

Capítulo 5

GRAFOS E ÁRVORES

OBJETIVOS DO CAPÍTULO

Após o estudo deste capítulo, você será capaz de:

- ❖ Compreender e utilizar os diversos termos associados a grafos, grafos direcionados e árvores.
- ❖ Avaliar a utilização de grafos, grafos direcionados e árvores como ferramentas de representação em uma ampla variedade de contextos.
- ❖ Provar que dois grafos são isomorfos ou dar uma razão de por que não o são.
- ❖ Usar a fórmula de Euler para grafos planares simples e conexos.
- ❖ Compreender o papel de dois grafos específicos, K_5 e $K_{3,3}$ na teoria de grafos planares.
- ❖ Provar propriedades elementares de grafos e árvores.
- ❖ Usar a matriz de adjacência e a lista de adjacência para representar grafos e grafos direcionados.
- ❖ Efetuar percursos em uma árvore em pré-ordem, em ordem simétrica e em pós-ordem.
- ❖ Usar alocação seqüencial ou encadeada e ponteiros para armazenar árvores binárias.
- ❖ Usar árvores de decisão para representar os comandos executados por um algoritmo de busca ou ordenação.
- ❖ Construir uma árvore binária de busca e efetuar uma busca em uma árvore binária.
- ❖ Expressar o número de comparações, no pior caso, para busca ou ordenação em uma lista com n elementos.
- ❖ Encontrar os códigos de Huffman para caracteres cuja freqüência de ocorrência é conhecida.

Você trabalha no Departamento de Sistemas de Informação da World Wide Widget (WWW), o líder mundial na produção de regenhocas¹. Regenhocas são aparelhos extremamente complexos, com um número muito grande de componentes muito simples. Cada peça é de um dos seguintes tipos: Arruela (A), Biela (B), Cavidha (C), Engrenagem (E) ou Parafuso (P). Existem muitas variações diferentes de cada tipo básico. Os números das peças começam com uma letra, A, B, C, E ou P, que identifica o tipo, seguida de um número com 8 dígitos. Assim,

B00347289

A11872432

P45003781

são todos números legítimos de componentes. Usando o princípio de multiplicação, existem 5×10^8 números de peças em potencial! WWW mantém um arquivo de dados com os números das peças que usa, que são a maioria dos números em potencial. A maior parte dos computadores, incluindo os usados na WWW, usa o sistema de códigos ASCII para converter caracteres em forma binária, sob o qual cada caractere necessita de 1 byte (8 bits) de armazenagem. Como cada número diferente de peça contém 9 caracteres, o arquivo de peças da WWW é de aproximadamente $9 \times 5 \times 10^8$ bytes, ou 4,5 Gb.

Pergunta: Como comprimir esse arquivo de número de peças de modo a usar menos espaço de armazenamento?

Uma resposta a essa pergunta envolve a utilização de uma estrutura de árvores binárias. Uma árvore é uma representação visual de dados e conexões entre eles. É um caso especial de uma estrutura mais geral chamada de grafo. Grafos ou árvores podem representar um número surpreendente de situações reais — organogramas, mapas rodoviários, redes de transporte e comunicação, e assim por diante. Mais tarde veremos outros usos de grafos e árvores para representar redes lógicas, máquinas de estado finito e derivações de linguagens formais.

¹A palavra utilizada no original inglês, *widget*, é uma palavra inventada, com um som semelhante a *gadget*, que significa “engenhoca”; traduzi por outra palavra inventada, com um som semelhante a engenhoca. (N.T.)

A teoria dos grafos é um tópico extenso. As Seções 5.1 e 5.2 apresentam parte da terminologia associada a grafos e árvores e alguns resultados elementares sobre essas estruturas. Para representar um grafo ou árvore na memória do computador, os dados precisam ser arrumados de tal forma que preserve toda a informação contida na representação visual. São discutidas diversas abordagens para a representação de grafos e árvores dentro de um computador.

Árvores de decisão são representações gráficas das atividades de certos tipos de algoritmos. Elas são apresentadas e usadas, na Seção 5.3, para encontrar limites inferiores para o comportamento, no pior caso, de algoritmos de busca e ordenação. É dado um algoritmo, na Seção 5.4, para a construção de árvores binárias que permitem a compressão dos dados em arquivos grandes.

SEÇÃO 5.1 GRAFOS E SUAS REPRESENTAÇÕES

DEFINIÇÕES DE UM GRAFO

Uma maneira de passar o tempo em uma viagem de avião é olhar os panfletos nos bolsos de assento. Esse material quase sempre inclui um mapa das rotas da companhia proprietária do avião, como na Fig. 5.1. Toda essa informação sobre rotas poderia ser expressa em um parágrafo; por exemplo, existe uma rota direta entre Chicago e Nashville mas não existe uma rota direta entre Nashville e São Luís. No entanto, esse parágrafo seria bastante longo e complicado, e não seríamos capazes de assimilar a informação tão rápida e claramente quanto a partir do mapa. Existem muitos casos onde “uma figura vale mais que mil palavras”.

A palavra gráfico é, muitas vezes, usada para qualquer representação visual de dados, como na Fig. 5.1; outras formas incluem o gráfico de barras, o gráfico pictórico e o gráfico em setores, ilustrados na Fig. 5.2. Falamos, também, sobre gráfico de funções em um sistema retangular de coordenadas. Os gráficos de que trataremos agora são chamados de grafos. Usaremos duas definições de grafos: uma é baseada em uma representação visual como a da Fig. 5.1 e a outra é uma definição mais formal que não fala nada sobre uma representação visual.

Definição (Informal): Grafo Um grafo é um conjunto não-vazio de nós (vértices) e um conjunto de arcos (arestas) tais que cada arco conecta dois nós.

Nossos grafos sempre terão um número finito de nós e de arcos.

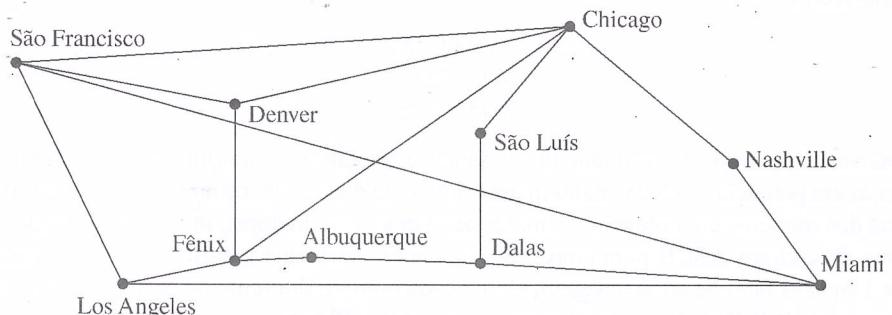


Fig. 5.1

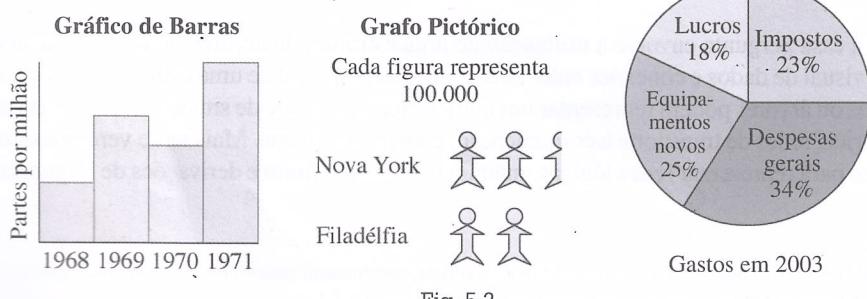


Fig. 5.2

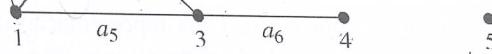


Fig. 5.3

A definição informal de um grafo funciona muito bem se tivermos a representação visual do grafo na nossa frente mostrando que arcos conectam que nós. Sem uma figura, no entanto, precisamos de uma forma concisa de mostrar essa informação. Isso nos leva à segunda definição de grafos.

Definição (Formal): Grafo Um **grafo** é uma tripla ordenada (N, A, g) , onde

N = um conjunto não-vazio de **nós** (vértices)

A = um conjunto de **arcos** (arestas)

g = uma função que associa a cada arco a um par *não-ordenado* $x-y$ de nós, chamados de **extremidades** de a .

♦ EXEMPLO 3

Para o grafo da Fig. 5.3, a função g que associa arcos a suas extremidades é a seguinte: $g(a_1) = 1-2$, $g(a_2) = 1-3$, $g(a_3) = 2-3$, $g(a_4) = 2-3$, $g(a_5) = 1-3$ e $g(a_6) = 3-4$.

PROBLEMA PRÁTICO 1

Esboce um grafo com nós $\{1, 2, 3, 4, 5\}$, arcos $\{a_1, a_2, a_3, a_4, a_5, a_6\}$ e função g dada por $g(a_1) = 1-2$, $g(a_2) = 1-3$, $g(a_3) = 3-4$, $g(a_4) = 3-4$, $g(a_5) = 4-5$ e $g(a_6) = 5-5$.

Podemos querer que os arcos de um grafo começem em um nó e terminem em outro, caso em que teríamos um **grafo direcionado**.

Definição: Grafo Direcionado Um **grafo direcionado** (dígrafo) é uma tripla ordenada (N, A, g) , onde

N = um conjunto não-vazio de nós

A = um conjunto de arcos

g = uma função que associa a cada arco um par *ordenado* (x, y) de nós, onde x é o **ponto inicial (extremidade inicial)** e y é o **ponto final (extremidade final)** de a .

Em um grafo direcionado, cada arco tem um sentido ou orientação.

♦ EXEMPLO 4

A Fig. 5.4 mostra um grafo direcionado, com 4 nós e 5 arcos. A função g que associa a cada arco suas extremidades satisfaz $g(a_1) = (1, 2)$, o que significa que o arco a_1 começa no nó 1 e termina no nó 2. Temos, também, $g(a_3) = (1, 3)$ e $g(a_4) = (3, 1)$.

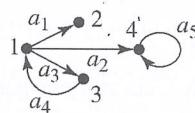


Fig. 5.4

Além de impor orientação aos arcos de um grafo, podemos querer modificar a definição básica de um grafo de outras maneiras. Queremos, muitas vezes, que os nós de um grafo contenham informações identificadoras, ou rótulos, como os nomes das cidades no mapa de rotas aéreas. Esse seria um **grafo rotulado**. Podemos querer usar um **grafo com pesos**, onde cada arco tem um valor numérico, ou peso, associado. Por exemplo, poderíamos querer indicar as distâncias nas várias rotas em nosso mapa da companhia aérea.

❖ EXEMPLO 1

O conjunto de nós no mapa das rotas aéreas na Fig. 5.1 é {Chicago, Nashville, Miami, Dallas, São Luís, Albuquerque, Fênix, Denver, São Francisco, Los Angeles}. O grafo tem 16 arcos; Fênix-Albuquerque é um arco (denominamos, aqui, os arcos pelos nós que ele conecta), Albuquerque-Dallas é outro, e assim por diante.

❖ EXEMPLO 2

O grafo da Fig. 5.3 tem cinco nós e seis arcos. O arco a_1 conecta os nós 1 e 2, a_3 conecta os nós 2 e 3, e assim por diante.

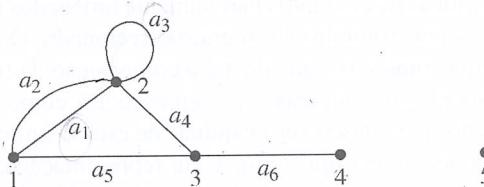


Fig. 5.3

A definição informal de um grafo funciona muito bem se tivermos a representação visual do grafo na nossa frente mostrando que arcos conectam que nós. Sem uma figura, no entanto, precisamos de uma forma concisa de mostrar essa informação. Isso nos leva à segunda definição de grafos.

Definição (Formal): Grafo Um grafo é uma tripla ordenada (N, A, g) , onde

N = um conjunto não-vazio de nós (vértices)

A = um conjunto de arcos (arestas)

g = uma função que associa a cada arco a um par não-ordenado $x-y$ de nós, chamados de extremidades de a .

❖ EXEMPLO 3

Para o grafo da Fig. 5.3, a função g que associa arcos a suas extremidades é a seguinte: $g(a_1) = 1-2$, $g(a_2) = 1-5$, $g(a_3) = 2-3$, $g(a_4) = 2-3$, $g(a_5) = 3-5$ e $g(a_6) = 3-4$.

PROBLEMA PRÁTICO 1

Esboce um grafo com nós $\{1, 2, 3, 4, 5\}$, arcos $\{a_1, a_2, a_3, a_4, a_5, a_6\}$ e função g dada por $g(a_1) = 1-2$, $g(a_2) = 1-3$, $g(a_3) = 3-4$, $g(a_4) = 3-4$, $g(a_5) = 4-5$ e $g(a_6) = 5-5$.

Podemos querer que os arcos de um grafo comecem em um nó e terminem em outro, caso em que teríamos um *grafo direcionado*.

Definição: Grafo Direcionado Um grafo direcionado (dígrafo) é uma tripla ordenada (N, A, g) , onde

N = um conjunto não-vazio de nós

A = um conjunto de arcos

g = uma função que associa a cada arco um par ordenado (x, y) de nós, onde x é o ponto inicial (extremidade inicial) e y é o ponto final (extremidade final) de a .

Em um grafo direcionado, cada arco tem um sentido ou orientação.

❖ EXEMPLO 4

A Fig. 5.4 mostra um grafo direcionado, com 4 nós e 5 arcos. A função g que associa a cada arco suas extremidades satisfaz $g(a_1) = (1, 2)$, o que significa que o arco a_1 começa no nó 1 e termina no nó 2. Temos, também, $g(a_3) = (1, 3)$ e $g(a_4) = (3, 1)$.

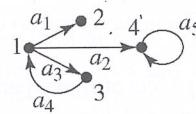


Fig. 5.4

Além de impor orientação aos arcos de um grafo, podemos querer modificar a definição básica de um grafo de outras maneiras. Queremos, muitas vezes, que os nós de um grafo contenham informações identificadoras, ou rótulos, como os nomes das cidades no mapa de rotas aéreas. Esse seria um *grafo rotulado*. Podemos querer usar um *grafo com pesos*, onde cada arco tem um valor numérico, ou peso, associado. Por exemplo, poderíamos querer indicar as distâncias nas várias rotas em nosso mapa da companhia aérea.

Neste livro a palavra “grafo” sempre indicará um grafo não-direcionado. Para nos referir a um grafo direcionado, sempre escreveremos “grafo direcionado”.

APLICAÇÕES DE GRAFOS

Embora a idéia de grafo seja bastante simples, um número surpreendente de situações envolvem relações entre itens que podem ser representadas por um grafo. Não é de espantar que este livro contenha muitos grafos. Vimos representações gráficas de conjuntos parcialmente ordenados (diagramas de Hasse) no Cap. 4. Um diagrama PERT (Fig. 4.7, por exemplo) é um grafo direcionado. O diagrama E-R (Fig. 4.10, por exemplo) é um grafo. O diagrama comutativo que ilustra a composição de funções (Fig. 4.23) é um grafo direcionado. O Cap. 7 introduzirá redes lógicas e as representará como grafos direcionados. Grafos direcionados também serão usados para descrever máquinas de estado finito no Cap. 8.

Vimos que o mapa de rotas era um grafo. Uma representação de qualquer rede de rotas de transporte (um mapa de estradas, por exemplo), rede de comunicação (como em uma rede de computadores), ou rotas de distribuição de produtos ou serviços, como dutos de gás ou água, é um grafo. A estrutura química de uma molécula também pode ser representada por um grafo.

Desenhe o grafo subjacente em cada um dos casos a seguir:

- A Fig. 5.5 é um mapa de estradas no Arizona.
- A Fig. 5.6 é uma representação de uma molécula de ozônio com três átomos de oxigênio.

PROBLEMA PRÁTICO 2

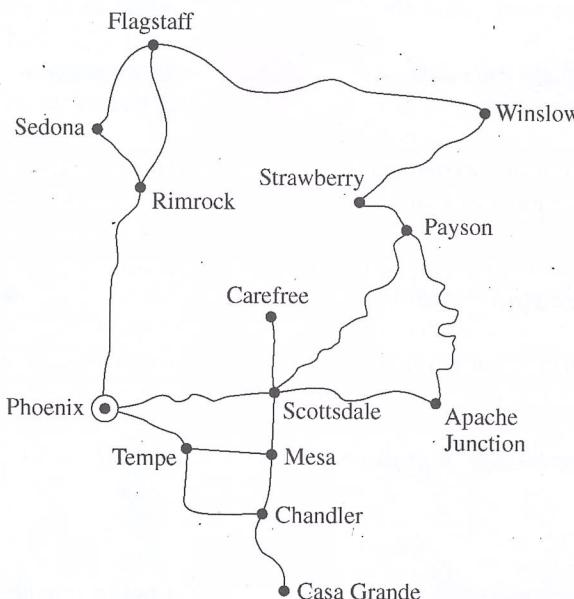


Fig. 5.5

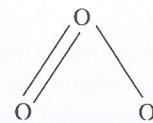


Fig. 5.6

Uma visão esquemática do fluxo de informação no DETRAN seria o primeiro passo para se desenvolver um novo sistema computadorizado para novas licenças. A Fig. 5.7 mostra o grafo direcionado resultante, muitas vezes chamado de **diagrama de fluxo**.

EXEMPLO 5

A Fig. 5.8 mostra um grafo que representa uma rede local de computadores em uma firma. Nessa “topologia estrela”, todas as máquinas se comunicam através de um servidor central. O grafo ilustra bem uma das fraquezas de um tal projeto de rede: sua dependência na operação confiável e constante do servidor central.

EXEMPLO 6

Redes neurais, ferramentas utilizadas em inteligência artificial para tarefas como o reconhecimento de padrões, são representadas por grafos direcionados com peso. A Fig. 5.9 mostra uma rede com camadas múltiplas consistindo em unidades de entrada, unidades de saída e unidades de “camadas escondidas”. Os pesos nos arcos dos grafos são ajustados à medida que a rede neural “aprende” a reconhecer certos padrões em teste.

EXEMPLO 7

TERMINOLOGIA DA TEORIA DOS GRAFOS

Antes de prosseguir, precisamos de alguma terminologia sobre grafos. Surpreendentemente, embora exista uma grande quantidade de livros sobre a teoria dos grafos, a terminologia não é completamente padronizada.

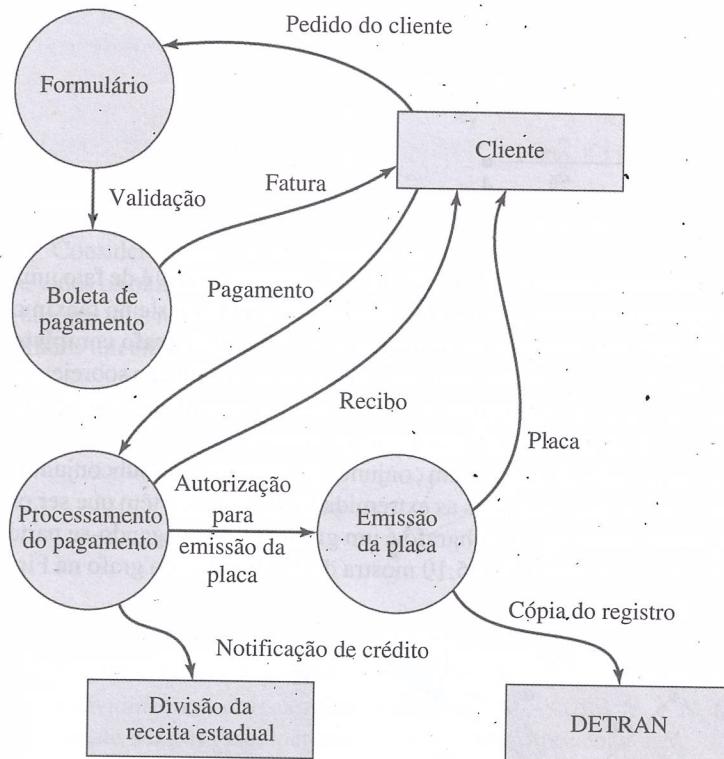


Fig. 5.7

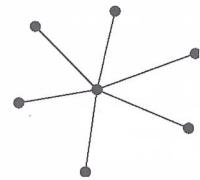


Fig. 5.8

Unidades de entrada Camadas ocultas Unidades de saída

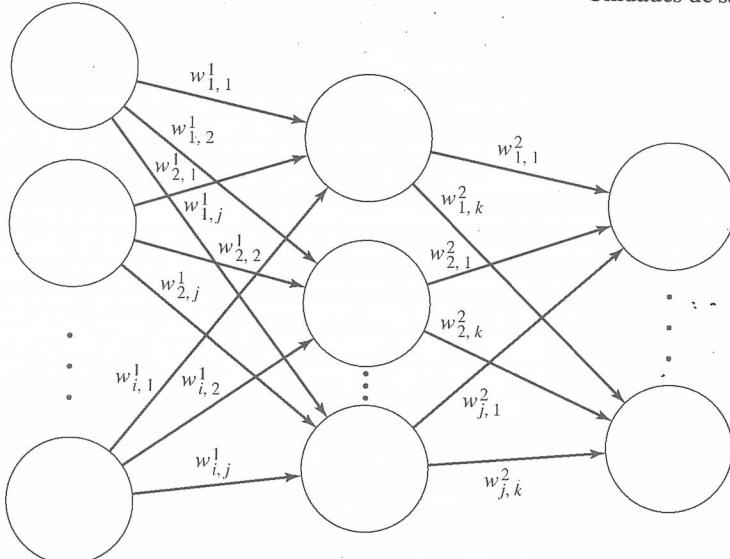


Fig. 5.9

da. Outros livros, portanto, podem ter nomes ligeiramente diferentes de alguns desses termos.

Dois nós em um grafo são ditos **adjacentes** se ambos são as extremidades de algum arco. Por exemplo, no grafo da Fig. 5.3, reproduzido novamente a seguir, 1 e 3 são nós adjacentes mas 1 e 4 não. O nó 2 é adjacente a si mesmo. Um **laço** em um grafo é um arco com extremidades $n-n$ para algum nó n ; na Fig. 5.3, o arco a_3 é um laço com extremidades 2–2. Usaremos a terminologia **grafo sem arcos** no caso em que o grafo não tiver nenhum laço. Dois arcos com as mesmas extremidades são ditos **arcos paralelos**; os arcos a_1 e a_2 na Fig. 5.3 são paralelos. Um **grafo simples** é um grafo sem laços nem arcos paralelos. Um **nó isolado** é um nó que não é adjacente a nenhum outro; na Fig. 5.3, o nó 5 é um nó isolado. O **grau** de um nó é o número de extremidades de arcos naquele nó. Na Fig. 5.3, os nós 1 e 3 têm grau 3, o nó 2 tem grau 5, o nó 4 tem grau 1 e o nó 5 tem grau 0.

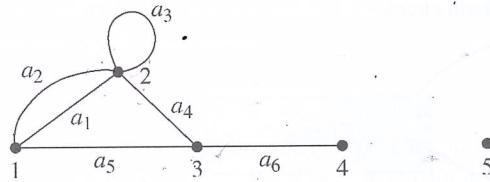


Fig. 5.3

Como a função g , que associa a cada arco suas extremidades na definição formal de grafo, é de fato uma função, cada arco tem um único par de extremidades. Se g é uma função injetora, então existe no máximo um arco associado a cada par de extremidades; um tal grafo não tem arcos paralelos. Um **grafo completo** é um grafo no qual dois nós distintos quaisquer são adjacentes. Nesse caso g é quase uma função sobrejetora — todo par $x-y$ de nós distintos é a imagem, sob g , de algum arco — mas não há a necessidade de se ter um laço em cada nó. Portanto, pares da forma $x-x$ podem não ter uma imagem inversa.

Um **subgrafo** de um grafo consiste em um conjunto de nós e um conjunto de arcos que são subconjuntos do conjunto original de nós e arcos, respectivamente, nos quais as extremidades de um arco têm que ser os mesmos nós que no grafo original. Em outras palavras, um subgrafo é um grafo obtido apagando-se parte do grafo original e deixando o resto sem modificações. A Fig. 5.10 mostra dois subgrafos do grafo na Fig. 5.3. Note que o grafo na Fig. 5.10a é simples e completo.

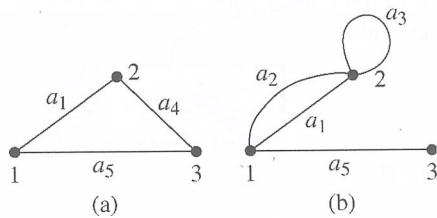


Fig. 5.10

Um **caminho** do nó n_0 para o nó n_k é uma seqüência

$$n_0, a_0, n_1, a_1, \dots, n_{k-1}, a_{k-1}, n_k$$

de nós e arcos onde, para cada i , as extremidades do arco a_i são n_i-n_{i+1} . No grafo da Fig. 5.3, um caminho do nó 2 para o nó 4 consiste na seqüência 2, a_1 , 1, a_2 , 2, a_4 , 3, a_6 , 4. O **comprimento** de um caminho é o número de arcos que ele contém; se um arco for usado mais de uma vez, ele é contado cada vez que é usado. O comprimento do caminho descrito neste parágrafo do nó 2 para o nó 4 é 4.

