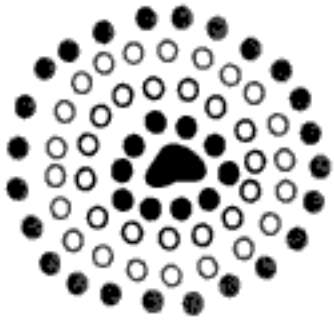
Estruturas de Dados e Algoritmos em JAVA

Capítulo

1 Conceitos Básicos de Programação Java



Conteúdo

R-1.10 Escreva uma pequena função em Java que receba um inteiro n e retorne a soma de todos os inteiros menores que n.

R-1.11 Escreva uma pequena função em Java que receba um inteiro n e retorne a soma de todos os inteiros pares menores que n.

Criatividade

C-1.1 Escreva uma pequena função Java que recebe um arranjo de valores **int** e determina se existe um par de números no arranjo cujo produto seja par.

C-1.2 Escreva um método Java que recebe um arranjo de valores **int** e determina se todos os números são diferentes entre si (isto é, se são valores distintos).

C-1.3 Escreva um método em Java que receba um arranjo contendo o conjunto de todos os inteiros no intervalo de 1 a 52 e embaralhe os mesmos de forma aleatória. O método deve exibir as possíveis seqüências com igual probabilidade.

C-1.4 Escreva um pequeno programa em Java que exiba todas as strings possíveis de serem formadas usando os caracteres `'c'`, `'a'`, `'r'`, `'b'`, `'o'` e `'n'` apenas uma vez.

C-1.5 Escreva um pequeno programa em Java que receba linhas de entrada pelo dispositivo de entrada padrão, e escreva as mesmas no dispositivo de saída padrão na ordem contrária. Isto é, cada linha é exibida na ordem correta, mas a ordem das linhas é invertida.

C-1.6 Escreva um pequeno programa em Java que receba dois arranjos a e b de tamanho n que armazenam valores **int** e retorne o produto escalar de a por b. Isto é, retorna um arranjo c de tamanho n onde $c[i] = a[i] \cdot b[i]$, para $i = 0,...,n-1$.

Projetos

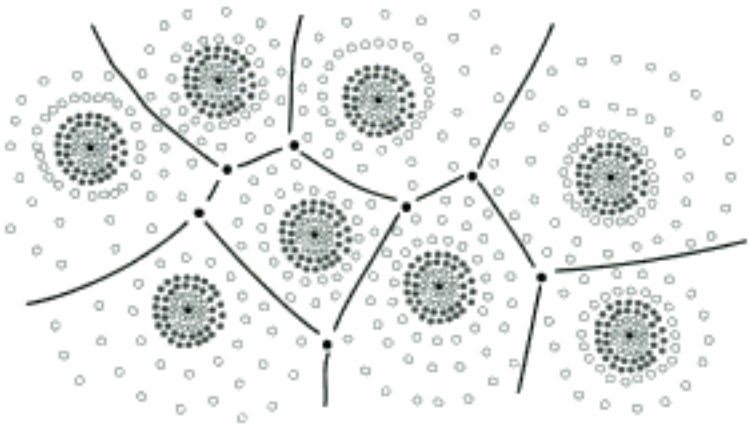
P-1.1 Uma punição comum para alunos de escola é escrever a mesma frase várias vezes. Escreva um programa executável em Java que escreva a mesma frase uma centena de vezes: "Eu não mandarei mais spam para meus amigos". Seu programa deve numerar as frases e "acidentalmente" fazer oito erros aleatórios diferentes de digitação.

P-1.2 (Para aqueles que conhecem os métodos de interface gráfica com o usuário em Java.) Defina uma classe GraphicalTest que teste a funcionalidade da classe CreditCard do Trecho de código 1.5, usando campos de entrada de texto e botões.

P-1.3 O ***paradoxo do aniversário*** diz que a probabilidade de duas pessoas em uma sala terem a mesma data de aniversário é maior que 50% desde que n, o número de pessoas na sala, seja maior que 23. Esta propriedade não é realmente um paradoxo, mas muitas pessoas se surpreendem. Projete um programa em Java que possa testar esse paradoxo por uma série de experimentos sobre aniversários gerados aleatoriamente, testando o paradoxo para $n = 5, 10, 15, 20, ..., 100$.

Capítulo

2 Projeto Orientado a Objetos



Conteúdo

R-2.12 Escreva um pequeno método Java que conte o número de vogais em uma string.

R-2.13 Escreva um pequeno método Java que remove toda a pontuação de um string *s* armazenando uma frase. Por exemplo, esta operação deve transformar a string `"Let's try, Mike"` em `"Lets try Mike"`.

R-2.14 Escreva um pequeno programa que recebe como entrada três inteiros, *a*, *b* e *c*, a partir do console Java e determina se eles podem ser usados em uma fórmula aritmética correta (na ordem em que foram fornecidos), como em "$a + b = c$", "$a = b - c$" ou "$a*b = c$".

R-2.15 Escreva um pequeno programa que cria uma classe Pair, que armazena dois objetos declarados como tipos genéricos. Faça um exemplo de uso deste programa criando e imprimindo pares de objetos Pair que contenham cinco tipos diferentes de pares, tais como <Integer,String> e <Float,Long>.

R-2.16 Parâmetros genéricos não são incluídos na assinatura da declaração de um método, de maneira que não se pode ter métodos diferentes na mesma classe que tenha diferentes parâmetros genéricos mas os mesmos nomes e tipos e quantidade de parâmetros. Como se pode alterar a assinatura dos métodos em conflito de maneira a contornar esse problema?

Criatividade

C-2.1 Explique por que o algoritmo de ativação dinâmica de Java que define o método que será ativado para uma determinada mensagem *o*.a( ) nunca irá entrar em laço infinito.

C-2.2 Escreva uma classe em Java que estende a classe Progression, criando uma progressão em que cada valor é o módulo da diferença entre os dois valores anteriores. Deve-se incluir um construtor que inicie com 2 e 200 como sendo os dois valores iniciais e um construtor parametrizado, que inicie com um par de valores informados como iniciais.

C-2.3 Escreva uma classe em Java que estende a classe Progression, criando uma progressão em que cada valor é a raiz quadrada do valor anterior. (Observe que não é mais possível representar os valores como inteiros.) Você deve incluir um construtor default que inicie com 65,536 como primeiro valor, e um construtor parametrizado que inicie com um valor informado (**double**) como primeiro valor.

C-2.4 Reescreva todas as classes da hierarquia da classe Progression de forma que todos os valores sejam da classe BigInteger, de maneira a evitar overflows.

C-2.5 Escreva um programa que consiste de três classes, *A*, *B* e *C*, tais que *B* estende *A* e *C* estende *B*. Cada classe deve definir uma variável de instância chamada "x" (isto é, cada uma tem sua própria variável chamada *x*). Descreva uma forma que permita a um método de *C* acessar e alterar a versão da variável *x* de *A* sem alterar as versões de *B* ou *C*.

C-2.6 Escreva um conjunto de classes Java que possam simular uma aplicação Internet onde um usuário, Alice, periodicamente cria pacotes que deseja enviar para Bob. Um processo da Internet esta sempre verificando se Alice tem algum pacote para enviar e, se tiver, despacha-os para o computador de Bob, e Bob está periodicamente verificando se o seu computador tem um pacote de Alice, e, se tiver, lê o mesmo e o deleta.

Projetos

P-2.1 Escreva um programa Java que recebe um documento e exibe um gráfico de barras com as freqüências de cada letra do alfabeto que aparece no documento.

P-2.2 Escreva um programa Java que simule uma calculadora portátil. O programa deve ser capaz de processar entradas tanto de uma GUI como do console Java, para acionar os botões e então exibir o conteúdo da tela após a execução de cada operação. No mínimo, a calculadora deve ser capaz de efetuar as operações aritméticas básicas e as operações zerar/limpar.

P-2.3 Complete o código da classe PersonPairDirectory do Trecho de código 2.14, assumindo que pares de pessoas são armazenadas em um arranjo com capacidade 1000. O diretório deve manter o registro de quantas pessoas estão registradas no momento.

P-2.4 Escreva um programa Java que recebe um inteiro positivo maior que 2 como entrada e exibe o número de vezes que alguém pode, repetidamente, dividir este número por 2 antes de obter um resultado menor que 2.

P-2.5 Escreva um programa Java que "faça troco". O programa deve receber dois números como entrada, um o valor pago e o outro o valor devido. Ele deve então retornar a quantidade de cada tipo de nota e moeda que deve ser devolvido como troco, devido a diferença entre o que foi pago e o que foi cobrado. Os valores das notas e moedas podem ser os do sistema monetário de qualquer governo. Tente projetar o programa de maneira que ele retorne a menor quantidade de notas e moedas possível.

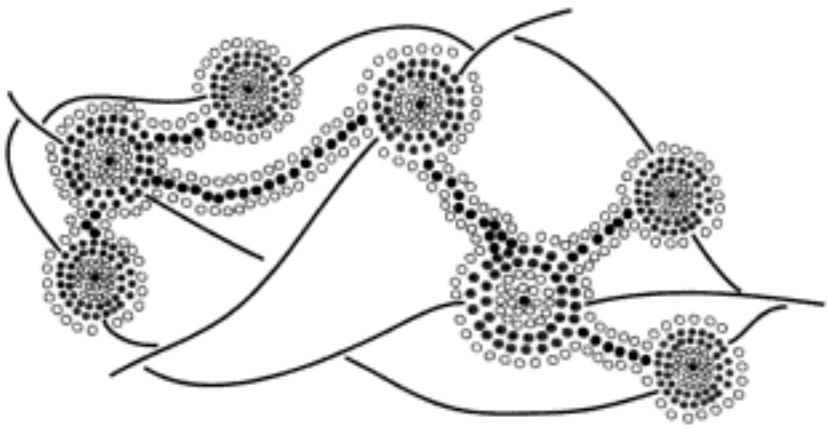
Observações sobre o capítulo

Para uma revisão abrangente dos desenvolvimentos recentes da ciência e da engenharia da computação, sugere-se *The Computer Science and Engineering Handbook* [92]. Para mais informações sobre o incidente com o Terac-25, ver o artigo de Leveson e Turner [66].

