Elementos de Teoria da Computação

Harry R. Lewis
Gordon McKay Professor of Computer Science
Harvard University
and Dean of Harvard College
Cambridge, Massachusetts

Christos H. Papadimitriou
C. Lester Hogan Professor of Electrical Engineering
and Computer Science
University of California
Berkeley, California

2ª edição

Tradução:
EDSON FURMANKIEWICZ

Consultoria, supervisão e revisão técnica desta edição:
PAULO GERALDO PINHEIRO
Mestre em Ciência da Computação pela USP.

Bookman

Porto Alegre, 2000

L673e Lewis, Harry R.
Elementos de teoria da computação / Harry R. Lewis e Christos H. Papadimitriou; trad. Edson Furmankiewicz. – 2a. ed. – Porto Alegre: Bookman, 2000.

1. Computação – Elementos de teoria.
I. Papadimitriou, Christos H. II. Título.

CDU 681.62

Catalogação na publicação: Mônica Ballejo Canto – CRB 10/1023

ISBN 85-7307-534-1

Prefácio à segunda edição

Muita coisa mudou nos últimos 15 anos, desde que *Elementos de Teoria da Computação* apareceu pela primeira vez – e muita coisa continuou igual. A ciência da computação é agora uma disciplina muito mais aprimorada e estabelecida, desempenhando um papel de grande importância em um mundo no qual a computação é onipresente, as informações são globalizadas, e a complexidade é cada vez maior – mais razões para se manter em contato com suas bases. Os autores de *Elementos* estão agora muito mais amadurecidos e ocupados – essa é a razão pela qual esta segunda edição demorou tanto. Envolvemo-nos nesse empreendimento porque achamos que algumas coisas poderiam ser ditas de uma maneira melhor, outras, simplificadas – e algumas outras, simplesmente omitidas. Sobretudo, queríamos que o livro refletisse como a teoria da computação e seus alunos desenvolveram-se nestes anos. Embora a teoria da computação agora seja ensinada mais amplamente em termos absolutos, sua posição relativa dentro do currículo da ciência da computação, por exemplo, em relação ao assunto algoritmos, não foi fortalecida. De fato, o campo do projeto e da análise de algoritmos está agora tão desenvolvido que seus princípios elementares são, pode-se argumentar, parte de um curso básico sobre a teoria da computação. Além disso, os universitários de hoje, por exemplo, com sua extensa experiência anterior em computação, estão muito mais cientes das aplicações de autômatos em compiladores, e mais desconfiados quando modelos simples como a máquina de Turing são apresentados como computadores genéricos. Evidentemente, o tratamento desses assuntos precisa ser atualizado.

Concretamente, as diferenças importantes em relação à primeira edição são estas:

- Os princípios básicos relacionados com projeto e análise de algoritmos são introduzidos informalmente já no Capítulo 1 (em conexão com os fechamentos) e questões algorítmicas são utilizadas ao longo de todo o livro. Há seções sobre problemas algorítmicos em conexão com autômatos finitos e gramáticas livres de contexto nos Capítulos 2 e 3 (incluindo minimização de estado e reconhecimento de estado livre do contexto), algoritmos para variantes fáceis de problemas de $\mathcal{NP}$-completos e uma seção que revisa técnicas algorítmicas para "lidar com a completude $\mathcal{NP}$" (algoritmos de caso especial, algoritmos de aproximação, reversão, ramificar-e-limitar, melhora local e algoritmos de recozimento simulados).
- O tratamento dado às máquinas de Turing no Capítulo 4 é mais informal e os argumentos de simulação são mais simples e mais quantitativos. Uma máquina de Turing, de acesso aleatório, é introduzida, ajudando a cobrir a lacuna entre os recursos desajeitados das máquina de Turing e o poder dos computadores e das linguagens de programação.
- Incluímos no Capítulo 5, sobre indecisibilidade, um pouco de teoria de funções recursivas (até o Teorema de Rice). As gramáticas e as funções

numéricas recursivas são introduzidas e são provadas como equivalente à máquina de Turing mais cedo e as provas estão mais simples. A impossibilidade na resolução dos problemas relacionados a gramáticas livre de contexto é provada por um argumento direto e simples, sem recorrer à questão da correspondência Post. Mantivemos o problema da organização, que revisitamos no capítulo sobre completude $\mathcal{NP}$.

◦ A complexidade é introduzida paulatinamente: no Capítulo 6, não definimos nenhum outro limite de tempo além dos polinomiais – assim, $\mathcal{P}$ é a primeira classe e o conceito de complexidade encontrado. A diagonalização, nesse caso, mostra que há problemas exponenciais que não em $\mathcal{P}$. Problemas práticos são introduzidos lado a lado com suas representações de linguagem (uma distinção que deliberadamente é obscurecida), e suas questões algorítmicas são examinadas extensamente.

◦ Há um capítulo separado sobre completude $\mathcal{NP}$ com um conjunto novo, extenso e, acreditamos, pedagogicamente útil de reduções dessa inteireza, culminando com o problema de equivalência para expressões regulares – fechando um círculo completo em torno do primeiro assunto do livro. Como mencionado antes, a obra acaba com uma seção sobre técnicas algorítmicas para "lidar com a completude $\mathcal{NP}$".

• Não há capítulos lógicos na nova edição. Esta foi uma decisão difícil, tomada por duas razões: de acordo com todas as evidências, esses capítulos eram os menos lidos e ensinados do livro; e hoje existem materiais que abordam melhor esse assunto. Entretanto, há um extenso tratamento da lógica booleana e seus problemas de satisfazibilidade no Capítulo 6.

◦ De maneira geral, as provas e exposições foram simplificadas, tornadas mais informais em alguns pontos-chave. Em várias ocasiões, como na prova da equivalência de linguagens livres de contexto e autômatos a pilha, longas provas técnicas de instruções indutivas tornaram-se exercícios. Há exercícios ao final de cada seção.

Como um resultado dessas alterações, existe agora, pelo menos, mais uma maneira de ensinar o conteúdo deste livro (além dos delineados na primeira edição e os que surgiram a partir de seu uso): um curso de um semestre de duração, objetivando a cobertura dos princípios básicos da teoria da computação *e* dos algoritmos, pode basear-se em uma seleção de conteúdos nos Capítulos de 2 a 7.

Queremos expressar nosso sincero agradecimento a todos os alunos e colegas que ofereceram *feedback*, idéias, indicações de erros e correções durante esses 15 anos – é impossível fazer uma lista completa. Agradecemos especialmente a Martha Sideri por sua ajuda na revisão do Capítulo 3. Além disso, somos muito gratos a nosso editor, Alan Apt, e à equipe da Prentice-Hall – Barbara Kraemer, Sondra Chavez e Bayani de Leon – que foram tão pacientes e prestativos.

Por fim, agradecemos o envio de relatórios de erro ou outros comentários, de preferência por correio eletrônico, para o endereço elements@cs.berkeley.edu. Confirmações de erros, correções e outras informações sobre o livro também podem ser obtidas enviando-se uma mensagem para esse endereço de correio eletrônico.

Prefácio à primeira edição

Este livro é uma introdução, em nível universitário, à teoria contemporânea clássica da computação. Resumidamente, os temas abordados são: a teoria de autômatos e linguagens formais, computabilidade por máquina de Turing e funções recursivas, não-computabilidade, complexidade computacional e lógica matemática. O tratamento é matemático, mas o ponto de vista é o de ciência da computação; assim, o capítulo sobre linguagens livres de contexto inclui uma discussão sobre análise sintática, e os capítulos sobre lógica estabelecem o vigor e a completude das soluções de prova de teorema.

No currículo universitário, a abordagem desse assunto tende a chegar atrasada, quando chega, e a ocorrer paralelamente aos cursos de projeto e análise de algoritmos. Nosso ponto de vista é de que os alunos de ciência da computação devem aprender esse assunto mais cedo – já no segundo ou terceiro ano – tanto pela visão mais profunda que ele oferece sobre tópicos específicos da ciência da computação como por servir para estabelecer paradigmas matemáticos essenciais. Mas descobrimos que ministrar um curso universitário rigoroso sobre o assunto é um empreendimento difícil por causa da maturidade matemática pressuposta pelos textos mais avançados. Nosso objetivo ao escrever este livro foi tornar o essencial do assunto acessível para uma ampla audiência universitária de maneira matematicamente vigorosa, mas sem pressupor nenhuma experiência matemática especial.

O livro representa cerca de um ano de trabalho em aula. Cada um de nós lecionou um curso de um semestre abrangendo grande parte do material que se encontra nos Capítulos de 1 a 6, omitindo, em várias ocasiões e em várias combinações, as seções sobre análise sintática, funções recursivas e problemas particulares sobre decisões insolúveis. Outras seleções são possíveis; por exemplo, um curso que pretenda enfatizar a computabilidade e as fundamentações da lógica mecânica talvez passe rapidamente pelos Capítulos de 1 a 3 e concentre-se nos Capítulos 4, 6, 8 e 9. Entretanto, se for utilizado, nossa maior esperança é que o livro possa contribuir para o desenvolvimento intelectual da próxima geração de cientistas da área da computação, introduzindo logo no início da sua formação o pensamento metódico e instigante sobre problemas computacionais.

Aproveitamos esta oportunidade para agradecer a todos com quem aprendemos, tanto os professores como os alunos. Somos especialmente gratos a Larry Denenberg e Aaron Temin, pelo seu trabalho de revisão dos originais, e a Michael Kahl e Oded Shmueli por sua ajuda e aconselhamento como assistentes de ensino. Na primavera de 1980, Albert Meyer ministrou um curso no M.I.T. a partir de um esboço deste livro, e agradecemos carinhosamente a ele por suas críticas e correções. Naturalmente, a culpa por quaisquer incorreções que restem são de nossa exclusiva responsabilidade. Renate D'Arcangelo digitou e ilustrou o manuscrito com seus característicos, mas extraordinários, perfeccionismo e rapidez.

Sumário

Introdução

Olhe à sua volta. A computação está em toda parte, todo o tempo. É iniciada por todo mundo e afeta a todos nós. Isso só acontece porque os cientistas descobriram nas últimas décadas métodos sofisticados de gerenciar recursos de computadores, permitir comunicação, traduzir programas, projetar *chips* e bancos de dados além de criar computadores e programas que são mais rápidos, mais baratos, mais fáceis de utilizar e mais seguros.

Como é normalmente o caso em todas as disciplinas importantes, o sucesso prático da ciência da computação baseia-se em suas *fundações sólidas e elegantes*. Na base das ciências físicas, estão perguntas fundamentais como *qual é a natureza da matéria?* e *qual é a base e a origem da vida orgânica?* A ciência da computação tem seu próprio conjunto de perguntas fundamentais: *O que é um algoritmo? O que pode e o que não pode ser computado? Quando um algoritmo deve ser considerado praticamente factível?* Por mais de 60 anos, (começando mesmo antes do advento do computador eletrônico) cientistas da computação têm ponderado estas questões e *sugerido respostas engenhosas que influenciaram profundamente a ciência da computação.*

O propósito deste livro é apresentar-lhe essas idéias, esses modelos e esses resultados fundamentais que permeiam a ciência da computação, os *paradigmas básicos* do nosso campo. Vale a pena estudá-los por muitas razões. Primeiro, grande parte da moderna ciência da computação é baseada, mais ou menos explicitamente, nesses paradigmas – e a maior parte do restante também deveria... Essas idéias e modelos também são belos e poderosos exemplos de aplicações matemáticas elegantes, produtivas e de valor duradouro. Além disso, são parte da história e do "subconsciente coletivo" do nosso campo, e é difícil entender a ciência da computação sem conhecer primeiro essas idéias e esses modelos.

Provavelmente, não deve surpreender o fato de que essas idéias e esses modelos são matemáticos por natureza. Embora um computador seja inegavelmente um objeto físico, também é verdade que muito pouca coisa útil pode ser dita sobre seus aspectos físicos, como suas moléculas e sua forma; as abstrações mais úteis de um computador são claramente matemáticas e assim o são, necessariamente, as técnicas utilizadas para argumentar sobre elas. Além disso, tarefas computacionais práticas requerem garantias estritas que só a matemática oferece (queremos que nossos compiladores traduzam corretamente, que nossos aplicativos terminem de modo adequado, e assim por diante). Entretanto, a matemática empregada na teoria da computação é bastante diferente da matemática utilizada em outras disciplinas aplicadas. Ela é geralmente discreta, no sentido de que a ênfase não está nos números reais e nas variáveis contínuas, mas em conjuntos finitos e seqüências. Ela é baseada em poucos conceitos elementares e extrai sua força e profundidade da manipulação paciente, cuidadosa e extensa, camada por camada, desses conceitos – exatamente como o computador. No pri-

meiro capítulo, você revisará esses conceitos e técnicas elementares (conjuntos, relações e indução, entre outros) e será introduzido ao estilo como eles são utilizados na teoria da computação.

Os dois Capítulos seguintes, 2 e 3, descrevem certos modelos de computação restritos, capazes de executar tarefas muito especializadas de manipulação de *string*, como dizer se uma determinada *string*, por exemplo a palavra *roto*, aparece em um dado texto, como as obras completas de Shakespeare; ou para testar se uma dada *string* de parênteses está adequadamente equilibrada – como () e (()) ()(), mas não)(). Esses dispositivos computacionais restritos (chamados *autômatos de estado finito* e *os autômatos de pilha*, respectivamente), sem dúvida, surgem na prática como componentes altamente otimizados e muito úteis de sistemas mais gerais como circuitos e compiladores. Aqui eles oferecem bons exercícios de preparação em nossa tarefa de construir uma definição formal genérica de algoritmo. Além disso, é instrutivo observar como o poder desses dispositivos se desvanece (ou, mais freqüentemente, é preservado) com a adição ou remoção de vários recursos, mais notadamente do *não-determinismo*, um aspecto intrigante da computação que é tão central quanto (paradoxalmente) não-realista.

No Capítulo 4, estudamos modelos gerais de algoritmos, dos quais o mais básico é a *máquina de Turing*[†], uma extensão relativamente simples dos dispositivos de manipulação de string dos Capítulos 2 e 3, que se tornam, surpreendentemente, uma estrutura geral para descrever algoritmos arbitrários. A fim de argumentar este ponto, conhecido como *tese de Church-Turing*, introduzimos modelos de computação cada vez mais elaborados (variantes mais poderosas da máquina de Turing, e até uma *máquina de Turing de acesso aleatório* e definições recursivas de funções numéricas) e mostramos que todas são equivalentes em poder ao modelo básico da máquina de Turing.

O capítulo seguinte trata da *indecisibilidade*, a surpreendente característica de certas tarefas computacionais próprias e bem-definidas residirem além do alcance da solução algorítmica. Por exemplo, suponha que lhe seja perguntado se podemos utilizar ladrilhos de uma dada lista finita de formas básicas para cobrir um plano inteiro. Se o conjunto de formas consistir em quadrados, ou qualquer triângulo regular, então a resposta é, obviamente, "sim". Mas, e se o conjunto consistir em algumas formas estranhas, ou se algumas forem obrigatórias, isto é, *devem* ser utilizadas pelo menos uma vez para a organização se qualificar? Isso é, seguramente, o tipo de pergunta complicada que gostaríamos de responder utilizando uma máquina. No Capítulo 5, empregamos o formalismo da máquina de Turing para provar que este e muitos outros problemas *simplesmente não podem ser resolvidos por computadores*.

Mesmo quando uma tarefa computacional é adequada para solução por *algum* algoritmo, pode ser o caso de que não haja um algoritmo *razoavelmente rápido, praticamente factível* que a resolva. Nos últimos dois capítulos deste livro, mostramos como problemas computacionais da vida real podem ser categorizados em termos de sua *complexidade*: certos problemas podem ser resolvidos dentro de limites de tempo *polinomiais* razoáveis, enquanto outros parecem requerer quantidades de tempo que crescem de modo astronômico, *exponencialmente*. No Capítulo 7, identificamos uma classe de problemas notoriamente difíceis, práticos e comuns, chamados $\mathcal{NP}$-completos (a questão do vendedor viajante é só um deles). Estabelecemos que todos esses problemas são *equivalentes* no sentido de que, se um deles tiver um algoritmo eficaz, então todos eles têm. É aceito de modo geral que todos os problemas $\mathcal{NP}$-completos são de complexidade inerentemente exponencial; quando essa conjectura é mesmo verdadeira, temos o famoso problema matemático $\mathcal{P} \neq \mathcal{NP}$, um dos mais importantes e profundos com que se defrontam os cientistas da computação de hoje.

Este livro dedica-se principalmente aos algoritmos e suas bases formais. Entretanto, o assunto algoritmos, envolvendo análise e projeto, é considerado, no currículo de ciência da computação de hoje, bem independente da teoria da computação. Na presente edição, tentamos restaurar a unidade do assunto. Como resultado, este livro também proporciona uma introdução um pouco especializada e não-convencional ao assunto algoritmos. Os algoritmos e sua análise são apresentados informalmente no Capítulo 1 e recapitulados, repetidas vezes no contexto dos modelos restritos de computação, estudados nos Capítulos 2 e 3, e dos problemas computacionais intrínsecos que eles geram. Dessa maneira, quando modelos gerais de algoritmos forem buscados mais tarde, o leitor estará em uma melhor posição para apreciar o escopo da missão e julgar seu êxito. Os algoritmos também desempenham um papel importante em nossa exposição sobre a complexidade, porque não há melhor maneira de apreciar um problema complexo do que compará-lo com outro, adequado a um algoritmo eficiente. O último capítulo culmina em uma seção sobre como lidar com a completude $\mathcal{NP}$, na qual apresentamos um conjunto de técnicas algorítmicas, utilizado com sucesso para resolver problemas $\mathcal{NP}$-completos (algoritmos de aproximação, algoritmos exaustivos, heurística de pesquisa local e assim por diante).

A computação é essencial, poderosa, bela, desafiadora, sempre em expansão – e assim também é sua teoria. Este livro conta apenas o começo de uma empolgante história. É uma modesta introdução para alguns temas básicos, cuidadosamente selecionados no baú de tesouros da teoria da computação. Esperamos que ele motive seus leitores a buscar mais; as referências no fim de cada tópico de capítulo são um bom lugar por onde começar.

[†] Nome atribuído em homenagem ao brilhante matemático e filósofo inglês Alan M. Turing (1912-1954), cujo papel seminal, em 1936, marcou o começo da teoria da computação. Turing também desbravou os campos da inteligência artificial e jogos de xadrez por computador, assim como a morfogênese na biologia, sendo responsável pela quebra de *Enigma*, o código naval alemão, durante a Segunda Guerra Mundial. Para descobrir mais coisas sobre sua vida fascinante (e seu fim trágico nas mãos da crueldade e intolerância oficiais), consulte o livro *Alan Turing: The Enigma*, de Andrew Hodges. Nova York: Schuster & Simon, 1983.

1 Conjuntos, relações e linguagens

1.1 CONJUNTOS

Dizem que a matemática é a linguagem da ciência – ela é certamente a linguagem da teoria da computação, a disciplina científica que estudaremos neste livro. E a linguagem da matemática lida com *conjuntos* e os modos complexos como se sobrepõem, se cruzam e, de fato, como eles próprios participam da formação de novos conjuntos.

Um *conjunto* é uma coleção de objetos. Por exemplo, a coleção de quatro letras a, b, c, d é um conjunto ao qual podemos atribuir o nome L; escrevemos $L = \{a, b, c, d\}$. Os objetos que constituem um conjunto são chamados *elementos* ou *membros* desse conjunto. Por exemplo, b é um elemento do conjunto L; em notação simbólica, $b \in L$. Às vezes, dizemos simplesmente que b está em L *ou* que L contém b. Por outro lado, z não é um elemento de L, e escrevemos $z \notin L$.

