

Recursividade, Relações de Recorrência e Análise de Algoritmos

OBJETIVOS DO CAPÍTULO

Depois de estudar este capítulo, você será capaz de:

- Compreender definições recorrentes de sequências, coleções de objetos e operações sobre objetos.
- Escrever definições recorrentes para determinadas sequências, coleções de objetos e operações sobre objetos.
- Compreender como os algoritmos recursivos funcionam.
- Escrever algoritmos recursivos para gerar sequências definidas recorrentemente.
- Encontrar soluções em forma fechada para determinadas relações de recorrência.
- Analisar algoritmos contando o número de operações de uma unidade básica de trabalho, diretamente ou resolvendo uma relação de recorrência.

Você está servindo no Conselho Municipal de Obras, que está considerando a proposta de uma firma para gerenciar um local de eliminação de resíduos químicos. O material a ser estocado no local decai a uma taxa de 5% ao ano. O empreiteiro afirma que, a essa taxa de estabilização, restará ao final de 20 anos apenas aproximadamente um terço do material ativo original.

Pergunta: A estimativa do empreiteiro está correta?

É possível verificar essa estimativa por meio de cálculos extensos: se existir certa quantidade inicialmente, então sobrarão um tanto no próximo, outro tanto no ano seguinte e assim por diante, até os 20 anos. Mas pode-se obter uma solução rápida e elegante resolvendo uma relação de recorrência; as relações de recorrência são discutidas na Seção 3.2.

A Seção 3.1 explora recorrência, que está intimamente ligada à indução matemática (discutida no capítulo anterior) e é importante para expressar muitas definições e até algoritmos. Algumas sequências definidas recorrentemente também podem ser definidas por uma fórmula. Encontrar tal fórmula envolve a resolução de uma relação de recorrência; métodos de solução para diversos tipos de relações de recorrência são desenvolvidos na Seção 3.2. Relações de recorrência formam uma ferramenta importante na análise de algoritmos, que determina matematicamente a quantidade de trabalho que determinado algoritmo tem que fazer. A análise de algoritmos é o tópico da Seção 3.3.

SEÇÃO 3.1

DEFINIÇÕES RECORRENTES

Uma definição na qual o item que está sendo definido aparece como parte da definição é chamada de uma **definição recorrente** ou **definição por recorrência** ou ainda **definição recursiva**. A princípio isso não parece fazer sentido — como podemos definir algo em termos de si mesmo? Isso funciona porque uma definição recorrente tem duas partes:

1. Uma base, ou condição básica, em que alguns casos simples do item que está sendo definido são dados explicitamente.
2. Um passo indutivo ou de recorrência, em que novos casos do item que está sendo definido são dados em função de casos anteriores.

A Parte 1 nos dá um lugar para começar, fornecendo alguns casos simples e concretos; a Parte 2 permite a construção de novos casos, a partir desses simples, e depois a construção de ainda outros casos a partir desses novos, e assim por

diante. (Isso parece análogo às demonstrações por indução matemática. Em uma demonstração por indução existe uma base da indução, a saber, mostrar que $P(1)$ — ou P em algum outro valor inicial — é verdadeira, e existe um passo indutivo, em que a veracidade de $P(k + 1)$ é estabelecida a partir da veracidade de P em valores anteriores. Essa semelhança é uma razão para o termo **definição por indução** ser usado algumas vezes no lugar de definição recorrente.)

Recorrência (ou recursividade) é uma ideia importante que pode ser usada para definir sequências de objetos, coleções mais gerais de objetos e operações com objetos. (O predicado Prolog *na cadeia alimentar* da Seção 1.5 foi definido de forma recorrente.) Até algoritmos podem ser recorrentes.

Sequências Definidas por Recorrência

Uma **sequência** S (uma **sequência infinita**) é uma lista de objetos que são numerados em determinada ordem; existem um primeiro objeto, um segundo e assim por diante. $S(k)$ denota o k -ésimo objeto na sequência. A lista não termina, de modo que uma sequência consiste em

$$S(1), S(2), \dots, S(k), \dots$$

Muitas vezes são usados índices para denotar os elementos de uma sequência, como

$$S_1, S_2, \dots, S_k, \dots$$

A letra S é uma “variável muda”, de modo que a sequência também poderia ser representada como

$$a_1, a_2, \dots, a_k, \dots \quad \text{ou} \quad w_1, w_2, \dots, w_k, \dots$$

e assim por diante.¹

Uma sequência é definida por recorrência nomeando-se, explicitamente, o primeiro valor (ou alguns poucos primeiros valores) na sequência e depois definindo valores subsequentes na sequência em termos de valores anteriores.

EXEMPLO 1

Considere a sequência S definida por recorrência por

- 1. $S(1) = 2$
- 2. $S(n) = 2S(n - 1)$ para $n \geq 2$

Pela proposição 1, $S(1)$, o primeiro objeto em S , é 2. Depois, pela proposição 2, o segundo objeto em S é $S(2) = 2S(1) = 2(2) = 4$. Novamente pela proposição 2, $S(3) = 2S(2) = 2(4) = 8$. Continuando desse modo, vemos que a sequência S é

$$2, 4, 8, 16, 32, \dots$$

Uma regra como a da proposição 2 no Exemplo 1, que define um valor de uma sequência em termos de um ou mais valores anteriores, é chamada de uma **relação de recorrência**.

PROBLEMA PRÁTICO 1

A sequência T é definida por recorrência por:

- 1. $T(1) = 1$
 - 2. $T(n) = T(n - 1) + 3$ para $n \geq 2$
- Escreva os cinco primeiros valores da sequência T . ■

EXEMPLO 2

A **sequência de Fibonacci**, introduzida no século XIII por um comerciante e matemático italiano, é definida por recorrência por

$$\begin{aligned} F(1) &= 1 \\ F(2) &= 1 \end{aligned}$$

$$F(n) = F(n-2) + F(n-1) \text{ para } n > 2$$

Aqui são dados os dois primeiros valores da sequência, e a relação de recorrência define o n -ésimo valor para $n > 2$ em termos dos dois valores precedentes. É melhor pensar na relação de recorrência em sua forma mais geral, que diz que F em qualquer valor — exceto em 1 e 2 — é a soma de F em seus dois valores anteriores. ■

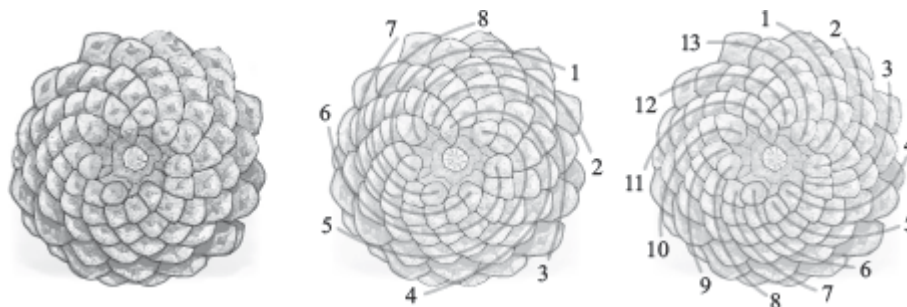
PROBLEMA PRÁTICO 2

Escreva os oito primeiros valores da sequência de Fibonacci. ■

A sequência de Fibonacci é famosa por causa de suas propriedades interessantes. Eis uma lista curta (sem demonstrações):

- Todo inteiro positivo pode ser escrito de maneira única como a soma de um ou mais números de Fibonacci distintos não consecutivos. Por exemplo, $11 = 3 + 8$, em que $3 = F(4)$ e $8 = F(6)$.
- $\text{mdc}(F(p), F(q)) = F(\text{mdc}(p, q))$. Por exemplo, se $p = 6$ e $q = 9$, então $F(6) = 8$, $F(9) = 34$ e $\text{mdc}(8, 34) = 2$. Por outro lado, $\text{mdc}(6, 9) = 3$ e $F(3) = 2$.
- Dois números de Fibonacci consecutivos são primos entre si, ou seja, o máximo divisor comum entre eles é 1. Em consequência, o algoritmo de Euclides executa a quantidade máxima de operações para encontrar o $\text{mdc}(a, b)$ quando a e b são dois números de Fibonacci consecutivos.

Outras propriedades matemáticas da sequência de Fibonacci são dadas no Exemplo 3 e nos exercícios ao final desta seção. Mas não são só matemáticos que se interessam pela sequência de Fibonacci. Os números de Fibonacci ocorrem com frequência na natureza. Muitas vezes, o número de pétalas em uma margarida é um número de Fibonacci. Olhando uma pinha, as sementes parecem estar arrumadas em espirais no sentido horário e no sentido anti-horário. Contando o número de cada tipo de espiral, frequentemente se obtêm dois números de Fibonacci consecutivos (8 e 13 aqui). O mesmo ocorre nas sementes de flores como o girassol ou nas espirais nos abacaxis.



E em arte e arquitetura, considera-se que a razão áurea cria proporções esteticamente agradáveis. A *razão áurea* é

$$\frac{1 + \sqrt{5}}{2} \approx 1,6180339$$

e seu valor pode ser aproximado pela razão entre dois números de Fibonacci consecutivos $F(n+1)/F(n)$, com precisão melhor para valores cada vez maiores de n .

EXEMPLO 3

Prove que, na sequência de Fibonacci,

$$F(n+4) = 3F(n+2) - F(n) \text{ para todo } n \geq 1$$

Como queremos provar que alguma coisa é verdadeira para todo $n \geq 1$, é natural pensar em uma demonstração por indução. E como o valor de $F(n)$ depende de $F(n-1)$ e de $F(n-2)$, devemos usar o segundo princípio de indução. Para a base da indução, vamos provar dois casos, $n = 1$ e $n = 2$. Para $n = 1$, obtemos

$$F(5) = 3F(3) - F(1)$$

ou (usando os valores calculados no Problema Prático 2)

$$5 = 3(2) - 1$$

que é verdade. Para $n = 2$,

$$F(6) = 3F(4) - F(2)$$

ou

$$8 = 3(3) - 1$$

que também é verdade. Suponha que, para todo r , $1 \leq r \leq k$,

$$F(r + 4) = 3F(r + 2) - F(r).$$

Vamos mostrar o caso $k + 1$, em que $k + 1 \geq 3$. (Já provamos os casos $n = 1$ e $n = 2$.) Queremos mostrar, então, que

$$F(k + 1 + 4) \stackrel{?}{=} 3F(k + 1 + 2) - F(k + 1)$$

ou

$$F(k + 5) \stackrel{?}{=} 3F(k + 3) - F(k + 1)$$

Da relação de recorrência para a sequência de Fibonacci, temos

$$F(k + 5) = F(k + 3) + F(k + 4)$$

(F em qualquer valor é a soma de F nos dois valores anteriores)

e, pela hipótese de indução, com $r = k - 1$ e $r = k$, respectivamente, temos

$$F(k + 3) = 3F(k + 1) - F(k - 1)$$

e

$$F(k + 4) = 3F(k + 2) - F(k)$$

Portanto,

$$F(k + 5) = F(k + 3) + F(k + 4)$$

$$= [3F(k + 1) - F(k - 1)] + [3F(k + 2) - F(k)]$$

$$= 3[F(k + 1) + F(k + 2)] - [F(k - 1) + F(k)]$$

$$= 3F(k + 3) - F(k + 1) \quad (\text{usando novamente a relação de recorrência})$$

Isso completa a demonstração por indução. 

PROBLEMA PRÁTICO 3

Na demonstração por indução do Exemplo 3, por que é necessário demonstrar o caso $n = 2$ como um caso particular? ■

EXEMPLO 4

A fórmula

$$F(n+4) = 3F(n+2) - F(n) \text{ para todo } n \geq 1$$

do Exemplo 3 também pode ser provada diretamente, sem indução, usando apenas a relação de recorrência na definição dos números de Fibonacci. A relação de recorrência

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

pode ser reescrita na forma

$$F(n+1) = F(n+1) - F(n) \quad (1)$$

Logo,

$$\begin{aligned} F(n+4) &= F(n+3) + F(n+2) \\ &= F(n+2) + F(n+1) + F(n+2) && \text{(reescrivendo } F(n+3) \text{)} \\ &= F(n+2) + [F(n+2) - F(n)] + F(n+2) && \text{(reescrivendo } F(n+1) \text{ usando (1))} \\ &= 3F(n+2) - F(n) \end{aligned}$$

Conjuntos Definidos por Recorrência

Os objetos em uma sequência são ordenados — existem um primeiro objeto, um segundo e assim por diante. Um conjunto de objetos é uma coleção na qual não há nenhuma ordem imposta. Alguns conjuntos podem ser definidos por recorrência.

EXEMPLO 5

Na Seção 1.1, notamos que certas cadeias de letras de proposição, conectivos lógicos e parênteses, tais como $(A \wedge B)' \vee C$, são consideradas legítimas, enquanto outras, como $\wedge \wedge A'' B$, não o são. A sintaxe para arrumar tais símbolos constitui a definição do conjunto de fórmulas proposicionais bem formuladas e é uma definição por recorrência.

1. Qualquer letra de proposição é uma fbf.
2. Se P e Q são fbfs, então $(P \wedge Q)$, $(P \vee Q)$, $(P \rightarrow Q)$, (P') , e $(P \leftrightarrow Q)$ também o são.²

Usando as prioridades para os conectivos lógicos estabelecidas na Seção 1.1, podemos omitir os parênteses quando isso não causar confusão. Assim, podemos escrever $(P \vee Q)$ como $P \vee Q$, ou (P') como P' ; as novas expressões, tecnicamente, não são fbfs pela definição que acabamos de dar, mas representam, sem ambiguidades, fbfs.

Começando com letras de proposição e usando, repetidamente, a regra 2, podemos construir todas as fbfs proposicionais. Por exemplo, A , B e C são fbfs pela regra 1. Pela regra 2,

$$(A \wedge B) \text{ e } (C)$$

são, ambas, fbfs. Novamente pela regra 2,

$$((A \wedge B) \rightarrow (C))$$

é uma fbf. Aplicando a regra 2 mais uma vez, obtemos a fbf

$$(((A \wedge B) \rightarrow (C))')$$

Eliminando alguns parênteses, podemos escrever essa fbf como

$$((A \wedge B) \rightarrow (C))'$$

PROBLEMA PRÁTICO 4

Mostre como construir a fbf $((A \vee (B')) \rightarrow C)$ da definição no Exemplo 5. ■

PROBLEMA PRÁTICO 5

Uma definição por recorrência para um conjunto de pessoas que são ancestrais de João poderia ter a seguinte base:

Os pais de João são seus ancestrais.

Dê o passo indutivo. ■

Cadeias de símbolos retiradas de um “alfabeto” finito são objetos encontrados com frequência em ciência da computação. Computadores guardam os dados como **cadeias binárias**, cadeias do alfabeto que consiste apenas em 0 e 1; compiladores veem proposições ou comandos em programas como cadeias de *marcas* ou *sinais*,[†] tais como palavras-chave e identificadores. A coleção de todas as cadeias de comprimento finito formada por símbolos de um alfabeto, chamadas de cadeias *de* um alfabeto, podem ser definidas de forma recorrente (veja o Exemplo 6). Muitos conjuntos de cadeias com propriedades particulares também têm definições recursivas.

EXEMPLO 6

O conjunto de todas as cadeias (de comprimento finito) de símbolos de um alfabeto A é denotado por A^* . A definição recorrente de A^* é

1. A **cadeia vazia** λ (a cadeia sem nenhum símbolo) pertence a A^* .
2. Um único elemento qualquer de A pertence a A^* .
3. Se x e y são cadeias em A^* , então a **concatenação** xy de x e y também pertence a A^* .

As partes 1 e 2 constituem a base, e a parte 3 é o passo indutivo dessa definição. Note que, para qualquer cadeia x , $x\lambda = \lambda x = x$. ■

PROBLEMA PRÁTICO 6

Se $x = 1011$ e $y = 001$, escreva as cadeias xy , yx e $yx\lambda x$. ■

PROBLEMA PRÁTICO 7

Dê uma definição recorrente para o conjunto de todas as cadeias binárias que são **palíndromos**, cadeias que são iguais se lidas normalmente ou de trás para a frente. ■

EXEMPLO 7

Suponha que, em determinada linguagem de programação, os identificadores podem ser cadeias alfanuméricas de comprimento arbitrário, mas têm que começar com uma letra. Uma definição recorrente para o conjunto dessas cadeias é

1. Uma única letra é um identificador.
2. Se A for um identificador, a concatenação de A e qualquer letra ou dígito também o será.