Para o leitor preocupado com estudos avançados em programação orientada a objetos, indicam-se os livros de Booch [14], Budd[17] e Liskov e Guttag[69]. Liskov e Guttag [69] também oferecem uma análise interessante sobre tipos abstratos de dados, da mesma forma que o artigo científico de Cardelli e Wegner [20], assim como o capítulo do livro de Demurjian[28] em *The Computer Science and Engineering Handbook* [92]. Padrões de projeto são descritos no livro de Gamma *et al* [38]. A notação dos diagramas de herança de classe que foi utilizada é derivada do livro de Gamma *et al*.

Capítulo

3 Arranjos, Listas Encadeadas e Recursão



Conteúdo

3.6 Exercícios

Para obter ajuda e o código-fonte dos exercícios, visite **Java.datastructures.net**.

Reforço

R-3.1 Os métodos add e remove do Trecho de código 3.3 e 3.4 não mantêm o número n de entradas não-nulas do arranjo a. Em vez disso, as células não usadas apontam para um objeto **null**. Mostre como alterar estes métodos de maneira que eles mantenham o tamanho atual de a em uma variável de instância n.

R-3.2 Descreva uma forma de usar recursão para acrescentar todos elementos de a em um arranjo $n \times n$ (bidimensional) de inteiros.

R-3.3 Explique como modificar o programa da Cifra de César (Trecho de código 3.9) de maneira que ele execute codificação e decodificação ROT13, que usa 13 como valor de deslocamento do alfabeto. Como você pode simplificar o código de maneira que o corpo do método de decodificação tenha apenas uma única linha?

R-3.4 Explique as alterações que devem ser feitas no programa do Trecho de código 3.9, de maneira que ele possa executar a Cifra de César para mensagens que são escritas em línguas baseadas em alfabetos diferentes do inglês, tais como grego, russo ou hebraico.

R-3.5 Qual a exceção que é lançada quando advance ou remove do Trecho de código 3.25 são chamados sobre uma lista vazia? Explique como modificar estes métodos de maneira a fornecer um nome de exceção mais instrutivo para esta condição.

R-3.6 Apresente uma definição recursiva para uma lista simplesmente encadeada.

R-3.7 Descreva um método para inserir um elemento no início de uma lista simplesmente encadeada. Assuma que a lista ***não*** usa um nodo sentinela e, em vez disso, usa a variável head para referenciar o primeiro nodo da lista.

R-3.8 Forneça um algoritmo para encontrar o penúltimo nodo em uma lista simplesmente encadeada onde o último elemento é indicado por uma referência next nula.

R-3.9 Descreva um método não-recursivo para encontrar, avançando pelos encadeamentos, o nodo do meio de uma lista duplamente encadeada com sentinelas de início e fim. (Observe: este método deve usar apenas navegação pelos encadeamentos; não deve usar um contador). Qual é o tempo de execução deste método?

R-3.10 Descreva um algoritmo recursivo para encontrar o maior elemento em um arranjo A de n elementos. Qual é o tempo de execução e a memória utilizada?

R-3.11 Desenhe o rastreamento recursivo da execução do método ReverseArray(A, 0, 4) (Trecho de código 3.32) sobre o arranjo $A = \{4, 3, 6, 2, 5\}$.

R-3.12 Desenhe o rastreamento recursivo para a execução do método PuzzleSolve(3, S, U) (Trecho de código 3.37) onde S está vazio e $U = \{a, b, c, d\}$.

R-3.13 Escreva um pequeno método Java que repetidamente seleciona e remove aleatoriamente uma entrada de um arranjo até que ele não armazene mais entradas.

R-3.14 Escreva um pequeno método Java para contar o número de nodos em uma lista encadeada circular.

Criatividade

C-3.1 Forneça o código Java dos métodos add(e) e remove(i) para entradas de jogos em um arranjo a, como nos Trechos de código 3.3 e 3.4, exceto que agora as entradas não são mantidas em ordem. Assuma que ainda é necessário manter n entradas armazenadas nos índices de 0 a $n - 1$. Tente implementar os métodos add e remove sem usar laços, de forma que o número de passos que eles executem não dependa de n.

C-3.2 Faça A ser um arranjo de tamanho $n \geq 2$ contendo inteiros de 1 a $n - 1$, inclusive, com exatamente um repetido. Descreva um algoritmo rápido para encontrar o inteiro de A que está repetido.

C-3.3 Seja B um arranjo de tamanho $n \geq 6$ contendo inteiros de 1 a $n - 5$, inclusive, com exatamente 5 repetidos. Descreva um bom algoritmo para encontrar os 5 inteiros de B que estão repetidos.

C-3.4 Suponha que você está projetando um jogo para vários jogadores que tem $n \geq 1000$ jogadores, numerados de 1 a n, interagindo em uma floresta encantada. O vencedor deste jogo é o primeiro jogador que puder encontrar todos os demais pelo menos uma vez (cordas são permitidas). Assumindo que existe um método meet(i, j) que é chamado cada vez que o jogador i encontra o jogador j (com $i \neq j$), descreva uma maneira de manter os pares de jogadores que se encontram e quem é o vencedor.

C-3.5 Apresente um algoritmo recursivo para calcular o produto de dois inteiros positivos, m e n, usando apenas adição e subtração.

C-3.6 Descreva um algoritmo recursivo rápido para inverter uma lista simplesmente encadeada L, de maneira que a ordem dos nodos seja o oposto do que era antes. Se a lista só tem uma posição, então não há nada a ser feito; a lista já está invertida. Em qualquer outro caso, remova.

C-3.7 Descreva um bom algoritmo para concatenar duas listas simplesmente encadeadas, L e M, com sentinelas cabeça, em uma única lista L' que contém todos os nodos de L seguidos por todos os nodos de M.

C-3.8 Forneça um algoritmo rápido para concatenar duas listas duplamente encadeadas L e M, com nodos sentinela cabeça e final, em uma única lista L'.

C-3.9 Descreva em detalhes como trocar dois nodos x e y de uma lista simplesmente encadeada L fornecidas apenas referências para x e y. Repita este exercício para o caso onde L é uma lista duplamente encadeada. Qual algoritmo consome mais tempo?

C-3.10 Descreva em detalhes um algoritmo para inverter uma lista simplesmente encadeada L usando apenas uma quantidade constante de espaço adicional e sem usar recursão.

C-3.11 No quebra-cabeça das ***Torres de Hanói***, existe uma plataforma com três pinos, a, b e c fincados na mesma. No pino a temos uma pilha de discos,

um maior que o outro, de maneira que o menor está no topo e o maior na base. O desafio é mover todos os discos do pino a para o pino c, movendo um disco de cada vez, de maneira que nunca seja colocado um disco maior sobre um disco menor. Veja a Figura 3.27 como exemplo do caso de $n = 4$. Descreva um algoritmo recursivo para resolver o quebra-cabeças das Torres de Hanói para qualquer valor de n. (Dica: considere primeiro o subproblema de mover todos menos o n-ésimo disco do pino a para outro pino usando o terceiro como "armazenamento temporário".)

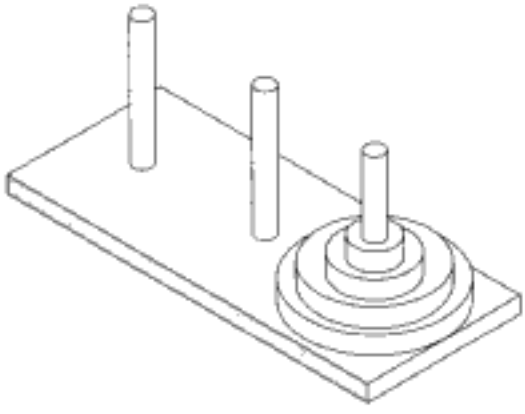


Figura 3.27 Desenho do quebra-cabeças das Torres de Hanói.

C-3.12 Descreva um método recursivo para converter um string de dígitos no inteiro que ele representa. Por exemplo, `"13531"` representa o inteiro 13.531.

C-3.13 Descreva um algoritmo recursivo que conte o número de nodos em uma lista simplesmente encadeada.

C-3.14 Escreva um programa Java recursivo que resultará em todos os suconjuntos de um conjunto de n elementos (sem repetir nenhum subconjunto).

C-3.15 Escreva um pequeno programa Java recursivo que encontre o menor e o maior valor de um arranjo de valores **int** sem usar nenhum laço.

C-3.16 Descreva um algoritmo recursivo que irá verificar se um arranjo A de inteiros contém um inteiro $A[i]$ que é a soma de dois inteiros que aparecem antes em A, isto é, tais que $A[i] = A[j]+A[k]$ para $j, k < i$.

C-3.17 Escreva um pequeno método Java recursivo que irá reorganizar um arranjo de valores **int** de maneira que todos os valores pares apareçam antes de todos os valores impares.

C-3.18 Escreva um pequeno método Java recursivo que pega um caracter string s e exibe seu inverso. Assim, por exemplo, o inverso de `"post&pans"` será `"snap&stop"`.

