

# ANÁLISE DE ALGORITMOS

## REPOSITÓRIO DE EXERCÍCIOS

Paulo Feofiloff

4 de outubro de 2018

Este é o meu repositório de exercícios Análise de Algoritmos. A maioria dos exercícios foi extraída da primeira e segunda edições do livro de Cormen, Leiserson, Rivest & Stein. Alguns exercícios vieram dos livros de Parberry, Kleinberg & Tardos, Aho & Ullman, Manber, Aho, Hopcroft & Ullman, Brassard & Bratley, Bentley. Alguns poucos exercícios foram extraídos da competição Programming Challenges. Seguem as siglas que usaremos para identificar os livros:

CLR Cormen, Leiserson & Rivest, primeira edição [[CLR91](#)]

CLRS Cormen, Leiserson, Rivest & Stein, segunda edição [[CLRS01](#)]

Par Parberry [[Par95](#)]

KT Kleinberg & Tardos [[KT05](#)]

Mnb Manber [[Man89](#)]

AU Aho & Ullman [[AU95](#)]

AHU Aho, Hopcroft & Ullman [[AHU74](#), [AHU87](#)]

PC Programming Challenges [[Pro](#), [SR03](#)]

BB Brassard & Bratley [[BB96](#)]

Bent1988 Bentley [[Ben00](#)]

Bent2000 Bentley [[Ben88](#)]

Sed Sedgewick [[Sed98](#)]

# Sumário

<b>1 Preliminares e ferramentas</b>	<b>5</b>
1.1 Pseudocódigo . . . . .	5
1.2 Somatórios, potências e logaritmos . . . . .	6
1.3 Inversas compositionais . . . . .	7
1.4 Provas por indução matemática . . . . .	7
1.5 Valor absoluto e módulo . . . . .	8
1.6 Piso e teto . . . . .	8
1.7 Ordens assintóticas: O, Ômega e Teta . . . . .	10
1.8 Uso avançado da notação O . . . . .	14
1.9 Probabilidades . . . . .	15
<b>2 Recursão</b>	<b>16</b>
2.1 Algoritmos recursivos . . . . .	16
<b>3 Recorrências</b>	<b>19</b>
3.1 Solução exata de recorrências . . . . .	19
3.2 Recorrências “sem base” . . . . .	25
3.3 Recorrências assintóticas . . . . .	26
<b>4 Medida de desempenho de algoritmos</b>	<b>27</b>
4.1 Algoritmos iterativos . . . . .	27
4.2 Algoritmos recursivos . . . . .	29
4.3 Generalidades . . . . .	34
<b>5 Alguns problemas simples</b>	<b>36</b>
<b>6 Dividir para conquistar</b>	<b>39</b>
6.1 Busca binária . . . . .	39
6.2 Mergesort . . . . .	40
6.3 Quicksort . . . . .	41
6.4 Mediana generalizada . . . . .	45

<b>SUMÁRIO</b>	3
----------------	---

<b>7 Heap</b>	<b>48</b>
7.1 Construção de um heap . . . . .	49
7.2 Heapsort . . . . .	50
7.3 Filas de prioridades . . . . .	53
<b>8 Árvores binárias</b>	<b>55</b>
8.1 Árvores binárias de busca . . . . .	55
8.2 Árvores rubro-negras . . . . .	56
<b>9 Análise probabilística e algoritmos aleatorizados</b>	<b>57</b>
<b>10 Programação dinâmica</b>	<b>59</b>
10.1 Generalidades . . . . .	59
10.2 Multiplicação de cadeias de matrizes . . . . .	60
10.3 Problema da mochila . . . . .	62
10.4 Subsequência comum máxima . . . . .	63
10.5 Mais programação dinâmica . . . . .	65
<b>11 Estratégia gulosa</b>	<b>70</b>
11.1 Instâncias especiais do problema da mochila . . . . .	70
11.2 Intervalos disjuntos . . . . .	71
11.3 Códigos de Huffman . . . . .	73
11.4 Outros problemas . . . . .	75
<b>12 Análise amortizada</b>	<b>76</b>
12.1 Função potencial . . . . .	79
<b>13 Conjuntos disjuntos dinâmicos</b>	<b>80</b>
13.1 Union by rank . . . . .	81
13.2 Path compression . . . . .	83
13.3 Union by rank & path compression . . . . .	84
<b>14 Grafos não-dirigidos</b>	<b>87</b>
14.1 Árvores . . . . .	89
<b>15 Árvores geradoras de peso mínimo</b>	<b>90</b>
15.1 Algoritmo de Kruskal . . . . .	91
15.2 Algoritmo de Prim . . . . .	92
15.3 Outras questões . . . . .	96
<b>16 Caminhos de peso mínimo</b>	<b>98</b>
<b>17 Grafos dirigidos</b>	<b>99</b>

17.1 Digrafos acíclicos e componentes fortes . . . . .	99
<b>18 Ordenação em tempo linear</b>	<b>101</b>
18.1 Algoritmos lineares de ordenação . . . . .	101
18.2 Cota inferior para o problema da ordenação . . . . .	103
<b>19 Problemas completos em NP</b>	<b>105</b>
19.1 Questões mais abstratas . . . . .	108
<b>Bibliografia</b>	<b>111</b>
<b>Índice Remissivo</b>	<b>112</b>

# Capítulo 1

## Preliminares e ferramentas

“Os logaritmos permanecem importantes para muitas aplicações científicas e técnicas.”  
— pérola do jornal *O Estado de São Paulo* (agosto de 1998)

### 1.1 Pseudocódigo

**Exr 1.1** Reescreva a função abaixo em linguagem C.

```
FUNC ( $A, n$ )
1  para  $j \leftarrow 2$  até  $n$  faça
2     $ch \leftarrow A[j]$ 
3     $i \leftarrow j - 1$ 
4    enquanto  $i \geq 1$  e  $A[i] > ch$  faça
5       $A[i + 1] \leftarrow A[i]$ 
6       $i \leftarrow i - 1$ 
7     $A[i + 1] \leftarrow ch$ 
```

**Exr 1.2** Reescreva a função abaixo em pseudocódigo (notação CLR):

```
int funcao (int n, int v[]) {
    int i, j;
    i = 0;
    while (i < n) {
        if (v[i] != 0) i++;
        else {
            for (j = i+1; j < n; j++) v[j-1] = v[j];
            n--;
        }
    }
    return n;
}
```

## 1.2 Somatórios, potências e logaritmos

Veja CLRS ap. A.1. Notação:  $\lg n := \log_2 n$ ,  $\lg n^2 := \lg(n^2)$ ,  $\lg^2 n := (\lg n)^2$  e  $\lg \lg n := \lg(\lg n)$ .

**Exr 1.3** Simplifique a expressão  $2^{\lg^2 n}$ . Simplifique a expressão  $n^{1/\lg n}$ .

**Exr 1.4** Mostre que  $n^{1+(\lg \lg n)/\lg n} = n \lg n$ .

**Exr 1.5** É verdade que  $2^{(n^k)} = (2^n)^k$ ? O que significa a expressão  $2^{n^k}$ ?

**Exr 1.6** Mostre que  $3^{\log_4 n} = n^{\log_4 3}$

**Exr 1.7** Mostre que para quaisquer  $n > 0$  e  $k > 0$  tem-se  $n^{\log k} = k^{\log n}$ .

**Exr 1.8** Dê o valor da soma  $\sum_{j=2}^{n-1} j$  e prove que sua resposta está certa.

**Exr 1.9** Quanto vale  $S$  no fim do seguinte fragmento de código?

```

1   S ← 0
2   para i ← 1 até n faça
3       para j ← 1 até i faça
4           S ← S + 1

```

Escreva um algoritmo mais eficiente que tenha o mesmo efeito.

**Exr 1.10** Quanto vale  $S$  no fim do algoritmo?

```

1   S ← 0
2   para i ← 2 até n - 2 faça
3       para j ← i até n faça
4           S ← S + 1

```

Escreva um algoritmo mais eficiente que tenha o mesmo efeito.

**Exr 1.11** Quanto vale  $S$  no fim do seguinte fragmento de código?

```

1   S ← 0
2   i ← n
3   enquanto i > 0 faça
4       para j ← 1 até i faça
5           S ← S + 1
6       i ← ⌊i/2⌋

```

**Exr 1.12** Quanto vale  $S$  no fim do seguinte fragmento de código?

```

1    $S \leftarrow 0$ 
2   para  $i \leftarrow 1$  até  $n$  faça
3        $j \leftarrow i$ 
4       enquanto  $j > 0$  faça
5            $S \leftarrow S + 1$ 
6            $j \leftarrow \lfloor j/2 \rfloor$ 

```

**Exr 1.13** Seja  $x$  um número real diferente de 1. Mostre que  $x^0 + x^1 + x^2 + \dots + x^n = \frac{x^{n+1} - 1}{x - 1}$ .

**Exr 1.14** Seja  $x$  um número real tal que  $|x| < 1$ . Mostre que  $x^0 + x^1 + x^2 + x^3 + \dots = 1/(1-x)$ . Mostre que  $1x^0 + 2x^1 + 3x^2 + \dots = 1/(1-x)^2$ . Mostre que  $1x^1 + 2x^2 + 3x^3 + \dots = x/(1-x)^2$ .

**Exr 1.15** Dê uma fórmula fechada para a soma  $2^2 + 2^4 + 2^6 + \dots + 2^{2k}$ .

**Exr 1.16** Prove que  $1^3 + 2^3 + 3^3 + \dots + (n-1)^3 + n^3 = \Theta(n^4)$ .

**Exr 1.17** Prove que  $1 + n + n^2 + n^3 + \dots + n^8 + n^9 = \Theta(n^9)$ .

### 1.3 Inversas composticionais

Uma função  $g$  é *inversa compostional* de uma função  $f$  se  $f(g(n)) = g(f(n)) = n$ . Em outras palavras,  $y = f(x)$  se e só se  $x = g(y)$ . (Estou supondo, ao longo desta seção, que todas as funções estão definidas sobre o conjunto dos números reais positivos.)

Leia CLRS sec. 3.2.

**Exr 1.18** Dê as inversas composticionais das funções<sup>1</sup>  $f(n) = n^3$ ,  $f(n) = n^{1/5} \equiv \sqrt[5]{n}$  e  $f(n) = 1/n$ .

**Exr 1.19** Dê as inversas composticionais das funções  $5^n$ ,  $\log_3 n$  e  $\lg n$ .

**Exr 1.20** [CLRS 3.2] Qual a relação entre  $\log_8 n$  e  $\log_2 n$ ?<sup>2</sup>

**Exr 1.21** Dê as inversas composticionais das funções  $\log_3 \log_3 n \equiv \log_3(\log_3 n)$  e  $\log_3^2 n \equiv (\log_3 n)^2$ .

**Exr 1.22** Desenhe gráficos das funções  $\log_2 n$  e  $2^n$  definidas sobre os inteiros positivos.

### 1.4 Provas por indução matemática

Indução matemática mirim: Suponha  $P(n)$  e prove  $P(n+1)$ . Indução matemática de gente grande: Suponha  $P(m)$  para todo  $m < n$  e prove  $P(n)$ .

---

<sup>1</sup> Em texto sem formatação (numa mensagem de e-mail, por exemplo), escreva “ $n^{1/5}$ ” no lugar de “ $n^{1/5}$ ”.

<sup>2</sup> Em texto sem formatação, escreva “ $\log_8 n$ ” no lugar de “ $\log_8 n$ ”.

**Exr 1.23** Prove que  $1 + 2 + 2^2 + \cdots + 2^n = 2^{n+1} - 1$  para todo número natural  $n$ .

**Exr 1.24** Prove que para qualquer número natural não-nulo  $n$  tem-se  $1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{1}{6}n(n+1)(2n+1)$ .

**Exr 1.25** Veja exercícios 11.23 e 8.2.

**Exr 1.26** Prove que  $2^2 + 2^4 + 2^6 + \cdots + 2^{2k} = (4/3)(2^{2k} - 1)$ .

**Exr 1.27** Prove que  $n \leq 2^{n/2}$ . Prove que  $\lg n = n/2$ .

**Exr 1.28** Prove que  $\lg n \leq \sqrt{n}$ .

## 1.5 Valor absoluto e módulo

**Exr 1.29** Suponha que  $a \geq b$  e  $x \geq y$ . Mostre que  $|x - a| + |y - b| \leq |x - b| + |y - a|$ .

## 1.6 Piso e teto

**Exr 1.30** Seja  $x$  um número real. Diga, da maneira mais precisa que puder, o que significam as expressões “ $\lfloor x \rfloor$ ” e “ $\lceil x \rceil$ ”.<sup>3</sup>

**Exr 1.31** Prove ou desprove a seguinte proposição: para todo par  $x, y$  de números racionais,  $x \leq y$  se e somente se  $\lfloor x \rfloor \leq \lfloor y \rfloor$ .

**Exr 1.32** Suponha que  $n/5 < m/5$ . É verdade que  $\lfloor n/5 \rfloor < \lfloor m/5 \rfloor$ ?

**Exr 1.33** Suponha que  $k$  é inteiro e  $k > n/5$ . É verdade que  $k \geq 1 + \lfloor n/5 \rfloor$ ?

**Exr 1.34** Sejam  $n, a$  e  $b$  números inteiros positivos. Suponha que  $b > 4$ . É verdade que  $\lfloor n/a - 1/b \rfloor = \lfloor n/a \rfloor$ ?

**Exr 1.35** É verdade que  $\lfloor x \rfloor + \lfloor y \rfloor = \lfloor x + y \rfloor$  para quaisquer  $x$  e  $y$ ?

**Exr 1.36** É verdade que  $\lfloor x \rfloor + 1 = \lceil x + 1 \rceil$  para qualquer  $x$ ?

**Exr 1.37** Mostre que  $\lfloor x \rfloor + \lfloor y \rfloor \leq \lfloor x + y \rfloor$ , com igualdade se e somente se  $x + y - 1 < \lfloor x \rfloor + \lfloor y \rfloor$ . Encontre uma fórmula análoga para  $\lceil \cdot \rceil$ .

**Exr 1.38** Suponha que  $c$  é um número inteiro e  $x$  um número racional. É verdade que  $\lceil cx \rceil = c\lceil x \rceil$ ?

**Exr 1.39** Use a notação  $\lfloor \cdot \rfloor$  para representar o resto da divisão de  $n$  por 7.

---

<sup>3</sup> Em texto sem formatação, escreva “`piso(x)`” ou “`floor(x)`” no lugar de “[ $x$ ]”.

**Exr 1.40** Simplifique a expressão  $\lfloor \frac{n+m}{2} \rfloor + \lfloor \frac{n-m+1}{2} \rfloor$  supondo que  $m$  e  $n$  são números inteiros. Repita com  $\lceil \cdot \rceil$  no lugar de  $\lfloor \cdot \rfloor$ .

**Exr 1.41** [CLRS 3.2] Mostre que para qualquer número real  $x$  tem-se  $x-1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x+1$ .

**Exr 1.42** Desenhe gráficos das funções  $\lfloor x \rfloor$  e  $\lceil x \rceil$ .

**Exr 1.43** Mostre que, para qualquer número inteiro  $n \geq 1$ ,

$$\frac{n-1}{2} \leq \left\lfloor \frac{n}{2} \right\rfloor \leq \frac{n}{2} \quad \text{e} \quad \frac{n}{2} \leq \left\lceil \frac{n}{2} \right\rceil \leq \frac{n+1}{2}.$$

Isso faz sentido se  $n$  não for inteiro?

**Exr 1.44** Seja  $n$  um inteiro positivo. Prove que  $\lfloor \lfloor n/2 \rfloor / 2 \rfloor = \lfloor n/4 \rfloor$ .

**Exr 1.45** Prove que para quaisquer inteiros positivos  $a$  e  $b$  tem-se  $\lfloor \lfloor n/a \rfloor / b \rfloor = \left\lfloor \frac{n}{ab} \right\rfloor$ .

**Exr 1.46** É verdade que  $\lceil 2\lceil 2n/3 \rceil / 3 \rceil = \lceil 4n/9 \rceil$ ? É verdade que  $\lfloor 2\lfloor 2n/3 \rfloor / 3 \rfloor = \lfloor 4n/9 \rfloor$ ?

**Exr 1.47** [CLRS 3.2] Suponha que  $n, a$  e  $b$  são inteiros positivos. Mostre que

$$\left\lfloor \frac{\lfloor n/a \rfloor}{b} \right\rfloor = \left\lfloor \frac{n}{ab} \right\rfloor.$$

É verdade que  $\lceil 2\lceil 2n/3 \rceil / 3 \rceil = \lceil 4n/9 \rceil$ ? É verdade que  $\lfloor 2\lfloor 2n/3 \rfloor / 3 \rfloor = \lfloor 4n/9 \rfloor$ ?

**Exr 1.48** Mostre que para todo número natural não-nulo  $n$  tem-se  $\lceil n/9 \rceil + \lfloor 8n/9 \rfloor = 1$ .

**Exr 1.49** Seja  $k$  um número natural não-nulo. É verdade que  $\lceil n/k \rceil + \lfloor (k-1)n/k \rfloor = 1$  para todo natural não-nulo  $n$ ?

**Exr 1.50** Seja  $n_0$  um inteiro positivo e  $n_1, n_2, n_3, \dots$  a sequência definida por

$$n_i = \left\lceil \frac{2n_{i-1}}{3} \right\rceil.$$

É verdade que  $n_i = \lceil 2^i n_0 / 3^i \rceil$ ? Prove essa afirmação ou dê uma boa cota superior para  $n_i$ .

**Exr 1.51** Seja  $x$  um número real,  $m$  um número e  $n$  um inteiro positivo. Mostre que  $\lfloor \frac{x+m}{n} \rfloor = \lfloor \frac{\lfloor x \rfloor + m}{n} \rfloor$ .

**Exr 1.52** [Converte teto em piso] Sejam  $a$  e  $b$  números inteiros positivos. Mostre que  $\lceil \frac{a}{b} \rceil = \lfloor \frac{a+b-1}{b} \rfloor$ .

**Exr 1.53** Seja  $n$  um inteiro estritamente positivo. Diga, em termos de potências, sem mencionar logaritmos, o que significa a expressão “ $\lceil \lg n \rceil$ ”.

**Exr 1.54** Mostre que  $\lfloor \lg n \rfloor$  é o maior inteiro  $k$  tal que  $2^k \leq n$ . Mostre que  $\lceil \lg n \rceil$  é o menor inteiro  $k$  tal que  $2^k \geq n$ .

**Exr 1.55** Escreva uma função que calcule  $\lfloor \lg n \rfloor$ . Escreva sua função em pseudocódigo. Escreva uma versão iterativa e uma recursiva (veja capítulo 2).

**Exr 1.56** Escreva um algoritmo que calcule  $\lceil \lg n \rceil$ .

**Exr 1.57** Prove que para qualquer inteiro  $n$  maior que 1 tem-se  $\lfloor \lg \lfloor \frac{n}{2} \rfloor \rfloor = \lfloor \lg \frac{n}{2} \rfloor$ .

**Exr 1.58** É verdade que  $\lfloor \lg n \rfloor \geq \lg(n - 1)$  para todo inteiro  $n \geq 2$ ? É verdade que  $\lceil \lg n \rceil \leq \lg(n + 1)$  para todo inteiro  $n \geq 1$ ?

**Exr 1.59** [Importante] É verdade que  $\lceil \lg(n + 1) \rceil = \lfloor \lg n \rfloor + 1$  para todo inteiro  $n > 1$ ? É verdade que  $\lfloor \lg(n - 1) \rfloor = \lceil \lg n \rceil - 1$  para todo inteiro  $n > 1$ ?

**Exr 1.60** Mostre que para todo número real  $x \geq 1$  tem-se  $\lfloor \lg x \rfloor \leq \lg \lfloor x \rfloor \leq \lg x \leq \lceil \lg x \rceil \leq \lceil \lg x \rceil$ .

**Exr 1.61** [Soma de teto de log] Prove que  $\sum_{k=2}^n \lceil \lg k \rceil = n \lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + 1$ .

**Exr 1.62** Para cada número racional positivo  $x$ , seja  $f(x)$  o número  $\sqrt{x}$ . É verdade que  $\lfloor f(x) \rfloor \leq f(\lfloor x \rfloor) \leq f(x) \leq f(\lceil x \rceil) \leq \lceil f(x) \rceil$ .

**Exr 1.63** Para cada número racional positivo  $x$ , seja  $f(x)$  o número  $2^x$ . É verdade que  $\lfloor f(x) \rfloor \leq f(\lfloor x \rfloor) \leq f(x) \leq f(\lceil x \rceil) \leq \lceil f(x) \rceil$ .

**Exr 1.64** [Maioria simples] Considere a seguinte proposição: “O candidato estará eleito se obtiver metade mais um dos votos válidos”. Considere esta outra proposição: “O candidato estará eleito se obtiver mais da metade dos votos válidos”. Compare e critique as duas proposições.

## 1.7 Ordens assintóticas: O, Ômega e Teta

Leia CLRS seção 3.1. Veja também seções 3.4 e 3.5 de Aho e Ullman [AU95], Veja também seções 2.3 e 2.4 de R. Sedgewick [Sed98].

Ao ver uma expressão como  $n + 10$  ou  $n^2 + 1$ , a maioria das pessoas pensa automaticamente em valores pequenos de  $n$ , valores próximos de 0. A análise de algoritmos faz o contrário: ignora os valores pequenos e concentra-se nos valores enormes de  $n$ .

Escrevemos  $f = O(g)$  (diga “ $f$  é O-grande de  $g$ ”) se existe uma constante positiva  $c$  tal que  $0 \leq f(n) \leq cg(n)$  para todo  $n$  suficientemente grande. Mais formalmente: dadas funções  $f$  e  $g$ , dizemos que  $f = O(g)$  se existem constantes  $c$  e  $n_0$  tais que  $0 \leq f(n) \leq cg(n)$  para todo  $n \geq n_0$ .

Uma função  $f$  é *assintoticamente não-negativa* se existe  $n_1$  tal que  $f(n) \geq 0$  para todo  $n \geq n_1$ . Com esse conceito, podemos dar a seguinte definição equivalente de  $O()$ : dadas funções assintoticamente não-negativas  $f$  e  $g$ , dizemos que  $f = O(g)$  se existem constantes positivas  $c$  e  $n_0$  tais que  $f(n) \leq cg(n)$  para todo  $n \geq n_0$ .

Notação: Em lugar de escrever “ $f = O(g)$ ” podemos dizer “ $f$  é  $O(g)$ ” ou “ $f$  está na ordem de  $g$ ” ou “ $f$  é da ordem de  $g$ ”. Podemos também escrever “ $f \in O(g)$ ”, uma vez que  $O(g)$  é um conjunto de funções.

Escrevemos  $f = \Omega(g)$  (diga “ $f$  é Ômega grande de  $g$ ”) se existe uma constante positiva  $c$  tal que  $0 \leq c g(n) \leq f(n)$  para todo  $n$  suficientemente grande. Dizemos que  $f = \Theta(g)$  se  $f = O(g)$  e  $f = \Omega(g)$ .

**Exr 1.65** Discuta a seguinte afirmação: “A função  $2n^2 + 100n - 1$  está em  $O(n^2)$  com  $c = 4$ ”, Discuta a seguinte afirmação: “A função  $2n^2 + 100n - 1$  está em  $O(n^2)$  para  $n \geq 100$ ”.

**Exr 1.66** Discuta a seguinte afirmação: “O consumo de tempo do algoritmo X é pelo menos  $O(n \lg n)$ ”. Discuta a seguinte afirmação: “A complexidade do algoritmo X é pelo menos  $O(n \lg n)$ ”.

**Exr 1.67** Seja  $f$  a função definida pela seguinte recorrência:  $f(1) = 1$  e  $f(n) = f(n - 1) + 2n$  para todo número natural  $n \geq 2$ . Quero provar que  $f(n) = O(n^2)$ . Discuta a seguinte prova:

A prova é por indução em  $n$ . Se  $n = 1$  então temos  $f(n) = f(1) = 1 \leq n^2 = O(n^2)$ . Suponha agora que  $n > 1$ . Temos então  $f(n) = f(n - 1) + 2n \leq 2(n - 1)^2 + 2n = 2n^2 - 4n + 2 + 2n = 2n^2 - 2n + 2 \leq 2n^2 = O(n^2)$ .

**Exr 1.68** Você quer tomar um empréstimo para comprar uma casa. Você quer escolher entre dois bancos, A e B. No banco A, a prestação do mês  $n$  é  $n^2$ . No banco B, a prestação do mês  $4n + 8$ . Qual você prefere?

**Exr 1.69** Exiba três pares  $(c, n_0)$  tais que  $100n^2 \leq c \frac{1}{100}n^3$  para todo  $n \geq n_0$ .

**Exr 1.70** Seja  $f$  a função definida por  $f(n) = 3n^2 + 7n - 8$  para todo inteiro não-negativo  $n$ . Mostre que  $f(n) = O(n^2)$ .

**Exr 1.71** Mostre que  $100n = O(2n)$ , que  $n + 100 = O(n)$ , e que  $100n = O(2n^2 + n)$ .

**Exr 1.72** Prove que  $n^2 + 999n + 9999 = O(n^2)$ .

**Exr 1.73** É verdade que  $\frac{1}{100}n^2 - 999n - 9999 = O(n)$ ?

**Exr 1.74** É verdade que  $10\sqrt{n} + 10 = O(n)$ ? É verdade que  $\frac{1}{10}n - 100 = O(\sqrt{n})$ ?

**Exr 1.75** É verdade que  $\lg n^{10} = O(\lg n)$ ?

**Exr 1.76** Mostre que  $\lg n = O(\log_3 n)$  e  $\log_3 n = O(\lg n)$ .

**Exr 1.77** Prove que  $\frac{1}{2}n(n - 1) = O(n^2)$ .

**Exr 1.78** Prove que  $\frac{1}{100}n^3 = O(n^2)$ .

**Exr 1.79** Prove que  $n = O(2^n)$ .

**Exr 1.80** Prove que  $4 \log n = O(n)$ .

**Exr 1.81** Prove que  $n^2 = O(\frac{1}{10} 2^n)$ .

**Exr 1.82** É verdade que  $n^5 = O(2^n)$ ? É verdade que  $2^{n/2} - 1000n^{99} = O(n^9)$ ?

**Exr 1.83** É verdade que  $n^9 = O((\frac{3}{2})^n)$ ? É verdade que  $(\frac{11}{10})^n = O(n^9)$ ?

**Exr 1.84** Suponha que  $f(n) = n^2$  para  $n = 1, \dots, 1000$  e  $f(n) = n$  para  $n = 1001, \dots$  É verdade que  $f(n) = O(n)$ ?

**Exr 1.85** Prove que  $\lg n = O(n)$ .

**Exr 1.86** Prove que  $\lceil \lg n \rceil = O(n)$ .

**Exr 1.87** Mostre que  $5 + \frac{1}{n}$  é  $O(n^0)$ . Mostre que  $5n + \frac{1}{n} = O(n)$ .

**Exr 1.88** Prove que  $n^2 - 999n - 9999 = \Omega(n^2)$ .

**Exr 1.89** É verdade que  $\frac{1}{100}n^2 + 999n + 9999 = \Omega(n^3)$ ? Justifique.

**Exr 1.90** [CLRS 8.1-2] Mostre que  $\lg(n!) = \Omega(n \lg n)$ .

**Exr 1.91** Prove que  $\frac{1}{2}n(n+1) = \Omega(n^2)$ .

**Exr 1.92** Prove que  $n = \Omega(\lg n)$ .

**Exr 1.93** Prove que  $\lg n = O(n^{1/2})$ . Prove que  $n^{1/2}$  não é  $O(\lg n)$ .

**Exr 1.94** É verdade que  $n^{1/3} = O(\lg n)$ ?

**Exr 1.95** Prove que  $n^{1/2} = \Omega(\lg n)$ .

**Exr 1.96** Prove que  $(\frac{11}{10})^n = \Omega(n^9)$ ?

**Exr 1.97** Prove que  $\lfloor n/3 \rfloor = O(n)$ . É verdade que  $n = O(\lfloor n/3 \rfloor)$ ?

**Exr 1.98** Prove que  $\lfloor n/3 \rfloor = \Omega(n)$ .

**Exr 1.99** É verdade que  $\lceil n/5 \rceil = O(n)$ ? É verdade que  $\lfloor n/5 \rfloor = \Omega(n)$ ?

**Exr 1.100** Sejam  $T$  e  $f$  duas funções que levam números inteiros em números reais. (a) O que significa a afirmação “ $T(n)$  é  $O(f(n))$ ”? (b) É verdade que  $20n^3 + 10n \lg n + 5$  é  $O(n^3)$ ? Justifique. (c) É verdade que  $\frac{1}{2}n^2$  é  $O(n)$ ? Justifique. (d) O que significa a afirmação “ $T(n)$  é  $\Omega(f(n))$ ”? (e) O que significa a expressão “ $T(n) = n + \Omega(n \lg n)$ ”?

**Exr 1.101** Prove que  $10 \lg n + 100 = O(\lg n)$ .

**Exr 1.102** Prove que  $\lfloor \lg n \rfloor = O(\lg n)$ ? É verdade que  $\lceil \lg n \rceil = O(\lg n)$ ?

**Exr 1.103** Mostre, da maneira mais direta que puder, que

$$\frac{1}{100}n \lg n - 100n$$

não é  $O(n)$ .

**Exr 1.104** [CLRS 3.1-2, simplificado] Mostre que  $(n + 10)^5 = O(n^5)$ .

**Exr 1.105** [CLRS 3.1-2] Seja  $k$  um inteiro positivo. Mostre que  $(n + 1)^k = \Theta(n^k)$ .

**Exr 1.106** Mostre que  $\sqrt{n + 100} = \Theta(\sqrt{n})$

**Exr 1.107** Encontre um inteiro positivo  $k$  tal que  $2^n$  é  $O(n^k)$ .

**Exr 1.108** [CLR 2.1-4] É verdade que  $2^n = O(2^{n/2})$ ? É verdade que  $2^{2n} = O(2^n)$ ?

**Exr 1.109** É verdade que  $2^{n+1} = \Theta(2^n)$ ? É verdade que  $3^n = \Theta(2^n)$ ? É verdade que  $2^{2n} = \Theta(2^n)$ ?

**Exr 1.110** É verdade que  $2^{\lg n} = O(n)$ ? É verdade que  $2^{\log_3 n} = O(n)$ ?

**Exr 1.111** É verdade que  $2^{\log_3 n} = O(2^{\log_{10} n})$ ?

**Exr 1.112** É verdade que  $n$  é  $O(\log^2 n)$ ?

**Exr 1.113** É verdade que  $\lg n = \Omega(n)$ ? É verdade que  $2^{\lg^2 n} = \Theta(n^{\lg n})$ ?

**Exr 1.114** É verdade que  $\sqrt{n} = O(\log^2 n)$ ?

**Exr 1.115** Discuta a seguinte afirmação: “ $n$  é  $O(2^n)$  para  $n \geq 4$ ”?

**Exr 1.116** Critique a seguinte afirmação: “ $f(n)$  é  $O(n^2)$  para  $n \geq 100$ ”?

**Exr 1.117** Discuta a seguinte afirmação: “ $n^2 + n = O(n^2)$  para  $c = 2$  e  $N = 1$ ”.

**Exr 1.118** Sejam  $f$  e  $g$  funções que levam números racionais positivos em números reais positivos. Suponha que  $f$  não é  $O(g)$ . É verdade então que  $g = O(f)$ ?

**Exr 1.119** [CLR 2-4 simplificado] É verdade que  $1/2 + 2/2^2 + 3/2^3 + \dots + n/2^n = O(1)$ ?

**Exr 1.120** Sejam  $f$  e  $g$  duas funções que levam números positivos em números positivos. Reformula a afirmação “ $f$  não é  $O(g)$ ” sem usar a expressão “não existe”.

**Exr 1.121** Mostre que  $m\lceil n/m \rceil = \Theta(m + n)$ . É verdade de  $n\lceil n/m \rceil = \Theta(m)$ ?

**Exr 1.122** Mostre que  $n \lg n = \Omega(n)$ . Mostre que  $n^2 - 100n - 200\sqrt{n} = \Theta(n^2)$ .

**Exr 1.123** Mostre que  $n \lg n - \lceil 2n/3 \rceil - \lg n + 4 = \Omega(2n \lg n)$ .

**Exr 1.124** Seja  $f$  a função definida assim para todo inteiro positivo  $k$ :

$$f(4k) = 0, \quad f(4k+1) = 1, \quad f(4k+2) = 0, \quad f(4k+3) = -1.$$

A função  $n^{2+f(n)}$  é  $O(n^2)$ ? É  $\Omega(n^2)$ ? Justifique.

**Exr 1.125** Suponha que  $T(n) = O(1)$ . Mostre que existe  $c$  tal que  $T(n) \leq c$  para todo  $n \geq 1$ .

**Exr 1.126** Sejam  $T$  e  $f$  funções tais  $f(n) > 0$  para  $n \geq 1$  e  $T(n) = O(f(n))$ . Mostre que existe  $c$  tal que  $T(n) \leq c f(n)$  para todo  $n \geq 1$ .

## 1.8 Uso avançado da notação O

Algumas expressões que se usam com notação O:

- “O( $f$ ) = O( $g$ )” significa “qualquer função  $f'$  em O( $f$ ) também está em O( $g$ )” (em outras palavras, “se  $f' = O(f)$  então  $f = O(g)$ ”) (seria, portanto, mais correto escrever “O( $f$ ) ⊆ O( $g$ )”);
- “ $g(n) = f(n) + O(n)$ ” significa “existe uma função  $h(n)$  em O( $n$ ) tal que  $g(n) = f(n) + h(n)$ ”;
- “O( $f$ ) + O( $g$ ) = O( $f+g$ )” significa “para qualquer  $f'$  em O( $f$ ) e qualquer  $g'$  em O( $g$ ), a função  $f' + g'$  está em O( $f+g$ )”.

**Exr 1.127** Qual é o significado exato da afirmação “ $g(n) = f(n) + O(n)$ ”?

**Exr 1.128** Suponha que a função  $f$  leva números inteiros em números inteiros. É verdade que  $2f(n) + 3 = O(f(n))$ ?

**Exr 1.129** O que significa a afirmação “ $O(n^2) + O(1) = O(n^2)$ ”? Ela é verdadeira? Justifique.

**Exr 1.130** O que significa a afirmação “ $O(n^2 + 9n) = O(n^2)$ ”? Ela é verdadeira?

**Exr 1.131** O que significa a afirmação “ $O(f) + O(g) = O(f + g)$ ”? Ela é verdadeira?

**Exr 1.132** Mostre que  $nO(f(n)) \subseteq O(nf(n))$ . Mostre que  $O(nf(n)) \subseteq nO(f(n))$ .

**Exr 1.133** Suponha que  $f = O(g)$ . Prove que  $g = \Omega(f)$ .

**Exr 1.134** Suponha que  $f(n) = \Omega(g(n))$  e  $g(n) = O(f(n))$ . É verdade que  $g(n) = O(n)$ ?

**Exr 1.135** Suponha que  $f(n) = \Omega(g(n))$  e  $g(n) = O(f(n))$ . É verdade que  $g(n) = \Omega(n)$ ?

**Exr 1.136** Suponha que  $f(n) = \Omega(g(n))$ . É verdade que  $2^{f(n)} = O(2^{g(n)})$ ?

**Exr 1.137** Suponha que  $\log f = O(\log g)$ . É verdade que  $f = O(g)$ ?

**Exr 1.138** Prove que  $f = \Omega(g)$  se e só se  $g = O(f)$ .

**Exr 1.139** [Transitividade, CLRS p.49] Suponha que as funções  $f, g$  e  $h$  levam números inteiros em números inteiros. Mostre que

$$\text{se } f = O(g) \text{ e } g = O(h) \text{ então } f = O(h).$$

(Por exemplo, como  $2n^2 + n = O(n^2)$ , podemos dizer que toda função em  $O(2n^2 + n)$  também está em  $O(n^2)$ .)

**Exr 1.140** [CLR 2-4 simplificado] É verdade que se  $f(n) = O(g(n))$  então também  $g(n) = O(f(n))$ ? É verdade que  $f(n) + g(n) = O(\min\{f(n), g(n)\})$ ? É verdade que  $f(n) = O(f(n)^2)$ ? É verdade que  $f(n) = O(f(n/2))$ ?

**Exr 1.141** É verdade que  $O(2^{\log_3 n}) = O(2^{\log_{10} n})$ ?

**Exr 1.142** [CLR 2-3 simplificado] Coloque as seguintes funções em ordem crescente de taxa de crescimento, ou seja, encontre uma permutação  $f_1, f_2, f_3, \dots$  das funções tal que  $f_1 = O(f_2)$ ,  $f_2 = O(f_3)$ , etc.

$$\begin{array}{ccccc} \lg \lg n & n^2 & n! & n^3 & (3/2)^n \\ 2^n & 1 & \sqrt{n} & \sqrt{\lg n} & n \lg n \\ \log_{10} n & n2^n & \lg^2 n & & \end{array}$$

**Exr 1.143** Suponha que as funções  $f$  e  $g$  levam números inteiros positivos em números reais. O que significa a afirmação " $O(f)O(g) = O(fg)$ "? Ela é verdadeira?

**Exr 1.144** Suponha que  $F(n) = O(n^2)$  e  $G(n) = O(\lg n)$ . Prove que  $F(n)G(n) = O(n^2 \lg n)$ .

**Exr 1.145** [Regra da Soma] Suponha que as funções  $T_1, T_2, f_1$  e  $f_2$  levam inteiros positivos em inteiros positivos. Suponha  $T_1(n) = O(f_1(n))$  e  $T_2(n) = O(f_2(n))$ . Mostre que se  $f_1(n) = O(f_2(n))$  então  $T_1 + T_2 = O(f_2)$ .

**Exr 1.146** Mostre que  $O(1) + O(1) + O(1) = O(1)$ . Mostre que  $\sum_{i=1}^n O(1) = O(n)$ .

## 1.9 Probabilidades

Veja capítulo 9.

# Capítulo 2

## Recursão

A recursão é uma técnica que resolve uma instância de um problema a partir das soluções de instâncias menores do mesmo problema.

A idéia de qualquer algoritmo recursivo é simples: se a instância é pequena, resolva-a diretamente, como puder; se a instância é grande, *reduza-a a uma instância menor*. Portanto, recursão é essencialmente o mesmo que indução matemática.

### 2.1 Algoritmos recursivos

**Exr 2.1** [Máximo de um vetor] Escreva uma função recursiva que encontre o valor de um elemento máximo<sup>1</sup> de um vetor de  $A[1..n]$ . Para que valores de  $n$  o problema faz sentido? (Use pseudocódigo ou linguagem C.)

**Exr 2.2** Escreva um algoritmo recursivo que calcule a soma dos elementos de um vetor. Use pseudocódigo CLR.

**Exr 2.3** Problema: dado um vetor  $A[1..n]$  de números inteiros, calcular o produto dos elementos não-nulos do vetor. Escreva um algoritmo recursivo que resolva o problema. (Declare claramente as eventuais hipóteses que seu algoritmo faz sobre os dados.)

**Exr 2.4** Escreva um algoritmo recursivo eficiente que calcule  $k^n$  para  $k$  e  $n$  inteiros positivos.

**Exr 2.5** Critique o seguinte algoritmo ( $n$  é inteiro não-negativo):

```
XIS (n)
1   se n = 0
2     então devolva 0
3     senão devolva XIS (└n/3┘ + 1)
```

**Exr 2.6** Escreva funções recursivas que calculem  $\lfloor \lg n \rfloor$  e  $\lceil \lg n \rceil$ . (Veja 1.55 e 1.56.)