Em um conjunto, não distinguimos repetições dos elementos. Portanto, o conjunto {vermelho, azul, vermelho} é o mesmo conjunto que {vermelho, azul}. De maneira semelhante, a ordem dos elementos é irrelevante; por exemplo, {3, 1, 9}, {9, 3, 1} e {1, 3, 9} são o mesmo conjunto. Para resumir: dois conjuntos são iguais (isto é, o mesmo) se e somente se eles tiverem os mesmos elementos.

Os elementos de um conjunto não precisam estar relacionados de modo algum (além do fato de serem todos membros do mesmo conjunto); por exemplo, {3, vermelho, {*d*, azul}} é um conjunto com três elementos, um dos quais é outro conjunto. Um conjunto pode ter somente um elemento; ele é então chamado **unitário.** Por exemplo, {1} é o conjunto com 1 como seu único elemento; portanto {1} e 1 são bem diferentes. Há também um conjunto sem elementos. Naturalmente, pode haver somente um conjunto assim: ele é chamado **conjunto vazio** e é denotado por $\emptyset$. Qualquer outro conjunto que não o conjunto vazio é referido como não-vazio.

Até aqui, especificamos conjuntos simplesmente listando todos os seus elementos, separados por vírgulas e incluídos entre chaves. Alguns conjuntos não podem ser escritos dessa maneira porque eles são infinitos. Por exemplo, o conjunto dos números naturais $\mathbf{N}$ é infinito; podemos indicar seus elementos escrevendo $\mathbf{N} = \{0, 1, 2, \ldots\}$, utilizando reticências e sua intuição em lugar de uma lista infinitamente longa. Um conjunto que é não-infinito, é finito.

Outro modo de especificar um conjunto é fazendo referência a diferentes conjuntos e propriedades que os elementos podem ou não ter. Portanto, se $I = \{1,3,9\}$ e $G = \{3,9\}$, G pode ser descrito como o conjunto de elementos de I, que são maiores que 2. Escrevemos esse fato desta maneira:

$$G = \{x : x \in I \text{ e } x \text{ é maior que } 2\}.$$

Em geral, se um conjunto A foi definido, e P é uma propriedade que elementos de A podem ou não ter, então é possível definir um novo conjunto

$$B = \{x : x \in A \text{ e } x \text{ tem a propriedade } P\}.$$

Como outro exemplo, o conjunto de números naturais ímpares é

$$O = \{x : x \in \mathbf{N} \text{ e } x \text{ não é divisível por } 2\}.$$

Um conjunto A é um **subconjunto** de um conjunto B – em notação simbólica, $A \subseteq B$ – se cada elemento de A também é um elemento de B. Portanto $O \subseteq \mathbf{N}$, uma vez que todo número natural ímpar é um número natural. Note que qualquer conjunto é um subconjunto dele próprio. Se A é um subconjunto de B, mas A é diferente de B, dizemos que A é um **subconjunto próprio** de B, e escrevemos $A \subset B$. Note também que o conjunto vazio é um subconjunto de todos os conjuntos. Se B é qualquer conjunto, então $\emptyset \subseteq B$, uma vez que cada elemento de $\emptyset$ (do qual não há nenhum) também é um elemento de B.

Para comprovar que dois conjuntos A e B são iguais, podemos provar que $A \subseteq B$ e $B \subseteq A$. Todo elemento de A deve, então, ser um elemento de B e vice-versa, de modo que A e B tenham os mesmos elementos e $A = B$.

Dois conjuntos podem ser combinados para formar um terceiro por várias *operações de conjunto*, exatamente do mesmo modo que os números são combinados por operações aritméticas como adição. Uma operação de conjuntos é a **união:** a união de dois conjuntos é o grupamento que tem como elementos os objetos que são os mesmos de, pelo menos, um dos dois conjuntos dados e, possivelmente, de ambos. Utilizamos o símbolo $\cup$ para denotar união, de tal modo que

$$A \cup B = \{x: x \in A \text{ ou } x \in B\}.$$

Por exemplo,

$$\{1,3,9\} \cup \{3,5,7\} = \{1,3,5,7,9\}.$$

A **intersecção** de dois conjuntos é a coleção de todos os elementos que dois conjuntos têm em comum; isto é,

$$A \cap B = \{x: x \in A \text{ e } x \in B\}.$$

Por exemplo,

$$\{1,3,9\} \cap \{3, 5, 7\} = \{3\},$$

e

$$\{1,3,9\} \cap \{a,b,c,d\} = \emptyset.$$

Por fim, a **diferença** de dois conjuntos A e B, denotada por $A - B$, é o conjunto de todos os elementos de A que não são elementos de B.

$$A - B = \{x: x \in A \text{ e } x \notin B\}.$$

Por exemplo,

$$\{1,3,9\} - \{3,5,7\} = \{1, 9\}.$$

Certas propriedades das operações de conjunto são facilmente derivadas de suas definições. Por exemplo, se A, B e C são conjuntos, as seguintes leis se aplicam:

Idempotência	$A \cup A = A$
	$A \cap A = A$
Comutatividade	$A \cup B = B \cup A$
	$A \cap B = B \cap A$
Associatividade	$(A \cup B) \cup C = A \cup (B \cup C)$
	$(A \cap B) \cap C = A \cap (B \cap C)$
Distributividade	$(A \cup B) \cap C = (A \cap C) \cup (B \cap C)$
	$(A \cap B) \cup C = (A \cup C) \cap (B \cup C)$
Absorção	$(A \cup B) \cap A = A$
	$(A \cap B) \cup A = A$
Leis de DeMorgan	$A - (B \cup C) = (A - B) \cap (A - C)$
	$A - (B \cap C) = (A - B) \cup (A - C)$

Exemplo 1.1.1: Provemos a primeira das leis de DeMorgan. Se

$$L = A - (B \cup C)$$

e

$$R = (A - B) \cap (A - C);$$

podemos demonstrar que $L = R$. Fazemos isso, mostrando que (a) $L \subseteq R$ e (b) $R \subseteq L$.

(a) Se x for qualquer elemento de L, então $x \in A$, mas $x \notin B$ e $x \notin C$. Logo, x é um elemento de ambos, $A - B$ e $A - C$, e é, assim, um elemento de R. Portanto $L \subseteq R$.

(b) Se $x \in R$, então x é um elemento de ambos, $A - B$ e $A - C$, e está portanto, em A, mas não em B, nem C. Logo, $x \in A$, mas $x \notin B \cup C$, assim $x \in L$.

Portanto, $R \subseteq L$ e estabelecemos que $L = R$.

Dois conjuntos são **disjuntos**, se eles não tiverem nenhum elemento em comum, isto é, se sua intersecção é vazia.

É possível formar intersecções e uniões de mais que dois conjuntos. Se S é qualquer coleção de conjuntos, escrevemos $\bigcup S$ para aquele cujos elementos são os de todos os conjuntos em S. Por exemplo, se $S = \{\{a, b\}, \{b, c\}, \{c, d\}\}$, então $\bigcup S = \{a, b, c, d\}$; e, se $S = \{\{n\}: n \in \mathbf{N}\}$, isto é, a coleção de todos os conjuntos unitários com números naturais como elementos, então $\bigcup S = \mathbf{N}$. Em geral,

$$\bigcup S = \{ x: x \in P \text{ para algum conjunto } P \in S\}.$$

Similarmente,

$$\bigcap S = \{x: x \in P \text{ para cada conjunto } P \in S\}.$$

A coleção de todos os subconjuntos de um conjunto A é ela própria um conjunto, chamado **conjunto das partes** de A e denotado 2^A. Por exemplo, os subconjuntos de $\{c, d\}$ são $\{c, d\}$ em si, os conjuntos unitários $\{c\}$ e $\{d\}$ e o conjunto vazio $\emptyset$, assim,

$$2^{\{c,d\}} = \{\{c, d\}, \{c\}, \{d\}, \emptyset\}.$$

Uma **partição** de um conjunto não-vazio A é um subconjunto Π de 2^A, tal que $\emptyset$ não é um elemento de Π, e tal que cada elemento de A *está* em um, e somente um conjunto em Π. Isto é, Π é uma partição de A, se Π é um conjunto de subconjuntos de A, tal que

(1) cada elemento de Π é não-vazio;
(2) membros distintos de Π são disjuntos;
(3) $\bigcup\Pi = A$.

Por exemplo, $\{\{a, b\}, \{c\}, \{d\}\}$ é uma partição de $\{a, b, c, d\}$, mas $\{\{b, c\},\{c, d\}\}$ não é. Os conjuntos de números naturais pares e ímpares formam uma partição de **N**.

Problemas da Seção 1.1

1.1.1. Determine se cada um dos seguintes itens é verdadeiro ou falso.
(a) $\emptyset \subseteq \emptyset$
(b) $\emptyset \in \emptyset$
(c) $\emptyset \in \{\emptyset\}$
(d) $\emptyset \subseteq \{\emptyset\}$
(e) $\{a, b\} \in \{a, b, c, \{a, b\}\}$
(f) $\{a,b\} \subseteq \{a,b,\{a,b\}\}$
(g) $\{a, b\} \subseteq 2^{\{a,b,\{a,b\}\}}$
(h) $\{\{a, b\}\} \in 2^{\{a,b,\{a,b\}\}}$
(i) $\{a, b, \{a, b\}\} - \{a, b\} = \{a, b\}$

1.1.2. Quais são esses conjuntos? Escreva-os, utilizando somente chaves, vírgulas e numerais.
(a) $(\{1,3,5\} \cup \{3,1\}) \cap \{3,5,7\}$
(b) $\bigcup\{\{3\},\{3,5\}, \bigcap \{\{5,7\},\{7,9\}\}\}$
(c) $(\{1,2,5\}-\{5,7,9\}) \cup (\{5,7,9\}-\{1,2,5\})$
(d) $2^{\{7,8,9\}} - 2^{\{7,9\}}$
(e) $2^{\emptyset}$

1.1.3. Prove cada um dos seguintes:
(a) $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
(b) $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
(c) $A \cap (A \cup B) = A$
(d) $A \cup (A \cap B) = A$
(e) $A - (B \cap C) = (A - B) \cup (A - C)$

1.1.4. Se $S = \{a,b,c,d\}$.
(a) Que partição de S tem menos membros? Qual tem mais membros?
(b) Liste todas as partições de S com exatamente dois membros.

1.2 RELAÇÕES E FUNÇÕES

A matemática lida com afirmações sobre objetos e as relações entre eles. É natural dizer, por exemplo, que "menor que" é uma relação entre objetos de um certo tipo – a saber, números – a qual se aplica entre 4 e 7, mas não entre 4 e 2, ou entre 4 e ele próprio. Porém, como podemos expressar relações entre objetos na única linguagem matemática de que dispomos neste ponto – isto é, uma linguagem de conjuntos? Simplesmente consideramos uma relação como sendo em si um conjunto. Os objetos que pertencem à relação são, em essência, as combinações de indivíduos para a qual essa relação se aplica no sentido intuitivo. Assim, a relação menor é o conjunto de todos os pares de números tais que o primeiro número é menor que o segundo.

Entretanto, estamos caminhando um pouco rápido demais. Em um par que pertence a uma relação, precisamos ser capazes de distinguir as duas partes do par, e não explicamos como fazer isso. Não podemos escrever esses pares como conjuntos, uma vez que $\{4,7\}$ é a mesma coisa que $\{7,4\}$. É mais fácil introduzir um novo dispositivo para agrupar objetos chamado **par ordenado.**†

Escrevemos o par ordenado de dois objetos a e b como (a,b); a e b são chamados **componentes** do par ordenado (a, b). O par ordenado (a, b) não é a mesma coisa que o conjunto $\{a, b\}$. Primeiro, a ordem importa: (a, b) *é* diferente de (b, a), onde $\{a, b\} = \{b, a\}$. Em segundo lugar, os dois componentes de um par ordenado não precisam ser distintos; $(7,7)$ é um par ordenado válido. Note que dois pares ordenados (a, b) e (c, d) são iguais somente quando $a = c$ e $b = d$.

O **produto cartesiano** de dois conjuntos, A e B, denotado por $A \times B$, é o conjunto de todos os pares ordenados (a, b), com $a \in A$ e $b \in B$. Por exemplo,

$$\{1,3,9\} \times \{b,c,d\} = \{(1, b), (1, c), (1, d), (3, b), (3, c), (3, d), (9, b), (9, c), (9, d)\}.$$

Uma relação binária sobre dois conjuntos A e B é um subconjunto de $A \times B$. Por exemplo, $\{(1, b), (1, c), (3, d), (9, d)\}$ é uma relação binária em $\{1,3,9\}$ e $\{b, c, d\}$ e $\{(i, j): i, j \in \mathbf{N} \text{ e } i < j\}$ é a relação *menor que*; é um subconjunto de $\mathbf{N} \times \mathbf{N}$ – freqüentemente dois conjuntos relacionados por uma relação binária são idênticos.

Mais geralmente, suponha que n seja qualquer número natural. Então, se $a_1, \ldots, a_n$ são quaisquer n objetos, não necessariamente distintos, $(a_1,\ldots,a_n)$ é uma **tupla ordenada;** para cada $i = 1,\ldots, n$, a_i é o i-ésimo componente de $(a_1,\ldots,a_n)$. Uma m-tupla ordenada $(b_1,\ldots,b_m)$, onde m é um número natural, é o mesmo que $(a_1,\ldots,a_n)$, se, e somente se, $m = n$ e $a_i = b_i$, para $i = 1,\ldots, n$. Portanto $(4, 4)$, $(4, 4, 4)$, $((4, 4), 4)$ e $(4,(4,4))$ são todos distintos. 2-tuplas

† Verdadeiros fundamentalistas veriam o par ordenado (a, b) não como um novo tipo de objeto, mas como idêntico a $\{a, \{a, b\}\}$.

ordenadas são a mesma coisa que os pares ordenados discutidos acima, e 3-, 4-, 5- e 6-tuplas ordenadas são chamadas **triplos**, **quádruplos**, **quíntuplos** e **sêxtuplos** ordenados, respectivamente. Por outro lado, uma seqüência é uma n-tupla ordenada para algum n não especificado (o **comprimento** da seqüência). Se $A_1, \ldots, A_n$ são quaisquer conjuntos, então o **produto cartesiano** n-**multiplicado** $A_1 \times \ldots \times A_n$ é o conjunto de todas as n-tuplas ordenadas $(a_1, \ldots, a_n)$, com $a_i \in A_i$, para cada $i = 1, \ldots, n$. No caso em que todos os A_i são o mesmo conjunto A, o produto cartesiano n-multiplicado $A \times \ldots \times A$ de A com ele próprio também é escrito A^n. Por exemplo, $\mathbf{N}^2$ é o conjunto de pares ordenados de números naturais. Uma **relação** n-**ária** em conjuntos $A_1, \ldots, A_n$ é um subconjunto de $A_1 \times \ldots \times A_n$; as relações 1-, 2- e 3-árias são chamadas **relações unária, binária e ternária,** respectivamente.

Outra idéia matemática fundamental é a de *função*. No nível intuitivo, uma função é uma associação de cada objeto de um tipo com um único objeto de outro tipo: de pessoas com suas idades, cães com seus donos, números com seus sucessores, e assim por diante. Mas, utilizando a idéia de uma relação binária como um conjunto de pares ordenados, podemos substituir essa idéia intuitiva por uma definição concreta. Uma função de um conjunto A em um conjunto B é uma relação binária R sobre A e B com a seguinte propriedade especial: para cada elemento $a \in A$, há *exatamente um* par ordenado em R com o primeiro componente a. Para ilustrar a definição, suponha que C seja o conjunto de cidades nos Estados Unidos e que S seja o conjunto de estados; e suponha que

$$R_1 = \{(x, y): x \in C, y \in S \text{ e } x \text{ é uma cidade no estado } y\},$$

$$R_2 = \{(x, y): x \in S, y \in C \text{ e } y \text{ é uma cidade no estado } x\}.$$

Então R_1 é uma função, uma vez que cada cidade está em um e somente um estado, mas R_2 não é uma função, já que alguns estados têm mais de uma cidade.†

Em geral, utilizamos letras como f, g e h para funções e escrevemos $f: A \to B$ para indicar que f é uma função de A para B. Chamamos A o **domínio** de f. Se a é qualquer elemento de A, escrevemos $f(a)$ para esse elemento b de B, tal que $(a, b) \in f$; uma vez que f é uma função, há exatamente um $b \in B$ com essa propriedade, assim, $f(a)$ denota um único objeto. O objeto $f(a)$ é chamado **imagem** de a sob f. Para especificar uma função $f: A \to B$, é suficiente determinar $f(a)$ para cada $a \in A$; por exemplo, para definir a função R_1 acima, é suficiente especificar, para cada cidade, o estado em que ela é localizada. Se $f: A \to B$ e A' é um subconjunto de A, então definimos $f[A'] = \{f(a): a \in A'\}$ (isto é, $\{b: b = f(a)$ para algum $a \in A'\}$). Chamamos $f[A']$ a **imagem** de A' sob f. O ***intervalo*** de f é a imagem de seu domínio.

Comumente, se o domínio de uma função é um produto cartesiano, um conjunto de parênteses é eliminado. Por exemplo, se $f: \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ é definido de tal modo que a imagem sob f de um par ordenado (m, n) é a soma de m e n, escrevemos $f(m, n) = m + n$ em vez de $f((m, n)) = m + n$, simplesmente por uma questão de conveniência na notação.

† Não consideramos Cambridge, Massachusetts, e Cambridge, Maryland, como a mesma cidade, mas duas cidades diferentes que casualmente têm o mesmo nome.

Se $f: A_1 \times A_2 \times \ldots \times A_n \to B$ é uma função e $f(a_1, \ldots, a_n) = b$, onde $a_i \in A_i$ para $i = 1, \ldots, n$ e $b \in B$, então, às vezes chamamos $a_1, \ldots, a_n$ de **argumentos** de f e b, de valor correspondente de f. Portanto, f pode ser especificado dando seu **valor** a cada n-tupla de argumentos.

Certos tipos de funções são de interesse especial. Uma função $f: A \to B$ é **injetora** se, para quaisquer dois elementos distintos $a, a' \in A$, $f(a) \neq f(a')$. Por exemplo, se C é o conjunto de cidades nos Estados Unidos, S é o conjunto de estados, e $g: S \to C$ é especificado por

$$g(s) = \text{a capital do estado } s$$

para cada $s \in S$; então g é injetora, uma vez que não há dois estados com a mesma capital. Uma função $f: A \to B$ é **sobrejetora**, se cada elemento de B é a imagem sob f de algum elemento de A. A função g especificada não é sobrejetora, mas a função R_1, definida acima, é, considerando que cada estado contém, pelo menos, uma cidade. Por fim, um mapeamento $f: A \to B$ é uma **bijeção** entre A e B, se ele for tanto injetora como sobrejetora; por exemplo, se C_0 é o conjunto de cidades capitais de estado, então a função $g: S \to C_0$ especificada, como antes, por

$$g(s) = \text{a capital do estado } s$$

é uma bijeção entre S e C_0.