C-3.19 Escreva um pequeno método Java recursivo que determina se uma string s é um palíndromo, isto é, se é igual ao seu inverso. Por exemplo `"racecar"` e `"gohangasalamiimalasagnahog"` são palíndromos.

C-3.20 Use recursão para escrever um método Java para determinar se uma string s tem mais vogais que consoantes.

C-3.21 Suponha que são fornecidas duas listas circulares, L e M, isto é, duas listas de nodos de forma que cada nodo possui uma referência **next** não-nula. Descreva um algoritmo rápido para dizer se L e M são na verdade a mesma lista de nodos, mas com diferentes (cursores) pontos de partida.

C-3.22 Dada uma lista circular encadeada L contendo um número par de nodos, descreva como dividir L em duas listas encadeadas circulares com a metade do tamanho.

Projetos

P-3.1 Escreva um programa Java para uma classe matriz que possa adicionar e multiplicar arranjos bidimensionais de inteiros quaisquer.

P-3.2 Execute o projeto anterior, mas use tipos genéricos de maneira que as matrizes involvidas possam conter tipos arbitrários de números.

P-3.3 Escreva uma classe que mantém os dez maiores escores para uma aplicação de jogo, implementando os métodos add e remove da Seção 3.11, mas usando uma lista simplesmente encadeada em vez de um arranjo.

P-3.4 Execute o projeto anterior, mas use uma lista duplamente encadeada. Além disso, sua implementação de remove(i) deve fazer o menor número de deslocamentos sobre as conexões para obter a entrada sob o índice i.

P-3.5 Execute o projeto anterior mas use uma lista que encadeada que seja tanto circular como duplamente encadeada.

P-3.6 Escreva um programa para resover os quebra-cabeças de soma pela enumeração e teste de todas as soluções possíveis. Usando seu programa, resolva os três quebra-cabeças fornecidos na Seção 3.5.3.

P-3.7 Escreva um programa que criptografe e decriptografe usando uma cifra de substituição arbitrária. Neste caso, o arranjo de criptografia é uma mistura aleatória de letras do alfabeto. Seu programa deve gerar o array aleatório de criptografia, seu arranjo correspondente de decodificação e use estes para codificar e decodificar uma mensagem.

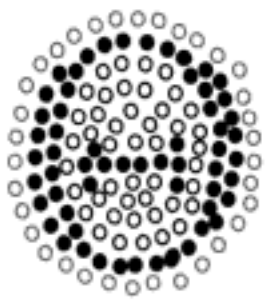
P-3.8 Escreva um programa que possa executar a Cifra de César para mensagens em inglês que incluam tanto caracteres minúsculos como maiúsculos.

Observações sobre o capítulo

As estruturas de dados fundamentais de arranjos e listas encadeadas, bem como recursão, discutidas neste capítulo pertencem ao folclore da Ciência da Computação. Elas foram descritas pela primeira vez na literatura de Ciência da Computação por Knuth no seu original livro sobre *Fundamentos de Algoritmos* [62].

Capítulo

4 Ferramentas de Análise



Conteúdo

R-4.29 Mostre que 2^{n+1} é $O(2^n)$.

R-4.30 Mostre que n é $O(n \log n)$.

R-4.31 Mostre que n^2 é $\Omega(n \log n)$.

R-4.32 Mostre que $n \log n$ é $\Omega(n)$.

R-4.33 Mostre que $\lceil Of(n) \rceil$ é $O(f(n))$, se $f(n)$ é uma função positiva não decrescente que é sempre maior que 1.

R-4.34 O algoritmo A executa uma computação em tempo $O(\log n)$ para cada entrada de um arranjo de n elementos. Qual o pior caso em relação ao tempo de execução de A?

R-4.35 Dado um arranjo X de n elementos, o algoritmo B escolhe $\log n$ elementos de x, aleatoriamente, e executa um cálculo em tempo $O(n)$ para cada um. Qual o pior caso em relação ao tempo de execução de B?

R-4.36 Dado um arranjo X de n elementos inteiros, o algoritmo C executa uma computação em tempo $O(n)$ para cada número par de X e uma computação em tempo $O(\log n)$ para cada elemento impar de X. Qual o melhor caso e o pior caso em relação ao tempo de execução de C?

R-4.37 Dado um arranjo X de n elementos, o algoritmo D chama o algoritmo E para cada elemento $X[i]$. O algoritmo E executa em tempo $O(i)$ quando é chamado sobre um elemento $X[i]$. Qual o pior caso em relação ao tempo de execução do algoritmo D?

R-4.38 Al e Bob estão discutindo sobre seus algoritmos. Al afirma que seu método de tempo $O(n \log n)$ é *sempre* mais rápido que o método de Bob de tempo $O(n^2)$. Para decidir a questão, eles executaram um conjunto de experimentos. Para o espanto de Al, eles encontraram que se $n < 100$, o algoritmo $O(n^2)$ executa mais rápido, e somente quando $n \geq 100$ é que o algoritmo $O(n \log n)$ é um pouco melhor. Explique como isso é possível.

Criatividade

C-4.1 Descreva um algoritmo recursivo para calcular a parte inteira do logaritmo de base 2 de n usando apenas somas e divisões inteiras.

C-4.2 Descreva como implementar um TAD fila usando duas pilhas. Qual o tempo de execução dos métodos `enqueue( )` e `dequeue( )` neste caso?

C-4.3 Suponha que seja fornecido um arranjo A de n elementos contendo inteiros distintos que são listados em ordem crescente. Dado um número k, descreva um algoritmo recursivo para encontrar dois inteiros em A cuja soma seja k, se tal par existir. Qual o tempo de execução do seu algoritmo?

C-4.4 Dado um arranjo A de n elementos inteiros não ordenado e um inteiro k, descreva um algoritmo recursivo para reorganizar os elementos de A de maneira que os elementos menores ou iguais a k antecedam qualquer elemento maior que k. Qual é o tempo de execução do seu algoritmo?

C-4.5 Mostre que $\sum_{i=1}^{n} i^2$ é $O(n^3)$.

C-4.6 Mostre que $\sum_{i=1}^{n} i / 2^i < 2$. (Dica: tente limitar esta soma, termo a termo, usando uma progressão geométrica).

C-4.7 Mostre que é $\log_b f(n)$ é $\Theta(\log f(n))$ se $b > 1$ é uma constante.

C-4.8 Descreva um método para encontrar tanto o mínimo como o máximo entre n números usando menos que $3n/2$ comparações. (Dica: primeiro crie um grupo de candidatos a mínimo e um grupo de candidatos a máximo).

C-4.9 Bob construiu um site Web e forneceu a URL apenas para os seus n amigos, que ele numerou de 1 a n. Ele disse ao amigo número i que ele pode visitar o site no máximo i vezes. Agora Bob tem um contador, C, que mantém o total de visitas ao site (mas não as identidades dos visitantes). Qual é o valor mínimo de C tal que Bob possa ficar sabendo que um de seus amigos está visitando o site mais do que o número permitido de vezes?

C-4.10 Considere a seguinte "justificativa" para o fato da função Fibonacci, $F(n)$, (veja a Proposição 4.20) ser $O(n)$:
Caso Base ($n \leq 2$): $F(1) = 1$ e $F(2) = 2$;
Passe de indução ($n > 2$): Assume-se a afirmação como verdadeira para $n < n$. Considere n. $F(n) = F(n-1)+F(n-2)$. Por indução, $F(n-1)$ é $O(n' - 1)$ e $F(n-2)$ é $O(n-2)$. Então $F(n)$ é $O((n-1) + (n-2))$, pela identidade apresentada no Exercício R-4.22. Conseqüentemente, $F(n)$ é $O(n)$.
O que está errado nesta justificativa?

C-4.11 Seja $p(x)$ uma polinomial de grau n, isto é, $\sum_{i=0}^{n} a_i x^i$.

(a) Descreva um método simples de tempo $O(n^2)$ para calcular $p(x)$.
(b) Agora considere re-escrever $p(x)$ como

$$p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + \cdots + x(a_{n-1} + xa_n)\cdots))),$$

o que é conhecido como ***método de Horner***. Usando a notação O, caracterize a quantidade de operações aritméticas que este método executa.

C-4.12 Considere a função Fibonacci, $F(n)$ (veja Proposição 4.20). Mostre por indução que $F(n)$ é $\Omega((3/2)^n)$.

C-4.13 Dado um conjunto $A = \{a_1, a_2, \ldots, a_n\}$ de n inteiros, descreva em pseudocódigo um método eficiente para calcular cada uma das somas parciais $s_k = \sum_{i=1}^{k} a_i$ para $k = 1, 2, \ldots, n$. Qual o tempo de execução deste método?

C-4.14 Desenhe uma justificativa visual para a Proposição 4.3 análoga a da Figura 4.1(b) para o caso onde n é ímpar.

C-4.15 Um arranjo A contém $n-1$ inteiros únicos no intervalo $[0, n-1]$, isto é, existe um número neste arranjo que não está em A. Projete um algoritmo de tempo $O(n)$ para encontrar este número. Pode-se usar somente $O(1)$ espaço adicional além do arranjo A propriamente dito.

C-4.16 Seja S um conjunto de n linha no plano tal que não existem duas paralelas a não existe um trio que se encontre no mesmo ponto. Mostre, por indução, que as linhas de S determinam $\Theta(n^2)$ pontos de intersecção.

C-4.17 Mostre que o somatório $\sum_{i=1}^{n} \lceil \log_2 i \rceil$ é $O(n \log n)$.

