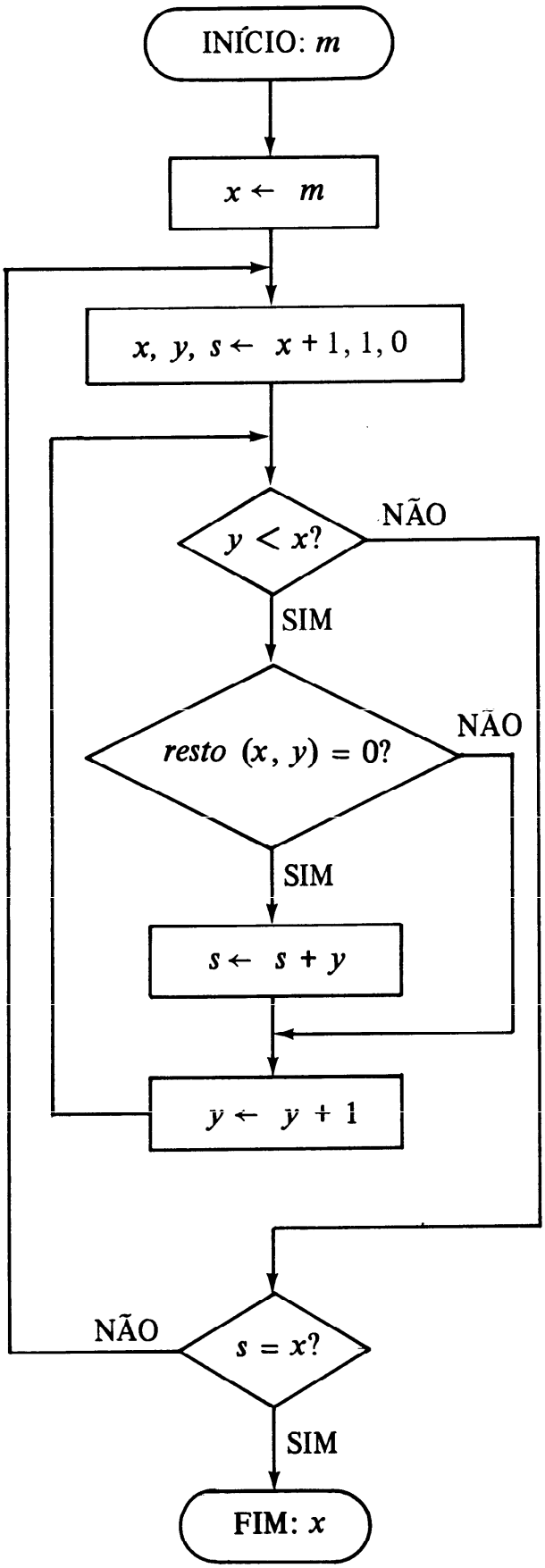

INÍCIO: m
x ← m
x, y, s ← x + 1, 1, 0
y < x?
NÃO
SIM
resto (x, y) = 0?
NÃO
SIM
s ← s + y
y ← y + 1
s = x?
NÃO
SIM
FIM: x

Figura 2

Note-se que não chegamos a definir precisamente o conceito de procedimento, e nem o faremos. Este é um conceito primitivo, correspondente à abstração das noções exemplificadas acima, independentemente da sua representação. Um procedimento que satisfaz as propriedades enumeradas acima é chamado também *procedimento efetivo*. Continuaremos, entretanto, a utilizar o termo procedimento para designar o mesmo conceito.

A descrição de um procedimento pode assumir várias formas distintas, e mais ou menos formalizadas. Uma maneira comum de se representar um procedimento é através do chamado *diagrama de blocos*. O Algoritmo de Euclides poderia ser representado pelo diagrama da Figura 1. O significado do diagrama deve ficar óbvio ao compará-lo com a descrição do procedimento dada anteriormente.

Um outro exemplo de procedimento está representado na Figura 2. A sua função é determinar o menor número perfeito maior do que um inteiro positivo m dado (um número k é perfeito se for igual à soma de todos os seus divisores exceto o próprio k). Mais um exemplo de procedimento está indicado na Figura 3.

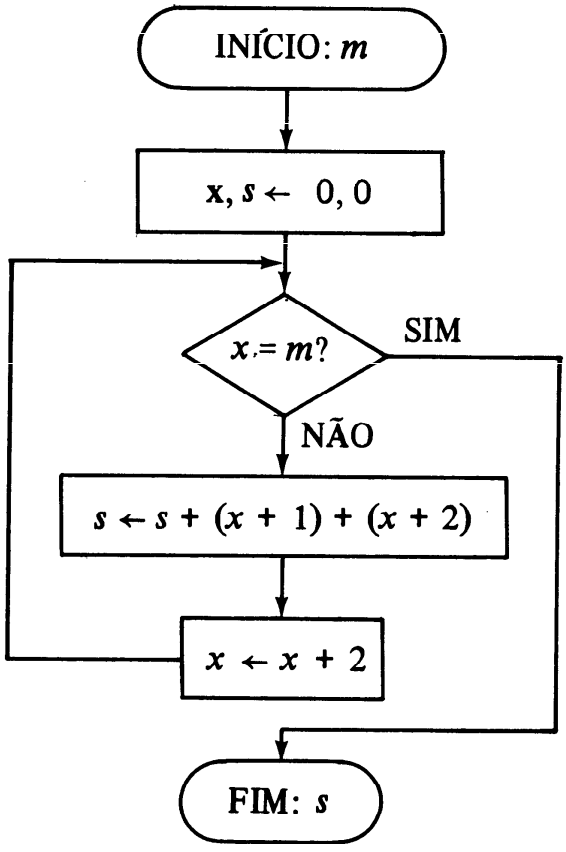


Figura 3

Uma pergunta que surge naturalmente é se um procedimento, partindo de certos dados iniciais, executa uma seqüência finita de cálculos, produzindo resultados finais, ou se então essa seqüência de cálculos nunca termina. No caso do Algoritmo de Euclides podemos mostrar que a seqüência de cálculos é finita provando a seguinte proposição: se no passo 2 do procedimento os valores de x e y são inteiros e positivos, então os passos 2, 3 e 4 serão executados apenas um número finito de vezes, com os cálculos terminando no passo 4. A demonstração é por indução sobre o valor de y. Se $y = 1$, então teremos, pela execução do passo 2, $r = 0$. Conseqüentemente, os passos 2, 3 e 4 são executados uma única vez, e os cálculos terminam no passo 4. Suponhamos agora que a proposição é verdadeira para qualquer $x > 0$ e qualquer y, com $1 \leq y < k$, e demonstraremos que ela é verdadeira para $y = k$. Por definição do resto da divisão de inteiros positivos, teremos, após a execução do passo 2, $0 \leq r < k$. Se $r = 0$, então a execução termina, como anteriormente, numa única vez. Se $r > 0$, então, com a execução dos passos 3 e 4, teremos $x = k > 0$ e $y = r$ com $0 < r < k$, e a execução volta ao passo 2. Por hipótese de indução, os passos 2, 3 e 4 serão executados um número finito p de vezes, com os cálculos terminando no passo 4. Ao todo teremos, então, $p + 1$ execuções para $y = k$. Notemos ainda que os valores iniciais $x=m$ e $y=n$ resultantes da execução do passo 1 satisfazem as condições da proposição acima. Podemos concluir, portanto, que a execução do Algoritmo de Euclides termina para quaisquer inteiros positivos m e n.

No caso do segundo procedimento, sabemos que o cálculo termina para certos valores de m. Por exemplo, se $m = 4$ então obteremos o resultado $x = 6$ pois $5 \neq 1$ e $6 = 1 + 2 + 3$. Entretanto, no caso geral a resposta não é conhecida, pois a existência ou não de um número infinito de números perfeitos é um problema em aberto. Se existirem infinitos números perfeitos, então a execução do procedimento termina para qualquer m; caso contrário, se K é o maior número perfeito, então o procedimento executa uma seqüência infinita de cálculos para todo $m \geq K$. Finalmente, no caso do terceiro exemplo podemos ver que a execução do procedimento termina com $s = \sum_{0}^{m} i$ para valores pares de m, pois os valores consecutivos de x são 0, 2, 4, 6, ... Para valores ímpares de m, a igualdade $x = m$ nunca será satisfeita, e a execução do procedimento não termina.

Entre todos os procedimentos, teremos um interesse especial naqueles cuja execução termina para quaisquer valores dos dados. Estes procedimentos serão chamados *algoritmos*(1).

Em conseqüência da discussão anterior, conclui-se que o procedimento que chamamos Algoritmo de Euclides é realmente um algoritmo. Quanto ao segundo procedimento, a resposta não é conhecida, e o terceiro exemplo representa um procedimento que não é um algoritmo.

Neste ponto fica claro que o problema de decidir se um dado procedimento é ou não um algoritmo deve ser difícil. Caso contrário, já saberíamos as respostas sobre várias conjeturas, tais como a existência de um número infinito de números perfeitos, a veracidade do Último Teorema de Fermat, e outros. Este problema será discutido de maneira mais completa no Capítulo B.II.

Note-se que todo procedimento é um método para o cálculo de alguma função, eventualmente não definida para certos argumentos. Assim, o terceiro diagrama de blocos corresponde à função h que pode ser descrita por:

$$h(m) = \begin{cases} \sum_{0}^{m} i & \text{se } m \text{ é par} \\ \text{não definida} & \text{se } m \text{ é ímpar.} \end{cases}$$

Por outro lado, uma mesma função pode ser calculada por vários procedimentos distintos. O procedimento da Figura 4 calcula a mesma função h definida acima.

3. Programas e linguagens de programação

Vimos na seção anterior que um procedimento pode ser especificado por uma mistura de palavras e símbolos, como foi feito com a primeira versão do Algoritmo de Euclides. Esta maneira é, em geral, bastante adequada quando se trata de descrever procedimentos que serão interpretados por pessoas. Entretanto, para que um procedimento possa ser executado por uma máquina, é necessário que a sua descrição seja feita numa linguagem que não tenha as imprecisões nem a varia-

(1) O termo "algoritmo" vem do nome Muhammad ibn-Musa al-Khwarizmi, autor do famoso texto matemático *Kitab al-jabr wa al-muqabalah* escrito em árabe no Século IX, e que deu origem ao termo "álgebra". Khwarezm, a cidade de origem do autor, é hoje a pequena cidade de Khiva na república soviética de Uzbequistão.

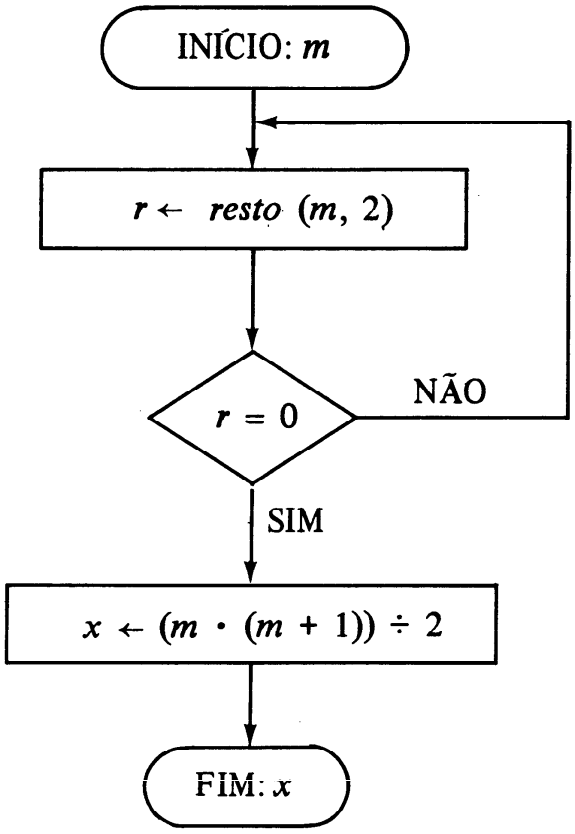


Figura 4

bilidade de uma língua natural. Uma outra vantagem da formalização é que ela permite maior rigor nas definições e demonstrações sobre procedimentos.

O fato de que a maneira mais conveniente de transmitir informação aos computadores modernos é através de seqüências de caracteres exclui, em geral, o uso de diagramas de blocos, se bem que estes podem ser definidos de maneira rigorosa. O método mais comum, portanto, para se especificar um procedimento de maneira formal é através de uma *linguagem de programação*.

Uma linguagem de programação é definida por um conjunto de símbolos, chamado *alfabeto*, que podem ser usados na representação de procedimentos, e por um conjunto de regras que especificam como compor estas representações e quais são as ações associadas a estas representações. Uma seqüência de símbolos de uma linguagem de programação que representa um ou mais procedimentos será chamada *programa*.

As várias linguagens de programação têm características bastante diferentes, conforme a sua finalidade. Existem linguagens muito simples, que incluem um número muito pequeno de operações primitivas, mas

que têm um grande interesse para a teoria da computação, como por exemplo a linguagem de Turing, a ser vista no Capítulo B.I. Tais linguagens dificilmente são usadas para programar procedimentos utilizados na prática. As chamadas *linguagens de máquina*, que são "compreendidas" diretamente pelos computadores, variam bastante de um computador ao outro, e a programação nestas linguagens pode ser muito trabalhosa. A tendência moderna é a utilização das chamadas *linguagens de alto nível*, mais adequadas para a representação de procedimentos. Um programa escrito em linguagem de alto nível é traduzido para a linguagem de máquina de um dado computador, para que possa ser executado. Essa tradução, por sua vez, é feita pelo próprio computador, executando um programa especial chamado *compilador*. Conseqüentemente, tudo se passa como se o computador "compreendesse" a linguagem de alto nível.

É importante saber, em geral, se numa dada linguagem de programação podem-se representar todos os procedimentos efetivos, isto é, se a linguagem é universal. Uma discussão pormenorizada deste problema está fora do alcance deste texto, mas devemos considerar os seguintes fatos. Em primeiro lugar, é aceita universalmente a chamada *Tese de Church*, segundo a qual qualquer procedimento pode ser representado em linguagem de Turing. Evidentemente, esta tese não pode ser demonstrada, pois não temos uma definição de procedimento. Entretanto, o volume de evidência empírica, que abrange todos os procedimentos já formulados, faz com que se aceite esta tese. Em segundo lugar, pode-se provar que os programas escritos em linguagem de Turing podem ser "traduzidos" em programas equivalentes em outras linguagens de programação, contanto que estas contenham certos conjuntos mínimos de operações primitivas, como por exemplo soma, subtração, teste de zero, e um mecanismo para indicar cálculos repetitivos. Em geral, as linguagens de programação incluem um número de operações maior do que o citado acima, tornando o trabalho de programação mais conveniente, e resultando numa execução mais rápida de certas operações que poderiam ser programadas através das operações básicas. A Parte B do texto inclui uma discussão deste aspecto de linguagens.

Neste texto utilizaremos uma linguagem de programação bastante simples, mas que se assemelha às linguagens de alto nível usadas na prática, e em particular à linguagem chamada Algol 60. Antes de descrever esta linguagem, que denominaremos *LP*, mostramos nas Figuras

5 e 6 programas que representam os procedimentos das Figuras 1 e 2, respectivamente.

Um programa em *LP* é constituído por uma seqüência de símbolos, sendo que o espacejamento entre os mesmos, as mudanças de linha e, em geral, a disposição física são irrelevantes, e têm por finalidade

```
procedimento Euclides(m, n):
    início
        x, y ← m, n;
        repita
            r ← resto(x, y);
            x, y ← y, r
        até que r = 0;
        devolva x
    fim
```

Figura 5

```
procedimento número perfeito(m):
    início
        x ← m;
        repita
            x, y, s ← x + 1, 1, 0;
            enquanto y < x faça
                início
                    se resto(x, y) = 0 então s ← s + y
                                       senão nada;
                    y ← y + 1
                fim;
        até que s = x;
        devolva x
    fim
```

Figura 6

aumentar a legibilidade e compreensão pelo leitor humano. A estrutura de um programa é indicada pelos símbolos de pontuação tais como vírgula, dois-pontos, ponto-e-vírgula, seta e outros, bem como por símbolos compostos de letras impressas em negrito. Neste texto, estes símbolos são palavras como **início, fim, se, então, senão** e outras, e o seu significado ajuda na compreensão dos programas. Os nomes dos programas (como *Euclides* e *número perfeito*), dos argumentos (como

m e n) e das variáveis (como x, y e s) podem ser escolhidos de maneira arbitrária.

Passaremos a descrever agora de maneira mais precisa, se bem que informal, a linguagem *LP*. Uma indicação de como tal definição poderia ser formalizada será dada no final desta seção.

Em primeiro lugar, deveríamos especificar os objetos que podem ser manipulados nesta linguagem e quais as operações primitivas disponíveis para manipulá-los. Como já foi mencionado, bastaria adotar os números naturais e um mínimo de operações primitivas para conseguir representar, segundo a Tese de Church, qualquer procedimento. Entretanto, esta decisão não permitiria a manipulação direta de outros objetos tais como caracteres e símbolos, seqüências de números, vetores, matrizes, etc. Todos estes objetos teriam que ser representados por meio de uma codificação conveniente através dos números naturais. Assim, os dados para os programas teriam que ser codificados, os resultados teriam que ser decodificados, e ao programar teríamos que estar sempre atentos ao fato de utilizarmos números para representar objetos de outra natureza. Por conveniência, incluímos então em *LP* a manipulação de vários tipos de objetos, além dos números naturais. A especificação precisa destes tipos de objetos e das operações primitivas correspondentes será feita ao apresentarmos os exemplos particulares de programas que os utilizam. Suporemos apenas que existe uma maneira de denotar as constantes como 0 (inteiro zero), $\langle 2, 7, 10 \rangle$ (seqüência dos três inteiros 2, 7 e 10), "*aspectos teóricos*" (cadeia de caracteres), etc. As *expressões* da linguagem serão formadas utilizando-se estas constantes, nomes de argumentos e variáveis, operações primitivas, ou então nomes de procedimentos representados no programa. Neste último caso, devem ser calculados os valores dos argumentos, e executado o procedimento correspondente, que deverá devolver os valores calculados.

Um programa em *LP* será constituído da representação de um procedimento, seguido eventualmente da representação dos procedimentos auxiliares.

A representação de um procedimento tem a forma geral:

procedimento $f(m_1, m_2, \ldots, m_n)$: S

com $n \geq 0$, onde f será o nome do procedimento, os m_i são os nomes dos argumentos (dados), e S é um *comando* que indica como devem ser calculados os resultados da execução de f, para os valores $m_1, \ldots, m_n$ dos dados.

Os comandos em *LP* podem assumir várias formas. O *comando de atribuição de valor* tem a forma

$$x_1, \ldots, x_n \leftarrow E_1, \ldots, E_n$$

com $n \geq 1$, onde os x_i são nomes de variáveis, e os E_i são expressões que envolvem variáveis, constantes, argumentos, as operações primitivas e, eventualmente, nomes de procedimentos. A ação que corresponde à execução deste comando consiste em se calcular, em primeiro lugar, os valores e_i das expressões E_i, e em seguida adotá-los como os novos valores das variáveis x_i ($i = 1, \ldots, n$), respectivamente. Uma outra forma possível para o comando de atribuição é

$$x_1, \ldots, x_n \leftarrow f(E_1, \ldots, E_p)$$

com $n \geq 2$ e $p \geq 0$, onde f é o nome de um procedimento definido no programa, cuja execução deve devolver n resultados. (O caso $n = 1$ está incluído na primeira forma do comando.)

O *comando condicional* tem a forma

se E **então** S_1 **senão** S_2

onde E é uma expressão, e S_1 e S_2 são comandos quaisquer. Durante a execução deste comando, é calculado em primeiro lugar o valor e da expressão E, que deve ser verdadeiro ou falso. Se e for verdadeiro então será executado em seguida o comando S_1; caso contrário será executado o comando S_2. Quando se utilizam diagramas de blocos, o comando condicional corresponde à construção indicada na Figura 7.

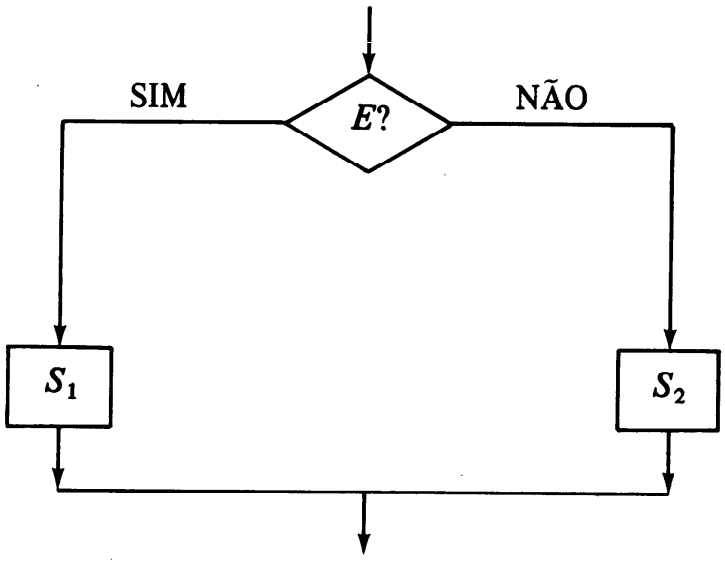


Figura 7

O comando **nada**, que não gera ação alguma, será usado, às vezes, como uma das alternativas S_1 ou S_2.

O *comando repetitivo* tem a forma

enquanto E **faça** S

onde E é uma expressão cujo valor deve ser verdadeiro ou falso, e S é um comando qualquer. Durante a execução, se o valor de E é falso, então a execução do comando repetitivo termina; caso contrário é executado o comando S, e o comando inteiro "**enquanto** E **faça** S" volta a ser executado. A Figura 8 indica o diagrama de blocos correspondente.

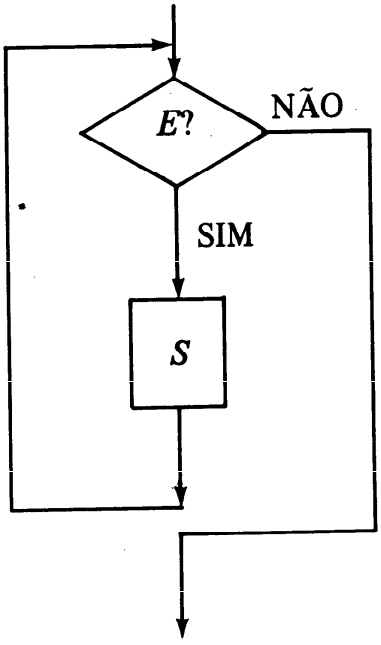


Figura 8

Um outro comando repetitivo tem a forma:

repita $S_1; S_2; \ldots; S_n$ **até que** E

com $n \geq 1$, onde os S_i são comandos quaisquer, e E é uma expressão cujo valor deve ser verdadeiro ou falso. O significado deste comando está indicado através do diagrama de blocos da Figura 9.

Em certos casos aparece a necessidade de se transformar uma seqüência de comandos num único comando como, por exemplo, quando esta seqüência deve constituir uma das alternativas de um comando condicional. Neste caso será usado o *comando composto* da forma:

início S_1 ; S_2 ; ... ; S_n **fim**

com $n \geq 1$. Note-se que os símbolos **início** e **fim** funcionam apenas como parênteses, indicando um agrupamento de comandos quando necessário. Os comandos S_1, S_2, ..., S_n serão executados um após o outro, nesta ordem.

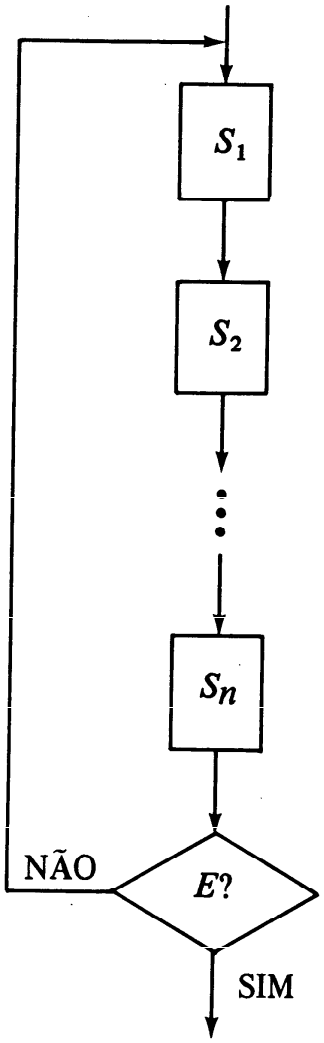


Figura 9

Finalmente, os resultados a serem devolvidos por um procedimento são indicados pelo *comando de retorno* da forma:

devolva $E_1, E_2, \ldots, E_n$

com $n \geq 1$. Os valores e_i das expressões E_i são devolvidos como resultados do procedimento dentro do qual o comando de retorno ocorre, encerrando a sua execução.

Uma convenção importante que adotaremos para a linguagem *LP* é a de que o *escopo* dos nomes dos argumentos e das variáveis é constituído pelo próprio texto da definição do procedimento no qual eles ocorrem. Em outras palavras, se num programa temos duas definições de procedimentos, ambas fazendo referência à variável x, na realidade trata-se de duas variáveis distintas.

Ocasionalmente, haverá observações no meio do programa visando facilitar a compreensão do mesmo. Tais observações, contidas sempre entre as chaves { e }, serão ignoradas durante a execução do programa.

O leitor atento deve ter notado que a definição da forma de um comando em *LP* é uma definição indutiva, e poderia ser formulada de maneira mais concisa. Se V é o conjunto de nomes de argumentos e de variáveis, F é o conjunto de nomes de procedimentos, e X é o conjunto de seqüências de símbolos que representam expressões, então o conjunto C de seqüências de símbolos que representam comandos de *LP* pode ser definido por

C é o menor conjunto tal que:

(i) a seqüência "**nada**" está em C;

(ii) se $x_1, \ldots, x_n$ estão em V, e $E_1, \ldots, E_n$ em X $(n \geq 1)$, então a seqüência "$x_1, \ldots, x_n \leftarrow E_1, \ldots, E_n$" está em C;

(iii) se $x_1, \ldots, x_n$ estão em V, f em F, e $E_1, \ldots, E_p$ em X $(n \geq 2, p \geq 0)$, então a seqüência "$x_1, \ldots, x_n \leftarrow f(E_1, \ldots, E_p)$" está em C;

(iv) se $E_1, \ldots, E_n$ estão em X $(n \geq 1)$, então a seqüência "**devolva** $E_1, E_2, \ldots, E_n$" está em C;

(v) se E está em X, e S_1 e S_2 em C, então a seqüência "**se** E **então** S_1 **senão** S_2" está em C;

(vi) se E está em X e S em C, então a seqüência "**enquanto** E **faça** S" está em C;

(vii) se E está em X, e $S_1, \ldots, S_n$ em C $(n \geq 1)$, então a seqüência "**repita** S_1 ; ...; S_n **até que** E" está em C;

(viii) se $S_1, \ldots, S_n$ estão em C $(n \geq 1)$, então a seqüência "**início** S_1 ; ...; S_n **fim**" está em C.

Na prática, estas definições são abreviadas usando-se a notação chamada FNB[2]. Nesta notação, os conjuntos de seqüências de sím-

(2) A abreviação vem de "Forma Normal de Backus". Essa notação foi usada pela primeira vez por J. Backus para descrever a linguagem Algol 60 num relatório editado por P. Naur [87]. Em inglês utiliza-se a abreviação BNF que indica tanto "Backus Normal Form" como "Backus Naur Form".

bolos recebem nomes mnemônicos colocados entre ⟨ e ⟩. Assim, a definição acima ficaria:

⟨comando⟩ :: =

nada
| ⟨variável⟩ [, ⟨variável⟩] ← ⟨expressão⟩ [, ⟨expressão⟩]
| **devolva** ⟨expressão⟩ [, ⟨expressão⟩]
| **se** ⟨expressão⟩ **então** ⟨comando⟩ **senão** ⟨comando⟩
| **enquanto** ⟨expressão⟩ **faça** ⟨comando⟩
| **repita** ⟨comando⟩ [; ⟨comando⟩] **até que** ⟨expressão⟩
| **início** ⟨comando⟩ [; ⟨comando⟩] **fim**

A notação [α] indica a repetição da seqüência α zero ou mais vezes. Note-se que os casos (ii) e (iii) da definição foram reunidos num único caso, por motivo de simplicidade. A rigor, isto não seria correto, pois permitiria comandos como "$x, y, z \leftarrow 0, 1$". Utilizando-se a mesma notação, a definição de *LP* pode ser completada com:

⟨programa⟩ :: =
⟨procedimento⟩ [⟨procedimento⟩]
⟨procedimento⟩ :: =
procedimento ⟨nome⟩ ( ): ⟨comando⟩
| **procedimento** ⟨nome⟩ (⟨nome⟩ [, ⟨nome⟩]): ⟨comando⟩

Note-se que estas definições determinam apenas a forma de um programa em *LP*, isto é, a sua *sintaxe*. Para que a definição fosse completa, deveríamos especificar também o significado, ou seja, a *semântica* de cada construção. Os métodos para a especificação da semântica de linguagens de programação são bastante complicados, e não serão discutidos neste texto.

Nas Partes *B* e *C*, a linguagem *LP* será utilizada para representar vários algoritmos. A fim de aumentar a clareza, serão introduzidas certas construções não incluídas na nossa definição, mas cujo significado deve ser óbvio dada a notação usual que será empregada. Uma convenção comum será a eliminação da seqüência "**senão nada**", sempre que possível. Esta eliminação não poderá ser feita nas construções da forma "**se** E_1 **então se** E_2 **então** S_1 **senão nada senão** S_2" e "**se** E_1 **então se** E_2 **então** S_1 **senão** S_2 **senão nada**", pois obteríamos a mesma forma "**se** E_1 **então se** E_2 **então** S_1 **senão** S_2" que seria, portanto, ambígua.

4. Iteração e recursão

Um conceito fundamental em programação é o de execução repetitiva de um comando, ou de uma série de comandos. Como vimos, a descrição de um procedimento é sempre finita. Na ausência de mecanismos repetitivos, o número de cálculos executados por um programa será limitado, então, por uma constante cujo valor depende do próprio programa. Assim, por exemplo, o Algoritmo de Euclides não poderia ser descrito por um programa desta espécie, uma vez que dado um inteiro K, é sempre possível escolher valores m e n tais que o número de divisões necessárias será maior do que K.

Todas as linguagens de programação de aplicação geral incluem mecanismos que causam execução repetitiva de comandos. Um mecanismo explícito existente em *LP* são os comandos repetitivos, chamados também de *comandos iterativos*. Por exemplo, no programa que representa o Algoritmo de Euclides, a seqüência de comandos "$r \leftarrow resto\ (x, y)$; $x, y \leftarrow y, r$" será executada repetitivamente até que o valor de r seja zero. Assim, se os valores dados são $m = 119$ e $n = 544$, os valores consecutivos das variáveis x, y e r durante o cálculo repetitivo serão:

x	y	r
119	544	119
544	119	68
119	68	51
68	51	17
51	17	0
17	0	

Analisemos agora o significado dos comandos iterativos, escolhendo para isto a forma "**enquanto** E **faça** S". Suponhamos que $m_1, \ldots, m_n$ são os argumentos e $x_1, \ldots, x_p$ são as variáveis de um programa no qual ocorre este comando iterativo. O comando S, ao ser executado, calcula, a partir dos valores $m_1, \ldots, m_n$ e $x_1, \ldots, x_p$, os novos valores atribuídos às variáveis $x_1, \ldots, x_p$ (alguns podem permanecer inalterados). Podemos supor, portanto, a existência das funções $g_i(u_1, \ldots, u_n, v_1, \ldots, v_p)$ $(i = 1, \ldots, p)$ que representam este cálculo, e reescrever o comando iterativo sob a forma:

enquanto E **faça**

$$x_1, x_2, \ldots, x_p \leftarrow g_1(m_1, \ldots, m_n, x_1, \ldots, x_p), g_2(m_1, \ldots, m_n, x_1, \ldots, x_p), \ldots, g_p(m_1, \ldots, m_n, x_1, \ldots, x_p).$$

Abreviando a notação para as seqüências de variáveis e funções, e lembrando que o valor de E também é função de $m_1, \ldots, m_n$ e $x_1, \ldots, x_p$, podemos escrever:

enquanto $E(m, x)$ **faça** $x \leftarrow g(m, x)$.

Sejam $a^0 = (a_1^0, \ldots, a_p^0)$ os valores iniciais das variáveis $x_1, \ldots, x_p$, antes de começar a execução do comando iterativo. Enquanto prosseguir a repetição de S, teremos uma seqüência de valores de x: a^0, a^1, a^2, ..., onde

$$a^i = g(m, a^{i-1}) \qquad (i \geq 1).$$

Esta seqüência será finita ou não conforme exista ou não um $k \geq 0$ tal que $E(m, a^k)$ seja falso. Caso a seqüência seja finita, o seu último valor a^k fornecerá os valores finais das variáveis x.

Estas considerações sugerem a seguinte regra para a demonstração de proposições que relacionam os valores de m e de x, baseada no princípio de indução matemática:

Se (i) a proposição $P(m, x)$ é verdadeira antes da execução de "**enquanto** E **faça** S" .

e (ii) a veracidade de $P(m, x)$ e de $E(m, x)$ implica a veracidade de $P(m, x)$ após a execução de S,

então

ou (iii) a execução do comando "**enquanto** E **faça** S" não termina,

ou (iv) após o seu término a proposição $P(m, x)$ é verdadeira e $E(m, x)$ é falsa.

Esquematicamente podemos representar esta regra num diagrama de blocos como indicado na Figura 10.

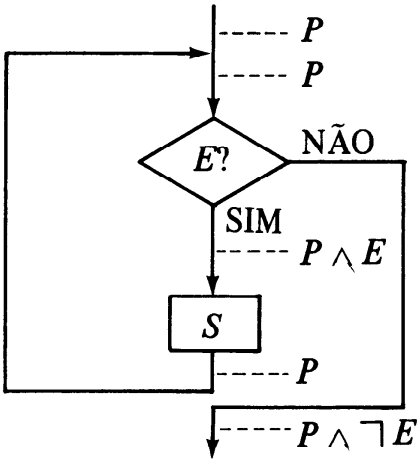


Figura 10

No texto de um programa apresentaremos o uso desta regra colocando as proposições como comentários:

$\{P\}$ **enquanto** $\{P\}\ E$
faça $\{P \wedge E\}\ S\ \{P\}$;
$\{P \wedge \hat{\neg}\ E\}$

Note-se que a indução está sendo feita sobre o número de vezes que o comando S é executado. A premissa (i) é a base da indução, e a premissa (ii) é o passo indutivo.

Considerações análogas levam à formulação de uma regra para o comando "**repita** S_1; ...; S_r **até que** E" que pode ser enunciada por:

$\{P\}$ **repita**
$\{P\}\ S_1; \ldots; S_r\ \{Q\}$
até que E;
$\{Q \wedge E\}$

onde $Q \wedge \neg E$ implica P. A Figura 11 indica esta regra num diagrama de blocos.

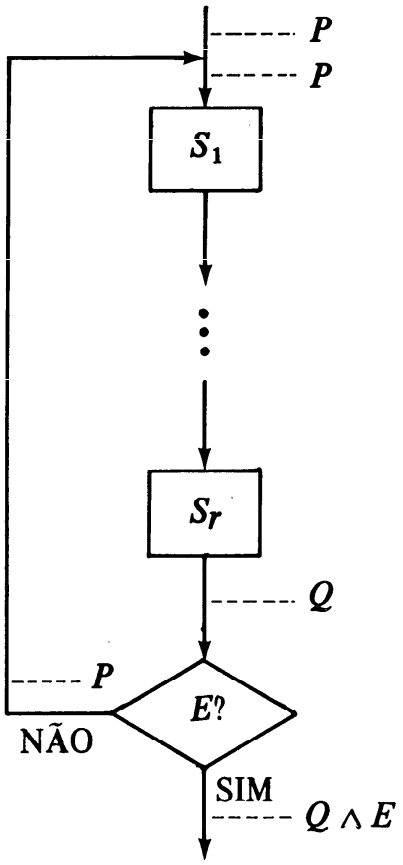


Figura 11

Um outro mecanismo existente em *LP*, e em várias linguagens de programação usadas na prática, é a *recursão*. Esta consiste em utilizar, direta ou indiretamente, um procedimento dentro do mesmo procedimento que o define. Consideremos a seguinte definição indutiva da função fatorial:

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n \cdot (n-1)! & \text{se } n > 0. \end{cases}$$

Esta definição pode ser transformada no seguinte programa em *LP*:

```
procedimento fat(n):
    se n = 0 então devolva 1
             senão devolva n · fat(n − 1)
```

Para compreender melhor o mecanismo de recursão, simulemos a execução deste programa para $n = 4$. Inicialmente, obtém-se:

$fat(4) = 4 \cdot fat(3)$

ou seja neste ponto deve-se suspender temporariamente o cálculo de $fat(4)$, "lembrando" para o uso posterior o valor $n = 4$ e passar a usar o mesmo programa para $n = 3$, e assim por diante:

$fat(3) = 3 \cdot fat(2)$
$fat(2) = 2 \cdot fat(1)$
$fat(1) = 1 \cdot fat(0)$
$fat(0) = 1$

Neste ponto devemos ir "relembrando" os valores anteriores de n:

$fat(1) = 1 \cdot 1 = 1$
$fat(2) = 2 \cdot 1 = 2$
$fat(3) = 3 \cdot 2 = 6$
$fat(4) = 4 \cdot 6 = 24$

Note que o programa será usado $n + 1$ vezes para calcular $n!$. Não é difícil, por outro lado, escrever uma versão iterativa do programa que calcula $n!$:

```
procedimento fat(n):
    início
        s, x ← 1, 1;
        enquanto x ≤ n faça s, x ← s · x, x + 1;
        devolva s
    fim
```

Simulando a execução deste programa para $n = 4$, obtêm-se os seguintes valores consecutivos de x e s:

x	s
1	1
2	1
3	2
4	6
5	24

Um outro exemplo de programa recursivo é o que calcula os números de Fibonacci. A seqüência de Fibonacci é definida indutivamente por:

$$F_n = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ F_{n-1} + F_{n-2} & \text{se } n > 1 \end{cases}$$

e esta definição pode ser diretamente transformada num programa em *LP*:

```
procedimento fib(n):
    se n = 0 então devolva 0
             senão
    se n = 1 então devolva 1
             senão
    devolva fib(n − 1) + fib(n − 2)
```

A Figura 12 indica como os cálculos seriam executados para $n = 4$:

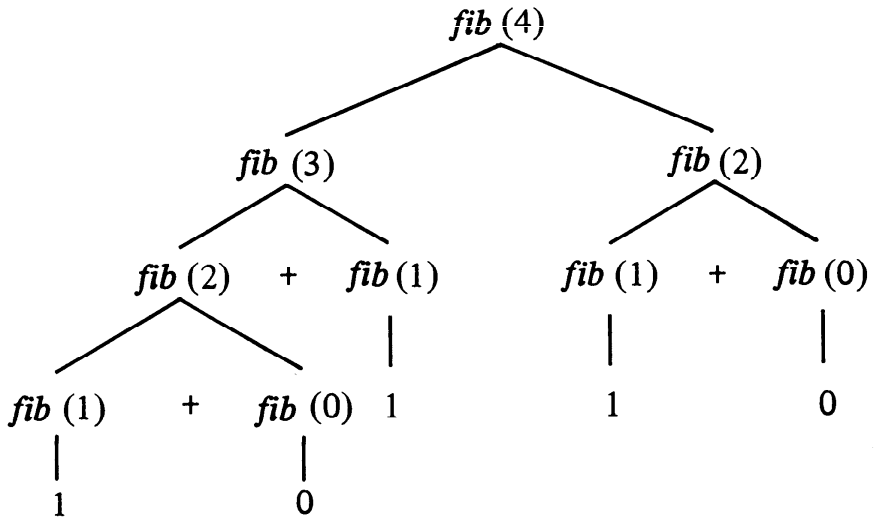


Figura 12

Devido à maneira como foi escrito o programa, vários valores são calculados mais do que uma vez.

Um conceito importante em programação é o de *eficiência* de programas, e que será discutido na Parte B deste texto. Adiantar-nos-emos um pouco, entretanto, e tentaremos avaliar a eficiência do programa acima. Uma medida aceitável de eficiência poderia ser o número total de operações primitivas (comparações, somas e subtrações) executadas para calcular F_n. Este número depende de n, e observando o programa, pode-se ver que será dado pela função:

$$c(n) = \begin{cases} 1 & \text{se } n = 0 \\ 2 & \text{se } n = 1 \\ 5 + c(n-1) + c(n-2) & \text{se } n > 1. \end{cases}$$

Esta é uma outra definição indutiva, cuja solução pode ser escrita em termos de números de Fibonacci:

$$c(n) = F_{n+2} + 5 \cdot (F_{n+1} - 1) \qquad (n \geq 0).$$

Pode-se obter uma fórmula fechada para $c(n)$ lembrando que

$$F_k = (\alpha^k - \beta^k)/\sqrt{5} \qquad (k \geq 0)$$

onde

$$\alpha = (1 + \sqrt{5})/2 \qquad \text{e} \qquad \beta = (1 - \sqrt{5})/2.$$

Usando o fato de que $|\beta| < 1$, para os valores crescentes de n obtém-se a aproximação:

$$c(n) \simeq 2{,}96 \cdot (1{,}62)^{n+1}.$$

Alguns exemplos dos valores de $c(n)$ são:

n	$c(n)$
25	803.378 (exato)
50	13×10^{10}
100	38×10^{20}

Para se ter uma idéia do valor de $c(100)$, basta dizer que um computador que realizasse um milhão de operações básicas por segundo, levaria cerca de $1{,}2 \times 10^8$ anos para calcular F_{100}! Na realidade, a memória deste computador teria sido esgotada muito antes. Por outro lado, o programa iterativo a seguir calcula F_n executando apenas $3 \cdot n + 1$ operações (isto é 301 operações para calcular F_{100}):

procedimento *fib*(n):
  início
    $x, y, k \leftarrow 0, 1, 0$;
    enquanto $k < n$ **faça** $x, y, k \leftarrow y, x + y, k + 1$;
    devolva x
  fim

Uma conclusão óbvia é que uma mesma função pode ser calculada por procedimentos de eficiência muito diferente (veja também o Exercício 7). Abordaremos novamente este problema na Parte B.

Uma pergunta que surge naturalmente é se os dois mecanismos, iteração e recursão, são em algum sentido equivalentes. A resposta a esta pergunta depende da definição exata do conceito de equivalência. Uma definição razoável seria a seguinte: dois programas são equivalentes se, e somente se, calculam a mesma função parcial, isto é, quando partindo dos mesmos dados, ou as execuções de ambos não terminam, ou então ambas terminam, sendo devolvidos os mesmos resultados.

Adotada esta definição, pode-se provar que um programa sempre pode ser transformado num programa recursivo equivalente. Na realidade, a discussão da iteração no início desta seção sugere a maneira de substituir a iteração por recursão. Sejam $m_1, \ldots, m_n$ e $x_1, \ldots, x_p$ os nomes dos argumentos e das variáveis usados na descrição de um procedimento, e "**enquanto** E **faça** S" o comando iterativo a ser eliminado desta descrição. Deve-se definir, então, o procedimento auxiliar:

procedimento $g(m_1, \ldots, m_n, y_1, \ldots, y_p)$:
  início
    $x_1, \ldots, x_p \leftarrow y_1, \ldots, y_p$;
    se E **então**
      início
        S; **devolva** $g(m_1, \ldots, m_n, x_1, \ldots, x_p)$
      fim
    senão devolva $x_1, \ldots, x_p$
  fim

O comando "**enquanto** E **faça** S" deve ser substituído por:

$$x_1, \ldots, x_n \leftarrow g(m_1, \ldots, m_n, x_1, \ldots, x_p).$$

Repetindo-se a aplicação desta transformação a todos os comandos iterativos do programa, chega-se a uma versão sem comandos iterativos.

Na prática, várias simplificações são possíveis pois nem todas as variáveis do programa precisam ser modificadas por *S*. Note-se que nesta transformação ignoramos totalmente a natureza das operações executadas pelo programa, e dos valores manipulados, fazendo uma transformação puramente simbólica do programa dado.

Aplicando-se esta transformação à versão iterativa do programa *fat*, obtém-se a seguinte versão recursiva equivalente:

```
procedimento fat(n):
    início
        s, x ← 1, 1;
        s, x ← g(n, s, x);
        devolva s
    fim

procedimento g(n, y1, y2):
    início
        s, x ← y1, y2;
        se x ≤ n então
                    início
                        s, x ← s • x, x + 1;
                        devolva g(n, s, x)
                    fim
                senão devolva s, x
    fim
```

A transformação contrária, isto é, de um programa que usa recursão em um programa que usa apenas iterações, depende da natureza dos objetos manipulados, e da existência de certas operações primitivas. Este problema está relacionado com a Tese de Church mencionada na Seção 2. No caso particular da *LP*, supondo a manipulação de inteiros e as operações primitivas de soma, subtração e comparação com zero, esta transformação será sempre possível se bem que bastante complicada. Deve-se notar que esta transformação pode ser feita de maneira sistemática, mas os programas iterativos obtidos têm essencialmente a mesma eficiência dos programas recursivos originais. Este fato deve ser contrastado com a transformação do procedimento *fib* indicada anteriormente. Como mostra o Exercício 7, a melhoria de eficiência não se deve à eliminação da recursão, mas sim à mudança do método de cálculo, ou seja do algoritmo. Pode-se provar que para

certas funções, como por exemplo a função $S(m_1, m_2, k)$ da Seção C.V.2, este tipo de melhoria é impossível.

Por outro lado, pode-se mostrar que num certo sentido a recursão é mais poderosa do que a iteração. Consideremos o seguinte exemplo:

procedimento $PH(m)$:
  se $r(m)$ **então devolva** $f(PH(g(m)), PH(h(m)))$
  senão devolva $k(m)$

em que não foi especificado o significado dos símbolos de função f, g, h e k, e de predicado r, nem o conjunto de objetos manipulados pelos mesmos. Note-se que este exemplo representa na realidade uma coleção de programas distintos, um para cada interpretação dos símbolos f, g, h, k e r. Em outras palavras, o exemplo representa um *esquema de programas*. Pode-se demonstrar então, que não existe um esquema puramente iterativo que seja equivalente ao esquema PH para todas as interpretações possíveis dos símbolos f, g, h, k e r.

Isto não quer dizer, entretanto, que para interpretações particulares não existem transformações equivalentes. Por exemplo, se o programa deve manipular números naturais, e:

$$f(y, z) \equiv y + z$$
$$g(y) \equiv h(y) \equiv y - 1$$
$$k(y) \equiv 1$$
$$r(y) \equiv y > 0$$

então o programa calcula 2^m, e um programa equivalente seria:

procedimento $T(m)$:
  início
    $y, s \leftarrow m, k(m)$;
    enquanto $r(y)$ **faça** $y, s \leftarrow g(y), f(s, s)$;
    devolva s
  fim

Também no caso da recursão podemos estabelecer uma regra de demonstração baseada no princípio de indução matemática. Para simplificar a exposição, suponhamos um programa composto de uma única definição recursiva da forma:

procedimento $f(m_1, \ldots, m_n)$: S

onde dentro do comando S ocorrem aplicações recursivas do procedimento f sob a forma $f(e_1, \ldots, e_n)$. Suponhamos, também, que deseja-se

demonstrar a proposição $P(m_1, \ldots, m_n, f(m_1, \ldots, m_n))$ que relaciona os dados $m_1, \ldots, m_n$ com os resultados $f(m_1, \ldots, m_n)$. Para cada chamada $f(e_1, \ldots, e_n)$ pode-se adotar, então, a hipótese de indução $P(e_1, \ldots, e_n, f(e_1, \ldots, e_n))$.

Evidentemente, esta regra corresponde à indução sobre o número de vezes que o procedimento f é aplicado de maneira recursiva durante o cálculo de $f(m_1, \ldots, m_n)$. Note-se que, como é comum em provas por indução em geral, a parte mais difícil da demonstração é achar uma hipótese de indução conveniente, seja no caso da iteração ou da recursão. A verificação da hipótese é, em geral, bastante simples.

Note-se, também, que as regras de demonstração estabelecidas permitem relacionar dados com resultados, caso a computação termine. As provas de terminação de programas são freqüentemente feitas à parte, muitas vezes por indução sobre o valor dos argumentos, como foi feito no caso do Algoritmo de Euclides.

5. Exemplos de aplicação

Mostraremos, nesta seção, vários exemplos de aplicação do princípio de indução, tanto sob a forma discutida na seção anterior, como sob outras formas.

EXEMPLO 1. Neste exemplo demonstraremos que o programa iterativo *Euclides* apresentado na Seção 3 realmente calcula o *mdc* de dois inteiros positivos m e n. A demonstração será feita indicando-se a relação existente entre m, n, e os valores das variáveis do programa antes e depois de cada comando. Estas relações, dadas pelas proposições Q_0 a Q_5, estão indicadas no texto do programa da Figura 13.

Q_0 é uma proposição verdadeira por hipótese. A execução de "$x, y \leftarrow m, n$" acarreta Q_1, e Q_1 implica de maneira trivial Q_2. Q_3 resulta de Q_2, e da definição do resto da divisão de dois inteiros positivos. Para concluir Q_4, basta tomar a proposição Q_3 e substituir y por x e r por y devido à execução de "$x, y \leftarrow y, r$". Na proposição assim obtida basta considerar os casos $y = 0$ e $y > 0$, e aplicar algumas propriedades do *mdc* para obter Q_4. Finalmente, vem a verificação do passo indutivo, ao notar que Q_4 e $r \neq 0$ implicam Q_2. Q_5 é uma conseqüência trivial de Q_4 e $y = 0$.

Fica provado, então, que o valor calculado é o $mdc(m, n)$ se a execução termina, e este fato já foi provado na Seção 2. Note-se que as

procedimento *Euclides*(m, n):
  início $\{Q_0 : m > 0 \wedge n > 0\}$
    $x, y \leftarrow m, n;$
      $\{Q_1 : m > 0 \wedge n > 0 \wedge x = m \wedge y = n\}$
    repita
      $\{Q_2 : m > 0 \wedge n > 0 \wedge x > 0 \wedge y > 0 \wedge mdc(m, n) =$
        $= mdc(x, y)\}$
      $r \leftarrow resto\ (x, y);$
      $\{Q_3 : m > 0 \wedge n > 0 \wedge x > 0 \wedge y > 0 \wedge mdc(m, n) =$
        $= mdc(x, y) \wedge \exists q(q \cdot y + r = x \wedge 0 \leq r < y)\}$
      $x, y \leftarrow y, r$
      $\{Q_4 : m > 0 \wedge n > |0 \wedge x > 0 \wedge y = r \wedge [y = 0 \wedge$
      $\wedge mdc(m, n) = x \vee y > 0 \wedge mdc(m, n) = mdc(x, y)]\}$
    até que $r = 0;$
      $\{Q_5 : mdc(m, n) = x\}$
    devolva x
  fim

Figura 13

proposições Q_2 e Q_4 correspondem às proposições P e Q usadas para formular a regra de demonstração para o comando "**repita** ... **até que** ...".

EXEMPLO 2. A Figura 14 indica as proposições necessárias para demonstrar que o programa *fib*, apresentado na Seção 4, calcula F_n segundo a sua definição indutiva.

procedimento *fib*(n):
  início
      $\{n \geq 0\}$
    $x, y, k \leftarrow 0, 1, 0;$
      $\{n \geq 0 \wedge x = F_0 \wedge y = F_1 \wedge k = 0\}$
    enquanto
      $\{n \geq 0 \wedge x = F_k \wedge y = F_{k+1} \wedge 0 \leq k \leq n\}$
      $k < n$
      faça
        $x, y, k \leftarrow y, x + y, k + 1$
        $\{n \geq 0 \wedge x = F_k \wedge y = F_{k+1} \wedge 0 \leq k \leq n\};$
      $\{n \geq 0 \wedge x = F_n\}$
    devolva x
  fim

Figura 14

Para mostrar que o programa termina basta verificar que dentro do único comando repetitivo, os valores sucessivos da variável k formam a seqüência 0, 1, 2, 3, ..., e como $n \geq 0$, depois de um número finito de repetições, exatamente n, teremos $k = n$.

EXEMPLO 3. Consideremos o seguinte programa[3]:

procedimento $f(n)$:
  se $n = 1$ **então devolva** 1
    senão
  se $resto(n, 2) = 0$ **então devolva** $n \div 2$
    senão devolva $f(f((3 \cdot n + 1) \div 2))$

onde n é um inteiro positivo, e $\div$ representa a divisão de inteiros. Este exemplo é interessante, pois não se sabe se o cálculo de $f(n)$ termina para qualquer $n \geq 1$. É instrutivo calcular o valor de f para alguns argumentos; assim (omitindo os parênteses):

$$f(33) = ff50 = f25 = ff38 = f19 = ff29 = fff44 = ff22 = \\ = f11 = ff17 = fff26 = ff13 = fff20 = ff10 = f5 = \\ = ff8 = f4 = 2.$$

A proposição a ser demonstrada neste caso é:

Se $n > 1$ então ou $f(n)$ não está definido (isto é, o cálculo de $f(n)$ não termina), ou então $f(n) \leq n \div 2$.

Usaremos a regra de demonstração dada na Seção 4 para programas recursivos, sendo que a proposição acima será a hipótese de indução. Seja um inteiro $n > 1$. Se o cálculo de $f(n)$ não termina então não há nada a demonstrar. Caso contrário, consideremos as duas possibilidades:

(a) n é par: $f(n) = n \div 2 \leq n \div 2$

(b) n é ímpar: $f(n) = f(a)$ com $a = f((3 \cdot n + 1) \div 2)$. Como o cálculo de $f(n)$ termina, então terminam também os de a e de $f(a)$. Por hipótese de indução aplicada duas vezes tem-se então:

$$a \leq (3 \cdot n + 1) \div 4$$

e

$$f(a) \leq a \div 2 \leq (3 \cdot n + 1) \div 8.$$

É fácil mostrar que para todo $n \geq 1$ tem-se $(3 \cdot n + 1) \div 8 \leq n \div 2$. Portanto, $f(n) \leq n \div 2$.

---

(3) O programa foi proposto por Morris [86].

EXEMPLO 4. Consideremos o seguinte programa[4]:

procedimento $g(n)$:
se $n > 100$ **então devolva** $n - 10$
senão devolva $g(g(n + 11))$

onde n é um inteiro qualquer. Demonstraremos que a função g definida pelo procedimento é igual à função h definida por:

$$h(n) = \begin{cases} n - 10 & \text{se } n > 100 \\ 91 & \text{se } n \leq 100. \end{cases}$$

Consideremos os três casos possíveis:

(a) $n > 100$: neste caso $g(n) = n - 10 = h(n)$.

(b) $90 \leq n \leq 100$: neste caso o valor de $g(n)$ será dado por $g(g(n + 11))$, onde as duas ocorrências de g correspondem a chamadas recursivas do procedimento; usando, portanto, a hipótese de indução duas vezes, teremos:

$$g(n) = g(g(n + 11)) = g(h(n + 11)) = h(h(n + 11)).$$

Por outro lado, $n + 11 > 100$ e, portanto:

$$h(h(n + 11)) = h(n + 1) = 91 = h(n)$$

pela definição da função h. Temos finalmente: $g(n) = h(n)$.

(c) $n < 90$: ainda neste caso, aplicando a hipótese de indução duas vezes, teremos:

$$g(n) = g(g(n + 11)) = g(h(n + 11)) = h(h(n + 11)).$$

Como $n + 11 \leq 100$, temos:

$$h(h(n + 11)) = h(91) = 91 = h(n).$$

Finalmente: $g(n) = h(n)$.

O que acabamos de demonstrar é, na realidade, que para todo inteiro n ou o cálculo de $g(n)$ não termina, ou então $g(n) = h(n)$. A demonstração de que o cálculo de $g(n)$ sempre termina pode ser feita considerando os três casos acima, e fazendo indução conveniente sobre os argumentos.

---

(4) O programa foi proposto por Burstall [12].

EXEMPLO 5. Suponhamos que em *LP* podemos manipular, além de inteiros, também seqüências de inteiros, e que para isto existem as seguintes operações primitivas:

$$car(\langle s_1, s_2, \ldots, s_n\rangle) = s_1 \text{ se } n \geq 1$$
$$cdr(\langle s_1, s_2, \ldots, s_n\rangle) = \langle s_2, \ldots, s_n\rangle \text{ se } n \geq 1$$
$$s \,\&\, \langle s_1, \ldots, s_n\rangle = \langle s, s_1, \ldots, s_n\rangle$$
$$\langle s_1, \ldots, s_n\rangle \,\#\, s = \langle s_1, \ldots, s_n, s\rangle$$
$$\langle s_1, \ldots, s_n\rangle\,[i] = s_i \text{ para } i = 1, \ldots, n.$$

A seqüência de zero elementos será indicada por $\langle\ \rangle$. Algumas dessas operações serão utilizadas no exemplo seguinte.

O programa da Figura 15 produz a seqüência reversa a partir da seqüência t dada. As proposições Q_0 a Q_4 inseridas no texto do programa indicam a maneira de mostrar que o programa produz resultados corretos:

procedimento *reverso*(t):
  início
    $\{Q_0\}$
    $x, y \leftarrow \langle\ \rangle, t;$
    $\{Q_1\}$
    enquanto
      $\{Q_2\}$
      $y \neq \langle\ \rangle$
    faça
      $x, y \leftarrow car(y)\&x, cdr(y)$
      $\{Q_3\};$
    $\{Q_4\}$
    devolva x
  fim

Figura 15

$Q_0 : t = \langle s_1, \ldots, s_n\rangle \wedge n \geq 0$
$Q_1 : t = \langle s_1, \ldots, s_n\rangle \wedge n \geq 0 \wedge x = \langle\ \rangle \wedge y = t$
$Q_2 : t = \langle s_1, \ldots, s_n\rangle \wedge n \geq 0 \wedge \exists\, i[0 \leq i \leq n \wedge x =$
$= \langle s_i, s_{i-1}, \ldots, s_1\rangle \wedge y = \langle s_{i+1}, s_{i+2}, \ldots, s_n\rangle]$
$Q_3 \equiv Q_2$ (hipótese de indução)
$Q_4 : t = \langle s_1, \ldots, s_n\rangle \wedge n \geq 0 \wedge x = \langle s_n, s_{n-1}, \ldots, s_1\rangle$

EXEMPLO 6. Neste exemplo, mostraremos uma maneira indutiva de resolver um problema[5], garantindo assim que o programa obtido será correto, isto é, satisfará as especificações do problema.

Seja M o menor conjunto que contém o elemento 1, e está fechado sob as operações $g(x) = 2x + 1$ e $f(x) = 3x + 1$. Deseja-se escrever um programa que constrói a seqüência dos n menores elementos de M ($n \geq 1$), em ordem crescente. Por exemplo, para $n = 15$ teríamos:

$$\langle 1, 3, 4, 7, 9, 10, 13, 15, 19, 21, 22, 27, 28, 31, 39 \rangle.$$

Para resolver o problema, suponhamos que já conseguimos construir a seqüência parcial:

$$\langle s_1, s_2, \ldots, s_m \rangle \qquad 1 \leq m \leq n.$$

É fácil verificar que existem p e q tais que $1 \leq p, q \leq m$, e

$$\begin{aligned} &g(s_i) \leq s_m \quad \text{para todo} \quad 1 \leq i < p \\ &f(s_i) \leq s_m \quad \text{para todo} \quad 1 \leq i < q \\ &g(s_p) > s_m \\ &f(s_q) > s_m. \end{aligned}$$

Em outras palavras, o menor entre $g(s_p)$ e $f(s_q)$ será o valor de s_{m+1}, e os novos valores de p e q serão:

$$p', q' = \begin{cases} p, q + 1 & \text{se } f(s_q) < g(s_p) \\ p + 1, q & \text{se } f(s_q) > g(s_p) \\ p + 1, q + 1 & \text{se } f(s_q) = g(s_p). \end{cases}$$

Foi feito, dessa maneira, o passo indutivo, estendendo-se com mais um elemento a seqüência parcial, e calculando-se os novos valores de p e q, a partir dos anteriores. É fácil verificar que os valores $m = p = q = 1$ e a seqüência $\langle 1 \rangle$ constituem uma base conveniente para esta indução. Todas estas considerações levam diretamente à construção do programa apresentado na Figura 16.

EXEMPLO 7. Vimos na Seção 3 como definições indutivas podem ser usadas para descrever a forma dos programas numa linguagem de programação. Mostraremos neste exemplo como estas definições podem ser usadas para escrever programas que decidem se uma dada seqüência de símbolos é ou não um programa. Indicaremos apenas o procedimento que reconhece comandos em *LP*, e

(5) Este problema aparece em Wirth [130].

```
procedimento h(n):
    início
        p, q, m, t ← 1, 1, 1, ⟨1⟩;
        enquanto m < n faça
            início
            a, b ← 2 • t[p] + 1, 3 • t[q] + 1;
                m ← m + 1;
                se a < b então p, t ← p + 1, t # a
                          senão
                se a > b então q, t ← q + 1, t # b
                          senão
                                p, q, t ← p + 1, q + 1, t # a
            fim;
        devolva t
    fim
```

Figura 16

suporemos que já foi definido o procedimento que reconhece expressões. O argumento do procedimento será sempre uma seqüência de símbolos da linguagem *LP* da forma $\langle s_1, s_2, \ldots, s_m \rangle$. Suporemos que símbolo s_m será sempre um símbolo especial □ que não faz parte de nenhum comando. Desta maneira evitaremos que em alguns pontos do programa a função *car* seja aplicada a uma seqüência vazia. Se para algum valor i ($1 \leq i \leq m - 1$), a seqüência $\langle s_1, s_2, \ldots, s_i \rangle$ representa um comando, então o resultado do procedimento é a seqüência restante $\langle s_{i+1}, \ldots, s_m \rangle$ (pode-se demonstrar que tal i é único). Se não existir tal i, então o resultado será $\langle$"**erro**"$\rangle$. Suporemos que o procedimento que reconhece expressões tem funcionamento semelhante. O predicado *var*(x) indica se o símbolo x é o nome de uma variável. O procedimento está apresentado na Figura 17.

```
procedimento comando(t):
    se car(t) = "nada"
        então devolva cdr(t)
        senão
    se var(car(t))
        então início
                  v ← cdr(t);
```

Figura 17 (continua)

```
            enquanto car(v) = "," faça
               se var(car(cdr(v))) então v ← cdr(cdr(v))
                                   senão devolva ⟨"erro"⟩;
            se car(v) = "←"
               então repita
                        v ← expressão(cdr(v));
                        se v = ⟨"erro"⟩ então devolva v
                                       senão nada
                     até que car(v) ≠ ","
               senão devolva ⟨"erro"⟩;
            devolva v
         fim
   senão
se car(t) = "devolva"
   então início
            v ← t;
            repita
               v ← expressão(cdr(v));
               se v = ⟨"erro"⟩ então devolva v
                              senão nada
            até que car(v) ≠ ",";
            devolva v
         fim
   senão
se car(t) = "se"
   então início
            v ← expressão(cdr(t));
            se car(v) ≠ "então"
               então devolva ⟨"erro"⟩
               senão início
                        v ← comando(cdr(v));
                        se car(v) ≠ "senão" então devolva ⟨"erro"⟩
                                           senão devolva
                                                 comando(cdr(v))
                     fim
         fim
   senão
se car(t) = "enquanto"
```

Figura 17 (continua)

```
        então início
                  v ← expressão(cdr(t));
                  se car(v) ≠ "faça" então devolva ⟨"erro"⟩
                                    senão devolva comando(cdr(v))
              fim
        senão
    se car(t) = "repita"
        então início
                  v ← t;
                  repita
                      v ← comando(cdr(v))
                  até que car(v) ≠ ";";
                  se car(v) ≠ "até" ∨ car(cdr(v)) ≠ "que"
                      então devolva ⟨"erro"⟩
                      senão devolva expressão(cdr(cdr(v))
              fim
        senão
    se car(t) = "início"
        então início
                  v ← t;
                  repita
                      v ← comando(cdr(v))
                  até que car(v) ≠ ";";
                  se car(v) ≠ "fim" então devolva ⟨"erro"⟩
                                    senão devolva cdr(v)
              fim
        senão devolva ⟨"erro"⟩
```

Figura 17 (conclusão)

EXERCÍCIOS

1. Descreva em *LP* um procedimento que verifica se um inteiro positivo dado é ou não primo. O seu procedimento é um algoritmo?
2. Descreva em *LP* um procedimento que calcula o número de divisores distintos de um inteiro positivo.
3. Descreva um procedimento cuja execução termina se, e somente se, o Último Teorema de Fermat é falso. Se a execução terminar, o procedimento deve devolver como resultados valores inteiros positivos x, y e z e valor inteiro $k > 2$ tais que $x^k + y^k = z^k$.

4. A construção indicada através do diagrama de blocos da Figura 18 não tem um equivalente direto em *LP*. Indique como tal construção poderia ser representada em *LP*, sem introduzir um comando repetitivo novo.

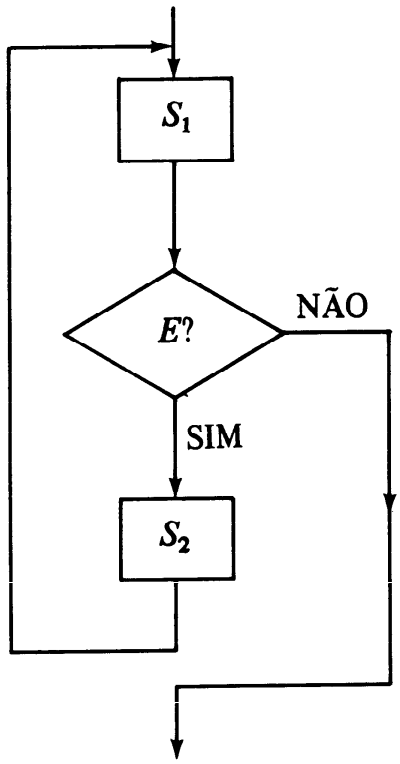


Figura 18

5. Indique as funções calculadas pelos seguintes programas:

(a) **procedimento** $f(n)$:
início
$k, s \leftarrow 0, 1$;
enquanto $k < n$ **faça** $s, k \leftarrow 2 \cdot s, k + 1$;
devolva s
fim

(b) **procedimento** $c(m, n)$:
início
$k, s \leftarrow m, 1$;
repita
$s, k \leftarrow s \cdot k, k + 1$
até que $k > n$;
devolva s
fim

(c) **procedimento** *sr*(*s*, *t*):
início
$n \leftarrow s[1] \cdot t[2] + s[2] \cdot t[1]$;
$d \leftarrow s[2] \cdot t[2]$;
$k \leftarrow$ *Euclides*(n, d);
devolva $\langle n \div k, d \div k \rangle$
fim

Neste último programa, s e t são pares de números inteiros positivos; *Euclides* é o nome do procedimento definido na Seção 3.

6. Mostre que dado um inteiro K, é sempre possível escolher valores m e n tais que o número de divisões executadas pelo Algoritmo de Euclides é maior que K.

7. Escreva um programa para calcular o n-ésimo número de Fibonacci, sem usar comandos iterativos, e executando apenas $3 \cdot n + 1$ operações primitivas para calcular F_n.

8. Determine a função $f(m, n)$ que indica o número de vezes que o procedimento *fib* da Figura 11 é chamado com argumento m durante o cálculo de *fib*(n), $n \geq m \geq 0$.

9. Ache interpretações dos símbolos f, g, h, k e r para que o programa *PH* da Seção 4 calcule as funções
(a) $2 \cdot m$
(b) F_m
(c) $m \div 2$.

10. Escreva um programa não recursivo que calcula a mesma função que o procedimento f do Exemplo 3 da Seção 5.

11. Mostre que o procedimento apresentado a seguir devolve o máximo divisor comum e o mínimo múltiplo comum de inteiros positivos m e n:

procedimento *mdcmmc*(m, n):
início
$x, y, v, w \leftarrow m, n, n, m$;
enquanto $x \neq y$ **faça**
se $x > y$ **então** $x, w \leftarrow x - y, v + w$
senão $y, v \leftarrow y - x, v + w$;
devolva $x, (v + w) \div 2$
fim

Mostre, também, que a execução do procedimento termina para quaisquer valores inteiros positivos m e n.

12. Complete o programa da Figura 17, incluindo a verificação de que os comandos de atribuição são da forma indicada na Seção 3:

$$x_1, \ldots, x_n \leftarrow E_1, \ldots, E_n \qquad (n \geq 1)$$

ou

$$x_1, \ldots, x_n \leftarrow f(E_1, \ldots, E_p) \quad (n \geq 2, p \geq 0)$$

13. Usando a notação FNB, defina o conjunto de seqüências de expressões formadas por nomes de variáveis, símbolos de operações $+$, $-$, $\cdot$ e $\div$, e parênteses. Escreva um procedimento que reconhece expressões análogo ao do Exemplo 7 da Seção 5.

NOTAS BIBLIOGRÁFICAS

Uma discussão em nível introdutório dos conceitos de procedimento e algoritmo pode ser encontrada em Knuth [60]. Davis [18] e Rogers [101] são dois textos mais avançados, e clássicos, sobre a teoria da computabilidade.

Existem inúmeras publicações sobre linguagens de programação e sua utilização. Recomendamos, em particular, o texto de Wirth [130]. A notação FNB foi usada pela primeira vez em Naur [87] para descrever a linguagem Algol 60.

A literatura sobre a demonstração de propriedades de programas é bastante vasta, mas não muito consolidada. O texto de Wirth mencionado acima introduz estas noções. Outras publicações recomendadas são Elspas et al. [27], Manna [77] e Manna et al. [78]. O problema da relação entre a iteração e a recursão é discutido de uma maneira mais profunda em Paterson e Hewitt [93] e Walker e Strong [125].

PARTE B

COMPLEXIDADE DE ALGORITMOS

CAPÍTULO I

COMPLEXIDADE DE ALGORITMOS: NOÇÕES BÁSICAS

1. Introdução

Na Parte A, examinamos alguns aspectos importantes da síntese, descrição e correção de algoritmos. Todas estas questões são de grande interesse, não só do ponto de vista teórico, mas também prático. Certamente é fundamental sabermos como escrever programas corretos para algoritmos de nosso interesse, bem como demonstrar que os algoritmos calculam corretamente as funções desejadas, isto é, que após um número finito de passos nos fornecerão as respostas de acordo com as especificações. Em geral, porém, não basta garantir que um algoritmo forneça as respostas corretas em um número finito de passos. Para que o algoritmo possa ser útil, é necessário que este número de passos seja não só "finito", mas "muito finito". Este fato já foi ilustrado na Seção A.I. 4, onde exibimos um programa recursivo para o cálculo do n-ésimo número de Fibonacci, F_n, correto mas inútil para valores moderados de n. Assim, vimos que levaria mais de cem milhões de anos para calcular F_{100} por meio deste algoritmo, dispondo de um computador que efetue um milhão de operações por segundo! Evidentemente, podemos calcular F_{100} de maneira muito mais rápida utilizando-se o algoritmo iterativo descrito na mesma seção, ou melhor ainda, a fórmula fechada para F_n, $F_n = (\alpha^n - \beta^n)/\sqrt{5}$, onde $\alpha = (1 + \sqrt{5})/2$ e $\beta = (1 - \sqrt{5})/2$. Fica claro portanto, que considerações sobre a eficiência de desempenho de nossos algoritmos não podem ser ignoradas, e são estas as questões que são o objeto da Teoria da *Complexidade de Algoritmos*.

Embora a preocupação com eficiência de algoritmos seja possivelmente tão antiga quanto a própria noção de algoritmo, um estudo sistemático destas questões é relativamente recente. Na década de 1950 vários algoritmos foram analisados. Os primeiros trabalhos que estabeleceram as bases desta teoria, porém, só surgiram em torno de 1960 [97, 98, 54, 55].

Vale a pena ilustrar algumas das questões neste campo com mais um exemplo. Consideremos o seguinte programa:

```
procedimento mdc(m, n):
    início
        se m < n então d ← m senão d ← n;
        enquanto resto (m, d) ≠ 0 ∨ resto (n, d) ≠ 0 faça d ← d − 1;
        devolva d
    fim
```

Um pouco de reflexão deve convencer o leitor que, se m e n forem inteiros positivos, o procedimento acima calcula o máximo divisor comum entre m e n. De fato, a condição do comando repetitivo garante que o valor final de d é um divisor comum de m e n, e como o valor inicial de d é $min\,(m, n)$, claramente maior ou igual a qualquer divisor comum, e d decresce de um em um, resulta que o valor final de d é o maior de tais divisores. Na Parte A vimos que o Algoritmo de Euclides também calcula o máximo divisor comum de m e n. É natural, portanto, perguntar qual destes dois algoritmos é melhor. Um critério para comparar estes dois algoritmos poderia ser a avaliação da sua eficiência pelo tempo que levam para calcular a resposta; esse tempo é diretamente proporcional ao número de operações efetuadas durante a computação[1]. Sejam $T_A(m, n)$ e $T_E(m, n)$ o número de operações efetuadas pelo algoritmo acima e o Algoritmo de Euclides respectivamente. Pelo Exercício 1, $T_A(m, n) = 7\,(min\,(m, n) - mdc\,(m, n)) + 8$. Em particular vemos que fixado n, $T_A(m, n)$ varia entre 8 e $7n + 1$, e portanto no pior caso vale $7n + 1$. O valor de $T_E(m, n)$ é mais difícil de determinar. Pelo Exercício 2 temos, no entanto, que $T_E(m, n) \leq 10\ log_2\ n + 8$. Para cada n, portanto, o Algoritmo de Euclides é, no pior caso, mais eficiente do que o algoritmo *mdc*, exceto para valores pequenos de n. Veja também os Exercícios 3 e 4, para outras questões interessantes sobre o desempenho destes dois algoritmos.

O exemplo simples acima ilustra que a análise do desempenho de nossos algoritmos pode nos fornecer subsídios úteis para a escolha do algoritmo mais adequado, além de sugerir muitas questões interessantes do ponto de vista matemático dignas de serem estudadas. Existe considerável conhecimento acumulado nos últimos vinte anos acerca das técnicas que se revelaram úteis na solução dos problemas que surgem na análise de algoritmos, e muitos algoritmos já foram analisados com êxito usando-se estas técnicas (veja [60, 61, 62]). Além de

---

(1) Vamos supor que cada operação elementar como soma, comparação, atribuição, etc., leve uma unidade de tempo.

ser útil na escolha do algoritmo mais adequado em cada caso, a análise de algoritmos aumenta a nossa compreensão dos algoritmos analisados, sugerindo muitas vezes idéias importantes para a sua melhoria.

Existem, porém, questões importantes que a simples análise dos algoritmos já conhecidos não pode responder, pois existem infinitos algoritmos para calcular cada função computável. Por exemplo, em geral queremos não só saber qual a quantidade de recursos computacionais utilizada por algum algoritmo conhecido, mas também se este algoritmo pode ainda ser melhorado. Enquanto a análise do algoritmo conhecido nos fornece um *limite superior* para a quantidade de recursos que é suficiente para resolver esta tarefa, o primeiro passo para responder se este algoritmo pode ser melhorado é estabelecer um *limite inferior* na quantidade de recursos necessária. Naturalmente, temos interesse em estabelecer o maior limite inferior que conseguirmos, e analogamente, o menor limite superior possível. Na situação ideal os dois limites deveriam ser iguais, caso este em que conheceríamos exatamente a quantidade de recursos que é tanto necessária como suficiente. Se dispusermos de um algoritmo que utilize exatamente esta quantidade de recursos, então, obviamente, temos um *algoritmo ótimo* para a tarefa, no sentido de que a quantidade de recursos utilizada por qualquer outro algoritmo para a tarefa será maior, ou no melhor dos casos igual, à do algoritmo que temos.

Na prática, são raros os casos em que temos esta situação ideal. Pode-se mesmo demonstrar que existem problemas que não possuem algoritmos ótimos. Normalmente, a diferença entre o limite superior e o limite inferior é uma indicação de quanto poderíamos, no máximo, melhorar o algoritmo de que dispomos. É possível, no entanto, que a melhoria efetivamente obtenível seja bem menor do que a diferença entre o limite superior e inferior nos faria acreditar, porque o limite inferior é demasiadamente baixo. Para responder este tipo de questão, é preciso, pois, estudar classes de algoritmos em vez de algoritmos individuais.

Estabelecer limites inferiores para a quantidade de recursos necessária para uma tarefa é, em geral, um problema muito difícil. Nos três capítulos seguintes voltaremos a este problema sob vários pontos de vista. No Capítulo II mostraremos que existem tarefas naturais de computação para as quais não existem algoritmos. Isto pode ser encarado como um problema extremo de limites inferiores, no sentido de que nenhuma quantidade de recursos, por maior que seja, é suficiente para computar estas funções. No Capítulo III estudaremos a comple-

xidade de algoritmos que calculam o produto de duas matrizes. A medida de complexidade aqui, será o número de multiplicações utilizadas. Como veremos, o algoritmo clássico de multiplicação de matrizes, surpreendentemente, não é o melhor algoritmo, e apresentaremos um algoritmo que lhe é assintoticamente superior. A seguir esboçaremos uma teoria algébrica que permite provar limites inferiores para este tipo de problema. A relativa sofisticação do método, quando comparada com os resultados, às vezes modestos, que podem ser provados mediante o seu uso, ilustra bem a dificuldade de se provar limites inferiores.

No Capítulo IV isolaremos uma classe de problemas que num certo sentido preciso têm aproximadamente a mesma complexidade. Esta classe inclui, como veremos, muitos problemas de interesse em diversas áreas do conhecimento, no entanto não se sabe se estes problemas podem ou não ser resolvidos eficientemente. Temos aqui, portanto, a interessante situação em que todos os limites superiores conhecidos para resolver estes problemas são "ineficientes", mas não se conseguiu até hoje provar limites inferiores que mostrem que este é necessariamente o caso.

Como indica a discussão acima, um dos problemas fundamentais da Teoria de Complexidade é a estimativa dos recursos computacionais necessários e suficientes para o cálculo de funções computáveis específicas de nosso interesse. Poderia parecer à primeira vista que, em virtude disto, a teoria tenderia a ser uma coleção de técnicas, cada uma adequada a um problema específico. Na realidade, porém, existem relações entre as complexidades de muitos problemas aparentemente diferentes, permitindo reduzir um problema a outro. Esta técnica de reduções aparece com destaque nos três capítulos seguintes sob diversas formas.

Existem naturalmente outras questões importantes da Teoria de Complexidade que não examinaremos nos capítulos seguintes. Por exemplo, uma pergunta fundamental da teoria é determinar qual a relação entre as diversas medidas de complexidade de algoritmos, tais como tempo e memória utilizados. Não estudaremos aqui tais questões, embora acreditemos que os capítulos seguintes constituam uma amostra significativa não só de alguns dos resultados mais relevantes da teoria, mas também dos métodos nela utilizados.

2. Máquinas de Turing

Para se poder estudar a complexidade computacional de uma tarefa é necessário escolher um modelo formal de algoritmos. Já vimos uma possível formalização por meio da linguagem de programação *LP* descrita na Parte A. Em muitos casos, porém, este tipo de formalização é por demais complexo para ser útil do ponto de vista teórico. No Capítulo III, por exemplo, é introduzida uma simplificação deste modelo para eliminar os aspectos da linguagem, irrelevantes para a questão lá estudada, facilitando assim o estudo do número de operações algébricas utilizadas pelos algoritmos.

Provavelmente o modelo formal de algoritmo mais utilizado em Teoria da Computação é o de Turing. Esta preferência se deve ao fato de que este modelo é suficientemente simples, facilitando as demonstrações, mas ao mesmo tempo suficientemente poderoso para que os resultados provados se apliquem a modelos aparentemente com mais recursos. Em particular, o modelo é suficientemente poderoso para que qualquer algoritmo possa ser nele representado. Passaremos agora a definir este modelo.

Uma *máquina de Turing determinística* é um sistema formal que pode ser visualizado como um computador consistindo de uma *fita de entrada/trabalho/saída* manipulada por um *controle central* (Figura 1). A fita é subdividida em *células*, cada uma das quais contém um símbolo de um alfabeto finito Σ[2], e é infinita para a direita. Embora isto não seja essencial, suporemos que Σ contém pelo menos os quatro símbolos $\{\vdash, \not b, 0, 1\}$, podendo, no entanto, conter símbolos adicionais. A primeira célula à esquerda contém sempre o símbolo $\vdash$, indicando que esta é a célula mais à esquerda da fita. Este símbolo não é usado para nenhum outro propósito. O controle central tem acesso a informações na fita, e pode modificar tais informações, por meio de uma *cabeça* móvel de *leitura* e *gravação* que, em cada instante, encontra-se sobre uma das células da fita. O controle central, por sua vez, está em algum *estado* q, de um conjunto finito de estados Q. Inicialmente, o controle central está no estado q_0, chamado de *estado inicial*, a cabeça encontra-se

---

(2) Um *alfabeto* Σ é um conjunto de símbolos. Uma seqüência finita de símbolos de Σ é uma *palavra* sobre Σ. O conjunto de todas as palavras sobre Σ é denotado por Σ^*. O número de símbolos de uma palavra x é denotado por $|x|$, e é chamado o *comprimento* de x. A palavra de comprimento zero é denotado por Λ e é chamada a *palavra vazia*.

sobre a célula mais à esquerda, contendo o símbolo $\vdash$, e as n células seguintes contém uma palavra x, chamada de *entrada*, sobre o *alfabeto de entrada e saída*, $\Sigma_{e/s} \subseteq \Sigma\backslash\{\vdash, \not b\}$. As demais células da fita contêm o símbolo *branco*, $\not b$. A computação da máquina começa então, prosseguindo passo a passo. Um *passo* da máquina consiste das seguintes operações:

(a) ler o símbolo σ sob a cabeça;
(b) escrever um símbolo σ' no lugar de σ;
(c) reposicionar a cabeça, que pode ser movida uma célula à direita ou à esquerda, ou deixada no mesmo lugar;
(d) mudar o controle central para um novo estado q'.

As operações (b), (c) e (d) dependem apenas do símbolo σ lido em (a) e do estado q do controle central antes deste passo. Ademais, o símbolo σ' é necessariamente diferente de $\vdash$ sempre que σ for diferente de $\vdash$; no caso em que $\sigma = \vdash$, a cabeça encontra-se sobre a célula mais à esquerda da fita, e então $\sigma' = \vdash$ obrigatoriamente e a cabeça não pode ser movida para a esquerda.