Um grafo é **conexo** se existe um caminho de qualquer nó para qualquer outro. Ambos os grafos na Fig. 5.10 são conexos mas o grafo na Fig. 5.3 não é. Um **ciclo** em um grafo é um caminho de algum nó n_0 para ele mesmo tal que nenhum arco aparece mais de uma vez, n_0 é o único nó que aparece mais de uma vez e n_0 aparece apenas nas extremidades. (Nós e arcos podem ser repetidos em um caminho mas não, com exceção do nó n_0 , em um ciclo.) No grafo da Fig. 5.3,

$$1, a_1, 2, a_4, 3, a_5, 1$$

é um ciclo. Um grafo sem ciclos é dito **acíclico**.

Considere o grafo criado no Problema Prático 1.

PROBLEMA PRÁTICO 3

- Encontre dois nós que não são adjacentes.
- Encontre um nó adjacente a si mesmo.
- Encontre um laço.
- Encontre dois arcos paralelos.
- Encontre o grau do nó 3.
- Encontre um caminho de comprimento 5.
- Encontre um ciclo.
- Esse grafo é completo?
- Esse grafo é conexo?

A Fig. 5.11 ilustra os grafos simples completos com 1, 2, 3 e 4 vértices. O grafo simples completo com n vértices é denotado por K_n .

EXEMPLO 8

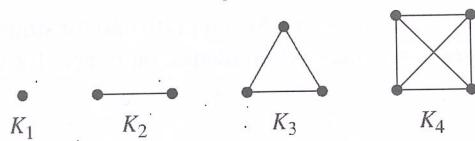


Fig. 5.11

PROBLEMA PRÁTICO 4

Desenhe K_5 .

Considere agora o grafo simples da Fig. 5.12. Esse grafo não é completo, já que nem todo nó é adjacente a todos os outros nós. No entanto, os nós podem ser divididos em dois conjuntos disjuntos, $\{1, 2\}$ e $\{3, 4, 5\}$, tais que dois nós quaisquer escolhidos no mesmo conjunto não são adjacentes, mas dois nós quaisquer escolhidos um em cada conjunto são adjacentes. Um grafo desse tipo é chamado de *grafo bipartido completo*.

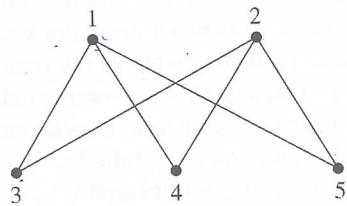


Fig. 5.12

Definição: Grafo Bipartido Completo Um grafo é um grafo bipartido completo se seus nós podem ser divididos em dois conjuntos disjuntos não-vazios N_1 e N_2 tais que dois nós são adjacentes se, e somente se, um deles pertence a N_1 e o outro pertence a N_2 . Se $|N_1| = m$ e $|N_2| = n$, um tal grafo é denotado por $K_{m,n}$.

A Fig. 5.12, portanto, ilustra $K_{2,3}$.

PROBLEMA PRÁTICO 5

Desenhe $K_{3,3}$.

O conceito de caminho pode ser estendido a um grafo direcionado de maneira natural: um **caminho** de um nó n_0 para um nó n_k em um grafo direcionado é uma seqüência

$$n_0, a_0, n_1, a_1, \dots, n_{k-1}, a_{k-1}, n_k$$

onde, para cada i , n_i é o ponto inicial e n_{i+1} o ponto final do arco a_i . Se existe um caminho de n_0 para n_k , então n_k é **acessível** de n_0 . A definição de ciclo também pode ser estendida para grafos direcionados.

EXEMPLO 9

No grafo direcionado ilustrado na Fig. 5.13, existem muitos caminhos do nó 1 para o nó 3: duas possibilidades são 1, a_4 , 3 e 1, a_1 , 2, a_2 , 2, a_2 , 2, a_3 , 3. O nó 3 é certamente acessível do nó 1. O nó 1, no entanto, não é acessível de nenhum outro nó. Os ciclos nesse grafo são o laço a_2 e o caminho 3, a_5 , 4, a_6 , 3.

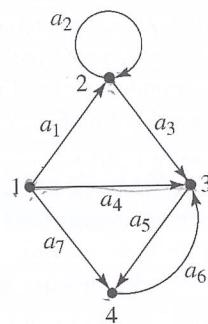


Fig. 5.13

Podemos provar algumas proposições (razoavelmente triviais) sobre grafos que seguem diretamente das definições.

EXEMPLO 10

Prove que um grafo acíclico é simples.

Faremos uma demonstração por contraposição. Se um grafo não for simples, ele tem arcos paralelos ou um laço. Então, os dois arcos paralelos e suas extremidades, ou o laço, formam um ciclo, e o grafo não é acíclico.

Note que a recíproca da proposição no Exemplo 10 não é verdadeira: a Fig. 5.10a é um grafo simples mas contém um ciclo.

- Prove que todo grafo completo é conexo.
- Encontre um grafo conexo que não é completo.

PROBLEMA PRÁTICO 6

GRAFOS ISOMORFOS

Dois grafos podem parecer muito diferentes em sua representação visual mas ainda assim serem o mesmo grafo de acordo com nossa definição formal. Queremos distinguir entre dois grafos que têm diferenças visuais cosméticas e os que têm estruturas fundamentalmente diferentes. Os grafos nas Figs. 5.14 e 5.15 são iguais — eles têm os mesmos nós, os mesmos arcos e a mesma função que associa as extremidades a cada arco. (Na representação de um grafo, arcos podem se intersectar em pontos que não são nós do grafo.) O grafo na Fig. 5.16 também é, essencialmente, o mesmo grafo. Se trocarmos os nomes dos nós e dos arcos na Fig. 5.14 através das funções a seguir, os grafos seriam iguais:

$$\begin{array}{ll} f_1: 1 \rightarrow a & f_2: a_1 \rightarrow e_2 \\ 2 \rightarrow c & a_2 \rightarrow e_1 \\ 3 \rightarrow b & \\ 4 \rightarrow d & \end{array}$$

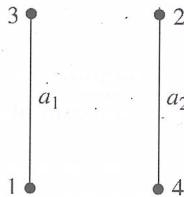


Fig. 5.14

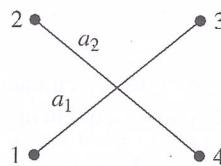


Fig. 5.15

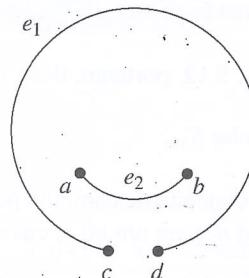


Fig. 5.16

Estruturas que são iguais, exceto por uma mudança de nomes, são ditas *isomorfas*. Para mostrar que duas estruturas são isomorfas, precisamos obter uma mudança de nomes (uma bijeção entre os elementos das duas estruturas) e depois mostrar que as propriedades importantes das estruturas são “preservadas” (mantidas) sob essa mudança de nomes. No caso de grafos, os elementos são os nós e os arcos. A “propriedade importante” em um grafo é quais arcos conectam quais nós.

As funções dadas f_1 e f_2 são bijeções entre os nós e os arcos do grafo na Fig. 5.14 e os nós e os arcos, respectivamente, do grafo na Fig. 5.16. Além disso, se um arco a na Fig. 5.14 tem extremidades $x-y$, então o arco $f_2(a)$ na Fig. 5.16 tem extremidades $f_1(x)-f_1(y)$, e vice-versa. Por exemplo, o arco a_1 na Fig. 5.14 tem extremidades 1-3, enquanto o arco correspondente e_2 na Fig. 5.16 tem extremidades $a-b$, que são os nós na Fig. 5.16 os quais correspondem aos nós 1 e 3 na Fig. 5.14. Podemos formalizar essa idéia.

Definição: Grafos Isomorfos Dois grafos (N_1, A_1, g_1) e (N_2, A_2, g_2) são **isomorfos** se existem bijeções $f_1: N_1 \rightarrow N_2$ e $f_2: A_1 \rightarrow A_2$ tais que, para cada arco $a \in A_1$, $g_2[f_2(a)] = f_1(g_1(a))$.

Os grafos ilustrados na Fig. 5.17 são isomorfos. As bijeções que estabelecem o isomorfismo são dadas parcialmente a seguir:

$$\begin{array}{ll} f_1: 1 \rightarrow c & f_2: a_1 \rightarrow e_1 \\ 2 \rightarrow e & a_2 \rightarrow e_4 \\ 3 \rightarrow d & a_3 \rightarrow e_2 \\ 4 \rightarrow b & \\ 5 \rightarrow a & \vdots \end{array}$$

EXEMPLO 11

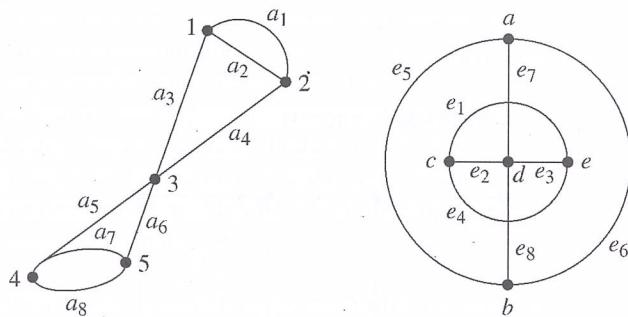


Fig. 5.17

Usando essas bijeções, $g_1(a_3) = 1-3$ e $g_2[f_2(a_3)] = g_2(e_2) = c-d = f_1(1)-f_1(3)$. Isso mostra que a relação entre arco e extremidades é preservada sob a mudança de nomes para o caso do arco a_3 . Para mostrar que os grafos são isomorfos, teríamos que completar a definição da função f_2 e depois demonstrar que a relação entre arcos e extremidades é preservada sob essas funções examinando todos os casos possíveis. ♦

PROBLEMA PRÁTICO 7

Isomorfismos entre grafos é mais fácil de provar se restringirmos nossa atenção a grafos simples. Se pudermos encontrar uma função apropriada f_1 que leva nós em nós, então a função f_2 que leva arcos em arcos é trivial, já que existe no máximo um arco entre qualquer par de extremidades. Portanto, o teorema a seguir é válido.

Teorema sobre Isomorfismos de Grafos Simples Dois grafos simples (N_1, A_1, g_1) e (N_2, A_2, g_2) são isomorfos se existe uma bijeção $f: N_1 \rightarrow N_2$ tal que, quaisquer que sejam os nós n_i e n_j de N_1 , n_i e n_j são adjacentes se, e somente se, $f(n_i)$ e $f(n_j)$ são adjacentes. (A função f é chamada um **isomorfismo** do grafo 1 no grafo 2.)

PROBLEMA PRÁTICO 8

Encontre um isomorfismo do grafo na Fig. 5.18a para o na Fig. 5.18b.

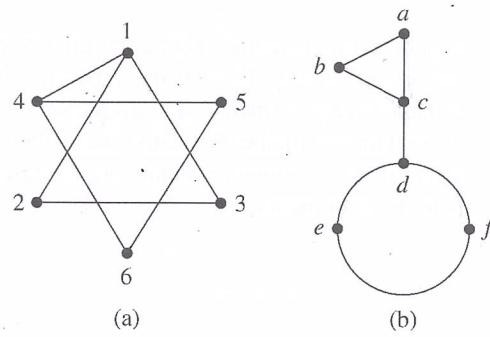


Fig. 5.18

Para provar que dois grafos são isomorfos é necessário encontrar uma bijeção (ou, para grafos que não são simples, duas bijeções) e depois mostrar que a propriedade de adjacência (ou relação entre arcos e extremidades) é preservada. Para provar que dois grafos não são isomorfos, precisamos mostrar que a(s) bijeção(s) necessária(s) não existe(m). Poderíamos tentar todas as bijeções possíveis (como existe apenas um número finito de nós e arcos, existe um número finito de bijeções). No entanto, esse método torna-se inviável em qualquer grafo um pouquinho maior. Ao invés disso, podemos tentar encontrar alguma outra razão para essas bijeções não existirem. Embora isso nem sempre seja fácil, existem certas condições sob as quais é claro que os grafos não são isomorfos (veja o Exercício 15). Estas incluem as seguintes:

1. Um grafo tem mais nós do que o outro.
2. Um grafo tem mais arcos do que o outro.
3. Um grafo tem arcos paralelos e o outro não.
4. Um grafo tem um laço e o outro não.
5. Um grafo tem um nó de grau k e o outro não.

6. Um grafo é conexo e o outro não.
 7. Um grafo tem um ciclo e o outro não.

Prove que os dois grafos na Fig. 5.19 não são isomorfos.

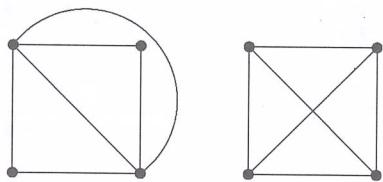


Fig. 5.19

**PROBLEMA
PRÁTICO 9**

Os dois grafos na Fig. 5.20 não são isomorfos. Note que cada grafo tem seis nós e sete arcos. Nenhum deles tem arcos paralelos ou laços. Ambos são conexos. Ambos têm três ciclos, quatro nós de grau 2 e dois nós de grau 3. Portanto, nenhum dos testes óbvios para a não existência de isomorfismo se aplica. No entanto, o grafo na Fig. 5.20b tem um nó de grau 2 adjacente a dois nós de grau 3; isso não acontece na Fig. 5.20a, de modo que os grafos não são isomorfos.

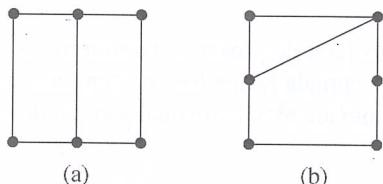


Fig. 5.20

EXEMPLO 12

Mais uma vez, grafos isomorfos são essencialmente “iguais”, independentemente das diferenças cosméticas em como estão desenhados ou em como os vértices e arcos são identificados, enquanto grafos não-isomorfos têm diferenças estruturais.

GRAFOS PLANARES

Um **grafo planar** é um grafo que pode ser representado (em uma folha de papel, isso é, em um plano) de modo que seus arcos se intersectam apenas em nós. Pessoas que projetam circuitos integrados querem que todos os componentes em uma camada do *chip* formem um grafo planar, de modo a não haver cruzamento de conexões. O grafo na Fig. 5.14 é certamente planar. Por outro lado, sabemos que ele é isomorfo ao grafo na Fig. 5.15, de modo que o grafo na Fig. 5.15 também é planar. A palavra-chave na definição de um grafo planar é que ele *pode* ser desenhado de certa maneira.

Prove que K_4 é um grafo planar.

**PROBLEMA
PRÁTICO 10**

Considere K_5 , o grafo simples completo com cinco vértices. Vamos tentar construir K_5 sem arcos que se intersectam no meio, começando com alguns arcos e depois adicionando tantos novos arcos quanto possível sem cortar os arcos já existentes. Colocamos, primeiro, os cinco vértices e, depois, os conectamos como na Fig. 5.21a. (Como todos os vértices em K_5 são simétricos, não importa como colocamos os nomes.)

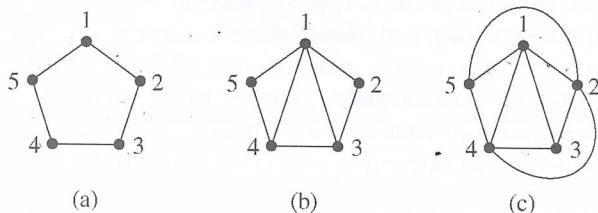


Fig. 5.21

EXEMPLO 13

A seguir, ligamos o nó 1 aos nós 3 e 4, como na Fig. 5.21b. Agora, precisamos conectar o nó 2 aos nós 4 e 5. Isso pode ser feito, preservando a planaridade, colocando-se os arcos por fora, como na Fig. 5.21c. Falta

ainda a ligação final entre os nós 3 e 5. Mas não é possível desenhar um arco do nó 3 para o nó 5 sem cruzar o arco 2–4 ou um ou mais dos arcos interiores, como o 1–4.

Tivemos que fazer uma escolha sobre onde colocar os arcos 1–3 e 1–4; escolhemos desenhá-los no interior. Poderíamos verificar se faria alguma diferença colocar esses arcos na parte de fora, mas acontece que isso não vai mudar nada (veja o Problema Prático 11). Parece, portanto, que o grafo K_5 não é planar. Entretanto, ainda gostaríamos de ver uma demonstração desse fato baseada em alicerces firmes — o que fizemos parece muito com um argumento do tipo “não consegui fazer, então não pode ser feito”. Daremos tal demonstração um pouco mais adiante.

PROBLEMA PRÁTICO 11

Mostre que colocar os arcos 1–3 e 1–4 como arcos exteriores, ao construir K_5 , ainda nos leva a uma situação na qual os arcos têm que se intersectar.

PROBLEMA PRÁTICO 12

O matemático suíço do século XVIII, Leonhard Euler (que se lê “óiler”), descobriu um fato sobre grafos planares. Um grafo planar (quando desenhado em uma representação planar, sem cruzamento de arcos) simples e conexo divide o plano em um determinado número de regiões, incluindo regiões totalmente limitadas por arcos e uma região exterior ilimitada. Euler observou uma relação entre o número n de nós, o número a de arcos e o número r de regiões em um tal grafo. Essa relação é conhecida como a **fórmula de Euler**:

$$\text{nós} - \text{arcos} + \text{regiões} = 2 \quad (1)$$

PROBLEMA PRÁTICO 13

Verifique a fórmula de Euler para o grafo planar simples e conexo na Fig. 5.18b.

Para provar a fórmula de Euler, vamos fazer uma demonstração por indução no número de arcos a . A base da indução é o caso $a = 0$, quando temos apenas um nó; a única região é a região externa (Fig. 5.22a). Aqui, $n = 1$, $a = 0$ e $r = 1$, logo a Eq. (1) é válida. Suponhamos agora que a fórmula é válida para a representação planar de qualquer grafo planar simples e conexo com k arcos, e considere um tal grafo com $k + 1$ arcos. Como de hábito, precisamos relacionar o “caso $k + 1$ ” ao “caso k ” de modo a usar a hipótese de indução. Vamos considerar dois casos para o grafo com $k + 1$ arcos.

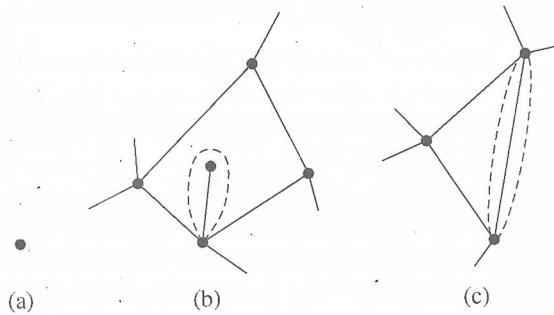


Fig. 5.22

Caso 1. O grafo tem um nó de grau 1. Apague, temporariamente, esse nó e o arco do qual ele é uma das extremidades (Fig. 5.22b); isso nos deixa um grafo planar simples e conexo com k arcos, um determinado número n de nós e algum número r de regiões tal que (pela hipótese de indução)

$$n - k + r = 2$$

No grafo original, temos um arco a mais, um nó a mais e o mesmo número de regiões, logo a fórmula apropriada é

$$(n + 1) - (k + 1) + r = 2$$

que, pela hipótese de indução, é válida.

Caso 2. O grafo não tem nós de grau 1. Então apague, temporariamente, um arco que ajuda a definir uma região limitada (Fig. 5.22c). (Se não existem arcos que ajudam a definir uma região limitada, então o grafo é uma cadeia e existe um nó de grau 1.) Isso nos deixa um grafo planar simples e conexo com k arcos, algum número n de nós e um número r de regiões tal que (pela hipótese de indução)

$$n - k + r = 2$$

No grafo original, tínhamos um arco a mais e uma região a mais, mas o mesmo número de nós, logo a fórmula adequada é

$$n - (k + 1) + (r + 1) = 2$$

que é válida, pela hipótese de indução.

Na demonstração da fórmula de Euler, explique por que, no caso 2, o arco a ser apagado tem que ajudar a definir uma região limitada. Dê duas razões.

PROBLEMA
PRÁTICO 14

A fórmula de Euler tem duas consequências se colocarmos outras restrições sobre o grafo. Suponha que o grafo não só é planar simples e conexo, mas também tem pelo menos três nós. Em uma representação planar de um tal grafo, podemos contar o número de arcos que são adjacentes a cada região (formam a fronteira de cada região), incluindo a região externa. Arcos inteiramente no interior de uma região contribuem duas arestas para aquela região; por exemplo, ao percorrer a fronteira da região interior ilustrada na Fig. 5.22b, percorremos seis arestas, incluindo o arco que sai do nó de grau 1 e depois o mesmo arco de volta. Arcos que separam duas regiões contribuem uma aresta para cada região. Portanto, se o grafo tem a arcos, o número de arestas das regiões é $2a$.

Não existem regiões com exatamente uma aresta adjacente, já que o grafo não tem laços. Não existem regiões com exatamente duas arestas adjacentes, já que não há arestas paralelas e o grafo consistindo inteiramente em um arco unindo dois nós (que teria duas arestas adjacentes à região exterior) está excluído. Portanto, cada região tem pelo menos três arestas adjacentes, logo $3r$ é o número mínimo de arestas de regiões. Então,

$$2a \geq 3r$$

ou, da Eq. (1),

$$2a \geq 3(2 - n + a) = 6 - 3n + 3a$$

e, finalmente,

$$a \leq 3n - 6 \quad (2)$$

Se colocarmos uma última restrição sobre o grafo, a de que não existem ciclos de comprimento 3, então cada região terá, pelo menos, quatro arestas adjacentes, de modo que $4r$ será o número mínimo de arestas das regiões. Isso nos leva à desigualdade

$$2a \geq 4r$$

que fica

$$a \leq 2n - 4 \quad (3)$$

Esses resultados estão resumidos no teorema a seguir:

Teorema sobre o Número de Nós e Arcos Para um grafo planar simples e conexo com n nós e a arcos:

1. Se a representação planar divide o plano em r regiões, então

$$n - a + r = 2 \quad (1)$$

2. Se $n \geq 3$, então

$$a \leq 3n - 6 \quad (2)$$

3. Se $n \geq 3$ e se não existem ciclos de comprimento 3, então

$$a \leq 2n - 4 \quad (3)$$

Note que a desigualdade (3) coloca uma limitação mais estrita sobre o número de arcos do que a desigualdade (2), mas foi colocada uma condição adicional sobre o grafo.

Podemos usar esse teorema para provar que certos grafos não são planares.

❖ EXEMPLO 14

K_5 é um grafo simples e conexo com 5 nós (e 10 arcos). Se fosse um grafo planar, a desigualdade (2) do nosso teorema seria válida, mas $10 > 3(5) - 6$. Logo, como o nosso argumento construtivo indicou, K_5 não é planar. $K_{3,3}$ é um grafo simples e conexo com 6 nós (e 9 arcos). Ele não tem ciclos de comprimento 3, já que isso necessitaria que dois nós em um dos subconjuntos fossem adjacentes. Se fosse um grafo planar, a desigualdade (3) seria válida, mas $9 > 2(6) - 4$. Portanto, $K_{3,3}$ não é planar.

PROBLEMA PRÁTICO 15

Mostre que a desigualdade (2) é válida para $K_{3,3}$, o que mostra que essa desigualdade é uma condição necessária, mas não suficiente, para um grafo com $n \geq 3$ ser planar.

Os grafos não-planares K_5 e $K_{3,3}$ têm um papel central na teoria dos grafos não-planares. Para enunciar qual é esse papel, precisamos de mais uma definição.

Definição: Grafos Homeomorfos Dois grafos são ditos **homeomorfos** se ambos podem ser obtidos do mesmo grafo por uma seqüência de subdivisões elementares, nas quais um único arco $x-y$ é substituído por dois novos arcos, $x-v$ e $v-y$, ligando um novo nó v .

❖ EXEMPLO 15

Os grafos nas partes (b) e (c) da Fig. 5.23 são homeomorfos, pois cada um deles pode ser obtido do grafo na Fig. 5.23a por uma seqüência de subdivisões elementares. (No entanto, nenhum deles pode ser obtido do outro por uma seqüência de subdivisões elementares.)

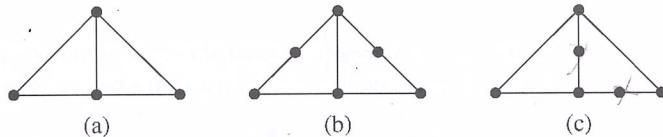


Fig. 5.23

Um grafo planar não pode ser transformado em um grafo não-planar por subdivisões elementares e um grafo não-planar não pode ser transformado em um planar por subdivisões elementares (veja o Exercício 26). Como consequência, grafos homeomorfos ou são ambos planares, ou são ambos não-planares. O teorema a seguir, do matemático polonês Kazimierz Kuratowski, caracteriza os grafos não-planares.

Teorema de Kuratowski Um grafo é não-planar se, e somente se, ele contém um subgrafo que é homeomorfo a K_5 ou a $K_{3,3}$.

Não demonstraremos esse teorema, embora uma direção seja fácil de ver. Se um grafo tem um subgrafo homeomorfo ao grafo não-planar K_5 ou a $K_{3,3}$, então o subgrafo — e, portanto, todo o grafo — não é planar.

❖ EXEMPLO 16

A Fig. 5.24a mostra o “grafo de Petersen”. Provaremos que esse grafo não é planar encontrando um subgrafo homeomorfo a $K_{3,3}$. Olhando o topo do grafo, podemos ver que o nó a é adjacente aos nós e, f e b , e nenhum desses é adjacente a um dos outros. Além disso, o nó e é adjacente aos nós d e j , além do nó a , e nenhum dos nós a, d e j é adjacente a um dos outros. Essa informação está incorporada no grafo da Fig. 5.24b, que também é um subgrafo de $K_{3,3}$. Os arcos necessários para completar $K_{3,3}$ estão ilustrados como linhas tracejadas na Fig. 5.24c. Esses arcos não estão no grafo de Petersen; por exemplo, não há arco $j-f$. No entanto, existe um caminho no grafo de Petersen de j para f usando o nó intermediário h , isso é, $j-h$ e $h-f$. Analogamente, existem caminhos $j-g$ e $g-b$, $d-i$ e $i-f$, e $d-c$ e $c-b$. Adicionando esses caminhos à Fig. 5.24b, obtemos a Fig. 5.24d, que é um subgrafo do grafo de Petersen e pode também ser obtido do grafo na Fig. 5.24c por uma seqüência de subdivisões elementares.