Uma notação mais simbólica para descrever conjuntos de cadeias definidas por recorrência é chamada de **forma de Backus Naur**, ou **FBN**, desenvolvida originalmente para definir a linguagem de programação ALGOL. Em notação FBN, os itens que são definidos em termos de outros itens são envolvidos pelos símbolos de menor e maior, enquanto itens específicos que não podem ser divididos não aparecem dessa forma. Um segmento vertical $|$ denota uma escolha e tem o mesmo significado que a palavra *ou*. A definição em FBN de um identificador é

```
<identificador> ::= <letra> | <identificador> <letra> | <identificador> <dígito>
<letra> ::= a | b | c | ... | z
<dígito> ::= 1 | 2 | ... | 9
```

Assim, o identificador `me2` pode ser obtido da definição por uma sequência de escolhas como

<identificador>	pode ser	<identificador> <dígito>
	que pode ser	<identificador>2
	que pode ser	<identificador> <letra>2
	que pode ser	<identificador>e2
	que pode ser	<letra>e2
	que pode ser	me2

Como outro exemplo de ligação entre recorrência e indução, existe uma forma de indução chamada de **indução estrutural** que pode ser aplicada a conjuntos definidos por recorrência. Suponha que S é um conjunto definido por recorrência e suponha que existe uma propriedade $P(x)$ que pode ser válida ou não para um elemento x de S . Se pudermos provar que:

1. A propriedade P é válida para todos os elementos descritos na base.
2. Se a propriedade P for válida para alguns elementos de S , então será válida para todos os elementos novos de S construídos desses elementos usando o passo indutivo.

então a propriedade P será válida para todos os elementos de S .

EXEMPLO 8

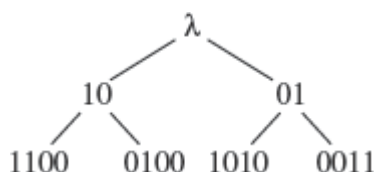
Um conjunto S de cadeias é definido recursivamente por

1. λ pertence a S .
2. Se x pertencer a S , então $1x0$ e $0x1$ também pertencerão a S .

Podemos usar a indução estrutural para provar que toda cadeia em S tem o mesmo número de zeros e de uns. A base, Regra 1, identifica uma única cadeia em S , a cadeia vazia λ , que tem o mesmo número de zeros e de uns (0 zeros e 0 uns). Suponha que a cadeia x tem o mesmo número de zeros e de uns. Usando a Regra 2, as duas cadeias novas que podem ser construídas a partir de x adicionam um único 0 e um único 1 a x , de modo que o número de zeros e o número de uns foram aumentados de 1, que assim os números de zeros e de uns continuam iguais. Pela indução estrutural, toda cadeia em S tem o mesmo número de zeros e de uns.

Note que nem todas as cadeias contendo o mesmo número de zeros e de uns pertencem a S . Por exemplo, não é possível gerar a cadeia 1001 usando as regras dadas.

A indução matemática usual prova propriedades sobre valores inteiros positivos e os inteiros positivos são ordenados: $1, 2, \dots, k, k+1, \dots$. Um conjunto, no entanto, não precisa ser ordenado. Se considerarmos o conjunto S definido no Exemplo 8, ele parece com



e a indução estrutural nos ajuda a tratar essa “disseminação” de valores no conjunto.

Operações Definidas por Recorrência

Certas operações em objetos podem ser definidas de forma recorrente, como nos Exemplos 9 e 10.

EXEMPLO 9

Uma definição recorrente da operação de potenciação a^n de um número real não nulo a , em que n é um inteiro não negativo, é

1. $a_0 = 1$
2. $a_n = (a^{n-1})a$ para $n \geq 1$

EXEMPLO 10

Uma definição recorrente para a multiplicação de dois inteiros positivos m e n .

1. $m(1) = m$
2. $m(n) = m(n-1) + m$ para $n \geq 2$

PROBLEMA PRÁTICO 8

Seja x uma cadeia de determinado alfabeto. Dê uma definição recorrente para a operação x^n (concatenação de x consigo mesmo n vezes) para $n \geq 1$. ■

Na Seção 1.1 definimos a operação de disjunção lógica de duas letras de proposição. Isso pode servir como base para uma definição recorrente da disjunção de n letras de proposição, $n \geq 2$:

1. $A_1 \vee A_2$ definido como na Seção 1.1
2. $A_1 \vee \cdots \vee A_n = (A_1 \vee \cdots \vee A_{n-1}) \vee A_n$ para $n > 2$ (2)

Usando essa definição, podemos generalizar a associatividade da disjunção (equivalência tautológica 2a) dizendo que, em uma disjunção de n letras de proposição, o agrupamento entre parênteses é desnecessário porque todos esses agrupamentos são equivalentes à expressão geral para a disjunção de n letras de proposição. Em forma simbólica, para qualquer n com $n \geq 3$ e qualquer p com $1 \leq p \leq n-1$,

$$(A_1 \vee \cdots \vee A_p) \vee (A_{p+1} \vee \cdots \vee A_n) \Leftrightarrow A_1 \vee \cdots \vee A_n$$

Essa equivalência pode ser demonstrada por indução em n . Para $n = 3$,

$$\begin{aligned} A_1 \vee (A_2 \vee A_3) &\Leftrightarrow (A_1 \vee A_2) \vee A_3 && \text{(pela equivalência 2a)} \\ &= A_1 \vee A_2 \vee A_3 && \text{(pela Equação (2))} \end{aligned}$$

Suponha que, para $n = k$ e $1 \leq p \leq k-1$,

$$A_1 \vee \cdots \vee A_p) \vee (A_{p+1} \vee \cdots \vee A_k) \Leftrightarrow A_1 \vee \cdots \vee A_k$$

Então, para $n = k+1$ e $1 \leq p \leq k$,

$$\begin{aligned} &(A_1 \vee \cdots \vee A_p) \vee (A_{p+1} \vee \cdots \vee A_{k+1}) \\ &= (A_1 \vee \cdots \vee A_p) \vee [(A_{p+1} \vee \cdots \vee A_k) \vee A_{k+1}] && \text{(pela Equação (2))} \\ &\Leftrightarrow [(A_1 \vee \cdots \vee A_p) \vee (A_{p+1} \vee \cdots \vee A_k)] \vee A_{k+1} && \text{(pela equivalência 2a)} \\ &\Leftrightarrow (A_1 \vee \cdots \vee A_k) \vee A_{k+1} && \text{(pela hipótese de indução)} \\ &= A_1 \vee \cdots \vee A_{k+1} && \text{(pela Equação (2))} \end{aligned}$$

Algoritmos Definidos por Recorrência

O Exemplo 1 dá uma definição recorrente para uma sequência S . Suponha que queremos escrever um programa de computador para calcular $S(n)$ para algum inteiro positivo n . Temos duas abordagens possíveis. Se quisermos encontrar $S(12)$, por exemplo, podemos começar com $S(1) = 2$ e depois calcular $S(2)$, $S(3)$, e assim por diante, como fizemos no Exemplo 1, até chegar, finalmente, em $S(12)$. Sem dúvida, essa abordagem envolve iteração em alguma espécie de laço. A seguir, vamos dar uma função S em pseudocódigo que usa esse algoritmo iterativo. A base, com $n = 1$, é obtida na primeira cláusula do comando **se**; o valor 2 é retornado. A cláusula **senão**, para $n > 1$, tem uma atribuição inicial e entra em um laço **enquanto** que calcula valores maiores da sequência até atingir o limite superior correto. Você pode seguir a execução desse algoritmo para alguns valores de n para se convencer de que ele funciona.

ALGORITMO

```
S(inteiro positivo  $n$ )
//função que calcula iterativamente o valor  $S(n)$ 
//para a sequência  $S$  do Exemplo 1
Variáveis locais:
inteiro  $i$     //índice do laço
ValorAtual   //valor atual da função  $S$ 

    se  $n = 1$  então
        retorne 2
    senão
         $i = 2$ 
         $ValorAtual = 2$ 
        enquanto  $i \leq n$  faça
             $ValorAtual = 2 * ValorAtual$ 
             $i = i + 1$ 
        fim do enquanto

    //agora  $ValorAtual$  tem o valor  $S(n)$ 
    retorne  $ValorAtual$ 
fim do se
fim da função  $S$ 
```

A segunda abordagem para calcular $S(n)$ usa diretamente a definição recorrente de S . A versão a seguir é de um *algoritmo recursivo*, escrito, novamente, como uma função em pseudocódigo.

ALGORITMO

```
S(inteiro positivo  $n$ )
//função que calcula o valor  $S(n)$  de forma recorrente
//para a sequência  $S$  do Exemplo 1

    se  $n = 1$  então
        retorne 2
    senão
        retorne  $2 * S(n - 1)$ 
    fim do se
fim da função  $S$ 
```

O corpo dessa função consiste em uma única proposição do tipo **se-então-senão**. Para compreender como essa função funciona, vamos seguir a execução para calcular o valor de $S(3)$. Chamamos primeiro a função com um valor de entrada $n = 3$. Como n não é 1, a execução é direcionada para a cláusula **senão**. Nesse instante, a atividade de calcular $S(3)$ tem que ser suspensa até se conhecer o valor de $S(2)$. Qualquer informação conhecida, relevante para o cálculo de $S(3)$, é armazenada na memória do computador em uma pilha, que será recuperada quando o cálculo for completado. (Uma pilha é uma coleção de dados tal que qualquer novo item vai para o topo da pilha e, em qualquer instante, apenas o item no topo é acessível ou pode ser removido da pilha. Portanto, uma pilha é uma estrutura LIFO — do inglês *last in, first out*, ou seja, o último a entrar é o primeiro a sair.) A função é chamada novamente com um valor de entrada $n = 2$. Mais uma vez, a cláusula **senão** é executada e o cálculo de $S(2)$ é suspenso, com as informações relevantes armazenadas na pilha, enquanto a função é chamada novamente com $n = 1$ como valor de entrada.

Dessa vez é executada a primeira cláusula da proposição **se**, e o valor da função, 2, pode ser calculado diretamente. Essa chamada final da função está completa, e o valor 2 é usado na penúltima chamada da função, que remove agora da pilha qualquer informação relevante ao caso $n = 2$, calcula $S(2)$ e usa esse valor na invocação prévia (inicial) da função. Finalmente, essa chamada original de S é capaz de esvaziar a pilha e completar seu cálculo, retornando o valor de $S(3)$.

Quais são as vantagens relativas dos dois algoritmos, o iterativo e o recursivo, ao executar a mesma tarefa? Nesse exemplo, a versão recursiva é certamente mais curta, já que não precisa gerenciar um cálculo em laço. A descrição da execução do algoritmo recursivo parece soar mais complicada do que a do algoritmo iterativo, mas todos os passos são executados automaticamente. Não precisamos estar conscientes do que está acontecendo internamente, exceto para observar que uma série longa de chamadas recursivas pode usar muita memória ao armazenar na pilha as informações relevantes para as invocações prévias. Se a utilização da memória for excessiva, pode acontecer um “transbordamento” (*overflow*) da pilha. Além de usar mais memória, algoritmos recursivos podem necessitar de mais cálculos e executar mais lentamente do que os não recursivos (veja o Exercício 3 na seção No Computador, ao final deste capítulo).

Apesar disso, a recorrência (ou recursividade) fornece um modo natural de pensar em muitas situações, algumas das quais necessitariam de soluções não recursivas muito complexas. Uma solução recursiva é bem adequada para o problema de calcular os valores de uma sequência definida de maneira recorrente. Muitas linguagens de programação aceitam recursividade.

PROBLEMA PRÁTICO 9

Escreva o corpo de uma função recursiva para calcular $T(n)$ para a sequência T definida no Problema Prático 1. ■

EXEMPLO 11

No Exemplo 10, foi dada uma definição recorrente para a multiplicação de dois inteiros positivos m e n . A seguir temos uma função em pseudocódigo para a multiplicação baseada nessa definição. ■

ALGORITMO

```
Produto(inteiro positivo  $m$ ; inteiro positivo  $n$ )
//Função que calcula de forma recursiva o produto de  $m$  e  $n$ 

se  $n = 1$  então
    retorne  $m$ ;
senão
    retorne  $\text{Produto}(m, n - 1) + m$ 
fim do se
fim da função Produto
```

Um algoritmo recursivo chama a si mesmo com valores de entrada “menores”. Suponha que um problema pode ser resolvido encontrando-se soluções para as versões menores do mesmo problema e que as versões menores acabam se

tornando casos triviais, facilmente solucionados. Então um algoritmo recursivo pode ser útil, mesmo que o problema original não tenha sido enunciado de forma recorrente.

Para nos convencer de que determinado algoritmo recursivo funciona, não precisamos começar com um dado particular de entrada, ir diminuindo, tratando de casos cada vez menores, e depois ir voltando, percorrendo todo o caminho inverso. Fizemos isso ao discutir o cálculo de $S(3)$, mas foi só para ilustrar a mecânica de um cálculo recursivo. Em vez disso, podemos verificar o caso trivial (como demonstrar a base em uma demonstração por indução) e verificar que, se o algoritmo funciona corretamente ao ser chamado com valores de entrada menores, então ele resolve, de fato, o problema para o valor de entrada original (o que é semelhante a provar $P(k + 1)$ da hipótese $P(k)$ em uma demonstração por indução).

LEMBRETE

Pense em um algoritmo recursivo sempre que você puder resolver o problema a partir de soluções de versões menores do problema.

EXEMPLO 12

Uma das tarefas mais comuns em processamento de dados é colocar uma lista L de n itens em ordem numérica ou alfabética, crescente ou decrescente. (Essa lista pode conter nomes de clientes, por exemplo, e, em ordem alfabética, “Vargas, Joana” deve vir depois de “Teixeira, José”.) O algoritmo de **ordenação por seleção** — um algoritmo simples, mas particularmente eficaz — é descrito em pseudocódigo adiante.

Essa função ordena os j primeiros itens em L em ordem crescente; quando a função é chamada pela primeira vez, j tem o valor n (de modo que a primeira chamada acaba ordenando toda a lista). A parte recursiva do algoritmo está dentro da cláusula **senão**; o algoritmo examina a seção da lista sob consideração e encontra o valor de i para o qual $L[i]$ tem o valor máximo. Ele então permuta $L[i]$ e $L[j]$ e, depois disso, o valor máximo ocorre na posição j , a última posição na parte da lista que está sendo considerada. $L[j]$ está correto agora e não deve mais ser modificado, de modo que esse processo é repetido na lista de $L[1]$ a $L[j - 1]$. Se essa parte da lista for ordenada corretamente, então a lista inteira será ordenada corretamente. Quando j tiver o valor 1, a parte da lista que está sendo considerada terá apenas um elemento, que tem que estar no lugar certo. Nesse instante a lista toda estará ordenada. ■

ALGORITMO	ORDENAÇÃO POR SELEÇÃO
	<pre>OrdenaçãoPorSeleção(lista L; inteiro positivo j) //algoritmo recursivo para ordenar os itens de 1 a j em uma lista L em ordem //crescente se j = 1 então a ordenação está completa, escreva a lista ordenada senão encontre o índice i do maior item em L entre 1 e j permuta L[i] e L[j] OrdenaçãoPorSeleção(L, j - 1) fim do se fim da função OrdenaçãoPorSeleção</pre>

Outros algoritmos recursivos de ordenação estão discutidos nos exercícios da Seção 3.3.

EXEMPLO 13

Agora que ordenamos nossa lista, outra tarefa comum é procurar um item particular na lista. (Joana Vargas já é uma cliente?) Uma técnica eficiente de busca em uma lista ordenada é o **algoritmo de busca binária**, um algoritmo recursivo descrito em pseudocódigo a seguir.