---

<sup>1</sup> Eu não disse “do maior elemento” pois o vetor pode ter mais de um máximo.

**Exr 2.7** [Busca linear, CLRS 2.1-3] Escreva um algoritmo recursivo que verifique se  $v$  é elemento de um vetor  $A[p..r]$ . (Para que valores de  $p$  e  $r$  o problema faz sentido?) O algoritmo deve devolver  $i$  tal que  $A[i] = v$  ou NIL se tal  $i$  não existe.<sup>2</sup>

**Exr 2.8** [Busca em vetor ordenado] Dado um vetor crescente  $A[p..r]$  e um número  $v$ , verificar se  $v$  é elemento do vetor. Escreva um algoritmo recursivo que resolva o problema. Faça com que o algoritmo devolva  $j$  tal que

$$A[j] \leq v < A[j+1]$$

(isso é mais simples que devolver  $i$  tal que  $A[i] = v$ ). Quais os valores possíveis de  $j$ ? Escreva duas versões: uma de busca linear e outra de busca binária.

**Exr 2.9** O algoritmo abaixo recebe um número  $v$  e um vetor crescente  $A[p..r]$  com  $p \leq r$ . Se  $v = A[j]$  para algum  $j$  em  $p..r$ , o algoritmo promete devolver  $j$ ; senão, promete devolver NIL.

EXERCÍCIO ( $v, A, p, r$ )

- 1 se  $p = r$
- 2 então se  $A[p] = v$  então devolva  $p$  senão devolva NIL
- 3 senão  $q \leftarrow \lfloor (p+r)/2 \rfloor$
- 4 se  $A[q] \leq v$
- 5 então devolva EXERCÍCIO ( $v, A, q, r$ )
- 6 senão devolva EXERCÍCIO ( $v, A, p, q$ )

O algoritmo faz o que promete?

**Exr 2.10** Suponha que um vetor  $A[1..n]$  tem exatamente  $m$  elementos não-nulos e que os índices desses elementos são  $i_1 < i_2 < \dots < i_m$ . Escreva um algoritmo recursivo eficiente que tenha o mesmo efeito que a sequência de atribuições

$$A[1] \leftarrow A[i_1], \quad A[2] \leftarrow A[i_2], \quad \dots, \quad A[m] \leftarrow A[i_m].$$

O seu algoritmo deve receber apenas  $n$  e  $A[1..n]$  e devolver  $m$ .

**Exr 2.11** Escreva um algoritmo recursivo que receba um inteiro não-negativo  $n$  e devolva as coordenadas  $(i, j)$  de  $n$  na tabela abaixo. (Use notação CLRS.)

	0	1	2	3	4	...	$j$
0	0	2	5	9	14		
1	1	4	8	13			
2	3	7	12				
3	6	11					
4	10						
$\vdots$						$\vdots$	
$i$						$\dots$	$n$

<sup>2</sup> Acho que o algoritmo ficaria mais bonito se devolvesse  $p - 1$  (ou  $r + 1$ ) quando  $v$  não está em  $A[p..r]$ .

**Exr 2.12** Escreva um algoritmo que remova todos os elementos nulos de um vetor  $A[p..r]$  mas preserve a ordem relativa dos demais elementos. Faça duas versões: uma iterativa e uma recursiva.

Repita o problema depois de substituir o vetor por uma lista encadeada: o seu algoritmo deverá aceitar como entrada o endereço do primeiro nó da lista.

**Exr 2.13** Um vetor  $A[1..m]$  é *segmento* de um vetor  $B[1..n]$  se existe  $i$  tal que  $1 \leq i \leq n - m + 1$  e  $A[1..m] = B[i..i+m-1]$ . Escreva (em notação CLRS) um algoritmo recursivo que devolva 1 se  $A[1..m]$  é segmento de  $B[1..n]$  e devolva 0 em caso contrário. Escreva uma documentação correta de seu algoritmo.

**Exr 2.14** [Ordenação por seleção] Escreva uma versão recursiva do algoritmo de ordenação por seleção (veja exercício 5.2). O algoritmo deve rearranjar em ordem crescente qualquer vetor dado  $A[p..r]$ .

**Exr 2.15** [CLR 1.3-6, inserção binária] O algoritmo de ordenação-por-inserção rearranja um vetor  $A[1..n]$  de modo que ele fique em ordem crescente. Para  $j = 1, 2, \dots, n$ , o algoritmo insere  $A[j+1]$  na sua posição correta dentro do vetor  $A[1..j]$ , que já está em ordem crescente. Suponha agora que isso seja feito assim:

primeiro, o algoritmo faz uma busca binária para determinar  $i$  tal que  $A[i] \leq A[j+1] < A[i+1]$ ; depois, o algoritmo desloca  $A[i+1..j]$  para as posições  $i+2..j+1$  e insere  $A[j+1]$  na posição  $i+1$ .

Faça uma análise do consumo de tempo do algoritmo. Dê a resposta em notação O.

**Exr 2.16** [Ordenação por inserção] Escreva uma versão recursiva do algoritmo de ordenação por inserção (veja exercício 5.1). O algoritmo deve rearranjar em ordem crescente qualquer vetor dado  $A[p..r]$ .

**Exr 2.17** Esta questão envolve retângulos cuja altura e largura são inteiros positivos. Um retângulo é *elementar* se tem altura 1 ou largura 1. Vamos considerar partições de um retângulo  $R$  em retângulos elementares.<sup>3</sup> Seja  $p(m, n)$  o número de diferentes partições de um retângulo de altura  $m$  e largura  $n$ . Por exemplo,  $p(2, 2) = 7$ .

1. Escreva uma recorrência para  $p(1, n)$ .
2. Resolva a recorrência do item 1.
3. Escreva uma recorrência para  $p(2, n)$ .
4. Descreva um algoritmo para determinar  $p(2, n)$ .
5. Determine a complexidade de seu algoritmo.

---

<sup>3</sup> Cada ponto de um dos retângulos da partição deve pertencer a  $R$ . Reciprocamente, cada ponto de  $R$  deve pertencer ao interior de um e apenas um dos retângulos da partição ou à fronteira de pelo menos um dos retângulos da partição.

# Capítulo 3

## Recorrências

Uma *recorrência* é uma “fórmula” que define uma única função em termos d’ela mesma. Em outras palavras, uma recorrência é um algoritmo recursivo que calcula uma função.

Resolver uma recorrência é obter uma “fórmula fechada” para a função que a recorrência define. Nem sempre queremos uma fechada exata; às vezes uma fórmula “em termos de O” é suficiente.

Veja CLRS secs.4.1–4.2.

Mais uma observação prática: Como no bilhar, primeiro anuncie o que você vai provar e depois prove. (Veja também a página <https://www.ime.usp.br/~pf/amostra-de-prova/>.)

### 3.1 Solução exata de recorrências

**Exr 3.1** Seja  $f$  a função definida sobre os inteiros positivos pela recorrência

$$\begin{aligned} f(1) &= 1 \\ f(n) &= f(n-1) + 3n + 1 \quad \text{para } n \geq 2. \end{aligned}$$

Resolva a recorrência exatamente.

**Exr 3.2** Seja  $f$  a função definida pela recorrência  $f(1) = 1$ ,  $f(2) = 2$  e

$$f(n) = f(n-2) + 2 \quad \text{para } n \geq 2.$$

Resolva a recorrência exatamente.

**Exr 3.3** Suponha que  $f(1) = 1$  e  $f(n) = f(n-1) + 2n$  para  $n = 2, 3, 4, \dots$ . Mostre que  $f(n) = \Omega(n^2)$ .

**Exr 3.4** Seja  $T(n)$  uma função definida sobre  $\{0, 1, 2, \dots\}$ . Suponha que existem  $\alpha > 0$  e  $\beta > 0$  tais que  $T(1) \leq \alpha$  e  $T(n) \leq T(n-k) + \beta k$  para todo  $n > 1$ . Mostre que  $T(n) \leq \max(\alpha, \beta)n$

**Exr 3.5** [BB 4.21: algoritmo recursivo para determinantes] Sejam  $a$  e  $b$  duas constantes reais positivas. Seja  $f$  a função definida sobre os inteiros positivos pela seguinte recorrência:  $f(1) = a$  e

$$f(n) = n f(n-1) + bn$$

para  $n > 1$ . Dê uma fórmula fechada para  $f(n)$ . (Sua fórmula pode ter um termo da forma  $\sum_{i=1}^n 1/i!$ .) Prove sua fórmula por indução.<sup>1</sup> Escreva  $f(n)$  em notação  $\Theta$  da maneira mais simples que puder. Qual o valor de  $\lim_{n \rightarrow \infty} f(n)/n!$  em função de  $a$  e  $b$ ? (Lembre-se de que  $\sum_{i=1}^{\infty} 1/i! = e - 1$ , sendo  $e = 2.7182818\dots$ )

**Exr 3.6** Seja  $F$  a função definida pela recorrência  $F(0) = F(1) = 0$  e

$$F(n) = F(n-1) + F(n-2) + 1$$

para  $n \geq 2$ . Mostre que  $F(n) = \Omega((3/2)^n)$ .

**Exr 3.7** [CLRS 3.2-6 e CLRS 4-5] Seja  $f$  a função definida sobre os inteiros não-negativos pela recorrência  $f(0) = 0$ ,  $f(1) = 1$  e  $f(n) = f(n-1) + f(n-2) + 2$  para  $n \geq 2$ . Resolva a recorrência exatamente.

**Exr 3.8** Discuta a seguinte recorrência:  $f(7) = 1$  e  $f(n) = f(\lfloor n/2 \rfloor) + n$  para  $n = 8, 9, 10, 11, 12, \dots$

**Exr 3.9** Considere a seguinte recorrência:  $f(1) = 3$  e

$$f(n) = f(n/2) + 1$$

para  $n = 2^1, 2^2, 2^3, \dots$  Quanto vale  $f(16)$ ? Mostre que  $f(n) = \lg(n) + 3$  para  $n = 2^0, 2^1, 2^2, 2^3, \dots$

**Exr 3.10** Considere a seguinte recorrência:  $g(1) = 3$  e  $g(n) = g(\lfloor n/2 \rfloor) + 1$  para todo inteiro  $n > 1$ . Dê uma fórmula fechada para a recorrência.

**Exr 3.11** Seja  $f$  a função definida sobre os inteiros positivos pela recorrência

$$f(1) = 1 \quad \text{e} \quad f(n) = f(\lceil \frac{n-1}{2} \rceil) + 1 \quad \text{para } n \geq 2.$$

Resolva a recorrência.

**Exr 3.12** [CLRS 2.3-5] Seja  $f$  a função definida sobre os inteiros positivos pela recorrência  $f(1) = 1$  e

$$f(n) = \max\{f(\lceil \frac{n-1}{2} \rceil), f(\lfloor \frac{n-1}{2} \rfloor)\} + 1$$

para  $n \geq 2$ . Resolva a recorrência.

**Exr 3.13** Seja  $f$  a função definida sobre os inteiros positivos pela recorrência

$$\begin{aligned} f(1) &= 0 \\ f(n) &= f(\lceil n/2 \rceil) + f(\lfloor n/2 \rfloor) + n + 1 \quad \text{para } n \geq 2. \end{aligned}$$

Calcule  $f(n)$  para  $n = 1, 2, 3, 4, 5$ . Mostre que  $f(n) = n\lceil \lg n \rceil - 2^{\lceil \lg n \rceil} + 2n - 1$ .

---

<sup>1</sup> A rigor, eu deveria dizer “indução matemática”.

**Exr 3.14** Seja  $f$  a função definida sobre inteiros positivos pela recorrência

$$\begin{aligned} f(1) &= 1 \\ f(n) &= f(\lfloor \frac{n}{2} \rfloor) + f(\lceil \frac{n}{2} \rceil) + 6n + 7 \quad \text{para } n \geq 2. \end{aligned}$$

Verifique que a recorrência é honesta, ou seja, de fato define uma função sobre os inteiros positivos. Resolva a recorrência. Não é preciso resolver a recorrência de maneira exata: basta mostrar que  $f(n) = O(g(n))$  para alguma função  $g$  razoavelmente pequena.

**Exr 3.15** Considere a seguinte recorrência:  $f(r) = 3$  para todo racional no intervalo  $[\frac{1}{2}, 1)$  e  $f(r) = f(r/2) + 1$  para todo racional  $r \geq 1$ . Resolva a recorrência.

**Exr 3.16 [CLR 1.3-3]** Resolva a seguinte recorrência:  $T(2) = 2$  e  $T(n) = 2T(n/2) + n$  para  $n = 4, 8, 16, 32, 64, \dots$

**Exr 3.17** Sejam  $a, b$  e  $c$  números positivos. Seja  $T$  a função definida pela seguinte recorrência:  $T(1) = a$  e

$$T(n) = 2T(n/2) + bn + c$$

para  $n = 2^1, 2^2, 2^3, \dots$  Dê uma solução da recorrência, ou seja, dê uma fórmula fechada para  $T(n)$ . Prove que sua resposta está correta.

**Exr 3.18** Considere a recorrência  $f(1) = 1$  e  $f(n) = 2f(n/2) + 3$  para  $n = 2^1, 2^2, 2^3, \dots$  Quero provar que  $f(n) = O(n)$ . Comente a seguinte prova: “Seja  $n \geq 2$  uma potência inteira de 2. Como  $f(n) = 2f(n/2) + 3$ , podemos supor, por hipótese de indução, que existe  $c$  tal que  $f(n) \leq 2(c \cdot n/2) + 3$ . Logo,  $f(n) \leq cn + 3$ . Como  $n \geq 2$ , temos  $2n > 3$  e portanto  $f(n) < cn + 2n = (c+2)n$ . Logo,  $f(n)$  é menor que um múltiplo de  $n$  para todo  $n \geq 2$ . Isto prova que  $f(n) = O(n)$ .”

**Exr 3.19** Considere a recorrência  $T(2^0) = 1$  e  $T(n) = 4T(n/2) + n$  para  $n = 2^1, 2^2, 2^3, \dots$  Comente a seguinte solução da recorrência: “Para provar que  $T(n) = O(n^2)$ , vou mostrar, por indução, que  $T(n) \leq 2n^2$ . Isto é obviamente verdade quando  $n = 2^0$ . Agora suponha que  $n = 2^k$  com  $k > 0$  e suponha que  $T(n/2) \leq 2(n/2)^2$ . Então  $T(n) = 4T(n/2) + n \leq 4 \cdot 2(n/2)^2 + n = 2n^2 + n = O(n^2)$ , como prometi.”

**Exr 3.20** Considere a recorrência  $F(2^0) = 1$  e  $F(n) = 2F(n/2) + 3n + 4$  para  $n = 2^1, 2^2, 2^3, \dots$  Quero provar que  $F(n)$  está em  $O(n \lg n)$ . Comente a seguinte solução: “Vou provar que existe uma constante  $c > 0$  tal que  $F(n) \leq cn \lg n$  para  $n = 2^2, 2^3, 2^4, \dots$  A desigualdade é certamente verdadeira quando  $n = 2^2$ . Agora tome  $n > 2^2$ . Podemos supor que existe um número  $c'$  tal que  $F(n/2) \leq c' \frac{n}{2} \lg \frac{n}{2}$ . Portanto,  $F(n) = 2F(\frac{n}{2}) + 3n + 4 \leq 2c' \frac{n}{2} \lg \frac{n}{2} + 3n + 4 \leq c'n \lg n + 3n + 4 \leq c'n \lg n + 3n \lg n + 4n \lg n = (c' + 3 + 4)n \lg n$ . Está provado que existe uma constante  $c$  tal que  $F(n) \leq cn \lg n$  para  $n = 2^2, 2^3, 2^4, \dots$ ”

**Exr 3.21** Considere a recorrência  $T(1) = 1$  e  $T(n) = 4T(n/2) + n$  para todo  $n > 1$  que seja potência inteira de 2. Prove, por indução, que  $T(n) = O(n^2)$ .

**Exr 3.22** Seja  $T$  a função definida da seguinte maneira:  $T(1) = 1$  e

$$T(n) = 4T(\lfloor n/2 \rfloor) + n$$

para  $n = 2, 3, 4, 5, \dots$ . Verifique que a recorrência é honesta (isto é, define uma função). A partir da árvore da recorrência (= recurrence tree), adivinhe uma boa cota superior assintótica para  $T(n)$  (ou seja, algo da forma “ $T(n) = O(?)$ ”). Prove a cota pelo método da substituição.

**Exr 3.23** Seja  $T$  a função definida pela recorrência  $T(1) = 1$  e  $T(n) = 4T(\lfloor n/2 \rfloor) + n$  para todo inteiro  $n > 1$ . A que ordem  $\Theta$  pertence  $T$ ?

**Exr 3.24** [CLRS 4.2-1 simplificado, Karatsuba & Ofman] Seja  $T$  a função definida sobre os inteiros positivos pela recorrência

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 3T(\lfloor n/2 \rfloor) + n \quad \text{para } n \geq 2. \end{aligned}$$

Verifique que a recorrência é honesta (isto é, define uma função). A partir da árvore da recorrência (= recurrence tree), adivinhe uma boa cota superior assintótica para  $T(n)$  (ou seja, algo da forma “ $T(n) = \Theta(?)$ ”). Prove a cota pelo método da substituição.

**Exr 3.25** [Generalização de Karatsuba & Ofman] Seja  $T$  a função definida sobre os inteiros positivos pela recorrência

$$\begin{aligned} T(1) &= c \\ T(n) &= 3T(\lfloor n/2 \rfloor) + n \quad \text{para } n \geq 2, \end{aligned}$$

onde  $c$  é uma constante. A que classe  $\Theta$  a função pertence?

**Exr 3.26** [Generalização de Karatsuba & Ofman] Seja  $T$  a função definida sobre os inteiros positivos pela recorrência

$$\begin{aligned} T(1) &= c \\ T(n) &= 3T(\lfloor n/2 \rfloor) + dn \quad \text{para } n \geq 2, \end{aligned}$$

onde  $c$  e  $d$  são constantes. A que classe  $\Theta$  a função pertence?

**Exr 3.27** [Karatsuba & Ofman generalizado] Seja  $T$  a função definida sobre os inteiros positivos pela recorrência

$$\begin{aligned} T(1) &= \alpha \\ T(n) &= 3T(\lfloor n/2 \rfloor) + \beta n + \gamma \quad \text{para } n \geq 2, \end{aligned}$$

onde  $\alpha, \beta$  e  $\gamma$  são constantes. A que classe  $\Theta$  a função pertence?

**Exr 3.28** [Verdadeiro Karatsuba & Ofman] Seja  $T$  a função definida sobre os inteiros positivos pela recorrência

$$\begin{aligned} T(1) &= 1 \\ T(n) &\leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(1 + \lceil n/2 \rceil) + \beta n \quad \text{para } n \geq 2, \end{aligned}$$

sendo  $\beta$  uma constante positiva. A que classe  $O$  a função pertence?

**Exr 3.29** Seja  $Q$  o conjunto de todos os números racionais maiores que  $1/2$ . A seguinte recorrência define uma função que leva  $Q$  no conjunto dos números racionais positivos?

$$\begin{aligned} f(x) &= 1 && \text{para } 1/2 < x \leq 1 \\ f(x) &= 2f(x/2) + 7x - 2 && \text{para } x > 1 \end{aligned}$$

Resolva a recorrência.

**Exr 3.30** Resolva a recorrência  $T(1) = 1$  e  $T(n) = 2T(\lceil n/2 \rceil) + 7n + 2$  para todo inteiro  $n > 1$ .

**Exr 3.31** Seja  $f$  a função definida pela recorrência

$$\begin{aligned} f(1) &= 1 \\ f(n) &= 2f(n/2) + 7n + 2 && \text{para } n = 2, 4, 8, 16, 32, \dots \end{aligned}$$

e  $f'$  a função definida pela recorrência

$$\begin{aligned} f'(1) &= 1 \\ f'(n) &= 2f'(\lceil n/2 \rceil) + 7n + 2 && \text{para } n = 2, 3, 4, 5, 6, \dots \end{aligned}$$

Verifique que as recorrências são honestas (isto é, que as funções  $f$  e  $f'$  estão bem definidas). Encontre funções  $F$  e  $F'$  tais que  $f = O(F)$  e  $f' = O(F')$ . Prove suas afirmações.

**Exr 3.32** Considere a função  $T$  definida pela seguinte recorrência:  $T(r) = 1$  para todo  $r$  racional tal que  $\frac{1}{3} < r \leq 1$  e

$$T(r) = T(r/3) + T(2r/3) + 5r$$

para todo racional  $r > 1$ . Mostre que  $T(r) = O(r \lg r)$ .

**Exr 3.33** Considere a função  $F$  definida pela recorrência  $F(1) = 1$  e

$$F(n) = F(\lceil n/3 \rceil) + F(\lfloor 2n/3 \rfloor) + 5n$$

para  $n = 2, 3, 4, \dots$ . Mostre que  $F(n) = O(n \lg n)$ . (Dicas: Veja exercício 3.32. Veja exercício 1.60.)<sup>2</sup>

**Exr 3.34** Resolva a seguinte recorrência:  $T(n) = 1$  e

$$T(n) = T(n/2) + n \lg n$$

para todo  $n > 1$  que seja potência inteira de 2.

**Exr 3.35** Resolva a seguinte recorrência:

$$\begin{aligned} f(1) &= 1 \\ f(n) &= 2f(n/2) + n \lg n && \text{para } n = 2, 4, 8, \dots, 2^k, \dots \end{aligned}$$

---

<sup>2</sup> Esta recorrência é relevante no estudo do consumo de tempo médio do Quicksort.

**Exr 3.36** Seja  $T$  a função definida pela recorrência  $T(1) = 0$  e

$$T(n) = \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1)$$

para todo inteiro  $n > 1$ . Mostre que  $T(n) = \Omega(2^n)$ .

**Exr 3.37** Considere a função  $f$  definida pela seguinte recorrência:  $f(1) = 0$  e

$$f(n) = n + \frac{2}{n} \sum_{j=\lfloor n/2 \rfloor}^{n-1} f(j) \quad (3.1)$$

para  $n \geq 2$ . É claro que  $f$  leva números inteiros positivos em números racionais. A que ordem O pertence a função  $f$ ? (Esta recorrência aparece na análise do algoritmo SELECTL-ALEATORIZADO.)

**Exr 3.38** Considere a função  $f$  definida pela seguinte recorrência:  $f(0) = 1$ ,  $f(1) = 1$  e

$$f(n) = \max_{0 \leq k \leq n-1} \{f(k) + f(n-k-1)\} + n$$

para  $n \geq 2$ . É claro que  $f$  leva números inteiros não-negativos em números racionais. A que ordem O pertence a função  $f$ ? (Esta recorrência aparece na análise do algoritmo QUICKSORT.)

**Exr 3.39** [CLR 4.3-1, modificado] Dê uma cota superior simples e razoavelmente justa para a função  $T$  definida da seguinte maneira:  $T(1) = 1$  e  $T(n) = T(\lceil n/2 \rceil) + n^2$  para todo inteiro  $n > 1$ .

**Exr 3.40** Considere a função  $f$  definida pela seguinte recorrência:  $f(1) = 2$  e

$$f(n) = f(3n/4) + 3$$

para  $n > 1$ . Para que valores de  $n$  a função  $f$  está definida? Enuncie e prove uma fórmula fechada exata para  $f$ .

**Exr 3.41** Seja  $f$  a função definida sobre as potências inteiras de  $\frac{4}{3}$  pela recorrência

$$\begin{aligned} f(1) &= 77 \\ f(n) &= f(3n/4) + 88 \quad \text{para } n = \frac{4}{3}, (\frac{4}{3})^2, (\frac{4}{3})^3, \dots \end{aligned}$$

Resolva a recorrência. Em seguida, resolva a recorrência

$$\begin{aligned} f'(1) &= 77 \\ f'(n) &= f'(\lfloor 3n/4 \rfloor) + 88 \quad \text{para } n > 1, \end{aligned}$$

que define uma função  $f'$  sobre os inteiros positivos.

**Exr 3.42** Seja  $T$  a função definida pela seguinte recorrência:  $T(0) = 0$  e  $T(n) = T(\lfloor n - \sqrt{n} \rfloor) + 1$  para todo inteiro  $n > 0$ . (Aqui, “ $\sqrt{n}$ ” representa a determinação positiva da raiz.) Mostre que  $T(n) = \Theta(\sqrt{n})$ .

**Exr 3.43** Seja  $T$  a função definida pela seguinte recorrência:  $T(1) = 1$  e  $T(n) = T(\lfloor \sqrt{n} \rfloor) + 1$  para todo inteiro  $n > 1$ . (Aqui, “ $\sqrt{n}$ ” representa a determinação positiva da raiz quadrada de  $n$ .) Mostre que  $T(n) = \Theta(\lg \lg n)$ .

**Exr 3.44** A. Suponha que  $f(1) = 1$  e  $f(n) = f(\lfloor n/2 \rfloor) + 1$  para  $n = 2, 3, 4, \dots$  Mostre que  $f(n) = \Omega(\lg n)$ . B. Suponha que  $f'(1) = 1$  e  $f'(n) = f'(\lfloor n/2 \rfloor) + n$  para  $n = 2, 3, 4, \dots$  Mostre que  $f'(n) = \Omega(n)$ . C. Suponha que  $f''(1) = 1$  e  $f''(n) = 2f''(\lfloor n/2 \rfloor) + n$  para  $n = 2, 3, 4, \dots$  Mostre que  $f''(n) = \Omega(n \lg n)$ .

**Exr 3.45** Seja  $f$  a função definida sobre os inteiros positivos pela seguinte recorrência:  $f(1) = 1$ ,  $f(2) = 2$ ,  $f(3) = 3$  e

$$f(n) = 5f(\lfloor n/4 \rfloor) + 6n$$

para  $n \geq 4$ . Seja  $g$  a função definida sobre os inteiros positivos pela seguinte recorrência:  $g(1) = 1$  e

$$g(n) = 3g(\lfloor n/2 \rfloor) + 4n$$

para  $n \geq 2$ . Verifique que as recorrências são honestas. É verdade que  $f(n) = O(g(n))$ ? É verdade que  $g(n) = O(f(n))$ ?

## 3.2 Recorrências “sem base”

Suponha que  $F$  é uma função definida sobre os inteiros positivos. Considere a afirmação

$$F \text{ satisfaz a recorrência } F(n) = F(n - 1) + 3n + 2. \quad (3.2)$$

O que ela significa? A afirmação deve ser entendida assim: existe um inteiro positivo  $N$  e constantes  $c_1, c_2, \dots, c_{N-1}$  tais que

$$\begin{aligned} F(n) &= c_n && \text{para } 1 \leq n \leq N - 1 \\ F(n) &= F(n - 1) + 3n + 2 && \text{para } n \geq N. \end{aligned}$$

Essas informações não bastam para determinar a função  $F$  exatamente, mas são suficientes determinar a ordem a que  $F$  pertence: quaisquer que sejam  $N, c_1, c_2, \dots, c_{N-1}$ ,

$$F(n) = O(n^2).$$

É claro que  $F(n)$  também é  $O(n^3)$ , mas não é  $O(n^{3/2})$ , nem  $O(n \lg n)$ , nem ...

**Exr 3.46** Suponha que a função  $F$  está definida sobre os inteiros positivos e que

$$F(n) = 2F(n - 1) + 4n.$$

A que ordem  $O$  pertence  $F$ ?

**Exr 3.47** Suponha que a função  $F$  está definida sobre os inteiros positivos e que

$$F(n) = F(\lceil n/2 \rceil) + 1.$$

Resolva a recorrência em termos da notação  $O$ .

**Exr 3.48** Que família de recorrências a expressão

$$T(n) = 3T(\lfloor n/2 \rfloor) + n$$

define? A que ordem O pertencem todas as recorrências da família?

### 3.3 Recorrências assintóticas

Suponha que  $F$  é uma função definida sobre os inteiros positivos. Considere a afirmação

$$F \text{ satisfaz a recorrência } F(n) = F(n - 1) + O(n). \quad (3.3)$$

O que ela significa? A afirmação deve ser entendida assim: existe um inteiro positivo  $N$ , uma  $f(n)$  sobre os inteiros positivos e uma constante  $c$  tais que

$$\begin{aligned} F(n) &= f(n) && \text{para } 1 \leq n \leq N - 1 \\ F(n) &= F(n - 1) + f(n) && \text{para } n \geq N \end{aligned}$$

e  $0 \leq f(n) \leq cn$  sempre que  $n \geq N$ .

Essas informações não bastam para determinar a função  $F$  exatamente, mas são suficientes determinar a ordem a que  $F$  pertence: quaisquer que sejam  $N, f$  e  $c$ ,

$$F(n) = O(n^2).$$

É claro que  $F(n)$  também é  $O(n^3)$ , mas não é  $O(n^{3/2})$ , nem  $O(n \lg n)$ , nem ...

**Exr 3.49** Uma função  $T$  está definida sobre os inteiros positivos. A única informação que tenho sobre  $T$  é que  $T(n) \leq T(n - 1) + n$  para todo  $n$  suficientemente grande. A que ordem O a função  $T$  pertence?

**Exr 3.50** Considere todas as recorrências do tipo  $T(n) \leq T(n - 1) + O(n)$ . Mostre que toda função  $T$  que satisfaz uma recorrência desse tipo pertence à ordem  $O(n^2)$ .

**Exr 3.51** Resolva as recorrências da classe  $T(n) = T(n - 2) + O(n)$ .

**Exr 3.52** Resolva as recorrências do tipo  $T(n) = 5T(n - 1) + O(n)$

**Exr 3.53** [Karatsuba & Ofman, CLRS 4.2-1] Considere a classe de recorrências  $T(n) = 3T(\lfloor n/2 \rfloor) + \Theta(n)$ . Dê uma boa cota superior assintótica para  $T(n)$ . Prove sua cota.

**Exr 3.54** A que ordem  $\Theta$  pertencem as soluções das recorrências do tipo  $T(n) = T(n/3) + \Theta(1)$ ?

**Exr 3.55** Considere as recorrências da forma  $T(n) = aT(n/5) + 4n^3$ , com  $a \geq 1$ . (Estou supondo que  $n$  é inteiro positivo. No lugar de “ $n/5$ ” podemos ter “ $\lfloor n/5 \rfloor$ ” ou “ $\lceil n/5 \rceil$ ”.) Mostre que se  $a < 5^3$  então  $T(n) = \Theta(n^3)$ . Mostre que se  $a = 5^3$  então  $T(n) = \Theta(n^3 \lg n)$ . Mostre que se  $a > 5^3$  então  $T(n) = \Theta(n^{\log_5 a})$ . (Sugestão: use o Teorema Mestre.)

**Exr 3.56** Suponha que a função  $F$  está definida sobre os inteiros positivos e que

$$F(n) = F(\lceil n/2 \rceil) + O(1).$$

A que ordem O pertence  $F$ ?

## Capítulo 4

# Medida de desempenho de algoritmos

### 4.1 Algoritmos iterativos

**Exr 4.1** Faça a análise de desempenho do algoritmo abaixo usando notação O.

```
ORDENAÇÃO-POR-INSERÇÃO ( $A, p, r$ )
1  para  $j \leftarrow p + 1$  até  $r$  faça
2     $c \leftarrow A[j]$ 
3     $i \leftarrow j - 1$ 
4    enquanto  $i \geq p$  e  $A[i] > c$  faça
5       $A[i + 1] \leftarrow A[i]$ 
6       $i \leftarrow i - 1$ 
7     $A[i + 1] \leftarrow c$ 
```

**Exr 4.2** Quanto tempo consome o algoritmo ORDENAÇÃO-POR-INSERÇÃO (veja exercício 4.1), no melhor caso, para ordenar um vetor com  $n$  elementos distintos. Dê sua resposta em notação assintótica.

**Exr 4.3** [AU 3.7.1] O algoritmo abaixo opera sobre um vetor  $A[1..n]$ . Qual o consumo de tempo do algoritmo? Use notação O, mas procure dar a resposta mais justa possível.

```
Xxx ( $n, A$ )
1   $s \leftarrow 0$ 
2  para  $i \leftarrow 1$  até  $n$  faça
3     $s \leftarrow s + A[i]$ 
4   $m \leftarrow s/n$ 
5   $k \leftarrow 1$ 
6  para  $i \leftarrow 2$  até  $n$  faça
7    se  $(A[i] - m)^2 < (A[k] - m)^2$ 
8      então  $k \leftarrow i$ 
9  devolva  $k$ 
```

**Exr 4.4** [AU 3.7.2] O fragmento abaixo opera sobre uma matriz  $A[1..n, 1..n]$ . Qual o consumo de tempo do fragmento? Use notação O, mas procure dar a resposta mais justa possível.

```

1 para  $i \leftarrow 1$  até  $n - 1$  faça
2   para  $j \leftarrow i + 1$  até  $n$  faça
3     para  $k \leftarrow i$  até  $n$  faça
4        $A[j, k] \leftarrow A[j, k] - A[i, k] \cdot A[j, i]/A[i, i]$ 

```

**Exr 4.5** Quanto tempo consome o algoritmo abaixo?

INTERVALOS-DISJUNTOS ( $a, b, n$ )

```

1  $x \leftarrow 0$ 
2  $j \leftarrow 1$ 
3 enquanto  $j \leq n$  faça
4    $x \leftarrow x + 1$ 
5    $k \leftarrow j + 1$ 
6   enquanto  $k \leq n$  e  $a_k \leq b_j$  faça
7      $k \leftarrow k + 1$ 
8    $j \leftarrow k$ 
9 devolva  $x$ 

```

**Exr 4.6** O algoritmo abaixo recebe vetores  $s[1..n]$  e  $f[1..n]$  e devolve um número inteiro entre 0 e  $n$ :

ALGO ( $s, f, n$ )

```

1  $t \leftarrow 0$ 
2  $i \leftarrow 1$ 
3 enquanto  $i \leq n$  faça
4    $t \leftarrow t + 1$ 
5    $m \leftarrow i + 1$ 
6   enquanto  $m \leq n$  e  $s[m] < f[i]$  faça
7      $m \leftarrow m + 1$ 
8    $i \leftarrow m$ 
9 devolva  $t$ 

```

Mostre que o consumo de tempo do algoritmo é  $O(n)$  (apesar dos dois loops encaixados).

**Exr 4.7** Um aluno alega que o seguinte algoritmo consome  $O(n)$  unidades de tempo. Ele está certo?

FAZ-ALGO ( $s, f, n$ )

```

1  $j \leftarrow 1$ 
2  $t \leftarrow 0$ 
3  $s[1] \leftarrow 0$ 
4 enquanto  $j < n$  faça
5   enquanto  $t > 0$  e  $f[t] \neq f[j]$  faça
6      $t \leftarrow s[t]$ 
7    $t \leftarrow t + 1$ 
8    $j \leftarrow j + 1$ 
9   se  $f[j] = f[t]$ 
0     então  $s[j] \leftarrow s[t]$ 

```

0            setão  $s[j] \leftarrow t$

**Exr 4.8** O algoritmo abaixo recebe um vetor  $A[0..n-1]$  de números inteiros. Quanto tempo consome? Expresse sua resposta em notação O, mas procure ser o mais justo possível.

```

CAIXA-PRETA ( $A, n$ )
1  $k \leftarrow 0$     $i \leftarrow 1$     $j \leftarrow 0$ 
2 enquanto  $i < n$  e  $k + j + 1 < n$  faça
3     se  $A[k + j] = A[(i + j) \bmod n]$ 
4       então  $j \leftarrow j + 1$ 
5       senão se  $A[k + j] < A[(i + j) \bmod n]$ 
6           então  $i \leftarrow i + j + 1$     $j \leftarrow 0$ 
7           senão se  $A[k + j] > A[(i + j) \bmod n]$ 
8              então  $h \leftarrow \max(i, k + j + 1)$     $k \leftarrow h$     $i \leftarrow h + 1$     $j \leftarrow 0$ 
9 devolva  $k$ 

```

**Exr 4.9** Eis um exemplo bobo:

```

EXEMPLO-BOBO ( $n$ )
1  $s \leftarrow 0$ 
2 para  $i \leftarrow 1$  até  $n$  faça
3      $s \leftarrow s + i$ 
4 devolva  $s$ 

```

Quanto tempo o algoritmo consome? Escreva um algoritmo MENOS-BOBO que consuma menos tempo e produza o mesmo efeito. Quanto tempo MENOS-BOBO consome?

**Exr 4.10** Parte 1: Escreva o pseudocódigo de uma função iterativa simples que procure um número  $x$  num vetor estritamente crescente  $A$ : ao receber um número  $x$  e um vetor  $A[1..n]$  tal que  $A[1] < A[2] < \dots < A[n]$ , a função deve devolver o único índice  $i$  em  $\{1, \dots, n+1\}$  tal que  $A[i-1] < x \leq A[i]$ . (Se  $i = 1$ , imagine  $-\infty$  no lugar de  $A[0]$ . Se  $i = n+1$ , imagine  $+\infty$  no lugar de  $A[n+1]$ .) Sua função deve usar busca sequencial e não busca binária. Parte 2: Suponha agora que  $x = A[i]$  para algum  $i$ , com igual probabilidade para cada  $i$  em  $\{1, \dots, n+1\}$ . Mostre que o número médio de iterações do algoritmo é  $(n+1)/2$ .

## 4.2 Algoritmos recursivos

A análise de consumo de tempo de um algoritmo recursivo envolve a solução de uma recorrência. Veja CLRS sec.2.3.

**Exr 4.11** Seja  $T(n)$  o consumo de tempo de seguinte trecho de programa.

```

1 para  $i \leftarrow 1$  até  $n$ 
2     faça  $j \leftarrow n$ 
3       enquanto  $j > 1$ 
4           faça  $j \leftarrow \lfloor j/2 \rfloor$ 

```

Dê uma fórmula para  $T(n)$ , supondo a execução de cada linha do código consome 1 unidade de tempo.

**Exr 4.12 [CLRS 2.3-4]** Analise a versão recursiva do algoritmo ORDENAÇÃO-POR-INSERÇÃO. (Veja exercício 4.1.) Qual a recorrência que dá o consumo de tempo do algoritmo? Resolva a recorrência.

**Exr 4.13** Analise a versão recursiva do algoritmo ORDENAÇÃO-POR-SELEÇÃO.

**Exr 4.14** O algoritmo abaixo supõe  $n \geq 1$  e devolve o valor de um elemento máximo de  $A[1..n]$ . Analise o desempenho do algoritmo.

```

MAX ( $A, n$ )
1   se  $n = 1$ 
2     então devolva  $A[1]$ 
3     senão  $x \leftarrow \text{MAX} (A, n - 1)$ 
4       se  $x \geq A[n]$ 
5         então devolva  $x$ 
6       senão devolva  $A[n]$ 
```

**Exr 4.15** O algoritmo abaixo supõe  $p \leq r$  e devolve o valor de um elemento máximo de  $A[p..r]$ . Analise o desempenho do algoritmo:

```

MAX ( $A, p, r$ )
1   se  $p = r$ 
2     então devolva  $A[1]$ 
3     senão  $q \leftarrow \lfloor (p + r)/2 \rfloor$ 
4        $x \leftarrow \text{MAX} (A, p, q)$ 
5        $y \leftarrow \text{MAX} (A, q + 1, r)$ 
6       se  $x \geq y$ 
7         então devolva  $x$ 
8       senão devolva  $y$ 
```

**Exr 4.16** Diga o que faz o algoritmo abaixo. Faça uma análise do consumo de tempo. Escreva e resolva a recorrência que define o consumo de tempo de pior caso do algoritmo em função de  $n := r - p + 1$ .

```

FIB-SORT ( $A, p, r$ )
 $n \leftarrow r - p + 1$ 
se  $n > 1$ 
  então se  $A[p] > A[r]$  então  $A[p] \leftrightarrow A[r]$ 
    FIB-SORT ( $A, p + 1, r - 1$ )
    se  $A[p] > A[p + 1]$  então  $A[p] \leftrightarrow A[p + 1]$ 
    FIB-SORT ( $A, p + 1, r$ )
```

Quantas vezes o algoritmo compara dois componentes do vetor  $A$ ? Escreva uma relação de recorrência que descreva esse número em função do comprimento  $n$  do vetor. (Não é necessário resolver a recorrência.)