A **inversa** de uma relação binária $R \subseteq A \times B$, denotado $R^{-1} \subseteq B \times A$, é simplesmente a relação $\{(b, a): (a, b) \in R\}$. Por exemplo, a relação R_2 definida acima é a inversa de R_1. Portanto, a inversa de uma função não necessariamente é uma função. No caso de R_1, sua inversa falha em ser uma função, pois alguns estados têm mais de uma cidade; isto é, há distintas cidades c_1 e c_2, tal que $R_1(c_1) = R_1(c_2)$. Uma função $f: A \to B$ também pode falhar em ter uma inversa se houver algum elemento $b \in B$, tal que $f(a) \neq b$ para **todo** $a \in A$. Se $f: A \to B$ é uma bijeção, porém, nenhuma dessas eventualidades pode ocorrer, e f^{-1} é uma função - na realidade, uma bijeção entre B e A. Além disso, $f^{-1}(f(a)) = a$ para cada $a \in A$, e $f(f^{-1}(b)) = b$ para cada $b \in B$.

Quando uma bijeção particularmente simples entre dois conjuntos é especificada, às vezes é possível ver um objeto no domínio e sua imagem no intervalo como virtualmente indistinguíveis: um pode ser visto como um outro nome ou um modo de reescrever o outro. Por exemplo, conjuntos unitários e 1-tuplas ordenadas são, estritamente falando, diferentes, mas não perdemos muito se ocasionalmente obscurecemos a distinção, devido à óbvia bijeção f, tal que $f(\{a\}) = (a)$ para qualquer unitário $\{a\}$. Esse tipo de bijeção é chamado *isomorfismo natural*; evidentemente essa não é uma definição formal, uma vez que a definição do que é "natural" e de quais distinções podem ser obscurecidas depende do contexto. Alguns exemplos ligeiramente mais complexos devem esclarecer melhor essa questão.

Exemplo 1.2.1: Para três conjuntos A, B e C quaisquer, há um isomorfismo natural de $A \times B \times C$ para $(A \times B) \times C$, a saber

$$f(a, b, c) = ((a, b), c)$$

para qualquer $a \in A$, $b \in B$ e $c \in C$.✧

Exemplo 1.2.2: Para quaisquer conjuntos A e B, há um isomorfismo natural ϕ de

$$2^{A \times B},$$

isto é, o conjunto de todas as relações binárias em A e B, para o conjunto

$$\{f: f \text{ é uma função de } A \text{ para } 2^B\}.$$

A saber, para qualquer relação $R \subseteq A \times B$, suponhà que $\phi(R)$ seja a função $f : A \to 2^B$, tal que

$$f(a) = \{b: b \in B \text{ e } (a, b) \in R\}.$$

Por exemplo, se S é o conjunto de estados, e $R \subseteq S \times S$ contém qualquer par ordenado de estados com uma borda comum, então, a função naturalmente associada $f: S \to 2^S$ é especificada por $f(s) = \{s': s' \in S \text{ e } s' \text{ compartilham uma borda com } s\}$. ✧

Exemplo 1.2.3: Às vezes consideramos a inversa de uma função $f: A \to B$ como uma função mesmo par, quando f não é bijetora. A idéia é considerar $f^{-1} \subseteq B \times A$ como uma função de B para 2^A, utilizando o isomorfismo natural descrito no Exemplo 1.2.2. Portanto, $f^{-1}(b)$ é, para qualquer $b \in B$, o conjunto de todo $a \in A$, tal que $f(a) = b$. Por exemplo, se R_1 é como definido acima – a função que atribui a cada cidade o estado em que ela está localizada – então $R_1^{-1}(s)$, onde s é um estado, é o conjunto de todas as cidades contidas nele.

Se Q e R são relações binárias, então sua composição $Q \circ R$, ou simplesmente QR, é a relação $\{(a, b) :$ para algum c, $(a, c) \in Q$ e $(c, b) \in R\}$. Note que a composição de duas funções $f: A \to B$ e $g: B \to C$ é uma função h de A para C, tal que $h(a) = g(f(a))$ para cada $a \in A$. Por exemplo, se f é a função que atribui a cada *cão* seu dono, e g atribui a cada pessoa sua idade, então $f \circ g$ atribui a cada cão a idade do seu dono. ✧

Problemas da Seção 1.2

1.2.1. Escreva explicitamente cada uma das seguintes expressões.
(a) $\{1\} \times \{1, 2\} \times \{1, 2, 3\}$
(b) $\emptyset \times \{1, 2\}$
(c) $2^{\{1, 2\}} \times \{1, 2\}$

1.2.2. Suponha que $R = \{(a, b), (a, c), (c, d), (a, a), (b, d)\}$. O que é $R \circ R$, a composição de R com ela própria? O que é R^{-1}, a inversa de R? R, $R \circ R$ ou R^{-1} é uma função?

1.2.3. Suponha que $f: A \to B$ e $g: B \to C$. Suponha que $h: A \to C$ seja sua composição. Em cada um dos seguintes casos, declare as condições necessárias e suficientes em f e g, de modo que h seja conforme especificado.
(a) Sobrejetora.
(b) Injetora.
(c) Bijetora.

1.2.4. Se A e B são dois conjuntos quaisquer, escrevemos B^A para o conjunto de todas as funções de A para B. Descreva um isomorfismo natural entre $\{0, 1\}^A$ e 2^A.

1.3 TIPOS ESPECIAIS DE RELAÇÕES BINÁRIAS

As relações binárias serão encontradas com freqüência cada vez maior nestas páginas; assim, será útil dispor de maneiras convenientes de representá-las e de uma terminologia para discutir suas propriedades. Uma relação binária completamente "aleatória" não tem estrutura interna significativa alguma; mas muitas relações que encontraremos surgem de contextos específicos e, portanto, têm regularidades importantes. Por exemplo, a relação que se aplica entre duas cidades, se elas pertencem ao mesmo estado, têm certas "simetrias" e outras propriedades que merecem ser mencionadas, discutidas e exploradas.

Nesta seção, estudaremos relações que exibem essas regularidades e outras similares. Devemos tratar das relações binárias somente dentro de um conjunto e em relação a elas próprias. Portanto, suponha que A seja um conjunto, e $R \subseteq A \times A$ seja uma relação em A. A relação R pode ser representada por um **grafo dirigido.** Cada elemento de A é representado por um pequeno círculo – que chamamos de **vértice** do grafo dirigido – e uma seta é desenhada de a para b, se, e somente se, $(a, b) \in R$. As setas são os **arcos** do grafo dirigido. Por exemplo, a relação $R = \{(a, b), (b, a), (a, d), (d, c), (c, c), (c, a)\}$ é representada pelo grafo na Figura 1-1. Note, em particular, o laço de c para ele próprio, correspondente ao par $(c, c) \in R$. De um vértice do grafo a outro, há um ou nenhum arco – não permitimos "setas paralelas".

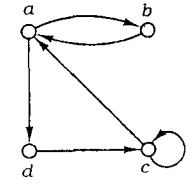


Figura 1-1

Não existe nenhuma distinção formal entre relações binárias em um conjunto A e grafos dirigidos com vértice de A. Utilizamos o termo *grafo dirigido* quando queremos enfatizar que o conjunto no qual a relação é definida não tem nenhum interesse independente para nós, fora do contexto dessa relação em particular. Grafos dirigidos, assim como os *grafos não-dirigidos* que logo serão apresentados, são úteis como modelos e abstrações de sistemas complexos (tráfego e redes de comunicação, estruturas e processos computacionais etc.). Na Seção 1.6 e, em muito mais detalhes, nos Capítulos 6 e 7, discutiremos muitos *problemas computacionais* interessantes que surgem em conexão com grafos dirigidos.

Para outro exemplo de uma relação binária/grafo dirigido, a relação menor que, ou igual a, $\leq$, definida nos números naturais, é ilustrada na

Figura 1-2. Naturalmente, o grafo dirigido não pôde ser desenhado em sua totalidade, uma vez que seria infinito.

Uma relação $R \subseteq A \times A$ é **reflexiva,** se $(a, a) \in R$ para cada $a \in A$. O grafo dirigido que representa uma relação reflexiva tem um laço de cada vértice para ele mesmo. Por exemplo, o grafo dirigido da Figura 1-2 representa uma relação reflexiva, mas o da Figura 1-1, não.

Uma relação $R \subseteq A \times A$ é **simétrica** se $(b, a) \in R$, sempre que $(a, b) \in R$.

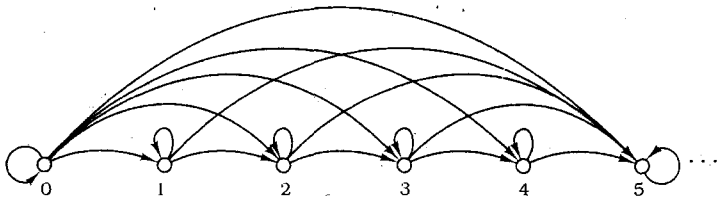


Figura 1-2

No grafo dirigido correspondente, sempre que há uma seta entre dois vértices, há setas entre esses vértices em ambas as direções. Por exemplo, o grafo dirigido da Figura 1-3 representa uma relação simétrica. Esse grafo dirigido pode representar a relação de "amizade" entre cinco pessoas, uma vez que sempre que x é um amigo de y, y também é um amigo de x. A relação de amizade não é reflexiva, já que não consideramos uma pessoa como amiga de si própria; evidentemente, uma relação poderia ser tanto simétrica como reflexiva; por exemplo, $\{(a, b)$: a e b são pessoas com o mesmo pai$\}$ é uma relação assim.

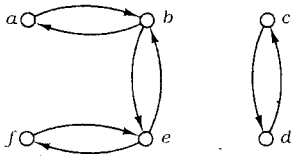


Figura 1-3

Uma relação simétrica *sem pares da forma* (a, a) é representada como um **grafo não-dirigido,** ou simplesmente um **grafo.** Grafos são desenhados sem pontas de seta, combinando pares de setas para trás e para frente entre os mesmos vértices. Por exemplo, a relação mostrada na Figura 1-3 poderia também ser representada pelo grafo na Figura 1-4.

Uma relação R é **anti-simétrica** sempre que $(a, b) \in R$, e a e b são distintos, então $(b, a) \notin R$. Por exemplo, suponha que P seja o conjunto de todas as pessoas. Então,

$\{(a, b)$: $a, b \in P$ e a é o pai de $b\}$

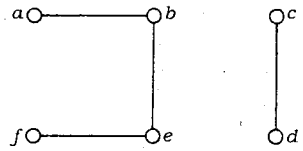


Figura 1-4

é anti-simétrica. Uma relação pode não ser simétrica, nem anti-simétrica; por exemplo, a relação

$\{(a, b)$: $a, b \in P$ e a é o irmão de $b\}$

e a relação representada na Figura 1-1 não são simétricas nem anti-simétricas.

Uma relação binária R é **transitiva** sempre que $(a, b) \in R$ e $(b, c) \in R$, então $(a, c) \in$ R. A relação

$\{(a, b)$: $a, b \in P$ e a é um ancestral de $b\}$

é transitiva, pois, se a é um ancestral de b, e b é um ancestral de c, então a é um ancestral de c. Assim é a relação menor que ou igual a. Em termos da representação do grafo dirigido, a transitividade é equivalente ao requisito de que, sempre que houver uma seqüência de arcos conduzindo de um elemento a para um elemento z, haverá uma seta diretamente de a para z. Por exemplo, a relação ilustrada na Figura 1-5 é transitiva.

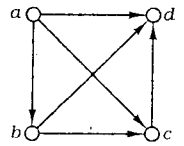


Figura 1-5

Uma relação que é reflexiva, simétrica e transitiva é chamada de **relação de equivalência.** A representação de uma relação de equivalência por um grafo não-dirigido consiste em um número de *componentes*; dentro de cada componente, cada par de vértices é conectado por um linha (veja a Figura 1-6). As "componentes" de uma relação de equivalência são chamadas de **classes de equivalência.** Normalmente, escrevemos $[a]$ para a equivalência de classe contendo um elemento a, desde que a relação de equivalência R seja compreendida pelo contexto. Isto é, $[a] = \{b$: $(a, b) \in R\}$, ou, uma vez que R é simétrica, $[a] = \{b$: $(b, a) \in R\}$. Por exemplo, a relação de equivalência na Figura 1-6 tem três classes de equivalência, uma com quatro, outra com três e outra com um elemento.

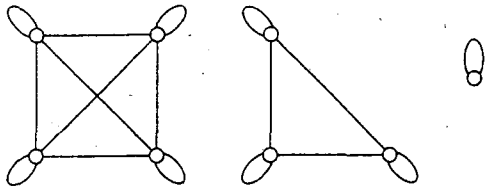


Figura 1-6

Teorema 1.3.1: *Suponha que R seja uma relação de equivalência em um conjunto A não-vazio. Então, as classes de equivalência de R constituem uma partição de A.*

Prova: Suponha que $\prod = \{[a]: a \in A\}$. Devemos demonstrar que os conjuntos em $\prod$ são não-vazios, disjuntos e que, juntos, esgotam A. Todas as classes de equivalência são não-vazias, desde que $a \in [a]$ para todos os $a \in A$, por reflexividade. Para demonstrar que eles são disjuntos, considere duas classes de equivalência distintas quaisquer, $[a]$ e $[b]$, e suponha que $[a] \cap [b] \neq \emptyset$. Portanto, há um elemento c, tal que $c \in [a]$, e $c \in [b]$. Logo, $(a, c) \in R$ e $(c, b) \in R$; já que R é transitiva, $(a, b) \in R$; e uma vez que R é simétrica, $(b, a) \in R$. Mas agora pegue qualquer elemento $d \in [a]$; então $(d, a) \in R$ e, por transitividade, $(d, b) \in R$. Assim, $d \in [b]$, de tal modo que $[a] \subseteq [b]$. Igualmente $[b] \subseteq [a]$. Por seguinte $[a] = [b]$. Mas isso contradiz o pressuposto de que $[a]$ e $[b]$ são distintos.

Para ver que $\bigcup \prod = A$, basta notar que cada elemento a de A *está em* algum conjunto em $\prod$ – ou seja, $a \in [a]$, por reflexividade. ■

Deste modo, começando de uma relação de equivalência R, podemos sempre construir uma partição $\prod$ correspondente. Por exemplo, se

$$R = \{(a, b): a \text{ e } b \text{ são pessoas, e } a \text{ e } b \text{ têm os mesmos pais}\},$$

então as classes de equivalência de R são todos os grupos de irmãos. Note que a construção do Teorema 1.3.1 pode ser invertida: a partir de qualquer partição, podemos construir uma correspondente relação de equivalência. Ou seja, se $\prod$ é uma partição de A, então

$$R = \{(a, b): a \text{ e } b \text{ pertencem ao mesmo conjunto de } \prod\}$$

é uma relação de equivalência. Assim, há um isomorfismo natural entre o conjunto de relações de equivalência em um conjunto A e o conjunto de partições de A.

Uma relação que é reflexiva, anti-simétrica e transitiva é chamada de uma **ordem parcial.** Por exemplo,

$$\{(a, b): a, b \text{ são pessoas e } a \text{ é um ancestral de } b\}$$

é uma ordem parcial (desde que consideremos cada pessoa como sendo um ancestral de si próprio). Se $R \subseteq A \times A$ é uma ordem parcial, um elemento $a \in A$ é chamado **mínimo** se o seguinte é verdadeiro: $(b, a) \in R$, somente se $a = b$. Por exemplo, na relação de ancestral definida acima, Adão e Eva são os únicos elementos mínimos. Uma ordem parcial finita deve ter, pelo menos, um elemento mínimo, mas uma ordem parcial infinita não precisa ter um.

Uma ordem parcial $R \subseteq A \times A$ é uma **ordem total** se, para todo $a, b \in A$, ou $(a, b) \in R$ ou $(b, a) \in R$. Portanto, a relação de ancestral não é uma ordem total, considerando que não há duas pessoas quaisquer que sejam ancestralmente relacionadas (por exemplo, irmãos não são); mas a relação menor que ou igual a, em números, é uma ordem total. Uma ordem total não pode ter dois ou mais elementos mínimos.

Um **caminho** em uma relação binária R é uma seqüência $(a_1, \ldots, a_n)$ para algum $n \geq 1$, tal que $(a_i, a_{i+1}) \in R$ para $i = 1, \ldots, n - 1$; esse caminho é referido como de a_1 para a_n. O **comprimento** de um caminho $(a_1, \ldots, a_n)$ é n. O caminho $(a_1, \ldots, a_n)$ é um **circuito**, se todos os a_i's são distintos e também $(a_n, a_1) \in R$.

Problemas da Seção 1.3

1.3.1. Suponha que $R = \{(a, c), (c, e), (e, e), (e, b), (d, b), (d, d)\}$. Desenhe grafos dirigidos representando cada um dos seguintes.
(a) R
(b) R^{-1}
(c) $R \cup R^{-1}$
(d) $R \cap R^{-1}$

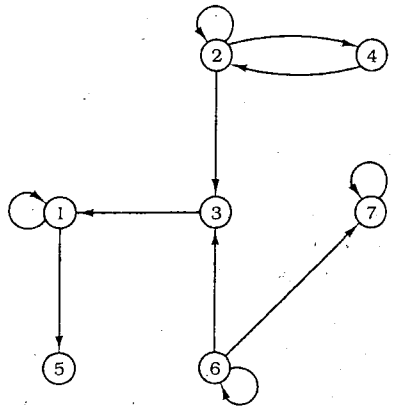
1.3.2. Suponha que R e S são as relações binárias em $A = \{1, \ldots, 7\}$ com as representações gráficas mostradas na próxima página.
(a) Indique se R e S são (i) simétricas, (ii) reflexivas e (iii) transitivas.
(b) Repita (a) para a relação $R \cup S$.

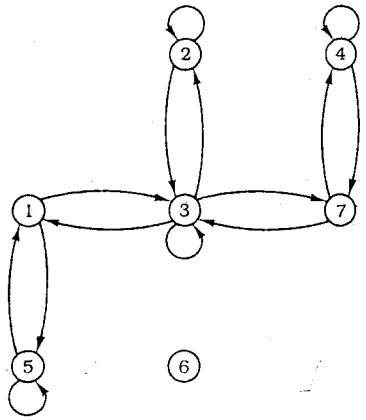
1.3.3. Desenhe grafos dirigidos representando relações dos seguintes tipos:
(a) Reflexiva, transitiva e anti-simétrica.
(b) Reflexiva, transitiva e nem simétrica nem anti-simétrica.

1.3.4. Suponha que A seja um conjunto não-vazio e que $R \subseteq A \times A$ seja o conjunto vazio. Que propriedades R tem?
(a) Reflexividade.
(b) Simetria.
(c) Anti-simetria.
(d) Transitividade.

1.3.5. Suponha que $f: A \rightarrow B$. Demonstre que a seguinte relação R é de equivalência em A: $(a, b) \in R$, se, e somente se, $f(a) = f(b)$.

1.3.6. Suponha que $R \subseteq A \times A$ seja uma relação binária, como definida abaixo. Em que casos R é uma ordem parcial? Em que casos R é uma ordem total?
(a) A = os inteiros positivos; $(a, b) \in R$ se e somente se b é divisível por a.
(b) $A = \mathbf{N} \times \mathbf{N}$; $((a, b) (c, d)) \in R$ se e somente se $a \leq c$ ou $b \leq d$.
(c) $A = \mathbf{N}$; $(a, b) \in R$ se e somente se $b = a$ ou $b = a + 1$.
(d) A é o conjunto de todas as palavras do idioma: $(a, b) \in R$ se e somente se a é não maior que b.





(e) A é o conjunto de todas as palavras do idioma; $(a, b) \in R$ se e somente se a é o mesmo que b, ou ocorre mais freqüentemente que b *no* presente livro.