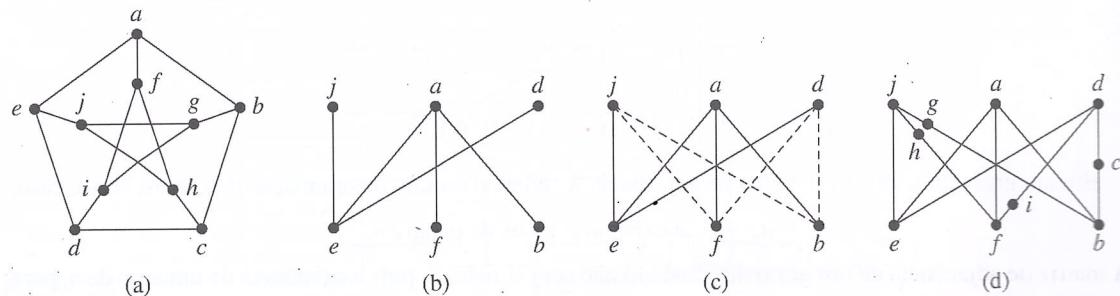


Fig. 5.24

REPRESENTAÇÃO DE GRAFOS NO COMPUTADOR

Dissemos que a maior vantagem de um grafo é sua representação visual da informação. E se quisermos armazenar um grafo em forma digital? Embora seja possível armazenar uma imagem digital de um grafo, isso necessita de muito espaço. Além disso, uma tal imagem permanece como imagem — ela não pode ser manipulada. O que precisamos armazenar são os dados essenciais que fazem parte da definição de grafo — quais são os nós e quais deles são extremidades de arcos. A partir dessa informação, pode-se construir uma representação visual se quisermos. As representações computacionais usuais envolvem uma entre duas estruturas de dados, uma matriz de adjacência ou uma lista de adjacências.

Matriz de Adjacência

Suponha que um grafo tem n nós, numerados n_1, n_2, \dots, n_n . Essa numeração impõe uma ordem arbitrária nos nós; lembre-se de que um conjunto é uma coleção não-ordenada. No entanto, isso é feito simplesmente para identificar os nós — não é dado significado algum ao fato de um nó aparecer antes de outro. Após a ordenação dos nós, podemos formar uma matriz $n \times n$ onde o elemento i, j é o número de arcos entre os nós n_i e n_j . Essa matriz é chamada de **matriz de adjacência A** do grafo em relação a essa ordem. Assim,

$$a_{ij} = p, \text{ se existem } p \text{ arcos entre } n_i \text{ e } n_j.$$

A matriz de adjacência do grafo na Fig. 5.25 em relação à ordenação 1, 2, 3, 4 é uma matriz 4×4 .

EXEMPLO 17

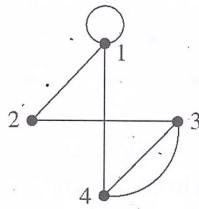


Fig. 5.25

O elemento $1,1$ é 1, devido ao laço no nó 1. Todos os outros elementos na diagonal principal são 0. O elemento $2,1$ (segunda linha, primeira coluna) é 1 porque existe um arco entre os nós 1 e 2, o que também implica que o elemento $1,2$ é 1. Temos, até agora,

$$A = \begin{bmatrix} 1 & 1 & - & - \\ 1 & 0 & - & - \\ - & - & 0 & - \\ - & - & - & 0 \end{bmatrix}$$

Complete a matriz de adjacência da Fig. 5.25.

PROBLEMA PRÁTICO 16

A matriz de adjacência no Problema Prático 16 é simétrica, o que vai ser verdade para qualquer matriz de adjacência de um grafo não-direcionado — se existem p arcos entre os nós n_i e n_j , certamente existem p arcos entre n_j e n_i . A simetria da matriz significa que precisamos apenas armazenar os elementos pertencentes à diagonal principal ou abaixo dela. Portanto, toda a informação contida no grafo da Fig. 5.25 está contida no arranjo “triangular inferior” ilustrado e o grafo pode ser reconstruído desse arranjo.

$$\begin{bmatrix} 1 & & & \\ 1 & 0 & & \\ 0 & 1 & 0 & \\ 1 & 0 & 2 & 0 \end{bmatrix}$$

Em um grafo direcionado, a matriz de adjacência A reflete o sentido de orientação dos arcos. Nesse caso,

$$a_{ij} = p \text{ se existem } p \text{ arcos de } n_i \text{ para } n_j$$

A matriz de adjacência de um grafo direcionado não será simétrica, pois a existência de um arco de n_i para n_j não implica a existência de um arco de n_j para n_i .

❖ EXEMPLO 18 Considere o grafo direcionado da Fig. 5.26.

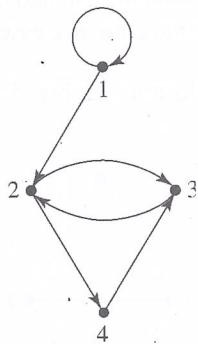


Fig. 5.26

A matriz de adjacência é

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



Em um grafo simples com peso, os elementos na matriz de adjacência podem indicar o peso de um arco pelo número apropriado, ao invés de apenas indicar a presença de um arco pelo número 1.

Lista de Adjacência

Muitos grafos, longe de serem completos, têm relativamente poucos arcos. Esses grafos têm matrizes de adjacência **esparsas**, isso é, a matriz de adjacência contém muitos zeros. Ainda assim, se o grafo tem n nós, precisamos de n^2 dados para representar a matriz de adjacência (ou mais de $n^2/2$, se for usada uma matriz triangular), mesmo que muitos desses dados sejam iguais a zero. Qualquer algoritmo ou procedimento no qual todo arco tem que ser examinado necessita olhar todos os n^2 elementos na matriz, já que não há como saber quais dos elementos são nulos sem examiná-los. Para encontrar todos os nós adjacentes a um determinado nó n_i é necessário olhar toda a i -ésima linha da matriz de adjacência, um total de n elementos.

Um grafo com relativamente poucos arcos pode ser representado de modo mais eficiente armazenando-se apenas os elementos não-nulos da matriz de adjacência. Essa representação consiste em uma lista, para cada nó, de todos os nós adjacentes a ele. São usados ponteiros para se ir de um item na lista para o próximo. Essa estrutura é chamada de uma **lista encadeada**. Existe um arranjo de n ponteiros, um para cada nó, para começar cada lista. Essa **lista de adjacência**, embora precise de armazenagem extra para os ponteiros, ainda pode ser mais eficiente do que uma matriz de adjacência. Para encontrar todos os nós adjacentes a n_i , é preciso percorrer a lista encadeada para n_i , que pode ter muito menos do que os n elementos que teríamos que examinar na matriz de adjacência. Existem algumas desvantagens, no entanto; se quisermos determinar se um determinado nó n_i é adjacente a n_j , podemos ter que percorrer toda a lista encadeada de n_i , ao passo que, na matriz de adjacência, podemos acessar o elemento i, j diretamente.

❖ EXEMPLO 19

A lista de adjacência para o grafo na Fig. 5.25 contém um arranjo de ponteiros com quatro elementos, um para cada nó. O ponteiro de cada nó aponta para um nó adjacente, que aponta para outro nó adjacente, e assim por diante. A estrutura de lista de adjacência está ilustrada na Fig. 5.27.

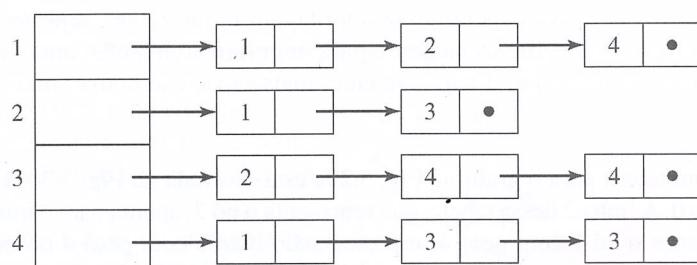


Fig. 5.27

Na figura, o ponto indica um **ponteiro nulo**, que significa que não há nada mais a se indicar, ou que se chegou ao final da lista. Tratamos os arcos paralelos listando um determinado nó mais de uma vez na lista de adjacência para n_i se existir mais de um arco entre n_i e esse nó.

Desenhe a lista de adjacência para o grafo ilustrado na Fig. 5.28.

PROBLEMA PRÁTICO 17

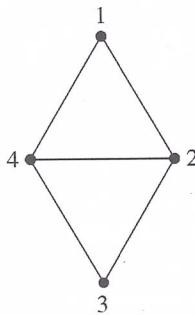
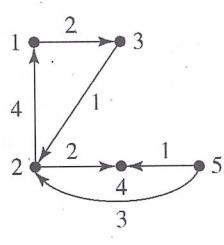


Fig. 5.28

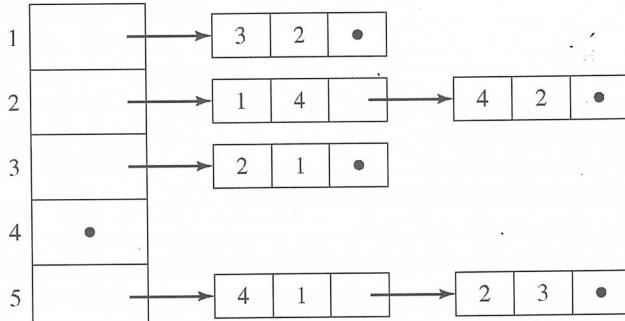
Em um grafo não-direcionado, cada arco é representado duas vezes. Se n_j está na lista de adjacência de n_i , então n_i também está na lista de adjacência de n_j . A lista de adjacência para um grafo direcionado coloca n_j na lista de adjacência de n_i se existir um arco de n_i para n_j ; n_i não precisa, necessariamente, estar na lista de adjacência de n_j . Para um grafo rotulado ou com pesos, dados adicionais podem ser armazenados junto com o rótulo do nó na lista de adjacência.

A Fig. 5.29a mostra um grafo direcionado com pesos. A lista de adjacência para esse grafo está ilustrada na Fig. 5.29b. Para cada registro na lista, a primeira componente é o nó, a segunda é o peso do arco que termina no nó e a terceira é o ponteiro. Note que a quarta componente no arranjo dos ponteiros iniciais é nulo, já que não existe nenhum arco saindo do nó 4.

EXEMPLO 20



(a)



(b)

Fig. 5.29

Em uma linguagem de programação que não usa ponteiros, ainda podemos obter o mesmo efeito de uma lista de adjacência usando um arranjo em várias colunas (ou uma tabela de registros), onde uma coluna contém os nós e uma outra coluna contém o número do próximo nó na lista de adjacência — um “pseudoponteiro”. A desvantagem dessa abordagem é que talvez seja necessário reservar a quantidade máxima de espaço de armazenamento para um grafo com n nós; uma vez que começamos a encher a tabela, não podemos criar, dinamicamente, mais espaço se houver mais nós adjacentes do que o esperado.

Uma representação em tabela para o grafo na Fig. 5.29a está ilustrada na Fig. 5.30. Um ponteiro nulo é indicado pelo número 0. A linha 2 dessa tabela, que representa o nó 2, aponta para o número 7. No número 7 da tabela, encontramos o nó 1 com peso 4, representando o arco com peso 4 do nó 2 para o nó 1. O ponteiro com o número 8 indica que a lista de adjacência do nó 2 tem mais elementos. No número 8, vemos que existe um arco de 2 para 4 com peso 2, o que completa a lista de adjacência para o nó 2.

EXEMPLO 21

Nó	Peso	Ponteiro
1		6
2		7
3		9
4		0
5		10
6	3	2
7	1	4
8	4	2
9	2	1
10	4	1
11	2	3

Fig. 5.30

SEÇÃO 5.1 REVISÃO

TÉCNICAS

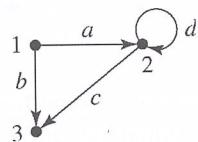
- ❖ Utilização da terminologia de grafos.
- W ❖ Demonstração de que dois grafos são, ou não são, isomorfos.
- ❖ Determinação de uma representação planar de um grafo simples ou demonstração de que não existe tal representação.
- W ❖ Construção de matrizes de adjacência e listas de adjacências para grafos e grafos direcionados.

IDÉIAS PRINCIPAIS

- ❖ Situações variadas podem ser modeladas com grafos.
- ❖ Grafos podem ser representados em um computador por matrizes ou listas encadeadas.

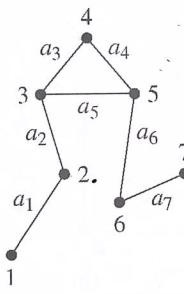
EXERCÍCIOS 5.1

1. Dê a função g que é parte da definição formal do grafo direcionado ilustrado.



2. Responda as perguntas a seguir sobre o grafo na figura abaixo:

- O grafo é simples?
- O grafo é completo?
- O grafo é conexo?
- Você pode encontrar dois caminhos de 3 para 6?
- Você pode encontrar um ciclo?
- Você pode encontrar um arco cuja remoção transformará o grafo em um grafo acíclico?
- Você pode encontrar um arco cuja remoção transformará o grafo em um grafo não-conexo?

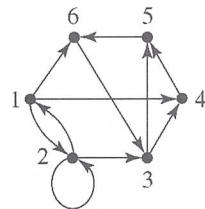


★3. Esboce um desenho para cada um dos grafos indicados a seguir:

- um grafo simples com três nós, cada um de grau 2;
- um grafo com quatro nós e ciclos de comprimento 1, 2, 3 e 4;
- um grafo não completo com quatro nós, cada um de grau 4.

★4. Use o grafo direcionado na figura para responder às perguntas abaixo:

- Quais nós são acessíveis a partir do nó 3?
- Qual o comprimento do caminho mais curto do nó 3 para o nó 6?
- Qual o caminho de comprimento 8 do nó 1 para o nó 6?



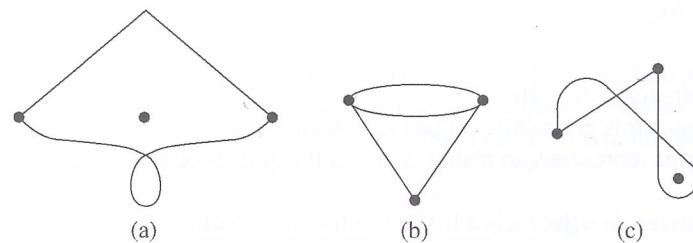
5. a. Desenhe K_6 .

b. Desenhe $K_{3,4}$.

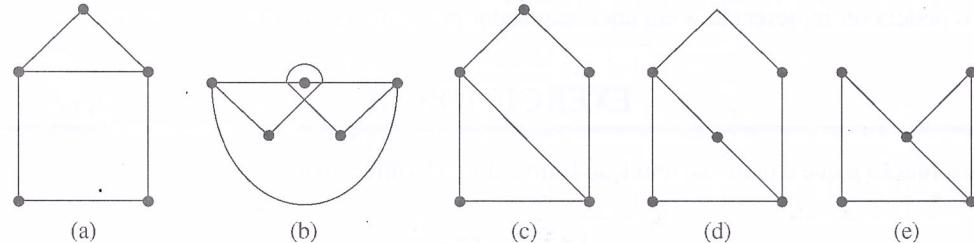
6. Desenhe um grafo com cada uma das características a seguir, ou explique por que não existe um tal grafo:

- quatro nós de graus 1, 2, 3 e 4, respectivamente;
- simples com quatro nós de graus 1, 2, 3 e 4, respectivamente;
- quatro nós de graus 2, 3, 3 e 4, respectivamente;
- quatro nós de graus 2, 3, 3 e 3, respectivamente.

7. Qual dos grafos a seguir não é isomorfo aos outros e por quê?

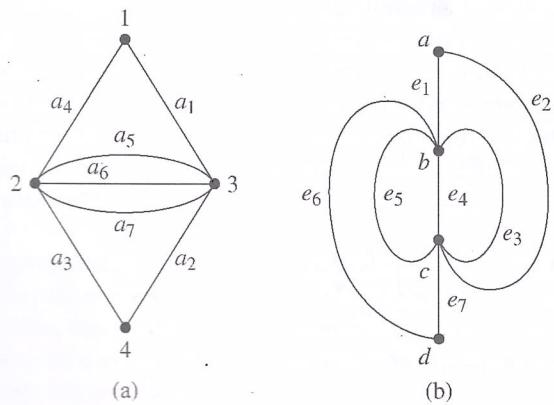


8. Qual dos grafos a seguir não é isomorfo aos outros e por quê?

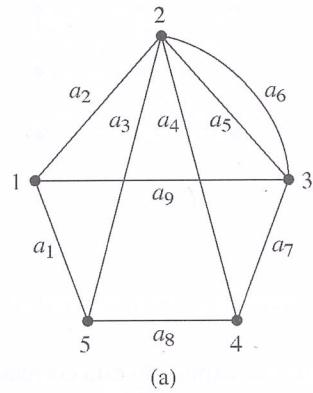


Nos Exercícios 9 a 14, decida se os dois grafos são isomorfos. Se forem, dê uma função ou funções que estabeleçam o isomorfismo; se não forem, explique por quê.

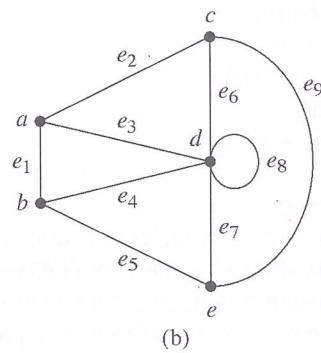
★9.



10.

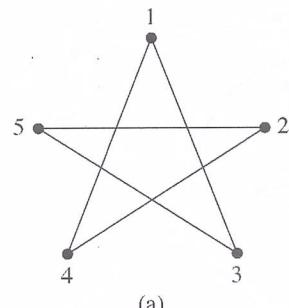


(a)

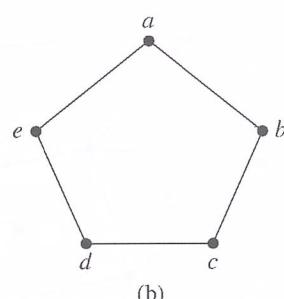


(b)

11.

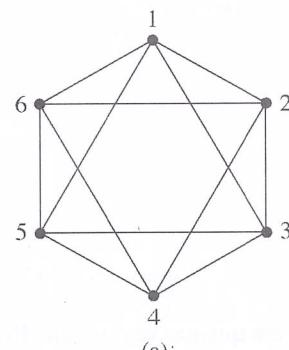


(a)

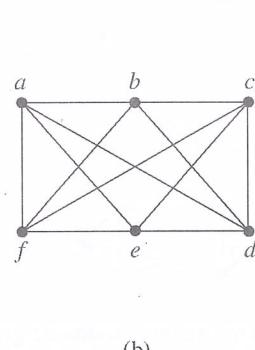


(b)

12.

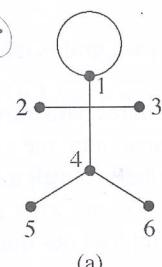


(a)

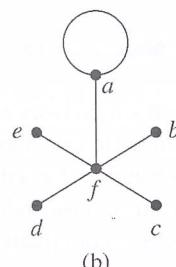


(b)

★13.

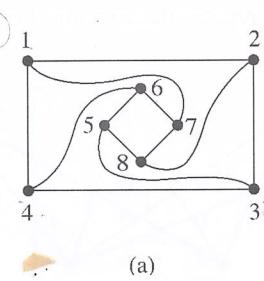


(a)

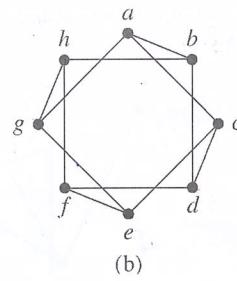


(b)

14.

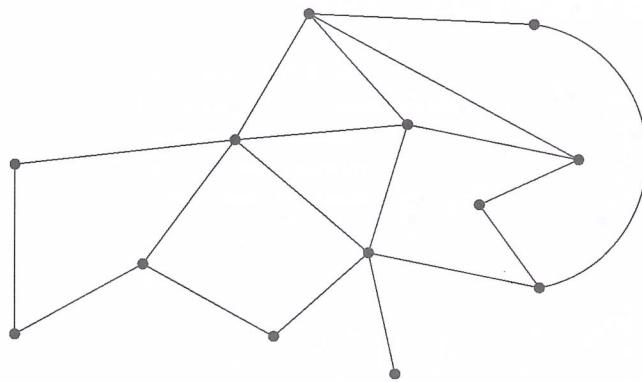


(a)

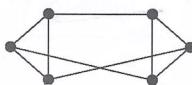


(b)

15. Prove que dois grafos não são isomorfos se
- um tem mais nós do que o outro;
 - um tem mais arcos do que o outro;
 - um tem arcos paralelos e o outro não;
 - um tem um laço e o outro não;
 - um tem um nó de grau k e o outro não;
 - um é conexo e o outro não;
 - um tem um ciclo e o outro não.
- ★16. Desenhe todos os grafos não-isomorfos simples com dois nós.
17. Desenhe todos os grafos não-isomorfos simples com três nós.
18. Desenhe todos os grafos não-isomorfos simples com quatro nós.
19. Encontre uma expressão para o número de arcos em K_n e prove que sua expressão está correta.
20. Verifique a fórmula de Euler para o grafo planar simples e conexo na figura abaixo.

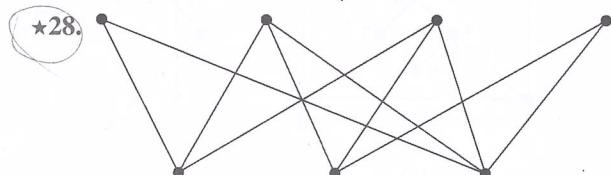
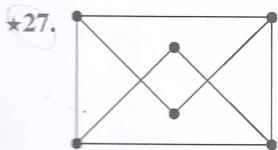


- ★21. Prove que $K_{2,3}$ é um grafo planar.
22. Prove que o grafo na figura a seguir é planar.

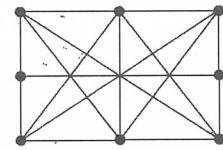


- ★23. Se um grafo planar simples e conexo tem seis nós, todos de grau 3, em quantas regiões ele divide o plano?
24. Se todos os nós de um grafo planar simples e conexo têm grau 4 e se o número de arcos é 12, em quantas regiões ele divide o plano?
25. A fórmula de Euler (Eq. (1) do teorema sobre o número de nós e arcos) é válida para grafos que não são simples? E as desigualdades (2) e (3) nesse teorema?
26. O que está errado no argumento a seguir que indica como usar subdivisões elementares para transformar um grafo que não é planar em um grafo planar? Em um grafo que não é planar, têm que existir dois arcos a_i e a_j que se intersectam em um ponto v que não é um nó. Faça uma subdivisão elementar em a_i inserindo um nó em v e uma subdivisão elementar em a_j inserindo um nó em v . No grafo resultante, o ponto de interseção é um nó. Repita esse processo com quaisquer interseções que não sejam nós; o resultado é um grafo planar.

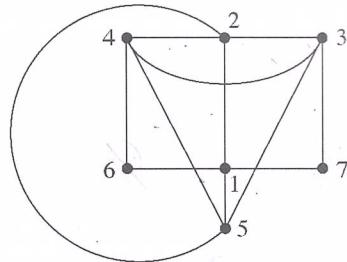
Para os Exercícios 27 a 30, determine se o grafo é planar (encontrando uma representação planar) ou não (encontrando um subgrafo homeomorfo a K_5 ou $K_{3,3}$).



★29.

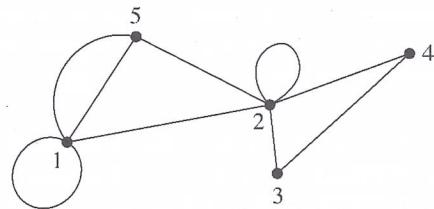


30.

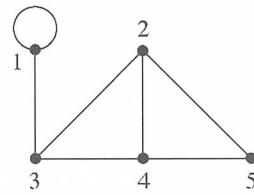


Para os Exercícios 31 a 36, escreva a matriz de adjacência do grafo na figura dada.

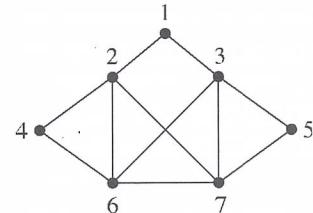
★31.



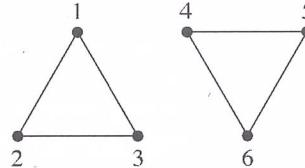
32.



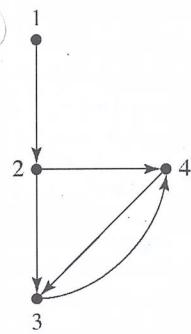
33.



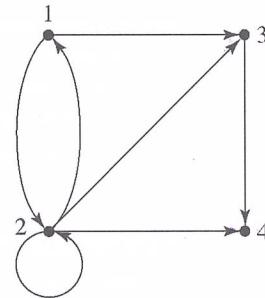
34.



★35.



36.



Para os Exercícios 37 a 40, desenhe o grafo representado pela matriz de adjacência.