**Exr 4.17** Suponha que a execução de cada linha do algoritmo abaixo, exceto a linha 3, consome 1 unidade de tempo. (Não é preciso entender o que o algoritmo faz.) Para  $n = r - p + 1$ , seja  $T(n)$  o consumo de tempo do algoritmo no pior caso. Dê uma recorrência que caracterize  $T(n)$ .

```
LIMPA ( $A, p, r$ )
1   se  $p > r$ 
2     então devolva  $r$ 
3     senão  $q \leftarrow \text{LIMPA} (A, p, r - 1)$ 
4       se  $A[r] \neq 0$ 
5         então  $q \leftarrow q + 1$ 
6          $A[q] \leftarrow A[r]$ 
7       devolva  $q$ 
```

Dê uma “fórmula” exata para a função definida pela recorrência. Justifique.

**Exr 4.18** Considere o seguinte algoritmo recursivo:

```
CLEANUP ( $A, n$ )
1   se  $n = 0$ 
2     então devolva 0
3     senão  $m \leftarrow \text{CLEANUP} (A, n - 1)$ 
4       se  $A[n] = 0$ 
5         então devolva  $m$ 
6       senão  $A[m + 1] \leftarrow A[n]$ 
7         devolva  $m + 1$ 
```

Seja  $T(n)$  o número de execuções da comparação “ $A[n] = 0$ ”. Escreva uma recorrência para  $T(n)$ . Com base na recorrência, dê uma fórmula fechada para  $T(n)$ . Dê uma cota superior, em notação O, para  $T(n)$ . O que faz o algoritmo?

**Exr 4.19** No seguinte algoritmo (não importa o que ele faz),  $n$  é uma potência inteira de 2:

```
ALGOR ( $n$ )
1   se  $n \leq 1$ 
2     então devolva 1
3     senão para  $i \leftarrow 1$  até 8 faça  $z \leftarrow \text{ALGOR}(n/2)$ 
4       para  $i \leftarrow 1$  até  $n^3$  faça  $z \leftarrow 0$ 
```

Seja  $T(n)$  o número de execuções de “ $z \leftarrow 0$ ”. Escreva uma recorrência para  $T(n)$ . Mostre que  $T(n) = \Omega(n^3 \lg n)$  (não use o Teorema Mestre). Troque “8” por “7” no algoritmo e mostre que  $T(n) = \Omega(n^3)$  (não use o Teorema Mestre).

**Exr 4.20** Seja  $T(n)$  o número de linhas impressas pela invocação  $\text{WASTE}(n)$ . Escreva uma recorrência para  $T(n)$ . Use o Teorema Mestre para determinar a classe  $\Theta$  a que  $T(n)$  pertence.

```
WASTE ( $n$ )
1   para  $i \leftarrow 1$  até  $n$  faça
2     para  $j \leftarrow 1$  até  $i$  faça
```

```

3      imprima  $i, j, n$ 
4  se  $n > 0$  então
5      para  $i \leftarrow 1$  até 4 faça
6          WASTE ( $\lfloor n/2 \rfloor$ )

```

**Exr 4.21** No seguinte algoritmo,  $n$  é uma inteiro positivo:

```

PRINTX ( $n$ )
1  se  $n > 0$ 
2      então PRINTX ( $n - 1$ )
3      para  $i \leftarrow 1$  até  $n$  faça imprima "x"
4          PRINTX ( $n - 1$ )

```

Quantos "x" a invocação PRINTX ( $n$ ) imprime?

**Exr 4.22** O algoritmo abaixo recebe um inteiro positivo  $n$  e imprime asteriscos.

```

ASTERIX ( $n$ )
1  se  $n > 1$ 
2      então  $k \leftarrow \lfloor n/2 \rfloor$ 
3          ASTERIX ( $n - k$ )
4          para  $i \leftarrow 1$  até  $n^2$  faça
5              imprima "*"

```

Seja  $S(n)$  o número de asteriscos que o algoritmo imprime ao receber  $n$ . Escreva uma recorrência para  $S(n)$ . A que ordem  $O$  pertence a função  $S$ ?

**Exr 4.23** Para  $n$  inteiro não-negativo, quantos asteriscos serão impressos em uma chamada de ASTERISCO ( $n$ )? Justifique.

```

ASTERISCO ( $n$ )
1  se  $n > 0$ 
2      então ASTERISCO ( $n - 1$ )
3          para  $i \leftarrow 1$  até  $n$ 
4              faça imprima "*"
5          ASTERISCO ( $n - 1$ )

```

**Exr 4.24** O algoritmo abaixo recebe um inteiro positivo  $n$  e imprime estrelas.

```

ASTERIX ( $n$ )
1  se  $n > 1$ 
2      então  $k \leftarrow \lfloor n/2 \rfloor$ 
3          ASTERIX ( $n - k$ )
4          para  $i \leftarrow 1$  até  $n^2$  faça
5              imprima "*"

```

Que acontece se a linha 1 for trocada por "se  $n \geq 1$ "? Seja  $S(n)$  o número de estrelas que o algoritmo imprime ao receber  $n$ . Escreva uma recorrência para  $S(n)$ . A que ordem  $\Theta$  pertence a

função  $S$ ?

**Exr 4.25** Que número ALGOX devolve ao receber um inteiro não-negativo  $n$ ?

```
ALGOX ( $n$ )
1   se  $n = 0$ 
2     então devolva 0
3    $t \leftarrow \text{ALGOX} (n - 1)$ 
4    $t \leftarrow t + \text{ALGOX} (n - 1) + 3$ 
5   devolva  $t$ 
```

Estime, em notação  $\Theta$ , o consumo de tempo do algoritmo em função de  $n$ .

**Exr 4.26** Que número ALGOY devolve ao receber um inteiro não-negativo  $n$ ?

```
ALGOY ( $n$ )
1   se  $n = 0$ 
2     então devolva 0
3   devolva  $\text{ALGOY} (n - 1) + 2n - 1$ 
```

Estime, em notação  $\Theta$ , o consumo de tempo do algoritmo em função de  $n$ .

**Exr 4.27** Que números ALGOX e ALGOY devolvem ao receber um inteiro não-negativo  $n$ ?

ALGOY ( $n$ )	ALGOX ( $n$ )
1   se $n = 0$	1   se $n = 0$
2     então devolva 0	2     então devolva 0
3 $t \leftarrow 2 \cdot \text{ALGOY} (n - 1) + 4$	3 $t \leftarrow \text{ALGOX} (n - 1) + \text{ALGOX} (n - 1) + 4$
4   devolva $t$	4   devolva $t$

Estime, em notação  $\Theta$ , o consumo de tempo de cada um dos algoritmos em função de  $n$ .

**Exr 4.28** O seguinte algoritmo recebe um vetor  $P[1..n]$  com  $n \geq 0$  e devolve um vetor  $X[0..n]$ .

```
DUMMY ( $P, n$ )
1   para  $m \leftarrow 0$  até  $n$  faça
2      $k \leftarrow m$ 
3     enquanto  $k \geq 1$  e  $\text{TESTE} (P, k) = 0$  faça
4        $k \leftarrow k - 1$ 
5      $X[m] \leftarrow k$ 
6   devolva  $X$ 
```

Ao receber argumentos  $(P, n)$ , a função TESTE devolve 1 ou 0 e consome  $O(n)$  unidades de tempo para dar a resposta.

1. Quais são os possíveis valores de  $X[i]$  para  $i = 0, \dots, n$ ?
2. Calcule detalhadamente o consumo de tempo de DUMMY no pior e no melhor caso.

### 4.3 Generalidades

**Exr 4.29** [CLRS 3.1-3] Explique por que a afirmação “o consumo de tempo do algoritmo  $\mathcal{A}$  é pelo menos  $O(n^2)$ ” não faz sentido.

**Exr 4.30** Explique por que a afirmação “o consumo de tempo do algoritmo  $\mathcal{A}$  é  $O(n^2)$  no pior caso” é redundante. Reescreva a afirmação sem a redundância.

**Exr 4.31** Um algoritmo processa um vetor de  $n$  componentes. O algoritmo tem três passos, executados em sequência. Os consumos de tempo dos três passos são  $O(n)$ ,  $O(n^2)$  e  $O(n\sqrt{n})$  respectivamente. Prove, rigorosamente, que o consumo de tempo do algoritmo todo é  $O(n^2)$ .

**Exr 4.32** [CLR 1.4-1] Suponha que estamos comparando o desempenho de dois algoritmos para um mesmo problema. Quando aplicado a instâncias de tamanho  $n$  do problema, o primeiro consome  $8n^2$  unidades de tempo enquanto o segundo consome  $64n \lg n$ . Para que valores de  $n$  o primeiro é mais rápido que segundo?

**Exr 4.33** [CLR 1.4-2] Suponha que, numa determinada máquina, um algoritmo A consome  $100n^2$  unidades de tempo e um algoritmo B consome  $2^n$  unidades de tempo. Para que valores de  $n$  o algoritmo A é mais rápido que B?

**Exr 4.34** Suponha que estamos estudando o desempenho de um algoritmo em função do tamanho,  $n$ , das instâncias de um problema. Discuta as seguinte afirmações: (1) “o consumo de tempo do algoritmo é  $O(n^2)$  no pior caso”, (2) “o consumo de tempo do algoritmo é  $O(n^2)$  no melhor caso”, (3) “o algoritmo consome pelo menos  $O(n^2)$  unidades de tempo”, (4) “o consumo de tempo do algoritmo é  $O(n^2)$  para  $n \geq 100$ ”.

**Exr 4.35** Suponha que estamos estudando o desempenho de um algoritmo em função do tamanho,  $n$ , das instâncias de um problema. Considere as seguintes afirmações: (1) “o consumo de tempo do algoritmo é  $O(n^2)$  no pior caso” e (2) “o consumo de tempo do algoritmo é  $O(n^2)$  para toda instância do problema”. Qual a diferença entre essas afirmações?

**Exr 4.36** Suponha que estamos estudando o desempenho de um algoritmo em função do tamanho,  $n$ , das instâncias de um problema. Considere as seguintes afirmações: (1) “o consumo de tempo do algoritmo é  $O(n^2)$  no melhor caso” e (2) “o consumo de tempo do algoritmo é  $O(n^2)$  para alguma instância de tamanho  $n$ ”. Qual a diferença entre essas afirmações?

**Exr 4.37** Suponha que estamos estudando o desempenho de um algoritmo em função do tamanho,  $n$ , das instâncias de um problema. Considere as seguintes afirmações: (1) “o consumo de tempo do algoritmo é  $\Omega(n^2)$  no pior caso” e (2) “existe uma instância para a qual o consumo de tempo do algoritmo é  $\Omega(n^2)$ ”. Qual a diferença entre essas afirmações?

**Exr 4.38** Suponha que um algoritmo  $\mathcal{A}$  opera sobre um vetor com  $n$  elementos. Qual o significado das expressões “o consumo de tempo de  $\mathcal{A}$  é  $\Omega(n^2)$ ” e “o consumo de tempo de  $\mathcal{A}$  no pior caso é  $\Omega(n^2)$ ”?

**Exr 4.39** Suponha que o cálculo da expressão  $\text{AAA}(k)$  consome  $O(k)$  unidades de tempo. Quanto tempo consome o seguinte algoritmo?

```
BBB(n)
1   para k ← 1 até n faça
2       AAA(k)
```

**Exr 4.40** Considere o algoritmo clássico para o cálculo do determinante de uma matriz quadrada. Qual o consumo de tempo do algoritmo. Compare isso com o algoritmo de Gauss–Jordan para cálculo de determinantes.

## Capítulo 5

# Alguns problemas simples

Um vetor  $X[1..n]$  é *crescente* se  $X[1] \leq X[2] \leq \dots \leq X[n]$ . O mesmo vetor é *estritamente crescente* se  $X[1] < X[2] < \dots < X[n]$ . Os conceitos de vetor *decrescente* e *estritamente decrescente* são definidos de maneira análoga.

Esses exercícios correspondem às seções 2.1–2.2 de CLRS.

**Exr 5.1** [Insertion sort] Considere o algoritmo “de inserção” para o seguinte problema: rearranjar um vetor  $A[p..r]$  de modo que ele fique em ordem crescente. Para que valores de  $p$  e  $r$  o problema faz sentido? Analise a correção do algoritmo (ou seja, encontre e prove os invariantes<sup>1</sup> apropriados).

**Exr 5.2** [Selection sort, CLRS 2.2-2] Escreva um algoritmo “de seleção” para o seguinte problema: rearranjar um vetor  $A[p..r]$  de modo que ele fique em ordem crescente. Analise a correção do algoritmo (ou seja, encontre e prove os invariantes apropriados). Analise o consumo de tempo do algoritmo.

**Exr 5.3** O seguinte recebe um número natural  $n \geq 2$  e devolve outro número natural. Que função o algoritmo calcula?

```
RAIZQ ( $n$ )
1    $p \leftarrow 1$ 
2    $r \leftarrow n$ 
3   enquanto  $p + 1 < r$  faça
4        $q \leftarrow \lfloor (p + r)/2 \rfloor$ 
5       se  $q^2 \leq n$ 
6           então  $p \leftarrow q$ 
7       senão  $r \leftarrow q$ 
8   devolva  $p$ 
```

(Bom exercício: se devolver  $q$  está errado.) Quanto tempo o algoritmo consome?

---

<sup>1</sup> Um invariante é essencialmente o mesmo que a hipótese de indução numa prova por indução matemática.

**Exr 5.4** Considere o seguinte problema: dado um conjunto  $S$  de inteiros positivos distintos dois a dois e um inteiro positivo  $x$ , decidir se  $S$  contém dois elementos cuja soma é  $x$ . Escreva (em notação CLRS) um algoritmo que resolva o problema em tempo  $O(n \lg n)$ , onde  $n = |S|$ . Prove que o seu algoritmo está correto.

**Exr 5.5** [CLRS 2.3.1] Suponha dado um vetor  $A[p..r]$  e um índice  $q$  tais que os subvetores  $A[p..q]$  e  $A[q+1..r]$  são crescentes. Nossa problema: Rearranjar o vetor  $A[p..r]$  de modo que ele fique em ordem crescente. Esse é o *problema da intercalação (= merge)*. Escreva um algoritmo que resolva o problema; procure fazer um algoritmo diferente do CLRS 2.3.1. Analise a correção do seu algoritmo (diga quais os invariantes). Analise o consumo de tempo do seu algoritmo.

**Exr 5.6** Suponha dado um algoritmo INTERCALA que recebe vetores  $X[1..m]$  e  $Y[1..n]$ , ambos crescentes, e consome tempo  $\Theta(m + n)$  para produzir um vetor crescente  $Z[1..m+n]$  que contém todos os elementos de  $X$  e todos os de  $Y$ . Suponha dados vetores crescentes  $A_0[1..2^0]$ ,  $A_1[1..2^1]$ ,  $A_2[1..2^2]$ , ...,  $A_k[1..2^k]$ . Queremos usar INTERCALA para reunir todos os elementos desses vetores em um vetor crescente  $B[1..2^{k+1}-1]$ . Há duas maneiras de fazer isso. Na primeira, intercalamos  $A_0$  com  $A_1$ , depois intercalamos o resultado com  $A_2$ , e assim por diante. Na segunda, intercalamos  $A_k$  com  $A_{k-1}$ , depois intercalamos o resultado com  $A_{k-2}$ , e assim por diante. Transforme essa descrição informal em código. Quanto tempo consome cada um dos dois algoritmos? (Dê sua resposta em notação  $\Theta$ .)

**Exr 5.7** Considere a sequência de vetores

$$A_k[1..2^k], A_{k-1}[1..2^{k-1}], \dots, A_1[1..2^1], A_0[1..2^0].$$

Suponha que cada um dos vetores é crescente. Queremos reunir, por meio de sucessivas operações de intercalação (*= merge*), o conteúdo dos vetores  $A_0, \dots, A_k$  em um único vetor crescente  $B[1..n]$ , onde  $n = 2^{k+1} - 1$ . Escreva um algoritmo que faça isso em  $O(n)$  unidades de tempo. Justifique.

**Exr 5.8** Seja  $\mathcal{B}$  um conjunto de objetos para os quais está definida a relação “=” mas não estão definidas as relações “<” e “>”. Seja  $\langle x_1, \dots, x_n \rangle$  uma sequência de objetos de  $\mathcal{B}$ . Um *elemento majoritário* da sequência é qualquer  $y$  em  $\{x_1, \dots, x_n\}$  tal que  $|\{i : x_i = y\}| > n/2$ . Uma sequência  $\langle x_1, \dots, x_n \rangle$  de objetos de  $\mathcal{B}$  está *agrupada* se, para cada par  $j < k$  de índices tal que  $x_j = x_k$  tem-se  $x_j = x_{j+1} = \dots = x_k$ . Dê um algoritmo  $O(n)$  que receba uma sequência agrupada  $\langle x_1, \dots, x_n \rangle$  e devolva um elemento majoritário da sequência ou constate que um tal elemento não existe.

**Exr 5.9** [Elemento majoritário] Um número  $x$  é *valor majoritário* num vetor  $A[1..n]$  se a cardinalidade do conjunto  $\{i : A[i] = x\}$  for estritamente maior que  $n/2$ . Um *valor majoritário* de um vetor  $A[1..n]$  é qualquer número  $x$  que seja majoritário em  $A[1..n]$ . Um vetor  $A[1..n]$  é *forte* se tem um valor majoritário. Descreva um algoritmo que decida, em tempo  $O(n)$ , se um vetor  $A[1..n]$  de números inteiros positivos<sup>2</sup> é forte e em caso afirmativo devolva o seu valor majoritário.

---

<sup>2</sup> Os elementos de  $A$  não precisam ser necessariamente positivos e nem mesmo números inteiros: basta que eu possa decidir se dois elementos são iguais ou não. Mas é cômodo restringir a atenção a inteiros positivos pois assim posso usar “-1” para indicar ausência de elemento majoritário.

**Exr 5.10 [Elemento majoritário]** Um número  $a$  é *valor majoritário* de um vetor  $A[1..n]$  se a cardinalidade do conjunto  $\{i : A[i] = a\}$  for estritamente maior que  $n/2$ . Um vetor  $A[1..n]$  é *bom* se tem um valor majoritário. Descreva um algoritmo que decide se um vetor  $A[1..n]$  bom e em caso afirmativo devolve o seu valor majoritário. Os elementos do vetor não pertencem a um conjunto ordenado e portanto somente comparações de igualdade fazem sentido (comparações do tipo " $A[i] < A[j]$ " não fazem sentido). O seu algoritmo deve executar  $O(n)$  comparações entre elementos do vetor.

**Exr 5.11** Suponha que os algoritmos  $A$  e  $B$  transformam cadeias de caracteres em outras cadeias de caracteres. O algoritmo  $A$  consome  $O(n^2)$  unidades de tempo e o algoritmo  $B$  consome  $O(n^4)$  unidades de tempo, onde  $n$  é o número de caracteres da cadeia de entrada. Considere agora o algoritmo  $AB$  que consiste na composição de  $A$  e  $B$ , com  $B$  recebendo como entrada a saída de  $A$ . Qual o consumo de tempo de  $AB$ ?

**Exr 5.12** É dado um vetor contendo uma permutação de  $1, 2, \dots, n$ . A operação  $\text{mpf}(i)$  consiste em mover o  $i$ -ésimo elemento do vetor para o fim. Sua tarefa é determinar uma sequência de operações  $\text{mpf}$ , a mais curta possível, que rearrange o vetor em ordem crescente. Um exemplo com  $n = 5$ :

vetor inicial:	5 1 4 6 2 3
após $\text{mpf}(3)$ :	5 1 6 2 3 4
após $\text{mpf}(1)$ :	1 6 2 3 4 5
após $\text{mpf}(2)$ :	1 2 3 4 5 6

Descreva um algoritmo que determine uma solução do problema em tempo  $O(n \lg n)$ .

**Exr 5.13** Seja  $s_1, \dots, s_n$  e  $t_1, \dots, t_n$  elementos do conjunto  $\{1, \dots, k\}$ . Suponha que  $s_i < t_i$  para cada  $i$ . Suponha também que  $s_1 \leq s_2 \leq \dots \leq s_n$ . Para cada  $i$ , o par  $(s_i, t_i)$  representa o intervalo  $\{s_i, \dots, t_i\}$ . Problema: Queremos decidir se existem dois intervalos encaixados, ou seja, se existem  $i$  e  $j$  tais que  $s_i \leq s_j$  e  $t_i \geq t_j$ . Desafio: resolver o problema em tempo  $O(k + n)$ .

**Exr 5.14 [Degree computation]** Alice deseja fazer uma festa e está decidindo a quem convidar. Há  $n$  pessoas que são candidatas a serem convidadas. Ela preparou uma lista de pares de pessoas que se conhecem. Alice deseja convidar o maior número possível de pessoas de tal forma que na festa cada pessoa conheça pelo menos cinco pessoas e não conheça pelo menos outras cinco. Descreva um algoritmo eficiente que receba a lista das  $n$  pessoas e a lista dos pares que se conhecem e devolva uma solução do problema de Alice. Qual o consumo de tempo do seu algoritmo? Uma resposta satisfatória deve ser um algoritmo que consome  $O(n^3)$ .

# Capítulo 6

## Dividir para conquistar

### 6.1 Busca binária

**Exr 6.1** [Busca linear, CLRS 2.1-3] Escreva e analise um algoritmo iterativo que verifique se  $v$  é elemento de um vetor  $A[p \dots r]$ . (Para que valores de  $p$  e  $r$  o problema faz sentido?) O algoritmo deve devolver  $i$  tal que  $A[i] = v$  ou NIL se tal  $i$  não existe.<sup>1</sup>

**Exr 6.2** [Busca em vetor ordenado] Dado um vetor crescente  $A[p \dots r]$  e um número  $v$ , verificar se  $v$  é elemento do vetor. Escreva e analise um algoritmo iterativo que resolva o problema devolvendo  $j$  tal que

$$A[j] \leq v < A[j + 1]$$

(isso é mais simples que devolver  $i$  tal que  $A[i] = v$ ). Quais os valores possíveis de  $j$ ? Escreva duas versões: uma de busca linear e outra de busca binária.

**Exr 6.3** [CLRS 2.3-5] Considere o algoritmo de busca binária (os dados do algoritmo são um vetor de  $n$  elementos e um número).<sup>2</sup> Mostre que consumo de tempo do algoritmo no pior caso é  $\Omega(\lg n)$ . Mostre que consumo de tempo do algoritmo (em todos os casos) é  $\Omega(1)$ . (Na versão mais “sophisticada” do algoritmo, o consumo de tempo do algoritmo é  $\Omega(\lg n)$ .)

**Exr 6.4** Um vetor  $A[1 \dots n]$  de inteiros é dito *semi-compacto* se  $A[i+1] - A[i] \leq 1$  para  $i = 1, \dots, n-1$ . Escreva um algoritmo que receba um vetor semi-compacto  $A[1 \dots n]$  e um inteiro  $x$  tais que  $A[1] \leq x \leq A[n]$  e devolva um índice  $i$  em  $\{1, \dots, n\}$  tal que  $A[i] = x$ . Seu algoritmo deve consumir tempo  $O(\lg n)$ . Explique sucintamente por que seu algoritmo está correto e tem o consumo de tempo pedido.

**Exr 6.5** [Busca binária, CLRS 2.3-5] Faça uma análise do consumo de tempo da versão recursiva da busca binária. (Veja exercício 2.8.)

---

<sup>1</sup> Acho que o algoritmo ficaria mais bonito se devolvesse  $p - 1$  (ou  $r + 1$ ) quando  $v$  não está em  $A[p \dots r]$ .

<sup>2</sup> Veja também o exercício 6.6.

**Exr 6.6** [CLRS 2.3-5] Escreva uma versão recursiva do algoritmo de busca binária (o algoritmo deve decidir se um dado número  $x$  é igual a algum dos elementos de um vetor crescente  $A$ ).<sup>3</sup> Analise o consumo de tempo do algoritmo.

**Exr 6.7** Escreva um algoritmo que calcule  $\lfloor \sqrt{n} \rfloor$  para qualquer número natural  $n \geq 2$ . O seu algoritmo só pode usar as operações de soma, subtração, multiplicação e divisão e deve consumir tempo  $O(\lg n)$ .

**Exr 6.8** Se uma busca binária em um vetor com  $n$  elementos consome tempo  $T$  no pior caso, quanto tempo consome (no pior caso) uma busca binária em um vetor com  $n^2$  elementos?

**Exr 6.9** É dado um número inteiro  $s$  e um vetor crescente  $A[1..n]$  de números inteiros. Quero saber se existem dois elementos do vetor cuja soma é exatamente  $s$ . Dê um algoritmo que resolva o problema em tempo  $O(\lg n)$ .

**Exr 6.10** [BB 7.12] Se  $A[1..n]$  um vetor estritamente crescente de inteiros (não necessariamente todos positivos). Problema: encontrar  $i$  tal que  $A[i] = i$ . Escreva um algoritmo eficiente que resolva o problema. Faça uma análise do consumo de tempo do algoritmo.

**Exr 6.11** Suponha dado um vetor  $A[1..99500]$  cujos elementos pertencem ao conjunto  $\{0, \dots, 99999\}$ . Escreva um algoritmo eficiente para encontrar um número do conjunto  $\{0, \dots, 99999\}$  que não esteja no vetor.

**Exr 6.12** Suponha dado um vetor de  $A[1..n]$  cujos elementos são cadeias de caracteres (*strings*). Suponha dado um programa  $P$  que recebe um segmento inicial  $A[1..k]$  do vetor e decide que a posição  $k$  é *boa* ou *ruim*. O programa  $P$  consome muito tempo. Escreva um algoritmo eficiente que use  $P$  para encontrar uma posição ruim do vetor.

## 6.2 Mergesort

O algoritmo INTERCALA (veja exercício 5.5) recebe um vetor  $A[p..r]$  tal que  $A[p..q]$  e  $A[q+1..r]$  são crescentes e rearranja o vetor todo em ordem crescente:

```

INTERCALA ( $A, p, q, r$ )
1    $n_1 \leftarrow q - p + 1$ 
2    $n_2 \leftarrow r - q$ 
3   aloque vetores  $L[1..n_1 + 1]$  e  $R[1..n_2 + 1]$ 
4   para  $i \leftarrow 1$  até  $n_1$  faça  $L[i] \leftarrow A[p + i - 1]$ 
5   para  $j \leftarrow 1$  até  $n_2$  faça  $R[j] \leftarrow A[q + j]$ 
6    $L[n_1 + 1] \leftarrow R[n_2 + 1] \leftarrow \infty$ 
7    $i \leftarrow j \leftarrow 1$ 
8   para  $k \leftarrow p$  até  $r$  faça
9     se  $L[i] \leq R[j]$ 
```

---

<sup>3</sup> Veja exercício 6.3.

```

0     então  $A[k] \leftarrow L[i]$ 
1          $i \leftarrow i + 1$ 
2     senão  $A[k] \leftarrow R[j]$ 
3          $j \leftarrow j + 1$ 

```

O algoritmo MERGESORT rearranja um vetor  $A[p..r]$  em ordem crescente:

```

MERGESORT ( $A, p, r$ )
1  se  $p < r$ 
2      então  $q \leftarrow \lfloor (p+r)/2 \rfloor$ 
3          MERGESORT ( $A, p, q$ )
4          MERGESORT ( $A, q+1, r$ )
5          INTERCALA ( $A, p, q, r$ )

```

**Exr 6.13** Que acontece se trocarmos “ $\lfloor (p+r)/2 \rfloor$ ” por “ $\lceil (p+r)/2 \rceil$ ” no código do algoritmo MERGESORT?

**Exr 6.14** Escreva e analise uma versão iterativa do algoritmo de ordenação por intercalação (MERGE-SORT). O algoritmo deve rearranjar um vetor  $A[p..r]$  de modo que ele fique em ordem crescente. Use o resultado do exercício 5.5.

### 6.3 Quicksort

(Veja CLRS cap. 7.) Eis o algoritmo de separação que aparece (sob o nome de PARTITION) no livro de Cormen *et al.* [CLRS01]:<sup>4</sup> O algoritmo devolve um índice  $q$  tal que  $p \leq q \leq r$  depois de rearranjar o vetor  $A[p..r]$  de modo que  $A[p..q-1] \leq A[q] < A[q+1..r]$ :

```

SEPARL ( $A, p, r$ )
1   $x \leftarrow A[r]$ 
2   $i \leftarrow p - 1$ 
3  para  $j \leftarrow p$  até  $r - 1$ 
4      faça se  $A[j] \leq x$ 
5          então  $i \leftarrow i + 1$ 
6           $A[i] \leftrightarrow A[j]$ 
7   $A[i+1] \leftrightarrow A[r]$ 
8  devolva  $i + 1$ 

```

O algoritmo QUICKSORT rearranja  $A[p..r]$  em ordem crescente:

```

QUICKSORT ( $A, p, r$ )
1  se  $p < r$ 
2      então  $q \leftarrow \text{SEPARL} (A, p, r)$ 
3          QUICKSORT ( $A, p, q - 1$ )

```

---

<sup>4</sup> A primeira edição do livro [CLR91], p.168 atribuía uma versão muito semelhante a N. Lomuto.

4

QUICKSORT ( $A, q + 1, r$ )

**Exr 6.15** [CLRS 7.1-3] Mostre que o algoritmo SEPAREL consome tempo  $\Theta(n)$  quando aplicado a um vetor com  $n$  elementos.

**Exr 6.16** Submeta ao algoritmo SEPAREL um vetor com  $n$  elementos iguais. Como o algoritmo permuta o vetor recebido? Quantas trocas faz (linhas 6 e 7) entre elementos do vetor?

**Exr 6.17** Suponha que  $A[p \dots r]$  é uma permutação de  $1, 2, \dots, n$ . Que índice SEPAREL ( $A, 1, n$ ) devolve se  $A[n] = 1$ ? E se  $A[n] = 2$ ? E se  $A[n] = n - 1$ ? E se  $A[n] = n$ ?

**Exr 6.18** Digamos que um vetor  $A[1 \dots n]$  está *arrumado* se existe um índice  $i$  em  $1 \dots n$  tal que

$$A[1 \dots i - 1] \leq A[i] \leq A[i + 1 \dots n]$$

(Expressões da forma “ $A[h \dots k] \leq x$ ” significam “ $A[j] \leq x$  para  $j = h, \dots, k$ ”. Note que a definição não exige que  $i$  seja dado explicitamente!) Escreva um algoritmo que decida se um vetor  $A[1 \dots n]$  está ou não arrumado. Em caso afirmativo, o seu algoritmo deve devolver um índice  $i$  que satisfaça as condições da definição. Quanto tempo o seu algoritmo consome?

**Exr 6.19** [CLRS 7.2-2] Qual o consumo de tempo do QUICKSORT quando aplicado a um vetor com  $n$  elementos iguais?

**Exr 6.20** [CLRS 7.2-3] Mostre que o consumo de tempo do QUICKSORT é  $\Omega(n^2)$  quando aplicado a um vetor crescente com  $n$  elementos distintos.

**Exr 6.21 (Variante “Median of five”)** Invente uma variante de SEPAREL que funcione assim: rearranja  $A[p \dots r]$  e devolve  $q$  tal que  $A[p \dots q-1] \leq A[q] < A[q+1 \dots r]$  e  $p + 2 \leq q \leq r - 2$ . (É claro que isso só faz sentido se o vetor tem pelo menos 5 elementos; para vetores menores, use o SEPAREL usual.) Quanto tempo o seu algoritmo consome? Se QUICKSORT usar o seu algoritmo no lugar de SEPAREL, qual será o seu consumo de tempo no pior caso?

**Exr 6.22 (Variante “Median of  $\alpha n$ ”)** Seja  $\alpha$  um número real tak que  $0 < \alpha < \frac{1}{2}$ . Invente uma variante de SEPAREL que funcione assim: rearranja  $A[p \dots r]$  e devolve  $q$  tal que  $A[p \dots q-1] \leq A[q] < A[q+1 \dots r]$  e  $p + \alpha n \leq q \leq r - \alpha n$ , sendo  $n = r - p + 1$ . Quanto tempo o seu algoritmo consome? Se QUICKSORT usar o seu algoritmo no lugar de SEPAREL, qual será o seu consumo de tempo no pior caso?

**Exr 6.23** [CLRS 7.2-5] Seja  $\alpha$  um número real tal que  $0 < \alpha \leq \frac{1}{2}$ . Suponha que a rotina SEPAREL (veja introdução desta seção) divide o vetor sempre na proporção  $(\alpha, 1-\alpha)$ . Mostre que a profundidade mínima de uma folha na árvore de recursão do algoritmo QUICKSORT é aproximadamente  $-\lg n / \lg \alpha$ . Mostre que a profundidade máxima de uma folha é aproximadamente  $-\lg n / \lg(1-\alpha)$ .

**Exr 6.24** Considere a classe de recorrências  $F(n) = F(\alpha n) + F((1-\alpha)n) + \Theta(n)$ , onde  $\alpha$  é uma constante no intervalo semi-aberto  $(0, \frac{1}{2}]$ . Mostre que  $F(n) = O(n \lg n)$ .

**Exr 6.25** Considere a função  $F$  definida pela recorrência  $F(1) = 1$  e  $F(n) = F(\lceil n/3 \rceil) + F(\lfloor 2n/3 \rfloor) + 5n$  para  $n = 2, 3, 4, \dots$ . Mostre que  $F(n) = O(n \lg n)$ . (Veja exercícios 3.33 e 1.46.)

**Exr 6.26 [CLRS 7.4-3]** Mostre que  $k^2 + (n - k - 1)^2$  atinge o máximo para  $0 \leq k \leq n - 1$  quando  $k = 0$  ou  $k = n - 1$ .<sup>5</sup>

**Exr 6.27** Seja  $S$  a função definida sobre os inteiros positivos pela seguinte recorrência:  $S(0) = S(1) = 1$  e

$$S(n) = \max_{0 \leq k < n} (S(k) + S(n-k-1)) + n$$

quando  $n \geq 2$ . Mostre que  $S(n) \leq n^2 + 1$  para  $n \geq 0$ .<sup>6</sup>

**Exr 6.28 [CLRS 7.4-1, modificado]** Considere a função  $S$  definida pela recorrência  $S(0) = S(1) = 1$  e  $S(n) = \max_{0 \leq k < n} (S(k) + S(n-k-1)) + n$  para todo inteiro  $n > 1$ . Mostre que  $S(n) \geq \frac{1}{2}n^2$  para todo  $n \geq 1$ .<sup>7</sup>

**Exr 6.29** Considere a função  $Q$  definida pela recorrência  $Q(0) = Q(1) = 1$  e

$$Q(n) = \min_{0 \leq k < n} (Q(k) + Q(n-k-1)) + n$$

para todo inteiro  $n > 1$ . Mostre que  $Q(n) = \Omega(n \lg n)$ .<sup>8</sup>

**Exr 6.30 [CLRS 7.4-2]** Mostre que o algoritmo QUICKSORT é  $\Omega(n \lg n)$ . Em outras palavras, mostre que o algoritmo consome  $\Omega(n \lg n)$  unidades de tempo para toda instância de tamanho  $n$ .

**Exr 6.31 [Quicksort com variante do Separe]** Suponha dado um algoritmo SEPARAH (o “H” é de Hoare) rearanja qualquer vetor  $A[p..r]$  e devolve um índice  $q$  tal que  $p \leq q < r$  e  $A[i] \leq A[j]$  para todo  $i$  em  $p..q$  e todo  $j$  em  $q+1..r$ . (O algoritmo consome  $O(n)$  unidades de tempo, sendo  $n := r - p + 1$ .) Escreva uma versão do algoritmo QUICKSORT que use o algoritmo SEPARAH. Prove que sua versão do QUICKSORT está correta. Qual a importância da condição  $p \leq q < r$ ? Por que não exigir que o índice  $q$  devolvido por SEPARAH seja tal que  $p + n/4 \leq q < r - n/4$ , onde  $n := r - p + 1$ ?

**Exr 6.32 [Quicksort com variante do Separe]** Suponha dado um algoritmo SEP que permuta os elementos de um vetor dado  $B[p..r]$  e devolve um índice  $q$  tal que  $p \leq q \leq r - 1$  e  $B[i] \leq B[q] \leq B[q + 1] \leq B[j]$  para todo  $i$  em  $\{p, \dots, q - 1\}$  e todo  $j$  em  $\{q + 2, \dots, r\}$ . Use esse algoritmo para escrever uma implementação do algoritmo Quicksort que faça uma permutação em ordem crescente de um vetor  $A[p..r - 1]$ .

**Exr 6.33 [Stooge Sort, CLRS 7-3]** O seguinte algoritmo rearanja  $A[p..r]$  em ordem crescente (mas você não precisa se preocupar com a correção do algoritmo).

<sup>5</sup> Relevante para o exercício 6.27.

<sup>6</sup> Isso mostra, essencialmente, que o QUICKSORT é  $O(n^2)$ .

<sup>7</sup> Isso mostra, essencialmente, que o pior caso do QUICKSORT é  $\Omega(n^2)$ .

<sup>8</sup> Veja exercício 6.30.

```

ORDENA ( $A, p, r$ )
1  se  $A[p] > A[r]$ 
2    então  $A[p] \leftrightarrow A[r]$      $\triangleright$  troque  $A[p]$  com  $A[r]$ 
3  se  $p + 1 < r$ 
4    então  $k \leftarrow \lfloor (r - p + 1)/3 \rfloor$ 
5      ORDENA ( $A, p, r - k$ )
6      ORDENA ( $A, p + k, r$ )
7      ORDENA ( $A, p, r - k$ )

```

Determine, em função de  $n := r - p + 1$ , a ordem de grandeza do número de comparações na linha 1. Dê a resposta em notação O.

**Exr 6.34** O seguinte algoritmo rearranja  $A[p \dots r]$  em ordem crescente.

```

SORT ( $A, p, r$ )
1  se  $A[p] > A[r]$ 
2    então  $A[p] \leftrightarrow A[r]$      $\triangleright$  troque  $A[p]$  com  $A[r]$ 
3  se  $p + 1 < r$ 
4    então  $k \leftarrow \lfloor (r - p + 1)/3 \rfloor$ 
5      SORT ( $A, p, r - k$ )
6      SORT ( $A, p + k, r$ )
7      SORT ( $A, p, r - k$ )

```

Determine, em função de  $n = r - p + 1$ , a ordem de grandeza do número de comparações na linha 1. Dê a resposta em notação O.

**Exr 6.35** [Stack depth, CLRS 7-4] Considere a seguinte variante do algoritmo QUICKSORT:

```

QUICKSORT' ( $A, p, r$ )
1  enquanto  $p < r$  faça
2     $q \leftarrow \text{SEPARL} (A, p, r)$ 
3    QUICKSORT' ( $A, p, q - 1$ )
4     $p \leftarrow q + 1$ 

```

Mostre que a pilha de recursão pode atingir altura proporcional a  $n$ , onde  $n := r - p + 1$ . Modifique o código de modo que a pilha de recursão tenha altura  $O(\lg n)$ . (Veja enunciado completo em CLRS p.162.)