A computação prossegue passo a passo, só sendo interrompida se o controle central assumir um dos dois *estados finais* q_a ou q_r. Se isto acontecer, dizemos que a computação *pára*. Neste caso, se o estado final for q_a, dizemos que a máquina *aceita* a entrada x, e se for q_r, dizemos que a máquina *rejeita* a entrada x. Se a máquina nunca parar, x não é nem aceita nem rejeitada. A *saída* da máquina, definida apenas se a máquina parar, é a palavra sobre $\Sigma_{e/s}$, escrita após a primeira célula à esquerda contendo $\vdash$, até o primeiro símbolo de $\Sigma\backslash\Sigma_{e/s}$ na fita, exclusive.

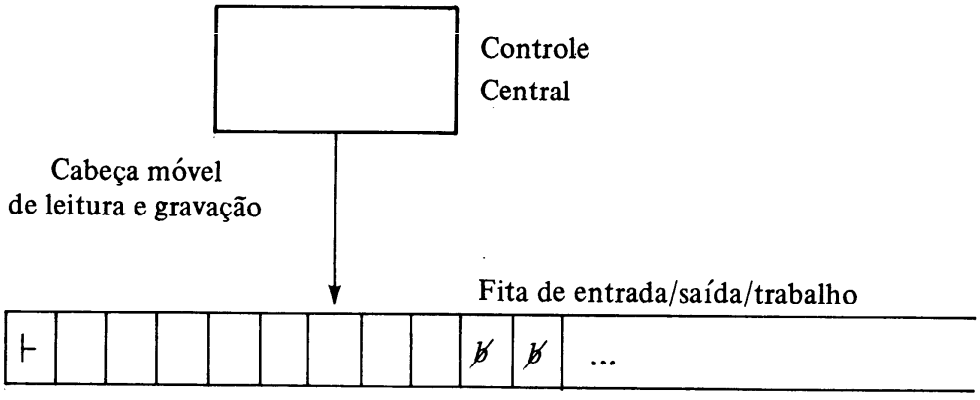


Figura 1 — Uma máquina de Turing.

Uma máquina de Turing M *reconhece* um conjunto $A \subseteq \Sigma^*_{e/s}$ se para toda entrada $x \in \Sigma^*_{e/s}$, M pára, e aceita x sse $x \in A$. Uma máquina

de Turing M *calcula uma função* f: $\Sigma^*_{e/s} \to \Sigma^*_{e/s}$ se para toda entrada $x \in \Sigma^*_{e/s}$, M pára, e a saída de M é a palavra $f(x)$.

Resumindo formalmente a discussão acima: uma máquina de Turing M é um sistema $(Q, q_0, q_a, q_r, \Sigma, \Sigma_{e/s}, \vdash, \not b, \delta)$ onde $q_0, q_a, q_r \in Q$, $\vdash, \not b \in \Sigma$, $\Sigma_{e/s} \subseteq \Sigma \backslash \{\vdash, \not b\}$ e δ é a *função de transferência* de M, $\delta: Q \times \Sigma \to Q \times \Sigma \times \{-1, 0, 1\}$. A interpretação de $\delta(q, \sigma) = (q', \sigma', \Delta)$ é que a máquina no estado q, tendo lido σ, escreve σ', move a cabeça Δ células para a direita, e muda para o estado q', sendo portanto sujeita às restrições:

$$\begin{array}{l} \text{se } \sigma = \vdash \text{ então } \sigma' = \vdash, \text{ e } \Delta = +1 \text{ ou } \Delta = 0, \\ \text{se } \sigma \neq \vdash \text{ então } \sigma' \neq \vdash, \\ \text{e} \quad \delta(q_a, \sigma) = (q_a, \sigma, 0), \\ \quad\quad \delta(q_r, \sigma) = (q_r, \sigma, 0). \end{array}$$

Finalmente, a transformação da fita associada à máquina M pode ser formalmente definida pela relação $\vDash_M$ abaixo sobre $\hat{\Sigma}^*$, onde $\hat{\Sigma} = \Sigma \cup Q \times \Sigma$. Para $x, y \in \Sigma^*$ e $\delta(q, \sigma) = (q', \sigma', \Delta)$:

(i) se $\Delta = 0$ então $x(q, \sigma)y \vDash_M x(q', \sigma')y$;
(ii) se $\Delta = +1$ e $y = \tau y'$, ou se $\Delta = +1$, $y' = y = \Lambda$ e $\tau = \not b$, então $x(q, \sigma)y \vDash_M x\sigma'(q', \tau)y'$;
(iii) se $\Delta = -1$ e $x = x'\tau$ então $x(q, \sigma)y \vDash_M x'(q', \tau)\sigma' y$.

Uma *máquina de Turing não determinística* é um sistema formal como o acima descrito, com a única diferença que a máquina não determinística possui uma fita adicional chamada *fita de sugestões*. O controle central tem acesso a esta fita, mas não pode modificar as informações nela armazenadas. O acesso é por meio de uma *cabeça de leitura apenas*, que não pode gravar. Esta cabeça encontra-se inicialmente sobre a primeira célula à esquerda da fita, que contém o símbolo $\vdash$, as células seguintes contêm uma palavra y de $\Sigma^*_{e/s}$, e as demais células da fita de sugestões contém brancos $\not b$. A computação da máquina não determinística prossegue passo a passo como descrito antes. Naturalmente, a função de transferência que descreve um passo da máquina é agora uma função δ: $Q \times \Sigma \times \Sigma \to Q \times \Sigma \times \{-1, 0, +1\}^2$, refletindo que cada passo depende agora não só do estado q do controle central e do símbolo σ lido da fita de entrada/trabalho/saída, mas também do símbolo lido da fita de sugestões, com a seguinte interpretação: se $\delta(q, \sigma_1, \sigma_2) = (q', \sigma', \Delta_1, \Delta_2)$ então o controle central no estado q, lendo σ_1 na fita de entrada/trabalho/saída e σ_2 na fita de sugestões, escreve σ' no lugar de σ_1, move as duas cabeças respectivamente Δ_1 e Δ_2 células para a direita e muda para o estado q'.

A idéia da fita de sugestões é dar à máquina informações auxiliares, daí o nome de fita de sugestões, para ajudar a determinar se a entrada x da máquina deve ou não ser aceita. Isto sugere a definição abaixo.

Um conjunto $A \subseteq \Sigma_{e/s}^*$ é *aceito* por uma máquina não determinística M, se com entrada x:

(a) se $x \in A$ então com alguma sugestão y a máquina M pára e aceita x;

(b) se $x \notin A$ então qualquer que seja a sugestão y, M ou nunca pára, ou rejeita x.

O leitor atento terá observado que a definição de máquina de Turing não determinística estende a definição do modelo determinístico. De fato, qualquer máquina de Turing determinística pode ser encarada como uma máquina não determinística que ignora a sua fita de sugestões. Deste modo, uma máquina determinística M aceita um conjunto A se M, encarada como uma máquina não determinística, aceitar A. Note que se uma máquina determinística reconhecer A, então ela também aceita A. A recíproca não é verdadeira: é possível a uma máquina determinística M aceitar um conjunto A sem reconhecê-lo.

Vamos ver agora um exemplo simples aplicando os conceitos acima. Considere a máquina de Turing determinística

$M = (Q, q_0, q_a, q_r, \Sigma, \Sigma_{e/s}, \vdash, \not b, \delta)$ que reconhece[3] $\{0^n1^n \mid n \geq 1\}$, onde

$Q = \{q_0, q_1, q_2, q_3, q_4, q_a, q_r\}$, $\Sigma = \{\vdash, \not b, 0, 1, A, B\}$, $\Sigma_{e/s} = \{0, 1\}$ e δ é dado por:

(i) $\delta(q_0, \vdash) = (q_0, \vdash, +1)$
$\delta(q_0, 0) = (q_1, A, +1)$

(M verifica se a entrada é da forma 0^n1^n substituindo alternadamente um 0 por um A e um 1 por um B. Assim no estado q_0, o 0 lido é substituído por um A e em seguida M move para a direita no estado q_1 procurando o primeiro símbolo 1.);

(ii) $\delta(q_1, 0) = (q_1, 0, +1)$
$\delta(q_1, B) = (q_1, B, +1)$
$\delta(q_1, 1) = (q_2, B, -1)$

---

(3) Para $\sigma \in \Sigma$ a notação σ^n denota a palavra $\sigma\sigma \ldots \sigma$ de comprimento n.

(No estado q_1, M passa por cima dos símbolos 0 e B até encontrar o primeiro símbolo 1, que é substituído por B. M agora moverá à esquerda no estado q_2.);

(iii) $\delta(q_2, B) = (q_2, B, -1)$
$\delta(q_2, 0) = (q_3, 0, -1)$
$\delta(q_2, A) = (q_4, A, +1)$

(No estado q_2 a máquina vai à esquerda, passando por cima de $B's$. Se houver mais zeros a serem substituídos, o primeiro símbolo que não seja B será um 0, e neste caso a máquina procurará o 0 mais à esquerda no estado q_3. Caso contrário, se o primeiro símbolo diferente de B for um A, então não há mais zeros, e precisamos apenas verificar se todos os $1's$ foram substituídos por $B's$.);

(iv) $\delta(q_3, 0) = (q_3, 0, -1)$
$\delta(q_3, A) = (q_0, A, +1)$

(No estado q_3 movemos por cima dos zeros até um símbolo A. Quando este é encontrado movemos à direita no estado q_0 para substituir o próximo símbolo 0 por A.);

(v) $\delta(q_4, B) = (q_4, B, +1)$
$\delta(q_4, \not b) = (q_a, \not b, 0)$

(M verifica se todos os $1's$ foram substituídos por $B's$.);

(vi) Para todo $\sigma \in \Sigma$,
$\delta(q_a, \sigma) = (q_a, \sigma, 0)$
$\delta(q_r, \sigma) = (q_r, \sigma, 0)$;

(vii) Para todo $(q, \sigma) \in Q \times \Sigma$ tal que $\delta(q, \sigma)$ não foi definido anteriormente, $\delta(q, \sigma) = (q_r, \sigma, 0)$.

Para a entrada 0011 temos a seguinte seqüência de passos que aceitam esta entrada:

$$
\begin{aligned}
(q_0, \vdash)0011 &\vDash_M \vdash(q_0, 0)011 \vDash_M \vdash A(q_1, 0)11 \vDash_M \vdash A0(q_1, 1)1 \\
&\vDash_M \vdash A(q_2, 0)B1 \vDash_M \vdash(q_3, A)0B1 \vDash_M \vdash A(q_0, 0)B1 \\
&\vDash_M \vdash AA(q_1, B)1 \vDash_M \vdash AAB(q_1, 1) \vDash_M \vdash AA(q_2, B)B \\
&\vDash_M \vdash A(q_2, A)BB \vDash_M \vdash AA(q_4, B)B \vDash_M \vdash AAB(q_4, B) \\
&\vDash_M \vdash AABB(q_4, \not b) \vDash_M \vdash AABB(q_a, \not b).
\end{aligned}
$$

O Exercício 15 mostra que esta máquina de Turing reconhece o conjunto $\{0^n1^n \mid n \geq 1\}$.

Pelo exemplo acima podemos notar que descrever uma máquina de Turing pela sua função de transferência é bastante trabalhoso, mesmo para resolver problemas relativamente simples. Para facilitar a construção de máquinas de Turing, certas "técnicas de programação" destas máquinas tornam-se indispensáveis. Com tais construções podemos descrever máquinas de Turing mais complexas sem precisarmos dar explicitamente funções de transferência envolvendo possivelmente centenas de estados. Por falta de espaço não daremos aqui estas técnicas. Quando precisarmos mais adiante construir máquinas de Turing, daremos descrições que esperamos que sejam suficientemente pormenorizadas para que o leitor se convença de que, seguindo a descrição, pode construir a função de transferência da máquina desejada.

Existem vários textos de computabilidade mencionados nas notas bibliográficas em que o leitor interessado pode encontrar estas técnicas. A resolução dos Exercícios 7 a 12 pode também ser de utilidade para adquirir alguma prática na construção de máquinas de Turing.

3. A tese de Church

"Qualquer procedimento efetivo pode ser realizado por uma máquina de Turing". Esta proposição é conhecida como a *Tese de Church*.

Uma máquina de Turing tem uma descrição finita, pois o conjunto de estados Q e o alfabeto Σ são conjuntos finitos por definição, e portanto a função de transferência δ também é um objeto finito. Cada operação da máquina é claramente efetiva. Assim, examinando as propriedades que exigimos de um procedimento descritas na parte A, fica intuitivamente claro que toda máquina de Turing descreve um procedimento efetivo. O modelo de Turing, porém, é tão simples que à primeira vista pode parecer injustificado supor que qualquer procedimento efetivo possa nele ser implementado. No entanto é exatamente isto que afirma a Tese de Church.

A noção de procedimento efetivo é um conceito informal, e assim a Tese de Church afirma a equivalência de um conceito informal e um conceito formal, não sendo portanto passível de demonstração. Não obstante, existe considerável evidência que suporta esta proposição. De um lado temos uma vasta evidência empírica: cada um dos algoritmos e procedimentos efetivos usados em matemática pode ser

descrito por meio de uma máquina de Turing, ou diretamente, ou através de um formalismo mais conveniente equivalente ao formalismo de Turing. Em particular, pode-se demonstrar que para cada programa em linguagem *LP* existe uma máquina de Turing que calcula a mesma função. Mais ainda, existe um algoritmo que tendo por entrada um programa na linguagem *LP* produz por saída a descrição de uma máquina de Turing correspondente.

Por outro lado, todas as tentativas independentes de formalizar os conceitos de procedimento efetivo e algoritmo se mostraram equivalentes, apesar de haver uma grande variedade de tais formalizações, aparentemente muito dissimilares.

Em virtude da evidência acima, a Tese de Church é geralmente aceita pelos matemáticos.

Supondo que a Tese de Church seja aceita, podemos utilizá-la para dois tipos de aplicações. Às vezes, para economizar tempo e esforço, podemos especificar um procedimento informalmente e, invocando a Tese de Church, concluir que existe um programa formal correspondente. Este tipo de aplicação pode ser evitado, especificando-se formalmente o procedimento em questão.

Existe uma certa analogia entre este tipo de aplicação da Tese de Church e demonstrações em matemática. Em lógica define-se precisamente o conceito de demonstração num sistema formal. Mesmo assim, na maioria dos casos em matemática, usamos métodos informais em nossas demonstrações, subentendendo-se que, se desejado, pode-se transformar as demonstrações informais em demonstrações formais numa teoria formal conveniente.

O segundo tipo de aplicação é porém inevitável e ocorre quando demonstrarmos que não existe um programa formal de uma certa espécie, mas pela Tese de Church afirmarmos que não exista tampouco nenhum procedimento informal do tipo desejado. Usaremos nas seções abaixo ambas as formas de aplicação.

4. Medidas de complexidade de máquinas de Turing

A eficiência de um algoritmo pode ser descrita pela função que para cada entrada dá por resultado o tempo de execução do algoritmo com esta entrada. No nosso modelo, "tempo" corresponde de modo natural ao número de passos tomados pela máquina até parar. Assim,

para cada máquina de Turing determinística M definimos a função $t_M: \Sigma^*_{e/s} \to \mathbb{N} \cup \{\infty\}$ dada por[4]:

(a) Se M com entrada x parar então $t_M(x)$ é o número de passos tomados por M até a sua parada;

(b) Se M com entrada x não parar então $t_M(x) = \infty$.

Para uma máquina de Turing não determinística M a função $t_M: \Sigma^*_{e/s} \to \mathbb{N} \cup \{\infty\}$ é definida por:

(a) Se para alguma sugestão y a máquina M aceitar a entrada x então $t_M(x)$ é o número de passos mínimo dentre todas as computações de M que aceitam x. (Note que podemos ter várias sugestões y com as quais M aceita x, tomando um número maior ou menor de passos conforme a sugestão.);

(b) Se M não aceitar x com nenhuma sugestão, mas parar com alguma sugestão, então $t_M(x)$ é o número de passos mínimo dentre todas as computações de M que rejeitam x;

(c) Se M não parar com nenhuma sugestão então $t_M(x) = \infty$.

É muitas vezes mais conveniente medir a eficiência de um algoritmo em termos do tamanho de sua entrada. Assim, dada uma função $g: \mathbb{N} \to \mathbb{N}$ dizemos que uma máquina de Turing M é *limitada em tempo* por g se $t_M(x) \leq g(|x|)$ para toda entrada x. Seja G uma família de funções $g: \mathbb{N} \to \mathbb{N}$. Um conjunto A é *reconhecível em tempo G* se existir uma máquina de Turing determinística M que reconhece A e é limitada em tempo por algum $g \in G$. Um conjunto A é *aceitável em tempo G* se existir uma máquina de Turing não determinística M que aceita A e é limitada em tempo por algum $g \in G$.

Uma outra medida de complexidade de algoritmos importante é a "memória" utilizada na computação. Isto corresponde naturalmente ao número de células da fita visitadas pela cabeça de leitura/gravação da máquina de Turing durante a computação. Por convenção define-se que o espaço utilizado numa computação que não pára é infinito, mesmo se a máquina visita indefinidamente um trecho finito da fita. Deixamos ao leitor, como exercício, as definições da função análoga a $t_M(x)$ para a medida de complexidade de espaço, bem como os conceitos de conjunto reconhecível e aceitável em espaço G.

Os limites superiores de tempo e espaço para máquinas de Turing podem, sem perda de generalidade, ser expressos a menos de uma

(4) $n < \infty$ para todo $n \in \mathbb{N}$.

constante multiplicativa, conforme o Exercício 11. Para isto freqüentemente é muito conveniente utilizar-se da notação O definida do seguinte modo: dadas as funções f, g: $\mathbb{N} \to \mathbb{N}$ dizemos que $f \in O(g)$, leia-se f é da *ordem* de g, se existir uma constante $K > 0$ e $n_0 \in \mathbb{N}$ tal que $f(n) \leq K \cdot g(n)$ para todo $n \geq n_0$.

EXERCÍCIOS

1. Mostre que o número de operações efetuadas pelo procedimento *mdc* é $T_A(m, n) = 7\,(min\,(m, n) - mdc\,(m, n)) + 8$. Conte cada soma (+), subtração (−), atribuição (←), cálculo de resto, comparação, $\vee$, etc. como uma operação. Assim, por exemplo, cada execução de **se** $m < n$ **então** $d \leftarrow m$ **senão** $d \leftarrow n$ conta como duas operações.
2. Mostre que o número de operações efetuadas pelo Algoritmo de Euclides, $T_E(m, n)$, é limitada superiormente por $10\,log_2\,n + 8$. (Sugestão: Mostre que para cada duas execuções do comando repetitivo o valor de y decresce pelo menos para $y/2$.)
3. No texto comparamos os algoritmos *mdc* e de Euclides pelo seu desempenho no pior caso, para cada n fixo. Defina o conceito de *tempo médio* de execução para n fixo, para o algoritmo *mdc*. Demonstre que existe uma função $\overline{t}_A(n)$ que corresponde à sua definição.
4. Estude as propriedades de $\overline{t}_A(n)$ do Exercício 3.
   (a) Determine o valor de $\overline{t}_A(45)$;
   (b) Determine o valor de $\overline{t}_A(p)$ onde p é primo;
   (c) Determine o valor de $\overline{t}_A(p \cdot q)$ onde p e q são primos;
   (d) O que você pode dizer no caso geral?
5. Considere um problema para o qual dispomos de quatro algoritmos de tempos de execução $200n$, $40n\,\lfloor log_2\,n \rfloor$, $10\,n^2$ e 2^n. Para cada algoritmo indique a faixa de valores de n para os quais ele é o melhor dos quatro algoritmos. (Para um número real α, $\lfloor \alpha \rfloor$ denota o maior inteiro contido em α, isto é, o maior inteiro não maior do que α.)
6. Contraste as definições de conjunto reconhecido por uma máquina de Turing determinística, e conjunto aceito por uma máquina de Turing não determinística.

7. Construa uma máquina de Turing determinística que reconhece o subconjunto de $\{0, 1\}^*$:
   (a) $\{xx^R \mid x \in \{0, 1\}^*\}$ onde x^R denota a palavra reversa correspondente a x definida por: $\Lambda^R = \Lambda$, $(x0)^R = 0x^R$, $(x1)^R = 1x^R$ (exemplo: $(01001)^R = 10010$);
   (b) $\{x \mid x$ é a representação binária de um número par$\}$;
   (c) $\{x \mid x$ é a representação binária de um número divisível por 3$\}$;
   (d) $\{1^{n^2} \mid n \in \mathbb{N}\}$.
8. Mostre que as linguagens (b) e (c) do Exercício 7 são reconhecíveis no sentido definido no Capítulo D. II.
9. Mostre que o conjunto (d) do Exercício 7 é reconhecível em espaço $|x|$, onde x é a entrada da máquina de Turing.
10. No texto foram dadas duas definições para $t_M(x)$, uma para M determinística e uma para M não determinística. Mostre que se M for uma máquina determinística então $t_M(x)$ calculada por ambas as definições dá a mesma função. Considere uma máquina determinística como uma máquina não determinística que ignora a sua fita de sugestões.
11. (a) Mostre que se L é reconhecido por uma máquina de Turing limitada em tempo por $t(n)$, onde n é o comprimento da entrada, e $\lim_{n\to\infty} t(n)/n^2 = \infty$ então para qualquer constante $c > 0$ o conjunto L é reconhecido por uma máquina de Turing de fita **única limitada em tempo por** $max(n, c, \cdot t(n))$.
    (Sugestão: construa uma nova máquina de Turing que reconheça L cujo alfabeto tem mais símbolos do que o alfabeto da máquina original. Note que basta demonstrar a proposição para $c = 1/2$.)
    (b) Mostre que se L é reconhecido por uma máquina de Turing limitada em espaço por $e(n)$ então para qualquer constante $c > 0$ o conjunto L é reconhecido por uma máquina de Turing limitada em espaço por $max(n, c \cdot e(n))$.
12. Seja $f: \mathbb{N} \to \mathbb{N}$ uma função. Diremos que f é computável por uma máquina de Turing se existir uma máquina de Turing determinística que com entrada 1^n produz a saída $1^{f(n)}$. Demonstre que
    (a) $f(n) = n + 1$ é computável;
    (b) se f e g são computáveis então $f \circ g$ também é computável;
    (c) se f é uma função computável e $K \in \mathbb{N}$ então a função g definida por

$$\begin{cases} g(0) = K \\ g(n) = f^n(K) \text{ para } n \geq 1, \end{cases}$$

também é computável.

13. (a) Seja $f(n) = 15n^2 + 7n \lfloor log_2 n \rfloor + 5n + 3$. Mostre que $f \in O(n^2)$ mas $f \notin O(n \lfloor log_2 n \rfloor)$.
    (b) Mostre que $\lfloor log_a n \rfloor \in O(\lfloor log_b n \rfloor)$ para quaisquer $a, b > 1$.
14. Mostre que
    (a) toda função constante pertence a $O(1)$;
    (b) existem funções f e g tais que $g \in O(f)$ mas $f \notin O(g)$.
    (c) Seja $O(f) + O(g) = \{h: \mathbb{N} \to \mathbb{N} \mid h = f' + g' \text{ e } f' \in O(f) \text{ e } g' \in O(g)\}$. Então $O(f+g) = O(f) + O(g) = O(max(f, g))$.
15. Demonstre que a máquina de Turing construída na Seção 2 reconhece o conjunto $\{0^n1^n \mid n \geq 1\}$.

NOTAS BIBLIOGRÁFICAS

O modelo de Turing foi definido em [119]. A Tese de Church exprime a convicção de que este é um modelo correto de procedimento efetivo, e coroa os esforços desenvolvidos na primeira metade deste século em lógica matemática para isolar este conceito. [39] é uma referência sobre a evolução histórica de artefatos de computação mecânica. Algumas destas contribuições já anteviam a importância da complexidade dos cálculos, muito antes do aparecimento dos primeiros computadores digitais.

A análise de algoritmos específicos, em muitos casos antecedeu o estudo mais abstrato e geral de questões de complexidade computacional, e se desenvolveu em paralelo com esta teoria. Muito do que se conhece sobre técnicas de análise de algoritmos pode ser encontrado nos livros de Knuth [60, 61, 62].

A primeira tentativa para medir a dificuldade de computação de um ponto de vista axiomático foi feito por Rabin [97]. Mais tarde o trabalho de Hartmanis e Stearns em [54, 55] estabeleceu alguns resultados fundamentais da teoria, e constitui o primeiro estudo sistemático de uma medida específica de complexidade. Desde então, o campo de complexidade tornou-se uma das áreas de estudo autônomo de Teoria da Computação, e vem se desenvolvendo vigorosamente.

Uma formalização elegante e abstrata do conceito de medida de complexidade é a teoria axiomática de Blum [5].

CAPÍTULO II

PROBLEMAS INDECIDÍVEIS

1. Introdução

Em 1900, no Segundo Congresso Internacional de Matemática realizado em Paris, David Hilbert propôs uma lista de problemas de envergadura em Matemática, cuja solução "desafiaria futuras gerações de matemáticos". Vários problemas desta famosa lista foram desde então resolvidos. Alguns, porém, resistiram até hoje a todas as tentativas de solução, e permanecem ainda em aberto.

O décimo problema da lista de Hilbert era de enunciado bastante simples: descreva um algoritmo que determina se uma dada equação diofantina, a diversas variáveis, isto é, uma equação do tipo $P(u_1, u_2, \ldots, u_n) = 0$, onde P é um polinômio com coeficientes inteiros, tem uma solução no anel dos inteiros. Em linguagem ligada à computação: escreva um programa que tem por entrada uma equação diofantina, e que dá por saída *sim* ou *não*, conforme a equação dada tenha ou não soluções inteiras. O programa não precisa encontrar a solução da equação. Precisa apenas determinar se tal solução existe. Por exemplo, se a entrada for $x^2 - y^2 + xy - 11 = 0$, então o programa, após um número finito de passos, deve responder *sim*, pois esta equação tem solução inteira ($x = 3$, $y = 2$ é uma solução). Já se a entrada ao programa fosse $x^2 + y^2 + xy - 11 = 0$, então a resposta deveria ser *não*, pois esta equação não possui soluções inteiras.

A simplicidade do enunciado do décimo problema de Hilbert é ilusória, pois apesar do intenso esforço para resolvê-lo, não foi senão 70 anos depois que a solução foi finalmente encontrada, por Matijasevič, um matemático russo de apenas 22 anos na época. A solução de Matijasevič é bastante complexa, dependendo tanto de resultados gerais de Teoria dos Números, conhecidos há centenas de anos, como do trabalho anterior, específico sobre o problema de Hilbert, de três americanos, Martin Davis, Julia Robinson e Hilary Putnam, que por sua vez baseia-se em certos resultados fundamentais sobre lógica e algoritmos descobertos na década de 30 deste século por Kurt Gödel, Alan Turing, Emil Post, Alonzo Church e Stephen Kleene.

Embora os pormenores da solução sejam complexos, o resultado de todo este trabalho pode ser descrito com mais facilidade ainda do que o próprio problema: tal algoritmo não existe. O décimo problema de Hilbert é, portanto, um exemplo de um problema indecidível por meio de um algoritmo. Tais problemas de tipo *sim* ou *não* são denominados problemas *recursivamente* (isto é, por meio de algoritmos) *indecidíveis*, ou simplesmente *problemas indecidíveis*.

Neste capítulo exibiremos alguns problemas naturais de computação que são indecidíveis. Na verdade, as técnicas que exporemos podem ser usadas para demonstrar limitações inerentes a qualquer sistema formal mecanizável. E uma vez demonstrado que um problema é indecidível, pode-se freqüentemente demonstrar que outros problemas também são indecidíveis, usando uma técnica de redução de um problema ao outro que estudaremos na Seção 5 deste capítulo. Foi por uma redução deste tipo, embora muito mais complexa do que os exemplos que veremos, que se provou que o décimo problema de Hilbert é indecidível. No fim do capítulo daremos uma lista de outros problemas indecidíveis em lógica, álgebra e computação.

2. Um problema indecidível de computação

Na Parte A deste livro foram apresentados os conceitos de procedimento efetivo, ou simplesmente procedimento, e de algoritmo. Um algoritmo, como vimos, é um procedimento efetivo que pára após um número finito de passos, para todos os valores possíveis de seus argumentos. Vimos, por exemplo, uma demonstração de que o Algoritmo de Euclides sempre pára, isto é, que este procedimento é de fato um algoritmo.

Obviamente, a questão de determinar se um dado procedimento é ou não um algoritmo é importante. Em geral, porém, esta questão é difícil de ser respondida como indica o seguinte exemplo, apresentado na linguagem *LP* definida na Parte A do livro, e que é a solução do Exercício A.I.3.

procedimento *Fermat*( ):
  início
    $x, y, z, n \leftarrow 1, 1, 1, 3$;
    repita $x, y, z, n \leftarrow$ *sucessor* (x, y, z, n) **até que** $x^n + y^n = z^n$;
    devolva 1
  fim

```
procedimento sucessor(x, y, z, n) : {seleciona a próxima quádrupla}
  início
    se n > 3 então devolva x, y, z + 1, n − 1
             senão {n = 3}
    se z > 1 então devolva x, y + 1, 1, z + 1
             senão {z = 1, n = 3}
    se y > 1 então devolva x + 1, 1, 1, y + 1
             senão {y = 1, z = 1, n = 3}
    devolva 1, 1, 1, x + 3
  fim
```

Não é difícil verificar que este procedimento pára se e somente se existirem inteiros positivos x, y, z e um inteiro $n \geq 3$ tais que $x^n + y^n = z^n$. (Verifique. Sugestão: verifique primeiro que sucessor enumera todas as quádruplas (x, y, z, n) válidas.) Não se sabe se tais números existem, embora Fermat tivesse anotado na margem de seu exemplar do livro de Diofanto que tinha achado "uma prova maravilhosa" de que tais números não existem, mas a prova infelizmente era demasiadamente longa para caber no reduzido espaço da margem de seu livro. A prova de que Fermat falava nunca foi encontrada, e se de fato era uma prova correta, deve ter sido realmente maravilhosa, pois em 300 anos ninguém conseguiu demonstrar esta proposição, geralmente conhecida como o Último "Teorema" de Fermat. De qualquer modo, deve ser difícil decidir se o procedimento acima é ou não um algoritmo, pois decidir esta questão, como vimos, é equivalente a resolver um problema de Matemática, em aberto há três centenas de anos!

Seria ótimo se pudessemos desenvolver um algoritmo que decidisse se um procedimento arbitrário dado é ou não um algoritmo. Se tivéssemos um tal algoritmo poderíamos, pelo menos em princípio, programar um computador que após um tempo finito nos daria a resposta se o procedimento *Fermat* acima é um algoritmo e, portanto, se o Último "Teorema" de Fermat é ou não verdadeiro. Na Seção 5, no entanto, mostraremos que esse problema também é indecidível, e portanto tal algoritmo não existe.

3. Conceitos básicos

Para poder provar que um problema é indecidível precisaremos formalizar algumas noções.

Um *predicado* θ sobre um conjunto X é uma função $\theta: X \to \{0, 1\}$. Se $\theta(x) = 1$ para $x \in X$, então dizemos que x *satisfaz* θ, ou que θ é *verdadeiro* em x. Caso contrário, isto é, se $\theta(x) = 0$, então dizemos que x *não satisfaz* θ, ou que θ é *falso* em x.

Um predicado θ sobre X é *decidível* se existir um algoritmo F que calcula θ, isto é, se F com argumento $x \in X$ pára, e $F(x) = \theta(x)$. Neste caso dizemos que F é um *processo efetivo de decisão* para θ. O predicado θ é *indecidível* se não for decidível.

Certamente, qualquer coleção de perguntas que admitem por resposta *sim* ou *não*, isto é, qualquer problema *sim* ou *não*, pode ser representado por meio de um predicado. Um membro desta coleção é uma *instância* do problema correspondente. Um problema é *decidível* se o predicado que o representa for decidível.

Um programa na linguagem *LP* é uma palavra sobre um alfabeto Γ conveniente, que inclui todos os símbolos necessários para representar programas em *LP*. O alfabeto Γ portanto inclui, entre outros, as letras maiúsculas e minúsculas A, a, B, b, ..., Z, z, numerais 0, 1, ..., 9, sinais de pontuação „ :, ;„ sinais que denotam operações e símbolos $+$, $-$, $\times$, $\div$, $>$, $=$, $\leftarrow$, **procedimento**, **início**, **fim**, **repita**, **até que**, **faça**, **devolva**, etc. Naturalmente nem todas as seqüências de símbolos de Γ, isto é, nem todas as palavras de Γ^*, descrevem programas válidos. Existe porém um algoritmo que decide se uma palavra de Γ^* representa ou não um programa válido em *LP*. (Vimos parte deste algoritmo descrito em *LP* no Exemplo A.I.7.)

Cada programa em *LP* tem um *nome* que o identifica. Como sabemos, este nome é a cadeia de caracteres que aparece logo após o símbolo **procedimento**.

Um programa, quando executado, pode aceitar dados de entrada de vários tipos, como números inteiros, matrizes, textos, etc. Cada um destes objetos pode ser descrito por uma palavra sobre Γ. Assim, podemos supor que os dados d para um programa são codificados numa única palavra de Γ^*, que chamaremos de *texto de dados* do programa e que denotaremos por d_t. Deve-se especificar precisamente, o que por motivos didáticos não foi feito no Capítulo A.I, o que acontece se o texto de dados for inadequado, por conter um número incorreto de argumentos, ou argumentos de tipo errado, ou argumentos codificados incorretamente. Os pormenores destas especificações não nos interessam tampouco, pois qualquer convenção razoável é adequada para os nossos propósitos. O que suporemos simplesmente é que esteja precisamente especificada a seqüência de ações de um pro-

grama qualquer quando executado com qualquer texto de dados. Por exemplo se o programa *sucessor* descrito na Seção 2 for executado como um texto de dados que codifica cinco inteiros em vez dos quatro necessários, então o último pode ser ignorado; se o texto de dados não codifica pelo menos quatro inteiros então a execução do programa pode ser interrompida imediatamente tendo por resultado um símbolo especial de Γ denotando dados incorretos, e assim por diante.

4. O resultado principal

Estamos agora em condições de enunciar o principal teorema deste capítulo.

TEOREMA 1. *Seja θ o seguinte predicado sobre o conjunto $\Gamma^* \times \Gamma^*$ de pares de palavras de Γ^*:*

(i) $\theta(P_t, d_t) = 1$ *se P_t for um programa em LP que, executado com texto de dados d_t, pára.*

(ii) $\theta(P_t, d_t) = 0$ *em caso contrário.*

Nestas condições não existe um programa que calcula o predicado θ. Em outras palavras, θ é indecidível[1].

É importante entender que o teorema não afirma que dado um particular programa e seu texto de dados de entrada, seja impossível determinar se o programa pára ou não com estes dados. Na Parte A já vimos uma prova de que o Algoritmo de Euclides sempre pára para quaisquer entradas m e n de inteiros positivos, e portanto, às vezes é perfeitamente possível determinar o valor de $\theta(P_t, d_t)$. O que o teorema afirma é que não existe um algoritmo que decida esta questão para qualquer programa P_t e texto de dados de entrada d_t.

Podemos agora passar à prova do Teorema 1.

Demonstração. Por contradição, suponhamos que existe um programa, de nome *Decide*, que calcula o predicado θ. Considere então o programa seguinte:

---

(1) O problema de decidir se um dado procedimento P pára com dados de entrada d é às vezes chamado de "o problema da parada". Uma outra forma de enunciar este teorema é, portanto, dizer que o problema da parada é indecidível. Note que o teorema afirma que não existe um programa em *LP* com as características desejadas. Pela Tese de Church isto "significa" que não existe um algoritmo para decidir o problema da parada.

procedimento *Confunde* (A, B):
se *Decide* $(A, B) = 1$ **então repita** $x \leftarrow x$ **até que** $1 = 0$
{$A(B)$ pára – *Confunde* não pára}
senão devolva 1
{$A(B)$ não pára – *Confunde* pára}

Analisemos a execução de *Confunde* com dados de entrada P_t e d_t, supondo que P_t é um programa de nome P. Claramente, se *Decide* $(P_t, d_t) = 1$, isto é, pela condição (i), se P pára com dados d, então *Confunde* (P_t, d_t) nunca pára, pois repetirá incessantemente a execução do comando $x \leftarrow x$, uma vez que a condição $1 = 0$ é sempre falsa. Por outro lado, se *Decide* $(P_t, d_t) = 0$, isto é se P não pára com dados d, então *Confunde* (P_t, d_t) pára com resposta 1. Portanto, para um programa P_t, de nome P, com texto de dados d_t:

$$\textit{Confunde}\ (P_t, d_t) = \begin{cases} 1 \text{ se } P \text{ não pára com os dados } d \text{ descritos por } d_t \\ \\ \text{não pára, se } P \text{ pára com estes dados.} \end{cases}$$

Com o auxílio de *Confunde* definimos o seguinte programa:

procedimento *Diagonal* (A): **devolva** *Confunde* (A, A).

Analisemos agora a execução de *Diagonal*, supondo que A seja um programa. *Diagonal* chama *Confunde* (A, A), e portanto *Diagonal* pára com resposta 1 se o programa A não pára com dados de entrada que é o seu próprio texto. Caso contrário *Diagonal* não pára.

Agora estamos em condições de provar a contradição que é acarretada pela hipótese da existência de *Decide*. De fato, qual é o resultado de *Diagonal* $(Diagonal_t)$? Se *Diagonal* tendo seu próprio texto por dados de entrada parar, então pela discussão acima de como *Diagonal* funciona, sabemos que *Diagonal* $(Diagonal_t)$ não pára. Se, por outro lado, *Diagonal* tendo seu próprio texto por dados de entrada não parar, então da mesma discussão sabemos que *Diagonal* $(Diagonal_t)$ pára. Em suma:

Diagonal $(Diagonal_t)$ pára sse *Diagonal* $(Diagonal_t)$ não pára.

Isto é uma contradição e portanto somos forçados a concluir que *Decide* não existe. ■

A prova deste teorema é relativamente simples e curta, mas é pouco intuitiva. Vale a pena analisarmos um pouco a estratégia da demonstração. Queremos demonstrar que nenhum programa com as

características de *Decide* pode existir. Como não sabemos como tal programa poderia ser, agiremos contra um "adversário" que alega (falsamente) estar de posse de um tal programa. Nós então lhe solicitamos o texto descrevendo *Decide*, e de sua posse construímos *Confunde* e *Diagonal*. *Diagonal* opera sobre um programa P_t, de nome P, e consultando *Decide* sobre o que P faz quando apresentado com um texto P_t, faz exatamente o contrário. A contradição surge então se *Diagonal* é apresentado com o seu próprio texto $Diagonal_t$, pois neste caso se *Decide* diz que *Diagonal* pára então *Diagonal* não pára, e se *Decide* diz que *Diagonal* não pára então *Diagonal* pára. Deste modo mostramos ao nosso adversário que *Decide* não pode dar a resposta correta com entrada $(Diagonal_t, Diagonal_t)$[2]. Em outras palavras, *Diagonal* é o procedimento que, tendo seu texto por entrada, faz exatamente o contrário do que *Decide* afirma que *Diagonal* deveria fazer.

5. Redutibilidades

Existe uma grande variedade de problemas indecidíveis. Embora para muitos destes problemas uma prova direta, como a que vimos na seção anterior, seja possível, é muitas vezes mais conveniente utilizar-se uma prova indireta empregando o conceito de redutibilidade. Intuitivamente, um predicado θ_1 é redutível a um predicado θ_2, se tivermos um método efetivo para decidir θ_1 desde que nos seja fornecido o valor de θ_2 sempre que for necessário. Este conceito pode ser formalizado de diversas maneiras, com diferentes graus de generalidade. Daremos abaixo uma destas formalizações.

Um predicado $\theta_i : X_i \to \{0, 1\}$ é *m-redutível* a um predicado $\theta_2 : X_2 \to \{0, 1\}$ sse existir um algoritmo f que para qualquer entrada $x_1 \in X_1$ produz uma saída $x_2 \in X_2$ tal que $\theta_1(x_1) = 1$ sse $\theta_2(x_2) = 1$ (e $\theta_1(x_1) = 0$ sse $\theta_2(x_2) = 0$). Se θ_1 é m-redutível a θ_2 então escrevemos $\theta_1 \leq_m \theta_2$.

Uma das aplicações de m-redutibilidade é dada pelo seguinte teorema:

TEOREMA 2. *Se* θ_1 *for um predicado indecidível e* $\theta_1 \leq_m \theta_2$ *então* θ_2 *é também indecidível.*

---

(2) Este método de demonstração é essencialmente a técnica de *diagonalização* concebida por Cantor em suas investigações sobre a Teoria dos Conjuntos.

Demonstração. Suponha que θ_2 é decidível e $\theta_1 \leq_m \theta_2$ via f. Seja g o algoritmo que calcula θ_2, e seja h o procedimento:

procedimento $h(x)$: **devolva** $g(f(x))$

Este procedimento sempre pára, pois tanto f como g sempre param. Mas por hipótese $\theta_1(x) = 1$ sse $\theta_2(f(x)) = 1$ e como g calcula θ_2, $\theta_2(f(x)) = 1$ sse $g(f(x)) = 1$. Portanto h é um algoritmo que calcula θ_1 o que é uma contradição, pois θ_1 é indecidível. ■

A seguir aplicaremos este teorema para provar que alguns outros problemas são indecidíveis.

COROLÁRIO 1 (Problema da parada uniforme). *O problema de determinar se um procedimento é ou não um algoritmo é indecidível. Mais precisamente, não existe um programa Pára que tem por entrada um programa P_t, de nome P, e que pare para qualquer entrada P_t, produzindo saída* 1 *se P é um algoritmo (isto é, parar com quaisquer dados de entrada), e* 0 *em caso contrário.*

Demonstração. Mostraremos que o problema da parada é m-redutível ao problema da parada uniforme. Portanto, como o problema da parada é indecidível (Teorema 1), pelo Teorema 2 o problema da parada uniforme também o é.

Queremos, então, construir um algoritmo f que, tendo por entrada um programa P_t e seu texto de dados d_t, dê por saída um programa Q_t, de nome Q, tal que Q com quaisquer dados pára se e somente se P pára em d. A saída de f será então o texto

procedimento $Q(\ )$: $P(d)$
P_t

onde P é o nome de P_t e d são os dados correspondentes a d_t. É óbvio que tal agoritmo f pode ser construído. Pela Tese de Church, portanto, podemos construir um programa para $f(P_t, d_t)$. Note que f manipula o texto P_t dele extraindo P, e d_t extraindo d, e em seguida escreve o texto Q_t acima.

Analisemos agora o funcionamento de Q. O texto P_t é incluído para definir P dentro de Q. A execução de Q consiste simplesmente em chamar P com dados d, e portanto é óbvio que Q pára com quaisquer dados sse P pára com d. ■

COROLÁRIO 2 (Problema da equivalência de programas). *Não existe um algoritmo Eq que decide se dois procedimentos dados P_1 e P_2 são equivalentes; mais precisamente, não existe um programa Eq (P_t, Q_t) tal que Eq pára com quaisquer dados de entrada, e Eq $(P_t, Q_t) = 1$ se os procedimentos P e Q calculam a mesma função e Eq $(P_t, Q_t) = 0$ em caso contrário. Note que P e Q calculam a mesma função se para qualquer entrada ou ambos não param, ou ambos param com a mesma resposta.*

Demonstração.

Eq (P_t, **procedimento** *Diverge* ( ): **repita** $x \leftarrow x$ **até que** $1 = 0$) $= 1$

sse P não pára com nenhuma entrada. Isto mostra que o problema do Exercício 4 é m-redutível ao problema de equivalência de programas, e portanto, pelo mesmo exercício, este problema é indecidível. ■

6. Outros problemas indecidíveis em matemática

Nesta seção daremos alguns exemplos de problemas indecidíveis em vários ramos da Matemática, sem dar a demonstração destes resultados, limitando-nos simplesmente a enunciá-los. No fim do capítulo damos indicações bibliográficas onde as demonstrações podem ser encontradas.

Em geral, as demonstrações destes resultados são por meio de reduções do tipo que vimos na seção anterior, às vezes muito mais complicadas. Já mencionamos um destes problemas na introdução deste capítulo: é o décimo problema de Hilbert. Passemos agora a outros exemplos de problemas indecidíveis.

(a) — O Problema da Correspondência de Post

Dado um alfabeto Σ e um conjunto finito de pares de palavras sobre Σ, $\{(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)\}$, é indecidível se existe uma seqüência finita de índices $i_1, i_2, \ldots, i_n$ tal que $1 \leq i_j \leq k$ e $x_{i_1} x_{i_2} \ldots x_{i_n} = y_{i_1} y_{i_2} \ldots y_{i_n}$.

(b) — Teorias de Primeira Ordem

É indecidível se uma expressão formada com os símbolos 0, 1, $+$, $\cdot$, $=$ conectivos lógicos $\wedge$, $\vee$, $\neg$, variáveis, quantificadores lógicos $\exists$, $\forall$, é um Teorema da Aritmética.

(c) – O Problema das Palavras em Grupos

É indecidível se duas palavras dadas representam o mesmo elemento de um grupo finitamente apresentado.

Estes exemplos mostram a diversidade dos problemas que já foram demonstrados serem indecidíveis. Existem muitos outros problemas em áreas que vão de Topologia a Linguagens Formais, de Biologia Matemática a Álgebra.

Em muitos destes casos, até que o problema foi demonstrado ser indecidível, acreditava-se ser possível construir um algoritmo para decidir o problema. Deste modo uma das "vantagens" de se ter uma demonstração de indecidibilidade, é exatamente em mostrar a futilidade de se continuar a tentar desenvolver um algoritmo para o problema em questão. Note, no entanto, que mesmo que um problema seja indecidível, uma subclasse menos geral do problema pode ser decidível. Pode, portanto, ser interessante identificar uma tal subclasse e desenvolver algoritmos de decisão para a mesma.

EXERCÍCIOS

1. Verifique que a contradição a que chegamos, supondo a existência do algoritmo *Decide* no Teorema 1, desaparece se supusermos apenas que *Decide* é um procedimento, isto é se supusermos que se *Decide* (P_t, d_t) pára, então *Decide* $(P_t, d_t) = 1$ se $P(d)$ pára, e *Decide* $(P_t, d_t) = 0$ se $P(d)$ não pára. *Decide*, porém, pode nunca parar para alguns argumentos.
2. Esboce um procedimento efetivo *Decide* que com dados P_t e d_t pára sempre que P em d parar, e se parar, então dá o resultado 1 se P em d parar, e 0 em caso contrário.
3. Verifique que dado um conjunto X, a relação $\leq_m$ é reflexiva e transitiva sobre o conjunto de predicados sobre X.
4. Prove que o problema de determinar se um procedimento não pára com nenhuma entrada é indecidível.
5. Prove que o problema de determinar se um programa P com dados de entrada d executa um dado comando do programa é indecidível.
6. Prove que o problema de determinar se o valor da função inteira calculada por um programa P é maior do que 5 para alguma entrada é indecidível.

7. Considere a função $f: \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}^+$ de pares ordenados de naturais não-nulos em naturais não-nulos:

$$f(x, y) = \frac{(x+y-1)\,(x+y-2)}{2} + y.$$

(a) Mostre que f é bijetora.

(b) Ache as funções "inversas" $g: \mathbb{N}^+ \to \mathbb{N}^+$ e $h: \mathbb{N}^+ \to \mathbb{N}^+$ tais que $g(f(x, y)) = x$ e $h(f(x, y)) = y$.

(c) Observe que f enumera os pontos de $\mathbb{N}^+ \times \mathbb{N}^+$; o j-ésimo ponto de $\mathbb{N}^+ \times \mathbb{N}^+$ é o único par ordenado (x, y) tal que $f(x, y) = j$, com $x = g(j)$ e $y = h(j)$. Mostre graficamente esta enumeração, colocando nos pontos (x, y) do plano cartesiano o valor de $f(x, y)$. Descreva em termos geométricos a regra pela qual f enumera $\mathbb{N}^+ \times \mathbb{N}^+$.

(d) Generalizando a regra geométrica acima, obtenha uma função t que enumera $\mathbb{N}^{+3}$, isto é, uma função bijetora $t: \mathbb{N}^+ \times \times \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}^+$.

(e) Sejam

$$t_1(x, y, z) = f(x, f(y, z))$$

e

$$t_2(x, y, z) = f(f(x, y), z)$$

Mostre que as funções t_1, t_2 e a função t, dada em (d), são três bijeções distintas de $\mathbb{N}^{+3}$ em $\mathbb{N}^+$.

(f) Construa uma bijeção de $\mathbb{N}^{+k}$ em $\mathbb{N}^+$, para k fixo.

(g) Construa uma bijeção de $\bigcup_{k=1}^{\infty} \mathbb{N}^{+k}$ em $\mathbb{N}^+$.

8. Um conjunto A se diz *recursivamente enumerável* sse ou $A = \varnothing$ ou existe um algoritmo F que calcula uma função sobrejetora $f: \mathbb{N} \to A$.

(a) Mostre que a união e a intersecção de conjuntos recursivamente enumeráveis é recursivamente enumerável.

(b) Mostre que um subconjunto A de X é recursivamente enumerável sse existe um programa P que com uma entrada $x \in X$ pára sse $x \in A$.

9. Um subconjunto A de X se diz *recursivo* sse existe um algoritmo que calcula sua função característica, isto é, quando o problema "$x \in A$?" é decidível.

(a) Mostre que se A é recursivo, então A é recursivamente enumerável.
(b) Mostre que existe um conjunto recursivamente enumerável que não é recursivo.
(c) Seja A um subconjunto dos números naturais. Mostre que se A é infinito e recursivamente enumerável em ordem não decrescente, isto é, se $A = \{f(i) \mid i \in \mathbb{N}\}$ onde f é computável, e $f(i) \leq f(j)$ se $i \leq j$, então A é recursivo.
(d) Mostre que um subconjunto A de um conjunto X é recursivo sse A e $X \backslash A$ são recursivamente enumeráveis.
(e) Exiba um conjunto que não é recursivamente enumerável.

10. Prove que nem o conjunto T de todos os programas em LP que são algoritmos, nem seu complemento em relação a Γ^* (ou em relação ao conjunto de todos os programas em LP) são recursivamente enumeráveis.

11. Sejam A e B subconjuntos de Γ^*. Dizemos que A é *m-redutível* a B, e escrevemos $A \leq_m B$, se o predicado "$x \in A$" é m-redutível ao predicado "$y \in B$".
(a) Mostre que se $A \leq_m B$ e B é recursivo (recursivamente enumerável) então A é recursivo (recursivamente enumerável).
(b) Seja $K = \{P_t \mid P_t \in \Gamma^*, P_t$ é um programa que com dados P_t pára$\}$. Mostre que K é recursivamente enumerável, mas não é recursivo.
(c) Mostre que um subconjunto A de Γ^* é recursivamente enumerável sse $A \leq_m K$.

12. Sejam A e B subconjuntos de Γ^*. Dizemos que A é *m-equivalente* a B e escrevemos $A \equiv_m B$ sse $A \leq_m B$ e $B \leq_m A$.
(a) Mostre que $\equiv_m$ é uma relação de equivalência sobre 2^{Γ^*} e que a relação $\leq_m$ se mantém entre as classes de equivalência (chamadas *graus de indecidibilidade*).
(b) O que você pode dizer sobre os conjuntos T e K dos Exercícios 10 e 11, quanto à relação $\leq_m$?

13. Quais dos subconjuntos de Γ^* abaixo são (i) recursivos, (ii) recursivamente enumeráveis ou (iii) tem complemento recursivamente enumerável?
(a) $\{P_t \mid P_t$ é um programa que pára num conjunto infinito de entradas$\}$
(b) $\{P_t \mid P$ é equivalente a um programa Q, dado$\}$

(c) $\{P_t \mid P$ calcula a função que é identicamente $0\}$
(d) $\{P_t \mid$ o conjunto de entradas para os quais P pára é um conjunto recursivo$\}$.

Em todos os casos, a entrada dos programas é uma palavra de Γ^*.

14. Mostre que dado um programa P_t que calcula uma função não decrescente, o problema de decidir se a imagem de P é finita é indecidível.

15. Mostre que o problema da parada para uma máquina de Turing que dispõe de apenas k células em sua fita de trabalho é decidível.

16. Dizemos que um algoritmo P é *limitado em tempo* por $f(n)$, se para toda entrada x de comprimento n, $P(x)$ pára em no máximo $f(n)$ passos (veja Capítulo B.I).
    (a) Mostre que o problema "dado o algoritmo P, de tempo limitado por n^3, P é de tempo limitado por n^2?" é indecidível.
    (b) Exiba um algoritmo P que calcula uma função f, tal que f é computável em tempo limitado por n^3 mas que não pode ser calculada por nenhum algoritmo de tempo limitado por n^2.
    (c) Mostre que o problema "dados os algoritmos P_1 e P_2, ambos de tempo limitado por n^2, e uma entrada x, $P_1(x) = P_2(x)$?" é decidível, mas o problema "dados os algoritmos P_1 e P_2, ambos de tempo limitado por n^2, P_1 e P_2 calculam a mesma função?" é indecidível.

NOTAS BIBLIOGRÁFICAS

Existem vários textos excelentes que discutem os conceitos tratados neste capítulo. O artigo original de Turing [119] contém uma análise excepcionalmente lúcida do modelo computacional de Turing a partir da noção intuitiva de procedimento efetivo. A mesma discussão pode ser encontrada no livro de Minsky [83]. Davis [18] é um texto clássico em computabilidade e Rogers [101] é um tratado avançado da Teoria das Funções Recursivas, recomendado para um estudo mais aprofundado. Os livros de Minsky [83], Hopcroft e Ullman [56], Manna [77], Simon [113] e Machtey e Young [71] são bons livros texto em que os conceitos deste capítulo são examinados, e o artigo de Borodin [7] é um "survey" bastante completo da área.

A solução do décimo problema de Hilbert é apresentada em [20] e [21]. O problema da correspondência de Post está no artigo [95] e também nos textos [83], [56]. A indecidibilidade da aritmética foi o

primeiro resultado na área e deve-se a Gödel [38] (veja também o texto de Shoenfield[111]). O problema das palavras em grupos é um resultado de Novikov [89] e também está no texto [111]. [101] e [112] contêm uma exposição clara da teoria dos graus de indecidibilidade. Uma coleção dos artigos pioneiros mais importantes nesta área contendo os artigos de Gödel, Post e Turing mencionados acima, além de outros trabalhos pioneiros, pode ser encontrada em [19]. O Teorema 1 é baseado no trabalho de Turing [119].

CAPÍTULO III

PRODUTO EFICIENTE DE MATRIZES

1. Introdução

Quantas multiplicações são necessárias para obter o produto de duas matrizes?

Para responder a esta pergunta devemos precisar os métodos admissíveis para o cálculo do produto. Por exemplo, se os elementos das matrizes forem números reais positivos, e se permitirmos as operações *logaritmo* e *exponencial*, então nenhuma multiplicação é necessária, já que

$$a \cdot b = \textit{exponencial}\,(\textit{logaritmo}\,(a) + \textit{logaritmo}\,(b)).$$

Utilizando a identidade acima podemos eliminar todas as multiplicações para achar o produto de duas matrizes de números reais e positivos.

Suporemos então, que os elementos das matrizes a serem multiplicadas pertencem a um anel com unidade A. Assim, as únicas operações permitidas são as operações do anel, além de testes de igualdade e desigualdade. Note que não supusemos que o anel é comutativo, isto é, $a \cdot b$ pode não ser igual a $b \cdot a$[1]. A nossa pergunta, portanto, fica: dadas duas matrizes $n \times n$ X e Y sobre A, quantas operações de A serão necessárias para calcular o produto $X \cdot Y$?

---

(1) Lembramos que as operações $+$ e $\cdot$ gozam das propriedades:

1. $(A, +)$ é um *grupo abeliano*, isto é,
   (a) $a + (b + c) = (a + b) + c$ para todo $a, b, c \in A$;
   (b) $a + b = b + a$ para todo $a, b \in A$;
   (c) Existe um elemento $0 \in A$ tal que $a + 0 = a$ para todo $a \in A$;
   (d) Para cada $a \in A$, existe um elemento $(-a) \in A$, chamado de o *simétrico* de a, tal que $a + (-a) = 0$;
2. $(A, \cdot)$ é um *monóide*, isto é,
   (e) $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ para todo $a, b, c \in A$;
   (f) Existe um elemento $1 \in A$, $1 \neq 0$, tal que $a \cdot 1 = 1 \cdot a = a$ para todo $a \in A$;
3. A operação $\cdot$ *distribui* sobre $+$, isto é, para todo a, b, $c \in A$
   (g) $((a + b) \cdot c) = (a \cdot c) + (b \cdot c)$;
   (h) $(c \cdot (a + b)) = (c \cdot a) + (c \cdot b)$.

Esta pergunta, como vimos no Capítulo I, é típica dos problemas de complexidade de computação, no sentido de pedir a determinação da quantidade de recursos computacionais, (operações • no caso), necessárias para executar uma certa tarefa. O algoritmo clássico de produto de matrizes $n \times n$ nos fornece um limite superior de n^3 operações. Com efeito, se

$$Z = [z_{ij}] = X \cdot Y, \text{ onde } X = [x_{ij}] \text{ e } Y = [y_{ij}]$$

então

$$z_{ij} = \sum_{k=1}^{n} x_{ik} \cdot y_{kj}, \quad 1 \leq i, j \leq n,$$

onde Σ denota a somatória no anel A. Portanto, para encontrar z_{ij} utilizamos uma multiplicação para cada valor de k, e assim, são utilizadas n multiplicações neste cálculo. Como Z possui n^2 elementos, utilizamos $n \cdot n^2 = n^3$ multiplicações no cálculo de Z. O número de somas, pelo mesmo raciocínio, será $n - 1$ para o cálculo de cada z_{ij}, e portanto,

$$n^2 \cdot (n - 1) = n^3 - n^2$$

para a matriz Z toda.

Intuitivamente, parece à primeira vista que o algoritmo acima é ótimo, e assim nada nos restaria fazer para melhorá-lo, exceto demonstrar que de fato são necessárias n^3 multiplicações para determinar Z. Como veremos entretanto, nossa intuição nos engana neste caso, pois, surpreendentemente, o algoritmo clássico pode ser consideravelmente melhorado.

2. O algoritmo de Strassen

Para achar o produto de duas matrizes 1×1, certamente precisaremos de uma multiplicação (vide Exercício 8). Para matrizes 1×1, portanto, n^3 multiplicações são necessárias e suficientes. Já para matrizes 2×2 não se consegiu provar o limite inferior $n^3 = 2^3 = 8$ multiplicações. Isto não é de se admirar, pois em 1969 Strassen observou que é possível multiplicar matrizes 2×2 usando apenas 7 mul-

tiplicações, com o algoritmo abaixo (expresso na linguagem *LP*, definida na Parte *A*, onde $+$, $-$ e $\cdot$ denotam as respectivas operações no anel *A*):

procedimento *Prodmat2*(X, Y):
início
$p_1 \leftarrow (x_{11} + x_{22}) \cdot (y_{11} + y_{22})$;
$p_2 \leftarrow (x_{21} + x_{22}) \cdot y_{11}$;
$p_3 \leftarrow x_{11} \cdot (y_{12} - y_{22})$;
$p_4 \leftarrow x_{22} \cdot (y_{21} - y_{11})$;
$p_5 \leftarrow (x_{11} + x_{12}) \cdot y_{22}$;
$p_6 \leftarrow (x_{21} - x_{11}) \cdot (y_{11} + y_{12})$;
$p_7 \leftarrow (x_{12} - x_{22}) \cdot (y_{21} + y_{22})$;
$z_{11} \leftarrow p_1 + p_4 - p_5 + p_7$;
$z_{12} \leftarrow p_3 + p_5$;
$z_{21} \leftarrow p_2 + p_4$;
$z_{22} \leftarrow p_1 + p_3 - p_2 + p_6$;
devolva Z $\{Z = [z_{ij}]\}$
fim

Com as técnicas recursivas que foram vistas na Parte *A*, podemos utilizar o algoritmo *Prodmat2* para obter um método mais eficiente que o algoritmo clássico, para o produto de duas matrizes quadradas $n \times n$ X e Y. Inicialmente vamos supor $n = 2^k$, para algum $k \geq 2$. Mais tarde mostraremos como eliminar esta restrição. Se $n = 2^k$, podemos subdividir as matrizes X e Y, cortando-as em quatro partes de mesma dimensão:

$$X = \left[\begin{array}{c|c} X_{11} & X_{12} \\ \hline X_{21} & X_{22} \end{array}\right] \quad \text{e} \quad Y = \left[\begin{array}{c|c} Y_{12} & Y_{12} \\ \hline Y_{21} & Y_{22} \end{array}\right].$$

Assim X_{ij} e Y_{ij} são por sua vez matrizes quadradas $n/2 \times n/2$ chamadas *blocos*. Por exemplo, X_{11} é a matriz formada pelas $n/2$ primeiras linhas e $n/2$ primeiras colunas de X, enquanto que X_{12} é a matriz formadas pelas $n/2$ primeiras linhas e $n/2$ últimas colunas de X, e assim por diante. É um fato bem conhecido da Álgebra Linear que o produto de matrizes pode ser feito por blocos, isto é

$$Z = X \cdot Y = \left[\begin{array}{c|c} Z_{11} & Z_{12} \\ \hline Z_{21} & Z_{22} \end{array}\right], \text{ onde } Z_{ij} = X_{i1} \cdot Y_{1j} + X_{i2} \cdot Y_{2j}$$

e as multiplicações e somas indicadas são agora sobre o anel das matrizes $n/2 \times n/2$. Em outras palavras, podemos efetuar o produto $X \cdot Y$ como se X e Y fossem matrizes 2×2 com elementos tomados do anel das matrizes $n/2 \times n/2$, que reinterpretado como uma matriz $n \times n$ sobre o anel A, coincide com o produto de matrizes $n \times n$ $X \cdot Y$. Por exemplo, para $n = 4$ considere

$$X = \left[\begin{array}{cc|cc} 1 & 0 & 2 & 1 \\ 2 & 1 & -1 & 0 \\ \hline 2 & 0 & 0 & 1 \\ 0 & 3 & 1 & 0 \end{array}\right] \text{ e } Y = \left[\begin{array}{cc|cc} 0 & 1 & 2 & 0 \\ 1 & 2 & -1 & 2 \\ \hline 1 & 1 & 0 & 0 \\ 2 & 0 & 0 & -2 \end{array}\right]$$

Portanto,

$$Z = X \cdot Y = \left[\begin{array}{cc|cc} 4 & 3 & 2 & -2 \\ 0 & 3 & 3 & 2 \\ \hline 2 & 2 & 4 & -2 \\ 4 & 7 & -3 & 6 \end{array}\right].$$

Por outro lado,

$$X_{11} = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}; X_{12} = \begin{bmatrix} 2 & 1 \\ -1 & 0 \end{bmatrix}; Y_{11} = \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix}; Y_{12} = \begin{bmatrix} 2 & 0 \\ -1 & 2 \end{bmatrix}$$

$$X_{21} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}; X_{22} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}; Y_{21} = \begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix}; Y_{22} = \begin{bmatrix} 0 & 0 \\ 0 & -2 \end{bmatrix}$$

Assim,

$$Z_{11} = X_{11} \cdot Y_{11} + X_{12} \cdot Y_{21} = \begin{bmatrix} 0 & 1 \\ 1 & 4 \end{bmatrix} + \begin{bmatrix} 4 & 2 \\ -1 & -1 \end{bmatrix} = \begin{bmatrix} 4 & 3 \\ 0 & 3 \end{bmatrix};$$

$$Z_{12} = X_{11} \cdot Y_{12} + X_{12} \cdot Y_{22} = \begin{bmatrix} 2 & 0 \\ 3 & 2 \end{bmatrix} + \begin{bmatrix} 0 & -2 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 2 & -2 \\ 3 & 2 \end{bmatrix};$$

$$Z_{21} = X_{21} \cdot Y_{11} + X_{22} \cdot Y_{21} = \begin{bmatrix} 0 & 2 \\ 3 & 6 \end{bmatrix} + \begin{bmatrix} 2 & 0 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 4 & 7 \end{bmatrix};$$

$$Z_{22} = X_{21} \cdot Y_{12} + X_{22} \cdot Y_{22} = \begin{bmatrix} 4 & 0 \\ -3 & 6 \end{bmatrix} + \begin{bmatrix} 0 & -2 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 4 & -2 \\ -3 & 6 \end{bmatrix}.$$

A idéia do algoritmo para calcular o produto de X e Y é subdividir as matrizes em blocos $n/2 \times n/2$, e em seguida aplicar a rotina *Prodmat*2, com X e Y consideradas como matrizes 2×2 sobre o anel das matrizes $n/2 \times n/2$. Os produtos e somas de *Prodmat*2 são portanto produtos e somas de matrizes $n/2 \times n/2$. Para efetuar cada um destes 7 produtos de matrizes, chamamos o algoritmo recursivamente, agora com argumentos menores $n/2 \times n/2$. Damos abaixo a descrição precisa do algoritmo em *LP*. A notação $X[i : j, k : \ell]$ significará o bloco de X formado pelas linhas de i a j, e as colunas de k a ℓ, inclusive.

procedimento *Prodmat*(X, Y, n): {Produto de matrizes $n \times n$, $n = 2^k$}
início

se $n = 2$ **então devolva** *Prodmat*2(X, Y) **senão** $m \leftarrow n/2$;
$X_{11} \leftarrow X[1 : m, 1 : m]$; $Y_{11} \leftarrow Y[1 : m, 1 : m]$;
$X_{12} \leftarrow X[1 : m, m + 1 : n]$; $Y_{12} \leftarrow Y[1 : m, m + 1 : n]$;
$X_{21} \leftarrow X[m + 1 : n, 1 : m]$; $Y_{21} \leftarrow Y[m + 1 : n, 1 : m]$;
$X_{22} \leftarrow X[m + 1 : n, m + 1 : n]$; $Y_{22} \leftarrow Y[m + 1 : n, m + 1 : n]$;
$P_1 \leftarrow$ *Prodmat*$(S(X_{11}, X_{22}, m, \text{“+”}), S(Y_{11}, Y_{22}, m, \text{“+”}), m)$;
$\{P_1 = (X_{11} + X_{22}) \cdot (Y_{11} + Y_{22})\}$

$P_2 \leftarrow Prodmat(S(X_{21}, X_{22}, m, \text{“+”}), Y_{11}, m);$
$\{P_2 = (X_{21} + X_{22}) \cdot Y_{11}\}$
$P_3 \leftarrow Prodmat(X_{11}, S(Y_{12}, Y_{22}, m, \text{“−”}), m);$
$\{P_3 = X_{11} \cdot (Y_{12} - Y_{22})\}$
$P_4 \leftarrow Prodmat(X_{22}, S(Y_{21}, Y_{11}, m, \text{“−”}), m);$
$\{P_4 = X_{22} \cdot (Y_{21} - Y_{11})\}$
$P_5 \leftarrow Prodmat(S(X_{11}, X_{12}, m, \text{“+”}), Y_{22}, m);$
$\{P_5 = (X_{11} + X_{12}) \cdot Y_{22}\}$
$P_6 \leftarrow Prodmat(S(X_{21}, X_{11}, m, \text{“−”}), S(Y_{11}, Y_{12}, m, \text{“+”}), m);$
$\{P_6 = (X_{21} - X_{11}) \cdot (Y_{11} + Y_{12})\}$
$P_7 \leftarrow Prodmat(S(X_{12}, X_{22}, m, \text{“−”}), S(Y_{21}, Y_{22}, m, \text{“+”}), m);$
$\{P_7 = (X_{12} - X_{22}) \cdot (Y_{21} + Y_{22})\}$
$Z_{11} \leftarrow S(S(S(P_1, P_4, m, \text{“+”}), P_5, m, \text{“−”}), P_7, m, \text{“+”});$
$\{Z_{11} = P_1 + P_4 - P_5 + P_7\}$
$Z_{12} \leftarrow S(P_3, P_5, m, \text{“+”});$
$\{Z_{12} = P_3 + P_5\}$
$Z_{21} \leftarrow S(P_2, P_4, m, \text{“+”});$
$\{Z_{21} = P_2 + P_4\}$
$Z_{22} \leftarrow S(S(S(P_1, P_3, m, \text{“+”}), P_2, m, \text{“−”}), P_6, m, \text{“+”});$
$\{Z_{22} = P_1 + P_3 - P_2 + P_6\}$
$Z[1 : m, 1 : m] \leftarrow Z_{11}$; $Z[1 : m, m + 1 : n] \leftarrow Z_{12}$;
$Z[m + 1 : n, 1 : m] \leftarrow Z_{21}$; $Z[m + 1 : n, m + 1 : n] \leftarrow Z_{22}$;
devolva Z $\{Z = X \cdot Y\}$

fim

procedimento $S(A, B, m, s)$: $\{A\ s\ B$, onde $s =$ “+” ou $s =$ “−”$\}$

início
$i \leftarrow 1$;
enquanto $i \leq m$ **faça**
início
$j \leftarrow 1$;
enquanto $j \leq m$ **faça**
início
se $s =$ “+” **então** $c_{ij} \leftarrow a_{ij} + b_{ij}$
senão $c_{ij} \leftarrow a_{ij} - b_{ij}$;
$j \leftarrow j + 1$
fim;
$i \leftarrow i + 1$
fim;
devolva C $\{C = A\ s\ B\}$
fim

Vamos agora calcular o número de multiplicações $M(n)$ utilizadas por *Prodmat* para calcular o produto de duas matrizes $n \times n$, onde $n = 2^k$. Se $k = 1$ então o número de multiplicações é 7 pois neste caso chamamos diretamente *Prodmat*2. Para $k \geq 2$ chamamos *Prodmat* recursivamente, para multiplicar matrizes $n/2 \times n/2$, sete vezes. O número de multiplicações utilizadas em cada uma destas chamadas recursivas é $M(2^{k-1})$. Portanto, temos

$$\begin{cases} M(2^1) = 7 \\ M(2^k) = 7 \cdot M(2^{k-1}) \text{ para } k > 1. \end{cases}$$

A solução desta recorrência é $M(2^k) = 7^k$ (vide o Exercício 3). Lembrando que $n = 2^k$, temos:

$$M(n) = 7^k = 7^{\log_2 n} = n^{\log_2 7}.$$

Conseguimos, portanto, reduzir o número de multiplicações de n^3 para $n^{\log_2 7}$, que é uma economia considerável, já que $\log_2 7 = 2{,}80735\ldots$.

O leitor atento terá observado que *Prodmat*2 economiza uma multiplicação à custa de quatorze somas. Como *Prodmat* é baseado em *Prodmat*2, pareceria, à primeira vista, que a economia que obtivemos no número de multiplicações poderia ser anulada por um aumento no número de somas. Um exame mais minucioso da situação revela, no entanto, que *Prodmat* na realidade utiliza menos operações de soma do que o algoritmo clássico, exceto para valores pequenos de n. Com efeito seja $A(n)$ o número de adições que *Prodmat* utiliza para $n = 2^k$. Se $n = 2$ então este número é 18, o número de somas (ou subtrações) que *Prodmat*2 utiliza. Para $k > 1$ chamamos *Prodmat* recursivamente 7 vezes com matrizes $n/2 \times n/2$. Cada uma destas chamadas utiliza portanto $A(2^{k-1})$ somas. Antes de chamar *Prodmat*, no entanto, somas são utilizadas também na rotina de soma de matrizes S, que usa $(n/2)^2$ somas[2]. A rotina S é chamada 18 vezes em *Prodmat*.

(2) Contamos aqui apenas as somas no anel A. Não estamos contando as somas necessárias para o controle do processo, nos incrementos às variáveis i e j.

Portanto, temos

$$\begin{cases} A(2^1) = 18 \\ A(2^k) = 7 \cdot A(2^{k-1}) + 18 \cdot (2^{k-1})^2. \end{cases}$$

A solução desta recorrência é $A(2^k) = 6 \cdot (7^k - 4^k)$ (vide o Exercício 3). O número de somas utilizadas por *Prodmat* é, portanto, menor do que $6 \cdot 7^k$, isto é, menor do que $6 \cdot n^{\log_2 7} < n^3 - n^2$ para n suficientemente grande.

Vejamos agora o que fazer se n não for uma potência de 2. Seja $2^{k-1} < n < 2^k$. Essencialmente o algoritmo que usaremos é o *Prodmat* visto acima, chamado com a potência de 2 mais próxima. Para isto preenchemos com zeros as $2^k - n$ linhas e colunas extras criadas, conforme o esquema abaixo.

$$X = \begin{array}{c} \overbrace{}^{n} \\ \left.\begin{array}{|c|c|} \hline X & 0\ 0\ \ldots\ 0 \\ & 0\ 0\ \ldots\ 0 \\ & \ldots\ldots \\ \hline 0\ 0\ \ldots & \ldots\ldots\ 0 \\ \ldots\ldots & \ldots\ldots\ldots \\ 0\ 0\ \ldots & \ldots\ldots\ 0 \\ \hline \end{array}\right\} n \\ \underbrace{}_{2^k} \end{array}; \qquad Y' = \left[\begin{array}{|c|c|} \hline Y & 0\ 0\ \ldots\ 0 \\ & 0\ 0\ \ldots\ 0 \\ & \ldots\ldots \\ \hline 0\ 0\ \ldots & \ldots\ldots\ 0 \\ \ldots\ldots & \ldots\ldots\ldots \\ 0\ 0\ \ldots & \ldots\ldots\ 0 \\ \hline \end{array}\right];$$

$$Z' = X' \cdot Y' = \left[\begin{array}{|c|c|} \hline X \cdot Y & 0\ 0\ \ldots\ 0 \\ & 0\ 0\ \ldots\ 0 \\ & \ldots\ldots \\ \hline 0\ 0\ \ldots & \ldots\ldots\ 0 \\ \ldots\ldots & \ldots\ldots\ldots \\ 0\ 0\ \ldots & \ldots\ldots\ 0 \\ \hline \end{array}\right].$$

É fácil ver que o número de multiplicações será menor que $7n^{\log_2 7}$ e o número de somas menor que $42n^{\log_2 7}$.

Vamos entender melhor o mecanismo pelo qual Strassen conseguiu reduzir, assintoticamente, isto é, para n suficientemente grande, tanto o número de multiplicações como o número de somas. Inicialmente, vamos admitir que é o número de multiplicações que queremos

realmente minimizar, isto é, admitamos que o custo de uma multiplicação no anel seja muito maior que o custo de uma soma. A idéia de Strassen é então a seguinte: vamos minimizar o número de multiplicações num caso particular ($n = 2$), sem nos preocupar com o número de somas, já que o custo de cada multiplicação é muito maior que o custo de uma soma. Se conseguirmos um algoritmo melhor para o caso particular, que se aplique em qualquer anel não comutativo, então no caso geral podemos tirar proveito desta economia pelo artifício de dividir as matrizes de entrada originais em blocos, de tal maneira que possamos aplicar o algoritmo particular às matrizes de entrada, consideradas como matrizes de blocos. (A hipótese da não comutatividade do anel é aqui usada, já que o produto de matrizes é em geral não comutativo.) Desta forma reduzimos o problema original a produtos e somas de matrizes menores ($n/2 \times n/2$, no nosso caso). A economia no número de multiplicações no caso particular, se transformará aqui num número menor de produtos de matrizes do que no algoritmo clássico, e poderemos aumentar ainda mais esta economia se reaplicarmos recursivamente o método para calcular estes produtos de matrizes. Por outro lado, o aumento no número de somas no algoritmo particular se transforma num aumento no número de somas de matrizes ($n/2 \times n/2$, no nosso caso). Mas somar matrizes é mais fácil do que multiplicar matrizes, exigindo, pelo método clássico, da ordem de n^2 somas contra da ordem de n^3 multiplicações e somas para cada produto de matrizes, e portanto é vantajoso trocar um número menor de produtos de matrizes por um número maior de somas de matrizes, ainda que a hipótese original, de que multiplicações são muito mais caras do que somas no nosso anel, fosse falsa. É devido a este fato que, surpreendentemente, o algoritmo acaba economizando até no número de somas para n suficientemente grande.

Como vimos, o algoritmo de Strassen economiza no número de multiplicações utilizadas para calcular o produto de duas matrizes, mas para n pequeno aumenta o número de somas. Se o custo de cada multiplicação for muito maior que o custo de cada soma então o algoritmo poderá ser aplicado com vantagem mesmo para n relativamente pequeno. Se isto não acontecer, então o algoritmo só será vantajoso na multiplicação de matrizes muito grandes. Admitindo que o custo de cada multiplicação seja m, e de cada adição a, a fórmula do custo do algoritmo de Strassen é $7^k m + 6(7^k - 4^k)a$, para cal-

cular o produto de duas matrizes $2^k \times 2^k$, enquanto que o algoritmo clássico terá custo total $8^k m + (8^k - 4^k)a$[3]. O algoritmo começa a economizar tanto no número de somas como no número de multiplicações a partir de $k = 14$, isto é $n = 2^{14}$. O número total de operações no anel, (que é proporcional ao custo se $m = a$), começa a ser melhor para o algoritmo de Strassen a partir de $k = 10$, isto é, $n = 1.024$. Na prática, portanto, em muitos casos será mais conveniente continuar a usar o algoritmo clássico para calcular produtos de matrizes de tamanho relativamente pequeno, embora existam métodos de melhorar o desempenho do algoritmo de Strassen, além do indicado aqui. O artigo de P. Fischer mencionado no fim do capítulo apresenta alguns destes métodos e discute as condições em que o algoritmo de Strassen é superior ao algoritmo clássico.

Em geral, pelo mesmo método, utilizando recursivamente um algoritmo que, para um r dado, multiplica duas matrizes $r \times r$ usando m multiplicações, podemos obter um algoritmo que multiplica duas matrizes $n \times n$ para um n arbitrário usando $O(n^{\log_r m})$ multiplicações e somas. Assim, se para algum r existir um tal algoritmo com $\log_r m < \log_2 7$, então a partir dele poderíamos obter um algoritmo assintoticamente melhor do que o de Strassen. Recentemente V. Ya. Pan encontrou um tal algoritmo que multiplica matrizes $r \times r$ usando $m = (r^3 - 4r)/3 + 6r^2$ multiplicações. Em particular para $r = 70$ resulta $m = 143.640$, e como $\log_{70} 143.640 = 2,79512\ldots < \log_2 7$, seu algoritmo quando utilizado recursivamente é assintoticamente mais rápido do que o de Strassen. Talvez matrizes possam ser multiplicadas ainda mais rapidamente por métodos completamente diferentes, com possivelmente $n^2 \log n$ ou até mesmo n^2 multiplicações. Para que possamos ter uma idéia de quanta melhoria podemos ter alguma esperança de conseguir no futuro, é preciso que achemos limites inferiores para o número de operações necessárias. É este o assunto que discutiremos na Seção 4. Antes porém, veremos alguns outros problemas relacionados com o produto de matrizes.

---

(3) Estamos computando aqui apenas os custos devidos a operações no anel. Na realidade há um custo adicional de "controle" do processo, para tomar conta das chamadas recursivas e percorrer as matrizes da maneira correta. Este custo poderá ser desprezível ou não, dependendo do anel considerado, e dos métodos de acesso às matrizes.

3. Problemas correlatos

Nesta seção vamos mostrar que existem alguns problemas, cuja complexidade se relaciona com a do produto de matrizes, de tal modo que um algoritmo mais eficiente para o produto de matrizes também resulta em algoritmos melhores para estes problemas. A apresentação será algo esquemática, deixando detalhes e algumas demonstrações fáceis como exercícios. Assim, essa seção é de leitura algo mais difícil e pode ser omitida num primeiro estudo, sem prejuízo para a compreensão do restante do livro.

Seja A um anel com unidade. Um elemento $a \in A$ é *inversível* em A se existir um elemento indicado por a^{-1}, chamado de *inverso* de a em A tal que

$$a\,a^{-1} = a^{-1}a = 1.$$

É fácil ver que se um inverso de um elemento de A existir então ele é único. Dados $a, b \in A$, dizemos que eles *comutam* se $ab = ba$. Note que se b for inversível em A então a e b comutam sse a e b^{-1} também comutam. Neste caso definimos a operação de *divisão* de a por b, cujo resultado, o *quociente* entre a e b, é dado por

$$\frac{a}{b} = ab^{-1} = b^{-1}a.$$

Como a unidade comuta com todos os elementos de A, resulta que o inverso de qualquer elemento inversível de A pode ser obtido com uma operação de divisão pois $a^{-1} = 1/a$.

Estudaremos agora o número de multiplicações e/ou divisões em A envolvidas na determinação da inversa de uma matriz inversível no anel das matrizes $n \times n$ sobre A (vide Exercício 23(d)). Examinaremos apenas uma subclasse do conjunto das matrizes inversíveis. O método pode ser estendido a classes mais amplas conforme os Exercícios 10 e 11. Seja X uma matriz inversível $n \times n$ sobre A e escrevamos X como a matriz de blocos

$$X = \left[\begin{array}{c|c} X_{11} & X_{12} \\ \hline X_{21} & X_{22} \end{array}\right]$$

e suponhamos que a matriz X_{11} possui uma inversa X_{11}^{-1}, que n é

par, que todos os blocos têm dimensão $n/2 \times n/2$, e que a matriz

$$Y = X_{22} - X_{21}X_{11}^{-1}X_{12}$$

também é inversível, com inversa Y^{-1}. Então a inversa X^{-1} da matriz X pode ser escrita como

$$X^{-1} = \begin{bmatrix} X_{11}^{-1} + X_{11}^{-1}X_{12}Y^{-1}X_{21}X_{11}^{-1} & -X_{11}^{-1}X_{12}Y^{-1} \\ -Y^{-1}X_{21}X_{11}^{-1} & Y^{-1} \end{bmatrix}.$$

Podemos usar a identidade acima para calcular X^{-1}, usando duas inversões de matrizes $n/2 \times n/2$ e cinco produtos de matrizes $n/2 \times n/2$ (vide Exercício 9). Se n for uma potência de dois, isto sugere um método rápido de inversão de matrizes. Note, porém, que nem sempre é possível continuar o processo recursivamente: há matrizes inversíveis X tais que X_{11} ou Y não é inversível. Nos exercícios indicamos condições suficientes para que todas as submatrizes encontradas no processo sejam inversíveis, e, para matrizes de números reais, mostramos como reduzir o problema de inverter uma matriz inversível arbitrária, ao de inverter uma matriz que satisfaz estas condições. Assim, suporemos que a matriz X é inversível, n é potência de dois e que todas as submatrizes X_{11} e Y são inversíveis. Neste caso, usando as técnicas recursivas desenvolvidas para o método de Strasssen, temos a recorrência

$$\begin{cases} I(1) = 1, \\ I(n) = 2I\left(\dfrac{n}{2}\right) + 5\,M\left(\dfrac{n}{2}\right), \end{cases}$$

onde $I(n)$ denota o número de multiplicações e/ou divisões usadas pelo método para inverter matrizes $n \times n$, e $M(n)$ é o número de tais operações usadas no cálculo do produto de duas matrizes $n \times n$. Supondo que

$$\begin{cases} M(1) \geq 1, \\ M(n) \geq 4M\left(\dfrac{n}{2}\right), \end{cases} \tag{1}$$

Segue por indução que

$$I(n) \leq 3M(n).$$

Assim, é possível inverter matrizes deste tipo usando um número de multiplicações e/ou divisões da mesma ordem que o usado por qualquer algoritmo de multiplicação de matrizes, desde que a condição (1) seja satisfeita. Em particular, se usarmos o algoritmo de Strassen para a multiplicação de matrizes, a recorrência acima fica

$$\begin{cases} I(1) = 1, \\ I(n) = 2I\left(\frac{n}{2}\right) + 5\left(\frac{n}{2}\right)^{\log_2 7} \end{cases}$$

cuja solução é

$$I(n) = n^{\log_2 7}.$$

Reciprocamente, suponhamos agora que dispomos de um algoritmo que inverte matrizes $m \times m$ usando $I(m)$ operações de multiplicação e/ou divisão. Então, para calcular o produto de duas matrizes $n \times n$ X e Y considere a matriz $3n \times 3n$.

$$Z = \begin{bmatrix} I_n & X & 0_n \\ 0_n & I_n & Y \\ 0_n & 0_n & I_n \end{bmatrix}$$

onde I_n denota a matriz identidade $n \times n$ e 0_n a matriz $n \times n$ que tem todos os elementos nulos. É fácil ver que Z é inversível e Z^{-1} é dada por

$$Z^{-1} = \begin{bmatrix} I_n & -X & X \cdot Y \\ 0_n & I_n & -Y \\ 0_n & 0_n & I_n \end{bmatrix}$$

Como Z^{-1} pode ser obtido usando $I(3n)$ multiplicações e/ou divisões e ela contém a submatriz $X \cdot Y$, vemos que $I(3n)$ operações são suficientes também para o cálculo do produto de duas matrizes $n \times n$. Se denotarmos por $M(n)$ o número mínimo de multiplicações e/ou divisões necessário para multiplicar duas matrizes $n \times n$, isto mostra que

$$M(n) \leq I(3n).$$

Supondo que $I(3n) \in O(I(n))$ resulta que

$$M(n) \in O(I(n))$$

(para uma classe de funções que satisfazem a hipótese acima sobre $I(n)$ veja o Exercício 13 (c)).