★37.

$$\begin{bmatrix} 0 & 2 & 0 \\ 2 & 0 & 2 \\ 0 & 2 & 0 \end{bmatrix}$$

38.

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 2 & 0 \end{bmatrix}$$

★39.

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

40.

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

- ★41. A matriz de adjacência de um grafo não-direcionado é dada, em forma triangular inferior, por

$$\begin{bmatrix} 2 & & & & \\ 1 & 0 & & & \\ 0 & 1 & 1 & & \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Desenhe o grafo.

42. A matriz de adjacência de um grafo direcionado é dada por

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 2 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Desenhe o grafo.

- ★43. Descreva o grafo cuja matriz de adjacência é a matriz identidade $n \times n$, I_n .

44. Descreva a matriz de adjacência de K_n , o grafo simples completo com n nós.

45. Dada a matriz de adjacência A de um grafo direcionado G , descreva o grafo representado pela matriz de adjacência A^T (veja o Exercício 17 na Seção 4.5).

Para os Exercícios 46 a 51, desenhe a lista de adjacência do grafo no exercício indicado.

46. Exercício 31.

47. Exercício 32.

- ★48. Exercício 33.

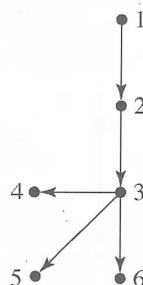
49. Exercício 34.

50. Exercício 35.

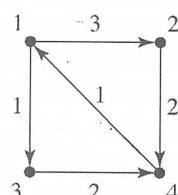
51. Exercício 36.

52. Considere o grafo ilustrado na figura.

- Desenhe a lista de adjacência.
- Quantos locais de armazenagem são necessários para a lista de adjacência?
(Um ponteiro usa um local de armazenagem.)
- Quantos locais de armazenagem seriam necessários para a matriz de adjacência desse grafo?



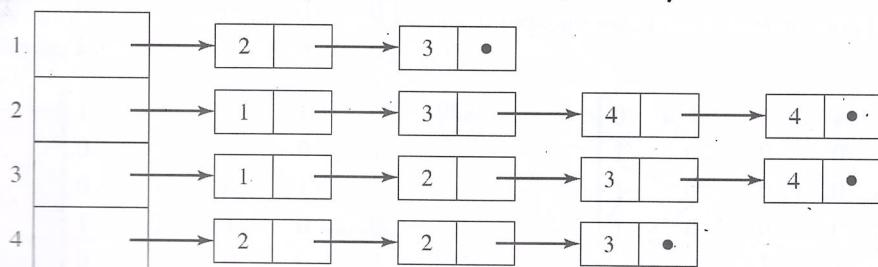
- ★53. Desenhe a lista de adjacência para o grafo direcionado com pesos da figura a seguir:



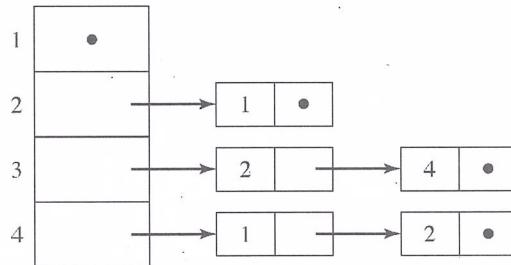
- ★54. Para o grafo direcionado no Exercício 36, construa uma representação em tabela.

55. Para o grafo direcionado com pesos no Exercício 53, construa uma representação em tabela.

56. Desenhe o grafo não-direcionado representado pela lista de adjacência a seguir:



57. Desenhe o grafo não-direcionado representado pela lista de adjacência a seguir:



Os Exercícios 58 a 64 tratam do *complementar* de um grafo. Se G é um grafo simples, o complementar de G , denotado por G' , é o grafo simples com o mesmo conjunto de nós que G e onde os nós $x-y$ são adjacentes se, e somente se, eles não são adjacentes em G .

- ★58. Desenhe G' para o grafo na Fig. 5.18a.

59. Desenhe K_4' .

60. Mostre que, se dois grafos simples G_1 e G_2 são isomorfos, então seus complementares G_1' e G_2' também o são.

61. Um grafo simples é *autocomplementar* se é isomorfo a seu complemento. Prove que em um grafo autocomplementar com n nós ($n > 1$) $n = 4k$ ou $n = 4k + 1$ para algum inteiro k . (Sugestão: Use o resultado do Exercício 19.)

- ★62. a. Prove que em qualquer grafo simples G com pelo menos dois nós, se G não for conexo, então G' é conexo. (Sugestão: Se G não for conexo, então G' é uma coleção de subgrafos conexos “disjuntos”.)
b. Encontre um grafo G tal que G e G' são ambos conexos, mostrando, dessa forma, que a recíproca do item (a) é falsa.

63. Dada a matriz de adjacência A de um grafo simples, descreva a matriz de adjacência de G' .

64. Prove que, se $|N| \geq 11$ em um grafo simples e conexo G , então os grafos G e G' não podem ser ambos planares.

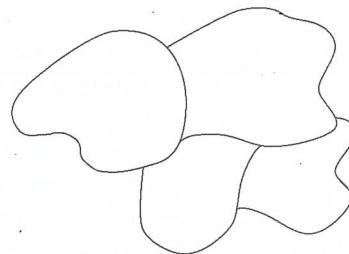
65. Prove que, em qualquer grafo simples G com n nós e a arcos, $2a \leq n^2 - n$.

- ★66. Prove que um grafo simples conexo com n nós tem pelo menos $n - 1$ arcos. (Sugestão: Mostre que essa proposição pode ser enunciada na forma “Um grafo simples conexo com m arcos tem, no máximo, $m + 1$ nós”. Depois use o segundo princípio de indução em m .)

67. Prove que um grafo simples com n nós ($n \geq 2$) e mais de $C(n - 1, 2)$ arcos é conexo. (Sugestão: Use os Exercícios 62 e 66.)

Os Exercícios 68 a 72 se referem ao problema de *colorir grafos*. A origem desses problemas é o *problema de colorir um mapa*: suponha que um mapa contendo vários países, desenhado em uma folha de papel, deve ser colorido de modo que dois países tendo uma fronteira comum não fiquem com a mesma cor. (Não precisamos nos preocupar com países que têm apenas um ponto de fronteira em comum e vamos supor que cada país é “conexo”.) Qual o número mínimo de cores necessário para se executar essa tarefa para qualquer mapa?

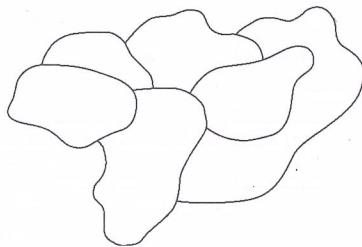
68. Mostre que, para colorir o mapa a seguir, são necessárias três cores e não mais do que isso.



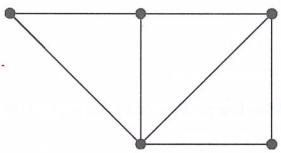
69. Desenhe um mapa que necessite de quatro cores.

70. Associado a qualquer mapa existe um grafo, chamado de *grafo dual* do mapa, formado da seguinte maneira: coloque um nó em cada região do mapa e um arco entre dois nós que representam países adjacentes.

- a. Desenhe o grafo dual do mapa no Exercício 68.
 b. Desenhe o grafo dual do mapa a seguir:

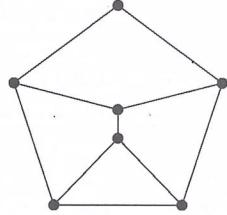


- c. Desenhe um mapa para o qual o grafo a seguir serviria como dual.

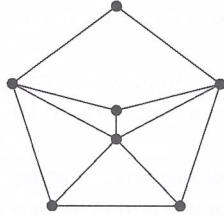


71. Colorir um grafo é atribuir uma cor a cada nó de modo que dois nós adjacentes nunca tenham a mesma cor. O número cromático de um grafo é o menor número de cores necessário para se colorir o grafo. Encontre o número cromático dos grafos a seguir:

★a.



b.



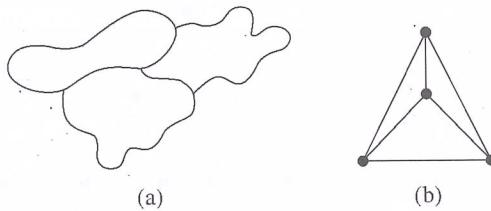
72. Pelo menos quatro cores são necessárias para se resolver o problema geral de colorir um mapa (veja o Exercício 69). Como ninguém conseguiu produzir um mapa que necessitasse de mais de quatro cores, foi formulada uma conjectura de que quatro cores são, de fato, suficientes. Essa conjectura ficou conhecida como o *problema das quatro cores*. Ele foi proposto, pela primeira vez, ao matemático Augustus de Morgan por um de seus alunos em 1852 e recebeu, mais tarde, bastante atenção. Permaneceu sem demonstração, no entanto, por mais de 100 anos. Em 1976, dois matemáticos da Universidade de Illinois, Wolfgang Haken e Kenneth Appel, usaram um computador para analisar um grande número de casos em uma demonstração por absurdo, verificando, assim, a conjectura das quatro cores.

O grafo dual de um mapa (veja o Exercício 70), pela maneira como é construído, será sempre um grafo planar simples e conexo. Além disso, qualquer grafo planar simples e conexo pode ser visto como o grafo dual de um mapa. Enuncie a conjectura das quatro cores em termos do número cromático (veja o Exercício 71) de um grafo.

73. Prove que um grafo planar simples e conexo com três ou mais nós tem pelo menos um nó com grau menor ou igual a 5. (Sugestão: Faça uma demonstração por absurdo.)

74. (Problema desafiador) O *problema das cinco cores* diz que o número cromático de um grafo planar simples e conexo é, no máximo, 5. Enquanto o teorema das quatro cores (Exercício 72) é muito difícil de provar, o teorema das cinco cores pode ser provado por indução no número de nós do grafo. Prove o teorema das cinco cores usando o resultado do Exercício 73.

75. O teorema das seis cores pode ser provado como um problema de colorir um mapa sem usar o grafo dual. Ao invés de criar o grafo dual, coloque nós nas interseções das fronteiras e retifique as fronteiras de modo que o problema de colorir o mapa ilustrado na parte (a) da figura a seguir é representado pelo problema de colorir as regiões limitadas do grafo na parte (b) da figura. Suponha, primeiro, que nenhum país tem um “buraco” no meio. Então o grafo não vai ter laços e vai ser planar e conexo. Além disso, todo nó terá grau pelo menos 3.



- ★a. Mostre que podemos supor que o grafo é simples provando que, se seis cores são suficientes para se colorir um grafo simples, então elas também são suficientes para um grafo com arcos paralelos. (*Sugestão:* Use países pequenos temporários nos nós.)
 - b. Prove que, em um grafo planar simples e conexo com R regiões limitadas, $n - a + R = 1$.
 - c. Considere um grafo planar simples e conexo e suponha que toda região limitada interior ao grafo tem, pelo menos, seis arestas adjacentes. Mostre que $2a \leq 3n - 3$.
 - d. Considere agora um grafo planar simples e conexo onde cada nó tem grau pelo menos 3. Mostre que um tal grafo tem pelo menos uma região limitada interior que não tem mais de cinco arestas adjacentes.
 - e. Prove que seis cores são suficientes para se colorir qualquer mapa onde nenhum país tem um buraco no meio.
 - f. Prove que seis cores são suficientes para se colorir qualquer mapa planar. (*Sugestão:* Faça alguns cortes temporários no mapa.)
76. Cinco lobistas estão visitando sete congressistas (denominados A a G) no mesmo dia. Os congressistas que cada um dos lobistas tem que ver estão indicados a seguir:
1. A, B, D
 2. B, C, F
 3. A, B, D, G
 4. E, G
 5. D, E, F
- Cada congressista estará disponível para se encontrar com os lobistas por uma hora. Qual o número mínimo de intervalos de tempo que serão usados para se marcar os encontros de uma hora de modo que nenhum lobista tenha conflitos de horário? (*Sugestão:* Trate esse problema como um problema de colorir um grafo.) E se o lobista 3 descobre que não precisa conversar com B e o lobista 5 descobre que não precisa se encontrar com D ?
77. Em uma máquina com processadores múltiplos, seis processadores indicados por letras de A até F dividem blocos em uma parte da memória para armazenar dados em comum. Dois processadores não podem escrever simultaneamente no mesmo bloco. A tabela a seguir indica quais os processadores que vão gravar dados no mesmo instante. Quantos blocos distintos são necessários? (*Sugestão:* trate esse como um problema de colorir um grafo.)
- | |
|-----------|
| A, F, C |
| B, D |
| F, D, A |
| B, E |
| F, C, E |

SEÇÃO 5.2 ÁRVORES E SUAS REPRESENTAÇÕES

TERMINOLOGIA DE ÁRVORES

Um tipo especial de grafo, chamado de *árvore*, é muito útil para representar dados.

Definição: Árvore Uma **árvore** é um grafo conexo acíclico com um nó especial, denominado **raiz** da árvore.

A Fig. 5.31 ilustra duas árvores. Só para ser do contra, o pessoal de computação insiste em desenhar as árvores com a raiz no topo. Um grafo conexo acíclico sem nenhum nó designado de raiz é chamado uma **árvore sem raiz** ou **árvore livre**. (Mais uma vez, a terminologia não é padrão. Alguns livros definem árvores como sendo grafos conexos acíclicos e depois chamam de “árvore enraizada” quando existe um nó designado por raiz.)

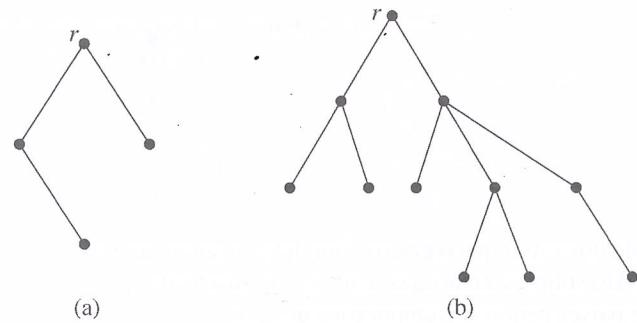


Fig. 5.31

Uma árvore também pode ser definida de maneira recorrente. Um único nó é uma árvore (com esse nó como raiz). Se T_1, T_2, \dots, T_r são árvores disjuntas com raízes r_1, r_2, \dots, r_r , o grafo formado colocando-se um novo nó r ligado, por um único arco, a cada um dos nós r_1, r_2, \dots, r_r é uma árvore com raiz r . Os nós r_1, r_2, \dots, r_r são os **filhos** de r e r é **pai** de r_1, r_2, \dots, r_r . A Fig. 5.32 mostra a última etapa na construção por recorrência da árvore na Fig. 5.31b. É muitas vezes útil percorrer uma estrutura de árvore de modo recorrente, considerando as subárvore como árvores menores.

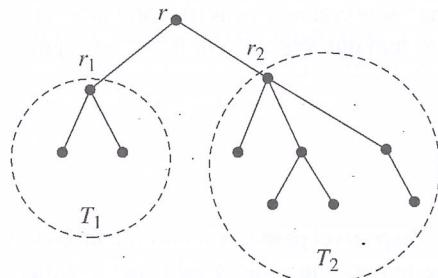


Fig. 5.32

Como uma árvore é um grafo conexo, existe um caminho da raiz para qualquer outro nó na árvore; como a árvore é acíclica, esse caminho é único. A **profundidade de um nó** em uma árvore é o comprimento do caminho da raiz ao nó; a raiz tem profundidade 0. A **profundidade (altura) de uma árvore** é a maior profundidade dos nós na árvore; em outras palavras, é o comprimento do caminho mais comprido da raiz até um dos nós. Um nó sem filhos é chamado de **folha** da árvore; todos os nós que não são folhas são **nós internos**. Uma **floresta** é um grafo acíclico (não necessariamente conexo); logo, uma floresta é uma coleção de árvores disjuntas. As Figs. 5.31a e 5.31b juntas formam uma floresta.

De interesse especial são as **árvores binárias**, onde cada nó tem, no máximo, dois filhos. Em uma árvore binária, cada filho de um nó é chamado de **filho esquerdo** ou **filho direito**. Uma **árvore binária cheia** é uma árvore que tem todos os nós internos com dois filhos e onde todas as folhas estão à mesma profundidade. A Fig. 5.33 mostra uma árvore binária de altura 4 e a Fig. 5.34 mostra uma árvore binária cheia de altura 3. Uma **árvore binária completa** é uma árvore binária que é quase cheia; o nível mais baixo da árvore vai se enchendo da esquerda para a direita mas pode ter folhas faltando. A Fig. 5.35 mostra uma árvore binária completa de altura 3. (Note que, embora uma árvore seja um grafo, uma árvore completa não é um grafo completo!)

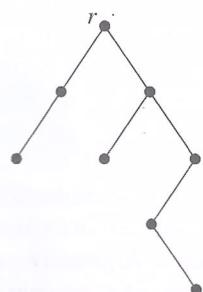


Fig. 5.33

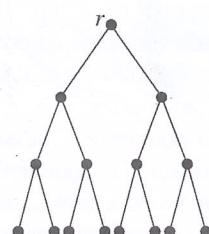


Fig. 5.34

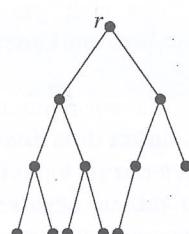


Fig. 5.35

PROBLEMA PRÁTICO 18

Responda as perguntas a seguir sobre a árvore binária ilustrada na Fig. 5.36. (Suponha que o nó 1 é a raiz.)

- Qual é a altura?
- Qual é o filho esquerdo do nó 2?
- Qual é a profundidade do nó 5?

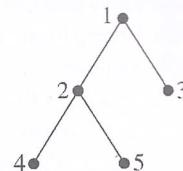


Fig. 5.36

APLICAÇÕES DE ÁRVORES

Árvores de decisão foram usadas para resolver problemas de contagem no Cap. 3 e serão usadas na Seção 5.3 para ajudar a estabelecer limites inferiores sobre a quantidade de operações de determinados algoritmos. O Exercício 31 da Seção 4.1 descreve a organização de dados em uma estrutura de árvore binária. Usando essas árvores, pode-se fazer uma busca eficiente em uma coleção de registros para localizar um registro particular ou verificar se um registro não pertence à coleção. Exemplos dessa busca seriam procurar um livro na biblioteca, procurar um registro médico de um paciente em um hospital ou procurar uma ficha de cadastro de um cliente em um banco. Vamos também considerar buscas em árvores binárias na Seção 5.3. As derivações de palavras em certas linguagens formais serão mostradas como árvores no Cap. 8 (essas são árvores de análise, geradas por um compilador ao analisar um programa de computador).

Uma árvore genealógica é, em geral, uma árvore de fato embora, se houver casamento entre parentes, seja um grafo mas não uma árvore no sentido técnico. (Informações obtidas de uma árvore genealógica não são apenas interessantes mas são também úteis para pesquisas em genética.) O fluxo organizacional que indica quem responde a quem em uma companhia grande ou outro empreendimento é, em geral, uma árvore (veja a Fig. 5.37).

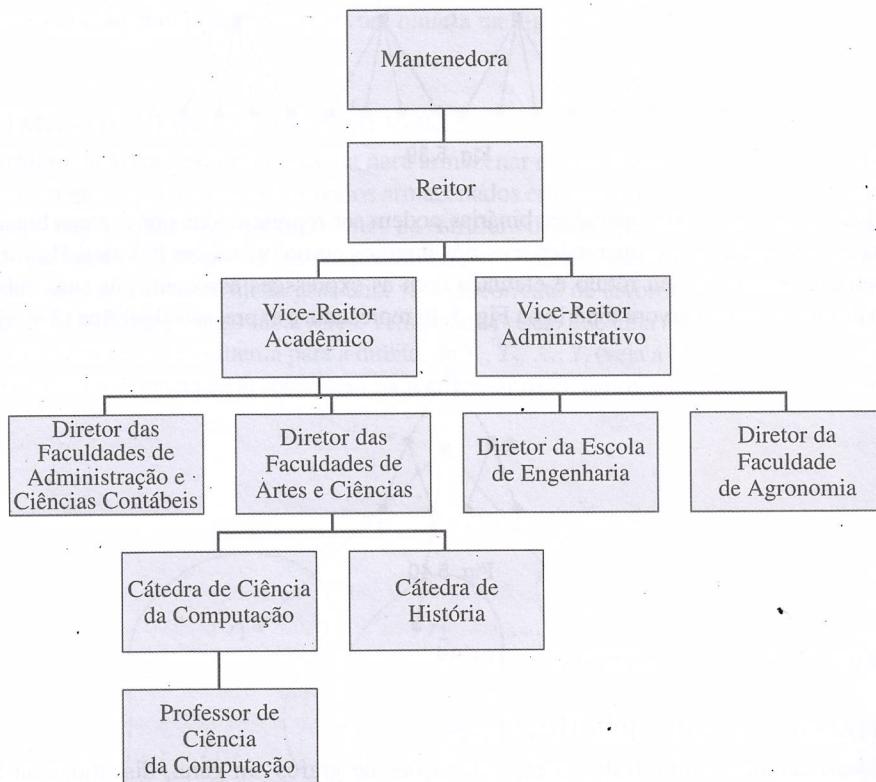


Fig. 5.37

Os arquivos em seu computador estão organizados em uma estrutura hierárquica (de árvore), assim como os tópicos em muitos sistemas de ajuda (*on-line Help systems*; veja a Fig. 5.38, onde há diversas árvores, cada uma enraizada à esquerda).

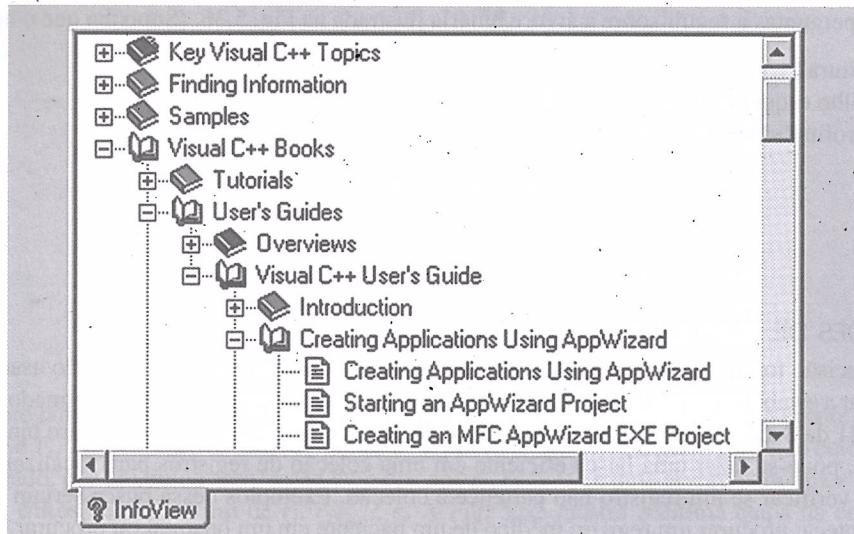


Fig. 5.38

Um vírus de computador espalha-se através do correio eletrônico. A cada segundo, 4 novas máquinas são infectadas. Uma estrutura de árvores quaternária (Fig. 5.39) mostra a disseminação do vírus. Pelo princípio de multiplicação, após n segundos 4^n máquinas terão sido infectadas.

◆ EXEMPLO 22

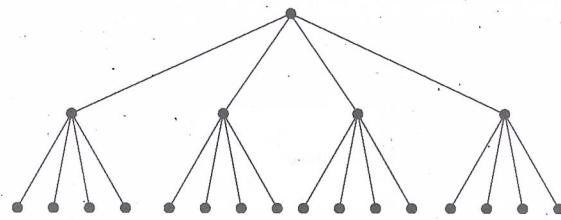


Fig. 5.39

Expressões algébricas envolvendo operações binárias podem ser representadas por árvores binárias rotuladas. As folhas são rotuladas como operandos e os nós internos como operações binárias. Para qualquer nó interno, a operação binária de seu rótulo é efetuada com as expressões associadas às suas subárvores da esquerda e da direita. Assim, a árvore binária na Fig. 5.40 representa a expressão algébrica $(2 + x) - (y * 3)$.

◆ EXEMPLO 23

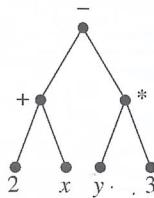


Fig. 5.40

Qual a árvore que representa a expressão $(2 + 3) * 5$?

◆ PROBLEMA PRÁTICO 19

REPRESENTAÇÃO DE ÁRVORES BINÁRIAS

Como uma árvore também é um grafo, as representações de grafos em geral, discutidas na Seção 5.1, também podem ser usadas para árvores. Árvores binárias, no entanto, têm características especiais que gostaríamos de capturar na representação, a saber, a identidade do filho esquerdo e do direito. O equivalente de uma matriz de adjacência é uma tabela com duas colunas (ou uma tabeleta de registros) onde os dados para cada nó são os filhos esquerdo e direito daquele nó. O equivalente de uma lista de adjacência é uma coleção de registros com três campos contendo, respectivamente, o nó em questão, um ponteiro para o registro do nó filho esquerdo e um ponteiro para o registro do nó filho direito.

❖ EXEMPLO 24

Para a árvore binária ilustrada na Fig. 5.41, a representação em tabela, com filhos esquerdo e direito, é dada na Fig. 5.42a. Mais uma vez, os zeros indicam ponteiros nulos. A representação com ponteiros é dada na Fig. 5.42b.