C-4.18 Um rei malvado tem n garrafas de vinho e um espião envenenou apenas uma delas. Infelizmente ele não sabe qual. O veneno é extremamente mortal: apenas uma gota diluída em um bilhão ainda mata. Mesmo assim, leva um mês para o veneno fazer efeito. Desenhe um esquema para determinar exatamente qual das garrafas de vinho foi envenenada, em apenas um mês, usando apenas $O(\log n)$ testadores.

C-4.19 Suponha que cada linha de um arranjo A, $n \times n$, consiste de zeros e uns tais que, em qualquer linha de A, todos os uns antecedem todos os zeros.

Assumindo que A ainda está na memória, descreva um método que rode em tempo $O(n)$ (não $O(n^2)$) para encontrar a linha de A que tem o maior número de uns.

C-4.20 Descreva em pseudocódigo um método para multiplicar uma matriz A $n \times m$ e uma matriz B $m \times p$. Lembre que o produto $C = AB$ é definido como $C[i][j] = \sum_{k=1}^{m} A[i][k] \cdot B[k][j]$. Qual o tempo de execução de seu método?

C-4.21 Suponha que cada linha de um arranjo A, $n \times n$, consiste em zeros e uns tais que, em qualquer linha de A, todos os uns antecedem todos os zeros. Também suponha que o número de uns na linha i é pelo menos o número na linha $i + 1$, para $i = 0,1, \ldots, n - 2$. Assumindo que A está na memória, descreva um método que execute em tempo $O(n)$ (não $O(n^2)$) para contar o número de uns em A.

C-4.22 Descreva um método recursivo para calcular o n-ésimo ***número harmônico***, $H_n = \sum_{i=1}^{n} 1/i$.

Projetos

P-4.1 Implemente prefixAverages1 e prefixAverages2, da Seção 4.2.5, e execute uma análise experimental dos seus tempos de execução. Visualize seus tempos de execução como uma função do tamanho da entrada usando um gráfico di-log.

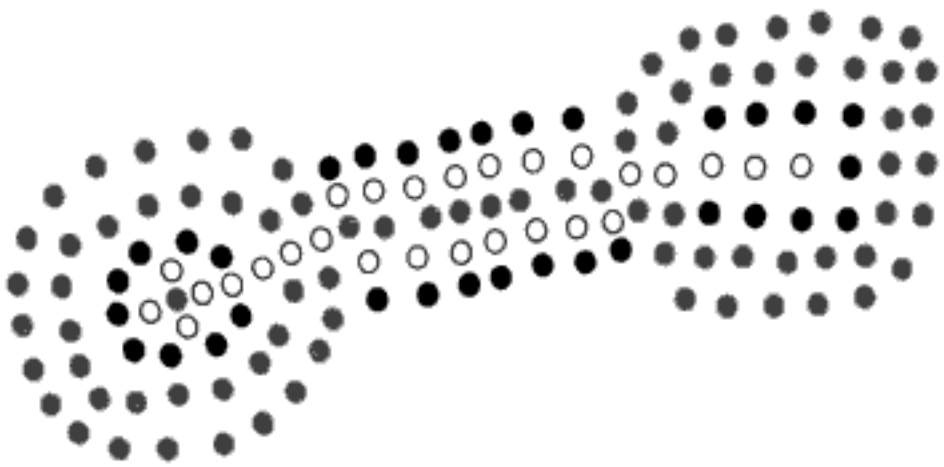
P-4.2 Execute uma análise experimental cuidadosa que compare os tempos relativos de execução dos métodos apresentados no Trecho de código 4.5.

Observações sobre o capítulo

A notação O tem gerado vários comentários sobre seu uso [16, 47, 61]. Knuth [62,61] a define usando a notação $f(n) = O(g(n))$, mas diz que esta igualdade funciona apenas em um sentido. Foi escolhida uma visão mais tradicional de igualdade e considerou-se a notação O como um conjunto, seguindo Brassard [16]. O leitor interessado em estudar análise do caso médio pode procurar o capítulo do livro de Vitter e Flajolet [97]. A história de Arquimedes é encontrada em [77]. Para algumas ferramentas matemáticas adicionais, consulte o Apêndice A.

Capítulo

5 Pilhas e Filas



Conteúdo

C-5.12 Suponha que Bob tem quatro vacas e que ele quer levá-las através da ponte. Mas ele possui apenas um cambão, que une apenas duas vacas lado a lado. O cambão é muito pesado para que ele possa carregá-lo pela ponte, mas ele pode amarrar (e soltar) as vacas no mesmo, rapidamente. De suas quatro vacas, Mazie pode atravessar a ponte em 2 minutos, Dayse pode atravessar em 4 minutos, Crazy leva 10 minutos e Lazy pode fazê-lo em 20 minutos. Naturalmente, quando duas vacas estão presas ao cambão, elas devem andar na velocidade da vaca mais lenta. Descreva como Bob pode atravessar suas vacas pela ponte em 34 minutos.

Projetos

P-5.1 Implemente o TAD pilha usando uma lista duplamente encadeada.

P-5.2 Implemente o TAD pilha usando a classe ArrayList de Java (sem usar a classe predefinida de Java, Stack).

P-5.3 Implemente um programa que possa receber uma expressão em notação pós-fixada (ver Exercício C-5.8) e exibir seu valor.

P-5.4 Implemente o TAD fila usando um arranjo.

P-5.5 Implemente todo o TAD fila usando uma lista simplesmente encadeada.

P-5.6 Projete um TAD para uma pilha dupla de duas cores que consiste em duas pilhas – uma "vermelha" e outra "azul" – e tem suas versões coloridas das operações normais de um TAD pilha. Por exemplo, este TAD pode admitir tanto uma operação push azul como vermelha. Apresente uma implementação eficiente deste TAD usando um único arranjo cuja capacidade é definida em um valor N que se assume ser maior que os tamanhos das pilhas vermelho e azul combinadas.

P-5.7 Implemente o TAD deque usando uma lista duplamente encadeada.

P-5.8 Implemente o TAD deque usando um arranjo tratado de forma circular.

P-5.9 Implemente as interfaces Stack e Queue com uma única classe que estende a classe NodeQueue (Trecho de código 5.8).

P-5.10 Quando um lote de ações de uma companhia é vendido, o ***capital obtido*** (ou às vezes perdido) é a diferença entre o preço de venda e o preço pago originalmente pelas ações. Esta regra é fácil de entender para uma única ação, mas se vendemos vários lotes de ações comprados ao longo de um período de tempo, então é necessário identificar as ações que estão sendo vendidas. Um princípio padrão em contabilidade para a identificação de lotes de ações vendidas, neste caso, é o uso de um protocolo FIFO – as ações vendidas são aquelas que foram armazenadas mais tempo (na verdade este é o princípio padrão adotado em vários pacotes de software de finanças pessoais). Por exemplo, suponha que se deseja comprar 100 ações a R$ 20,00 cada, no dia 1; 20 ações a R$ 24,00, no dia 2; 200 ações a R$ 36,00, no dia 3; e então vender 150, ações, no dia 4, a R$ 30,00 cada. Então, aplicando o princípio FIFO, significa que das 150 ações vendidas, 100 foram compradas no dia 1, 20 no dia 2 e 30 no dia 3. O capital obtido neste caso foi $100 \cdot 10 + 20 \cdot 6 + 30 \cdot (-6)$, ou R$ 940,00. Escreva um programa que recebe como entrada uma seqüência de transações do tipo "`compre ações(s) por R$y cada`" ou "`venda x ações(s) por`

R$y cada", assumindo que as transações ocorrem em dias consecutivos e que os valores de x e y são inteiros. Dada a seqüência de entrada, a saída pode ser o capital total ganho (ou perdido) para a seqüência completa, usando um protocolo FIFO para identificar as ações.

Observações sobre o capítulo

A abordagem de definir primeiro as estruturas de dados em termos de seus TADs e depois de suas implementações concretas foi introduzida pela primeira vez pelos livros clássicos de Aho, Hopcroft e Ullman [4,5], que, não por acaso, são os primeiros trabalhos em que se vê um problema similar ao do exercício C-5.6. Os exercícios C-5.10, C-5.11 e C-5.12 são similares às questões da entrevista ditas originárias de uma companhia de software bem conhecida. Para aprofundar seus estudos de tipos abstratos de dados, veja Liskov e Guttag [69], Cardelli e Wegner [20] ou Demurjian [28].

Capítulo

6 Listas e Iteradores

Conteúdo

C-6.19 Uma operação útil sobre bancos de dados é a ***junção natural****. Se entendermos um banco de dados como uma lista de pares ordenados de objetos, então a junção natural dos bancos de dados A e B é a lista de todas as triplas ordenadas (x, y, z), tal que o par (x,y) se encontra em A e o par (y, z), em B. Descreva e analise um algoritmo eficiente para computar a junção natural de uma lista A de n pares e uma lista B de m pares.

C-6.20 Quando Bob deseja enviar uma mensagem M para Alice via Internet, ele quebra M em ***pacotes de dados***, numera os pacotes consecutivamente e injeta-os na rede. Quando os pacotes chegam no computador de Alice, eles podem estar fora de ordem, de maneira que Alice precisa remontar a seqüência em ordem antes de se certificar de ter recebido toda mensagem. Descreva um esquema eficiente para Alice fazer isso. Qual o tempo de execução deste algoritmo?

C-6.21 Forneça uma lista L com n inteiros positivos, cada um representado usando $k = \lceil \log n \rceil + 1$ bits, descreva um método de tempo $O(n)$ para encontrar um inteiro de k bits que não esteja em L.