Um outro problema relacionado com produto de matrizes é o de multiplicar matrizes booleanas. *Matrizes booleanas* são definidas sobre o conjunto $\{0, 1\}$ munido das operações $\vee$ (*disjunção*), $\wedge$ (*conjunção*) e $\neg$ (*negação*), definidas por:

$\vee$	0	1
0	0	1
1	1	1

$\wedge$	0	1
0	0	0
1	0	1

$\neg$	0	1
	1	0

Dadas duas matrizes booleanas $n \times n$ X e Y, seu *produto* é definido como a matriz booleana $Z = X \cdot Y$, cujos elementos z_{ij} são dados por

$$z_{ij} = \bigvee_{k=1}^{n} (x_{ik} \wedge y_{kj}) .$$

Matrizes booleanas podem ser utilizadas para representar relações sobre conjuntos finitos: dada uma relação R sobre o conjunto $\{1, 2, \ldots, n\}$, fazemos $r_{ij} = 1$ sse o elemento i está na relação R com o elemento j. É fácil ver que a relação representada pelo produto de matrizes booleanas é a composta das relações que as matrizes representam. Assim, o produto de matrizes booleanas é associativa, e a matriz booleana $n \times n$ I_n que representa a relação identidade sobre $\{1, 2, \ldots, n\}$ é a unidade em relação a este produto.

A *soma* das matrizes booleanas $n \times n$ X e Y, que indicaremos por $X \vee Y$, é a matriz booleana $n \times n$ W, cujos elementos w_{ij} são dados por

$$w_{ij} = x_{ij} \vee y_{ij}.$$

Definimos o *fecho reflexivo e transitivo* da matriz booleana $n \times n$ X como a matriz booleana $n \times n$ X^* dada pela soma

$$X^* = I_n \vee X \vee (X \cdot X) \vee ((X \cdot X) \cdot X) \vee \ldots .$$

A matriz booleana X^* representa o fecho reflexivo e transitivo R^* da relação R representada por X. (Veja os exercícios no fim deste capítulo e do Capítulo D. I.)

A seguir daremos algoritmos eficientes para o cálculo do fecho reflexivo e transitivo e para a multiplicação de matrizes booleanas. A medida de complexidade que adotaremos é o número de operações booleanas usado pelos algoritmos.

Em primeiro lugar, mostraremos que, sob certas hipóteses, os dois problemas têm complexidades assintóticas de mesma ordem. As reduções são muito semelhantes às vistas para a inversão de matrizes.

Suponhamos primeiro que temos um algoritmo que calcula o fecho reflexivo e transitivo de uma matriz booleana $m \times m$ usando $F(m)$ operações booleanas e que desejamos calcular o produto das matrizes booleanas $n \times n$ X e Y. Suponhamos ainda que $F(3n) \in O(F(n))$. Consideremos a matriz booleana $3n \times 3n$

$$Z = \begin{bmatrix} 0_n & X & 0_n \\ 0_n & 0_n & Y \\ 0_n & 0_n & 0_n \end{bmatrix}.$$

Então,

$$Z^2 = \begin{bmatrix} 0_n & 0_n & X \cdot Y \\ 0_n & 0_n & 0_n \\ 0_n & 0_n & 0_n \end{bmatrix}$$

e $Z^i = 0_{3n}$ para $i > 2$, onde 0_k é a matriz booleana $k \times k$ que tem todos os elementos nulos. Portanto, o fecho reflexivo e transitivo Z^* é dado por

$$Z^* = \begin{bmatrix} I_n & X & X \cdot Y \\ 0_n & I_n & Y \\ 0_n & 0_n & I_n \end{bmatrix},$$

e o produto $X \cdot Y$ aparece como uma submatriz de Z^*. Denotando por $B(n)$ o número mínimo de operações booleanas necessário para o cálculo do produto de matrizes booleanas $n \times n$, a construção acima mostra que

$$B(n) \leq F(3n),$$

e portanto, pela hipótese sobre F temos que $B(n) \in O(F(n))$.

Suponhamos agora que dispomos de um algoritmo de multiplicação de matrizes booleanas $n \times n$ usando $B(n)$ operações booleanas e que desejamos calcular o fecho reflexivo e transitivo de uma matriz

booleana X. Para simplificar o algoritmo, suporemos que n é uma potência de dois. Escrevamos X como a matriz de blocos $n/2 \times n/2$

$$X = \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix}.$$

Então X^* é dado por

$$X^* = \begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{bmatrix},$$

onde os blocos Y_{11}, Y_{12}, Y_{21} e Y_{22} são dados por

$$\begin{aligned} Y_{11} &= (X_{11} \vee (X_{12} \cdot X_{22}^* \cdot X_{21}))^* \\ Y_{12} &= Y_{11} \cdot X_{12} \cdot X_{22}^* \\ Y_{21} &= X_{22}^* \cdot X_{21} \cdot Y_{11} \\ Y_{22} &= X_{22}^* \vee (X_{22}^* \cdot X_{21} \cdot Y_{11} \cdot X_{12} \cdot X_{22}^*) \end{aligned}$$

(vide Exercício 15). Assim, para calcular Y_{11}, Y_{12}, Y_{21} e Y_{22} por estas fórmulas, usamos dois fechos reflexivos e transitivos, seis produtos e duas somas, todas operações sobre matrizes booleanas $n/2 \times n/2$. Temos portanto, para o número $F(n)$ de operações booleanas usadas por este método, que

$$F(n) \leq 2F\left(\frac{n}{2}\right) + 6B\left(\frac{n}{2}\right) + 2\left(\frac{n}{2}\right)^2,$$

já que m^2 operações booleanas são suficientes para o cálculo da soma de duas matrizes booleanas $m \times m$. Supondo que a função $B(n)$ satisfaça

$$\begin{cases} B(1) \geq 1, \\ B(n) \geq 4B\left(\frac{n}{2}\right), \end{cases}$$

resulta, por indução, que

$$F(n) \leq 5B(n).$$

Agora veremos métodos para o cálculo do produto de matrizes booleanas (e, pelo visto acima, para o fecho reflexivo e transitivo).

É imediato verificar que n^3 operações $\wedge$ e $n^3 - n^2$ operações $\vee$ são suficientes para o cálculo do produto de duas matrizes booleanas, já que o algoritmo sugerido pela definição (análogo ao algoritmo clássico de multiplicação de matrizes) tem essa complexidade. Infelizmente as técnicas de Strassen, vistas na seção anterior, não se aplicam diretamente para obter um algoritmo eficiente, pois o conjunto $\{0, 1\}$ com as operações $\vee$ e $\wedge$ não é um anel, já que não existe o simétrico do elemento 1 para a operação $\vee$. (Veja os Exercícios 24 a 26 para o estudo de semianéis completos.) Apesar disto, é possível obter um algoritmo mais eficiente para a multiplicação de matrizes booleanas, como veremos a seguir.

Sejam X e Y duas matrizes booleanas $n \times n$. A matriz W, produto usual das matrizes X e Y consideradas como matrizes sobre o o anel dos inteiros, é uma matriz cujos elementos w_{ij} serão inteiros não-negativos. Seja Z a matriz booleana que é o produto (booleano) das matrizes booleanas X e Y. Então é fácil ver que $w_{ij} \neq 0$ sse $z_{ij} \neq 0$. A matriz W pode ser obtida em $O(n^{\log_2 7})$ operações aritméticas usando-se o algoritmo de Strassen. De posse de W, é fácil calcular Z.

O método acima descrito obtém o produto em $O(n^{\log_2 7})$ operações aritméticas. Estamos interessados no número de operações booleanas. Para transformar o algoritmo acima em um que use apenas operações booleanas ($\wedge$, $\vee$ e $\neg$) devemos implementar as operações aritméticas por meio de operações booleanas. É bem conhecido que a soma e o produto de números naturais pode ser calculado usando-se operações booleanas — é assim que os computadores modernos efetuam as quatro operações! Os algoritmos usuais têm complexidade $O(k)$ para a soma e $O(k^2)$ para o produto de dois números binários de k dígitos (vide Exercício 16). Para estimar o número de operações booleanas suficientes para o nosso algoritmo precisamos obter um limite superior para o número de dígitos suficiente para representar cada número usado no cálculo de W. Como cada elemento w_{ij} de W resulta da soma de n termos, cada um dos quais é no máximo 1, temos que $w_{ij} \leq n$, onde n é a dimensão da matriz W. Note que isto não significa que no cálculo dos w_{ij} pelo método de Strassen não possam surgir resultados intermediários cujo valor excede a n (vide Exercício 17). Para evitar a perda de eficiência que resultaria das operações com estes números grandes, podemos efetuar todas as somas e produtos módulo $n + 1$: isto corresponde a calcular o produto das matrizes inteiras X e Y sobre o anel $\mathbb{Z}_{n+1}$ dos inteiros módulo $n + 1$.

Como $\mathbb{Z}_{n+1}$ é um anel, o algoritmo de Strassem pode ser usado. Não é difícil provar que o resultado final será a mesma matriz W que seria obtida efetuando-se o produto no anel dos inteiros. Lembrando que o número $x \in \mathbb{N}$ pode sempre ser representado por $\lfloor log_2 x \rfloor + 1$ dígitos[4], vemos que se calcularmos W em $\mathbb{Z}_{n+1}$, nenhum resultado intermediário necessitará mais do que $\lfloor log_2 n \rfloor + 1$ dígitos.

Usando todas estas técnicas, obtemos um algoritmo que calcula o produto de duas matrizes booleanas com $O(n^{log_2 7} (log_2 n)^2)$ operações booleanas. De modo geral, se $M(n)$ for o número de operações aritméticas necessárias para o cálculo do produto de matrizes $n \times n$, e se $P(k)$ for um limite superior para o cálculo da soma e do produto de dois números de k dígitos, então $O(M(n)P(log_2 n))$ operações booleanas serão suficientes para o cálculo do produto de duas matrizes booleanas $n \times n$. Como vimos, este também é o número de operações suficiente para o cálculo do fecho reflexivo e transitivo de uma matriz booleana $n \times n$.

Existem outros problemas relacionados com multiplicação de matrizes: alguns são vistos nos exercícios. Mencionamos dois outros, que fogem do escopo deste livro:

(a) decomposição LUP: dada uma matriz $n \times n$ não singular X sobre um anel qualquer, é possível achar em $O(n^{2,795\ldots})$ operações aritméticas matrizes L, U e P, tais que $X = L \cdot U \cdot P$, P é uma matriz de permutação, L é uma matriz triangular inferior, U é uma matriz triangular superior.

(b) reconhecimento de linguagens livres de contexto: para cada gramática livre de contexto, é possível obter um algoritmo que, dada uma palavra x, decide se x está na linguagem gerada pela gramática em $O(n^{2,795\ldots})$ operações, onde n é o comprimento de x.

A seguir trataremos do problema de estabelecer limites inferiores para a complexidade do cálculo do produto de matrizes, e de alguns outros problemas algébricos.

4. Limites inferiores

Como já mencionamos no Capítulo I, é em geral muito difícil provar que uma determinada tarefa necessita de certos recursos com-

(4) $\lfloor x \rfloor$ é o maior inteiro menor ou igual a x.

putacionais, como um número mínimo de multiplicações, de comparações, tempo, espaço, etc. A dificuldade vem do fato de que para se provar um limite inferior de algum recurso computacional, devemos demonstrar que qualquer algoritmo que calcule a função desejada deverá usar pelo menos esta quantidade de recursos. É muitas vezes possível, e às vezes até fácil, mostrar que um particular algoritmo que conhecemos para a função utilize certos recursos. Mas para provar um limite inferior, este deve aplicar-se a métodos que calculem a função, mesmo a métodos que ainda não conhecemos, utilizando possivelmente propriedades ainda não descobertas. Precisamos enfim provar que *qualquer* algoritmo para a função terá de utilizar pelo menos tais e tais recursos. O exemplo da Seção 2 é um caso típico: antes de sua descoberta acreditava-se, erroneamente, que qualquer método para o cálculo do produto de duas matrizes $n \times n$ precisa utilizar pelo menos n^3 multiplicações. Consideremos agora um outro exemplo deste tipo. Suponha que queremos determinar o produto de dois números complexos. As operações permitidas são soma e multiplicação de números reais e queremos minimizar o número de multiplicações. Como $(x + iy)\ (u + iv) = (xu - yv) + i\,(xv + yu)$, aparentemente quatro multiplicações são necessárias. O seguinte algoritmo, no entanto, usa apenas três multiplicações.

procedimento *Prodcomplexo* (x, y, u, v):
  início
    $t_1, t_2, t_3 \leftarrow x + y,\ v - u,\ u + v$;
    $p_1 \leftarrow t_1 \cdot u$;
    $real \leftarrow p_1 - y \cdot t_3$;
    imaginária $\leftarrow x \cdot t_2 + p_1$;
    devolva (*real*, *imaginária*) $\{(x+iy)(u+iv) = (real + i \cdot imaginária)\}$
  fim

No caso de problemas algébricos, como produto de matrizes, cálculo de polinômios, e outros, existem algumas técnicas, que apesar das dificuldades apontadas, podem ser usadas para obter limites inferiores para o número de operações necessárias. Infelizmente, não será possível apresentarmos estas técnicas com detalhe e rigor suficientes no espaço limitado de que dispomos. Esperamos que as páginas seguintes dêem uma idéia do tipo de resultados que se pode obter e das técnicas utilizadas para prová-los, e que os leitores interessados consultem a bibliografia indicada. Vale a pena salientar que esta é uma área de pesquisa em franco desenvolvimento. Assim, é possível que

muitos dos resultados obtidos até agora sejam passíveis de melhoras substanciais.

Começaremos por um exemplo simples destas técnicas.

PROPOSIÇÃO 1. *Seja A um anel e seja*

$$f_n(a_0, a_1, \ldots, a_n) = \sum_{0 \le i \le n} a_i,$$

onde $a_i \in A$. *Qualquer programa, cujas únicas instruções sejam as operações do anel, e que calcule a função* f_n *requer n somas ou subtrações.*

Demonstração. Por indução em n. Se $n = 0$ então não há nada a provar. Suponhamos então que o cálculo de $f_n(a_0, a_1, \ldots, a_n)$ requer n operações $+$ ou $-$. Seja P um programa que calcula

$$f_{n+1}(a_0, a_1, \ldots, a_{n+1}) = \sum_{0 \le i \le n+1} a_i,$$

usando o número mínimo de somas e subtrações. O programa P deve ter pelo menos uma operação $+$ ou $-$ pois se todas as operações de P são multiplicações então somente valores da forma

$$c_1 \cdot a_{i_1}^{j_1} \cdot c_2 \cdot a_{i_2}^{j_2} \cdot \ldots \cdot c_k \cdot a_{i_k}^{j_k} \cdot c_{k+1}$$

podem ser calculadas. Em particular se $a_0 = a_1 = \ldots = a_{n-1} = 0$, então $f_{n+1}(a_0, a_1, \ldots, a_{n+1}) = a_n + a_{n+1}$, que para um anel arbitrário não pode ser expresso como um produto no anel. Consideremos então a primeira soma ou subtração de P. Todos os valores até agora calculados são produtos da forma já vista, pois esta é a primeira soma ou subtração utilizada. Esta soma ou subtração, então, precisa ser ou da forma $t_1 \pm t_2$ onde t_1 e t_2 são termos previamente calculados, e portanto da forma

$$c_1 \cdot a_{i_1}^{j_1} \cdot c_2 \cdot a_{i_2}^{j_2} \cdot \ldots \cdot c_k \cdot a_{i_k}^{j_k} \cdot c_{k+1}$$

ou uma variável simples, a_i, que é também da forma indicada. A soma ou subtração então equivale a uma instrução da forma

$$t \leftarrow c_1 \cdot a_{i_1}^{j_1} \cdot c_2 \cdot a_{i_2}^{j_2} \cdot \ldots \cdot c_k \cdot a_{i_k}^{j_k} \cdot c_{k+1} \pm d_1 \cdot a_{g_1}^{h_1} \cdot d_2 \cdot a_{g_2}^{h_2} \cdot \ldots \cdot d_e \cdot a_{g_e}^{h_e} d_{e+1}.$$

Pelo menos um dos termos t_1 ou t_2 precisa conter uma variável a_i, que suporemos sem perda de generalidade ser a_{n+1}. Seja P' o pro-

grama obtido de P onde todas as ocorrências de a_{n+1} são substituídas por 0. Tal programa calcula

$$f_n(a_0, a_1, \ldots, a_n).$$

A instrução correspondente à primeira soma ou subtração de P pode agora ser eliminada, pois o termo em que a_{n+1} aparece será igual a zero, e portanto esta instrução atribui a t ou o valor de uma variável, ou o valor de um termo anterior já calculado.[5] O programa assim obtido, P'', é equivalente a P', e calcula portanto $f_n(a_0, a_1, \ldots, a_n)$, e tem exatamente uma soma ou subtração a menos que P. Pela hipótese indutiva P'' precisa ter ao menos n somas ou subtrações. Conseqüentemente P precisa ter pelo menos n + 1 somas ou subtrações, o que prova a Proposição. ■

A demonstração acima ilustra o tipo de raciocínio utilizado neste ramo da teoria. Nos parágrafos seguintes elaboraremos os mecanismos necessários para provar limites inferiores do número de multiplicações necessárias para calcular o produto de duas matrizes.

Seja A um corpo,[6] e $x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m$ variáveis que não sejam elementos de A. Considere o anel $A[x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m]$, o menor anel com unidade comutativo que contém A e as variáveis $x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m$ e que estenda o corpo A. (Isto é, o anel de polinômios sobre A nas $n + m$ variáveis $x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m$.) Portanto, para elementos a e b de A, as operações $a + b$, $a - b$, $a \cdot b$, no anel $A[x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m]$ coincidem com as operações correspondentes de A. A idéia intuitiva de incluir as variáveis x_i e y_j é a de representar desta forma variáveis de entrada do algoritmo. Para cada execução do algoritmo, que operará sobre A, x_i e y_j assumirão valores em A, porém, como o algoritmo deve funcionar para quaisquer valores assumidos pelas variáveis de entrada, queremos que a priori não tenhamos nenhuma relação com qualquer elemento fixo de A.

Note que, como todo corpo é um anel, se um algoritmo calcular uma função sobre um anel arbitrário, em particular calcula tal função sobre todo corpo. Assim, podemos estudar limites inferiores no nú-

---

(5) Naturalmente, ao se eliminar esta instrução, é preciso também substituir t pela variável equivalente já conhecida, em todas as demais instruções em que t apareça.

(6) Um *corpo* é um anel com unidade comutativo em que todos os elementos não nulos são inversíveis.

mero de operações de algoritmos que operam sobre um corpo, que tais limites inferiores serão também válidos para algoritmos operando sobre anéis. A recíproca não é verdadeira: um algoritmo pode calcular uma função sobre um corpo usando um número de operações menor do que o usado por qualquer algoritmo que calcule a função sobre um anel arbitrário. Por exemplo, $xy + yx$ pode ser calculada com uma só multiplicação se x e y pertencerem a um corpo, mas requer duas multiplicações se eles forem elementos de um anel arbitrário.

Os algoritmos que estudaremos serão de um tipo especial, que denominaremos de algoritmos em linha reta. Um *algoritmo em linha reta* é um algoritmo, no sentido definido na Parte A, sem instruções de controle como **enquanto**, **repita**, **se então senão**, etc. ou de chamadas de procedimento. O algoritmo consiste de uma seqüência finita de instruções de atribuição de valor do tipo $u \leftarrow v \; op \; w$ onde v e w são:

(a) ou um dos x_i ou y_j;
(b) ou uma constante, isto é, um elemento de A;
(c) ou uma variável u cujo valor foi previamente calculado.

A operação *op* é uma das operações $+$, $-$, ou $\cdot$, do anel $A[x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m]$. Como queremos provar limites inferiores no número de operações aritméticas, as instruções de controle não são em geral necessárias, pois em princípio para cada n fixo, pode-se "desenrolar" um conjunto de instruções a ser repetida n vezes por meio de uma operação de controle como **enquanto** ou **repita**, pelo artifício de repetir no programa as operações envolvidas o número requerido de vezes. Desta maneira, o conceito de algoritmo em linha reta elimina os problemas irrelevantes ao aspecto que queremos estudar. Os limites inferiores que provaremos serão para algoritmos deste tipo, mas são igualmente aplicáveis a qualquer algoritmo mais geral, desde que este possa ser reduzido por um artifício, como o indicado acima, a um algoritmo em linha reta equivalente. Doravante quando usarmos a palavra "algoritmo" neste capítulo, subentenderemos ser um algoritmo em linha reta.

Um algoritmo, assim, *calcula uma função* f: $A^{n+m} \rightarrow A^p$ se, após a execução do algoritmo, um determinado p-subconjunto das variáveis u à esquerda das suas instruções contiver os p componentes do valor da função $f(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m)$ para cada valor das variáveis de entrada.

Nosso objetivo é calcular limites inferiores para o número de multiplicações necessárias no cálculo de algumas funções. Para isto

contaremos apenas multiplicações que não possam ser eliminadas (por exemplo, a multiplicação $2 \cdot x$ pode ser eliminada, pela soma $x + x$), que chamaremos de essenciais. Como estamos provando limites inferiores, podemos contar apenas o número de multiplicações essenciais, pois este é menor do que ou igual ao número de multiplicações. Em outras palavras, provaremos limites inferiores sobre multiplicações essenciais, e por conseguinte tais limites inferiores aplicam-se a qualquer multiplicação, essencial ou não.

Não contaremos multiplicações entre elementos de A, nem multiplicações em que ambos os operandos não dependem de alguma das variáveis de entrada x_i e y_j. Assim, uma multiplicação é *essencial* se ela é uma operação $u \leftarrow v \cdot w$, onde v e w são variáveis essenciais. Uma variável é *essencial* se for

ou (a) um dos x_i ou y_j,

ou (b) uma variável cujo valor foi anteriormente calculado numa instrução, em que pelo menos um dos operandos é uma variável essencial.

A seguir, indicaremos como obter um limite inferior para o número de multiplicações essenciais necessárias para o cálculo de $B\mathbf{x}$, onde $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ é o vetor coluna dos valores x_i, e B é uma matriz $p \times n$, cujos elementos são funções lineares dos y_i. Os limites serão válidos para programas em linha reta que calculem a função $B\mathbf{x}$ para qualquer corpo A. Assim, não podemos usar propriedades de corpos particulares, nem comandos condicionais para testar propriedades dos argumentos.

A razão pela qual estudaremos o número de multiplicações necessárias para o cálculo de $B\mathbf{x}$ é que muitas funções podem ser representadas desta forma.

EXEMPLO 1. Sejam $U = [u_{ij}]$ e $V = [v_{ij}]$ matrizes $n \times n$ sobre A.

A matriz $Z = UV$ pode ser obtida efetuando-se o produto da matriz $n^2 \times n^2$ B com o vetor $\mathbf{x}$, onde, na forma de blocos,

$$B = \begin{bmatrix} U & 0_n & 0_n & \ldots & 0_n \\ 0_n & U & 0_n & \ldots & 0_n \\ 0_n & 0_n & 0_n & \ldots & U \end{bmatrix}$$

e

$$\mathbf{x}^T = (v_{11}, v_{21}, \ldots, v_{n1}, v_{12}, v_{22}, \ldots, v_{n2}, \ldots, v_{1n}, v_{2n}, \ldots, v_{nn}).$$

Obtém-se então

$$(z_{11}, z_{21}, \ldots, z_{n1}, z_{12}, z_{22}, \ldots, z_{n2}, \ldots, z_{n1}, z_{n2}, \ldots, z_{nn})^T.$$

EXEMPLO 2. As partes real e imaginária do produto de dois números complexos $u + iv$ e $t + iw$ podem ser calculadas efetuando-se o produto

$$\begin{bmatrix} u & -v \\ v & u \end{bmatrix} \quad \begin{bmatrix} t \\ w \end{bmatrix}.$$

Considere o espaço vetorial (sobre o corpo A) $A^k[y_1, y_2, \ldots, y_m]$ das k-tuplas de $A[y_1, y_2, , y_m]$. Um conjunto $\{\alpha_i\}$ de r vetores de $A^k[y_1, y_2, \ldots, y_m]$ é *linearmente independente módulo A* sse

$$\sum_{i=1}^{r} c_i \alpha_i \in A^k; \qquad c_i \in A$$

implica que $c_i = 0$ para todo i. Se um conjunto de vetores não é linearmente independente módulo A, dizemos que ele é *linearmente dependente módulo A*. Em outras palavras, uma coleção de vetores é linearmente independente módulo A se não existe uma combinação linear dos vetores com coeficientes em A, nem todos nulos, que resulte num vetor em cujas componentes as variáveis y_i não aparecem.

Seja B uma matriz $r \times s$ com elementos em $A[y_1, y_2, \ldots, y_m]$. O *posto de linhas* de B *módulo* A é a cardinalidade do maior conjunto de linhas de B (consideradas como vetores de $A^s[y_1, y_2, \ldots, y_m]$) que é linearmente independente módulo A.

A ferramenta essencial para a demonstração dos resultados desta seção é o teorema que provamos agora.

TEOREMA 1. *Seja B uma matriz com elementos em* $A[y_1, y_2, \ldots, y_m]$ *e seja* $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$. *Se o posto de linhas de B módulo A é r, o cálculo de* $B\mathbf{x}$ *requer pelo menos r multiplicações essenciais.*

Demonstração. Em primeiro lugar, note que podemos supor, sem perda de generalidade, que B tem r linhas. (Caso contrário, considere a matriz B' formada por r linhas independentes de B. Todo cálculo de $B\mathbf{x}$ deve calcular $B'\mathbf{x}$, e, portanto, usar pelo menos tantas multiplicações quanto $B'\mathbf{x}$.)

Suponha agora que s multiplicações são suficientes para o cálculo de $B\mathbf{x}$, e sejam $t_1, t_2, \ldots, t_s$ as expressões calculadas por essas multiplicações. Como as únicas outras operações permitidas são somas, subtrações ou multiplicações não essenciais, os elementos de $B\mathbf{x}$ podem ser expressos como funções lineares dos t_i e das incógnitas x_j e y_k, com os coeficientes c_ℓ das funções lineares sendo elementos de A. Isto é, se

$$\mathbf{t} = (t_1, t_2, \ldots, t_s)^T$$

podemos escrever o produto $B\mathbf{x}$ como

$$B\mathbf{x} = D\mathbf{t} + \mathbf{u}, \tag{2}$$

onde os elementos d_{ij} da matriz $r \times s$ D estão no corpo A e o vetor $\mathbf{u}$ tem por componentes funções lineares das incógnitas x_i e y_j, isto é, para todo ℓ, $1 \leq \ell \leq r$, existem c_i, $c_j \in A$, tais que

$$u_\ell = c_0 + \sum_{i=1}^{n} c_i x_i + \sum_{j=1}^{m} c_j y_j.$$

Suponhamos agora que $s < r$. Então, como sabemos da Álgebra Linear, as linhas $\mathbf{d}_i$ de D são linearmente dependentes (como vetores de A^s), e portanto existem coeficientes w_i, $w_i \in A$, nem todos nulos, tais que

$$\sum_{i=1}^{r} w_i \mathbf{d}_i = 0.$$

Seja $\mathbf{w}$ o vetor não nulo $\mathbf{w} = (w_1, w_2, \ldots, w_r)$. Multiplicando ambos os lados da igualdade (2) por $\mathbf{w}$, temos

$$\mathbf{w}B\mathbf{x} = \mathbf{w}D\mathbf{t} + \mathbf{w}\mathbf{u} = \mathbf{w}\mathbf{u}.$$

Podemos escrever esta igualdade como

$$(\mathbf{w}B)\mathbf{x} = \mathbf{w}\mathbf{u}.$$

Já que as componentes de $\mathbf{w}$ estão em A, o lado direito é uma função linear de incógnitas com coeficientes em A, e portanto o lado esquerdo não poderá ter produtos de incógnitas. Isto só é possível se o resultado de multiplicar o vetor linha $\mathbf{w}$ por B for um vetor cujas componentes estão em A (porque a linhas distintas de $\mathbf{x}$ correspondem variáveis x_i distintas). Mas $\mathbf{w}B \in A^n$ equivale a dizer que as linhas de B são linearmente dependentes módulo A, e, portanto, o posto de linhas de B módulo A não é r, contrariando a hipótese.

Como a contradição resultou do fato de supormos que $s < r$, devemos ter $s \geq r$, o que prova o teorema. ■

COROLÁRIO 1. *Pelo menos n^2 multiplicações são necessárias para calcular o produto de duas matrizes $n \times n$.*

Demonstração. O posto de linhas módulo A da matriz B do Exemplo 1 é n^2. ■

Existem várias maneiras de generalizar o teorema anterior: pode-se provar versões mais fortes do teorema que dão limites inferiores sobre algoritmos em linha reta em que divisões também são permitidas; pode-se demonstrar que o posto de colunas de B módulo A também é um limite inferior para o número de multiplicações essenciais necessárias para o cálculo de $B\mathbf{x}$, e pode-se obter uma formulação geral de limites inferiores sobre o número de multiplicações necessárias para o cálculo de formas bilineares baseada no posto de um tensor de ordem três associado à forma bilinear. Não podemos expor estas técnicas neste capítulo, apenas mencionamos alguns dos limites inferiores que podem ser provados por meio delas. Alguns destes tópicos são vistos nos exercícios no fim do capítulo.

Citamos os seguintes limites inferiores conhecidos para o número de multiplicações necessárias para o cálculo de algumas funções por algoritmos em linha reta:

cálculo do valor de um polinômio de grau n num ponto	n
produto de números complexos usando produtos de reais	3
produto de quaternions, usando produtos de reais	8
produto de matrizes quadradas $n \times n$	$2n^2 - n$.

Nos três primeiros casos, o limite superior é também o limite inferior: a regra de Horner usa n multiplicações para o cálculo do valor de um polinômio; vimos um algoritmo que calcula o produto de números complexos usando apenas 3 multiplicações; e o algoritmo de "multiplicação rápida" de quaternions pode ser encontrado em [66]. Produto de matrizes, no entanto, é um problema que continua em aberto: vimos que, até hoje, o melhor limite superior é da ordem de $n^{2,795\ldots}$ multiplicações, enquanto o melhor limite inferior é da ordem de n^2. A diferença entre os dois é muito grande, e é um problema fascinante (e aparentemente difícil) tentar diminuí-la.

EXERCÍCIOS

1. Verifique que o algoritmo *Prodmat2* calcula o produto $X \cdot Y$ corretamente, usando 7 multiplicações (não comutativas) e 18 somas.
2. Prove que o produto de matrizes pode ser efetuado por blocos.
3. (a) Prove que a solução da recorrência

$$\begin{cases} M(2^1) = 7 \\ M(2^k) = 7M(2^{k-1}) \text{ para } k > 1 \end{cases}$$

é $M(2^k) = 7^k$.

(b) Prove que a solução da recorrência

$$\begin{cases} A(2^1) = 18 \\ A(2^k) = 7 \cdot A(2^{k-1}) + 18(2^{k-1})^2 \text{ para } k > 1 \end{cases}$$

é $A(2^k) = 6 \cdot (7^k - 4^k)$.

4. Prove que a solução da recorrência

$$\begin{cases} X(1) = b, \\ X(n) = aX\left(\frac{n}{c}\right) + bn, \end{cases}$$

satisfaz

$$X(n) \in \begin{cases} O(n) & \text{se } a < c, \\ O(n \log n) & \text{se } a = c, \\ O(n^{\log_c a}) & \text{se } a > c. \end{cases}$$

5. Ache o menor n_0 tal que $6n_0^{\log_2 7} < n_0^3 - n_0^2$.
6. Mostre que o programa abaixo calcula o produto de duas matrizes 2×2 usando apenas 7 multiplicações e 15 somas (ou subtrações).

procedimento *Outroprodmat2*(X, Y):
início

$s_1 \leftarrow x_{21} + x_{22}$;
$s_2 \leftarrow s_1 - x_{11}$;
$s_3 \leftarrow x_{11} - x_{21}$;
$s_4 \leftarrow x_{12} - s_2$;
$s_5 \leftarrow y_{12} - y_{11}$;
$s_6 \leftarrow y_{22} - s_5$;
$s_7 \leftarrow y_{22} - y_{12}$;
$s_8 \leftarrow s_6 - y_{21}$;

$p_1 \leftarrow s_2 \cdot s_6$;
$p_2 \leftarrow x_{11} \cdot y_{11}$;
$p_3 \leftarrow x_{12} \cdot y_{21}$;
$p_4 \leftarrow s_3 \cdot s_7$;
$p_5 \leftarrow s_1 \cdot s_5$;
$p_6 \leftarrow s_4 \cdot y_{22}$;
$p_7 \leftarrow x_{22} \cdot s_8$;
$s_9 \leftarrow p_1 + p_2$;
$s_{10} \leftarrow s_9 + p_4$;
$z_{11} \leftarrow p_2 + p_3$;
$z_{12} \leftarrow s_9 + p_5 + p_6$;
$z_{21} \leftarrow s_{10} - p_7$;
$z_{22} \leftarrow s_{10} + p_5$;
devolva Z
fim

7. Escreva um programa em *LP* que calcula o produto de matrizes sobre $\mathbb{Z}_2$, o corpo dos inteiros módulo 2, sem usar a soma ou o produto de $\mathbb{Z}_2$.
8. Prove que uma multiplicação é necessária para o cálculo do produto de duas matrizes 1×1. (Sugestão: o programa deve funcionar para quaisquer anéis. Considere o anel dos polinômios com coeficientes em $\mathbb{Z}$ sobre duas variáveis não comutativas a e b, e mostre que o produto ab, resultado da multiplicação das matrizes 1×1 $[a]$ e $[b]$ não pode ser expresso como o resultado de somas e subtrações envolvendo a e b.)
9. Verifique a fórmula do texto para inversão de matrizes por blocos, e que ela pode ser programada usando apenas cinco multiplicações de matrizes.
10. Dê um exemplo de uma matriz inversível X, tal que
    (a) a submatriz X_{11} não seja inversível.
    (b) X_{11} seja inversível, mas $Y = X_{22} - X_{21} X_{11}^{-1} X_{21}$ não seja.
    Mostre que as matrizes X_{11} e Y são sempre inversíveis, se X for não-singular e
    (c) X é uma matriz triangular superior (triangular inferior).
    (d) A é um corpo ordenado e X é simétrica, positiva definida.

11. Mostre que se X é uma matriz não singular sobre um corpo ordenado, e se X^T é a transposta de X, então
    (a) $S = XX^T$ é simétrica positiva definida;
    (b) $X^{-1} = X^T \cdot S^{-1}$
    Usando os fatos acima, esboce um algoritmo para inverter matrizes sobre corpos ordenados que usa $O(M(n))$ multiplicações e divisões.

12. Prove que o método de Gauss para inversão de matrizes usa $O(n^3)$ operações de divisão e multiplicação.

13. (a) Prove que se $M(n) \geq 4M\left(\frac{n}{2}\right)$ então $I(n) \leq 3M(n)$ (Note que $M(1) \geq 1$ pelo Exercício 8.)
    (b) Prove que a solução de recorrência

$$\begin{cases} I(1) = 1, \\ I(n) = 2I\left(\frac{n}{2}\right) + 5 \cdot \left(\frac{n}{2}\right)^{\log_2 7}, \end{cases}$$

    satisfaz $I(n) < 7n^{\log_2 7}$. Obtenha a solução em seguida.
    (c) Mostre que a família de funções que sejam positivas, monotônicas não decrescentes, não limitadas e tais que $f(cn) \in \in O(f(n))$ para toda constante $c > 1$, contém as funções $a \cdot \log n$ e $a \cdot n^b$ para a e b reais e positivos. Ademais esta família é fechada sob soma, produto e composição de funções.

14. Mostre que
    (a) a operação $\wedge$ pode ser expressa por meio de composições de $\vee$ e $\neg$;
    (b) a operação $\vee$ pode ser expressa por meio de composições de $\wedge$ e $\neg$;
    (c) a operação $\neg$ não pode ser expressa por meio de composições de $\wedge$ e $\vee$.

15. Verifique a fórmula da Seção 3 para o fecho reflexivo e transitivo.

16. Dê algoritmos para a soma e para o produto de números naturais usando operações booleanas. Analise a complexidade dos algoritmos.

17. Considere a seguinte técnica para multiplicação de naturais escritos em notação binária:
    para multiplicar números de um dígito $x \cdot y = x \wedge y$;

para multiplicar números de n dígitos, onde $n = 2^k$, escreva

$$x = a \cdot 2^{n/2} + b$$
$$y = c \cdot 2^{n/2} + d$$

com a, b, c e d números de $n/2$ dígitos. Calcule $x \cdot y$ pelo algoritmo

$$u \leftarrow (a + b) \cdot (c + d);$$
$$v \leftarrow a \cdot c;$$
$$w \leftarrow b \cdot d;$$
$$z \leftarrow v \cdot 2^n + (u - v - w) \cdot 2^{n/2} + w$$

Lembrando que podemos multiplicar em notação binária por 2^n acrescentando n zeros à direita, vemos que o algoritmo acima usa três multiplicações de números de comprimento $n/2$.

(a) Escreva um algoritmo para multiplicação de inteiros usando a técnica acima. Mostre que sua complexidade satisfaz a recorrência

$$\begin{cases} P(1) = k, \\ P(n) = 3P(n/2) + kn, \end{cases}$$

onde k é uma constante.

(b) Mostre que a solução da recorrência acima é

$$P(n) = 3kn^{\log_2 3} - 2kn.$$

(c) O raciocínio acima contém um erro. Ache-o. Escreva as somas $(a + b)$ e $(c + d)$ como $\alpha 2^{n/2} + e$, e $\gamma 2^{n/2} + f$ com α e γ dígitos binários e obtenha um algoritmo correto. Mostre que sua complexidade é da ordem $n^{\log_2 3}$.

18. (a) Dê um exemplo de matrizes $n \times n$ sobre $\{0, 1\}$, cujo produto, pelo método de Strassen, produz resultados intermediários maiores do que n.

(b) Sejam X e Y matrizes de inteiros positivos, $Z = X \cdot Y$ e suponha que todos os elementos de X, Y e Z são limitados por n. Mostre que os resultados do produto $X \cdot Y$, calculados nos anéis $\mathbb{Z}$ e $\mathbb{Z}_{n+1}$ são idênticos. (Identificamos o inteiro $i \leq n$ com o elemento $1 + 1 + \ldots + 1$ (i vezes) de $\mathbb{Z}_{n+1}$.)

19. (a) Mostre que os vetores (y_1, y_2), $(y_2, 0)$ e $(0, y_2)$ de $A^2[y_1, y_2]$ são linearmente independentes módulo A.

(b) Defina o posto de colunas módulo A de uma matriz B com elementos em $A[y_1, y_2, \ldots, y_m]$. Mostre que o posto de colunas de B não é necessariamente igual ao posto de linhas de B.

20. Prove o seguinte teorema: Seja B uma matriz $p \times n$ cujos elementos são funções lineares das variáveis $y_1, y_2, \ldots, y_m$, e $\mathbf{x}$ um vetor coluna de n componentes. Se o posto de colunas de B módulo A for r, o cálculo de $B\mathbf{x}$ requer pelo menos r multiplicações essenciais.

21. Usando o teorema acima prove
    (a) o cálculo do valor de um polinômio de grau n num ponto requer n multiplicações (note que a regra de Horner usa exatamente este número de multiplicações e é, portanto ótima).
    (b) o cálculo do produto de uma matriz $n \times m$ por um vetor coluna de m componentes requer nm multiplicações.

    Sugestão: para a parte (a) considere o produto da matriz $1 \times (n+1)$ $[1\ x\ x^2\ \ldots\ x^n]$ pelo vetor $[a_0\ a_1\ \ldots\ a_n]^T$.
    Para a parte (b) considere o produto da matriz $n \times nm$

$$\begin{bmatrix} v_1 & v_2 & \ldots & v_m & 0 & 0 & \ldots & 0 & 0 & \ldots & 0 & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 & v_1 & v_2 & \ldots & v_m & 0 & \ldots & 0 & 0 & \ldots & 0 \\ \multicolumn{14}{c}{\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots} \\ 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 & 0 & \ldots & v_1 & v_2 & \ldots & v_m \end{bmatrix}$$

    pelo vetor

$$[b_{11}\ b_{12}\ \ldots\ b_{1m}\ b_{21}\ b_{22}\ \ldots\ b_{2m}\ \ldots\ b_{n1}\ b_{n2}\ \ldots\ b_{nm}]^T.$$

22. Mostre como estender os algoritmos de inversão de matrizes e de fecho reflexivo e transitivo para o caso de matrizes $n \times n$, onde n não é uma potência de 2. Analise a complexidade dos algoritmos.

23. Considere o cálculo da função $p_n(x) = x^n$ sobre um corpo arbitrário.
    (a) mostre que se $n = 2^k$, então k multiplicações são suficientes para o cálculo de x^n.
    (b) mostre que se $\gamma(n)$ = número de algarismos 1 na representação binária de n, então $\lceil \log_2 n \rceil + \gamma(n) - 1$ multiplicações são suficientes para o cálculo de x^n.
    (c) mostre que se somente somas, subtrações e multiplicações são permitidas, então pelo menos $\lceil \log_2 n \rceil$ multiplicações são necessárias.
    (d) mostre que se divisões também são permitidas, então 6 multiplicações e/ou divisões são suficientes para calcular x^{31}, mas 7 multiplicações são necessárias em caso contrário.

24. Um *semianel completo* é um conjunto S, munido de operações de soma $+$ e produto $\cdot$ que satisfazem as seguintes propriedades
$(S, \cdot)$ é um monóide, isto é,
(i) $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ para todo $a, b, c \in S$;
(ii) Existe um elemento $1 \in S$ tal que

$$1 \cdot a = a \cdot 1 = a$$

para todo $a \in S$.
$(S, +)$ é um monóide comutativo, isto é,
(iii) $a + (b + c) = (a + b) + c$ para todo $a, b, c \in S$;
(iv) $a + b = b + a$ para todo $a, b \in S$;
(v) Existe um elemento $0 \in S$, tal que

$$a + 0 = 0 + a = a,$$

para todo $a \in S$.
A operação $\cdot$ distribui sobre $+$, isto é, para todo $a, b, c \in S$
(vi) $(a + b) \cdot c = a \cdot c + b \cdot c$;
(vii) $a \cdot (b + c) = a \cdot b + a \cdot c$.
O elemento 0 satisfaz
(viii) $a \cdot 0 = 0 \cdot a = 0$,
para todo $a \in S$.
A soma $\sum_{i \in I} a_i$ é definida para todo conjunto enumerável de índices I, com $a_i \in S$, e satisfaz
(ix) $\sum_{i \in \emptyset} a_i = 0$;
(x) $\sum_{i \in \{i\}} a_i = a_i$;
(xi) se $\{I_j \mid j \in J\}$ é uma partição de I, então

$$\sum_{i \in I} a_i = \sum_{j \in J} \left[\sum_{\ell \in I_j} a_\ell \right];$$

(xii) para todo $b \in S$

$$b \cdot \left[\sum_{i \in I} a_i \right] = \sum_{i \in I} b \cdot a_i.$$

Mostre que
(a) O conjunto $\{0, 1\}$ com as operações $\wedge$ e $\vee$ correspondendo a $\cdot$ e $+$, é um semianel completo.
(b) Seja $S = \mathbb{R}_+ \cup \{\infty\}$ onde $\mathbb{R}_+$ é o conjunto dos números reais não-negativos, com as operações de soma e *min* correspondendo a $\cdot$ e $+$ respectivamente. Então $(S, min, +)$ é um semianel completo.

(c) Seja S um semianel completo. Considere a família $M_n(S)$ das matrizes $n \times n$ com coeficientes em S. Mostre que $M_n(S)$ com as operações de soma e de produto de matrizes induzidas pelas operações $+$ e $\cdot$ de S, é um semianel completo.

(d) Mostre que

(d_1) as propriedades (iii) e (iv) da soma num semianel completo decorrem das propriedades (ix), (x) e (xi).

(d_2) a propriedade (viii) não é conseqüência das demais propriedades, mas seria conseqüência de (viii'): Para todo $a \in S$ existe um elemento $-a$, tal que $a + (-a) = (-a) + a = 0$. Mostre que a propriedade (viii') não vale nos semianéis (a) e (b) acima.

(d_3) Mostre que num semianel completo

$$\left[\sum_{i \in I} a_i\right] \cdot \left[\sum_{j \in J} b_j\right] = \sum_{i \in I,\, j \in J} a_i \cdot b_j .$$

25. Dado um semianel completo S, com as operações $+$ e $\cdot$, defina a^i para todo $a \in S$ e $i \in N$, por

$$\begin{cases} a^0 = 1, \\ a^{i+1} = a^i \cdot a. \end{cases}$$

Defina a^* como $a^* = \sum_{i \in N} a^i = 1 + a + a^2 + a^3 + \ldots$

No caso do semianel $M_n(S)$ do Exercício 24, a operação $*$ é chamada de fecho reflexivo e transitivo.

(a) Mostre que o produto de dois elementos de $M_n(S)$ pode ser obtido em $O(n^3)$ operações $+$ e $\cdot$ do semianel S.

(b) Mostre que num semianel completo S, o número $P(n)$ de operações $+$ e $\cdot$ de S necessárias para o cálculo do produto de duas matrizes de $M_n(S)$ é assintoticamente o mesmo que o número $F(n)$ de operações necessárias para o cálculo do fecho reflexivo e transitivo de um elemento de $M_n(S)$, supondo que $F(3n) \in \in O(F(n))$, $P(1) \geq 1$ e $P(n) \geq P(n/2)$. (Sugestão: a prova para o caso particular do semianel da parte (a) do Exercício 24 está no texto.)

26. (Cf. o Capítulo C. VI) Considere um grafo orientado cujas arestas são rotuladas com elementos de um semianel completo S. Seja M a matriz cujo elemento $m_{ij} = a$, $a \in S$ sse a soma dos rótulos das arestas com extremo inicial i e extremo final j é a. (E portanto, se não há aresta com extremo inicial i e extremo final j, então $m_{ij} = 0$.)

Dados dois vértices i e j e um passeio orientado P de arestas α_1, α_2, ..., α_k, rotulados por $a_1, a_2, \ldots, a_k$, o rótulo do passeio orientado P é o elemento $a_1 \cdot a_2 \cdot \ldots \cdot a_k$ de S.

(a) Verifique que o elemento m_{ij} de M é a soma dos rótulos de todos os passeios orientados de comprimento 1 de i a j.

(b) Verifique que o elemento (i, j) de M^2 é a soma dos rótulos de todos os passeios orientados de comprimento 2 de i a j.

(c) Verifique que o elemento (i, j) de M^* é a soma dos rótulos de todos os passeios orientados de i a j.

(d) Considere o caso particular do semianel $(\{0, 1\}, \vee, \wedge)$ do Exercício 24 (a). Mostre que o elemento (i, j) de M^* é $m^*_{ij} = 1$ sse existe um caminho orientado de i até j.

(e) Considere o semianel $(\mathbb{R}_+ \cup \{\infty\}, min, +)$ do Exercício 24 (b). Mostre que se o elemento (i, j) de M^* é d, então d é o comprimento de um caminho orientado de menor comprimento de i a j (onde o comprimento de cada aresta é igual ao seu rótulo).

NOTAS BIBLIOGRÁFICAS

O trabalho pioneiro na área é o artigo de Ostrowski [91] que coloca o problema de quantas multiplicações são necessárias para o cálculo de um polinômio num ponto, questão hoje totalmente esclarecida (cf. o Exercício 20 e o livro de Borodin e Munro [8]), embora um problema essencialmente equivalente ao do Exercício 22 (cadeias de adição) tenha sido anteriormente estudado em Matemática. O artigo de Strassen [116] expõe o método rápido de multiplicação de matrizes. A decomposição LUP mencionada no texto é do artigo [10] e o Exercício 10 é uma idéia não publicada de Schönhage. O Exercício 6 é devido a Winograd. O algoritmo de Pan a que referimos no texto está em [92]. A discussão do uso do algoritmo de Strassen é de [34]. O único livro-texto na área é [1]. Existe também uma monografia [8] que trata dos problemas de complexidade algébrica vistas no capítulo, expondo o que se sabia na área em 1975. Muitos dos resultados podem ser encontrados apenas como artigos de jornais.

Os resultados citados no texto sobre quaternions são de [66], o limite inferior de $2n^2 - n$ multiplicações para o produto de matrizes de [67] e o método rápido de multiplicação de inteiros é de [105].

CAPÍTULO IV

É MAIS FÁCIL VERIFICAR A SOLUÇÃO DO QUE ENCONTRÁ-LA?

1. Introdução

No encontro da Sociedade Americana de Matemática de 1903, Frank Cole apresentou um resultado com uma característica bastante incomum. De um lado, seu resultado desprovou uma conjetura em aberto há mais de duzentos e cinqüenta anos; no entanto, não levou mais do que alguns minutos para que Cole convencesse a sua audiência de matemáticos de que a conjetura em questão era falsa, e sua prova consistia de uma única identidade:

$$2^{67} - 1 = 147.573.952.589.676.412.927 = \\ = 193.707.721 \times 761.838.257.287.$$

Evidentemente, a conjetura a que nos referimos no parágrafo anterior era a proposição de Mersenne de que números da forma $2^p - 1$ seriam primos para uma lista de certos primos p. A lista de Mersenne foi laboriosamente conferida à mão, sem o emprego de computadores, antes de 1950, sendo que o resultado de Cole foi um dos poucos erros encontrados. Desde então a lista foi consideravelmente estendida com o auxílio de computadores.

O exemplo de Cole ilustra de maneira dramática que, aparentemente, encontrar os fatores de um número composto pode em geral ser muito difícil, tanto é que levou mais de dois séculos para que alguém encontrasse os fatores de $2^{67} - 1$, mas uma vez encontrados tais fatores torna-se um problema relativamente trivial verificar que o número em questão é realmente um número composto, bastando para isso efetuar uma única multiplicação. Existe uma variedade de problemas conhecidos que parecem exibir o mesmo fenômeno, no sentido de que conhecemos algoritmos que verificam rapidamente se um canditado proposto como uma solução é ou não de fato uma solução, mas não conhecemos nenhum algoritmo rápido que encontra esta solução. Por exemplo, seja SAT o conjunto de todas as fórmulas satisfazíveis do cálculo proposicional, isto é, todas as expressões envolvendo variáveis lógicas $p, q, \ldots,$ e os operadores lógicos $\neg$ (negação), $\wedge$

(conjunção) e $\vee$ (disjunção), tais que existe uma interpretação que dê valores *verdadeiro* ou *falso* às variáveis da fórmula de tal maneira, que a fórmula com esta interpretação se torne verdadeira. Para decidir se uma fórmula F com n variáveis é satisfazível, precisamos, aparentemente, examinar 2^n possíveis interpretações. De fato não se conhece nenhum algoritmo rápido que decida se F pertence ou não a SAT. Se F for satisfazível, no entanto, uma vez conhecida a interpretação que a torne verdadeira, umas poucas operações lógicas são suficientes para verificar este fato.

Embora não se conheçam algoritmos rápidos para os problemas mencionados, tampouco conseguiu-se até agora demonstrar que tais algoritmos não existem. É então natural perguntar, intuitivamente, se verificar a solução de problemas deste tipo é inerentemente um problema mais fácil do que encontrar essa solução. Neste capítulo nos dedicaremos a formalizar esta pergunta e mostrar alguns dos resultados obtidos tentando respondê-la. Como veremos, o problema formal correspondente a nossa pergunta é uma das questões centrais em aberto em Teoria da Computação. Informalmente, o resultado principal que iremos provar afirma que existem problemas naturais de computação (SAT é um exemplo), cuja solução pode ser verificada "rapidamente" por um algoritmo, e tal que, se conseguirmos um algoritmo "rápido" para encontrar a solução de um só destes problemas, então a solução de todos os problemas que podem ser verificados rapidamente por um algoritmo pode também ser encontrada rapidamente por um algoritmo.

2. Conceitos básicos

É evidente que antes que possamos estudar a pergunta levantada informalmente na introdução é preciso caracterizar as classes de problemas cuja solução pode ser respectivamente encontrada e verificada por algoritmos rápidos.

Antes de mais nada, ocupar-nos-emos apenas com problemas que admitem por resposta *sim* ou *não*. Já examinamos alguns problemas deste tipo no Capítulo II. Restringir nossa atenção a estes problemas permite simplificar as nossas considerações e, além do mais, a maioria dos problemas de nosso interesse pode ser formulada desta maneira. Assim são, por exemplo, os problemas de determinar se um número dado é ou não composto, se uma fórmula do cálculo proposicional

é ou não satisfazível, e assim por diante. Estes problemas, portanto, equivalem a decidir se um objeto dado pertence ou não a um certo conjunto.

Em segundo lugar, podemos restringir nossa atenção a conjuntos de palavras sobre um alfabeto finito, porque qualquer objeto de nosso interesse pode ser representado por uma palavra. Por exemplo, qualquer número natural pode ser representado pela sua representação decimal que é uma palavra sobre o alfabeto $\{0, 1, 2, \ldots, 9\}$. Similarmente, qualquer fórmula proposicional pode ser vista como uma palavra sobre o alfabeto $\{p, q, \neg, \vee, \wedge, (,), v, f\}$ onde os nomes das variáveis lógicas são palavras não vazias sobre $\{p, q\}$. Na Seção 4 definiremos este conjunto com mais pormenores.

O número de símbolos do alfabeto é relativamente sem importância, desde que seja pelo menos dois, pois por uma codificação símbolo a símbolo podemos representar qualquer palavra sobre um alfabeto Σ com $m \geq 2$ símbolos, por uma palavra sobre, por exemplo, o alfabeto $\{0, 1\}$. Para isto bastarão[1] $k = \lceil log_2 m \rceil$ símbolos de $\{0, 1\}$ para representar cada símbolo de Σ, e portanto uma palavra de Σ^* de comprimento n pode ser codificada por uma palavra de $\{0, 1\}^*$ de comprimento $k \cdot n$.

Pela discussão acima, todos os problemas que nos interessarão neste capítulo serão do tipo geral de decidir se um objeto dado representado por uma palavra sobre um alfabeto conveniente pertence ou não a um certo conjunto de palavras A. Conforme vimos no Capítulo I, uma máquina de Turing determinística que reconhece A corresponde de modo natural a decidir esta questão por meio de um algoritmo, o algoritmo representado pela máquina de Turing. Similarmente, uma máquina de Turing não determinística que aceita A corresponde de modo natural a verificar, por meio de um algoritmo, se uma solução proposta ao problema correspondente a A é correta. A solução que se quer verificar, ou de maneira mais geral, informações adicionais relativas ao problema, são armazenadas na fita de sugestões. Por exemplo, para o caso dos números compostos que vimos na introdução, a fita de sugestões pode conter os fatores do número representado pela entrada. A máquina não determinística multiplicaria estes fatores e verificaria se o produto é o número representado pela entrada, aceitando se for, e rejeitando em caso contrário. É claro que tal máquina

(1) — $\lceil x \rceil$ denota o menor inteiro maior ou igual a x.

não determinística aceita o conjunto de palavras que representam números compostos, pois se a entrada representar um número composto então com uma sugestão apropriada a máquina aceita a entrada, e se a entrada não representar um número composto então esta máquina rejeitará a entrada qualquer que seja a sugestão.

3. Eficiência

Até agora não tratamos neste capítulo do problema da eficiência de nossos algoritmos. Na introdução vimos no entanto que nosso interesse é em algoritmos "rápidos".

De acordo com o Capítulo I, a complexidade de tempo associada a uma máquina de Turing determinística ou não determinística M é dada pela função $t_M(x)$, que pode ser limitada superiormente por alguma função $g : \mathbb{N} \to \mathbb{N}$. Mas qual o limite de tempo que devemos considerar "rápido"? É claro que tal pergunta só pode ser respondida num contexto empírico. Certamente tal limite deve crescer menos que uma exponencial no comprimento da entrada, pois exponenciais já crescem tão rapidamente que, se dispusermos apenas de um algoritmo exponencial para resolver um problema, então conseguiremos usá-lo apenas para entradas de tamanho relativamente reduzido. A tabela abaixo ilustra este fato. Suponhamos que dispomos de cinco algoritmos para a resolução de um certo problema, de tempos de execução n, $n \log n$, n^2, n^3 e 2^n passos respectivamente. Suponhamos ainda que dispomos de um computador que execute mil passos por segundo. A Tabela 1, fornece para cada tipo de algoritmo o comprimento máximo da entrada para vários tempos de execução, onde n representa o comprimento da entrada.

ALGORITMO	TEMPO (PASSOS)	n MÁXIMO PARA TEMPO DE EXECUÇÃO		
		1 *s*	1 min	1 h
1	n	1.000	6×10^4	$3{,}6 \times 10^6$
2	$n \log n$		4.893	$2{,}0 \times 10^5$
3	n^2	31	244	1.897
4	n^3	10	39	153
5	2^n	9	15	21

Tabela 1

Note que multiplicando o tempo disponível por uma constante multiplica o tamanho máximo da entrada por pelo menos uma constante para os primeiros quatro algoritmos, que são polinomiais, mas aumenta apenas de uma constante para o último, que é um algoritmo exponencial.

Para que consideremos um algoritmo "rápido", o limite de tempo deve então ser subexponencial. Por outro lado, para qualquer problema não trivial precisaremos certamente pelo menos tempo suficiente para ler toda a entrada, o que implica que tal limite deve ser pelo menos linear no comprimento da entrada. Uma família de limites superiores que parece razoável sob este ponto de vista é a classe de polinomiais com coeficientes inteiros não negativos $p : \mathbb{N} \to \mathbb{N}$, que denotaremos por *POL*.

A escolha de *POL* traz outras vantagens, de ordem matemática. De fato, *POL* possui propriedades de fechamento que permitem combinar algoritmos polinomiais de diversas maneiras para obter novos algoritmos polinomiais. *POL* é fechada sob soma, produto, e composição funcional. Por exemplo, um algoritmo polinomial em que cada passo e substituído por um algoritmo polinomial dá origem a um outro algoritmo polinomial.

Seja $\mathscr{P}$ a família de todos os conjuntos reconhecíveis em tempo *POL* por uma máquina de Turing determinística, e seja $\mathscr{NP}$ a família de todos os conjuntos aceitáveis em tempo *POL* por uma máquina de Turing não determinística. Pelos motivos acima parece razoável identificar a classe de problemas cuja solução pode ser encontrada rapidamente por um algoritmo com $\mathscr{P}$ e a classe de problemas cuja solução pode ser verificada rapidamente por um algoritmo com $\mathscr{NP}$. A questão levantada na introdução se traduz então na pergunta se $\mathscr{P}$ é ou não igual a $\mathscr{NP}$, ou em outras palavras, já que $\mathscr{P} \subseteq \mathscr{NP}$ pelo Exercício 1, se $\mathscr{P}$ está ou não propriamente contido em $\mathscr{NP}$. Esta questão, levantada por Cook, encontra-se ainda hoje em aberto. Nas seções seguintes veremos parte do progresso conseguido tentando resolvê-la, que ao mesmo tempo mostrará porque esta questão é um dos problemas centrais de Teoria da Computação.

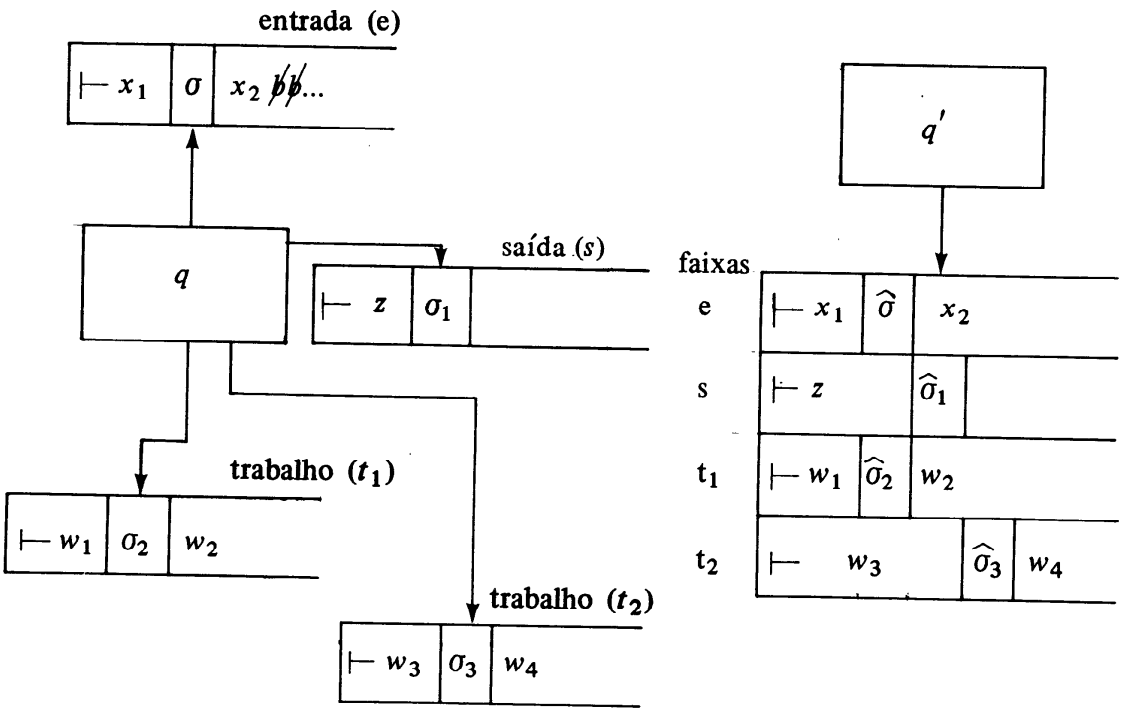
O modelo de algoritmo que definimos no Capítulo I, a máquina de Turing, é evidentemente um modelo extremamente simples. Pela Tese de Church qualquer algoritmo pode ser "programado" neste modelo. Mas será que qualquer algoritmo eficiente pode ser programado de maneira eficiente em modelo tão simples? Pareceria, à primeira vista, que poderia acontecer que uma máquina de Turing fosse ineficiente, não porque representasse um algoritmo ineficiente, mas

devido à própria simplicidade, e portanto relativa ineficiência, do modelo em si. Assim, poderíamos pensar que talvez existam algoritmos rápidos, isto é, polinomiais, em computadores reais, mas que implementados numa máquina de Turing deixariam de ser polinomiais. Um exame mais minuncioso, no entanto, revela que isto não pode acontecer. Dentre os modelos formais definidos, o mais próximo dos computadores reais é chamado *máquina de acesso aleatório*. Pode-se demonstrar que qualquer passo de uma máquina deste tipo pode ser simulado em tempo polinomial numa máquina de Turing. Para se ter uma idéia do tipo de técnicas envolvidas nestas simulações esboçamos abaixo a prova de que máquinas de Turing com várias fitas de trabalho podem ser simuladas eficientemente por uma máquina de Turing com uma única fita.

Uma *máquina de Turing com fitas múltiplas* é um sistema formal como o definido no Capítulo I, exceto que o controle central tem acesso a várias fitas de trabalho em vez de uma única fita de entrada/trabalho/saída. Normalmente neste tipo de modelo admite-se uma fita separada, de leitura apenas, contendo a entrada, e uma fita de gravação apenas, onde a máquina escreve a sua saída. Inicialmente as fitas de trabalho estão totalmente brancas. Um passo de uma máquina com fitas múltiplas é similar ao da máquina de fita única, exceto que as ações da máquina dependem agora do estado de controle central e dos símbolos lidos da fita de entrada e das fitas de trabalho. Cada cabeça das fitas de trabalho e da fita de saída escreve um símbolo na respectiva fita independentemente das demais, e cada cabeça é movida independentemente das demais no máximo uma célula para a esquerda ou para a direita na sua fita. A cabeça da fita de saída não pode ser movida para a esquerda. A função $t_M(x)$ define-se analogamente ao caso da máquina de fita única.

TEOREMA 1. *Dada uma máquina de Turing M determinística com fitas múltiplas, existe uma máquina de Turing M determinística, com uma única fita de entrada/trabalho/saída tal que (a) se M com entrada x parar em t passos então M' pára com entrada x em no máximo* $4t^2$ *passos; (b) M' aceita x sse M aceita x; (c) A saída de M' é igual à saída de M.*

Demonstração. Daremos apenas as idéias principais da demonstração. Cada uma das fitas de M é representada por uma faixa na fita única de M' conforme a Figura 1.



Máquina M com m fitas de trabalho e alfabeto $\Sigma = \{\vdash, ⊔, \sigma_0, \sigma_1, \ldots, \sigma_i\}$

Máquina M' com alfabeto $\Sigma'^{|m+2}$
$\Sigma' = \{\vdash, \hat{\vdash}, ⊔, \hat{⊔}, \sigma_0, \hat{\sigma}_0, \ldots, \sigma_i, \hat{\sigma}_i\}$ (2)

Figura 1

Um símbolo na fita de M' é uma $(m + 2)$-tupla representando o conteúdo de uma célula em cada uma das faixas correspondentes às fitas de entrada, saída e às fitas de trabalho de M. Se a cabeça numa fita de M estiver sobre esta célula então o símbolo na posição correspondente na faixa apropriada é assinalado com a marca "^". Um passo de M é simulado por M' do seguinte modo (supomos que a cabeça de M', ao se iniciar a simulação de um passo de M, encontra-se sobre a célula mais à esquerda da fita):

(a) M' desloca a cabeça para a direita até encontrar a célula em cada faixa que contém o símbolo marcado com "^". Deste modo M'

(2) Pelas convenções admitidas o alfabeto de uma máquina de Turing deve incluir pelo menos os símbolos $\{\vdash, ⊔, 0, 1\}$. Para enquadrar M' nestas convenções e para que M' calcule a mesma função que M, é preciso escolher $(m + 2)$-tuplas apropriadas para representar os símbolos de Σ. Deixamos ao leitor a implementação de pormenores deste tipo.

descobre os símbolos lidos por M neste passo e os armazena no seu controle central;

(b) Uma vez descobertos estes símbolos, M' desloca a cabeça para a esquerda escrevendo os novos símbolos escritos por M nesse passo, nas células das faixas correspondentes;

(c) Deslocando a cabeça para a direita, M' atualiza as marcas "^" das cabeças de M que se deslocam para a direita neste passo;

(d) Finalmente, deslocando a cabeça para a esquerda, M' atualiza as marcas "^" das cabeças de M que se deslocam para a esquerda neste passo, voltando também a cabeça para a célula mais à esquerda da fita, e refletindo no estado do controle central a mudança de estado de M.

Note que nenhuma cabeça de M pode afastar-se mais do que t células para a direita na respectiva fita, já que em cada passo cada cabeça pode ser deslocada de no máximo uma célula para a direita. Deste modo cada passo de M requer no máximo $4t$ passos de M' para ser simulado. Portanto a simulação toda requer no máximo $4t^2$ passos. ■

COROLÁRIO 1. *Se uma máquina de Turing M com fitas múltiplas reconhece um conjunto A em tempo polinomial, então existe uma máquina de Turing M' com uma única fita que reconhece A em tempo polinomial.*

TEOREMA 2. *Dada uma máquina de Turing não determinística M com fitas múltiplas, existe uma máquina de Turing não determinística M', com uma única fita de entrada/trabalho/saída que aceita o mesmo conjunto que M. Ademais, se M parar com entrada x e sugestão y em t passos então M' pára com entrada x e sugestão y em no máximo $4t^2$ passos.*

Demonstração. Análoga à do Teorema 1. ■

4. Fórmulas do cálculo proporcional

Vejamos agora algumas noções fundamentais de lógica que, como veremos, serão de importância para a questão $\mathscr{P} \stackrel{?}{=} \mathscr{NP}$.

Considere o alfabeto $\Sigma = \{p, q, v, f, (,), \neg, \vee, \wedge\}$.

Uma *constante proposicional* ou simplesmente *constante*, é um elemento do conjunto $\{v, f\}$. Interpretaremos v como representando *verdadeiro*, e f como representando *falso*.

Uma *variável proposicional*, ou simplesmente *variável*, é uma palavra não vazia sobre o alfabeto $\{p, q\}$. Exemplos: p, q, pq, etc.

Uma *fórmula atômica* ou *átomo* é uma variável, ou uma constante, ou uma palavra da forma $\neg w$ onde w é uma variável ou constante.

Uma *fórmula* é uma fórmula atômica ou uma palavra da forma $\neg A$, $(A \vee B)$, ou $(A \wedge B)$ onde A e B são fórmulas. Exemplos de fórmulas são: $p, f, \neg p, v, \neg v, \neg(p \wedge q), \neg(\neg\neg(p \vee q) \wedge pq)$, etc. As primeiras cinco destas fórmulas são fórmulas atômicas.

Seja A uma fórmula. Denotamos por $V(A)$ o *conjunto de variáveis* de A, definido por:

(i) Se A é uma constante então $V(A) = \varnothing$;
(ii) Se A é uma variável então $V(A) = \{A\}$;
(iii) Se A é da forma $\neg B$ onde B é uma fórmula então $V(A) = V(B)$;
(iv) Se A é da forma $(B \vee C)$ ou $(B \wedge C)$ então $V(A) = V(B) \cup V(C)$.

Seja V um conjunto de variáveis. Uma *interpretação sobre* V é uma função $I : V \to \{v, f\}$. Se $V(A) \subseteq V$ então qualquer interpretação sobre V se diz uma *interpretação de* A.

Dada uma fórmula A e uma interpretação I de A, o *valor* de A segundo I, denotado por $|| A ||_I$, é o elemento de $\{v, f\}$ definido por:

(i) Se A é uma constante então $|| A ||_I = A$;
(ii) Se A é uma variável então $|| A ||_I = I(A)$;
(iii) Se A é da forma $\neg B$ onde B é uma fórmula então

$$|| A ||_I = \begin{cases} v \text{ se } || B ||_I = f \\ f \text{ se } || B ||_I = v; \end{cases}$$

(iv) Se A é da forma $(B \vee C)$ onde B e C são fórmulas então

$$|| A ||_I = \begin{cases} v \text{ se } || B ||_I = v \text{ ou } || C ||_I = v \\ f \text{ se } || B ||_I = || C ||_I = f; \end{cases}$$

(v) Se A é da forma $(B \wedge C)$ onde B e C são fórmulas então

$$|| A ||_I = \begin{cases} v \text{ se } || B ||_I = || C ||_I = v \\ f \text{ se } || B ||_I = f \text{ ou } || C ||_I = f. \end{cases}$$

Duas fórmulas A e B são *logicamente equivalentes* se para toda interpretação I de ambas, $|| A ||_I = || B ||_I$.

Uma *disjução* de átomos (fórmulas) é uma fórmula do tipo $(((A_1 \vee A_2) \vee A_3) \vee \ldots \vee A_n)$ onde $A_1, A_2, A_3, \ldots, A_n$ são átomos (fórmulas).

Uma *conjução* de átomos (fórmulas) é uma fórmula do tipo $(((A_1 \wedge A_2) \wedge A_3) \wedge \ldots \wedge A_n)$ onde $A_1, A_2, A_3, \ldots, A_n$ são átomos (fórmulas).

Uma fórmula está em *forma normal disjuntiva* (*conjuntiva*) se for uma disjunção (conjunção) de fórmulas, cada uma das quais sendo uma conjunção (disjunção) de átomos. Exemplos: $\neg p$, $(p \vee q)$, $(p \wedge q)$, e $((p \wedge q) \vee (\neg pp \wedge pq))$ estão em forma normal disjuntiva; $\neg p$, $(p \vee q)$, $(p \wedge q)$, e $(p \wedge ((p \vee \neg q) \vee pq))$ estão em forma normal conjuntiva; $(((p \vee q) \wedge pp) \vee \neg p)$ não está nem em forma normal conjuntiva nem em forma normal disjuntiva.

Devido à propriedade associativa das operações lógicas $\wedge$ e $\vee$ (Exercício 2, (iii) e (iv)), muitos parênteses são desnecessários numa fórmula. Assim, convencionaremos, no intuito de reduzir os parênteses nas nossas expressões, que

$A_1 \vee A_2 \vee A_3 \vee \ldots \vee A_n$ abrevia $(((A_1 \vee A_2) \vee A_3) \vee \ldots \vee A_n)$,

e $A_1 \wedge A_2 \wedge A_3 \wedge \ldots \wedge A_n$ abrevia $(((A_1 \wedge A_2) \wedge A_3) \wedge \ldots \wedge A_n)$,

onde $A_1, A_2, A_3, \ldots, A_n$ são fórmulas. É também costumeiro convencionar que $\wedge$ tem precedência sobre $\vee$, e que $\neg$ tem precedência sobre $\wedge$ e $\vee$. Assim, $\neg A_1 \vee A_2$ denota $(\neg A_1 \vee A_2)$ e não $\neg(A_1 \vee A_2)$; $A_1 \vee A_2 \wedge A_3$ denota $(A_1 \vee (A_2 \wedge A_3))$ e não $((A_1 \vee A_2) \wedge A_3)$. Expressões obtidas de fórmulas, eliminando parênteses de acordo com estas regras, serão denominadas de *fórmulas abreviadas*. Seguem alguns exemplos:

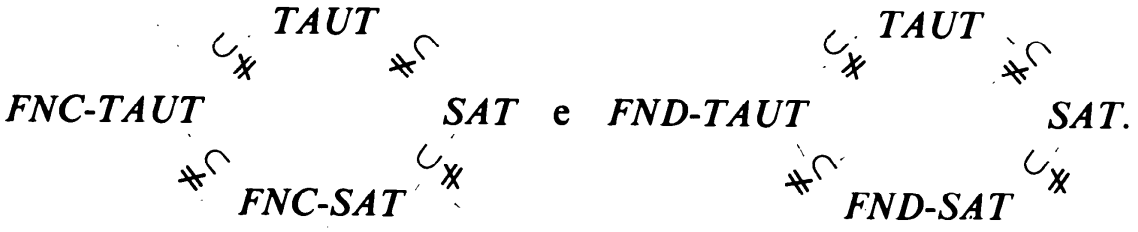
FÓRMULA	FÓRMULA ABREVIADA
$((p \wedge ((\neg p \vee q) \vee pq)) \wedge pp)$	$p \wedge (\neg p \vee q \vee pq) \wedge pp$
$\neg(A_1 \vee A_2)$	$\neg(A_1 \vee A_2)$
$(((\neg p \wedge q) \wedge pp) \vee (q \wedge pq))$	$\neg p \wedge q \wedge pp \vee q \wedge pq.$

Uma fórmula A é *satisfazível* se existe uma interpretação I de A tal que $\| A \|_I = v$.

Uma fórmula A é uma *tautologia* se for logicamente equivalente a v.

Denotaremos o conjunto de todas as fórmulas satisfazíveis, abreviadas ou não, por *SAT* e o conjunto de todas as tautologias, abreviadas ou não, por *TAUT*. *FNC-SAT* denotará o conjunto de todas as fórmulas abreviadas satisfazíveis em forma normal conjuntiva.

Analogamente, *FND-SAT* denotará o conjunto de todas as fórmulas abreviadas satisfazíveis em forma normal disjuntiva; *FNC-TAUT*, o conjunto de todas as tautologias abreviadas, em forma normal conjuntiva, e *FND-TAUT*, o conjunto de todas as tautologias abreviadas, em forma normal disjuntiva. Note que:



TEOREMA 3. *FNC-SAT pertence a* $\mathcal{NP}$.

Demonstração. Queremos construir uma máquina de Turing não determinística M que aceita *FNC-SAT* em tempo polinomial. Pelo Teorema 2 podemos supor que M tem uma fita de trabalho, e as fitas de entrada e de sugestões que são de leitura apenas. Seja x a entrada de comprimento n, e y a sugestão. A máquina M será construída de tal forma que se y for da forma $a_1b_1a_2b_2 \ldots a_mb_m$, onde $a_i \in \{p, q\}^*$, $b_i \in \{v, f\}$, $1 \leq i \leq m$, $m \leq |x| = n$ e $|y| \leq 2n$, codificando a interpretação b_i de cada variável a_i de x que torna a fórmula x *verdadeira*, e se x estiver em forma normal conjuntiva abreviada, então M aceita x. Caso contrário x é rejeitado. Note que $m \leq n$ e $|y| \leq 2n$. Para cada entrada x e sugestão y, a máquina M levará no máximo um número polinomial de passos para parar. Estas condições claramente garantem que M aceita *FNC-SAT* em tempo polinomial.

Na construção de M utilizaremos várias submáquinas com funções específicas. A máquina M será uma composição conveniente destas submáquinas. Note que uma fórmula abreviada em forma normal conjuntiva é uma expressão da forma[3]: $D_1 \wedge D_2 \wedge \ldots \wedge D_s$, onde $s \geq 1$ e cada D_i é da forma A_{1i} ou $(A_{1i} \vee A_{2i} \vee \ldots \vee A_{r_i i})$, $r_i > 1$, onde os A_{ji} são átomos. Descreveremos agora as submáquinas de M.

(3) Para simplificar a demonstração, vamos admitir a restrição adicional de que uma fórmula abreviada que consiste de uma só disjunção de vários átomos deve estar entre parênteses. Assim, a fórmula $(p \vee q)$ será aceita por M enquanto $p \vee q$ não será aceita. Note, porém, que o teorema é válido independentemente desta restrição.

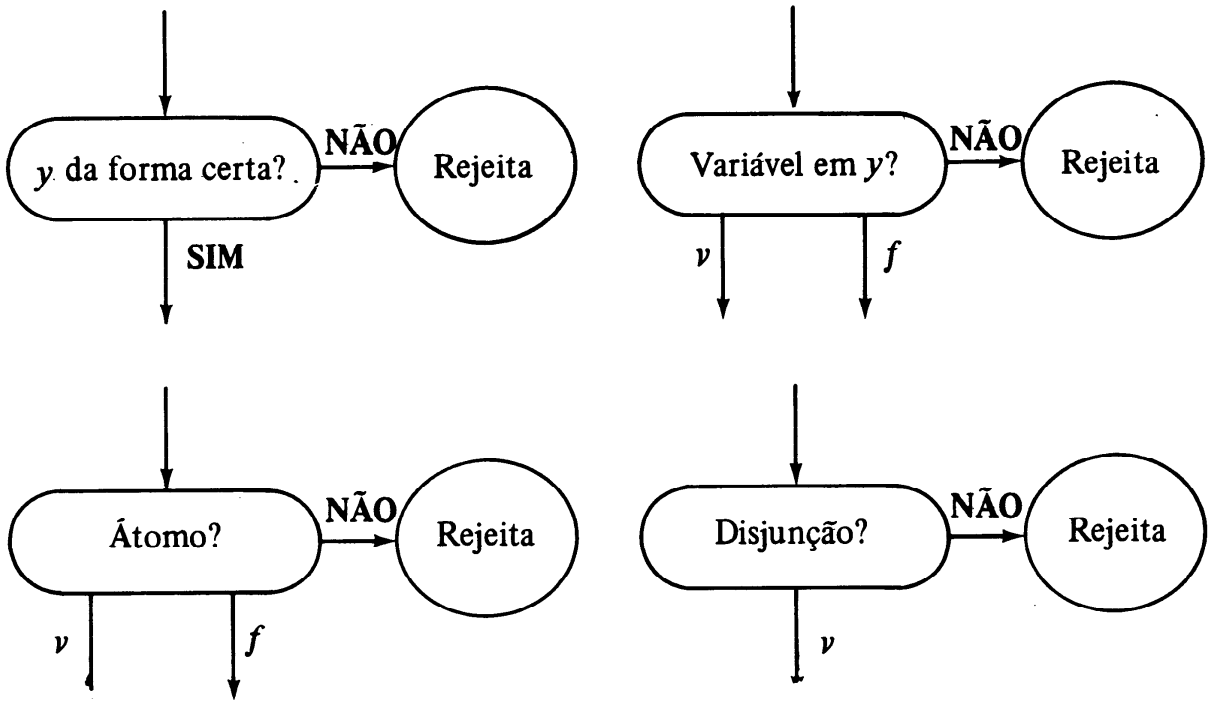


Figura 2 — As submáquinas de M.

(i) Submáquina "y é da forma certa?".

Descrição: Lê x e y simultaneamente verificando se $|y| \leq 2|x|$ e se y é da forma $a_1b_1a_2b_2 \ldots a_mb_m$ onde $a_i \in \{p, q\}^*$ e $b_i \in \{v, f\}$. Se y não for desta forma ou então se $|y| > 2|x|$ então rejeita. Caso contrário volta ambas as cabeças para o começo das respectivas fitas e devolve o comando.

Tempo de execução: $\leq 4n$.

(ii) Submáquina "Variável em y?".