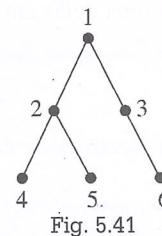
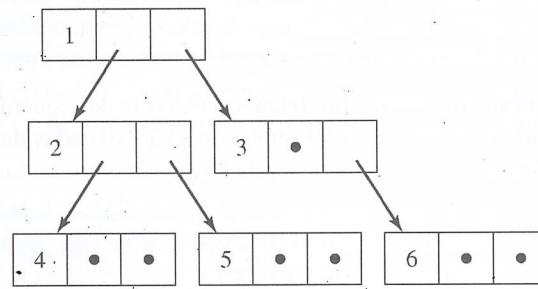


Fig. 5.41

Filho esquerdo Filho direito

	Filho esquerdo	Filho direito
1	2	3
2	4	5
3	0	6
4	0	0
5	0	0
6	0	0

(a)



(b)

Fig. 5.42

PROBLEMA
PRÁTICO 20

- Dê a representação em tabela, com filhos esquerdo e direito, da árvore binária na Fig. 5.43.
- Dê a representação com ponteiros da árvore binária na Fig. 5.43.

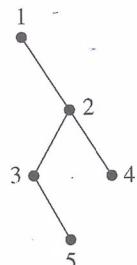


Fig. 5.43

ALGORITMOS DE PERCURSO EM ÁRVORES

Se uma estrutura de árvore está sendo usada para armazenar dados, é muitas vezes útil ter um mecanismo sistemático para escrever os valores dos dados armazenados em todos os nós. Isto pode ser feito *percorrendo-se* a árvore, isto é, visitando-se todos os nós na estrutura de árvore. Os três algoritmos mais comuns de **percurso em árvores** são os percursos em pré-ordem, em ordem simétrica e em pós-ordem.

Nesses métodos de percurso, ajuda usar uma visão recorrente de árvores, onde a raiz de uma árvore tem ramificações para as raízes de suas subárvore. Vamos supor então que uma árvore T tem uma raiz r ; quaisquer subárvore são chamadas, da esquerda para a direita, de T_1, T_2, \dots, T_t (veja a Fig. 5.44). Como estamos usando uma definição por recorrência de árvores, será fácil enunciar os algoritmos de percurso em forma recorrente.

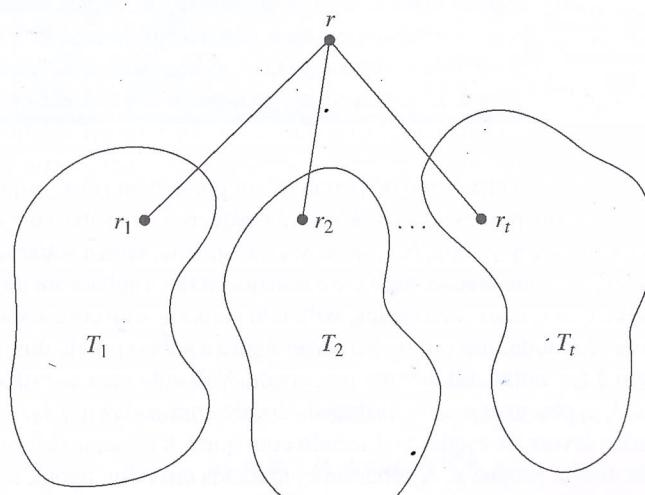


Fig. 5.44

Os termos *pré-ordem*, *ordem simétrica* e *pós-ordem* referem-se à ordem de visita da raiz em comparação aos nós das subárvore. No **percurso em pré-ordem**, a raiz é visitada primeiro e depois processam-se as subárvore, da esquerda para a direita, cada uma delas em pré-ordem.

ALGORITMO PRÉ-ORDEM

```

Pré-ordem(árvore  $T$ )
// Escreve os nós de uma árvore com raiz  $r$  em pré-ordem

    escreva( $r$ )
    para  $i = 1$  até  $t$  faça
        Pré-ordem( $T_i$ )
    fim do para
fim Pré-ordem

```

No **percurso em ordem simétrica**, a subárvore da esquerda é percorrida em ordem simétrica, depois a raiz é visitada e depois as outras subárvore são visitadas da esquerda para a direita, sempre em ordem simétrica. Se a árvore for binária, a raiz é visitada entre as duas subárvore.

ALGORITMO ORDEM SIMÉTRICA

```

OrdemSimétrica(árvore  $T$ )
// Escreve os nós de uma árvore com raiz  $r$  em ordem simétrica

    OrdemSimétrica( $T_1$ )
    escreva( $r$ )
    para  $i = 2$  até  $t$  faça
        OrdemSimétrica( $T_i$ )
    fim do para
fim OrdemSimétrica

```

Finalmente, no **percurso em pós-ordem**, a raiz é a última a ser visitada, após o percurso, em pós-ordem, de todas as subárvore da esquerda para a direita.

ALGORITMO PÓS-ORDEM

```

Pós-ordem(árvore  $T$ )
// Escreve os nós de uma árvore com raiz  $r$  em pós-ordem

    para  $i = 1$  até  $t$  faça
        Pós-ordem( $T_i$ )
    fim do para
    escreva( $r$ )
fim Pós-ordem

```

Para a árvore binária na Fig. 5.45, o algoritmo de percurso em pré-ordem (raiz, esquerda, direita) diz para escrever a raiz a primeiro e depois processar a subárvore da esquerda. Na subárvore da esquerda, com raiz b , um percurso em pré-ordem escreve a raiz, b , e move-se, novamente, para a subárvore da esquerda, que é o nó d . Esse único nó é a raiz de uma árvore, logo ele é escrito. Então a subárvore de d da esquerda (vazia) e a subárvore de d da direita (vazia) são percorridas. Voltando para a árvore enraizada em b , já percorremos sua subárvore da esquerda, de modo que vamos percorrer agora a subárvore da direita, escrevendo o nó e . A subárvore enraizada em b foi, então, totalmente percorrida. Voltando para a , está na hora de percorrer a subárvore de a da direita. Um percurso em pré-ordem da árvore enraizada em c faz com que c seja escrito, depois o percurso vai para a árvore da esquerda, fazendo com que f , h e i sejam escritos. Voltando para c , o percurso da subárvore da direita produz g . A subárvore enraizada em c foi, agora, totalmente percorrida e o algoritmo termina. O percurso em pré-ordem produziu

$a, b, d, e, c, f, h, i, g$

EXEMPLO 25

LEMBRETE:

Para uma árvore binária:

Percurso em pré-ordem é raiz, esquerda, direita.

Percurso em ordem simétrica é esquerda, raiz, direita.

Percurso em pós-ordem é esquerda, direita, raiz.

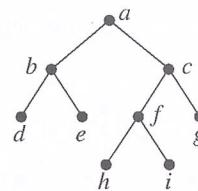


Fig. 5.45

♦ EXEMPLO 26

Usando novamente a árvore na Fig. 5.45, o percurso em ordem simétrica (esquerda, raiz, direita) vai até a subárvore da esquerda mais longe, enraizada em d . No percurso em ordem simétrica percorremos a subárvore da esquerda (vazia), escrevemos a raiz d e percorremos a subárvore da direita (vazia). Voltando para a árvore enraizada em b , já visitamos a subárvore da esquerda, logo está na hora de escrever a raiz, b . Indo agora para a subárvore de b da direita, escrevemos e . Voltando para a , já percorremos a subárvore da esquerda, logo a raiz, a , é escrita. Indo agora para a subárvore de a da direita, um percurso em ordem simétrica diz para se ir, primeiro, para a subárvore da esquerda mais longe, o que faz com que h seja escrito. Depois disso, escreve-se f e i , depois a raiz c e depois a subárvore de c da direita, que é g . Os nós, portanto, são escritos como

$$d, b, e, a, h, f, i, c, g$$

Um percurso em pós-ordem (esquerda, direita, raiz) produziria

$$d, e, b, h, i, f, g, c, a$$

♦ EXEMPLO 27

Considere a árvore ilustrada na Fig. 5.46, que não é uma árvore binária. Um percurso em pré-ordem escreve primeiro a raiz a e depois percorre a subárvore da esquerda, enraizada em b , em pré-ordem. O percurso em pré-ordem dessa subárvore escreve b e percorre em pré-ordem a subárvore da esquerda, enraizada em d . O nó d é escrito e depois percorre-se em pré-ordem a subárvore de d da esquerda, que está enraizada em i . Depois de escrever i , o percurso volta para considerar quaisquer outras subárvore de d ; não existe nenhuma outra.

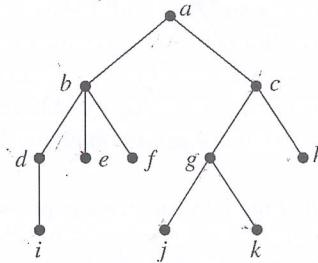


Fig. 5.46

Voltando para b , existem outras subárvore de b . Processando essas subárvore da esquerda para a direita, escreve-se os nós e e f . Já percorremos, então, todas as subárvore de b . Voltando para a para procurar outras subárvore, encontramos uma enraizada em c . O algoritmo escreve a raiz c , depois move-se para sua subárvore mais à esquerda enraizada em g e escreve g . Processando as subárvore de g , escrevemos j e k ; depois, voltando para c , sua subárvore restante é processada, produzindo h . O nó c não tem outras subárvore; voltando para a , a não tem outras subárvore e o algoritmo termina. A lista dos nós no percurso em pré-ordem é

$$a, b, d, i, e, f, c, g, j, k, h$$

Para fazer um percurso em ordem simétrica na árvore na Fig. 5.46, processa-se primeiro as subárvore da esquerda. Isso nos leva ao nó i , que não tem subárvore. Então escreve-se i . Voltando para d , a subárvore de d da esquerda já foi percorrida, logo escreve-se d . Como o nó d não tem outras subárvore, o algoritmo volta para b . A subárvore de b da esquerda já foi percorrida, logo escreve-se b e as subárvore restantes de b são percorridas, escrevendo-se e e f . Voltando para a , escreve-se a e depois processa-se a subárvore de a da direita. Isso nos leva aos nós j , g , k , c e h , nessa ordem, e terminamos. Portanto, a lista dos nós em ordem simétrica é

$$i, d, b, e, f, a, j, g, k, c, h$$

A lista a seguir resulta de um percurso em pós-ordem:

$$i, d, e, f, b, j, k, g, h, c, a$$

Percorra a árvore na Fig. 5.47 em pré-ordem, em ordem simétrica e em pós-ordem.

PROBLEMA PRÁTICO 21

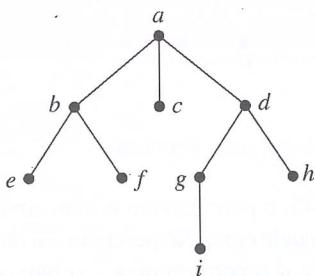


Fig. 5.47

O Exemplo 23 mostrou como expressões algébricas podem ser representadas como árvores binárias. Se fizermos um percurso em ordem simétrica da árvore, obteremos a expressão algébrica original. Para a árvore na Fig. 5.48, por exemplo, um percurso em ordem simétrica nos dá a expressão

$$(2 + x) * 4$$

onde os parênteses são adicionados ao se completar o processamento de uma subárvore. Essa forma de expressão algébrica, onde o símbolo da operação aparece entre os dois operandos, é chamada de **notação infixada**. Aqui os parênteses são necessários para indicar a ordem das operações. Sem parênteses, a expressão torna-se $2 + x * 4$, que também é uma expressão infixada, mas, devido à ordem de precedência da multiplicação em relação à soma, não é o que queremos.

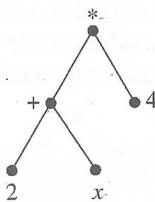


Fig. 5.48

Um percurso em pré-ordem na árvore da Fig. 5.48 fornece a expressão

$$* + 2 x 4$$

Aqui o símbolo da operação precede o operando. Essa forma de uma expressão é chamada de **notação prefixada** ou **notação polonesa**². Essa expressão pode ser colocada em forma infixada da seguinte maneira:

$$* + 2 x 4 \rightarrow * (2 + x) 4 \rightarrow (2 + x) * 4$$

Um percurso em pós-ordem nos dá a expressão

$$2 x + 4 *$$

onde o símbolo da operação vem após os operandos. Essa forma de uma expressão é chamada de **notação posfixada** ou **notação polonesa reversa** (ou, simplesmente, **NPR**). A expressão pode ser colocada em forma infixada da seguinte maneira:

$$2 x + 4 * \rightarrow (2 + x) 4 * \rightarrow (2 + x) * 4$$

Nem a notação polonesa nem a polonesa inversa precisa de parênteses para evitar ambigüidades. Essas notações fornecem, portanto, representações mais eficientes, embora menos familiares, de expressões algébricas do que em notação infixada. Tais formas podem ser calculadas seqüencialmente, sem precisar “olhar adiante” para procurar expressões entre parênteses. Compiladores mudam, muitas vezes, expressões algébricas em programas de computador de notação infixada para a polonesa reversa para obter um processamento mais eficiente.

²Assim chamada por causa do lógico polonês, J. Lukasiewicz, que foi o primeiro a usá-la.

EXEMPLO 28

PROBLEMA PRÁTICO 22

Escreva a árvore que representa a expressão

$$a + (b * c - d)$$

e escreva a expressão em notações polonesa e polonesa reversa.

RESULTADOS SOBRE ÁRVORES

Árvores formam um campo fértil (isso não é uma piada) para demonstrações por indução no número de nós, ou no número de arcos, ou na altura.

EXEMPLO 29

Após desenhar algumas árvores e fazer algumas contagens, parece que o número de arcos em uma árvore é sempre o número de nós menos 1. Mais formalmente, parece que

Uma árvore com n nós tem $n - 1$ arcos.

Provaremos essa proposição por indução em n , $n \geq 1$. Para a base da indução, $n = 1$, a árvore consiste em um único nó e nenhum arco (Fig. 5.49), logo o número de arcos é o número de nós menos 1.

- $n = 1, a = 0$

Fig. 5.49

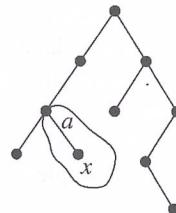


Fig. 5.50

Suponha que qualquer árvore com k nós tem $k - 1$ arcos e considere uma árvore com $k + 1$ nós. Queremos mostrar que essa árvore tem k arcos. Seja x uma folha da árvore (tem que existir uma folha, já que a árvore é finita). Então x tem um único pai. Remova da árvore o nó x e o único arco a que o liga a seu pai (veja a Fig. 5.50). O grafo restante ainda é uma árvore e tem k nós. Portanto, pela hipótese de indução, ele tem $k - 1$ arcos e a árvore original, contendo o arco a , tem $(k - 1) + 1 = k$ arcos. A demonstração está completa.

Note que, na demonstração por indução do Exemplo 29, tivemos que usar muito mais palavras do que em algumas de nossas demonstrações por indução anteriores. No Exemplo 15 do Cap. 2, por exemplo, a demonstração por indução de que

$$1 + 2 + 2^2 + \cdots + 2^n = 2^{n+1} - 1$$

consistiu, basicamente, de manipulações de expressões matemáticas nessa equação, mas agora tivemos que fazer um argumento mais verbal. Não somente não tem problema usar palavras em uma demonstração, como elas podem formar a parte principal da demonstração.

A demonstração por indução no Exemplo 29 difere ainda de outra maneira de demonstrações como a do Exemplo 15 no Cap. 2. Naquelas demonstrações, havia sempre uma parcela na série (a última parcela) cuja remoção nos levaria ao “caso $P(k)$ ”, a hipótese de indução. Em demonstrações envolvendo árvores com $k + 1$ nós, que nó deveria ser removido para se obter o caso $P(k)$? Em geral, o nó a ser removido não é único mas também não é completamente arbitrário. Na demonstração do Exemplo 29, por exemplo, a remoção de um nó interno (e os arcos ligados a ele) de uma árvore com $k + 1$ nós resultaria em um grafo com k nós, mas não em uma árvore com k nós, logo não poderíamos usar a hipótese de indução.

PROBLEMA PRÁTICO 23

Prove que, em qualquer árvore com n nós, o número total de extremidades de arcos é $2n - 2$. Use indução no número de nós.

EXEMPLO 30

Algumas vezes uma observação astuta pode tomar o lugar de uma demonstração por indução. Todos os arcos de uma árvore ligam um nó a seu pai. Todos os nós de uma árvore, exceto a raiz, têm um pai, e há $n - 1$ desses nós, logo há $n - 1$ arcos. Cada arco tem duas extremidades, de modo que há $2(n - 1)$ extremidades de arcos. Esse argumento nos dá os resultados do Exemplo 29 e do Problema Prático 23.

Percorra a árvore na Fig. 5.47 em pré-ordem, em ordem simétrica e em pós-ordem.

PROBLEMA PRÁTICO 21

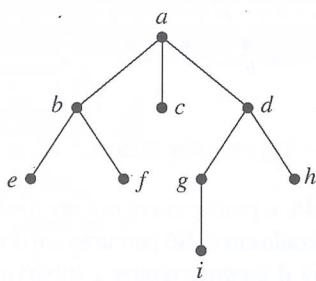


Fig. 5.47

O Exemplo 23 mostrou como expressões algébricas podem ser representadas como árvores binárias. Se fizermos um percurso em ordem simétrica da árvore, obteremos a expressão algébrica original. Para a árvore na Fig. 5.48, por exemplo, um percurso em ordem simétrica nos dá a expressão

$$(2 + x) * 4$$

onde os parênteses são adicionados ao se completar o processamento de uma subárvore. Essa forma de expressão algébrica, onde o símbolo da operação aparece entre os dois operandos, é chamada de **notação infixa**. Aqui os parênteses são necessários para indicar a ordem das operações. Sem parênteses, a expressão torna-se $2 + x * 4$, que também é uma expressão infixia mas, devido à ordem de precedência da multiplicação em relação à soma, não é o que queremos.

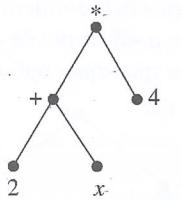


Fig. 5.48

Um percurso em pré-ordem na árvore da Fig. 5.48 fornece a expressão

$$* + 2 x 4$$

Aqui o símbolo da operação precede o operando. Essa forma de uma expressão é chamada de **notação prefixa** ou **notação polonesa**². Essa expressão pode ser colocada em forma infixia da seguinte maneira:

$$* + 2 x 4 \rightarrow * (2 + x) 4 \rightarrow (2 + x) * 4$$

Um percurso em pós-ordem nos dá a expressão

$$2 x + 4 *$$

onde o símbolo da operação vem após os operandos. Essa forma de uma expressão é chamada de **notação posfixa** ou **notação polonesa reversa** (ou, simplesmente, **NPR**). A expressão pode ser colocada em forma infixia da seguinte maneira:

$$2 x + 4 * \rightarrow (2 + x) 4 * \rightarrow (2 + x) * 4$$

Nem a notação polonesa nem a polonesa inversa precisa de parênteses para evitar ambigüidades. Essas notações fornecem, portanto, representações mais eficientes, embora menos familiares, de expressões algébricas do que em notação infixia. Tais formas podem ser calculadas seqüencialmente, sem precisar “olhar adiante” para procurar expressões entre parênteses. Compiladores mudam, muitas vezes, expressões algébricas em programas de computador de notação infixia para a polonesa reversa para obter um processamento mais eficiente.

²Assim chamada por causa do lógico polonês, J. Lukasiewicz, que foi o primeiro a usá-la.

EXEMPLO 28

PROBLEMA PRÁTICO 22

Escreva a árvore que representa a expressão

$$a + (b * c - d)$$

e escreva a expressão em notações polonesas e polonesa reversa.

RESULTADOS SOBRE ÁRVORES

Árvores formam um campo fértil (isso não é uma piada) para demonstrações por indução no número de nós, ou no número de arcos, ou na altura.

EXEMPLO 29

Após desenhar algumas árvores e fazer algumas contagens, parece que o número de arcos em uma árvore é sempre o número de nós menos 1. Mais formalmente, parece que

Uma árvore com n nós tem $n - 1$ arcos.

Provaremos essa proposição por indução em n , $n \geq 1$. Para a base da indução, $n = 1$, a árvore consiste em um único nó e nenhum arco (Fig. 5.49), logo o número de arcos é o número de nós menos 1.

- $n = 1, a = 0$

Fig. 5.49

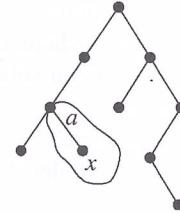


Fig. 5.50

Suponha que qualquer árvore com k nós tem $k - 1$ arcos e considere uma árvore com $k + 1$ nós. Queremos mostrar que essa árvore tem k arcos. Seja x uma folha da árvore (tem que existir uma folha, já que a árvore é finita). Então x tem um único pai. Remova da árvore o nó x e o único arco a que o liga a seu pai (veja a Fig. 5.50). O grafo restante ainda é uma árvore e tem k nós. Portanto, pela hipótese de indução, ele tem $k - 1$ arcos e a árvore original, contendo o arco a , tem $(k - 1) + 1 = k$ arcos. A demonstração está completa.

Note que, na demonstração por indução do Exemplo 29, tivemos que usar muito mais palavras do que em algumas de nossas demonstrações por indução anteriores. No Exemplo 15 do Cap. 2, por exemplo, a demonstração por indução de que

$$1 + 2 + 2^2 + \dots + 2^n = 2^{n+1} - 1$$

consistiu, basicamente, de manipulações de expressões matemáticas nessa equação, mas agora tivemos que fazer um argumento mais verbal. Não somente não tem problema usar palavras em uma demonstração, como elas podem formar a parte principal da demonstração.

A demonstração por indução no Exemplo 29 difere ainda de outra maneira de demonstrações como a do Exemplo 15 no Cap. 2. Naquelas demonstrações, havia sempre uma parcela na série (a última parcela) cuja remoção nos levaria ao “caso $P(k)$ ”, a hipótese de indução. Em demonstrações envolvendo árvores com $k + 1$ nós, que nó deveria ser removido para se obter o caso $P(k)$? Em geral, o nó a ser removido não é único mas também não é completamente arbitrário. Na demonstração do Exemplo 29, por exemplo, a remoção de um nó interno (e os arcos ligados a ele) de uma árvore com $k + 1$ nós resultaria em um grafo com k nós, mas não em uma árvore com k nós, logo não poderíamos usar a hipótese de indução.

PROBLEMA PRÁTICO 23

Prove que, em qualquer árvore com n nós, o número total de extremidades de arcos é $2n - 2$. Use indução no número de nós.

EXEMPLO 30

Algumas vezes uma observação astuta pode tomar o lugar de uma demonstração por indução. Todos os arcos de uma árvore ligam um nó a seu pai. Todos os nós de uma árvore, exceto a raiz, têm um pai, e há $n - 1$ desses nós, logo há $n - 1$ arcos. Cada arco tem duas extremidades, de modo que há $2(n - 1)$ extremidades de arcos. Esse argumento nos dá os resultados do Exemplo 29 e do Problema Prático 23.

SEÇÃO 5.2 REVISÃO

TÉCNICAS

- ❖ Construção de árvores que representam expressões algébricas.
- ❖ Construção de representações em tabela e com ponteiros de árvores binárias.
- ❖ Percursos em árvores em pré-ordem, em ordem simétrica e em pós-ordem.

W

IDÉIAS PRINCIPAIS

- ❖ Árvores binárias podem ser representadas por tabelas e por estruturas encadeadas.
- ❖ Existem procedimentos recorrentes para se visitar, de maneira sistemática, todos os nós de uma árvore binária.

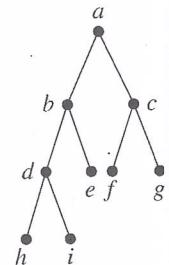
EXERCÍCIOS 5.2

- ★1. Esboce uma figura para cada uma das seguintes árvores:

- árvore com cinco nós e altura 1;
- árvore binária cheia de altura 2;
- árvore de altura 3 onde cada nó de profundidade i tem $i + 1$ filhos.

2. Responda as perguntas a seguir sobre o grafo na figura correspondente com raiz a .

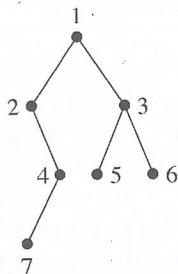
- Essa árvore é binária?
- É uma árvore binária cheia?
- É uma árvore binária completa?
- Qual nó é pai de e ?
- Qual nó é o filho direito de e ?
- Qual a profundidade do nó g ?
- Qual a altura da árvore?



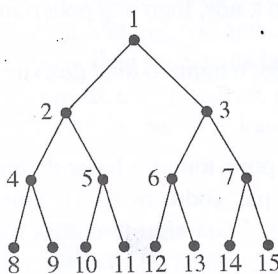
Nos Exercícios 3 a 6, desenhe a árvore que representa a expressão dada.

- $[(x - 2) * 3] + (5 + 4)$
- $[(2 * x - 3 * y) + 4 * z] + 1$
- $1 - (2 - [3 - (4 - 5)])$
- $[(6 \div 2) * 4] + [(1 + x) * (5 + 3)]$

- ★7. Escreva a representação em tabela, com filhos esquerdo e direito, para a árvore binária na figura correspondente.



8. Escreva a representação em tabela, com filhos esquerdo e direito, para a árvore binária na figura correspondente.



- ★9. Desenhe a árvore binária representada em tabela, com filhos esquerdo e direito, na figura correspondente. (1 é a raiz.)

Filho esquerdo Filho direito

1	2	3
2	4	0
3	5	0
4	6	7
5	0	0
6	0	0
7	0	0

10. Desenhe a árvore binária representada em tabela, com filhos esquerdo e direito, na figura correspondente. (1 é a raiz.)