C-6.22 Argumente porque qualquer solução para o problema anterior deve executar em tempo $\Omega(n)$.

C-6.23 Apresente uma lista L com n inteiros arbitrários, projete um método de tempo $O(n)$ para encontrar um inteiro que não possa ser formado pela soma de dois inteiros que estão em L.

C-6.24 Isabel tem uma forma interessante de totalizar a soma dos valores de um arranjo A de n inteiros, onde n é uma potência de dois. Ela criou um arranjo B com a metade do tamanho de A e fez $B[i] = A[2i] + A[2i + 1]$, para $i = 0,1, \ldots, (n/2) - 1$. Se B tem tamanho 1, então ela retorna $B[0]$. Em qualquer outro caso, ela substitui A por B e repete o processo. Qual o tempo de execução de seu algoritmo?

Projetos

P-6.1 Implemente um TAD lista arranjo como um arranjo extensível usado de maneira circular de forma que inserções e deleções no início e no fim do arranjo sejam executadas em tempo constante.

P-6.2 Implemente o TAD lista arranjo usando uma lista duplamente encadeada. Mostre experimentalmente que esta implementação é pior do que a abordagem baseada em arranjo.

P-6.3 Escreva um editor de textos simples que armazena e exibe uma string de caracteres usando o TAD lista juntamente com um objeto cursor que destaca a posição de um dos caracteres da string. O editor deve suportar as seguintes operações:

- left: move o cursor um caractere para a esquerda (ou não faz nada se estiver no fim do texto).
- rigth: move o cursor um caractere para a direita (ou não faz nada se estiver no fim do texto).

* N. de T. Em inglês, *natural join*.

- cut: apaga o caractere à direita do cursor (ou não faz nada se estiver no fim do texto).
- paste (c): insere o caractere c após o cursor.

P-6.4 Implemente uma ***lista de favoritos com fases***. Uma fase consiste em N acessos na lista para um dado parâmetro N. Durante uma fase, a lista deve manter seus elementos ordenados em ordem decrescente dos contadores de acesso. Ao final da fase, ela deve limpar os contadores de acesso e iniciar a próxima fase. Experimentalmente, determine quais são os melhores valores de N para vários tamanhos de lista.

P-6.5 Escreva uma classe adaptadora completa que implemente o TAD seqüência usando um objeto java.util.ArrayList.

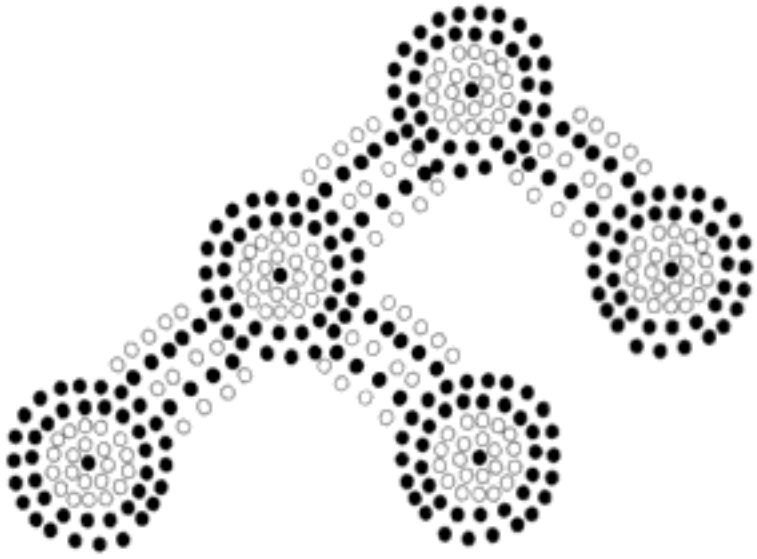
P-6.6 Implemente a lista de favoritos usando uma lista arranjo em vez de uma lista. Compare-a, experimentalmente, com uma implementação que use lista.

Observações sobre o capítulo

A concepção de entender estruturas de dados como coleções (e outros princípios de projeto orientado a objetos) podem ser encontrados nos livros de projeto orientado a objetos de Booch [14], Budd[17], Golberg e Robson [40] e Liskov e Guttag[69]. Listas e iteradores são conceitos impregnados no framework de coleções de Java. Nosso TAD lista é derivado da abstração "posição", introduzida por Aho, Hopcroft e Ullman [5] e o TAD lista de Wood [100]. Implementações de listas usando arranjos e listas encadeadas são discutidas por Knuth [62].

Capítulo

7 Árvores



Conteúdo

nodos v e w de T, dados $p(v)$ e $p(w)$. Não é necessário determinar u, apenas calcular o número que identifica seu nível.

C-7.24 Justifique os limites da Tabela 7.3 com uma análise detalhada dos tempos de execução dos métodos de uma árvore binária T, implementada sobre uma lista arranjo S, onde S é definida sobre um arranjo.

C-7.25 Justifique a Tabela 7.1 resumindo o tempo de execução dos métodos de uma árvore representada com uma estrutura encadeada apresentando, para cada método, uma descrição de sua implementação e uma análise do tempo de execução.

C-7.26 Descreva um método não-recursivo para avaliar uma árvore binária que representa uma expressão aritmética.

C-7.27 Seja T uma árvore binária com n nodos, defina um ***nodo romano*** como sendo um nodo v de T, de maneira que o número de descendentes na subárvore esquerda de v diferencie-se do número de descendentes da subárvore direita de v por no máximo 5. Descreva um método com tempo de execução linear para encontrar cada nodo v de T, tal que v não seja um nodo romano, mas que todos os seus descendentes o sejam.

C-7.28 Descreva um método não-recursivo para executar o caminhamento de Euler sobre uma árvore binária que execute em tempo linear e não use uma pilha.

C-7.29 Descreva em pseudocódigo um método não-recursivo para executar o caminhamento interfixado sobre uma árvore binária em tempo linear.

C-7.30 Seja T uma árvore binária com n nodos (T pode ser implementada usando uma lista arranjo ou uma estrutura encadeada). Forneça um método de tempo linear que use métodos da interface BinaryTree para percorrer os nodos de T através do incremento dos valores da função de numeração por nível p apresentada na Seção 7.3.5. Esse caminhamento é conhecido como ***caminhamento por nível***.

C-7.31 O ***tamanho do caminho*** de uma árvore T é a soma das profundidades de todos os nodos de T. Descreva um método com tempo de execução linear para calcular o tamanho do caminho de uma árvore T (que não é necessariamente binária).

C-7.32 Defina o ***tamanho do caminho interno***, $I(T)$, de uma árvore T como sendo a soma das profundidades de todos os nodos internos de T. Da mesma forma, defina o ***tamanho do caminho externo***, $E(T)$, de uma árvore T como sendo a soma das profundidades de todos os nodos externos de T. Mostre que se T é uma árvore binária com n nodos internos, então $E(T) = I(T) + n - 1$.

Projetos

P-7.1 Implemente o TAD árvore binária usando uma lista arranjo.

P-7.2 Implemente o TAD árvore binária usando uma lista encadeada.

P-7.3 Escreva um programa capaz de desenhar uma árvore binária.

P-7.4 Escreva um programa capaz de desenhar uma árvore genérica.

P-7.5 Escreva um programa que permita tanto entrar como exibir a árvore genealógica de alguém.

P-7.6 Implemente o TAD árvore usando a representação por árvores binárias descrita no Exercício C-7.12. Você pode reusar a implementação de LinkedBinaryTree para árvores binárias.

P-7.7 Uma ***planta baixa fatiada*** é a decomposição de um retângulo com lados horizontais e verticais usando ***cortes*** horizontais e verticais. (Veja a Figura 7.23a.) Uma planta baixa fatiada pode ser representada por uma árvore binária chamada de ***árvore de fatias***, cujos nodos internos representam os cortes e os nodos externos representam os retângulos básicos em que o chão é dividido pelos cortes. (Veja a Figura 7.23b.) O ***problema da compactação*** é definido como segue. Pressuponha que para cada retângulo básico de uma planta baixa fatiada atribui-se uma largura mínima w e uma altura mínima h. O problema da compactação é encontrar a menor largura e altura para cada retângulo da planta baixa que seja compatível com as dimensões mínimas de cada retângulo básico. Em outras palavras, este problema requer a atribuição de valores $h(v)$ e $w(v)$ para cada nodo v da árvore de fatias de maneira que:

$$w(v) = \begin{cases} w & \text{se } v \text{ for um nodo externo cujo retângulo base tem a largura mínima } w \\ \max(w(w), w(z)) & \text{se } v \text{ for um nodo interno associado com um corte horizontal com o filho da esquerda } w \text{ e da direita } z \\ w(w) + w(z) & \text{se } v \text{ for um nodo interno associado com um corte vertical com o filho da esquerda } w \text{ e da direita } z \end{cases}$$

$$h(v) = \begin{cases} h & \text{se } v \text{ for um nodo externo cujo retângulo base tem a altura mínima } h \\ h(w) + h(z) & \text{se } v \text{ for um nodo interno associado com um corte horizontal com o filho da esquerda } w \text{ e da direita } z \\ \max(h(w), h(z)) & \text{se } v \text{ for um nodo interno associado com um corte vertical com o filho da esquerda } w \text{ e da direita } z \end{cases}$$

Projete uma estrutura de dados para plantas baixas fatiadas que suporte as operações:

- criar uma planta baixa composta de retângulos básicos individuais;
- decompor um retângulo básico através de um corte horizontal;
- decompor um retângulo básico através de um corte vertical;
- atribuir uma altura e largura mínima a um retângulo básico;
- desenhar a árvore de fatias associada à planta baixa;
- compactar e desenhar a planta baixa.