1.3.7. Suponha que R_1 e R_2 sejam quaisquer duas ordens parciais no mesmo conjunto A. Demonstre que $R_1 \cap R_2$ é uma ordem parcial.

1.3.8. (a) Prove que, se S é qualquer coleção de conjuntos, então $R_S = \{(A, B) : A, B \in S, \text{ e } A \subseteq B\}$ é uma ordem parcial.
(b) Suponha que $S = 2^{\{1, 2, 3\}}$. Desenhe um grafo dirigido representando a ordem parcial R_S definida em (a). Quais são os elementos mínimos de R_S?

1.3.9. Sob que circunstâncias um grafo dirigido representa uma função?

1.3.10. Demonstre que qualquer função de um conjunto finito para ele próprio contém um circuito.

1.3.11. Suponha que S seja qualquer conjunto e que P seja o conjunto de todas as partições de S. Suponha também que R seja a relação binária em P, tal que $(\Pi_1, \Pi_2) \in R$, se e somente se, para todo $S_1 \in \Pi_1$, há um $S_2 \in \Pi_2$, tal que $S_1 \subseteq S_2$; se $(\Pi_1, \Pi_2) \in R$, dizemos que Π_1 **refina** Π_2. Demonstre que R é uma ordem parcial em P. Quais elementos de P são máximos e quais são mínimos? Suponha que P seja uma coleção arbitrária de subconjuntos de 2^S, os quais não precisam ser partições de S. R seria necessariamente uma ordem parcial?

1.4 CONJUNTOS FINITOS E INFINITOS

Uma propriedade básica de um conjunto finito é seu tamanho, isto é, o número de elementos que ele contém. Alguns fatos sobre os tamanhos de conjuntos finitos são tão óbvios que dificilmente precisam de prova. Por exemplo, se A $\subseteq$B, então o tamanho de A é menor ou igual ao de B; o tamanho de A é zero, se e somente se, A é um conjunto vazio.

Entretanto, uma extensão da noção de "tamanho" para conjuntos infinitos conduz a dificuldades se tentarmos seguir nossa intuição. Há mais múltiplos de 17 (0, 17, 34, 51, 68, ...) do que quadrados perfeitos (0, 1, 4, 9, 16, ...)? Você é bem-vindo a especular sobre alternativas, mas a experiência tem demonstrado que a única convenção satisfatória é considerar esses conjuntos como tendo o mesmo tamanho.

Chamamos dois conjuntos, A e B, **equinumerosos**, se há uma função bijetora $f: A \to B$. Lembre-se de que, se há uma bijeção $f: A \to B$, então há uma bijeção $f^{-1}: B \to A$; logo, a equinumerosidade é uma relação simétrica. De fato, como é facilmente demonstrado, ela é uma relação de equivalência. Por exemplo, {8, vermelho, {$\emptyset$, b}} e {1, 2; 3} são equinumerosos; suponha que f (8) = 1, f (vermelho) = 2, f ({$\emptyset$, b}) = 3. Assim são os múltiplos de 17 e os quadrados perfeitos; uma bijeção é dada por $f(17n) = n^2$ para cada $n \in \mathbf{N}$.

Em geral, chamamos um conjunto de finito se, intuitivamente, ele é equinumeroso com {1, 2, ..., n} algum número natural n. (Para n = 0, {1, ..., n} é o conjunto vazio, assim, $\emptyset$ é finito, sendo equinumeroso com ele próprio.) Se A e {1, ..., n} são equinumerosos, então dizemos que a cardinalidade de A (em símbolos, $|A|$) é n. A cardinalidade de um conjunto finito é, portanto, o número de elementos contidos nele mesmo.

Um conjunto é infinito se ele não é finito. Por exemplo, o conjunto de números naturais **N** é infinito, assim como o conjunto dos inteiros, o conjunto dos reais e o conjunto dos quadrados perfeitos. Entretanto, nem todos os conjuntos infinitos são equinumerosos.

Um conjunto é considerado contavelmente infinito, se ele é equinumeroso com **N** e, contável, se ele é finito, ou contavelmente infinito. Um conjunto que não é contável, é incontável. Para demonstrar que um conjunto A é contavelmente infinito, devemos exibir uma função bijetora f entre A e **N**; do mesmo modo, precisamos somente sugerir uma forma em que A pode ser enumerado como

$$A = \{a_0, a_1, a_2, \ldots\},$$

e assim por diante, uma vez que essa enumeração imediatamente sugere uma bijeção – simplesmente assuma $f(0) = a_0$, $f(1) = a_1$,...

Por exemplo, podemos demonstrar que a união de qualquer número finito de conjuntos, contavelmente infinitos, é contavelmente infinita. Permita-nos apenas ilustrar a prova para o caso de três conjuntos contavelmente infinitos e disjuntos dois a dois; um argumento similar funciona no caso geral. Chame os conjuntos *de A, B* e *C*. Eles podem ser listados como acima: $A = \{a_0, a_1, \ldots\}$, $B = \{b_0, b_1, \ldots\}$, $C = \{c_0, c_1, \ldots\}$. Então, sua união pode ser listada como $A \cup B \cup C$ $\{a_0, b_0, c_0, a_1, b_1, c_1, a_2, \ldots\}$. Essa listagem leva a um modo de "visitar" todos os elementos em $A \cup B \cup C$, alternando entre diferentes conjuntos, como ilustrado na Figura 1-7. A técnica de intercalar a enumeração de diversos conjuntos é chamada "formação em rabo de pombo" (por razões que qualquer carpinteiro pode entender depois de ver a Figura 1-7).

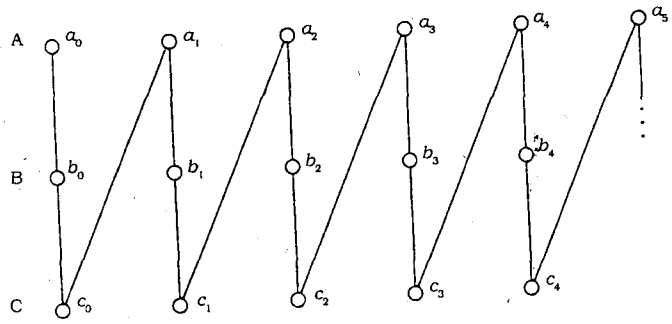


Figura 1-7

A mesma idéia pode ser utilizada para demonstrar que a união de uma coleção contavelmente infinita de conjuntos contavelmente infinitos é contavelmente infinita. Por exemplo, permita-nos demonstrar que $\mathbf{N} \times \mathbf{N}$ é contavelmente infinito; note que $\mathbf{N} \times \mathbf{N}$ é a união de $\{0\} \times \mathbf{N}$, $\{1\} \times \mathbf{N}$, $\{2\} \times \mathbf{N}$ e assim por diante, isto é, a união de uma coleção contavelmente infinita de conjuntos contavelmente infinitos. A formação em rabo de pombo deve ser mais sutil aqui do que no exemplo acima: não podemos, como fizemos lá, visitar um elemento de cada conjunto antes de visitar o segundo elemento do primeiro conjunto, porque com muitos conjuntos a visitar, poderíamos nunca terminar a primeira rodada! Em vez disso, procedemos da seguinte maneira (veja a Figura 1-8).

(1) Na primeira rodada, visitamos um elemento do primeiro conjunto: (0, 0).
(2) Na segunda rodada, visitamos o próximo elemento do primeiro conjunto, (0, 1) e também o primeiro elemento do segundo conjunto, (1, 0).
(3) Na terceira rodada, visitamos os próximos elementos não-visitados do primeiro e segundo conjuntos, (0, 2) e (1, 1) e também o primeiro elemento do terceiro conjunto, (2, 0).
(4) Em geral, na *n*-ésima rodada, visitamos o *n*-ésimo elemento do primeiro conjunto, o $(n - 1)$-ésimo elemento do segundo conjunto e o primeiro elemento do *n*-ésimo conjunto.

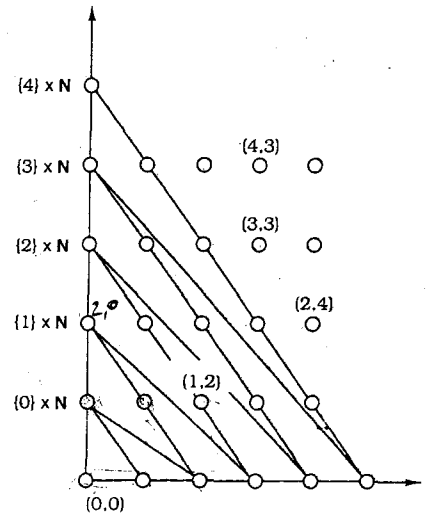


Figura 1-8

Outro modo de ver esse uso da formação em rabo de pombo é observar que o par (i, j) é o *m*-ésimo visitado, onde $m = \frac{1}{2}[(i+j)^2 + 3i + j]$; isto é, a função $f(i, j) = \frac{1}{2}[(i+j)^2 + 3i + j]$ é bijetora em $\mathbf{N} \times \mathbf{N}$ para $\mathbf{N}$ (veja o Problema 1.4.4).

No final da próxima seção, apresentamos uma técnica para demonstrar que dois conjuntos infinitos não são equinumerosos.

Problemas da Seção 1.4

1.4.1. Prove que os seguintes conjuntos são contáveis.
(a) A união de três conjuntos contáveis quaisquer, não necessariamente infinitos ou disjuntos.
(b) O conjunto de todos os subconjuntos finitos de $\mathbf{N}$.

1.4.2. Forneça explicitamente bijeções entre cada um dos seguinte pares.
(a) $\mathbf{N}$ e os números naturais ímpares.
(b) $\mathbf{N}$ e o conjunto de todos os inteiros.
(c) $\mathbf{N}$ e $\mathbf{N} \times \mathbf{N} \times \mathbf{N}$.
(Procuramos fórmulas que são o mais simples possível e envolvem apenas operações como adição e multiplicação.)

1.4.3. Suponha que *C* seja um conjunto de conjuntos, definido como segue:
1. $\emptyset \in C$
2. Se $S_1 \in C$ e $S_2 \in C$, então $\{S_1, S_2\} \in C$.
3. Se $S_1 \in C$ e $S_2 \in C$, então $S_1 \times S_2 \in C$.
4. Nada está em *C*, exceto aquilo que se deduz de (1), (2) e (3).
(a) Explique cuidadosamente por que é uma conseqüência de (1-4) o fato de que $\{\emptyset, \{\emptyset\}\} \in C$.

(b) Dê um exemplo de um conjunto de pares ordenados S tal que $S \in C$ e $|S| > 1$.
(c) C contém quaisquer conjuntos infinitos? Explique.
(d) C é contável ou incontável? Explique.

1.4.4. Demonstre que no método da formação em rabo de pombo da Figura 1-8, o par (i, j) é o m-ésimo a ser visitado, onde

$$m = \frac{1}{2}[(i+j)^2 + 3i + j].$$

1.5 TRÊS TÉCNICAS DE PROVA FUNDAMENTAIS

Toda prova é diferente, uma vez que é projetada para estabelecer um resultado distinto. Mas, como os jogos de xadrez ou beisebol, a observação de muitas leva-nos a perceber que há padrões, regras máximas e truques do negócio que podem ser encontrados e explorados com muita freqüência. O principal propósito desta seção é introduzir três princípios fundamentais recorrentes, sob várias nuances, em muitas provas: *indução matemática*, o *princípio da casa de pombo* e *diagonalização*.

O princípio da indução matemática: *Suponha que A seja um conjunto de números naturais tal que*

(1) $0 \in A$, e que,
(2) para cada número natural n, se $\{0, 1, \ldots, n\} \subseteq A$, então $n + 1 \in A$.

Portanto, $A = \mathbf{N}$.

Em termos menos formais, o princípio da indução matemática sustenta que qualquer conjunto de números naturais que contenha o zero e o $n+1$ (pelo fato de possuir todos os números até e incluindo n) deve ser de fato o conjunto de todos os números naturais.

A justificação para esse princípio deve ser intuitivamente clara: todo número natural deve acabar em A, já que ele pode ser "atingido", a partir do zero, em uma sucessão finita de etapas adicionando um de cada vez. Outro modo para argumentar a mesma idéia é por contradição; suponha que (1) e (2) se aplicam, mas $A \neq \mathbf{N}$. Então, algum número está omitido de A. Em particular, suponha que n seja o primeiro número entre 0, 1, 2,... que é omitido de $\mathbf{N}$.† Então, n não pode ser zero, uma vez que $0 \in A$ por (1); e uma vez que $0, 1, \ldots, n-1 \subseteq A$ pela escolha de n, então $n \in A$ por (2), o que é uma contradição.

† Este é o emprego de outro princípio, chamado *princípio do número mínimo*, que é de fato equivalente ao princípio de indução matemática, assim não iremos realmente "prover o princípio da indução matemática. O princípio do número mínimo é: Se $A \subseteq \mathbf{N}$ e $A \neq \mathbf{N}$, então há um único número mínimo $n \in \mathbf{N} - A$; isto é, o único número n tal que $n \notin A$ mas $0, 1, \ldots, n-1 \in A$. Um exemplo algo frívolo do princípio do número mínimo é o fato de que *não existem números não interessantes*. Se fizéssemos a suposição de que eles existem, então deveria existir um número mínimo assim, digamos n. Mas então, n teria a notável propriedade de ser o número mínimo não interessante, o que com certeza tornaria n interessante...

Na prática, a indução é utilizada para provar asserções desta forma: "Para todos os números naturais n, a propriedade P é verdadeira". O princípio acima é aplicado para o conjunto $A = \{n: P \text{ é verdadeiro de } n\}$ da seguinte maneira:
(1) Na *etapa de base*, demonstramos que $0 \in A$, isto é, que P é verdadeiro de 0.
(2) A *hipótese da indução* é o pressuposto de que, para algum fixo, mas arbitrário $n \geq 0$, P aplica-se a cada número natural $0, 1, \ldots, n$.
(3) Na *etapa da indução*, demonstramos, utilizando a hipótese da indução, que P é verdadeiro para $n + 1$. Pelo princípio da indução, A é, então, igual a $\mathbf{N}$, ou seja, P aplica-se a todo número natural.

Exemplo 1.5.1: Permita-nos demonstrar que, para qualquer $n \geq 0$, $1 + 2 + \ldots + n = \frac{n^2+n}{2}$.

Etapa de base. Suponha que $n = 0$. Então, a soma à esquerda é zero, pois não há nada para adicionar. A expressão à direita também é zero.

Hipótese da indução. Suponha que, para algum $n \geq 0$, $1 + 2 + \ldots + m = \frac{m^2+m}{2}$ sempre que $m \leq n$.

Etapa da indução.

$$\begin{aligned} 1 + 2 + \ldots + n + (n+1) &= (1 + 2 + \ldots + n) + (n+1) \\ &= \frac{n^2+n}{2} + (n+1) \text{ (pela hipótese da indução)} \\ &= \frac{n^2+n+2n+2}{2} \\ &= \frac{(n+1)^2+(n+1)}{2} \end{aligned}$$

como se queria demonstrar. ✧

Exemplo 1.5.2: Para qualquer conjunto finito A, $|2^A| = 2^{|A|}$, isto é, a cardinalidade do conjunto das partes de A é 2 elevado à cardinalidade de A. Devemos provar essa instrução *por indução* na *cardinalidade* de A.

Etapa de base. Suponha que A seja o conjunto de cardinalidade $n = 0$. Então, $A = \emptyset$ e $2^{|A|} = 2^0 = 1$; por outro lado, $2^A = \{\emptyset\}$ e $|2^A| = |\{\emptyset\}| = 1$.
Hipótese da indução. Suponha que $n > 0$ e que $|2^A| = 2^{|A|}$, desde que $|A| \leq n$.

Etapa da indução. Suponha que A seja semelhante a $|A| = n + 1$. Uma vez que $n > 0$, A contém, pelo menos, um elemento a. Suponha que $B = A - \{a\}$; então, $|B| = n$. Pela hipótese da indução, $|2^B| = 2^{|B|} = 2^n$. Agora, o conjunto das partes de A pode ser dividido em dois, os conjuntos contendo o elemento a e os conjuntos não contendo a. A última parte é simplesmente 2^B, e a primeira parte é obtida introduzindo a em cada membro de 2^B. Portanto,

$$2^A = 2^B \cup \{C \cup \{a\}: C \in 2^B\}.$$

Essa divisão, de fato, particiona 2^A em duas partes equinumerosas disjuntas. Assim, a cardinalidade do todo é duas vezes $2^{|B|}$, o que, pela hipótese da indução, é $2 \cdot 2^n = 2^{n+1}$, como se queria demonstrar. ✧

Em seguida, utilizamos a indução para estabelecer nosso segundo princípio fundamental, o *da casa de pombo.*

O princípio da casa de pombo: Se A e B *são conjuntos finitos*, e $|A| > |B|$, *então não há nenhuma função injetora de* A *para* B.

Em outras palavras, se tentarmos corresponder os elementos de A (os "pombos") com elementos de B (as casas de pombo), cedo ou tarde, teremos de colocar mais de um pombo em uma mesma casa.

Prova: *Etapa de base.* Suponha $|B| = 0$, isto é, $B = \emptyset$. Então, não há nenhuma função $f: A \to B$ *qualquer*, quanto mais, uma função injetora.

Hipótese da indução. Suponha que f não seja injetora sempre que $f: A \to B$, $|A| > |B|$ e $|B| \leq n$, onde $n \geq 0$.

Etapa da indução. Suponha que $f: A \to B$ e $|A| > |B| = n + 1$. Escolha algum $a \in A$ (uma vez que $|A| > |B| = n + 1 \geq 1$, A é não-vazio e, portanto, tal escolha é possível). Se houver outro elemento de A, digamos a', de forma que $f(a) = f(a')$, então, obviamente f não é uma função injetora e acabamos. Assim, suponha que a seja o único elemento mapeado por f para $f(a)$. Considere, então, os conjuntos $A - \{a\}$, $B - \{f(a)\}$ e a função g de $A - \{a\}$ para $B - \{f(a)\}$ que concorda com f em todos os elementos, de $A - \{a\}$. Agora a hipótese da indução se aplica, porque $B - \{f(a)\}$ tem n elementos e $|A - \{a\}| = |A| - 1 > |B| - 1 = |B - \{f(a)\}|$. Portanto, há dois elementos de $A - \{a\}$ distintos – $\{a\}$ que são mapeados por g (e, portanto, por f) para o mesmo elemento de $B - \{b\}$ e, assim, f não é injetora. ■

Esse simples fato é, surpreendentemente, muito utilizado em uma ampla variedade de provas. Apresentaremos apenas uma aplicação simples aqui, mas apontaremos outros casos, à medida que eles surgirem, nos capítulos seguintes.

Teorema 1.5.1: *Suponha que R seja uma relação binária em um conjunto finito A e que $a, b \in A$. Se há um caminho de a para b em R, então há um caminho com comprimento de, no máximo, $|A|$.*

Prova: Suponha que $(a_1, a_2, \ldots, a_n)$ é o caminho *mais curto* de $a_1 = a$ para $a_n = b$, isto é, o caminho com o menor comprimento, e suponha que $n > |A|$. Pelo princípio da casa de pombo, há um elemento de A que se repete no caminho, digamos, $a_i = a_j$ para algum $1 \leq i < j \leq n$. Mas, deste modo, $(a_1, a_2, \ldots, a_i, a_{j+1}, \ldots, a_n)$ é um caminho mais curto de a para b, contradizendo nosso pressuposto de que $(a_1, a_2, \ldots, a_n)$ é o caminho mais curto de a para b. ■

Por fim, chegamos à nossa terceira prova técnica básica, o *princípio da diagonalização.* Apesar de esse princípio não ser tão amplamente utilizado em matemática como os outros dois que acabamos de discutir, ele parece particularmente adequado para provar certos resultados importantes na teoria da computação.