Filho esquerdo Filho direito

1	2	0
2	3	4
3	0	0
4	5	6
5	0	0
6	0	0

11. Escreva a representação em tabela, com filhos esquerdo e direito, para a árvore binária de busca que é criada ao se processar a seguinte lista de palavras: "Toda a Gália está dividida em três partes" (veja o Exercício 31 da Seção 4.1). Armazene, também, o nome de cada nó.

- ★12. A tabela a seguir representa uma árvore binária onde são dados o filho esquerdo e o pai de cada nó. Desenhe a árvore binária. (A raiz é 1.)

Filho esquerdo Pai

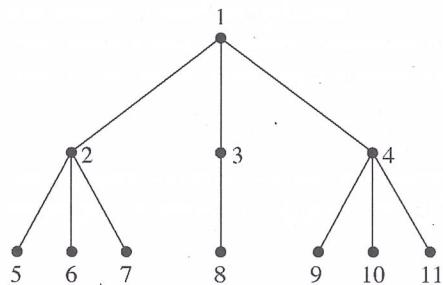
1	2	0
2	4	1
3	0	1
4	0	2
5	0	2
6	0	3

13. A tabela a seguir representa uma árvore (não necessariamente binária) onde, para cada nó, são dados o filho mais à esquerda e seu irmão à direita mais próximo. Desenhe a árvore. (A raiz é 1.)

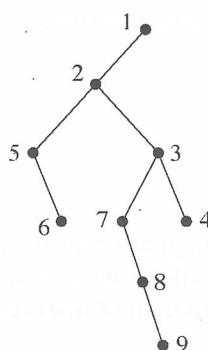
Filho esquerdo Irmão direito

1	2	0
2	5	3
3	0	4
4	8	0
5	0	6
6	0	7
7	0	0
8	0	0

14. a. Para a árvore a seguir, escreva a representação em tabela com o filho mais à esquerda e seu irmão mais próximo à direita, como descrito no Exercício 13.

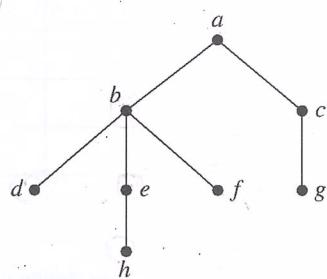


- b. Desenhe, agora, a árvore binária que resulta ao se tratar a resposta do item (a) como uma representação em tabela de uma árvore binária com os filhos esquerdo e direito. Dessa forma, pode-se obter uma representação em árvore binária de uma árvore arbitrária.
 15. A árvore binária a seguir é uma representação de uma árvore genérica (como no item (b) do Exercício 14). Desenhe a árvore.

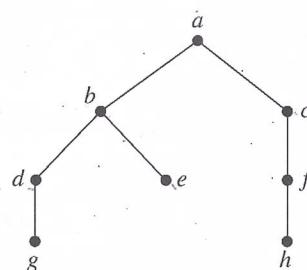


Para os Exercícios 16 a 21, escreva as listas de nós que resultam de um percurso em pré-ordem, um percurso em ordem simétrica e um percurso em pós-ordem na árvore.

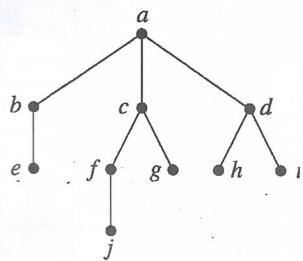
★16.



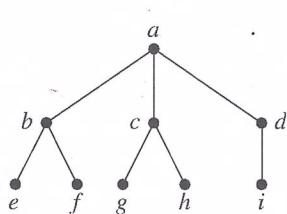
17.



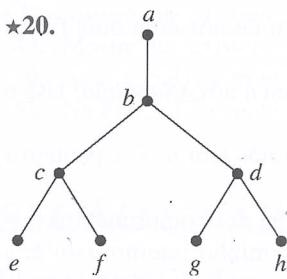
18.



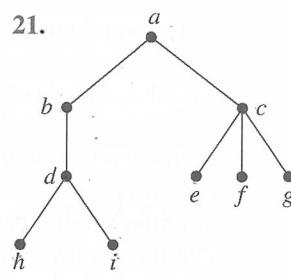
19.



★20.



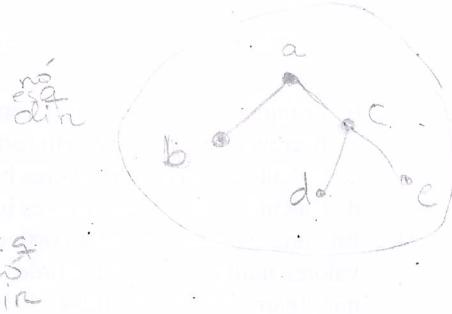
21.



- ★22. Escreva nas notações polonesa e polonesa reversa: $3/4 + (2 - y)$.
 23. Escreva nas notações polonesa e polonesa reversa: $(x * y + 3/z) * 4$.
 24. Escreva nas notações infixa e polonesa reversa: $- * + 2 3 * 6 x 7$.
 25. Escreva nas notações infixa e polonesa reversa: $- + - x y z w$.
 ★26. Escreva nas notações polonesa e infixa: $4 7 x - * z +$.
 27. Escreva nas notações polonesa e infixa: $x 2 w + y z * - /$.
 28. Desenhe uma árvore cujo percurso em pré-ordem é

a, b, c, d, e

e cujo percurso em ordem simétrica é

b, a, d, c, e

29. Desenhe uma árvore cujo percurso em ordem simétrica é

f, a, g, b, h, d, i, c, j, e

e cujo percurso em pós-ordem é

f, g, a, h, i, d, j, e, c, b

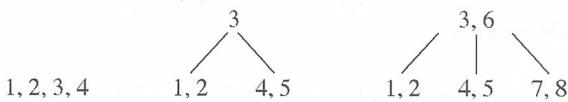
- ★30. Encontre um exemplo de uma árvore cujos percursos em ordem simétrica e em pós-ordem fornecem a mesma lista de nós.
 31. Encontre duas árvores diferentes que têm a mesma lista de nós se percorridas em pré-ordem.
 32. Descreva, informalmente, um algoritmo recorrente para calcular a altura de uma árvore binária sendo dado o nó raiz.
 33. Descreva, informalmente, um algoritmo recorrente para calcular o número de nós em uma árvore binária sendo dado o nó raiz.
 ★34. Prove que um grafo simples é uma árvore sem raiz se, e somente se, existe um único caminho entre dois nós quaisquer.
 35. Seja G um grafo simples. Prove que G é uma árvore sem raiz se, e somente se, G é conexo e a remoção de um único arco qualquer em G torna-o não conexo.
 36. Seja G um grafo simples. Prove que G é uma árvore sem raiz se, e somente se, G é conexo e a adição de um arco a G resulta em um grafo com exatamente um ciclo.
 37. Prove que uma árvore com n nós, $n \geq 2$, tem pelo menos dois nós de grau 1.
 ★38. Prove que uma árvore binária tem no máximo 2^d nós com profundidade d .
 39. a. Desenhe uma árvore binária cheia de altura 2. Quantos nós ela tem?
 b. Desenhe uma árvore binária cheia de altura 3. Quantos nós ela tem?
 c. Faça uma conjectura sobre a quantidade de nós em uma árvore binária cheia de altura h .
 d. Prove sua conjectura. (Sugestão: Use o Exercício 38.)
 40. a. Prove que uma árvore binária cheia com x nós internos tem $2x + 1$ nós ao todo.
 b. Prove que uma árvore binária cheia com x nós internos tem $x + 1$ folhas.
 c. Prove que uma árvore binária cheia com n nós tem $(n - 1)/2$ nós internos e $(n + 1)/2$ folhas.

41. Prove que o número de folhas em qualquer árvore binária é o número de nós com dois filhos mais 1.
- ★42. Encontre uma expressão para a altura de uma árvore binária completa com n nós. (Sugestão: Use o Exercício 39.)
43. Prove que a representação com ponteiros de uma árvore binária com n nós tem $n + 1$ ponteiros nulos. (Sugestão: Use o Exercício 41.)
44. Seja E o comprimento do caminho externo de uma árvore, isto é, a soma dos comprimentos dos caminhos da raiz a cada uma das folhas. Seja I o comprimento do caminho interno; isto é, a soma dos comprimentos dos caminhos da raiz a cada um dos nós internos. Seja i o número de nós internos. Prove que, em uma árvore binária onde todos os nós internos têm dois filhos, $E = I + 2i$.
45. Suponha que $B(n)$ representa o número de árvores binárias diferentes com n nós.
- Defina $B(0)$ como tendo o valor 1 (existe uma árvore binária com 0 nós). Prove que $B(n)$ é dada pela relação de recorrência

$$B(1) = 1$$

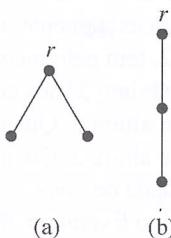
$$B(n) = \sum_{k=0}^{n-1} B(k)B(n-1-k)$$

- Compare a seqüência $B(n)$ com a seqüência dos números de Catalan (Exercício 82, Seção 3.4). Escreva uma expressão em forma fechada para $B(n)$.
 - Calcule o número de árvores binárias diferentes com 3 nós. Desenhe todas essas árvores.
 - Calcule o número de árvores binárias diferentes com 6 nós.
46. Em uma estrutura de dados conhecida como uma árvore B de ordem 5, cada nó da árvore pode conter valores múltiplos de dados ordenados. Entre e em torno dos valores em um nó interno estão arcos que ligam o nó a seus filhos. Novos valores são inseridos nas folhas da árvore, mas quando uma folha (ou nó interno) atinge 5 valores, ele se divide em dois e a mediana³ dos valores sobe para o próximo nível da árvore. A figura a seguir mostra a árvore à medida que os valores de 1 a 8 são inseridos em uma árvore inicialmente vazia.



- O número mínimo de valores inseridos em uma árvore B de ordem 5 de modo a forçá-la a ter dois níveis é 5. Encontre o número mínimo de valores necessários para que a árvore tenha três níveis.
 - Prove que, para $n \geq 2$, quando uma árvore B de ordem 5 tem o número mínimo de valores que a obriga a ter n níveis, o nível de baixo contém $2 \cdot 3^{n-2}$ nós.
 - Encontre uma expressão geral (e justifique) para o número mínimo de valores necessários para que uma árvore B de ordem 5 tenha n níveis. (Sugestão: $3^0 + 3^1 + \dots + 3^{n-2} = (3^n - 3)/6$.)
47. Encontre o número cromático de uma árvore (veja o Exercício 71 da Seção 5.1).

Nos Exercícios 48 e 49, duas árvores são *isomórfas* se existe uma bijeção $f: N_1 \rightarrow N_2$ tal que f leva a raiz de uma árvore na raiz da outra e $f(y)$ é filho de $f(x)$ na segunda árvore sempre que y é filho de x na primeira. Assim, na figura apresentada, as duas árvores são grafos isomórfos mas não árvores isomórficas (na Fig. (a) a raiz tem dois filhos e na Fig. (b) não). Essas são as duas únicas árvores não-isomórficas com três nós.



³A mediana de um conjunto finito de valores é o valor central desse conjunto. (N.T.)

- ★48. Mostre que existem quatro árvores não-isomórficas com quatro nós.
 49. Mostre que existem nove árvores não-isomórficas com cinco nós.
 50. Na Fig. 5.37, suponha que o nó 4 é a raiz e redesenhe a árvore com a raiz no topo.
 51. Na Fig. 5.37, suponha que o nó 2 é a raiz e redesenhe a árvore com a raiz no topo.

SEÇÃO 5.3 ÁRVORES DE DECISÃO

Usamos árvores de decisão no Cap. 3 para resolver problemas de contagem. A Fig. 5.51 mostra a árvore usada no Exemplo 39 do Cap. 3 para representar as diversas possibilidades de cinco jogadas de moedas com a restrição de que duas caras consecutivas não podem ocorrer. Cada nó interno da árvore representa uma ação (uma jogada da moeda) e os arcos que o une aos filhos representam os resultados da ação (cara (C) ou coroa (K)). As folhas da árvore representam os resultados finais, isto é, os diversos resultados diferentes que podem ocorrer depois de cinco jogadas.

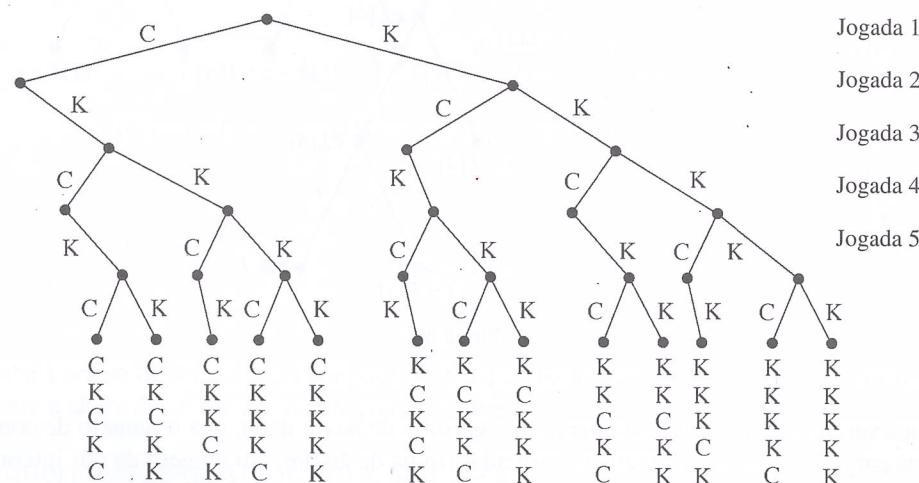


Fig. 5.51

Embora tenhamos usado árvores de decisão, não definimos formalmente o que é uma árvore de decisão.

Definição: Árvore de Decisão Uma árvore de decisão é uma árvore na qual os nós internos representam ações, os arcos representam os resultados de uma ação e as folhas representam resultados finais.

Algumas vezes pode-se obter informações úteis usando-se uma árvore de decisão para representar as atividades de um algoritmo; ações executadas pelo algoritmo estão representadas nos nós internos, os filhos de um nó interno representam a próxima ação a ser executada baseada no resultado da ação anterior e as folhas representam alguma coisa que pode ser inferida ao final do algoritmo. Note que, ao contrário das árvores discutidas na Seção 5.2, uma árvore de decisão não é uma estrutura de dados, isto é, os nós da árvore não têm valores de dados associados. Nem os algoritmos que estamos representando precisam estar agindo necessariamente em uma estrutura de árvore. De fato, usaremos árvores de decisão nesta seção para aprender mais sobre algoritmos de busca e de ordenação, e esses algoritmos agem em uma lista de dados.

ALGORITMOS DE BUSCA

Um algoritmo de busca encontra um elemento desejado x em uma lista de elementos ou verifica que x não pertence à lista. Esse algoritmo, em geral, compara x sucessivamente com os elementos na lista. Já vimos dois desses algoritmos, busca seqüencial e busca binária. Podemos modelar as atividades desses algoritmos usando árvores de decisão. Os nós representam as ações de comparar x com os elementos na lista, onde a comparação entre x e o i -ésimo elemento na lista é denotado por $x:L(i)$.

O algoritmo de busca seqüencial tem apenas dois resultados possíveis para uma comparação entre x e $L(i)$. Se $x = L(i)$, o algoritmo termina, já que x foi encontrado na lista. Se $x \neq L(i)$, a próxima comparação a ser feita é $x:L(i + 1)$, independente de se x era maior ou menor do que $L(i)$. As folhas dessa árvore de decisão correspondem aos resultados finais, onde x é um dos elementos na lista ou x não pertence à lista.

A Fig. 5.52 mostra a árvore de decisão para o algoritmo de busca sequencial agindo em uma lista ordenada com cinco elementos.

❖ EXEMPLO 31

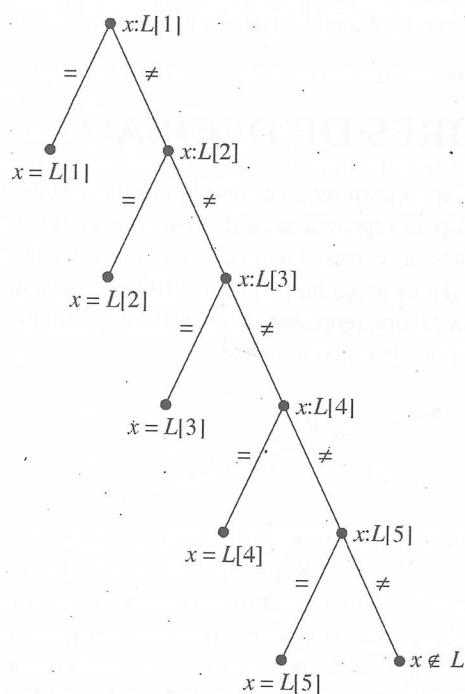


Fig. 5.52

Podemos ver, na árvore de decisão para um algoritmo de busca dado, que o número de comparações necessárias para se chegar a um resultado particular (folha da árvore) é o número de nós internos da raiz àquela folha. Esse número é igual ao comprimento do caminho da raiz até a folha. O caso pior, isto é, o que tem o número maior de comparações, é o comprimento máximo desses caminhos, que é a altura da árvore.

Como toda árvore de decisão para uma busca sequencial parece com a da Fig. 5.52, é claro que a altura de uma tal árvore, para uma lista com n elementos, é n . Isso está de acordo com o que já sabemos, isto é, que o caso pior para um algoritmo de busca sequencial em uma lista de n elementos é

- A árvore de decisão para o algoritmo de busca binária é mais interessante. A busca binária age em uma lista ordenada e tem três resultados possíveis para cada comparação:

- $x = L(i)$: o algoritmo termina, x foi encontrado
 - $x < L(i)$: o algoritmo vai para a metade esquerda da lista
 - $x > L(i)$: o algoritmo vai para a metade direita da lista

Seguiremos o costume e não colocaremos a folha que corresponde ao “ramo do meio”, $x = L(i)$ (veja o Exercício 21 para uma discussão das consequências dessa convenção). Se $x < L(i)$, a próxima comparação do algoritmo está no filho esquerdo desse nó; se $x > L(i)$, a próxima comparação do algoritmo está no filho direito. Se não existe filho, o algoritmo termina porque x não pertence à lista. A árvore que descrevemos é uma árvore binária cujas folhas representam todos os resultados possíveis quando x não pertence à lista. Existem muito mais folhas representando os casos em que x não pertence à lista em uma busca binária do que em uma busca seqüencial, pois a busca binária indica *como* x não está na lista (por exemplo, $x < L(1)$ ou $L(1) < x \leq L(2)$).

A Fig. 5.53 mostra a árvore de decisão para o algoritmo de busca binária agindo em uma lista com oito elementos. O caso pior, isto é, o que tem o número máximo de comparações, novamente vai ser a altura da árvore, que é 4 na Fig. 5.53. Resolvemos, no Cap. 2, uma relação de recorrência para obter o comportamento do pior caso para a busca binária, com n uma potência de 2, e encontramos $1 + \log n$ (lembre-se de que estamos usando logaritmo em base 2). Note que $1 + \log 8 = 4$, logo o resultado encontrado com a árvore de decisão coincide com nosso resultado anterior. A restrição de que n seja uma potência de 2 torna mais simples a resolução da relação de recorrência. Se n não for uma potência de 2, então a altura da árvore é dada pela expressão $1 + \lfloor \log n \rfloor$.

❖ EXEMPLO 32

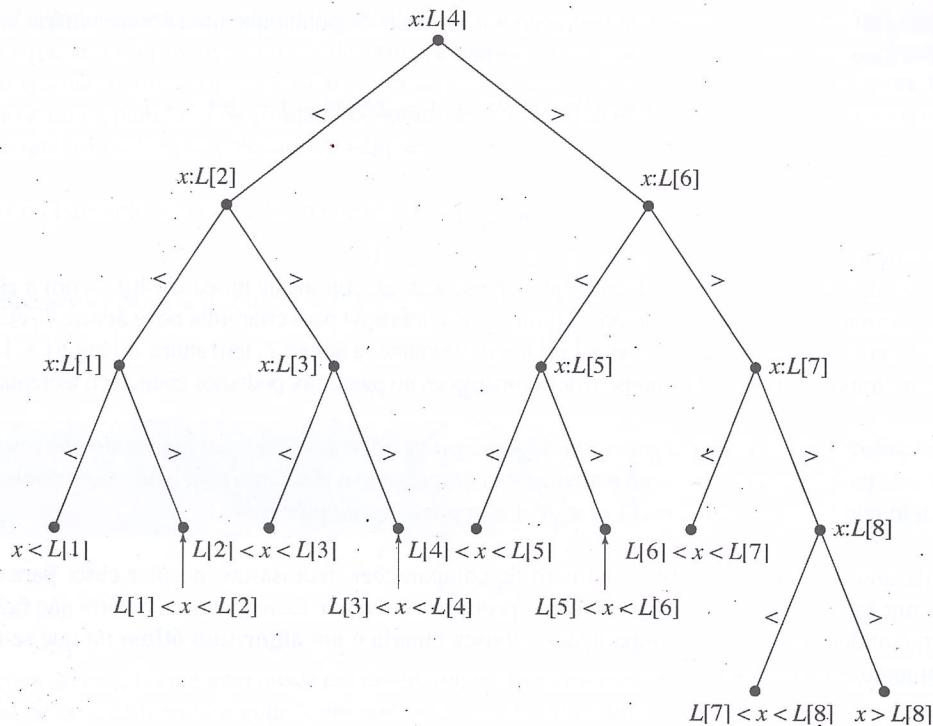


Fig. 5.53

PROBLEMA PRÁTICO 24

- a Desenhe a árvore de decisão para o algoritmo de busca binária em um conjunto com cinco elementos.
 b. Encontre a altura da árvore e compare com $1 + \lfloor \log 5 \rfloor$.

Cotas Inferiores para Algoritmos de Busca

Usamos árvores de decisão para representar as ações de dois algoritmos particulares de busca. Uma tal árvore poderia ser usada para representar as ações de qualquer algoritmo que resolva o problema de busca comparando o elemento desejado com os elementos na lista. Os nós internos de uma tal árvore representariam as comparações feitas e a altura da árvore daria o número de comparações no caso pior entre todos os casos possíveis. O que se pode dizer sobre uma tal árvore quando não se conhece detalhes do algoritmo envolvido? Podemos dizer que x precisa ser comparado a todos os elementos na lista pelo menos uma vez (talvez mais de uma vez, se o algoritmo for burro). De fato, se x não for comparado a algum elemento na lista, o algoritmo não pode dizer se aquele elemento é igual a x e, portanto, não pode decidir com certeza se x pertence à lista. Comparações correspondem aos nós internos na árvore de decisão. Logo, se m é o número de nós internos em uma árvore de decisão T , para um algoritmo de busca qualquer em uma lista com n elementos, então $m \geq n$.

Antes de continuar nosso estudo de árvores de decisão, precisamos de alguns fatos adicionais sobre árvores binárias em geral. O número de nós em cada nível em uma árvore binária cheia segue uma progressão geométrica: 1 nó no nível 0, 2^1 nós no nível 1, 2^2 nós no nível 2, e assim por diante. Em uma árvore binária cheia de altura d , o número total de nós, portanto, é:

$$1 + 2 + 2^2 + 2^3 + \dots + 2^d = 2^{d+1} - 1$$

(veja o Exemplo 15 do Cap. 2). Uma árvore binária cheia tem o número máximo de nós para uma dada altura de qualquer árvore binária. Isso nos dá o fato 1:

1. Qualquer árvore binária de altura d tem, no máximo, $2^{d+1} - 1$ nós.

O fato 2, que provaremos a seguir, é:

2. Qualquer árvore binária com m nós tem altura $\geq \lfloor \log m \rfloor$.

Para provar o fato 2, faremos uma demonstração por absurdo. Suponha que uma árvore binária tem m nós e altura $d < \lfloor \log m \rfloor$. Então $d \leq \lfloor \log m \rfloor - 1$. Do fato 1,

$$\begin{aligned} m \leq 2^{d+1} - 1 &\leq 2^{\lfloor \log m \rfloor - 1 + 1} - 1 = 2^{\lfloor \log m \rfloor} - 1 \leq 2^{\log m} - 1 \\ &= m - 1 \end{aligned}$$

ou

$$m \leq m - 1$$

uma contradição. Portanto, $d \geq \lfloor \log m \rfloor$.

Vamos agora voltar às árvores de decisão que representam algoritmos de busca em listas com n elementos. Retire, temporariamente, as folhas da árvore T_1 (com m nós internos) para criar uma nova árvore T_2 com m nós, $m \geq n$. Pelo fato 2, T_2 tem altura $d \geq \lfloor \log m \rfloor \geq \lfloor \log n \rfloor$. Portanto, a árvore T_1 tem altura $\geq \lfloor \log n \rfloor + 1$. Como a altura de uma árvore de decisão dá o número de comparações no pior caso, podemos enunciar o teorema a seguir:

Teorema sobre a Cota Inferior para um Algoritmo de Busca Qualquer algoritmo que resolva um problema de busca em uma lista com n elementos comparando o elemento desejado x com os elementos na lista tem que fazer, pelo menos, $\lfloor \log n \rfloor + 1$ comparações no pior caso.

Isso nos dá uma cota inferior sobre o número de comparações necessárias, no pior caso, para qualquer algoritmo que faça comparações para resolver o problema de busca. Como a busca binária não faz mais do que essa quantidade mínima de comparações, a busca binária é um **algoritmo ótimo** no que se refere ao comportamento no pior caso.