P-7.8 Escreva um programa que efetivamente possa jogar o jogo-da-velha (ver Seção 3.1.5). Para tanto, será necessário criar uma ***árvore de jogadas*** T, que é uma árvore na qual cada nodo representa uma ***configuração de jogada***, o que, neste caso, corresponde a uma representação do tabuleiro do jogo-da-velha. O nodo raiz corresponde à configuração inicial. Para cada nodo interno v de T, os filhos de v correspondem aos estados do jogo possíveis de serem alcançados a partir do estado inicial em uma única

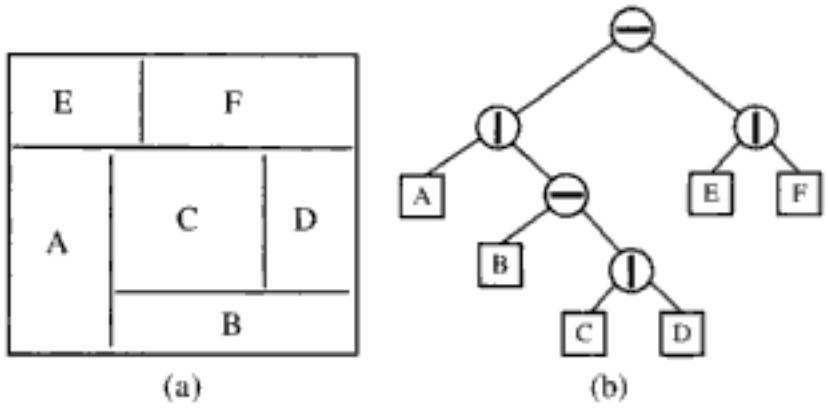


Figura 7.23 (a) Planta baixa fatiada; (b) árvore de fatias associada com a planta baixa.

jogada do jogador da vez, A (o primeiro jogador) ou B (o segundo jogador). Nodos de profundidade par correspondem a jogadas de A e nodos de profundidade ímpar correspondem a jogadas de B. Nodos externos podem ser tanto estados finais do jogo ou estarem localizados em uma profundidade que não se deseja explorar. Para cada nodo externo atribui-se um valor que indica quão bom este estado é para o jogador A. Em jogos mais complexos, como o xadrez, é necessário adotar uma função heurística para atribuir este valor, mas para jogos simples, como jogo-da-velha, pode-se construir toda a árvore de jogadas e atribuir valor para os nodos com $+1$, 0, -1, indicando se o jogador A tem a ganhar ou a perder com está configuração. Um bom algoritmo de seleção de jogadas é o ***minimax***. Neste algoritmo, atribui-se um valor para cada nodo interno v de T, de maneira que se v representa a vez de A, calcula-se o valor de v como o valor máximo dos filhos de v (que corresponde a melhor jogada para A a partir de v). Se um nodo interno v representa a vez de B, então calcula-se o valor de v como o menor valor dos filhos de v (o que corresponde a melhor jogada para B a partir de v).

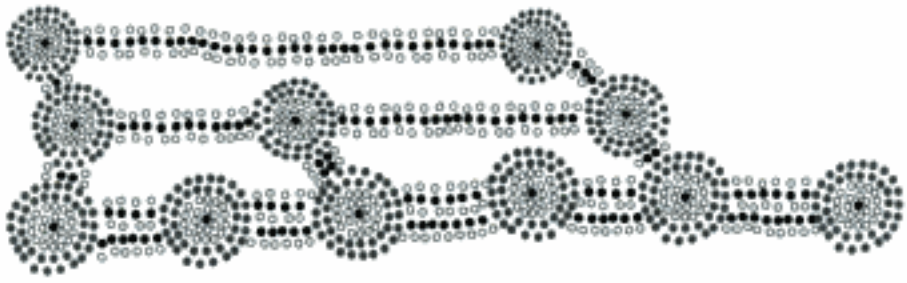
P-7.9 Escreva um programa que receba como entrada uma expressão aritmética toda entre parênteses e a converta em uma árvore binária que representa uma expressão. Seu programa deve exibir a árvore de alguma forma e também imprimir o valor associado com a raiz. Como desafio adicional, permita que se armazene nas folhas variáveis da forma x_1, x_2, x_3 e, assim por diante que são inicializadas com 0 e que podem ser atualizadas interativamente por meio do programa, atualizando de forma coerente o valor impresso que corresponde ao valor da raiz da árvore que representa a expressão.

P-7.10 Escreva um programa que visualiza o caminhamento de Euler sobre uma árvore binária própria, incluindo os movimentos nodo a nodo e as ações associadas com as visitas pela esquerda, por baixo e pela direita. Demonstre seu programa fazendo-o computar e mostrar os rótulos prefixados, interfixados e pós-fixados, bem como contadores de ancestrais e contadores de descendentes para cada nodo da árvore (não necessariamente todos ao mesmo tempo).

P-7.11 A codificação de expressões aritmética apresentada nos Trechos de código 7.29-7.32 funciona apenas para expressões inteiras com operador de soma. Escreva um programa Java que possa calcular expressões arbitrárias com qualquer tipo numérico de objeto.

Capítulo

9 Mapas e Dicionários



Conteúdo

C-9.13 Suponha que cada linha de um arranjo A de tamanho $n \times n$ consiste em 1 e 0, tal que em qualquer linha de A todos os valores 1 venham antes de todos os valores 0. Assumindo que A esteja em memória, descreva um método com tempo $O(n \log n)$ (e não tempo $O(n^2)$!) para contar o número de valores 1 em A.

C-9.14 Descreva uma estrutura ordenada eficiente para um dicionário que armazena n elementos e tem um conjunto de ordem total associado de $k < n$ chaves. Ou seja, o conjunto de chaves é menor que o elemento. Sua estrutura deverá executar a operação findAll no tempo esperado de $O(\log r + s)$, onde s é o número de elementos retornados, a operação entries( ) no tempo $O(n)$ e operações restantes do TAD dicionário no tempo esperado de $O(\log r)$.

C-9.15 Descreva uma estrutura eficiente de dicionário para armazenar n elementos com $r < n$ e chaves que contenham distintos códigos de hash. Sua estrutura deverá executar a operação findAll no tempo esperado de $O(1 + s)$, onde s é o número de elementos retornados, e a operação entries( ) no tempo $O(n)$, e operações restantes do TAD dicionário no tempo esperado de $O(1)$.

C-9.16 Descreva uma eficiente estrutura de dados para implementar o TAD ***sacola***, o qual suporta um método add(e), para adicionar um elemento arbitrário e na sacola, e um método remove( ), o qual remove um elemento arbitrário da sacola. Mostre que ambos os métodos podem ser feitos no tempo $O(1)$.

C-9.17 Descreva como modificar uma skip list para suportar o método atIndex(i), que retorna a posição do elemento "base" na lista S_0 com colocação i, pois $i \in [0, n - 1]$. Mostre que sua implementação deste método tem tempo esperado $O(\log n)$.

Projetos

P-9.1 Implemente uma classe que implemente o TAD dicionário pela adaptação da classe java.util.HashMap

P-9.2 Implemente o TAD dicionário com uma tabela de hash que trata as colisões com encadeamento separado (não adapte qualquer classe do pacote java.util).

P-9.3 Implemente o TAD dicionário ordenado usando uma seqüência ordenada.

P-9.4 Implemente os métodos de um TAD dicionário ordenado usando uma skip list.

P-9.5 Estenda o projeto anterior fornecendo uma animação gráfica da operação da skip list. Visualize como os itens se movem para cima na skip list durante a operação de inserção e como são retirados durante a remoção. Na operação de procura, visualize as varreduras e descidas para o nível inferior.

P-9.6 Implemente um dicionário que suporta métodos baseados em localizadores através de uma seqüência ordenada

P-9.7 Faça uma análise comparativa que estuda as taxas de colisão para vários códigos de hash para cadeias de caracteres, tais como códigos hash polinomiais para diferentes valores do parâmetro a. Use uma tabela de hash para determinar colisões, mas apenas conte colisões em que cadeias diferentes levam ao mesmo código de hash (e não se elas levam à mesma posição da tabela de hash). Teste os códigos hash em arquivos encontrados na Internet.

P-9.8 Faça uma análise comparativa, como no exercício anterior, para números de telefone de dez dígitos, em vez de cadeias de caracteres.

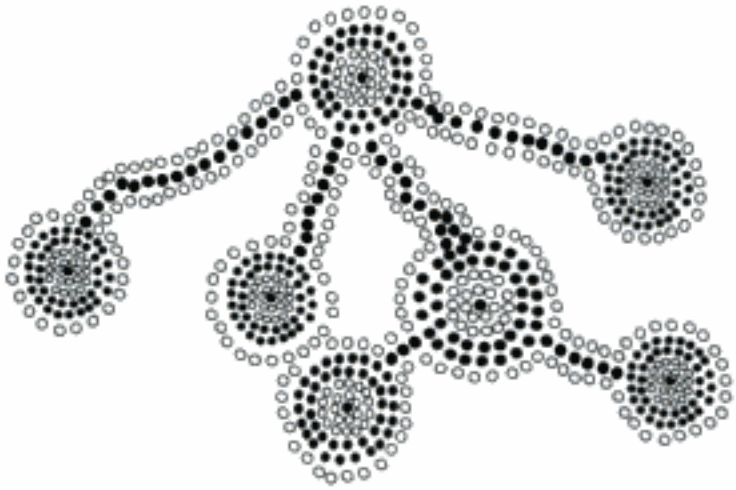
P-9.9 Projete uma classe Java que implemente a estrutura de dados skip list. Use esta classe para criar implementações de TAD mapa e TAD dicionário, incluindo métodos localizadores para o dicionário.