Descrição: Verifica se a variável escrita na fita de trabalho ocorre em y como um dos a_i. Se não ocorrer então rejeita. Caso contrário devolve o comando no estado v se $b_i = v$, e f se $b_i = f$. A computação de "Variável em y?" começa com as cabeças da fita de sugestões e da fita de trabalho no começo das respectivas fitas, e termina com as cabeças na mesma posição. A comparação de cada a_i com a variável na fita de trabalho é feita símbolo a símbolo. Se uma das duas palavras acabar primeiro ou se algum símbolo não for o mesmo nas duas palavras então a cabeça da fita de sugestões é avançada para o próximo a_i e a cabeça da fita de trabalho é reposicionada para o símbolo mais à esquerda da variável, repetindo-se a comparação.

Tempo de execução: $\leq 2n^2 + 2n$. Cada comparação da variável na fita de trabalho com um dos a_i leva no máximo $2n$ passos, e fazemos no máximo $m \leq n$ comparações. Assim levamos no máximo $2n^2$ passos para ou encontrar a variável em y ou concluir que a mesma não ocorre em y. Levamos no máximo mais $2n$ passos para retornar as cabeças ao começo das respectivas fitas.

(iii) Submáquina "Átomo?".

Descrição: Verifica se o símbolo sob a cabeça da fita de entrada e eventualmente símbolos seguintes constituem um átomo, isto é uma seqüência de $p's$ e $q's$, ou v, ou f, ou $\neg$ seguido de uma seqüência de $p's$ e $q's$, ou $\neg v$, ou $\neg f$. Se for um átomo, devolve o valor correspondente segundo a interpretação codificada em y (para isto poderemos chamar "Variável em y?"); caso contrário, rejeita.

Tempo de execução: $\leq 2n^2 + 4n$. Leva no máximo $2n$ passos para descobrir a variável envolvida, caso haja uma, e escrevê-la na fita de trabalho retornando a cabeça da fita de trabalho ao começo da fita. Chamando "Variável em y?" levamos mais $2n^2 + 2n$ passos no máximo para descobrir o valor desta variável e conseqüentemente do átomo. Note que se a entrada for $\neg a$ então "átomo?" devolve v se o valor de a for f, e f em caso contrário.

(iv) Submáquina "Disjunção?"

Descrição: Verifica se o símbolo sob a cabeça de entrada e eventualmente símbolos seguintes constituem uma disjunção, isto é, uma palavra do tipo A_1 ou $(A_1 \vee A_2 \vee \ldots \vee A_r)$ onde $r > 1$ e A_i são átomos, e se a disjunção é verdadeira, isto é, se pelo menos um dos átomos tem valor v. Se não for, rejeita. "Disjunção?" começa verificando se o símbolo sob a cabeça é (. Se não for, então chama "Átomo?" e devolve o comando se o valor devolvido por "Átomo?" for v, e rejeita em caso contrário. Se for, então chama "Átomo?", vê se o símbolo seguinte é $\vee$, e repete este procedimento até encontrar o símbolo) em vez de $\vee$. Se qualquer dos átomos for v, devolve o comando. Caso contrário rejeita. Na Figura 3 damos o diagrama de blocos de "Disjunção?".

Tempo de execução: $\leq 2n^3 + 4n^2 + n$. Cada chamada de "Átomo?" leva no máximo $2n^2 + 4n$ passos, e mais um passo para verificar o símbolo seguinte. Há no máximo n chamadas.

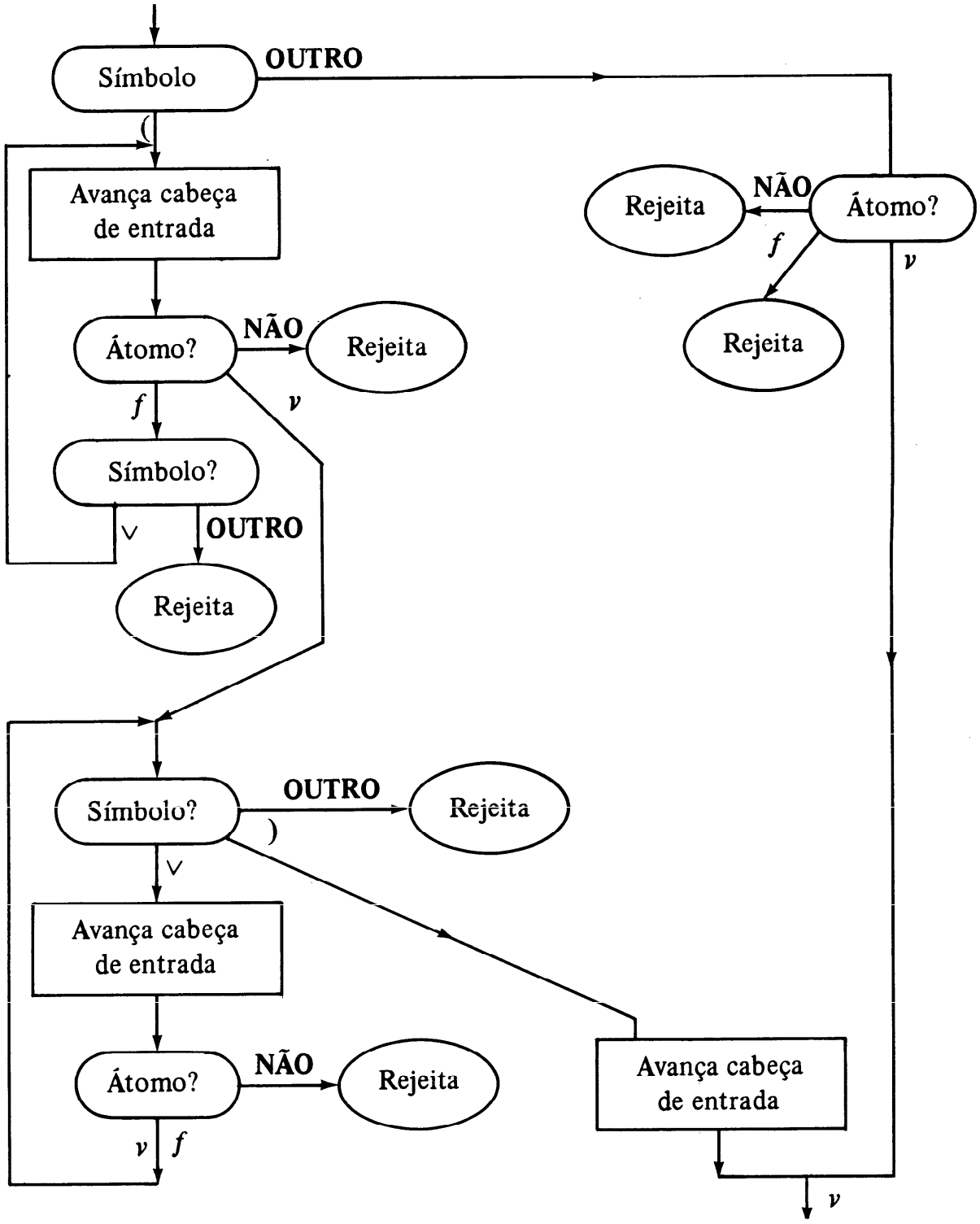


Figura 3 — Diagrama de blocos de "Disjunção?".

Com as submáquinas acima, M pode ser construido como indicado na Figura 4.

Tempo de execução: $\leq 2n^4 + 4n^3 + n^2 + 5n$. "Disjunção?" é chamada no máximo n vezes. ■

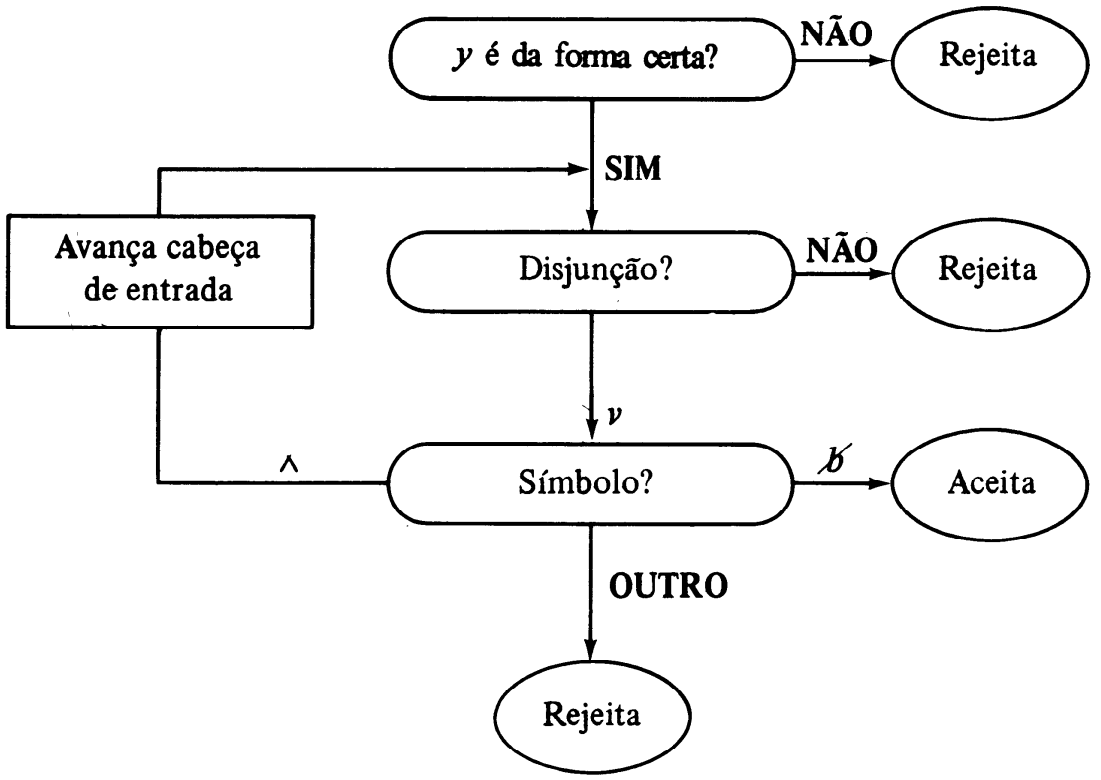


Figura 4 — Diagrama de blocos de M.

5. Redutibilidade e conjuntos NP-m-completos

No Capítulo II vimos uma noção de redutibilidade recursiva, que foi utilizada para mostrar que certos problemas são indecidíveis. Aqui introduzimos um conceito similar, o de *m*-redutibilidade polinomial.

Um conjunto $A \subseteq \Sigma^*$ é *m-redutível em tempo polinomial* a um conjunto $B \subseteq \Sigma^*$, e indicado por $A \leq_m^{\mathscr{P}} B$, se existir uma função $f : \Sigma^* \to \Sigma^*$ computável em tempo polinomial tal que para todo $x \in \Sigma^*$ temos $x \in A$ sse $f(x) \in B$.

PROPOSIÇÃO 1. *Sejam A, B e C subconjuntos de Σ^*.*
(i) *Se $A \in \mathscr{P}$ então para todo $B \subsetneq \Sigma^*$ não vazio $A \leq_m^{\mathscr{P}} B$;*
(ii) *Se $A \leq_m^{\mathscr{P}} C$ e $C \in \mathscr{P}$ então $A \in \mathscr{P}$;*
(iii) *Se $A \leq_m^{\mathscr{P}} C$ e $C \in \mathscr{NP}$ então $A \in \mathscr{NP}$.*

Demonstração. (i) Seja $b \in B$ e $b' \in \Sigma^* \setminus B$. As palavras b e b' existem porque B é não vazio e $B \subsetneq \Sigma^*$. Seja

$$f(x) = \begin{cases} b & \text{se } x \in A, \\ b' & \text{se } x \notin A. \end{cases}$$

A função f é computável em tempo polinomial porque $A \in P$, e $A \leq_m^{\mathcal{P}} B$ via f.

(ii) Se $A \leq_m^{\mathcal{P}} C$ via f então $x \in A$ sse $f(x) \in C$.

A máquina de Turing determinística M seguinte reconhece A em tempo polinomial:

(1) Calculamos $f(x)$ sobre a fita de trabalho (isto leva tempo polinomial, porque f é computável em tempo polinomial);

(2) Aplicamos a máquina de Turing que reconhece C em tempo polinomial à fita de entrada/trabalho/saída que contém $f(x)$. (Esta máquina pára em tempo polinomial $q(|\ f(x)\ |)$ no comprimento da sua entrada, que é $f(x)$. Como f é computável em tempo polinomial, temos $|\ f(x)\ | \leq p(|\ x\ |)$ para alguma polinomial p, pois em cada passo no máximo um símbolo é escrito na saída. Portanto o "passo" (2) leva no máximo tempo $q(p(|\ x\ |))$, que é polinomial.)

Claramente, M reconhece A em tempo polinomial.

(iii) A prova é idêntica ao item (ii), exceto que a máquina de Turing do passo (2) é agora não determinística, e aceita C em tempo polinomial. Com esta modificação obtemos uma máquina não determinística que aceita A em tempo polinomial. ■

Um conjunto $B \subseteq \Sigma^*$ é *$\mathcal{NP}$-m-completo* se

(i) $B \in \mathcal{NP}$,

e

(ii) Para todo $A \in \mathcal{NP}$ temos $A \leq_m^{\mathcal{P}} B$.

COROLÁRIO 2. *Se B é $\mathcal{NP}$-m-completo então $B \in \mathcal{P}$ sse $\mathcal{P} = \mathcal{NP}$.*

Demonstração. Se $\mathcal{P} = \mathcal{NP}$ então $B \in \mathcal{P}$ pois B é *$\mathcal{NP}$-m-completo.*

Reciprocamente, se $B \in \mathcal{P}$, então para todo $A \in \mathcal{NP}$ temos $A \leq_m^{\mathcal{P}} B$. Pela Proposição 1 segue que $A \in \mathcal{P}$. Isto mostra que $\mathcal{NP} \subseteq \mathcal{P}$. Mas pelo Exercício 1, $\mathcal{P} \subseteq \mathcal{NP}$. Portanto $\mathcal{P} = \mathcal{NP}$. ■

O Corolário 2 mostra que se identificarmos algum conjunto B que seja *$\mathcal{NP}$-m-*completo e se acharmos um algoritmo polinomial para B, então podemos reconhecer todos os conjuntos de $\mathcal{NP}$ em tempo polinomial. A classe de conjuntos de $\mathcal{NP}$ é muito ampla e contém um grande número de problemas de interesse prático para os quais não se conhece nenhum algoritmo polinomial. Seria, portanto, um avanço muito grande encontrar um algoritmo polinomial para um dos problemas *$\mathcal{NP}$-m-*completos. Reciprocamente, pela mesma razão parece provável que $\mathcal{P} \neq \mathcal{NP}$. Para provar isto os problemas

$\mathcal{NP}$-*m*-completos são os melhores candidatos para um problema em $\mathcal{NP} \setminus \mathcal{P}$.

Estamos agora em condições de provar o resultado principal deste capítulo.

TEOREMA 4 (Cook). *O conjunto FNC-SAT é $\mathcal{NP}$-m-completo.*

Demonstração. Já vimos no Teorema 2 que *FNC-SAT* $\in \mathcal{NP}$. Precisamos então provar que se $A \in \mathcal{NP}$ então $A \leq_m^{\mathcal{P}}$ *FNC-SAT*. Para isto construiremos, para cada conjunto A aceito em tempo polinomial por uma máquina de Turing não determinística M de fita única, uma função $f_M : \Sigma^*_{e/s} \to \Sigma^*_{e/s}$, computável em tempo polinomial tal que:

(i) $f_M(x)$ é uma fórmula abreviada em forma normal conjuntiva;
(ii) $f_M(x)$ é satisfazível sse $x \in A$, isto é, sse M aceita x.

Estas condições garantem que $A \leq_m^{\mathcal{P}}$ *FNC-SAT* via f_M, e portanto que *FNC-SAT* é $\mathcal{NP}$-*m*-completo.

Seja então $A \in \mathcal{NP}$ e M uma máquina de Turing não determinística de fita única que aceita A em tempo polinomial p. Seja $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_s\}$ o alfabeto de M, onde $\sigma_1 = \vdash$ e $\sigma_2 = \not b$, e $Q = \{q_1, q_2, \ldots, q_e\}$ o conjunto de estados de M, onde q_1 é o estado inicial, q_2 é o estado final q_a, e q_3 o estado final q_r. Seja x uma entrada e seja $T = p(|x|)$. A fórmula $f_M(x)$ terá as seguintes variáveis lógicas com as interpretações padrão indicadas[4]:

VARIÁVEL	INTERPRETAÇÃO PADRÃO
$Estado_t^k$, $1 \leq k \leq e$, $1 \leq t \leq T$;	v sse M no passo t estiver no estado q_k;
$Símbolo\text{-}T_{c,t}^i$, $1 \leq i \leq s$, $1 \leq c, t \leq T$	v sse a célula c da fita de entrada/trabalho/saída contiver o símbolo σ_i no passo t;
$Símbolo\text{-}S_c^i$, $1 \leq i \leq s$, $1 \leq c \leq T$;	v sse a célula c da fita de sugestões contiver o símbolo σ_i;
$Posição\text{-}T_t^c$, $1 \leq c, t \leq T$;	v sse a cabeça da fita de entrada/trabalho/saída estiver sobre a célula c no passo t;
$Posição\text{-}S_t^c$, $1 \leq c, t \leq T$;	v sse a cabeça da fita de sugestões estiver sobre a célula c no passo t.

(4) Na realidade os nomes mnemônicos das variáveis devem ser substituidos por palavras de $\{p, q\}^*$, de acordo com a Seção 4.

Note que o número de variáveis é $eT + sT^2 + sT + T^2 + T^2$ Como $T = p(|\ x\ |)$, e s e e são constantes, temos um número de variáveis polinomial no comprimento da entrada x. Note ainda que, em no máximo T passos, as cabeças de M não podem afastar-se mais que T células para a direita.

A fórmula $f_M(x)$ será construída de tal forma que se M com sugestão y aceitar x em t passos, com $t \leq T$, então $f_M(x)$ com a interpretação padrão torna-se verdadeira. Em outras palavras, se M aceitar x então $f_M(x)$ será satisfazível.

Reciprocamente, se uma interpretação de $f_M(x)$ a tornar verdadeira, então dela podemos extrair uma sugestão y, tal que M com sugestão y aceita x em tempo $t \leq T$.

A fórmula $f_M(x)$ é a conjunção:

$$B \wedge C \wedge D \wedge E \wedge F \wedge G \wedge H \wedge I$$

onde as subfórmulas com a interpretação padrão afirmam:

B – a cabeça da fita de entrada/trabalho/saída em cada passo encontra-se sobre uma e só uma célula;

C – a cabeça da fita de sugestões em cada passo encontra-se sobre uma e só uma célula;

D – cada célula da fita entrada/trabalho/saída em cada passo contém um e um só símbolo;

E – cada célula da fita de sugestões contém um e um só símbolo (note que o conteúdo da fita de sugestões, por ser de leitura apenas, não depende do passo t);

F – em cada passo, M encontra-se em um e um só estado;

G – inicialmente o estado de M é q_1, a fita de entrada/trabalho/saída contém $\vdash x$, a primeira célula da fita de sugestões contém $\vdash$, e as cabeças encontram-se sobre a primeira célula das respectivas fitas;

H – descreve as transições de M;

I – M aceita x em algum passo $t \leq T$.

Cada uma das subfórmulas estará em forma normal conjuntiva abreviada, e assim $f_M(x)$ estará em forma normal conjuntiva abreviada. Descrevemos agora cada uma das subfórmulas. Note que usaremos $\bigwedge_{1 \leq i \leq \ell} A_i$ para denotar a fórmula abreviada $A_1 \wedge A_2 \wedge \ldots \wedge A_\ell$, e $\bigvee_{1 \leq i \leq \ell} A_i$ para denotar a fórmula abreviada $A_1 \vee A_2 \vee \ldots \vee A_\ell$.

$B = \bigwedge_{1 \le t \le T} B_t$, onde

$$B_t = \left(\bigvee_{1 \le c \le T} \textit{Posição-}T_t^c \right) \wedge \bigwedge_{1 \le i < j \le T} (\neg \textit{Posição-}T_t^i \vee \neg \textit{Posição-}T_t^j).$$

Note que $(\neg \textit{Posição-}T_t^i \vee \neg \textit{Posição-}T_t^j)$ afirma que no passo t a cabeça não pode estar simultaneamente sobre as células i e j, onde $i < j$. Assim, B_t afirma que no passo t a cabeça está sobre uma e somente uma das células da fita de entrada/saída/trabalho, e B afirma que isto é verdade em cada passo. As fórmulas C, D, E e F se obtêm analogamente:

$C = \bigwedge_{1 \le t \le T} C_t$, onde

$$C_t = \left(\bigvee_{1 \le c \le T} \textit{Posição-}S_t^c \right) \wedge \bigwedge_{1 \le i < j \le T} (\neg \textit{Posição-}S_t^i \vee \neg \textit{Posição-}S_t^j).$$

$D = \bigwedge_{1 \le c,t \le T} D_{c,t}$ onde

$$D_{c,t} = \left(\bigvee_{1 \le i \le s} \textit{Símbolo-}T_{c,t}^i \right) \wedge$$

$$\bigwedge_{1 \le i < j \le s} (\neg \textit{Símbolo-}T_{c,t}^i \vee \neg \textit{Símbolo-}T_{c,t}^j).$$

$E = \bigwedge_{1 \le c \le T} E_c$, onde

$$E_c = \left(\bigvee_{1 \le i \le s} \textit{Símbolo-}S_c^i \right) \wedge$$

$$\bigwedge_{1 \le i < j \le s} (\neg \textit{Símbolo-}S_c^i \vee \neg \textit{Símbolo-}S_c^j).$$

$F = \bigwedge_{1 \le i \le T} F_t$, onde

$$F_t = \left(\bigvee_{1 \le k \le e} \textit{Estado}_t^k \right) \wedge \bigwedge_{1 \le i < j \le e} (\neg \textit{Estado}_t^i \vee \neg \textit{Estado}_t^j).$$

$G = \textit{Estado}_1^1 \wedge \textit{Posição-}T_1^1 \wedge \textit{Posição-}S_1^1 \wedge$
$\wedge \textit{Símbolo-}S_1^1 \wedge \textit{Símbolo-}T_{1,1}^1 \wedge$
$\wedge \textit{Símbolo-}T_{2,1}^{i_1} \wedge \textit{Símbolo-}T_{3,1}^{i_2} \wedge \ldots \wedge \textit{Símbolo-}T_{n+1,1}^{i_n} \wedge$
$\wedge \textit{Símbolo-}T_{n+2,1}^2 \wedge \textit{Símbolo-}T_{n+3,1}^2 \wedge \ldots \wedge \textit{Símbolo-}T_{T,1}^2$,
onde $x = \sigma_{i_1} \sigma_{i_2} \ldots \sigma_{i_n}$.

$H = H^1 \wedge H^2$, onde

$$H^1 = \bigwedge_{\substack{1 \le i \le s \\ 1 \le c \le T \\ 1 \le t < T}} (\textit{Posição-}T_t^c \vee \neg \textit{Símbolo-}T_{c,t}^i \vee \textit{Símbolo-}T_{c,t+1}^i).$$

(Note que H^1 afirma que, se a cabeça não estiver sobre a célula c da fita de entrada/trabalho/saída no passo t, então o símbolo contido nesta célula não é modificado neste passo.)

$$H^2 = \bigwedge_{\substack{1 \le k \le e \\ 1 \le i,j \le s \\ 1 \le c_1, c_2, t < T}} H_{k,i,j,c_1,c_2,t}, \text{ onde,}$$

supondo que $\delta(q_k, \sigma_i, \sigma_j) = (q_{k'}, \sigma_{i'}, \Delta_1, \Delta_2)$, temos:

$$\begin{aligned}
&H_{k,i,j,c_1,c_2,t} = \\
&(\neg \mathit{Estado}_t^k \vee \neg \mathit{Posição\text{-}T}_t^{c_1} \vee \neg \mathit{Posição\text{-}S}_t^{c_2} \vee \neg \mathit{Símbolo\text{-}T}_{c_1,t}^i \vee \neg \mathit{Símbolo\text{-}S}_{c_2}^j \vee \mathit{Estado}_{t+1}^{k'}) \wedge \\
\wedge\ &(\neg \mathit{Estado}_t^k \vee \neg \mathit{Posição\text{-}T}_t^{c_1} \vee \neg \mathit{Posição\text{-}S}_t^{c_2} \vee \neg \mathit{Símbolo\text{-}T}_{c_1,t}^i \vee \neg \mathit{Símbolo\text{-}S}_{c_2}^j \vee \mathit{Símbolo\text{-}T}_{c_1,t+1}^{i'}) \wedge \\
\wedge\ &(\neg \mathit{Estado}_t^k \vee \neg \mathit{Posição\text{-}T}_t^{c_1} \vee \neg \mathit{Posição\text{-}S}_t^{c_2} \vee \neg \mathit{Símbolo\text{-}T}_{c_1,t}^i \vee \neg \mathit{Símbolo\text{-}S}_{c_2}^j \vee \mathit{Posição\text{-}T}_{t+1}^{c_1+\Delta_1}) \wedge \\
\wedge\ &(\neg \mathit{Estado}_t^k \vee \neg \mathit{Posição\text{-}T}_t^{c_1} \vee \neg \mathit{Posição\text{-}S}_t^{c_2} \vee \neg \mathit{Símbolo\text{-}T}_{c_1,t}^i \vee \neg \mathit{Símbolo\text{-}S}_{c_2}^j \vee \mathit{Posição\text{-}S}_{t+1}^{c_2+\Delta_2})
\end{aligned}$$

(Assim, H^2 afirma que para todo passo $t < T$, se M estiver no estado q_k, a cabeça da fita de entrada/saída/trabalho estiver sobre a célula c_1 que contiver o símbolo σ_i, e a cabeça da fita de sugestões estiver sobre a célula c_2 que contiver o símbolo σ_j, então no passo $t+1$ a máquina M estará no estado $q_{k'}$, a célula c_1 da fita de entrada/saída/trabalho conterá $\sigma_{i'}$, e as cabeças estarão respectivamente sobre as células $c_1 + \Delta_1$ e $c_2 + \Delta_2$).

Finalmente

$$I = \bigvee_{1 \le t \le T} \mathit{Estado}_t^2.$$

Claramente, se M aceitar x com alguma sugestão $y = \sigma_{j_1} \sigma_{j_2} \ldots \sigma_{j_m}$, então a interpretação padrão das variáveis torna $f_M(x)$ verdadeira, e assim, a fórmula $f_M(x)$ é neste caso satisfazível. Reciprocamente, se $f_M(x)$ for satisfazível para alguma interpretação das variáveis então sejam

$$\mathit{Símbolo\text{-}S}_1^{j_1},\ \mathit{Símbolo\text{-}S}_2^{j_2},\ \ldots,\ \mathit{Símbolo\text{-}S}_T^{j_T}$$

as variáveis $\mathit{Símbolo\text{-}S}_c^j$ verdadeiras nesta interpretação. (Como a interpretação torna E verdadeira, para cada c existe um e um só j tal que $\mathit{Símbolo\text{-}S}_c^j$ é verdadeira.) Conseqüentemente M aceita x com sugestão $y = \sigma_{j_1} \sigma_{j_2} \ldots \sigma_{j_T}$, pois as variáveis de $f_M(x)$ que são verdadeiras nesta interpretação descrevem, quando reinterpretadas pela interpretação padrão, a computação de M que aceita x com esta sugestão. Assim, $f_M(x)$ é satisfazível sse M aceita x com alguma sugestão y, isto é

$$f_M(x) \in \mathit{FNC\text{-}SAT} \text{ sse } x \in A.$$

Como o comprimento de cada subfórmula de $f_M(x)$ é polinomial em $|x|$, temos que $|f_M(x)|$ é polinomial em $|x|$. Seguindo a descrição acima de $f_M(x)$, podemos construir uma máquina de Turing que com entrada x escreve a fórmula $f_M(x)$ em tempo polinomial. Assim, $f_M(x)$ é computável em tempo polinomial, o que completa a demonstração. ■

COROLÁRIO 3. *O conjunto SAT é $\mathcal{NP}$-m-completo.*

Demonstração. A função $f_M(x)$ do Teorema 4 está em forma normal conjuntiva abreviada para qualquer x. Portanto, como $A \leq_m^{\mathcal{P}} FNC\text{-}SAT$ via $f_M(x)$, é imediato que $f_M(x)$ também m-reduz A a SAT em tempo polinomial. Pelo Exercício 8, $SAT \in \mathcal{NP}$. Portanto SAT é $\mathcal{NP}$-m-completo. ■

6. Outros problemas NP-m-completos

Após a descoberta de Cook de que $FNC\text{-}SAT$ e SAT são $\mathcal{NP}$-m-completos, constatou-se que uma lista extensa de problemas de importância prática, lista que continua crescendo a cada ano, são também $\mathcal{NP}$-m-completos. Note que uma vez demonstrado que um conjunto A é $\mathcal{NP}$-m-completo, basta mostrar que $A \leq_m^{\mathcal{P}} B$ e $B \in \mathcal{NP}$, para demonstrar que B também é $\mathcal{NP}$-m-completo. Estas reduções, em geral, são bem mais simples do que a redução que vimos no Teorema 4. Karp [57] exibiu uma lista de 21 problemas $\mathcal{NP}$-m-completos dos quais extraímos seis no teorema abaixo.

Percebeu-se, então, que as soluções rápidas para um grande número de problemas de interesse estão interrelacionadas no sentido de que ou cada um deles ou nenhum deles pode ser resolvido rapidamente por um algoritmo. Isto explica a importância prática da questão $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$.

Também do ponto de vista teórico, vê-se que esta é uma questão central a ser respondida para se ter um melhor entendimento de algoritmos rápidos. Com efeito, devido ao Teorema 4, responder à questão $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$ resultará num avanço significativo nesta área, qualquer que seja a resposta: se $\mathcal{P} = \mathcal{NP}$ então teremos algoritmos polinomiais para todos os problemas em $\mathcal{NP}$, que, como se viu, inclui um grande número de problemas para os quais não dispomos de algoritmos polinomiais no momento; se, por outro lado $\mathcal{P} \neq \mathcal{NP}$, então sabe-

remos que um grande número de problemas, os problemas $\mathcal{NP}$-*m*-completos, não podem ser resolvidos por algoritmos polinomiais.

Damos abaixo, sem demonstração, alguns dos problemas $\mathcal{NP}$-*m*-completos descobertos por Karp.

TEOREMA 5. *Os problemas seguintes são* $\mathcal{NP}$-*m-completos*:

(i) *Programação inteira* $0 - 1$:
$\{(C, \mathbf{d}) \mid C$ *é uma matriz inteira,* $\mathbf{d}$ *é um vetor, e existe um vetor* $\mathbf{x}$ *sobre* $\{0, 1\}$, *tal que* $C\mathbf{x} \geq \mathbf{d}\}$;

(ii) *Classificação ótima (veja as definições necessárias no Capítulo C. VI)*:
$\{(D, k) \mid o$ *grafo orientado* D *admite uma classificação* f, *tal que no máximo* k *arestas de* D *são inversões com relação a* $f\}$;

(iii) *Circuito hamiltoniano (veja as definições necessárias no Capítulo C. I)*:
$\{G \mid o$ *grafo* G *tem um circuito que passa por todos os vértices de* $G\}$;

(iv) *Número cromático (veja as definições necessárias no Capítulo C.IV)*:
$\{(G, k) \mid o$ *grafo* G *é* k-*colorável*$\}$;

(v) *Emparelhamento em três dimensões (confronte com o Capítulo C.III)*:
$\{(T, U) \mid U \subseteq T \times T \times T, T$ *é um conjunto finito, e existe um subconjunto* W *de* U, *tal que* $|W| = |T|$ *e para cada* $i = 1, 2, 3$ $\bigcup_{(w_1, w_2, w_3) \in W} \{w_i\} = T\}$;

(vi) *Problema da mochila*:
$\{(n, a_1, a_2, \ldots, a_n, b) \mid n, a_1, \ldots, a_n, b \in \mathbb{N}, n \geq 1,$ *e existem* $x_1, x_2, \ldots, x_n \in \{0, 1\}$, *tais que* $\sum_{1 \leq i \leq n} a_i x_i = b\}$.

Como vimos, não se sabe se $\mathcal{P} = \mathcal{NP}$ ou $\mathcal{P} \neq \mathcal{NP}$. A hipótese mais popular, embora não unânime, entre os cientistas de computação, parece ser $\mathcal{P} \neq \mathcal{NP}$, que aparenta ser mais plausível tendo em vista o grande número de problemas de interesse em $\mathcal{NP}$ para os quais não se conhecem algoritmos polinomiais. Não se sabe, tampouco, se $\mathcal{NP}$ é fechado sob complementação. Como $\mathcal{P}$ é obviamente fechado sob complementação é claro que se $\mathcal{NP}$ não for fechado sob complementação então $\mathcal{P} \neq \mathcal{NP}$. Note que $\mathcal{NP}$ é fechado sob complementação sse o complemento de algum conjunto $\mathcal{NP}$-*m*-completo pertencer a $\mathcal{NP}$. Outra vez, a hipótese mais popular, e pelas mesmas razões, é que $\mathcal{NP}$ não é fechado sob complementação.

Na Seção 2 descrevemos uma máquina de Turing não determinística que aceita o conjunto *COMPOSTO* de todas as palavras que

são a representação decimal de um número composto. Esta máquina é polinomial, e portanto *COMPOSTO* pertence a $\mathcal{NP}$. Menos evidente é o fato de que o complemento de *COMPOSTO*, o conjunto *PRIMO* das palavras que representam números primos na notação decimal, também pertence a $\mathcal{NP}$. Não se sabe se *PRIMO* pertence a $\mathcal{P}$, embora Miller tenha provado que se a hipótese estendida de Riemann for verdadeira então *PRIMO* pertence a $\mathcal{P}$. Não se sabe, tampouco, se *PRIMO* é $\mathcal{NP}$-*m*-completo, mas parece provável que não seja, pois se fosse então $\mathcal{NP}$ seria fechado sob complementação.

EXERCÍCIOS

1. Verifique que $\mathcal{P}$ é um subconjunto de $\mathcal{NP}$.
2. Verifique que os pares de fórmulas abaixo são logicamente equivalentes
   (i) $(p \vee \neg p)$ e v;
   (ii) $(p \wedge \neg p)$ e f;
   (iii) $(p \wedge (pp \wedge pq))$ e $((p \wedge pp) \wedge pq)$ (propriedade associativa de $\wedge$);
   (iv) $(p \vee (pp \vee pq))$ e $((p \vee pp) \vee pq)$ (propriedade associativa de $\vee$);
   (v) $(p \wedge q)$ e $(q \wedge p)$ (propriedade comutativa de $\wedge$);
   (vi) $(p \vee q)$ e $(q \vee p)$ (propriedade comutativa de $\vee$);
   (vii) $(p \wedge (pp \vee pq))$ e $((p \wedge pp) \vee (p \wedge pq))$ (propriedade distributiva de $\wedge$ sobre $\vee$);
   (viii) $(p \vee (pp \wedge pq))$ e $((p \vee pp) \wedge (p \vee pq))$ (propriedade distributiva de $\vee$ sobre $\wedge$);
   (ix) $\neg(p \wedge q)$ e $(\neg p \vee \neg q)$ (lei de DeMorgan);
   (x) $\neg(p \vee q)$ e $(\neg p \wedge \neg q)$ (lei de DeMorgan);
   (xi) p e $\neg\neg p$;
   (xii) v e $\neg f$;
   (xiii) f e $\neg v$.
3. Mostre que para toda fórmula A existem fórmulas B e C, logicamente equivalentes a A, respectivamente em forma normal disjuntiva e conjuntiva. Note que B e C podem ser fórmulas exponencialmente mais compridas do que A.
4. Mostre que *FND-SAT* e *FNC-TAUT* pertencem a $\mathcal{P}$.
5. Seja $\equiv_m^{\mathcal{P}}$ a relação definida por $A \equiv_m^{\mathcal{P}} B$ sse $A \leq_m^{\mathcal{P}} B$ e $B \leq_m^{\mathcal{P}} A$. Mostre que $\equiv_m^{\mathcal{P}}$ é uma relação de equivalência.

6. Seja $p(x)$ um polinômio com coeficientes inteiros, e seja $\leq_m^{p(n)}$ a relação definida por: $A \leq_m^{p(n)} B$ sse existir uma função $f: \Sigma^* \to \Sigma^*$ computável em tempo $p(n)$ tal que $x \in A$ sse $f(x) \in B$. Prove que $\leq_m^{p(n)}$ não é transitiva.
7. Prove que existe uma máquina de Turing determinística que tendo por entrada uma fórmula em forma normal conjuntiva dá por saída a fórmula abreviada correspondente em tempo polinomial.
8. Prove que *SAT* pertence a $\mathcal{NP}$.
9. Como *SAT* pertence a $\mathcal{NP}$ pelo exercício anterior, o Teorema 4 garante que dada uma fórmula x existe uma fórmula z de comprimento polinomial em $|x|$ e em forma normal conjuntiva, tal que z é satisfazível sse x é satisfazível. Exiba uma tal fórmula explicitamente. Confronte com o Exercício 3.
10. Mostre que cada um dos problemas do Teorema 5 está em $\mathcal{NP}$.
11. Seja *k-FNC-SAT* o subconjunto de *FNC-SAT* tal que cada fórmula de *k-FNC-SAT* é uma conjunção de disjunções de no máximo k átomos. Mostre que
    (i) *k-FNC-SAT* pertence a $\mathcal{NP}$.
    (ii) Se $k \geq 3$ *k-FNC-SAT* é $\mathcal{NP}$-*m*-completo.
    (iii) 2-*FNC-SAT* pertence a $\mathcal{P}$.
12. Sejam $H_1 = \{G \mid G$ é um grafo orientado que tem um circuito orientado que passa por todos as vértices de $G\}$, e $H_2 = \{G \mid G$ é um grafo que tem um circuito que passa por todos os vértices de $G\}$. Prove que:
    (i) $H_2 \leq_m^{\mathcal{P}} H_1$;
    (ii) $H_1 \leq_m^{\mathcal{P}} H_2$.
13. Construa um grafo orientado G com três vértices *iniciais* v_1, v_2 e v_3, e três correspondentes vértices *finais* v_1', v_2' e v_3' tal que:
    (i) Dados quaisquer i vértices iniciais ($i = 1, 2$ ou 3), existem i caminhos orientados disjuntos nos vértices, com origens respectivamente nos vértices dados e términos respectivamente nos vértices finais correspondentes, caminhos tais que em cada um dos vértices de G passa exatamente um dos i caminhos. (A união disjunta dos vértices dos i caminhos é, portanto, VG.)
    (ii) Para cada $i = 1, 2$ ou 3, *não* existem i caminhos orientados disjuntos nos vértices, com origens em vértices iniciais e

términos em vértices finais, e tais que em cada vértice de G passe exatamente um dos i caminhos, *exceto* se para cada um dos i caminhos o seu término for o vértice final correspondente à sua origem.
(Veja as definições necessárias nos Capítulos C.I e C.VI.)

14. Mostre que $3\text{-}FNC\text{-}SAT \leq_m^{\mathscr{P}} H_1$ (vide Exercícios 11 e 12). Note que isto demonstra, com os Exercícios 10, 11, e 12, o item (iii) do Teorema 5. (Sugestão: Use um grafo orientado como o construído no Exercício 13 para representar cada disjunção da fórmula.)
15. Demonstre o Teorema 5.

NOTAS BIBLIOGRÁFICAS

A classe $\mathscr{P}$ foi primeiro definida em [16] por Cobham. O artigo de Edmonds [23] contém uma discussão informal de o que deve ser considerado um algoritmo eficiente e sugere também nas entrelinhas a classe $\mathscr{P}$. Cook provou em [17] os resultados principais deste capítulo, e caracterizou pela prima vez a questão $\mathscr{P} \stackrel{?}{=} \mathscr{NP}$ precisamente. Karp [57] exibiu uma lista de 21 problemas $\mathscr{NP}$-m-completos, demonstrando o impacto da questão levantada por Cook. Meyer e Stockmeyer, [80] e [115], provaram os primeiros resultados estabelecendo limites inferiores superpolinomiais, e mesmo superexponenciais para vários problemas naturais. O resultado que $PRIMO \in \mathscr{NP}$ encontra-se em [96], e [81] contém o algoritmo mais rápido conhecido para reconhecer *PRIMO*. O melhor limite superior conhecido para este algoritmo é exponencial, mas a hipótese estendida de Riemann implicaria que o algoritmo é polinomial [81].

PARTE C

TEORIA DOS GRAFOS

CAPÍTULO I

GRAFOS E SUBGRAFOS

1. Grafos e grafos simples

Muitas situações podem ser convenientemente descritas através de diagramas que consistem de um conjunto de pontos, juntamente com linhas que ligam alguns pares destes pontos. Por exemplo, os pontos poderiam representar pessoas, as linhas ligando pares de amigos; os pontos poderiam representar centros de comunicações, as linhas ligações entre os centros. A abstração matemática de situações desse tipo dá lugar ao conceito de grafo (veja Figura 1).[1]

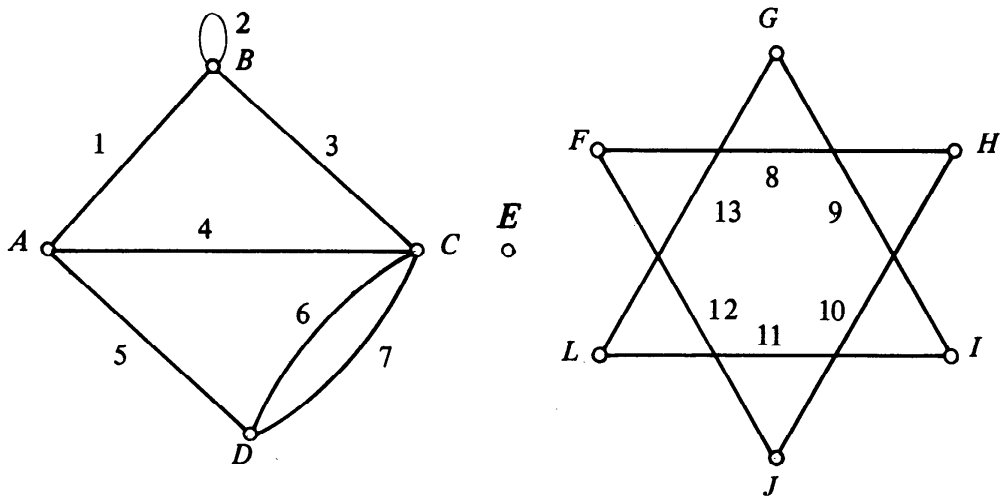
Um *grafo* G consiste de um conjunto VG de elementos chamados *vértices*, um conjunto aG de elementos chamados *arestas* e uma *função de incidência* ψG que associa a cada aresta α de G um par não ordenado de vértices (não necessariamente distintos) de G, chamados de *extremos* de α.

Grafos podem ser representados por diagramas, onde cada vértice é representado por um ponto e cada aresta por uma linha ligando os pontos que representam seus extremos. (Subentende-se que nenhuma linha passa por pontos que representem vértices outros que os extremos da aresta correspondente.)

Convém ressaltar que duas arestas no diagrama de um grafo podem se interceptar num ponto que não representa um vértice. Os grafos que têm um diagrama cujas arestas não se interceptam a não ser nos extremos são ditos *planares*. O grafo da Figura 1 é planar (veja Exercício 1), ao passo que o grafo da Figura 2 não é planar.

Muitos termos utilizados na teoria dos grafos advêm da representação em diagramas. Assim, os extremos de uma aresta são *incidentes* à aresta, e vice-versa. Os extremos de uma aresta são *adjacentes* (mesmo que coincidam); são *adjacentes* também arestas com pelo menos um extremo em comum. Para X um subconjunto de VG, $AdjG(X)$ denota o conjunto dos vértices de G adjacentes a, pelo menos, um dos vértices de X. Uma aresta é um *laço* se seus extremos coincidem, uma

(1) – O termo *grafo* não consta, ainda, nos dicionários da língua portuguesa. No entanto, o uso é unânime, e preferível em nossa opinião, ao uso do termo gráfico.



Grafo Z: $VZ = \{A, B, C, D, E, F, G, H, I, J, L\}$
$aZ = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13\}$

α	$\psi Z(\alpha)$
1	$\{A, B\}$
2	$\{B\}$
3	$\{B, C\}$
4	$\{A, C\}$
5	$\{A, D\}$
6	$\{C, D\}$
7	$\{C, D\}$
8	$\{F, H\}$
9	$\{G, I\}$
10	$\{H, J\}$
11	$\{I, L\}$
12	$\{F, J\}$
13	$\{G, L\}$

Figura 1 — Um grafo Z e uma representação, em diagrama, de Z.

ligação caso contrário. A aresta 2 do grafo Z da Figura 1 é um laço; as demais arestas desse grafo são ligações.

Um grafo G é *finito* se VG e aG forem ambos finitos; o termo "grafo" designará sempre um grafo finito. O *tamanho* de um grafo G é o inteiro $|VG| + |aG|$. O grafo *vazio* é o grafo de tamanho zero, sem arestas nem vértices.

Um grafo G é *simples* se não tem laços nem duas ligações distintas com o mesmo par de extremos. Um grafo *completo* é um grafo simples cujos vértices são dois a dois adjacentes (veja Figura 2). Um *triângulo* é um grafo completo com exatamente 3 vértices.

Dois grafos G e H são *complementares* se forem ambos simples, com $VG = VH$ e tais que quaisquer dois vértices distintos são adjacentes em G se e somente se não o forem em H.

Um *n-cubo* ($n \geq 1$) é um grafo simples cujos vértices são n-uplas, ordenadas, sobre o conjunto $\{0, 1\}$, e no qual dois vértices são adjacentes se e somente se diferem em exatamente uma coordenada (veja diagrama do 3-cubo na Figura 3).

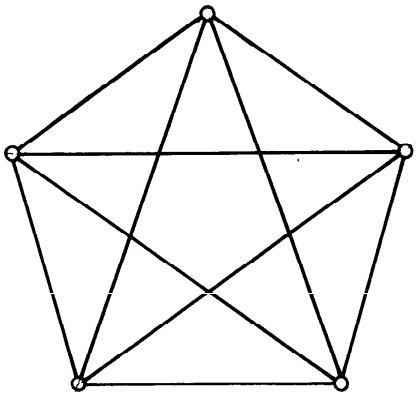


Figura 2 — Um grafo completo com 5 vértices.

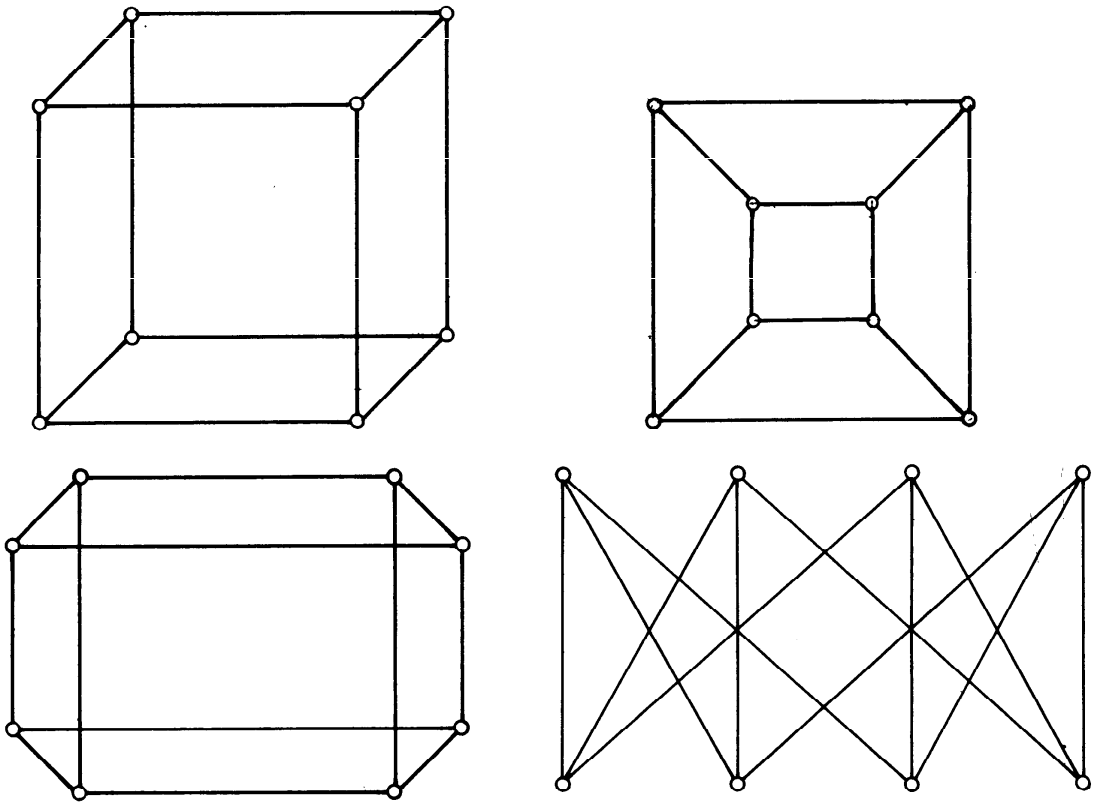


Figura 3 — Quatro representações do cubo (3-cubo).

Um par não ordenado $\{X, Y\}$ de conjuntos de vértices de um grafo G é uma *bipartição* de G se $X \cup Y = VG$, $X \cap Y = \varnothing$ e cada aresta de G tem um extremo em X, o outro em Y. Um grafo G é *biparticionável* se tiver uma bipartição. (Uma das representações do cubo dadas na Figura 3 sugere que o cubo é biparticionável — veja Exercício 3.)

Usaremos normalmente a letra G para designar um grafo; assim, quando não houver possibilidade de ambigüidade omitiremos a letra G e escreveremos por exemplo, V, a, ψ e Adj, ao invés de VG, aG, ψG e $AdjG$, respectivamente.

2. Algumas representações de grafos no computador

Uma maneira de representar um grafo G no computador, e que reflete diretamente a definição de grafo, consiste em representar os conjuntos VG e aG, e a tabela que define a função de incidência ψG. Para tanto, é conveniente "dar novos nomes" aos vértices e às arestas; assim, é comum impor as igualdades[2]

$$VG = \{1, 2, \ldots, |VG|\}$$
$$aG = \{1, 2, \ldots, |aG|\}.$$

A representação de VG torna-se então trivial: basta ter a cardinalidade de VG; a representação de aG fica igualmente simples. A função de incidência ψG pode nesse caso ser representada pelo *vetor de vértices incidentes*, um vetor com $|aG|$ coordenadas em que a j-ésima coordenada é um par (u, v), onde u e v são os extremos da aresta j.

Uma outra estrutura que pode ser usada para representar um grafo G é o *vetor de arestas incidentes*, um vetor com $|VG|$ coordenadas em que a j-ésima coordenada é uma lista das arestas incidentes ao vértice j.

Tanto o vetor de vértices incidentes quanto o vetor de arestas incidentes são representações razoavelmente "concisas" de um grafo. Nas aplicações em que freqüentemente é necessário obter um extremo de uma aresta arbitrária e uma aresta incidente a um vértice arbitrário,

---

(2) — O leitor atento notará que nada há na definição de grafo que exija que os conjuntos de vértices e de arestas de um grafo sejam disjuntos. Contudo, se um elemento é usado como uma aresta e como um vértice, deve-se deixar claro, a cada uso, se se trata do vértice ou da aresta.

pode-se usar o *par de vetores de incidências*, que consiste do vetor de vértices incidentes e do vetor de arestas incidentes.

Nas aplicações em que freqüentemente se torna necessário determinar se dois vértices arbitrários são ou não adjacentes, pode-se lançar mão da *matriz de adjacências*, uma matriz quadrada com $|VG|$ linhas e $|VG|$ colunas, onde o elemento correspondente à i-ésima linha e j-ésima coluna é igual ao número de arestas cujos extremos são os vértices i e j. Convém ressaltar que esta matriz a rigor não representa um grafo, pois "ignora os nomes das arestas".

Outra representação de um grafo $|G|$ é a *matriz de incidências*, uma matriz com $|VG|$ linhas e $|aG|$ colunas, em que o elemento da i-ésima linha e j-ésima coluna é o número de vezes (0, 1 ou 2) que a aresta j incide no vértice i.

3. Isomorfismo entre grafos

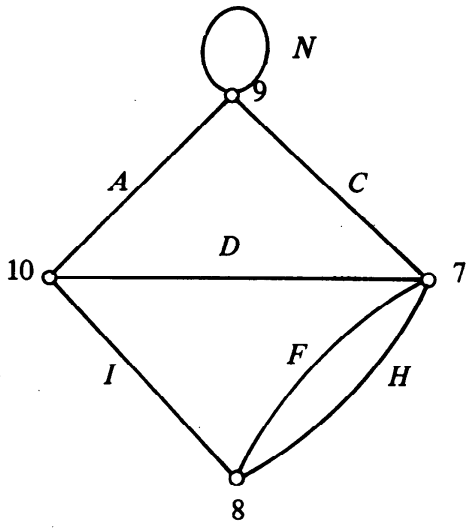
Dois grafos G e H são *iguais* se $VG = VH$, $aG = aH$ e $\psi G = \psi H$. Se dois grafos são iguais então podem certamente ser representados pelo mesmo diagrama. Por outro lado, grafos distintos podem, às vezes, ser representados pelo mesmo diagrama, a menos dos nomes dos vértices e arestas. Por exemplo, o grafo Y, ilustrado na Figura 4, é distinto do grafo Z ilustrado na Figura 1; no entanto, ambos são representados pelo mesmo diagrama. Dizemos então que Y e Z são isomorfos. De fato, é possível "dar novos nomes aos vértices e às arestas de Y, de forma a torná-lo igual a Z".

Formalmente, um grafo G é *isomorfo* a outro H se existirem bijeções $f: VG \to VH$ e $\phi: aG \to aH$ tais que para cada aresta α de G, se u e v são os extremos de α em G, então $f(u)$ e $f(v)$ são os extremos de $\phi(\alpha)$ em H[3]; o par (f, ϕ) é então um *isomorfismo* de G em H. Convém ressaltar que se G é isomorfo a H então H é isomorfo a G: portanto pode-se simplesmente dizer que G e H são isomorfos (veja Exercícios 10 e 22).

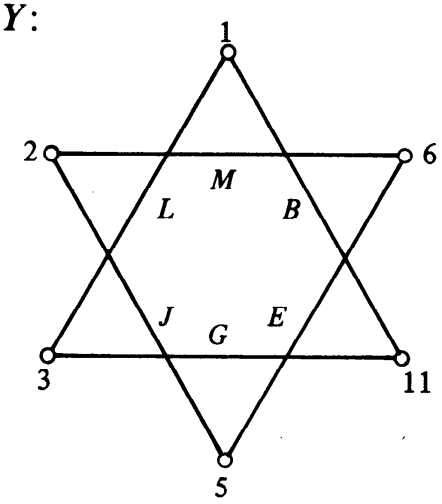
Note que se G e H são grafos simples, então G e H são isomorfos se e somente se existe uma bijeção f de VG em VH tal que, para quaisquer vértices u e v de G, u e v são adjacentes (em G) se e somente se $f(u)$ e $f(v)$ são adjacentes (em H) (veja Exercício 16).

---

(3) Assim, $f[\psi G(\alpha)] = \psi H[\phi(\alpha)]$.



Grafo Y:



4

Isomorfismo (f, ϕ) de Y em Z:

u	$f(u)$
1	G
2	F
3	L
4	E
5	J
6	H
7	C
8	D
9	B
10	A
11	I

α	$\phi(\alpha)$
A	1
B	9
C	3
D	4
E	10
F	6
G	11
H	7
I	5
J	12
L	13
M	8
N	2

Figura 4 — Um grafo Y isomorfo ao grafo Z da figura 1, e um isomorfismo (f, ϕ) de Y em Z.

Um grafo G é *autocomplementar* se for simples e isomorfo a outro H, com G e H complementares.

Um *automorfismo* de G é um isomorfismo de G em G. Um grafo G é *transitivo nos vértices* se para quaisquer vértices u e v existe um automorfismo (f, ϕ) de G tal que $f(u) = v$. Analogamente, G é *transitivo nas arestas* se para quaisquer arestas α e β existe um automorfismo (f, ϕ) de G tal que $\phi(\alpha) = \beta$.

4. Cardinalidade e inclusão. Subgrafos

Dado um conjunto C de conjuntos, dizemos que um conjunto m em C é *máximo* em C se nenhum conjunto em C tem cardinalidade maior do que a de m; analogamente, um conjunto m' de C é *mínimo* em C se nenhum conjunto em C tem cardinalidade menor do que a de m'.

EXEMPLO 1. Dado um grafo G, um *emparelhamento* em G é um conjunto t de ligações de G duas a duas não adjacentes. Considere o grafo da Figura 5, seja C o conjunto dos emparelhamentos nesse grafo. Então nem o emparelhamento $\{2\}$ nem o emparelhamento $\{2, 5\}$ são máximos em C; os emparelhamentos $\{1, 3, 5\}$ e $\{1, 3, 6\}$, por exemplo, são ambos máximos em C. O emparelhamento vazio é o único mínimo em C.

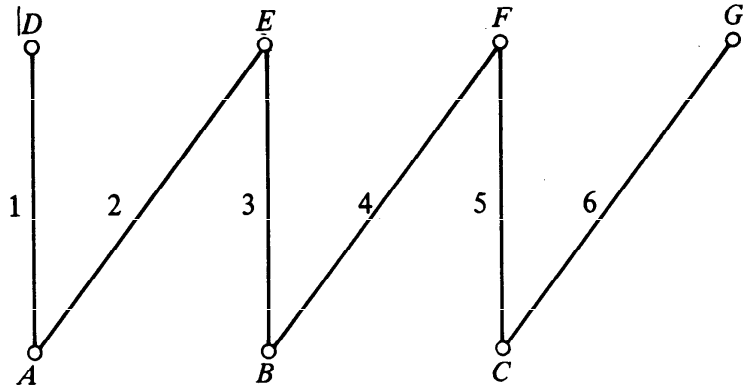


Figura 5 — O grafo dos Exemplos 1 e 2.

Dado um conjunto C de conjuntos, dizemos que um conjunto m em C é *maximal* em C se nenhum conjunto em C inclui propriamente m; analogamente, um conjunto m' em C é *minimal* se nenhum conjunto em C é um subconjunto próprio de m'.

EXEMPLO 2. Utilizando ainda o conjunto C do exemplo anterior, nem o emparelhamento $\{2\}$ nem o emparelhamento $\{2, 4\}$ são maximais em C; os emparelhamentos $\{2, 5\}$ e $\{2, 4, 6\}$ são ambos maximais em C. O emparelhamento vazio é o único minimal em C.

Pode-se demonstrar que todo conjunto finito e não vazio de conjuntos finitos tem elementos máximos e mínimos, e que todo elemento máximo é maximal, todo elemento mínimo é minimal (vide Exercício 27).

Um grafo H é um *subgrafo* de outro G ($H \subseteq G$) se VG inclui VH, aG inclui aH e, para toda aresta de H, seus extremos em H são também seus extremos em G; H é um subgrafo *próprio* de G ($H \subsetneq G$) se H é um subgrafo de G, mas distinto de G. O grafo da Figura 6 é um subgrafo próprio do grafo Z da Figura 1.

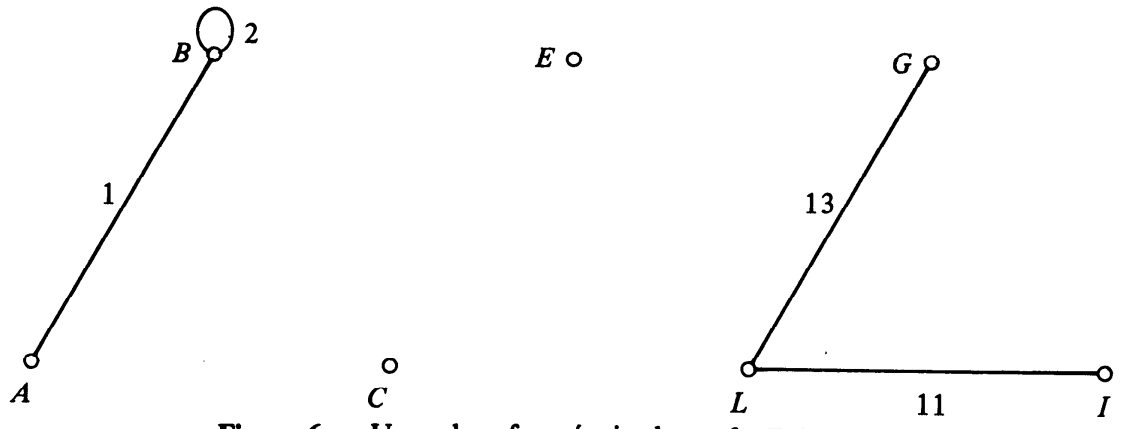


Figura 6 — Um subgrafo próprio do grafo Z da Figura 1.

Dado um conjunto C de grafos, um grafo G é *máximo* em C se nenhum outro grafo em C tem tamanho maior do que o de G; um grafo H é *maximal* em C se H não é subgrafo próprio de nenhum grafo em C. Analogamente se definem grafos mínimos e minimais.

Para X um subconjunto de V, o *subgrafo de G gerado por X*, $G[X]$, é o subgrafo H de G tal que $VH = X$ e aH é o conjunto das arestas de G que têm ambos os extremos em X (vide Exercício 29). Para X um subconjunto de V, v um vértice de V, G-X abrevia $G[V \backslash X]$ e G-v abrevia G-$\{v\}$.

Um grafo H *gera* outro G se H é um subgrafo de G tal que $G = G[VH]$. Grafo H é um *subgrafo gerador* de G se H gera G. Assim, um grafo H gera G se e somente se H é um subgrafo de G com $VH = VG$ (vide Exercício 30).

Para x um subconjunto de a, o *subgrafo de G gerado por x*, $G[x]$, é o subgrafo H de G tal que $aH = x$ e VH é o conjunto dos extremos das arestas em x; por outro lado, G-x denota o subgrafo gerador de G que tem $a \backslash x$ como conjunto de arestas; em particular, para α uma aresta em aG, G-α abrevia G-$\{\alpha\}$ (vide Exercícios 31 e 32).

5. Graus

O *grau* $gG(v)$ de um vértice v em G é o número de arestas incidentes a v, cada laço sendo contado duas vezes. Em todo grafo, a soma

dos graus de seus vértices é igual ao dobro do número de arestas (Exercício 33).

Seja G um grafo com n vértices $1, 2, \ldots, n$; a seqüência $(g(1), g(2), \ldots, g(n))$ é a *seqüência de graus* de G. Uma seqüência de naturais é *gráfica* se for a seqüência de graus de algum grafo simples. Convém ressaltar que existem grafos não isomorfos com a mesma seqüência de graus (vide Exercício 34).

Um grafo é *regular* se todos os seus vértices têm o mesmo grau; caso haja necessidade de se explicitar o grau, k, comum a todos os vértices, diremos então que o grafo é *k-regular*.

6. Passeios

Um *passeio* P em G é uma seqüência finita e não vazia $(v_0, \alpha_1, v_1, \ldots, \alpha_n, v_n)$, cujos termos são alternadamente vértices v_i e arestas α_j, e tal que, para todo i, $1 \leq i \leq n$, v_{i-1} e v_i são os extremos de α_i. Dizemos que P é um passeio de v_0 a v_n; os vértices v_0 e v_n são a *origem* e o *término* de P, respectivamente; os vértices $v_1, \ldots, v_{n-1}$ são *vértices internos* de P; VP e aP denotam os conjuntos $\{v_0; \ldots, v_n\}$ e $\{\alpha_1, \ldots, \alpha_n\}$ respectivamente; o passeio P *passa* por α_i e por v_j ($\alpha_i \in aP$ e $v_j \in VP$). Admite-se o caso em que $n = 0$, no qual P é então dito *degenerado*. O *comprimento* de P é o inteiro n. O passeio $(v_n, \alpha_n, v_{n-1}, \ldots, \alpha_1, v_0)$ é o *reverso* $R(P)$ de P. Uma *seção* de P é um passeio que é uma subseqüência de termos consecutivos de P. Se as arestas $\alpha_1, \ldots, \alpha_n$ de P forem duas a duas distintas então P é uma *trilha*. Se os vértices $v_0, \ldots, v_n$ forem dois a dois distintos então P é um *caminho*. Se a origem e o término de P coincidirem e P não for degenerado então P é *fechado*. Se P for uma trilha fechada e se $v_1, \ldots, v_n$ forem dois a dois distintos então P é um *circuito*.

Sejam P e Q passeios $(v_0, \alpha_1, v_1, \ldots, \alpha_n, v_n)$ e $(u_0, \beta_1, u_1, \ldots, \beta_m, u_m)$ em G, respectivamente, tais que $v_n = u_0$: o passeio $(v_0, \alpha_1, v_1, \ldots, \alpha_n, v_n, \beta_1, u_1, \ldots, \beta_m, u_m)$ é o *produto* PQ de P e Q nesta ordem.

EXEMPLO 3. No grafo Z da Figura 1, $P = (A, 1, B, 3, C, 4, A, 5, D, 5, A, 1, B, 2, B)$ é um passeio de A a B de comprimento 7, A é sua origem, B é seu término, B, C, A, D são internos em P, $VP = \{A, B, C, D\}$, $aP = \{1, 2, 3, 4, 5\}$; P não é uma trilha, nem um caminho, nem um passeio fechado. Por outro lado, $Q = (B, 2, B, 3, C, 4, A)$ é uma trilha não fechada, mas não é um ca-

minho. A seção $(A, 1, B, 3, C)$ de P é um caminho. A seção $(A, 1, B, 3, C, 4, A)$ de P é um circuito, ao passo que a trilha fechada $(A, 1, B, 2, B, 3, C, 4, A)$ não é um circuito.

A *distância* de um vértice u a outro v em G é o mínimo dos comprimentos dos passeios de u a v em G. (Se não existir nenhum passeio de u a v então dizemos que a distância de u a v é infinita.) O *diâmetro* de G é o máximo das distâncias entre vértices de G. A *cintura* de G é o comprimento do menor circuito em G. (Se não houver nenhum circuito então dizemos que a cintura é infinita.)

Vértice u de G é *ligado* ao vértice v de G se existe um passeio de u a v em G; a relação de ligação é certamente de equivalência.

TEOREMA 1. *Seja u um vértice em G. Defina a seqüência $S_0, S_1, \ldots, S_k, \ldots$ de subconjuntos de V indutivamente da seguinte maneira, onde $S_{\leq j}$ denota $\bigcup_{0 \leq i \leq j} S_i$*

$$S_k = \begin{cases} Adj\,(S_{k-1}) \backslash S_{\leq k-1} & \text{se } k > 0, \\ \{u\} & \text{se } k = 0. \end{cases}$$

Então

(i) *para todo k, S_k é o conjunto dos vértices de G cuja distância a u é k;*

(ii) *existe um inteiro ℓ tal que $\ell \leq |V|$, $S_0, S_1, \ldots, S_{\ell-1}$ são todos não vazios e $S_\ell, S_{\ell+1}, \ldots$ são todos vazios.*

Demonstração de (i). Para cada k, seja D_k o conjunto dos vértices que distam k de u, seja $D_{\leq k}$ o conjunto $\bigcup_{0 \leq i \leq k} D_i$.

Provaremos, por indução em k, que $D_k = S_k$. Evidentemente, a igualdade vale para $k = 0$, uma vez que D_0 e S_0 são ambos iguais a $\{u\}$ Suponha pois que $k > 0$. Admita, como hipótese de indução, que $D_r = S_r$ para cada $r \leq k - 1$.

Para provar que $S_k \subseteq D_k$, seja v um vértice em S_k. Por definição de S_k, v pertence a $Adj\,(S_{k-1}) \backslash S_{\leq k-1}$. Por hipótese de indução, v pertence a $Adj\,(D_{k-1}) \backslash D_{\leq k-1}$.

Como v pertence a $Adj\,(D_{k-1})$, então existe um vértice, w, em D_{k-1} e uma aresta, α, com extremos v e w. Como w pertence a D_{k-1}, então existe um passeio, P, de comprimento $k - 1$, de u a w. Ora, o produto $P(w, \alpha, v)$ é um passeio de comprimento k, de u a v; portanto, v pertence a $D_{\leq k}$. Mas v pertence também a $V \backslash D_{\leq k-1}$. Logo, v per-

tence a $D_{\leq k} \backslash D_{\leq k-1}$. Ou seja, v pertence a D_k. Como esta conclusão vale para todo vértice v em S_k, então $S_k \subseteq D_k$.

Para provar que $D_k \subseteq S_k$, seja v um vértice em D_k. Então existe um passeio $P = (u_0, \alpha_1, u_1, \ldots, \alpha_k, u_k)$ de u a v, e de comprimento k. Como $k > 0$, então, $(u_0, \alpha_1, u_1, \ldots, \alpha_{k-1}, u_{k-1})$ é uma seção de P de comprimento $k-1$, de u a u_{k-1}. Logo, u_{k-1} pertence a $D_{\leq k-1}$. Seja $r \leq k$ tal que u_{k-1} pertence a D_{r-1}. Pela hipótese de indução, u_{k-1} pertence a S_{r-1}. Como α_k tem extremos u_{k-1} e v, então v pertence a $Adj\,(S_{r-1})$. Por definição, v pertence então a $S_{\leq r}$ (pois ou v pertence a $S_{\leq r-1}$ ou v pertence a $S_r = Adj\,(S_{r-1}) \backslash S_{\leq r-1}$). Seja $s \leq r \leq k$ tal que v pertence a S_s. Pela hipótese de indução, $s = k$, pois se $s < k$ então v pertence a D_s, contradição, pois v pertence a D_k. Ou seja, o vértice v, de D_k, pertence a S_k. Como esta conclusão vale para todo vértice v em D_k, então $D_k \subseteq S_k$.

De fato, $D_k = S_k$ para todo k. A demonstração de (i) está completa.

Demonstração de (ii) – Como todo passeio de comprimento mínimo entre dois vértices é um caminho (veja Exercício 44), então nenhum vértice dista $|V|$ de u (pois se um passeio tem comprimento $|V|$ então passa pelo menos duas vezes por algum vértice). De (i), $S_{|V|} = \varnothing$. Seja ℓ o menor inteiro tal que $S_\ell = \varnothing$. Então $\ell \leq |V|$ e $S_0, S_1, \ldots, S_{\ell-1}$ são todos não vazios. Ademais, como $Adj\,(\varnothing) = \varnothing$, segue de um argumento indutivo que $S_\ell, S_{\ell+1}, \ldots$ são todos vazios. A demonstração de (ii) completa a demonstração do teorema. ■

Baseados no teorema, descreveremos o algoritmo *distância* (vide Figura 7), que determina o *vetor de distâncias* de u, um vetor com $|V|$ coordenadas onde a i-ésima coordenada é igual à distância de u ao vértice i. O algoritmo pressupõe que G é dado pela quádrupla (*m*, *n*, *arestas*, *vértices*), onde $m = |VG|$, $n = |aG|$, *arestas* é o vetor de arestas incidentes de G e *vértices* o vetor de vértices incidentes.

O primeiro passo do algoritmo é a inicialização do vetor s de distâncias de u e da fila S. O vetor s é inicializado fazendo cada uma de suas coordenadas igual a ∞, exceto a u-ésima, que é igual a zero. A fila S é inicialmente unitária, e consiste do vértice u.

Durante o processo, a fila S consiste dos vértices cujos adjacentes precisam ser examinados. Ademais, se $S = \langle v_1, \ldots, v_p \rangle$, então para algum r, com $0 \leq r \leq p$, todos os vértices $v_1, \ldots, v_r$ têm a mesma distância de u, digamos $k-1$; todos os vértices $v_{r+1}, \ldots, v_p$ distam k de u; cada vértice que dista não mais do que $k-1$ de u ou pertence a $\{v_1, \ldots, v_r\}$ ou seus adjacentes já foram todos examinados.

O marcador * é utilizado nas coordenadas de *arestas* a fim de examinar os conjuntos que representam, deixando o vetor *arestas* intacto ao fim do processo.

Deixamos ao leitor verificar que existem constantes b, c e d tais que *distância* executa no máximo $bm + cn + d$ operações. Dizemos então que *distância* é um algoritmo *linear* (no tamanho do grafo).

```
procedimento distância (m, n, arestas, vértices, u):
    {s é um vetor com n coordenadas que ao fim do processo será o
            vetor de distâncias de u}
  início
    i ← 1;
    enquanto i ≤ n faça s[i], i ← ∞, i + 1;
    s[u], S ← 0, ⟨u⟩;
    enquanto S ≠ ⟨ ⟩ faça
      início
        w, S ← car(S), cdr(S);
        arestas[w], k ← arestas[w] # *, s[w] + 1;
        enquanto car(arestas[w]) ≠ * faça
          início
            α ← car(arestas[w]);
            arestas[w] ← cdr(arestas[w]) # α;
            x1, x2 ← vértices[α][1], vértices[α][2];
            se x1 = w então z ← x2 senão z ← x1;
            se s[z] = ∞ então s[z], S ← k, S # z
          fim;
        arestas[w] ← cdr(arestas[w]) {remove *}
      fim;
    devolva s
  fim
```

Figura 7 — Algoritmo *distância*.

7. Componentes, conexão e cortes

Conforme foi visto na seção anterior, a relação de ligação é de equivalência; portanto, a relação de ligação induz uma partição p de V tal que dois vértices são ligados se e somente se pertencem ao mesmo elemento de p. O *conjunto de componentes* de G, cG, é então definido como sendo igual a $\{G[X] \mid X \in p\}$. Cada grafo em cG é um *componente* de G.

Um grafo é *conexo* se quaisquer dois de seus vértices são ligados. Evidentemente, cada componente de um grafo é conexo. (O grafo Z da Figura 1 não é conexo; a Figura 8 mostra os quatro componentes do grafo Z.)

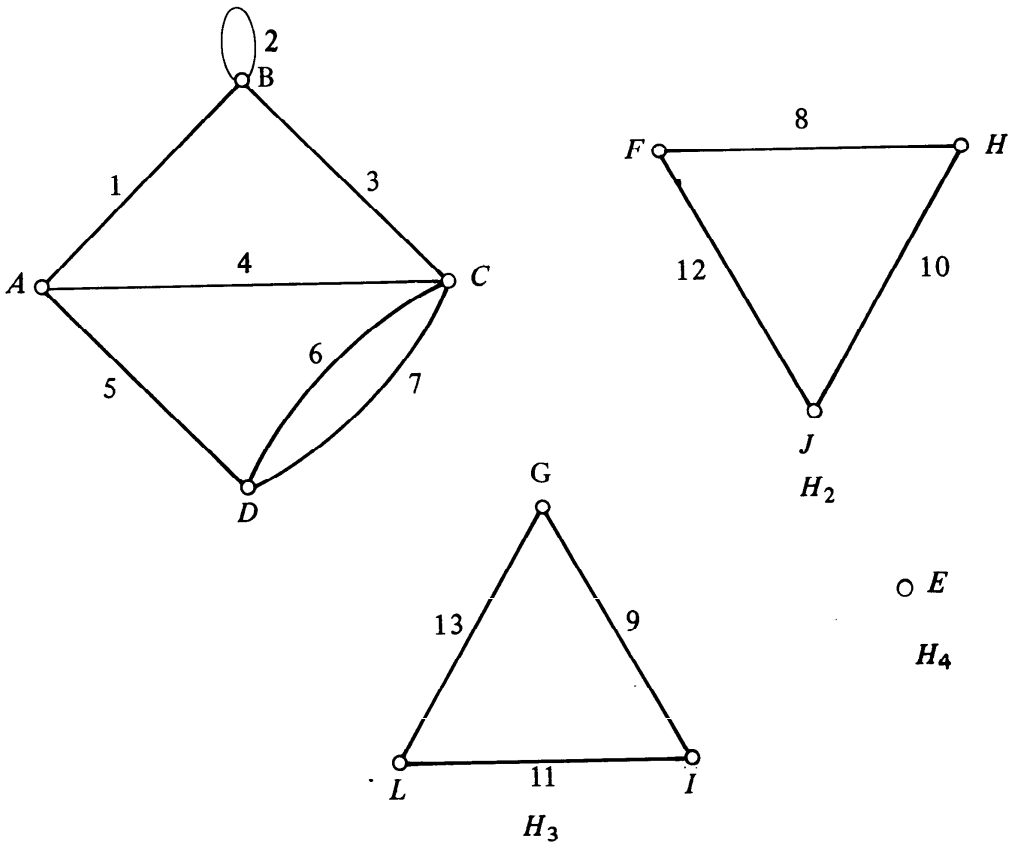


Figura 8 — O grafo Z da Figura 1 e seus quatro componentes H_1, H_2, H_3 e H_4.

Um subconjunto S de V *separa* os vértices u e v se um dos vértices u e v pertence a S, o outro a $V\backslash S$. A cada subconjunto S de V podemos associar o conjunto das arestas cujos extremos S separa; esse conjunto é chamado *o corte associado a* S e é denotado por $\delta G(S)$; assim, $\delta G(S)$ é o conjunto das arestas de G que têm um extremo em S, o outro em $V\backslash S$. Convém ressaltar que $\delta(S) = \delta(V\backslash S)$ e $\delta(\varnothing) = \varnothing = \delta(V)$. Um subconjunto d de aG é um *corte* em G se for o corte associado a algum subconjunto de V. Uma aresta α é *de corte* se $\{\alpha\}$ é um corte. Um corte d *separa* os vértices u e v se for o corte associado a algum subconjunto de V que separa u e v.

Um subgrafo H de G é *isolado* em G se toda aresta de G com pelo menos um extremo em VH é uma aresta de H; assim, H é isolado se e somente se $H = G[VH]$ e $\delta G(VH) = \varnothing$.

EXEMPLO 4. No grafo Z da Figura 8,

(i) o conjunto $S = \{A, C, D, J\}$ separa B e D, separa J e E, mas não separa nem A e J, nem E e I.

(ii) o corte $\delta Z(S)$ associado ao conjunto $S = \{B, C, F\}$ é o conjunto $\{1, 4, 6, 7, 8, 12\}$.

(iii) o conjunto $\{3, 4, 5, 9, 11\}$ é um corte (associado, por exemplo, ao conjunto $\{A, B, F, H, J, I\}$); nem o conjunto $\{8, 10, 12\}$ nem o conjunto $\{1, 4, 6\}$ são cortes.

(iv) não há arestas de corte.

(v) o corte vazio separa os vértices D e J, mas não separa os vértices B e D.

(vi) o subgrafo H_1 de Z é isolado; também é isolado o subgrafo de Z que tem dois componentes, H_1 e H_2; o subgrafo $Z[A, B, C]$ não é isolado; tampouco são isolados os subgrafos X de Z tais que $VX = \{A, B, C, D\}$ e $aX \neq \{1, 2, 3, 4, 5, 6, 7\}$.

TEOREMA 2. *Um grafo H é um componente de G se e somente se H é um subgrafo não vazio, isolado e conexo, de G.*

Demonstração. Afirmar que H é um componente de G equivale a afirmar que

(i) VH é não vazio,

e

(ii) $H = G[VH]$,

e

(iii) para qualquer vértice v em VH, VH é o conjunto dos vértices ligados a v em G.

Considere agora a afirmação

(iii′) $\delta G(VH) = \varnothing$ e $G[VH]$ é conexo.

Pelo Lema 1, abaixo, a validade de (iii′) implica na validade de (iii). Pelos Lemas 1 e 2, abaixo, a validade de (iii) implica na validade de (iii′). Logo (iii) e (iii′) são equivalentes.

Portanto, H é um componente de G se e somente se VH é não vazio, $H = G[VH]$, $\delta G(VH) = \varnothing$ e $G[VH]$ é conexo. De fato, H é um componente de G se e somente se H é um subgrafo não vazio, isolado e conexo, de G.

LEMA 1. *Sejam u e v vértices de G, S um subconjunto de V que separa u e v, P = $(v_0, \alpha_1, v_1, \ldots, \alpha_n, v_n)$ um passeio de u a v. Então aP encontra*[4] *δS.*

Demonstração. Ajuste a notação, permutando S e $V \backslash S$ se necessário, de forma que u pertença a S e v a $V \backslash S$. Como $v_0 = u$, então $v_0 \in S$: seja k o maior inteiro tal que $0 \leq k \leq n$ e $v_k \in S$. Como $v_n = v$, então v_n pertence a $V \backslash S$ e portanto $k < n$: ou seja, $v_k \in S$ e $v_{k+1} \in V \backslash S$. Logo $\alpha_{k+1} \in \delta S$. De fato, aP encontra δS. ■

LEMA 2. *Seja u um vértice de G, S o conjunto dos vértices ligados a u. Então δS é vazio.*

Demonstração. Suponha que, pelo contrário, δS contém uma aresta, α. Seja w o extremo de α em S, v seu extremo em $V \backslash S$. Como w pertence a S, então u e w são ligados; como w e v são adjacentes, então u e v são ligados e portanto $v \in S$, contradição. De fato, δS é vazio. A demonstração do Lema 2 completa a demonstração do Teorema 2. ■■

COROLÁRIO 1 (dos Lemas 1 e 2). *Dois vértices são ligados se e somente se o corte vazio não os separa.*

COROLÁRIO 2. *Um grafo G é conexo se e somente se o corte associado a cada subconjunto próprio e não vazio de V é não vazio.*

TEOREMA 3. *Seja α uma aresta de um grafo G. Então $| cG | \leq$ $\leq | c(G - \alpha) | \leq | cG | + 1$. Ademais, as seguintes afirmações são equivalentes:*

(i) $| c(G - \alpha) | = | cG | + 1$,
(ii) *os extremos de α não são ligados em $G - \alpha$,*
(iii) *a aresta α é de corte,*
(iv) *nenhum circuito passa pela aresta α.*

Demonstração. A equivalência das afirmações (ii), (iii) e (iv) fica a cargo do leitor (vide Exercícios 46 e 70). Para de-

(4) Dizemos que *A encontra B* se *A* e *B* não forem disjuntos.

monstrar a desigualdade constante da asserção, bem como a equivalência das afirmações (i) e (ii), sejam u e v os extremos de α, G' o grafo $G - \alpha$. Para cada vértice x em V, sejam G_x e G'_x as componentes de G e de G', respectivamente, que têm x como vértice. Denotaremos por c_0G e c_0G' os conjuntos dos componentes de G e de G', respectivamente que não têm nem u nem v como vértices.

Valem então as seguintes igualdades:

$$| cG | = | c_0G | + | \{G_u, G_v\} | \tag{1}$$

e

$$| cG' | = | c_0G' | + | \{G'_u, G'_v\} | \tag{2}$$

Como u e v, os extremos de α, são ligados em G, então $G_u = G_v$. Portanto,

$$| \{G_u, G_v\} | = 1. \tag{3}$$

Analogamente,

$$| \{G'_u, G'_v\} | = \begin{cases} 1 \text{ se } u \text{ e } v \text{ são ligados em } G', \\ 2 \text{ caso contrário.} \end{cases} \tag{4}$$

De (1), (2), (3) e (4), pela Proposição 1, abaixo, seguem a desigualdade constante da asserção e a equivalência das afirmações (i) e (ii).