Árvore de Busca Binária

O algoritmo de busca binária precisa que os dados já estejam ordenados. Dados arbitrários podem ser organizados em uma estrutura chamada de **árvore binária de busca**, que pode, então, ser pesquisada usando-se um algoritmo diferente, chamado de **busca em árvore binária**. Para construir uma árvore binária de busca, o primeiro dado é a raiz da árvore. Dados sucessivos são colocados comparando-os com os nós já existentes, a começar pela raiz. Se um dado é menor do que um nó, o próximo nó a ser testado é o filho esquerdo; caso contrário, é o filho direito. Quando o nó não tem filho, o novo dado torna-se um filho.

Os dados

$$5, 8, 2, 12, 10, 14, 9$$

vão ser organizados em uma árvore binária de busca. A Fig. 5.54 mostra os estados sucessivos na construção dessa árvore.

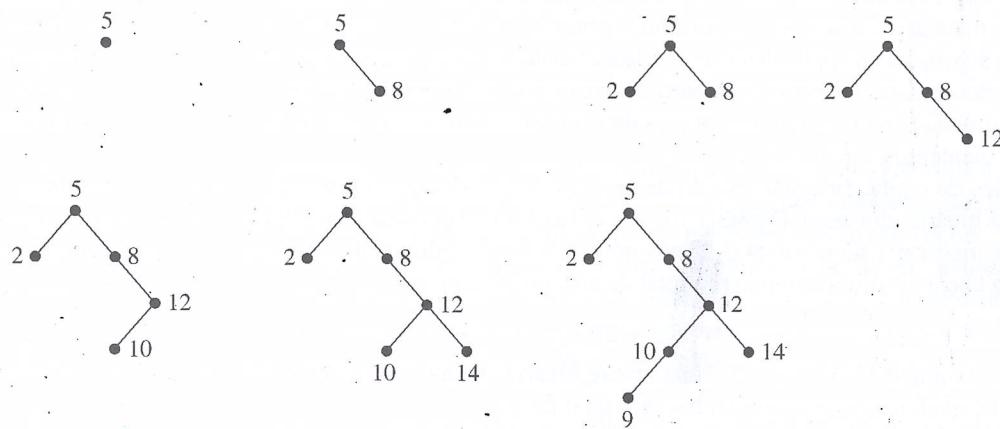


Fig. 5.54

EXEMPLO 33

Uma **árvore binária de busca**, pela própria construção, tem a propriedade de que o valor em cada nó é maior do que todos os valores em sua subárvore esquerda (a subárvore enraizada em seu filho esquerdo) e menor do que todos os valores em sua subárvore direita. Uma **busca em árvore binária** compara o item x com uma sucessão de nós, começando pela raiz. Se x é igual ao valor do nó, o algoritmo termina; se x é menor, compara-se a seguir com o filho esquerdo; se x é maior, compara-se a seguir com o filho direito. Se o nó não tem filhos, o algoritmo termina porque x não pertence à lista. Assim, a árvore binária de busca,

exceto pelas folhas, torna-se a árvore de decisão para o algoritmo de busca em árvore binária. (Temos aqui um caso em que o próprio algoritmo é descrito em termos de uma árvore.) O número de comparações no pior caso é igual à altura da árvore mais 1 (pelas folhas que faltam). No entanto, uma árvore binária de busca não é única para um determinado conjunto de dados; a árvore (e, portanto, sua altura) depende da ordem na qual os dados são colocados na árvore.

♦ EXEMPLO 34 Os dados no Exemplo 32 colocados na ordem

9, 12, 10, 5, 8, 2, 14

produzem a árvore binária de busca na Fig. 5.55.

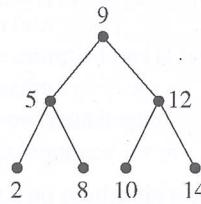


Fig. 5.55

As ações executadas em uma busca em árvore binária certamente são semelhantes às executadas por um algoritmo de busca binária “regular”; em ambos os casos, faz-se uma comparação e, se os elementos não forem iguais, segue-se para a esquerda ou para a direita (na árvore, se for uma busca em árvore binária, ou na lista, se for uma busca binária). É possível ordenar os dados para uma busca em árvore binária de modo que a árvore de busca construída por esses dados coincida com a árvore de decisão (menos as folhas) para uma busca binária dos mesmos dados ordenados. Isso está ilustrado no Exemplo 34 (note que a árvore não foi construída a partir dos dados ordenados). A árvore binária de busca nesse caso tem altura mínima e necessita da quantidade mínima de trabalho no pior caso.

A altura de uma árvore binária de busca para um determinado conjunto de dados pode variar. A altura da árvore na Fig. 5.54 é 4, enquanto a da Fig. 5.55 é 2. Assim, o número de comparações no pior caso para se procurar por um elemento também pode variar. O processo de construção da árvore pode ser modificado para tornar a árvore mais “balanceada”, isto é, baixa e larga, ao invés de alta e fina; uma tal modificação reduz a altura da árvore e, portanto, o tempo de busca. No entanto, como sabemos pelo teorema sobre a cota inferior para um algoritmo de busca, é necessária uma quantidade mínima determinada de trabalho, não importa o quanto esperto for a nossa construção da árvore.

PROBLEMA PRÁTICO 25

- a. Construa a árvore binária de busca para os dados do Exemplo 33 colocados na ordem

12, 9, 14, 5, 10, 8, 2

- b. Qual a altura da árvore?

ALGORITMOS DE ORDENAÇÃO

Árvores de decisão também podem modelar algoritmos que ordenam uma lista de itens através de uma seqüência de comparações entre dois itens da lista. Os nós internos de uma tal árvore de decisão são rotulados $L[i]:L[j]$ para indicar a comparação do item i da lista com o item j . Para simplificar nossa discussão, vamos supor que a lista não contém itens duplicados. Então o resultado de uma tal comparação é $L[i] < L[j]$ ou $L[i] > L[j]$. Se $L[i] < L[j]$, o algoritmo prossegue para a comparação indicada pelo filho esquerdo desse nó; se $L[i] > L[j]$, o algoritmo vai para o filho direito. Se o nó não tem filho, o algoritmo termina, pois a ordenação terminou. A árvore é uma árvore binária e as folhas representam os resultados finais, isto é, as diversas ordenações.

♦ EXEMPLO 35

A Fig. 5.56 mostra a árvore de decisão para um algoritmo de ordenação agindo em uma lista com três elementos. Esse algoritmo não é particularmente esperto, já que ignora a propriedade transitiva de $<$ e, portanto, faz algumas comparações desnecessárias. As folhas da árvore indicam os diversos resultados, incluindo dois casos (marcados com X) que resultam de informações contraditórias. Por exemplo, um X resulta da seguinte seqüência inconsistente de resultados: $L[1] < L[2]$, $L[2] < L[3]$, $L[1] > L[3]$.

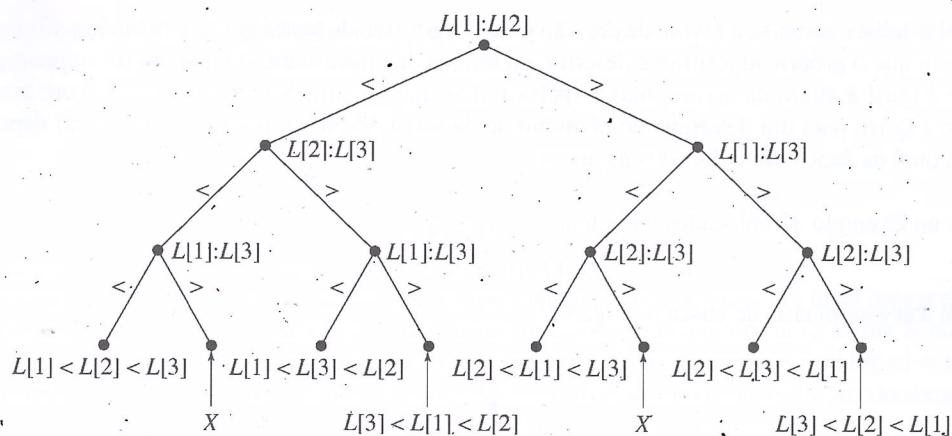


Fig. 5.56

Desenhe a árvore de decisão que resultaria se o algoritmo do Exemplo 35 fosse modificado para eliminar comparações desnecessárias.

PROBLEMA
PRÁTICO 26

Um argumento baseado em uma árvore de decisão também pode ser usado para se estabelecer uma cota inferior para o número de comparações necessárias, no pior caso, para ordenar uma lista de n elementos. Como fizemos para os algoritmos de busca, vamos ver o que podemos dizer sobre uma árvore de decisão para ordenar através de comparações, independente do algoritmo que representa. As folhas de uma tal árvore representam os resultados finais, isto é, os vários arranjos ordenados dos n elementos. Existem $n!$ desses arranjos, de modo que, se p é o número de folhas, $p \geq n!$. O pior caso é igual à altura da árvore. Mas também é verdade que, se a árvore tem altura d , então $p \leq 2^d$ (Exercício 38 da Seção 5.2). Tomando o logaritmo em base 2 nessa equação, obtemos $\log p \leq d$, ou, como d é inteiro, $d = \lceil \log p \rceil$. Finalmente, temos

$$d = \lceil \log p \rceil \geq \lceil \log n! \rceil$$

Isso prova o teorema a seguir:

Teorema sobre Cotas Inferiores para Algoritmos de Ordenação Qualquer algoritmo que ordena uma lista de n elementos comparando pares de elementos na lista tem que fazer, pelo menos, $\lceil \log n! \rceil$ comparações no pior caso.

Pode-se mostrar (Exercício 23 ao final desta seção) que $\log n! = \Theta(n \log n)$. Provamos, portanto, que o número de comparações para ordenar n elementos comparando pares de itens da lista é limitado inferiormente por $\Theta(n \log n)$, ao passo que o número de comparações em uma busca que compara o elemento desejado com os itens na lista é limitado inferiormente por $\Theta(\log n)$. Como esperado, dá mais trabalho ordenar do que procurar.

SEÇÃO 5.3 REVISÃO

TÉCNICAS

- W ❖ Desenho de árvores de decisão para buscas seqüencial e binária em uma lista com n elementos.
- W ❖ Criação de uma árvore binária de busca.

IDÉIAS PRINCIPAIS

- ❖ Árvores de decisão representam ações possíveis para determinados algoritmos.
- ❖ A análise de uma árvore de decisão genérica para algoritmos que resolvem certo tipo de problema pode nos levar a cotas inferiores para a quantidade mínima de trabalho necessário para resolver o problema no pior caso.
- ❖ A tarefa de procurar um valor x em uma lista com n elementos, se feita comparando-se x com os elementos na lista, necessita de pelo menos $\lceil \log n \rceil + 1$ comparações no pior caso.
- ❖ A tarefa de ordenar uma lista com n elementos, se feita comparando-se pares de elementos na lista, necessita de pelo menos $\lceil \log n! \rceil$ comparações no pior caso.

EXERCÍCIOS 5.3

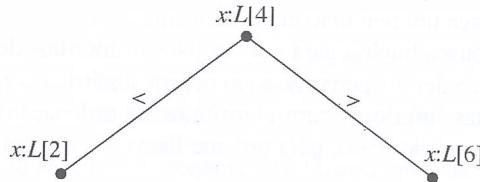
- ★1. Desenhe a árvore de decisão para a busca seqüencial em uma lista com três elementos.
2. Desenhe a árvore de decisão para a busca seqüencial em uma lista com seis elementos.
3. Desenhe a árvore de decisão para a busca binária em uma lista ordenada com sete elementos. Qual a altura da árvore?
4. Desenhe a árvore de decisão para a busca binária em uma lista ordenada com quatro elementos. Qual a altura da árvore?
- ★5. Considere um algoritmo de busca que compara um item com o último elemento de uma lista, depois com o primeiro elemento, depois com o penúltimo, depois com o segundo, e assim por diante. Desenhe a árvore de decisão para a busca em uma lista ordenada com seis elementos. Qual a altura da árvore? Esse parece ser um algoritmo ótimo no pior caso?
6. Considere um algoritmo de busca que compara um item com um elemento na lista que está no final do primeiro terço da lista; baseado nessa comparação, ele procura no primeiro terço ou nos dois últimos terços da lista. Desenhe a árvore de decisão para a busca em uma lista ordenada com nove elementos. Qual a altura da árvore? Esse parece ser um algoritmo ótimo no pior caso?
- ★7. a. A partir dos dados
- $$\begin{array}{c} 1 \cdot 2 \cdot 3 \cdot 2 \cdot 4 \cdot \\ 9, 5, 6, 2, 4, 7 \end{array}$$
- construa a árvore binária de busca. Qual a altura da árvore?
- b. Encontre o número médio de comparações feitas para se procurar um item, que se sabe que está na lista, usando busca em árvore binária na árvore do item (a). (Sugestão: Encontre o número de comparações para cada um dos elementos.)
8. a. A partir dos dados
- $$g, d, r, s, b, q, c, m$$
- construa a árvore binária de busca. Qual a altura da árvore?
- b. Encontre o número médio de comparações feitas para se procurar um item, que se sabe que está na lista, usando busca em árvore binária na árvore do item (a). (Sugestão: Encontre o número de comparações para cada um dos elementos.)
9. a. Para um conjunto de seis dados, qual o número mínimo de comparações, no pior caso, que um algoritmo de busca tem que fazer?
- b. Para o conjunto de dados $\{a, d, g, i, k, s\}$, encontre uma ordem na qual se coloque os dados para que a árvore binária de busca correspondente tenha altura mínima.
10. a. Para um conjunto de nove dados, qual o número mínimo de comparações, no pior caso, que um algoritmo de busca tem que fazer?
- b. Para o conjunto de dados $\{4, 7, 8, 10, 12, 15, 18, 19, 21\}$, encontre uma ordem na qual se coloque os dados para que a árvore binária de busca correspondente tenha altura mínima.
- ★11. Um percurso em ordem simétrica em uma árvore binária de busca produz uma lista dos três nós em ordem alfabética ou numérica. Construa uma árvore binária para a frase “To be or not to be, that is the question”⁴, e depois faça um percurso em ordem simétrica.
12. Construa uma árvore de busca binária para a frase “Nós primórdios dos tempos, o Elefante, Ó Meu Amor, não tinha tromba”, e depois percorra-a em ordem simétrica (veja o Exercício 11).
- ★13. Use o teorema sobre cotas inferiores para algoritmos de ordenação para encontrar o número de comparações necessárias, no pior caso, para ordenar listas dos seguintes tamanhos:
- a. 4 b. 8 c. 16
14. Compare o número de comparações necessárias no pior caso para os algoritmos de ordenação por seleção e ordenação por fusão com as cotas inferiores encontradas no Exercício 13 (veja o Exercício 17 na Seção 2.5). Quais são suas conclusões?

Os Exercícios 15 a 20 tratam do problema de identificação de uma moeda falsa (pesada demais ou leve demais) em um conjunto de n moedas. Uma balança de pratos é utilizada para colocar um grupo qualquer de moedas do conjunto em um dos pratos e um número equivalente de moedas no outro prato. O resultado de uma tal pesagem pode ser que o grupo A pesa menos do que o grupo B,

⁴“Ser ou não ser, essa é a questão.” Citação da peça *Hamlet*, de Shakespeare. (N.T.)

ou ambos têm o mesmo peso, ou A pesa mais do que B. Uma árvore de decisão representando a seqüência de comparações feitas será então uma *árvore ternária*, onde um nó interno pode ter três filhos.

15. Uma entre cinco moedas é falsa e mais leve do que as outras quatro. O problema é identificar a moeda falsa.
 - a. Qual o número de resultados finais (o número de folhas na árvore de decisão)?
 - b. Encontre uma cota inferior para o número de comparações necessárias para resolver esse problema no pior caso.
 - c. Pense em algum algoritmo que use essa cota inferior (desenhe sua árvore de decisão).
- ★16. Uma entre cinco moedas é falsa e é pesada demais ou leve demais. O problema é identificar a moeda falsa e determinar se ela é mais pesada ou mais leve do que as outras.
 - a. Qual o número de resultados finais (o número de folhas na árvore de decisão)?
 - b. Encontre uma cota inferior para o número de comparações necessárias para resolver esse problema no pior caso.
 - c. Pense em algum algoritmo que use essa cota inferior (desenhe sua árvore de decisão).
17. Uma entre quatro moedas é falsa e é pesada demais ou leve demais. O problema é identificar a moeda falsa mas não determinar se ela é mais pesada ou mais leve do que as outras.
 - a. Qual o número de resultados finais (o número de folhas na árvore de decisão)?
 - b. Encontre uma cota inferior para o número de comparações necessárias para resolver esse problema no pior caso.
 - c. Pense em algum algoritmo que use essa cota inferior (desenhe sua árvore de decisão).
18. Uma entre quatro moedas é falsa e é pesada demais ou leve demais. O problema é identificar a moeda falsa e determinar se ela é mais pesada ou mais leve do que as outras.
 - a. Qual o número de resultados finais (o número de folhas na árvore de decisão)?
 - b. Encontre uma cota inferior para o número de comparações necessárias para resolver esse problema no pior caso.
 - c. Prove que não existe nenhum algoritmo que tenha essa cota inferior. (*Sugestão:* A primeira comparação é feita com duas ou quatro moedas. Considere cada um desses casos.)
19. Projete um algoritmo para resolver o problema no Exercício 18 que faz três comparações no pior caso..
20. Uma entre oito moedas é falsa e é pesada demais ou leve demais. O problema é identificar a moeda falsa e determinar se ela é mais pesada ou mais leve do que as outras.
 - a. Qual o número de resultados finais (o número de folhas na árvore de decisão)?
 - b. Encontre uma cota inferior para o número de comparações necessárias para resolver esse problema no pior caso.
 - c. Pense em algum algoritmo que use essa cota inferior (desenhe sua árvore de decisão).
- ★21. Na árvore de decisão para o algoritmo de busca binária (e para o algoritmo de busca em árvore binária), contamos cada nó interno como uma comparação. Por exemplo, o topo da Fig. 5.53 é



Para se obter um dos filhos da raiz, supusemos que uma comparação foi feita. No entanto, o resultado da comparação em cada nó interno corresponde, de fato, a três ramificações possíveis:

$$\begin{aligned} x &= \text{elemento no nó} \\ x &< \text{elemento no nó} \\ x &> \text{elemento no nó} \end{aligned}$$

Pense em como implementar essas três ramificações na maioria das linguagens de programação e escreva uma expressão mais precisa do que $1 + \lfloor \log n \rfloor$ para o número de comparações no pior caso.

22. Nossa algoritmo de busca binária (Exemplo 40 no Cap. 2) contém a instrução em pseudocódigo

encontre o índice k do item do meio na lista $L[i] - L[j]$

e, após isso, o elemento desejado x é comparado com o item na lista na posição do índice k , o “item do meio”. Suponha que essa instrução seja substituída por

```

se  $i = j$  então
   $k = j$ 
senão
   $k = i + 1$ 
fim do se

```

- Desenhe a árvore de decisão que resulta da utilização do algoritmo modificado em uma lista ordenada com $n = 8$.

- Dê o número exato de comparações necessárias (veja o Exercício 21) no pior caso para $n = 8$.
- Dê uma expressão para a ordem de grandeza do número de comparações necessárias no pior caso em função de n e justifique sua expressão. Compare esse algoritmo com o de busca binária original, que é $\Theta(\log n)$.

23. Para provar que $\log n! = \Theta(n \log n)$, podemos usar a definição de ordem de grandeza (veja a Seção 4.4 do Cap. 4) e mostrar que existem constantes positivas n_0, c_1 e c_2 tais que, se $n \geq n_0$, então $c_1(n \log n) \leq \log n! \leq c_2(n \log n)$.
- Mostre que, para $n \geq 1$, $\log n! \leq n \log n$. (*Sugestão:* Use a definição de $n!$ e propriedades dos logaritmos.)
 - Mostre que, para $n \geq 4$, $\log n! \geq (1/4)(n \log n)$. (*Sugestão:* Use a definição de $n!$ e propriedades dos logaritmos, mas pare em $\log \lceil n/2 \rceil$.)

SEÇÃO 5.4 CÓDIGOS DE HUFFMAN

PROBLEMA E SOLUÇÃO TENTATIVA

Caracteres consistem em letras do alfabeto (maiúsculas e minúsculas), símbolos de pontuação e outros símbolos de teclado, como @ e %. Os computadores armazenam os caracteres em forma binária, como uma seqüência de 0 e 1. A abordagem usual é fixar um comprimento n de modo que 2^n seja tão grande quanto o número de caracteres distintos e codificar cada caractere distinto como uma seqüência particular de n bits⁵. Cada caractere tem que ser codificado em sua seqüência binária fixa e depois decodificado quando for mostrado. O sistema de codificação mais comum é ASCII (*American Standard Code for Information Interchange*⁶), que usa $n = 8$, de modo que cada caractere usa 8 bits de armazenagem. Mas, qualquer que seja o valor de n , cada caractere usa a mesma quantidade de espaço de armazenamento.

Suponha que uma coleção de caracteres a ser armazenada em um arquivo em forma binária é suficientemente grande para que a quantidade de espaço de armazenamento seja uma preocupação. Suponha, também, que o arquivo tenha uma natureza relativamente permanente, que seu conteúdo não será modificado com freqüência. Pode valer a pena, então, gastar um certo esforço extra no processo de codificação para que se reduza a quantidade de espaço de armazenamento necessário para o arquivo.

Ao invés de se usar um número fixo de bits por caractere, um esquema de codificação poderia usar um número variável de bits e armazenar caracteres que apareçam freqüentemente como seqüências com menos bits. Para que se armazene todos os caracteres distintos, algumas seqüências ainda terão que ser longas mas, se as seqüências mais longas forem usadas para caracteres que ocorrem com menos freqüência, a quantidade total de espaço deverá ser reduzida. Essa abordagem necessita de conhecimento sobre o conteúdo do arquivo em questão, razão pela qual funciona melhor para arquivos cujo conteúdo não será modificado com freqüência. Estudaremos aqui um tal esquema de **compressão de dados** ou **compactificação de dados**, já que a melhor maneira de descrevê-lo é como uma série de ações executadas em árvores binárias.

EXEMPLO 36

Como um exemplo trivial, suponha que uma coleção de dados contém 50.000 vezes os seis caracteres a, c, g, k, p e ?, que ocorrem com as seguintes freqüências percentuais:

Caractere	a	c	g	k	p	?
Freqüência	48	9	12	4	17	10

⁵Do inglês *binary digits*, dígitos binários. (N.T.)

⁶Código Americano Padrão para Troca de Informação. (N.T.)

Como seis caracteres distintos têm que ser armazenados, o esquema de comprimento fixo necessaria de um mínimo de três bits por caractere ($2^3 = 8 \geq 6$). O espaço total necessário seria então de $50.000 * 3 = 150.000$ bits. Suponha agora que, ao invés desse esquema, usássemos o seguinte esquema de codificação:

Caractere	<i>a</i>	<i>c</i>	<i>g</i>	<i>k</i>	<i>p</i>	?
Esquema de codificação	0	1101	101	1100	111	100

O espaço necessário (número de bits) seria, então,

$$50.000(0,48 * 1 + 0,09 * 4 - 0,12 * 3 + 0,04 * 4 + 0,17 * 3 + 0,10 * 3) = 108.500$$

que é, aproximadamente, dois terços do espaço anterior.

- No esquema de armazenagem de comprimento fixo com n bits para cada caractere, a cadeia de bits longa do arquivo codificado pode ser quebrada para se ler o código dos caracteres sucessivos olhando-se, simplesmente, n bits de cada vez. Isso facilita a decodificação do arquivo. No código com comprimento variável, é preciso ter uma maneira de saber quando termina a sequência de um símbolo e começa a do seguinte.

Usando o código de comprimento variável dado no Exemplo 36, decodifique cada uma das cadeias a seguir:

- 1111111010100
- 1101010101100
- 100110001101100

No Problema Prático 27, as cadeias podem ser quebradas na representação dos diversos caracteres de uma única maneira. Ao se analisar cada dígito novo, vão se diminuindo as possibilidades de qual caractere está sendo representado até que ele seja identificado pelo final de sua representação. Nunca existe a necessidade de se conjecturar qual deveria ser o caractere e depois voltar atrás se nossa conjectura mostrou-se errada. Essa possibilidade de se decodificar de maneira única, sem falsos começos e voltas para trás, é devida ao fato de o código ser um exemplo de um código de prefixo. Em um código de prefixo, o código para qualquer caractere nunca é o prefixo do código de qualquer outro caractere. (Um código de prefixo é, portanto, um código “antiprefixo”!)

Considere o código

Caractere	<i>a</i>	<i>b</i>	<i>c</i>
Esquema de codificação	01	101	011

que não é um código de prefixo. A cadeia 01101 poderia representar *ab* (01–101) ou *ca* (011–01). Além disso, ao se processar 011011 dígito por dígito, como um computador faria, a decodificação começaria com *ab* (01–101) e só encontraria o erro no último dígito. O processo teria, então, que voltar até o primeiro dígito para poder reconhecer *cc* (011–011).

Em nosso estudo de códigos de prefixo, vamos construir árvores binárias tendo os caracteres como folhas. Uma vez construída a árvore, um código binário pode ser atribuído a cada caractere simplesmente percorrendo-se o caminho da raiz até a folha correspondente, usando 0 para a ramificação da esquerda e 1 para a da direita. Como nenhuma folha precede outra em algum caminho começando na raiz, o código será um código de prefixo. A Fig. 5.57 ilustra a árvore binária para o código do Exemplo 36.