Observações sobre o capítulo

É interessante notar que o algoritmo de pesquisa binária foi publicado pela primeira vez em 1946, mas só foi publicado em uma versão completamente correta em 1962. Para mais discussões sobre as lições a serem aprendidas desta história, veja o livro de Knuth [60] e os artigos de Bentley [12] e Levisse [65]. As skip lists foram introduzidas por Pugh [84]. A presente análise das skip lists é uma simplificação da apresentação feita no livro de Motwani e Raghavan [79]. Além disso, a discussão de hashing também é uma simplificação da apresentação daquele livro. O leitor interessado em outras construções probabilísticas para suportar dicionários (incluindo mais informação sobre hashing) pode verificar o livro de Motwani e Raghavan [79]. Para uma análise mais profunda sobre skip lists, o leitor pode consultar os artigos na literatura de estruturas de dados [57,81,82]. O Exercício C-9.9 foi uma contribuição de James Lee.

Capítulo

10 Árvores de Pesquisa



Conteúdo

heap unificável h com o presente, destruindo as versões antigas de ambos. Descreva uma implementação concreta para o TAD heap unificável que obtenha performance $O(\log n)$ para todas as suas operações.

C-10.19 Considere uma variação da árvore splay chamada ***árvores half-splay,*** onde a expansão de um nodo com profundidade d para assim que o nodo consiga a profundidade $\lfloor d/2 \rfloor$. Execute uma análise de amortização das árvores half-splay.

C-10.20 A etapa de expansão padrão requer duas passagens, uma descida para encontrar o nodo x para expansão, seguida por uma subida para expandir o nodo x. Descreva um método para expansão e pesquisa pelo nodo x em um passo de descida. Cada subpasso requer que você considere os próximos dois nodos no caminho abaixo de x, com um possível subpasso zig executado no final. Descreva como executar os passos zig-zig, zig-zag e zig.

C-10.21 Descreva uma seqüência de acessos a um nodo n da árvore splay T, onde n é ímpar que resulta em T, consistindo em uma simples cadeia de nodos internos com filhos que são nodos externos, no qual o caminho do nodo interno abaixo T alterna entre o filho à esquerda e o filho à direita.

C-10.22 Explique como implementar um arranjo de n elementos onde os métodos add e get levam o tempo $O(\log n)$ no pior caso (sem a necessidade de um arranjo expansível).

Projetos

P-10.1 Simulações de n-corpos são ferramentas de modelagem importantes na física, astronomia e química. Neste projeto, você tem que escrever um programa que execute uma simples simulação de n-corpos chamada "Duendes Saltitantes"*. Esta simulação envolve n duendes, numerados de 1 até n. Ela mantém um valor ouro g_i para cada duende i, e se inicia com cada duende começando com o valor do ouro em um milhão de dólares, isto é, $g_i = 1\,000\,000$ para cada $i = 1, 2, \ldots, n$. Além disso, a simulação também mantém, para cada duende i, um lugar no horizonte, que é representado com um número de ponto flutuante de dupla precisão, x_i. Em cada iteração da simulação, esta processa os duendes na ordem. O processamento da um duende durante esta iteração inicia pela computação de um novo lugar no horizonte para i, que é determinado pela seguinte tarefa

$$x_i \leftarrow x_i + rg_i,$$

onde r é um número de ponto flutuante gerado randomicamente dentro do intervalo -1 e 1. O duende i então rouba metade do ouro do duende mais próximo de um dos seus lados e adiciona este ouro no seu valor de ouro g_i. Escreva um programa que possa executar uma série de iterações nesta simulação para um dado número, n, de duendes. Tente incluir uma visualização dos duendes nesta simulação, incluindo seus valores de ouro e posições no horizonte. Você pode manter o conjunto de posições do horizonte usando uma estrutura de dados de dicionário ordenado descrita neste capítulo.

* N. de T. O autor utiliza a expressão "Jumping Leprechauns".

P-10.2 Estenda a classe BinarySearchTree (Trecho de código 10.3 – 10.5) para suportar os métodos de um TAD dicionário ordenado (ver Seção 9.5.2).

P-10.3 Implemente um classe RestructurableNodeBinaryTree que suporte os métodos de um TAD árvore binária, mais um método restructure para execução de uma operação de rotação. Esta classe é um componente da implementação de uma árvore AVL apresentada na Seção 10.2.2.

P-10.4 Escreva uma classe Java que implemente todos os métodos de um TAD dicionário ordenado (ver Seção 9.5.2) usando uma árvore AVL.

P-10.5 Escreva uma classe Java que implemente todos os métodos de um TAD dicionário ordenado (ver Seção 9.5.2) usando uma árvore (2,4).

P-10.6 Escreva uma classe Java que implemente todos os métodos de um TAD dicionário ordenado (ver Seção 9.5.2) usando uma árvore vermelho-preta.

P-10.7 Forme uma equipe de três programadores e que cada membro implemente um dos três projetos apresentados anteriormente. Faça um extensivo estudo para comparar a velocidade de cada um destas três implementações. Projete três conjuntos de experimentos, cada um favorecendo uma diferente implementação.

P-10.8 Escreve uma classe Java que possa pegar qualquer árvore vermelho-preta e convertê-la em uma árvore (2,4) correspondente e possa pegar qualquer árvore (2,4) e convertê-la em uma árvore vermelho-preta correpondente.

P-10.9 Execute um estudo experimental para comparar o desempenho de uma árvore vermelho-preta com uma skip list.

P-10.10 Prepare uma implementação de árvores splay que utilizem expansão bottom-up como descrito neste capítulo e outra que utilize uma expansão top-down como descrito no Exercício C-10.20. Execute um estudo experimental extensivo para verificar qual implementação é melhor na prática, se houver.

Observações sobre o capítulo

Algumas das estruturas de dados discutidas neste capítulo são descritas em detalhes por Knuth no seu livro *Sorting and Searching* [63] e por Mehlhorn em [74]. As árvores AVL são atribuídas a Adel'son-Vel'skii e Landis [1], que inventaram essa classe de árvores de pesquisa balanceadas em 1962. Árvores de pesquisa binária, árvores AVL e estruturas de hash são descritas por Knuth no seu livro *Sorting and Searching* [63]. Análises de altura média para árvores de pesquisa binária podem ser encontradas nos livros de Aho, Hopcroft e Ulman [5] e Cormen, Leiserson e Rivest [25]. O manual de Gonnet e Bazea-Yates [41] contém uma boa quantidade de comparações experimentais e teóricas entre implementações de dicionários. Ahos, Hopcroft e Ulman [4], discutem árvores (2,3), que são similares a árvores (2,4). Árvores vermelho-pretas são definidas por Bayer [10]. Variações e propriedades interessantes de árvores vermelho-pretas são apresentadas em um artigo de Guibas e Sedgewick [46]. O leitor interessado em aprender mais sobre diferentes tipos de estruturas de árvores balanceadas deve procurar os livros de Mehlhorn [74] e Tarjan [91] e o capítulo de livro de Mehlhorn e Tsakalidis [76]. Knuth [63] é uma leitura adicional excelente que inclui abordagens mais recentes de árvores balanceadas. Árvores splay foram inventadas por Sleator and Tarjan [86] (ver também [91]).

Capítulo

11 Ordernação, Conjuntos e Seleção

Conteúdo

algoritmo de Karen está correto e analise seu tempo de execução para um caminho de tamanho h.

C-11.31 Este problema trata de uma modificação do algoritmo de quick-select para torná-lo determinístico e ainda podendo ser executado em tempo $O(n)$ em uma seqüência de n elementos. A idéia é modificar a forma com que se escolhe o pivô de modo que seja escolhido de forma determinística, e não aleatoriamente, como segue:

> Divida o conjunto S em $\lceil n/5 \rceil$ grupos de tamanho 5 cada (exceto, talvez, um grupo). Ordene cada pequeno conjunto e identifique a mediana do conjunto. Com estas $\lceil n/5 \rceil$ medianas "pequenas", aplique o algoritmo de seleção recursivamente para encontrar a mediana das medianas "pequenas". Use este elemento como pivô e continue como no algoritmo de quick-select.

Mostre que este método determinístico tem tempo $O(n)$, respondendo as seguintes questões (ignore pisos e tetos se isto simplificar os cálculos, pois os resultados assintóticos continuam os mesmos):

a. Quantas medianas pequenas são menores do que ou iguais ao pivô escolhido? Quantas são maiores ou iguais?
b. Para cada mediana pequena menor ou igual ao pivô, quantos elementos são menores ou iguais ao pivô? O mesmo é verdadeiro para aquelas maiores ou iguais ao pivô?
c. Discuta por que o método para encontrar o pivô deterministicamente e usá-lo para particionar S custa tempo $O(n)$.
d. Baseado nestas estimativas, escreva uma relação de recorrência que limita o tempo de execução de pior caso $t(n)$ para este algoritmo de seleção. (Note que no pior caso há duas chamadas recursivas – uma para encontrar a mediana das medianas pequenas e outra para fazer a recorrência com o maior valor entre L e G.)
e. Usando essa relação de recorrência, mostre por indução que $t(n)$ é $O(n)$.

Projetos

P-11.1 Experimentalmente, compare o desempenho do quick-sort in-place e uma versão de um quick-sort que não seja in-place.

P-11.2 Projete e implemente uma versão estável do algoritmo de bucket-sort para ordenar uma seqüência de n elementos com chaves inteiras no intervalo $[0, N-1]$ para $N \geq 2$. O algoritmo deve ser executado em tempo $O(n + N)$.