O princípio da diagonalização: *Suponha que R seja uma relação binária em um conjunto A e que D, o conjunto diagonal para R, seja $\{a: a \in A \text{ e } (a, a) \notin R\}$. Para cada $a \in A$, suponha que $R_a = \{b: b \in A \text{ e } (a, b) \in R\}$. Logo, D é distinto de cada R_a.*

Se A é um conjunto finito, então R pode ser representado como um array quadrado; as linhas e as colunas são rotuladas com os elementos de A, e há um × na caixa com uma linha rotulada a e uma coluna rotulada b, apenas no caso $(a, b) \in R$. O conjunto diagonal D corresponde ao complemento da seqüência de caixas ao longo da diagonal principal, as caixas com ×, sendo substituídas por caixas sem × e vice-versa. Os conjuntos R_a correspondem às linhas do array. O princípio da diagonalização pode, em vista disso, ser repetido: o complemento da diagonal é diferente a partir de cada linha.

Exemplo 1.5.3: Considere a relação $R = \{(a, b), (a, d), (b, b), (b, c), (c, c), (d, b), (d, c), (d, e), (d, f), (e, e), (e, f), (f, a), (f, c), (f, d), (f, e)\}$; note que $R_a = \{b, d\}$, $R_b = \{b, c\}$, $R_c = \{c\}$, $R_d = \{b, c, e, f\}$, $R_e = \{a, e\}$, e $R_f = \{c, d, e\}$. Levando tudo isso em conta, R pode ser representado assim:

	a	b	c	d	e	f
a		×		×		
b		×	×			
c			×			
d		×	×		×	×
e					×	×
f	×		×	×	×	

A seqüência de caixas ao longo da diagonal é

	×	×		×	

Seu complemento é

×			×		×

o qual corresponde ao conjunto diagonal $D = \{a, d, f\}$. Na realidade, D é diferente a partir de cada linha do array; pois D, devido ao modo como é construído, difere a contar da primeira linha, na primeira posição, a partir da segunda linha, na segunda posição, e assim por diante. ✧

O princípio da diagonalização também se aplica a conjuntos infinitos, pela mesma razão: O conjunto diagonal D sempre difere do conjunto R_a quantó à questão de que a é um elemento e, assim, não pode ser o mesmo que R_a para qualquer a.

Ilustramos a utilização da diagonalização com um teorema clássico de Georg Cantor (1845-1918).

Teorema 1.5.2: *O conjunto $2^{\mathbf{N}}$ é incontável.*

Prova: Imagine que $2^{\mathbf{N}}$ é contavelmente infinito. Isto é, supomos que há um modo de enumerar todos os membros de $2^{\mathbf{N}}$ como

$$2^{\mathbf{N}} = \{R_0, R_1, R_2, \ldots\}$$

(note que esses são os conjuntos R_a na instrução do princípio da diagonalização, uma vez que consideramos a relação $R = \{(i, j): j \in R_i\}$). Agora observe o conjunto

$$D = \{n \in \mathbf{N}: n \notin R_n\}$$

(esse é o conjunto diagonal). D é um conjunto de números naturais e, portanto, deve aparecer em algum lugar na enumeração $\{R_0, R_1, R_2, \ldots\}$. Mas D não pode ser R_0, porque difere dele com relação ao fato de conter ou não 0 (ele contém se e somente se R_0 não contiver); e não pode ser R_1, porque difere dele com relação a 1; e assim por diante. Devemos concluir que D não aparece na enumeração em absoluto, e essa é a contradição.

Para reafirmar o argumento um pouco mais formalmente, admita que $D = R_k$ para algum $k \geq 0$ (uma vez que D é um conjunto de números naturais e supomos que $\{R_0, R_1, R_2, \ldots\}$ é uma enumeração completa de todos esses conjuntos, tal k deve existir). Chegamos a uma contradição, perguntando se $k \in R_k$:

(a) Suponha que a resposta seja sim, $k \in R_k$. Uma vez que $D = \{n \in \mathbf{N}: n \notin R_n\}$, segue-se que $k \notin D$; mas $D = R_k$, uma contradição.
(b) Suponha que a resposta seja não, $k \notin R_k$; então, $k \in D$. Mas D é R_k, assim $k \in R_k$, outra contradição.

Chegamos a essa contradição começando do pressuposto de que $2^{\mathbf{N}}$ é contavelmente infinito e continuando por um impecavelmente rigoroso raciocínio matemático; devemos, portanto, concluir que esse pressuposto foi um erro. Logo, $2^{\mathbf{N}}$ é incontável.■

Para uma diferente produção dessa prova, em termos de estabelecer que o conjunto de números reais no intervalo [0, 1] é incontável, veja o Problema 1.5.11.

Problemas da Seção 1.5

1.5.1. Demonstre por indução que:

$$1 \cdot 2 + 2 \cdot 3 + 2 \cdot 3 \cdot 4 + \ldots + n \cdot (n+1).(n+2) = \frac{n.(n+1).(n+2).(n+3)}{4}.$$

1.5.2. Demonstre por indução que $n^4 - 4n^2$ é divisível por 3 para todos os $n \geq 0$.

1.5.3. O que há de errado com a seguinte pretensa prova de que todos os cavalos são da mesma cor?
Prova por indução sobre o número de cavalos:
Etapa de base. Há somente um cavalo. Então, logicamente, todos os cavalos têm a mesma cor.
Hipótese da indução. Em qualquer grupo de até n cavalos, todos os cavalos têm a mesma cor.
Etapa da indução. Considere um grupo de $n + 1$ cavalos. Descarte um cavalo; pela hipótese da indução, todos os cavalos restantes têm a mesma cor. Agora traga de volta aquele cavalo e descarte outro; novamente todos os cavalos restantes têm a mesma cor. Assim, todos os cavalos têm a mesma cor daqueles que não são descartados a cada vez, e, assim, todos têm a mesma cor.

1.5.4. Demonstre que, se A e B são quaisquer conjuntos finitos, então há $|B|^{|A|}$ funções de A para B.

1.5.5. Prove por indução: toda ordem parcial em um conjunto finito não-vazio tem, pelo menos, um elemento mínimo. Essa afirmação precisa ser verdadeira se o requisito de finitude for eliminado?

1.5.6. Demonstre que, em qualquer grupo de, no mínimo, duas pessoas, há pelo menos duas que têm o mesmo número de conhecidos dentro do grupo. (Utilize o princípio da casa de pombo.)

1.5.7. Suponha que tentássemos provar, por meio de um argumento exatamente paralelo ao da prova do Teorema 1.5.2, que o conjunto de todos os subconjuntos finitos de $\mathbf{N}$ é incontável. O que sai errado?

1.5.8. Dê exemplos para demonstrar que a intersecção de dois conjuntos contavelmente infinitos pode ser ou finita ou contavelmente infinita e que a intersecção de dois conjuntos incontáveis pode ser finita, contavelmente infinita ou incontável.

1.5.9. Demonstre que a diferença de um conjunto incontável e um conjunto contável é incontável.

1.5.10. Demonstre que, se S é qualquer conjunto, então há uma função injetora de S para 2^S, mas não vice-versa.

1.5.11. Demonstre que o conjunto de todos os números reais no intervalo [0, 1] é incontável. (*Dica*: Sabe-se que cada um desses números pode ser escrito em notação binária como uma seqüência infinita de 0s e 1s – tal como.0110011100000.... Suponha que uma enumeração dessas seqüências exista e crie uma seqüência "diagonal" mediante a "alternância" do i-ésimo bit da i-ésima seqüência.).

1.6 FECHAMENTOS E ALGORITMOS

Considere os dois grafos dirigidos R e R^* na Figura 1-9(a) e (b). R^* contém R; além disso, R^* é reflexivo e transitivo (enquanto R não o é). De fato, é fácil ver

que R^* é *o menor grafo dirigido possível* que tem essas propriedades – isto é, contém R, é reflexivo e é transitivo (por "menor" queremos dizer aquele com o menor número de arcos). Por essa razão, R^* é chamado *fechamento transitivo reflexivo* de R. Definimos esse útil conceito formalmente a seguir:

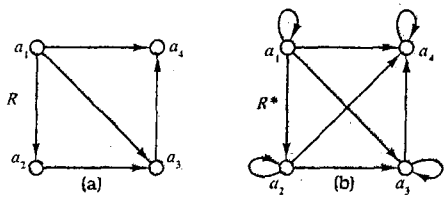


Figura 1-9

Definição 1.6.1: Suponha que $R \subseteq A^2$ seja um grafo dirigido definido no conjunto A. O **fechamento transitivo reflexivo** de R é a relação

$$R^* = \{(a, b) : a, b \in A \text{ e há um caminho de } a \text{ para } b \text{ em } R.\}$$

Note o interessante contraste entre a definição "a partir de baixo" articulada na Definição 1.6.1, e a definição informal "a partir de cima", em que introduzimos o fechamento transitivo reflexivo (a menor relação que contém R e é reflexiva e transitiva). Talvez seja intuitivamente claro que as duas definições são equivalentes. No final desta seção, devemos estudar mais profundamente essas "definições a partir de cima", e a razão por que elas sempre correspondem a uma definição "a partir de baixo" alternativa.

Algoritmos

A definição 1.6.1 imediatamente sugere um *algoritmo* para *computar* o fechamento transitivo reflexivo R^* de qualquer relação binária *dada* R sobre algum conjunto finito $A = \{a_1, a_2, \ldots, a_n\}$:

Inicialmente $R^* := \emptyset$
Para $i = 1, \ldots, n$ faça
  Para cada i-tupla $(b_1, \ldots, b_i) \in A^i$ faça
  se $(b_1, \ldots, b_i)$ é um caminho em R, então adicione (b_1, b_i) a R^*.

O estudo rigoroso de algoritmos é, em certo sentido, tudo de que trata este livro; mas dar a definição formal de um algoritmo neste ponto seria estragar a bela história que temos para contar. Até que um modelo formal geral para algoritmos seja introduzido, devemos tratá-los de maneira informal e um tanto não-rigorosa, aumentando gradualmente nossa visão e familiaridade. Então, ficará claro que nossa definição de algoritmo, quando sua hora chegar, na realidade, captura corretamente esse importante conceito. Seguindo essa idéia, a presente seção contém apenas um tratamento intuitivo de algoritmos e uma introdução informal à sua análise.

Felizmente, é fácil reconhecer um algoritmo quando você vê um. O procedimento que descrevemos acima para computar R^* é uma seqüência detalhada e não-ambígua de instruções que produz um resultado – o que chamamos R^*. Ele consiste em etapas elementares, as quais, podemos razoavelmente assumir, são fáceis de realizar: iniciando R^* como $\emptyset$, adicionando novos elementos a R^*, testando se $(b_j, b_{j+1}) \in R$ – este último tem de ser feito $i-1$ vezes na última linha do algoritmo para testar se $(b_1, \ldots, b_i)$ é um caminho de R. Supomos que, de algum modo, esse algoritmo opera em elementos de A e R diretamente, assim não precisamos especificar como tais conjuntos e relações são *representados* para a manipulação pelo algoritmo.

Devemos, em seguida, argumentar que a relação R^*, computada por esse algoritmo, é, na realidade, o fechamento transitivo reflexivo de R (isto é, o algoritmo está *correto*). A razão é que nosso algoritmo é somente uma simples e direta *implementação* da Definição 1.6.1. Ele adiciona a R^*, inicialmente vazio, todos os pares de elementos de A que são conectados pelo caminho em R. Possíveis caminhos são verificados, um por um, com o aumento do comprimento. Paramos em seqüências de comprimento n porque, pelo Teorema 1.5.1, se dois vértices de A estiverem conectados ao longo de um caminho, então há um caminho de comprimento n, ou menor, que os conecta.

É, portanto, claro que nosso algoritmo acabará terminando com a resposta correta. Uma questão que se deseja provar ser a mais importante e relevante para nossas preocupações neste livro é: *após quantas etapas ele terminará?* Na última parte do livro, devemos desenvolver toda uma teoria de como a resposta para tais questões é calculada, e quando o resultado é considerado satisfatório. Mas permita-nos proceder informalmente por enquanto.

O que precisamos é de uma indicação de quantas etapas elementares, quanto "tempo", o algoritmo requer quando recebe a relação R como uma entrada. Como é razoável esperar que o algoritmo levará mais tempo em relações de entrada maiores, a resposta dependerá do tamanho da relação de entrada – mais concretamente, dependerá do número n de elementos no conjunto A. Portanto, procuraremos a função f: $\mathbf{N} \to \mathbf{N}$, tal que para cada $n \geq 1$, se o algoritmo receber a relação binária $R \subseteq A \times A$ com $|A| = n$, ele terminará após, no máximo, $f(n)$ etapas. Como é típico na análise de algoritmos, devemos permitir que $f(n)$ seja uma superestimativa grosseira, contanto que ela tenha a correta *taxa de crescimento*. Taxas de crescimento são, assim, nosso próximo tópico.

O crescimento das funções

Destas três funções do conjunto de números naturais para ele próprio, qual é a maior?

$$f(n) = 1.000.000 \cdot n;\ g(n) = 10 \cdot n^3;\ h(n) = 2^n$$

Apesar de, para todos os valores de n até mais ou menos 12 termos $f(n) > g(n) > h(n)$, deve ser intuitivamente claro que a classificação correta é exata-

mente a oposta; se pegarmos *n grande o suficiente*, devemos acabar tendo $f(n) < g(n) < h(n)$. Nessa subseção, devemos desenvolver os conceitos necessários para classificar tais funções, de acordo com seu potencial, em última instância, de produzir valores grandes.

Definição 1.6.2: Suponha que f: $\mathbf{N} \to \mathbf{N}$ seja a função dos números naturais para os números naturais. A **ordem de** f, denotada $O(f)$, é o conjunto de todas as funções g: $\mathbf{N} \to \mathbf{N}$, com a propriedade que há números naturais positivos $c > 0$ e $d > 0$, tal que, para todos os $n \in \mathbf{N}$, $g(n) \leq c.\ f(n) + d$. Se, de fato, essa desigualdade se aplica a todo n, dizemos que $g(n) \in O(f(n))$, **com constantes** c e d.

Se para duas funções f, g: $\mathbf{N} \to \mathbf{N}$ temos que $f \in O(g)$ e $g \in O(f)$, então escrevemos $f \asymp g$. É claro que a relação $\asymp$ definida em funções é de equivalência: ela é reflexiva (porque sempre $f \in O(f)$, com constantes 1 e 0) e simétrica (devido ao fato de os papéis de f e g serem completamente intercambiáveis em uma definição de $\asymp$). Por fim, ela é transitiva. Porque, supondo que $f \in O(g)$, com constantes c, d, e $g \in O(h)$, com constantes c', d', então, para todos os n,

$$f(n) \leq c.\ g(n) + d \leq c.\ (c'\ h(n) + d') + d = (c.\ c')h(n) + (d + c.\ d').$$

Desse modo $f \in O(h)$ com constantes $c.\ c'$ e $d + c.\ d'$.

Portanto, todas as funções do conjunto de números naturais para ele próprio são particionadas por $\asymp$ em classes de equivalência. A classe de equivalência de f com relação $a \asymp$ é chamada **taxa de crescimento** de f.

Exemplo 1.6.1: Considere o *polinômio* $f(n) = 31n^2 + 17n + 3$. Estabelecemos que $f(n) \in O(n^2)$. Para ver isso, note que $n^2 \geq n$ e, assim, $f(n) \leq 48n^2 + 3$ e, portanto, $f(n) \in O(n^2)$ com constantes 48 e 3. Naturalmente, também $n^2 \in O(f(n))$ – com constantes 1 e 0. Logo, $n^2 \asymp 31n^2 + 17n + 3$, e as duas funções têm a mesma taxa de crescimento.

Similarmente, para qualquer polinômio de grau d com coeficientes não negativos

$$f(n) = a_d n^d + a_{d-1} n^{d-1} + \ldots + a_1 n + a_0$$

onde $a_i \geq 0$ para todos os i e $a_d > 0$, é fácil provar que $f(n) \in O(n^d)$ – com constantes $\sum_{i=1}^{d} a_i$ e a_0. *Todos os polinômios de mesmo grau têm a mesma taxa de crescimento.*

Considere, então, dois polinômios com diferentes graus; eles também têm a mesma taxa de crescimento? A resposta aqui é negativa. Uma vez que todos os polinômios com o mesmo grau têm a mesma taxa de crescimento, é suficiente levar em conta os dois mais simples representativos, a saber, dois polinômios das formas n^i e n^j, com $0 < i < j$. Obviamente, $n^i \in O(n^j)$ com constantes 1 e 0. Devemos provar que $n^j \notin O(n^i)$.

Suponha que, em função da contradição, na realidade, $n^j \in O(n^i)$ com constantes c e d. Isto é, para todos os $n \in \mathbf{N}$ $n^j \leq cn^i + d$. Mas descobre-se facilmente que isso é um absurdo, experimentando $n = c + d$:

$$c(c + d)^i + d < (c + d)^{i+1} \leq (c + d)^j.$$

Resumindo, para quaisquer dois polinômios f e g, se eles têm o **mesmo** grau, então $f \asymp g$. Caso contrário, se g tem grau maior que f, então $f \in O(g)$ mas $g \notin O(f)$; isto é, *g tem taxa de crescimento mais alta que f.* ✧

Exemplo 1.6.2: O exemplo anterior sugere que a taxa de crescimento de um polinômio é capturada por seu grau. Quanto maior o grau de f, mais alta a taxa de crescimento. Mas há funções com taxa de crescimento mais alta que qualquer polinômio. Um exemplo simples é a função *exponencial* 2^n. Vamos primeiro estabelecer que, para todos os $n \in \mathbf{N}$, $n \leq 2^n$. Devemos utilizar indução em n. O resultado certamente se aplica quando $n = 0$. Assim, suponha que isso se aplica a todos os números naturais até e incluindo n. Então, temos

$$n + 1 \leq 2^n + 1 \leq 2^n + 2^n = 2^{n+1},$$

onde, na primeira desigualdade, empregamos a hipótese da indução e, na segunda, utilizamos o fato de que $1 \leq 2^n$ para todo n.

Devemos agora estender esse simples fato para todos os polinômios. Devemos demonstrar que, para qualquer $i \geq 1$, $n^i \in O(2^n)$; isto é,

$$n^i \leq c\,2^n + d \qquad (1)$$

para constantes c e d apropriadas. Tome $c = (2i)^i$ e $d = (i^2)^i$. Há dois casos. Se $n \leq i^2$, então a desigualdade se aplica, porque $n^i \leq d$. Se, por outro lado, $n \geq i^2$, então devemos demonstrar que a desigualdade (1) novamente se aplica, porque agora $n^i \leq c^{2n}$. Na prova, suponha que m seja o quociente de n dividido por i – o único inteiro tal que $im \leq n < im + i$. Então, temos:

$$\begin{aligned} n^i &\leq (im + i)^i, \text{ pela definição de } m \\ &= i^i(m + 1)^i \\ &\leq i^i(2^{m+1})^i, \text{ pela desigualdade } n \leq 2^n \text{ aplicada a } n = m + 1 \\ &\leq c2^{mi}, \text{ lembrando que } c = (2i)^i \\ &\leq c2^n, \text{ novamente pela definição de } m. \end{aligned}$$

Portanto, a taxa de crescimento de qualquer polinômio não é mais rápida do que a de 2^n. Um polinômio pode ter a mesma taxa de crescimento que 2^n? Se pode, então qualquer polinômio de maior grau teria a mesma taxa de crescimento também; e dissemos no exemplo anterior que polinômios de diferentes graus têm diferentes taxas de crescimento. Concluímos que 2^n *tem uma taxa de crescimento mais alta que qualquer polinômio.* Outras funções exponenciais, tais como 5^n, n^n, $n!$, 2^{n^2}, ou, pior, 2^{2^n}, têm taxas de crescimento ainda mais altas. ✧

Análise de algoritmos

Taxas de crescimento polinomiais e exponenciais aparecem muito naturalmente na *análise de algoritmos*. Por exemplo, vamos derivar uma estimativa

grosseria do número de etapas exigidas pelo algoritmo para o fechamento transitivo reflexivo apresentado no início desta seção.