ALGORITMO BUSCABINÁRIA

	<pre>BuscaBinária(lista L; inteiro positivo i; inteiro positivo j; tipo item x) //procura na lista ordenada L, de L[i] a L[j], pelo item x</pre>
--	--

```

se  $i > j$  então
  escreva("não encontrado")
senão
  encontre o índice  $k$  do item do meio entre  $i$  e  $j$ 
  se  $x = \text{item do meio } L[k]$  então
    escreva("encontrado")
  senão
    se  $x < \text{item do meio } L[k]$  então
      BuscaBinária( $L, i, k - 1, x$ )
    senão
      BuscaBinária( $L, k + 1, j, x$ )
  fim do se
fim do se
fim do se
fim da função BuscaBinária

```

Esse algoritmo procura na seção da lista entre $L[i]$ e $L[j]$ por um item x ; inicialmente, i e j têm os valores 1 e n , respectivamente. A primeira cláusula do primeiro **se** é o passo básico que diz que x não pode ser encontrado em uma lista vazia, uma em que o primeiro índice é maior do que o último. Na cláusula **senão** principal, o item do meio em uma seção da lista tem que ser encontrado. (Se a seção tiver um número ímpar de itens, existirá, de fato, um item do meio; se a seção contiver um número par de itens, bastará escolher como “item do meio” o que fica no final da primeira metade da seção da lista.) Comparando x com o item do meio, localizamos a metade da lista onde devemos procurar a seguir, a metade antes do ponto do meio ou a metade depois do ponto do meio. ■

EXEMPLO 14

Vamos aplicar o algoritmo de busca binária à lista

3, 7, 8, 10, 14, 18, 22, 34

onde o item x a ser encontrado é o número 25. A lista inicial não é vazia, logo o item do meio é localizado e encontra-se o valor 10. Então x é comparado com o item do meio. Como $x > 10$, a busca é feita na segunda metade da lista, a saber, entre os itens

14, 18, 22, 34

Novamente, essa lista não é vazia, e o item do meio é 18. Como $x > 18$, procura-se na segunda metade da lista, ou seja, entre os itens

22, 34

Nessa lista não vazia, o item do meio é 22. Como $x > 22$, a busca continua na segunda metade da lista, a saber,

34

Essa é uma lista de apenas um elemento, com o item do meio sendo esse único elemento. Como $x < 34$, começa uma busca na “primeira metade” da lista; mas a primeira metade é vazia. O algoritmo termina aqui com a informação de que x não está na lista. Essa execução necessita de quatro comparações ao todo; x é comparado com 10, 18, 22 e 34. ■

PROBLEMA PRÁTICO 10	Em uma busca binária da lista no Exemplo 14, nomeie os elementos que serão comparados com x se x tiver o valor 8. ■
----------------------------	---

Vimos uma série de definições recorrentes. A Tabela 3.1 resume suas características.

TABELA 3.1

Definições por Recorrência	
O que Está Sendo Definido	Características
Sequência recorrente	O primeiro ou os dois primeiros valores da sequência são conhecidos; outros valores na sequência são definidos em termos de valores anteriores.
Conjunto recorrente	Alguns poucos elementos específicos do conjunto são conhecidos; outros elementos do conjunto são construídos a partir de combinações de elementos já pertencentes ao conjunto.
Operação recorrente	Um caso “pequeno” da operação fornece um valor específico; outros casos da operação são definidos em termos de casos menores.
Algoritmo recursivo	Para os valores menores dos argumentos, o comportamento do algoritmo é conhecido; para valores maiores dos argumentos, o algoritmo chama a si mesmo com valores menores dos argumentos.

SEÇÃO 3.1 REVISÃO

TÉCNICAS

- Geração de valores de uma sequência definida por recorrência.
- Demonstração de propriedades da sequência de Fibonacci.
- ❶ Reconhecimento de objetos em uma coleção definida por recorrência.
- Obtenção de definições por recorrência para conjuntos particulares de objetos.
- Obtenção de definições por recorrência para determinadas operações em objetos.
- Desenvolvimento de algoritmos recursivos para gerar sequências definidas por recorrência.

IDEIAS PRINCIPAIS

- Definições por recorrência podem ser dadas para sequências de objetos, para conjuntos de objetos e para operações em objetos, com informação básica conhecida e informações novas dependendo de informações já conhecidas.
- Algoritmos recursivos fornecem uma solução natural para determinados problemas invocando a mesma tarefa em versões menores do problema.

EXERCÍCIOS 3.1

Para os Exercícios 1 a 12, escreva os cinco primeiros valores da sequência.

1. $S(1) = 10$

$$S(n) = S(n - 1) + 10 \text{ para } n \geq 2$$

2. $C(1) = 5$

$$C(n) = 2C(n - 1) + 5 \text{ para } n \geq 2$$

3. $A(1) = 2$

$$A(n) = \frac{1}{A(n - 1)} \text{ para } n \geq 2$$

4. $B(1) = 1$

$$B(n) = B(n - 1) + n^2 \text{ para } n \geq 2$$

5. $S(1) = 1$

$$S(n) = S(n - 1) + \frac{1}{n} \text{ para } n \geq 2$$

6. $T(1) = 1$

$$T(n) = nT(n - 1) \text{ para } n \geq 2$$

7. $P(1) = 1$
 $P(n) = n^2 P(n-1) + (n-1)$ para $n \geq 2$
8. $A(1) = 2$
 $A(n) = nA(n-1) + n$ para $n \geq 2$
9. $M(1) = 2$
 $M(2) = 2$
 $M(n) = 2$
 $M(n-1) + M(n-2)$ para $n > 2$
10. $D(1) = 3$
 $D(2) = 5$
 $D(n) = (n-1)D(n-1) + (n-2)D(n-2)$ para $n > 2$
11. $W(1) = 2$
 $W(2) = 3$
 $W(n) = W(n-1)W(n-2)$ para $n > 2$
12. $T(1) = 1$
 $T(2) = 2$
 $T(3) = 3$
 $T(n) = T(n-1) + 2T(n-2) + 3T(n-3)$ para $n > 3$

Nos Exercícios 13 a 18, prove a propriedade dada dos números de Fibonacci diretamente da definição.

13. $F(n+1) + F(n-2) = 2F(n)$ para $n \geq 3$
14. $F(n) = 5F(n-4) + 3F(n-5)$ para $n \geq 6$
15. $F(n) = 3F(n-3) + 2F(n-4)$ para $n \geq 5$
16. $[F(n+1)]^2 = [F(n)]^2 + F(n-1)F(n+2)$ para $n \geq 2$
17. $F(n+3) = 2F(n+1) + F(n)$ para $n \geq 1$
18. $F(n+6) = 4F(n+3) + F(n)$ para $n \geq 1$

Nos Exercícios 19 a 22, prove a propriedade dada dos números de Fibonacci para todo $n \geq 1$. (*Sugestão*: O primeiro princípio de indução vai funcionar.)

19. $F(1) + F(2) + \cdots + F(n) = F(n+2) - 1$
20. $F(2) + F(4) + \cdots + F(2n) = F(2n+1) - 1$
21. $F(1) + F(3) + \cdots + F(2n-1) = F(2n)$
22. $[F(1)]^2 + [F(2)]^2 + \cdots + [F(n)]^2 = F(n)F(n+1)$

Nos Exercícios 23 a 26, prove a propriedade dada dos números de Fibonacci usando o segundo princípio de indução.

23. Exercício 17.
24. Exercício 18.
25. $F(n) < 2^n$ para $n \geq 1$
26. $F(n) > \left(\frac{3}{2}\right)^{n-1}$ para $n \geq 6$
27. Escreva um algoritmo recursivo em pseudocódigo para uma função que calcule $F(n)$, o n -ésimo número de Fibonacci.
28. Siga seu algoritmo do Exercício 27 para calcular $F(6)$.
 - a. Quantas vezes a função é chamada?
 - b. Quantas vezes $F(4)$ é calculado?
 - c. Quantas vezes $F(3)$ é calculado?
 - d. Quantas vezes $F(2)$ é calculado?

Os Exercícios 29 e 30 tratam de uma demonstração de correção do algoritmo iterativo a seguir para calcular $F(n)$, o n -ésimo número de Fibonacci.

```

F(inteiro positivo  $n$ )
//função que calcula de modo iterativo o valor do
//do  $n$ -ésimo número de Fibonacci
Variáveis locais:
inteiro positivo  $i$            //índice do laço
inteiros positivos  $p, q, r$     //termos na sequência de Fibonacci

se  $n = 1$  então
    retorne 1
senão
    se  $n = 2$  então
        retorne 1
    senão
         $i = 2$ 
         $p = 1$       // $p$  = primeiro dos termos anteriores na sequência de Fibonacci
         $q = 1$       // $q$  = segundo dos termos anteriores na sequência de Fibonacci
        enquanto  $i < n$  faça
             $r = p + q$  //forma o próximo termo como a
                        //soma dos dois termos anteriores
             $p = q$      //atualiza  $p$ 
             $q = r$      //atualiza  $q$ 
             $i = i + 1$ 
        fim do enquanto
        //agora  $q$  tem o valor de  $F(n)$ 
        retorne  $q$ 
    fim do se
fim do se
fim da função  $F$ 

```

29. a. No algoritmo iterativo de Fibonacci, a condição B para a continuação do laço é $i < n$, logo B' é $i \geq n$, mas qual é o valor exato de i quando o laço termina?
- b. Ao sair do laço, você quer que $q = F(n)$; o que você quer para o valor de p naquele instante?
30. a. Escreva o invariante de laço Q para o algoritmo iterativo de Fibonacci
- b. Prove que Q é um invariante do laço.
31. Os valores p e q são definidos da seguinte maneira:

$$p = \frac{1 + \sqrt{5}}{2} \quad \text{e} \quad q = \frac{1 - \sqrt{5}}{2}$$

- a. Prove que $1 + p = p^2$ e $1 + q = q^2$.
- b. Prove que

$$F(n) = \frac{p^n - q^n}{p - q}$$

- c. Use o item (b) para provar que

$$F(n) = \frac{\sqrt{5}}{5} \left(\frac{1 + \sqrt{5}}{2} \right)^n - \frac{\sqrt{5}}{5} \left(\frac{1 - \sqrt{5}}{2} \right)^n$$

é uma solução em forma fechada para a sequência de Fibonacci.

32. A sequência de Lucas é definida por

$$L(1) = 1$$

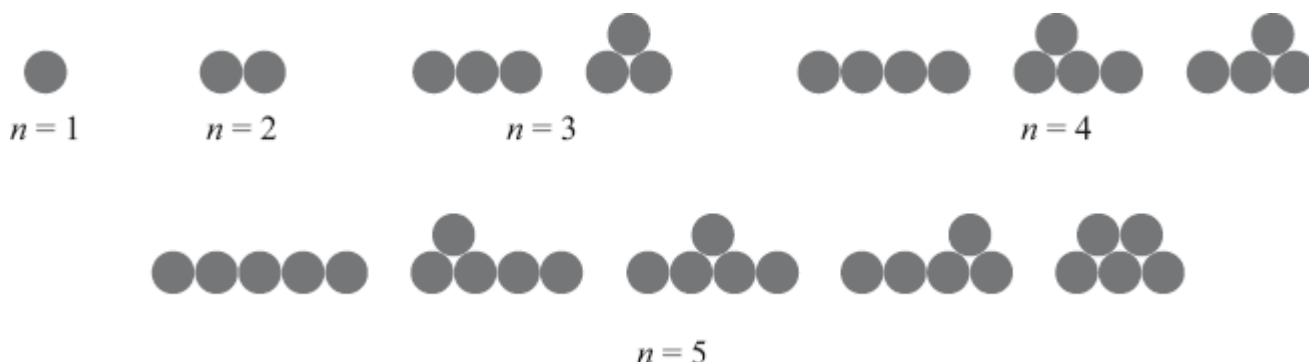
$$L(2) = 3$$

$$L(n) = L(n-1) + L(n-2) \text{ para } n \geq 2$$

- Escreva os cinco primeiros termos da sequência.
- Prove que $L(n) = F(n+1) + F(n-1)$ para $n \geq 2$, em que F é a sequência de Fibonacci.

Para os Exercícios 33 a 36, decida se as sequências descritas são subsequências da sequência de Fibonacci, ou seja, se seus elementos são alguns ou todos os elementos, na ordem correta, da sequência de Fibonacci.³

- A sequência $A(n)$, em que $A(n) = 1 +$ (a soma dos n primeiros elementos da sequência de Fibonacci), $n \geq 1$. Os quatro primeiros valores são 2, 3, 5, 8, os quais — até agora — formam uma subsequência da sequência de Fibonacci.
- A sequência $B(n)$, em que $B(n) = (n-1)2^{n-2} + 1$, $n \geq 1$. Os quatro primeiros valores são 1, 2, 5, 13, os quais — até agora — formam uma subsequência da sequência de Fibonacci.
- A sequência $C(n)$, em que $C(n)$ é o número de maneiras diferentes que podemos arrumar n moedas em fileiras horizontais, de modo que todas as moedas em cada fileira fiquem encostadas e toda moeda acima da fileira mais embaixo fique encostada em duas moedas na fileira logo abaixo, $n \geq 1$. Os cinco primeiros valores são 1, 1, 2, 3, 5, os quais — até agora — formam uma subsequência da sequência de Fibonacci.



- A sequência $D(n)$, em que $D(n)$ descreve o número de maneiras diferentes de pintar o chão de um prédio com n andares, de modo que o chão de cada andar seja pintado de amarelo ou verde e que dois andares adjacentes não podem, ambos, ter o chão verde (embora dois andares adjacentes possam ter o chão amarelo), $n \geq 1$. Os quatro primeiros valores são 2, 3, 5, 8, os quais — até agora — formam uma subsequência da sequência de Fibonacci. Por exemplo, $D(3) = 5$, já que um prédio de 3 andares pode ter o chão pintado de

A	A	A	V	V
A	A	V	A	A
A	V	A	V	A

(Sugestão: Pense em uma expressão recorrente para $D(n+1)$.)

- O problema original apresentado por Fibonacci tratava de pares de coelhos. Dois coelhos não cruzam até terem 2 meses de idade. Depois disso, cada par de coelhos gera um novo par cada mês. Suponha que nenhum coelho

morre. Denote por $C(n)$ o número de pares de coelhos que você terá ao final de n meses se começar com um único par de coelhos. Mostre que $C(n)$ é a sequência de Fibonacci.

b. Escreva 27 e 62 como soma de números de Fibonacci distintos não consecutivos.

38. a. A sequência de *números de Catalan* é definida por recorrência por

$$C(0) = 1$$

$$C(1) = 1$$

$$C(n) = \sum_{k=1}^n C(k-1)C(n-k) \text{ para } n \geq 2$$

Calcule os valores de $C(2)$, $C(3)$ e $C(4)$ usando essa relação de recorrência.

b. Tanto Frank quanto José são candidatos à presidência da Câmara de Vereadores. O número de votos foi igual a $2n$, com n votos para Frank e n votos para José. Os votos são contados sequencialmente. O *problema de votação* é o seguinte: de quantas maneiras é possível contar os votos de modo que o total de votos para José nunca seja maior do que o de votos para Frank? Acontece que a resposta é $C(n)$, o n -ésimo número de Catalan. Por exemplo, se $n = 5$, uma sequência de contagem possível é

FFJJFJJFFJJ

Usando $n = 3$, escreva todas as sequências de contagem que satisfazem a propriedade e compare o resultado com o número de Catalan $C(3)$.

39. Uma sequência é definida por recorrência por

$$S(1) = 2$$

$$S(2) = 2$$

$$S(3) = 6$$

$$S(n) = 3S(n-3) \text{ para } n \geq 3$$

Prove que $S(n)$ é um número par para $n \geq 1$.

40. Uma sequência é definida por recorrência por

$$T(5) = 6$$

$$T(6) = 10$$

$$T(n) = 2T(n-2) + 2 \text{ para } n \geq 7$$

Prove que $T(n) \geq 2n$ para $n \geq 7$.

41. Uma sequência é definida por recorrência por

$$S(0) = 1$$

$$S(1) = 1$$

$$S(n) = 2S(n-1) + S(n-2) \text{ para } n \geq 2$$

a. Prove que $S(n)$ é um número ímpar para $n \geq 0$.

b. Prove que $S(n) < 6S(n-2)$ para $n \geq 4$.

42. Uma sequência é definida por recorrência por

$$T(0) = 1$$

$$T(1) = 2$$

$$T(n) = 2T(n-1) + T(n-2) \text{ para } n \geq 2$$

Prove que $T(n) \leq \left(\frac{5}{2}\right)^n$ para $n \geq 0$.