**Exr 6.36** [Randomized Quicksort, CLRS 7-2] Considere a seguinte versão aleatorizada (= *randomized*) do QUICKSORT:

```

QUICKSORT-ALEATORIZADO ( $A, p, r$ )
1  se  $p < r$ 
2    então  $q \leftarrow \text{SEPARL-ALEATORIZADO} (A, p, r)$ 
3    QUICKSORT-ALEATORIZADO ( $A, p, q - 1$ )
4    QUICKSORT-ALEATORIZADO ( $A, q + 1, r$ )

```

SEPARL-ALEATORIZADO ( $A, p, r$ )

- 1  $i \leftarrow \text{RANDOM}(p, r)$
- 2  $A[i] \leftrightarrow A[r]$
- 3 devolva  $\text{SEPARL}(A, p, r)$

Faça uma análise do desempenho médio do **QUICKSORT-ALEATORIZADO** com base no seguinte roteiro. A. Defina a variável aleatória  $X_i$  como 1 se o  $i$ -ésimo menor elemento de  $A[p..r]$  é escolhido como pivô e como 0 em caso contrário. Calcule  $E[X_i]$ . B. Seja  $T(n)$  a variável aleatória que dá o consumo de tempo do Quicksort ( $n$  é o número de componentes do vetor). Mostre que

$$E[T(n)] = E \left[ \sum_{k=1}^n X_k (T(k-1) + T(n-k) + \Theta(n)) \right].$$

C. Mostre que essa recorrência se reduz a  $E[T(n)] = \left(\frac{2}{n} \sum_{k=0}^{n-1} E[T(k)]\right) + \Theta(n)$ . D. Mostre que  $\sum_{k=1}^{n-1} k \lg k \leq \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2$ . E. Mostre que  $E[T(n)] = \Theta(n \lg n)$ .

**Exr 6.37** É verdade o consumo de tempo do algoritmo **QUICKSORT-ALEATORIZADO** do exercício 6.36 é  $\Omega(n^2)$  no pior caso?

**Exr 6.38** Mostre que  $(n!)^2 > n^n$  para todo número natural não-nulo  $n$ .

## 6.4 Mediana generalizada

Veja CLRS cap.9.

Os problemas desta seção estão definidos sobre *conjuntos* e não *sequências*. Um conjunto é, muitas vezes, representado por um vetor cujos elementos são distintos dois a dois.

O  $i$ -ésimo menor elemento de um conjunto  $S$  de números é o elemento  $x$  de  $S$  tal que  $|\{s \in S : s \leq x\}| = i$ . (Se  $s_1, s_2, \dots, s_n$  é uma enumeração dos elementos de  $S$  tal que  $s_1 < s_2 < \dots < s_n$  então o  $i$ -ésimo menor elemento é  $s_i$ .) A *mediana* de  $S$  é o  $\lfloor \frac{1}{2}(|S| + 1) \rfloor$ -ésimo<sup>9</sup> elemento de  $S$ .

O seguinte algoritmo recebe um índice  $i$  no intervalo  $1 \dots r-p+1$  e um vetor  $A[p..r]$  cujos elementos são dois a dois diferentes e devolve o valor do  $i$ -ésimo menor elemento do vetor.

- ```

SELECTL ( $A, p, r, i$ )
1  se  $p = r$ 
2    então devolva  $A[p]$ 
3   $q \leftarrow \text{SEPARL}(A, p, r)$ 
4   $k \leftarrow q - p + 1$ 
5  se  $k = i$ 
6    então devolva  $A[q]$ 
7  se  $k > i$ 
8    então devolva  $\text{SELECTL}(A, p, q - 1, i)$ 
9    senão devolva  $\text{SELECTL}(A, q + 1, r, i - k)$ 

```

Versão melhor: recebe um índice  $t$  tal que  $p \leq t \leq r$  e um vetor  $A[p..r]$  cujos elementos são dois a dois diferentes e devolve o valor do  $(t - p + 1)$ -ésimo menor elemento do vetor:

---

<sup>9</sup> Poderia ter definido a mediana como o  $\lceil \frac{1}{2}(|S| + 1) \rceil$ -ésimo elemento. Isso só faz diferença quando  $|S|$  é par.

```

SELECTL ( $A, p, t, r$ )
1    $q \leftarrow \text{SEPARL} (A, p, r)$ 
2   se  $q = t$ 
3     então devolva  $A[q]$ 
4   senão se  $q > t$ 
5     então devolva SELECTL ( $A, p, t, q - 1$ )
6     senão devolva SELECTL ( $A, q + 1, t, r$ )

```

(O algoritmo SEPARL devolve um índice  $q$  tal que  $p \leq q \leq r$  depois de rearranjar o vetor  $A[p..r]$  de modo que  $A[p..q-1] \leq A[q] < A[q+1..r]$ . Veja início da seção 6.3.)

**Exr 6.39** Escreva uma algoritmo que calcule o segundo menor elemento de um vetor com  $n$  elementos. Escreva uma algoritmo que calcule o terceiro menor elemento de um vetor com  $n$  elementos. O consumo de tempo dos seus algoritmos deve ser  $O(n)$ .

**Exr 6.40** [CLRS 9.1-1] Mostre que o segundo menor elemento de um vetor  $A[1..n]$  pode ser encontrado com não mais que  $n + \lceil \lg n \rceil - 2$  comparações. (Dica: Encontre também o menor elemento.)

**Exr 6.41** Mostre o algoritmo SELECTL descrito acima consome  $O(n^2)$  unidades de tempo no pior caso, sendo  $n := r - p + 1$ . Mostre que o algoritmo consome  $O(n \lg n)$  unidades de tempo em média (supondo todas as permutações de  $A[p..r]$  igualmente prováveis).

**Exr 6.42** Troque “SEPARL” por “SEPARL-ALEATORIZADO” no código do algoritmo SELECTL. O resultado é o algoritmo SELECTL-ALEATORIZADO (= RANDOMIZED-SELECT). Mostre que esse algoritmo funciona corretamente, ou seja, devolve o valor do  $i$ -ésimo menor elemento do vetor  $A[p..r]$ .

**Exr 6.43** [CLRS 9.2-3] Escreva uma versão iterativa do algoritmo SELECTL-ALEATORIZADO.

**Exr 6.44** É verdade que existe um algoritmo que consome  $O(n)$  unidades de tempo em média para encontrar o  $\lfloor \lg n \rfloor$ -ésimo menor elemento de um conjunto de  $n$  números? Dê uma justificativa curta para sua resposta.

**Exr 6.45** Suponha dado um algoritmo SEPAREH, com parâmetros  $(A, p, r)$ , que rearranja os elementos de um vetor  $A[p..r]$  e devolve um índice  $q$  tal que  $p < q \leq r$  e  $A[p..q-1] \leq A[q..r]$ . Suponha que o consumo de tempo do algoritmo é  $\Theta(n)$ , sendo  $n := r - p + 1$ . Use SEPAREH para escrever um algoritmo SELECTH que receba um vetor  $A[p..r]$  e um índice  $i$  tal que com  $1 \leq i \leq r - p + 1$  e devolva o valor do  $i$ -ésimo menor elemento de  $A[p..r]$ .

**Exr 6.46** [CLRS 9.3-5] Suponha dado um algoritmo MEDIANA, com parâmetros  $(A, p, r)$ , que rearranja os elementos de um vetor  $A[p..r]$  de números inteiros e devolve um índice  $q$  tal que  $p \leq q \leq r$  e  $A[q]$  é a mediana<sup>10</sup> de  $A[p..r]$  e  $A[p..q-1] \leq A[q] \leq A[q+1..r]$ . Suponha ainda que o consumo de tempo do algoritmo é linear, ou seja,  $\Theta(n)$ , sendo  $n := r - p + 1$ . Escreva um algoritmo que receba um vetor  $A[p..r]$  de inteiros e um inteiro positivo  $i$ , com  $1 \leq i \leq r - p + 1$ , e devolva o valor

---

<sup>10</sup> Ou uma das medianas.

do  $i$ -ésimo menor elemento de  $A[p..r]$ . O seu algoritmo deve utilizar MEDIANA como subrotina e deve consumir tempo linear. Explique sucintamente por que o seu algoritmo está correto e tem o consumo de tempo pedido.

**Exr 6.47** [CLRS 9.3-8] Sejam  $\langle x_1, \dots, x_n \rangle$  e  $\langle y_1, \dots, y_n \rangle$  duas sequências numéricas estritamente crescentes sem elementos em comum. Escreva um algoritmo que consuma  $O(\lg n)$  unidades de tempo para encontrar a mediana do conjunto  $\{x_1, \dots, x_n\} \cup \{y_1, \dots, y_n\}$ .

**Exr 6.48** [Os  $k$  maiores elementos em ordem] Dados  $n$  números, queremos encontrar os  $k$  menores, em ordem crescente. Mostre como resolver o problema em tempo  $O(n \lg k)$ .

**Exr 6.49** [Os  $k$  maiores elementos em ordem, CLRS 9-1] Dado um vetor  $S[1..n]$  de números inteiros distintos dois a dois, queremos construir um vetor crescente  $B[1..k]$  contendo os  $k$  menores elementos de  $S$ . Para cada uma das idéias abaixo, dê um algoritmo com o melhor desempenho assintótico de pior caso que você puder. Analise o consumo de tempo de cada algoritmo em função de  $n$  e  $k$ .

- Recolha os  $k$  primeiros elementos de  $S$  depois de rearranjo  $S$  em ordem crescente.
- Transforme  $S$  num heap e chame a função EXTRACT-MIN  $k$  vezes.
- Encontre o valor do  $k$ -ésimo menor elemento de  $S$ , faça uma partição de  $S$  em torno desse valor, e finalmente rearranjo os  $k$  menores elementos em ordem crescente.

# Capítulo 7

## Heap

Veja CLRS cap. 6. A estrutura *heap* (em português diz-se, às vezes, árvore hierárquica) é útil para implementar algoritmos de ordenação e filas de prioridades. As posições  $1, \dots, m$  de um vetor  $A[1 \dots m]$  são chamadas *nós*. O nó 1 é a *raiz*. O *filho esquerdo* de um nó  $i$  é  $2i$ , o *filho direito* é  $2i + 1$  e o *pai* é  $\lfloor i/2 \rfloor$ .

Um vetor  $A[1 \dots m]$  é um *max-heap* se  $A[i] \leq A[\lfloor i/2 \rfloor]$  para cada  $i \geq 2$ . O vetor é um *min-heap* se  $A[i] \geq A[\lfloor i/2 \rfloor]$  para cada  $i \geq 2$ .

A *profundidade* (= *depth*) ou *nível* de um nó  $i$  em um heap  $A[1 \dots m]$  é  $\lfloor \lg i \rfloor$ . A *altura* de um nó  $i$  é o comprimento da mais longa sequência de índices em  $1 \dots m$  que têm a forma  $i, 2i, 4i, \dots$ . Os nós de altura 0 são *folhas*. A altura do heap é a altura do nó 1.

**Exr 7.1** Critique a seguinte definição de max-heap: Um vetor  $A[0 \dots n-1]$  é um max heap se  $A[i] \geq A[2i]$  e  $A[i] \geq A[2i+1]$  sempre que os índices fizerem sentido.

**Exr 7.2** É verdade que todo heap tem  $2^p$  nós de profundidade  $p$ ?

**Exr 7.3** Mostre que todo heap de altura  $h$  tem entre  $2^h$  e  $2^{h+1} - 1$  nós.

**Exr 7.4** [CLRS 6.1-2] Mostre que todo heap  $A[1 \dots m]$  tem altura  $\lfloor \lg m \rfloor$ .

**Exr 7.5** [CLRS 6.1-2] Seja  $i$  um nó de um heap  $A[1 \dots m]$ . Mostre que a altura de  $i$  no heap é  $\lfloor \lg \frac{m}{i} \rfloor$ .

**Exr 7.6** Suponha que  $A[1 \dots m]$  é um heap. Quais são os índices que estão no subheap do nó  $i$ ?

**Exr 7.7** Seja  $h$  a altura de um nó  $i$  em um heap. Considere o subheap que tem raiz  $i$ . Mostre que o subheap tem entre  $2^h$  e  $2^{h+1} - 1$  nós.

**Exr 7.8** [CLRS 6.1-7] Mostre que as folhas de um heap  $A[1 \dots m]$  são  $\lfloor \frac{m}{2} \rfloor + 1, \lfloor \frac{m}{2} \rfloor + 2, \dots, m$ .

**Exr 7.9** [CLRS 6.3-3] Suponha que  $A[1 \dots m]$  é um heap. Mostre que o heap tem no máximo  $\lceil m/2^{h+1} \rceil$  nós com altura  $h$ . Mostre que o heap tem no mínimo  $\lfloor m/2^{h+1} \rfloor$  nós com altura  $h$ .

**Exr 7.10** [CLRS p.135] Mostre que  $\lceil m/2^{h+1} \rceil \leq m/2^h$  quando  $h \leq \lfloor \lg m \rfloor$ .

**Exr 7.11** [CLR p.144] Considere um heap  $A[1..m]$ . A raiz do heap é o elemento de índice 1. Seja  $m'$  o número de elementos do “subheap esquerdo”, cuja raiz é o elemento de índice 2. Seja  $m''$  o número de elementos do “subheap direito”, cuja raiz é o elemento de índice 3. Mostre que  $m'' \leq m' < 2m/3$ .

**Exr 7.12** [CLRS 6.1-4] Suponha que os valores todos os elementos de um max-heap são distintos. Onde pode estar o elemento de valor mínimo?

**Exr 7.13** Suponha os elementos de um vetor  $A[1..2^k-1]$ , com  $k > 3$ , são distintos dois a dois. Suponha ainda que o vetor está organizado como um max-heap. Onde pode estar o terceiro maior elemento do vetor? Onde pode estar o terceiro menor elemento?

**Exr 7.14** O que é um min-heap? O que é um max-heap?

## 7.1 Construção de um heap

Queremos rearranjar um vetor  $A[1..n]$  de tal forma que ele seja um heap.

**Exr 7.15** Os três algoritmos abaixo rearranjam um vetor  $A[1..n]$  de números inteiros positivos. Qual dos três produz um heap? Calcule uma cota superior, em notação O, do consumo de tempo de cada um dos algoritmos.

ALGOH1 ( $A, n$ )  
 1 para  $m \leftarrow 2$  até  $n$  faça  
 2      $i \leftarrow m$   
 3     enquanto  $i > 1$  e  $A[\lfloor i/2 \rfloor] < A[i]$  faça  
 4         troque  $A[\lfloor i/2 \rfloor] \leftrightarrow A[i]$   
 5          $i \leftarrow \lfloor i/2 \rfloor$

ALGOH2 ( $A, n$ )  
 1 para  $m \leftarrow \lfloor n/2 \rfloor$  decrescendo até 1 faça  
 2      $j \leftarrow 2m$   
 3     enquanto  $j \leq n$  faça  
 4         se  $j < n$  e  $A[j] < A[j+1]$  então  $j \leftarrow j + 1$   
 5         se  $A[\lfloor j/2 \rfloor] \geq A[j]$   
 6             então  $j \leftarrow n + 1$   
 7         senão troque  $A[\lfloor j/2 \rfloor] \leftrightarrow A[j]$   
 8          $j \leftarrow 2j$

ALGOH3 ( $A, n$ )  
 1 para  $m \leftarrow 1$  até  $\lfloor n/2 \rfloor$  faça  
 2      $j \leftarrow 2m$   
 3     enquanto  $j \leq n$  faça  
 4         se  $j < n$  e  $A[j] < A[j+1]$  então  $j \leftarrow j + 1$

```

5      se  $A[\lfloor j/2 \rfloor] \geq A[j]$ 
6          então  $j \leftarrow n + 1$ 
7      senão troque  $A[\lfloor j/2 \rfloor] \leftrightarrow A[j]$ 
8           $j \leftarrow 2j$ 

```

**Exr 7.16** Quanto tempo é necessário para rearranjar um vetor  $A[1..n]$  de modo a transformá-lo em um max-heap?

## 7.2 Heapsort

O algoritmo ACERTA-DESCENDO (também conhecido como FIXDOWN ou PENEIRA) recebe  $A[1..m]$  e  $i \geq 1$  tais que os subheaps com raizes  $2i$  e  $2i + 1$  são max-heaps. O algoritmo rearranja o vetor de modo que o subheap com raiz  $i$  seja um max-heap.

```

ACERTA-DESCENDO ( $A, m, i$ )
1    $e \leftarrow 2i$ 
2    $d \leftarrow 2i + 1$ 
3   se  $e \leq m$  e  $A[e] > A[i]$ 
4       então  $x \leftarrow e$ 
5       senão  $x \leftarrow i$ 
6   se  $d \leq m$  e  $A[d] > A[x]$ 
7       então  $x \leftarrow d$ 
8   se  $x \neq i$ 
9       então  $A[i] \leftrightarrow A[x]$ 
0           ACERTA-DESCENDO ( $A, m, x$ )

```

O algoritmo HEAPSORT rearranja  $A[1..n]$  em ordem crescente:

```

HEAPSORT ( $A, n$ )
1  para  $i \leftarrow \lfloor n/2 \rfloor$  decrescendo até 1
2      faça ACERTA-DESCENDO ( $A, n, i$ )
3  para  $m \leftarrow n$  decrescendo até 2
4      faça  $A[1] \leftrightarrow A[m]$ 
5          ACERTA-DESCENDO ( $A, m - 1, 1$ )

```

**Exr 7.17** Discuta a seguinte variante do algoritmo ACERTA-DESCENDO:

```

ACERTA-DESCENDO2 ( $A, m, i$ )
1    $j \leftarrow i$ 
2   enquanto  $2j \leq m$  faça
3       se  $2j < m$  e  $A[2j] > A[2j+1]$ 
4           então  $f \leftarrow 2j$  senão  $f \leftarrow 2j+1$ 
6       se  $A[j] \geq A[f]$ 
7           então  $j \leftarrow m$ 

```

```

8      senão troque  $A[j] \leftrightarrow A[f]$ 
9           $j \leftarrow f$ 

```

**Exr 7.18** Discuta a seguinte variante do algoritmo ACERTA-DESCENDO:

```

ACERTA-DESCENDO3 ( $A, m, i$ )
0    $j \leftarrow f \leftarrow i$ 
1   se  $2j \leq m$ 
2       então se  $2j < m$  e  $A[2j] > A[2j+1]$ 
3           então  $f \leftarrow 2j$  senão  $f \leftarrow 2j+1$ 
4   enquanto  $A[j] < A[f]$  faça
5       troque  $A[j] \leftrightarrow A[f]$ 
6        $j \leftarrow f$ 
7       se  $2j \leq m$ 
8           então se  $2j < m$  e  $A[2j] > A[2j+1]$ 
9               então  $f \leftarrow 2j$  senão  $f \leftarrow 2j+1$ 

```

**Exr 7.19** Discuta a seguinte variante do algoritmo ACERTA-DESCENDO:

```

M-H ( $A, m, i$ )
1    $l \leftarrow 2i$ 
2    $r \leftarrow 2i + 1$ 
3   se  $l \leq m$  e  $A[l] > A[i]$ 
4       então  $A[i] \leftrightarrow A[l]$ 
5       M-H ( $A, m, l$ )
6   se  $r \leq m$  e  $A[r] > A[i]$ 
7       então  $A[i] \leftrightarrow A[r]$ 
8       M-H ( $A, m, r$ )

```

**Exr 7.20** Escreva um algoritmo que receba um min-heap  $A[1..n]$  e rearranja o vetor de modo que ele fique decrescente. O código do seu algoritmo deve estar “completo”, ou seja, não deve invocar outros algoritmos.

**Exr 7.21** Escreva um algoritmo que receba um heap  $A[1..n]$  e um número  $x$  e devolve um índice  $i$  tal que  $A[i] = x$  ou devolve 0 se tal  $i$  não existe. Procure escrever um algoritmo eficiente. Qual o consumo de tempo do seu algoritmo?

**Exr 7.22** Considere a recorrência  $T(1) = 1$  e  $T(m) \leq T(\lfloor 2m/3 \rfloor) + 5$  para  $m \geq 2$ . Mostre que  $T(m) = O(\lg m)$ . Mais geral: mostre que se  $T(m) = T(\lfloor 2m/3 \rfloor) + O(1)$  então  $O(\log m)$ . (Curiosidade: Essa é a recorrência do ACERTA-DESCENDO ( $A, m, i$ ) se interpretarmos  $m$  como sendo o número de nós na subárvore com raiz  $i$ ).

**Exr 7.23** Seja  $z$  um número inteiro não-negativo. Considere a função  $f$  definida por  $f(0) = z$  e  $f(n) = \lceil 2f(n-1)/3 \rceil$  para  $n = 1, 2, 3, 4, \dots$ . Itere a recorrência para mostrar que  $f(n) < 2^n z / 3^n + 2$  para todo  $n$ .

**Exr 7.24** [CLRS 6.2-5] Escreva uma versão iterativa do algoritmo ACERTA-DESCENDO. Faça uma análise do consumo de tempo do algoritmo.

**Exr 7.25** [CLRS 6-1] O algoritmo abaixo transforma  $A[1 \dots n]$  em um max-heap? Qual o invariante no início de cada iteração do bloco de linhas 2–5? Qual o consumo de tempo do algoritmo?

```
B-H ( $A, n$ )
1 para  $m \leftarrow 2$  até  $n$  faça
2    $i \leftarrow m$ 
3   enquanto  $i > 1$  e  $A[\lfloor i/2 \rfloor] < A[i]$  faça
4      $A[\lfloor i/2 \rfloor] \leftrightarrow A[i]$  ▷ troca
5      $i \leftarrow \lfloor i/2 \rfloor$ 
```

**Exr 7.26** [CLRS 6-1] O algoritmo abaixo transforma  $A[1 \dots n]$  em um max-heap? Qual o invariante no início de cada iteração da linha 2? Qual o consumo de tempo do algoritmo?

```
B-H ( $A, n$ )
1 para  $m \leftarrow 1$  até  $n - 1$  faça
2   MAX-HEAP-INSERT( $A, m, A[m + 1]$ )
```

**Exr 7.27** O que faz o algoritmo BUILD-MAX-HEAP? Escreva o algoritmo BUILD-MAX-HEAP (não é preciso escrever o código do algoritmo auxiliar ACERTA-DESCENDO). Quanto tempo BUILD-MAX-HEAP consome se for aplicado a um vetor decrescente (dê a resposta em notação  $\Theta$ )? Explique.

**Exr 7.28** A seguinte afirmação é verdadeira ou falsa? “Qualquer algoritmo que rearranje um vetor  $A[1 \dots n]$  de modo que ele se torne um max-heap consome  $\Omega(n \lg n)$  unidades de tempo.” Comente e explique.

**Exr 7.29** Aplique o algoritmo HEAPSORT a um vetor  $A[1 \dots n]$  em que  $n = 2^{p+1} - 1$  para algum  $p$ . Faça um cálculo detalhado do consumo de tempo.

**Exr 7.30** Mostre que o consumo de tempo do HEAPSORT é  $\Omega(n)$ , sendo  $n$  o número de elementos do vetor. (Em outras palavras, mostre que para toda instância do problema o algoritmo consome  $\Omega(n)$  unidades de tempo.)

**Exr 7.31** Mostre que o consumo de tempo do HEAPSORT no melhor caso é  $O(n)$ , sendo  $n$  o número de elementos do vetor. (Em outras palavras, mostre que existe uma instância do problema para a qual o algoritmo consome  $O(n)$  unidades de tempo.)

**Exr 7.32** [CLRS 6.4-4] Mostre que o consumo de tempo do HEAPSORT no pior caso é  $\Omega(n \lg n)$ , sendo  $n$  o número de elementos do vetor. (Em outras palavras, mostre que existe uma instância do problema para a qual o algoritmo consome  $\Omega(n \lg n)$  unidades de tempo.)

**Exr 7.33** [CLRS 6.4-5 \*] Consider as instâncias  $A[1 \dots n]$  do Heapsort cujos elementos são dois a dois distintos. Mostre o consumo de tempo do HEAPSORT no melhor caso é  $\Omega(n \lg n)$ .

### 7.3 Filas de prioridades

Nesta seção vamos analisar não um determinado algoritmo mas toda uma família de algoritmos semelhantes. É irrelevante o que cada algoritmo faz, mas todos eles têm em comum a manipulação de uma estrutura de dados conhecida como fila de prioridades. Nossa estudo se ocupa, exatamente, da parte do consumo de tempo do algoritmo devida à manipulação dessa estrutura.

**Exr 7.34 [CLRS 6.5-5]** Prove que HEAP-INCREASE-KEY está correto. Use o seguinte invariante: no início de cada iteração,  $A[1 \dots m]$  é um max-heap exceto talvez pela violação da relação  $A[\lfloor i/2 \rfloor] \geq A[i]$ .

**Exr 7.35 [CLRS 6.5-7]** Escreva uma implementação eficiente da operação MAX-HEAP-DELETE com parâmetros  $A, m, i$ . Ela deve remover o nó  $i$  do max-heap  $A[1 \dots m]$  e armazenar os elementos restantes, em forma de max-heap, no vetor  $A[1 \dots m-1]$ .

**Exr 7.36** Uma fila de prioridades é *estável* se os itens com uma mesma chave são removidos da fila na mesma ordem em que foram inseridos. Descreva como implementar uma fila de prioridades estável de modo que as operações de remoção e inserção consumam tempo logarítmico (isto é, se a fila tem  $n$  itens então o consumo deve ser  $O(\lg n)$ ).

**Exr 7.37 [CLRS 6.5-8]** Suponha dados  $k$  vetores crescentes com um total de  $n$  elementos. (Você pode imaginar um vetor  $A[1 \dots n]$  e índices  $j_0, j_1, \dots, j_k$  tais que  $1 = j_0 < j_1 < j_2 < \dots < j_k = n$  e  $A[j_{i-1} \dots j_i]$  é crescente para  $i = 1, \dots, k$ .) Dê um algoritmo que gaste tempo  $O(n \lg k)$  para reunir os  $k$  vetores em um único vetor crescente. (Sugestão: Use um min-heap.)

**Exr 7.38 [BB 5.22]** Escreva um algoritmo que receba um heap  $A[1 \dots n]$  e um número  $x$  e devolva um índice  $i$  tal que  $A[i] = x$  ou devolva 0 se tal  $i$  não existe. Procure escrever um algoritmo eficiente. Qual o consumo de tempo do seu algoritmo?

**Exr 7.39** Cada paciente de um hospital tem uma certa altura  $a$  e um certo peso  $p$ . O banco de dados do hospital mantém os pares  $(a, p)$  de todos os seus  $n$  pacientes. Queremos executar cada uma das seguintes operações em tempo  $O(\log n)$ :

**INSIRA( $a, p$ )**: insere um paciente de altura  $a$  e peso  $p$  no banco de dados e incrementa  $n$  de 1.

**MEDIA( $a, a'$ )**: devolve a média dos pesos dos pacientes que têm altura entre  $a$  e  $a'$ .

Descreva a estrutura de dados e o método usado para atualizar a estrutura. (Dica: para determinar a média dos elementos de um subconjunto basta saber a soma dos elementos do subconjunto e sua cardinalidade.)

**Exr 7.40 [CLRS 6-3]** Um *tablô de Young* é uma matriz cada uma de cujas linhas é crescente (da esquerda para a direita) e cada uma de cujas colunas é crescente (de cima para baixo). Alguns dos componentes da matriz podem ter valor  $\infty$ . Assim, um tablô de Young com  $m$  linhas e  $n$  colunas armazena  $r \leq mn$  números finitos.

(a) Desenhe um tablô de Young  $4 \times 4$  com componentes  $\{9, 16, 3, 2, 4, 8, 5, 14, 12\}$ . (b) Seja  $Y$  um tablô de Young  $m \times n$ . Mostre que  $Y$  está vazio se  $Y[1, 1] = \infty$ . Mostre que  $Y$  está cheio se  $Y[m, n] < \infty$ . (c) Dê um algoritmo que implemente a operação EXTRACT-MIN em um tablô de

Young  $m \times n$  não-vazio. O consumo de tempo de seu algoritmo deve ser  $O(m + n)$ . Seu algoritmo deve usar uma subrotina recursiva que resolve uma instância de tamanho  $m \times n$  reduzindo-a a uma instância de tamanho  $(m - 1) \times n$  ou a uma instância de tamanho  $m \times (n - 1)$ . (Sugestão: Pense em ACERTA-DESCENDO.) Defina  $T(p)$ , com  $p = m + n$ , com sendo o consumo de tempo máximo de EXTRACT-MIN quando aplicado a um tablô  $m \times n$ . Escreva e resolva uma recorrência para  $T(p)$  que produza a cota superior  $O(m + n)$ . (d) Mostre como inserir um elemento novo em um tablô de Young  $m \times n$  não-cheio em tempo  $O(m + n)$ . (e) Mostre como usar um tablô de Young  $n \times n$  para ordenar uma sequência de  $n^2$  números em tempo  $O(n^3)$ .

**Exr 7.41** Esta questão trata de matrizes com elementos em  $Z \cup \{\infty\}$ , sendo  $Z$  o conjunto dos números inteiros. Um elemento com valor  $\infty$  é tratado como uma posição vaga da matriz. Assim, uma matriz com  $m$  linhas e  $n$  colunas pode armazenar no máximo  $mn$  inteiros. Dizemos que uma matriz está *vazia* se todos os seus elementos são  $\infty$  e está *cheia* se todos os seus elementos estão em  $Z$ . Dizemos que uma matriz está *ordenada* se cada linha está em ordem crescente<sup>1</sup> (da esquerda para a direita), e cada coluna estão em ordem crescente (de cima para baixo).

- a. Desenhe uma matriz ordenada 3-por-3 cujos elementos são 9, 16, 3, 2, 4, 8, 5, 14, 12.
- b. Escreva um algoritmo EXTRAI-MIN que recebe uma matriz ordenada  $A$  com  $m$  linhas e  $n$  colunas, retira da matriz um elemento de valor mínimo, insere um  $\infty$  no lugar do elemento que acabou de retirar e rearranja a matriz de modo que ela continue sendo ordenada. Seu algoritmo deve consumir tempo  $O(m + n)$ . (Dica: Pense no algoritmo de extração de mínimo de um heap.)
- c. Escreva um algoritmo INSERE que recebe um inteiro  $x$  e uma matriz ordenada não-cheia  $A$  com  $m$  linhas e  $n$  colunas, insere  $x$  na matriz (no lugar de algum  $\infty$ ) e rearranja a matriz de modo que ela continue sendo ordenada. Seu algoritmo deve consumir tempo  $O(m + n)$ .
- d. Explique como usar uma matriz ordenada com  $n$  linhas e  $n$  colunas e as operações EXTRAI-MIN e INSERE para ordenar  $n$  números em tempo  $O(n^3)$  (sem recorrer a um outro algoritmo de ordenação).
- e. Escreva um algoritmo BUSCA que recebe como argumentos um número  $x$  e uma matriz ordenada  $A$  com  $m$  linhas e  $n$  colunas e determina se  $x$  está em  $A$  ou não. O seu algoritmo deve consumir tempo  $O(m + n)$ .

---

<sup>1</sup> Uma sequência  $x_1, x_2, \dots, x_n$  está em ordem crescente se  $x_1 \leq x_2 \leq \dots \leq x_n$ .

# Capítulo 8

## Árvores binárias

**Exr 8.1** [CLRS B.5-3] Seja  $T$  uma árvore binária não-vazia com  $k$  folhas. Mostre que  $T$  tem exatamente  $k - 1$  nós de grau 2 (ou seja, nós com 2 filhos). Sugestão: Use indução matemática.

**Exr 8.2** Uma árvore binária é *cheia* se cada um de seus nós tem 0 ou 2 filhos. Um nó é *interno* se tiver 2 filhos. Prove (por indução matemática) que toda árvore binária cheia com  $n$  folhas tem exatamente  $n - 1$  nós internos.

### 8.1 Árvores binárias de busca

**Exr 8.3** Suponha que uma árvore binária tem a seguinte propriedade: para cada nó  $x$ ,

se  $\text{esq}[x] \neq \text{NIL}$  então  $\text{chave}[\text{esq}[x]] \leq \text{chave}[x]$  e  
se  $\text{dir}[x] \neq \text{NIL}$  então  $\text{chave}[\text{dir}[x]] \geq \text{chave}[x]$ .

É verdade que nossa árvore binária é de busca (= *binary search-tree*)? Justifique.

**Exr 8.4** Escreva um algoritmo que receba (a raiz de) uma árvore binária e devolva a altura da árvore.

**Exr 8.5** [CLR 13.1-3] Escreva uma versão iterativa do algoritmo de varredura esquerda-raiz-direita (= *inorder traversal*) de uma árvore binária de busca. (Dica: Há uma solução fácil que usa uma pilha como estrutura de dados auxiliar; há uma solução mais complicada mas elegante que não usa pilha mas supõe que a igualdade de dois ponteiros pode ser testada.) Calcule uma cota superior do consumo de tempo do algoritmo. Diga qual o invariante principal (ou invariantes principais) do seu algoritmo. (A propósito, calcule também uma cota superior do consumo de tempo da versão recursiva do algoritmo, que está no CLR.)

**Exr 8.6** Sejam  $A$  e  $B$  duas árvore binárias. Suponha que cada nó  $x$  de qualquer das árvore armazena um inteiro positivo  $\text{chave}[x]$ . Suponha agora que  $A$  e  $B$  são árvores de busca em relação ao atributo  $\text{chave}$ .

Dizemos que  $A$  se encaixa em  $B$  se (1) para todo nó  $a$  de  $A$  existe um nó  $b$  de  $B$  tal que  $\text{chave}[b] = \text{chave}[a]$  e (2) para todo par  $a, a'$  de nós de  $A$ , se  $a'$  é descendente de  $a$  então existem nós  $b$  e  $b'$  em  $B$  tais que  $b'$  é descendente de  $b$ ,  $\text{chave}[b] = \text{chave}[a]$  e  $\text{chave}[b'] = \text{chave}[a']$ . Escreva um algoritmo

que decide se uma árvore binária de busca  $A$  se encaixa em uma árvore binária de busca  $B$ . O seu algoritmo deve consumir tempo  $O(n)$ , sendo  $n$  a soma dos números de nós de  $A$  e  $B$ .

**Exr 8.7** Considere uma estrutura de dados padrão de dicionário com operações INSERIR, BUSCAR e REMOVER. Desejamos ampliar esse repertório com a operação MIN-GAP que devolve a menor diferença entre dois números do dicionário. Explique como implementar esse dicionário ampliado de tal forma que o consumo de tempo de cada operação seja  $O(\lg n)$  no pior caso, sendo  $n$  o número de elementos do dicionário. Você pode usar qualquer árvore balanceada de busca como caixa preta.

## 8.2 Árvores rubro-negras

**Exr 8.8 [CLR 14.1-3]** Seja  $x$  um nó de uma árvore rubro-negra e considere os caminhos que descem de  $x$  até uma folha. Suponha que um caminho máximo desse tipo tem comprimento  $c^*$  e que um caminho mínimo desse tipo tem comprimento  $c_*$ . Mostre que  $c^* \leq 2c_*$ .

## Capítulo 9

# Análise probabilística e algoritmos aleatorizados

Veja CLRS cap.5 e ap.C.

**Exr 9.1** [CLRS 5.2-3] Calcule o valor esperado da soma de  $n$  dados (*dice*).

**Exr 9.2** Um maço de 10 cartas numeradas de 1 a 10 é embaralhado. A seguir, três cartas são retiradas, uma de cada vez, do maço. Qual a probabilidade de que os números das cartas retiradas estejam em ordem crescente? Especifique precisamente o espaço de probabilidades que está sendo considerado para resolver a questão.

**Exr 9.3** [CLRS 5.2-2] Se  $A[1 \dots n]$  é uma permutação aleatória uniforme (= *random uniform permutation*) de  $1 \dots n$ , qual a probabilidade de que a linha 4 seja executada exatamente duas vezes?

```
MAX ( $A, n$ )
1    $max \leftarrow -\infty$ 
2   para  $i \leftarrow 1$  até  $n$ 
3     faça se  $A[i] > max$ 
4       então  $max \leftarrow A[i]$ 
5   devolva  $max$ 
```

**Exr 9.4** Suponha dado um vetor  $A[1 \dots n]$  com elementos distintos dois a dois. Prove que o seguinte algoritmo gera uma permutação aleatória uniforme de  $A[1 \dots n]$  (ou seja, todas as  $n!$  permutações têm a mesma probabilidade). (*Sugestão:* Use indução em  $n$ .)

```
1  enquanto  $n > 1$ 
2    faça  $i \leftarrow \text{RANDOM} (1, n)$ 
3     $A[i] \leftrightarrow A[n]$ 
4     $n \leftarrow n - 1$ 
```

**Exr 9.5** Suponha dado um gerador de bits aleatórios (distribuição uniforme). Suponha dados números inteiros  $a$  e  $b$ . Construa um gerador de inteiros aleatórios, com probabilidade uniforme, no conjunto  $\{a, a + 1, \dots, b - 1, b\}$ . O seu gerador deve consumir tempo esperado  $O(1)$ .

**Exr 9.6** Considere o seguinte algoritmo, que devolve o índice de um elemento mínimo do vetor  $A[1..n]$ :

```
MÍNIMO ( $A, n$ )
1   se  $n = 1$ 
2     então  $k \leftarrow n$ 
3     senão  $k \leftarrow \text{MÍNIMO} (A, n-1)$ 
4       se  $A[k] > A[n]$ 
5         então  $k \leftarrow n$ 
6   devolva  $k$ 
```

Suponha que os elementos de  $A[1..n]$  são distintos dois a dois. Suponha também que, para cada  $i$ , a probabilidade de que  $A[i]$  é o elemento mínimo do vetor é  $1/n$ . Calcule o número médio, digamos  $T(n)$ , de execuções da atribuição “ $k \leftarrow n$ ” (linhas 2 e 5). Não use notação O. (Sugestão: Escreva uma recorrência para  $T(n)$ .)

**Exr 9.7** Cada componente de  $A[1..n]$  e de  $B[1..n]$  vale 0 ou 1. Se  $A[j] = 1 = B[j]$  para algum  $j$ , o algoritmo devolve 1; caso contrário, devolve 0.

```
ELEMENTO-COMUM ( $A, B, n$ )
1    $j \leftarrow 1$ 
2   enquanto  $j \leq n$  e ( $A[j] \neq 1$  ou  $B[j] \neq 1$ )
3     faça  $j \leftarrow j + 1$ 
4   se  $j \leq n$ 
5     então devolva 1
6   senão devolva 0
```

Suponha que o valor de cada componente de  $A[1..n]$  e  $B[1..n]$  é escolhida ao acaso e independentemente para ser 0 ou 1 com probabilidade  $1/2$ . Mostre que o consumo de tempo esperado do algoritmo é  $O(1)$ . (Dica:  $1/(1-x)^2 = 1 + 2x + 3x^2 + \dots$ )

**Exr 9.8** O código abaixo é uma descrição do algoritmo de Karger para grafos conexos. Modifique o código para que o algoritmo aceite qualquer grafo, mesmo desconexo.

```
KARGER ( $V, E$ )
1    $\mathcal{P} \leftarrow \{\{v\} : v \in V\}$ 
2    $F \leftarrow E$ 
3   enquanto  $|\mathcal{P}| > 2$  faça
4     seja  $\{x, y\}$  um elemento aleatório de  $F$ 
5     seja  $X$  o elemento de  $\mathcal{P}$  que contém  $x$ 
6     seja  $Y$  o elemento de  $\mathcal{P}$  que contém  $y$ 
7      $\mathcal{P} \leftarrow (\mathcal{P} - \{X, Y\}) \cup \{X \cup Y\}$ 
8      $F \leftarrow F - \{f \in F : f \subseteq X \cup Y\}$ 
9   devolva  $\mathcal{P}$ 
```

# Capítulo 10

## Programação dinâmica

Veja CLRS cap.15. Ian Parberry [Par95] diz: “Dynamic programming is a fancy name for divide-and-conquer with a table. Instead of solving subproblems recursively, solve them sequentially and store their solutions in a table. The trick is to solve them in the right order so that whenever the solution to a subproblem is needed, it is already available in the table. Dynamic programming is particularly useful on problems for which divide-and-conquer appears to yield an exponential number of subproblems, but there are really only a small number of subproblems repeated exponentially often. In this case, it makes sense to compute each solution the first time and store it away in a table for later use, instead of recomputing it recursively every time it is needed.”