O algoritmo examina cada seqüência $(b_1, \ldots, b_j)$ de comprimento até n para determinar se ele é um caminho de R, e, se a resposta for "sim," ele adiciona (b_1, b_j) a R^*. Cada repetição pode definitivamente ser realizada em n ou menos "operações elementares" – testando se $(a, b) \in R$, adicionando (a, b) a R^*. Enfim, o número total de operações não pode ser maior que

$$n \cdot (1 + n + n^2 + \ldots + n^n),$$

o qual está em $O(n^{n+1})$. Determinamos, em vista disso, que a taxa de crescimento dos *requisitos de tempo* desse algoritmo é $O(n^{n+1})$.

Esse resultado é relativamente desapontador, pois é uma função *exponencial*, com uma taxa de crescimento mais alta ainda do que 2^n. Esse algoritmo não é eficiente em absoluto!

Logo surge a questão: há um algoritmo mais rápido para computar o fechamento transitivo reflexivo? Considere o seguinte:

Inicialmente $R^* := R \cup \{(a_i, a_i): a_i \in A\}$
Enquanto existirem três elementos $a_i, a_j, a_k \in A$, *tais que*
$(a_i, a_j), (a_j, a_k) \in R^*$, mas $(a_i, a_k) \notin R^*$ faça:
  adicione (a_i, a_k) a R^*.

Esse algoritmo é, talvez, mais natural e intuitivo do que o primeiro. Ele inicia adicionando a R^* todos os pares em R e todos os pares da forma (a_i, a_i) – garantindo, assim, que R^* contém R e é reflexivo. Ele, então, repetidamente, procura violações da propriedade de transitividade. Presumivelmente essa busca de violações é feita de uma maneira organizada, não claramente expressa no algoritmo, digamos, passando por todos os valores de a_i, para cada um deles, através de todos os valores de a_j e, por fim, de a_k. Se tal violação é descoberta, então ela é corrigida adicionando um par apropriado para R^*, e a busca por uma violação deve começar do zero novamente. Se, em algum ponto, todos os triplos são examinados e nenhuma violação é encontrada, o algoritmo termina.

Quando o algoritmo terminar, R^* certamente conterá R e será reflexivo; além disso, uma vez que nenhuma violação foi encontrada na última iteração do laço, R^* deve ser transitivo. Além do mais, R^* é a menor relação que tem todas essas propriedades (e é, portanto, o fechamento transitivo reflexivo de R buscado). Para ver o porquê, suponha que haja um subconjunto apropriado de R^*, chamado R_0, que contém R, sendo reflexivo e transitivo. Considere, então, o primeiro par que não pertence a R_0 e ainda foi adicionado a R^* pelo algoritmo. Ele não pode ser um par em R *ou* um par da forma (a_i, a_i), porque todos esses pares pertencem a R_0. Diante disso, ele deve ser um par (a_i, a_k), tal que ambos (a_i, a_j) e (a_j, a_k) pertencem a R^* nesse ponto – e, assim, também a R_0, pois (a_i, a_k) é o primeiro par em que as duas relações diferem. Mas então R_0, por hipótese uma relação transitiva, deve também conter (a_i, a_k) – uma contradição. Logo, o resultado será o fechamento transitivo reflexivo de R, e o algoritmo está correto.

Mas quando o algoritmo terminará? A resposta é *após, no máximo, n^2 iterações do laço*. Essa é a razão por que, depois de cada bem-sucedida iteração, um par (a_i, a_k) que não existia antes é adicionado a R^* e há, no máximo, n^2 pares a serem adicionados. Então, o algoritmo terminará após, no máximo, n^2 iterações. E, uma vez que cada iteração pode ser realizada no tempo $O(n^3)$ (todos os triplos dos elementos de A precisam ser pesquisados), a complexidade do novo algoritmo é $O(n^2 \times n^3) = O(n^5)$ – uma taxa de crescimento *polinomial* e, portanto, uma espetacular melhoria em relação à nossa primeira tentativa exponencial.

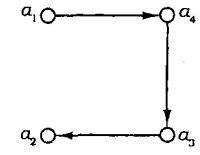


Figura 1-10

Exemplo 1.6.3: Vejamos o desempenho do novo algoritmo no grafo da Figura 1-10. Começamos na primeira linha com o grafo e todos os laços. Suponha que a busca de triplos (a_i, a_j, a_k) que violam a transitividade é conduzida na ordem "natural"

$$(a_1, a_1, a_1), (a_1, a_1, a_2), \ldots, (a_1, a_1, a_4), (a_1, a_2, a_1), (a_1, a_2, a_2), \ldots, (a_4, a_4, a_4).$$

A primeira violação descoberta, então, é (a_1, a_4, a_3), assim um arco (a_1, a_3) é adicionado. Em seguida, começamos a verificar todos os triplos do início – porque a introdução de (a_1, a_3) pode ter causado uma violação de transitividade em um triplo que foi examinado na iteração anterior. Na realidade, a próxima violação descoberta é (a_1, a_3, a_2), e assim (a_1, a_2) é adicionado. Começamos, novamente, tudo de novo desde o início, desta vez para descobrir uma violação para (a_4, a_3, a_2) – o arco (a_4, a_2) é adicionado. Na próxima iteração, passamos por todos os triplos sem descobrir uma violação, e logo concluímos que computamos o fechamento transitivo reflexivo do grafo. ✧

Esse exemplo ilustra a origem da ineficiência relativa do nosso algoritmo, refletida no íngreme crescimento n^5 de sua complexidade: um triplo (a_i, a_j, a_k) deve ser testado várias vezes para uma possível violação de transitividade, uma vez que um par recentemente inserido pode criar novas violações *em triplos que já foram verificados*.

Podemos fazer melhor? Em particular, há um modo de ordenar os triplos de tal maneira que, recentemente, pares adicionados nunca introduzam violações de transitividade nos triplos já examinados? Uma ordenação desse tipo produziria um algoritmo $O(n^3)$, porque cada triplo, então, precisaria ser examinado uma única vez.

Como veremos, tal organização é possível: ordenamos todos os triplos (a_i, a_j, a_k) para serem pesquisados com o incremento de j – o índice intermediário! Primeiro procuramos sistematicamente violações de transitividade da forma (a_i, a_1, a_k); as encontradas são corrigidas adicionando os pares

apropriados, e a busca continua do ponto onde parou. Uma vez que todos os triplos da forma (a_i, a_1, a_k) foram examinados, procuramos de todo jeito triplos da forma (a_i, a_2, a_k), depois (a_i, a_3, a_k), e assim por diante. A ordem exata em que os triplos, dentro de cada grupo, são examinados é insignificante. Uma vez que examinamos o último grupo, todos os triplos da forma (a_i, a_n, a_k), então paramos, tendo computado o fechamento transitivo reflexivo. O algoritmo completo é este:

Inicialmente: $R^* := R \cup \{(a_i, a_i): a_i \in A\}$
Para cada $j = 1, 2, \ldots, n$ faça
  Para cada $i = 1, 2, \ldots, n$ e $k = 1, 2, \ldots, n$ faça
  se $(a_i, a_j), (a_j, a_k) \in R^*$, mas $(a_i, a_k) \notin R^*$, então adicione (a_i, a_k) a R^*.

Por que essa idéia funciona? Considere um caminho $(a_{i0}, a_{i1}, \ldots, a_{ik-1}, a_{ik})$ de a_{i0} para a_{ik}, onde $1 \leq i_j \leq n$ para todos os j. Defina a *classificação* do caminho como sendo o maior inteiro entre $i_1, \ldots, i_{k-1}$; isto é, o maior índice aparecendo em qualquer vértice intermediário. Caminhos triviais, tais como arcos e laços, têm classificação zero, uma vez que eles não têm vértices intermediários.

Com essa terminologia podemos argumentar sobre a correção de nosso último algoritmo. Em particular, podemos provar a seguinte instrução: *para cada $j = 0, \ldots, n$, imediatamente após a j-ésima execução do laço externo, R^* contém todos os pares (a_i, a_k), tal que há um caminho de classificação j, ou menor, a partir de a_i para a_k em R.* Note que, uma vez que todos os caminhos têm classificação de no máximo n, isso implica que, no fim, todos os pares unidos por qualquer caminho terão sido adicionados a R^*.

A prova dessa instrução é por indução em j. O resultado é certamente verdadeiro quando $j = 0$ – ainda nenhuma iteração, e de acordo com isso, R^* contém somente pares conectados por caminhos triviais de classificação 0. Para a hipótese da indução, suponha que o resultado se aplique a j até algum valor, digamos, $m < n$, e considere a iteração $(m + 1^a)$. Devemos demonstrar que a iteração $(m + 1)$ adiciona a R^* precisamente esses pares que são conectados em R por caminhos de classificação iguais a $m + 1$, isto é, em que o vértice mais alto indexado é a_{m+1}. Se dois vértices, a_i e a_k são conectados por um caminho, de tal modo que devem ser conectados por caminho semelhante em que a_{m+1} aparece exatamente uma vez – se a_{m+1} aparece mais de uma vez, então omita a parte do caminho entre a primeira e a última aparição – enquanto todos os outros vértices intermediários são indexados, m ou menos. E tal caminho consistirá de um caminho de classificação m ou menos de a_i a a_{m+1}, seguido por um caminho de classificação m ou menos de a_{m+1} a a_k. Pela hipótese da indução, ambos os pares, (a_i, a_{m+1}) e (a_{m+1}, a_k), devem estar em R^* nesse ponto. Assim, o algoritmo descobrirá que $(a_i, a_{m+1}), (a_{m+1}, a_k) \in R^*$, mas $(a_i, a_k) \notin R^*$ e adicionará o par (a_i, a_k) a R^*. Inversamente, a iteração m + 1 adicionará a R^* somente pares (a_i, a_j) que sejam conectados por um caminho de classificação exatamente $m + 1$. O algoritmo está correto.

Exemplo 1.6.3 (continuação): Se aplicarmos esse algoritmo ao grafo na Figura 1-10, nenhum arco é adicionado quando $j = 1$ e $j = 2$. Então, quando $j = 3$, o arco (a_4, a_2) é adicionado, e quando $j = 4$, os arcos (a_1, a_3) e (a_1, a_4) são adicionadas. ✧

Fechamentos e propriedades de fechamento

O fechamento transitivo de uma relação é simplesmente um tipo importante de estilo para definir conjuntos maiores (ou relações), começando com conjuntos pequenos.

Definição 1.6.3: Imagine que D seja um conjunto e suponha que $n \geq 0$ e que $R \subseteq D^{n+1}$ seja uma relação $(n + 1)$-ária em D. Então se diz que um subconjunto B de D é **fechado** *sob* R, se $b_{n+1} \in B$, sempre que $b_1, \ldots, b_n \in B$ e $(b_1, \ldots, b_n, b_{n+1}) \in R$. Qualquer propriedade na forma "o conjunto B é fechado *sob* as relações $R_1, R_2, \ldots, R_m$" é chamada **propriedade de fechamento** de B.

Exemplo 1.6.4: O conjunto de ancestrais de uma pessoa é fechado sob a relação

$$\{(a, b): a \text{ e } b \text{ são pessoas, e } b \text{ é um pai de } a\},$$

uma vez que o pai de um ancestral também é um ancestral. ✧

Exemplo 1.6.5: Suponha que A seja um conjunto fixado. Dizemos que o conjunto S satisfaz a *inclusão da propriedade associada com A*, se $A \subseteq S$. Qualquer propriedade de inclusão é uma propriedade de fechamento, assumindo a relação R como sendo a relação unária $\{(a): a \in A\}$ (note que devemos pegar $n = 0$ na Definição 1.6.3). ✧

Exemplo 1.6.6: Devemos, ocasionalmente, dizer que um conjunto $A \subseteq D$ é fechado sob uma função $f: D^k \to D$. Não deve haver mistério quanto ao que isso significa, considerando que uma função é um tipo de relação especial. Por exemplo, podemos dizer que os números naturais são *fechados sob adição*. Com isso queremos dizer que qualquer $m, n \in \mathbf{N}$, também temos $m + n \in \mathbf{N}$, uma vez que $(m, n, m + n)$ é o triplo na "relação de adição" nos números naturais. $\mathbf{N}$ também é fechado sob multiplicação, mas ele não é fechado sob subtração. ✧

Exemplo 1.6.7: Uma vez que as relações são conjuntas, podemos falar de uma relação como sendo fechada sob uma ou outras mais. Suponha que D seja um conjunto e que Q seja a relação ternária em D^2 (isto é, um subconjunto de $(D \times D)^3$), tal que

$$Q = \{((a, b), (b, c), (a, c)): a, b, c \in D\}.$$

Então, uma relação $R \subseteq D \times D$ será fechada sob Q, se e somente se ela for transitiva. Concluímos que transitividade é uma propriedade de fechamento. Do mesmo modo, a reflexividade é uma propriedade de fechamento, porque ela é a propriedade de inclusão do conjunto $\{(d, d): d \in D\}$. ✧

Um tipo comum de construção matemática é passar de um conjunto A para *o conjunto minimal B, que contém A* e tem a propriedade P. Por "conjun-

to mínimo B" queremos dizer "o conjunto B que não inclui propriamente qualquer outro conjunto B', que também contém A e tem a propriedade P". Deve-se tomar cuidado quando uma definição dessa forma é utilizada, em que o conjunto B é bem definido, isto é, que existe somente um conjunto minimal assim. Uma vez que um conjunto de conjuntos pode ter muitos elementos mínimos ou absolutamente nenhum, o fato de B ser bem definido depende da natureza da propriedade P. Por exemplo, se P é a propriedade "tem b ou c como um elemento" e $A = \{a\}$, então B não é bem definido, já que ambos, $\{a, b\}$ e $\{a, c\}$, são conjuntos minimais com A como um subconjunto e com propriedade P.

Entretanto, como o seguinte resultado garante, se P é uma propriedade de fechamento, então o conjunto B é sempre bem definido:

Teorema 1.6.1: *Suponha que P seja a propriedade de fechamento definida por relações no conjunto D e suponha que A seja um subconjunto de D. Então, há um único conjunto minimal B, que contém A e tem a propriedade P.*

Prova: Considere o conjunto de todos os subconjuntos de D que são fechados sob $R_1, \ldots, R_m$ e que têm A como um subconjunto. Chamamos de S esse conjunto de conjuntos. Precisamos demonstrar que S tem um único elemento minimal B. É fácil ver que S não é um vazio porque compreende o "universo" D – ele próprio trivialmente fechado sob cada R_i e, certamente, contendo A.

Considere, então, o conjunto B, o qual é a intersecção de todos os conjuntos em S,

$$B = \bigcap S.$$

Primeiro, B é bem definido, pois é a intersecção de uma coleção de conjuntos não-vazios. Além disso, é fácil ver que ele contém A – uma vez que todos os conjuntos em S também o contêm. Em seguida, defendemos que B é fechado sob todos os R_i's. Suponha que $a_1, \ldots, a_{ni-1} \in B$ e $(a_1, \ldots, a_{ni-1}, a_{ni}) \in R_i$. Uma vez que B é a intersecção de todos os conjuntos em S, segue-se que todos os conjuntos em S contêm $a_1, \ldots, a_{ni-1}$. Mas, uma vez que todos os conjuntos em S são fechados sob R_i, todos eles contêm também a_{ni}. Portanto, B deve conter a_{ni} e, assim, B é fechado sob R_i. Por fim B é mínimo, porque não pode haver nenhum subconjunto adequado B' de B que essas propriedades (B' contém A e é fechado sob o R_i's). Então, B' seria um membro de S e portanto incluiria B. ■

Chamamos B, no Teorema 1.6.1 de **fechamento**, de A sob as relações $R_1, \ldots, R_m$.

Exemplo 1.6.8: O conjunto de todos os seus ancestrais (no qual supomos que cada pessoa é um ancestral de si próprio) é o fechamento do conjunto simples contendo somente você sob a relação $\{(a, b)$: a e b são pessoas e b é um pai de $a\}$. ✧

Exemplo 1.6.9: O conjunto dos números naturais **N** é o fechamento sob adição do conjunto {0, 1}. **N** é fechado sob adição e multiplicação, mas não sob subtração. O conjunto de inteiros (positivos, negativos e zero) é o fechamento de **N** sob subtração. ✧

Exemplo 1.6.10: O fechamento transitivo reflexivo de uma relação binária R sobre algum conjunto finito A, definido como

$$R^* = \{(a, b)\text{: há um caminho em } R \text{ de } a \text{ para } b\}$$

(lembre-se da Definição 1.6.1) realmente merece seu nome: ele se revela como o fechamento de R sob transitividade e reflexividade – ambas propriedades de fechamento.

Primeiro, R^* é reflexivo e transitivo; porque há um caminho trivial de a para a para qualquer elemento a, e, se há um caminho de a para b e um caminho de b para c, então há um caminho de a para c. Além disso, evidentemente $R \subseteq R^*$, porque há um caminho de a para b sempre que $(a, b) \in R$.

Por fim, R^* é minimal. Suponha que $(a, b) \in R^*$. Uma vez que $(a, b) \in R^*$, há um caminho $(a = a_1, \ldots, a_k = b)$ de a para b. Segue-se, por indução, em k que (a, b) deve pertencer a qualquer relação que inclui R, sendo transitiva e reflexiva.

O fechamento transitivo reflexivo de uma relação binária é somente um dos diversos fechamentos possíveis. Por exemplo, o *fechamento transitivo* de uma relação R, denotado R^+, é o conjunto de todos os (a, b), tal que há um *caminho não-trivial* de a para b em R – não precisa ser reflexivo. E o *fechamento transitivo simétrico reflexivo* de qualquer relação (não há símbolo especial) é sempre uma relação de equivalência. Como devemos demonstrar em seguida, há algoritmos polinomiais para computar todos esses fechamentos. ✧

Qualquer propriedade de fechamento sobre um conjunto finito pode ser computada em tempo polinomial! Suponha que nos sejam dadas as relações $R_1 \subseteq D^{r1}, \ldots, R_k \subseteq D^{rk}$ de várias aridades sobre o conjunto finito D e um conjunto $A \subseteq D$; e nos seja pedido para computar o fechamento A^* de A sob $R_1, \ldots, R_k$. Isso pode ser realizado em tempo polinomial, por meio de uma simples e direta generalização do algoritmo $O(n^5)$ que utilizamos para o problema do fechamento transitivo na última subseção:

```
Inicialmente A*:= A
enquanto existir um índice i, 1 ≤ i ≤ k e r_i elementos a_j1, ..., a_j(r_i-1) ∈ A*
    e a_jri ∈ D – A* tal que (a_j1, ..., a_jri) ∈ R_i faça:
        adicione a_jri a A*.
```

Trata-se de uma extensão simples e direta de nosso argumento para o algoritmo de fechamento transitivo, o qual deixamos como um exercício (Problema 1.6.9) para demonstrar que o algoritmo acima é correto e que ele termina após $O(n^{r+1})$ etapas, onde $n = |D|$ e r é o maior inteiro entre $r_1, \ldots, r_k$. Segue-se que o fechamento de qualquer conjunto finito dado, sob qualquer propriedade de fechamento definida em termos de relações *de aridade fixa*

pode ser computado em tempo polinomial. Aliás, no Capítulo 7 devemos provar uma instrução de *conversão* muito interessante: qualquer algoritmo de tempo polinomial pode ser produzido como a computação do fechamento de um conjunto sob algumas relações de aridade fixa. Em outras palavras, o algoritmo polinomial para fechamento, demonstrado acima, é *a mãe de todos os algoritmos polinomiais.*

Problemas da Seção 1.6

1.6.1. Os seguintes conjuntos são fechados sob as operações correspondentes? Se não, quais são os respectivos fechamentos?
(a) Os inteiros ímpares sob multiplicação.
(b) Os inteiros positivos sob divisão.
(c) Os inteiros negativos sob subtração.
(d) Os inteiros negativos sob multiplicação.
(e) Os inteiros ímpares sob divisão.