43. Escreva uma definição recorrente para uma progressão geométrica com termo inicial a e razão r (veja o Exercício 27, Seção 2.2).

44. Escreva uma definição recorrente para uma progressão aritmética com termo inicial a e parcela a ser somada d (veja o Exercício 28, Seção 2.2).
45. Em um experimento, determinada colônia de bactérias tem uma população inicial de 50.000. A população é contada a cada 2 horas, e, ao final de cada intervalo de 2 horas, a população triplica.
- Escreva uma definição recorrente para a sequência $A(n)$, o número de bactérias presentes no início do n -ésimo período de tempo.
 - No início de que intervalo haverá 1.350.000 bactérias presentes?
46. Uma quantia de R\$500,00 é investida em uma aplicação que paga juros de 1,2% capitalizados anualmente.
- Escreva uma definição recorrente para $P(n)$, a quantia total na aplicação no início do n -ésimo ano.
 - Depois de quantos anos a aplicação vai ter um saldo maior do que R\$570,00?
47. Uma coleção T de números é definida por recorrência por
- 2 pertence a T .
 - Se x pertencer a T , então $x + 3$ e $2 * x$ também pertencerão.
- Quais dos números a seguir pertencem a T ?
- 6
 - 7
 - 19
 - 12
48. Uma coleção M de números é definida por recorrência por
- 2 e 3 pertencem a M .
 - Se x e y pertencerem a M , então $x * y$ também pertencerá.
- Quais dos números a seguir pertencem a M ?
- 6
 - 9
 - 16
 - 21
 - 26
 - 54
 - 72 h. 218
49. Uma coleção S de cadeia de caracteres é definida por recorrência por
- a e b pertencem a S .
 - Se x pertencer a S , então xb também pertencerá.
- Quais das cadeias a seguir pertencem a S ?
- a
 - ab
 - aba
 - $aaab$
 - $bbbbbb$
50. Uma coleção W de cadeias de símbolos é definida por recorrência por
- a , b e c pertencem a W .
 - Se x pertencer a W , então $a(x)c$ também pertencerá.
- Quais das cadeias a seguir pertencem a W ?
- $a(b)c$
 - $a(a(b)c)c$
 - $a(abc)c$
 - $a(a(a(a)c)c)c$ e $a(aacc)c$
51. Um conjunto S de inteiros é definido por recorrência por
- 0 e 3 pertencem a S .

2. Se x e y pertencerem a S , então $x + y$ também pertencerá.

Use indução estrutural para provar que todo inteiro é um múltiplo de 3.

52. Um conjunto T de cadeias de símbolos é definido por recorrência por

1. pqq pertence a T .

2. Se x e y pertencerem a T , então $pxqq$, $qqxp$ e xy também pertencerão.

Use indução estrutural para provar que o número de símbolos iguais a q em qualquer cadeia em T é o dobro do número de símbolos iguais a p .

53. Dê uma definição recorrente para o conjunto de todas as fbfs predicadas unárias em x .

54. Dê uma definição recorrente para o conjunto de todas as fórmulas bem formuladas de aritmética inteira envolvendo números inteiros e as operações aritméticas $+$, $-$, $*$ e $/$.

55. Dê uma definição recorrente para o conjunto de todos os inteiros ímpares.

56. Dê uma definição recorrente para o conjunto de todas as cadeias de parênteses bem balanceados.

57. Dê uma definição recorrente para o conjunto de todas as cadeias binárias contendo um número ímpar de elementos iguais a 0.

58. Dê uma definição recorrente para o conjunto de todas as cadeias binárias contendo um número par de elementos iguais a 1.

59. Dê uma definição recorrente para o conjunto de todas as cadeias binárias que terminam com 0.

60. Dê uma definição recorrente para o conjunto de todas as cadeias binárias com o mesmo número de zeros e de uns.

61. Use a notação FBN para definir o conjunto dos inteiros positivos.

62. Use a notação FBN para definir o conjunto dos números decimais, que consiste em um sinal opcional ($+$ ou $-$) seguido de um ou mais dígitos, seguido de uma vírgula, seguida de zeros ou mais dígitos.

63. Dê uma definição recorrente para x^R , a cadeia em ordem inversa da cadeia x .

64. Dê uma definição recorrente para $|x|$, o comprimento da cadeia x .

65. Dê uma definição recorrente para a operação fatorial $n!$, para $n \geq 1$.

66. Dê uma definição recorrente para a soma de dois inteiros não negativos m e n .

67. a. Dê uma definição recorrente para a operação de escolher o máximo entre n inteiros a_1, \dots, a_n , $n \geq 2$.

b. Dê uma definição recorrente para a operação de escolher o mínimo entre n inteiros a_1, \dots, a_n , $n \geq 2$.

68. a. Dê uma definição recorrente para a conjunção de n letras de proposição em lógica proposicional, $n \geq 2$.

b. Escreva uma generalização da associatividade para a conjunção (equivalência tautológica 2b da Seção 1.1) e use indução para prová-la.

69. Sejam A e B_1, B_2, \dots, B_n letras de proposição. Prove a extensão finita para as equivalências da distributividade na lógica proposicional:

$$A \vee (B_1 \wedge B_2 \wedge \dots \wedge B_n) \Leftrightarrow (A \vee B_1) \wedge (A \vee B_2) \wedge \dots \wedge (A \vee B_n)$$

e

$$A \wedge (B_1 \vee B_2 \vee \dots \vee B_n) \Leftrightarrow (A \wedge B_1) \vee (A \wedge B_2) \vee \dots \vee (A \wedge B_n)$$

para $n \geq 2$.

70. Sejam B_1, B_2, \dots, B_n letras de proposição. Prove a extensão finita para as leis de De Morgan:

$$(B_1 \vee B_2 \vee \dots \vee B_n)' \Leftrightarrow B'_1 \wedge B'_2 \wedge \dots \wedge B'_n$$

e

$$(B_1 \wedge B_2 \wedge \dots \wedge B_n)' \Leftrightarrow B'_1 \vee B'_2 \vee \dots \vee B'_n$$

para $n \geq 2$.

Nos Exercícios 71 a 76, escreva o corpo de uma função recorrente para calcular $S(n)$, em que S é a sequência dada.

71. 1, 3, 9, 27, 81, ...

72. 2, 1, 1/2, 1/4, 1/8, ...

73. 1, 2, 4, 7, 11, 16, 22, ...

74. 2, 4, 16, 256, ...

75. $a, b, a + b, a + 2b, 2a + 3b, 3a + 5b, \dots$

76. $p, p - q, p + q, p - 2q, p + 2q, p - 3q, \dots$

77. Qual o valor retornado pela função recorrente *Mistério* a seguir para um valor de entrada n ?

Mistério(inteiro positivo n)

se $n = 1$ **então**

retorne 1

senão

retorne *Mistério*($n - 1$) + 1

fim do se

fim da função *Mistério*

78. A função recorrente a seguir é chamada inicialmente com um valor de i igual a 1. L é uma lista (*array*) de 10 inteiros. O que a função faz?

g (lista L ; inteiro positivo i ; inteiro x)

se $i > 10$ **então**

retorne 0

senão

se $L[i] = x$ **então**

retorne 10

senão

retorne $g(L, i + 1, x)$

fim do se

fim do se

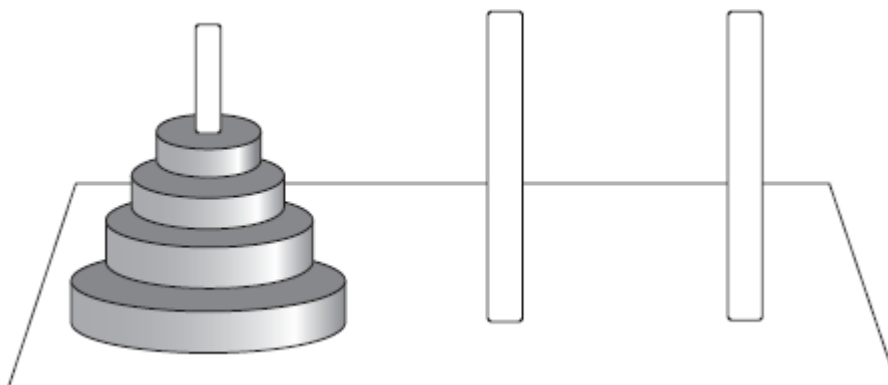
fim da função g

79. Descreva informalmente um algoritmo recursivo que inverte a ordem dos elementos em uma lista de itens.

80. Descreva informalmente um algoritmo recursivo para calcular a soma dos dígitos de um inteiro positivo.

81. Descreva informalmente um algoritmo recursivo para calcular o máximo divisor comum de dois inteiros positivos a e b , em que $a > b$. (*Sugestão*: A solução baseia-se no algoritmo de Euclides, discutido na Seção 2.3. Em particular, use a expressão (5) logo após o Exemplo 27 na Seção 2.3.)

82. O famoso quebra-cabeça da Torre de Hanói envolve três pinos e n discos de tamanhos variados empilhados em um dos pinos em ordem de tamanho, com o maior debaixo de todos e o menor no topo da pilha. O objetivo é empilhar os discos da mesma forma em outro pino; só pode ser movido um disco de cada vez, e nunca um disco maior pode ser colocado em cima de um menor. Descreva informalmente um algoritmo recursivo para resolver o quebra-cabeça da Torre de Hanói.



83. Simule a execução do algoritmo *OrdenaçãoPorSeleção* na lista L a seguir; escreva a lista após cada troca que muda a lista.

4, 10, -6, 2, 5

84. Simule a execução do algoritmo *OrdenaçãoPorSeleção* na lista L a seguir; escreva a lista após cada troca que muda a lista.

9, 0, 2, 6, 4

85. O algoritmo de busca binária é usado com a lista a seguir; x tem o valor “Curitiba”. Diga com quais elementos x é comparado.

Brasília, Campos, Itapemirim, Nova Friburgo, Petrópolis, São Paulo, Varginha

86. O algoritmo de busca binária é usado com a lista a seguir; x tem o valor “manteiga”. Diga com quais elementos x é comparado.

açúcar, chocolate, farinha, manteiga, óleo, ovos

87. Demonstre a correção da função iterativa dada nesta seção para calcular $S(n)$ do Exemplo 1, em que $S(n) = 2^n$.
88. A *Enciclopédia Online de Sequências Inteiras* (OEIS na sigla em inglês) foi desenvolvida e mantida durante muitos anos por Neil Sloane, um matemático que trabalhava na AT&T e que escreveu diversos livros sobre sequências. A Fundação OEIS gerencia, agora, o banco de dados, que contém mais de 200.000 sequências de inteiros que foram apreciadas e estudadas por muitas pessoas. (Veja oeis.org.) Há até um filme no YouTube sobre a OEIS! A *sequência de Recaman* (número A005132 no catálogo da OEIS) é uma sequência definida por recorrência da seguinte maneira:

$$a(1) = 1$$

Para $n > 1$,

$$a(n) = \begin{cases} a(n-1) - n & \text{se esse número é positivo e não está ainda na sequência} \\ \text{caso contrário} \\ a(n-1) + n \end{cases}$$

- Confirme que os primeiros termos dessa sequência são 1, 3, 6, 2, 7, 13.
- Foi conjecturado que todo inteiro não negativo vai acabar aparecendo nessa sequência. Encontre os índices dessa sequência onde aparecem os números 10, 12 e 23.

SEÇÃO 3.2

RELAÇÕES DE RECORRÊNCIA

Desenvolvemos dois algoritmos, um iterativo e o outro recorrente, para calcular um valor $S(n)$ para a sequência S do Exemplo 1. No entanto, existe um modo ainda mais fácil de calcular $S(n)$. Lembre que

$$S(1) = 2 \quad (1)$$

$$S(n) = 2S(n-1) \text{ para } n \geq 2 \quad (2)$$

Como

$$S(1) = 2 = 2^1$$

$$S(2) = 4 = 2^2$$

$$S(3) = 8 = 2^3$$

$$S(4) = 16 = 2^4$$

e assim por diante, vemos que

$$S(n) = 2^n \quad (3)$$

Usando a Equação (3), podemos substituir um valor para n e calcular diretamente $S(n)$ sem ter que calcular antes — explícita ou implicitamente, por recorrência — todos os valores menores de S . Uma equação como (3), onde podemos substituir um valor e obter diretamente o que queremos, é chamada uma **solução em forma fechada** para a relação de

recorrência (2) sujeita à condição básica (1). Quando encontramos uma solução em forma fechada, dizemos que **resolvemos** a relação de recorrência.

Relações de recorrência podem ser usadas para diversas coisas, do decaimento químico (veja o problema no início deste capítulo) ao saldo em uma aplicação bancária, do crescimento populacional de determinada espécie à proliferação de vírus computacionais. É claro que, sempre que possível, seria bom encontrar uma solução em forma fechada.

Relações de Recorrência Lineares de Primeira Ordem

Expandir, Conjecturar e Verificar

Uma técnica para resolver relações de recorrência é uma abordagem do tipo “expandir, conjecturar e verificar”, que usa repetidamente a relação de recorrência para expandir a expressão a partir do n -ésimo termo até podermos ter uma ideia da forma geral. Finalmente essa conjectura é verificada por indução matemática.

EXEMPLO 15

Considere novamente a condição básica e a relação de recorrência para a sequência S do Exemplo 1:

$$S(1) = 2 \quad (4)$$

$$S(n) = 2S(n - 1) \text{ para } n \geq 2 \quad (5)$$

Vamos fingir que não sabemos a solução em forma fechada e usar a abordagem de expandir, conjecturar e verificar para encontrá-la. Começando com $S(n)$, expandimos usando repetidamente a relação de recorrência. Lembre-se sempre de que a relação de recorrência é uma receita que diz que qualquer elemento de S pode ser substituído por duas vezes o elemento anterior. Aplicamos essa receita a S para $n, n - 1, n - 2$, e assim por diante:

$$\begin{aligned} S(n) &= 2S(n - 1) \\ &= 2[2S(n - 2)] = 2^2S(n - 2) \\ &= 2^2[2S(n - 3)] = 2^3S(n - 3) \end{aligned}$$

Olhando o padrão que está se desenvolvendo, conjecturamos que, após k tais expansões, a equação tenha a forma

$$S(n) = 2^k S(n - k)$$

Essa expansão dos elementos de S em função de elementos anteriores tem que parar quando $n - k = 1$, isto é, quando $k = n - 1$. Nesse ponto, temos

$$\begin{aligned} S(n) &= 2^{n-1} S[n - (n - 1)] \\ &= 2^{n-1} S(1) = 2^{n-1}(2) = 2^n \end{aligned}$$

que expressa a solução em forma fechada.

Ainda não terminamos, no entanto, pois conjecturamos qual deveria ser a forma geral. Precisamos confirmar nossa solução em forma fechada por indução no valor de n . A proposição que queremos provar, portanto, é que $S(n) = 2^n$ para $n \geq 1$.

Para a base da indução, $S(1) = 2^1$. Isso é verdadeiro pela Equação (4). Vamos supor que $S(k) = 2^k$. Então

$$\begin{aligned} S(k + 1) &= 2S(k) && \text{(pela Equação (5))} \\ &= 2(2^k) && \text{(pela hipótese de indução)} \\ &= 2^{k+1} \end{aligned}$$

Isso prova que nossa solução em forma fechada está correta. ■

LEMBRETE

Não fique preocupado com “ n ” e “ $n - 1$ ” na relação de recorrência. Pense nisso como “ S em algum valor é 2 vezes S no valor anterior”.