Na expressão ‘programação dinâmica’, a palavra ‘programação’ é sinônimo de ‘planejamento’ e não tem relação direta com a programação de computadores. De acordo com Robert Sedgewick, ‘programação dinâmica’ é o nome antigo de pesquisa operacional.

### 10.1 Generalidades

**Exr 10.1** O algoritmo recursivo abaixo calcula os números de Fibonacci (veja exercício 3.6). Seja  $T(n)$  o número de somas realizado pelo algoritmo na linha 5. Mostre que  $T(n) = \Omega((\frac{3}{2})^n)$ .

```
FIBO-REC ( $n$ )
1   se  $n \leq 1$ 
2     então devolva  $n$ 
3     senão  $a \leftarrow \text{FIBO-REC} (n - 1)$ 
4            $b \leftarrow \text{FIBO-REC} (n - 2)$ 
5           devolva  $a + b$ 
```

**Exr 10.2** [CLRS 15.3-2] Desenhe a árvore de recursão para o algoritmo MERGESORT aplicado a um vetor de 16 elementos. Por que a técnica de programação dinâmica não é capaz de acelerar o algoritmo?

## 10.2 Multiplicação de cadeias de matrizes

Seja  $A_1$  uma matriz  $p_0$  linhas e  $p_1$  colunas e  $A_2$  uma matriz  $p_1$  linhas e  $p_2$  colunas. Para calcular a matriz  $A_1 A_2$  é preciso fazer  $p_0 p_1 p_2$  multiplicações escalares entre componentes das matrizes.

Uma *cadeia de matrizes* ( $= matrix chain$ ) é uma sequência  $A_1, A_2, \dots, A_n$  de matrizes tal que o número de colunas de cada matriz é igual ao número de linhas da matriz seguinte.

Problema da Multiplicação de uma Cadeia de Matrizes (*Matrix-Chain Multiplication Problem*): Dada uma cadeia de matrizes, encontrar a ordem em que as matrizes devem ser multiplicadas de modo a minimizar o número total de multiplicações escalares.

O seguinte algoritmo recursivo determina o número mínimo de multiplicações escalares necessário para multiplicar a cadeia  $A_i, \dots, A_j$  de dimensões  $p_{i-1}, p_i, \dots, p_j$ :

```
REC-MATRIXCHAIN ( $p, i, j$ )
1   se  $i = j$ 
2     então devolva 0
3    $x \leftarrow \infty$ 
4   para  $k \leftarrow i$  até  $j - 1$  faça
5      $q_1 \leftarrow$  REC-MATRIXCHAIN ( $p, i, k$ )
6      $q_2 \leftarrow$  REC-MATRIXCHAIN ( $p, k + 1, j$ )
7      $q \leftarrow q_1 + p_{i-1} p_k p_j + q_2$ 
8     se  $q < x$ 
9       então  $x \leftarrow q$ 
0   devolva  $x$ 
```

O algoritmo abaixo aplica a técnica da programação dinâmica ao problema:

```
MATRIXCHAINORDER ( $p, n$ )
1   para  $i \leftarrow 1$  até  $n$  faça
2      $m[i, i] \leftarrow 0$ 
3   para  $l \leftarrow 2$  até  $n$  faça
4     para  $i \leftarrow 1$  até  $n - l + 1$  faça
5        $j \leftarrow i + l - 1$ 
6        $m[i, j] \leftarrow \infty$ 
7       para  $k \leftarrow i$  até  $j - 1$  faça
8          $q \leftarrow m[i, k] + p_{i-1} p_k p_j + m[k+1, j]$ 
9         se  $q < m[i, j]$ 
0           então  $m[i, j] \leftarrow q$ 
1   devolva  $m[1, n]$ 
```

**Exr 10.3** Considere a cadeia de matrizes  $A_1, A_2, A_3, A_4$ . Faça uma lista completa de todas as maneiras de multiplicar essas matrizes, ou seja, de todas as maneiras de inserir parênteses na cadeia.

**Exr 10.4 [CLRS 15.2-1 ]** Suponha dada uma cadeia de matrizes de dimensões  $(5, 10, 3, 12, 5, 50, 6)$ . Encontre uma maneira de fazer a multiplicação da cadeia que minimize o número de multiplicações escalares.

**Exr 10.5** [CLRS 15.2-5] Mostre que são necessários exatamente  $n - 1$  pares de parênteses para especificar uma ordem de multiplicação da cadeia de matrizes  $A_1, A_2, \dots, A_n$ .

**Exr 10.6** [Matrix-chain multiplication] Enuncie e prove a “propriedade da subestrutura ótima” para o Problema da Multiplicação de uma Cadeia de Matrizes.

**Exr 10.7** Seja  $T(n)$  o número de comparações entre  $q$  e  $x$  na linha 8 do algoritmo REC-MATRIXCHAIN. Aqui,  $n$  é o número  $j - i + 1$ . (Observe que  $T(n)$  é proporcional ao consumo de tempo do algoritmo.) Mostre que  $T(n) = \Omega(2^n)$ .

**Exr 10.8** Discuta o comportamento do algoritmo REC-MATRIXCHAIN no caso em que temos apenas uma matriz (ou seja,  $i = j$ ). Repita o exercício no caso de duas matrizes e três matrizes. Discuta o comportamento do algoritmo no caso em que  $p_{i-1} = p_i = \dots = p_j$ . Repita no caso em que  $p_{i-1} \leq p_i \leq \dots \leq p_j$ .

**Exr 10.9** [CLRS 15.3-5] Seja  $A_i, A_{i+1}, \dots, A_j$  uma cadeia de matrizes de dimensões  $p_{i-1}, p_i, \dots, p_j$ . Considere o seguinte algoritmo: primeiro, escolha  $k$  que minimize  $p_k$ ; depois, determine recursivamente as ordens de multiplicação das cadeias  $A_i, \dots, A_k$  e  $A_{k+1}, \dots, A_j$ . Esse algoritmo resolve o problema da multiplicação da cadeia de matrizes?

**Exr 10.10** É verdade que MISTÉRIO ( $1, n$ ) consome  $O(n^3)$  unidades de tempo?

```

MISTÉRIO ( $i, k$ )
1   se  $k = i + 1$ 
2     então devolva 1
3     senão  $x \leftarrow -\infty$ 
4       para  $j \leftarrow i + 1$  até  $k - 1$  faça
5          $y_1 \leftarrow$  MISTÉRIO ( $i, j$ )
6          $y_2 \leftarrow$  MISTÉRIO ( $j, k$ )
7          $x \leftarrow \max(x, y_1 + y_2 + 2j)$ 
8       devolva  $x$ 
```

**Exr 10.11** [CLRS 15.3-3] Considere a seguinte variante do problema da multiplicação de cadeia de matrizes: dada uma cadeia de matrizes  $A_1, A_2, \dots, A_n$ , determinar a maneira de inserir parênteses na cadeia que maximize o número de multiplicações escalares. Este problema tem estrutura recursiva (ou seja, o problema possui a “propriedade da subestrutura ótima”)?

**Exr 10.12** Considere o algoritmo MATRIXCHAINORDER. Mostre que o número de execuções da linha 8 (e portanto também o consumo de tempo) é  $\Theta(n^3)$ .

**Exr 10.13** Discuta o comportamento do algoritmo MATRIXCHAINORDER no caso em que temos apenas uma matriz (ou seja,  $i = j$ ). Repita o exercício no caso de duas matrizes e três matrizes. Discuta o comportamento do algoritmo no caso em que  $p_{i-1} = p_i = \dots = p_j$ . Repita no caso em que  $p_{i-1} \leq p_i \leq \dots \leq p_j$ .

### 10.3 Problema da mochila

Problema Booleano da Mochila (= *Boolean Knapsack Problem*): Dados números inteiros não-negativos  $v_1, \dots, v_n, w_1, \dots, w_n$  e  $W$ ,<sup>1</sup> encontrar um subconjunto  $K$  de  $\{1, \dots, n\}$  que

$$\text{satisfaca } \sum_{k \in K} w_k \leq W \quad \text{e} \quad \text{maximize } \sum_{k \in K} v_k.$$

(Motivação: Tenho  $n$  objetos, numerados de 1 a  $n$ . Cada objeto  $i$  com peso  $w_i$  e valor  $v_i$ . Quero colocar uma seleção dos objetos numa mochila de capacidade  $W$  de modo a maximizar a soma dos valores dos objetos escolhidos.)

**Exr 10.14** [Subset sum, Problema dos cheques, CLRS 16.2-2 simplificado] Escreva um algoritmo de programação dinâmica para o seguinte problema: dados números inteiros não-negativos  $w_1, \dots, w_n$  e  $W$ , encontrar um subconjunto  $K$  de  $\{1, \dots, n\}$  que satisfaca  $\sum_{k \in K} w_k \leq W$  e  $\sum_{k \in K} w_k$  é máximo. (Imagine que  $w_1, \dots, w_n$  são os valores dos cheques que você emitiu durante o mês e  $W$  é o valor que o banco debitou em sua conta no fim do mês. Quais dos cheques foram descontados?)<sup>2</sup> Compare com o problema booleano da mochila e com os exercícios 10.16 e 5.4. (Outra maneira de formular o problema: dado um inteiro não-negativo  $W$  e uma sequência  $\langle w_1, \dots, w_n \rangle$  de inteiros não-negativos, encontrar uma subsequência  $\langle s_1, \dots, s_k \rangle$  de  $\langle w_1, \dots, w_n \rangle$  tal que  $s_1 + \dots + s_k = W$ .)

**Exr 10.15** Discuta o seguinte caso especial do problema *subset sum* (veja exercício 10.14): dados números inteiros não-negativos  $a, b, m, n$  e  $W$  encontrar inteiros não-negativos  $i \leq m$  e  $j \leq n$  tais que  $ia + jb = W$ .

**Exr 10.16** [Instâncias “ $v=w$ ” da mochila booleana] Escreva um algoritmo de programação dinâmica para o seguinte problema: dados números inteiros não-negativos  $w_1, \dots, w_n$  e  $W$ , encontrar um subconjunto  $K$  de  $\{1, \dots, n\}$  que maximize  $\sum_{k \in K} w_k$  sem violar a restrição  $\sum_{k \in K} w_k \leq W$ .<sup>3</sup> Exiba e discuta a propriedade da subestrutura ótima (*optimal substructure property*). (Para simplificar, basta que seu algoritmo devolva a soma  $\sum_{k \in K} w_k$ . Compare com os exercícios 10.14 e 11.3.)

**Exr 10.17** [Mochila booleana, CLRS 16.2-2] Use a técnica da programação dinâmica para resolver o Problema Booleano da Mochila. A solução do problema certamente envolverá uma tabela bidimensional, digamos  $t$ ; diga qual o significado de  $t[i, j]$ . Qual a *optimal substructure property* para esse problema?

**Exr 10.18** [Partição equilibrada] Seja  $S$  o conjunto das raízes quadradas dos números  $1, 2, \dots, 500$ . Escreva e teste um programa que determine uma partição<sup>4</sup>  $\{A, B\}$  de  $S$  tal que a soma dos números

<sup>1</sup> É errado dizer “dado um conjunto  $\{v_1, \dots, v_n\}$ ” pois os números  $v_1, \dots, v_n$  podem não ser distintos dois a dois.

<sup>2</sup> CLRS exige que os  $w_i$  sejam distintos dois a dois. Esse caso particular do problema pode ser formulado assim: dado um conjunto  $T$  de inteiros não-negativos e um inteiro não-negativo  $W$ , encontrar um subconjunto  $S$  de  $T$  tal que  $\sum_{s \in S} s = W$ . Mas este caso especial do problema não é mais fácil que o caso geral.

<sup>3</sup> Acho que CLRS exigiria que os  $w_i$  sejam distintos dois a dois. Esse caso particular do problema pode ser formulado assim: dado um conjunto  $T$  de inteiros não-negativos e um inteiro não-negativo  $W$ , encontrar um subconjunto  $S$  de  $T$  que maximize  $\sum_{s \in S} s$  sem violar a restrição  $\sum_{s \in S} s \leq W$ . Mas este caso particular do problema não é mais fácil que o caso geral.

<sup>4</sup> Uma *partição* de um conjunto  $S$  é uma coleção  $\mathcal{P}$  de subconjuntos de  $S$  tal que cada elemento de  $V$  pertence a exatamente um dos elementos de  $\mathcal{P}$ . Em particular,  $\{A, B\}$  é uma partição de  $S$  se  $A \cup B = S$  e  $A \cap B = \emptyset$ .

em  $A$  seja tão próxima quanto possível da soma dos números em  $B$ . (O seu algoritmo resolve o problema ou dá apenas uma solução “aproximada”?)

Uma vez calculados  $A$  e  $B$ , seu programa deve imprimir a diferença entre a soma de  $A$  e a soma de  $B$  e depois imprimir a lista dos quadrados dos números em um dos conjuntos.

## 10.4 Subsequência comum máxima

Nesta seção, convém falar em *sequências* em lugar de vetores e escrever  $\langle a_1, \dots, a_n \rangle$  em lugar de  $A[1..n]$ .

Uma *subsequência* de uma sequência  $\langle a_1, \dots, a_n \rangle$  é o que sobra depois que um conjunto arbitrário de termos de  $\langle a_1, \dots, a_n \rangle$  é apagado.<sup>5</sup>

Uma subsequência  $\langle z_1, \dots, z_k \rangle$  de  $\langle a_1, \dots, a_n \rangle$  é *crescente* se  $z_1 \leq \dots \leq z_k$ . Uma subsequência  $\langle z_1, \dots, z_k \rangle$  de  $\langle a_1, \dots, a_n \rangle$  é *estritamente decrescente* se  $z_1 > \dots > z_k$ .

**Problema da Subsequência Comum Máxima** (= *Longest Common Subsequence* = *LCS*): Encontrar uma subsequência comum máxima de duas sequências dadas.

**Problema da Subsequência Crescente Máxima:** Encontrar uma subsequência crescente máxima de uma sequência de números.

**Exr 10.19** Suponha que  $\langle b_1, \dots, b_k \rangle$  é uma subsequência de  $\langle a_1, \dots, a_n \rangle$ . 1. Seja  $m$  um índice tal que  $b_k = a_m$ . É verdade que  $\langle b_1, \dots, b_{k-1} \rangle$  é uma subsequência de  $\langle a_1, \dots, a_{m-1} \rangle$ ? 2. Seja  $m$  o maior índice tal que  $b_k = a_m$ . É verdade que  $\langle b_1, \dots, b_{k-1} \rangle$  é uma subsequência de  $\langle a_1, \dots, a_{m-1} \rangle$ ?

**Exr 10.20** Escreva um algoritmo para decidir se  $\langle z_1, \dots, z_k \rangle$  é subsequência de  $\langle x_1, \dots, x_m \rangle$ . Prove rigorosamente que o seu algoritmo está correto. Calcule o consumo de tempo do seu algoritmo.

**Exr 10.21** Escreva um algoritmo para contar o número de ocorrências de  $\langle z_1, \dots, z_k \rangle$  como subsequência de  $\langle x_1, \dots, x_m \rangle$ . O seu algoritmo pode consumir tempo  $O(km)$ .

**Exr 10.22** Dizemos que um vetor  $Z[1..m+n]$  é um *embaralhamento* dos vetores  $X[1..m]$  e  $Y[1..n]$  se  $Z$  é formado pela intercalação de  $X$  e  $Y$  de forma que a ordem relativa dos elementos de  $X$  é mantida e a ordem relativa dos elementos de  $Y$  é mantida. (É claro que cada elemento de  $X[1..m]$  e de  $Y[1..n]$  deve aparecer em  $Z[1..m+n]$ .) Exemplo A: aALnÁGloIrITsMeOS é um embaralhamento de análise e ALGORITMOS. Exemplo B: aALniGloÁRITsMeOS não é um embaralhamento de análise e ALGORITMOS.

Escreva um algoritmo EMBARALHAMENTO que receba vetores  $Z[1..m+n]$ ,  $X[1..m]$  e  $Y[1..n]$  e devolva 1 se  $Z$  é embaralhamento de  $X$  e  $Y$  e devolva 0 em caso contrário. O consumo de tempo do seu algoritmo deve ser  $O(mn)$ . Mostre que o seu algoritmo está correto e tem o consumo de tempo exigido.

**Exr 10.23** Suponha que os elementos de uma sequência  $\langle a_1, \dots, a_n \rangle$  são distintos dois a dois. Quantas subsequências tem a sequência? Quantos segmentos diferentes tem a sequência?

---

<sup>5</sup> Não confunda subsequência com segmento. Um *segmento* de  $\langle a_1, \dots, a_n \rangle$  é o que sobra depois que apagamos um número arbitrário de termos no início da sequência e um número arbitrário de termos no fim da sequência.

**Exr 10.24** Invente uma heurística<sup>6</sup> gulosa para o problema da subsequência comum máxima. Mostre que ela não resolve o problema.

**Exr 10.25** [Longest common subsequence] Digamos que  $c[m, n]$  é o comprimento de uma subsequência comum máxima de  $\langle x_1, \dots, x_m \rangle$  e  $\langle y_1, \dots, y_n \rangle$ . Escreva uma recorrência para  $c[m, n]$  (ou seja, uma recorrência a partir da qual  $c[m, n]$  possa ser calculado).

**Exr 10.26** [CLRS 15.4-2] Suponha dadas sequências  $X = \langle x_1, \dots, x_m \rangle$  e  $Y = \langle y_1, \dots, y_n \rangle$ . Para cada  $i$ , seja  $X_i$  a sequência  $\langle x_1, \dots, x_i \rangle$ . Defina  $Y_j$  analogamente. Suponha dada também uma matriz  $c[0..m, 0..n]$  tal que  $c[i, j]$  é comprimento de uma subsequência comum máxima de  $X_i$  e  $Y_j$ . Escreva um algoritmo que imprima uma subsequência comum máxima de  $X$  e  $Y$ . O consumo de tempo do seu algoritmo deve ser  $O(m + n)$ .

**Exr 10.27** [Subsequência crescente máxima] Uma subsequência crescente  $Z$  de uma sequência  $X$  é máxima se não existe outra subsequência crescente mais longa. A subsequência  $\langle 5, 6, 9 \rangle$  de  $\langle 9, 5, 6, 9, 6, 2, 7 \rangle$  é máxima? Dê uma sequência crescente máxima de  $\langle 9, 5, 6, 9, 6, 2, 7 \rangle$ . Mostre que o algoritmo guloso óbvio não é capaz, em geral, de encontrar uma subsequência crescente máxima de uma sequência dada. (Algoritmo guloso óbvio: escolha o menor elemento de  $X$ ; a partir daí, escolha sempre o próximo elemento de  $X$  que seja maior ou igual ao último escolhido.)

**Exr 10.28** [CLRS 15.4-5] Mostre como o algoritmo da subsequência comum máxima pode ser usado para resolver o problema da subsequência crescente máxima de uma sequência de números inteiros (veja exercício 10.27). Dê uma cota justa, em notação  $\Theta$ , do consumo de tempo de sua solução.

**Exr 10.29** Considere o seguinte problema SSECM: dada uma sequência  $\langle x_1, x_2, \dots, x_n \rangle$  de números, encontrar uma subsequência máxima dentre as que são estritamente crescentes. (Veja exercício 10.28.) Parte 1: Suponha conhecido o clássico algoritmo LCS para o problema da subsequência comum máxima de duas sequências. Mostre (em português, sem escrever pseudocódigo) como o algoritmo LCS pode ser usado para resolver o problema SSECM. Parte 2: Prove, cuidadosamente, que sua proposta na parte 1 produz uma solução correta do problema SSECM. Parte 3: Qual o consumo de tempo de sua proposta em função de  $n$ ? Dê uma cota justa, em notação  $\Theta$ .

**Exr 10.30** Considere o problema da subsequência crescente máxima (veja exercício 10.27). Um analista de algoritmos afirma que o problema possui a seguinte propriedade da subestrutura ótima: Se  $\langle z_1, \dots, z_k \rangle$  é uma subsequência crescente máxima de  $\langle a_1, \dots, a_n \rangle$  então (1)  $\langle z_1, \dots, z_k \rangle$  é uma subsequência crescente máxima de  $\langle a_1, \dots, a_{n-1} \rangle$  ou (2)  $z_k = a_n$  e  $\langle z_1, \dots, z_{k-1} \rangle$  é uma subsequência crescente máxima de  $\langle a_1, \dots, a_{n-1} \rangle$ . O analista está certo?

**Exr 10.31** Digamos que  $\langle z_1, \dots, z_k \rangle$  é uma subsequência crescente máxima de  $\langle a_1, \dots, a_n \rangle$ . Suponha que  $z_k = a_n$ . É verdade que  $\langle z_1, \dots, z_{k-1} \rangle$  é uma subsequência crescente máxima de  $\langle a_1, \dots, a_{n-1} \rangle$ ?

**Exr 10.32** Escreva um algoritmo de programação dinâmica para resolver o problema da subsequência crescente máxima (veja exercício 10.27). O seu algoritmo deve resolver o problema diretamente, sem usar o algoritmo da subsequência comum máxima como subrotina.

---

<sup>6</sup> Uma heurística é um “algoritmo” que não garante resolver o problema. A palavra *heurística* é às vezes usada como sinônimo de *estratégia*.

**Exr 10.33** [CLR 16.3-6★] Dê um algoritmo que gaste  $O(n \lg n)$  unidades de tempo para encontrar uma subsequência crescente (SSC) de comprimento máximo de uma sequência de  $n$  números. (Dica: Observe que o último elemento de uma subsequência-candidata de comprimento  $i$  não é menor que o último elemento de uma candidata de comprimento  $i - 1$ . Mantenha uma lista encadeada de subsequências-candidatas.)

**Exr 10.34** [Cobertura por subsequências. Minimax] Seja  $\langle a_1, \dots, a_n \rangle$  uma sequência de números. Uma coleção  $\mathcal{C}$  de sequências *cobre*  $\langle a_1, \dots, a_n \rangle$  se cada termo de  $\langle a_1, \dots, a_n \rangle$  está em alguma sequência da coleção  $\mathcal{C}$ . Uma *cobertura* de  $\langle a_1, \dots, a_n \rangle$  é uma coleção  $\mathcal{D}$  de subsequências estritamente decrescentes (SSEDs) de  $\langle a_1, \dots, a_n \rangle$  que cobre  $\langle a_1, \dots, a_n \rangle$ .

Suponha que  $\langle a_1, \dots, a_n \rangle$  tem uma cobertura com  $k$  sequências; mostre que toda subsequência crescente (SSC) de  $\langle a_1, \dots, a_n \rangle$  tem comprimento no máximo  $k$ . Suponha que  $\langle a_1, \dots, a_n \rangle$  tem uma SSC de comprimento  $k$ ; mostre que toda cobertura de  $\langle a_1, \dots, a_n \rangle$  contém pelo menos  $k$  sequências.

Dê um algoritmo que, ao receber uma sequência  $\langle a_1, \dots, a_n \rangle$  devolva uma tem uma SSC máxima e uma cobertura mínima.

**Exr 10.35** Encontre uma *supersequência* comum de comprimento mínimo das duas sequências abaixo:

abaabababbab  
babaabaab

Dê um algoritmo que encontre uma supersequência comum de comprimento mínimo de duas sequências  $\langle x_1, \dots, x_m \rangle$  e  $\langle y_1, \dots, y_n \rangle$  dadas. O consumo de tempo do seu algoritmo deve ser  $O(mn)$ .

## 10.5 Mais programação dinâmica

**Exr 10.36** [Segmento de soma máxima] Escreva um algoritmo de programação dinâmica para encontrar um segmento de soma máxima de um sequência dada de números inteiros (os elementos da sequência podem ser positivos e negativos). Quanto tempo o seu algoritmo consome?

**Exr 10.37** [Quebra de strings, Parberry 410, [Pro, PC 111105]] Uma certa linguagem de processamento de texto permite que o programador quebre uma cadeia de caracteres em duas. Como isso envolve a cópia da cadeia original, a quebra de uma cadeia de  $n$  caracteres em duas tem custo  $n$ . A ordem em que as quebras são feitas pode afetar o consumo total da operação. Por exemplo, suponha que queremos quebrar uma cadeia de 20 caracteres depois dos caracteres 3, 8 e 10 (estamos numerando os caracteres da esquerda para a direita e começando a numeração com 1). Se as quebras são feitas da esquerda para a direita, a primeira quebra custa 20, a segunda custa 17 e a última custa 12; o total é 49. Se as quebras são feitas da direita para a esquerda, a primeira custa 20, a segunda custa 10 e a terceira custa 8; o total é 38.

|                                                |           |
|------------------------------------------------|-----------|
| <u>t h i</u> s i s a s t r i n g o f c h a r s | custo: 20 |
| <u>s i s a s</u> t r i n g o f c h a r s       | custo: 17 |
| <u>t r</u> i n g o f c h a r s                 | custo: 12 |

Escreva um algoritmo de programação dinâmica que receba uma cadeia de caracteres e as posições das quebras desejadas e determine a ordem das quebras que minimize o custo. (Não se restrinja

às ordens esquerda-para-direita e direita-para-esquerda dadas no exemplo!) O consumo de tempo do seu algoritmo deve ser  $O(n^3)$ .

**Exr 10.38** [Redação melhorada da Quebra de strings, Parberry 410, [Pro, PC 111105]] Suponha dada uma haste metálica de comprimento  $L$ . A haste tem uma ponta inicial e uma final. A haste tem marcas a  $x_1, x_2, \dots, x_n$  unidades de sua ponta inicial. (Suponha que  $x_n < L$ .) Queremos cortar a haste nos pontos marcados de modo que, ao final do processo, a haste seja dividida em  $n + 1$  hastes menores. Para fazer os cortes terei que usar, repetidas vezes, uma máquina de cortar. Em cada iteração, a máquina recebe uma haste de comprimento  $c$  e corta a haste no ponto especificado. Por algum motivo, o custo dessa operação é proporcional a  $c$  (qualquer que seja o ponto de corte). A ordem em que os cortes são feitos pode afetar o consumo total da operação. Por exemplo, suponha que  $L = 20$ ,  $n = 3$ ,  $x_1 = 3$ ,  $x_2 = 8$  e  $x_3 = 10$ . Se os cortes são feitas na ordem  $x_1, x_2, x_3$ , o primeira quebra custa 20, o segundo custa 17 e o último custa 12; o total é 49. Se os cortes são feitos na ordem  $x_3, x_2, x_1$ , o primeiro custa 20, o segundo custa 10 e o terceiro custa 8; o total é 38. Escreva um algoritmo de programação dinâmica que determine a ordem em que os cortes devem ser feitos para minimizar o custo. (Não se restrinja às ordens esquerda-para-direita e direita-para-esquerda dadas no exemplo!) O consumo de tempo do seu algoritmo deve ser  $O(n^3)$ .

**Exr 10.39** [Expressão aritmética de valor máximo] Suponha dada uma sequência  $\langle x_1, \dots, x_{2n+1} \rangle$  em que  $x_1, x_3, \dots, x_{2n+1}$  são inteiros positivos e  $x_2, x_4, \dots, x_{2n}$  pertencem ao conjunto  $\{+, *\}$ . É claro que “+” representa soma e “\*” representa multiplicação. Queremos inserir parênteses na sequência  $\langle x_1, \dots, x_{2n+1} \rangle$  de tal forma que o valor da expressão aritmética resultante seja máximo. Por exemplo, se a sequência dada é  $2+4*1+5$  então uma solução é a expressão  $((2+4)*(1+5))$ , que vale 36. Escreva um algoritmo de programação dinâmica que devolva o valor máximo da expressão. Mostre que o seu algoritmo está correto. Escreva a recorrência que serve de base para o seu algoritmo. Calcule o consumo de tempo do seu algoritmo.

**Exr 10.40** [Tabela de multiplicação] Seja  $S$  o conjunto  $\{a, b, c\}$  e considere a seguinte tabela de “multiplicação” sobre  $S$ :

|     | $a$ | $b$ | $c$ |
|-----|-----|-----|-----|
| $a$ | $b$ | $b$ | $a$ |
| $b$ | $c$ | $b$ | $a$ |
| $c$ | $a$ | $c$ | $c$ |

Assim,  $ab = b$ ,  $ba = c$ , e assim por diante. (Note que a operação de multiplicação definida pela tabela não é comutativa nem associativa.) Escreva um algoritmo eficiente que receba uma sequência  $x_1x_2 \cdots x_n$  de elementos de  $S$  e decida se é possível ou não inserir parênteses na sequência de tal modo que o valor da expressão resultante seja  $a$ . Por exemplo, se a sequência dada for  $bbba$  então a resposta deve ser afirmativa uma vez que  $(b(bb))(ba) = a$  (e também  $(b(b(b(ba)))) = a$ ). Especifique claramente a relação de recorrência que serve de base para o seu algoritmo. Quanto tempo seu algoritmo consome (em função de  $n$ )?

Parte 2: Modifique o seu algoritmo de modo que ele devolva o número de diferentes maneiras de inserir parênteses em  $x_1x_2 \cdots x_n$  de modo que o valor da expressão resultante seja  $a$ .

**Exr 10.41** [Número de diferentes ordenações] Queremos colocar  $n$  objetos em ordem usando as relações “ $<$ ” e “ $=$ ”. Por exemplo, para  $n = 3$  há 13 diferentes ordenações:

$$\begin{array}{ccccc} a = b = c & a = b < c & a < b = c & a < b < c & a < c < b \\ a = c < b & b < a = c & b < a < c & b < c < a & b = c < a \\ c < a = b & c < a < b & c < b < a & & \end{array}$$

Escreva um algoritmo de programação dinâmica que calcule, em função de  $n$ , o número de diferentes ordenações dos  $n$  objetos. (Especifique claramente a relação de recorrência que serve de base para o seu algoritmo.) O seu algoritmo deve consumir tempo  $O(n^2)$  e espaço  $O(n)$ .

**Exr 10.42** [Printing neatly, CLRS 15-2, Parberry 413] Considere a sequência  $P_1, P_2, \dots, P_n$  de palavras que constitui um parágrafo de texto. A palavra  $P_i$  tem  $l_i$  caracteres. Queremos imprimir as palavras em linhas, na ordem dada, de modo que cada linha tenha no máximo  $M$  caracteres (sem quebrar palavras entre linhas). Se uma determinada linha contém as palavras  $P_i, P_{i+1}, \dots, P_j$  (com  $i \leq j$ ) e há exatamente um espaço entre cada par de palavras consecutivas, o número de espaços no fim da linha é

$$M - (l_i + l_{i+1} + \dots + l_j + (j - i)).$$

É claro que não devemos permitir que esse número seja negativo. Queremos minimizar, com relação a todas as linhas exceto a última, a soma dos cubos dos números de espaços no fim de cada linha. (Assim, se temos linhas  $1, 2, \dots, L$  e  $b_p$  espaços no fim da linha  $p$ , queremos minimizar  $b_1^3 + b_2^3 + \dots + b_{L-1}^3$ ).<sup>7</sup>

Dê um exemplo para mostrar que algoritmos inocentes não resolvem o problema. Dê um algoritmo de programação dinâmica que resolva o problema. Qual a *optimal substructure property* para esse problema? Faça uma análise do consumo de tempo do algoritmo.

**Exr 10.43** [BB 8.32: Descendo o rio] Temos  $n$  estações ao longo de um rio, numeradas de 1 a  $n$  na direção da correnteza. Você pode alugar um barco em qualquer estação  $i$ , descer o rio até uma estação  $k > i$ , devolver o barco e pagar uma taxa  $c(i, k)$  pelo passeio. É possível que  $c(i, k)$  seja maior que  $c(i, j) + c(j, k)$ , sendo  $j$  uma estação intermediária entre  $i$  e  $k$ . Nesse caso, é mais barato alugar um barco de  $i$  a  $j$  e depois outro de  $j$  até  $k$ .

Dê um algoritmo eficiente que calcule o custo mínimo de uma viagem de 1 a  $n$ . Exiba a recorrência que serve de base para o seu algoritmo. Em função de  $n$ , quanto tempo o seu algoritmo consome?

**Exr 10.44** [Programando para maximizar o lucro, CLRS 15-7] Suponha que você tem um conjunto de tarefas, numeradas de 1 a  $n$ , que devem ser processados em uma máquina. Cada tarefa  $j$  tem um tempo de processamento  $t_j$ , um prazo  $d_j$  e um lucro  $p_j$ . A máquina só pode processar um tarefa de cada vez e cada tarefa  $j$  deve ser processada ininterruptamente por  $t_j$  unidades de tempo. Para cada tarefa  $j$ , você tem lucro  $p_j$  se a tarefa for concluída no prazo  $d_j$  e lucro 0 em caso contrário. Escreva um algoritmo que determine a ordem de execução das tarefas que maximiza a soma dos lucros. Suponha que  $t_1, \dots, t_n$  são inteiros entre 1 e  $n$  inclusive. Estime o consumo de tempo do seu algoritmo.

<sup>7</sup> Como é uma solução ótima se  $l_i = 1$  para todo  $i$ ? E se  $l_i = 0$  para todo  $i$ ?

**Exr 10.45** [Carregamento ótimo de balsa [Pro, PC 111106]] Uma balsa é usada para transportar carros de uma margem do rio à outra. A balsa tem duas pistas: a pista  $B$  (de bombordo) e a pista  $E$  (de estibordo). Cada pista é capaz de acomodar uma fila de carros. Há uma fila de carros esperando para embarcar na balsa. O operador da balsa estima os comprimentos dos carros e encaminha os carros para uma ou outra pista, respeitando a fila, de modo a embarcar o maior número possível de carros. Escreva um programa que faça o papel do operador da balsa. Todos os comprimentos (balsa e cada um dos carros) são inteiros positivos.

**Exr 10.46** [Intervalos disjuntos, CLRS 16.1-1] Dois intervalos  $[a, b)$  e  $[c, d)$  da reta real são *disjuntos* se  $[a, b) \cap [c, d) = \emptyset$ . Uma coleção de intervalos é *disjunta* se os intervalos da coleção são disjuntos dois a dois. O problema da coleção disjunta máxima de intervalos<sup>8</sup> consiste no seguinte: Dada uma coleção de intervalos, digamos  $S$ , determinar uma subcoleção disjunta máxima de  $S$ . Qual a *optimal substructure property* para esse problema? Escreva e analise um algoritmo de programação dinâmica para o problema da coleção disjunta máxima de intervalos. Para simplificar, basta que o algoritmo determine o *tamanho* de uma coleção disjunta máxima.

**Exr 10.47** [CLRS 15-3a, Parberry 414] Resumo do problema (veja enunciado completo em Cormen *et al.* [CLRS01]): Queremos transformar uma cadeia de caracteres  $x[1..m]$  em uma cadeia de caracteres  $y[1..n]$  usando uma sequência de operações; as seis operações válidas são descritas abaixo. A sequência de operações “percorre” o vetor  $x$  (do índice 1 ao índice  $m$ ) e constrói um vetor auxiliar  $z[1..n]$ . Ao final da sequência de operações, deveremos ter  $i = m + 1$  e  $z[1..n] = y[1..n]$ . No começo de cada iteração tenho índices  $i$  e  $j$  para  $x$  e  $z$  respectivamente. Eis as operações válidas:

- *copy*:  $z[j] \leftarrow x[i], i \leftarrow i + 1, j \leftarrow j + 1;$
- *replace* (por algum caractere  $c$ ):  $z[j] \leftarrow c, i \leftarrow i + 1, j \leftarrow j + 1;$
- *delete*<sup>9</sup>:  $i \leftarrow i + 1;$
- *insert* (de um caractere  $c$ ):  $z[j] \leftarrow c, j \leftarrow j + 1;$
- *twiddle*:  $z[j] \leftarrow x[i + 1], z[j + 1] \leftarrow x[i], i \leftarrow i + 2, j \leftarrow j + 2;$
- *kill*:  $i \leftarrow m + 1.$

(Note que  $i$  e  $j$  jamais decrescem.) Suponha que as operações os custos indicados abaixo.<sup>10</sup>

| <i>copy</i> | <i>replace</i> | <i>delete</i> | <i>insert</i> | <i>twiddle</i> | <i>kill</i> |
|-------------|----------------|---------------|---------------|----------------|-------------|
| 2           | 3              | 2             | 2             | 3              | 1           |

Uma sequência de operações é ótima se a soma dos custos das operações da sequência é mínima. A *distância de edição* (= *edit distance*) de  $x[1..m]$  a  $y[1..n]$  é o custo de uma sequência ótima de operações que transforma  $x$  em  $y$ .

Dê um algoritmo de programação dinâmica que calcule a distância de edição de  $x$  a  $y$ . Qual a *optimal substructure property* para esse problema? Analise o consumo de tempo e de espaço do algoritmo.

<sup>8</sup> Cormen *et al.* [CLRS01] dizem *Activity-Selection Problem*.

<sup>9</sup> Isso é palavra da língua inglesa. Em português, diga *apagar, remover, retirar*.

<sup>10</sup> Quaisquer outros custos poderiam ser adotados. Mas é claro que o problema ficaria menos interessante se a soma dos custos de *delete* e *insert* fosse menor que o custo de *copy* ou o custo de *replace*.

**Exr 10.48** [CLRS 15-3b] (Veja antes o exercício 10.47). Um alinhamento de duas cadeias de caracteres  $x[1..m]$  e  $y[1..n]$  consiste na inserção de espaços (caracteres  $\_$ ) em posições arbitrárias de  $x$  e  $y$  de tal modo que as cadeias de caracteres resultantes, digamos  $x'$  e  $y'$ , tenham o mesmo comprimento mas não tenham espaços em posições correspondentes. O valor de um alinhamento é a soma de pontos definida assim:

- $+1$  se  $x'[j] = y'[j]$  e nenhum dos dois é um espaço,
- $-1$  se  $x'[j] \neq y'[j]$  e nenhum dos dois é um espaço,
- $-2$  se  $x'[j]$  ou  $y'[j]$  é um espaço.

Mostre como o problema de determinar um alinhamento de valor máximo pode ser formulado como um problema de distância de edição. Use um subconjunto apropriado do conjunto de operações  $\{copy, replace, delete, insert, twiddle, kill\}$ .

**Exr 10.49** Uma loja de aluguel de esquis tem  $m$  pares de esquis para alugar, sendo  $s_i$  a altura do  $i$ -ésimo par de esquis. Temos também  $n$  esquiadores querendo alugar pares de esquis, sendo  $h_i$  a altura do  $i$ -ésimo esquiador. Sabe-se que um esquiador deve usar um esqui cuja altura é tão próxima quanto possível de sua altura. Supondo que  $n = m$ , escreva um algoritmo eficiente de alocação de esquis aos esquiadores de forma a minimizar a soma dos valores absolutos das diferenças entre as alturas do esquiador e do seu esqui. Analise a eficiência do seu algoritmo e justifique sua corretude. Sugestão: Mostre que existe uma solução ótima sem cruzamentos de alturas, i.e., se esquiadores  $i$  e  $j$  têm alturas  $h_i < h_j$  seus respectivos esquis terão alturas  $s_i \leq s_j$ .

# Capítulo 11

## Estratégia gulosa

Veja CLRS cap.16.