PROPOSIÇÃO 1. $c_0G = c_0G'$.

Demonstração. Seja S um subconjunto de $V\backslash\{u, v\}$. Como u e v são os extremos de α, então $\delta G(S) = \delta G'(S)$ e $G[S] = G'[S]$. Portanto, $G[S]$ é um subgrafo não vazio, conexo e isolado em G se e somente se $G'[S]$ é um subgrafo não vazio, conexo e isolado em G'. Pelo Teorema 2, $G[S]$ é um componente de G se e somente se $G'[S]$ é um componente de G'. Como esta conclusão vale para todo subconjunto S de $V\backslash\{u, v\}$, então, de fato, $c_0G = c_0G'$. A demonstração da proposição completa a demonstração do teorema. ■■

EXERCÍCIOS

1. Dê uma representação para o grafo Z da Figura 1 tal que as arestas não se interceptam a não ser nos extremos. Portanto, Z é um grafo planar.

2. Quantos grafos H existem tais que

$$VH = \{1, 2, 3, 4, 5\} \quad \text{e} \quad aH = \{A, B, C, D, E\}?$$

Quantos grafos simples H existem com os citados conjuntos de vértices e arestas?

3. Mostre que todo k-cubo tem 2^k vértices, $k2^{k-1}$ arestas e é bipartici-onável.

4. Mostre que se G é simples, H é completo e ambos têm exatamente n vértices, então

$$|aG| \leq |aH| = \binom{n}{2}.$$

5. Dê um exemplo de um grafo simples não bipartici-onável que não seja um triângulo, e de tamanho o menor possível.

6. Mostre que o número de mulheres é igual ao de homens em toda festa em que cada pessoa é amiga de precisamente k outras do sexo oposto presentes à festa ($k \geq 1$). *Sugestão*: obtenha o número de pares de amigos em função de k e do número de mulheres, em função de k e do número de homens, representando a situação por um grafo bipartici-onável.

7. (*Teorema da amizade*) Mostre que, se numa festa com pelo menos duas pessoas quaisquer duas pessoas têm exatamente um amigo comum presente à festa, então existe uma pessoa na festa que é amiga de todas.

8. Represente o grafo Z da Figura 1 através de (i) uma matriz de adjacências, (ii) um par de vetores de incidências, dando novos nomes aos vértices e às arestas, se necessário. Escreva algoritmos, tão eficientes quanto possível, que obtêm a matriz de adjacências e o vetor de vértices incidentes, respectivamente, dado o vetor de arestas incidentes, o número de vértices e o número de arestas de um grafo.

9. Obtenha f e ϕ tais que (f, ϕ) seja um isomorfismo de Z (Figura 1) em Y (Figura 4). Quantos isomorfismos de Z em Y existem? Quantos automorfismos tem um grafo completo?

10. Se dois grafos são isomorfos então ambos têm o mesmo número de vértices e o mesmo número de arestas. Mostre, através de um exemplo, que a recíproca é falsa.

11. Mostre que os quatro grafos indicados na Figura 3 são isomorfos.

12. Escreva um algoritmo para determinar se dois grafos dados são ou não isomorfos. Quão eficiente é seu algoritmo? (confronte com Exercício 81)
13. Verifique que, a menos de isomorfismo, existem onze grafos simples com quatro vértices. Quantos grafos existem, a menos de isomorfismo, com quatro vértices?
14. Verifique que, a menos de isomorfismo, existem 34 grafos simples com cinco vértices.
15. Escreva um algoritmo que obtém a lista de grafos com n vértices não isomorfos, dado n. Quão eficiente é seu algoritmo?
16. Mostre que, restrito a grafos simples, o conceito de isomorfismo pode ser encarado como uma bijeção (entre os conjuntos de vértices) que preserva adjacência.
17. Obtenha um grafo autocomplementar com quatro vértices. Dê dois grafos autocomplementares, não isomorfos e com cinco vértices cada um. Mostre que não existem grafos autocomplementares com três vértices.
18. Mostre que se G é autocomplementar então
$$|VG| \equiv 0 \text{ ou } 1 \ (mod\ 4).$$
19. Mostre que para todo natural n tal que $n \equiv 0$ ou $1\ (mod\ 4)$, existem grafos autocomplementares com n vértices.
20. Mostre que existem, a menos de isomorfismo, 8 grafos simples regulares com 6 vértices.
21. Dados isomorfismos (f, ϕ) e (f', ϕ') de G em H e de H em K, respectivamente, a composição destes isomorfismos é o par $(f'f, \phi'\phi)$: mostre que este é um isomorfismo de G em K.
22. Verifique que a relação de isomorfismo entre grafos é de equivalência.
23. Verifique que o conjunto de automorfismos de um grafo é um grupo com relação à operação de composição.
24. Obtenha um grafo simples com seis vértices que tenha exatamente um automorfismo. Mostre que todo grafo simples com mais de um e menos de seis vértices tem mais de um automorfismo.
25. Que pode ser dito a respeito dos conjuntos de automorfismos de dois grafos complementares?

26. Dê um grafo simples que seja transitivo nos vértices mas não nas arestas e outro que seja transitivo nas arestas mas não nos vértices.

27. Seja C um conjunto finito e não vazio de conjuntos finitos. Mostre que C tem elementos máximos e mínimos. Mostre que todo elemento máximo (mínimo) é maximal (minimal).

28. Demonstre as seguintes afirmações:
    (a) todo conjunto não vazio de conjuntos finitos tem um elemento mínimo e todo elemento mínimo é minimal.
    (b) todo conjunto finito e não vazio de conjuntos tem um elemento máximo.
    (c) todo conjunto finito e não vazio de conjuntos tem elementos maximais e minimais.
    (d) conjuntos infinitos de conjuntos podem não ter nem elementos minimais nem elementos maximais.

29. Mostre que $H = G[X]$ se e somente se H é maximal no conjunto dos subgrafos de G cujos vértices pertencem a X.

30. Mostre que H gera G se e somente se H é um subgrafo de G tal que $VH = VG$.

31. Dê um grafo G e um subconjunto x de a tais que $G[a\backslash x]$ e $G - x$ sejam distintos.

32. Mostre que $H = G[x]$ se e somente se H é um subgrafo minimal no conjunto dos subgrafos de G que têm todas as arestas de x.

33. Mostre que para qualquer grafo G,
$$2\,|a| = \sum_{v \in V} g(v).$$
Conclua então que o número de vértices de grau ímpar é par, em qualquer grafo.

34. Dê uma seqüência que seja a seqüência de graus de (pelo menos) dois grafos simples não isomorfos.

35. Obtenha um grafo simples e bipartitionável que não seja isomorfo a nenhum subgrafo de nenhum k-cubo.

36. Mostre que todo grafo é o subgrafo gerado por algum subconjunto do conjunto de vértices de algum grafo regular.

37. Mostre que em toda cidade (com pelo menos dois habitantes) residem duas pessoas com o mesmo número de amigos habitantes na cidade. Formule a asserção em termos de grafos simples.

38. Mostre que para todo $n \geq 3$ existe um grafo simples 3-regular com $2n$ vértices que não tem triângulos como subgrafos.

39. Mostre que uma seqüência $(g(1), \ldots, g(n))$ de naturais é a seqüência de graus de algum grafo se e somente se $\sum_{1 \leq i \leq n} g(i)$ for par.

40. Seja $g = (g(1), \ldots, g(n))$ uma seqüência não vazia de naturais tais que $g(1) \geq g(2) \geq \ldots \geq g(n)$. Mostre que g é gráfica se e somente se $n > g(1)$ e $(g(2) - 1, \ldots, g(i+1) - 1, g(i+2), \ldots, g(n))$ é gráfica, onde i denota $g(1)$. Com base nessa demonstração, descreva um algoritmo que determina se uma seqüência dada é ou não gráfica, e, em caso afirmativo, obtém um grafo (simples) com a seqüência de graus dada.

41. Mostre que existe um grafo simples k-regular com n vértices se e somente se $k < n$ e kn é par.

42. Mostre que todo grafo G sem laços tem um subgrafo gerador bipartitionável H tal que $gH(v) \geq gG(v)/2$ para cada vértice v de G.

43. Dê um exemplo de, ou mostre que não existe:
    (i) um passeio que não seja uma trilha,
    (ii) uma trilha que não seja um caminho,
    (iii) um caminho que não seja uma trilha,
    (iv) um passeio fechado que não seja um circuito,
    (v) dois caminhos cujo produto não seja um caminho.

44. Mostre que um passeio de comprimento mínimo de u a v em G é um caminho.

45. Mostre que existe um passeio de u a v em G se e somente se existe um caminho de u a v em G.

46. Mostre que existe um circuito em G que passa por α se e somente se os extremos de α são ligados em G-α.

47. Mostre que o passeio fechado e não degenerado $P = (v_0, \alpha_1, v_1, \ldots, \alpha_n, v_n)$ $(n \neq 2)$ é um circuito se e somente se $v_1, \ldots, v_n$ forem dois a dois distintos. (Confronte com o Exercício VI.3.)

48. Prove que por toda aresta numa trilha fechada passa algum circuito. Mostre que a afirmação seria falsa se substituíssemos "trilha" por "passeio".

49. Mostre que a operação de produto de passeios é associativa.

50. Mostre que a relação de ligação é de equivalência.

51. Seja γG o mínimo dos graus dos vértices de um grafo G não vazio. Mostre que se G for simples então G tem um caminho de comprimento γG. Para cada n, ache um grafo simples G tal que $n = \gamma G$ e G não tem caminhos de comprimento $n + 1$.

52. Prove que se γG (vide exercício anterior) é maior do que 1 então G tem um circuito. Prove também que se G for simples e $\gamma G > 1$ então G tem um circuito cujo comprimento é maior do que γG.

53. Sejam u, v e w três vértices num grafo G e sejam C e D caminhos de comprimento mínimo de u a v e de u a w, respectivamente. Mostre que se x e y são ambos vértices de VC e de VD então x precede y em C se e somente se x precede y em D. (Um vértice z' *precede* outro z'' num caminho A se existem seções A', A'', A''' tais que $A = A'A''A'''$, z' é o término de A' e z'' o término de A''.)

54. Mostre que se modificarmos a definição de S_k $(k > 0)$ no enunciado do Teorema 1 para
$$S_k = Adj(S_{k-1}) \backslash S_{k-2} \backslash S_{k-1}$$
e adicionarmos a igualdade $S_{-1} = \varnothing$, então o teorema continua válido.

55. Escreva um algoritmo linear que, dados G, vértices u e w de G e vetor de distâncias d de u, determina um caminho de comprimento mínimo de u a w, se u e w são ligados.

56. Dê um algoritmo linear que determina um circuito de comprimento mínimo que passe por uma dada aresta, se existir tal circuito.

57. Dê um algoritmo que determina a cintura de um grafo em tempo $O(n(n+m))$, onde $n = |VG|$ e $m = |aG|$.

58. Mostre que cada elemento $a_{ij}^{(k)}$ da k-ésima potência A^k da matriz de adjacências A é o número de passeios de i e j de comprimento k. Baseando-se nesta propriedade, escreva um algoritmo que determina os vetores de distâncias de cada vértice. Qual é a complexidade do seu algoritmo?

59. Um passeio é *par* (*ímpar*) se seu comprimento é par (ímpar). Da mesma forma como distância, diâmetro e cintura foram definidos com relação a passeios, podemos também definir distâncias, diâmetros e cinturas pares e ímpares. Por exemplo, a *distância par* de u a v é o mínimo dos comprimentos dos passeios pares de u a v. Dê exemplos de passeios pares (ímpares) de u a v, de comprimento igual à distância par (ímpar) de u a v mas que não sejam trilhas.

60. Defina uma seqüência (P_0, I_0), (P_1, I_1), ... de pares ordenados de subconjuntos de V de forma que para cada k, P_k (I_k) é o conjunto dos vértices cuja distância par (ímpar) de u é $2k$ $(2k+1)$. Escreva um algoritmo linear que determina os vetores das distâncias pares e ímpares de um vértice u.

61. Prove que existe um passeio ímpar fechado em G se e somente se existe um circuito ímpar em G. Mostre, através de um exemplo, que a afirmação fica falsa se substituirmos "ímpar" por "par", mesmo se substituirmos "passeio" por "trilha". Dê um algoritmo polinomial que determina a cintura ímpar de um grafo.

62. Dado um conjunto t de arestas de um grafo G, um passeio $P = (v_0, \alpha_1, v_1, \ldots, \alpha_n, v_n)$ é *alternado* (com relação a t) se, para cada $i (1 \leq i \leq n-1)$, uma das arestas α_i e α_{i+1} pertence a t, a outra a $aG \backslash t$. Podemos então definir *distância alternada* da maneira natural. Escreva um algoritmo linear que determina o vetor de distâncias alternadas de um vértice u dado.

63. Faça uma pequena modificação no algoritmo distância de forma a obter também o conjunto dos vértices do componente que tem u como um de seus vértices. Dê um algoritmo linear para a determinação da partição de V induzida pela relação de ligação.

64. Mostre que para quaisquer subconjuntos X e Y de V, $\delta(X \oplus Y) = \delta(X) \oplus \delta(Y)$ onde $\oplus$ denota a operação de diferença simétrica, ou seja, $A \oplus B = A \cup B \backslash A \cap B$.

65. Mostre que todo corte é a união de uma coleção disjunta de cortes não vazios minimais.

66. Mostre que se G é conexo, então $\delta(S)$ é um corte não vazio minimal se e somente se $G[S]$ e $G[V \backslash S]$ são ambos subgrafos próprios e conexos, de G.

67. Seja S um subconjunto de V, $P = (v_0, \alpha_1, v_1, \ldots, \alpha_n, v_n)$ um passeio. Mostre que $|\{i \mid \alpha_i \in \delta(S)\}|$ é ímpar se e somente se S separa v_0 e v_n.

68. Mostre que um conjunto d de arestas é um corte se e somente se $|d \cap aC|$ é par, para todo circuito C. Dê um algoritmo linear que determina se um conjunto dado de arestas é ou não um corte.

69. Mostre que as seguintes afirmações são equivalentes:
    (i) G é biparticionável,
    (ii) aG é um corte em G,
    (iii) nenhum circuito em G tem comprimento ímpar.

70. Demonstre a seguinte afirmação como um corolário do Corolário 1: uma aresta α é de corte se e somente se seus extremos não são ligados em G-α.

71. Mostre que as seguintes afirmações são equivalentes:

    (i) H é um componente de G,
    (ii) H é um subgrafo conexo, não vazio, maximal, de G,
    (iii) H é um subgrafo isolado, não vazio, minimal, de G.

72. Mostre que todo grafo simples com n vértices e mais do que $\binom{n-1}{2}$ arestas é conexo.

73. Mostre que se G é um grafo simples com n vértices e cada um de seus vértices tem grau pelo menos $(n-1)/2$ então G é conexo.

74. Demonstre que dois caminhos de comprimento máximo num grafo conexo têm um vértice em comum.

75. Mostre que se G é não conexo e H e G são complementares então H é conexo. Dê exemplos de grafos complementares G e H que sejam ambos conexos.

76. Mostre que se G tem diâmetro pelo menos 3 e H e G são complementares então H tem diâmetro no máximo 3.

77. Prove que existe um corte com um número ímpar de arestas num grafo se e somente se existe um vértice no grafo cujo grau é ímpar.

78. Um grafo é *euleriano* se é conexo e se todos os seus vértices têm grau par. Mostre que um grafo admite uma trilha que passa por todos os seus vértices e todas as suas arestas se e somente se o grafo é euleriano. (Informalmente, um grafo é euleriano se e somente se um seu diagrama pode ser desenhado sem tirar o lápis do papel, sem passar mais de uma vez por alguma linha, mas voltando o lápis ao ponto de partida.) [32].

79. Mostre que para quaisquer vértices distintos u e v ligados em G, a distância de u a v é igual à cardinalidade de um elemento máximo do conjunto de coleções disjuntas de cortes que separam u e v.

80. Suponha que a cada aresta α de um grafo G seja associado um real não negativo, $p(\alpha)$, chamado o *peso* de α. O *peso* de um passeio $P = (v_0, \alpha_1, v_1, \ldots, \alpha_n, v_n)$ é a soma $p(\alpha_1) + p(\alpha_2) + \ldots + p(\alpha_n)$. Dê um algoritmo polinomial que determina um passeio de peso mínimo entre dois dados vértices. ([22] e [127].) Generalize a asserção do exercício anterior. (Confronte com o Exercício B.III.26.)

81. Alguns problemas em aberto, reconhecidamente difíceis:
   (a) demonstrar que existe um algoritmo polinomial para determinar se a cintura par de um grafo dado é ou não menor do que um inteiro dado, ou então mostrar que o problema é $\mathcal{NP}$-m-completo.
   (b) demonstrar que existe um algoritmo polinomial para determinar se dois grafos dados são ou não isomorfos, ou então mostrar que o problema é $\mathcal{NP}$-m-completo.
   (c) (*conjetura da reconstrução de Ulam*). Sejam G e H grafos com $VG = VH = \{1, 2, \ldots, n\}$ $(n \geq 3)$. Se, para cada i, G-i e H-i são isomorfos, então G e H são isomorfos [123].

82. Mostre que todo grupo finito é isomorfo ao grupo de automorfismos de um grafo simples 3-regular conexo [35].

NOTAS BIBLIOGRÁFICAS

A teoria dos grafos tem sido utilizada em áreas tão díspares do conhecimento humano como análise de circuitos elétricos, pesquisa operacional, teoria da computação, análise numérica, química orgânica, física, topologia, genética e psicologia (vide, por exemplo, [49]).

Leonhard Euler é considerado o primeiro autor em teoria dos grafos; em 1736, o famoso físico e matemático resolveu o problema das pontes de Königsberg [32] (vide Exercício 78).

O conceito de planaridade foi aqui apenas abordado, e como de caráter puramente topológico; no entanto, Kuratowski [65] deu uma caracterização estritamente combinatorial de planaridade.

Algoritmos em grafos têm recentemente recebido muita atenção dos cientistas da computação, especialmente em conexão com o problema da igualdade de $\mathcal{P}$ e $\mathcal{NP}$, conforme foi visto no Capítulo B.IV.

O número de textos em teoria dos grafos é razoavelmente grande. Destacamos [6], [3] e [49] como textos de caráter geral, [1] e [60] como textos que enfocam complexidade de algoritmos e estruturas de dados e [4] como um enfoque algébrico de teoria dos grafos.

CAPÍTULO II

FLORESTAS

1. Florestas e árvores

Uma *floresta* é um grafo sem circuitos. Uma *árvore* é uma floresta conexa. (O grafo da Figura 1 é uma floresta com seis componentes, cada um dos quais é uma árvore.)

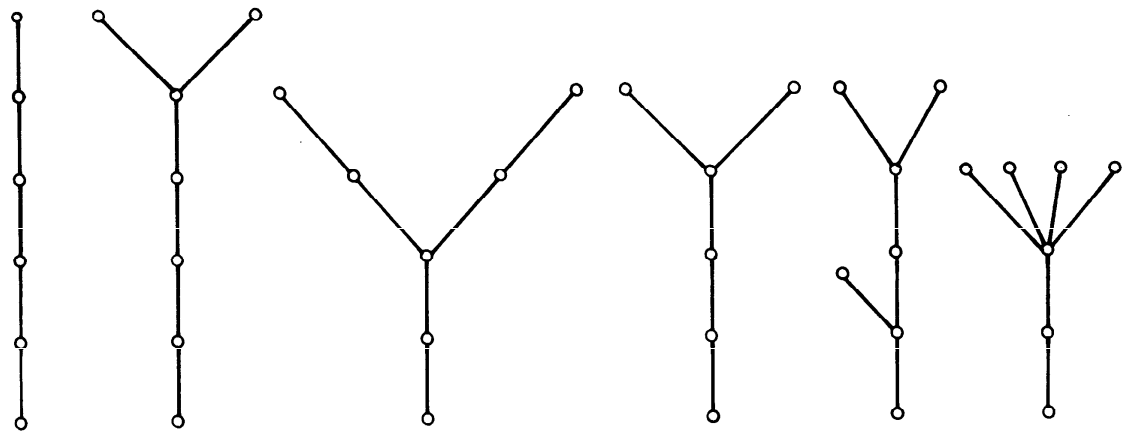


Figura 1 — Uma floresta com seis árvores.

O Teorema 1, abaixo enunciado, é um corolário do Teorema I.3:

TEOREMA 1. *Um grafo é uma floresta se e somente se cada uma de suas arestas é de corte.*

TEOREMA 2. *Para todo grafo* G, $|VG| \leq |aG| + |cG|$. *Ademais a igualdade subsiste se e somente se* G *é uma floresta.*

Demonstração. Por indução em $|aG|$. Se aG for vazio então certamente G é uma floresta e $|VG| = |aG| + |cG|$.

Suponha então que aG contém uma aresta, α. Por indução,

$$|V(G\text{-}\alpha)| \leq |a(G\text{-}\alpha)| + |c(G\text{-}\alpha)|,$$

e a igualdade subsiste se e somente se $G\text{-}\alpha$ é uma floresta.

Pelo Teorema I.3,

$$|c(G\text{-}\alpha)| \leq |cG| + 1,$$

e a igualdade subsiste se e somente se não existe em G circuito que passe por α.

Ora, $V(G\text{-}\alpha) = VG$ e $|a(G\text{-}\alpha)| = |aG| - 1$. Portanto $|VG| \leq |aG| + |cG|$. Ademais a igualdade subsiste se e somente se $G\text{-}\alpha$ é uma floresta e não existe em G circuito que passe por α. Ou seja, $|VG| = |aG| + |cG|$ se e somente se G é uma floresta. ■

TEOREMA 3. *Um grafo G é uma floresta se e somente se cada um de seus subgrafos com pelo menos uma aresta tem pelo menos dois vértices com grau igual a* 1.

Demonstração. Considere inicialmente, o caso em que G é uma floresta. Seja H um subgrafo de G com pelo menos uma aresta, α, e sejam u e v os extremos de α. Como G é uma floresta, então u e v são distintos. Portanto, (u, α, v) é um caminho em H que passa por α. Dentre os caminhos em H que passam por α, escolha um, C, que não seja seção de nenhum outro caminho em H que passe por α. Como VH é finito, então C existe. Sejam $n, v_0, v_1, \ldots, v_n, \alpha_1, \ldots, \alpha_n$ tais que $C = (v_0, \alpha_1, v_1, \ldots, \alpha_n, v_n)$. Vamos agora mostrar que v_0 e v_n são distintos e têm ambos grau 1 em H.

Para mostrar que v_0 e v_n são distintos, basta observar que C é um caminho que passa por α: portanto $n > 0$ e $v_0, v_1, \ldots, v_n$ são dois a dois distintos.

Para mostrar que o grau de v_0 é igual a 1 em H, basta provar que a ligação α_1 é a única aresta em H incidente a v_0. Para tanto, seja β uma aresta em H incidente a v_0, seja x o extremo de β distinto de v_0. Pela escolha de C, x pertence a VC (pois caso contrário $(x, \beta, v_0)C$ é um caminho em H que passa por α, é distinto de C e tem C como seção). Seja i tal que $x = v_i$ $(0 \leq i \leq n)$. Como G é uma floresta, então o caminho $(v_0, \alpha_1, v_1, \ldots, \alpha_i, v_i)$ passa por β (pois caso contrário $(v_0, \alpha_1, v_1, \ldots, \alpha_i, v_i)$ (v_i, β, v_0) seria um circuito em G). Finalmente, β, uma aresta incidente a v_0 pela qual passa o caminho C, é igual a α_1, pois nenhuma aresta em $\{\alpha_2, \ldots, \alpha_n\}$ incide em v_0. De fato, a ligação α_1 é a única aresta em H que incide em v_0 e portanto $gH(v_0) = 1$. Analogamente, a ligação α_n é a única aresta em H que incide em v_n e portanto $gH(v_n) = 1$.

Para completar a demonstração do teorema, resta agora considerar o caso em que G não é uma floresta. Seja então C um circuito em G. O subgrafo $G[aC]$ de G tem pelo menos uma aresta e é 2-regular. A demonstração do Teorema 3 está completa. ■

TEOREMA 4. *Um grafo G é uma floresta se e somente se não existem duas trilhas distintas com mesma origem e mesmo término.*

Demonstração. Considere inicialmente o caso em que G é uma floresta. Sejam T_1 e T_2 trilhas em G, ambas com a mesma origem, u, e mesmo término, v. Provaremos, por indução na soma dos comprimentos das trilhas T_1 e T_2, que $T_1 = T_2$. Para tanto, observe que $G[aT_1 \cup aT_2]$, um subgrafo de G, é também uma floresta; pelo Teorema 3, ou $aT_1 \cup aT_2$ é vazio ou $G[aT_1 \cup aT_2]$ tem dois vértices distintos, ambos com grau 1.

Se $aT_1 \cup aT_2$ é vazio, então $T_1 = (u) = (v) = T_2$. Suponha pois que $G[aT_1 \cup aT_2]$ tem dois vértices distintos, ambos com grau 1. Como cada vértice interno de T_1 ou de T_2 tem grau maior do que 1, então u e v são distintos, e ambos têm grau 1 em $G[aT_1 \cup aT_2]$. Portanto, existem α, u', T_1' e T_2' tais que $T_1 = (u, \alpha, u')T_1'$ e $T_2 = (u, \alpha, u')T_2'$. Por indução, $T_1' = T_2'$. Logo, $T_1 = T_2$.

Resta agora considerar o caso em que G não é uma floresta. Seja então C um circuito em G, u a sua origem. As trilhas (u) e C são distintas, ambas têm a mesma origem e o mesmo término. A demonstração do teorema está completa. ■

2. Subflorestas maximais

Dado um grafo G, um subgrafo H de G e uma aresta α de G, $H + \alpha$ denota o subgrafo de G obtido a partir de H adicionando-se a aresta α (e, eventualmente, os extremos de α); assim, $H + \alpha$ é o subgrafo de G que tem $VH \cup \psi(\alpha)$ como conjunto de vértices e $aH \cup \{\alpha\}$ como conjunto de arestas.

LEMA 1. *Seja H uma subfloresta de um grafo G, α uma aresta de G. Se $H + \alpha$ não é uma floresta então o conjunto $\{aC \mid C$ é um circuito em $H + \alpha\}$ é unitário.*

Demonstração Suponha que $H + \alpha$ não é uma floresta, seja C um circuito em $H + \alpha$. Como H é uma floresta, então C passa por α. Logo, se u e v denotam os extremos de α, então ou (u, α, v) ou (v, α, u) é uma seção de C. Uma vez que $aC = aR(C)$, podemos ajustar a notação, considerando o reverso $R(C)$ de C se necessário, de forma que (u, α, v) seja uma seção de C. Sejam então C_1 e C_2 trilhas em H tais que $C = C_1(u, \alpha, v)C_2$; C_2C_1 é então uma trilha de v a u em H. Logo, $aC = \{\alpha\} \cup aT$, onde T é uma trilha de v a u em H. Como esta conclusão valê para todo circuito C de $H + \alpha$ e como T é única pelo Teorema 4, então $\{aC \mid C$ é um circuito em $H + \alpha\}$ é de fato unitário. ■

TEOREMA 5. *Cada subfloresta F de um grafo G é um subgrafo de alguma subfloresta máxima de G.*

Demonstração. Dentre as subflorestas máximas de G, escolha uma, H, tal que $aF \backslash aH$ seja minimal.

Vamos provar que F é um subgrafo de H. Ora, H, sendo máxima, certamente gera G. Basta portanto provar que aF é um subconjunto de aH.

Para tanto, suponha que, pelo contrário, $aF \backslash aH$ contém uma aresta, α. Pela definição de H, $H + \alpha$ não é uma floresta: seja C um circuito em $H + \alpha$. Como F é uma floresta e α é uma de suas arestas, então C passa por alguma aresta em $aH \backslash aF$, digamos, β. Seja H' o grafo $(H + \alpha) - \beta$. Pelo Lema 1, H' é uma floresta. De fato, H' é uma subfloresta de G cujo tamanho é idêntico ao de H; ademais, $aF \backslash aH'$, igual a $aF \backslash aH \backslash \{\alpha\}$, é um subconjunto próprio de $aF \backslash aH$, em contradição à definição de H. Portanto, F é um subgrafo da subfloresta máxima H de G. ■

COROLÁRIO 1. *Uma subfloresta de um grafo é máxima se e somente se for maximal.*

Com base no Corolário 1, apresentamos o algoritmo da Figura 2, para a obtenção de uma subfloresta máxima de um grafo G.[1]

---

(1) Tanto neste como noutros algoritmos apresentados a seguir, incluimos novas construções, de significado óbvio, mas que não constam da definição dada no Capítulo A.I.

procedimento *subfloresta máxima* (G):
  início
    $VF, aF \leftarrow VG, \varnothing$; {$F$ é o subgrafo de G, sem arestas, que gera G}
    para cada *aresta* α *de* aG **faça**
      se $F + \alpha$ *é uma floresta* **então** $F \leftarrow F + \alpha$;
    devolva F
  fim

Figura 2 — Um algoritmo para a determinação de uma subfloresta máxima de um grafo G (para determinar se $F+\alpha$ é ou não uma floresta, basta determinar se passa ou não um circuito por α em $F+\alpha$; vide Exercício 1.46).

EXERCÍCIOS

1. Mostre que um grafo é uma floresta se e somente se cada um de seus componentes é uma árvore.
2. Mostre que para todo grafo G não vazio, as seguintes afirmações são equivalentes:
   (i) G é uma árvore,
   (ii) $|V| = |a| + 1$ e G é uma floresta,
   (iii) $|V| = |a| + 1$ e G é conexo,
   (iv) para w e x vértices quaisquer de G existe exatamente uma trilha de w a x em G.
3. Seja G um grafo tal que $|a| \geq |V| + 1$, α uma aresta de G. Mostre que G tem um circuito que não passa por α.
4. Um emparelhamento t de um grafo G *cobre* um subconjunto X de VG se cada vértice em X é incidente a uma aresta de t. Mostre que nenhuma floresta G tem mais do que um emparelhamento que cobre VG.
5. Prove que todo grafo conexo G com pelo menos uma ligação tem dois vértices distintos, v_1 e v_2, tais que $G - v_1$ e $G - v_2$ são ambos conexos.
6. Dado um grafo conexo G e um vértice v de G, a *excentricidade* de v em G é igual ao máximo do conjunto de distâncias de v aos vértices de G. Um vértice de G é um *centro* de G se sua excentricidade em G for mínima. Mostre que toda árvore tem no máximo dois centros, e se tiver exatamente dois centros então estes são adjacentes.

7. Caracterize a classe dos grafos G nos quais não existe mais do que um caminho de u a v em G, quaisquer que sejam os vértices u e v de G.

8. Seja $d = (d_1, d_2, \ldots, d_n)$ uma seqüência de números naturais positivos. Mostre que existe uma árvore cuja seqüência de graus é d se e somente se $\Sigma d_i = 2(n-1)$. Mostre que existe uma floresta cuja seqüência de graus é d se e somente se Σd_i for par e menor do que $2n$.

9. Um *matróide* M consiste de um conjunto finito aM de elementos chamados *arestas* e um conjunto CM de elementos chamados *circuitos*, que satisfazem as seguintes propriedades:
   (i) cada circuito em CM é um subconjunto não vazio de aM,
   (ii) nenhum circuito em CM é subconjunto de outro circuito em CM,
   (iii) se C_1 e C_2 são circuitos distintos e α é uma aresta que pertence a C_1 e a C_2 então existe um circuito em CM que é subconjunto de $(C_1 \cup C_2)\backslash\{\alpha\}$.

   Mostre que para qualquer grafo G, se *cir* G denota o conjunto $\{aC \mid C$ é um circuito em $G\}$ então $(aG, \textit{cir } G)$ é um matróide. Mostre que para qualquer grafo G, se *cor* G denota o conjunto de cortes minimais não vazios de G, então $(aG, \textit{cor } G)$ é um matróide.

10. Considere a seguinte definição alternativa de matróide: um matróide M consiste de um conjunto finito aM de elementos chamados *arestas* e um conjunto IM de subconjuntos de aM, chamados *conjuntos independentes*, que satisfazem as seguintes propriedades:
   (i) o conjunto vazio é independente,
   (ii) todo subconjunto de um conjunto independente é independente,
   (iii) para cada subconjunto x de aM, todos os subconjuntos de x independentes e maximais têm a mesma cardinalidade.

   Mostre que se IM denota o conjunto $\{aF \mid F$ é uma subfloresta de $G\}$ então (aG, IM) é um matróide, de acordo com esta definição alternativa. Mostre que se IM denota o conjunto $\{z \mid z \subseteq aG$ e todo corte não vazio de G tem uma aresta em $aG\backslash z\}$ então (aG, IM) é um matróide, de acordo com esta definição alternativa.

11. Mostre que se (aM, CM) é um matróide, de acordo com a definição dada no Exercício 9, e se IM denota o conjunto $\{z \mid z \subseteq aM$ e todo elemento em CM tem uma aresta em $aM\backslash z\}$ então (aM, IM)

é um matróide, de acordo com a definição alternativa, dada no Exercício 10.

12. Mostre que se (aM, IM) é um matróide, de acordo com a definição alternativa, dada no Exercício 10, e se CM denota o conjunto de elementos minimais do conjunto $\{z \mid z \subseteq aM \text{ e } z \notin IM\}$ então (aM, CM) é um matróide de acordo com a definição dada no Exercício 9.

13. Mostre que se S é um conjunto finito e não vazio e A é um conjunto de subconjuntos de S tal que $|A| = |S|$, então existe um elemento x de S tal que $|Ax| = |S|$, onde Ax denota o conjunto $\{X \cup \{x\} \mid X \in A\}$.

14. Mostre que um grafo G é conexo se e somente se G tem uma subárvore geradora. Escreva um algoritmo linear que determina se G é conexo e, em caso afirmativo, dê também uma subárvore geradora de G. Escreva um algoritmo linear que obtém uma subfloresta máxima de um grafo.

15. (*Algoritmo de Kruskal*). Suponha que associado a cada aresta α de um grafo G existe um número real $p(\alpha)$, chamado *peso* de α. O *peso* $p(H)$ de um subgrafo H de G é então definido como a soma dos pesos de suas arestas. Demonstre a seguinte proposição, emulando a demonstração do Teorema 5: "seja F uma subfloresta de G, seja z o conjunto das arestas α em $aG \backslash aF$ tais que $F + \alpha$ é uma floresta, seja α uma aresta em z, de peso máximo, seja $\mathscr{F}$ o conjunto das subflorestas de G que têm F como subfloresta. Então $F + \alpha$ é subfloresta de alguma floresta de peso máximo em $\mathscr{F}$". Com base nesta propriedade, dê um algoritmo polinomial que determina uma subfloresta de peso máximo de um grafo, dados o grafo e a função peso, esta através de uma tabela.

16. Mostre que as seguintes afirmações são equivalentes, para qualquer grafo G conexo:
    (i) H é um subgrafo gerador conexo minimal de G,
    (ii) H é um subgrafo gerador conexo mínimo de G,
    (iii) H é uma subárvore geradora de G,
    (iv) H é uma subárvore maximal de G,
    (v) H é uma subárvore máxima de G.

17. Modifique o algoritmo de Kruskal (Exercício 15) de forma a obter outro que determina uma subárvore geradora de peso mínimo, de um grafo conexo G.

18. Mostre que um grafo G é uma floresta se e somente se cada subconjunto de aG é um corte em G.

19. Mostre que se uma floresta G tem um vértice v cujo grau $g(v)$ é positivo, então G tem pelo menos $g(v)$ vértices com grau 1.

20. Sejam F e H subflorestas maximais de um grafo G. Mostre que para cada aresta α em $aF \backslash aH$ existe uma aresta, β, em $aH \backslash aF$ tal que $(H+\alpha)-\beta$ e $(F+\beta)-\alpha$ são ambas subflorestas maximais de G.

21. Considere o "problema da mochila": dados $n+1$ números reais positivos $x_1, \ldots, x_n$ e y, determine um subconjunto I de $\{1, 2, \ldots, n\}$ tal que
    (i) $\sum_{i \in I} x_i \leq y$
    e
    (ii) $\sum_{i \in I} x_i$ seja o maior possível.
    Mostre que um algoritmo gananciosо não obtém um I ótimo (vide Notas Bibliográficas).

NOTAS BIBLIOGRÁFICAS

O algoritmo da Figura 2 é uma versão simplificada do algoritmo de Kruskal [64] (vide Exercício 15). Algoritmos como esses são exemplos típicos de algoritmos "gananciosos" (o algoritmo da Figura 2, por exemplo, "toma a primeira aresta α" que pode ser adicionada a F de forma que $F+\alpha$ seja ainda uma floresta). Este tipo de algoritmo pode ser usado sempre que maximal e máximo forem indistingüíveis (vide Exercícios 9, 10, 11 e 12). O conceito de matróide (Exercício 9) foi introduzido por Whitney [128] e bastante desenvolvido por Tutte [121], [122]. A definição alternativa de matróide (Exercício 10) foi dada por Edmonds, que também relaciona matróides e algoritmos gananciosos em [24].

Uma versão bastante eficiente ($O(|aG| log_2 log_2 |VG|)$) do algoritmo de Kruskal é dada em [131].

CAPÍTULO III

EMPARELHAMENTOS E COBERTURAS

1. O problema dos casamentos

"Dado um conjunto P de rapazes e outro N de moças, qual a condição necessária e suficiente para que seja possível casar todas as moças em N com rapazes em P, de forma que o esposo de cada moça seja um dos rapazes de quem ela goste?" Esta pergunta é conhecida como o problema dos casamentos e tem a seguinte resposta: "para cada conjunto X de moças em N, o número de rapazes dos quais as moças em X gostam tem que ser pelo menos tão grande quanto o número de moças em X".

Damos a seguir uma versão formal do problema e de sua solução, esta conhecida como o Teorema de Hall.

Um conjunto t de arestas de um grafo G *cobre* um conjunto X de vértices de G se cada vértice em X incide em pelo menos uma aresta em t. Um conjunto t de arestas de G é um *emparelhamento* se t consiste de ligações duas a duas não adjacentes.

TEOREMA 1 (Hall). *Um grafo G com bipartição $\{P, N\}$ tem um emparelhamento que cobre N se e somente se* $|\, Adj(X) \,| \geq |\, X \,|$ *para cada subconjunto X de N.*

LEMA 1. *A condição de Hall é necessária (ou seja, se existe um emparelhamento em G que cobre N então* $|\, Adj(X) \,| \geq |\, X|$ *para cada subconjunto X de N).*

Demonstração. Suponha que G tem um emparelhamento que cobre N. Seja t um tal emparelhamento, X um subconjunto de N. Vamos provar que $|\, Adj(X) \,| \geq |\, X \,|$ definindo uma injeção f de X em $Adj(X)$.

Para tanto, seja v um vértice em X. Como t cobre N, então v é o extremo de (exatamente) uma aresta em t, digamos, $\alpha(v)$; seja $f(v)$ o extremo de $\alpha(v)$ em P. Ora, para v e v' vértices distintos em X, $\alpha(v)$ e $\alpha(v')$ são arestas distintas e como t é um emparelhamento então $f(v)$ e

$f(v')$ são vértices distintos em $Adj(X)$. De fato, f é uma injeção de X em $Adj(X)$ e portanto $|\,Adj(X)\,| \geq |\,X\,|$. Como esta conclusão vale para qualquer subconjunto X de N, então o lema está demonstrado. ■

LEMA 2. *A condição de Hall é suficiente (ou seja, se* $|\,Adj(X)\,| \geq |\,X\,|$ *para todo subconjunto* X de N, *então* G *tem um emparelhamento que cobre* N*).*

Demonstração. Suponha que $|\,Adj(X)\,| \geq |\,X\,|$ para todo subconjunto X de N. Suponha, como hipótese indutiva, que para todo grafo G' com bipartição $\{P', N'\}$ e com $|\,VG'\,| < |\,VG\,|$, se $|\,AdjG'(X)\,| \geq |\,X\,|$ para todo subconjunto X de N', então G' tem um emparelhamento que cobre N'.

Note que se aG é vazio então $Adj(N)$ é vazio: nesse caso, como, por hipótese, $|\,Adj(N)\,| \geq |\,N\,|$, então N é vazio e portanto o emparelhamento vazio cobre N. Suponha então que aG contém (pelo menos) uma aresta, digamos, α. Seja u o extremo de α em P, v seu extremo em N.

Seja L o grafo G-$\{u, v\}$. Note que $\{P\backslash\{u\}, N\backslash\{v\}\}$ é uma bipartição de L. Ora, se $|\,AdjL(X)\,| \geq |\,X\,|$ para todo subconjunto X de $N\backslash\{v\}$, então, por indução, L tem um emparelhamento, t, que cobre $N\backslash\{v\}$; nesse caso, $t \cup \{\alpha\}$ é um emparelhamento em G que cobre N.

Podemos então supor que $N\backslash\{v\}$ tem um subconjunto, S, tal que $|\,AdjL(S)\,| < |\,S\,|$. Por hipótese, $|\,Adj(S)\,| \geq |\,S\,|$. Pela definição de L, $Adj(S) \subseteq \{u\} \cup AdjL(S)$. Logo, u pertence a $Adj(S)$ e $|\,Adj(S)\,| = |\,S\,|$ (vide Figura 1).

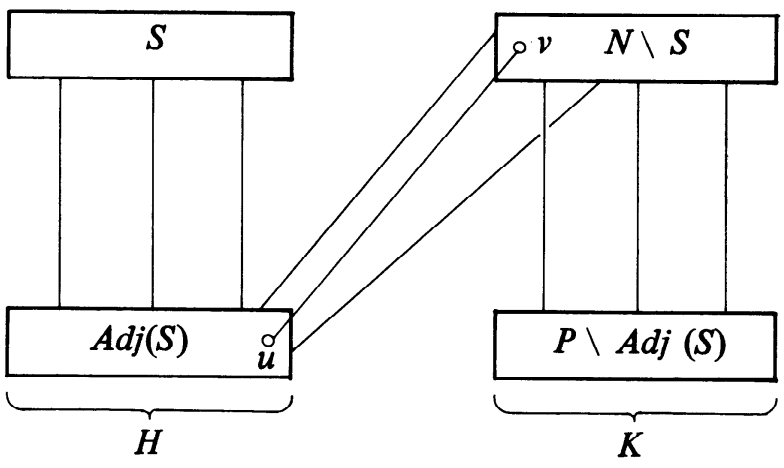


Figura 1 — O conjunto S em G e os subgrafos H e K de G.

Seja H o grafo $G[S \cup Adj(S)]$ e K o grafo $G\text{-}VH$ (vide Figura 1).

Note que $\{S, Adj(S)\}$ e $\{N\backslash S, P\backslash Adj(S)\}$ são bipartições de H e K, respectivamente. Note também que como $|AdjL(S)| < |S|$ e como S é um subconjunto de $N\backslash\{v\}$, então S não é nem o vazio nem N; portanto $|VH|$ e $|VK|$ são ambos menores do que $|V|$.

Basta então provar que $|AdjH(X)| \geq |X|$ para todo subconjunto X de S e $|AdjK(X)| \geq |X|$ para todo subconjunto X de $N\backslash S$. Porque então, por indução, H e K tem emparelhamentos tH e tK que cobrem S e $N\backslash S$, respectivamente; nesse caso, $tH \cup tK$ é um emparelhamento em G que cobre N.

Para provar que $|AdjH(X)| \geq |X|$ para todo subconjunto X de S, basta notar que se X é um subconjunto de S, então $AdjH(X) = Adj(X)$ por definição de H e $|Adj(X)| \geq |X|$ por hipótese.

Para provar que $|AdjK(X)| \geq |X|$ para todo subconjunto X de $N\backslash S$, note inicialmente que $AdjK(X) = Adj(X \cup S)\backslash Adj(S)$ por definição de K. Portanto,

$$|AdjK(X)| = |Adj(X \cup S)| - |Adj(S)|.$$

Por hipótese, $|Adj(X \cup S)| \geq |X \cup S| = |X| + |S|$; logo,

$$|AdjK(X)| \geq |X| + |S| - |Adj(S)|.$$

Finalmente, como $|S| = |Adj(S)|$, então $|AdjK(X)| \geq |X|$. A demonstração do Lema 2 completa a demonstração do Teorema de Hall. ■■

Com base na demonstração do Lema 2 (e na asserção do Lema 1), representamos na Figura 2 o algoritmo *Hall*(G, N, P): dados um grafo G e uma bipartição $\{P, N\}$ de G, *Hall*(G, N, P) determina se todo subconjunto X de N satisfaz a desigualdade

$$|Adj(X)| \geq |X|.$$

Em caso negativo, *Hall*(G, N, P) determina também um subconjunto X de N tal que $|Adj(X)| < |X|$; em caso afirmativo, *Hall*(G, N, P) determina também um emparelhamento em G que cobre N.

Na descrição do algoritmo, certos comandos têm nome, para facilidade de referência. Três desses comandos, a saber, recursão L, recursão H e recursão K, correspondem a chamadas recursivas do algoritmo. Assim, o comando recursão H deve ser interpretado da seguinte maneira: "chame *Hall* recursivamente para determinar se todo subconjunto X de S satisfaz à desigualdade $|AdjH(X)| \geq |X|$;

em caso afirmativo, atribua o valor *sim* a *satisfaz* e um emparelhamento que cobre S a *touX*; em caso negativo atribua *não* a *satisfaz* e um subconjunto de S que não satisfaz a desigualdade a *touX*".

```
procedimento Hall(G, N, P):
        início
            se aG é vazio então se N é vazio
                                  então devolva sim, ∅
                                  senão devolva não, N;
    {escolha}  seja α uma aresta em G;
               u, v ← extremo de α em P, extremo de α em N;
{remoção}      L ← G\{u, v};
{recursão L}   satisfaz, touX ← Hall(L, N\{v}, P\{u});
               se satisfaz então devolva sim, touX ∪ {α};
               S ← touX;
               se | Adj(S) | ≠ | S | então devolva não, S;
               H ← G[S ∪ Adj(S)];
{recursão H}   satisfaz, touX ← Hall(H, S, Adj(S));
               se não satisfaz então devolva não, touX
                                 senão tH ← touX;
               K ← G-VH;
{recursão K}   satisfaz, touX ← Hall(K, N\S, P\Adj(S));
               se não satisfaz então devolva não, touX
                                 senão devolva sim, tH ∪ touX
        fim
```

Figura 2 — O algoritmo *Hall* ("*G* satisfaz a condição de Hall?").

EXEMPLO 1. Considere o grafo G da Figura 3, com bipartição $\{P, N\}$, onde $N = \{v_0, v_2, v_4, v_6, v_8\}$. Durante a execução do algoritmo *Hall*, os comandos escolha e remoção deverão ser executados várias vezes: cada vez, alguma aresta α do grafo deverá ser escolhida, seus extremos removidos do grafo; suponhamos que se dê preferência à aresta α_{2i} com o maior índice $2i$ (se existir alguma). Se este critério de escolha for adotado e o algoritmo mecanicamente seguido, então o comando escolha será executado 31 vezes, até que o emparelhamento $\{\alpha_1, \alpha_3, \alpha_5, \alpha_7, \alpha_9\}$ seja obtido. Esta afirmação segue da Proposição 1, cuja demonstração fica a cargo do leitor, como exercício.

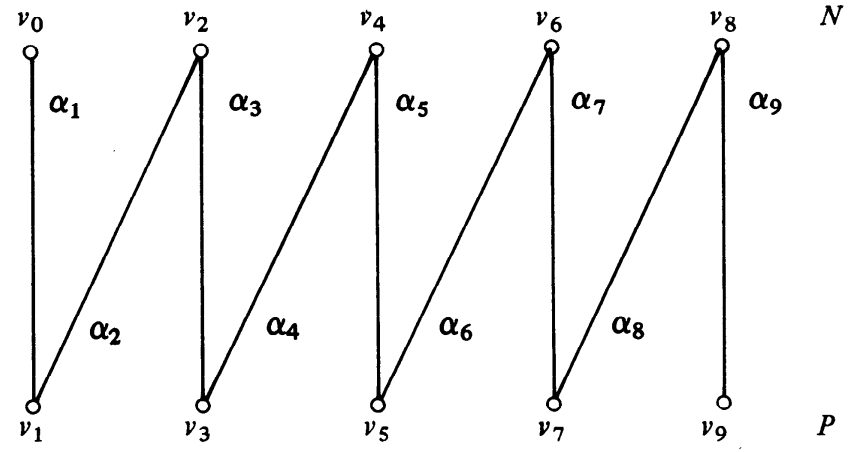


Figura 3 — O grafo do Exemplo 1.

PROPOSIÇÃO 1. *Para cada natural positivo n, seja G_n o grafo que tem $\{v_0, v_1, \ldots, v_{2n-1}\}$ como conjunto de vértices, $\{\alpha_1, \ldots, \alpha_{2n-1}\}$ como conjunto de arestas, onde, para cada i $(1 \leq i \leq 2n-1)$, a aresta α_i tem os vértices v_{i-1} e v_i como extremos; sejam N_n e P_n os conjuntos $\{v_0, v_2, \ldots, v_{2n-2}\}$ e $\{v_1, v_3, \ldots, v_{2n-1}\}$ respectivamente. Então $\{P_n, N_n\}$ é uma bipartição de G_n. Dada preferência à aresta α_{2i} com 2i máximo para a execução do comando escolha, então este será executado $2^n - 1$ vezes até que o emparelhamento $\{\alpha_1, \alpha_3, \ldots, \alpha_{2n-1}\}$ (que cobre N_n) seja obtido.*

O leitor poderia argumentar que não é possível existir um algoritmo polinomial equivalente a *Hall*, "uma vez que $|\, Adj(X) \,| \geq |\, X \,|$ precisa ser verificada para cada um dos $2^{|N|}$ subconjuntos X de N". Felizmente, tal raciocínio é falso: daremos a seguir outra demonstração do Lema 2, que induz um algoritmo eficiente.

Dado um conjunto t de arestas de um grafo G e um conjunto X de vértices de G, Xt denota o subconjunto de X que consiste daqueles vértices que incidem em pelo menos uma aresta em t.

Segunda demonstração do Lema 2 (suficiência da condição de Hall) — Suponha que G é um grafo com bipartição $\{P, N\}$ tal que $|\, Adj(X) \,| \geq |\, X \,|$ para todo subconjunto X de N. Dentre os emparelhamentos em G (e como o vazio é um deles então existe pelo menos um), seja t um tal que Nt seja maximal.

Pelo Lema 3, abaixo, N tem um subconjunto S tal que $|\, Adj(S) \,| + |\, N \backslash Nt \,| = |\, S \,|$. Ora, se t não cobre N então $N \backslash Nt$ não é vazio e portanto $|\, Adj(S) \,| < |\, S \,|$, contradição. Logo, t cobre N.

LEMA 3. *Seja G um grafo com bipartição* $\{P, N\}$, *t um emparelhamento em G. Então*

ou (i) *existe um subconjunto S de N que inclui* $N\backslash Nt$ *e tal que*

$$| Adj(S) | + | N\backslash Nt | = | S |,$$

ou (ii) *existe um emparelhamento* t_+ *em G tal que* Nt_+ *inclui (propriamente) Nt e* $| Nt_+\backslash Nt | = 1$.

Demonstração. Suponha, como hipótese de indução, que a asserção vale para todo grafo G' com $| VG' | < | VG |$.

Note inicialmente que se $Adj(N\backslash Nt)$ for vazio então (i) vale trivialmente, com $S = N\backslash Nt$.

Suponha pois que existe (pelo menos) uma aresta, α, com um extremo em $N\backslash Nt$. Seja u o extremo de α em P, v seu extremo em $N\backslash Nt$.

Se u pertence a $P\backslash Pt$, então (ii) vale, com $t_+ = t \cup \{\alpha\}$ e $Nt_+\backslash Nt = \{v\}$.

Suponha pois que u pertence a Pt. Seja β a aresta de t que incide em u, v' seu extremo em Nt. Note que v pertence a $N\backslash Nt$ e portanto v e v' são distintos (vide Figura 4).

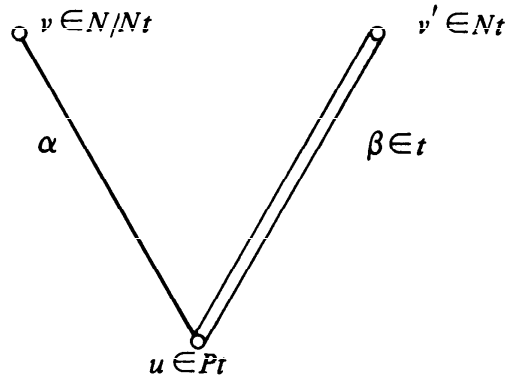


Figura 4

Seja G' o grafo $G - u$. Observe que $\{P\backslash\{u\}, N\}$ é uma bipartição de G', e $t' = t\backslash\{\beta\}$ um emparelhamento em G'. Note também que a hipótese de indução é aplicável a G', com bipartição $\{P\backslash\{u\}, N\}$ e emparelhamento t'.

Considere inicialmente o caso em que N tem um subconjunto S que inclui $N\backslash Nt'$ e tal que

$$| AdjG'(S) | + | N\backslash Nt' | = | S |.$$

Ora, $AdjG'(S) = Adj(S)\backslash\{u\}$, por definição de G'; como v', um vértice em $N\backslash Nt'$, pertence a S, então u pertence a $Adj(S)$. Logo,

$$| AdjG'(S) | = | Adj(S) | - 1.$$

Por outro lado, $N \backslash Nt' = (N \backslash Nt) \cup \{v'\}$ e v' pertence a Nt. Logo, S inclui $N \backslash Nt$ e

$$| N \backslash Nt' | = | N \backslash Nt | + 1.$$

De fato, S inclui $N \backslash Nt$ e $| Adj(S) | + | N \backslash Nt | = | S |$.

Resta agora considerar o caso em que G' tem um emparelhamento, t'_+, tal que Nt'_+ inclui Nt' e $Nt'_+ \backslash Nt'$ é unitário. Para tanto, seja w o vértice em $Nt'_+ \backslash Nt'$. Se w e v' forem distintos, então (ii) vale, com $t_+ = t'_+ \cup \{\beta\}$ e $Nt_+ \backslash Nt = \{w\}$. Se w e v' coincidirem, então $Nt'_+ = Nt$ e nesse caso (ii) vale, com $t_+ = t'_+ \cup \{\alpha\}$ e $Nt_+ \backslash Nt = \{v\}$. Em ambos os casos, (ii) vale. A demonstração do Lema 3 completa a demonstração da suficiência da condição de Hall. ■■

Com base na segunda demonstração da suficiência da condição de Hall (bem como na asserção do Lema 1), apresentamos nas Figuras 5 e 6 os algoritmos *Hall2* e *Hallauxiliar*. Dados um grafo G, uma bipartição $\{P, N\}$ de G e um emparelhamento t em G, *Hallauxiliar* determina, se existir, um emparelhamento t_+ em G tal que Nt_+ inclui Nt e $Nt_+ \backslash Nt$ é unitário; caso um tal t_+ não exista, então *Hallauxiliar* determina um subconjunto S de N que inclui $N \backslash Nt$ e tal que $| Adj(S) | + | N \backslash Nt | = | S |$.

O algoritmo *Hall2* determina, dado G e uma bipartição $\{P, N\}$, se $| Adj(X) | \geq | X |$ para todo subconjunto X de N; em caso afirmativo, *Hall2* determina também um emparelhamento que cobre N; em caso negativo, *Hall2* determina um subconjunto S de N tal que $| Adj(S) | < | S |$.

```
procedimento Hall2 (G, N, P):
    início
        touS ← ∅;
        repita
            t ← touS;
            obtidot+, touS ← Hallauxiliar (G, N, P, t)
        até que não obtidot+;
        se N\Nt = ∅ então devolva sim, t
                    senão devolva não, touS {S}
    fim
```

Figura 5 — O algoritmo *Hall2* ("G satisfaz $|Adj(X)| \geq |X|$ para cada subconjunto X de N?").

```
procedimento Hallauxiliar (G, N, P, t):
    início
        se Adj(N\Nt) = ∅ então devolva não, N\Nt;
{escolha} seja α uma aresta com um extremo em N\Nt;
        u, v ← extremo de α em P, extremo de α em N;
        se u ∉ Pt então devolva sim, t ∪ {α};
        β ← aresta de t que incide em u;
        v', G', t' ← extremo de β em N, G − u, t\{β};
        obtido t+, touS ← Haullauxiliar(G', N, P\{u}, t');
        se não obtidot+ então devolva não, touS    {S}
                       senão
        se v' ∈ Nt'+ então devolva sim, touS ∪ {α}    {t+}
                       senão devolva sim, touS ∪ {β}    {t+}
    fim
```

Figura 6 — O algoritmo *Hallauxiliar*.

Deixamos ao leitor verificar a seguinte afirmação.

PROPOSIÇÃO 2. *O comando escolha de Hallauxiliar é executado não mais do que* $|aG|$ *vezes, e portanto Hall2 causa a execução do referido comando não mais do que* $|N|\,|aG|$ *vezes.*

2. Uma igualdade minimax

Um conjunto X de vértices de um grafo G *cobre* um conjunto x de arestas do grafo (é uma *cobertura* de x) se cada aresta em x incide em pelo menos um vértice em X.

TEOREMA 2 (König). *Seja G um grafo com bipartição* $\{P, N\}$. *Então um emparelhamento máximo em G tem cardinalidade igual à de uma cobertura mínima de aG.*

Demonstração. Pela segunda demonstração da suficiência da condição de Hall, existe um emparelhamento t em G e um subconjunto S de N tais que

$$|Adj(S)| + |N\backslash Nt| = |S|.$$

Seja Z o conjunto $Adj(S) \cup (N\backslash S)$. Então $|Z| = |Nt| = |t|$.

Resta mostrar que Z é uma cobertura mínima de aG e t um emparelhamento máximo em G.

Para mostrar que Z é uma cobertura de aG, basta observar que para cada aresta α em aG, ou seu extremo em P pertence a $Adj(S)$ ou seu extremo em N pertence a $N\backslash S$ e em ambos os casos α tem pelo menos um extremo em Z.

Para mostrar que Z é mínima e t é máximo, seja Z' uma cobertura de aG e t' um emparelhamento em G. Como Z' cobre aG então cada aresta α em t' tem pelo menos um extremo, digamos $v\alpha$, em Z'. Como t' é um emparelhamento então arestas distintas α e β em t' têm extremos distintos $v\alpha$ e $v\beta$ em Z'. Portanto $|t'| \leq |Z'|$. Em particular, $|t'| \leq$ $\leq |Z| = |t| \leq |Z'|$ e portanto $|t'| \leq |t|$ e $|Z| \leq |Z'|$. De fato, t é máximo e Z é mínima. A demonstração do teorema de König está completa. ■

EXERCÍCIOS

1. Seja G um grafo com bipartição $\{P, N\}$ e k um inteiro. Mostre que se $g(v) \geq k \geq 1$ para cada vértice v em N e $g(v) \leq k$ para cada vértice v em P então G tem um emparelhamento que cobre N. Mostre ainda que se G é k-regular, $k \geq 1$, então existem k emparelhamentos que cobrem V, dois a dois disjuntos.
2. Sejam n prismas triangulares regulares ($n \geq 1$) tais que cada face (lateral) de cada prisma é colorida com uma de n cores de forma que existam exatamente três das $3n$ faces com cada cor. Mostre que é possível empilhar os prismas pelas bases de forma a obter outro prisma triangular no qual cada face exiba todas as n cores.
3. Mostre que se um grafo G qualquer tem um emparelhamento que cobre um conjunto especificado M de vértices então $|Adj(X)| \geq$ $\geq |X|$, para todo subconjunto X de M. Dê um exemplo para mostrar que a recíproca é falsa. Prove que a recíproca é verdadeira para grafos biparticionáveis. Mostre ainda que se Δ é o maior grau dos vértices de um grafo biparticionável G, então aG é a união de uma coleção (disjunta) que consiste de Δ emparelhamentos.
4. Um grafo G é *par* se $|VG|$ for par, e *ímpar* se $|VG|$ for ímpar. Denota-se por iG o conjunto dos componentes ímpares de um grafo G. Mostre que para todo emparelhamento t de um grafo G, $|Adj(X)\backslash X| \geq |iG[X]|$ para todo subconjunto X de Vt.
5. Sejam t e t_+ emparelhamentos em um grafo G. Descreva os componentes do grafo $G[t \oplus t_+]$, onde $t \oplus t_+ = t \cup t_+ \backslash t \cap t_+$. Mostre que se $|t_+| > |t|$ então existe em G um emparelhamento x tal

que Vx inclui Vt e $Vx \backslash Vt$ consiste de precisamente dois vértices, ambos em $Vt_+ \backslash Vt$.

6. Seja G um grafo, $I(G)$ o conjunto dos subconjuntos Z de V tais que G tem emparelhamento que cobre Z. Mostre que $(V, I(G))$ é um matróide (vide Exercício II.10).

7. Um conjunto X de vértices de um grafo G é *independente* se nenhuma aresta de G tem ambos os extremos em X. Mostre que um conjunto X é independente se e somente se $V \backslash X$ cobre aG. Conclua que S é um conjunto independente máximo se e somente se $V \backslash X$ é uma cobertura mínima de aG. Dê um algoritmo eficiente para obter um conjunto independente máximo num grafo bipartícionável.

8. Dê exemplos de grafos em que a cardinalidade de uma cobertura mínima de aG e a de um emparelhamento máximo são distintas.

9. Mostre que se t é um emparelhamento num grafo G sem vértices de grau zero, então uma cobertura $c(t)$ de VG pode ser obtida a partir de t adicionando a t arestas incidentes em vértices de $V \backslash Vt$. Mostre que existe uma cobertura $c(t)$ de V que inclui t e tal que $| c(t) | + | t | \leq | V |$.

10. Mostre que se c é uma cobertura de VG então c inclui um emparelhamento $t(c)$ tal que $| c | + | t(c) | \geq | V |$.

11. Mostre que se t é um emparelhamento máximo em G e c uma cobertura mínima de VG, então $| c | + | t | = | V |$.

12. Mostre que se t é um emparelhamento em G e c uma cobertura de VG tais que $| c | + | t | = | V |$, então c é mínima se e somente se t é máximo.

13. Dê um algoritmo polinomial que obtém uma cobertura mínima de VG, G bipartícionável sem vértices de grau zero.

14. Mostre que em todo grafo G, se I é um conjunto independente máximo de vértices (vide Exercício 7) e c uma cobertura mínima de VG, então $| I | \leq | c |$. Dê exemplos em que a desigualdade estrita é observada. Mostre que para grafos bipartícionáveis sem vértices de grau zero vale sempre a igualdade.

15. Mostre que se G, com bipartição $\{P, N\}$, tem um emparelhamento que cobre N e outro que cobre P então G tem um emparelhamento que cobre V.

16. Deduza o Teorema de Hall a partir do Teorema de König.

17. (*Teorema de Tutte*) – Um grafo G tem um emparelhamento que cobre V se e somente se $|\, Adj(X)\backslash X \,| \geq |\, iG[X] \,|$, para todo subconjunto X de V (a necessidade da condição é o enunciado do Exercício 4). Demonstre a suficiência da condição, por indução em $|\, V \,|$, da seguinte maneira:

    (a) Defina um subconjunto S de V como *crítico* se $|\, Adj(S)\backslash S \,| = |\, iG[S] \,|$ e se S é não vazio.
    (b) Mostre que V é crítico quando G é não vazio.
    (c) Mostre que se S é crítico minimal, então nenhum componente de $G[S]$ é par (remova um vértice de um componente par de $G[S]$).
    (d) Para cada conjunto crítico S, defina um grafo $H(S)$ com bipartição $\{iG[S], Adj(S)\backslash S\}$, cujas arestas são as arestas do corte de S e tal que se α é uma aresta de $\delta(S)$ com um extremo, u, num componente ímpar, C, de $G[S]$, e o outro extremo, v, em $Adj(S)\backslash S$, então C e u são os extremos de α em H. Mostre que $H(S)$ tem um emparelhamento $t(S)$ que cobre $VH(S)$.
    (e) Mostre que para cada conjunto crítico minimal S, e cada componente ímpar K de $H[S]$, se v é o vértice de VK incidente à aresta de $t(S)$, então $K' = K - v$ tem um emparelhamento que cobre VK' pois se $|\, AdjK'(T)\backslash T \,| < |\, iK'[T] \,|$, para algum $T \subseteq VK'$, então $S\backslash\{v\}\backslash(AdjK'(T)\backslash T)$ é crítico.
    (f) Mostre que se S é crítico então $G' = G - [S \cup Adj(S)]$ tem um emparelhamento que cobre VG'.

18. Mostre que todo grafo 3-regular sem arestas de corte tem um emparelhamento que cobre VG. Sugestão: use o Teorema de Tutte (Exercício 17).

19. Mostre que para toda aresta α de um grafo G 3-regular sem arestas de corte existe um emparelhamento em G que cobre VG e contém α.

20. Dê um exemplo de um grafo simples 3-regular que não têm um emparelhamento que cobre VG.

21. O grafo 3-regular da Figura 7, o grafo de Petersen, tem, para cada uma de suas arestas α, um emparelhamento, $t\alpha$, que contém α e cobre VG; mas G não possui três emparelhamentos dois a dois disjuntos cuja união é aG. Demonstre estas afirmações.

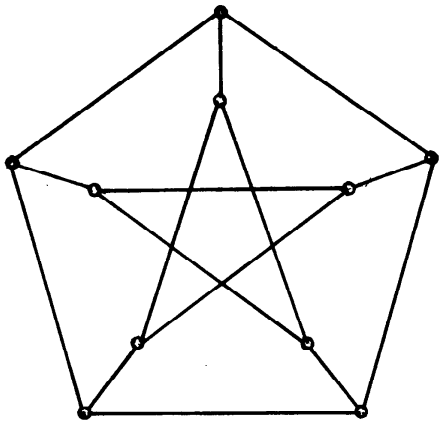


Figura 7 — O grafo de Petersen.

NOTAS BIBLIOGRÁFICAS

O Teorema de Hall foi provado pela primeira vez por Philip Hall [46] em 1935, e sua demonstração era relativamente complicada. Várias outras demonstrações apareceram deste então (veja Mirsky [84]). A demonstração do Teorema de Hall aqui apresentada é de Halmos e Vaughan [48]. A demonstração do Lema 3 é nova, de Lucchesi e Younger. Um algoritmo eficiente equivalente ao algoritmo *Hall2* foi dado por M. Hall Jr. [44].

König provou o teorema que leva seu nome em 1931 [63]. Aqui no entanto damos uma demonstração que usa o Teorema de Hall como lema, apesar de historicamente este ter sido provado posteriormente.

Existem inúmeros teoremas na literatura cuja primeira demonstração induz um algoritmo exponencial; somente mais tarde é obtida outra que induz um algoritmo polinomial. Assim temos por exemplo o Teorema de Tutte [120] (vide Exercício 17), do qual Edmonds [23] deu uma demonstração que induz um algoritmo eficiente. Outro exemplo pode ser encontrado em Lucchesi [69], Lovász [68] e em Lucchesi e Younger [70].

Para um grafo bipartitionável G, existem algoritmos polinomiais que determinam emparalhamentos máximos, conjuntos independentes máximos e coberturas mínimas de aG e VG (vide Exercícios 7, 13 e 14). Para grafos G arbitrários os problemas de determinar se G tem uma cobertura de aG com no máximo k arestas ou um conjunto independente com pelo menos k vértices (k dado) são $\mathcal{NP}$-m-completos [57] (vide Capítulo B.IV.). Por outro lado existem algoritmos polinomiais para determinar emparelhamentos máximos e coberturas mínimas de VG, G genérico [23].

CAPÍTULO IV

COLORAÇÃO DE VÉRTICES E O TEOREMA DE BROOKS

1. Coloração de vértices

Uma *coloração de vértices* de um grafo G é uma partição p de V em conjuntos independentes. (Podemos dizer que uma coloração de vértices de G consiste em dar a cada vértice uma cor de forma que vértices adjacentes tenham cores diferentes). Para um inteiro s, uma *s-coloração* de G é uma coloração p de G tal que $|p| \leq s$. Se G admite uma s-coloração então G é *s-colorável*. (Observe que se G tem laços então G não admite colorações, ao passo que se G não tem laços então G é $|V|$-colorável). O *número cromático* $crom(G)$ de G é o menor inteiro s tal que G é s-colorável. Os grafos das Figuras 1 e 2 têm números cromáticos 4 e 3, respectivamente.

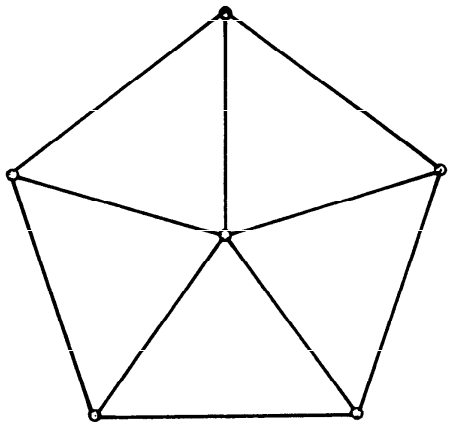


Figura 1 — A roda de cinco pontas tem número cromático 4.

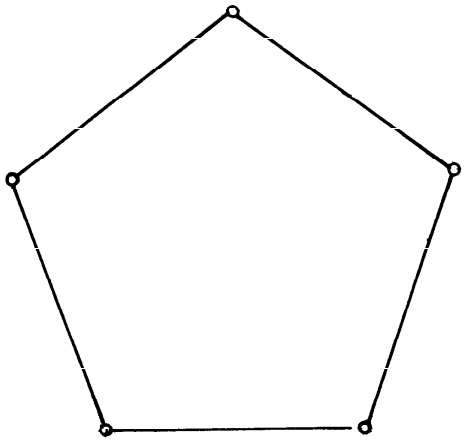


Figura 2 — O pentágono tem número cromático 3.

TEOREMA 1 (Brooks). *Seja s um inteiro e G um grafo sem laços, e sem subgrafos completos com $s+1$ vértices. Se $s \geq 3$ e $g(v) \leq s$ para todo v em V então G é s-colorável.*

Demonstração. Suponha que, pelo contrário, existam grafos que satisfaçam todas as condições do teorema mas que não sejam s coloráveis. Seja G um tal grafo com um número mínimo de vértices. Observe que pela definição de G, removendo-se um vértice

x qualquer de G obtém-se o grafo $G' = G - x$ que é s-colorável. Para cada s-coloração de G' é necessário que se usem todas as s cores para os t vértices $x_1, x_2, \ldots, x_t$ adjacentes a x em G, pois caso contrário haveria uma cor disponível a x e portanto G seria s-colorável.

Como $g(x) \leq s$, então $t = s$ e podemos supor que os vértices $x_1, \ldots, x_s$ são coloridos com as cores $1, \ldots, s$, respectivamente.

Supondo o grafo G' assim colorido, temos então:

LEMA 1. *Vértices x_i e x_j $(1 \leq i, j \leq s \text{ e } i \neq j)$ estão no mesmo componente C_{ij} do subgrafo B_{ij} induzido pelo conjunto de vértices de cores i e j.*

Demonstração. Caso contrário a permuta das cores i e j no componente de B_{ij} que tem x_i como um de seus vértices fornece uma s-coloração de G' em que os vértices x_i e x_j têm ambos a mesma cor j, em contradição à obrigatoriedade de colorir $x_1, \ldots, x_s$ com s cores diferentes. ■

LEMA 2. *C_{ij} é uma cadeia, $(1 \leq i, j \leq s \text{ e } i \neq j)$. Isto é, C_{ij} é gerado pelas arestas de um caminho em C_{ij}.*

Demonstração. Pelo Lema 1, existe um caminho (não degenerado) $(v_0, \alpha_1, v_1, \ldots, \alpha_n, v_n)$ de x_i a x_j em C_{ij} (onde $v_0 = x_i$ e $v_n = x_j$).

Observe que x_i é adjacente ao vértice v_1, que por sua vez tem cor j. Ademais, como $g(x_i) \leq s$ então $gG'(x_i) \leq s - 1$ e portanto v_1 é o único vértice de cor j adjacente a x_i, pois caso contrário x_i poderia ser recolorido com uma cor diferente de i. Portanto, o grau de x_i em C_{ij} é 1. Analogamente, o grau de x_j em C_{ij} é 1.

Para completar a demonstração do Lema 2, basta agora mostrar que para todo k tal que $0 < k < n$, o grau de v_k em C_{ij} é 2. Para tanto, suponha o contrário, seja k o menor inteiro tal que $0 < k < n$ e o grau de v_k em C_{ij} é distinto de 2. Como v_k é adjacente a v_{k-1} e a v_{k+1}, por sua vez distintos, então o grau de v_k em C_{ij} é maior do que 2. Nesse caso, v_k pode ser recolorido com uma cor diferente de i e j e o novo componente C_{ij} terá $x_i = v_0, v_1, \ldots, v_{k-1}$ como vértices, não terá nenhum dos vértices $v_k, v_{k+1}, \ldots, v_n = x_j$, em contradição ao Lema 1. De fato, C_{ij} é a cadeia $G[aC]$. ■

LEMA 3. *O vértice* x_i *é o único comum a* C_{ij} *e* C_{ik} $(1 \leq i, j, k \leq s, i \neq j \neq k \neq i)$.

Demonstração. Suponha que w é um vértice em $V\backslash\{x_i\}$ comum a C_{ij} e C_{ik}. Então o grau de w em C_{ij} e em C_{ik} é 2 e portanto w pode ser recolorido com uma cor diferente de i, de j e de k; neste caso, os vértices x_i e x_j ficarão em componentes distintos, do novo B_{ij}, em contradição ao Lema 1. ■

Como G não tem um subgrafo completo com $s+1$ vértices, podemos supor então que x_1 e x_2 não são adjacentes. Nesse caso, a cadeia C_{12} contém um vértice y adjacente a x_1, mas distinto de x_2. Após intercambiar as cores 1 e 3 na cadeia C_{13}, as novas cadeias C_{21} e C_{23} conterão ambas o vértice y distinto de x_2, em contradição ao Lema 3. A demonstração do Teorema de Brooks está completa. ■

EXERCÍCIOS

1. Acompanhe a demonstração do Teorema de Brooks para o grafo G da Figura 3.

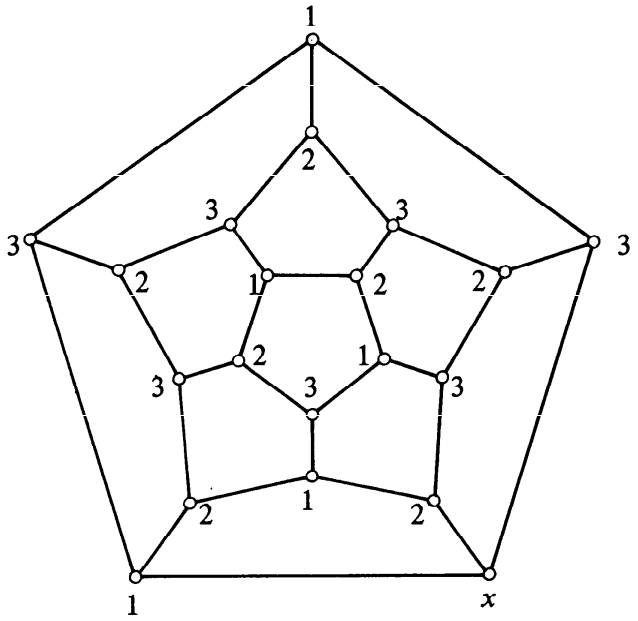


Figura 3

2. Mostre que a demonstração do Teorema de Brooks induz um algoritmo polinomial (linear).
3. Por que existe a restrição $s \geq 3$ no enunciado do Teorema de Brooks?
4. (*Teorema das cinco cores*) — Pode-se provar que todo grafo planar simples não vazio tem pelo menos um vértice com grau menor do

que ou igual a cinco. Baseado nesta propriedade, demonstre que todo grafo planar simples é 5-colorável.

5. Mostre que todo natural k é o número cromático de algum grafo sem triângulos.
6. Dê um algoritmo linear para determinar se um grafo é ou não 2-colorável.

NOTAS BIBLIOGRÁFICAS

Em 1852, Francis Guthrie concebeu a famosa "conjetura das quatro cores": todo grafo planar sem laços é 4-colorável [90] (veja também [103]). Observe que o grafo da Figura 1 é planar e tem número cromático 4; portanto a "conjetura das três cores" é falsa. Por outro lado, a "conjetura das cinco cores" é verdadeira, sua demonstração é bastante simples. (Veja, por exemplo, [129], ou [6]; veja Exercício 4).

A conjetura das quatro cores foi provada por Appel e Haken em 1976 [2], com o auxílio do computador, após resistir durante mais de cem anos ao ataque de combinatóricos, algebristas e topólogos, que forneceram inúmeras demonstrações falsas. Dentre essas, a mais famosa parece ter sido a de Kempe [58], durante dez anos aceita como correta. Apesar disso, a técnica por ele utilizada, de "cadeias de Kempe", foi aqui aplicada com sucesso na demonstração do Teorema de Brooks. Esta não é a demonstração original [9], mas sim a de Melnikov e Vizing [79].

Determinar se um grafo é ou não 2-colorável é bastante simples (vide Exercício 6). Por outro lado, determinar se um grafo é k-colorável (k fixo, $k \geq 3$) é um problema $\mathcal{NP}$-m-completo [57]. Mesmo determinar se um grafo planar é ou não 3-colorável é $\mathcal{NP}$-m-completo [114]. Por outro lado, determinar se um grafo planar é k-colorável ($k \geq 4$) é trivial, à luz do Teorema das quatro cores; basta verificar se o grafo tem ou não laços.

CAPÍTULO V

O TEOREMA DE RAMSEY E SUAS APLICAÇÕES

1. Introdução

Numa festa com seis participantes existem três que conhecem-se dois a dois ou existem três que desconhecem-se dois a dois.

A afirmação acima é um corolário de um teorema de F.P. Ramsey que será visto na Seção 2. Nas Seções 3 e 4 damos algumas aplicações do Teorema de Ramsey à Teoria dos Grafos, bem como a outras áreas da Matemática. Antes, porém, é instrutivo provar a afirmação inicial, pois a sua demonstração, apesar de muito simples, ilustra bem as idéias básicas na demonstração do Teorema de Ramsey.

Inicialmente, reformulamos a afirmação dada, usando agora a linguagem da Teoria dos Grafos. A uma dada festa, com participantes $P = \{p_1, p_2, p_3, p_4, p_5, p_6\}$, associamos um grafo completo G com conjunto de vértices P. Para $p_i \neq p_j$, colorimos a aresta de G, com extremos p_i e p_j, de vermelho ou de azul, conforme os participantes p_i e p_j se conheçam ou não. Resulta que três participantes p_i, p_j e p_k conhecem-se dois a dois (desconhecem-se dois a dois) se e somente se as arestas do triângulo $G[p_i, p_j, p_k]$ são todas vermelhas (todas azuis). Assim a afirmação dada é equivalente à proposição que segue.

PROPOSIÇÃO 1. *Para qualquer coloração, em vermelho e azul, das arestas de um grafo completo com seis vértices, este contém um triângulo vermelho ou um triângulo azul.*

Demonstração. Seja G o grafo completo cujas arestas foram coloridas, e seja v um vértice de G. Como v é incidente a cinco arestas, pelo menos três destas são vermelhas ou pelo menos três são azuis. Suponhamos o primeiro caso, e sejam α_1, α_2 e α_3 três arestas vermelhas incidentes a v. Sejam u_1, u_2 e u_3 os outros extremos destas arestas.

Se alguma aresta de $G[u_1, u_2, u_3]$ for vermelha, digamos aquela com extremos u_i e u_j, então o triângulo $G[v, u_i, u_j]$ é vermelho. Caso contrário, o triângulo $G[u_1, u_2, u_3]$ é azul. (Veja a Figura 1.)

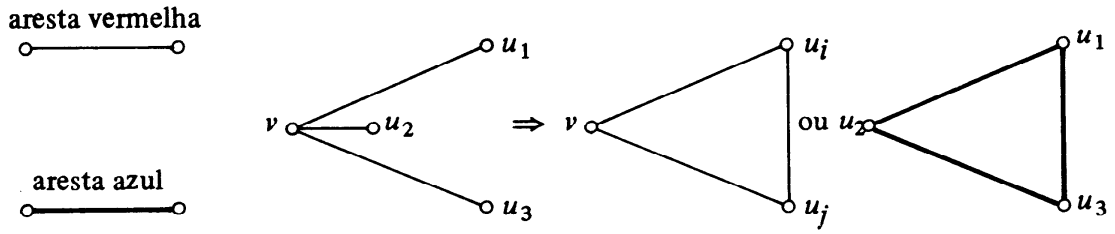


Figura 1 — Ilustração para a Proposição 1.

O caso em que v é incidente a pelo menos três arestas azuis é tratado analogamente. ■

Observamos que as arestas de um grafo completo G com cinco vértices podem ser coloridas de forma que G não tenha nem triângulos vermelhos, nem triângulos azuis, conforme mostra a Figura 2.

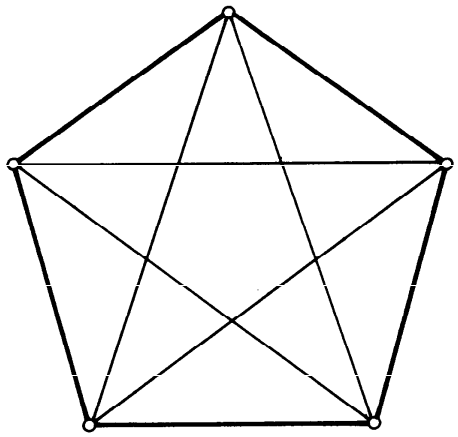


Figura 2 — Uma coloração de arestas sem triângulos monocromáticos.

2. O Teorema de Ramsey

Para um natural k, um *k-conjunto* é um conjunto (finito) de cardinalidade k. Um *k-subconjunto* de um conjunto A é um subconjunto de A, cuja cardinalidade é k. Denotamos por $\mathfrak{p}_k A$ o conjunto dos k-subconjuntos de A, isto é

$$\mathfrak{p}_k A = \{B \subseteq A \mid |B| = k\}.$$

Por uma *partição* de um conjunto A em n *blocos* $B_1, B_2, \ldots, B_n$, entendemos que os blocos são subconjuntos de A, dois a dois disjuntos, e cuja união é A. Note que permitimos que alguns dos blocos sejam vazios.