P-11.3 Implemente o merge-sort e o quick-sort determinístico e realize testes para verificar qual dos dois é mais rápido. Seus testes devem incluir seqüências que aparentemente são "aleatórias" e seqüências que parecem "quase" ordenadas.

P-11.4 Implemente o quick-sort determinístico e sua versão randomizada e realize testes para verificar qual dos dois é mais rápido. Seus testes devem incluir seqüências que aparentemente são "aleatórias" e seqüências que parecem "quase" ordenadas.

P-11.5 Implemente uma versão in-place do insertion-sort e uma versão in-place do quick-sort. Realize testes para determinar os valores de n para os quais o quick-sort é melhor (em média) do que o insertion-sort.

P-11.6 Projete e implemente uma animação para um dos algoritmos de ordenação apresentados neste capítulo. Sua animação deve ilustrar as propriedades essenciais do algoritmo de forma intuitiva.

P-11.7 Implemente os algoritmos quick-sort randomizado e quick-select, e projete uma série de experimentos para testar suas velocidades relativas.

P-11.8 Implemente um TAD para conjuntos estendidos, incluindo os métodos union(B), intersect(B), size( ) e isEmpty( ) mais os métodos equals(B), contains(e), insert(e) e remove(e).

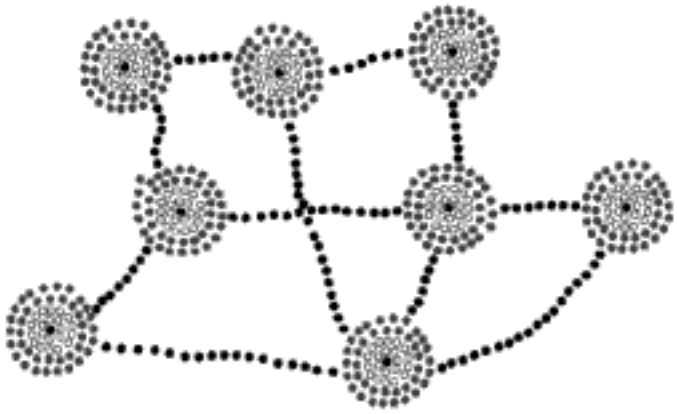
P-11.9 Implemente a estrutura de dados de partição union/find baseada em árvores com ambas as heurísticas: união-pelo-tamanho e compressão do caminho.

Observações sobre o capítulo

O clássico texto de Knuth *Sorting and Searching* [63] contém uma extensa história do problema da classificação e algoritmos para resolvê-lo. Huang e Langston [52] descrevem como unificar duas listas ordenadas de uma forma in-place e com tempo linear. Nosso TAD conjunto é derivado do TAD conjunto de Aho, Hopcroft e Ullman [5]. O algoritmo padrão para quick-sort foi feito por Hoare [49]. Uma análise mais profunda do quick-sort randomizado pode ser encontrada no livro de Motwani e Raghavan [79]. A análise do quick-sort apresentada neste capítulo é uma combinação de uma análise apresentada ne edição anterior deste livro, e a análise de Kleinberg e Tardos [59]. A análise do quick-sort do Exercício C-11.7 é devido a Littman. Gonnet e Baeza-Yates [41] fornecem comparações experimentais e análises teóricas de uma série de algoritmos diferentes de ordenação. O termo "poda e busca" é originário da literatura de geometria computacional (como no livro de Clarkson [22] e Meggido [72,73]). O termo "diminuição e conquista" é de Levitin [68].

Capítulo

13 Grafos



Conteúdo

a. Seja o produto de M consigo mesma (M^2) definido, para $1 \leq i, j \leq n$, como segue:

$$M^2(i,j) = M(i,1) \odot M(1,j) \oplus \cdots \oplus M(i,n)\, M(n,j),$$

onde "$\oplus$" é o operador booleano **or** e "$\odot$" é o operador booleano **and**. Dada esta definição, o que é que $M^2(i,j) = 1$ informa sobre os vértices i e j? E se $M^2(i,j) = 0$?

b. Suponha que M^4 é o produto de M^2 consigo mesma. O que representam as entradas de M^4? E as entradas de $M^5 = (M^4)(M)$? Em geral, que informação está contida na matriz M^p?

c. Suponha que é ponderado e assuma o seguinte:
   1. para $1 \leq i \leq n$, $M(i,i) = 0$.
   2. para $1 \leq i, j \leq n$, $M(i,j) = w(i,j)$ se (i,j) ? E.
   3. para $1 \leq i, j \leq n$, $M(i,j) = \infty$ se (i,j) ? E.

   Também defina M^2 para $1 \leq i, j \leq n$, como segue:

$$M^2(i,j) = \min\{M(i,j) + M(1,j), \ldots, M(i,n) + M(n,j)\}.$$

   Se $M^2(i,j) = k$, o que se pode concluir sobre a relação entre os vértices i e j?

C-13.27 Um grafo G é ***bipartido*** se seus vértices podem ser divididos em dois conjuntos S e Y sendo que toda aresta de G tem um vértice final em X e outro em Y. Projete e analise um algoritmo eficiente para determinar se um grafo não-dirigido G é bipartido (sem ter o conhecimento dos conjuntos X e Y).

C-13.28 Um método MST antigo, chamado ***Algoritmo de Baruvka***, trabalha sobre um grafo G com n vértices e m arestas com pesos distintos:

```
Seja T um subgrafo de G inicialmente contendo somente os vértices de V.
enquanto T tem menos que n − 1 arestas faça
para cada componente conexo C_i de T faça
    Procure a aresta com menor peso (v,u) de E com v C_i e u C_i.
    Adicione (v,u) em T (a menos que ele já esteja em T).
retorne T
```

Argumente porque este algoritmo está correto e porque ele executa no tempo $O(m \log n)$.

C-13.29 Seja G um grafo com n vértices e m arestas sendo que todos os pesos das arestas em G sejam inteiros no intervalo $[1,n]$. Apresente um algoritmo para procurar as árvores de cobertura mínimas de G no tempo $O(m \log^* n)$.

Projetos

P-13.1 Escreva uma classe implementando um TAD simplificado para grafos que têm os métodos relevantes para grafos não-dirigidos e que não inclui métodos de atualização, usando uma matriz de adjacência. Sua classe deve incluir um método construtor que recebe duas coleções (por exemplo, seqüências) — uma coleção V de vértices e uma coleção E de pares de vértices — e produz o grafo G que estas duas coleções representam.

P-13.2 Implemente o TAD simplificado descrito no Projeto P-13.1 usando uma lista de adjacências.

P-13.3 Implemente o TAD simplificado descrito no Projeto P-13.1 usando uma lista de arestas.

P-13.4 Estenda a classe do Projeto P-13.2 para suportar métodos de atualização.

P-13.5 Estenda a classe do Projeto P-13.2 para suportar todos os métodos do TAD grafo (incluindo métodos para arestas dirigidas).

P-13.6 Implemente um caminhamento em largura genérico usando o padrão de templates.

P-13.7 Implemente o algoritmo de ordenação topológica.

P-13.8 Implemente o algoritmo de Floyd-Warshall para o fechamento transitivo.

P-13.9 Planeje uma comparação experimental de vários caminhamentos em profundidade em relação ao algoritmo de Floyd-Warshall para determinar o fechamento transitivo de um dígrafo.

P-13.10 Implemente o algoritmo de Kruskal assumindo que os pesos das arestas sejam inteiros.

P-13.11 Implemente o algoritmo de Prim-Jarník assumindo que os pesos das arestas sejam inteiros.

P-13.12 Realize uma comparação experimental de dois dos algoritmos para MST discutidos neste capítulo (Kruskal e Prim-Jarník). Desenvolva um conjunto de experimentos para testar os tempos de execução dos algoritmos usando grafos gerados aleatoriamente.

P-13.13 Uma forma de construir um ***labirinto*** inicial com uma matriz de $n \times n$, sendo que cada célula da matriz é cercada por quatro paredes de tamanho único. Então removemos duas paredes de tamanho único para representar o início e o final. Para cada parede restante que não seja fronteira, definimos um valor randômico e criamos um grafo G, chamado de ***dual***, sendo que cada célula seja um vértice em G e exista uma aresta ligando os vértices de duas células se e somente se as células compartilharem uma parede em comum. Construímos o labirinto pela procura de uma árvore de cobertura mínima T em G e removemos todas as paredes correspondentes as arestas de T. Escreva um programa que use este algoritmo para gerar labirintos e então solucione-os. De forma resumida, seu programa deverá desenhar o labirinto e, idealmente, ele deverá visualizar a solução do mesmo.

P-13.14 Escreva um programa que crie as tabelas de roteamento para os nodos de uma rede de computadores, baseado na rota do menor caminho, onde a distância é medida pelo contador de saltos, isto é, o número de aresta em um caminho. A entrada deste problema é a informação de conectividade para todos os nodos em uma rede, como no exemplo a seguir:

241.12.31.14: 241.12.31.15 241.12.31.18 241.12.31.19

que indica três nodos de rede que estão conectados a 241.12.31.14, isto é, três nodos que estão um salto adiante. A tabela de roteamento para o nodo no endereço A é o conjunto de pares (B,C), que indicam que para transferir uma mensagem de A para B, o próximo nodo para enviar (no menor caminho entre A e B) é o C. Seu programa deverá ter como saída a tabela de roteamento para cada nodo em uma rede, dada como entrada a lista de conectividade de nodos, cada qual é a entrada com a sintaxe apresentada acima, uma por linha.