1.6.2. Qual é o fechamento transitivo reflexivo R^* da relação $R = \{(a, b), (a, c), (a, d), (d, c), (d, e)\}$? Desenhe um grafo dirigido representando R^*.

1.6.3. O fechamento transitivo do fechamento simétrico de uma relação binária é necessariamente reflexivo? Prove isso ou dê um exemplo contrário.

1.6.4. Suponha que $R \subseteq A \times A$ seja qualquer relação binária.
(a) Suponha que $Q = \{(a, b)$: $a, b \in A$ e há caminhos em R de a para b e de b para $a\}$. Demonstre que Q é uma relação de equivalência em A.
(b) Suponha que $\prod$ seja a partição de A correspondente à relação de equivalência Q. Suponha que R seja a relação $\{(S, T)$: $S, T \in \prod$ e que há um caminho em R de algum membro de S para algum membro de $T\}$. Demonstre que R é uma ordem parcial em $\prod$.

1.6.5. Dê um exemplo de uma relação binária que não é reflexiva mas tem um fechamento transitivo que é reflexivo.

1.6.6. Lembre-se das três funções em taxas de crescimento do início da subseção:

$$f(n) = 1.000.000.n; \quad g(n) = 10.n^3; \quad h(n) = 2^n.$$

Quais são as constantes c e d apropriadas para as inclusões $f(n) \in O(g(n))$, $f(n) \in O(h(n))$ e $g(n) \in O(h(n))$? Qual é o menor inteiro n, tal que os valores $f(n) \leq g(n) \leq h(n)$?

1.6.7. Organize essas funções na ordem ascendente da taxa de crescimento. Identifique qualquer função com a mesma taxa de crescimento: n^2, 2^n, $n\lfloor \log n \rfloor$, $n!$, 4^n, n^n, $n^{\lfloor \log n \rfloor}$, 2^{2^n}, 2^{2n}, $2^{2^{n+1}}$.

1.6.8. Você tem cinco algoritmos para um problema, com estes tempos de execução:

$$10^6 n, \quad 10^4 n^2, \quad n^4, \quad 2^n, \quad n!$$

(a) Seu computador executa 10^8 etapas por segundo. Qual é o maior tamanho n que você pode resolver por algoritmo em um segundo?
(b) Em um dia? (Suponha que um dia tenha 10^5 segundos).
(c) Como os números em (a) e (b) iriam alterar-se, caso você comprasse um computador dez vezes mais rápido?

1.6.9. Demonstre que o algoritmo dado no fim desta seção calcula corretamente o fechamento de um conjunto $A \subseteq D$ sob as relações $R_1 \subseteq D^{r_1}$, ..., $R_k \subseteq D^{r_k}$, em $O(n^r)$ tempo, onde $n = |D|$ e r é o máximo das aridades $r_1, \ldots, r_k$ – (*Dica:* O argumento é uma generalização simples e direta daquele para um algoritmo de fechamento transitivo $O(n^5)$.)

1.7 ALFABETOS E LINGUAGENS

Os algoritmos que estudamos informalmente na última seção deixaram muita coisa vaga. Por exemplo, não especificamos exatamente como as relações R e R^* que precisam para serem acessadas e modificadas são representadas e armazenadas. Na prática computacional, esses *dados* são codificados na memória do computador como strings de bits, ou outros símbolos apropriados para manipulação pela máquina. O estudo matemático da teoria da computação deve, portanto, começar pela compreensão da *matemática de strings de símbolos.*

Começamos com a noção de um **alfabeto**: um conjunto finito de **símbolos.** Um exemplo é, naturalmente, o alfabeto romano $\{a, b, \ldots, z\}$. Um alfabeto em particular, pertinente para a teoria da computação, é o **alfabeto binário** $\{0, 1\}$. De fato, qualquer objeto pode estar em um alfabeto; do ponto de vista formal, um alfabeto é simplesmente um conjunto finito de qualquer tipo. Para simplificar, porém, utilizamos como símbolos somente letras, numerais e outros caracteres comuns, tais como $ ou #.

Uma string em um alfabeto é uma seqüência finita de símbolos desse alfabeto. Em vez disso, ao escrever strings com parênteses e vírgulas, como escrevemos outras seqüências, simplesmente justapomos os símbolos. Portanto, *melancia* é uma string no alfabeto $\{a, b, \ldots, z\}$ e 0111011 é uma string em $\{0, 1\}$. Além disso, utilizando o isomorfismo natural, identificamos uma string de somente um símbolo com o próprio símbolo; portanto, o símbolo a é o mesmo que a string a. Uma string pode não ter nenhum símbolo; nesse caso, ela é chamada string **vazia** e é denotada por e. Geralmente utilizamos u, v, x, y, z e letras gregas para denotar strings; por exemplo, podemos empregar w como um nome para a string *abc*. Naturalmente, para evitar confusão é uma boa prática não utilizar como símbolos letras que também adotamos como nomes de strings. O conjunto de todas as strings, incluindo a string vazia, sobre um alfabeto Σ é denotado por Σ^*.

O **comprimento** de uma string é seu comprimento como uma seqüência; logo, o comprimento da string *acrd* é 4. Denotamos o comprimento de uma string w por $|w|$; portanto, $|101| = 3$ e $|e| = 0$. Alternativamente (isto é, via um isomorfismo natural), uma string $w \in \Sigma^*$ pode ser considerada como uma função w: $\{1, \ldots, |w|\} \to \Sigma$; o valor de $w(j)$, onde $1 \leq j \leq |w|$, é o símbolo na *j-ésima* posição de w. Por exemplo, se $w = accordion$, então $w(3) = w(2) = c$ e $w(1) = a$. Esse ponto de vista alternativo pode confundir as

coisas. Naturalmente, o símbolo c na terceira posição é idêntico ao que aparece na segunda. Se, porém, precisarmos distinguir símbolos idênticos em posições diferentes em uma string, devemos referir-nos a eles como diferentes **ocorrências** do símbolo. Isto é, o símbolo $\sigma \in \Sigma$ ocorre na posição *j-ésima* da string $w \in \Sigma^*$, se $w(j) = \sigma$.

Duas strings sobre o mesmo alfabeto podem ser combinadas para formar uma terceira pela operação de **concatenação.** A concatenação de strings x e y, que se escreve $x \circ y$, ou simplesmente xy, é a string x seguida pela string y; formalmente, $w = x \circ y$, se, e somente se, $|w| = |x| + |y|$, $w(j) = x(j)$ para $j = 1,..., |x|$ e $w(|x| + j) = y(j)$ para $j = 1, ..., |y|$. Por exemplo, $01 \circ 001 = 01001$ e *beach* $\circ$ *boy* = *beachboy*. Naturalmente, $w \circ e = e \circ w = w$ para qualquer string w. E a concatenação é associativa: $(wx)y = w(xy)$ para quaisquer strings w, x e y. Uma string v é uma **substring** de uma string w, se, e somente se, há strings x e y, tal que $w = xvy$. Tanto x como y poderiam ser e assim toda string é uma substring dela própria; e, assumindo $x = w$ e $v = y = e$, vemos que e é uma substring de toda string. Se $w = xv$ para algum x, então v é um **sufixo** de w; se $w = vy$ para algum y, então v é um **prefixo** de w. Portanto, *road* é um prefixo de *roadrunner*, o sufixo de *abroad* é uma substring dessas duas e de *broader*. Uma string pode ter diversas ocorrências da mesma substring; por exemplo, *ababab* tem três ocorrências de *ab* e duas de *abab*.

Para cada string w e cada número natural i, a string w^i é definida como

$$w^0 = e, \text{ a string vazia}$$
$$w^{i+1} = w^i \circ w \text{ para cada } i \geq 0$$

logo, $w^1 = w$ e $do^2 = dodo$.

Essa definição é nossa primeira instância de um tipo muito comum: **definição por indução**. Já vimos provas por indução, e a idéia subjacente é muito parecida. Há um caso de base da definição, aqui a especificação de w^i como $i = 0$; então, quando isso que está sendo definido foi determinado que para todo $j \leq i$, ele é definido para $j = i + 1$. No exemplo acima, w^{i+1} é explicitado em termos de w^i. Para ver exatamente como qualquer caso da definição pode ser rastreado para o caso de base, considere o exemplo de do^2. De acordo com a definição (com $i = 1$), $(do)^2 = (do)^1 \circ do$. Novamente, de acordo com a definição, (com $i = 0$) $(do)^1 = (do)^0 \circ do$. Agora o caso de base se aplica: $(do)^0 = e$. Assim, $(do)^2 = (e \circ do) \circ do = dodo$.

O **inverso** de uma string w, denotado por w^R, é a string "escrita de trás para a frente": por exemplo, $reverse^R = esrever$. Uma definição formal pode ser dada por indução no comprimento de uma string:

(1) Se w é uma string de comprimento 0, então $w^R = w = e$.
(2) Se w é uma string de comprimento $n + 1 > 0$, então $w = ua$ para algum $a \in \Sigma$ e $w^R = au^R$.

Vamos utilizar essa definição para ilustrar como uma prova por indução pode depender de uma definição por indução. Devemos demonstrar que para quaisquer strings w e x, $(wx)^R = x^R w^R$. Por exemplo, $(dogcat)^R = (cat)^R (dog)^R = tacgod$. Procedemos por indução sobre o comprimento de x.

Etapa de base. $|x| = 0$. Então, $x = e$, e $(wx)^R = (we)^R = w^R = ew^R = e^R w^R = x^R w^R$.

Hipótese da indução. Se $|x| \leq n$, então $(wx)^R = x^R w^R$.

Etapa da indução. Suponha que $|x| = n + 1$. Então, $x = ua$ para algum $u \in \Sigma^*$, e $a \in \Sigma$, tal que $|u| = n$.

$(wx)^R = (w(ua))^R$, uma vez que $x = ua$
$= ((wu)a)^R$, uma vez que a concatenação é associativa
$= a(wu)^R$ pela definição do inverso de $(wu)a$
$= au^R w^R$ pela hipótese da indução
$= (ua)^R w^R$ pela definição do inverso de ua
$= x^R w^R$, uma vez que $x = ua$

Agora passaremos do estudo das strings individuais para o de *conjuntos* finitos e infinitos de strings. Os modelos simples de máquinas de computação que devemos estudar em breve serão caracterizados em termos de regularidades no modo como tratam muitas strings diferentes; assim, é importante primeiro entender maneiras genéricas de descrever e combinar classes de strings.

Qualquer conjunto de strings em um alfabeto Σ – isto é, qualquer subconjunto de Σ^* – será chamado de **linguagem.** Portanto, Σ^*, $\emptyset$ e Σ são linguagens. Uma vez que uma linguagem é simplesmente um tipo especial de conjunto, podemos especificar uma linguagem finita listando todas as suas strings. Por exemplo, $\{aba, czr, d, f\}$ é a linguagem em $\{a, b, ..., z\}$. Entretanto, a maioria das linguagens de interesse são infinitas, de modo que não é possível listar todas as strings. Linguagens que podem ser consideradas são $\{0, 01, 011, 0111, ...\}$, $\{w \in \{0, 1\}^*: w \text{ tem um número igual de 0's e 1's}\}$ e $\{w \in \Sigma^*: w = w^R\}$. Desse modo, devemos especificar linguagens infinitas por meio do esquema

$$L = \{w \in \Sigma^*: w \text{ tem a propriedade } P\},$$

seguindo a forma geral que temos utilizado para especificar conjuntos infinitos.

Se Σ é um alfabeto finito, então Σ^* é certamente infinito; mas é ele um conjunto contavelmente infinito? Não é difícil ver que este é, na realidade, o caso. Para construir uma função bijetora f: $\mathbf{N} \rightarrow \Sigma^*$, primeiro fixe alguma ordenação do alfabeto, digamos $\Sigma = \{a_1, ..., a_n\}$, onde $a_1, ..., a_n$ são distintos. Os membros de Σ^*, podem então ser enumerados do seguinte modo:

(1) Para cada $k \geq 0$, todas as strings de comprimento k são enumeradas antes de todas as strings de comprimento $k + 1$.
(2) As n^k strings de comprimento exato k são enumeradas **lexicograficamente,** isto é, $a_{i_1} ... a_{i_k}$ precede $a_{j_1} ... a_{j_k}$, desde que, para algum m, $0 \leq m \leq k - 1$, $i_l = j_l$ para $l = 1, ..., m$, e $i_{m+1} < j_{m+1}$.

Por exemplo, se $\Sigma = \{0, 1\}$, a ordem seria como segue:

$$e, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, ...$$

Se Σ é o alfabeto romano, e a ordenação de $\Sigma = \{a_1, ..., a_{26}\}$ é a usual $\{a, ..., z\}$, então a ordem lexicográfica para strings de igual comprimento é a ordem

utilizada em dicionários; porém, a ordenação descrita por (1) e (2) para todas as strings em Σ^* difere da ordenação do dicionário por listar strings mais curtas antes das mais compridas.

Levando em conta que as linguagens são conjuntas, elas podem ser combinadas pelas operações de conjunto de união, intersecção e diferença. Quando um alfabeto particular Σ envolve um contexto, devemos escrever $\overline{A}$ — o **complemento** de A – em vez disso, a diferença $\Sigma^* - A$.

Além disso, certas operações são significativas somente para linguagens. A primeira dessas é a **concatenação de linguagens.** Se L_1 e L_2 são linguagens sobre Σ, sua concatenação é $L = L_1 \circ L_2$, *ou* simplesmente $L = L_1L_2$, onde

$$L = \{w \in \Sigma^* : w = x \circ y \text{ para algum } x \in L_1 \text{ e } y \in L_2\}.$$

Por exemplo, se $\Sigma = \{0, 1\}$, $L_1 = \{w \in \Sigma^*$: w tem um par número de 0's$\}$ e $L_2 = \{w$: w inicia com um 0, e os demais símbolos são 1's$\}$, então $L_1 \circ L_2 = \{w$: w tem um número ímpar de 0's$\}$.

Outra operação de linguagem é a **estrela de Kleene** de uma linguagem L, denotado por L^*. L^* é o conjunto de todas as strings obtidas pela concatenação de zero ou mais strings de L. (A concatenação de zero strings é e, e a concatenação de uma string é a própria string.) Portanto,

$$L^* = \{w \in \Sigma^* : w = w_1 \circ \ldots \circ w_k \text{ para algum } k \geq 0, \text{ e algum } w_1, \ldots, w_k \in L\}.$$

Por exemplo, se $L = \{01, 1, 100\}$, então $110001110011 \in L^*$, uma vez que $110001110011 = 1 \circ 100 \circ 01 \circ 1 \circ 100 \circ 1 \circ 1$, e cada uma dessas strings está em L.

Note que a utilização de Σ^* para denotar o conjunto de todas as strings sobre Σ é consistente com a notação para a estrela de Kleene de Σ, considerada como uma linguagem finita. Isto é, se supomos que $L = \Sigma$ e aplicamos a definição acima, então Σ^* é o conjunto de todas as strings que podem ser escritas como $w_1 \circ \ldots \circ w_k$ para algum $k \geq 0$, e algum $w_1, \ldots, w_k \in \Sigma$. Posto que os w_i são então simplesmente símbolos individuais em Σ, segue-se que Σ^* é, como originalmente definido, o conjunto de todas as strings finitas cujos símbolos estão em Σ.

Para outro exemplo extremo, observe que $\emptyset^* = \{e\}$. Suponha que $L = \emptyset$ na definição acima. A única possível concatenação $w_1 \circ w_2 \circ \ldots \circ w_k$ com $k \geq 0$ e $w_1, \ldots, w_k \in L$ é aquela com $k = 0$, isto é, a concatenação de zero strings; assim, o único membro de L^*, nesse caso, é e!

Como um exemplo final, permita-nos demonstrar que, se L é uma linguagem $\{w \in \{0, 1\}^*$: w tem um número desigual de 0's e 1's$\}$, então $L^* = \{0, 1\}^*$. Para ver isso, primeiro note que para quaisquer linguagens L_1 e L_2, se $L_1 \subseteq L_2$ então $L_1^* \subseteq L_2^*$, como é evidente a partir da definição da estrela de Kleene. Segundo, $\{0, 1\} \subseteq L$, uma vez que cada 0 e 1, considerados como uma string, tem um número desigual de 0's e 1's. Logo, $\{0, 1\}^* \subseteq L^*$; mas $L^* \subseteq \{0, 1\}^*$ por definição, assim $L^* = \{0, 1\}^*$.

Escrevemos L^+ para uma linguagem LL^*. De maneira equivalente, L^+ é a linguagem

$$\{w \in \Sigma^* : w = w_1 \circ w_2 \circ \ldots \circ w_k \text{ para algum } k \geq 1, \text{ e algum } w_1, \ldots, w_k \in L\}.$$

Note que L^+ pode ser considerado como o *fechamento* de L sob a função de concatenação. Ou seja, L^+ é a menor linguagem que inclui L e todas as strings que são concatenações de strings em L.

Problemas da Seção 1.7

1.7.1. (a) Prove, utilizando a definição de concatenação dada no texto, que a concatenação de strings é associativa.
(b) Dê uma definição indutiva da concatenação de strings.
(c) Utilizando a definição indutiva de (b), prove que a concatenação de strings é associativa.

1.7.2. Prove cada uma das seguintes afirmações utilizando a definição indutiva de inverso dada no texto.
(a) $(w^R)^R = w$ para qualquer string w.
(b) Se v é um substring de w, então v^R é um substring de w^R.
(c) $(w^i)^R = (w^R)^i$ para qualquer string w, e $i \geq 0$.

1.7.3. Suponha que $\Sigma = \{a_1, \ldots, a_{26}\}$ seja o alfabeto romano. Defina cuidadosamente a relação binária $<$ em Σ^*, tal que $x < y$, se, e somente se, x precedesse y em um dicionário padrão.

1.7.4. Demonstre cada uma das seguintes afirmações.
(a) $\{e\}^* = \{e\}$
(b) Para qualquer alfabeto Σ e qualquer $L \subseteq \Sigma^*$, $(L^*)^* L^*$.
(c) Se a e b são símbolos distintos, então $\{a,b\}^* = \{a\}^* (\{b\} \{a\}^*)^*$.
(d) Se Σ é qualquer alfabeto e $e \in L_1 \subseteq \Sigma^*$ e $e \in L_2 \subseteq \Sigma^*$, então $(L_1\Sigma^*L_2)^* = \Sigma^*$.
(e) Para qualquer linguagem L, $\emptyset L = L\emptyset = \emptyset$.