TEOREMA 1 (Ramsey). *Dados naturais n, m e k tais que $m \geq k \geq 1$ e $n \geq 2$, existe um natural $\ell = R(n, m, k)$, tal que dados um ℓ-conjunto X e uma partição de $\mathfrak{p}_k X$ em n blocos $B_1, B_2, \ldots, B_n$, existem um bloco B_i e um m-subconjunto Y de X, tais que $\mathfrak{p}_k Y \subseteq B_i$.*

Já que o enunciado do teorema acima é um pouco complicado, damos uma descrição pictórica do processo envolvido. Dado um ℓ-conjunto X, formamos uma lista dos seus k-subconjuntos e colorimos cada um destes com uma de n cores disponíveis. O Teorema 1 afirma que, se ℓ for suficientemente grande, qualquer que seja a coloração, sempre encontraremos um m-subconjunto Y de X, tal que todos os k-subconjuntos de Y têm uma mesma cor.

Para demonstrarmos o Teorema 1, necessitamos de um estudo mais detalhado do caso no qual particionamos os k-subconjuntos de X em apenas dois blocos, isto é, $n = 2$. Este caso pode ser assim enunciado:

TEOREMA 2. *Dados naturais m_1, m_2 e k, tais que $m_1, m_2 \geq k \geq 1$, existe um natural $\ell = S(m_1, m_2, k)$, tal que dados um ℓ-conjunto X e uma partição de $\mathfrak{p}_k X$ em dois blocos B_1 e B_2,*

existe um m_1-subconjunto Y_1 de X, tal que $\mathfrak{p}_k Y_1 \subseteq B_1$,

ou

existe um m_2-subconjunto Y_2 de X, tal que $\mathfrak{p}_k Y_2 \subseteq B_2$.

Antes de demonstrar estes teoremas, apresentamos definições indutivas das funções R e S, cuja existência é afirmada nos Teoremas 1 e 2, respectivamente. Quanto a S, temos:

$$S(m_1, m_2, 1) = m_1 + m_2 - 1 \qquad \text{para } m_1, m_2 \geq 1 \qquad (1)$$

$$S(k, m_2, k) = m_2 \qquad \text{para } m_2 \geq k > 1 \qquad (2)$$

$$S(m_1, k, k) = m_1 \qquad \text{para } m_1 > k > 1 \qquad (3)$$

$$\text{e } S(m_1, m_2, k) = 1 + S(S(m_1 - 1, m_2, k), S(m_1, m_2 - 1, k), k - 1) \qquad \text{para } m_1, m_2 > k > 1. \qquad (4)$$

Deixamos ao leitor a demonstração de que o valor $S(m_1, m_2, k)$ está bem definido para todo m_1, m_2, k, tais que $m_1, m_2 \geq k \geq 1$ e que

$$S(m_1, m_2, k) = S(m_2, m_1, k) \geq k.$$

Isto pode ser feito por indução em k, e para cada valor de $k > 1$, por indução em $m_1 + m_2$. Note que na prova do Teorema 2 usaremos um esquema de indução análogo a este.

Os valores de $S(m_1, m_2, k)$ são "enormes", mesmo para valores "pequenos" de m_1, m_2 e k. De fato, pode-se demonstrar que a função S cresce mais rapidamente do que qualquer função primitiva recursiva [94]. Aqui nos contentaremos em mostrar que

$$S(5, 5, 3) \simeq 10^{6395},$$

ilustrando ao mesmo tempo a definição de S.

Inicialmente, observamos que, para $m_1, m_2 \geq 2$,

$$S(m_1, m_2, 2) = \binom{m_1 + m_2 - 2}{m_1 - 1} \tag{5}$$

onde o lado direito representa um coeficiente binomial. De fato, de (4) e de (1),

$$\begin{aligned} S(m_1, m_2, 2) &= 1 + S(S(m_1 - 1, m_2, 2), S(m_1, m_2 - 1, 2), 1) = \\ &= S(m_1 - 1, m_2, 2) + S(m_1, m_2 - 1, 2). \end{aligned}$$

Agora (5) segue por indução sobre $m_1 + m_2$, aplicando-se a fórmula

$$\binom{p}{q} = \binom{p-1}{q} + \binom{p-1}{q-1}$$

Temos então:

$$S(5, 5, 3) = 1 + S(S(4, 5, 3), S(5, 4, 3), 2).$$

Torna-se necessário calcular $S(4, 5, 3) = S(5, 4, 3)$:

$$S(4, 5, 3) = 1 + S(S(3, 5, 3), S(4, 4, 3), 2) = 1 + S(5, S(4, 4, 3), 2).$$

Calculando $S(4, 4, 3)$, obtém-se

$$\begin{aligned} S(4, 4, 3) &= 1 + S(S(3, 4, 3), S(4, 3, 3), 2) = \\ &= 1 + S(4, 4, 2) = 1 + \binom{6}{3} = 21. \end{aligned}$$

Assim,

$$S(4, 5, 3) = 1 + \binom{24}{4} = 10627, \text{ e}$$

$$S(5, 5, 3) = 1 + \binom{21252}{10626}$$

Usando-se a aproximação de Stirling:

$$n! \simeq \sqrt{2\pi n}\left[\frac{n}{e}\right]^n,$$

obtém-se

$$S(5, 5, 3) \simeq 10^{6395}.$$

Isto é, para escrevermos o valor exato de $S(5, 5, 3)$ necessitamos de mais de 6 000 algarismos, ou seja pelo menos três páginas deste texto! Recomendamos ao leitor que reflita sobre a estimativa do valor de $S(6, 6, 4)$.

Quanto à função R, temos a definição indutiva:

$$R(2, m, k) = S(m, m, k) \qquad \text{para } m \geq k \geq 1 \tag{6}$$

e

$$R(n, m, k) = S(R(n-1, m, k), m, k) \text{ para } n > 2 \text{ e } m \geq k \geq 1. \tag{7}$$

O leitor não terá dificuldades em verificar que o valor de $R(n, m, k)$ está bem definido para $n \geq 2$ e $m \geq k$. Notamos apenas que

$$R(n, m, 1) = n(m-1) + 1 \text{ para } n \geq 2 \text{ e } m \geq 1. \tag{8}$$

Passemos às demonstrações dos Teoremas 1 e 2.

Demonstração do Teorema 2. No que segue, consideramos triplas ordenadas de naturais (m_1, m_2, k) com $m_1, m_2 \geq k \geq 1$. Denotamos $S(m_1, m_2, k)$ por ℓ e supomos dados um ℓ-conjunto X e uma partição de $\mathfrak{p}_k X$ em dois blocos B_1 e B_2. A afirmação do teorema estará provada para a tripla (m_1, m_2, k), ao exibirmos um m_1-subconjunto Y_1 de X, tal que $\mathfrak{p}_k Y_1 \subseteq B_1$ ou um m_2-subconjunto Y_2 de X, tal que $\mathfrak{p}_k Y_2 \subseteq B_2$.

Demonstramos o teorema por indução sobre k e chamamos esta indução de externa. Para $k = 1$, $\ell = m_1 + m_2 - 1$ (de (1)), e para todo conjunto Z, podemos identificar $\mathfrak{p}_1 Z$ com Z. Resulta então:

$$|B_1| + |B_2| = |X| = \ell = m_1 + m_2 - 1. \tag{9}$$

Se $|B_1| \geq m_1$, então para qualquer m_1-subconjunto Y_1 de B_1

$$\mathfrak{p}_1 Y_1 = Y_1 \subseteq B_1.$$

Se por outro lado $|B_1| < m_1$, resulta de (9) que $|B_2| \geq m_2$, logo para qualquer m_2-subconjunto Y_2 de B_2,

$$\mathfrak{p}_1 Y_2 = Y_2 \subseteq B_2.$$

Isto prova a base da indução externa.

Quanto ao passo da indução externa, fixamos um valor de $k > 1$ e supomos a afirmação verdadeira para triplas $(m_1', m_2', k-1)$. Demonstramos a afirmação para triplas (m_1, m_2, k) por indução sobre $m_1 + m_2$, indução esta que chamamos de interna.

A base da indução interna é dada pelo caso em que $m_1 + m_2 = 2k$. Como $m_1, m_2 \geq k$, segue que $m_1 = m_2 = k$. De (2), temos que $\ell = k$, logo $|\mathfrak{p}_k X| = 1$. Assim B_1 ou B_2 é igual a $\mathfrak{p}_k X$, bastando tomar $Y_1 = X$ ou $Y_2 = X$, respectivamente.

Para ver o passo da indução interna, fixamos uma tripla (m_1, m_2, k), com $m_1 + m_2 > 2k$ e supomos a afirmação ser verdadeira para triplas (m_1'', m_2'', k), com

$$m_1'' + m_2'' < m_1 + m_2.$$

Se $m_1 = k$, então de (2), $\ell = m_2$ e se $B_1 \neq \varnothing$, tomamos qualquer elemento de B_1 para Y_1. Se por outro lado $B_1 = \varnothing$, então $\mathfrak{p}_k X = B_2$, logo $Y_2 = X$ satisfaz a afirmação. Raciocínio análogo resolve o caso em que $m_2 = k$. Finalmente, se $m_1, m_2 > k$, colocamos

$$m_1' = S(m_1 - 1, m_2, k) \text{ e } m_2' = S(m_1, m_2 - 1, k),$$

e assim, de (4),

$$\ell = 1 + S(m_1', m_2', k-1).$$

Seja x um elemento de X e seja $X' = X \backslash \{x\}$. Então

$$|X'| = S(m_1', m_2', k-1). \tag{10}$$

Definimos ainda, para $i = 1, 2$

$$B_i' = \{A \in \mathfrak{p}_{k-1} X' \mid A \cup \{x\} \in B_i\}.$$

B_1' e B_2' formam uma partição de $\mathfrak{p}_{k-1} X'$, logo de (10) e da hipótese da indução externa, X' contém

um m_1'-subconjunto Y_1', com $\mathfrak{p}_{k-1} Y_1' \subseteq B_1'$,

ou

um m_2'-subconjunto Y_2', com $\mathfrak{p}_{k-1} Y_2' \subseteq B_2'$.

É suficiente considerar o primeiro caso, pois demonstração análoga vale para o segundo. Temos então:

$$|Y_1'| = m_1' \text{ e } \mathfrak{p}_{k-1} Y_1' \subseteq B_1'. \tag{11}$$

Definindo agora, para $i = 1, 2$

$$B_i'' = (\mathfrak{p}_k Y_1') \cap B_i,$$

B_1'' e B_2'' formam uma partição de $\mathfrak{p}_k Y_1'$. Como

$$|Y_1'| = m_1' = S(m_1 - 1, m_2, k),$$

pela hipótese da indução interna, Y_1' contém

um $(m_1 - 1)$-subconjunto Y_1'', com $\mathfrak{p}_k Y_1'' \subseteq B_1''$,

ou

um m_2-subconjunto Y_2'', com $\mathfrak{p}_k Y_2'' \subseteq B_2''$.

No segundo caso, é suficiente tomar $Y_2 = Y_2''$, já que

$$\mathfrak{p}_k Y_2'' \subseteq B_2'' \subseteq B_2.$$

No primeiro caso tomamos

$$Y_1 = Y_1'' \cup \{x\}.$$

Afirmamos que $\mathfrak{p}_k Y_1 \subseteq B_1$. De fato, os k-subconjuntos de Y_1 que não contêm x são k-subconjuntos de Y_1''. Logo, pertencem a B_1'' que é um subconjunto de B_1. Por outro lado, seja A um k-subconjunto de Y_1 que contém x. Então $A' = A \backslash \{x\}$ é um $(k-1)$-subconjunto de Y_1'. De (11), $A' \in B_1'$. Pela construção de B_1', $A' \cup \{x\} = A \in B_1$. Isto completa a demonstração do teorema. ■

Demonstração do Teorema 1. Procedemos por indução sobre o número n de blocos da partição de $\mathfrak{p}_k X$. Para $n = 2$, de (6),

$$R(2, m, k) = S(m, m, k).$$

A tese segue do Teorema 2. Fixamos agora um valor de $n > 2$ e supomos o teorema verdadeiro para todo n', tal que $2 \leq n' < n$.

Sejam $\ell = R(n, m, k)$, X um ℓ-conjunto e $B_1, B_2, \ldots, B_n$ os blocos de uma partição de $\mathfrak{p}_k X$. Definimos

$$B_1' = B_1 \cup B_2 \cup \ldots \cup B_{n-1} \text{ e } B_2' = B_n.$$

B_1' e B_2' formam uma partição de $\mathfrak{p}_k X$ em dois blocos. Como de (7),

$$\ell = R(n, m, k) = S(R(n-1, m, k), m, k),$$

resulta do Teorema 2, que X contém

um $R(n-1, m, k)$-subconjunto Y_1, com $\mathfrak{p}_k Y_1 \subseteq B'_1$,

ou

um m-subconjunto Y_2, com $\mathfrak{p}_k Y_2 \subseteq B'_2$.

No segundo caso a tese é verificada para $i = n$ e $Y = Y_2$. No primeiro caso, os conjuntos

$$B''_i = (\mathfrak{p}_k Y_1) \cap B_i \qquad (i = 1, 2, \ldots, n-1),$$

formam uma partição de $\mathfrak{p}_k Y_1$. Como

$$|Y_1| = R(n-1, m, k),$$

a hipótese da indução implica que existem um i e um m-subconjunto Y de Y_1, tais que $\mathfrak{p}_k Y \subseteq B''_i$. A tese segue diretamente, pois $B''_i \subseteq B_i$. ■

Encerramos esta seção com três observações.

Recorrendo ao princípio da indução transfinita (veja por exemplo no livro de Halmos [47]), podemos eliminar a indução dupla usada na demonstração do Teorema 2. Para tanto, consideramos o subconjunto Q de $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$, definido por

$$Q = \{(m_1, m_2, k) \in \mathbb{N} \times \mathbb{N} \times \mathbb{N} \mid m_1, m_2 \geq k \geq 1\}.$$

Sobre o conjunto Q definimos a relação $\leq$, como segue:

$$\begin{aligned}(m_1, m_2, k) \leq (m'_1, m'_2, k') \text{ sse } & k < k' \\ & \text{ou } k = k' \text{ e } m_1 + m_2 < m'_1 + m'_2 \\ & \text{ou } k = k' \text{ e } m_1 + m_2 = m'_1 + m'_2 \text{ e} \\ & \qquad m_1 \leq m'_1.\end{aligned}$$

É fácil mostrar que esta relação é uma boa-ordem para Q. Demonstrar o Teorema 2 pelo princípio da indução transfinita, usando esta boa-ordem, corresponde exatamente ao que foi feito na demonstração apresentada.

Queremos observar agora, que o Teorema de Ramsey é uma generalização do "princípio da casa do pombo". Este princípio afirma que ao distribuir $n+1$ pombos em n casas, haverá pelo menos uma casa contendo dois pombos (confronte isto com o Exercício D.I. 31). Pois bem, o caso particular do Teorema 1 para $k = 1$, isto é, quando particionamos os 1-subconjuntos de X, vale dizer o próprio X, afirma que (veja (8)):

"Se X é um conjunto de cardinalidade $R(n, m, 1) = n(m-1) + 1$, e $B_1, B_2, \ldots, B_n$ é uma partição de X, então existe um i, tal que $|B_i| \geq m$."

Esta afirmação é uma conseqüência simples do princípio da casa

do pombo. Porém, uma análise cuidadosa da demonstração do Teorema de Ramsey, revela que este é obtido através de uma iteração complexa do caso particular em que $k = 1$. Tal iteração é descrita pelo processo de indução dupla empregada na demonstração do Teorema 2, acrescido da indução com a qual provamos o Teorema 1.

Finalmente observamos que os Teoremas 1 e 2 afirmam a existência de naturais, para valores dados de n, m e k e de m_1, m_2 e k, respectivamente, possuindo as propriedades enunciadas. Daí segue a existência de um menor número natural satisfazendo estas propriedades, chamado de *número de Ramsey*. Denotamos estes por

$$r(n, m, k) \text{ e } s(m_1, m_2, k),$$

nos casos dos Teoremas 1 e 2, respectivamente. Com esta notação, os Teoremas 1 e 2 provam que

$$r(n, m, k) \leq R(n, m, k) \text{ e } s(m_1, m_2, k) \leq S(m_1, m_2, k).$$

Conhece-se muito pouco a respeito das funções r e s; suspeita-se no entanto que os limites superiores R e S excedem de muito os valores respectivos de r e s. A obtenção de limites inferiores e superiores "razoáveis" para r e s é um dos problemas importantes (e aparentemente difíceis) da Combinatória. Alguns dos poucos resultados conhecidos estão relacionados na Seção 3.

3. O caso particular dos grafos ($k = 2$)

O Teorema 2, no caso particular em que $k = 2$, isto é, quando particionamos os 2-subconjuntos de X, admite várias interpretações na linguagem da Teoria dos Grafos. Damos em seguida uma destas, que é particularmente útil para generalizações. Deixamos outras para os exercícios. Note que a Proposição 1 é um corolário do seguinte resultado:

TEOREMA 3. *Dados* $m_1, m_2 \geq 2$, *seja* G *um grafo completo com*

$$\binom{m_1 + m_2 - 2}{m_1 - 1}$$

vértices. Para qualquer coloração das arestas de G *com as cores vermelha e azul,* G *tem um subgrafo completo com* m_1 *vértices cujas arestas são todas vermelhas, ou um subgrafo completo com* m_2 *vértices cujas arestas são todas azuis.*

Demonstração. Seja X o conjunto dos vértices do grafo G. A coloração dada das arestas determina uma partição de $\mathfrak{p}_2 X$ nos blocos B_1 e B_2, correspondentes às arestas vermelhas e azuis, respectivamente. Agora, de (5)

$$S(m_1, m_2, 2) = \binom{m_1 + m_2 - 2}{m_1 - 1}$$

Pelo Teorema 2, X contém um m_1-subconjunto Y_1, com $\mathfrak{p}_2 Y_1 \subseteq B_1$ ou um m_2-subconjunto Y_2, com $\mathfrak{p}_2 Y_2 \subseteq B_2$. Assim, $G[Y_1]$ (ou $G[Y_2]$) é um subgrafo completo de G, com m_1 (ou m_2) vértices, cujas arestas são todas vermelhas (ou azuis). ■

Já mencionamos que o Teorema 2 deixa em aberto a determinação dos valores de $s(m_1, m_2, k)$, limitando-se a garantir a sua existência e afirmar que

$$s(m_1, m_2, k) \leq S(m_1, m_2, k).$$

Naturalmente, $s(m_1, m_2, k) = S(m_1, m_2, k)$ quando $k = 1$ ou $m_1 = k$ ou $m_2 = k$. Todos os outros valores conhecidos de $s(m_1, m_2, k)$ estão contidos na Tabela 1, cujos valores referem-se ao caso particular dos

$k = 2$

m_1 \ m_2	3	4	5	6	7
3	6	9	14	18	23
4	9	18			
5	14				
6	18				
7	23				

Tabela 1 — Valores conhecidos de $s(m_1, m_2, k)$.

grafos ($k = 2$). As demonstrações de alguns destes resultados são complexas, não sendo possível generalizá-las. Quanto aos valores de $r(n, m, k)$ a situação é ainda pior. A única informação não trivial é:

$$r(3, 3, 2) = 17.$$

Além destes valores, existem vários resultados do tipo:

$$102 \leq s(6, 6, 2) \leq 210,$$
$$13 \leq s(4, 4, 3) \leq 19,$$

ou

$$49 \leq r(4, 3, 2) \leq 65.$$

Conhecem-se ainda alguns limites inferiores gerais. Mencionamos dois aqui. O primeiro, demonstrado construtivamente, afirma que

$$r(n, m, 2) \geq \frac{1}{2}[(2m-3)^n + 1] = \ell.$$

Por "demonstrado construtivamente" entendemos que baseando-se na demonstração, podemos exibir uma n-coloração das arestas de um grafo completo G, com ℓ vértices, que satisfaça a seguinte propriedade: nenhum subgrafo completo de G, com m vértices, é monocromático.

O segundo limite inferior é demonstrado não-construtivamente, sendo obtido pelo "método probabilístico".

TEOREMA 4. *Para* $n, m \geq 2$, $r(n, m, 2) \geq 2n^{m/2-1}$. *Em particular,* $s(m, m, 2) \geq 2^{m/2}$.

Demonstração. Consideremos um grafo completo G com $\ell \geq m$ vértices. O número de colorações das arestas de G com as n cores $1, 2, \ldots, n$ é:

$$a = n^{\binom{\ell}{2}}.$$

O número destas colorações para as quais G tem um subgrafo completo de m vértices, cujas arestas são todas de cor i $(i = 1, 2, \ldots, n)$, é b, onde

$$b \leq \binom{\ell}{m} n^{\binom{\ell}{2} - \binom{m}{2}}.$$

Pelo Lema 1, a seguir, se $m \leq \ell < 2n^{m/2-1}$, então $nb < a$. Resulta que neste caso, existe alguma coloração das arestas de G, tal que nenhum dos subgrafos completos de G com m vértices tenha todas as suas arestas de uma mesma cor. Logo,

$$r(n, m, 2) \geq 2n^{m/2-1}.$$

A desigualdade $s(m, m, 2) \geq 2^{m/2}$ segue da observação de que $s(m, m, 2) = r(2, m, 2)$.

LEMA 1. *Se* $m \leq \ell < 2n^{m/2-2}$, *então* $nb < a$.

Demonstração. É fácil ver que

$$nb \leq a\left[\binom{\ell}{m} n^{1-\binom{m}{2}}\right].$$

Por outro lado,

$$\binom{\ell}{m} n^{1-\binom{m}{2}} = \frac{\ell(\ell-1)\ldots(\ell-m+1)}{1\,.\,2\ldots m} n^{1-m(m-1)/2} \leq$$
$$\leq \frac{\ell^{m-1}(\ell-1)n^{1-m(m-1)/2}}{2^{m-1}}.$$

Como $\ell < 2n^{m/2-1}$ por hipótese, segue que

$$\binom{\ell}{m} n^{1-\binom{m}{2}} < \frac{(2n^{m/2-1})^{m-1}(2n^{m/2-1}-1)n^{1-m(m-1)/2}}{2^{m-1}} =$$
$$= n^{2-m}(2n^{m/2-1}-1) = 1-(n^{1-m/2}-1)^2 \leq 1.$$

Assim, $nb < a$. ■ ■

Além da determinação dos valores de $s(m_1, m_2, 2)$, também são de interesse, para um entendimento completo do problema, as colorações, em vermelho e azul, das arestas de grafos completos G com $s(m_1, m_2, 2) - 1$ vértices, para as quais G nem contém um grafo completo vermelho com m_1 vértices, nem um grafo completo azul com m_2 vértices. Os grafos induzidos pelas arestas vermelhas de tais colorações são chamados de *grafos de Ramsey do tipo* (m_1, m_2). Eles possuem em geral estruturas interessantes e um alto grau de simetria. Dois casos conhecidos são dados na Figura 3, outros podem ser encontrados nos Exercícios 6 e 7.

Finalmente mostramos que o Teorema de Ramsey dá lugar a uma infinidade de problemas extremos na Teoria dos Grafos. Sejam G_1 e G_2 dois grafos simples, com m_1 e m_2 vértices. O Teorema 3 implica que para valores suficientemente grandes de ℓ, qualquer coloração das arestas de um grafo completo com ℓ vértices em vermelho e azul, contém um grafo isomorfo a G_1, cujas arestas são vermelhas, ou um grafo isomorfo de G_2, cujas arestas são azuis. Chama-se *número generalizado de Ramsey* $r(G_1, G_2)$, ao menor valor de ℓ satisfazendo a afirmação acima. Definem-se ainda *grafos de Ramsey do tipo* (G_1, G_2), aos grafos induzidos pelas arestas vermelhas das colorações de um

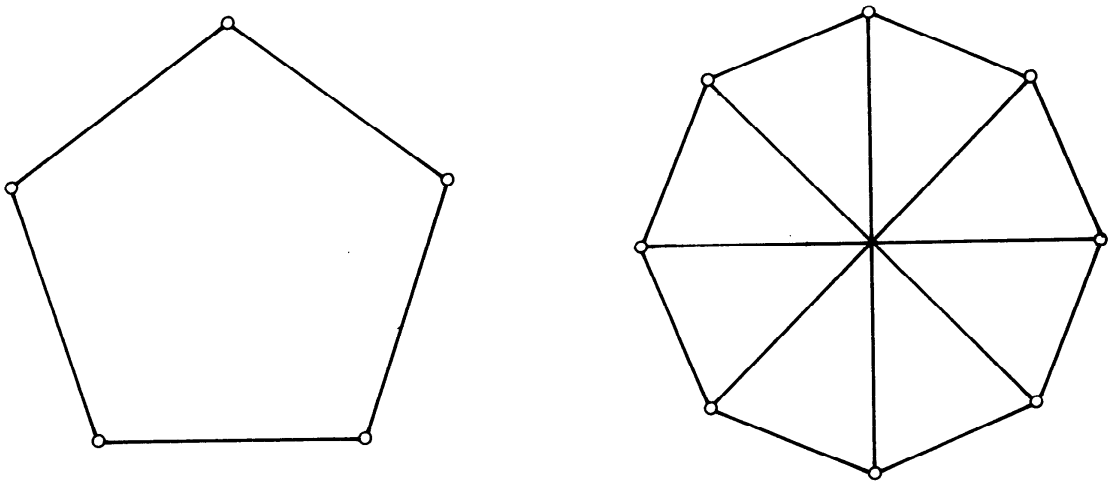


Figura 3 — Grafos de Ramsey do tipo (3,3) e (3,4).

grafo completo com $r(G_1, G_2) - 1$ vértices que nem contêm um G_1 vermelho, nem um G_2 azul.

Alguns destes problemas estão resolvidos, em outros casos têm-se apenas resultados parciais.

4. Outras aplicações do Teorema de Ramsey

Nesta seção apresentamos três aplicações do Teorema de Ramsey, referentes às áreas de Geometria, Teoria dos Números e Teoria dos Semigrupos Finitos.

Uma aplicação à Geometria

TEOREMA 5. *Dado um natural $m \geq 4$, existe um menor natural $f(m)$, tal que quaisquer $f(m)$ pontos no plano, três a três não colineares, contêm m pontos que são vértices de um polígono convexo.*

Demonstração. A demonstração utiliza dois lemas geométricos, cujas provas omitimos, mas que podem ser encontradas em [45] ou [102].

LEMA 2. *Entre cinco pontos no plano, três a três não colineares, existem quatro que são os vértices de um quadrilátero convexo.*

LEMA 3. *Se entre $m \geq 4$ pontos no plano, três a três não colineares, cada quatro são os vértices de um quadrilátero convexo, então os m pontos são os vértices de um polígono convexo.*

Mostramos agora que

$$f(m) \leq S(m, 5, 4).$$

De fato, seja X um conjunto de $S(m, 5, 4)$ pontos no plano, três a três não colineares. Particionamos $\mathfrak{p}_4 X$ em dois blocos, como segue:

$$B_1 = \{A \in \mathfrak{p}_4 X \mid \text{os pontos em } A \text{ são vértices de um quadrilátero convexo}\},$$
$$B_2 = (\mathfrak{p}_4 X) \backslash B_1.$$

Pelo Teorema 2, X contém um m-subconjunto Y_1, com $\mathfrak{p}_4 Y_1 \subseteq B_1$ ou um 5-subconjunto Y_2, com $\mathfrak{p}_4 Y_2 \subseteq B_2$. Do Lema 2 segue que o segundo caso é impossível. Do Lema 3 segue que os m pontos em Y_1 são os vértices de um polígono convexo. ■

Quanto aos valores da função $f(m)$, sabe-se que, para $m \geq 4$

$$2^{m-2} < f(m) \leq \binom{2m-4}{m-2} + 1.$$

Conjetura-se, por outro lado, que $f(m) = 2^{m-2} + 1$ para todo $m \geq 4$. Observamos que o limite inferior acima e a demonstração dada do Teorema 5, implicam que

$$s(m, 5, 4) > 2^{m-2}.$$

Uma aplicação à Teoria dos Números

TEOREMA 6. *Dado um natural $n \geq 2$, existe um menor natural $g(n)$, tal que para qualquer partição do conjunto*

$$\{1, 2, \ldots, g(n)\}$$

em n blocos, existem números x, y e z num mesmo bloco, tais que $x + y = z$.

Demonstração. Vamos mostrar que

$$g(n) \leq R(n, 3, 2).$$

De fato, seja $\ell = R(n, 3, 2)$. Dada uma partição de $X = \{1, 2, \ldots, \ell\}$ em n blocos $C_1, C_2, \ldots, C_n$, definimos a seguinte partição de $\mathfrak{p}_2 X$:

$$B_i = \{\{j_1, j_2\} \in \mathfrak{p}_2 X \mid |j_1 - j_2| \in C_i\} \qquad (i = 1, 2, \ldots, n).$$

Pelo Teorema 1, existe um bloco B_i e um 3-subconjunto $\{j_1, j_2, j_3\}$ de X, tais que $\{j_1, j_2\}$, $\{j_2, j_3\}$, $\{j_1, j_3\}$ pertencem a B_i. Assim

$$|j_1 - j_2|,\ |j_2 - j_3|,\ |j_1 - j_3| \in C_i.$$

Sem perda de generalidade, podemos supor que $j_1 > j_2 > j_3$. Colocando

$$x = j_1 - j_2,\ y = j_2 - j_3 \text{ e } z = j_1 - j_3,$$

resulta que $x, y, z \in C_i$. Por outro lado, $x + y = z$. ■

Quanto aos valores da função $g(n)$, sabe-se que:

$$g(2) = 5,\ g(3) = 14,\ g(4) = 45 \text{ e } \frac{1}{2}\,3^n < g(n) \leq \lfloor n!e \rfloor + 1.$$

Notamos que um limite inferior para $g(n)$ é também um limite inferior para $r(n, 3, 2)$. Esta é uma das razões dos esforços para determinar $g(n)$.

Uma aplicação à Teoria dos Semigrupos Finitos

TEOREMA 7. *Dado um semigrupo finito S e um natural $m \geq 1$, existe um menor natural $k = k(S, m)$ tal que toda seqüência $(a_1, a_2, \ldots, a_k)$ de elementos de S contém m segmentos consecutivos, o produto de cada segmento sendo um mesmo idempotente e de S. Mais precisamente, existem $e \in S$ e $1 \leq j_1 < j_2 < \ldots < j_m < j_{m+1} \leq k$, tais que*

$$a_{j_1} \ldots a_{j_2 - 1} = a_{j_2} \ldots a_{j_3 - 1} = \ldots = a_{j_m} \ldots a_{j_{m+1} - 1} = e = e^2.$$

Demonstração. Mostramos que para $m \geq 2$,

$$k(S, m) \leq R(|S|, m + 1, 2).$$

(Para $m = 1$ o resultado segue do caso em que $m = 2$.) De fato, seja $\ell = R(|S|, m + 1, 2)$. Definimos a seguinte partição dos 2-subconjuntos de $X = \{1, 2, \ldots, \ell\}$:

$$B_a = \left\{\{i, j\} \in \mathfrak{p}_2 X \mid i < j, \text{ e } a_i \ldots a_{j-1} = a\right\} \qquad (a \in S).$$

Pelo Teorema 1, existem $e \in S$ e um $(m + 1)$-subconjunto Y de X, tais que $\mathfrak{p}_2 Y \subseteq B_e$. Sendo $j_1 < j_2 < \ldots < j_{m+1}$ os elementos de Y, resulta que $\{j_1, j_2\}$, $\{j_2, j_3\}$ e $\{j_1, j_3\}$ pertencem a B_e isto é

$$\begin{aligned} e &= a_{j_1} \ldots a_{j_2 - 1} = a_{j_2} \ldots a_{j_3 - 1} = \\ &= (a_{j_1} \ldots a_{j_2 - 1})\,(a_{j_2} \ldots a_{j_3 - 1}) = e^2. \end{aligned}$$

Logo $e = e^2$, isto é, e é um idempotente de S. A tese segue da pertinência de $\{j_1, j_2\}$, $\{j_2, j_3\}$, ..., $\{j_m, j_{m+1}\}$ a B_e. ■

EXERCÍCIOS

1. Dados naturais $m_1, m_2 \geq 1$, mostre que existe um menor natural $\ell = h(m_1, m_2)$ tal que toda seqüência $s = (r_1, r_2, \ldots, r_\ell)$ de números reais contém uma subseqüência crescente de m_1 termos ou uma decrescente de m_2 termos. Mostre que $h(m_1, m_2) = (m_1 - 1)(m_2 - 1) + 1$. (Uma *subseqüência crescente* de m termos de s é uma seqüência $(r_{i_1}, r_{i_2}, \ldots, r_{i_m})$, tal que $1 \leq i_1 < i_2 < \ldots < i_m \leq \ell$ e $r_{i_1} \leq r_{i_2} \leq \ldots \leq r_{i_m}$.)
2. Demonstre a seguinte versão generalizada do Teorema de Ramsey: Dados $n \geq 2$ e $m_1, m_2, \ldots, m_n \geq k \geq 1$, existe um menor natural $\ell = N(m_1, m_2, \ldots, m_n, k)$, tal que dados um ℓ-conjunto X e uma partição de $\mathfrak{p}_k X$ em n blocos $B_1, B_2, \ldots, B_n$, existe um i $(1 \leq i \leq n)$ e um m_i-subconjunto Y de X, tais que $\mathfrak{p}_k Y \subseteq B_i$.
3. Prove que os números definidos no exercício anterior, satisfazem as relações abaixo:
   (a) $N(m_1, m_2, \ldots, m_n, k) \leq 1 + N(m_1', m_2', \ldots, m_n', k-1)$, onde $m_i' = N(m_1, \ldots, m_{i-1}, m_i - 1, m_{i+1}, \ldots, m_n, k)$,
   (b) $N(m_1, m_2, \ldots, m_n, 1) = 1 + \sum_{i=1}^{n} (m_i - 1)$,
   (c) $N(m_1, m_2, \ldots, m_n, 2) \leq \dfrac{(m_1 + m_2 + \ldots + m_n - n)!}{(m_1 - 1)!\,(m_2 - 1)! \ldots (m_n - 1)!}$.
4. Mostre que todo grafo simples com $s(m_1, m_2, 2)$ vértices tem um subgrafo completo com m_1 vértices ou um conjunto independente de m_2 vértices.
5. Mostre que se $s(m_1 - 1, m_2, 2)$ e $s(m_1, m_2 - 1, 2)$ são ambos pares, então $s(m_1, m_2, 2) \leq s(m_1 - 1, m_2, 2) + s(m_1, m_2 - 1, 2) - 1$. Como uma aplicação, mostre que $s(3, 4, 2) \leq 9$.
6. Mostre que o grafo cujos vértices são os resíduos $0, 1, \ldots, 7$ (*mod* 8) e no qual vértices i e j são adjacentes se e somente se $i - j \equiv \pm 1, \pm 4$ (*mod* 8) é um grafo de Ramsey do tipo (3, 4). Conclua que $s(3, 4, 2) = 9$.
7. Mostre que o grafo cujos vértices são os resíduos $0, 1, \ldots, 16$ (*mod* 17) e no qual vértices i e j são adjacentes se e somente se $i - j = \pm 1, \pm 2, \pm 4, \pm 8$ (*mod* 17) é um grafo de Ramsey do tipo (4, 4). Mostre que $s(4, 4, 2) = 18$.
8. Mostre que $s(4, 4, 3) \leq 19$. Ache o melhor limite inferior que você consegue para $s(4, 4, 3)$.

9. Mostre que $f(4) = 5$ e $f(5) = 9$. (Veja Teorema 5.)
10. Mostre que para $n \geq 3$, $r(n, 3, 2) \leq nr(n-1, 3, 2) + 2$. Usando $r(2, 3, 2) = 6$, mostre que $r(n, 3, 2) \leq \lfloor n!e \rfloor + 1$. Conclua que $g(n) \leq \lfloor n!e \rfloor + 1$. (Veja Teorema 6.)
11. Mostre que $g(2) = 5$ e $g(3) = 14$ (veja Teorema 6). Mostre que para todo $n \geq 3$, $g(n) \geq 3g(n-1) - 1$. Mostre que $g(n) > \frac{1}{2} 3^n$ para $n \geq 2$. Conclua que $r(n, 3, 2) > \frac{1}{2} 3^n$.
12. Mostre que se S for um grupo finito, então $k(S, m) = m \mid S \mid$. Mostre que se S é um semigrupo finito, então $k(S, 1) \leq 2^{|S|}$. (Veja Teorema 7.)
13. Usando os Exercícios 3 e 5 e os resultados na Tabela 1, mostre que $s(5, 5, 3) \leq 2 \times 10^{4403}$.
14. Compare os limites inferiores para $r(n, m, 2)$, dados pelo Teorema 5 e pelo limite inferior construtivo.
15. Mostre que a seguinte versão "orientada" do Teorema de Ramsey é falsa para todo $k \geq 2$: Dados naturais n, m e k, tais que $m \geq k \geq 1$ e $n \geq 2$, existe um natural $\ell = R_0(n, m, k)$, tal que dados um ℓ-conjunto X e uma partição de X^k em n blocos $B_1, B_2, \ldots, B_n$, existem um bloco B_i e um m-subconjunto Y de X, tais que $Y^k \subseteq B_i$. (Aqui, X^k denota o produto cartesiano de X com si mesmo, k vezes.)
16. Dados naturais $m, p \geq 1$, mostre que existe um menor natural $\ell = \ell(m, p)$, tal que para todo ℓ-subconjunto X de um domínio de integridade D, de característica distinta de 3, X contém um m-subconjunto Y, tal que para todo x, y, z em Y, $x^p + y^p + z^p \neq 0$. Sugestão – Observe que para $a, b \in D$, o polinômio $ax^p + b$ tem no máximo p raízes em D. Mostre que $\ell(m, p) \leq p + s(p+3, s(2p+2, m, 2), 3)$.
17. Dados naturais $m_1, m_2 \geq 2$, prove que existe um menor natural $\ell = h(m_1, m_2)$, tal que toda ordem parcial finita de cardinalidade pelo menos ℓ contém uma cadeia de cardinalidade m_1 ou uma anticadeia de cardinalidade m_2. Prove que $h(m_1, m_2) = (m_1 - 1)(m_2 - 1) + 1$. Obtenha o Exercício 1 como corolário da afirmação acima. (Um subconjunto de uma ordem parcial é uma *cadeia* (*anticadeia*) se seus elementos forem dois a dois comparáveis (incomparáveis).)

18. Prove a versão infinita do Teorema de Ramsey: Dados naturais $n \geq 2$ e $k \geq 1$, um conjunto infinito X e uma partição de $\mathfrak{p}_k X$ em n blocos $B_1, B_2, \ldots, B_n$, existem um bloco B_i e um subconjunto infinito Y de X, tais que $\mathfrak{p}_k Y \subseteq B_i$.

19. Dados naturais $n, m \geq 1$, prove que existe um menor natural $\ell = p(n, m)$, tal que dada uma partição do semigrupo livre Σ^+ ($\Sigma^+ = \Sigma^* \backslash 1$), gerado por Σ, em n blocos $B_1, B_2, \ldots, B_n$, toda palavra de Σ^* de comprimento pelo menos ℓ pertence a $\Sigma^* B_i^m \Sigma^*$ para algum i, $1 \leq i \leq n$. Prove que $p(m, n) = m^n$. Conclua que $r(n, m, 2) > m^n$. (Veja Seção D.I.2.)

20. Dados naturais $n \geq 2$, $m \geq k \geq 1$ e $p \geq 1$, prove que existe um menor natural $\ell = t(n, m, k, p)$, tal que dados um alfabeto Σ de cardinalidade p e uma partição de Σ^k em n blocos $B_1, B_2, \ldots, B_n$, toda palavra em Σ^*, de comprimento pelo menos ℓ, contém uma subpalavra de comprimento m, cujas subpalavras de comprimento k pertencem a um mesmo bloco B_i. (Dadas palavras x e y em Σ^*, x é uma *subpalavra* de y, $x \leq y$, se existem um natural q e palavras $x_1, x_2, \ldots, x_q, y_0, y_1, \ldots, y_q$, tais que $x = x_1 x_2 \ldots x_q$ e $y = y_0 x_1 y_1 \ldots x_q y_q$. Assim, $x \leq y$ se e somente se y pertence ao embaralhamento de $\{x\}$ com Σ^* (veja Exercício D.II.8).)

NOTAS BIBLIOGRÁFICAS

O Teorema 1 foi demonstrado por F. P. Ramsey em 1930 [100] e, independentemente, por P. Erdös e G. Szekeres em 1935 [30]. A demonstração apresentada segue [30]. Ramsey apresentou também uma versão infinita de seu teorema (Exercício 18), e que vem sendo estudada em Lógica Matemática. Recentemente o Teorema de Ramsey tem inspirado avanços significativos em Combinatória [40], [82], [88].

Os números de Ramsey contidos na Tabela 1 foram determinados por Greenwood, Gleason, Graver e Yackel [41], [43], [33]. O limite inferior construtivo para $r(n, m, 2)$ é de Giraud [37], enquanto o Teorema 5 é baseado no trabalho de Erdös [28]. Neste artigo clássico, Erdös introduziu o "método probabilístico" que tem sido aplicado extensamente na obtenção de limites inferiores e superiores para problemas extremos [29]. Os números generalizados de Ramsey foram introduzidos por Chvátal e Harary [14]. Progressos mais recentes podem ser encontrados em [11].

O Teorema 5, bem como os limites e a conjetura para a função $f(m)$ são devidos a Erdös e Szekeres [30], [31], [117].

O Teorema 6 é de Schur [106], antecedendo o próprio Teorema de Ramsey. Maiores detalhes sobre a função $g(n)$ podem ser encontrados em [126].

O Teorema 7 é de McNaughton [74], e tem aplicações na demonstração da decidibilidade de problemas na Teoria dos Autômatos Finitos [74], [76]. O Exercício 18 é de Schützenberger [109].

CAPÍTULO VI

GRAFOS ORIENTADOS

1. Introdução

Neste capítulo introduzimos grafos orientados, como sendo aqueles em que a cada aresta é dada uma "direção". Desta forma falaremos de cortes e circuitos orientados, como sendo aqueles em que todas as arestas tem a mesma "direção". O Teorema da dicotomia, da Secção 2, afirma que as arestas de um grafo orientado são de dois tipos: aquelas que pertencem a cortes orientados e aquelas pelas quais passam circuitos orientados, cada caso excluindo o outro. Na Seção 3, estudaremos os grafos fortemente conexos. Na Seção 4, os grafos acíclicos.

Uma *orientação* de um grafo G é um par ordenado (i, f) de funções, ambas de aG em VG, tais que para cada aresta α em aG, $i\alpha$ e $f\alpha$ são os extremos de α em G: $i\alpha$ é o *extremo inicial* de α e $f\alpha$ o *extremo final* de α. Um *grafo orientado* D consiste de um grafo GD, chamado de *grafo subjacente* de D, e de uma orientação de GD.

Na representação de um grafo orientado por um diagrama, colocamos, em cada aresta α, uma flecha que aponta de $i\alpha$ para $f\alpha$. Assim, no grafo orientado da Figura 1, v_1 é o extremo inicial das arestas α_1 e α_2 e é o extremo final de α_1, α_5 e α_7.

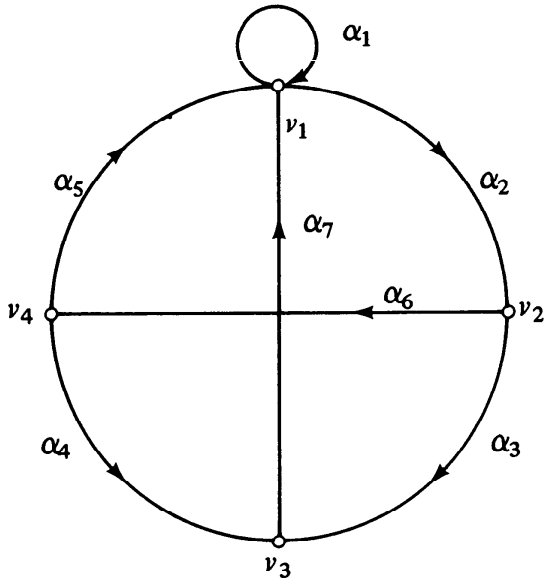


Figura 1 — Um grafo orientado.

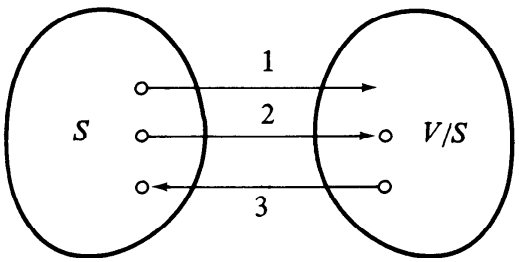
No que segue, ao aplicarmos um conceito ou propriedade, definido para grafos (não orientados) a um grafo orientado, subentendemos que estamos aplicando o conceito ou propriedade ao seu grafo subjacente. Assim podemos dizer que um grafo orientado é conexo, ou que P é um passeio num grafo orientado, etc.

Seja D um grafo orientado, (i, f) sua orientação. Para um subconjunto X de V, o *subgrafo orientado* $D[X]$ *gerado* por X é o grafo orientado que tem $GD[X]$ como grafo subjacente e (i_X, f_X) como orientação, onde $i_X\alpha = i\alpha$ e $f_X\alpha = f\alpha$, para cada aresta α de $GD[X]$.

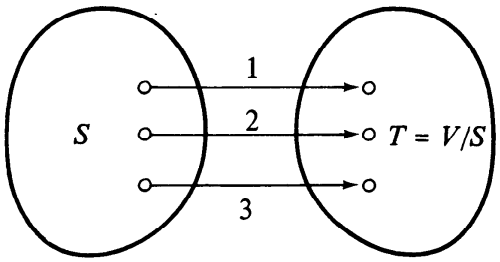
Um passeio $P = (v_0, \alpha_1, v_1, \ldots, \alpha_k, v_k)$ em D é *orientado* se para todo j tal que $1 \leq j \leq k$, $i\alpha_j = v_{j-1}$ (ou de maneira equivalente, $f\alpha_j = v_j$). O circuito $(v_1, \alpha_2, v_2, \alpha_6, v_4, \alpha_4, v_3, \alpha_7, v_1)$ no grafo orientado da Figura 1 é orientado.

Definimos a seguir duas relações sobre V. A primeira é a *relação de acesso*, denotada por $\rightarrow_D$ (ou simplesmente, $\rightarrow$ se D for subentendido): u *tem acesso* a v em D ($u \rightarrow v$) se existe um caminho orientado de u a v em D. A segunda é a *relação de ligação forte*, denotada por $\leftrightarrow_D$ (ou simplesmente, $\leftrightarrow$ se D for subentendido): u é *fortemente ligado* a v em D ($u \leftrightarrow v$) se $u \rightarrow v$ e $v \rightarrow u$. Naturalmente, a relação de acesso é reflexiva e transitiva, e a de ligação forte é de equivalência. Em vista da simetria da relação de ligação forte, dizemos simplesmente que u e v são *fortemente ligados* em D quando u é fortemente ligado a v em D.

Considere o corte $\delta(S)$ associado a um subconjunto S de V; se todas as arestas de $\delta(S)$ têm seu extremo inicial (final) em S então S é uma *fonte* (*sorvedouro*) e em ambos os casos o corte $\delta(S)$ de S é *orientado*. Um subconjunto d de a é um *corte orientado* se existe um subconjunto S de V tal que o corte $\delta(S)$ de S é orientado e igual a d. A Figura 2 ilustra esses conceitos.



o corte $\delta(S)$ de S igual a $\{1, 2, 3\}$.



S é uma fonte, T um sorvedouro, $\delta(S) = \delta(T)$ é orientado, $\{1, 2, 3\}$ é um corte orientado.

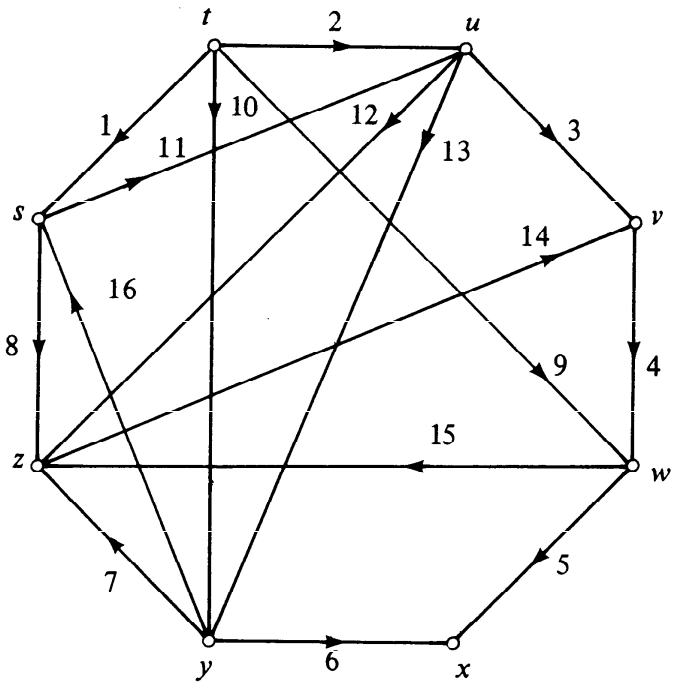
Figura 2

2. O Teorema da dicotomia

Para D um grafo orientado, seja *cir* D o conjunto das arestas de D pelas quais passam circuitos orientados; seja *cor* D o conjunto das arestas de D que pertencem a cortes orientados.

TEOREMA 1 (Teorema da dicotomia). *Seja D um grafo orientado. Então a união de cir D e cor D é aD, sua interseção o vazio. (Ou seja, toda aresta de D pertence a um circuito orientado, ou a um corte orientado, mas não a ambos.)*

A título de ilustração, verificamos o Teorema 1 para o grafo orientado da Figura 3.



$C_1 = (s, 11, u, 13, y, 16, s)$
$C_2 = (v, 4, w, 15, z, 14, v)$
$\delta(\{x\}) = \{5, 6\}$
$\delta(\{v, w, x, z\}) = \{3, 6, 7, 8, 9, 12\}$
$\delta(\{t\}) = \{1, 2, 9, 10\}$

Figura 3 — Uma ilustração do Teorema da dicotomia.

Demonstração do Teorema 1. Seja α uma aresta de D, considere as seguintes afirmações:

(i) $\alpha \in cor\ D$,

(ii) existe em D um sorvedouro que contém $f\alpha$ mas não $i\alpha$,

(iii) $f\alpha \not\to i\alpha$,

(iv) $\alpha \notin cir\ D$.

A equivalência de (i) e (ii) é imediata. As equivalências de (ii) e (iii), e de (iii) e (iv) seguem dos Lemas 2 e 1, abaixo, respectivamente. Assim, (i) e (iv) são equivalentes, o que mostra o teorema.

LEMA 1. *Seja D um grafo orientado, α uma aresta de D. Então $f\alpha \to i\alpha$ se e somente se $\alpha \in cir\ D$.*

LEMA 2. *Seja D um grafo orientado, u e v vértices de D. Então $u \to v$ se e somente se todo sorvedouro que contém u contém também v.*

Demonstração do Lema 1. Se C é um circuito orientado que passa por α então existem seções C_1 e C_2 de C tais que $C = C_1(i\alpha, \alpha, f\alpha)C_2$; nesse caso, C_2C_1 é um caminho orientado de $f\alpha$ a $i\alpha$ e portanto $f\alpha \to i\alpha$.

Por outro lado, se $C = (v_0, \alpha_1, v_1, \ldots, \alpha_n, v_n)$ é um caminho orientado de $f\alpha$ a $i\alpha$ em D então $C(i\alpha, \alpha, f\alpha)$ é um circuito orientado que passa por α, pois para todo inteiro j tal que $1 \leq j \leq n$, temos que $v_0 \neq v_j$, ou seja $f\alpha \neq f\alpha_j$ e portanto $\alpha \notin aC$. A demonstração do lema está completa. ■

Demonstração do Lema 2. Suponha que $u \to v$, seja S um sorvedouro que contém u, $C = (v_0, \alpha_1, v_1, \ldots, \alpha_n, v_n)$ um caminho orientado de u a v em D. Como $v_0 = u$, então $v_0 \in S$. Como S é um sorvedouro, então, para todo j tal que $0 < j < n$, se v_{j-1} pertence a S então $v_j \in S$. Segue, por indução, que $VC \subseteq S$. Em particular, $v_n \in S$. Ou seja, $v \in S$. De fato, se $u \to v$ então todo sorvedouro que contém u contém v também.

Suponha agora que todo sorvedouro que contém u contém v também. Seja S o conjunto dos vértices a que u tem acesso. Para toda aresta α tal que $i\alpha \in S$, temos que $u \to i\alpha$ e $i\alpha \to f\alpha$, portanto $u \to f\alpha$, ou seja, $f\alpha \in S$; S é então um sorvedouro. Como $u \to u$, então $u \in S$ e portanto $v \in S$. Ou seja, $u \to v$. De fato, se todo sorvedouro que contém u contém v então $u \to v$. A demonstração do lema completa a do Teorema 1. ■ ■

O leitor atento notará a semelhança entre a demonstração do Lema 2 e as dos Lemas I.1 e I.2.

3. Grafos fortemente conexos

Conforme foi visto na Seção 1, a relação de ligação forte é de equivalência. Seja p a partição de V que consiste das classes de equivalência da relação de ligação forte em um grafo orientado D. Para

cada elemento X de p, $D[X]$ é um *componente forte* de D. O conjunto cfD dos componentes fortes de D é então igual a $\{D[X] \mid X \in p\}$. A Figura 4 mostra os componentes fortes de um grafo D. Um grafo orientado D é *fortemente conexo* se quaisquer dois de seus vértices são fortemente ligados. O grafo da Figura 1 é fortemente conexo.

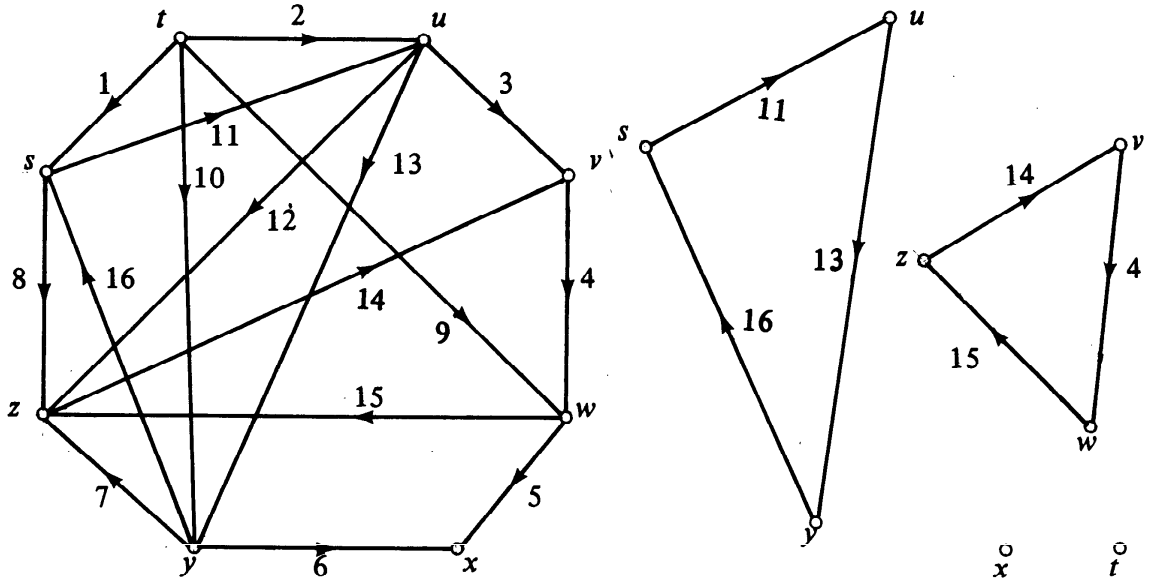


Figura 4 — Os 4 componentes fortes do grafo orientado da Figura 3.

Caracterizamos agora os grafos fortemente conexos:

TEOREMA 2. *Seja D um grafo orientado. Então as seguintes afirmações são equivalentes:*

(i) *D é fortemente conexo,*
(ii) *D é conexo e $cir\ D = aD$,*
(iii) *D é conexo e $cor\ D = \varnothing$,*
(iv) *VD e o vazio são os únicos sorvedouros em D.*

Demonstração. Para mostrar que (i) implica (ii), suponha que D é fortemente conexo. Então D é certamente conexo. Ademais, para cada aresta α de D, $f\alpha \to i\alpha$ e portanto $\alpha \in cir\ D$, pelo Lema 1. De fato, (i) implica (ii).

Pelo Teorema da dicotomia, (ii) implica (iii).

Para mostrar que (iii) implica (iv), suponha que (iii) vale, seja S um subconjunto próprio e não vazio de V. Como D é conexo, então o corte $\delta(S)$ não é vazio, pelo Corolário I.2. Como $cor\ D = \varnothing$ então $\delta(S)$ não é orientado. Portanto, S não é um sorvedouro. De fato, (iii) implica (iv).

Finalmente, (iv) implica (i), pelo Lema 2. A demonstração do teorema está completa. ■

Uma orientação de um grafo é *fortemente conexa* se o grafo orientado resultante for fortemente conexo.

TEOREMA 3. *Um grafo G admite uma orientação fortemente conexa se e somente se G é conexo e não tem arestas de corte.*

Demonstração. Suponha inicialmente que G admite uma orientação fortemente conexa, D. Pelos itens (i) e (iii) do Teorema 2, D é conexo e $cor\ D = \varnothing$. O grafo G não tem arestas de corte pois se uma aresta, α, fosse de corte, então $\{\alpha\}$ seria um corte orientado, qualquer que fosse a orientação de G.

Reciprocamente, suponha que G é conexo e não tem arestas de corte. Dentre os grafos orientados que têm G como grafo subjacente, escolha um, D, tal que $cor\ D$ seja minimal. Pelos itens (ii) e (iii) do Teorema 2, basta agora provar que $cor\ D = \varnothing$.

Para tanto, suponha o contrário, seja α uma aresta em $cor\ D$. Como α não é de corte, então, pelo Teorema I.3, existe em D um circuito, C, que passa por α. Altere a orientação de algumas arestas em aC, obtendo então um novo grafo orientado, D', de forma que C seja orientado em D'. Assim, $aC \subseteq cir\ D'$. Pelo Teorema da dicotomia,

$$cor\ D' \subseteq aD \backslash aC. \tag{1}$$

Provamos em seguida que $cor\ D'$ é um subconjunto próprio de $cor\ D$. De fato, seja S um sorvedouro em D'; de (1), $\delta D'(S) \subseteq aD \backslash aC$. Portanto, S é um sorvedouro em D, pois não foi alterada a orientação de arestas em $aD \backslash aC$. Logo, $cor\ D' \subseteq cor\ D \backslash aC$. Como α pertence tanto a $cor\ D$ como a aC, então $cor\ D'$ é um subconjunto próprio de $cor\ D$, em contradição à escolha de D. De fato, $cor\ D$ é vazio. A demonstração do Teorema 3 está completa. ■

4. Grafos acíclicos

Um grafo orientado D é *acíclico* se não existe em D circuito orientado. O próximo resultado mostra como se obtém um grafo acíclico de um grafo orientado D, mediante a contração de cada componente forte a um vértice.

Seja D um grafo orientado, (i, f) sua orientação. A *função de condensação* g de D é a função de VD em cfD que associa a cada vértice v de D o componente forte de D da qual v é um vértice. Para cada subconjunto X de V, gX denota o conjunto $\{gv \mid v \in X\}$. A *condensação* CD de D é o grafo orientado tal que cfD é o conjunto de vértices de CD, $cor\ D$ o conjunto de arestas de CD, e para cada aresta α em aCD, o extremo inicial de α em CD é $gi\alpha$, seu extremo final $gf\alpha$.

TEOREMA 4. *A condensação CD de um grafo orientado D é acíclica.*

Demonstração. Seja S um sorvedouro em D. Temos então:

(1) Para cada vértice v em V, v pertence a S se e somente se gv pertence a gS.

Obviamente, se v pertence a S, então gv pertence a gS. Por outro lado, se gv pertence a gS então para algum vértice u fortemente ligado a v em D, u pertence a S; nesse caso, pelo Lema 2, v pertence a S.

De (1), seguem imediatamente as seguintes afirmações:

(2) O conjunto gS é um sorvedouro em CD.

(3) O corte de gS em CD é igual ao corte de S em D.

De (2) e (3), temos então que todo corte orientado em D é um corte orientado em CD. Portanto,

$$cor\ D \subseteq cor\ CD \subseteq aCD = cor\ D.$$

Ou seja, $cor\ CD = aCD$. Pelo Teorema da dicotomia, CD é acíclico. A demonstração do teorema está completa. ■

Damos a seguir caracterizações de grafos acíclicos.

Uma *classificação* de um grafo orientado D é uma função h de V em $\mathbb{N}$, onde $\mathbb{N}$ é o conjunto dos números naturais. A classificação h é *consistente* se $hi\alpha < hf\alpha$ para toda aresta α de D. Uma aresta α é uma *inversão* em relação a h se $hia \geq hf\alpha$. Assim, h é consistente se nenhuma aresta em D é uma inversão em relação a h.

TEOREMA 5. *Seja D um grafo orientado. Então as seguintes afirmações são equivalentes:*

(i) *D é acíclico,*

(ii) *$cor\ D = aD$,*

(iii) *D admite uma classificação (injetora) consistente,*

(iv) *D não tem laços e a relação de acesso é uma ordem parcial sobre VD (isto é, $\rightarrow$ é reflexiva, anti-simétrica e transitiva).*

A Figura 5 mostra um grafo orientado acíclico D e uma classificação injetora consistente de D.

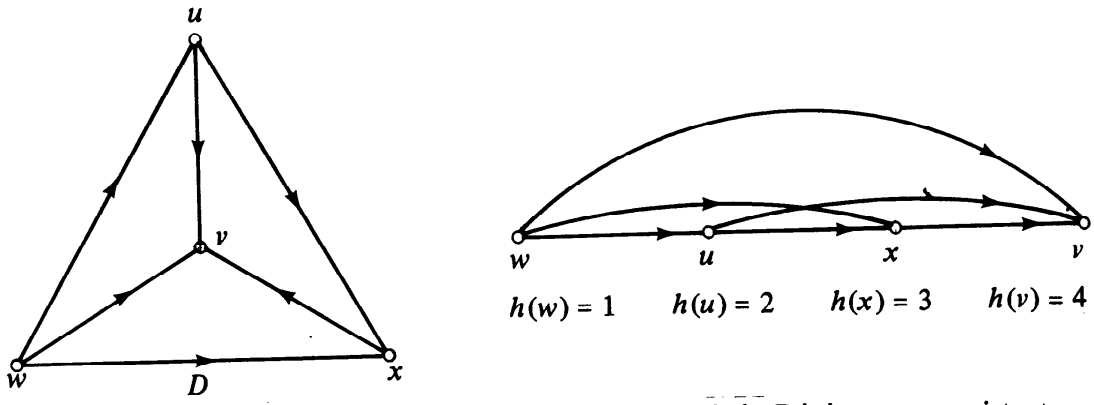


Figura 5 — Um grafo acíclico D e uma classificação h de D injetora e consistente.

Demonstração do Teorema 5. A equivalência de (i) e (ii) segue do Teorema da dicotomia.

Provemos agora que (i) implica (iii), por indução em $|VD|$. Se VD for vazio então (iii) é trivialmente válida. Suponha pois que VD não é vazio. Para cada vértice v em VD, seja Sv o conjunto dos vértices a que v tem acesso. Dentre os vértices de D, escolha um, v, tal que Sv seja minimal.

LEMA 3. *Nenhuma aresta de D tem v como extremo inicial.*

Demonstração. Seja α uma aresta de D. Como $i\alpha \rightarrow f\alpha$, então $Sf\alpha \subseteq Si\alpha$. Como D é acíclico, então, pelo Lema 1, $f\alpha \not\rightarrow i\alpha$. Portanto, $Sf\alpha \subseteq Si\alpha \backslash \{i\alpha\} \subsetneq Si\alpha$. Ou seja, $Si\alpha$ não é minimal. Como esta conclusão vale para toda aresta α de D, então, pela definição de v, nenhuma aresta de D tem v como extremo inicial. A demonstração do lema está completa. ■

Seja D' o grafo $D[V\backslash\{v\}]$. Como D é acíclico, D' também é acíclico. Por indução, D' admite uma classificação injetora consistente, digamos, h'.

Seja h a extensão de h' a VD tal que $hv = 1 + max\{h'u \mid u \in VD'\}$. Certamente h é uma classificação injetora de D. Para verificar que h é consistente, seja α uma aresta de aD. Se α pertence a aD' então $hi\alpha = h'i\alpha < h'f\alpha = hf\alpha$; se α pertence a $aD \backslash aD'$ então, pelo Lema 3, $i\alpha \in VD'$ e $f\alpha = v$, portanto $hi\alpha = h'i\alpha < hv = hf\alpha$. Em ambos os casos, $hi\alpha < hf\alpha$. De fato, h é uma classificação injetora consistente, de D.

Para provar que (iii) implica (iv), seja h uma classificação consistente de D. Para toda aresta α de D, $hf\alpha > hi\alpha$ e portanto $f\alpha \neq i\alpha$; logo, D não tem laços. Como $\rightarrow$ é reflexiva e transitiva independente-

mente de (iii) valer ou não, resta então mostrar que $\rightarrow$ é anti-simétrica. Para tanto, seja $P = (v_0, \alpha_1, v_1, \ldots, \alpha_n, v_n)$ um passeio orientado e não degenerado. Como h é consistente, então $hv_j < hv_{j+1}$, para todo j tal que $0 \leq j \leq n-1$. Como P é não degenerado, então $hv_0 < hv_n$. Portanto, se u e v são vértices distintos em D e $u \rightarrow v$ então $hu < hv$; consequentemente, $u \leftrightarrow v$ se e somente se $u = v$. De fato, (iii) implica (iv).

Para mostrar que (iv) implica (i), seja α uma aresta de D; como D não tem laços, então $i\alpha \neq f\alpha$; como $\rightarrow$ é anti-simétrica e $i\alpha \rightarrow f\alpha$, então $f\alpha \not\rightarrow i\alpha$: pelo Lema 1, $\alpha \notin cir\ D$. De fato, (iv) implica (i).

A demonstração do teorema está completa. ■

Uma orientação de um grafo é *acíclica* se o grafo orientado resultante for acíclico.

COROLÁRIO 1. *Um grafo G admite uma orientação acíclica se e somente se G não tem laços.*

Um vértice v de um grafo orientado D é um *vértice-fonte* de D (*vértice-sorvedouro* de D) se nenhuma aresta de D tem v como seu extremo final (inicial).

COROLÁRIO 2. *Todo grafo orientado acíclico e não vazio tem um vértice-fonte e um vértice-sorvedouro.*

EXERCÍCIOS

1. Mostre que um passeio orientado de comprimento mínimo de u a v em D é um caminho (orientado). O comprimento desse caminho é a *distância orientada* de u a v em D.
2. Mostre que existe um passeio orientado de u a v se e somente se existe um caminho orientado de u a v.
3. Mostre que o passeio orientado, fechado e não degenerado $C = (v_0, \alpha_1, v_1, \ldots, \alpha_n, v_n)$ é um circuito (orientado) se e somente se $v_1, \ldots, v_n$ forem dois a dois distintos. Confronte esta afirmação com aquela do Exercício I.47.
4. Mostre que se P é um passeio orientado fechado então para cada aresta α em aP existe um circuito orientado que passa por α. Mostre que a versão não orientada desta afirmação é falsa.
5. Escreva versões orientadas do Teorema I.1 e do procedimento *distância* da Seção I.6. Modifique o algoritmo assim obtido de

forma a obter outro que determine um caminho orientado de comprimento mínimo entre dois vértices dados. Escreva um algoritmo que determina os componentes fortes de um grafo orientado.

6. Escreva um algoritmo que determina um circuito orientado de comprimento mínimo em um grafo orientado.
7. Mostre que D não vazio é fortemente conexo se e somente se existe um passeio orientado e fechado P em D tal que $V = VP$. Mostre que a afirmação não é necessariamente verdadeira se substituirmos "passeio" por "trilha".
8. (*Generalização do Lema* 1) – Seja P um passeio orientado em um grafo orientado D, u a origem de P, v seu término. Mostre que $v \rightarrow u$ se e somente se $aP \subseteq cir\ D$.
9. Ache um grafo orientado D, um subconjunto d de a e um subconjunto S de V tais que o corte $\delta(S)$ de S não é orientado, $d = \delta(S)$ e d é um corte orientado.
10. Ache um grafo orientado D com 6 vértices tal que $aD = cor\ D = cir\ D$.
11. Prove que para um grafo orientado D, as afirmações seguintes são equivalentes:
    (i) H é um componente forte de D,
    (ii) H é um componente forte de $D - cor\ D$,
    (iii) H é um componente de $D - cor\ D$.
12. Prove que para um grafo orientado D as afirmações abaixo são equivalentes:
    (i) Cada componente de D é fortemente conexo,
    (ii) $cir\ D = aD$,
    (iii) $cor\ D = \varnothing$,
    (iv) A relação de acesso é de equivalência,
    (v) O conjunto vazio é o único corte orientado em D.
13. Prove que para um grafo orientado D as afirmações abaixo são equivalentes:
    (i) D é acíclico,
    (ii) cada componente forte de D consiste de um só vértice e não possui arestas,
    (iii) GD e GCD são isomorfos.
14. Seja D um grafo orientado acíclico e C um caminho orientado de comprimento máximo em D. Mostre que a origem (término) de C é um vértice-fonte (vértice-sorvedouro) de D.

15. Escreva um algoritmo eficiente que para um dado grafo orientado não vazio encontra um vértice-sorvedouro ou um circuito orientado.
16. Escreva um algoritmo eficiente que determina se um grafo orientado D é acíclico ou não; em caso afirmativo o algoritmo produz uma classificação consistente de D, em caso negativo o algoritmo produz um circuito orientado em D. Modifique o algoritmo assim obtido de forma a obter outro que determine os componentes fortes de um grafo orientado.
17. Escreva um algoritmo eficiente que encontra um caminho orientado de comprimento máximo num grafo acíclico.
18. Escreva um algoritmo que encontra um caminho orientado de comprimento máximo num grafo orientado. Este seu algoritmo é eficiente?
19. Seja D um grafo acíclico e não vazio. Um caminho orientado C em D é *crítico* se nenhum caminho orientado em D tem comprimento maior do que o de C. Mostre que uma coleção mínima de cortes orientados cuja união é aD tem cardinalidade igual ao comprimento de um caminho crítico em D.
20. Seja G um grafo. Mostre que existe uma coloração dos vértices de G com não mais do que s cores se e somente se existe uma orientação de G na qual todo passeio orientado tem comprimento menor do que s.

NOTAS BIBLIOGRÁFICAS

Os resultados deste capítulo fazem parte do folclore da Teoria dos Grafos. Maiores informações sobre grafos orientados podem ser encontrados em [13], [50] e [110].

Pelo Teorema 5, os grafos que não são acíclicos não admitem uma classificação consistente. Dizemos, assim, que uma classificação g de D é *ótima* se o número de arestas de D que são inversões com relação a g é o menor possível. Sabe-se que a determinação de uma classificação ótima para D é um problema $\mathcal{NP}$-m-completo [57]. (Veja o Capítulo B.IV.) No entanto, no caso particular em que o grafo D é planar existe um algoritmo eficiente para encontrar uma classificação ótima para D [69]. Observamos ainda que a determinação de uma classificação consistente para um grafo acíclico também é conhecida como o "problema da classificação topológica" [60].

PARTE D

TEORIA DOS AUTÔMATOS FINITOS

CAPÍTULO I

RELAÇÕES, FUNÇÕES E MONÓIDES

Neste capítulo apresentamos as definições de alguns conceitos básicos, bem como algumas de suas propriedades. No texto omitimos a maioria das demonstrações. Em particular, as afirmações seguidas de um ponto de exclamação são enunciados de propriedades cuja demonstração fica a cargo do leitor.

1. Relações e funções

Dados conjuntos A e B, uma *relação* r de A para B é um subconjunto do produto cartesiano $A \times B$. Os conjuntos A e B são ditos *domínio* e *co-domínio* da relação r. Para um subconjunto S de A, definimos a *imagem* de S por r:

$$Sr = \{b \in B \mid (a, b) \in r \text{ para algum } a \in S\}.$$

Convencionamos nesta parte do livro denotar um subconjunto unitário $\{a\}$ de A pelo seu único elemento a. Assim, $ar = b$ significa que (a, b) é o único par ordenado em r cujo primeiro elemento é a. Note que

$$(a, b) \in r \quad \text{e} \quad b \in ar$$

são notações equivalentes. Além disso, uma relação r é completamente determinada pelos conjuntos ar, no seguinte sentido: Duas relações $r, s \subseteq A \times B$ são iguais sse para todo $a \in A$ $ar = as$!

A *inversa* da relação $r \subseteq A \times B$ é a relação $r^{-1} \subseteq B \times A$, definida por:

$$r^{-1} = \{(b, a) \in B \times A \mid (a, b) \in r\}.$$

Para toda relação $r \subseteq A \times B$,

$$(r^{-1})^{-1} = r!$$

A *composição* da relação $r \subseteq A \times B$ com a relação $s \subseteq B \times C$ é a relação $rs \subseteq A \times C$, definida por:

$$rs = \{(a, c) \in A \times C \mid (a, b) \in r \quad \text{e} \quad (b, c) \in s \text{ para algum } b \in B\}.$$

Dada outra relação $t \subseteq C \times D$, resulta que

$$(rs)t = r(st)!$$

Isto é, a composição de relações é uma operação associativa, não havendo necessidade de utilizar parêntesis ao escrevermos *rst*. Analogamente, para $S \subseteq A$, *Srs* é uma expressão bem definida, pois

$$(Sr)s = S(rs)!$$

No caso de *r* ser uma relação de *A* para *A*, diremos também que *r* é uma relação *sobre A*.

Funções são casos particulares muito importantes de relações. Assim, uma *função f* de *A* para *B*, escreve-se $f : A \to B$, é uma relação $f \subseteq A \times B$, tal que, para todo *a* em *A*, *af* é um subconjunto unitário de *B*. Neste caso, as fórmulas

$$(a, b) \in f,\ b \in af \text{ e } af = b$$

são todas equivalentes.

Dadas funções $f : A \to B$ e $g : B \to C$, a sua composição *fg* é uma função $fg : A \to C$! Observe que nesta parte do livro a imagem de *a* por *f* é denotada por *af* em vez da notação mais comum *f*(*a*). Além disso a função composta *fg* consiste em aplicar primeiro *f* e depois *g*, ao contrário da notação tradicional.

Uma função $f : A \to B$ é *injetora* se para todo $a_1, a_2 \in A$, $a_1 f = a_2 f$ implica que $a_1 = a_2$; ela é *sobrejetora* se $Af = B$; e é *bijetora* se ela for injetora e sobrejetora. Nestes casos, podemos dizer também que a função *f* é, respectivamente, uma *injeção*, *sobrejeção* ou *bijeção*.

A *relação identidade* 1_A sobre *A* é definida por

$$1_A = \{(a, a) \in A \times A \mid a \in A\},$$

e é uma função bijetora $1_A : A \to A$. Para toda relação $r \subseteq A \times A$,

$$r1_A = 1_A r = r!$$

Note que a inversa f^{-1} de uma função $f : A \to B$ é uma relação $f^{-1} \subseteq B \times A$ que em geral não é uma função!

A proposição seguinte caracteriza funções injetoras, sobrejetoras e bijetoras.

PROPOSIÇÃO 1. *Seja* $g : A \to B$ *uma função com domínio não vazio. Então,*

(i) *g é injetora sse existe uma função* $h : B \to A$, *tal que* $gh = 1_A$;

(ii) *g é sobrejetora sse existe uma função* $f : B \to A$, *tal que* $fg = 1_B$;
(iii) *g é bijetora sse a relação* g^{-1} *é uma função* $g^{-1} : B \to A$, *tal que* $gg^{-1} = 1_A$ e $g^{-1}g = 1_B$.

Demonstração. (i) Suponhamos que $h : B \to A$ seja uma função, tal que $gh = 1_A$. Sejam $a_1, a_2 \in A$, tais que $a_1g = a_2g$. Então $a_1gh = a_2gh$. Como $gh = 1_A$, resulta que $a_1gh = a_1$ e $a_2gh = a_2$; logo $a_1 = a_2$. Isto é, a função g é injetora.

Reciprocamente, suponhamos que a função g seja injetora. Consideramos inicialmente a relação $g^{-1} \subseteq B \times A$, e mostramos que para cada b em B, bg^{-1} é vazio ou é um subconjunto unitário de A. De fato, suponha que $a_1, a_2 \in bg^{-1}$, então $a_1g = a_2g$, logo $a_1 = a_2$ pois g é injetora. Como $A \neq \varnothing$, podemos escolher um elemento a_0 em A e definir uma função h como segue:

$$bh = \begin{cases} a_0 \text{ se } bg^{-1} = \varnothing \\ bg^{-1} \text{ caso contrário} \end{cases} \qquad (b \in B).$$

Resta demonstrar que $gh = 1_A$. De fato, seja $a \in A$, e seja $ag = b$. Então $a \in bg^{-1}$, e pela construção $bh = a$. Logo, $agh = bh = a$, resultando que $gh = 1_A$.

(ii) Suponhamos que $f : B \to A$ seja uma função, tal que $fg = 1_B$. Está claro que $Bf \subseteq A$, logo $Bfg \subseteq Ag$. Por outro lado $Bfg = B1_B = B$, isto é, $B \subseteq Ag$. Como $Ag \subseteq B$, resulta que $Ag = B$, isto é, g é sobrejetora.

Reciprocamente, suponhamos que a função g seja sobrejetora. Consideremos inicialmente a relação $g^{-1} \subseteq B \times A$. Para todo b em B, bg^{-1} é não vazio. De fato, como g é sobrejetora, $Ag = B$ e para todo b em B existe a em A, tal que $ag = b$, isto é $a \in bg^{-1}$. Nestas condições, para cada b em B, escolhemos um elemento a em bg^{-1} e colocamos $bf = a$. Assim, $bfg = ag = b$. Como $bfg = b$ para todo b em B, resulta que $fg = 1_B$.[1]

(iii) Suponhamos que a relação g^{-1} seja uma função $g^{-1} : B \to A$, tal que $gg^{-1} = 1_A$ e $g^{-1}g = 1_B$. Por (i) e (ii), g é injetora e sobrejetora, logo a função g é bijetora.

---

(1) Observe que nesta demonstração estamos usando (implicitamente) o axioma da escolha. Na verdade demonstra-se que a afirmação (ii) é equivalente a este axioma.

Reciprocamente, suponhamos que a função g seja bijetora. Por (i) e (ii), existem funções $f, h : B \to A$, tais que $gh = 1_A$ e $fg = 1_B$. Inicialmente mostramos que $f = h$. De fato,

$$f = f1_A = f(gh) = (fg)h = 1_B h = h.$$

Por outro lado, $fg = 1_B$ implica que $f \subseteq g^{-1}$. De fato, se $(b, a) \in f$, isto é, $bf = a$, então $bfg = ag$. Como $fg = 1_B$, resulta que $ag = b$, isto é $(b, a) \in g^{-1}$. Analogamente, $gh = 1_A$ implica que $g \subseteq h^{-1}$, logo $g^{-1} \subseteq (h^{-1})^{-1} = h$. Assim, $f \subseteq g^{-1} \subseteq h$; mas como $f = h$, resulta que $f = g^{-1} = h$. Portanto g^{-1} é uma função, $gg^{-1} = 1_A$ e $g^{-1}g = 1_B$. ■

Dadas funções $f : A \to B$ e $f' : A' \to B'$, dizemos que f é uma *restrição* de f' se $A \subseteq A'$, $B \subseteq B'$ e $f \subseteq f'$, isto é, para todo a em A $af = af'$. Neste caso dizemos também que f' é uma *extensão* de f.

Dado um conjunto A, uma relação $r \subseteq A \times A$ é de *equivalência* sobre A sse

$$1_A \subseteq r, \quad r \subseteq r^{-1} \quad \text{e} \quad rr \subseteq r,$$

isto é, r é *reflexiva*, *simétrica* e *transitiva*, respectivamente. Escrevendo $a \sim b$ ao invés de $(a, b) \in r$, as propriedades acima tomam sua forma habitual! É bem conhecido que para $a_1, a_2 \in A$, $a_1 r$ e $a_2 r$ são conjuntos não vazios que são disjuntos ou são iguais! Eles são iguais sse $(a_1, a_2) \in r$! Para $a \in A$, ar é a *classe de equivalência* contendo a, e o *conjunto quociente* de A por r é

$$A/r = \{ar \mid a \in A\}.$$

A *projeção canônica* de A em A/r é a função

$$p : A \to A/r$$

que associa a cada elemento a de A a classe de equivalência ar. A projeção canônica é sobrejetora e $pp^{-1} = r$!

Dada uma função $f : A \to B$, a relação $ff^{-1} \subseteq A \times A$ é uma relação de equivalência sobre A! Observe que para a_1, a_2 em A, $(a_1, a_2) \in ff^{-1}$ sse $a_1 f = a_2 f$! Assim, para cada a em A, a classe de equivalência aff^{-1} que contém a é dada por:

$$aff^{-1} = \{b \in A \mid bf = af\}!$$

PROPOSIÇÃO 2. *Seja $g : A \to B$ uma função sobrejetora e seja $f : A \to C$ uma função. Se $gg^{-1} \subseteq ff^{-1}$ então existe*

uma única função $h : B \to C$, *tal que* $f = gh$. *Ademais, h é sobrejetora sse f é sobrejetora; e h é injetora sse* $gg^{-1} = ff^{-1}$.

As funções f, g e h no enunciado da proposição acima podem ser representadas como no diagrama da Figura 1. Costuma-se dizer que este diagrama é comutativo para expressar o fato de que $f = gh$.

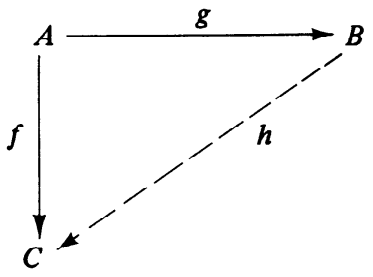


Figura 1 — Diagrama para a Proposição 2.

Demonstração. Se A for vazio, então B é vazio também, pois a função g é sobrejetora. Assim, $f = g = \varnothing$, e a função $h = \varnothing$ satisfaz trivialmente a proposição! Caso contrário, A, B e C são todos não-vazios! Pela Proposição 1 (ii), existe uma função $g_e : B \to A$, tal que

$$g_e g = 1_B. \tag{1}$$

Colocamos então

$$h = g_e f. \tag{2}$$

Para provar que $f = gh$, observamos inicialmente que $g = g1_B = gg_e g$. Seja a em A, então $ag = agg_e g$, logo $(a, agg_e) \in gg^{-1}$. Como $gg^{-1} \subseteq ff^{-1}$ por hipótese, resulta que $(a, agg_e) \in ff^{-1}$, isto é, $af = agg_e f$. Como o raciocínio acima vale para todo a em A, concluimos que $f = gg_e f$. Finalmente, como $g_e f = h$, resulta que

$$f = gh. \tag{3}$$

Para provar a unicidade de h, suponhamos que $h' : B \to C$ seja outra função tal que $f = gh'$. Usando (2) e (1), vem que $h = g_e f = g_e gh' = h'$, isto é $h = h'$.