Suponha que existe uma árvore T para um código, com as folhas representando caracteres. Para qualquer folha i , sua profundidade $d(i)$ em T é igual ao número de bits no código para o símbolo correspondente. Denote por $f(i)$ a freqüência percentual daquele símbolo no texto a ser armazenado e seja S o número total de caracteres no texto. Então, como no Exemplo 36, o número total de bits necessário é dado pela expressão

$$\text{LIV}: S * \left[\sum_{\text{todas as folhas } i} (d(i)f(i)) \right]$$

Queremos construir uma árvore ótima T , para a qual a expressão

$$E(T) = \sum_{\text{todas as folhas } i} (d(i)f(i)) \quad (1)$$

seja mínima e, portanto, o tamanho do arquivo seja mínimo.

PROBLEMA PRÁTICO 27

EXEMPLO 37

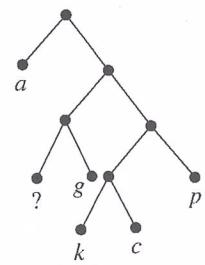


Fig. 5.57

Esse processo poderia ser feito por tentativa e erro, já que existe apenas um número finito de caracteres e, portanto, um número finito de maneiras de se construir uma árvore atribuindo-se caracteres a suas folhas. No entanto, esse número finito se torna, rapidamente, muito grande! Ao invés disso, usaremos o algoritmo conhecido como **código de Huffman**.

ALGORITMO DE CODIFICAÇÃO DE HUFFMAN

Suponha então que temos m caracteres em um arquivo e que conhecemos a freqüência percentual de cada um deles. O algoritmo para se construir a árvore funciona mantendo-se uma lista L de nós que são raízes de árvores binárias. Inicialmente, L contém m raízes, cada uma rotulada com a freqüência de um dos caracteres; as raízes serão ordenadas pela freqüência em ordem crescente e nenhuma delas terá filho.

Segue uma descrição do algoritmo em pseudocódigo.

```

ALGORITMO ÁRVOREDEHUFFMAN
ÁrvoreDeHuffman(lista de nós L; inteiro m)
// A cada um dos  $m$  nós em  $L$  está associada uma freqüência  $f$ , e  $L$ 
// é ordenada pela freqüência em ordem crescente; o algoritmo constrói a
// árvore de Huffman.
    para  $i = 1$  até  $m - 1$  faça
        crie novo nó  $z$ 
        sejam x, y os dois primeiros nós em L // nós de
        //freqüência mínima
         $f(z) = f(x) + f(y)$ 
        insira  $z$  em ordem em  $L$ 
        filho esquerdo de  $z$  = nó  $x$ 
        filho direito de  $z$  = nó  $y$            //x, y não pertencem mais a L
    fim do para
fim da ÁrvoreDeHuffman

```

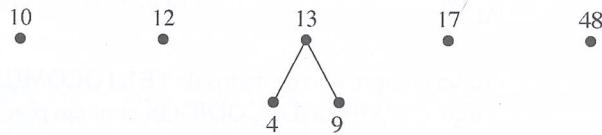
Quando esse algoritmo termina, L tem apenas um nó, que é a raiz da árvore binária final. Podemos atribuir códigos, então, a cada folha da árvore percorrendo o caminho da raiz até a folha e acumulando 0 para os ramos da esquerda e 1 para os da direita. Pela construção de árvore, cada nó interno terá exatamente dois filhos.

EXEMPLO 38

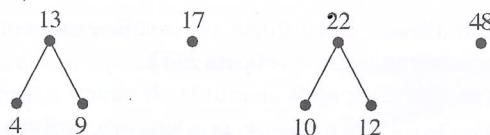
Vamos usar o algoritmo *ÁrvoreDeHuffman* para construir a árvore na Fig. 5.57, que é baseada nos dados do Exemplo 36. L inicialmente contém seis nós, ordenados por freqüência:

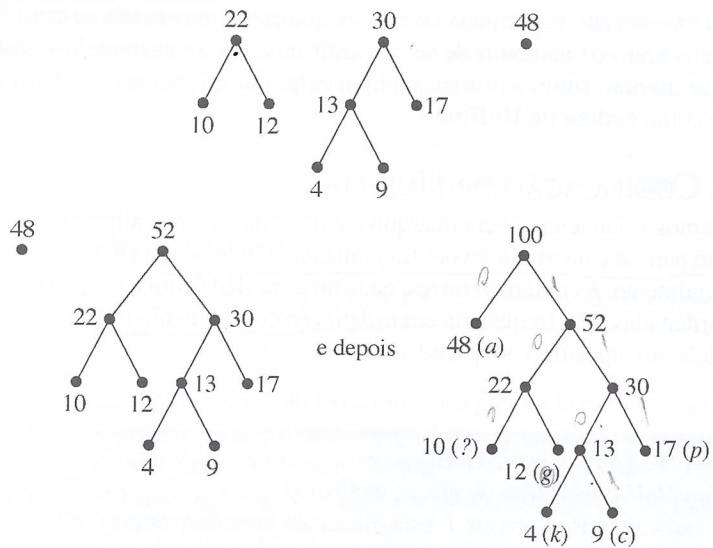
4 9 10 12 17 48

Seguindo o algoritmo, entramos no laço do **para** pela primeira vez. Os nós x e y são os com freqüência 4 e 9, respectivamente. Um novo nó com freqüência $4 + 9 = 13$ é criado e inserido, em ordem, na lista L , com o nó x como seu filho esquerdo e o nó y como seu filho direito. A nova lista L fica assim:



Esse processo é repetido mais quatro vezes. As listas novas L resultantes em cada etapa são:





Nesse ponto a árvore está completa e os códigos podem ser atribuídos. O código para c , por exemplo, é 1101 (ramos da direita, direita, esquerda, direita).

Construa a árvore de Huffman para os seguintes caracteres com suas freqüências:

Caractere	w	q	h	e
Freqüência	10	12	20	58

PROBLEMA
PRÁTICO 28

Encontre os códigos de Huffman para os caracteres do Problema Prático 28.

PROBLEMA
PRÁTICO 29

A Tabela 5.1 mostra as etapas necessárias para se usar a codificação/decodificação de Huffman para compressão de dados.

TABELA 5.1

Codificação Etapa 1	No arquivo original TEXTOCOMUM, faça uma análise de freqüência, isso é, crie um arquivo FREQÜÊNCIA que contém dados da forma $a-18$ $b-7$ e assim por diante.
Codificação Etapa 2	Usando TEXTOCOMUM e FREQÜÊNCIA, crie um arquivo TABELADECÓDIGOS que contém o código de Huffman para cada caractere, por exemplo, $a-001$ $b-1110$
Codificação Etapa 3	Usando TEXTOCOMUM e TABELADECÓDIGOS, crie um arquivo chamado CODIFICADO que contém os dados comprimidos.
Decodificação	Usando CODIFICADO e TABELADECÓDIGOS, decodifique o texto para recuperar TEXTOCOMUM.

O arquivo CODIFICADO é a versão com compressão de dados de TEXTOCOMUM e, presume-se, necessita de menos espaço. No entanto, o arquivo TABELADECÓDIGOS também precisa ser armazenado para se poder decodificar o arquivo.

JUSTIFICATIVA

Embora o algoritmo para construir a árvore de Huffman T seja suficientemente fácil de ser descrito, precisamos mostrar que ela nos dá o menor valor possível para $E(T)$ ⁷.

⁷Comprimento do caminho externo de uma árvore. Veja o Exercício 44 na Seção 5.2. (N.T.)

Primeiro, se tivermos uma árvore ótima T para m caracteres, pode-se supor que os nós de menor freqüência são os filhos esquerdo e direito de algum nó. Para provar isso, chame os nós de menor freqüência de x e y . Se x e y não são irmãos na árvore, encontre dois irmãos p e q no último nível da árvore e considere o caso em que x e y não pertencem a esse nível (veja a Fig. 5.58a). Como $f(x) \leq f(p)$, sabemos que $f(x) \leq f(p)$. Se $f(x) < f(p)$, então permutando os lugares de x e p na árvore resultaria em uma árvore nova T' com $E(T') < E(T)$ (Fig. 5.58b): a freqüência maior está agora a uma profundidade menor — veja o Exercício 16a), o que contradiz o fato de que T era ótima. Portanto, $f(x) = f(p)$ e x e p podem ser permutados na árvore sem mudança em $E(T)$. Analogamente, y e q podem ser permutados, resultando na Fig. 5.58c, na qual x e y são irmãos. Se x e y estão no mesmo nível que p e q no início, então eles certamente podem ser permutados com p ou q sem afetar $E(T)$ (Fig. 5.58d).

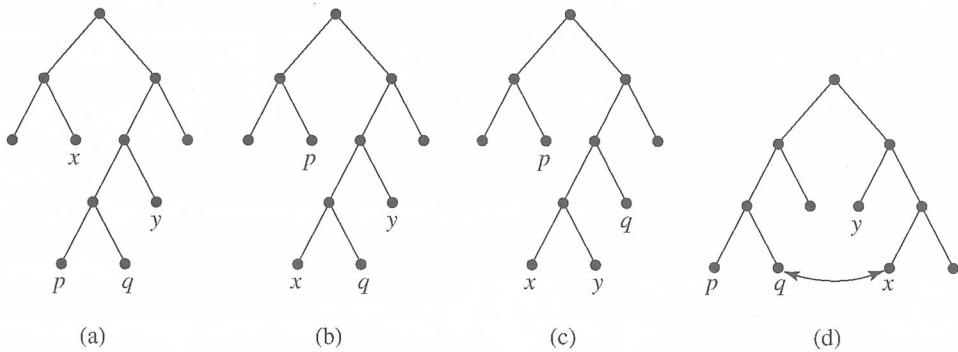


Fig. 5.58

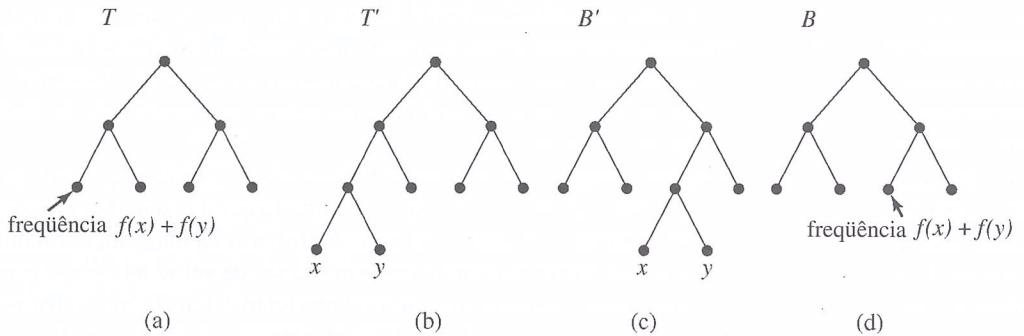


Fig. 5.59

Suponha, novamente, que $f(x)$ e $f(y)$ são as freqüências mínimas e suponha que temos uma árvore T que é ótima para outras freqüências junto com a soma $f(x) + f(y)$ (Fig. 5.59a). Essa soma será a freqüência de uma folha; crie uma árvore T' que tem esse nó como um nó interno com filhos x e y , tendo freqüências $f(x)$ e $f(y)$ (Fig. 5.59b). T' será ótima para as freqüências $f(x)$, $f(y)$ e as outras. A demonstração desse fato começa com alguma árvore ótima B' para as freqüências $f(x)$, $f(y)$ e as outras. Sabemos que existe uma tal árvore ótima (já que ela poderia ser encontrada por tentativa e erro) e, do parágrafo precedente, podemos supor que x e y são irmãos em B' (Fig. 5.59c). Crie agora uma árvore B retirando os nós x e y de B' e dando-se a freqüência $f(x) + f(y)$ ao nó pai, que agora é uma folha (Fig. 5.59d). Como T é ótimo para as outras freqüências junto com $f(x) + f(y)$, temos

$$E(T) \leq E(B) \quad (1)$$

Mas a diferença entre $E(B)$ e $E(B')$ é um arco para cada, x e y , isso é, $E(B') = E(B) + f(x) + f(y)$ (veja o Exercício 16b). Analogamente, temos $E(T') = E(T) + f(x) + f(y)$. Logo, se somarmos $f(x) + f(y)$ à equação (1), obteremos

$$E(T') \leq E(B') \quad (2)$$

Como B' era ótima, não podemos ter $E(T') < E(B')$, logo $E(T') = E(B')$ e T' é ótima.

Finalmente, uma árvore com um único nó cuja freqüência é a soma de todas as freqüências é trivialmente ótima para essa soma. Podemos, repetidamente, ir separando as parcelas dessa soma e colocando filhos de tal maneira a terminar com a árvore de Huffman final. Pelo parágrafo anterior, cada uma dessas árvores, incluindo a árvore de Huffman final, é ótima.

◆ EXEMPLO 39

Se aplicarmos o processo que preserva otimização à árvore da Fig. 5.57, começariam com um único nó com freqüência 100 e “cresceríamos” a árvore para baixo, como ilustrado na Fig. 5.60.

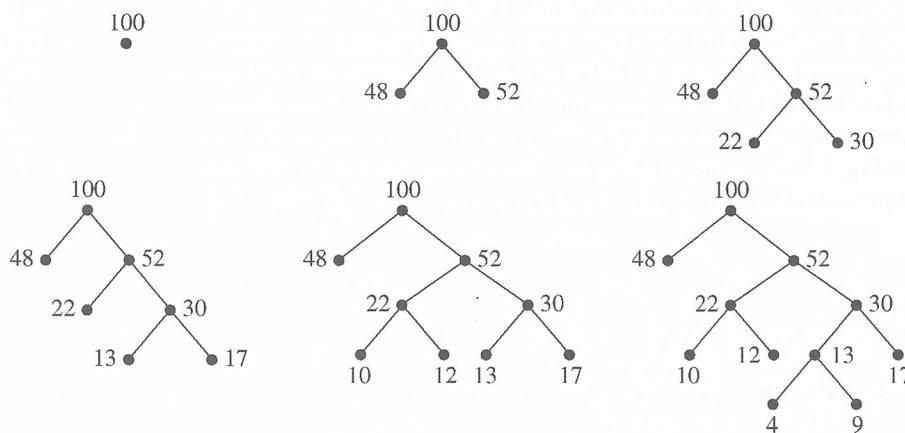


Fig. 5.60

APlicações de Códigos de Huffman

JPEG é um mecanismo padrão de compressão de imagens para imagens com qualidade fotográfica. JPEG é a abreviatura de *Joint Photographic Experts Group*, o nome do grupo que desenvolveu esse padrão internacional. A necessidade para a melhoria da compressão de imagens tornou-se maior devido ao desejo de se transmitir imagens pela Internet. Existem atualmente duas versões da codificação JPEG, uma com perda, outra sem perda, mas a versão com perda é muito mais comum. Um *esquema de compressão com perda* significa que uma vez decodificados os dados comprimidos eles não são idênticos aos originais — alguma informação foi “perdida”. No caso de compressão JPEG com perda, a perda de dados vem do pré-processamento da imagem antes da aplicação do código de Huffman; a codificação/decodificação de Huffman restaura fielmente os dados com os quais começou.

A compressão JPEG foi projetada para imagens a serem vistas por seres humanos e se aproveita do fato de que o olho humano é muito mais sensível a gradientes de claro e escuro do que a pequenas variações em cor. O primeiro passo no processo JPEG, portanto, é obter a informação de cores na imagem, normalmente dada como 24 bits por pixel, 8 bits para cada uma das componentes primárias de vermelho, verde e azul, e transformar cada pixel em componentes que capturem a luminosidade (claro/escuro) com informações reduzidas sobre as componentes das cores. A seguir, pixels que têm informações de cores semelhantes são agrupados e é usado um valor “médio” de cores, enquanto os dados mais precisos sobre a luminância são mantidos. Os dados são então transformados em dados de freqüência, que passam, por sua vez, por um processo de “quantização” (basicamente arredondamento dos resultados de cálculos) para terminar em forma inteira. Variações de alta freqüência (para as quais o olho humano é menos sensível) são perdidas nesse processo, mas, mais uma vez, os dados sobre a luminância são tratados de modo mais preciso do que os dados de cor. O código de Huffman é aplicado ao resultado. Áreas da imagem cuja representação ocorrem freqüentemente serão codificados em cadeias menores de bits.

Um arquivo de imagem JPEG contém não só a imagem comprimida, mas também a informação necessária para reverter o processo de compressão (incluindo a informação para decodificar o código de Huffman). A imagem resultante perdeu mudanças de alta freqüência e variações de cores eliminadas nos estágios anteriores à aplicação do código de Huffman. Parâmetros no processo de codificação JPEG permitem escolhas entre a quantidade da compressão a ser obtida e a fidelidade da imagem recuperada em relação à imagem original. Devido à natureza dos algoritmos utilizados, a codificação JPEG tem pouco ou nenhum efeito sobre desenhos em branco e preto, onde não existem dados a serem desprezados.

Uma segunda aplicação do código de Huffman ocorre em alguns controles remotos para aparelhos de videocassete. Esses controles mais sofisticados simplificam a programação de um videocassete que, como todos sabem, só são compreendidos por crianças de onze anos. Para programar o vídeo, digita-se um código⁸, que consiste em um número de 1 a 8 dígitos, para o programa a ser gravado. Os códigos dos programas são dados nos jornais e revistas que trazem a programação; um número de código contém informação sobre a data, o horário de início, o tempo de duração e o canal de televisão onde vai ser mostrado o programa a ser gravado. O controle remoto controla o vídeo. Embora os detalhes do esquema de codificação estejam paten-

teados, é usado um código de Huffman para se produzir números de código mais curtos para os programas de televisão vistos com mais freqüência. A Tabela 5.2 mostra exemplos retirados de um guia para a televisão para quarta-feira, 10 de outubro de 2001, em uma cidade do Meio-Oeste dos Estados Unidos.

TABELA 5.2

Nome	Canal	Código
Felicidade	WB	9923
Ala Oeste	NBC	39107
Cheers	Nickelodeon	490381
Tarântulas e seus Parentes Peçonhentos	Animal Planet	9789316

SEÇÃO 5.4 REVISÃO

TÉCNICAS

- ❖ Dado um conjunto de caracteres e suas freqüências, encontrar os códigos de Huffman.

IDÉIA PRINCIPAL

- ❖ Dada a freqüência dos caracteres em uma coleção de dados, pode-se encontrar um esquema de codificação binária que minimiza o número de bits necessários para armazenar os dados mas ainda tem uma decodificação fácil.

EXERCÍCIOS 5.4

- ★1. Dados os códigos

Caractere	<i>a</i>	<i>e</i>	<i>i</i>	<i>o</i>	<i>u</i>
Esquema de codificação	00	01	10	110	111

decodifique as seqüências

- a. 11011011101 b. 1000110111 c. 010101
2. Dados os códigos

Caractere	<i>b</i>	<i>h</i>	<i>q</i>	<i>w</i>	<i>%</i>
Esquema de codificação	1000	1001	0	11	101

decodifique as seqüências

3. Dados os códigos

Caractere	<i>a</i>	<i>p</i>	<i>w</i>	(.)
Esquema de codificação	001	1010	110	1111 1110

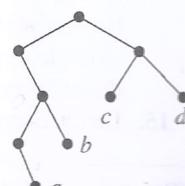
decodifique as seqüências

4. Dados os códigos, que não são de prefixo

Caractere	1	3	5	7	9
Esquema de codificação	1	111	101	10	1010

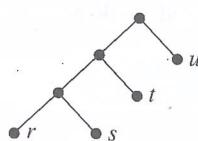
dê todas as decodificações possíveis para a seqüência 111110101.

5. Escreva os códigos de Huffman para a , b , c e d na árvore binária ilustrada.



⁸Esses códigos são utilizados nos Estados Unidos. (N.T.)

6. Escreva os códigos de Huffman para r , s , t e u na árvore binária da figura.



- ★7. a. Construa a árvore de Huffman para os caracteres a seguir com as freqüências dadas:

Caractere	c	d	g	m	r	z
Freqüência	28	25	6	20	3	18

- b. Encontre os códigos de Huffman para esses caracteres.

8. a. Construa a árvore de Huffman para os caracteres a seguir com as freqüências dadas:

Caractere	b	n	p	s	w
Freqüência	6	32	21	14	27

- b. Encontre os códigos de Huffman para esses caracteres.

9. a. Construa a árvore de Huffman para os caracteres a seguir com as freqüências dadas:

Caractere	a	z	t	e	b
Freqüência	27	12	15	31	15

- b. Encontre os códigos de Huffman para esses caracteres.

Nos Exercícios 10 e 11, os inteiros poderiam representar a etapa de “quantização” em uma compressão JPEG de uma imagem e o número de ocorrência de cada um na imagem. (Note que esses são exemplos de ocorrência e não de freqüências percentuais; isso significa simplesmente que a árvore de Huffman não vai terminar com o valor na raiz igual a 100.)

- ★10. Construa a árvore de Huffman e encontre os códigos de Huffman para o seguinte:

Inteiro	82	664	327	349	423	389
Ocorrências	416	97	212	509	446	74

11. Construa a árvore de Huffman e encontre os códigos de Huffman para o seguinte:

Inteiro	190	205	514	333	127	901	277
Ocorrências	52	723	129	233	451	820	85

12. Explique por que a codificação JPEG resulta em uma compressão menor para imagens contendo apenas tons de cinza do que para imagens coloridas.

- ★13. Alguém fez uma substituição global, em todo o texto do Exercício 9, substituindo todos os lugares onde tinha a letra “z” por “ch”; encontre os códigos de Huffman novos.

14. Considere o seguinte parágrafo:

No entanto, em meu íntimo, não podia deixar de me maravilhar com a intrepidez desses mortais diminutos, que ousavam subir e caminhar em cima de meu corpo enquanto uma de minhas mãos estava livre, sem tremer com a visão de uma criatura tão imensa como devo parecer a eles.⁹

Se esse parágrafo vai ser comprimido usando-se um código de Huffman, que caractere, diferente de símbolo de pontuação ou letras maiúsculas, deve ter um dos códigos mais longos? E qual deve ter um dos mais curtos?

15. Lembre-se do problema colocado no início deste capítulo:

⁹Das *Viagens de Gulliver*, de Johnathan Swift (Londres, 1726).

Você trabalha no Departamento de Sistemas de Informação da World Wide Widget (WWW), o líder mundial na produção de regenhocas. Os números das peças começam com uma letra, A, B, C, E ou P, que identifica o tipo, seguida de um número com 8 dígitos. Assim,

B00347289

A11872432

P45003781

são todos números legítimos de componentes. WWW mantém um arquivo de dados com os números das peças que usa, que são a maioria dos números em potencial.

Como comprimir esse arquivo de número de peças de modo a usar menos espaço de armazenamento do que os aproximadamente 4,5 Gb necessários para a codificação em ASCII de oito bits por caractere?

- Fazendo uma contagem de freqüência no arquivo da WWW, obtemos a seguinte informação:

Caractere	A	B	C	E	P	0	1	2	3	4	5	6	7	8	9
Freqüência	2	5	1	2	1	18	13	7	12	9	6	11	7	2	4

Construa um código de Huffman para esses caracteres.

- Calcule o espaço de armazenamento necessário para o arquivo comprimido como um percentual do arquivo não-comprimido.

★16. Na justificativa de que o algoritmo de Huffman produz uma árvore ótima, foram feitas as duas afirmações a seguir. Prove que cada uma delas é verdadeira.

- $E(T') < E(T)$
- $E(B') = E(B) + f(x) + f(y)$

REVISÃO DO CAPÍTULO 5

Terminologia

algoritmo ótimo (Seção 5.3)
altura de uma árvore (Seção 5.2)
arco (aresta) (Seção 5.1)
arcos paralelos (Seção 5.1)
árvore (Seção 5.2)
árvore binária (Seção 5.2)
árvore binária cheia (Seção 5.2)
árvore binária completa (Seção 5.2)
árvore binária de busca (Seção 5.3)
árvore de decisão (Seção 5.3)
árvore sem raiz (livre) (Seção 5.2)
busca em árvore binária (Seção 5.3)
caminho (Seção 5.1)
ciclo (Seção 5.1)
código de Huffman (Seção 5.4)
código de prefixo (Seção 5.4)
compressão de dados (compactação de dados) (Seção 5.4)
comprimento de um caminho (Seção 5.1)
diagrama de fluxo (Seção 5.1)
extremidades (Seção 5.1)
filho direito (Seção 5.2)
filho esquerdo (Seção 5.2)
filhos (Seção 5.2)
floresta (Seção 5.2)
folha (Seção 5.2)
fórmula de Euler (Seção 5.1)
grafo (Seção 5.1)

grafo acíclico (Seção 5.1)
grafo bipartido completo (Seção 5.1)
grafo com pesos (Seção 5.1)
grafo completo (Seção 5.1)
grafo conexo (Seção 5.1)
grafo direcionado (dígrafo) (Seção 5.1)
grafo planar (Seção 5.1)
grafo rotulado (Seção 5.1)
grafo sem laços (Seção 5.1)
grafo simples (Seção 5.1)
grafos homeomorfos (Seção 5.1)
grafos isomorfos (Seção 5.1)
grau de um nó (Seção 5.1)
isomorfismo (Seção 5.1)
laço (Seção 5.1)
lista de adjacência (Seção 5.1)
lista encadeada (Seção 5.1)
matriz de adjacência (Seção 5.1)
matriz esparsa (Seção 5.1)
nó (vértice) (Seção 5.1)
nó acessível (Seção 5.1)
nó interno (Seção 5.2)
nó isolado (Seção 5.1)
nós adjacentes (Seção 5.1)
notação infixa (Seção 5.2)
notação polonesa (Seção 5.2)
notação polonesa reversa (Seção 5.2)
notação posfixa (Seção 5.2)
notação prefixa (Seção 5.2)