1. $T(1) = 1$
2. $T(n) = T(n-1) + 3$ para $n \geq 2$

(Sugestão: Expanda, conjecture e verifique.) ■

Uma Fórmula para a Solução

Alguns tipos de relações de recorrência têm fórmulas conhecidas para suas soluções. Uma relação de recorrência para uma sequência $S(n)$ é dita **linear** se os valores anteriores de S que aparecem na definição aparecem apenas na primeira potência. A relação de recorrência linear mais geral tem a forma

$$S(n) = f_1(n)S(n-1) + f_2(n)S(n-2) + \cdots + f_k(n)S(n-k) + g(n)$$

em que os coeficientes f_i e g podem ser expressões envolvendo n . A relação de recorrência tem **coeficientes constantes** se todos os f_i forem constantes. Ela é de **primeira ordem** se o n -ésimo termo depende apenas do termo $n-1$. Relações de recorrência lineares de primeira ordem com coeficientes constantes têm, portanto, a forma

$$S(n) = cS(n-1) + g(n) \quad (6)$$

Finalmente, a relação de recorrência é **homogênea** se $g(n) = 0$ para todo n

Vamos encontrar a fórmula para a solução da Equação (6), a relação de recorrência linear de primeira ordem geral com coeficientes constantes, sujeita à condição básica de que $S(1)$ seja conhecida. Vamos usar a abordagem de expandir, conjecturar e verificar. O que vamos fazer é uma generalização do que foi feito no Exemplo 15. Usando a Equação (6) repetidamente e simplificando, obtemos

$$\begin{aligned} S(n) &= cS(n-1) + g(n) \\ &= c[cS(n-2) + g(n-1)] + g(n) \\ &= c^2S(n-2) + cg(n-1) + g(n) \\ &= c^2[cS(n-3) + g(n-2)] + cg(n-1) + g(n) \\ &= c^3S(n-3) + c^2g(n-2) + cg(n-1) + g(n) \end{aligned}$$

Após k expansões, a forma geral parece ser

$$S(n) = c^kS(n-k) + c^{k-1}g(n-(k-1)) + \cdots + cg(n-1) + g(n)$$

Se o primeiro termo da sequência for conhecido, então a expansão terminará quando $n-k=1$, ou $k=n-1$, onde temos

$$\begin{aligned} S(n) &= c^{n-1}S(1) + c^{n-2}g(2) + \cdots + cg(n-1) + g(n) \\ &= c^{n-1}S(1) + c^{n-2}g(2) + \cdots + c^1g(n-1) + c^0g(n) \end{aligned} \quad (7)$$

Podemos usar a **notação de somatório** para escrever parte dessa expressão de forma mais compacta. A letra grega maiúscula sigma, Σ , denota uma soma. A notação

$$\sum_{i=p}^q (\text{expressão})$$

diz para substituir na expressão os valores sucessivos de i , o **índice do somatório**, do limite inferior p até o limite superior q , e depois somar os resultados. (Veja o Apêndice B para uma discussão da notação de somatório.) Assim, por exemplo,

$$\sum_{i=1}^n (2i - 1) = 1 + 3 + 5 + \cdots + (2n - 1)$$

No Exemplo 14, Seção 2.2, provamos por indução que o valor dessa soma é n^2 .

Com a notação de somatório, a Equação (7) fica

$$S(n) = c^{n-1}S(1) + \sum_{i=2}^n c^{n-i}g(i)$$

Pode-se usar indução, como no Exemplo 15, para verificar que essa fórmula é a solução da relação de recorrência (6) (veja o Exercício 26).

Portanto, a solução da relação de recorrência (6) é

$$S(n) = c^{n-1}S(1) + \sum_{i=2}^n c^{n-i}g(i) \quad (8)$$

Essa ainda não é, no entanto, uma solução em forma fechada, porque precisamos encontrar uma expressão para o somatório. Em geral, ou é trivial encontrar a soma ou já encontramos seu valor na Seção 2.2 usando indução matemática. (Se não conseguirmos encontrar uma expressão para o somatório, não estamos em situação melhor do que antes. Precisamos iterar para calcular o somatório, em vez de iterar usando a relação de recorrência, para obter o valor desejado.)

Encontramos, portanto, uma solução geral — Equação (8) — de uma vez por todas para qualquer relação de recorrência da forma (6); esse processo *não precisa ser repetido*. Tudo que é necessário é colocar seu problema na forma (6) para encontrar o valor de c e a fórmula para $g(n)$, e depois substituir esses valores na expressão em (8). A notação $g(n)$ na Equação (6) é a notação usual para uma função de n ; só estudaremos funções formalmente no Capítulo 5, mas você pode pensar em $g(n)$ como uma “receita” para o que fazer com seu argumento n . Se, por exemplo,

$$g(n) = 2n$$

então g dobra o valor de qualquer que seja seu argumento:

$$g(3) = 2(3) = 6 \quad g(27) = 2(27) = 54 \quad \text{e} \quad g(i) = 2i$$

Esse último valor, $2i$, seria usado na Equação (8) se $g(n) = 2n$.

EXEMPLO 16

A sequência $S(n)$ do Exemplo 15,

$$\begin{aligned} S(1) &= 2 \\ S(n) &= 2S(n-1) \text{ para } n \geq 2 \end{aligned}$$

é uma relação de recorrência linear homogênea de primeira ordem com coeficientes constantes. Em outras palavras, ela coincide com a Equação (6) com $c = 2$ e $g(n) = 0$. Como $g(n) = 0$, a função g é sempre igual a 0 qualquer que seja seu argumento. Da fórmula (8), a solução em forma fechada é

$$S(n) = 2^{n-1}(2) + \sum_{i=2}^n 0 = 2^n$$

o que está de acordo com nosso resultado anterior. ■

Você tem agora duas maneiras alternativas para resolver uma relação de recorrência linear de primeira ordem com coeficientes constantes. A Tabela 3.2 resume essas abordagens.

TABELA 3.2	
Para Resolver Relações de Recorrência da Forma $S(n) = cS(n - 1) + g(n)$ Sujeita à Condição Básica $S(1)$	
Método	Passos
Expandir, conjecturar, verificar	<ol style="list-style-type: none"> 1. Use repetidamente a relação de recorrência até poder compreender o padrão. 2. Decida qual será o padrão quando $n - k = 1$. 3. Verifique a fórmula resultante por indução.
Fórmula da solução	<ol style="list-style-type: none"> 1. Coloque sua relação de recorrência na forma $S(n) = cS(n - 1) + g(n) \text{ para encontrar } c \text{ e } g(n).$ 2. Use c, $g(n)$ e $S(1)$ na fórmula $S(n) = c^{n-1}S(1) + \sum_{i=2}^n c^{n-i}g(i)$ 3. Calcule o somatório resultante para obter a expressão final.

EXEMPLO 17

Encontre uma solução em forma fechada para a relação de recorrência

$$S(n) = 2S(n - 1) + 3 \text{ para } n \geq 2$$

sujeita à condição básica

$$S(1) = 4$$

Usaremos o método da fórmula da solução. Comparando nossa relação de recorrência

$$S(n) = 2S(n - 1) + 3$$

com a forma geral $S(n) = cS(n - 1) + g(n)$, vemos que

$$c = 2 \quad g(n) = 3$$

O fato de que $g(n) = 3$ diz que g tem um valor constante de 3 independentemente do valor de seu argumento; em particular, $g(i) = 3$. Substituindo na forma geral da solução

$$S(n) = c^{n-1}S(1) + \sum_{i=2}^n c^{n-i}g(i)$$

Obtemos

$$\begin{aligned}
S(n) &= 2^{n-1}(4) + \sum_{i=2}^n 2^{n-i}(3) \\
&= 2^{n-1}(2^2) + 3 \sum_{i=2}^n 2^{n-i} \\
&= 2^{n+1} + 3[2^{n-2} + 2^{n-3} + \dots + 2^1 + 2^0] \\
&= 2^{n+1} + 3[2^{n-1} - 1] \quad (\text{do Exemplo 15, Seção 2.2})
\end{aligned}$$

Logo, o valor de $S(5)$, por exemplo, é $2^6 + 3(2^4 - 1) = 64 + 3(15) = 109$. De maneira alternativa, usando o método de expandir, conjecturar e verificar, expandindo temos

$$\begin{aligned}
S(n) &= 2S(n-1) + 3 \\
&= 2[2S(n-2) + 3] + 3 = 2^2S(n-2) + 2 \cdot 3 + 3 \\
&= 2^2[2S(n-3) + 3] + 3 \cdot 2 + 3 = 2^3S(n-3) + 2^2 \cdot 3 + 2 \cdot 3 + 3 \\
&\vdots
\end{aligned}$$

O padrão geral parece ser

$$S(n) = 2^k S(n-k) + 2^{k-1} \cdot 3 + 2^{k-2} \cdot 3 + \dots + 2^2 \cdot 3 + 2 \cdot 3 + 3$$

o qual, quando $n-k=1$ ou $k=n-1$, fica

$$\begin{aligned}
S(n) &= 2^{n-1}S(1) + 2^{n-2} \cdot 3 + 2^{n-3} \cdot 3 + \dots + 2^2 \cdot 3 + 2 \cdot 3 + 3 \\
&= 2^{n-1}(4) + 3[2^{n-2} + 2^{n-3} + \dots + 2^2 + 2 + 1] \\
&= 2^{n+1} + 3[2^{n-1} - 1] \quad (\text{do Exemplo 15, Seção 2.2})
\end{aligned}$$

Finalmente, precisamos provar por indução que $S(n) = 2^{n+1} + 3[2^{n-1} - 1]$.

Caso básico: $n=1$: $S(1) = 4 = 2^2 + 3[2^0 - 1]$, verdadeiro

Suponha $S(k) = 2^{k+1} + 3[2^{k-1} - 1]$

Mostre que $S(k+1) = 2^{k+2} + 3[2^k - 1]$

$$\begin{aligned}
S(k+1) &= 2S(k) + 3 && (\text{pela relação de recorrência}) \\
&= 2(2^{k+1} + 3[2^{k-1} - 1]) + 3 && (\text{pela hipótese de indução}) \\
&= 2^{k+2} + 3 \cdot 2^k - 6 + 3 && (\text{multiplicando}) \\
&= 2^{k+2} + 3[2^k - 1]
\end{aligned}$$

LEMBRETE

Ao expandir, certifique-se de que colocou a receita completa da relação de recorrência, com todos os termos, como o $+3$ neste exemplo.

EXEMPLO 18

Encontre uma solução em forma fechada para a relação de recorrência

$$T(n) = T(n-1) + (n+1) \text{ para } n \geq 2$$

sujeita à condição básica

$$T(1) = 2$$

Usando a fórmula para a solução e comparando a relação de recorrência com a forma geral da Equação (6), $S(n) = cS(n-1) + g(n)$, vemos que $c = 1$ e $g(n) = n + 1$. Vamos substituir na Equação (8), $S(n) = c^{n-1}S(1) + \sum_{i=2}^n c^{n-i}g(i)$, em que $g(i)$ será $i + 1$.

$$\begin{aligned} T(n) &= (1)^{n-1}(2) + \sum_{i=2}^n (1)^{n-i}(i+1) \\ &= 2 + \sum_{i=2}^n (i+1) \\ &= 2 + (3 + 4 + \cdots + (n+1)) \\ &= \frac{(n+1)(n+2)}{2} - 1 \end{aligned} \quad (\text{do Problema Prático 7, Seção 2.2})$$

EXEMPLO 19

Considere o problema de ler dados em um disco rígido em um computador.⁴ O disco é organizado em uma série de sulcos circulares concêntricos, divididos em setores. Cada setor contém um bloco de dados (Figura 3.1).

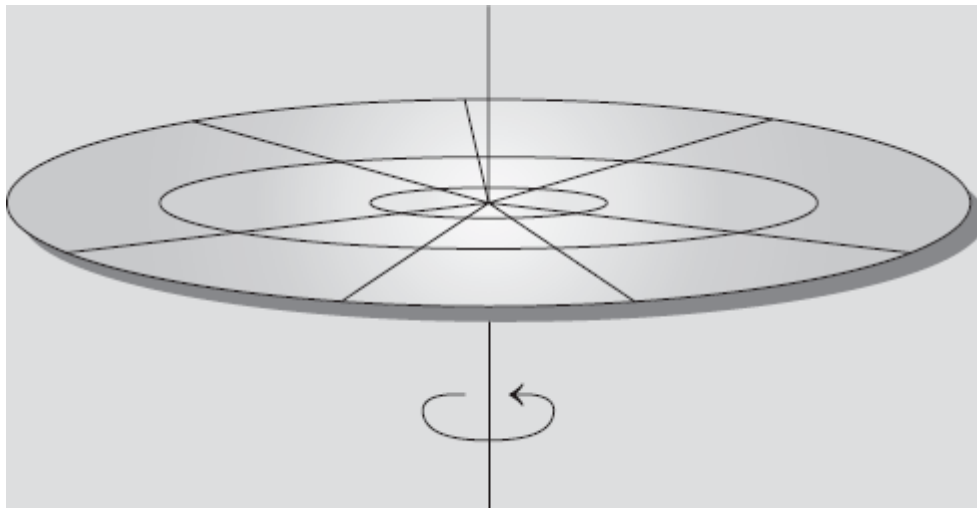


Figura 3.1

O tempo para colocar um bloco particular de dados na memória é composto de três partes:

1. *Tempo de busca* — o tempo necessário para posicionar a cabeça de leitura sobre o sulco apropriado. Esse tempo varia de acordo com as posições relativas entre a cabeça de leitura e o sulco apropriado quando é feito o comando de leitura. No melhor caso, a cabeça de leitura já está sobre o sulco em questão e o tempo de busca é 0. No pior caso, supondo que existem n sulcos, a cabeça de leitura poderia estar sobre o sulco 1 e ter que se mover para o sulco n , um movimento de $n - 1$ unidades, em que uma *unidade* é a distância entre sulcos adjacentes. Podemos supor que o tempo de busca é algum múltiplo do número de unidades.
2. *Tempo de latência* — tempo necessário para que o setor desejado fique sob a cabeça de leitura. Esse tempo também varia dependendo se o setor correto está chegando sob a cabeça de leitura (tempo de latência mínimo) ou se acabou de passar e é necessária uma rotação completa

(tempo de latência máximo).

3. *Tempo de transferência* — tempo necessário para ler um bloco posicionado sob a cabeça de leitura, em geral um tempo constante para todos os blocos.

O problema é encontrar o tempo de busca médio, na verdade, encontrar o número médio $A(n)$ de unidades. As hipóteses são que existem n sulcos, que a cabeça de leitura está posicionada sobre algum sulco i e que é igualmente provável que a cabeça de leitura tenha que se movimentar para qualquer sulco j .

A Tabela 3.3 mostra o número de unidades ao se ir de um sulco para outro, em que as linhas denotam os sulcos de saída e as colunas, os de chegada. Por exemplo, se a cabeça de leitura está sobre o sulco 3 e tem que ir para o sulco n , são necessárias $n - 3$ unidades, como mostra o elemento na linha 3, coluna n . O elemento na linha n , coluna 3 é igual, já que leva o mesmo tempo para a cabeça se mover na direção contrária.

TABELA 3.3						
Sulco de destino						
Sulco de saída	1	2	3	...	$n - 1$	n
1	0	1	2	...	$n - 2$	$n - 1$
2	1	0	1	...	$n - 3$	$n - 2$
3	2	1	0	...	$n - 4$	$n - 3$
...			
$n - 1$	$n - 2$	$n - 3$	$n - 4$...	0	1
n	$n - 1$	$n - 2$	$n - 3$...	1	0

A Tabela 3.3 ilustra os n^2 movimentos possíveis entre os sulcos. Encontramos o número médio $A(n)$ de unidades para esses n^2 casos calculando o número total de unidades na tabela, $T(n)$, e dividindo por n^2 . Para calcular $T(n)$, note que $T(n) = T(n - 1) +$ (o total da última linha mais a última coluna) e que a última linha mais a última coluna contribuem com

$$2[1 + 2 + 3 + \cdots + (n - 1)] = 2 \left[\frac{(n - 1)n}{2} \right] \quad \text{(usando o Problema Prático 7, Seção 2.2)}$$

$$= (n - 1)n$$

de modo que

$$T(n) = T(n - 1) + (n - 1)n$$

O caso básico é $T(1) = 0$ (tempo de busca nulo para um disco com 1 sulco). Essa é uma relação de recorrência linear de primeira ordem com coeficientes constantes. Podemos resolvê-la usando a Equação (8), em que $c = 1$ e $g(n) = (n - 1)n$. A solução é

$$T(n) = 0 + \sum_{i=2}^n (i - 1)i$$

$$= 1 \cdot 2 + 2 \cdot 3 + 3 \cdot 4 + \cdots + (n - 1)n$$

$$= \frac{(n - 1)n(n + 1)}{3} \quad \text{(do Exercício 19, Seção 2.2)}$$

Logo, o número médio de unidades é

$$A(n) = \frac{(n-1)n(n+1)}{3} / n^2 = \frac{n^3 - n^2 + n^2 - n}{3n^2} = \frac{n^3 - n}{3n^2} = \frac{n^2 - 1}{3n} = \frac{n}{3} - \frac{1}{3n}$$

Como o melhor caso é 0 e o pior é $n-1$, poderíamos esperar uma média próxima de $n/2$, mas ela é, de fato, um pouco menor do que $n/3$. 