Suponhamos agora que h seja sobrejetora. Pela Proposição 1 (ii), existe uma função $h_e : C \to B$, tal que $h_e h = 1_C$. Usando (1) e (3), vem que $1_C = h_e h = h_e 1_B h = h_e g_e gh = h_e g_e f$, isto é, $(h_e g_e)f = 1_C$. A função f é sobrejetora pela Proposição 1 (ii). Reciprocamente, suponhamos que f é sobrejetora. Pela Proposição 1 (ii), existe uma função

$f_e : C \to A$, tal que $f_e f = 1_C$. Resulta de (3) que $(f_e g)h = 1_C$, logo a função h é sobrejetora pela Proposição 1 (ii).

Suponhamos finalmente que h seja injetora. Como $gg^{-1} \subseteq ff^{-1}$, basta demonstrar que $ff^{-1} \subseteq gg^{-1}$. De fato, seja $(a_1, a_2) \in ff^{-1}$, isto é, $a_1 f = a_2 f$. De (3), $f = gh$, logo $a_1 gh = a_2 gh$. Como h é injetora, resulta que $a_1 g = a_2 g$, isto é, $(a_1, a_2) \in gg^{-1}$. Reciprocamente, suponhamos que $gg^{-1} = ff^{-1}$. Para demonstrar que h é injetora, sejam b_1 e b_2 em B, tais que $b_1 h = b_2 h$; basta provar então que $b_1 = b_2$. De fato, como $h = g_e f$ por (2), resulta que $b_1 g_e f = b_2 g_e f$, isto é, $(b_1 g_e, b_2 g_e) \in ff^{-1}$. Segue da hipótese que $(b_1 g_e, b_2 g_e) \in gg^{-1}$, isto é $b_1 g_e g = b_2 g_e g$. Porém, $g_e g = 1_B$, por (1), logo $b_1 = b_2$. ■

Talvez seja interessante examinar a Proposição 2 de outro ângulo. Vimos que gg^{-1} e ff^{-1} são relações de equivalência sobre o conjunto A. Seja agora b um elemento de B. Como g é sobrejetora, bg^{-1} é uma classe de equivalência de gg^{-1}. A condição $gg^{-1} \subseteq ff^{-1}$ garante que cada classe de equivalência de gg^{-1} está contida numa classe de equivalência de ff^{-1}. Logo, existe um único c em C, tal que $bg^{-1} \subseteq cf^{-1}$ Nestas condições $bh = c$.

Descrevemos a seguir uma aplicação importante da Proposição 2. Dada uma função $f : A \to B$, vimos que ff^{-1} é uma relação de equivalência sobre A. Seja $p : A \to A/ff^{-1}$ a projeção canônica de ff^{-1}; então p é sobrejetora e $pp^{-1} = ff^{-1}$. Segue da Proposição 2 que existe uma única função $h : A/ff^{-1} \to B$, tal que $f = ph$. Ademais, pela mesma proposição, h é injetora. Observe que h é bijetora sse f é sobrejetora.

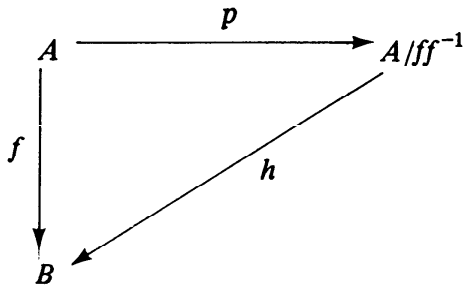


Figura 2 — Funções e relações de equivalência.

Terminamos esta seção com duas notações relativas à teoria dos conjuntos. O *conjunto potência* de um conjunto A será denotado por $\mathfrak{p}(A)$, isto é

$$\mathfrak{p}(A) = \{B \mid B \subseteq A\}.$$

A *cardinalidade* de um conjunto finito A será denotada (nesta parte do livro) por *card* A.

2. Monóides

Um *monóide* consiste de um conjunto M, munido de uma operação binária associativa (escrita multiplicativamente), e que contém um *elemento identidade* $1 \in M$. Isto é, para todo $m_1, m_2, m_3 \in M$,

$$(m_1m_2)m_3 = m_1(m_2m_3) \quad \text{e} \quad m_1 1 = 1m_1 = m_1.$$

Assim, dado um conjunto A, o conjunto das relações (funções) de A para A, com a operação de composição, é um monóide cuja identidade é 1_A, e que será denotado por *Rel* A (*Fun* A). Analogamente, $\mathfrak{p}(A)$ com a operação de união é um monóide cuja identidade é o conjunto vazio $\varnothing$.

Dados subconjuntos A e B de um monóide M, definimos o seu *produto* por:

$$AB = \{m_1m_2 \in M \mid m_1 \in A \quad \text{e} \quad m_2 \in B\}.$$

Com esta multiplicação de subconjuntos, $\mathfrak{p}(M)$ transforma-se num monóide, com identidade 1. Note que para subconjuntos de M, o produto distribui sobre a união, isto é, para $A, B, C \subseteq M$,

$$A(B \cup C) = AB \cup AC \quad \text{e} \quad (A \cup B)C = AC \cup BC!$$

Para um natural n e um subconjunto A de M, definimos A^n indutivamente, por:

$$A^0 = 1 \quad \text{e} \quad A^{n+1} = A^nA.$$

Um subconjunto T de M é um *submonóide* de M sse

$$1 \in T \quad \text{e} \quad T^2 \subseteq T.$$

Em outras palavras, T é um submonóide de M sse T contém a identidade de M e é fechado sob a operação binária de M.

Dado um subconjunto A do monóide M, consideremos a intersecção A^* de todos os submonóides de M que contêm A, isto é,

$$A^* = \cap T,$$

onde a intersecção se estende sobre todos os submonóides T de M, tais que $A \subseteq T$. Observe que a intersecção de submonóides de M é um submonóide de M! Assim, A^* é um submonóide de M. De fato,

A^* é o único submonóide minimal de M que contém A! Por isso, costuma-se dizer que A^* é o *menor submonóide de M que contém A*, ou que A^* é o *submonóide de M gerado por A*. Ocasionalmente diremos também que A^* é a *estrela* de A. Demonstra-se que

$$A^* = 1 \cup A \cup A^2 \cup \ldots \cup A^n \cup \ldots !$$

Em outras palavras, A^* consiste exatamente daqueles elementos de M que podem ser expressos como o produto de um número finito de fatores que pertencem a A. Note também que

$$A^* = 1 \cup AA^* = 1 \cup A^*A !$$

Dados monóides M e M', com identidades 1 e 1', uma função $f : M \to M'$ é um *morfismo* sse

$$1f = 1' \text{ e } (m_1m_2)f = (m_1f)(m_2f) \text{ para todo } m_1, m_2 \in M.$$

A composição de morfismos é um morfismo! Um *monoforfismo*, *epimorfismo*, ou *isomorfismo* é, respectivamente, um morfismo injetor, sobrejetor, ou bijetor.

Seja M um monóide e r uma relação de equivalência sobre M. Dizemos que r é uma *relação de congruência* sobre M sse para todo m_1 e m_2 em M

$$(m_1r)(m_2r) \subseteq (m_1m_2)r,$$

isto é, o produto de classes de equivalência (como subconjuntos de M) está contido em uma outra classe de equivalência. Escrevendo $m_1 \sim m_2$, ao invés de $(m_1, m_2) \in r$, é fácil demonstrar que as afirmações seguintes são equivalentes:

(i) r é uma relação de congruência,
(ii) para todo $m, m', m'' \in M$, $m' \sim m''$ implica que $m'm \sim m''m$ e $mm' \sim mm''$,
(iii) para todo $m_1, m_1', m_2, m_2' \in M$, $m_1 \sim m_1'$ e $m_2 \sim m_2'$ implica que $m_1m_2 \sim m_1'm_2'$!

Se r for uma relação de congruência sobre M, podemos definir um produto (indicador por $\cdot$) no conjunto quociente M/r:

$$(m_1r) \cdot (m_2r) = (m_1m_2)r.$$

O conjunto M/r munido deste produto é um monóide com identidade $1r$! Assim, chamamos M/r de *monóide quociente* de M por r. A projeção canônica $p : M \to M/r$ é um epimorfismo neste caso!

Para monóides, a Proposição 2 toma a seguinte forma:

PROPOSIÇÃO 3. *Sejam M_1, M_2 e M_3 monóides, $g : M_1 \to M_2$ um epimorfismo e $f : M_1 \to M_3$ um morfismo. Se $gg^{-1} \subseteq ff^{-1}$ então existe um único morfismo $h : M_2 \to M_3$, tal que $f = gh$. Ademais, h é um epimorfismo sse f é um epimorfismo; e h é um monomorfismo sse $gg^{-1} = ff^{-1}$.*

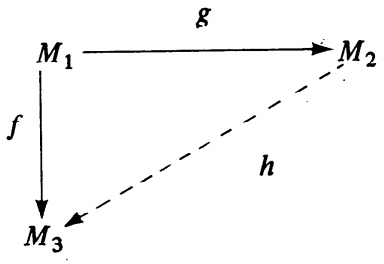


Figura 3 — Morfismos na Proposição 3.

Demonstração. Tendo em vista a Proposição 2, é suficiente mostrar que a função h dada por aquela proposição é um morfismo. De fato, sejam m_1, m_2 em M_2. Como g é sobrejetora, existem m'_1, m'_2 em M_1, tais que $m'_i g = m_i$ $(i = 1, 2)$. Como f é um morfismo, $(m'_1 m'_2)f = (m'_1 f)(m'_2 f)$; logo $(m'_1 m'_2)gh = (m'_1 gh)(m'_2 gh)$, já que $f = gh$. Agora, g é um morfismo, logo $[(m'_1 g)(m'_2 g)]h = (m'_1 gh)(m'_2 gh)$. Substituindo $m'_i g$ por m_i $(i = 1, 2)$, obtemos que $(m_1 m_2)h = (m_1 h)(m_2 h)$. Finalmente, denotando por 1_i a identidade de M_i $(i = 1, 2, 3)$, temos que $1_3 = 1_1 f = 1_1 gh = 1_2 h$. Assim, h é um morfismo. ■

Dado um morfismo $f : M_1 \to M_2$, a relação de equivalência ff^{-1} sobre M_1 é uma congruência! Segue da Proposição 3 que existe um monomorfismo $h : M_1/ff^{-1} \to M_2$ tal que $f = ph$, onde p é a projeção canônica $p : M_1 \to M_1/ff^{-1}$! (Vide Figura 4.) O monomorfismo h é um isomorfismo sse f é um epimorfismo!

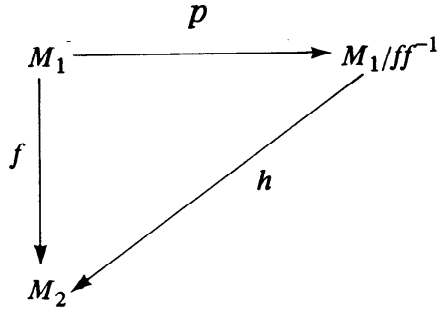


Figura 4 — Morfismos e relações de congruência.

Certos monóides, chamados de monóides livres, tem um papel importante nos capítulos seguintes. Passamos a descrevê-los.

Dado um conjunto Σ, consideremos o conjunto S de seqüências finitas de elementos de Σ:

$$s = (\sigma_1, \sigma_2, \ldots, \sigma_n) \quad (n \geq 0).$$

Definimos um produto em S por *concatenação*, isto é, se

$$t = (\tau_1, \tau_2, \ldots, \tau_m)$$

for outro elemento de S, o produto st de s e t é:

$$st = (\sigma_1, \sigma_2, \ldots, \sigma_n, \tau_1, \tau_2, \ldots, \tau_m).$$

A concatenação é associativa, logo S, munido deste produto, é um monóide cuja identidade é a seqüência vazia $1 = (\ )$.

Convencionamos denotar a seqüência unitária (σ) de S por σ. Desta forma, utilizando o produto já definido, o elemento s acima de S pode ser escrito (se $n > 0$):

$$s = \sigma_1\sigma_2 \ldots \sigma_n.$$

Devido a esta notação, chamamos Σ de *alfabeto*, seus elementos de *letras* e os elementos de S de *palavras*. Por outro lado, usando a convenção acima, podemos considerar Σ como um subconjunto de S. Note que o submonóide de S gerado por Σ é o próprio S, isto é, $S = \Sigma^*$. Assim, chamamos S de *monóide livre* gerado por Σ e o denotamos por Σ^*.

Dada a palavra s acima em Σ^*, o natural n é o seu *comprimento* e é denotado por $|s|$. Resulta que

$$|1| = 0 \text{ e } |st| = |s| + |t| \text{ para todo } s, t \text{ em } \Sigma^*.$$

Sejam s, t_1, t_2 e t_3 em Σ^*, tais que $s = t_1t_2t_3$. Dizemos que t_2 é um *segmento* de s e que t_1 (t_3) é um *segmento inicial* (*segmento final*) de s.

A propriedade fundamental dos monóides livres é dada por:

PROPOSIÇÃO 4. *Seja M um monóide e seja $f : \Sigma \to M$ uma função. Existe uma única extensão de f a um morfismo $f^* : \Sigma^* \to M$.*

Demonstração. Definimos a função f^* como segue:

$$1f^* = 1,$$

e para uma palavra $\sigma_1\sigma_2 \ldots \sigma_n$ de Σ^*, $n \geq 1$, com σ_i em Σ $(1 \leq i \leq n)$, colocamos

$$(\sigma_1\sigma_2 \ldots \sigma_n)f^* = (\sigma_1 f)(\sigma_2 f) \ldots (\sigma_n f).$$

É imediato verificar que f^* é uma extensão de f e que dadas palavras s e t em Σ^*, $(st)f^* = (sf^*)(tf^*)$. Logo, f^* é um morfismo.

Suponhamos agora que $g : \Sigma^* \to M$ seja uma outra extensão de f a um morfismo. Mostramos que $sg = sf^*$ para todo s em Σ^*. Para tanto procedemos por indução no comprimento de palavras em Σ^*. Se $|s| = 0$, então $s = 1$, logo $1g = 1$ pois g é um morfismo. Assim, $1g = 1f^*$. Se $|s| > 0$, então existem t em Σ^* e σ em Σ, tais que $s = t\sigma$. Como g é um morfismo, obtemos $sg = (t\sigma)g = (tg)(\sigma g)$. Como $|t| < |s|$, vem pela hipótese da indução que $tg = tf^*$. Por outro lado, $\sigma g = \sigma f = \sigma f^*$, pois g e f^* são ambas extensões de f. Assim, $(tg)(\sigma g) = (tf^*)(\sigma f^*) = (t\sigma)f^* = sf^*$, pois f^* é um morfismo. Segue que $sg = sf^*$. Logo, $g = f^*$. ■

EXERCÍCIOS

1. Prove as afirmações do texto que são seguidas de um ponto de exclamação.
2. Enumere todas as relações de A para A, onde A é o conjunto $\{a, b\}$ de dois elementos. Indique quais são funções, funções injetoras, funções sobrejetoras e funções bijetoras. Indique também o inverso de cada uma das relações.
3. Enumere todas as funções de A para A, onde A é o conjunto $\{a, b, c\}$ de três elementos. Indique as funções bijetoras e o inverso de cada uma destas.
4. Enumere todas as relações de equivalência sobre o conjunto $\{a, b, c\}$ de três elementos.
5. Em cada caso prove ou desprove a afirmação dada:
   (i) Para toda relação $r \subseteq A \times B$, $rr^{-1} \subseteq 1_A$.
   (ii) Para toda relação $r \subseteq A \times B$, $1_A \subseteq rr^{-1}$.
6. Sejam r_1 e r_2 relações de A para B, s_1 e s_2 relações de B para C e P_1 e P_2 subconjuntos de A. Prove que:
   (i) Se $P_1 \subseteq P_2$ e $r_1 \subseteq r_2$ então $P_1r_1 \subseteq P_2r_2$.
   (ii) Se $r_1 \subseteq r_2$ e $s_1 \subseteq s_2$ então $r_1s_1 \subseteq r_2s_2$.
   (iii) $(P_1 \cup P_2)r_1 = P_1r_1 \cup P_2r_1$.

(iv) $(P_1 \cap P_2)r_1 \subseteq P_1r_1 \cap P_2r_1$.
(v) $P_1(r_1 \cup r_2) = P_1r_1 \cup P_1r_2$.
(vi) $P_1(r_1 \cap r_2) \subseteq P_1r_1 \cap P_1r_2$.
(vii) $r_1(s_1 \cup s_2) = r_1s_1 \cup r_1s_2$.
(viii) $(r_1 \cup r_2)s_1 = r_1s_1 \cup r_2s_1$.
(ix) $r_1(s_1 \cap s_2) \subseteq r_1s_1 \cap r_1s_2$.
(x) $(r_1 \cap r_2)s_1 \subseteq r_1s_1 \cap r_2s_1$.
(xi) se $r_1 \subseteq r_2$ então $r_1^{-1} \subseteq r_2^{-1}$.
(xii) $(r_1 \cup r_2)^{-1} = r_1^{-1} \cup r_2^{-1}$.
(xiii) $(r_1 \cap r_2)^{-1} = r_1^{-1} \cap r_2^{-1}$.

7. Prove que as inclusões nos pontos (iv), (vi), (ix) e (x) do exercício anterior não podem, em geral, ser substituídas por igualdades.

8. Sejam $r \subseteq A \times B$ e $s \subseteq B \times C$ relações. Mostre que $(rs)^{-1} = s^{-1}r^{-1}$.

9. Seja $f : A \to B$ uma função. Mostre que se $A = \varnothing$, então $f = \varnothing$ e f é uma função injetora. Neste caso f é sobrejetora sse $B = \varnothing$. Prove que se $B = \varnothing$, então $A = \varnothing$ e f é uma função bijetora.

10. Prove que uma relação $r \subseteq A \times B$ é uma função sse $r^{-1}r \subseteq 1_B$ e $1_A \subseteq rr^{-1}$.

11. Seja $r \subseteq A \times B$ uma relação, onde $B \neq \varnothing$. Prove que as afirmações abaixo são equivalentes:
(i) r é uma função,
(ii) r é maximal com relação à propriedade $r^{-1}r \subseteq 1_B$,
(iii) r é minimal com relação à propriedade $1_A \subseteq rr^{-1}$.

12. Seja $r \subseteq A \times B$ uma relação, onde $B \neq \varnothing$. Prove que:
(i) existe uma função $f : A \to B$, tal que $r \subseteq f$ sse $r^{-1}r \subseteq 1_B$,
(ii) existe uma função $g : A \to B$, tal que $g \subseteq r$ sse $1_B \subseteq rr^{-1}$.

13. Seja $f : A \to B$ uma função. Prove que f é injetora sse $ff^{-1} = 1_A$, e que f é sobrejetora sse $1_B = f^{-1}f$.

14. Sejam $f : A \to B$ e $g : B \to C$ funções. Prove que:
(i) Se f e g forem injetoras então fg é injetora.
(ii) Se f e g forem sobrejetoras então fg é sobrejetora.
(iii) Se fg for injetora então f é injetora.
(iv) Se fg for sobrejetora então g é sobrejetora.

15. Dê exemplos de funções $f : A \to B$ e $g : B \to C$, tais que f é injetora, g é sobrejetora mas fg não é nem injetora nem sobrejetora.

16. Mostre que para toda função $f : A \to B$, $ff^{-1}f = f$.

17. Dada uma função $f : A \to B$, com $A \neq \varnothing$, construa uma outra função $g : B \to A$, tal que $fgf = f$.

18. Sejam $f : A \to B$, $g : B \to C$ e $h : C \to D$ funções e suponha que f e h são bijetoras. Mostre que g é injetora (sobrejetora, bijetora) sse fgh é injetora (sobrejetora, bijetora).

19. Seja $f : A \to B$ uma função. Prove que:
    (i) f é injetora sse para todo subconjunto X e Y de A $(X \cap Y)f = Xf \cap Yf$.
    (ii) f é injetora sse para todo subconjunto X de A $(A \backslash X)f \subseteq B \backslash (Xf)$.
    (iii) f é sobrejetora sse para todo subconjunto X de A $B \backslash (Xf) \subseteq (A \backslash X)f$.

20. Seja $f : A \to B$ uma função, e sejam X e Y subconjuntos de B. Prove que:
    (i) $(X \cup Y)f^{-1} = Xf^{-1} \cup Yf^{-1}$.
    (ii) $(X \cap Y)f^{-1} = Xf^{-1} \cap Yf^{-1}$.
    (iii) $(X \backslash Y)f^{-1} = Xf^{-1} \backslash Yf^{-1}$.

21. Sejam $f \subseteq A \times B$ e $g \subseteq B \times A$ relações, tais que $fg = 1_A$ e $gf = 1_B$. Prove que f e g são funções bijetoras e $f = g^{-1}$.

22. Sejam A e B conjuntos, A não vazio. Mostre que existe uma injeção $f : A \to B$ sse existe uma sobrejeção $g : B \to A$.

23. Seja $g : B \to C$ uma função. Mostre que g é injetora sse para todo conjunto A e funções $f_1, f_2 : A \to B$, se $f_1 g = f_2 g$ então $f_1 = f_2$. Enuncie e prove a caracterização análoga de funções sobrejetoras.

24. Sejam r e s relações simétricas sobre um conjunto A. Prove que se $rs \subseteq sr$, então $rs = sr$.

25. Ache duas relações de equivalência r e s sobre um conjunto A, tais que a relação rs não seja de equivalência.

26. Dê exemplos de relações r, s e t sobre um conjunto A, tais que:
    (i) r é reflexiva e simétrica mas não é transitiva,
    (ii) s é reflexiva e transitiva mas não é simétrica,
    (iii) t é simétrica e transitiva mas não é reflexiva.

27. Mostre que no enunciado da Proposição 2 a hipótese $gg^{-1} \subseteq ff^{-1}$ é necessária, isto é, ache funções $g : A \to B$ e $f : A \to C$, g sobrejetora, tais que não exista função $h : B \to C$, para qual $f = gh$.

28. Seja $f : A \to B$ uma função. Mostre que existem um conjunto C e funções $g : A \to C$ e $h : C \to B$, tais que $f = gh$, g é sobrejetora e h é injetora. Em outras palavras, toda função $f : A \to B$ admite uma fatoração

$$A \xrightarrow{g} C \xrightarrow{h} B,$$

com g sobrejetora e h injetora. Mostre que esta fatoração e "única" no seguinte sentido: se

$$A \xrightarrow{g'} C' \xrightarrow{h'} B$$

for uma outra fatoração de f, com g' sobrejetora e h' injetora, então existe uma (única) bijeção $b : C \to C'$ tal que $gb = g'$ e $bh' = h$.
Sugestão: Coloque $C = A/ff^{-1}$ e use a Proposição 2.

29. Mostre que toda função $f : A \to B$ admite uma fatoração

$$A \xrightarrow{g} C \xrightarrow{h} B,$$

com g injetora e h sobrejetora. Mostre que esta fatoração não é "única" no sentido do exercício anterior.
Sugestão: Se A e B forem disjuntos, coloque $C = A \cup B$.

30. Seja A um conjunto e sejam a e b elementos de A. A transposição de a com b é a função $t : A \to A$, definida por:

$$xt = \begin{cases} x \text{ se } x \neq a \text{ e } x \neq b \\ b \text{ se } x = a \\ a \text{ se } x = b \end{cases} \qquad (x \in A).$$

Prove que $tt = 1_A$ e conclua que t é uma bijeção.

31. (*Princípio da casa do pombo*, vide Capítulo C.V) – Se n for um número natural, $\mathbf{n}$ denotará o conjunto

$$\mathbf{n} = \{0, 1, \ldots, n-1\}.$$

Observe que $\mathbf{0} = \varnothing$ e que $n \leq m$ sse $\mathbf{n} \subseteq \mathbf{m}$. Sejam n e m naturais, e $f : \mathbf{n} \to \mathbf{m}$ uma função. Prove que se f for injetora, então $n \leq m$.
Sugestão: Use indução sobre m. Seja $k = (n-1)f$ e seja $t : \mathbf{m} \to \mathbf{m}$ a transposição de k com $m-1$. A função $g = ft$ é injetora. Note que $(n-1)g = m-1$ e aplique a hipótese da indução.

32. Sejam n e m naturais e $f : \mathbf{n} \to \mathbf{m}$ uma função. Prove que:
(i) se f for bijetora então $n = m$.
(ii) se $n = m$ então f é injetora sse f é sobrejetora.

33. Um conjunto A é *finito* se existir um número natural n e uma bijeção $f : \mathbf{n} \to A$. Mostre que se A for um conjunto finito então existe um único número natural n para qual existem bijeções $f : \mathbf{n} \to A$. Este número é chamado de *cardinalidade* de A e é denotado por *card* A.

34. Seja A um conjunto finito e $f : A \to A$ uma função. Mostre que f é injetora sse f é sobrejetora.

35. Um conjunto é *infinito* se ele não for finito. Mostre que o conjunto $\mathbb{N}$ dos números naturais não é finito. Usando o axioma da escolha mostre que se A for um conjunto infinito, então existe uma injeção $f : \mathbb{N} \to A$. Prove que um conjunto A é infinito sse existe uma injeção $f : A \to A$ que não é sobrejetora.

36. Dados conjuntos A e B, o conjunto de todas as funções com domínio B e co-domínio A é denotado por A^B, isto é

$$A^B = \{f \mid f : B \to A\}.$$

Ache bijeções entre os seguintes pares de conjuntos, onde A, B e C são conjuntos dados:

(i) $A \times A$ e A^2.
(ii) $\mathfrak{p}(A)$ e $\mathbf{2}^A$.
(iii) $(A \times B)^C$ e $A^C \times B^C$.
(iv) $(A^B)^C$ e $A^{B \times C}$.

37. Uma *permutação* de um conjunto A é uma bijeção $f : A \to A$. Denotamos por *Per* A o conjunto de todas as permutações de A. Para um conjunto A e um natural p, $\binom{A}{p}$ denota o conjunto de todos os subconjuntos de cardinalidade p de A.
Sejam p, n e m naturais, tais que $m = n + 1$. Ache bijeções entre os seguintes pares de conjuntos:

(i) $\mathbf{p}^m$ e $\mathbf{p} \times \mathbf{p}^n$.
(ii) *Per* $\mathbf{m}$ e $\mathbf{m} \times$ *Per* $\mathbf{n}$.
(iii) $\binom{\mathbf{m}}{p}$ e $\binom{\mathbf{n}}{p} \cup \binom{\mathbf{n}}{p-1}$, quando $p \geq 1$.

38. Sejam A e B conjuntos finitos e seja p um natural. Usando os Exercícios 36 e 37, mostre que:

(i) *card* $A^B = (\textit{card } A)^{\textit{card } B}$.
(ii) *card* $\mathfrak{p}(A) = 2^{\textit{card } A}$.
(iii) *card* (*Per* A) = (*card* A)!.

(iv) se $p \leq card\ A$ então $card\binom{A}{p} = \binom{card\ A}{p}$, onde, para naturais $n \leq m$, $\binom{m}{n}$ denota o coeficiente binomial dado por:

$$\binom{m}{n} = \frac{m!}{n!\,(m-n)!}.$$

39. (*Hall*) — Sejam A e B conjuntos finitos. Mostre que uma relação $r \subseteq A \times B$ contém uma função injetora sse para todo subconjunto S de A $card\ Sr \geq card\ S$.
Sugestão: Use o Teorema de Hall, demonstrado no Capítulo C. III.

40. (*Banach*) — Dados conjuntos A e B, bem como funções $f : A \to B$ e $g : B \to A$, mostre que existem conjuntos S, T, X e Y tais que:

$A = S \cup T$, $S \cap T = \varnothing$, $B = X \cup Y$, $X \cap Y = \varnothing$,
$Sf = X$ e $Yg = T$.

Sugestão: Defina uma função $h : \mathfrak{p}(A) \to \mathfrak{p}(A)$, colocando, para $V \subseteq A$, $Vh = A\backslash(B\backslash Vf)g$. Seja $\mathscr{F} = \{V \subseteq A \mid Vh \subseteq V\}$. A coleção $\mathscr{F}$ não é vazia, pois $A \in \mathscr{F}$. Seja $S = \bigcap_{V \in \mathscr{F}} V$. Se $V \subseteq W \subseteq A$, então $Vh \subseteq Wh$. Segue que $\mathscr{F}$ é fechada sob intersecção, bem como, se $V \in \mathscr{F}$ então $Vh \in \mathscr{F}$. Assim, tanto S como Sh pertencem a $\mathscr{F}$. Segue que $Sh = S$. Agora basta colocar $X = Sf$, $T = A\backslash S$ e $Y = B\backslash X$.

41. (*Schröder-Bernstein*) — Sejam $f : A \to B$ e $g : B \to A$ funções injetoras. Mostre que existe uma bijeção $h : A \to B$.
Sugestão: Use o Exercício 40. Alternativamente, defina $S_0 = A\backslash Bg$ e $S_{n+1} = S_n fg$, para todo $n \geq 0$. Sejam $S = \bigcup_{n \in \mathbb{N}} S_n$ e $X = Sf$. Prove que $(B\backslash X)g = A\backslash S$.

42. Seja $f : M_1 \to M_2$ um morfismo de monóides. Mostre que se S for um submonóide de M_1 então Sf é um submonóide de M_2. Mostre também que se T for um submonóide de M_2 então Tf^{-1} é um submonóide de M_1.

43. Monóides M_1 e M_2 são *isomorfos*, escreve-se $M_1 \simeq M_2$, se existir um isomorfismo de M_1 para M_2. Seja A um conjunto, e seja M_1 (M_2) o monóide que consiste do conjunto $\mathfrak{p}(A)$, munido da operação de união (intersecção). Mostre que $M_1 \simeq M_2$.

44. O conjunto $\mathbb{N}$ dos números naturais, munido da operação de soma é um monóide. Este monóide será também denotado por $\mathbb{N}$. Seja Σ um conjunto não vazio. Prove que a função comprimento $|| : \Sigma^* \to \mathbb{N}$ é um epimorfismo que é injetor sse Σ é unitário. Conclua que $\mathbb{N}$ é isomorfo a um monóide livre gerado por um único elemento.

45. Seja M um monóide e a um elemento de M. Mostre que existe um único morfismo $f : \mathbb{N} \to M$, tal que $1f = a$.

46. Um elemento z de um monóide M é um *zero à esquerda* (*à direita*) se para todo m em M $zm = z$ ($mz = z$). Um elemento z de M é um *zero* se ele for tanto um zero à esquerda como um zero à direita, isto é, para todo m em M, $mz = zm = z$. Mostre que um monoide tem no máximo um zero. Mostre também que se z_e for um zero à esquerda em M e z_d for um zero à direita em M, então $z_e = z_d$, e este elemento é o único zero de M.

47. Mostre que todo monóide M satisfaz exatamente uma das quatro afirmações seguintes:
    (i) M não tem zeros à direita nem zeros à esquerda.
    (ii) M não tem zeros à direita mas tem um ou mais zeros à esquerda.
    (iii) M tem um ou mais zeros à direita mas não tem zeros à esquerda.
    (iv) M tem um único zero e não tem outros zeros à esquerda nem à direita.

    Dê exemplos de monóides satisfazendo cada uma destas quatro afirmações.

48. Seja A um conjunto com pelo menos dois elementos. Classifique os monóides *Rel A* e *Fun A* de acordo com as afirmações do Exercício 47. Caracterize os zeros de cada tipo nos dois casos.

49. Sejam M_1 e M_2 monóides. O *produto direto* de M_1 e M_2 consiste do conjunto $M_1 \times M_2$, munido da seguinte operação: para todo $m_1, m_1' \in M_1$ e $m_2, m_2' \in M_2$ $(m_1, m_2)(m_1', m_2') = (m_1 m_1', m_2 m_2')$. Mostre que
    (i) $M_1 \times M_2$ é um monóide.
    (ii) $M_1 \times M_2 \simeq M_2 \times M_1$.
    (iii) Existem epimorfismos $p_i : M_1 \times M_2 \to M_i$ para $i = 1, 2$.
    (iv) Se M_1 e M_2 forem, respectivamente, dos tipos (ii) e (iii), de acordo com a classificação do Exercício 47, então $M_1 \times M_2$ é do tipo (i).

50. Um *grupo* é um monóide M, em que para cada a em M existe b em M, tal que $ab = ba = 1$. Mostre que para todo conjunto A, *Per* A é um grupo. (Veja Exercício 37.)

51. Mostre que um monóide M é um grupo sse para todo a em M, $aM = Ma = M$.

52. Um elemento e de um monóide M é *idempotente* se $e^2 = e$. Seja $f : M_1 \to M_2$ um morfismo de monóides. Mostre que se e for idempotente em M_1 então ef é um idempotente em M_2. Mostre por meio de um exemplo que a recíproca não é verdadeira.

53. Mostre que uma função f em *Fun* A é idempotente sse a restrição $g : Af \to Af$ de f a Af é a função identidade 1_{Af}.

54. Um subconjunto T de um monóide M é um *monóide em* M sse $T^2 \subseteq T$ e T contém um elemento e, tal que para todo t em T, $et = te = t$. Note que um monóide T em M é um submonóide de M sse $1 \in T$. Mostre que se e for um idempotente em M, então eMe é um monóide em M, enquanto eM ou Me podem não ser monóides em M. Porém, $eMe = eM \cap Me$.

55. Mostre que um grupo tem um único elemento idempotente.

56. Seja r uma relação em *Rel* A, e seja t dado por:

$$t = 1_A \cup r \cup r^2 \cup \ldots \cup r^n \cup \ldots .$$

Mostre que $(a, b) \in t$ sse $a = b$, ou existem um natural $n \geq 1$ e $c_0, c_1, \ldots, c_n \in A$, tais que $a = c_0$, $c_n = b$ e $(c_i, c_{i+1}) \in r$, para $i = 0, 1, \ldots, n-1$. Mostre que t é a única relação minimal reflexiva e transitiva que contém r. A relação t é chamada de *fecho reflexivo e transitivo* de r. (Confronte com o Exercício B.III.25.) Observação: Note que t e r^* são entidades distintas! De fato, enquanto t é um elemento de *Rel* A, r^* (isto é $\{r\}^*$) é um subconjunto de *Rel* A. No entanto, $t = \bigcup_{s \in r^*} s$.

57. Seja M um monóide. Mostre que a intersecção de relações de congruência sobre M é uma relação de congruência sobre M.

58. Mostre que dada uma relação r sobre um monóide M, existe uma única relação de congruência minimal sobre M que contém r. Esta relação é chamada de *relação de congruência gerada por* r.

59. Nas condições do exercício anterior, seja r_c a relação de congruência gerada por r. Seja s a relação sobre M dada por:

$(m, m') \in s$ sse existem $u, v, x, y \in M$, tais que $m = uxv$, $m' = uyv$ e $(x, y) \in r \cup r^{-1}$. Seja t o fecho reflexivo e transitivo de s (vide Exercício 56). Mostre que $t = r_c$.

Sugestão: Mostre que t é uma relação de congruência que contêm r, logo $r_c \subseteq t$. Mostre que $s \subseteq r_c$, pois r_c é uma relação de congruência que contém r. Segue que o fecho transitivo, t, de s está contido no fecho transitivo, r_c, de r_c. Logo, $t \subseteq r_c$.

60. Sejam A, B e C subconjuntos de um monóide. Mostre que:
   (i) Se $A \subseteq B$ então $AC \subseteq BC$ e $CA \subseteq CB$.
   (ii) $A(B \cup C) = AB \cup AC$.
   (iii) $(A \cup B)C = AC \cup BC$.
   (iv) $A(B \cap C) \subseteq AB \cap AC$.
   (v) $(A \cap B)C \subseteq AC \cap BC$.
   (vi) Se $A \subseteq B$ então $A^* \subseteq B^*$.
   (vii) $(A^*)^* = A^*$.

61. Seja $f : M_1 \to M_2$ um morfismo de monóides, e sejam A e B subconjuntos de M_1. Mostre que:
   (i) $(AB)f = (Af)(Bf)$.
   (ii) $A^*f = (Af)^*$.

62. Seja A um subconjunto do monóide M. Mostre que existem um monóide livre Σ^* e um morfismo $f : \Sigma^* \to M$, tais que $\Sigma f = A$ e $\Sigma^* f = A^*$.

63. Sejam A e B subconjuntos do monóide livre Σ^*, tais que $\Sigma = A \cup B$. Mostre que $\Sigma^* = A^*(BA^*)^*$.

64. Sejam A e B subconjuntos do monóide livre Σ^*. Mostre que:
   (i) $A(BA)^* = (AB)^*A$.
   (ii) $(A \cup B)^* = A^*(BA^*)^*$.
   (iii) $A^*(BA^*)^* = (B^*A)^*B^*$.

65. Sejam s e t palavras em Σ^*. Mostre que $st = ts$ sse existem naturais m e n e uma palavra x em Σ^*, tais que $s = x^m$ e $t = x^n$.

66. Sejam A e B subconjuntos de monóide livre Σ^*. Definimos:

$$A^{-1}B = \{t \in \Sigma^* \mid \text{existe } s \in A \text{ tal que } st \in B\},$$
$$\text{e} \quad AB^{-1} = \{s \in \Sigma^* \mid \text{existe } t \in B \text{ tal que } st \in A\}.$$

Mostre que para todo $A, B, C \subseteq \Sigma^*$,
   (i) $(AC^{-1})B^{-1} = A(BC)^{-1}$.
   (ii) $(A^{-1}B)C^{-1} = A^{-1}(BC^{-1})$.
   (iii) $(AB)^{-1}C = B^{-1}(A^{-1}C)$.

67. Seja Σ^* um monóide livre. A função *reversão* $\rho : \Sigma^* \to \Sigma^*$ é definida por: $1\rho = 1$ e $(s\sigma)\rho = \sigma(s\rho)$ para todo $s \in \Sigma^*$ e $\sigma \in \Sigma$. A imagem de $s \in \Sigma^*$ por ρ é chamada de *reverso* de s. Sendo $s, t \in \Sigma^*$ e $A, B \subseteq \Sigma^*$, mostre que:
    (i) $s\rho\rho = s$.
    (ii) $(st)\rho = (t\rho)\,(s\rho)$.
    (iii) $(AB)\rho = (B\rho)\,(A\rho)$.
    (iv) $A^*\rho = (A\rho)^*$.
    (v) $(A^{-1}B)\rho = B\rho(A\rho)^{-1}$ e $(AB^{-1})\rho = (B\rho)^{-1}(A\rho)$: (Vide Exercício 66).

68. Um subconjunto A de Σ^* é um *prefixo* se para todo s em A, o único segmento inicial de s que pertence a A é o próprio s. Em símbolos, para todo $s, t \in \Sigma^*$, se $s, st \in A$, então $t = 1$. Mostre que as afirmações abaixo são equivalentes (vide Exercício 66 para notação):
    (i) A é um prefixo.
    (ii) Para todo $s_1, s_2, t_1, t_2 \in \Sigma^*$, se $s_1t_1 = s_2t_2$ e $s_1, s_2 \in A$, entao $s_1 = s_2$ e $t_1 = t_2$.
    (iii) Para todo $s, t_1, t_2 \in \Sigma^*$, se $st_1, st_2 \in A$, então $t_1t_2 = 1$ ou $s \notin A$.
    (iv) $s^{-1}A = 1$ para todo $s \in A$.
    (v) $A = \varnothing$ ou $A^{-1}A = 1$.

69. Sejam A, B e C subconjuntos de Σ^*. Mostre que (confronte com o Exercício 60):
    (i) Se $A \cup B$ for um prefixo, então $(A \cap B)C = AC \cap BC$.
    (ii) Se A for um prefixo, então $A(B \cap C) = AB \cap AC$.
    (iii) Se A for um prefixo, então[2] $A(B \oplus C) = AB \oplus AC$.

70. Um subconjunto A de Σ^* é um *sufixo* se o reverso $A\rho$ de A for um prefixo. Enuncie e demonstre os análogos dos Exercícios 68 e 69, substituindo prefixos por sufixos.

71. Seja A um submonóide do monóide livre Σ^*. Um subconjunto B de A é um *gerador* de A se $B^* = A$. Mostre que

$$(A\backslash 1)\backslash(A\backslash 1)^2$$

é o único gerador minimal de A.

---

(2) Para conjuntos A e B, $A \oplus B$ denota a diferença simétrica de A e B, isto é, $A \oplus B = A\backslash B \cup B\backslash A$.

NOTAS BIBLIOGRÁFICAS

Os conceitos e resultados deste capítulo fazem parte do folclore da Matemática. Para um estudo mais detalhado de relações e funções recomendamos os livros de Halmos [47] ou de MacLane e Birkhoff [72]. O leitor interessado encontrará um tratamento detalhado de monóides no livro de Clifford e Preston [15].

CAPÍTULO II

CONJUNTOS RACIONAIS E O TEOREMA DE KLEENE

1. Autômatos, conjuntos reconhecíveis e conjuntos racionais

Um *autômato finito* $\mathscr{A}$ (ou simplesmente *autômato* $\mathscr{A}$) consiste de:

um conjunto finito Q de elementos chamados *estados*,
um alfabeto finito Σ,
uma função $\alpha\colon \Sigma \to Rel\ Q$,
um subconjunto I de Q, cujos elementos são chamados de *estados iniciais*,
um subconjunto F de Q, cujos elementos são chamados de *estados finais*.

O autômato acima será denotado por $\mathscr{A} = (Q, \Sigma, \alpha, I, F)$.

De acordo com a Proposição I.4, α tem uma única extensão a um morfismo

$$\Sigma^* \to Rel\ Q,$$

que também será denotada por α. Resulta que $1\alpha = 1_Q$. Ademais, dada a palavra s:

$$s = \sigma_1\sigma_2 \ldots \sigma_n \qquad (\sigma_i \in \Sigma,\ i = 1, 2, \ldots, n),$$

$s\alpha$ é a relação

$$s\alpha = (\sigma_1\alpha)(\sigma_2\alpha)\ldots(\sigma_n\alpha).$$

A definição de composição de relações implica que, dados estados q e q',

$$q' \in q(s\alpha)$$

see existem estados $q_0, q_1, \ldots, q_n$, com $q_0 = q$ e $q_n = q'$, tais que

$$q_i \in q_{i-1}(\sigma_i\alpha) \qquad (i = 1, 2, \ldots, n).$$

Passamos a dar uma descrição combinatória de autômatos, bem como das relações $s\alpha$. Definimos inicialmente o conjunto E de *arestas*

do autômato $\mathscr{A}$, como sendo o subconjunto de $Q \times \Sigma \times Q$, dado por

$$E = \{(q, \sigma, q') \in Q \times \Sigma \times Q \mid (q, q') \in \sigma\alpha\}.$$

Este conceito sugere uma representação gráfica de autômatos. Cada estado é representado por um pequeno círculo. Uma flecha entrando em (saindo de) um estado, indica que este é um estado inicial (final). Para cada aresta (q, σ, q') de $\mathscr{A}$ desenha-se uma flecha rotulada σ, saindo de q e entrando em q'. Na Figura 1 damos um exemplo, onde, como nos demais exemplos, usamos $\Sigma = \{\sigma, \tau\}$.

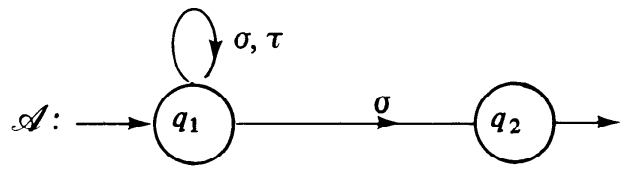


Figura 1 — Um autômato $\mathscr{A}$.

Um *passeio* c em $\mathscr{A}$ é uma seqüência finita de arestas, da forma

$$c = (q_0, \sigma_1, q_1)(q_1, \sigma_2, q_2) \ldots (q_{n-1}, \sigma_n, q_n) \quad (n \geq 1).$$

Dizemos que q_0 é a *origem* e q_n o *término* do passeio c. O *rótulo* $|c|$ do passeio c é a palavra de Σ^* dada por

$$|c| = \sigma_1\sigma_2 \ldots \sigma_n.$$

Definimos também, para cada estado q de A o *passeio degenerado* 1_q, que tem origem e término em q e não contém arestas. Por definição, o rótulo do passeio degenerado 1_q é a palavra vazia, isto é $|1_q| = 1$. O natural n é chamado de *comprimento* do passeio c; o comprimento do passeio degenerado é zero. Observe que o comprimento de um passeio c é igual ao comprimento de seu rótulo $|c|$, isto é o comprimento de c é dado por $||c||$. Sendo c um passeio com origem q, término q' e rótulo s, as seguintes notações serão utilizadas:

$$c: q \xrightarrow{s} q', \quad q \xrightarrow{s} q', \quad q \xrightarrow{c} q' \quad \text{e} \quad c: q \longrightarrow q'.$$

Em particular, uma aresta (q, σ, q'), poderá também ser denotada por

$$q \xrightarrow{\sigma} q'.$$

Dados passeios c_1 e c_2,

$$c_1 : q_1 \to q_2 \text{ e } c_2 : q_2 \to q_3,$$

tais que o término de c_1 é igual à origem de c_2, definimos o seu *produto*

$$c_1c_2 : q_1 \xrightarrow{c_1} q_2 \xrightarrow{c_2} q_3,$$

obtido por concatenação. Está claro que o produto de passeios é uma operação associativa, isto é,

$$(c_1c_2)c_3 = c_1(c_2c_3),$$

dado que os produtos c_1c_2 e c_2c_3 podem ser efetuados. Ademais, para todo passeio c com origem p e término q,

$$1_p c = c = c 1_q.$$

Temos também que:

$$|c_1c_2| = |c_1|\,|c_2| \text{ e } \|c_1c_2\| = \|c_1\| + \|c_2\|.$$

Para efeito de ilustração, consideremos os passeios c_1 e c_2 no autômato $\mathscr{A}$ da Figura 1:

$$c_1 = (q_1, \sigma, q_1)\,(q_1, \tau, q_1)\,(q_1, \sigma, q_1)$$

e

$$c_2 = (q_1, \sigma, q_1)\,(q_1, \tau, q_1)\,(q_1, \sigma, q_2).$$

Temos que $|c_1| = |c_2| = \sigma\tau\sigma$, $\|c_1\| = \|c_2\| = 3$ e o passeio

$$c_1c_2 : q_1 \to q_2$$

é dado por

$$c_1c_2 = (q_1, \sigma, q_1)(q_1, \tau, q_1)(q_1, \sigma, q_1)(q_1, \sigma, q_1)(q_1, \tau, q_1)(q_1, \sigma, q_2),$$

seu rótulo é $|c_1c_2| = \sigma\tau\sigma\sigma\tau\sigma$, e seu comprimento é $\|c_1c_2\| = 6$.

A proposição a seguir caracteriza as relações $s\alpha$ em termos de passeios:

PROPOSIÇÃO 1. *Seja $\mathscr{A} = (Q, \Sigma, \alpha, I, F)$ um autômato. Para toda palavra s em Σ^* e estados q e q', (q, q') pertence a $s\alpha$ sse existe em $\mathscr{A}$ um passeio c com origem q, término q' e rótulo s. Em símbolos,*

$$(q, q') \in s\alpha \text{ sse existe } c : q \xrightarrow{s} q'.$$

Demonstração. Procedemos por indução sobre o comprimento da palavra s em Σ^*. Se $|s| = 0$, então $s = 1$ e $s\alpha = 1_Q$. Por outro lado, os passeios com rótulo $s = 1$ são todos degenerados.

Segue que existe um passeio

$$c : q \xrightarrow{1} q'$$

sse $q = q'$ sse $(q, q') \in 1_Q = s\alpha$. Supondo agora que $|s| \geq 1$, existem t em Σ^* e σ em Σ, tais que $s = t\sigma$. Então, as afirmações seguintes são equivalentes:

(i) $(q, q') \in s\alpha$.

(ii) existe $r \in Q$, tal que $(q, r) \in t\alpha$ e $(r, q') \in \sigma\alpha$.

(iii) existem $r \in Q$, e passeios $c_1 : q \xrightarrow[s]{t} r$ e $c_2 : r \xrightarrow{\sigma} q'$.

(iv) existe em $\mathscr{A}$ um passeio $c : q \xrightarrow{s} q'$.

De fato, α sendo um morfismo, $s\alpha = (t\sigma)\alpha = (t\alpha)(\sigma\alpha)$, logo a equivalência de (i) e (ii) segue da definição de composição de relações. Agora, (ii) e (iii) são equivalentes pela hipótese da indução e a observação de que $(r, q') \in \sigma\alpha$ sse (r, σ, q') é uma aresta de $\mathscr{A}$. Finalmente, se (iii) estiver satisfeita, então o passeio $c = c_1c_2$ tem rótulo $|c| = t\sigma = s$, logo c satisfaz (iv). Reciprocamente, se (iv) estiver satisfeita, então o passeio c tem rótulo $s = t\sigma$, logo ele admite uma fatoração

$$c : q \xrightarrow{t} r \xrightarrow{\sigma} q'$$

para algum r em Q. Assim, (iii) está satisfeita. A equivalência de (i) e (iv) completa a demonstração. ■

A cada autômato $\mathscr{A} = (Q, \Sigma, \alpha, I, F)$ associamos um subconjunto de Σ^*, denotado por $|\mathscr{A}|$ e dado por

$$|\mathscr{A}| = \{s \in \Sigma^* \mid I(s\alpha) \cap F \neq \varnothing\}.$$

Em vista da Proposição 1, está claro que

$$|\mathscr{A}| = \{s \in \Sigma^* \mid \text{existem } i \in I, f \in F, \text{ e um passeio } c: i \xrightarrow{s} f\}.$$

Dizemos que $|\mathscr{A}|$ é o *comportamento* de $\mathscr{A}$, e também que $\mathscr{A}$ *reconhece* $|\mathscr{A}|$. Um subconjunto A de Σ^* é *reconhecível* se A for o comportamento de algum autômato, isto é, $A = |\mathscr{A}|$ para algum autômato $\mathscr{A}$. Denotamos o conjunto dos subconjuntos reconhecíveis de Σ^* por

$$Rec\ \Sigma.$$

No caso do autômato da Figura 1,

$$|\mathscr{A}| = (\sigma \cup \tau)^*\sigma,$$

portanto o conjunto $(\sigma \cup \tau)^*\sigma$ é reconhecível. Deixamos ao leitor a construção de autômatos que reconhecem os conjuntos $\varnothing$, Σ^*, 1 e σ (para $\sigma \in \Sigma$).

Um autômato $\mathscr{A} = (Q, \Sigma, \alpha, I, F)$ é *determinístico* se *card* $I = 1$ e para todo σ em Σ, a relação $\sigma\alpha$ é uma função. Neste caso α é uma função

$$\alpha : \Sigma \to Fun\ Q$$

e sua extensão é um morfismo

$$\alpha : \Sigma^* \to Fun\ Q.$$

Assim, se i for o estado inicial do autômato determinístico $\mathscr{A}$, então o seu comportamento pode também ser expresso por

$$|\mathscr{A}| = \{s \in \Sigma^* \mid i(s\alpha) \in F\}.$$

Por outro lado, pela Proposição 1, um autômato $\mathscr{A} = (Q, \Sigma, \alpha, i, F)$, onde i representa o estado inicial, é determinístico sse para cada par (q, s) em $Q \times \Sigma^*$, existe exatamente um passeio em $\mathscr{A}$ com origem em q e com rótulo s. A Figura 2 mostra um autômato determinístico $\mathscr{B}$. Deixamos ao leitor a verificação de que o comportamento de $\mathscr{B}$ também é $(\sigma \cup \tau)^*\sigma$.

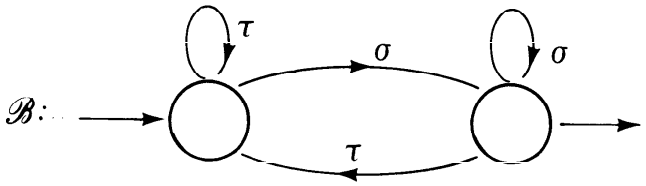


Figura 2 — Um autômato determinístico.

Um subconjunto $\mathscr{F}$ de $\mathfrak{p}(\Sigma^*)$ é *racionalmente fechado* se $\varnothing$ e 1 pertencem a $\mathscr{F}$ e se para todo A e A' em $\mathscr{F}$,

$$A \cup A',\ AA' \text{ e } A^*$$

também pertencem a $\mathscr{F}$. É fácil ver que a intersecção de subconjuntos racionalmente fechados de $\mathfrak{p}(\Sigma^*)$ também é racionalmente fechada. Denotamos por

$$Rac\ \Sigma$$

o menor subconjunto racionalmente fechado de $\mathfrak{p}(\Sigma^*)$ que contém os conjuntos unitários σ, para todo σ em Σ. Os elementos de $Rac\ \Sigma$ são chamados de *subconjuntos racionais* de Σ^*.

Alguns exemplos de subconjuntos racionais de Σ^* são

$$\Sigma^* = (\sigma \cup \tau)^*, \ (\sigma \cup \tau)^*\sigma, \ (\sigma\tau \cup \tau\sigma)^* \text{ e } (\sigma\sigma)^*.$$

Note que os elementos de *Rac* Σ são aqueles subconjuntos de Σ^* que podem ser obtidos a partir dos conjuntos unitários σ, utilizando-se um número finito de vezes as operações de união, concatenação e estrela.

O Teorema de Kleene, que demonstramos a seguir, afirma que para todo alfabeto finito Σ,

$$Rec\ \Sigma = Rac\ \Sigma.$$

2. Operações sobre conjuntos reconhecíveis

Tendo em vista a verificação da inclusão *Rac* $\Sigma \subseteq$ *Rec* Σ, mostramos nesta seção que *Rec* Σ é fechado sob várias operações. Nos exercícios indicamos outras propriedades de fecho de *Rec* Σ.

PROPOSIÇÃO 2. *Todo subconjunto reconhecível de* Σ^* *é o comportamento de um autômato determinístico.*

Demonstração. Sejam A em *Rec* Σ e $\mathscr{A} = (Q, \Sigma, \alpha, I, F)$ um autômato (não necessariamente determinístico), que reconhece A. Seja $\mathscr{B} = (\mathfrak{p}(Q), \Sigma, \beta, I, G)$ o autômato, onde

$$G = \{P \subseteq Q \mid P \cap F \neq \varnothing\},$$

e para todo $P \subseteq Q$ e $\sigma \in \Sigma$,

$$P(\sigma\beta) = P(\sigma\alpha).$$

É imediato que o autômato $\mathscr{B}$ é determinístico. Por outro lado, mostra-se, por indução no comprimento de s em Σ^*, que para todo $P \subseteq Q$,

$$P(s\beta) = P(s\alpha).$$

Assim,

$$s \in |\mathscr{A}| \quad \text{sse} \quad I(s\alpha) \cap F \neq \varnothing \quad \text{sse} \quad I(s\alpha) = I(s\beta) \in G \quad \text{sse} \quad s \in |\mathscr{B}|.$$

Logo, $|\mathscr{A}| = |\mathscr{B}| = A$. ■

A construção da Proposição 2 é chamada de *construção dos subconjuntos*. Na Figura 3 indicamos o autômato determinístico obtido a partir do autômato da Figura 1 através desta construção.

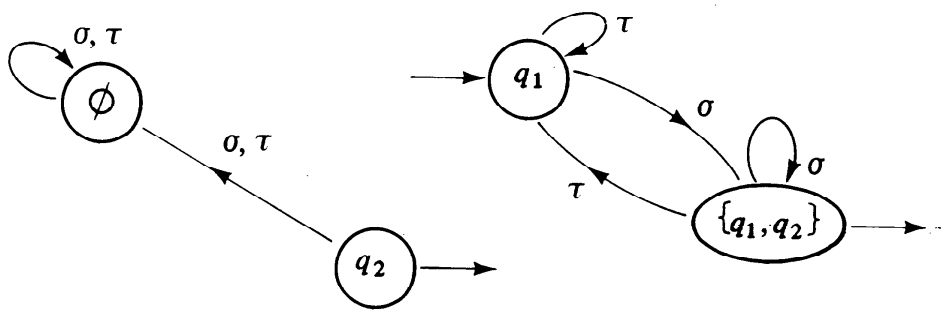


Figura 3 — Exemplo para a construção dos subconjuntos.

Para $A \subseteq \Sigma^*$, denotamos por $\bar{A}$ o *complemento* de A em Σ^*, isto é

$$\bar{A} = \Sigma^* \backslash A.$$

COROLÁRIO 1. *Rec* Σ *é fechado sob complementação.*

Demonstração. Seja A em *Rec* Σ e seja $\mathscr{A} = (Q, \Sigma, \alpha, i, F)$ um autômato determinístico, que reconhece A. Definimos

$$\mathscr{B} = (Q, \Sigma, \alpha, i, Q \backslash F).$$

Segue que

$$|\mathscr{B}| = \Sigma^* \backslash |\mathscr{A}| = \bar{A}.$$

(Note que a demonstração não é válida se $\mathscr{A}$ não for um autômato determinístico!) ■

PROPOSIÇÃO 3. *Rec* Σ *é fechado sob união.*

Demonstração. Sejam A_i em *Rec* Σ e $\mathscr{A}_i = (Q_i, \Sigma, \alpha_i, I_i, F_i)$ autômatos tais que $|\mathscr{A}_i| = A_i$, para $i = 1, 2$. Sem perda de generalidade, podemos supor que $Q_1 \cap Q_2 = \varnothing$. Definimos

$$\mathscr{B} = (Q_1 \cup Q_2, \Sigma, \beta, I_1 \cup I_2, F_1 \cup F_2);$$

onde para todo $\sigma \in \Sigma$, $\sigma\beta = \sigma\alpha_1 \cup \sigma\alpha_2$. Deixamos ao leitor a verificação de que

$$|\mathscr{B}| = |\mathscr{A}_1| \cup |\mathscr{A}_2| = A_1 \cup A_2. \blacksquare$$

COROLÁRIO 2. *Rec* Σ *é fechado sob intersecção e sob complementação relativa.*

Demonstração. Sejam A_1 e A_2 em *Rec* Σ. Então

$$A_1 \cap A_2 = \overline{\overline{A_1} \cup \overline{A_2}} \text{ e } A_1 \backslash A_2 = A_1 \cap \overline{A_2}.$$

O resultado segue do Corolário 1 e da Proposição 3. ■

PROPOSIÇÃO 4. *Rec* Σ *é fechado sob concatenação.*

Demonstração. Sejam A_1 e A_2 em *Rec* Σ e sejam $\mathscr{A}_i = (Q_i, \Sigma, \alpha_i, I_i, F_i)$ autômatos tais que

$$|\mathscr{A}_1| = A_1, |\mathscr{A}_2| = A_2 \text{ e } Q_1 \cap Q_2 = \varnothing.$$

Definimos

$$\mathscr{B} = (Q_1 \cup Q_2, \Sigma, \beta, I, F_2),$$

onde

$$I = \begin{cases} I_1 & \text{se } I_1 \cap F_1 = \varnothing \\ I_1 \cup I_2 & \text{se } I_1 \cap F_1 \neq \varnothing, \end{cases}$$

e, para todo σ em Σ,

$$\sigma\beta = \sigma\alpha_1 \cup \sigma\alpha_2 \cup \{(q_1, q_2) \in Q_1 \times Q_2 \mid q_1(\sigma\alpha_1) \cap F_1 \neq \varnothing \text{ e } q_2 \in I_2\}.$$

Observe que se E_1 e E_2 são os conjuntos de arestas de $\mathscr{A}_1$ e $\mathscr{A}_2$, respectivamente, então o conjunto E de arestas de $\mathscr{B}$ é dado por

$$E = E_1 \cup E_2 \cup \{(q_1, \sigma, q_2) \in Q_1 \times \Sigma \times Q_2 \mid q_2 \in I_2 \text{ e existem } f \in F_1 \text{ e } (q_1, \sigma, f) \in E_1\}.$$

Deixamos ao leitor a verificação trabalhosa, mas sem novidades, de que

$$|\mathscr{B}| = |\mathscr{A}_1| \, |\mathscr{A}_2| = A_1 A_2. \blacksquare$$

PROPOSIÇÃO 5. *Rec* Σ *é fechado sob estrela.*

Demonstração. Seja A em *Rec* Σ, e seja $\mathscr{A} = (Q, \Sigma, \alpha, I, F)$, um autômato que reconhece A. Seja p um estado que não pertence a Q. Definimos

$$\mathscr{B} = (Q \cup p, \Sigma, \beta, p, p),$$

onde para todo σ em Σ

$$\sigma\beta = X'_\sigma \cup \sigma\alpha \cup \{(p,q) \in p \times Q \mid q \in I(\sigma\alpha)\} \cup$$
$$\cup \{(q,p) \in Q \times p \mid q(\sigma\alpha) \cap F \neq \varnothing\},$$

com

$$X_\sigma = \begin{cases} \varnothing \text{ se } I(\sigma\alpha) \cap F = \varnothing \\ (p,p) \text{ caso contrário.} \end{cases}$$

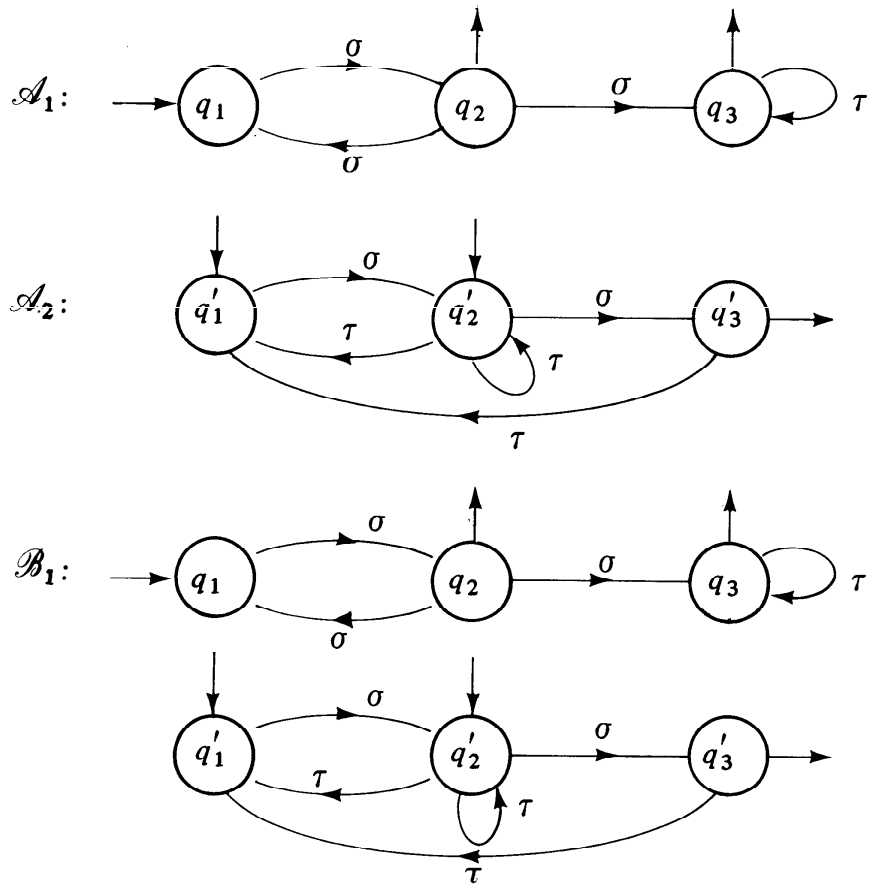
Observe que se $E_{\mathscr{A}}$ for o conjunto de arestas de $\mathscr{A}$, então o conjunto $E_{\mathscr{B}}$ de arestas de $\mathscr{B}$ é dado por

$$E_{\mathscr{B}} = E_{\mathscr{A}} \cup \{(p,\sigma,q) \in p \times \Sigma \times Q \mid \text{existem } i \in I \text{ e } (i,\sigma,q) \in E_{\mathscr{A}}\} \cup$$
$$\cup \{(q,\sigma,p) \in Q \times \Sigma \times p \mid \text{existem } f \in F \text{ e } (q,\sigma,f) \in E_{\mathscr{A}}\} \cup$$
$$\cup \{(p,\sigma,p) \in p \times \Sigma \times p \mid \text{existem } i \in I, f \in F \text{ e } (i,\sigma,f) \in E_{\mathscr{A}}\}.$$

Mais uma vez deixamos ao leitor a verificação de que

$$|\mathscr{B}| = |\mathscr{A}|^* = A^*. \blacksquare$$

Na Figura 4 são apresentados exemplos das construções das Proposições 3, 4 e 5.



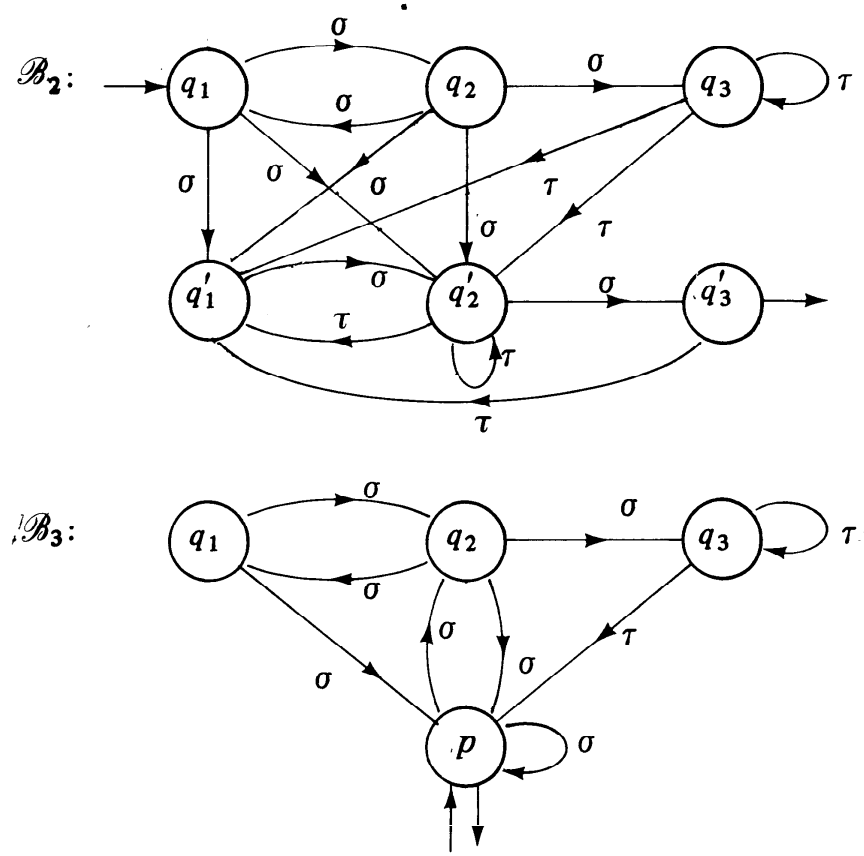


Figura 4 — Exemplos das construções $|\mathscr{B}_1| = |\mathscr{A}_1| \cup |\mathscr{A}_2|$, $|\mathscr{B}_2| = |\mathscr{A}_1|\,|\mathscr{A}_2|$ e $|\mathscr{B}_3| = |\mathscr{A}_1|^*$.

3. Sistemas de equações lineares

Tendo em vista a verificação da inclusão $Rec\,\Sigma \subseteq Rac\,\Sigma$, estudamos nesta seção sistemas de equações lineares, cujos coeficientes e incógnitas são subconjuntos de Σ^*. Mais precisamente, dados um natural $n \geq 1$ e subconjuntos de Σ^*

$$E_{ij} \text{ e } T_i, \text{ para } 1 \leq i, j \leq n,$$

estamos interessados em subconjuntos de Σ^*

$$X_i, \text{ para } 1 \leq i \leq n,$$

que satisfazem o sistema abaixo de n equações:

$$X_i = T_i \cup \bigcup_{j=1}^{n} E_{ij}X_j \quad (i = 1, 2, \ldots, n). \tag{1}$$

Mostraremos que sob certas condições o sistema (1) admite uma única solução que é constituída de subconjuntos racionais de Σ^*.

PROPOSIÇÃO 6. *Se* $1 \notin E_{ij}$, *para* $1 \leq i, j \leq n$, *então o sistema* (1) *admite no máximo uma solução.*

Demonstração. Suponhamos que os subconjuntos $X_1, \ldots, X_n$ e $Y_1, \ldots, Y_n$ de Σ^* são soluções de (1). Mostramos por indução em $|s|$, para $s \in \Sigma^*$, que

$$s \in X_i \text{ sse } s \in Y_i \quad (i = 1, 2, \ldots, n).$$

De fato, se $|s| = 0$, então $s = 1$ e como $1 \notin E_{ij}$ por hipótese, resulta de (1) que

$$1 \in X_i \text{ sse } 1 \in T_i \text{ sse } 1 \in Y_i \quad (i = 1, 2, \ldots, n).$$

Seja então s em Σ^*, tal que $|s| > 0$, e suponhamos que $s \in X_i$. Resulta de (1), que

$$s \in T_i \cup \bigcup E_{ij}X_j.$$

Se $s \in T_i$, então

$$s \in T_i \cup \bigcup E_{ij}Y_j = Y_i.$$

Caso contrário, para algum j, $1 \leq j \leq n$, $s \in E_{ij}X_j$. Assim, existem $t_1 \in E_{ij}$ e $t_2 \in X_j$, tais que $t_1t_2 = s$. Por hipótese $1 \notin E_{ij}$, logo $|t_1| > 0$. Consequentemente $|t_2| < |s|$, e pela hipótese da indução, $t_2 \in Y_j$. Segue que

$$s = t_1t_2 \in E_{ij}Y_j \subseteq T_i \cup \bigcup E_{ij}Y_j = Y_i.$$

Em ambos os casos, $s \in Y_i$. Argumento simétrico mostra que se $s \in Y_i$ então $s \in X_i$. Logo, $X_i = Y_i$, para $i = 1, 2, \ldots, n$. ■

PROPOSIÇÃO 7. *O sistema* (1) *tem pelo menos uma solução. Além disso, se* E_{ij} *e* T_i $(1 \leq i, j \leq n)$ *são subconjuntos racionais de* Σ^*, *então o sistema* (1) *tem uma solução* X_i $(1 \leq i \leq n)$, *constituída de subconjuntos racionais de* Σ^*.

Demonstração. Procedemos por indução em n. Se $n = 1$, a equação

$$X_1 = T_1 \cup E_{11}X_1$$

admite a solução

$$X_1 = E_{11}^*T_1.$$

De fato, usando a identidade $E_{11}^* = 1 \cup E_{11}E_{11}^*$, obtemos

$$X_1 = E_{11}^* T_1 = (1 \cup E_{11}E_{11}^*)T_1 = T_1 \cup E_{11}E_{11}^* T_1 = T_1 \cup E_{11}X_1.$$

Além disso, se E_{11} e T_1 são racionais, $E_{11}^* T_1$ também o é. Suponhamos então que $n > 1$, e consideremos o sistema abaixo de $n - 1$ equações:

$$X_i = T_i' \cup \bigcup_{j=1}^{n-1} E_{ij}' X_j \quad (1 \leq i \leq n - 1), \tag{2}$$

onde

$$E_{ij}' = E_{ij} \cup E_{in}E_{nn}^* E_{nj} \quad (1 \leq i, j \leq n - 1), \tag{3}$$

e

$$T_i' = T_i \cup E_{in}E_{nn}^* T_n \quad (1 \leq i \leq n - 1). \tag{4}$$

Pela hipótese da indução, o sistema (2) tem uma solução X_i $(1 \leq i \leq n-1)$. Colocamos agora

$$X_n = E_{nn}^* \left(T_n \cup \bigcup_{j=1}^{n-1} E_{nj} X_j \right) \tag{5}$$

Os X_i $(1 \leq i \leq n)$ assim obtidos satisfazem (1). De fato, para $1 \leq i \leq n - 1$,

$$\begin{aligned} X_i &= T_i' \cup \bigcup_{j=1}^{n-1} E_{ij}' X_j = \\ &= T_i \cup E_{in}E_{nn}^* T_n \cup \bigcup_{j=1}^{n-1} (E_{ij} \cup E_{in}E_{nn}^* E_{nj}) X_j = \\ &= T_i \cup \bigcup_{j=1}^{n-1} E_{ij}X_j \cup E_{in}E_{nn}^* \left(T_n \cup \bigcup_{j=1}^{n-1} E_{nj}X_j \right) = \\ &= T_i \cup \bigcup_{j=1}^{n} E_{ij}X_j. \end{aligned}$$

Temos ainda, colocando $E_{nn}^* = 1 \cup E_{nn}E_{nn}^*$,

$$\begin{aligned} X_n &= E_{nn}^* \left(T_n \cup \bigcup_{j=1}^{n-1} E_{nj}X_j \right) = \\ &= (1 \cup E_{nn}E_{nn}^*) \left(T_n \cup \bigcup_{j=1}^{n-1} E_{nj}X_j \right) = \end{aligned}$$

$$= T_n \cup \bigcup_{j=1}^{n-1} E_{nj}X_j \cup E_{nn}E_{nn}^*\left(T_n \cup \bigcup_{j=1}^{n-1} E_{nj}X_j\right) =$$

$$= T_n \cup \bigcup_{j=1}^{n} E_{nj}X_j.$$

Além disso, se E_{ij} e T_i $(1 \leq i, j \leq n)$ são racionais, resulta de (3) e (4) que E'_{ij} e T'_i $(1 \leq i, j \leq n-1)$ também são racionais. Pela hipótese da indução, (2) tem uma solução X_i $(1 \leq i \leq n-1)$, que é constituída de subconjuntos racionais de Σ^*. Vem de (5) que neste caso X_n é racional também. ■

Observamos que a solução obtida na proposição acima, pode também ser obtida do seguinte modo. Reescrevemos a n-ésima equação de (1):

$$X_n = T_n \cup \bigcup_{j=1}^{n} E_{nj}X_j = \left(T_n \cup \bigcup_{j=1}^{n-1} E_{nj}X_j\right) \cup E_{nn}X_n. \qquad (6)$$

Pela fórmula obtida na base da indução, (5) pode ser considerada uma solução formal de (6). Substituindo (5) nas $n-1$ primeiras equações de (1), obtém-se exatamente o sistema (2).

Juntando as proposições 6 e 7, obtemos:

PROPOSIÇÃO 8. *Se E_{ij} e T_i $(1 \leq i, j \leq n)$ são subconjuntos racionais de Σ^*, tais que $1 \notin E_{ij}$ $(1 \leq i, j \leq n)$, então o sistema (1) tem uma única solução X_i $(1 \leq i \leq n)$. Esta é constituída de subconjuntos racionais de Σ^*.*

4. O Teorema de Kleene

Estamos agora prontos para provar o Teorema de Kleene, isto é a igualdade dos conjuntos *Rec* Σ e *Rac* Σ.

TEOREMA 1 (Kleene). *Para todo alfabeto finito* Σ,

$$Rec\,\Sigma = Rac\,\Sigma.$$

Demonstração. As Proposições 3, 4 e 5 mostram que *Rec* Σ é fechado sob união, concatenação e estrela. Por outro lado, $\varnothing$, 1 e $\sigma \in \Sigma$ são reconhecíveis, logo *Rec* Σ é um subconjunto racio-

nalmente fechado de $\mathfrak{p}(\Sigma^*)$ que contém os conjuntos unitários σ para todo $\sigma \in \Sigma$. Segue da definição de $Rac\,\Sigma$, que

$$Rac\,\Sigma \subseteq Rec\,\Sigma.$$

Para mostrar a inclusão recíproca, seja $A \in Rec\,\Sigma$, e seja $\mathscr{A} = (Q, \Sigma, \alpha, I, F)$ um autômato que reconhece A. Sejam $n = card\,Q$ e $q_1, q_2, \ldots, q_n$ uma enumeração arbitrária dos elementos de Q. Definimos:

$$\mathscr{A}_i = (Q, \Sigma, \alpha, q_i, F) \qquad (1 \leq i \leq n),$$
$$X_i = |\mathscr{A}_i| \qquad (1 \leq i \leq n),$$
$$E_{ij} = \{\sigma \in \Sigma \mid q_j \in q_i(\sigma\alpha)\} \qquad (1 \leq i, j \leq n),$$

e

$$T_i = \begin{cases} 1 \text{ se } q_i \in F \\ \\ \varnothing \text{ se } q_i \notin F \end{cases} \qquad (1 \leq i \leq n).$$

Os conjuntos acima satisfazem as identidades

$$X_i = T_i \cup \bigcup_{j=1}^{n} E_{ij}X_j \quad (1 \leq i \leq n). \tag{1}$$

De fato,

$$\begin{aligned} X_i = |\mathscr{A}_i| &= \{s \in \Sigma^* \mid q_i(s\alpha) \cap F \neq \varnothing\} = \\ &= T_i \cup \{s \in \Sigma^* \mid |s| \geq 1 \text{ e } q_i(s\alpha) \cap F \neq \varnothing\} = \\ &= T_i \cup \bigcup_{q_j \in Q} \{\sigma \in \Sigma \mid q_j \in q_i(\sigma\alpha)\}\{t \in \Sigma^* \mid q_j(t\alpha) \cap F \neq \varnothing\} = \\ &= T_i \cup \bigcup_{j=1}^{n} E_{ij}X_j. \end{aligned}$$

Ora, T_i $(1 \leq i \leq n)$ é racional, e como Σ é finito e $E_{ij} \subseteq \Sigma$, E_{ij} $(1 \leq i, j \leq n)$ é racional também. Ademais, $1 \notin E_{ij}$, logo, pela Proposição 8, o sistema (1) admite uma única solução, e esta é constituída de subconjuntos racionais de Σ^*. Concluímos que X_i é racional, para $1 \leq i \leq n$. Finalmente,

$$A = |\mathscr{A}| = \{s \in \Sigma^* \mid I(s\alpha) \cap F \neq \varnothing\} = \bigcup_{q_i \in I} \{s \in \Sigma^* \mid q_i(s\alpha) \cap F \neq \varnothing\} = \bigcup_{q_i \in I} X_i.$$

Assim, A é um subconjunto racional de Σ^* e

$$Rec\,\Sigma \subseteq Rac\,\Sigma.$$

Isto estabelece o Teorema de Kleene. ■

COROLÁRIO 3. *Rac* Σ *é fechado sob complementação.*

Demonstração. Segue do Teorema de Kleene e do Corolário 1. ■

Observamos que o Corolário 3 é um resultado algébrico sobre um conjunto (*Rac* Σ) definido algebricamente. A sua demonstração porém, exige toda a força do Teorema de Kleene. Em outras palavras, para provar o Corolário 3, tornou-se necessário introduzir o conceito de autômato! De fato, não conhecemos demonstração deste resultado que não utilize autômatos (ou algo equivalente).

Para exemplificar o cálculo acima, considere o autômato da Figura 5.

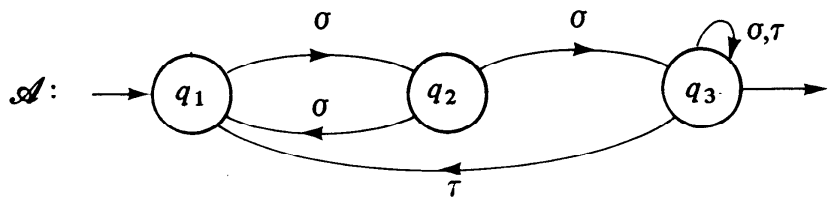


Figura 5 — Um exemplo.

Resulta o seguinte sistema de 3 equações:

$$\begin{aligned} X_1 &= \sigma X_2 \\ X_2 &= \sigma X_1 \cup \sigma X_3 \\ X_3 &= 1 \cup \tau X_1 \cup (\sigma \cup \tau) X_3 . \end{aligned}$$

Resolvendo a terceira equação em X_3, obtemos:

$$X_3 = (\sigma \cup \tau)^* (1 \cup \tau X_1).$$

Substituindo X_3 na segunda equação, obtemos:

$$X_2 = \sigma(\sigma \cup \tau)^* \cup [\sigma \cup \sigma(\sigma \cup \tau)^* \tau] X_1 .$$

Substituindo X_2 na primeira equação, obtemos:

$$X_1 = \sigma\sigma(\sigma \cup \tau)^* \cup \sigma[\sigma \cup \sigma(\sigma \cup \tau)^* \tau] X_1 .$$

Resolvendo a última equação em X_1, obtemos:

$$|\mathscr{A}| = X_1 = [\sigma(\sigma \cup \sigma(\sigma \cup \tau)^* \tau)]^* \sigma\sigma(\sigma \cup \tau)^* .$$

Em geral, existem várias expressões denotando o mesmo subconjunto racional de Σ^*. Algumas podem ser obtidas mudando-se a enumeração dos estados do autômato e aplicando-se o algoritmo do

Teorema de Kleene. No caso do autômato da Figura 5, deixamos ao leitor a verificação de que

$$|\mathscr{A}| = \sigma\sigma(\sigma \cup \tau)^*.$$

5. Monóide de um autômato e o monóide sintático

No que segue, associamos monóides $M_{\mathscr{A}}$ e M_A a cada autômato $\mathscr{A}$ e a cada subconjunto A de Σ^*. Estes monóides têm um papel fundamental, pois permitem tratar vários problemas em termos algébricos, utilizando a teoria dos semigrupos. No capítulo seguinte resolvemos um problema com estas características.

Seja $\mathscr{A} = (Q, \Sigma, \alpha, I, F)$ um autômato. Então

$$\alpha : \Sigma^* \to Rel\ Q$$

é um morfismo. A imagem $\Sigma^*\alpha$ de Σ^* por α é um submonóide de $Rel\ Q$, chamado de *monóide de* $\mathscr{A}$, e denotado por $M_{\mathscr{A}}$. Assim, α pode também ser visto como um epimorfismo

$$\alpha : \Sigma^* \to M_{\mathscr{A}}.$$

Como Q é um conjunto finito, $Rel\ Q$ é finito também, portanto o monóide $M_{\mathscr{A}}$ de um autômato $\mathscr{A}$ é um monóide finito.

No caso particular em que $\mathscr{A}$ é determinístico, o monóide $M_{\mathscr{A}}$ de $\mathscr{A}$ é um submonóide de $Fun\ Q$.