Veja o que diz Ian Parberry [Par95] sobre algoritmos gulosos: “A greedy algorithm starts with a solution to a very small subproblem and augments it successively to a solution for the big problem. The augmentation is done in a ‘greedy’ fashion, that is, paying attention to short-term or local gain, without regard to whether it will lead to a good long-term or global solution. As in real life, greedy algorithms sometimes lead to the best solution, sometimes lead to pretty good solutions, and sometimes lead to lousy solutions. The trick is to determine when to be greedy.”

E ainda: “One thing you will notice about greedy algorithms is that they are usually easy to design, easy to implement, easy to analyse, and they are very fast, but they are almost always difficult to prove correct.”

**Exr 11.1** Escreva um algoritmo para decidir se  $\langle z_1, \dots, z_k \rangle$  é subsequência de  $\langle x_1, \dots, x_m \rangle$ . Prove rigorosamente que o seu algoritmo está correto. Calcule o consumo de tempo do seu algoritmo. Veja o exercício 10.20.

### 11.1 Instâncias especiais do problema da mochila

**Exr 11.2** O problema subset sum do exercício 10.14 pode ser resolvido por um algoritmo guloso? O problema tem a propriedade da escolha gulosa?

**Exr 11.3** [Problema do disquete, CLRS 16.2-3 simplificado] Escreva um algoritmo guloso para o seguinte problema: dados números inteiros não-negativos  $w_1, \dots, w_n$  e  $W$ , encontrar um subconjunto máximo  $K$  de  $\{1, \dots, n\}$  dentre os que satisfazem  $\sum_{k \in K} w_k \leq W$ . (Imagine que  $w_1, \dots, w_n$  são os tamanhos de arquivos digitais que você deseja armazenar em um disquete de capacidade  $W$ . Compare com os exercícios 10.14 e 10.16.)<sup>1</sup>

---

<sup>1</sup> CLRS exige que os  $w_i$  sejam distintos dois a dois. Esse caso particular do problema pode ser formulado assim: dado um conjunto  $T$  de inteiros não-negativos e um inteiro não-negativo  $W$ , encontrar um subconjunto máximo  $S$  de  $T$  tal que  $\sum_{s \in S} s \leq W$ . Mas essas restrições não tornam o problema mais fácil.

**Exr 11.4** [Mochila fracionária, CLRS 16.2-1] O problema da mochila fracionária consiste no seguinte: dados números inteiros<sup>2</sup> não-negativos  $v_1, \dots, v_n, w_1, \dots, w_n$  e  $W$  (imagine que  $w_i$  é o peso e  $v_i$  é o valor do objeto  $i$ ), encontrar números racionais  $x_1, \dots, x_n$  no intervalo fechado  $[0, 1]$  que maximizem a soma  $x_1v_1 + \dots + x_nv_n$  enquanto respeitam a restrição  $x_1w_1 + \dots + x_nw_n \leq W$ . Escreva um algoritmo guloso para resolver o problema. (Para simplificar, basta que seu algoritmo devolva o valor de uma solução, ou seja, o valor da soma  $x_1v_1 + \dots + x_nv_n$ .) Prove que o seu algoritmo está correto. Dê um exemplo para mostrar que o algoritmo não resolve o problema booleano da mochila (exercício 10.17).

**Exr 11.5** O problema da mochila booleana (exercício 10.17) pode ser resolvido por um algoritmo guloso?

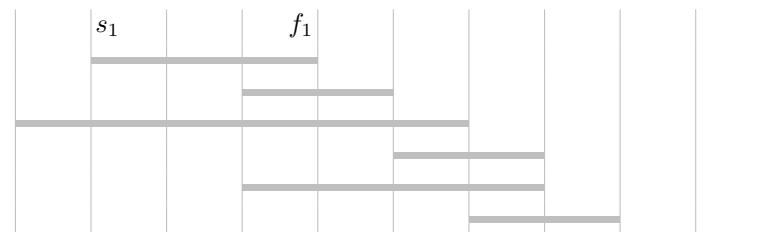
**Exr 11.6** Seja  $P$  um inteiro positivo e seja  $\langle p_1, \dots, p_n \rangle$  uma sequência de números inteiros não-negativos. Considere os seguintes problemas. Problema 1: Encontrar um subconjunto  $X$  de  $\{1, \dots, n\}$  tal que  $\sum_{i \in X} p_i \leq P$  e  $\sum_{i \in X} p_i$  é máximo. Problema 2: Encontrar um subconjunto  $X$  de  $\{1, \dots, n\}$  que maximize  $\sum_{i \in X} p_i$  sob a restrição  $\sum_{i \in X} p_i \leq P$ . Qual o consumo de tempo (em notação O) do melhor algoritmo conhecido para cada um dos problemas?

## 11.2 Intervalos disjuntos

Um *intervalo* é um par ordenado  $(s, f)$  de números inteiros positivos tal que  $s < f$ . O primeiro número do par é o *início* do intervalo e o segundo é o *término* do intervalo. Um intervalo  $(s, f)$  é *anterior* a um intervalo  $(s', f')$  se  $f \leq s'$ . Um intervalo  $(s, f)$  é *posterior* a um intervalo  $(s', f')$  se  $f' \leq s$ . Dois intervalos são *disjuntos* se um é anterior ou posterior ao outro. Dois intervalos são *incompatíveis* se não forem disjuntos. Uma coleção de intervalos é *disjunta* se os intervalos da coleção são disjuntos dois a dois.

Problema da Coleção Disjunta Máxima de Intervalos (ou *Activity-selection Problem*): Dada uma coleção  $S$  de intervalos, determinar uma subcoleção disjunta máxima de  $S$ .

Uma instância do problema:



Uma solução (subconjunto de  $\{1, \dots, n\}$ ) da instância:



<sup>2</sup> Esta condição não é essencial: os números poderiam ser racionais.

**Exr 11.7** [Maximal versus máximo] Seja  $\mathcal{S}$  uma coleção de intervalos na reta real. É verdade que todas as subcoleções disjuntas maximais de  $\mathcal{S}$  são máximas?

**Exr 11.8** Escreva um algoritmo que receba uma coleção  $\mathcal{S}$  de intervalos e devolva uma subcoleção disjunta máxima de  $\mathcal{S}$ .

**Exr 11.9** Suponha dada uma coleção de intervalos. Suponha que os intervalos são dadas em ordem crescente de início:  $f_1 \leq f_2 \leq \dots \leq f_n$ . Escreva um algoritmo guloso que resolva o problema tirando proveito da ordem em que os intervalos são dados.

**Exr 11.10** [CLRS 16.1-2] Mostre que a seguinte idéia também produz um algoritmo guloso correto para o problema da coleção disjunta máxima de intervalos: dentre os intervalos disjuntos dos já escolhidos, escolha um que tenha instante de início máximo.

Em outras palavras, repita o exercício 11.9 supondo que os intervalos são dadas em ordem decrescente de início:  $s_1 \geq s_2 \geq \dots \geq s_n$ . Escreva um algoritmo guloso que resolva o problema tirando proveito da ordem em que os intervalos são dadas (Veja também o exercício 10.46.)

**Exr 11.11** [CLRS 16.1-4] Nem todo algoritmo guloso resolve o problema da coleção disjunta máxima de intervalos. Mostre que nenhuma das três idéias a seguir resolve o problema. Idéia 1: Escolha um intervalo de menor tamanho dentre os que são disjuntos dos intervalos já escolhidos. Idéia 2: Escolha um intervalo que seja disjunto dos já escolhidos e intercepta o menor número possível de intervalos ainda não escolhidos. Idéia 3: Escolha o intervalo disjunto dos já selecionados que tenha o menor instante de início.

**Exr 11.12** Seja  $\mathcal{C}$  uma coleção de intervalos na reta real. Suponha que cada intervalo tem a forma  $[s, f]$ . Queremos encontrar uma coleção disjunta máxima de  $\mathcal{C}$ . Qual das alternativas está correta: (1) o intervalo com menor  $f$  pertence a alguma solução do problema; (2) o intervalo com maior  $f$  pertence a alguma solução; (3) o intervalo com menor  $s$  pertence a alguma solução; (4) o intervalo com maior  $s$  pertence a alguma solução.

**Exr 11.13** Para  $i = 1, \dots, n$ , sejam  $[s_i, f_i]$  intervalos na reta real. Suponha que  $f_1 \leq \dots \leq f_n$ . O algoritmo abaixo promete devolver a cardinalidade de uma coleção disjunta máxima de intervalos. Onde está o erro?

```

INTERV-DISJ ( $s, f, n$ )
1   num  $\leftarrow i \leftarrow 1$ 
2   enquanto  $i \leq n$  faça
3        $m \leftarrow i + 1$ 
4       enquanto  $m \leq n$  e  $s_m < f_i$  faça
5            $m \leftarrow m + 1$ 
6        $i \leftarrow m$ 
7       num  $\leftarrow num + 1$ 
8   devolva num

```

**Exr 11.14** Para  $i = 1, \dots, n$ , sejam  $[s_i, f_i]$  intervalos na reta real. Suponha que  $f_1 \leq \dots \leq f_n$ . O algoritmo abaixo promete devolver a cardinalidade de uma coleção disjunta máxima de intervalos. Critique o código.

INTERVALOS-DISJUNTOS ( $s, f, n$ )

- 1  $x \leftarrow i \leftarrow 1$
- 2 enquanto  $i \leq n$  faça
  - 3  $x \leftarrow x + 1$
  - 4  $m \leftarrow i + 1$
  - 5 enquanto  $m \leq n$  e  $s_m < f_i$  faça
    - 6  $m \leftarrow m + 1$
  - 7  $i \leftarrow m$
- 8 devolva  $x$

**Exr 11.15** [Interval clique cover] Dois intervalos, digamos  $[s_1, f_1)$  e  $[s_2, f_2)$ , são *incompatíveis* se  $[s_1, f_1) \cap [s_2, f_2) \neq \emptyset$ . Uma coleção de intervalos é uma *clique* se os intervalos da coleção são incompatíveis dois a dois (e os intervalos têm um ponto em comum). Uma *cobertura por cliques* de uma coleção  $\mathcal{S}$  de intervalos é uma coleção  $\mathcal{C}_1, \dots, \mathcal{C}_k$  de cliques tal que  $\mathcal{C}_i \subseteq \mathcal{S}$  para todo  $i$  e todo elemento de  $\mathcal{S}$  pertence a algum dos cliques da coleção. Problema: Encontrar uma cobertura mínima de uma coleção de intervalos dada.

Escreva um algoritmo guloso para o problema. Qual a *optimal substructure property* para esse problema? Qual a *greedy-choice property* para esse problema?

[Minimax.] Mostre que se  $\mathcal{C}_1, \dots, \mathcal{C}_k$  é uma cobertura de  $\mathcal{S}$  por cliques e  $\mathcal{A}$  é uma subcoleção disjunta máxima de  $\mathcal{S}$  então  $|\mathcal{A}| \leq k$ . Prove que existe uma cobertura  $\mathcal{C}_1, \dots, \mathcal{C}_k$  por cliques e uma coleção disjunta  $\mathcal{A}$  de  $\mathcal{S}$  tal que  $|\mathcal{A}| = k$ .

**Exr 11.16** [Max interval-clique] Dois intervalos, digamos  $[s_1, f_1)$  e  $[s_2, f_2)$ , são *incompatíveis* se  $[s_1, f_1) \cap [s_2, f_2) \neq \emptyset$ . Uma subcoleção  $\mathcal{C}$  de uma coleção  $\mathcal{S}$  de intervalos é uma *clique* se os intervalos em  $\mathcal{C}$  são incompatíveis dois a dois. Problema: Encontrar uma clique máxima em uma coleção de intervalos dada. Escreva um algoritmo guloso para o problema. Qual a *optimal substructure property* para esse problema? Qual a *greedy-choice property* para esse problema?

**Exr 11.17** [Interval-graph coloring, CLRS 16.1-3] Queremos distribuir um conjunto de atividades no menor número possível de salas. Cada atividade  $a_i$  ocupa um certo intervalo de tempo  $[s_i, f_i)$ ; duas atividades podem ser programadas para a mesma sala somente se os correspondentes intervalos são disjuntos. Descreva um algoritmo guloso que resolve o problema. (Represente cada sala por uma cor; use o menor número possível de cores para pintar todos os intervalos.) Prove que o número de salas dado pelo algoritmo é, de fato, mínimo.

### 11.3 Códigos de Huffman

Veja CLRS sec.16.3

**Exr 11.18** Suponha que uma árvore de Huffman tem uma raiz  $r$ , nós internos  $e$  e  $f$  e folhas  $a, b$ ,

c, d. A estrutura da árvore está representada na tabela abaixo:

| nó       | <i>esq</i> | <i>dir</i> | caractere |
|----------|------------|------------|-----------|
| <i>r</i> | <i>e</i>   | <i>d</i>   | —         |
| <i>e</i> | <i>a</i>   | <i>f</i>   | —         |
| <i>a</i> | —          | —          | A         |
| <i>f</i> | <i>b</i>   | <i>c</i>   | —         |
| <i>d</i> | —          | —          | D         |
| <i>b</i> | —          | —          | B         |
| <i>c</i> | —          | —          | C         |

Suponha que *esq* corresponde a 0 e *dir* corresponde a 1. Decodifique a mensagem 0 0 0 1 0 1 0 1 0 0 1 1.

**Exr 11.19** Considere o conjunto de caracteres {a, b, c, d, e, f, g, h}. A tabela abaixo dá o número de ocorrências de cada caractere num certo arquivo.

| a  | b  | c  | d   | e  | f  | g  | h  |
|----|----|----|-----|----|----|----|----|
| 10 | 40 | 56 | 120 | 55 | 50 | 56 | 10 |

Parte 1: Calcule o código de Huffman para o arquivo. Qual a propriedade mais importante de um código de Huffman (ou seja, por que o código é tão útil)? Parte 2: Para a codificação calculada na parte 1, qual o código da cadeia de caracteres abacade?

**Exr 11.20** [CLRS 16.3-2] Calcule o código de Huffman para a seguinte tabela de caracteres e frequências:

| c           | a | b | c | d | e | f | g  | h  |
|-------------|---|---|---|---|---|---|----|----|
| <i>f[c]</i> | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 |

Generalize esse exemplo e descreva a árvore de Huffman de um conjunto de *n* caracteres cujas frequências são os *n* primeiros números de Fibonacci.

**Exr 11.21** Suponha dado um arquivo em que só ocorrem os caracteres A, B, C, D, E, F. Cada caractere ocorre exatamente *x* vezes. Parte 1: Quantos bits são necessários para representar cada caractere supondo que todos os caracteres usam o mesmo número de bits? Qual será o tamanho total do arquivo (medido em número de bits) nesse caso? Parte 2: Qual o tamanho do arquivo se usarmos um código de Huffman?

**Exr 11.22** [CLRS 16.3-3] Considere uma árvore binária que representa um código livre de prefixos. Como se sabe, o custo da árvore é  $\sum_{c \in C} f[c] d(c)$ , onde *C* é o conjunto de folhas, *f[c]* é a frequência do caractere representado pela folha e *d(c)* é a profundidade da folha. Mostre que o custo da árvore é igual à soma  $\sum_{i \in I} (f[j] + f[k])$  onde *I* é o conjunto de nós internos (ou seja, não-folhas) da árvore e *j* e *k* são os dois filhos de *i*.

**Exr 11.23** Uma árvore binária é *cheia* se cada um de seus nós tem 0 ou 2 filhos. Um nó é *interno* se tiver 2 filhos. Prove (por indução) que toda árvore binária cheia com *n* folhas tem exatamente *n* – 1 nós internos.

## 11.4 Outros problemas

**Exr 11.24** [Pares de livros] Suponha dado um conjunto de livros numerados de 1 a  $n$ . Suponha que cada livro  $i$  tem um peso  $p_i$  tal que  $0 < p_i < 1$ . Problema: acondicionar os livros no menor número possível de envelopes de modo que cada envelope tenha 1 ou 2 livros e o peso do conteúdo de cada envelope não passe de 1. Escreva um algoritmo guloso que determine o número mínimo de envelopes. Aplique seu algoritmo a um exemplo interessante. Prove que o seu algoritmo está correto (ou seja, prove a *greedy-choice property* e a *optimal substructure* apropriadas). O consumo de tempo do algoritmo deve ser  $O(n \log n)$ .

**Exr 11.25** Seja  $\langle t_1, t_2, \dots, t_n \rangle$  uma sequência crescente de números inteiros que pertencem ao conjunto  $\{1, \dots, 99\}$ . Digamos que um subconjunto  $K$  de  $\{1, \dots, n\}$  é válido se  $|K| \leq 2$  e  $\sum_{k \in K} t_k \leq 100$ . Digamos que uma *embalagem* é uma partição de  $\{1, \dots, n\}$  em conjuntos válidos. O *tamanho* de uma embalagem  $\mathcal{E}$  é o número  $|\mathcal{E}|$ . Nossa problema: encontrar uma embalagem de tamanho mínimo. Parte 1: Escreva (em pseudocódigo CLR) um algoritmo guloso que devolva o tamanho de uma embalagem mínima de  $\langle t_1, t_2, \dots, t_n \rangle$ . Estime o consumo de tempo do algoritmo. Parte 2: Enuncie e prove a propriedade da escolha gulosa (*greedy choice property*) para nosso problema. Enuncie e prove a propriedade da subestrutura ótima (*optimal substructure property*) para nosso problema.

**Exr 11.26** [Bin-packing] São dados objetos  $1, \dots, n$  e um número ilimitado de “latas”. Cada objeto  $i$  tem “volume”  $w_i$  e cada lata tem “capacidade” 1: a soma dos volumes dos objetos colocados em uma lata não pode passar de 1. Problema: Distribuir os objetos pelo menor número possível de latas. Programe e teste as seguinte heurísticas. Heurística 1: examine os objetos na ordem dada; tente colocar cada objeto em uma lata já parcialmente ocupada que tiver mais “espaço” livre sobrando; se isso for impossível, pegue uma nova lata. Heurística 2: rearranje os objetos em ordem decrescente de volume; em seguida, aplique a heurística 1.

Essas heurísticas resolvem o problema? Compare com o exercício 11.24. (Para testar seu programa, sugiro escrever uma rotina que receba  $n \leq 100000$  e gere  $w_1, \dots, w_n$  aleatoriamente, todos no intervalo  $(0, u)$ , sendo  $u \leq 1$ .)

**Exr 11.27** [Escalonamento de tarefas, parte de CLRS 16-4] Seja  $1, \dots, n$  um conjunto de tarefas. Cada tarefa consome um dia de trabalho; durante um dia de trabalho somente uma das tarefas pode ser executada. Os dias de trabalho são numerados de 1 a  $n$ . A cada tarefa  $t$  está associado um prazo  $p_t$ : a tarefa deveria ser executada em algum dia do intervalo  $1 \dots p_t$ . A cada tarefa  $t$  está associada uma multa não-negativa  $m_t$ . Se uma dada tarefa  $t$  é executada depois do prazo  $p_t$ , sou obrigado a pagar a multa  $m_t$  (mas a multa não depende do número de dias de atraso). Problema: Programar as tarefas (ou seja, estabelecer uma bijeção entre as tarefas e os dias de trabalho) de modo a minimizar a multa total. Escreva um algoritmo guloso para resolver o problema. Prove que seu algoritmo está correto (ou seja, prove a *greedy-choice property* e a *optimal substructure* apropriadas). Analise o consumo de tempo.

# Capítulo 12

## Análise amortizada

Veja CLRS cap.17.

Neste capítulo vamos analisar não um determinado algoritmo mas toda uma família de algoritmos semelhantes. É irrelevante o que cada algoritmo faz, mas todos eles têm em comum a manipulação de uma determinada estrutura de dados (um contador binário, ou uma fila de prioridades, ou uma coleção de conjuntos disjuntos, por exemplo). Nossa estudo se ocupa, exatamente, da parte do consumo de tempo do algoritmo devida à manipulação dessa estrutura.

Como nos outros capítulos, vamos analisar o consumo de tempo no *pior caso*. Neste capítulo, o método apropriado para analisar o consumo de tempo no pior caso é o da *análise amortizada*.

**Exr 12.1** [AU 3.8] Qual o consumo de tempo do algoritmo?

```
POWERSOFTWO ( $n$ )
1    $i \leftarrow 0$ 
2   enquanto  $2\lfloor n/2 \rfloor = n$  faça
3        $n \leftarrow \lfloor n/2 \rfloor$ 
4        $i \leftarrow i + 1$ 
5   devolva  $i$ 
```

**Exr 12.2** [AU 3.7.3\*] Qual o consumo de tempo do algoritmo?

```
SUMPOWERSOFTWO ( $n$ )
1    $s \leftarrow 0$ 
2   para  $i \leftarrow 1$  até  $n$  faça
3        $j \leftarrow i$ 
4       enquanto  $2\lfloor j/2 \rfloor = j$  faça
5            $j \leftarrow \lfloor j/2 \rfloor$ 
6            $s \leftarrow s + 1$ 
7   devolva  $s$ 
```

**Exr 12.3** O algoritmo abaixo recebe vetores  $s[1..n]$  e  $f[1..n]$  e devolve um número inteiro entre 0 e  $n$ . Mostre que o consumo de tempo do algoritmo é  $O(n)$  (apesar dos dois loops encaixados).

```

ALGO ( $s, f, n$ )
1    $t \leftarrow 0$ 
2    $i \leftarrow 1$ 
3   enquanto  $i \leq n$  faça
4        $t \leftarrow t + 1$ 
5        $m \leftarrow i + 1$ 
6       enquanto  $m \leq n$  e  $s[m] < f[i]$  faça
7            $m \leftarrow m + 1$ 
8        $i \leftarrow m$ 
9   devolva  $t$ 

```

**Exr 12.4** O algoritmo BUILDMAXHEAP consiste em  $\lfloor n/2 \rfloor$  invocações de ACERTA-DESCENDO. Mostre que o consumo *amortizado* de uma invocação é  $O(1)$  (ainda que algumas invocações possam consumir  $O(\log n)$  unidades de tempo).

**Exr 12.5** [CLRS 17.1-3, 17.2-2, 17.3-2] Uma sequência de  $n$  operações é executada sobre uma certa estrutura de dados. Suponha que a  $i$ -ésima operação custa

- $i$  se  $i$  é uma potência inteira de 2 e
- 1 em caso contrário.

Mostre que o custo amortizado de cada operação é  $O(1)$ . Use o método da “análise agregada”; depois repita os cálculos usando o método contábil (= *accounting method*); finalmente, repita tudo usando o método da função potencial.

**Exr 12.6** [CLR 18.1-3] Uma sequência de  $n$  operações é executada sobre uma estrutura de dados. Para  $i = 1, \dots, n$ , a  $i$ -ésima operação custa  $i$  se  $i$  é potência inteira de 3 e custa 2 em caso contrário. Determine o custo amortizado de cada operação.

**Exr 12.7** [CLRS 17.3-6] Mostre como implementar uma fila (= *queue*) por meio de duas pilhas (= *stacks*) de modo que o custo amortizado de cada operação ENTRA-NA-FILA e cada operação SAI-DA-FILA seja  $O(1)$ .

**Exr 12.8** O seguinte fragmento de pseudocódigo opera sobre um vetor  $A[1 \dots n]$ :

```

1    $j \leftarrow 1$ 
2   para  $i \leftarrow 1$  até  $n$  faça
3       enquanto  $j \leq n$  e  $A[j] \leq i$  faça
4            $j \leftarrow j + 1$ 
5       se  $j > 1$  então  $j \leftarrow j - 1$ 

```

Mostre que o consumo de tempo deste fragmento é  $O(n)$ . (Use o método agregado de análise amortizada.)

**Exr 12.9** O seguinte algoritmo (veja exercício 18.5) supõe que cada elemento de  $A[1 \dots n]$  está em  $\{0, \dots, k\}$ . Qual o consumo de tempo do algoritmo?

C-SORT ( $A, n, k$ )

```

1 para  $i \leftarrow 0$  até  $k$ 
2   faça  $C[i] \leftarrow 0$ 
3 para  $j \leftarrow 1$  até  $n$ 
4   faça  $C[A[j]] \leftarrow C[A[j]] + 1$ 
5  $j \leftarrow 1$ 
6 para  $i \leftarrow 0$  até  $k$  faça
7   para  $c \leftarrow 1$  até  $C[i]$ 
8     faça  $A[j] \leftarrow i$ 
9      $j \leftarrow j + 1$ 

```

**Exr 12.10** Descreva um algoritmo que receba um inteiro positivo  $n$  e calcule  $n^n$  fazendo não mais que  $2\lg n$  multiplicações de números inteiros. (Suponha que a multiplicação de dois números requer apenas uma operação, quaisquer que sejam os tamanhos desses números.)

**Exr 12.11** [CLRS 17-2, Busca binária dinâmica] A busca binária em um vetor ordenado consome tempo logarítmico, mas o tempo necessário para inserir um novo elemento é linear no tamanho do vetor. Isso pode ser melhorado se mantivermos diversos vetores ordenados (em lugar de um só). Suponha que queremos implementar as operações BUSCA e INSERÇÃO em um conjunto de  $n$  elementos. Seja  $\langle n_{k-1}, n_{k-2}, \dots, n_0 \rangle$  a representação binária de  $n$ , onde  $k = \lceil \lg(n+1) \rceil$ . Temos vetores crescentes  $A_0, A_1, \dots, A_{k-1}$ , sendo que o comprimento de  $A_i$  é  $2^i$ . Um vetor típico  $A_i$  é relevante se  $n_i = 1$  e irrelevante se  $n_i = 0$ . O número total de elementos nos  $k$  vetores é, portanto,

$$\sum_{i=0}^{k-1} n_i 2^i = n.$$

Cada vetor é crescente, mas não há qualquer relação entre os valores dos elementos em dois vetores diferentes. A. Dê um algoritmo para a operação BUSCA. Dê uma cota superior para o consumo de tempo do algoritmo. B. Dê um algoritmo para a operação INSERÇÃO. Dê uma cota superior para o consumo de tempo do algoritmo. Calcule o consumo de tempo *amortizado*. C. Discuta uma implementação da operação REMOÇÃO.

**Exr 12.12** Suponha que dispomos de dois tipos de operação —  $V$  e  $A$  — sobre uma certa estrutura. Para facilitar a discussão, diremos que a operação do tipo  $V$  é *vermelha* e a operação do tipo  $A$  é *azul*. Considere uma sequência

$$O_1, O_2, \dots, O_{n-1}, O_n$$

de operações, cada  $O_i$  escolhida no conjunto  $\{V, A\}$ . Suponha que  $O_1 = V$  e  $O_2 = A$ . Suponha que cada operação azul consome tempo constante (ou seja, se  $O_i = A$  então o consumo de tempo de  $O_i$  não depende de  $i$ ). Suponha que cada operação vermelha (a partir da segunda) consumo o dobro do tempo da operação vermelha anterior. Para cada um dos cenários a seguir, calcule o consumo de tempo total da sequência das  $n$  operações. Dê suas respostas em notação assintótica, mas procure dar a resposta mais justa possível.

1. A sequência tem  $\Theta(1)$  operações azuis entre cada duas operações vermelhas consecutivas.
2. A sequência tem  $\Theta(\sqrt{n})$  operações azuis entre cada duas operações vermelhas consecutivas.
3. Para  $1 \leq i < j < k \leq n$ , se  $O_i, O_j, O_k$  são operações vermelhas consecutivas, então o número de operações azuis entre  $O_j$  e  $O_k$  é pelo menos o dobro do número de operações azuis entre  $O_i$  e  $O_j$ .

## 12.1 Função potencial

**Exr 12.13 [Pilha]** Considere uma pilha sujeita às operações usuais de “empilhamento” e “desempilhamento”; essas operações serão denotadas por E e D respectivamente. Vamos denotar por B a operação “backup”, que consiste simplesmente em fazer uma cópia do conteúdo da pilha. Imagine uma sequência de operações, cada uma pertencente ao conjunto {E, D, B}, tal que (1) o número de elementos da pilha jamais ultrapassa  $k$  e (2) uma operação B ocorre exatamente a cada  $k$  operações do conjunto {E, D}. Use o método potencial para mostrar que o custo amortizado de qualquer das três operações na sequência é  $O(1)$ .

**Exr 12.14 [Tabela dinâmica]** Suponha que tenho uma sequência de operações de algum tipo sobre uma tabela dinâmica (tabela que aumenta de tamanho conforme a necessidade). O consumo de tempo real da  $i$ -ésima é  $c_i$ . Tenho uma função  $\Phi \geq 0$  que atribui um número  $\Phi_i$  à tabela imediatamente depois da  $i$ -ésima operação. Esse número é interpretado como o potencial da tabela. Qual o consumo de tempo amortizado de uma operação? Mostre que a soma dos consumos de tempo amortizados de  $n$  operações é pelo menos tão grande quanto o consumo de tempo real nas  $n$  operações.

**Exr 12.15** Mostre que a função  $\Phi(T) = t[T] - n[T]$  não é uma boa função potencial para a análise da tabela dinâmica  $T$  sob a operação TABLEINSERT. Mostre que  $\Phi(T) = t[T]$  também não é um bom potencial para a análise da tabela dinâmica.

**Exr 12.16 [Tabela dinâmica]** Considere uma tabela dinâmica  $T$  submetida a uma sequência operações TABLEINSERT. O tamanho da tabela é  $t[T]$  e o número de posições ocupadas da tabela é  $n[T]$ . Se  $n[T] < t[T]$ , o custo de uma operação é 1. Se  $n[T] = t[T]$ , o custo de uma operação é  $n[T] + 1$ . Qual o custo amortizado de uma operação em relação à função potencial  $\Phi(T) = t[T] - n[T]$ ? Qual o custo amortizado de uma operação em relação à função potencial  $\Phi(T) = t[T]$ ?

**Exr 12.17 [CLRS 17.3-3]** Considere a estrutura de dados min-heap munida das operações INSERT e EXTRACT-MIN. Cada operação consome tempo  $O(\lg n)$ , onde  $n$  é o número de elementos na estrutura. Dê uma função potencial  $\Phi$  tal que o custo amortizado de INSERT seja  $O(\lg n)$  e o custo amortizado de EXTRACT-MIN seja  $O(1)$ . Prove que sua função potencial de fato tem essas propriedades.

**Exr 12.18 [CLRS 17.3-3]** Considere uma fila de prioridades sujeita às operações INSERE e EXTRAI-MÍNIMO, definidas da maneira usual. Suponha ainda que qualquer das duas operações consome não mais que  $\lg(n + 1)$  unidades de tempo quando aplicada a uma fila com  $n$  elementos.

Descreva o contexto da análise amortizada. Especifique uma função potencial tal que os correspondentes consumos amortizado de tempo sejam  $O(\lg n)$  para a operação INSERE e  $O(1)$  para a operação EXTRAI-MÍNIMO. (Sugestão: Considere algo da forma logaritmo-do-fatorial.<sup>1</sup>) Justifique seus cálculos.

---

<sup>1</sup>  $\lg(n!)$  fica entre  $\frac{1}{2}n \lg n$  e  $n \lg n$ .

# Capítulo 13

## Conjuntos disjuntos dinâmicos

Neste capítulo vamos analisar, tipicamente, não um determinado algoritmo mas toda uma família de algoritmos semelhantes. Não importa o que cada algoritmo da família faz, mas todos eles têm em comum a manipulação de uma estrutura de dados conhecida como floresta de conjuntos disjuntos. Nosso estudo se ocupa, exatamente, da parte do consumo de tempo do algoritmo devida à manipulação dessa estrutura.

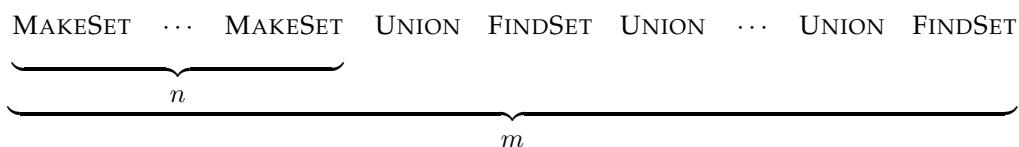
Este capítulo analisa a estrutura de dados “floresta de conjuntos disjuntos”. Considere as seguintes versões das operações FINDSET, UNION e MAKESET:

FINDSET ( $x$ )  
1 se  $pai[x] = x$   
2 então devolva  $x$   
3 senão devolva FINDSET ( $pai[x]$ )

UNION ( $x, y$ )  
1  $x' \leftarrow \text{FINDSET}(x)$   
2  $y' \leftarrow \text{FINDSET}(y)$   
3  $pai[x'] \leftarrow y'$

MAKESET ( $x$ )  
1  $pai[x] \leftarrow x$

Considere agora uma sequência de arbitrária de  $m$  operações MAKESET, UNION e FINDSET,  $n$  das quais são MAKESET. Sem perda de generalidade, podemos concentrar os  $n$  MAKESETS no início da sequência. Qual o consumo de tempo da sequência de operações?



Esta sequência produz uma coleção de árvores. Cada nó  $x$  da estrutura tem um pai  $pai[x]$ . Cada

árvore da floresta tem uma raiz  $r$  caracterizada pela condição  $pai[r] = r$ .

**Exr 13.1** [CLRS 21.2-2] Faça uma figura da floresta produzida pela seguinte sequência de operações:

- 1 para  $i \leftarrow 1$  até 16
- 2     faça  $\text{MAKESET}(x_i)$
- 3 para  $i \leftarrow 1$  até 15 em passos de 2
- 4     faça  $\text{UNION}(x_i, x_{i+1})$
- 5 para  $i \leftarrow 1$  até 13 em passos de 4
- 6     faça  $\text{UNION}(x_i, x_{i+2})$
- 7  $\text{UNION}(x_1, x_5)$
- 8  $\text{UNION}(x_{11}, x_{13})$
- 9  $\text{UNION}(x_1, x_{10})$
- 10  $\text{FINDSET}(x_2)$
- 11  $\text{FINDSET}(x_9)$

**Exr 13.2** Faça uma figura da floresta produzida pela seguinte sequência de operações:

- 0 para  $i \leftarrow 1$  até 9
- 1     faça  $\text{MAKESET}(x_i)$
- 2  $\text{UNION}(x_1, x_2)$
- 3  $\text{UNION}(x_3, x_4)$
- 4  $\text{UNION}(x_4, x_5)$
- 5  $\text{UNION}(x_2, x_4)$
- 6  $\text{UNION}(x_6, x_7)$
- 7  $\text{UNION}(x_8, x_9)$
- 8  $\text{UNION}(x_8, x_6)$
- 9  $\text{UNION}(x_5, x_7)$

### 13.1 Union by rank

O truque *union by rank* não altera a operação  $\text{FINDSET}$  mas modifica as duas outras operações da seguinte maneira:

$\text{UNIONUR}(x, y) \triangleright$  supõe que  $x$  e  $y$  estão em árvores diferentes

- 1  $x' \leftarrow \text{FINDSET}(x)$
- 2  $y' \leftarrow \text{FINDSET}(y)$
- 3 se  $\text{rank}[x'] = \text{rank}[y']$
- 4     então  $\text{pai}[y'] \leftarrow x'$
- 5          $\text{rank}[x'] \leftarrow \text{rank}[x'] + 1$
- 6 senão se  $\text{rank}[x'] > \text{rank}[y']$
- 7         então  $\text{pai}[y'] \leftarrow x'$
- 8         senão  $\text{pai}[x'] \leftarrow y'$

**MAKESETUR** ( $x$ )

- 1  $pai[x] \leftarrow x$
- 2  $rank[x] \leftarrow 0$

**Exr 13.3** O que há de errado com o seguinte código para UNIONUR?

**UNIONUR** ( $x, y$ )

- 1  $x' \leftarrow \text{FINDSET}(x)$
- 2  $y' \leftarrow \text{FINDSET}(y)$
- 3 se  $rank[x'] > rank[y']$
- 4     então  $pai[y'] \leftarrow x'$
- 5     senão  $pai[x'] \leftarrow y'$
- 6         se  $rank[x'] = rank[y']$
- 7             então  $rank[x'] \leftarrow rank[x'] + 1$

**Exr 13.4** [CLRS 21.3-1] Considere o repertório de operações MAKESETUR, UNIONUR e FINDSET. Faça uma figura da floresta produzida pela seguinte sequência de operações:

- 1 para  $i \leftarrow 1$  até 16
- 2     faça MAKESETUR ( $x_i$ )
- 3 para  $i \leftarrow 1$  até 15 em passos de 2
- 4     faça UNIONUR ( $x_i, x_{i+1}$ )
- 5 para  $i \leftarrow 1$  até 13 em passos de 4
- 6     faça UNIONUR ( $x_i, x_{i+2}$ )
- 7 UNIONUR ( $x_1, x_5$ )
- 8 UNIONUR ( $x_{11}, x_{13}$ )
- 9 UNIONUR ( $x_1, x_{10}$ )
- 10 FINDSET ( $x_2$ )
- 11 FINDSET ( $x_9$ )

**Exr 13.5** Considere o repertório de operações MAKESETUR, UNIONUR e FINDSET. Faça uma figura da floresta produzida pela seguinte sequência de operações:

- 0 para  $i \leftarrow 1$  até 9
- 1     faça MAKESETUR ( $x_i$ )
- 2 UNIONUR ( $x_2, x_1$ )
- 3 UNIONUR ( $x_4, x_3$ )
- 4 UNIONUR ( $x_5, x_4$ )
- 5 UNIONUR ( $x_4, x_2$ )
- 6 UNIONUR ( $x_7, x_6$ )
- 7 UNIONUR ( $x_9, x_8$ )
- 8 UNIONUR ( $x_6, x_8$ )
- 9 UNIONUR ( $x_7, x_5$ )

**Exr 13.6** Considere a estrutura gerada por uma sequências de operações MAKESETUR, UNIONUR e FINDSET. Prove que, para cada nó  $x$  da estrutura,  $\text{rank}[x]$  é a altura<sup>1</sup> de  $x$ .

**Exr 13.7** Considere a estrutura gerada por uma sequências de operações MAKESETUR, UNIONUR e FINDSET. Mostre que se  $\text{rank}[x] = k$  para algum nó  $x$  então a estrutura tem pelo menos  $2^k$  nós e pelo menos  $2^{k-1}$  ocorrências de UNIONUR. Mostre  $2^k$  MAKESETS e  $2^{k-1}$  UNIONUR são suficientes para produzir um nó  $x$  tal que  $\text{rank}[x] \geq k$ .

**Exr 13.8** [Variante de CLRS Corollary 21.5] Considere a estrutura gerada por uma sequências de operações MAKESETUR, UNIONUR e FINDSET. Mostre que para cada nó  $x$  na estrutura tem-se

$$\text{rank}[x] \leq \lg n_x ,$$

sendo  $n_x$  é o número de nós da árvore que contém  $x$ . (Sugestão: Faça indução na sequência de operações.)

**Exr 13.9** [Variante de CLRS 21.4-2] Considere uma sequência de operações MAKESETUR, UNIONUR e FINDSET,  $n$  das quais são do primeiro tipo. Mostre que  $\text{rank}[x] \leq \lfloor \lg n \rfloor$  para todo nó  $x$ . (Veja o exercício 13.8.)

**Exr 13.10** Deduza do exercício 13.8 que qualquer sequência de  $m$  operações MAKESETUR, UNIONUR e FINDSET, sendo  $n$  delas do primeiro tipo, consome  $O(m \lg n)$  unidades de tempo.

**Exr 13.11** [CLRS 21.4-4] Considere a estrutura gerada por uma sequências de operações MAKESETUR, UNIONUR e FINDSET,  $n$  das quais são MAKESETUR. Mostre que o consumo de tempo amortizado de cada operação da sequência é  $O(\lg n)$ . (Veja exercício 13.10.)

**Exr 13.12** [CLRS 21.3-3] Dê uma sequência de  $m$  operações MAKESETUR, UNIONUR e FINDSET, sendo  $n$  delas do primeiro tipo, que consuma  $\Omega(m \lg n)$  unidades de tempo.