1.7.5. Dê alguns exemplos de strings dentro e fora destes conjuntos, onde $\Sigma = \{a, b\}$.
(a) $\{w$: para algum $u \in \Sigma\Sigma$, $w = uu^Ru\}$.
(b) $\{w$: $uw = wuw\}$.
(c) $\{w$: para algum u, $v \in \Sigma^*$, $uvw = wvu\}$.
(d) $\{w$: para algum $u \in \Sigma^*$, $uuw = uu\}$.

1.7.6. Sob que circunstâncias $L^+ = L^* - \{e\}$?

1.7.7. A estrela de Kleene de uma linguagem L é o fechamento de L sob que relações?

1.8 REPRESENTAÇÕES FINITAS DE LINGUAGENS

Uma questão central na teoria da computação é a representação de linguagens por especificações finitas. Naturalmente, qualquer linguagem finita é adequada para representação finita por enumeração exaustiva de todas as strings na linguagem. A questão torna-se desafiadora somente quando linguagens infinitas são consideradas.

Sejamos um pouco mais precisos sobre a noção de "representação finita de uma linguagem". O primeiro ponto a ser esclarecido é que qualquer

representação assim, deve, ela própria, ser uma string, uma seqüência finita de símbolos sobre algum alfabeto Σ. Segundo, nós certamente queremos que linguagens diferentes tenham representações diferentes; de outro modo, o termo *representação* dificilmente poderia ser considerado apropriado. Mas esses dois requisitos já significam que as possibilidades para representação finita são severamente limitadas. Como o conjunto Σ^* de strings sobre um alfabeto Σ é contavelmente infinito, o número de possíveis representações de linguagens é contavelmente infinito. (Esse permaneceria verdadeiro, mesmo se não fôssemos limitados a utilizar um alfabeto Σ particular, conquanto o número total de símbolos disponíveis fosse contavelmente infinito.) Por outro lado, o conjunto de todas as possíveis linguagens sobre um dado alfabeto Σ – isto é, 2^{Σ^*} – é incontavelmente infinito, uma vez que 2^N e, portanto, o conjunto das partes de qualquer conjunto contavelmente infinito não é contavelmente infinito. Com apenas um número contável de representações e um número incontável de coisas para representar, não somos capazes de representar todas as linguagens finitamente. Assim, o máximo que podemos esperar é encontrar representações finitas, de um tipo ou de outro, para, pelo menos, algumas das linguagens mais interessantes.

Esse é nosso primeiro resultado na teoria da computação: independentemente do poder dos métodos que utilizamos para representar linguagens, somente uma quantidade contável de linguagens pode ser representada, contanto que as próprias representações sejam finitas. Como a quantidade da linguagem sobre um alfabeto Σ é incontável, a vasta maioria delas, inevitavelmente, será perdida sob qualquer esquema representacional finito.

Naturalmente, esta não é a última coisa que temos a dizer nestas linhas. Devemos apresentar diversas maneiras de descrever e representar linguagens, cada uma mais poderosa que a última no sentido de que uma é capaz de descrever linguagens que a anterior não pode. Essa hierarquia não se contradiz com o fato de que todos os métodos representacionais finitos são inevitavelmente limitados em escopo pelas razões que acabamos de explicar.

Devemos também querer derivar maneiras de exibir linguagens particulares que não podem ser descritas pelos vários métodos representacionais que estudaremos. Sabemos que o mundo das linguagens é habitado por vastos números dessas espécies não-representáveis, mas, estranhamente, talvez, pode ser muito difícil escolher um, colocá-lo em exibição e documentá-lo. Os argumentos de diagonalização acabarão nos ajudando aqui.

Para começar nosso estudo de representações finitas, consideramos expressões – strings de símbolos – que mostram como as linguagens podem ser construídas empregando as operações apresentadas na seção anterior.

Exemplo 1.8.1: Suponha que $L = \{w \in \{0, 1\}^*: w$ tem duas ou três ocorrências de 1, sendo que a primeira e a segunda não são consecutivas$\}$. Essa linguagem pode ser descrita utilizando apenas conjuntos unitários simples e os símbolos ∪, ∘ e * como

$$\{0\}^* \circ \{1\} \circ \{0\}^* \circ \{0\} \circ \{1\} \circ \{0\}^* \circ ((\{1\} \circ \{0\}^*) \cup \emptyset^*).$$

Não é difícil ver que uma linguagem representada pela expressão acima é precisamente a linguagem L definida antes. A coisa importante a notar é

que os únicos símbolos utilizados nessa representação são as chaves (e), os parênteses (e), ∅, 0, 1, *, ∘ e ∪. De fato, podemos dispensar as chaves e o símbolo ∘ e escrever simplesmente

$$L = 0^*10^*010^*(10^* \cup \emptyset^*).$$

✧

A grosso modo, uma expressão como aquela para L no Exemplo 1.8.1, é chamada **expressão regular.** Isto é, uma expressão regular descreve uma linguagem exclusivamente por meio de símbolos únicos e ∅, combinados, talvez, com os símbolos ∪ e *, possivelmente com a ajuda de parênteses. Mas, a fim de manter corretas as expressões sobre as quais estamos falando e o "português da matemática" que estamos utilizando para discuti-las, devemos seguir essa linha cuidadosamente. Em vez de utilizar ∪, * e ∅, que são os nomes neste livro para certas operações e conjuntos, introduzimos os símbolos especiais ∪, * e ∅, os quais devem ser considerados por enquanto de modo inteiro, sem nenhum significado mais profundo, simplesmente como os símbolos *a*, *b* e 0 utilizados em exemplos anteriores neste livro. Do mesmo modo, introduzimos os símbolos especiais (e), em vez dos parênteses (e), que viemos utilizando até aqui para fazer matemática. Todas as expressões regulares sobre um alfabeto Σ^* são strings sobre o alfabeto $\Sigma \cup \{(,), \emptyset, \cup, *\}$ que podem ser obtidas como segue.

(1) ∅ e cada membro de Σ são uma expressão regular.
(2) Se α e β são expressões regulares, então assim é (αβ).
(3) Se α e β são expressões regulares, então assim é (α ∪ β).
(4) Se α é uma expressão regular, então assim é α*.
(5) Nada é uma expressão regular, a menos que deduzida de (1) a (4).

Toda expressão regular representa uma linguagem, de acordo com a interpretação dos símbolos ∪ e * como a união de conjunto e a estrela de Kleene, e da justaposição de expressões como concatenação. Formalmente, a relação entre expressões regulares e as linguagens que elas representam é estabelecida por uma função $\mathcal{L}$, tal que, se α é uma expressão regular qualquer, então $\mathcal{L}(\alpha)$ é uma linguagem representada por α. Isto é, $\mathcal{L}$ é uma função de strings para linguagens. A função $\mathcal{L}$ é definida como segue.

(1) $\mathcal{L}(\emptyset) = \emptyset$ e $\mathcal{L}(a) = \{a\}$ para cada $a \in \Sigma$.
(2) Se α e β são expressões regulares, então $\mathcal{L}((\alpha\beta)) = \mathcal{L}(\alpha)\,\mathcal{L}(\beta)$.
(3) Se α e β são expressões regulares, então $\mathcal{L}((\alpha \cup \beta)) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta)$.
(4) Se α é uma expressão regular, então $\mathcal{L}(\alpha^*) = \mathcal{L}(\alpha)^*$.

A declaração 1 define $\mathcal{L}(\alpha)$ para cada expressão regular, α que consiste em um único símbolo; se $n > 1$, então (2) a (4) definem $\mathcal{L}(\alpha)$ para expressões regulares de algum comprimento em termos de $\mathcal{L}(\alpha')$ para uma ou duas expressões regulares α' de menor comprimento. Portanto, toda expressão regular é associada dessa maneira com alguma linguagem.

Exemplo 1.8.2: O que é $\mathcal{L}(((a \cup b)^* a))$? Temos o seguinte.

$$\begin{aligned}\mathcal{L}(((a \cup b)^* a)) &= \mathcal{L}((a \cup b)^*)\,\mathcal{L}(a) \text{ por (2)} \\ &= \mathcal{L}((a \cup b)^*)\{a\} \text{ por (1)}\end{aligned}$$

$= \mathcal{L}((a \cup b))^*\{a\}$ por (4)
$= (\mathcal{L}(a) \cup \mathcal{L}(b))^* \{a\}$ por (3)
$= (\{a\} \cup \{b\})^* \{a\}$ por (1) duas vezes
$= \{a, b\}^*\{a\}$
$= \{w \in \{a, b\}^*: w \text{ termina com um } a\}$

✧

Exemplo 1.8.3: Que linguagem é representada por $(c^*(a \cup (bc^*))^*)$? Essa expressão regular representa o conjunto de todas as strings sobre $\{a,b,c\}$ que não têm a substring ac. Evidentemente nenhuma string em $\mathcal{L}((c^*(a \cup (bc^*))^*))$ pode conter a substring ac, uma vez que cada incidência de a em uma string desse tipo está no fim da string, ou é seguida por outra ocorrência de a ou é seguida por uma ocorrência de b.
Por outro lado, suponha que w seja uma string sem nenhuma substring ac. Então, w começa com zero ou mais c's. Se eles são removidos, o resultado é uma string sem nenhuma substring ac e a não-iniciação com c. Qualquer string desse tipo está em $\mathcal{L}((a \cup (bc^*))^*)$; para ela pode ser lida, da esquerda para a direita, como uma seqüência de a's, b's e c's, com quaisquer blocos de c's imediatamente depois dos b's (não depois dos a's e não no início da string). Portanto, $w \in \mathcal{L}((c^*(a \cup (bc^*))^*))$. ✧

Exemplo 1.8.4: $(0^* \cup (((0^*(1 \cup (11)))((00^*)(1 \cup (11)))^*)0^*))$ representa o conjunto de todas as strings sobre $\{0, 1\}$ que não têm a substring 111. ✧

Toda linguagem que pode ser representada por uma expressão regular pode ser representada por muitas delas infinitamente. Por exemplo, α e $(\alpha \cup \emptyset)$ sempre representam a mesma linguagem; bem como $((\alpha \cup \beta) \cup \gamma)$ e $(\alpha \cup (\beta \cup \gamma))$. Considerando que a união de conjunto e concatenação são operações associativas – isto é, uma vez que $(L_1 \cup L_2) \cup L_3 = L_1 \cup (L_2 \cup L_3)$ para todos os L_1, L_2, L_3 e o mesmo para a concatenação – normalmente omitimos os símbolos (e) extras em expressões regulares; por exemplo, tratamos $a \cup b \cup c$ como uma expressão regular mesmo que "oficialmente" ela não o seja. Para outro exemplo, a expressão regular do Exemplo 1.8.4 pode ser reescrita como $0^* \cup 0^*(1 \cup 11)(00^*(1 \cup 11))^*0^*$.

Além disso, agora mostramos que expressões regulares e as linguagens que elas representam podem ser definidas formalmente e sem ambigüidade; sentimo-nos livres, quando nenhuma confusão pode resultar, para obscurecer a distinção entre as expressões regulares e o "português matemático" que estamos utilizando para falar sobre linguagens. Assim, podemos dizer em um ponto que a^*b^* é o conjunto de todas as strings consistindo de algum número de a's seguido por algum número de b's – para ser preciso, deveríamos ter escrito $\{a\}^* \circ \{b\}^*$. Em outro ponto, podemos dizer que a^*b^* é uma expressão regular representando esse conjunto; nesse caso, para ser específico, deveríamos ter escrito (a^*b^*).

A classe das **linguagens regulares** sobre um alfabeto Σ é definida como consistindo de todas as linguagens L tal que $L = \mathcal{L}(\alpha)$ para alguma expressão regular α sobre Σ. Isto é, todas as linguagens regulares podem ser descritas por expressões regulares. Alternativamente, linguagens regulares podem ser pensadas em termos de fechamentos. A classe de linguagens regulares sobre Σ é precisamente o fechamento do conjunto de linguagens

$$\{\{\sigma\}: \sigma \in \Sigma\} \cup \{\emptyset\}$$

com relação às funções de união, concatenação e estrela de Kleene.

Já vimos que expressões regulares realmente descrevem algumas linguagens não-triviais e interessantes. Infelizmente, não podemos descrever algumas linguagens com o uso de expressões regulares que têm descrições muito simples por outras técnicas. Por exemplo, $\{0^n1^n: n \geq 0\}$ será mostrada no Capítulo 2 como sendo não-regular. Certamente qualquer teoria da representação finita de linguagens terá de acomodar, pelo menos, linguagens simples como essa. Portanto, expressões regulares são um método de especificação geralmente inadequado.

Na busca de um método amplo para finitamente especificar linguagens, podemos voltar ao nosso esquema básico

$$L = \{w \in \Sigma^*: w \text{ tem propriedade } P\}.$$

Mas que propriedades P devemos implementar? Por exemplo, o que torna as propriedades precedentes, "w consiste em um número de 0's seguido por um igual número de 1's" e "w não tem nenhuma ocorrência de 111" candidatos tão óbvios? O leitor pode ponderar sobre a resposta certa; mas vamos, por enquanto, permitir *propriedades algorítmicas* e somente essas. Isto é, para uma propriedade P de strings ser admissível como uma especificação de uma linguagem, deve haver um algoritmo para decidir se uma dada string pertence à linguagem. Um algoritmo que seja especificamente projetado para alguma linguagem L, objetivando responder questões na forma "A string w é um membro de L?" pode ser chamada de um **dispositivo de reconhecimento de linguagem**. Por exemplo, um dispositivo para reconhecer a linguagem

$$L = \{w \in \{0, 1\}^*: w \text{ não tem } 111 \text{ como uma substring}\}.$$

por meio da leitura de strings, um símbolo por vez, da esquerda para a direita, pode operar como isto:

Mantenha uma contagem que começa em zero e é configurada de volta para zero toda vez que um 0 é encontrado na entrada; adicionando um toda vez que um 1 é encontrado na entrada; pára com uma resposta Não se a contagem atinge três e pára com uma resposta Sim se toda a string é lida sem atingir a contagem três.

Um método alternativo e relativamente ortogonal de especificar uma linguagem é descrever como um espécime genérico na linguagem é produzido. Por exemplo, uma expressão regular tal como $(e \cup b \cup bb)(a \cup ab \cup abb)^*$ pode ser vista como um modo de gerar membros de uma linguagem:

Para produzir um membro de L, primeiro escreva nada ou b ou bb; então, escreva a ou ab ou abb e faça isso qualquer número de vezes, incluindo zero: todos e somente os membros de L podem ser produzidos dessa maneira.

Esses **geradores de linguagem** não são algoritmos, uma vez que não são projetados para responder a perguntas e não são completamente explícitos sobre o que fazer (como escolheremos qual de *a*, *ab* ou *abb* deve ser escrito?) Mas são meios importantes e úteis de representar linguagens da mesma maneira. A relação entre dispositivos de reconhecimento e geradores de linguagem, ambos os quais são tipos de especificações de linguagens finitas, é outro importante assunto deste livro.

Problemas da Seção 1.8

1.8.1. Quais linguagens são representadas pela expressão regular $(((a^*a)b)\cup b)$?

1.8.2. Reescreva cada uma destas expressões regulares como uma expressão mais simples, representando o mesmo conjunto.
(a) $\varnothing^*\cup a^*\cup b^*\cup(a\cup b)^*$
(b) $((a^*b^*)^*(b^*a^*)^*)^*$
(c) $(a^*b)^*\cup(b^*a)^*$
(d) $(a\cup b)^*a(a\cup b)^*$

1.8.3. Suponha que $\Sigma = \{a, b\}$. Escreva expressões regulares para os seguintes conjuntos:
(a) Todas as strings em Σ^* com não mais que três *a*'s.
(b) Todas as strings em Σ^* com um número de *a*'s divisíveis por três.
(c) Todas as strings em Σ^* com, exatamente, uma ocorrência da substring *aaa*.

1.8.4. Prove que, se L é regular, então, assim é $L' = \{w: uw \in L$ para alguma string $u\}$. (Demonstre como construir uma expressão regular para L' a partir de uma para L.)

1.8.5. Quais das seguintes afirmações são verdadeiras? Explique.
(a) $baa \in a^*b^*a^*b^*$
(b) $b^*a^* \cap a^*b^* = a^* \cup b^*$
(c) $a^*b^* \cap b^*c^* = \emptyset$
(d) $abcd \in (a(cd)^*b)^*$

1.8.6. A **altura da estrela** $h(a)$ de uma expressão regular α é definida por indução como segue.

$$h(\varnothing) = 0$$
$$h(a) = 0 \text{ para cada } a \in \Sigma$$
$$h(\alpha\cup\beta) = h(\alpha\beta) = \text{o máximo de } h(\alpha) \text{ e } h(\beta).$$
$$h(\alpha^*) = h(\alpha) + 1$$

Por exemplo, se $\alpha = (((ab)^*\cup b^*)^*\cup a^*)$, então $h(\alpha) = 2$. Descubra, em cada caso, uma expressão regular, a qual represente a mesma linguagem e tenha a mínima altura da estrela possível.
(a) $((abc)^*ab)^*$
(b) $(a(ab^*c)^*)^*$
(c) $(c(a^*b)^*)^*$
(d) $(a^*\cup b^*\cup ab)^*$
(e) $(abb^*a)^*$

1.8.7. Uma expressão regular está na **forma normal disjuntiva** se ela está na forma $(\alpha_1\cup\alpha_2\cup \ldots \cup\alpha_n)$ para algum $n \geq 1$, onde nenhum dos α_i's contém uma ocorrência de $\cup$. Demonstre que toda linguagem regular é representada por uma outra na forma normal disjuntiva.

REFERÊNCIAS

Uma excelente fonte sobre teoria de conjuntos informais é o livro:
- P. Halmos *Naive Set Theory*. Princeton, N.J.: D. Van Nostrand, 1960.

Um esplêndido livro sobre indução matemática é:
- G. Polya *Induction and Analogy in Mathematics*, Princeton, N.J.: Princeton University Press, 1954.

Um número de exemplos de aplicações do princípio do ninho de pombo aparece no primeiro capítulo de:
- C. L. Liu Topics *in Combinatorial Mathematics*, Buffalo, N.Y.: Mathematical Association of America, 1972.

O argumento da diagonalização original de Cantor pode ser encontrado em:
- G. Cantor. *Contributions to the Foundations of the Theory of Transfinite Numbers*. New York: Dover Publications, 1947.

A notação o e diversas variantes são introduzidas em:
- D. E. Knuth "Big omicron and big omega and big theta", *ACM SIGACT News*, 8 (2), pp. 18-23, 1976.

O algoritmo $o(n^3)$ para o fechamento transitivo reflexivo é de:
- S. Warshall "A theorem on Boolean matrices," *Journal of the ACM*, 9, 1, pp. 1112, 1962.

Dois livros sobre algoritmos e sua análise são:
- T. H. Cormen, C. E. Leiserson, R. L. Rivest *Introduction to Algorithms*, Cambridge, Mass.: The MIT Press., 1990, e
- G. Brassard, P. Bratley *Fundamentals* of *Algorithms*, Englewood Cliffs, N.J.: Prentice Hall, 1996.

Dois livros avançados sobre teoria da linguagem são:
- A. Salomaa *Formal Languages* New York: Academic Press, 1973.
- M. A. Harrison *Introduction to Formal Language Theory*, Reading, Massach.: Addison-Wesley, 1978.