Seja A um subconjunto de Σ^*. Diz-se que palavras s_1 e s_2 em Σ^* são *congruentes módulo* A e escreve-se

$$s_1 \equiv s_2 \ (mod\ A)$$

sse

$$\text{para todo } u, v \text{ em } \Sigma^*\ us_1v \in A \text{ sse } us_2v \in A. \tag{1}$$

Em outras palavras, s_1 e s_2 são congruentes módulo A sse em qualquer contexto (u, v), s_1 e s_2 podem ser trocados um pelo outro, sem alterar a pertinência a A das palavras em questão. Está claro que a relação acima é de equivalência sobre Σ^*. Mais ainda, ela é uma relação de congruência, pois se $s_1 \equiv s_2\ (mod\ A)$ e $t \in \Sigma^*$, então (1) implica que

$$\text{para todo } u, v \text{ em } \Sigma^*\ us_1tv \in A \text{ sse } us_2tv \in A,$$

isto é, $s_1 t \equiv s_2 t \ (mod\ A)$. Analogamente $ts_1 \equiv ts_2 \ (mod\ A)$. O monóide quociente de Σ^* por $\equiv \ (mod\ A)$ é chamado de *monóide sintático* de A e é denotado por M_A. A projeção canônica

$$\gamma : \Sigma^* \to M_A$$

é chamada de *morfismo sintático* de A.

Observamos que se $s_1\gamma = s_2\gamma$, isto é $s_1 \equiv s_2 \ (mod\ A)$, então $s_1 \in A$ sse $s_2 \in A$. Assim,

$$\text{para todo } s \text{ em } \Sigma^* \ s \in A \text{ sse } s\gamma\gamma^{-1} \subseteq A. \tag{2}$$

Note que os conceitos acima aplicam-se a qualquer subconjunto de Σ^*. A proposição seguinte, que caracteriza *Rec* Σ, é o elo de ligação entre as teorias dos autômatos e dos semigrupos.

PROPOSIÇÃO 9. *Um subconjunto A de Σ^* é reconhecível sse o monóide sintático M_A de A é finito.*

Demonstração. Seja $\mathcal{A} = (Q, \Sigma, \alpha, I, F)$ um autômato que reconhece o conjunto reconhecível A. Consideremos o morfismo $\alpha : \Sigma^* \to M_{\mathcal{A}}$ e seja $\gamma : \Sigma^* \to M_A$ o morfismo sintático de A (veja Figura 6).

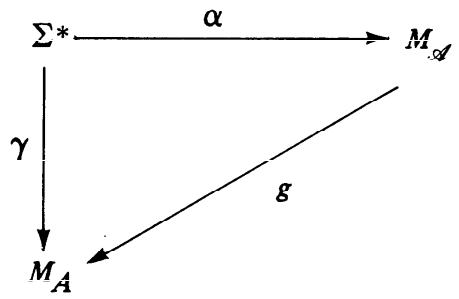


Figura 6.

Mostramos inicialmente que $\alpha\alpha^{-1} \subseteq \gamma\gamma^{-1}$. Para tanto, basta provar que para todo $s_1, s_2 \in \Sigma^*$,

$$s_1\alpha = s_2\alpha \text{ implica que } s_1 \equiv s_2 \ (mod\ A).$$

De fato, sejam u, v em Σ^*. Como $s_1\alpha = s_2\alpha$, resulta que

$$(us_1v)\alpha = (u\alpha)(s_1\alpha)(v\alpha) = (u\alpha)(s_2\alpha)(v\alpha) = (us_2v)\alpha.$$

Em particular:

$$I((us_1v)\alpha) = I((us_2v)\alpha),$$

logo

$$us_1v \in A \text{ sse } us_2v \in A.$$

Assim,

$$s_1 \equiv s_2 \ (mod\ A).$$

Pela Proposição I.3 existe um epimorfismo $g : M_{\mathscr{A}} \to M_A$, já que γ é um epimorfismo. Como $M_{\mathscr{A}}$ é finito, M_A é finito também.

Reciprocamente, suponhamos que M_A é finito, e seja

$$\gamma : \Sigma^* \to M_A$$

o morfismo sintático de A. Seja $\mathscr{A} = (M_A, \Sigma, \alpha, 1\gamma, F)$ um autômato determinístico, onde

$$F = \{m \in M_A \mid m\gamma^{-1} \subseteq A\},$$

e para $m \in M_A$ e $\sigma \in \Sigma$, a imagem de m por $\sigma\alpha$ é dado por

$$m(\sigma\alpha) = m \cdot (\sigma\gamma).$$

(Note que no lado direito, o ponto indica multiplicação em M_A.) É imediato verificar, que para todo s em Σ^*

$$m(s\alpha) = m \cdot (s\gamma). \tag{3}$$

Usando (3) e (2), resulta que:

$$s \in |\mathscr{A}| \text{ sse } (1\gamma)(s\alpha) = (1\gamma) \cdot (s\gamma) = s\gamma \in F \text{ sse } s\gamma\gamma^{-1} \subseteq A \text{ sse } s \in A.$$

Assim, $|\mathscr{A}| = A$ e A é reconhecível. ■

Terminamos esta seção com uma observação. Não apresentamos neste livro ferramentas suficientes para computar o monóide sintático de um subconjunto reconhecível A de Σ^*. Isto pode ser feito, utilizando o "autômato reduzido" que reconhece A.

EXERCÍCIOS

1. Dados conjuntos finitos Q, Σ, E, I e F, tais que $I, F \subseteq Q$ e $E \subseteq Q \times \Sigma \times Q$, mostre que existe um único autômato $\mathscr{A} = (Q, \Sigma, \alpha, I, F)$ cujo conjunto de arestas é E.
2. Complete as demonstrações das Proposições 4 e 5.

3. Dê um exemplo de um autômato $\mathscr{A} = (Q, \Sigma, \alpha, I, F)$, tal que $|\mathscr{A}| \neq |\mathscr{B}|$, onde $\mathscr{B} = (Q, \Sigma, \alpha, I, Q \backslash F)$.

4. Dado um autômato $\mathscr{A} = (Q, \Sigma, \alpha, I, F)$, seja $[\mathscr{A}] = \{s \in \Sigma^* \mid I(s\alpha) \subseteq F\}$. Mostre que para todo alfabeto finito Σ, $Rec\ \Sigma = \{A \subseteq \Sigma^* \mid A = [\mathscr{A}]$ para algum autômato $\mathscr{A}\}$.

5. Sejam $\mathscr{A}_j = (Q_j, \Sigma, \alpha_j, i_j, F_j)$ autômatos determinísticos para j = 1, 2. O *produto direto* $\mathscr{A}_1 \times \mathscr{A}_2$ de $\mathscr{A}_1$ e $\mathscr{A}_2$ é o autômato

$$\mathscr{A}_1 \times \mathscr{A}_2 = (Q_1 \times Q_2, \Sigma, \alpha, (i_1, i_2), F_1 \times F_2),$$

onde para todo par (q_1, q_2) em $Q_1 \times Q_2$,

$$(q_1, q_2)(\sigma\alpha) = (q_1(\sigma\alpha_1), q_2(\sigma\alpha_2)).$$

Mostre que $|\mathscr{A}_1 \times \mathscr{A}_2| = |\mathscr{A}_1| \cap |\mathscr{A}_2|$. Mostre que substituindo os estados finais $F_1 \times F_2$ de $\mathscr{A}_1 \times \mathscr{A}_2$ por conjuntos convenientes, obtém-se autômatos com comportamento

$$|\mathscr{A}_1| \cup |\mathscr{A}_2|, |\mathscr{A}_1| \backslash |\mathscr{A}_2| \text{ e } |\mathscr{A}_1| \oplus |\mathscr{A}_2|.$$

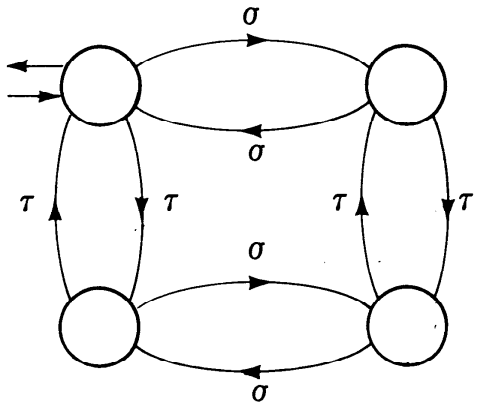
6. Generalize a construção do produto direto para autômatos em geral, de forma que a igualdade $|\mathscr{A}_1 \times \mathscr{A}_2| = |\mathscr{A}_1| \cap |\mathscr{A}_2|$ continue válida. Mostre que substituindo convenientemente os estados finais de $\mathscr{A}_1 \times \mathscr{A}_2$ obtém-se um autômato $\mathscr{B}$, tal que $|\mathscr{B}| \subseteq |\mathscr{A}_1| \cup |\mathscr{A}_2|$. Dê um exemplo de autômatos $\mathscr{A}_1$ e $\mathscr{A}_2$, tais que tanto $|\mathscr{A}_1|$ como $|\mathscr{A}_2|$ contém palavras não vazias, no entanto o autômato $\mathscr{A}_1 \times \mathscr{A}_2$ não tem arestas.

7. Seja $h : \Sigma^* \to \Gamma^*$ um morfismo de monóides livres. Mostre que se $A \subseteq \Sigma^*$ é reconhecível, então Ah é reconhecível também. Mostre que se $A \subseteq \Gamma^*$ é reconhecível, então Ah^{-1} é reconhecível também.

8. Sejam $A_1, A_2 \subseteq \Sigma^*$. O *embaralhamento* de A_1 e A_2 é definido por $A_1 \sqcup\!\sqcup A_2 = \{s_1 t_1 s_2 t_2 \ldots s_n t_n \mid n \geq 1, s_1 s_2 \ldots s_n \in A_1 \text{ e } t_1 t_2 \ldots t_n \in A_2\}$. Assim, por exemplo, $(\sigma\tau) \sqcup\!\sqcup \Sigma^* = \Sigma^* \sigma \Sigma^* \tau \Sigma^*$. Mostre que $A_1 \sqcup\!\sqcup A_2 = A_2 \sqcup\!\sqcup A_1$. Mostre também que *Rec* Σ é fechado sob embaralhamento.

9. Seja A um subconjunto reconhecível de Σ^*. Mostre que $\Sigma^{*-1}A$, $A\Sigma^{*-1}$ e $(\Sigma^{*-1}A)\Sigma^{*-1}$ são reconhecíveis também (veja Exercício I.66).

10. Ache uma expressão racional, tão "simples" quanto possível, para o comportamento do autômato da figura seguinte:



11. Mostre que se $1 \in E \subseteq \Sigma^*$, então X é uma solução da equação $X = T \cup EX$ sse existe $V \subseteq \Sigma^*$, tal que $X = E^*(T \cup V)$.

12. (*McNaughton e Yamada*) – Baseando-se nas idéias que seguem, dê uma nova demonstração da inclusão $Rec\ \Sigma \subseteq Rac\ \Sigma$. Seja $\mathscr{A} = (Q, \Sigma, \alpha, I, F)$ um autômato com n estados. Fixamos uma enumeração arbitrária $q_1, q_2, \ldots, q_n$ de Q. Para $0 \leq k \leq n$, dizemos que $c : q_i \xrightarrow{s} q_j$ é um k-passeio se para toda fatoração $q_i \xrightarrow{s_1} q_\ell \xrightarrow{s_2} q_j$ de c, $s_1 \neq 1 \neq s_2$ implica que $\ell \leq k$. Sejam, para $1 \leq i, j \leq n$ e $0 \leq k \leq n$,

$$A_{ij}^{(k)} = \{s \in \Sigma^* \mid \text{existe um } k\text{-passeio } q_i \xrightarrow{s} q_j\}.$$

Mostre que para $1 \leq i, j \leq n$ e $0 < k \leq n$,

$$A_{ij}^{(0)} = \{\sigma \in \Sigma \mid q_j \in q_i(\sigma\alpha)\}$$

e $$A_{ij}^{(k)} = A_{ij}^{(k-1)} \cup A_{ik}^{(k-1)} (A_{kk}^{(k-1)})^* A_{kj}^{(k-1)}.$$

Conclua que $A_{ij}^{(k)}$ é um subconjunto racional de Σ^*, para $1 \leq i, j \leq n$ e $0 \leq k \leq n$. Como

$$|\mathscr{A}| = \bigcup_{\substack{q_i \in I \\ q_j \in F}} A_{ij}^{(n)},$$

conclua que $|\mathscr{A}|$ é racional.

13. Mostre que um subconjunto A de Σ^* é reconhecível sse existe um monóide finito M e um morfismo $\beta : \Sigma^* \to M$, tais que $A\beta\beta^{-1} = A$.

14. Calcule o monóide sintático de $(\sigma\tau)^*$ e de $(\sigma\tau \cup \tau\sigma)^*$.

15. Calcule o monóide sintático de $A = \{\sigma^n\tau^n \mid n \geq 0\}$. Conclua que A não é racional. Confronte com o Exercício 24.

16. Sejam $\mathscr{A}_i = (Q_i, \Sigma_i, \alpha_i, I_i, F_i)$ autômatos para $i = 1, 2$. Dizemos que $\mathscr{A}_1$ é um *subautômato* de $\mathscr{A}_2$ se

$$Q_1 \subseteq Q_2,\ \Sigma_1 \subseteq \Sigma_2,\ I_1 \subseteq I_2,\ F_1 \subseteq F_2$$

e para todo σ em Σ_1, $\sigma\alpha_1 \subseteq \sigma\alpha_2$. Mostre que se $\mathscr{A}_1$ for um subautômato de $\mathscr{A}_2$, então $|\mathscr{A}_1| \subseteq |\mathscr{A}_2|$.

17. Um autômato $\mathscr{A} = (Q, \Sigma, \alpha, I, F)$ é *conexo* se para todo q em Q existe um s em Σ^*, tal que $q \in I(s\alpha)$. Mostre que todo autômato $\mathscr{A} = (Q, \Sigma, \alpha, I, F)$ tem um único subautômato conexo maximal. Este subautômato é chamado de *parte conexa* de $\mathscr{A}$. Mostre que a parte conexa de $\mathscr{A}$ é o autômato $\mathscr{A}_c = (Q_c, \Sigma, \alpha_c, I, F \cap Q_c)$, onde

$$Q_c = \{q \in Q \mid \text{existe } s \in \Sigma^* \text{ tal que } q \in I(s\alpha)\}$$

e para todo σ em Σ, $\sigma\alpha_c = \sigma\alpha \cap (Q_c \times Q_c)$. Mostre ainda que $|\mathscr{A}| = |\mathscr{A}_c|$.

18. Baseando-se nas idéias do Teorema C.I.1 dê um algoritmo para achar a parte conexa de um autômato dado.

19. Um autômato $\mathscr{A} = (Q, \Sigma, \alpha, I, F)$ é *fortemente conexo* se para todo par (q_1, q_2) de estados de $\mathscr{A}$ existe s em Σ^*, tal que $(q_1, q_2) \in s\alpha$. Mostre que todo autômato determinístico $\mathscr{A} = (Q, \Sigma, \alpha, i, F)$ contém um subautômato fortemente conexo $\mathscr{A}' = (Q', \Sigma, \alpha', I', F')$, no qual $\sigma\alpha'$ é não vazio para todo σ em Σ.

20. Sejam $\mathscr{A}_1$ e $\mathscr{A}_2$ autômatos determinísticos, e sejam n_1, n_2 e n o número de estados da parte conexa de $\mathscr{A}_1$, $\mathscr{A}_2$ e $\mathscr{A}_1 \times \mathscr{A}_2$, respectivamente. Mostre que $n \geq max\{n_1, n_2\}$.

21. Mostre que todo subconjunto reconhecível de Σ^* é o comportamento de infinitos autômatos conexos.
Sugestão: Use o Exercício 20.

22. Seja $\mathscr{A}$ um autômato com n estados. Mostre que toda palavra $s \in |\mathscr{A}|$ de comprimento pelo menos n tem uma fatoração $s = uvw$, tal que $1 \leq |v| \leq n$, e $uv^*w \subseteq |\mathscr{A}|$.

23. Usando o Exercício 22, mostre que o comportamento de um autômato dado $\mathscr{A}$, com n estados, é infinito sse existe uma palavra s em $|\mathscr{A}|$ tal que $n \leq |s| < 2n$.

24. Para $\sigma \in \Sigma$ e $s \in \Sigma^*$, $|s|_\sigma$ denota o número de ocorrências da letra σ na palavra s.
Usando o Exercício 22, mostre que os subconjuntos seguintes de Σ^* não são reconhecíveis ($\Sigma = \{\sigma, \tau\}$):

(i) $\{\sigma^n\tau^n \mid n \geq 0\}$,
(ii) $\{s \in \Sigma^* \mid |s|_\sigma = |s|_\tau\}$,

(iii) $\{s \in \Sigma^* \mid s = s\rho\}$,
(iv) $\{\sigma^{n^2} \mid n \geq 0\}$,
(v) $\{s^2 \mid s \in \Sigma^*\}$.

25. Um autômato $\mathscr{A} = (Q, \Sigma, \alpha, I, F)$ é *normalizado* se I e F são unitários, $I \neq F$ e nenhuma aresta de $\mathscr{A}$ tem término em I, ou origem em F. Mostre que se A for um conjunto reconhecível, então $A\backslash 1$ é o comportamento de algum autômato normalizado. Dados autômatos normalizados $\mathscr{A}_1$ e $\mathscr{A}_2$, construa autômatos com comportamento $|\mathscr{A}_1| \cup |\mathscr{A}_2|$, $|\mathscr{A}_1|\,|\mathscr{A}_2|$ e $|\mathscr{A}_1|^*$.

26. Mostre que um subconjunto A de Σ^* é reconhecível sse o conjunto

$$\{s^{-1}A \mid s \in \Sigma^*\}$$

é finito (veja o Exercício I.66). Mostre mais uma vez que o subconjunto $\{\sigma^n\tau^n \mid n \geq 0\}$ de $\{\sigma, \tau\}^*$ não é reconhecível.

27. Seja B o semianel booleano, isto é B tem suporte $B = \{0, 1\}$ e é munido das operações $+$ e $\cdot$, onde

$$0 + 0 = 1 \cdot 0 = 0 \cdot 1 = 0 \cdot 0 = 0,$$

e

$$1 + 1 = 1 + 0 = 0 + 1 = 1 \cdot 1 = 1.$$

Para um natural $n \geq 1$, seja M_nB o monóide cujo suporte é o conjunto das matrizes $n \times n$ com coeficientes em B, munido da operação de multiplicação de matrizes. Mostre que os monóides $\mathbf{Rel}\,\mathbf{n}$ e M_nB são isomorfos. Conclua que o monóide $M_{\mathscr{A}}$ de um autômato $\mathscr{A}$ é isomorfo a um submonóide de M_nB, onde n é a cardinalidade do conjunto de estados de A. (Confronte com o Exercício B.III.24.)

28. Um autômato $\mathscr{A} = (Q, \Sigma, \alpha, I, F)$ é *não-ambíguo* se para todo s em Σ^* $\mathscr{A}$ tem no máximo um passeio com origem em I, término em F e rótulo s. Mostre que existe um algoritmo para decidir se um autômato dado é ambíguo ou não.

29. Mostre que para todo subconjunto A de σ^*, A^* é reconhecível. Sugestão: Se n e m são naturais, tais que $0 \leq m < n$ e σ^n pertence a A, então $A \cap \{\sigma^p \mid p \equiv m \ (mod\ n)\}$ é reconhecível.

30. Seja $\mathbb{Z}$ o conjunto dos inteiros, munido da operação de soma. Seja ainda $h : \Sigma^* \longrightarrow \mathbb{Z}$ a extensão a um morfismo da função $h_1 : \Sigma \to \mathbb{Z}$, onde $\Sigma = \{\sigma, \tau\}$, $\sigma h_1 = 1$ e $\tau h_1 = -1$. Mostre que

$sh = |s|_\sigma - |s|_\tau$ (veja Exercício 24 para a notação). Para um natural n definimos

$$D_n = \{s \in \Sigma^* \mid sh = 0 \text{ e para toda fatoração } s = s_1 s_2 \text{ de } s,\ 0 \leq s_1 h \leq n\}.$$

Colocamos também

$$D = \bigcup_{n \in \mathbb{N}} D_n.$$

Mostre que para todo n D_n é reconhecível, mas D não é reconhecível.

31. Mostre que *Rec* Σ é fechado sob reversão (veja Exercício I.67).
32. Dê uma construção que associa a um autômato $\mathscr{A}$ uma máquina de Turing não determinística M que não utiliza as suas fitas de trabalho e que reconhece o comportamento de $\mathscr{A}$ (veja Capítulo B.I).

NOTAS BIBLIOGRÁFICAS

Autômatos finitos foram introduzidos na segunda metade da década de 1950. Os trabalhos pioneiros na área são de Kleene, Moore e Myhill. Alguns destes estão reimpressos no livro de Moore [85]. Entre os trabalhos da época recomendamos em especial a leitura da exposição muito lúcida de Rabin e Scott [99]. Posteriormente, a teoria foi substancialmente enriquecida por vários autores, entre os quais Brzozowski, McNaughton, Rhodes e Schützenberger.

O Teorema de Kleene apareceu originalmente em [59]. A Proposição 9 aparece no trabalho de Rabin e Scott, onde é creditado a J. R. Myhill.

Indicamos a seguir alguns livros sobre a Teoria dos Autômatos. Um tratamento mais elementar pode ser encontrado nos livros de Ginzburg [36], Harrison [51], [52], Minsky [83], Salomaa [104] e Veloso [124]. Um estudo completo e profundo da teoria encontra-se nos livros de Eilenberg [25], [26]. Aspectos combinatórios relativos à Teoria dos Autômatos são abordados no texto de Schützenberger [109].

CAPÍTULO III

CONJUNTOS INTEIROS E O TEOREMA DE SCHÜTZENBERGER

1. Conjuntos inteiros e monóides aperiódicos

Um subconjunto $\mathscr{F}$ de $\mathfrak{p}(\Sigma^*)$ é *integralmente fechado* se $\varnothing$ e 1 pertencem a $\mathscr{F}$ e se para todo A e A' em $\mathscr{F}$,

$$A \cup A',\ AA' \text{ e } \bar{A}$$

também pertencem a $\mathscr{F}$. É fácil ver que a intersecção de subconjuntos integralmente fechados de $\mathfrak{p}(\Sigma^*)$ também é integralmente fechada. Denotamos por

$$Int\ \Sigma$$

o menor subconjunto integralmente fechado de $\mathfrak{p}(\Sigma^*)$ que contém os conjuntos unitários σ, para todo σ em Σ. Os elementos de $Int\ \Sigma$ são chamados de *subconjuntos inteiros* de Σ^*.

Como $Int\ \Sigma$ é fechado sob união e complementação, ele é uma álgebra booleana de subconjuntos de Σ^* e é fechado portanto sob intersecção. Alguns exemplos de subconjuntos inteiros de Σ^* ($\Sigma = \{\sigma, \tau\}$), são:

$$\begin{aligned}
I &= \Sigma^* = \bar{\varnothing}, \\
I_1 &= (\sigma\tau)^* = 1 \cup (\sigma I \cap I\tau \cap \overline{I\sigma\sigma I} \cap \overline{I\tau\tau I}), \\
I_2 &= \tau(\sigma\tau)^* = \tau I_1, \\
I_3 &= \sigma(\tau\sigma)^* = I_1\sigma, \\
I_4 &= (\sigma\tau \cup \tau\sigma)^* = \overline{\overline{I\sigma I_2 \sigma I} \cup \overline{I\tau I_3 \tau I}}.
\end{aligned}$$

A verificação destas igualdades é deixada para o leitor, como um exercício interessante. Observamos que esta verificação pode ser feita algoritmicamente, embora neste livro não apresentemos os meios para isto.

Em vista do Corolário II.3, é imediato que

$$Int\ \Sigma \subseteq Rac\ \Sigma.$$

Um monóide M é *aperiódico* sse para todo m em M existe um natural n, tal que $m^n = m^{n+1}$. Mencionamos aqui que um monóide finito M é aperiódico sse todo grupo em M for trivial.

Os subconjuntos inteiros de Σ^* aparecem na Teoria dos Autômatos de diversas maneiras e possuem várias caracterizações. Aqui limitamo-nos a apresentar a seguinte caracterização:

TEOREMA 1 (Schützenberger). *Seja Σ um alfabeto finito. Um subconjunto A de Σ^* é inteiro sse o monóide sintático M_A de A é finito e aperiódico.*

Uma conseqüência do Teorema de Schützenberger é que existe um algoritmo para decidir se um conjunto racional dado A é inteiro ou não. Para tanto, basta construir o monóide sintático M_A de A e verificar se este é aperiódico ou não. Mais ainda, caso o monóide M_A de A seja aperiódico, seguindo-se a demonstração da parte "se" do teorema, podemos efetivamente construir uma expressão para A a partir dos conjuntos unitários σ, utilizando-se um número finito de vezes as operações de união, concatenação e complementação. Na Seção 4 apresentamos um exemplo desta construção.

2. Algumas propriedades de monóides aperiódicos

As propriedades que seguem serão utilizadas na demonstração do Teorema 1.

PROPOSIÇÃO 1. *Seja $\beta\colon M \to M_1$ um epimorfismo de monóides. Se M é aperiódico então M_1 também o é.*

Demonstração. Seja a em M_1. Como β é sobrejetora, $a = b\beta$ para algum b em M. Como M é aperiódico, $b^n = b^{n+1}$ para algum n, logo $a^n = (b\beta)^n = b^n\beta = b^{n+1}\beta = (b\beta)^{n+1} = a^{n+1}$. ■

PROPOSIÇÃO 2 (Lei de cancelamento para monóides aperiódicos). *Sejam a, b, e m elementos do monóide aperiódico M. Se $m = amb$ então $m = am = mb$.*

Demonstração. Seja n, tal que $a^n = a^{n+1}$. De $m = amb$, resulta que $m = a^n m b^n$, logo

$$m = a^n m b^n = a^{n+1} m b^n = a(a^n m b^n) = am.$$

Analogamente, $m = mb$. ■

Seja M um monóide, e J um subconjunto não vazio de M.

J é um *ideal à esquerda* sse $MJ = J$.
J é um *ideal à direita* sse $JM = J$.
J é um *ideal bilateral* sse $MJM = J$.

Os exercícios dão algumas propriedades de ideais. Em particular, se $m \in M$, então Mm, mM e MmM são ideais à esquerda, à direita e bilaterais, respectivamente, chamados de *ideais principais gerados por m*. Um elemento 0 em M é um *zero* sse $M0M = 0$, isto é, $\{0\} \subseteq M$ é um ideal bilateral. É fácil ver que 0 em M é um zero sse para todo m em M,

$$m0 = 0m = 0.$$

Para um subconjunto X de M, definimos

$$W_X = \{m \in M \mid MmM \cap X = \varnothing\}.$$

PROPOSIÇÃO 3. *Para todo subconjunto X de um monóide M, W_X é vazio ou é um ideal bilateral.*

Demonstração. Basta mostrar que $MW_XM = W_X$. De fato,

$$W_X = 1W_X1 \subseteq MW_XM.$$

Por outro lado, para mostrar que $MW_XM \subseteq W_X$, é suficiente provar que se $a, b \in M$ e $m \in W_X$, então $amb \in W_X$. Temos:

$$M(amb)M = (Ma)m(bM) \subseteq MmM.$$

Assim, se $MmM \cap X = \varnothing$, então $M(amb)M \cap X = \varnothing$, isto é, $amb \in W_X$. ■

PROPOSIÇÃO 4. *Seja m um elemento do monóide aperiódico M. Então*

$$m = (Mm \cap mM) \backslash W_m.$$

Demonstração. Está claro que $m \in Mm \cap mM$. Por outro lado, $m \in MmM$, isto é $MmM \cap m \neq \varnothing$. Logo $m \notin W_m$.

Assim,

$$m \subseteq (Mm \cap mM) \backslash W_m.$$

Seja agora m' um outro elemento de $(Mm \cap mM) \backslash W_m$. Como $m' \in Mm$ e $m' \in mM$, existem a e b em M, tais que

$$m' = am \quad \text{e} \quad m' = mb. \tag{1}$$

Como $m' \notin W_m$, resulta que $Mm'M \cap m \neq \varnothing$. Isto é, existem c e d em M, tais que

$$m = cm'd. \tag{2}$$

De (1) e (2), resulta que

$$m = cm'd = camd,$$

assim, pela Proposição 2,

$$m = cam = cm'. \tag{3}$$

De (1) e (3), resulta que

$$m = cm' = cmb,$$

logo, pela Proposição 2,

$$m = mb = m'.$$

Isto completa a demonstração. ■

3. Demonstração do Teorema 1

Começamos com a parte "se" do Teorema 1, que afirma que se o monóide sintático M_A de A é finito e aperiódico, então A é um subconjunto inteiro de Σ^*. De fato, mostramos a afirmação mais forte:

PROPOSIÇÃO 5. *Sejam Σ um alfabeto finito, M um monóide finito aperiódico e $\gamma: \Sigma^* \to M$ um epimorfismo. Para todo subconjunto X de M, $X\gamma^{-1}$ é um subconjunto inteiro de Σ^*.*

Demonstração. Procedemos por indução em *card* M. Se *card* $M = 1$, então $M = 1$ e para os dois subconjuntos de M, temos:

$$\varnothing\gamma^{-1} = \varnothing \quad \text{e} \quad 1\gamma^{-1} = \Sigma^* = \overline{\varnothing}.$$

Ambos os subconjuntos de Σ^* são inteiros, estabelecendo a base da indução.

No que segue, fixamos M e γ como no enunciado, e adotamos a seguinte hipótese de indução:

HI: Se $\gamma' : \Sigma^* \to M'$ é um epimorfismo, M' é finito e aperiódico, *card* $M' <$ *card* M e $X \subseteq M'$, então $X\gamma'^{-1}$ é inteiro.

Os Lemas 1, 2, 3 e 3', a seguir, serão demonstrados usando *HI*.

LEMA 1. *Seja J um ideal bilateral de M e seja $m \in M \backslash J$. Se card $J \geq 2$, então $m\gamma^{-1}$ e $J\gamma^{-1}$ são inteiros.*

LEMA 2. *Para todo m em M, $(MmM)\gamma^{-1}$ é inteiro.*

LEMA 3. *Para todo m em M, $(mM)\gamma^{-1}$ é inteiro.*

LEMA 3'. *Para todo m em M, $(Mm)\gamma^{-1}$ é inteiro.*

Adiamos a demonstração dos lemas e prosseguimos com a prova da Proposição 5. Seja $X \subseteq M$. Como X é finito e

$$X\gamma^{-1} = \bigcup_{m \in X} m\gamma^{-1},$$

basta mostrar que para todo $m \in M$, $m\gamma^{-1}$ é inteiro. De fato, pela Proposição 4,

$$m\gamma^{-1} = [(Mm \cap mM) \backslash W_m]\gamma^{-1} = [(Mm)\gamma^{-1} \cap (mM)\gamma^{-1}] \backslash (W_m\gamma^{-1}). \quad (1)$$

Pelos Lemas 3' e 3 $(Mm)\gamma^{-1}$ e $(mM)\gamma^{-1}$ são inteiros. Por outro lado, pela Proposição 3, W_m é vazio ou é um ideal bilateral. No segundo caso, pelo Exercício 3,

$$W_m = \bigcup_{m' \in W_m} Mm'M,$$

logo,

$$W_m\gamma^{-1} = \bigcup_{m' \in W_m} (Mm'M)\gamma^{-1}.$$

Como W_m é finito e pelo Lema 2 $(Mm'M)\gamma^{-1}$ é inteiro para cada m' em M, resulta que nos dois casos $W_m\gamma^{-1}$ é inteiro. Pela fórmula (1), $m\gamma^{-1}$ é um subconjunto inteiro de Σ^* para todo m em M. ■

Demonstração do Lema 1. Seja $\sim$ a relação de equivalência sobre M, dada por:

$$m_1 \sim m_2 \text{ sse } m_1 = m_2 \text{ ou } \{m_1, m_2\} \subseteq J.$$

Mostramos que $\sim$ é uma congruência. De fato, sejam $m_1, m_2, m_3 \in M$, tais que $m_1 \sim m_2$. Se $m_1 = m_2$ então $m_1m_3 = m_2m_3$. Se $m_1, m_2 \in J$, então $m_1m_3, m_2m_3 \in J$, já que J é um ideal bilateral. Assim $m_1m_3 \sim$ $\sim m_2m_3$. Analogamente $m_3m_1 \sim m_3m_2$. Temos então

$$card\ M/\sim\ = (card\ M) - (card\ J) + 1 < card\ M,$$

pois $card\ J \geq 2$. Por outro lado, sendo $\beta: M \to M/\sim$ a projeção canônica, $\gamma\beta : \Sigma^* \to M/\sim$ é um epimorfismo. Agora, para $m \notin J, m = m\beta\beta^{-1}$ e portanto

$$m\gamma^{-1} = m\beta\beta^{-1}\gamma^{-1} = (m\beta)\,(\gamma\beta)^{-1}.$$

Também $J = J\beta\beta^{-1}$, logo $J\gamma^{-1} = (J\beta)\,(\gamma\beta)^{-1}$. O resultado segue de *HI*, já que $M/\sim$ é aperiódico pela Proposição 1. ■

Demonstração do Lema 2. Seja $A = (MmM)\gamma^{-1}$. Já que MmM é um ideal bilateral, resulta que

$$A = \Sigma^* A \Sigma^*. \tag{2}$$

Se $MmM = M$, então $A = \Sigma^* = \varnothing$ que é inteiro. Caso contrário

$$1\gamma \notin MmM \quad \text{e} \quad 1 \notin A. \tag{3}$$

Seja

$$B = A \backslash (\Sigma^* \Sigma A \Sigma^* \cup \Sigma^* A \Sigma \Sigma^*).$$

Observamos que

$$B = \{s \in A \mid \text{nenhum segmento } t \text{ de } s,\ t \neq s, \text{ pertence a } A\}.$$

Seja

$$\Sigma_0 = \Sigma \cap B,$$

e consideremos

$$T = \{(\sigma, m', \tau) \mid \sigma, \tau \in \Sigma,\ m' \in M \text{ e } \sigma(m'\gamma^{-1})\tau \cap B \neq \varnothing\} =$$
$$= \{(\sigma, t\gamma, \tau) \mid \sigma, \tau \in \Sigma,\ t \in \Sigma^* \text{ e } \sigma t \tau \in B\}.$$

Vamos mostrar que

$$A = \Sigma^*\Sigma_0\Sigma^* \cup \left(\bigcup_{(\sigma,m',\tau)\in T} \Sigma^*\sigma(m'\gamma^{-1})\tau\Sigma^* \right) \tag{4}$$

De fato, seja $s \in A$ e seja u um segmento de comprimento mínimo de s que também pertença a A. Então $u \in B$, e de (3), $u \neq 1$. Se $|u| = 1$, então $u \in \Sigma_0$ e $s \in \Sigma^*\Sigma_0\Sigma^*$. Se $|u| > 1$, sejam σ, t e τ, tais que $u = \sigma t\tau$. Vem que $(\sigma, t\gamma, \tau) \in T$ e

$$s \in \Sigma^*\sigma(t\gamma\gamma^{-1})\tau\Sigma^*.$$

Reciprocamente, $\Sigma_0 \subseteq B \subseteq A$, logo, de (2),

$$\Sigma^*\Sigma_0\Sigma^* \subseteq \Sigma^*A\Sigma^* = A.$$

Por outro lado, sejam $(\sigma, m', \tau) \in T$ e $t \in m'\gamma^{-1}$, tais que $\sigma t\tau \in B \subseteq A$. Se $v \in m'\gamma^{-1}$ então $(\sigma t\tau)\gamma = (\sigma v\tau)\gamma$, logo $\sigma v\tau \in A$. De (2),

$$\Sigma^*\sigma(t\gamma\gamma^{-1})\tau\Sigma^* \subseteq A.$$

Isto prova (4).

Mostramos agora, que para todo m' em M, tal que $(\sigma, m', \tau) \in T$ para algum $\sigma, \tau \in \Sigma$,

$$card\ W_{m'} \geq 2. \tag{5}$$

De fato, seja $(\sigma, t\gamma, \tau) \in T$ com $\sigma t\tau \in B$. Sejam

$$m' = t\gamma,\ m_1 = \sigma\gamma \text{ e } m_2 = \tau\gamma.$$

Como $\sigma t\tau \in B \subseteq A = (MmM)\gamma^{-1}$, vem que $(\sigma t\tau)\gamma \in MmM$, isto é,

$$m_1m'm_2 \in MmM. \tag{6}$$

Por outro lado, como $\sigma t\tau \in B$, resulta da construção de B, que $\sigma t, t\tau \notin A$, logo

$$m_1m' \notin MmM \quad \text{e} \quad m'm_2 \notin MmM. \tag{7}$$

Suponhamos agora que $m_1m' \notin W_{m'}$. Resulta que $m' = am_1m'b$ para algum $a, b \in M$. Pela Proposição 2, $m' = am_1m'$, logo $m'm_2 = am_1m'm_2$. Ora, de (6), $m_1m'm_2 \in MmM$, logo, sendo MmM um ideal bilateral, $m'm_2 \in MmM$. Isto contradiz (7), portanto

$$m_1m' \in W_{m'}. \tag{8}$$

Pela Proposição 3, $W_{m'}$ é um ideal bilateral, logo

$$m_1 m' m_2 \in W_{m'}. \tag{9}$$

Agora (6) e (7) garantem que $m_1 m' \neq m_1 m' m_2$, e (5) segue de (8) e (9).

Finalmente, está claro que $m' \notin W_{m'}$. Assim, segue de (5) e do Lema 1, que $m'\gamma^{-1}$ é inteiro, para cada m' em M, tal que $(\sigma, m', \tau) \in T$ para algum σ e τ em Σ. Como T é finito, segue de (4), que $A = (MmM)\gamma^{-1}$ é um subconjunto inteiro de Σ^*. ■

Demonstração do Lema 3. Seja $A = (mM)\gamma^{-1}$. Já que mM é um ideal à direita, resulta que

$$A = A\Sigma^*. \tag{10}$$

Seja J o maior ideal bilateral contido em mM, ou $\varnothing$ se mM não contém ideais bilaterais. A existência de J é garantida, pois se J_1 e J_2 são ideais bilaterais contidos em mM, então $J_1 \cup J_2$ também é um tal ideal. Seja

$$B = A \backslash A\Sigma\Sigma^*.$$

Observamos que

$$B = \{s \in A \mid \text{nenhum segmento inicial } t \text{ de } s,\ t \neq s, \text{ pertence a } A\}.$$

Sejam

$$C = J\gamma^{-1} \quad \text{e} \quad D = B \backslash C.$$

Consideremos

$$T = \{(m', \sigma) \mid m' \in M,\ \sigma \in \Sigma \ \text{ e } \ (m'\gamma^{-1})\sigma \cap D \neq \varnothing\}.$$

Vamos mostrar que

$$A = C \cup \bigcup_{(m',\sigma)\in T} (m'\gamma^{-1})\sigma\Sigma^*. \tag{11}$$

De fato, suponhamos que $s \in A$. Se $s \in C$, não temos nada a mostrar, caso contrário

$$s \in A \backslash C.$$

Seja u o segmento inicial de comprimento mínimo de s que está em A. Por construção de B, $u \in B$. Por outro lado, $C = \Sigma^* C \Sigma^*$, logo $u \notin C$, pois caso contrário teríamos $s \in C$. Assim, $u \in B \backslash C = D$. Se $u = 1$, então $1 = u\gamma \in A\gamma = mM$. Vem que $M = mM$, logo $J = M$. Então $u \in J\gamma^{-1} = C$, uma contradição. Assim $u \neq 1$ e $u = t\sigma$ para algum

t em Σ^* e σ em Σ. Resulta que $(t\gamma, \sigma) \in T$ e $s \in (t\gamma\gamma^{-1})\sigma\Sigma^*$. Reciprocamente, já que $J \subseteq mM$,

$$C = J\gamma^{-1} \subseteq (mM)\gamma^{-1} = A.$$

Por outro lado, seja $(m', \sigma) \in T$ e seja $t \in m'\gamma^{-1}$, tal que $t\sigma \in D \subseteq A$. Seja $v \in m'\gamma^{-1}$. Então $(t\sigma)\gamma = (v\sigma)\gamma$, logo $v\sigma \in A$. De (10),

$$(m'\gamma^{-1})\sigma\Sigma^* \subseteq A.$$

Isto prova (11).

Mostramos agora, que para todo m' em M, tal que $(m', \sigma) \in T$ para algum σ em Σ,

$$card\ W_{m'} \geq 2. \tag{12}$$

De fato, seja $(t\gamma, \sigma) \in T$, com $t\sigma \in D$. Sejam

$$m' = t\gamma \quad \text{e} \quad m_1 = \sigma\gamma.$$

Como $t\sigma \in D \subseteq A = (mM)\gamma^{-1}$, vem que $(t\sigma)\gamma \in mM$, isto é

$$m'm_1 \in mM. \tag{13}$$

Por outro lado, como $t\sigma \in D \subseteq B$, resulta da construção de B, que $t \notin A = (mM)\gamma^{-1}$, logo $t\gamma \notin mM$, isto é,

$$m' \notin mM. \tag{14}$$

Suponhamos agora que $m'm_1 \notin W_{m'}$. Resulta que $m' = am'm_1b$ para algum $a, b \in M$. Pela Proposição 2, $m' = m'm_1b$. Como mM é um ideal à direita, resulta de (13), que $m' = m'm_1b \in mM$. Isto contradiz (14), logo

$$m'm_1 \in W_{m'}. \tag{15}$$

De (13) e (15):

$$W_{m'} \cap mM \neq \varnothing. \tag{16}$$

Suponhamos agora, que $W_{m'} \subseteq mM$. Pela Proposição 3, $W_{m'}$ é um ideal bilateral, segue portanto da construção de J, que $W_{m'} \subseteq J$. Vem de (15), que $m'm_1 \in J$ e conseqüentemente $t\sigma \in C = J\gamma^{-1}$. Isto contradiz a escolha de t, portanto

$$W_{m'} \backslash mM \neq \varnothing. \tag{17}$$

Agora, (16) e (17) implicam (12).

Finalmente, como $m' \notin W_{m'}$, segue de (12) e do Lema 1, que $(m'\gamma^{-1})\sigma\Sigma^*$ é inteiro para cada (m', σ) em T. Como T é finito e C é inteiro pelos Lemas 1 e 2, segue de (11) que $A = (mM)\gamma^{-1}$ é um subconjunto inteiro de Σ^*. ■

Observamos que o Lema 3′ prova-se de maneira análoga ao Lema 3. Estamos prontos para mostrar o Teorema 1:

Demonstração do Teorema 1. Suponhamos que o monóide sintático M_A de A é finito e aperiódico. Seja

$$\gamma: \Sigma^* \to M_A$$

o morfismo sintático de A. Segue da definição da congruência sintática, que para s e t em Σ^*

$$\text{se } s\gamma = t\gamma \text{ então } s \in A \text{ sse } t \in A.$$

Em outras palavras, $A\gamma\gamma^{-1} = A$. Aplicando-se a Proposição 5 para $A\gamma \subseteq M_A$, resulta que A é um subconjunto inteiro de Σ^*.

Reciprocamente, seja A um subconjunto inteiro de Σ^*, devemos mostrar que M_A é finito e aperiódico. De fato, segue do Corolário II.3 que

$$Int\ \Sigma \subseteq Rac\ \Sigma.$$

Assim, pelo Teorema de Kleene e a Proposição II.9, o monóide sintático M_A de A é finito. Para mostrar que M_A é aperiódico, consideremos o seguinte subconjunto de $\mathfrak{p}(\Sigma^*)$:

$$\mathcal{F} = \{R \subseteq \Sigma^* \mid M_R \text{ é aperiódico}\}.$$

É fácil verificar que $\varnothing$, 1 e σ, para σ em Σ, pertencem a $\mathcal{F}$. Por outro lado, vamos mostrar que $\mathcal{F}$ é fechado sob união, concatenação e complementação. Segue então da definição de $Int\ \Sigma$, que

$$Int\ \Sigma \subseteq \mathcal{F}.$$

Assim fica provada a parte "somente se" do Teorema 1.

Para mostrar que $\mathcal{F}$ é fechado sob complementação, observamos que $M_R = M_{\bar{R}}$. Assim $R \in \mathcal{F}$ implica $\bar{R} \in \mathcal{F}$.

Para mostrar que $\mathcal{F}$ é fechado sob união e concatenação, sejam R e S em $\mathcal{F}$ e s em Σ^*. Por hipótese, existem naturais n e m, tais que

$$s^n \equiv s^{n+1} \ (mod\ R) \text{ e } s^m \equiv s^{m+1} \ (mod\ S).$$

Definindo $p = max\{n, m\}$ é imediato que

$$s^p \equiv s^{p+1} \ (mod\ R) \text{ e } s^p \equiv s^{p+1} \ (mod\ S).$$

Resulta da definição de congruência sintática, que

$$s^p \equiv s^{p+1} \ (mod\ R \cup S).$$

Assim $R \cup S$ também pertence a $\mathscr{F}$. Definimos agora $q = n + m + 1$. Sejam u e v em Σ^*, tais que

$$us^q v \in RS.$$

Existem então t_1, $t_2 \in \Sigma^*$, tais que

$$us^q v = t_1 t_2, \ t_1 \in R \text{ e } t_2 \in S.$$

Segue da definição de q, que, para um w conveniente em Σ^*,

$$t_1 = us^n w \text{ ou } t_2 = ws^m v.$$

Assim, $us^{n+1}w \in R$ ou $ws^{m+1}v \in S$; em qualquer caso

$$us^{q+1}v \in RS.$$

Analogamente, $us^{q+1}v \in RS$ implica que $us^q v \in RS$. Portanto

$$s^q \equiv s^{q+1} \ (mod\ RS),$$

e RS também pertence a $\mathscr{F}$. Isto completa a demonstração do Teorema 1. ■

4. Um exemplo

Vamos aplicar o procedimento da Proposição 5, para obter uma expressão inteira que prove que

$$A = (\sigma\tau \cup \tau\sigma)^*$$

é um subconjunto inteiro de Σ^* ($\Sigma = \{\sigma, \tau\}$). Note que na Seção 1 já apresentamos uma tal expressão.

A Figura 1 mostra um autômato determinístico $\mathscr{A}$ que reconhece A e a Figura 2 representa o monóide $M_{\mathscr{A}} = M$ do autômato $\mathscr{A}$. Observamos que M é isomorfo ao monóide sintático de A, mas não vamos precisar deste fato. Os elementos de M são

$$\{a, b, c, d, e, f, g, h, i, j, k, \ell, m, n, p\},$$

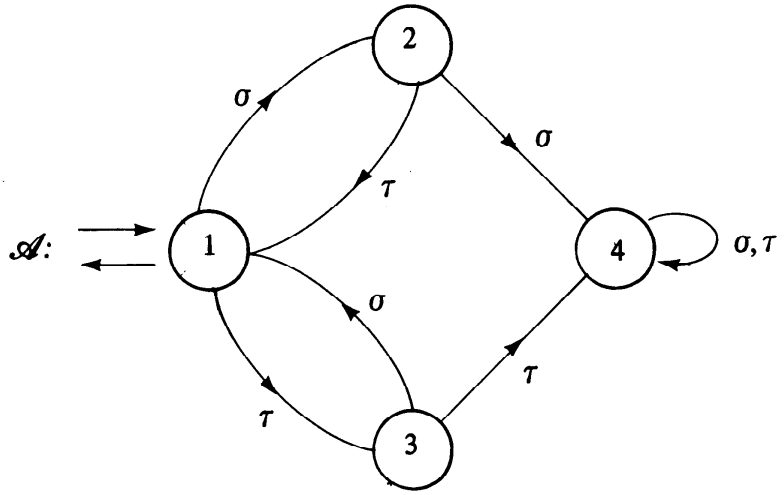


Figura 1 — Autômato $\mathscr{A}$ que reconhece A.

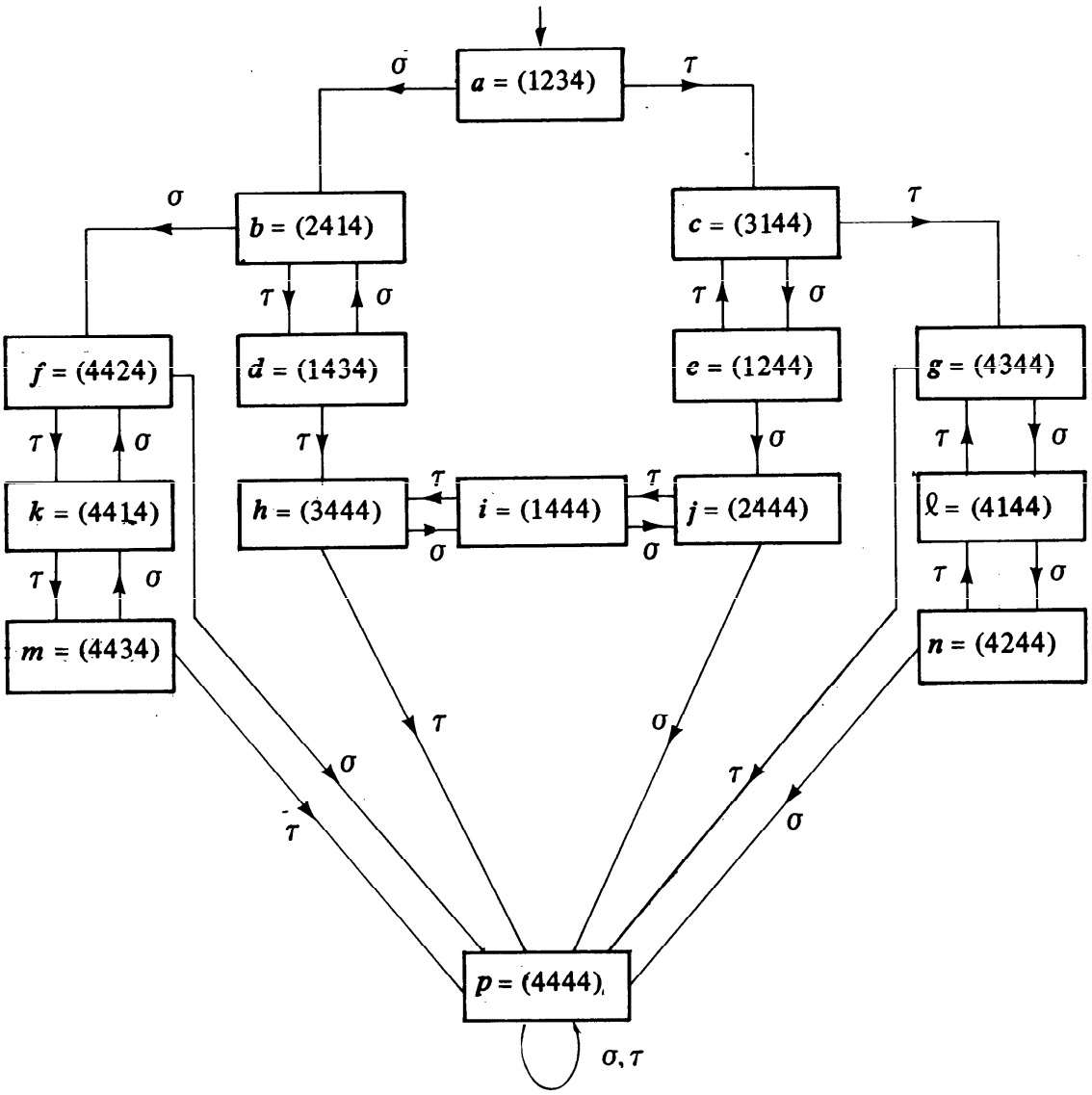


Figura 2 — O monóide $M = M_{\mathscr{A}}$ do autômato $\mathscr{A}$.

sendo a a identidade de M e p o seu zero. Na Figura 2, a notação $x = (n_1 n_2 n_3 n_4)$ indica que o elemento x de M é a função $P \to P$ que leva i em n_i $(i = 1, 2, 3, 4)$. Para cada s em Σ^*, $s\alpha = x$ sse na Figura 2, a palavra s leva a em x. Por exemplo:

$$(\sigma\tau)\alpha = d, \quad (\tau\sigma\tau)\alpha = c \quad \text{e} \quad (\sigma\tau\tau\sigma\tau)\alpha = h.$$

A multiplicação em M é obtida assim: sendo x e y elementos em M, sejam s e t em Σ^*, tais que $s\alpha = x$ e $t\alpha = y$. Resulta que $(st)\alpha = (s\alpha)(t\alpha) = xy$. Por exemplo: $dc = h$.

Ideais Principais à Direita

x	xM
a	$\{a, b, c, d, e, f, g, h, i, j, k, \ell, m, n, p\}$
b, d	$\{b, d, f, h, i, j, k, m, p\}$
c, e	$\{c, e, g, h, i, j, \ell, n, p\}$
f, k, m	$\{f, k, m, p\}$
g, ℓ, n	$\{g, \ell, n, p\}$
h, i, j	$\{h, i, j, p\}$
p	$\{p\}$

Ideais Principais à Esquerda

x	Mx
a	$\{a, b, c, d, e, f, g, h, i, j, k, \ell, m, n, p\}$
b, e	$\{b, e, f, i, j, k, \ell, n, p\}$
c, d	$\{c, d, g, h, i, k, \ell, m, p\}$
f, j, n	$\{f, j, n, p\}$
g, h, m	$\{g, h, m, p\}$
i, k, ℓ	$\{i, k, \ell, p\}$
p	$\{p\}$

Ideais Principais Bilaterais

x	MxM
a	$\{a, b, c, d, e, f, g, h, i, j, k, \ell, m, n, p\}$
b, c, d, e	$\{b, c, d, e, f, g, h, i, j, k, \ell, m, n, p\}$
$f, g, h, i, j, k, \ell, m, n$	$\{f, g, h, i, j, k, \ell, m, n, p\}$
p	$\{p\}$

Figura 3 — Ideais principais em M.

Deixamos ao leitor a verificação de que para todo x em M, $x^3 = x^4$, portanto M é aperiódico. A Figura 3 mostra os ideais principais gerados pelos elementos de M.

Sendo $\alpha\colon \Sigma^* \to M$ o morfismo do autômato $\mathscr{A}$, temos que

$$A = \{a, d, e, i\}\alpha^{-1}.$$

Aplicando o Lema 1 para o ideal bilateral: $\{f, g, h, i, j, k, \ell, m, n, p\}$, obtemos expressões inteiras B, C, D e E, tais que

$$a\alpha^{-1} = 1,\ b\alpha^{-1} = \sigma(\tau\sigma)^* = B,\ c\alpha^{-1} = \tau(\sigma\tau)^* = C,$$
$$d\alpha^{-1} = \sigma\tau(\sigma\tau)^* = D \text{ e } e\alpha^{-1} = \tau\sigma(\tau\sigma)^* = E.$$

Deixamos como exercício a obtenção destas expressões.

Para obter uma expressão inteira para A, basta agora calcular $i\alpha^{-1}$. Pela Proposição 4, é suficiente calcular

$$W_i\alpha^{-1} = p\alpha^{-1},\ (iM)\alpha^{-1} = \{h, i, j, p\}\alpha^{-1} \text{ e } (Mi)\alpha^{-1} = \{i, k, \ell, p\}\alpha^{-1},$$

e teremos

$$i\alpha^{-1} = ((Mi)\alpha^{-1} \cap (iM)\alpha^{-1})\backslash W_i\alpha^{-1}.$$

Para calcularmos $W_i\alpha^{-1} = p\alpha^{-1}$ aplicamos o Lema 2. Temos $\Sigma_0 = \varnothing$, e

$$T = \{(\sigma, b, \sigma),\ (\tau, c, \tau)\}.$$

Assim,

$$W_i\alpha^{-1} = p\alpha^{-1} = \Sigma^*\sigma(b\alpha^{-1})\sigma\Sigma^* \cup \Sigma^*\tau(c\alpha^{-1})\tau\Sigma^*.$$

Para calcularmos $(iM)\alpha^{-1}$, aplicamos o Lema 3. Temos

$$J = \{p\} \text{ e } T = \{(d, \tau),\ (e, \sigma)\}.$$

Assim,

$$(iM)\alpha^{-1} = p\alpha^{-1} \cup (d\alpha^{-1})\tau\Sigma^* \cup (e\alpha^{-1})\sigma\Sigma^*.$$

Para calcular $(Mi)\alpha^{-1}$, aplicamos o Lema 3'. O maior ideal bilateral contido em Mi é $\{p\}$ e resulta que

$$J = \{p\} \text{ e } T = \{(\sigma, d),\ (\tau, e)\}.$$

Assim,

$$(Mi)\alpha^{-1} = p\alpha^{-1} \cup \Sigma^*\sigma(d\alpha^{-1}) \cup \Sigma^*\tau(e\alpha^{-1}).$$

Finalmente, usando as expressões B, C, D e E, e colocando $\Sigma^* = \overline{\varnothing} = I$, obtém-se, após algumas simplificações evidentes

$$A = a\alpha^{-1} \cup d\alpha^{-1} \cup e\alpha^{-1} \cup i\alpha^{-1} =$$
$$= 1 \cup D \cup E \cup [(I\sigma D \cup I\tau E) \cap (D\tau I \cup E\sigma I)]\backslash(I\sigma B\sigma I \cup I\tau C\tau I).$$

Observamos que os conjuntos T podem ser obtidos algoritmicamente, porém o algoritmo correspondente é incômodo, pois envolve a construção de vários autômatos.

EXERCÍCIOS

1. Mostre que $(\sigma\sigma)^*$ não é um subconjunto inteiro de Σ^*.
2. Seja M um monóide finito. Mostre que M é aperiódico sse todo grupo em M é trivial. Mostre que esta afirmação não é verdadeira se M for infinito.
3. Seja J um ideal bilateral do monóide M; então $1 \cup J$ é um submonóide de M.

   União de ideais é um ideal do mesmo tipo.
   Intersecção não vazia de ideais é um ideal do mesmo tipo.
   Todo ideal J é a união dos ideais principais do mesmo tipo, gerados pelos elementos de J. Por exemplo,

$$\text{se } MJ = J, \text{ então } J = \bigcup_{m \in J} Mm.$$

4. Mostre a recíproca da Proposição 2.
5. Mostre o Lema 3′.
6. Um monóide M é *idempotente* sse para todo m em M, $m = m^2$. Mostre que todo monóide idempotente, finitamente gerado, é finito (vide [26] ou [42]).
7. Mostre que para cada subconjunto racional A de Σ^* existe um alfabeto finito Γ, um morfismo h de Γ^* em Σ^* e um subconjunto inteiro B de Γ^*, tais que $A = Bh$.

NOTAS BIBLIOGRÁFICAS

Conjuntos inteiros foram introduzidos por Trakhtenbrot [118] e independentemente por McNaughton [73]. O Teorema 1 é de M. P. Schützenberger [107], [108]. A demonstração apresentada é baseada no trabalho original de Schützenberger.

Um estudo mais detalhado de conjuntos inteiros pode ser encontrado nos livros de McNaughton e Papert [75] e de Eilenberg [26].

BIBLIOGRAFIA

[1] A. V. AHO, J. E. HOPCROFT e J. D. ULLMAN, *The design and analysis of computer algoritms*, Addison-Wesley, Reading, Mass., 1974.

[2] K. APPEL e W. HAKEN, Every planar map is four colorable, *Ill. J. Math.* **21** (1977), 429-567.

[3] C. BERGE, *Graphes et hypergraphes*, Dunod, Paris, 1970. Edição revista e traduzida para o inglês: *Graphs and hypergraphs*, North-Holland, Amsterdam, 1973.

[4] N. BIGGS, *Algebraic graph theory*, Cambridge University Press, Cambridge, 1974.

[5] M. BLUM, A machine independent theory of the complexity of recursive functions, *J. Assoc. Comput. Mach.* **14** (1967), 332-336.

[6] J. A. BONDY e U. S. R. MURTY, *Graph theory with applications*, MacMillan, London, 1976.

[7] A. BORODIN, Computational complexity – theory and practice, em A. V. Aho (ed.), *Currents in the theory of computing*, Prentice Hall, Englewood Cliffs, N. J., 1973, 35-89.

[8] A. BORODIN e I. MUNRO, *The computational complexity of algebraic and numeric problems*, American Elsevier, New York, 1975.

[9] R. L. BROOKS, On colouring the nodes of a network, *Proc. Cambridge Philos. Soc.* **37** (1941), 194-197.

[10] J. BUNCH e J. E. HOPCROFT, Triangular factorization and inversion by fast matrix multiplication, *Math. Comp.* **28** (1974), 231-236.

[11] S. A. BURR, Generalized Ramsey theory for graphs – a survey, em R. A. Bari e F. Harary (eds.), *Graphs and combinatorics*, Lecture Notes in Mathematics 406, Springer-Verlag, Berlin, 1974, 52-75.

[12] R. M. BURSTALL, Proving properties of programs by structural induction, *Comput. J.* **12** (1969), 41-48.

[13] R. G. BUSACKER e T. L. SAATY, *Finite graphs and networks: an introduction with applications*, McGraw-Hill, New York, 1965.

[14] V. CHVÁTAL e F. HARARY, Generalized Ramsey theory for graphs, *Bull. Amer. Math. Soc.* **78** (1972), 423-426.

[15] A. H. CLIFFORD e G. B. PRESTON, *The algebraic theory of semigroups*, vol. I e II, American Mathematical Society, Providence, R. I., 1961 e 1967.

[16] A. COBHAM, The intrinsic computational difficulty of functions, em Y. Bar-Hillel (ed.), *Logic, methodology and philosophy of science*, North-Holland, Amsterdam, 1965, 24-30.

[17] S. A. COOK, The complexity of theorem-proving procedures, em *Proc. 3rd. ACM Symposium on Theory of Computing*, Association for Computing Machinery, New York, 1971, 151-158.

[18] M. DAVIS, *Computability and unsolvability*, McGraw-Hill, New York, 1958.

[19] M. DAVIS (ed.), *The undecidable*, Raven Press, New York, 1965.

[20] M. DAVIS, Hilbert's tenth problem is unsolvable, *Amer. Math. Monthly* **80** (1973), 233-269.

[21] M. DAVIS e R. HERSCH, Hilbert's tenth problem, *Sci. Amer.* **299** (1973), 84-91.

[22] E. W. DIJKSTRA, A note on two problems in connexion with graphs, *Numer. Math.* **1** (1959), 269-271.

[23] J. EDMONDS, Paths, trees and flowers, *Canad. J. Math.* **17** (1965), 449-467.

[24] J. EDMONDS, Matroids and the greedy algorithm, *Math. Programming* **1** (1971), 127-136.

[25] S. EILENBERG, *Automata, languages, and machines*, vol. A, Academic Press, New York, 1974.

[26] S. EILENBERG, *Automata, languages, and machines*, vol. B, Academic Press, New York, 1976.

[27] B. ELSPAS, K. N. LEVITT, R. J. WALDINGER e A. WAKSMAN, An assessment of techniques for proving program correctness, *Comput. Surveys* **4** (1972), 97-147.

[28] P. ERDÖS, Some remarks on the theory of graphs, *Bull. Amer. Math. Soc.* **53** (1947), 292-294.

[29] P. ERDÖS e J. SPENCER, *Probabilistic methods in combinatorics*, Academic Press, New York, 1974.

[30] P. ERDÖS e G. SZEKERES, A combinatorial problem in geometry, *Compositio Math.* **2** (1935), 463-470. Reimpresso em P. ERDÖS, *The art of counting, selected writings*, (J. Spencer (ed.)), The MIT Press, Cambridge, Mass., 1973.

[31] P. ERDÖS e G. SZEKERES, On some extremum problems in elementary geometry, *Ann. Univ. Sci. Budapest. Eötvös Sect. Math* **3** (1960/61), 53-62. Reimpresso em P. Erdös, *The art of counting, selected writings*, (J. Spencer (ed.)), The MIT Press, Cambridge, Mass., 1973.

[32] L. EULER, Solutio problematis ad geometriam situs pertinentis, *Comentarii Academiae Petropolitanae* **8** (1736), 128-140. Traduzido para o inglês: The Königsberg bridges, *Sci. Amer.* **189** (1953), 66-70.

[33] P. FEOFILOFF, *Sobre os números de Ramsey*, Dissertação de Mestrado, Instituto de Matemática e Estatística da Universidade de São Paulo, 1974.

[34] P. C.FISCHER, Further schemes for combining matrix algorithms, em J. Loeckx (ed.), *Automata, languages and programming*, Lecture Notes in Computer Science 14, Springer-Verlag, Berlin, 1974, 428-436.

[35] R. FRUCHT, Graphs of degree three with a given abstract group, *Canad. J. Math.* **1** (1949), 365-378.

[36] A. GINZBURG, *Algebraic theory of automata*, Academic Press, New York, 1968.

[37] G. R.GIRAUD, Une généralisation des nombres et de l'inégalité de Schur, *C. R. Acad. Sci. Paris Ser. A* **266** (1968), 437-440.

[38] K. GODEL, Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I, *Monatshefte für Mathematik und Physik* **38** (1931), 173-198. Traduzido para o inglês em M. Davis (ed.), *The undecidable*, Raven Press, New York, 1965, 5-38.

[39] H. H. GOLDSTINE, *The computer from Pascal to von Neumann*, Princeton University Press, Princeton, N. J., 1972.

[40] R. L. GRAHAM e B. L. ROTHSCHILD, Ramsey's theorem for n-parameter sets, *Trans. Amer. Math. Soc.* **159** (1971), 257-292.

[41] J. E. GRAVER e J. YACKEL, Some graph theoretic results associated with Ramsey's theorem, *J. Combinatorial Theory* **4** (1968), 125-175.

[42] J. A. GREEN e D. REES, On semigroups in which $x^r = x$, *Proc. Cambridge Philos. Soc.* **48** (1952), 35-40.

[43] R. E. GREENWOOD e A. M. GLEASON, Combinatorial relations and chromatic graphs, *Canad. J. Math.* **7** (1955), 1-7.

[44] M. HALL JR., An algorithm for distinct representatives, *Amer. Math. Monthly* **63** (1956), 716-717.

[45] M. HALL JR., *Combinatorial theory*, Blaisdell Pu. Co., Waltham, Mass., 1967.

[46] P. HALL, On representatives of subsets, *J. London Math. Soc.* **10** (1935), 26-30.

[47] P. HALMOS, *Naive set theory*, van Nostrand Reinhold, New York, 1960. Traduzido para o português: *Teoria ingênua dos conjuntos*, Editora da Universidade de São Paulo e Editora Polígono, São Paulo, 1970.

[48] P. HALMOS e H. E. VAUGHAN, The marriage problem, *Amer. J. Math.* **72** (1950), 214-215.

[49] F. HARARY, *Graph theory*, Addison-Wesley, Reading, Mass., 1969.

[50] F. HARARY, R. Z. NORMAN e D. CARTWRIGHT, *Structural models: an introduction to the theory of directed graphs*, John Wiley & Sons, New York, 1965.

[51] M. A. HARRISON, *Introduction to switching and automata theory*, McGraw-Hill, New York, 1965.

[52] M. A. HARRISON, *Introduction to formal language theory*, Addison-Wesley, Reading, Mass., 1978.

[53] J. HARTMANIS e J. E. HOPCROFT, An overview of the theory of computational complexity, *J. Assoc. Comput. Mach.* **18** (1971), 444-475.

[54] J. HARTMANIS e R. E. STEARNS, Computational complexity of recursive sequences, em *Proc. 5th. IEEE Symposium on Switching Circuits and Logical Design*, Institute of Electrical and Electronics Engineers, Piscataway, N. J., 1964, 82-90.

[55] J. HARTMANIS e R. E. STEARNS, On the computational complexity of algorithms, *Trans. Amer. Math. Soc.* **117** (1965), 285-306.

[56] J. E. HOPCROFT e J. D. ULLMAN, *Formal languages and their relation to automata*, Addison-Wesley, Reading, Mass., 1969.

[57] R. M. KARP, Reducibility among combinatorial problems, em R. E. Miller e J. W. Thatcher (eds.), *Complexity of computer computations*, Plenum Press, New York, 1972, 85-104.

[58] A. B. KEMPE, On the geographical problem of four colours, *Amer. J. Math.* **2** (1879), 193-204.

[59] S. C. KLEENE, Representation of events in nerve nets and finite automata, em C. E. Shannon e J. McCarthy (eds.), *Automata studies*, Princeton University Press, Princeton, N. J., 1956, 3-41.

[60] D. E. KNUTH, *The art of computer programming, vol. 1: Fundamental algorithms*, Addison-Wesley, Reading, Mass., 1968.

[61] D. E. KNUTH, *The art of computer programming, vol. 2: Seminumerical algorithms*, Addison-Wesley, Reading, Mass., 1969.

[62] D. E. KNUTH, *The art of computer programming, vol. 3: Sorting and searching*, Addison-Wesley, Reading, Mass., 1973.

[63] D. KONIG, Grafos e matrizes (em húngaro), *Mat. Fiz. Lapok* **38** (1931), 116-119.
[64] J. B. KRUSKAL JR., On the shortest spanning subtree of a graph and the traveling salesman problem, *Proc. Amer. Math. Soc.* **7** (1956), 48-50.
[65] C. KURATOWSKI, Sur le problème des courbes gauches en topologie, *Fund. Math.* **15** (1930), 271-283.
[66] J. C. LAFON, Sur le produit de deux quaternions, *C. R. Acad. Sci. Paris Ser. A* **280** (1975), 665-668.
[67] J. VAN LEEUWEN e P. VAN EMDE BOAS, Some elementary proofs of lower-bounds in complexity theory, Publ. Matematisch Centrum IW41/76, Amsterdam, 1976.
[68] L. LOVÁSZ, On two minimax theorems in graph theory, *J. Combinatorial Theory Ser. B* **27** (1976), 96-103.
[69] C. L. LUCCHESI, *A minimax equality for directed graphs*, Ph. D. thesis, University of Waterloo, Waterloo, Ont., 1976.
[70] C. L. LUCCHESI e D. H. YOUNGER, A minimax theorem for directed graphs, *J. London Math. Soc.* (2) **17** (1978), 369-374.
[71] M. MACHTEY e P. YOUNG, *An introduction to the general theory of algorithms*, Elsevier North-Holland, New York, 1978.
[72] S. MACLANE e G. BIRKHOFF, *Algebra*, The Macmillan Co., New York, 1967.
[73] R. MCNAUGHTON, Symbolic logic for automata, Wright Air Development Division Tech. Note N.° 60-244, Cincinnati, Ohio, 1960.
[74] R. MCNAUGHTON, Algebraic decision procedures for local testability, *Math. Systems Theory* **8** (1974), 60-76.
[75] R. MCNAUGHTON e S. PAPERT, *Counter-free automata*, The MIT Press, Cambridge, Mass., 1971.
[76] A. MANDEL e I. SIMON, On finite semigroups of matrices, *Theor. Comput. Sci.* **5** (1977), 101-111.
[77] Z. MANNA, *Mathematical theory of computation*, McGraw-Hill, New York, 1974.
[78] Z. MANNA, S. NESS e J. VUILLEMIN, Inductive methods for proving properties of programs, em *Proc. of an ACM Conference on Proving Assertions about Programs*, SIGPLAN Notices 7 (1972) e SIGACT News 14 (1972), 27-50.
[79] L. S. MELNIKOV e V. G. VIZING, New proof of Brook's theorem, *J. Combinatorial Theory Ser. B* **7** (1969), 408-409.
[80] A. R. MEYER e L. J. STOCKMEYER, The equivalence problem for regular expressions with squaring requires exponential space, em *Proc. 13th. IEEE Conference on Switching and Automata Theory*, Institute of Electrical and Electronics Engineers, Piscataway, N. J., 1972, 125-129.
[81] G. L. MILLER, Riemann's hypothesis and tests for primality, *J. Comput. Systems Sci.* **13** (1976), 300-317.
[82] K. R. MILLIKEN, Ramsey's theorem with sums or unions, *J. Combinatorial Theory Ser. A* **18** (1975), 276-290.
[83] M. MINSKY, *Computation: finite and infinite machines*, Prentice-Hall, Englewood Cliffs, N. J., 1967.
[84] L. MIRSKY, *Transversal theory*, Academic Press, New York, 1971.
[85] E. F. MOORE (ed.), *Sequential machines: selected papers*, Addison-Wesley, Reading, Mass., 1964.

[86] J. H. MORRIS JR., Another recursion induction principle, *Comm. ACM* **14** (1971), 351-354.

[87] P. NAUR (ed.), Revised report on the algorithmic language Algol 60, *Comm. ACM* **6** (1963), 1-17 e *Comput. J.* **5** (1963), 349-367.

[88] J. NESETRIL e V. RODL, A structural generalization of the Ramsey theorem, *Bull. Amer. Math. Soc.* **83** (1977), 127-128.

[89] P. S. NOVIKOV, On the algorithmic insolvability of the word problem in group theory, *Trudy Mat. Inst. Steklov.* **44** (1955), 1-143. Traduzido para o inglês em *Amer. Math. Soc. Transl. Ser.* 2 **9** (1958), 1-122.

[90] O. ORE, *The four-color problem*, Academic Press, New York, 1967.

[91] A. M. OSTROWSKI, On two problems in abstract algebra connected with Horner's rule, em *Studies presented to R. von Mises*, Academic Press, New York, 1954, 40-48.

[92] V. YA. PAN, Strassen's algorithm is not optimal: trilinear technique of aggregating, uniting and canceling for constructing fast algorithms for matrix operations, em *Proc. 19th IEEE Symposium on Foundations of Computer Science*, Institute of Electrical and Electronics Engineers, Piscataway, N. J., 1978, 166-176.

[93] M. S. PATERSON e C. E. HEWITT, Comparative schematology, em *Record of Project MAC Conference on Concurrent Systems and Parallel Computation*, Association for Computing Machinery, New York, 1970, 119-128.

[94] R. PÉTER, *Rekursive Funktionen*, Akadémiai Kiadó, Budapest, 1951.

[95] E. L. POST, A variant of a recursively unsolvable problem, *Bull. Amer. Math. Soc.* **52** (1946), 264-268.

[96] V. R. PRATT, Every prime has a succint certificate, *SIAM J. Comput.* **4** (1975), 214-220.

[97] M. O. RABIN, Speed of computation of functions and classification of recursive sets, em *Proc. Third Convention of Scientific Societies*, Israel, 1959, 1-2.

[98] M. O. RABIN, Degrees of difficulty of computing a function and a partial ordering of recursive sets, Technical Report 2, Hebrew University, Jerusalem, 1960.

[99] M. O. RABIN e D. SCOTT, Finite automata and their decision problems, *IBM J. Res. Develop.* **3** (1959), 114-125. Reimpresso em E. F. Moore (ed.), *Sequential machines: selected papers*, Addison-Wesley, Reading, Mass., 1964, 63-91.

[100] F. P. RAMSEY, On a problem of formal logic, *Proc. London Math. Soc. 2nd. Ser.* **30** (1930), 264-286. Reimpresso em F. P. Ramsey, *The foundations of mathematics and other logical essays*, (R. B. Braithwaite (ed.)), Routledge & Kegan Paul, London, 1931, 82-111.

[101] H. ROGERS JR., *Theory of recursive functions and effective computability*, McGraw-Hill, New York, 1967.

[102] H. J. RYSER, *Combinatorial mathematics*, The Mathematical Association of America e John Wiley and Sons, 1963.

[103] T. L. SAATY, Thirteen colorful variations on Guthrie's four-colour conjecture, *Amer. Math. Monthly* **79** (1972), 2-43.

[104] A. SALOMAA, *Theory of automata*, Pergamon Press, Oxford, 1969.

[105] A. SCHONHAGE e V. STRASSEN, Schnelle Multiplikation grosser Zahlen, *Computing* **7** (1971), 281-292.

[106] I. SCHUR, Über die Kongruenz $x^m + y^m = z^m \pmod{p}$, *Jber. Deutsch. Math. Verein.* **25** (1916), 114-117.

[107] M. P. SCHÜTZENBERGER, On finite monoids having only trivial subgroups, *Information and Control* **8** (1965), 190-194.

[108] M. P. SCHÜTZENBERGER, On a family of sets related to McNaughton's L-language, em E. R. Caianiello (ed.), *Automata theory*, Academic Press, New York, 1966, 320-324.

[109] M. P. SCHÜTZENBERGER, *Quelques problemes combinatoires de la theorie des automates* (redigé par J. F. Perrot), Institut de Programmation, Paris, 1967.

[110] S. SESHU e M. REED, *Linear graphs and electrical networks*, Addison-Wesley, Reading, Mass., 1961.

[111] J. R. SHOENFIELD, *Mathematical logic*, Addison-Wesley, Reading, Mass., 1967.

[112] J. R. SHOENFIELD, *Degrees of unsolvability*, North-Holland, Amsterdam, 1971.

[113] I. SIMON, *Introdução à teoria de complexidade de algoritmos*, Escola de Computação, Instituto de Matemática e Estatística da Universidade de São Paulo, São Paulo, 1979.

[114] L. J. STOCKMEYER, Planar 3-colorability is polynomial complete, *SIGACT News* **5** (1973), 19-25.

[115] L. J. STOCKMEYER e A. R. MEYER, Word problems requiring exponential time, em *Proc. 5th ACM Symposium on Theory of Computing*, Association for Computing Machinery, New York, 1973, 1-9.

[116] V. STRASSEN, Gaussian elimination is not optimal, *Numer. Math.* **13** (1969), 354-356.

[117] G. SZEKERES, A combinatorial problem in geometry: reminiscences, em P. Erdös, *The art of counting, selected writings*, (J. Spencer (ed.)), The MIT Press, Cambridge, Mass., 1973, xix – xxii.

[118] B. A. TRAKHTENBROT, Synthesis of logic networks whose operators are described by means of single-place predicate calculus, *Dokl. Akad. Nauk SSSR* **118** (1958), 646-649.

[119] A. M. TURING, On computable numbers, with an application to the Entscheidungsproblem, *Proc. London Math. Soc. Ser.* 2 **42** (1936-37), 230-265 e **43** (1937) 544-546.

[120] W. T. TUTTE, The factorization of linear graphs, *J. London Math. Soc.* **22** (1947), 107-111.

[121] W. T. TUTTE, Lectures on matroids, *J. Res. Nat. Bur. Standards Sect. B* **69** (1965), 1-47.

[122] W. T. TUTTE, *Introduction to matroid theory*, Elsevier, New York, 1971.

[123] S. M. ULAM, *A collection of mathematical problems*, Wiley Interscience, New York, 1960.

[124] P. A. S. VELOSO, *Máquinas e linguagens: uma introdução à teoria de autômatos*, Escola de Computação, Instituto de Matemática e Estatística da Universidade de São Paulo, São Paulo, 1979.

[125] S. A. WALKER e H. R. STRONG, Characterizations of flowchartable recursions, *J. Comput. System Sci.* **7** (1973), 404-447.

[126] W. D. WALLIS, A. P. STREET e J. S. WALLIS, *Combinatorics: Room squares, sum-free sets, Hadamard matrices*, Lecture Notes in Mathematics 292, Springer-Verlag, Berlin, 1972.

[127] P. D. WHITING e J. A. HILLIER, A method for finding the shortest route through a road network, *Operational Res. Quart.* **11** (1960), 37-40.

[128] H. WHITNEY, On the abstract properties of linear dependence, *Amer. J. Math.* **57** (1935), 509-533.

[129] H. WHITNEY e W. T. TUTTE, Kempe chains and the four colour problem, *Utilitas Math.* **2** (1972), 241-281.

[130] N. WIRTH, *Systematisches Programmieren*, B. G. Teubner, Stuttgart, 1975. Traduzido para o português: *Programação sistemática*, Editora Campus, Rio de Janeiro, 1978.

[131] A. C. YAO, An $O(|E| \log\log |V|)$ algorithm for finding minimum spanning trees, *Information Processing Lett.* **4** (1975), 21-23.

ÍNDICE DE NOTAÇÕES

1. Comandos da linguagem de programação *LP*

2. Tabela de símbolos

3. Tabela de abreviações

Abreviação	*Uso*	*Referência*
a	*a*	136
	aG	133
	aP	141
Adj	*Adj*(*X*)	136
	Adj G(*X*)	133
c	*c*	136
	cG	144
car	$car(\langle s_1, s_2, \ldots, s_n \rangle)$	32
card	*card A*	219
cdr	$cdr(\langle s_1, s_2, \ldots, s_n \rangle)$	32
cf	*cfD*	204
cir	*cir D*	201
cor	*cor D*	201
crom	*crom* (*G*)	177
f	*f*	114
	$f\alpha$	200
F	F_n	23
FNB	*FNB*	17
FNC-SAT	*FNC-SAT*	115
FNC-TAUT	*FNC-TAUT*	116
FND-SAT	*FND-SAT*	116
FND-TAUT	*FND-TAUT*	116
Fun	*Fun A*	219
g	*g*(*v*)	136
	gG(*v*)	140
G	*GD*	200
i	$i\alpha$	200
I	I_n	84
Int	*Int* Σ	257
LP	*LP*	11
M	M_A	249
	$M_{\mathscr{A}}$	249
mdc	*mdc* (*m*, *n*)	4
n	**n**	226
$\mathbb{N}$	conjunto dos naturais	
$\mathscr{NP}$	$\mathscr{NP}$	110

ÍNDICE ALFABÉTICO

claudio l. lucchesi
imre simon istvan simon
janos simon tomasz kowaltowski

Os autores iniciaram-se em Computação em torno do saudoso computador IBM 1620, instalado em 1962 no então Centro de Cálculo Numérico da Escola Politécnica da Universidade de São Paulo. Formaram-se Engenheiros Eletrônicos por aquela Escola: Cláudio em 1968, Imre em 1966, Istvan em 1969, Janos em 1969, Tomasz em 1966. Realizaram estudos de pós-graduação no exterior, obtendo o grau de Ph.D. em Computação: Cláudio — Universidade de Waterloo em 1976, Imre — Universidade de Waterloo em 1972, Istvan — Universidade de Stanford em 1977. Janos — Universidade de Cornell em 1974, Tomasz — Universidade da Califórnia em Berkeley em 1973. Cláudio é Professor Colaborador (MS-4) na UNICAMP. Suas áreas de interesse são: teoria dos grafos e síntese de algoritmos eficientes. Imre é Professor Livre-Docente na USP. Suas áreas de interesse são: teoria dos autômatos e teoria dos grafos. Istvan é Professor Assistente Doutor na USP. Suas áreas de interesse são: análise de algoritmos e teoria de complexidade. Janos é Professor Assistente na Universidade Estadual da Pennsylvania, em afastamento da UNICAMP. Suas áreas de interesse são: teoria de computabilidade e complexidade concreta de algoritmos. Tomasz é Professor Colaborador (MS-4) da UNICAMP, em afastamento da USP. Suas áreas de interesse são: projeto, implementação e semântica de linguagens de programação e verificação formal da correção de programas.

aspectos teóricos da computação

claudio l. lucchesi
imre simon istvan simon

aspectos teóricos

CNPq Conselho Nacional de Desenvolvimento Científico e Tecnológico

impa Instituto de Matemática Pura e Aplicada