EXEMPLO 20

Nem toda relação de recorrência é da forma da Equação (6). Considere a relação de recorrência

$$T(1) = 1$$

$$T(n) = 2nT(n-1) \text{ para } n \geq 2$$

Embora essa seja uma relação de recorrência linear de primeira ordem, ela não tem coeficientes constantes. A Equação (8) não se aplica. Para encontrar uma solução em forma fechada, precisamos voltar para a técnica de expandir, conjecturar e verificar.

$$\begin{aligned} T(n) &= 2nT(n-1) \\ &= 2n[2(n-1)T(n-2)] = 2^2n(n-1)T(n-2) \\ &= 2^2n(n-1)[2(n-3)T(n-3)] = 2^3n(n-1)(n-2)T(n-3) \end{aligned}$$

Em geral, parece que

$$T(n) = 2^k n(n-1)(n-2) \dots (n-(k-1))T(n-k)$$

Quando $n-k=1$, $k=n-1$ e

$$T(n) = 2^{n-1} n(n-1)(n-2) \dots (2)T(1) = 2^{n-1} n(n-1)(n-2) \dots (2)(1) = 2^{n-1} n!$$

Essa é a nossa conjectura sobre a solução em forma fechada, que verificamos por indução em n .

Caso básico, $T(1)$: $T(1) = 2^{1-1} 1! = 2^0(1) = 1$, verdadeiro

Suponha $T(k)$: $T(k) = 2^{k-1} k!$

Prove que $T(k+1)$: $T(k+1) = 2^k(k+1)!$

$$T(k+1) = 2(k+1)T(k) \quad \text{(pela relação de recorrência)}$$

$$= 2(k+1)2^{k-1}k! \quad \text{(pela hipótese de indução)}$$

$$= 2^k(k+1)!$$

Portanto, nossa conjectura sobre a solução em forma fechada estava correta. 

Relações de Recorrência Lineares de Segunda Ordem

Em uma relação de recorrência de primeira ordem, o n -ésimo termo depende apenas do termo anterior. Em uma **relação de recorrência de segunda ordem**, o n -ésimo termo depende dos dois termos anteriores. Logo, relações de recorrência lineares de segunda ordem com coeficientes constantes têm a forma

$$S(n) = c_1 S(n-1) + c_2 S(n-2) \quad (9)$$

A sequência de Fibonacci é um exemplo (o Exercício 37 pede uma solução):

$$\begin{aligned} F(1) &= 1 \\ F(2) &= 1 \\ F(n) &= F(n-1) + F(n-2) \text{ para } n > 2 \end{aligned}$$

Em tais sequências, precisamos ter dois “casos básicos”, ou seja, dois valores conhecidos da sequência para gerar os valores subsequentes.

Gostaríamos de encontrar uma fórmula geral para a solução de uma relação de recorrência como (9). Se retirarmos o segundo termo, é claro que teríamos uma relação de primeira ordem linear homogênea com coeficientes constantes:

$$S(n) = c_1 S(n-1)$$

Da Equação (8), sabemos que a solução dessa relação de recorrência é da forma

$$S(n) = c_1^{n-1} S(1)$$

Vamos expressar essa solução como

$$S(n) = pr^{n-1} \quad (10)$$

em que r (ou seja, c_1) é a solução (raiz) da equação linear

$$t - c_1 = 0 \quad (11)$$

e p (ou seja, $S(1)$) satisfaz a Equação (10) para a condição inicial $n = 1$:

$$S(1) = pr^{1-1} = pr^0 = p$$

Esse ponto de vista sugere uma conjectura para a solução de (9). Como agora temos dois termos na equação propriamente dita, vamos adicionar um segundo termo em (10) e representar uma possível solução na forma

$$S(n) = pr_1^{n-1} + qr_2^{n-1} \quad (12)$$

em que r_1 e r_2 são raízes distintas (estendendo (11) a uma equação do segundo grau) de

$$t^2 - c_1 t - c_2 = 0 \quad (13)$$

Os valores p e q têm que ser escolhidos de modo a satisfazerem as duas condições iniciais:

$$\begin{aligned} S(1) &= pr_1^{1-1} + qr_2^{1-1} = p + q \\ S(2) &= pr_1^{2-1} + qr_2^{2-1} = pr_1 + qr_2 \end{aligned}$$

ou, simplificando,

$$\begin{aligned} p + q &= S(1) \\ pr_1 + qr_2 &= S(2) \end{aligned} \quad (14)$$

É claro que isso é um salto ousado de especulação da nossa parte, de modo que agora precisamos verificar que a Equação (12) é uma solução em forma fechada para a relação de recorrência (9).

Estamos tentando provar que

$$S(n) = pr_1^{n-1} + qr_2^{n-1}$$

(em que r_1, r_2, p e q são como descritos anteriormente) é uma solução de

$$S(n) = c_1 S(n-1) + c_2 S(n-2)$$

para todo $n \geq 1$. A frase “para todo $n \geq 1$ ” sugere uma demonstração por indução. Como $S(n)$ precisa “voltar atrás” dois valores para ser calculado, devemos usar o segundo princípio de indução.

Casos básicos:

Quando $n = 1$, a solução proposta nos dá

$$S(1) = pr_1^{1-1} + qr_2^{1-1} = p + q$$

Quando $n = 2$, a solução proposta nos dá

$$S(2) = pr_1^{2-1} + qr_2^{2-1} = pr_1 + qr_2$$

Ambas as equações são trivialmente verdadeiras, já que escolhemos p e q de modo que satisfaçam essas condições.

Suponha que para todo r , $1 \leq r \leq k$, $S(r) = pr_1^{r-1} + qr_2^{r-1}$. Temos que mostrar que $S(k+1) = pr_1^k + qr_2^k$. Antes de continuar, note que, como r_1 e r_2 são soluções da equação $t^2 - c_1 t - c_2 = 0$, segue que

$$\begin{aligned} r_1^2 - c_1 r_1 - c_2 &= 0 & \text{ou} & & r_1^2 &= c_1 r_1 + c_2 \\ r_2^2 - c_1 r_2 - c_2 &= 0 & \text{ou} & & r_2^2 &= c_1 r_2 + c_2 \end{aligned} \quad (15)$$

Temos

$$\begin{aligned} S(k+1) &= c_1 S(k) + c_2 S(k-1) && \text{(pela relação de recorrência)} \\ &= c_1 (pr_1^{k-1} + qr_2^{k-1}) + c_2 (pr_1^{k-2} + qr_2^{k-2}) && \text{(pela hipótese de indução)} \\ &= pr_1^{k-2} (c_2 + c_1 r_1) + qr_2^{k-2} (c_2 + c_1 r_2) && \text{aplicada duas vezes)} \\ &= pr_1^{k-2} r_1^2 + qr_2^{k-2} r_2^2 && \text{(pela Equação (15))} \\ &= pr_1^k + qr_2^k \end{aligned}$$

que é o resultado desejado. Isso confirma que a Equação (12) é uma solução da Equação (8).

O ponto-chave para a solução é a equação do segundo grau

$$t^2 - c_1 t - c_2 = 0$$

que é chamada de **equação característica** da relação de recorrência

$$S(n) = c_1 S(n-1) + c_2 S(n-2)$$

EXEMPLO 21

Resolva a relação de recorrência

$$S(n) = 2S(n-1) + 3S(n-2) \text{ para } n \geq 3$$

sujeita às condições iniciais

$$S(1) = 3$$

$$S(2) = 1$$

Nessa relação de recorrência, $c_1 = 2$ e $c_2 = 3$. Para encontrar a solução em forma fechada, formamos a equação característica

$$t^2 - 2t - 3 = 0$$

que tem raízes $r_1 = 3, r_2 = -1$. A Equação (12) nos dá a forma da solução:

$$S(n) = p3^{n-1} + q(-1)^{n-1}$$

em que, pela Equação (14), p e q satisfazem

$$\begin{aligned} p + q &= 3 \\ p(3) + q(-1) &= 1 \end{aligned}$$

Resolvendo esse sistema de equações, obtemos $p = 1, q = 2$. Portanto, a solução em forma fechada é

$$S(n) = 3^{n-1} + 2(-1)^{n-1}$$

PROBLEMA PRÁTICO 13

- a. Usando os casos básicos e a relação de recorrência, calcule os cinco primeiros termos da sequência $S(n)$ do Exemplo 21.
- b. Verifique que a solução em forma fechada no Exemplo 21 produz os cinco primeiros termos corretos. ■

PROBLEMA PRÁTICO 14

Resolva a relação de recorrência

$$T(n) = 6T(n - 1) - 5T(n - 2) \text{ para } n \geq 3$$

sujeita às condições iniciais

$$\begin{aligned} T(1) &= 5 \\ T(2) &= 13 \end{aligned} \blacksquare$$

Embora agora pareça que temos disponível o método de solução para qualquer relação de recorrência linear homogênea de segunda ordem com coeficientes constantes, isso não é bem verdade. Considere o sistema na Equação (14):

$$\begin{aligned} p + q &= S(1) \\ pr_1 + qr_2 &= S(2) \end{aligned}$$

Podemos resolver a primeira equação para p ,

$$p = S(1) - q$$

e depois substituir na segunda equação para q ,

$$\begin{aligned} [S(1) - q]r_1 + qr_2 &= S(2) \\ q(r_2 - r_1) &= S(2) - S(1)r_1 \\ q &= \frac{S(2) - S(1)r_1}{r_2 - r_1} \end{aligned}$$

Mas o que vai acontecer se a equação característica $t^2 - c_1t - c_2 = 0$ tiver uma raiz repetida, ou seja, se $r_1 = r_2$? Ih — não podemos resolver esse sistema de equações. A forma da solução quando a equação característica tem uma raiz repetida r é

$$S(n) = pr^{n-1} + q(n-1)r^{n-1} \text{ para todo } n \geq 1$$

em que p e q satisfazem as equações

$$\begin{aligned} p &= S(1) \\ pr + qr &= S(2) \end{aligned}$$

Isso pode ser provado por indução de maneira semelhante ao caso de raízes distintas (veja o Exercício 44).

EXEMPLO 22

Resolva a equação de recorrência

$$S(n) = 8S(n-1) - 16S(n-2) \text{ para } n \geq 3$$

sujeita às condições iniciais

$$\begin{aligned} S(1) &= 1 \\ S(2) &= 12 \end{aligned}$$

Nessa relação de recorrência, $c_1 = 8$ e $c_2 = -16$. Para encontrar a solução em forma fechada, formamos a equação característica

$$\begin{aligned} t^2 - 8t + 16 &= 0 \\ (t - 4)^2 &= 0 \end{aligned}$$

que tem uma raiz repetida $r = 4$. A solução é

$$S(n) = p4^{n-1} + q(n-1)4^{n-1}$$

em que

$$\begin{aligned} p &= 1 \\ p(4) + q(4) &= 12 \end{aligned}$$

Resolvendo esse sistema de equações, encontramos $p = 1$ e $q = 2$, de modo que a solução é

$$S(n) = 4^{n-1} + 2(n-1)4^{n-1} = (2n-1)4^{n-1}$$

A Tabela 3.4 resume os passos de resolução de um sistema linear homogêneo de segunda ordem com coeficientes constantes:

TABELA 3.4

Para Resolver Relações de Recorrência da Forma $S(n) = c_1 S(n-1) + c_2 S(n-2)$ Sujeitas às Condições Iniciais $S(1)$ e $S(2)$

1. Resolva a equação característica $t^2 - c_1 t - c_2 = 0$

2. Se a equação característica tiver raízes distintas r_1 e r_2 , a solução é

$$S(n) = pr_1^{n-1} + qr_2^{n-1}$$

em que

$$p + q = S(1)$$

$$pr_1 + qr_2 = S(2)$$

3. Se a equação característica tiver uma raiz repetida r , a solução é

$$S(n) = pr^{n-1} + q(n-1)r^{n-1}$$

em que

$$p = S(1)$$

$$pr + qr = S(2)$$

As demonstrações para os casos 2 e 3 permanecem iguais se as raízes da equação característica forem complexas. Em outras palavras, as fórmulas para a solução permanecem válidas.

Relações de Recorrência Dividir para Conquistar

Ainda outro tipo de relação de recorrência ocorre quando o valor de $S(n)$ não depende do termo anterior nem dos dois termos anteriores, mas depende do valor lá atrás, no meio da sequência, $S\left(\frac{n}{2}\right)$.

EXEMPLO 23

Considere a sequência com os seguintes valores:

$$S(1) = 2, S(2) = 4, S(4) = 8, S(8) = 16, S(16) = 32, \dots$$

Estamos olhando apenas para valores selecionados da sequência, ou seja, $S(n)$ quando n é uma potência de 2. Para esses termos, vemos que

$$S(n) = 2S\left(\frac{n}{2}\right). \quad \blacksquare$$

Tais relações de recorrência aparecem na análise de determinados algoritmos “dividir para conquistar”, algoritmos que resolvem um problema dividindo-o em versões menores, cada uma a metade do tamanho do problema original (veja a próxima seção). Por causa disso, tais relações de recorrência são chamadas de **relações de recorrência dividir para conquistar**. A forma geral é

$$S(n) = cS\left(\frac{n}{2}\right) + g(n) \text{ para } n \geq 2, n = 2^m \quad (16)$$

em que c é uma constante e g pode ser uma expressão envolvendo n . É conveniente considerar apenas valores de n que são potências de 2, pois, nesse caso, dividir n por 2 diversas vezes sempre resulta em um inteiro. Como veremos na próxima seção, essa não é uma restrição significativa.

Para resolver uma relação de recorrência dividir para conquistar, vamos voltar à abordagem de expandir, conjecturar e verificar. Além disso, a solução irá envolver a função logaritmo; para uma revisão da função logaritmo e suas propriedades, veja o Apêndice C.

EXEMPLO 24

Resolva a relação de recorrência

$$C(n) = 1 + C\left(\frac{n}{2}\right) \text{ para } n \geq 2, n = 2^m$$

sujeita à condição básica

$$C(1) = 1$$

Expandindo, obtemos

$$\begin{aligned} C(n) &= 1 + C\left(\frac{n}{2}\right) \\ &= 1 + \left(1 + C\left(\frac{n}{4}\right)\right) \\ &= 1 + 1 + \left(1 + C\left(\frac{n}{8}\right)\right) \\ &\quad \vdots \end{aligned}$$

e o termo geral parece ser

$$C(n) = k + C\left(\frac{n}{2^k}\right)$$

O processo para quando $n/2^k = 1$ ou $2^k = n$, o que significa que $k = \log_2 n$. (Omitiremos a notação da base 2 daqui por diante — $\log n$ irá significar $\log_2 n$. Veja o Apêndice C para uma discussão rápida de logaritmos.) Então

$$C(n) = \log n + C(1) = 1 + \log n$$

Agora vamos usar indução para verificar que $C(n) = 1 + \log n$ para todo $n \geq 1, n = 2^m$. Essa é uma forma de indução um pouco diferente porque os únicos valores em que estamos interessados são as potências de 2. Ainda usaremos 1 como o passo básico, mas depois provaremos que, se a proposição for verdadeira para k , então será verdadeira para $2k$. Assim, a proposição será verdadeira para 1, 2, 4, 8, ..., ou seja, para todas as potências inteiras não negativas de 2, que é exatamente o que queremos.

Para o caso básico,

$$C(1) = 1 + \log 1 = 1 + 0 = 1, \text{ verdadeiro}$$

Suponha que $C(k) = 1 + \log k$. Então

$$\begin{aligned} C(2k) &= 1 + C(k) && \text{(pela relação de recorrência)} \\ &= 1 + 1 + \log k && \text{(pela hipótese de indução)} \end{aligned}$$

$$= 1 + \log 2 + \log k$$

$$(\log 2 = 1)$$

$$= 1 + \log 2k$$

(propriedade dos logaritmos)

Esse cálculo completa a demonstração por indução. ■

Gostaríamos de encontrar uma solução em forma fechada para (16) sujeita à condição básica de que o valor de $S(1)$ é conhecido. Poderíamos usar a abordagem de expandir, conjecturar e verificar, mas, em vez disso, vamos transformar a Equação (16) convertendo-a em uma relação de recorrência de primeira ordem com coeficientes constantes, usar a fórmula para a solução de tal relação de recorrência que já conhecemos e depois inverter a transformação. A Figura 3.2 mostra essa abordagem que dá essa volta.