## 13.2 Path compression

O truque *path compression* não altera MAKESET, mas modifica as duas outras operações da seguinte maneira:

FINDSETPC ( $x$ )  
 1 se  $x \neq \text{pai}[x]$   
 2 então  $\text{pai}[x] \leftarrow \text{FINDSETPC}(\text{pai}[x])$   
 3 devolva  $\text{pai}[x]$

UNIONPC ( $x, y$ )  
 1  $x' \leftarrow \text{FINDSETPC}(x)$   
 2  $y' \leftarrow \text{FINDSETPC}(y)$   
 3  $\text{pai}[y'] \leftarrow x'$

---

<sup>1</sup> A altura de um nó  $x$  é a mais longa sequência da forma  $(u, \text{pai}[u], \text{pai}[\text{pai}[u]], \dots, x)$ . Não confunda altura com profundidade.

**Exr 13.13** Analise o comportamento de uma sequência arbitrária de operações MAKESET, FINDSETPC e UNIONPC.

### 13.3 Union by rank & path compression

Se acrescentarmos o truque *path compression* ao truque *union by rank*, a operação MAKESETUR não se altera mas as duas outras operações devem ser reescritas assim:

```
FINDSETPC ( $x$ )
1 se  $x \neq \text{pai}[x]$ 
2 então  $\text{pai}[x] \leftarrow \text{FINDSETPC}(\text{pai}[x])$ 
3 devolva  $\text{pai}[x]$ 
```

```
UNIONURPC ( $x, y$ )  $\triangleright$  supõe que  $x$  e  $y$  estão em árvores diferentes
1  $x' \leftarrow \text{FINDSETPC}(x)$ 
2  $y' \leftarrow \text{FINDSETPC}(y)$ 
3 se  $\text{rank}[x'] > \text{rank}[y']$ 
4 então  $\text{pai}[y'] \leftarrow x'$ 
5 senão  $\text{pai}[x'] \leftarrow y'$ 
6 se  $\text{rank}[x'] = \text{rank}[y']$ 
7 então  $\text{rank}[y'] \leftarrow \text{rank}[y'] + 1$ 
```

**Exr 13.14** [CLRS 21.3-2] Escreva uma versão iterativa da operação FINDSETPC.

**Exr 13.15** [CLRS 21.3-1] Considere o repertório de operações MAKESETUR, UNIONURPC e FINDSETPC. Faça uma figura da floresta produzida pela seguinte sequência de operações:

```
1 para  $i \leftarrow 1$  até 16
2 faça MAKESETUR ( $x_i$ )
3 para  $i \leftarrow 1$  até 15 em passos de 2
4 faça UNIONURPC ( $x_i, x_{i+1}$ )
5 para  $i \leftarrow 1$  até 13 em passos de 4
6 faça UNIONURPC ( $x_i, x_{i+2}$ )
7 UNIONURPC ( $x_1, x_5$ )
8 UNIONURPC ( $x_{11}, x_{13}$ )
9 UNIONURPC ( $x_1, x_{10}$ )
10 FINDSETPC ( $x_2$ )
11 FINDSETPC ( $x_9$ )
```

**Exr 13.16** Considere o repertório de operações MAKESETUR, UNIONURPC e FINDSETPC. Faça uma figura da floresta produzida pela seguinte sequência de operações:

```
0 para  $i \leftarrow 1$  até 9
1 faça MAKESETUR ( $x_i$ )
```

- 2 UNIONURPC ( $x_2, x_1$ )
- 3 UNIONURPC ( $x_4, x_3$ )
- 4 UNIONURPC ( $x_5, x_4$ )
- 5 UNIONURPC ( $x_4, x_2$ )
- 6 UNIONURPC ( $x_7, x_6$ )
- 7 UNIONURPC ( $x_9, x_8$ )
- 8 UNIONURPC ( $x_6, x_8$ )
- 9 UNIONURPC ( $x_7, x_5$ )

Repita com UNIONURPC ( $x_5, x_7$ ) no lugar da linha 9.

**Exr 13.17** Considere a estrutura gerada por uma sequências de operações MAKESETUR, UNIONURPC e FINDSETPC. Reescreva FINDSETPC de tal forma que, em qualquer instante,  $rank[x]$  seja igual à altura de  $x$ .

**Exr 13.18** [Variante de CLRS 21.4-2] Considere a estrutura gerada por uma sequências de operações MAKESETUR, UNIONURPC e FINDSETPC. Digamos que  $h[x]$  é a altura do nó  $x$ . Mostre que  $rank[x] \geq h[x]$ . Mostre que UNIONUR nem sempre “pendura” a árvore mais baixa na mais alta: dê um exemplo em que  $pai[x] = x$ ,  $pai[y] = y$ ,  $h[y] > h[x]$  mas UNIONUR ( $x, y$ ) faz  $pai[y] \leftarrow x$ .

**Exr 13.19** [CLRS Corollary 21.5] Considere a estrutura gerada por uma sequências de operações MAKESETUR, UNIONURPC e FINDSETPC. Mostre que para cada nó  $x$  na estrutura tem-se

$$rank[x] \leq \lg n_x ,$$

sendo  $n_x$  é o número de nós da árvore que contém  $x$ . (Veja o exercício 13.8.)

**Exr 13.20** [CLRS 21.4-2] Considere a estrutura gerada por uma sequências de operações MAKESETUR, UNIONURPC e FINDSETPC,  $n$  das quais são MAKESETUR. Mostre que  $rank[x] \leq \lfloor \lg n \rfloor$  para todo nó  $x$ . (Veja o exercício 13.19.)

**Exr 13.21** [Log estrela] A função função “torre” é definida assim:  $T(0) = 1$  e  $T(k) = 2^{T(k-1)}$  para todo inteiro positivo  $k$ . Portanto,<sup>2</sup>

$$T(1) = 2, \quad T(2) = 2^2, \quad T(3) = 2^{2^2}, \quad T(4) = 2^{2^{2^2}}, \quad \text{etc.}$$

Observe que  $\lg T(k) = T(k-1)$  para  $k = 1, 2, 3, \dots$

Para todo número  $n > 0$ , seja  $\lg^{(0)} n := n$ . Para todo inteiro positivo  $i$  e todo  $n > 2^{i-1}$ , seja  $\lg^{(i)} n := \lg \lg^{(i-1)} n$ . Em particular,  $\lg^{(1)} n = \lg n$ ,  $\lg^{(2)} n = \lg \lg n$  e  $\lg^{(3)} n = \lg \lg \lg n$ . É claro que  $\lg^{(i)} T(k) = T(k-i)$ . Em particular,  $\lg^{(k)} T(k) = 1$ . Assim,  $\lg^{(k)}$  é a inversa composicional de  $T(k)$ .

A função  $\lg^*$  é definida da seguinte maneira. Para todo número  $n > 1$ , seja  $\lg^* n$  o inteiro positivo  $k$  tal que

$$T(k-1) < n \leq T(k) .$$

Assim, para todo número  $n > 1$ ,  $\lg^* n$  é o único número natural  $k$  tal que  $0 < \lg^{(k)} n \leq 1$ . (CLRS [CLRS01, p.55] diz isso assim:  $\lg^* n := \min \{k \geq 0 : \lg^{(k)} n \leq 1\}$ .)

Verifique todos os fatos enunciados acima. Prove que  $\lg^* n$  não é  $O(1)$ .

---

<sup>2</sup> Não confunda  $2^{2^{2^2}}$  com  $((2^2)^2)^2$ .

**Exr 13.22** Qual a relação entre  $\lg^*(2^n)$  e  $\lg^* n$ ?

**Exr 13.23** É verdade que  $\lg^*(\lg n) = O(\lg(\lg^* n))$ ?

**Exr 13.24** [Difícil] Considere uma sequência de  $m$  operações MAKESETUR, UNIONURPC e FINDSETPC,  $n$  das quais são do primeiro tipo. Mostre que a sequência consome

$$O(m \lg^* n)$$

unidades de tempo. (Como  $\lg^* n$  cresce *muito* lentamente com  $n$ , o consumo de tempo é essencialmente  $O(m)$ .)

**Exr 13.25** É verdade que uma “floresta de conjuntos disjuntos” (*disjoint-set forest*) com  $n$  nós pode ser implementada de modo que cada operação UNIONURPC consuma  $O(\lg^* n)$  unidades de tempo?

**Exr 13.26** Imagine que temos  $n$  operações MAKESETUR seguidas de uma sequência arbitrária de operações UNIONURPC e FINDSETPC. É verdade que, em qualquer momento, todo caminho que leva de um nó até a correspondente raiz tem comprimento  $\leq \lg^* n$ ?

# Capítulo 14

## Grafos não-dirigidos

Um *grafo não-dirigido*, ou simplesmente *grafo*, é um par  $(V, E)$  de conjuntos finitos tal que  $E \subseteq V^{(2)}$ . Aqui,  $V^{(2)}$  denota o conjunto dos pares não-ordenados de elementos de  $V$ , ou seja, o conjunto dos pares  $\{u, v\}$  com  $u \in V, v \in V, u \neq v$ . Os elementos de  $V$  são chamados *vértices* e os elementos de  $E$  são chamados *arestas*. É claro que em qualquer grafo  $(V, E)$  tem-se  $0 \leq |E| \leq \frac{1}{2}n(n-1) < n^2$ , sendo  $n := |V|$ .

Às vezes é apropriado denotar uma aresta  $\{u, v\}$  por  $(u, v)$  ou até por  $uv$ . Dois vértices  $u$  e  $v$  são *vizinhos* ou *adjacentes* se  $uv$  é aresta. Se  $G$  é um grafo, o conjunto dos vértices de  $G$  é denotado por  $V[G]$  e o conjunto das arestas é denotado por  $E[G]$ . Exemplo de grafo:  $V = \{a, b, c, d, e, f, g, h, i\}$  e  $E = \{ab, bc, cd, de, ef, fg, gh, ha, bh, cf, ci, df, ig, ih, ic\}$ .

Um *caminho* é uma sequência de vértices em que cada vértice é adjacente ao anterior. Mais exatamente, um caminho é uma sequência  $(v_0, v_1, \dots, v_{k-1}, v_k)$  de vértices tal que, para todo  $i$ , o par  $(v_{i-1}, v_i)$  é uma aresta. Suporemos tacitamente que  $(v_{i-1}, v_i) \neq (v_{j-1}, v_j)$  sempre que  $i \neq j$ , ou seja, que o caminho não tem arestas repetidas. Um caminho é *simples* se não tem vértices repetidos.

Um grafo é *conexo* se, para cada par  $x, y$  de seus vértices, existe caminho de  $x$  a  $y$ . Em termos um tanto vagos, um *componente* de um grafo é um “pedaço” conexo maximal do grafo. É claro que um grafo é conexo sse tem apenas 1 componente.

Fato básico: Se um grafo  $(V, E)$  é conexo então  $|E| \geq |V| - 1$ .

**A estrutura recursiva de grafos.** Dentro de todo grafo existem muitos outros grafos. Dentre esses outros grafos destacam-se os subgrafos e as contrações.

Dado um grafo  $G = (V, E)$ , seja  $X$  um subconjunto de  $V$  e  $F$  um subconjunto de  $X^{(2)}$ . Se  $F \subseteq E$  então  $(X, F)$  é *subgrafo* de  $G$ . Exemplo: todo componente de  $G$  é um subgrafo de  $G$ . Outro exemplo: o mapa das ruas de uma cidade é subgrafo do mapa completo (ruas e estradas) do país.

Dado um grafo  $G = (V, E)$ , seja  $\mathcal{X}$  uma partição<sup>1</sup> de  $V$  e  $\mathcal{F}$  um subconjunto de  $\mathcal{X}^{(2)}$ . Suponha que  $(X, E \cap X^{(2)})$  é conexo para cada  $X$  em  $\mathcal{X}$ . Suponha ainda que  $\{X, Y\} \in \mathcal{F}$  sse existe  $\{x, y\}$  em  $E$  tal que  $x \in X$  e  $y \in Y$ . Se essas duas condições estiverem satisfeitas então o grafo  $(\mathcal{X}, \mathcal{F})$  é uma

---

<sup>1</sup> Uma *partição* de um conjunto  $V$  é uma coleção  $\mathcal{P}$  de subconjuntos de  $V$  tal que cada elemento de  $V$  pertence a exatamente um dos elementos de  $\mathcal{P}$ .

contração<sup>2</sup> de  $G$ . Exemplo: a coleção de todos os componentes de  $G$  é uma contração de  $G$ . Outro exemplo: o mapa das estradas de um país é contração do mapa completo (ruas e estradas) do país.

Seja  $uv$  uma aresta de um grafo  $G$ . Seja  $\mathcal{X}$  a partição  $\{\{u, v\}\} \cup \{\{x\} : x \in V - \{u, v\}\}$  de  $V[G]$ . Seja  $\mathcal{F}$  a subcoleção de  $\mathcal{X}^{(2)}$  induzido por  $E$  da maneira óbvia. Então  $(\mathcal{X}, \mathcal{F})$  é uma contração de  $G$ . Essa contração é denotada por  $G/uv$ .

**Cálculo de componentes.** Considere o problema de calcular o número de componentes de um grafo  $G$ . O seguinte algoritmo resolve o problema:

```
COMPONENTS-REC ( $G$ )
1  se  $E[G] = \emptyset$ 
2    então devolva  $|V[G]|$ 
3  seja  $uv$  um elemento de  $E[G]$ 
4  devolva COMPONENTS-REC ( $G/uv$ )
```

Como implementar a operação “ $G/uv$ ” de maneira eficiente? A resposta está na estrutura “floresta de conjuntos disjuntos” (veja capítulo 13). Cada árvore da estrutura é um bloco da partição da  $V[G]$ . Eis uma versão iterativa do algoritmo:

```
COMPONENTS ( $V, E$ )
1  para cada  $v$  em  $V$  faça
2    MAKESET ( $v$ )
3     $c \leftarrow |V|$ 
4  para cada  $(u, v)$  em  $E$  faça
5    se FINDSET ( $u$ )  $\neq$  FINDSET ( $v$ )
6      então UNION ( $u, v$ )
7       $c \leftarrow c - 1$ 
8  devolva  $c$ 
```

**Árvores.** Uma árvore é um grafo conexo  $(V, A)$  tal que  $|A| = |V| - 1$ . Portanto, uma árvore é um grafo “minimamente conexo”. Observe que árvores não têm os conceitos “raiz”, “pai”, “filho”.

Fato básico: Um grafo conexo é uma árvore sse não tem ciclos.

**Exr 14.1** Seja  $(V, E)$  um grafo conexo. Mostre que  $|E| \geq |V| - 1$ . A reciproca é verdadeira?

**Exr 14.2** Mostre que todo grafo conexo  $(V, E)$  tal que  $|E| = |V| - 1$  não tem ciclos. Mostre que todo grafo conexo sem ciclos é uma árvore.

**Exr 14.3 [CLRS 21.1-3]** Quando o algoritmo COMPONENTS é aplicado a um grafo  $G = (V, E)$  com  $k$  componentes, quantas vezes FINDSET é chamado? Quantas vezes UNION é chamado? Dê respostas em termos de  $k$ ,  $|V|$  e  $|E|$ .

---

<sup>2</sup> Uma contração é um tipo especial de um *minor* (diga “máior”) do grafo.

**Exr 14.4** Suponha que o algoritmo COMPONENTS usa as versões MAKESETUR, FINDSETPC e UNIONURPC de MAKESET, FINDSET e UNION. Mostre que o algoritmo consome  $O((V+E) \lg^* V)$  unidades de tempo, onde “ $V$ ” e “ $E$ ” são abreviaturas de “ $|V|$ ” e “ $|E|$ ” respectivamente. Se  $V = O(E)$ , o consumo de tempo será  $O(E \lg^* V)$ .

## 14.1 Árvores

**Exr 14.5** Seja  $T$  uma árvore com  $n$  vértices. Considere o seguinte problema  $CAM(x, y)$ : Dados dois vértices  $x$  e  $y$  de  $T$ , encontrar um caminho que liga  $x$  a  $y$  em  $T$ . (1) Elabore uma estrutura de dados para a árvore  $T$  que permita resolver  $CAM(x, y)$  em tempo  $O(d)$ , sendo  $d$  a distância de  $x$  a  $y$ . (2) Esboce um algoritmo que resolva  $CAM(x, y)$  em tempo  $O(d)$ . (3) Sugira como montar a sua estrutura em tempo  $O(n)$ . (Observação: No item (1),  $O(n)$  não basta. Sugestão: Para construir a estrutura, faça uma busca em largura a partir de um vértices arbitrário.)

## Capítulo 15

# Árvores geradoras de peso mínimo

Uma árvore geradora (= *spanning tree*) de um grafo  $(V, E)$  é uma árvore  $(V, A)$  que é subgrafo de  $(V, E)$ . Definição alternativa: uma árvore geradora de  $(V, E)$  é um subconjunto  $A$  de  $E$  tal que  $(V, A)$  é árvore. Um grafo  $G$  tem uma árvore geradora sse  $G$  é conexo.

Problema: Dado grafo conexo  $(V, E)$  com pesos nas arestas encontrar uma árvore geradora de peso mínimo.<sup>1</sup>

Cada aresta  $uv$  tem um peso  $w(uv)$ , que é um número real arbitrário (positivo, negativo ou nulo). É claro que  $w(vu) = w(uv)$ . O peso de  $A \subseteq E$  é  $w(A) := \sum_{uv \in A} w(uv)$ .

Para simplificar a linguagem, diremos às vezes “árvore ótima” no lugar de “árvore geradora de peso mínimo”. Para ressaltar o papel do peso  $w$ , diremos às vezes “árvore ótima de  $(G, w)$ ” no lugar de “árvore ótima de  $G$ ”.

Desafio: inventar um algoritmo que encontre uma árvore ótima em tempo  $O(V+E)$  ou, pelo menos, em tempo  $O((V+E) \lg^* V)$ .<sup>2</sup> Observe que o tempo necessário para arranjar as arestas em ordem crescente de peso é  $\Omega(E \lg E)$ .

**Exr 15.1** Discuta o seguinte algoritmo guloso, que promete resolver o problema da árvore geradora de peso mínimo: para cada vértice  $v$ , escolha uma aresta de peso mínimo dentre as que incidem em  $v$ ; devolva o conjunto de arestas assim escolhido.

**Exr 15.2** Seja  $uv$  uma aresta de um grafo conexo  $G$ . Seja  $A'$  o conjunto de arestas de uma árvore geradora do grafo  $G/uv$ . Seja  $A$  qualquer dos conjuntos de arestas de  $G$  que correspondem a  $A'$ . (Se algum vértice de  $G$  é vizinho de ambos  $u$  e  $v$  então  $A$  pode ser definido de diversas maneiras.) Mostre que  $A \cup \{uv\}$  é o conjunto de arestas de uma árvore geradora de  $G$ .

**Exr 15.3** Seja  $e$  uma aresta de peso mínimo em um grafo conexo com pesos nas arestas. Discuta as seguintes proposições. Proposição 1: “A aresta  $e$  pertence a toda árvore geradora de peso mínimo do grafo.” Proposição 2: “A aresta  $e$  pertence a alguma árvore geradora de peso mínimo do grafo.”

**Exr 15.4** [Greedy-choice property, CLRS 23.1-1] Seja  $(u, v)$  uma aresta de peso mínimo num grafo conexo  $G$ . Mostre que  $(u, v)$  pertence a alguma árvore geradora de peso mínimo de  $G$ .

<sup>1</sup> Minimum spanning tree ou MST.

<sup>2</sup> Existem algoritmos sofisticados que consomem tempo  $O(E)$  se o grafo for denso, ou seja, se  $E = \Omega(V^2)$ .

**Exr 15.5** [CLRS 23.1-5] Seja  $C$  um ciclo de um grafo conexo  $G$  e seja  $e$  uma aresta de peso máximo em  $C$  (portanto, nenhuma das arestas de  $C$  tem peso maior que o de  $e$ ). Prove que alguma árvore geradora de peso mínimo de  $G$  não usa  $e$ .

**Exr 15.6** [CLRS 23.1-8] Seja  $T$  uma árvore geradora de peso mínimo de um grafo conexo  $G$ . Seja  $w_1, \dots, w_{n-1}$  a lista de pesos de arestas de  $T$ , em ordem crescente. Mostre que toda árvore geradora de peso mínimo de  $G$  tem esta mesma lista de pesos de arestas.

## 15.1 Algoritmo de Kruskal

O algoritmo de Kruskal recebe um grafo conexo  $G$  com pesos  $w$  nas arestas e devolve uma árvore geradora de peso mínimo de  $(G, w)$ . O algoritmo supõe que o grafo dado é conexo:

```
RASCUNHO-REC ( $G, w$ )
1 se  $E[G] = \emptyset$ 
2 então devolva  $\emptyset$   $\triangleright G$  tem 1 só vértice
3 escolha  $uv$  em  $E[G]$  que tenha  $w$  mínimo
4  $w' \leftarrow \text{CONTRAIPELOS} (G, uv, w)$ 
5  $A \leftarrow \text{RASCUNHO-REC} (G/uv, w')$ 
6 devolva  $\{uv\} \cup A$ 
```

A rotina CONTRAIPELOS define os pesos  $w'$  assim: para toda aresta  $XY$  de  $G/uv$ ,  $w'(XY) := \min \{w(xy) : xy \in E[G], x \in X, y \in Y\}$ . Como implementar  $G/uv$  e a operação CONTRAIPELOS de maneira eficiente? Resposta: floresta de conjuntos disjuntos.

```
MSTKRUSKAL ( $V, E, w$ )
1 coloque  $E$  em ordem crescente de  $w$ 
2  $A \leftarrow \emptyset$ 
3 para cada  $v$  em  $V$ 
4   faça  $\text{MAKESETUR} (v)$ 
5 para cada  $uv \in E$  em ordem crescente de  $w$ 
6   faça se  $\text{FINDSETPC} (u) \neq \text{FINDSETPC} (v)$ 
7     então  $A \leftarrow A \cup \{uv\}$ 
8     UNIONURPC ( $u, v$ )
9 devolva  $A$ 
```

**Exr 15.7** Que acontece se o grafo  $G$  submetido ao algoritmo MSTKRUSKAL for desconexo?

**Exr 15.8** Critique a seguinte implementação do algoritmo de Kruskal:

```
KRSK ( $V, E, w$ )
1  $m \leftarrow |E|$ 
2 sejam  $e_1, \dots, e_m$  os elementos de  $E$  ordem crescente de  $w$ 
3  $n \leftarrow |V|$ 
```

```

4    $A \leftarrow \{e_1, \dots, e_{n-1}\}$ 
5   devolva  $A$ 

```

**Exr 15.9** Enuncie e prove a *optimal substructure property* e a *greedy-choice property* para o algoritmo de Kruskal. Essas propriedades são usadas para provar que o algoritmo está correto.

**Exr 15.10** Seja  $G$  um grafo conexo com pesos nas arestas. Seja  $w$  a função que associa pesos às arestas de  $G$ . Seja  $T$  uma árvore geradora de peso mínimo de  $(G, w)$  e  $uv$  uma aresta de  $T$ . Prove que  $T/uv$  é uma árvore geradora de peso mínimo de  $(G/uv, w')$ . Os pesos  $w'$  das arestas de  $G/uv$  são definidos assim: o peso de uma aresta  $XY$  de  $G/uv$  é o mínimo dos números  $w(xy)$  sendo  $xy$  uma aresta de  $G$  com  $x \in X$  e  $y \in Y$ .

**Exr 15.11** Escreva uma versão do algoritmo de Kruskal “por extenso”, isto é, com pseudocódigo explícito para as operações `MAKESET`, `FINDSET` e `UNION` incorporado ao pseudocódigo do algoritmo. (Escreva código “sob medida”; não escreva código mais geral que o necessário.)

**Exr 15.12** Seja  $G$  um grafo conexo com  $n$  vértices e  $m$  arestas. As arestas são numeradas de 1 a  $m$  e a  $i$ -ésima aresta tem um peso  $w_i$  que pertence ao conjunto  $\{1, 2, \dots, 9\}$ . Descreva informalmente a implementação assintoticamente mais rápida que você conhece para o algoritmo de Kruskal nesse caso. Qual o consumo de tempo, em notação O, de sua implementação? Escreva uma função `KRUSKAL` que implemente as idéias.

**Exr 15.13 [CLRS 23.2-4]** Seja  $G$  um grafo com pesos nas arestas. Suponha que todos os pesos pertencem ao conjunto  $\{1, 2, \dots, n\}$ , sendo  $n$  o número de vértices do grafo. Dê a melhor cota superior que puder para o algoritmo de Kruskal nesse caso. Repita o exercício supondo que os pesos pertencem ao conjunto  $\{1, 2, \dots, k\}$ , sendo  $k$  uma constante (ou seja, um inteiro que não depende do número de vértices e arestas de  $G$ ).

## 15.2 Algoritmo de Prim

Num grafo  $(V, E)$ , um corte é um par  $(S, \bar{S})$  de subconjuntos de  $V$  tal que  $\bar{S} = V - S$ . Uma aresta  $uv$  cruza um corte  $(S, \bar{S})$  se  $u \in S$  e  $v \in \bar{S}$  ou vice-versa.

Idéia do algoritmo de Prim: No começo de cada iteração, temos uma árvore  $(S, A)$  que é subgrafo de  $(V, E)$ . Cada iteração escolhe uma aresta de peso mínimo dentre as que cruzam o corte  $(S, \bar{S})$ . Todos os rascunhos do algoritmo de Prim a seguir supõem que o grafo dado é conexo e devolvem o peso de uma árvore ótima.

```

RASCUNHO-1 ( $V, E, w$ )
1   escolha qualquer  $r$  em  $V$ 
2    $S \leftarrow \{r\}$ 
3    $peso \leftarrow 0$ 
4   enquanto  $S \neq V$  faça
5        $min \leftarrow \infty$ 
6       para cada  $e$  em  $E$  faça
7           se  $e$  cruza o corte  $(S, \bar{S})$  e  $min > w(e)$ 

```

```

8       então  $min \leftarrow w(e)$ 
9       seja  $u_*$  a ponta de  $e$  em  $\bar{S}$ 
10       $S \leftarrow S \cup \{u_*\}$ 
11       $peso \leftarrow peso + min$ 
12      devolva  $peso$ 

```

No próximo rascunho, o grafo e os pesos são representados por uma matriz simétrica  $W$ : se  $uv$  é aresta então  $W[u, v] := w(uv)$ , senão  $W[u, v] := \infty$ . (A diagonal de  $W$  é irrelevante.) Cada vértice  $v$  tem uma  $chave[v]$  que é o peso de uma aresta mínima dentre as que ligam  $v$  a  $S$ .

RASCUNHO-2 ( $V, W, n$ )

```

1  escolha qualquer  $r$  em  $V$ 
2   $S \leftarrow \{r\}$ 
3  para cada  $u$  em  $V - \{r\}$  faça
4     $chave[u] \leftarrow W[r, u]$ 
5   $peso \leftarrow 0$ 
6  enquanto  $S \neq V$  faça
7     $min \leftarrow \infty$ 
8    para cada  $u$  em  $V - S$  faça
9      se  $min > chave[u]$ 
10     então  $min \leftarrow chave[u]$ 
11      $u_* \leftarrow u$ 
12      $S \leftarrow S \cup \{u_*\}$ 
13      $peso \leftarrow peso + chave[u_*]$ 
14     para cada  $v$  em  $V - S$  faça
15       se  $chave[v] > W[v, u_*]$ 
16         então  $chave[v] \leftarrow W[v, u_*]$ 
17  devolva  $peso$ 

```

No próximo rascunho, o grafo é representado por suas listas de adjacência:  $Adj[u]$  é a lista dos vértices vizinhos a  $u$ . O conjunto  $V - S$  é armazenado numa fila  $Q$ , com prioridades dadas por  $chave$ :

RASCUNHO-3 ( $V, Adj, w$ )

```

1  para cada  $u$  em  $V$  faça
2     $chave[u] \leftarrow \infty$ 
3  escolha qualquer  $r$  em  $V$ 
4  para cada  $u$  em  $Adj[r]$  faça
5     $chave[u] \leftarrow w(ru)$ 
6   $peso \leftarrow 0$ 
7   $Q \leftarrow FILAPRIORIDADES(V - \{r\}, chave)$ 
8  enquanto  $Q \neq \emptyset$  faça
9     $u \leftarrow EXTRAIMIN(Q)$ 
10    $peso \leftarrow peso + chave[u]$ 
11   para cada  $v$  em  $Adj[u]$  faça
12     se  $v \in Q$  e  $chave[v] > w(vu)$ 

```

```

13          então DECREMENTECHAVE ( $Q, chave, v, w(vu)$ )
14  devolva peso

```

A expressão FILAPRIORIDADES ( $U, chave$ ) constrói uma fila de prioridades com conjunto de elementos  $U$  e prioridades dadas por *chave*: quanto menor *chave*, maior a prioridade. A fila de prioridades pode ser implementada como um min-heap, por exemplo. A expressão DECREMENTECHAVE ( $Q, chave, v, novo$ ) faz  $chave[v] \leftarrow novo$  e ajusta a estrutura de  $Q$  de acordo.

O próximo rascunho começa com uma árvore sem vértices (e não a árvore  $(\{r\}, \emptyset)$ , como no rascunho anterior). Isso torna o código mais curto, embora a primeira iteração fique um pouco diferente das demais.

```

RASCUNHO-4 ( $V, Adj, w$ )
1  para cada  $u$  em  $V$  faça
2     $chave[u] \leftarrow \infty$ 
3  escolha qualquer  $r$  em  $V$ 
4   $chave[r] \leftarrow 0$ 
5   $peso \leftarrow 0$ 
6   $Q \leftarrow$  FILAPRIORIDADES ( $V, chave$ )
7  enquanto  $Q \neq \emptyset$  faça
8     $u \leftarrow$  EXTRAIMIN ( $Q$ )
9     $peso \leftarrow peso + chave[u]$ 
10   para cada  $v$  em  $Adj[u]$  faça
11     se  $v \in Q$  e  $chave[v] > w(vu)$ 
12       então DECREMENTECHAVE ( $Q, chave, v, w(vu)$ )
13  devolva peso

```

A versão final do algoritmo, devolve uma árvore geradora de peso mínimo (e não só o peso da árvore). A árvore é representada por um vetor de predecessores  $\pi$ .

```

MSTPRIM ( $V, Adj, w$ )
1  para cada vértice  $u$  faça
2     $chave[u] \leftarrow \infty$ 
3     $\pi[u] \leftarrow NIL$ 
4  escolha qualquer  $r$  em  $V$ 
5   $chave[r] \leftarrow 0$ 
6   $Q \leftarrow$  FILAPRIORIDADES ( $V, chave$ )
7  enquanto  $Q \neq \emptyset$  faça
8     $u \leftarrow$  EXTRAIMIN ( $Q$ )
9    para cada  $v$  em  $Adj[u]$  faça
10      se  $v \in Q$  e  $chave[v] > w(uv)$ 
11        então DECREMENTECHAVE ( $Q, chave, v, w(uv)$ )
12         $\pi[v] \leftarrow u$ 
13  devolva  $\pi$ 

```

No fim da execução do algoritmo, o conjunto de arestas  $A := \{(u, \pi(u)) : u \in V - \{r\}\}$  é o conjunto de arestas de uma árvore ótima.

**Exr 15.14** Discuta o seguinte algoritmo guloso, que promete resolver o problema da árvore geradora de peso mínimo. O algoritmo supõe que as arestas  $e_1, \dots, e_m$  grafo estão em ordem crescente de peso (ou seja, que  $w(e_1) \leq \dots \leq w(e_m)$ ).

```

1    $A \leftarrow \emptyset$ 
2   seja  $r$  um vértice qualquer
3    $S \leftarrow \{r\}$ 
4   para  $i \leftarrow 1$  até  $m$  faça
5     se  $e_i$  cruza o corte  $(S, \bar{S})$ 
6       então  $A \leftarrow A \cup \{e_i\}$ 
7         seja  $u_i$  a ponta de  $e_i$  em  $\bar{S}$ 
8          $S \leftarrow S \cup \{u_i\}$ 
9   devolva  $A$ 
```

**Exr 15.15** [CLRS 23.1-3] Imagine um grafo (não-dirigido) conexo com pesos associados às arestas. Suponha que uma aresta  $(u, v)$  pertence a uma árvore geradora de peso mínimo. Mostre que  $(u, v)$  é uma aresta de peso mínimo dentre as que cruzam algum corte do grafo.

**Exr 15.16** [Variante de CLRS 23.1-3] Imagine um grafo (não-dirigido) conexo com pesos associados às arestas. Suponha que uma aresta  $(u, v)$  pertence a uma árvore geradora de peso máximo. Mostre que  $(u, v)$  é uma aresta de peso máximo dentre as que cruzam algum corte do grafo.

**Exr 15.17** Que acontece se o grafo  $G$  submetido ao algoritmo MSTPRIM for desconexo?

**Exr 15.18** Explique o que é e para que serve o algoritmo de Prim. Faça um esboço vago do algoritmo, em português (não use pseudocódigo).

**Exr 15.19** Enuncie e prove a *greedy-choice property* para o algoritmo de Prim.

**Exr 15.20** Enuncie e prove a *optimal substructure property* para o algoritmo de Prim.

**Exr 15.21** Seja  $G$  um grafo conexo com pesos nas arestas. Seja  $(S, \bar{S})$  um corte de  $G$  e seja  $uv$  uma aresta que tem peso mínimo dentre as que cruzam o corte. (Portanto,  $w(uv) \leq w(xy)$  para toda aresta  $xy$  que cruza  $(S, \bar{S})$ , sendo  $w$  a função que atribui pesos às arestas.) Prove que  $uv$  pertence a alguma árvore geradora de peso mínimo de  $G$ .

**Exr 15.22** Qual o consumo de tempo dos algoritmos RASCUNHO-1, RASCUNHO-2, RASCUNHO-3, RASCUNHO-4 e MSTPRIM? Qual o mais eficiente?

**Exr 15.23** Considere o algoritmo MSTPRIM. Suponha que numa determinada iteração, para um determinado vértice  $v$ , o valor de  $\pi[v]$  é alterado. É verdade que o novo valor de  $\pi[v]$  é definitivo? Ou seja, é verdade que o valor de  $\pi[v]$  não será alterado pela iterações subsequentes?

**Exr 15.24** Escreva uma versão do algoritmo de Prim “por extenso” isto é, com pseudocódigo explícito para as operações de manipulação da fila de prioridades  $Q$  incorporado ao pseudocódigo do algoritmo.

**Exr 15.25** [Relacionado com CLRS 23.2-2] Suponha que  $G$  é um grafo (não-dirigido) completo (ou seja, para cada par  $u, v$  de vértices,  $(u, v)$  é uma aresta). Cada aresta  $(u, v)$  de  $G$  tem um peso arbitrário  $w(u, v)$ . Descreva, informalmente, a implementação mais rápida que você conhece para o algoritmo de Prim nesse caso.

**Exr 15.26** [CLRS 23.2-2] Uma matriz  $W[1..n, 1..n]$  representa um grafo com vértices  $1, \dots, n$ : o número  $W[u, v]$  é o peso da aresta  $uv$ . (Se  $W[u, v] = \infty$  então  $uv$  não é aresta do grafo. A diagonal da matriz é irrelevante.) Escreva uma versão do algoritmo de Prim que devolva o peso de uma árvore geradora de peso mínimo do grafo. O seu algoritmo deve consumir  $O(n^2)$  unidades de tempo.

**Exr 15.27** Escreva o pseudocódigo de um algoritmo que receba um grafo (não-dirigido) conexo  $G$  com pesos  $w$  associados às arestas (cada aresta  $(u, v)$  tem peso  $w(u, v)$ ) e devolva o peso de uma árvore geradora de peso máximo. Suponha que os vértices de  $G$  são  $1, \dots, n$  e o grafo é representado por sua matriz de adjacência. Escreva o algoritmo mais simples que puder, sem operações supérfluas. Escreva o algoritmo “por extenso”, isto é, sem chamar outros algoritmos. Faça um esboço da prova de correção do seu algoritmo. Diga, em notação  $O$ , qual o consumo de tempo de seu algoritmo.

### 15.3 Outras questões

**Exr 15.28** Considere o seguinte algoritmo que recebe um digrafo simétrico  $D$  com conjunto de vértices  $1, 2, \dots, n$  ( $n \geq 1$ ) e listas de adjacência  $Adj$  e pesos simétricos  $f$  nos arcos e devolve o peso de uma arborescência maximal com raiz 1.

1. Seja  $(U, W)$  uma bipartição de  $V$  tal que  $|U|$  e  $|V|$  diferem em no máximo 1.
2. Seja  $p_1$  o peso de uma árvore geradora mínima em  $G[U]$  (calculado recursivamente).
2. Seja  $p_2$  o peso de uma árvore geradora mínima em  $G[W]$  (calculado recursivamente).
3. Seja  $uw$  a aresta mais leve no corte  $(U, W)$ .
4. Devolva  $p_1 + f(uw) + p_2$ .

```

ÁRVORE ( $n, D, f, 1$ )
1   se  $n = 1$ 
1     então devolve 1
1     senão  $p \leftarrow \lfloor n \rfloor$ 
1       seja  $Adj$  as listas de adjacência restritas a  $\{1, 2, \dots, p\}$ 
1       seja  $Adj$  as listas de adjacência restritas a  $\{p + 1, \dots, n\}$ 
1       seja  $uv$  o arco mais leve dentre os ligam  $\{1, \dots, p\}$  a  $\{p + 1, \dots, n\}$ 
1        $t_1 \leftarrow$  ÁRVORE( $p, Adj_1, f_1, 1$ )
1        $t_2 \leftarrow$  ÁRVORE( $p + 1, \dots, n, Adj_2, f_2, v$ )
1       devolva  $t_1 + uv + t_2$ 
```

Prove que esse algoritmo devolve o peso de uma 1-arborescência maximal de peso mínimo ou dê um contra-exemplo.

**Exr 15.29** [Algoritmos alternativos para MST, CLRS 23-4] Considere o algoritmos abaixo. Cada um recebe um grafo  $(V, E)$  com pesos nas arestas e devolve um conjunto  $T$  de suas arestas. Para cada algoritmo, prove que  $T$  é uma árvore geradora mínima ou prove que  $T$  pode não ser uma árvore geradora mínima. Descreva a implementação mais eficiente de cada um dos algoritmos (quer ele calcule uma árvore geradora mínima quer não).

TALVEZ-MST-A  $(V, E, w)$

- 1 coloque as arestas em ordem decrescente de peso
- 2  $T \leftarrow E$
- 3 para cada  $e \in E$  em ordem não-decrescente de peso faça
  - 4 se  $T - \{e\}$  é um grafo conexo
  - 5 então  $T \leftarrow T - \{e\}$
- 6 devolva  $T$

TALVEZ-MST-B  $(V, E, w)$

- 1  $T \leftarrow \emptyset$
- 2 para cada  $e \in E$  em ordem arbitrária faça
  - 3 se  $T \cup \{e\}$  não tem ciclos
  - 4 então  $T \leftarrow T \cup \{e\}$
- 5 devolva  $T$

TALVEZ-MST-C  $(V, E, w)$

- 1  $T \leftarrow \emptyset$
- 2 para cada  $e \in E$  em ordem arbitrária faça
  - 3  $T \leftarrow T \cup \{e\}$
  - 4 se  $T$  tem um ciclo  $C$ 
    - 5 então seja  $e'$  uma aresta de peso máximo em  $C$
    - 6  $T \leftarrow T - \{e'\}$
- 7 devolva  $T$

**Exr 15.30** [Árvore geradora máxima] Escreva um algoritmo que receba um grafo conexo  $G$  com pesos nas arestas e determine o peso de uma árvore geradora de peso máximo.

**Exr 15.31** [Bottleneck spanning tree, CLRS 23-3] Uma *bottleneck spanning tree* de um grafo conexo  $G$  com pesos nas arestas é uma árvore geradora  $T$  de  $G$  cuja aresta mais pesada tem peso mínimo (entre todas as árvores geradoras de  $G$ ). O valor de uma bottleneck spanning tree  $T$  é o peso de uma aresta de peso máximo em  $T$ . Considere as seguintes questões: A. Mostre que toda árvore geradora de peso mínimo é uma bottleneck spanning tree. B. Dê um algoritmo linear (ou seja,  $O(V + E)$ ) que ao receber um grafo  $(V, E)$  com pesos nas arestas e um número  $b$  decide se o grafo tem uma bottleneck spanning tree de valor  $\leq b$ . C. Use o algoritmo da parte B como subrotina para para um algoritmo linear para o problema da bottleneck spanning tree. (Sugestão: Veja o algoritmo MST-REDUCE do problema CLRS 23-2.)

## Capítulo 16

# Caminhos de peso mínimo

**Exr 16.1** Considere a seguinte afirmação: “O algoritmo de Dijkstra pode dar uma resposta incorreta se o grafo tem arestas de peso nulo.” A afirmação é verdadeira ou falsa?

**Exr 16.2** Suponha que o algoritmo de Dijkstra está sendo executado sobre o grafo da figura, a partir do vértice  $s$ . Os números indicam os pesos das arestas. Suponha que as arestas em negrito já foram processadas pelo algoritmo. Qual será o próximo vértice e aresta escolhidos pelo algoritmo?

**Exr 16.3** Escreva uma versão do algoritmo de Dijkstra “por extenso” (ou seja, com o pseudocódigo das operações sobre a fila de vértices embutido no algoritmo). O algoritmo deve devolver o peso de um caminho de peso mínimo dentre os que ligam um vértice dado  $s$  a um vértice dado  $t$ . Suponha que os vértices do grafo são  $1, \dots, n$  e que o grafo é representado por sua matriz de adjacência. Faça um implementação simples, sem coisas supérfluas, que mantenha a fila de vértices em um vetor indexado pelos vértices. Diga quais os invariantes principais.

**Exr 16.4** Considere a seguinte afirmação: “Seja  $G$  um grafo com pesos não-negativos nas arestas. Sejam  $s$  e  $t$  dois vértices de  $G$  e  $P$  é um caminho de peso mínimo de  $s$  a  $t$ . Se somarmos 1 ao peso de cada aresta do grafo,  $P$  continua sendo um caminho de peso mínimo de  $s$  a  $t$ .” A afirmação é verdadeira ou falsa?

# Capítulo 17

## Grafos dirigidos

Grafos dirigidos também são conhecidos como *digrafos* e como *grafos orientados*.

### 17.1 Digrafos acíclicos e componentes fortes

**Exr 17.1** Suponha que  $G$  é um grafo dirigido estrito, isto é, sem arestas múltiplas e sem laços. Uma *celebridade* num tal grafo é um vértice  $v$  dotado da seguinte propriedade: para todo vértice  $w$  diferente de  $v$ , o par  $(w, v)$  é uma aresta (e portanto  $(v, w)$  não é uma aresta). Problema: encontrar uma celebridade num grafo estrito dado (ou decidir que o grafo não tem uma celebridade). A dificuldade está em resolver o problema em tempo  $O(n)$ , onde  $n$  é o número de vértices do grafo. Mais precisamente, resolver o problema com não mais que  $3n$  perguntas do tipo “ $(x, y)$  é uma aresta?”

**Exr 17.2** Considere a seguinte afirmação: “Uma enumeração acíclica (= ordenação topológica) dos vértices de qualquer grafo pode ser calculada em tempo linear (ou seja, em tempo  $O(n + m)$ , onde  $n$  é o número de vértices e  $m$  o número de arestas do grafo).” A afirmação é verdadeira ou falsa?

**Exr 17.3** Considere a seguinte afirmação: “Se acrescentarmos uma aresta a um DAG (grafo dirigido acíclico), o grafo resultante não será um DAG.” A afirmação é verdadeira ou falsa?

**Exr 17.4** Considere a seguinte afirmação: “Os componentes fortes de qualquer grafo podem ser calculados em tempo  $O(n^2)$ , onde  $n$  é o número de vértices do grafo.” A afirmação é verdadeira ou falsa?

**Exr 17.5 [CLRS 22.5-1]** Suponha que um grafo (dirigido)  $G$  tem  $f$  componentes fortes. Suponha que uma nova aresta é acrescentada a  $G$ . Quantos componentes fortes pode ter o novo grafo? Justifique.

**Exr 17.6** Quais os componentes fortes do grafo representado na figura? (Desenhe uma bola na figura em torno de cada componente forte.) Qual dos componentes será descoberto em primeiro lugar pelo algoritmo STRONGLY-CONNECTED-COMPONENTES? Qual dos componentes será descoberto por último? Desenhe o grafo  $G^{SCC}$  dos componentes fortes. Suponha que o algoritmo

percorre o conjunto  $V$  de vértices em ordem alfabética; suponha também que, para cada vértice  $v$ , a lista  $Adj[v]$  está arranjada em ordem alfabética.

# Capítulo 18

## Ordenação em tempo linear

Veja CLRS cap. 8.

### 18.1 Algoritmos lineares de ordenação

O algoritmo abaixo rearranja um vetor  $A[1 \dots n]$  em ordem crescente supondo que cada elemento de  $A$  pertence ao conjunto  $\{0, \dots, k\}$ :

```
COUNTINGSORT ( $A, n, k$ )
1   para  $i \leftarrow 0$  até  $k$ 
2     faça  $C[i] \leftarrow 0$ 
3   para  $j \leftarrow 1$  até  $n$ 
4     faça  $C[A[j]] \leftarrow C[A[j]] + 1$ 
5   para  $i \leftarrow 1$  até  $k$ 
6     faça  $C[i] \leftarrow C[i] + C[i - 1]$ 
7   para  $j \leftarrow n$  decrescendo até 1
8     faça  $B[C[A[j]]] \leftarrow A[j]$ 
9      $C[A[j]] \leftarrow C[A[j]] - 1$ 
10  para  $j \leftarrow 1$  até  $n$ 
11    faça  $A[j] \leftarrow B[j]$ 
```

O algoritmo abaixo, também conhecido como “ordenação digital”, rearranja um vetor  $A[1 \dots n]$  em ordem crescente supondo que cada elemento de  $A$  tem no máximo  $d$  dígitos decimais (ou seja,  $A[j] = a_d \dots a_2 a_1 := a_d 10^{d-1} + \dots + a_2 10^1 + a_1 10^0$ ):

```
RADIXSORT ( $A, n, d$ )
1   para  $i \leftarrow 1$  até  $d$  faça    $\triangleright$  de 1 até  $d$  e não o contrário!
2     ordenação  $A[1 \dots n]$  com dígito  $i$  como chave
```

**Exr 18.1** Dê o invariante que descreve o conteúdo do vetor  $C$  antes de cada execução da linha 8

do COUNTINGSORT. Mostre que COUNTINGSORT é estável. Mostre que COUNTINGSORT consome  $O(n + k)$  unidades de tempo.

**Exr 18.2** Considere o vetor  $A[1..n]$  definido por  $A[i] := i^2$  para todo  $i$ . Submeta esse vetor ao algoritmo COUNTINGSORT com argumento  $k = n^2$ . Discuta o comportamento do algoritmo. Quanto tempo o algoritmo consome em função  $n$ ?

**Exr 18.3** Qual o principal invariante do algoritmo RADIXSORT?

**Exr 18.4** Mostre que o consumo de tempo do RADIXSORT é  $O(dn)$ .

**Exr 18.5** O seguinte algoritmo promete rearranjar o vetor  $A[1..n]$  em ordem crescente supondo que cada  $A[i]$  está em  $\{0, \dots, k\}$ . O algoritmo está correto?

```
CSORT ( $A, n, k$ )
1  para  $i \leftarrow 0$  até  $k$ 
2    faça  $C[i] \leftarrow 0$ 
3  para  $j \leftarrow 1$  até  $n$ 
4    faça  $C[A[j]] \leftarrow C[A[j]] + 1$ 
5   $j \leftarrow 1$ 
6  para  $i \leftarrow 0$  até  $k$  faça
7    enquanto  $C[i] > 0$ 
8      faça  $A[j] \leftarrow i$ 
9       $j \leftarrow j + 1$ 
0       $C[i] \leftarrow C[i] - 1$ 
```

Qual o consumo de tempo do algoritmo?

**Exr 18.6** O seguinte algoritmo promete rearranjar o vetor  $A[1..n]$  em ordem crescente supondo que cada  $A[j]$  está em  $\{1, \dots, k\}$ . O algoritmo está correto? Estime, em notação  $O$ , o consumo de tempo do algoritmo.

```
WDSORT ( $A, n, k$ )
1   $i \leftarrow 1$ 
2  para  $a \leftarrow 1$  até  $k - 1$  faça
3    para  $j \leftarrow i$  até  $n$  faça
4      se  $A[j] = a$ 
5        então  $A[j] \leftrightarrow A[i]$  ▷ troca
6         $i \leftarrow i + 1$ 
```

**Exr 18.7** Suponha que os componentes do vetor  $A[1..n]$  estão todos em  $\{0, 1\}$ . Prove que  $n - 1$  comparações são suficientes para rearranjar o vetor em ordem crescente.

**Exr 18.8** Suponha que os elementos de  $A[1..n]$  pertencem todos ao conjunto  $\{1, 2, 3, 4\}$ . Escreva um algoritmo que rearranje o vetor em ordem crescente e gaste  $O(n)$  unidades de tempo.

**Exr 18.9** [CLRS 8.2-4] Suponha dados um número natural  $k$  e uma sequência  $\langle x_1, \dots, x_n \rangle$  de elementos do conjunto  $\{1, \dots, k\}$ . Parte 1: Escreva um algoritmo que pré-processe os dados de tal maneira que o problema descrito na parte 2 possa ser resolvido rapidamente. Escreva uma documentação correta do seu algoritmo. O tempo do pré-processamento deve ser  $O(n + k)$ . Parte 2: Escreva uma algoritmo que receba um par  $(a, b)$  de números do conjunto  $\{1, \dots, k\}$  e devolva o número de índices  $i$  para os quais  $a \leq x_i \leq b$ . O seu algoritmo pode usar o resultado de pré-processamento da parte 1. O consumo de tempo do seu algoritmo deve ser  $O(1)$ .

**Exr 18.10** [CLRS 8.3-4] Suponha dados  $n$  números inteiros, todos no intervalo  $[0, n^2 - 1]$ . É possível colocá-los em ordem crescente em tempo  $O(n)$ ? Em caso afirmativo, quanto espaço auxiliar é necessário para resolver o problema? Explique.

**Exr 18.11** Suponha que todos os elementos do vetor  $A[1..n]$  pertencem ao conjunto  $\{1, 2, \dots, n^3\}$ . Escreva um algoritmo que rearranje  $A[1..n]$  o vetor em ordem crescente em tempo  $O(n)$ .

**Exr 18.12** Descreva informalmente (não escreva código) um algoritmo que consome  $O(n)$  unidades de tempo para rearranjar em ordem crescente um vetor  $A[1..n]$  cujos elementos pertencem todos ao conjunto  $\{-2n, \dots, -1, 0, 1, \dots, 2n\}$ .

**Exr 18.13** Em que condições é possível aplicar o algoritmo RADIXSORT para rearranjar um conjunto de números em ordem crescente?

**Exr 18.14** Seja  $s_1, \dots, s_n$  e  $t_1, \dots, t_n$  elementos do conjunto  $\{1, \dots, k\}$ . Suponha que  $s_i < t_i$  para cada  $i$ . Para cada  $i$ , o par  $(s_i, t_i)$  representa o intervalo  $\{s_i, \dots, t_i\}$ . Problema: Queremos decidir se existem dois intervalos encaixados, ou seja, se existem  $i$  e  $j$  tais que  $s_i \leq s_j$  e  $t_i \geq t_j$ . Desafio: resolver o problema em tempo  $O(k + n)$ .

## 18.2 Cota inferior para o problema da ordenação

**Exr 18.15** Faça um desenho da árvore de decisão do algoritmo SELECTIONSORT aplicado a um vetor  $(a, b, c, d)$ .

**Exr 18.16** [CLRS 2.3-7] Nesta questão, considere algoritmos de ordenação baseados em comparações.

1. Mostre uma cota inferior justa para o número de comparações necessário para colocar 4 números em ordem crescente.
2. Mostre a correspondente cota superior.

**Exr 18.17** [CLRS 8.1-1] Qual a menor profundidade (= distância até a raiz) que uma folha pode ter em uma árvore de decisão que descreve um algoritmo de ordenação baseado em comparações?

**Exr 18.18** Mostre que qualquer algoritmo baseado em comparações necessita de pelo menos  $n - 1$  comparações para decidir se  $n$  números dados são todos iguais.

**Exr 18.19** [CLRS 8.1-2] Mostre que  $\lg(n!) = \Omega(n \lg n)$  sem usar a fórmula de Stirling. Sugestão: Calcule  $\sum_{k=n/2}^n \lg k$ . Use as técnicas de CLRS A.2.

**Exr 18.20** Um *algoritmo de busca* é qualquer algoritmo que resolva o seguinte *problema de busca*: dado um número inteiro  $x$  e uma sequência  $\langle a_1, \dots, a_n \rangle$  de números inteiros, encontrar  $j$  tal que  $x = a_j$  (vamos supor que um tal  $j$  sempre existe).

Mostre que qualquer algoritmo de busca baseado em comparações (ou seja, um algoritmo cujo controle de fluxo só depende de comparações entre  $x$  e elementos da sequência) faz  $\Omega(\lg n)$  comparações no pior caso.

**Exr 18.21** Suponha dadas sequências  $\langle a_1, \dots, a_n \rangle$  e  $\langle b_1, \dots, b_k \rangle$  de números inteiros positivos. Queremos produzir uma sequência  $\langle x_1, \dots, x_k \rangle$  definida assim:

$$x_i = \begin{cases} 1 & \text{se } b_i \in \{a_1, \dots, a_n\} \\ 0 & \text{caso contrário.} \end{cases}$$

Você tem duas alternativas: (a) ordenar  $\langle a_1, \dots, a_n \rangle$  e calcular  $x_i$  usando busca linear, (b) não ordenar  $\langle a_1, \dots, a_n \rangle$  e calcular  $x_i$  usando busca linear. Qual das duas alternativas é assintoticamente mais eficiente? Qual é assintoticamente mais eficiente se  $k = n$ ? Qual é assintoticamente mais eficiente se  $k = n$ ? e se  $k = \lfloor \lg n \rfloor$ ? e se  $k = \lfloor \sqrt[3]{n} \rfloor$ ? e se  $k = \lfloor \lg \lg n \rfloor$ ?

**Exr 18.22** [CLRS 8-6] Dadas sequências estritamente crescentes  $\langle a_1, \dots, a_n \rangle$  e  $\langle b_1, \dots, b_n \rangle$ , queremos ordenar a sequência  $\langle a_1, \dots, a_n, b_1, \dots, b_n \rangle$ . Mostre que  $2n - 1$  comparações são necessárias, no pior caso, para resolver o problema.

**Exr 18.23** [CLRS 8-5] Um vetor  $A[1..n]$  está *ordenado* se  $A[j] \leq A[j + 1]$  para  $j = 1, \dots, n - 1$ . Um vetor  $A[1..n]$  está  $k$ -*ordenado* se  $\frac{1}{k} \sum_{j=i}^{i+k-1} A[j] \leq \frac{1}{k} \sum_{j=i+1}^{i+k} A[j]$  para  $i = 1, 2, \dots, n - k$ . (A.) Descreva um vetor 1-ordenado. (B.) Mostre uma permutação de  $1, 2, \dots, 10$  que está 2-ordenada mas não ordenada. (C.) Mostre que  $A$  está  $k$ -ordenado se e somente se  $A[i] \leq A[i + k]$  para  $i = 1, 2, \dots, n - k$ . (D.) Descreva um algoritmo que gaste tempo  $O(n \lg \frac{n}{k})$  para  $k$ -ordenar um vetor  $A[1..n]$ . (E.) Mostre que um vetor  $k$ -ordenado pode ser ordenado em tempo  $O(n \lg k)$ . (Sugestão: use exercício 7.37.) (F.) Mostre que quando  $k$  é constante o consumo de tempo para  $k$ -ordenar um vetor é  $\Omega(n \lg n)$  no pior caso. (Sugestão: use a parte E junto com a cota inferior para ordenações baseadas em comparações.)

**Exr 18.24** Uma *ordem parcial* em um conjunto é uma relação binária, denotada usualmente por  $\prec$  tal que (i)  $a \not\prec a$  para cada  $a$  do conjunto e (ii) se  $a, b$  e  $c$  são tais que  $a \prec b$  e  $b \prec c$ , então  $a \prec c$ . Se  $a, b$  são tais que nem  $a \prec b$  nem  $b \prec a$ , eles são *incomparáveis*. Um vetor  $A[1..n]$  de elementos dessa ordem *estende* a ordem se, para todo  $i, j$ , se  $A[i] \prec A[j]$ , então  $i < j$ . Suponha definido um tipo `Op`, e dada uma função `compara(Op x, Op y)`, que devolve  $-1$  se  $x \prec y$ ,  $+1$  se  $y \prec x$  e  $0$  se  $x$  e  $y$  são incomparáveis.

Considere o problema: dado um vetor  $A[1..n]$  do tipo `Op`, reordenar esse vetor de modo a estender a ordem. Mostre que qualquer algoritmo que resolva esse problema usa  $\Omega(n^2)$  chamadas de `compara` no pior caso. Descreva um algoritmo que faz  $O(n^2)$  chamadas de `compara`.

## Capítulo 19

# Problemas completos em NP

**Teoria da complexidade.** As instâncias de um *problema de decisão* só admite resposta SIM ou NÃO. Toda instância de um problema de decisão é uma cadeia de caracteres. O *tamanho* da instância é o comprimento da cadeia.

Denotamos por P a classe dos problemas de decisão que podem ser resolvidos por algoritmos polinomiais. Denotamos por NP a classe dos problemas de decisão que têm certificados para a resposta SIM que podem ser verificados em tempo polinomial. Denotamos por co-NP a classe dos problemas de decisão que têm certificados para a resposta NÃO que podem ser verificados em tempo polinomial. Um problema de decisão está em co-NP se e somente se o seu complemento está em NP.

Dados problemas de decisão  $X$  e  $Y$ , dizemos que  $X \leq_P Y$  se existe uma redução polinomial de  $X$  a  $Y$ . Dizemos que um problema de decisão  $Y$  é NP-completo se  $X \leq_P Y$  para todo problema  $X$  em NP. O problema HAM-CYCLE é NP-completo.

O seguinte problema SUBSET-SUM é NP-completo: dados números naturais  $s_1, \dots, s_n$  e um número natural  $t$ , decidir se existe  $S \subseteq \{1, \dots, n\}$  tal que  $\sum_{i \in S} s_i = t$ .<sup>1</sup>

**Teoria dos grafos.** O *comprimento* de um caminho num grafo é o seu número de arestas. O *comprimento* de um ciclo é o seu número de arestas. Um caminho num grafo é *simples* se não tem vértices repetidos. Um ciclo num grafo é *simples* se não tem vértices repetidos (exceto o último, que é igual ao primeiro).

Um ciclo é *hamiltoniano* se for simples e passar por todos os vértices do grafo. Um caminho é *hamiltoniano* se for simples e passar por todos os vértices do grafo.

Um grafo  $G$  é *bipartido* se o seu conjunto de vértices admite uma bipartição  $(U, W)$  tal que toda aresta tem uma ponta em  $U$  e outra em  $W$ .

O problema HAM-CYCLE consiste no seguinte: decidir se um dado grafo tem um ciclo hamiltoniano. O problema HAM-PATH consiste no seguinte: dados vértices  $u$  e  $v$  de um grafo, decidir se o grafo tem um caminho hamiltoniano que começa em  $u$  e termina em  $v$ .

---

<sup>1</sup> CLRS exige que os  $s_i$  sejam distintos dois a dois. Esse caso particular do problema pode ser formulado assim: dado um conjunto  $T$  (finito, é claro) de números naturais e um número natural  $t$ , decidir se existe  $S \subseteq T$  tal que  $\sum_{s \in S} s = t$ . Esse caso particular do problema também é NP-completo.

Uma *clique* num grafo é um conjunto de vértices dois a dois adjacentes. Um *conjunto independente* é um conjunto de vértices dois a dois não-adjacentes. Uma *cobertura* de vértices é um conjunto de vértices que contém pelo menos uma ponta de cada aresta. O problema CLIQUE consiste no seguinte: dado um grafo  $G$  e um inteiro positivo  $k$ , decidir se  $G$  tem uma clique com  $\geq k$  vértices. O problema INDEPENDENT-SET consiste no seguinte: dado um grafo  $G$  e um inteiro positivo  $k$ , decidir se  $G$  tem um conjunto independente com  $\geq k$  vértices. O problema VERTEX-COVER consiste no seguinte: dado um grafo  $G$  e um inteiro positivo  $k$ , decidir se  $G$  tem uma cobertura de vértices com  $\leq k$  vértices.

**Exr 19.1** O algoritmo abaixo é polinomial?

```
EXEMPLO-BOBO ( $n$ )
1    $s \leftarrow 0$ 
2   para  $i \leftarrow 1$  até  $n$  faça
3        $s \leftarrow s + i$ 
4   devolva  $s$ 
```

Quanto tempo o algoritmo consome? Escreva um algoritmo MELHOR que consuma menos tempo e produza o mesmo efeito. Quanto tempo MELHOR consome? O algoritmo MELHOR é polinomial?

**Exr 19.2** Refaça o exercício 10.15.

**Exr 19.3** [CLRS 34.1-1] Seja LONGEST-PATH-LENGTH o seguinte problema: dados vértices  $u$  e  $v$  de um grafo  $G$ , encontrar o comprimento de caminho simples de comprimento máximo dentre os que começam em  $u$  e terminam em  $v$ . Seja LONGEST-PATH o seguinte problema de decisão: dados vértices  $u$  e  $v$  de um grafo  $G$ , e um número natural  $k$ , decidir se  $G$  tem caminho simples de comprimento  $\geq k$  que liga  $u$  a  $v$ . Mostre que LONGEST-PATH-LENGTH pode ser resolvido em tempo polinomial se e somente se LONGEST-PATH está em P.

**Exr 19.4** [CLRS 34.1-2] Enuncie o problema de encontrar um ciclo simples máximo num grafo. Enuncie o correspondente problema de decisão.

**Exr 19.5** [CLRS 34.1-4] O algoritmo de programação dinâmica para o problema booleano da mochila (veja exercício 10.17) é polinomial? Justifique sua resposta.

**Exr 19.6** [CLRS 34.2-2] Prove que um grafo bipartido com número ímpar de vértices não tem ciclo hamiltoniano.

**Exr 19.7** [CLRS 34.2-3] Prove que se o problema HAM-CYCLE estiver em P então o problema de encontrar a sequência de vértices de um ciclo hamiltoniano pode ser resolvido em tempo polinomial.

**Exr 19.8** [CLRS 34.2-6] Mostre que o problema HAM-PATH está em NP. O problema está em co-NP?

**Exr 19.9** Mostre que todo problema de decisão que pertence à classe P é polinomialmente redutível ao problema HAM-CYCLE.

**Exr 19.10** Mostre que o problema VERTEX-COVER é polinomialmente reduzível ao problema INDEPENDENT-SET. Mostre que INDEPENDENT-SET  $\leq_P$  CLIQUE.

**Exr 19.11** Seja INTERVALOS-DISJUNTOS o seguinte problema de decisão: dada uma coleção  $S$  de intervalos e um inteiro positivo  $k$ , decidir se  $S$  tem uma subcoleção com  $k$  ou mais intervalos dois a dois disjuntos. Mostre que INTERVALOS-DISJUNTOS  $\leq_P$  INDEPENDENT-SET. É verdade que INDEPENDENT-SET não é polinomialmente reduzível a INTERVALOS-DISJUNTOS?

**Exr 19.12** Mostre que o problema HAM-PATH é polinomialmente reduzível ao problema HAM-CYCLE. Mostre que o problema HAM-CYCLE é polinomialmente reduzível ao problema HAM-PATH.

**Exr 19.13** Suponha por um momento que o problema SUBSET-SUM está em P. Mostre que o seguinte problema pode ser resolvido em tempo polinomial: dado um conjunto  $S$  (finito, é claro) de números naturais e um número natural  $t$ , encontrar um subconjunto  $S'$  de  $S$  tal que  $\sum_{s \in S'} s = t$  (ou constatar que um tal  $S'$  não existe).

**Exr 19.14** Dados números inteiros  $s_1, s_2, \dots, s_n$ , diremos que uma partição  $(A, B)$  de  $\{1, \dots, n\}$  é *equilibrada* se  $\sum_{i \in A} s_i = \sum_{i \in B} s_i$ . Seja SET-PARTITION o seguinte problema de decisão: dados números inteiros  $s_1, s_2, \dots, s_n$ , decidir se existe uma partição equilibrada de  $\{1, \dots, n\}$ . Mostre como um algoritmo polinomial para o problema SET-PARTITION pode ser usado para encontrar uma partição equilibrada (ou dizer que uma tal partição não existe).

**Exr 19.15** Dados números inteiros  $s_1, s_2, \dots, s_n$ , diremos que uma tripartição  $(A, B, C)$  de  $\{1, \dots, n\}$  é *equilibrada* se  $\sum_{i \in A} s_i = \sum_{i \in B} s_i = \sum_{i \in C} s_i$ . Seja SETTRIPARTITIONPROB o seguinte problema de decisão: dados números inteiros  $s_1, s_2, \dots, s_n$ , decidir se existe uma tripartição equilibrada de  $\{1, \dots, n\}$ . Sabendo-se que o problema SET-PARTITION é NP-completo, mostre que SETTRIPARTITIONPROB também é NP-completo.

**Exr 19.16** Considere o seguinte problema SET-PARTITION: dados números inteiros  $s_1, s_2, \dots, s_n$ , decidir se existe uma partição  $(A, B)$  de  $\{1, \dots, n\}$  tal que  $\sum_{i \in A} s_i = \sum_{i \in B} s_i$ .<sup>2</sup> Prove que o problema SET-PARTITION é polinomialmente reduzível ao problema SUBSET-SUM.

**Exr 19.17** [Set-partition, CLRS 34.5-5] Prove que o problema SET-PARTITION (veja o exercício 19.16) é NP-completo. (Sugestão: o problema SUBSET-SUM é NP-completo.)

**Exr 19.18** [Bonnie and Clyde, CLRS 34-2] Bonnie e Clyde acabam de roubar um banco. Eles querem dividir o produto do roubo, que consiste em  $n$  objetos. Para cada um dos cenários abaixo, dê um algoritmo polinomial ou mostre que o problema é NP-completo. Em cada cenário, cada instância do problema é um conjunto de  $n$  números inteiros não-negativos. *Cenário a:* O produto do roubo é um conjunto de  $n$  moedas. Cada moeda tem um de dois possíveis valores (os valores são inteiros). Bonnie e Clyde insistem em ficar com exatamente o mesmo valor total cada um. *Cenário b:* O produto do roubo é um conjunto de  $n$  moedas e o valor de cada moeda é uma potência de 2 (ou seja, os possíveis valores das moedas são 1, 2, 4, etc.). Bonnie e Clyde insistem em ficar

<sup>2</sup> CLRS formulam o problema assim: Dado um conjunto  $S$  de números inteiros positivos (portanto, os números são distintos dois a dois), decidir se existe uma partição  $(X, Y)$  de  $S$  tal que  $\sum_{s \in X} s = \sum_{s \in Y} s$ . Mas esta simplificação não torna o problema mais fácil...

com exatamente o mesmo valor total cada um. *Cenário c:* O produto do roubo é um conjunto de  $n$  cheques. Cada cheque tem um valor inteiro, mas nada se sabe, a priori, sobre esse valor. Bonnie e Clyde insistem em ficar com exatamente o mesmo valor total cada um. *Cenário d:* Mesmo cenário que em c, mas Bonnie e Clyde aceitam uma divisão em que a diferença entre o valor total de um e de outro não passe de 100.

**Exr 19.19** Considere o seguinte problema FATORAÇÃO: dados dois números naturais  $m$  e  $n$  tais que  $2 < m \leq n$ , decidir se existe um número natural  $p$  tal que  $1 < p < m$  e  $p$  divide  $n$ .

Suponha que descobri um algoritmo polinomial, digamos  $A$ , que resolve o problema FATORAÇÃO. Mostre como  $A$  pode ser usado para resolver o seguinte problema em tempo polinomial: dados dois números naturais  $m$  e  $n$  tais que  $2 < m \leq n$ , encontrar um número natural  $p$  tal que  $1 < p < m$  e  $p$  divide  $n$  (ou constatar que um tal  $p$  não existe).

**Exr 19.20** Sejam  $s$  e  $t$  dois vértices de um grafo  $G$  que tem  $n$  vértices. Considere o seguinte par de problemas. Problema 1: Decidir se existe um caminho de  $s$  a  $t$  que passa por no mínimo  $n/2$  vértices. Problema 2: Decidir se existe um caminho de  $s$  a  $t$  que passa por no máximo  $n/2$  vértices. Quais dessas afirmativas são verdadeiras ou falsas com certeza?

- (a) Não existe algoritmo polinomial para nenhum destes problemas.
- (b) Existe algoritmo polinomial para pelo menos um destes problemas.
- (c) Existe algoritmo polinomial para exatamente um destes problemas.
- (d) Existem algoritmos polinomiais para ambos os problemas.

Como a resposta a esta pergunta será afetada se o problema “P = NP” for resolvido?

## 19.1 Questões mais abstratas

**Exr 19.21** Suponha que os algoritmos  $A$  e  $B$  transformam cadeias de caracteres em outras cadeias de caracteres. O algoritmo  $A$  consome  $O(n^2)$  unidades de tempo e o algoritmo  $B$  consome  $O(n^4)$  unidades de tempo, sendo  $n$  é o número de caracteres da cadeia de entrada. Considere agora o algoritmo  $AB$  que consiste na composição de  $A$  e  $B$ , com  $B$  recebendo a saída de  $A$  como entrada. Qual o consumo de tempo de  $AB$ ?

**Exr 19.22** [CLRS 34.2-5] Seja  $X$  um problema em NP. Mostre que existe um número  $k$  tal que qualquer instância de  $X$  pode ser resolvida em tempo  $2^{O(n^k)}$ .

**Exr 19.23** Discuta as seguintes afirmações: 1. “Todo problema de decisão está em NP.” 2. “Todo problema em NP é de decisão.”

**Exr 19.24** [Complemento] Formule a definição de complemento de um problema de decisão. Discuta a seguinte afirmação: “se um problema de decisão está em NP então o seu complemento também está em NP”.

**Exr 19.25** [CLRS 34.2-9] Prove que  $P \subseteq \text{co-NP}$ .

**Exr 19.26** [CLRS 34.3-2] Seja  $X$  um problema de decisão e  $\overline{X}$  o seu complemento. Mostre que  $X \leq_P \overline{X}$  se e somente se  $\overline{X} \leq_P X$ .

**Exr 19.27** Li em algum lugar a seguinte definição: “Dizemos que um problema de decisão  $X$  é polinomialmente redutível a um problema de decisão  $Y$  se, para toda instância  $I$  de  $X$ , existe um algoritmo polinomial que transforma  $I$  em uma instância de  $Y$  de tal forma que  $J$  é positiva se e somente se  $I$  é positiva.” O que há de errado com essa “definição”?

**Exr 19.28** Sejam  $X$  e  $Y$  dois problemas de decisão em NP. Suponha que  $X$  é NP-completo. Para mostrar que  $Y$  é NP-completo, devo mostrar que  $X$  é polinomialmente redutível a  $Y$  ou que  $Y$  é polinomialmente redutível a  $X$ ?

**Exr 19.29** Suponha que  $X$  e  $Y$  são dois problemas em NP. Considere a seguinte afirmação: “Se  $X$  pode ser polinomialmente reduzido a  $Y$  e  $Y$  está em P então  $X$  está em P.” A afirmação é verdadeira ou falsa?

**Exr 19.30** Suponha que  $X$  e  $Y$  são dois problemas em NP. Considere a seguinte afirmação: “Se  $X \leq_P Y$  então  $X$  está em P.” A afirmação é verdadeira ou falsa?

**Exr 19.31** Suponha que  $X$  e  $Y$  são problemas em NP e considere a seguinte afirmação: “Se  $X$  pode ser polinomialmente reduzido a  $Y$  e  $Y$  é NP-completo então  $X$  é NP-completo.” A afirmação é verdadeira ou falsa?

**Exr 19.32** Suponha que  $X$  e  $Y$  são problemas em NP e considere a seguinte afirmação: “Se  $X \leq_P Y$  e  $X$  é NP-completo então  $Y$  é NP-completo.” A afirmação é verdadeira ou falsa?

**Exr 19.33** Considere as seguintes afirmações: “todo problema NP-completo está em NP” e “todo problema em NP é NP-completo”. As afirmações são verdadeiras ou falsas?

**Exr 19.34** Considere a seguinte afirmação: “Há problemas em NP que não são NP-completos.” A afirmação é verdadeira ou falsa?

**Exr 19.35** Considere a seguinte afirmação: “ $P \cap NP \neq \emptyset$ .” A afirmação é verdadeira ou falsa?

**Exr 19.36** Considere a seguinte afirmação: “ $NP \subseteq P$ .” A afirmação é verdadeira ou falsa?

**Exr 19.37** Considere a seguinte afirmação: “Existem problemas NP-completos em P.” A afirmação é verdadeira ou falsa?

**Exr 19.38** Critique o seguinte texto extraído de um blog: “De forma bem resumida: Existem dois tipos de problemas: os triviais e os não triviais. Os problemas triviais são facilmente resolvidos, com operações matemáticas simples. Já os problemas não triviais só podem ser resolvidos com algoritmos. E é aí que mora o perigo. Existem os problemas não-triviais tratáveis. Esses problemas tem algoritmos do tipo polinomial (lembra da quinta série?). Aí é ‘tranquilo’. Qualquer Pentium 100 consegue liquidar em segundos. Esses algoritmos são tidos como de ‘classe P’. Alguns problemas não triviais não tem algoritmos de resolução conhecidos pelo homem. São chamados indecidíveis. Esses aí, a humanidade ainda vai levar alguns séculos pra resolver. Mas existem problemas intratáveis. São esses o xis da questão. Eles tem algoritmos conhecidos, mas os algoritmos não são polinomiais. Por isto, alguns deles são tão complexos que nosso poder computacional atual

*não consegue resolvê-los. Esses algoritmos são tidos como de ‘classe NP’. São esses problemas que demandam super-computadores e que levam horas, dias, anos, séculos para serem resolvidos por uma máquina. Por isso, os matemáticos estão sempre quebrando a cabeça pra transformar os algoritmos não-polinomiais em polinomiais. E eles tem tido sucesso em vários deles. Até que um dia alguém perguntou: ‘peraí! Será que eu consigo transformar QUALQUER algoritmo não-polinomial e um algoritmo polinomial?’ Em boa matemática: será que  $P=NP$ ? Se for, qualquer problema intratável passa a ser tratável.’*

# Bibliografia

- [AHU74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974. 1
- [AHU87] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1987. 1
- [AU95] A.V. Aho and J.D. Ullman. *Foundations of Computer Science (C edition)*. Computer Science Press, 1995. 1, 10
- [BB96] G. Brassard and P. Bratley. *Fundamentals of Algorithmics*. Prentice Hall, 1996. 1
- [Ben88] J.L. Bentley. *More Programming Pearls: Confessions of a Coder*. Addison-Wesley, 1988. 1
- [Ben00] J.L. Bentley. *Programming Pearls*. ACM Press, second edition, 2000. 1
- [CLR91] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 1991. 1, 41
- [CLRS01] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, second edition, 2001. 1, 41, 68, 85
- [KT05] J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley, 2005. 1
- [Man89] U. Manber. *Introduction to Algorithms: A Creative Approach*. Addison-Wesley, 1989. 1
- [Par95] I. Parberry. *Problems on Algorithms*. Prentice Hall, 1995. 1, 59, 70
- [Pro] Programming Challenges. Internet: <http://www.programming-challenges.com/>. 1, 65, 66, 68
- [Sed98] R. Sedgewick. *Algorithms in C, Parts 1–4*. Addison-Wesley Longman, third edition, 1998. 1, 10
- [SR03] S.S. Skiena and M.A. Revilla. *Programming Challenges (The Programming Contest Training Manual)*. Springer, 2003. Internet: <http://www.programming-challenges.com/>. 1

# Índice Remissivo

- ACERTA-DESCENDO, 50
- ACERTA-DESCENDO2, 50
- ACERTA-DESCENDO3, 51
- altura, 48
- amortizado, 76, 83
- análise amortizada, 76, 83
- árvore, 88
  - balanceada, 56
  - de busca, 56
  - de decisão, 103
  - geradora, 90
  - geradora mínima, 90
- árvore binária
  - cheia, 55
- balsa, 68
- bin packing*, 75
- Bonnie & Clyde, 107
- busca binária, 39
  - dinâmica, 78
- caminho hamiltoniano, 105
- ceiling*, 8
- certificado, 105
- ciclo hamiltoniano, 105
- classe
  - co-NP, 105
  - NP, 105
  - P, 105
- CLIQUE, 106
- clique, 106
- co-NP, 105
- cobertura
  - de vértices, 106
  - por cliques, 73
  - por subsequências, 65
- conjunto independente, 106
- counting sort, 101
- crescente, 36
- custo amortizado, 76, 83
- decrecente, 36
- dicionário, 56
- estável, 102
- estritamente
  - crescente, 36
  - decrescente, 36
- FIXDOWN, 50
- FIXUP, 49
- floor*, 8
- HAM-CYCLE, 105
- HAM-PATH, 105
- hamiltoniano
  - caminho, 105
  - ciclo, 105
- heap, 48
- Heapsort, 50
- HEAPSORT, 50
- Huffman, 73
- INDEPENDENT-SET, 106
- intercalação
  - de vetores, 40
  - múltipla de vetores, 37
- inversa composicional, 7
- KRSK, 91
- LCS, 63
- longest common subsequence, 63
- majoritário, 37
- mediana, 45
- MST, 90
- MSTKRUSKAL, 91
- MSTPRIM, 94
- nível, 48
- nó interno

de árvore binária, 55  
NP, 105  
NP-completo, 105  
  
ordenação  
estável, 102  
  
P, 105  
pares de livros, 75  
partição  
de um conjunto, 62, 87  
PENEIRA, 50  
piso, 8  
polinomialmente redutível, 105  
problema  
de decisão, 105  
NP-completo, 105  
profundidade, 48  
  
radix sort, 101  
RAIZQ, 36  
RASCUNHO-1, 92  
RASCUNHO-2, 93  
RASCUNHO-3, 93  
RASCUNHO-REC, 91  
recorrência de Karatsuba, 22, 26  
redução polinomial, 105  
redutível, 105  
  
SEPARÉH, 43  
SEPARÉL, 41  
sequênciia, 63  
SET-PARTITION, 107  
SETTRIPARTITIONPROB, 107  
subsequênciia, 63  
comum máxima, 63  
crescente máxima, 63  
SUBSET-SUM, 105  
  
TALVEZ-MST-A, 97  
teorema mestre, 26, 31  
teto, 8  
  
VERTEX-COVER, 106  
vetor arrumado, 42