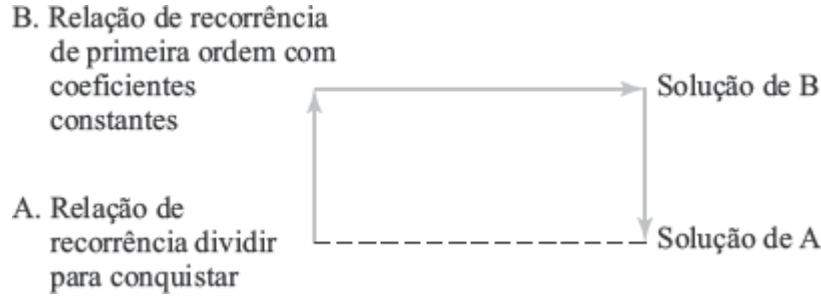


Figura 3.2

A Equação (16) supõe que $n = 2^m$ com $n \geq 2$. Segue que $m = \log n$ e $m \geq 1$. A substituição de n por 2^m na Equação (16) resulta em

$$S(2^m) = cS(2^{m-1}) + g(2^m) \quad (17)$$

Fazendo $T(m)$ representar $S(2^m)$ na Equação (17), obtemos

$$T(m) = cT(m-1) + g(2^m) \text{ para } m \geq 1 \quad (18)$$

A Equação (18) é uma relação de recorrência linear de primeira ordem com coeficientes constantes; da Equação (8), obtemos a solução

$$S(n) = c^{\log n} S(1) + \sum_{i=1}^{\log n} c^{(\log n)-i} g(2^i) \quad (21)$$

sujeita à condição básica de que $T(1)$ seja conhecida. Como a Equação (18) é válida para $m = 1$, sabemos que

$$T(1) = cT(0) + g(2)$$

Fazendo essa substituição em (19), obtemos

$$T(m) = c^m T(0) + \sum_{i=1}^m c^{m-i} g(2^i) \quad (20)$$

Invertendo a substituição $T(m) = S(2^m)$, (20) fica

$$S(2^m) = c^m S(2^0) + \sum_{i=1}^m c^{m-i} g(2^i)$$

Finalmente, fazendo $2^m = n$ ou $m = \log n$, obtemos

$$S(n) = c^{\log n} S(1) + \sum_{i=1}^{\log n} c^{(\log n)-i} g(2^i) \quad (21)$$

A Equação (21) representa, então, a solução para a relação de recorrência (16). Como antes, para usar essa solução geral você só precisa escrever sua relação de recorrência na forma (16) para determinar c e $g(n)$, e depois colocar esses valores na Equação (21). Como antes, $g(n)$ dá uma receita sobre o que fazer com o argumento n ; na Equação (21), o argumento é 2^i . Se você puder calcular esse somatório, então terá uma solução em forma fechada. A Tabela 3.5 indica os passos da solução.

LEMBRETE
Na parte do somatório na fórmula da solução geral, c está elevado à potência $(\log n) - i$ e não $(\log n) - 1$.

TABELA 3.5
Para Resolver Relações de Recorrência da Forma $S(n) = cS\left(\frac{n}{2}\right)$ para $n \geq 2, n = 2^m$, Sujeitas à Condição Inicial $S(1)$
<div>1. Escreva sua relação de recorrência na forma</div> <div>$S(n) = cS\left(\frac{n}{2}\right) + g(n)$</div> <div>para encontrar c e $g(n)$.</div>
<div>2. Use $c, g(n)$ e $S(1)$ na fórmula</div> <div>$S(n) = c^{\log n} S(1) + \sum_{i=1}^{\log n} c^{(\log n) - i} g(2^i)$</div>
<div>3. Calcule o somatório resultante para obter a expressão final.</div>


EXEMPLO 25

A relação de recorrência

$$C(1) = 1$$
$$C(n) = 1 + C\left(\frac{n}{2}\right) \text{ para } n \geq 2, n = 2^m$$

é do mesmo tipo que a Equação (16) com $c = 1$ e $g(n) = 1$. Como $g(n) = 1$, a função g tem sempre o valor 1, independentemente do argumento. A solução, de acordo com a fórmula (21), é

$$C(n) = 1^{\log n} C(1) + \sum_{i=1}^{\log n} 1^{(\log n) - i} (1)$$
$$= 1 + (\log n)(1) = 1 + \log n$$

o que está de acordo com nosso resultado anterior do Exemplo 24. 

EXEMPLO 26

Resolva a relação de recorrência

$$T(1) = 3$$

$$T(n) = 2T\left(\frac{n}{2}\right) + 2n$$

Essa relação é do mesmo tipo que a Equação (16) com $c = 2$ e $g(n) = 2n$. Logo, $g(2^i) = 2(2^i)$. A substituição na Equação (21) — a solução da Equação (16) — fornece o resultado a seguir, em que usamos o fato de que $2^{\log n} = n$.

$$\begin{aligned} T(n) &= 2^{\log n} T(1) + \sum_{i=1}^{\log n} 2^{\log n - i} 2(2^i) \\ &= 2^{\log n} (3) + \sum_{i=1}^{\log n} 2^{\log n + 1} \\ &= n(3) + (2^{\log n + 1}) \log n \\ &= 3n + (2^{\log n} \cdot 2) \log n \\ &= 3n + 2n \log n \end{aligned}$$

PROBLEMA PRÁTICO 15

Mostre que a solução da relação de recorrência

$$S(1) = 1$$

$$S(n) = 2S\left(\frac{n}{2}\right) + 1 \text{ para } n \geq 2, n = 2^m$$

é $2n - 1$. (Sugestão: Veja o Exemplo 15 na Seção 2.2 e note que $2^{\log n} = n$.) ■

SEÇÃO 3.2 REVISÃO

TÉCNICAS

- ❶ Resolução de relações de recorrência pela técnica de expandir, conjecturar e verificar.
- ❶ Resolução de relações de recorrência lineares de primeira ordem com coeficientes constantes usando uma fórmula para a solução.
- Resolução de relações de recorrência lineares homogêneas de segunda ordem com coeficientes constantes usando a equação característica.
- Resolução de relações de recorrência dividir para conquistar usando uma fórmula para a solução.

IDEIA PRINCIPAL

- Determinadas relações de recorrência têm soluções em forma fechada.

EXERCÍCIOS 3.2

Nos Exercícios 1 a 12, resolva a relação de recorrência sujeita à condição básica.

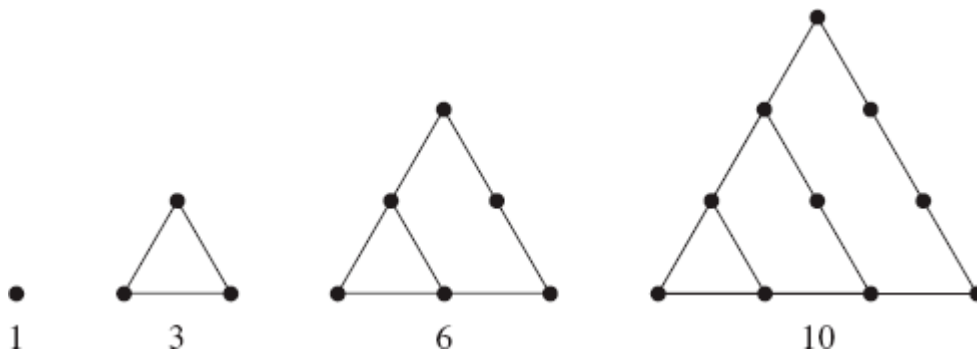
1. $S(1) = 5$
 $S(n) = S(n - 1) + 5$ para $n \geq 2$

2. $B(1) = 5$
 $B(n) = 3B(n - 1)$ para $n \geq 2$
3. $F(1) = 2$
 $F(n) = 2F(n - 1) + 2^n$ para $n \geq 2$
4. $T(1) = 1$
 $T(n) = 2T(n - 1) + 1$ para $n \geq 2$
(Sugestão: Veja o Exemplo 15 na Seção 2.2.)
5. $A(1) = 1$
 $A(n) = A(n - 1) + n$ para $n \geq 2$
(Sugestão: Veja o Problema Prático 7 na Seção 2.2.)
6. $S(1) = 1$
 $S(n) = S(n - 1) + (2n - 1)$ para $n \geq 2$
(Sugestão: Veja o Exemplo 14 na Seção 2.2.)
7. $T(1) = 1$
 $T(n) = T(n - 1) + n^2$ para $n \geq 2$
(Sugestão: Veja o Exercício 7 na Seção 2.2.)
8. $P(1) = 2$
 $P(n) = 2P(n - 1) + n2^n$ para $n \geq 2$
(Sugestão: Veja o Problema Prático 7 na Seção 2.2.)
9. $F(1) = 1$
 $F(n) = nF(n - 1)$ para $n \geq 2$
10. $S(1) = 1$
 $S(n) = nS(n - 1) + n!$ para $n \geq 2$
11. $A(1) = 1$
 $A(n) = 2(n - 1)A(n - 1)$ para $n \geq 2$
(Sugestão: $0!$ é definido como 1.)
12. $P(1) = 2$
 $P(n) = 3(n + 1)P(n - 1)$ para $n \geq 2$
13. No início deste capítulo, o empreiteiro afirmou:

O material a ser estocado no local de produtos químicos decai, tornando-se material inerte, a uma taxa de 5% ao ano. Portanto, ao final de 20 anos restará apenas aproximadamente um terço do material ativo original.

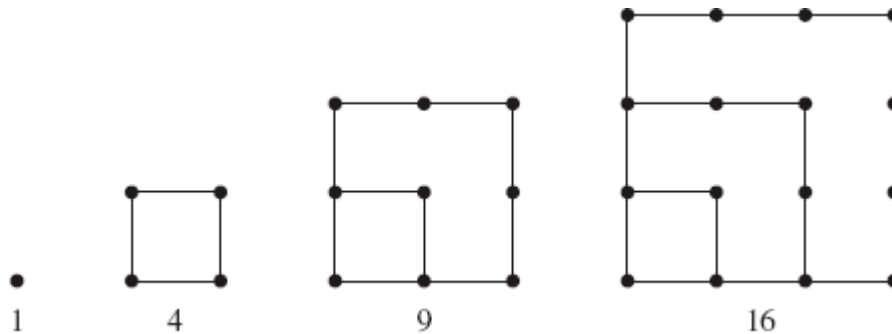
 - a. Escreva uma relação de recorrência $T(n)$ para a quantidade de material ativo no início do ano n . Suponha que $T(1) = X$, uma quantidade específica, mas desconhecida.
 - b. Resolva essa relação de recorrência.
 - c. Calcule $T(21)$ para verificar a afirmação do empreiteiro; note que ao final de 20 anos começa o vigésimo primeiro ano.
14. Uma colônia de morcegos é contada a cada 2 meses. As quatro primeiras contagens foram de 1200, 1800, 2700 e 4050.
 - a. Supondo que essa taxa de crescimento continue, escreva uma relação de recorrência para o número de morcegos na n -ésima contagem.
 - b. Resolva essa relação de recorrência.
 - c. Qual será a 2ª contagem?
15. Uma mensagem contendo um vírus foi enviada a 1000 endereços de correio eletrônico. Depois de 1 segundo, uma máquina recipiente transmite 10 novas mensagens com vírus e depois disso o vírus desabilita a si mesmo naquela máquina.
 - a. Escreva uma relação de recorrência para o número de mensagens com vírus enviadas no n -ésimo segundo.
 - b. Resolva essa relação de recorrência.
 - c. Quantas mensagens com vírus são enviadas no final de 20 segundos, ou seja, no início do 21º segundo?

16. O consumo de gás natural no estado de Nova Jersey, nos Estados Unidos, foi de 614.908 milhões de pés cúbicos ($\approx 17.217 \text{ m}^3$) em 2008 e de 653.459 milhões de pés cúbicos ($\approx 18.297 \text{ m}^3$) em 2010.
- Supondo uma taxa r de crescimento percentual anual constante, escreva uma relação de recorrência (em termos de r) para o consumo total de gás natural no estado de Nova Jersey no ano n .
 - Resolva essa relação de recorrência (em termos de r).
 - Usando os dados fornecidos, calcule o valor de r .
 - Qual será o consumo total de gás natural no estado de Nova Jersey no ano 2020?
17. Um empréstimo de R\$5.000,00 paga juros de 12% ao ano. É feito um pagamento de R\$80,00 por mês.
- Escreva uma relação de recorrência para o saldo do empréstimo no início do mês n .
 - Resolva essa relação de recorrência. (Veja o Exercício 27 da Seção 2.2 para a fórmula da soma de uma progressão geométrica.)
 - Quanto falta pagar no início do 19.º mês?
18. Investe-se R\$1.000,00 em uma aplicação que paga 3% de juros ao ano. Ao final de cada ano, aplica-se mais R\$100,00.
- Escreva uma relação de recorrência para o saldo do investimento no início do mês n .
 - Resolva essa relação de recorrência. (Veja o Exercício 27 da Seção 2.2 para a fórmula da soma de uma progressão geométrica.)
 - Qual será o total da aplicação no início do oitavo ano?
19. Estima-se que a população de mariscos em uma baía é de aproximadamente 1.000.000. Estudos mostram que a poluição reduz essa população em torno de 2% ao ano, enquanto outros fatores parecem reduzir essa população em 10.000 por ano.
- Escreva uma relação de recorrência para a população de mariscos no início do ano n .
 - Resolva essa relação de recorrência. (Veja o Exercício 27 da Seção 2.2 para a fórmula da soma de uma progressão geométrica.)
 - Qual será a população aproximada de mariscos no início do ano 10?
20. Determinada espécie protegida normalmente dobra sua população a cada mês. A população inicial era de 20, mas, no início do mês seguinte, um espécime morreu de infecção. Nos meses seguintes, a infecção matou 2, depois 4, depois 8 e assim sucessivamente.
- Escreva uma relação de recorrência para o tamanho da população no início do mês n .
 - Resolva essa relação de recorrência.
 - Qual o tamanho da população no início do mês 7?
21. Um vírus de computador que se prolifera por mensagens de correio eletrônico (*e-mail*) é colocado em 3 máquinas no primeiro dia. Diariamente, cada computador infectado no dia anterior infecta 5 novas máquinas. No final do segundo dia, é desenvolvido um programa para atacar o vírus, e se limpa 1 computador. Cada dia a partir daí, são limpas 6 vezes mais máquinas do que foram limpas no dia anterior.
- Escreva uma relação de recorrência para o número total de máquinas infectadas no dia n .
 - Resolva essa relação de recorrência.
 - Quantos dias irão se passar até os efeitos do vírus estarem completamente eliminados?
22. Este problema está relacionado com o quebra-cabeça da Torre de Hanói (veja o Exercício 82 na Seção 3.1).
- Com base no algoritmo recursivo do Exercício 82 na Seção 3.1, encontre uma relação de recorrência $M(n)$ para o número de movimentos necessários para se resolver o quebra-cabeça da Torre de Hanói com n discos.
 - Resolva essa relação de recorrência. (*Sugestão*: Veja o Exercício 15 na Seção 2.2.)
 - Faça os passos do algoritmo para $n = 3$ e registre o número de movimentos necessários. Compare esse número com o resultado do item (b) com $n = 3$.
 - A origem mítica do quebra-cabeça da Torre de Hanói trata de 64 discos de ouro que um grupo de monges está movendo de uma torre para outra. Quando eles terminarem sua tarefa, o mundo vai acabar. Supondo que os monges movem um disco por segundo, calcule o número de anos necessários para completar a tarefa.
23. Os primeiros membros da Associação de Pitágoras definiram *números poligonais* como o número de pontos em determinadas configurações geométricas. Os primeiros *números triangulares* são 1, 3, 6 e 10:



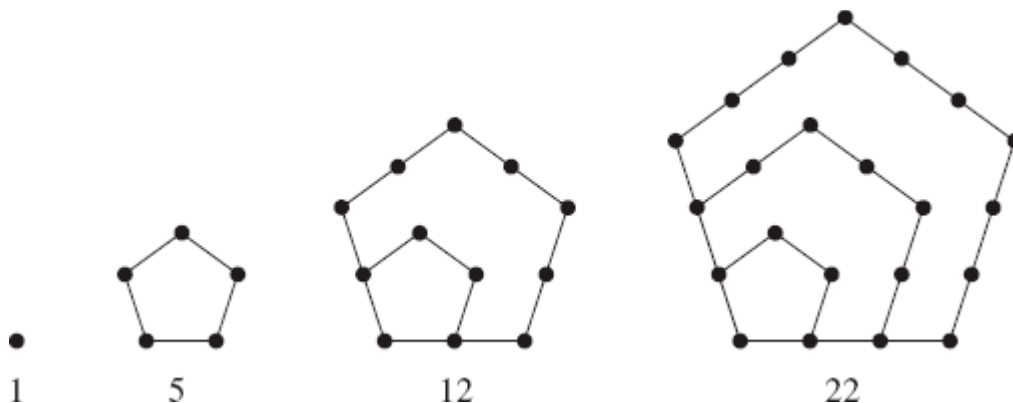
Encontre e resolva uma relação de recorrência para o n -ésimo número triangular. (*Sugestão:* Veja o Problema Prático 7 na Seção 2.2.)

24. Os primeiros *números quadrangulares* (veja o exercício anterior) são 1, 4, 9 e 16:



Encontre e resolva uma relação de recorrência para o n -ésimo número quadrangular. (*Sugestão:* Veja o Exemplo 14 na Seção 2.2.)

25. Os primeiros *números pentagonais* (veja o Exercício 23) são 1, 5, 12 e 22:



Encontre e resolva uma relação de recorrência para o n -ésimo número pentagonal. (*Sugestão:* Veja o Exercício 28 na Seção 2.2 para a fórmula da soma de uma progressão aritmética.)

26. Use indução para verificar que a Equação (8) desta seção é a solução da relação de recorrência (6) sujeita à condição básica de que $S(1)$ é conhecido.

Nos Exercícios 27 a 34, resolva a relação de recorrência sujeita às condições iniciais dadas.

27. $T(1) = 5$

$T(2) = 11$

$T(n) = 5T(n-1) - 6T(n-2)$ para $n \geq 3$

28. $A(1) = 7$

$A(2) = 18$

$A(n) = 6A(n-1) - 8A(n-2)$ para $n \geq 3$

29. $S(1) = 4$

$S(2) = -2$

$S(n) = -S(n-1) + 2S(n-2)$ para $n \geq 3$

30. $P(1) = 5$
 $P(2) = 17$
 $P(n) = 7P(n-1) - 12P(n-2)$ para $n \geq 3$
31. $F(1) = 8$
 $F(2) = 16$
 $F(n) = 6F(n-1) - 5F(n-2)$ para $n \geq 3$
32. $T(1) = -1$
 $T(2) = 7$
 $T(n) = -4T(n-1) - 3T(n-2)$ para $n \geq 3$
33. $B(1) = 3$
 $B(2) = 14$
 $B(n) = 4B(n-1) - 4B(n-2)$ para $n \geq 3$
34. $F(1) = -10$
 $F(2) = 40$
 $F(n) = -10F(n-1) - 25F(n-2)$ para $n \geq 3$

Nos Exercícios 35 e 36, resolva a relação de recorrência sujeita às condições iniciais dadas; as soluções envolvem números complexos.

35. $A(1) = 8$
 $A(2) = 8$
 $A(n) = 2A(n-1) - 2A(n-2)$ para $n \geq 3$
36. $S(1) = 4$
 $S(2) = -8$
 $S(n) = -4S(n-1) - 5S(n-2)$ para $n \geq 3$
37. Resolva a relação de Fibonacci

$$\begin{aligned} F(1) &= 1 \\ F(2) &= 1 \\ F(n) &= F(n-1) + F(n-2) \text{ para } n > 2 \end{aligned}$$

Compare sua resposta com o Exercício 31 da Seção 3.1.

38. Encontre uma solução em forma fechada para a sequência de Lucas

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

39. Em um condomínio novo, as casas começaram a ser vendidas por R\$200.000,00, em média. No início do segundo mês, o preço médio de venda subiu para R\$250.000,00. No início de cada mês a seguir, o aumento do preço médio foi de metade do aumento no mês anterior.
- Escreva e resolva uma relação de recorrência para $M(n)$, o preço médio de venda no início do mês n .
 - No início de que mês o preço médio estará em uma faixa de R\$2.000,00 em torno de R\$300.000,00?
40. Um sítio com solo contaminado é testado mensalmente para a presença de determinado microrganismo. Inicialmente, são encontrados 950 microrganismos por cm^3 de solo; no início do mês 2, há 1000 organismos por cm^3 . Sem tratamento, a taxa de crescimento desse microrganismo aumenta 25% por mês.
- Escreva e resolva uma relação de recorrência para $O(n)$, o número de organismos presentes no início do mês n .
 - No final de que mês o número de organismos ultrapassa, pela primeira vez, 5000 organismos por cm^3 ?

41. Prove que o número de cadeias binárias de comprimento n sem dois zeros consecutivos é dado pelo termo $F(n+2)$ da sequência de Fibonacci. (*Sugestão*: Escreva uma relação de recorrência; considere as cadeias de comprimento n que terminam em 1 e as que terminam em 0.)
42. a. Encontre uma relação de recorrência para o número de cadeias binárias de comprimento n sem dois uns consecutivos.
b. Quantas cadeias binárias de comprimento 4 têm dois uns consecutivos? Quais são essas cadeias?
43. Considere a relação de recorrência $S(n) = c_1S(n-1) + c_2S(n-2)$ uma relação de recorrência linear homogênea de segunda ordem com coeficientes constantes em que $c_2 = 0$. Resolva essa relação de recorrência usando sua equação característica e prove que a solução é a mesma que a da Equação (8).
44. Prove que

$$S(n) = pr^{n-1} + q(n-1)r^{n-1}$$

em que

$$\begin{aligned} p &= S(1) \\ pr + qr &= S(2) \end{aligned}$$

é uma solução da relação de recorrência $S(n) = c_1S(n-1) + c_2S(n-2)$ para todo $n \geq 1$ quando r for uma raiz repetida da equação característica.

Nos Exercícios 45 a 48, resolva a relação de recorrência sujeita à condição básica dada. (*Sugestão*: Veja o Exemplo 15 na Seção 2.2 e note que $2^{\log n} = n$.)

45. $P(1) = 1$

$$P(n) = 2P\left(\frac{n}{2}\right) + 3 \text{ para } n \geq 2, n = 2^m$$

46. $T(1) = 3$

$$T(n) = T\left(\frac{n}{2}\right) + n \text{ para } n \geq 2, n = 2^m$$

47. $S(1) = 1$

$$S(n) = 2S\left(\frac{n}{2}\right) + n \text{ para } n \geq 2, n = 2^m$$

48. $P(1) = 1$

$$P(n) = 2P\left(\frac{n}{2}\right) + n^2 \text{ para } n \geq 2, n = 2^m$$

SEÇÃO 3.3

ANÁLISE DE ALGORITMOS

A Ideia Geral

Muitas vezes, existe mais de um algoritmo para executar a mesma tarefa. Como supomos que todos esses algoritmos executem corretamente, precisamos de alguma base de comparação para decidir qual deles usar em determinada situação. Diversos critérios poderiam ser usados para se decidir qual é o algoritmo “melhor”. Poderíamos perguntar, por exemplo, qual é o mais fácil de entender, qual é o que usa a memória da máquina de modo mais eficiente ao ser implementado em alguma linguagem ou qual é o que roda mais eficientemente. Poder-se-ia esperar que julgar se um algoritmo está “rodando eficientemente” significa usar um cronômetro enquanto o algoritmo está executando. Mas o cronômetro pode nos dizer mais sobre a velocidade do processador do que sobre a eficiência inerente do algoritmo. Até mesmo a marcação do tempo que os códigos de algoritmos competidores levam no mesmo processador e usando os mesmos dados de entrada pode levar a resultados que nos induzam a erros sobre o que poderia ocorrer quando o conjunto de dados for maior.

Em vez disso, julgamos a eficiência de um algoritmo em relação ao tempo estimando o número de operações que ele tem que executar. Contamos apenas as operações básicas para a tarefa em questão e não operações de “manutenção” ou “contabilidade”, que contribuem pouco para o trabalho total necessário.

O estudo da eficiência de algoritmos, ou seja, o número de operações que eles executam, é chamado de **análise de algoritmos**. Foram desenvolvidas diversas técnicas para a análise de algoritmos. Algumas vezes pode ser feita uma análise direta apenas inspecionando o algoritmo.

EXEMPLO 27

A seguir é dado um algoritmo para escrever, para cada aluno em uma lista, a soma das notas de m testes menos a menor nota. O laço externo percorre cada um dos n alunos; o laço interno percorre as notas dos testes de cada aluno. Para cada aluno, as notas sucessivas dos testes são somadas, e, ao final, a menor nota é subtraída do total. Essas somas e subtrações parecem ser fundamentais em como o algoritmo funciona, de modo que contaremos o trabalho feito por essas operações aritméticas.

para $i = 1$ até n **faça**

$menor = lista[i].teste[1]$

$soma = lista[i].teste[1]$

para $j = 2$ até m **faça**

$soma = soma + lista[i].teste[j]$ //A

se $lista[i].teste[j] < menor$ **então**

$menor = lista[i].teste[j]$

fim do se

fim do para

$soma = soma - menor$ //S

 escreva(“Total para o aluno”, i , “é”, $soma$)

fim do para

A subtração ocorre na linha marcada //S, que é executada uma vez para cada passagem do laço exterior (uma vez para cada aluno), um total de n vezes. As somas, no entanto, ocorrem na linha marcada //A. Isso é feito no laço interno, que executa $m - 1$ vezes para cada aluno, ou seja, para cada uma das n passagens do laço exterior. Logo, o número total de somas é igual a $n(m - 1)$. O número total de operações aritméticas é $n + n(m - 1) = nm$. É claro que o valor dessa expressão depende de n (o número de alunos) e de m (o número de testes). As quantidades n e m medem a quantidade de dados de entrada; praticamente o trabalho de qualquer algoritmo irá depender do tamanho da entrada.

O algoritmo também faz algum trabalho de “contabilidade”. São atribuídos valores a variáveis, são feitas comparações (para encontrar a menor nota de teste para cada aluno), e os índices i e j dos laços têm que ser incrementados. Mas o número de vezes que essas operações são feitas também depende do número de passagens dos laços, de modo que seu efeito pode ser o de multiplicar o resultado nm por algum fator constante. Ao comparar dois algoritmos A e B , estamos procurando, em geral, diferenças maiores do que simplesmente um múltiplo constante, e essa é a razão pela qual ignoramos os detalhes de contabilidade. ■

Suponha agora que a tarefa é procurar, em uma lista ordenada contendo n itens, um item particular x . Já conhecemos um algoritmo que executa essa tarefa, o algoritmo de busca binária da Seção 3.1. Outro algoritmo que executa a mesma tarefa é o **algoritmo de busca sequencial**, que compara, simplesmente, x com cada elemento da lista até encontrar x ou até acabar a lista. (Esse algoritmo funciona, de fato, em qualquer lista, esteja ou não ordenada.) A seguir, fornecemos uma descrição em pseudocódigo para o algoritmo de busca sequencial.

ALGORITMO	BUSCASEQUENCIAL
	$BuscaSequencial(lista\ L; inteiro\ n; itemtipo\ x)$ //procura em uma lista L com n itens pelo item x Variável local: inteiro i //marca a posição na lista $i = 1$ enquanto $L[i] \neq x$ e $i < n$ faça

```
 $i = i + 1$   
fim do enquanto  
se  $L[i] = x$  então  
    escreva("Encontrado")  
senão  
    escreva("Não encontrado")  
fim do se  
fim da função BuscaSequencial
```

Ambos os algoritmos, de busca binária e de busca sequencial, trabalham comparando elementos da lista com x até encontrar um elemento igual. De fato, é difícil imaginar como qualquer algoritmo de busca poderia evitar tais comparações, de modo que essas comparações são as operações básicas que devem ser contadas para analisar esses dois algoritmos.

O trabalho executado (número de comparações) pelo algoritmo de busca sequencial vai ser máximo quando x for o último elemento da lista ou quando x não pertencer à lista. Em qualquer desses casos, todos os elementos são comparados com x , logo são feitas n comparações. Esse é o “pior caso” para esse algoritmo, e o trabalho depende do tamanho n da entrada (o comprimento da lista). O trabalho será mínimo quando x for o primeiro elemento da lista; só é feita uma comparação. Esse é o “melhor caso” para esse algoritmo. (No algoritmo do Exemplo 27, é sempre executado o mesmo número de operações aritméticas; não existe pior caso, nem melhor caso.)

Existem muitas possibilidades entre o melhor caso e o pior caso. Se x estiver exatamente no meio da lista, a busca necessitaria de aproximadamente $n/2$ comparações. Ajudaria se pudéssemos ter alguma medida da quantidade “média” de operações necessárias. Essa medida precisaria de alguma descrição da lista média e da relação média entre x e os itens daquela lista. Os Exercícios 35 e 36 desta seção exploram alguns aspectos da análise para o caso médio do algoritmo de busca sequencial. Para a maioria dos algoritmos, no entanto, o comportamento médio é muito difícil de determinar. Para comparar a eficiência de algoritmos, portanto, muitas vezes nos contentamos com a contagem do número de operações necessárias no pior caso.

EXEMPLO 28

Dada uma longa cadeia de caracteres de texto, podemos encontrar a primeira vez que aparece uma subcadeia particular ou um “padrão” neste texto? Esse problema tem várias aplicações importantes, como

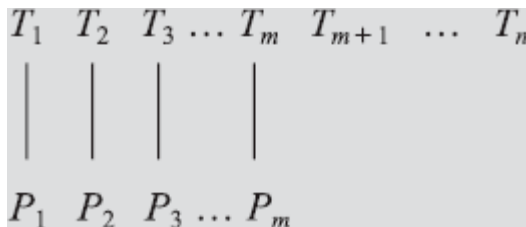
- buscar uma cadeia específica em um documento HTML que pode estar em um estilo determinado por uma folha de estilo em cascata (*Cascading Style Sheet*);
- usar o comando *grep* do UNIX para procurar determinada cadeia em um arquivo ou usar o comando “Localizar” (ou “Find”) em qualquer editor ou processador de texto;
- buscar uma sequência específica de genes em um trecho de DNA.

O DNA é uma molécula longa que é, basicamente, uma cadeia de moléculas menores, chamadas de nucleotídeos, ligados quimicamente. Existem quatro nucleotídeos, abreviados por *A*, *C*, *G* e *T*. Então uma seção de DNA poderia ser representada pela sequência

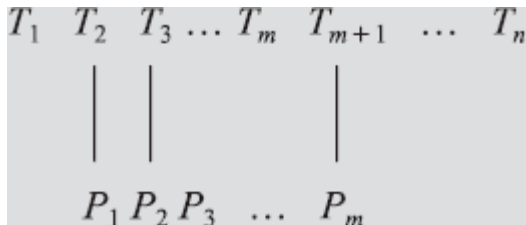
...TAATCATGGTCATAGCTGTTCTGTGTGAAATTG...

O DNA é armazenado dentro das células dos organismos vivos nos cromossomos; diversas seções desses cromossomos são identificadas como genes. Os genes, através de suas “instruções” no DNA, criam proteínas que controlam funções específicas ou características do organismo (cor do cabelo, tipo de sangue e assim por diante). Logo, todo o nosso código genético necessita de apenas quatro símbolos! O “mapeamento do genoma humano”, ou seja, a determinação da sequência inteira de DNA de seres humanos, foi um empreendimento científico enorme, essencialmente completado em 2003, embora o trabalho de identificação de genes específicos e de suas funções particulares continue. É sabido, por exemplo, que a fibrose cística é causada por uma mutação em um gene (sequência de DNA) determinado que é composto por cerca de 230.000 nucleotídeos.

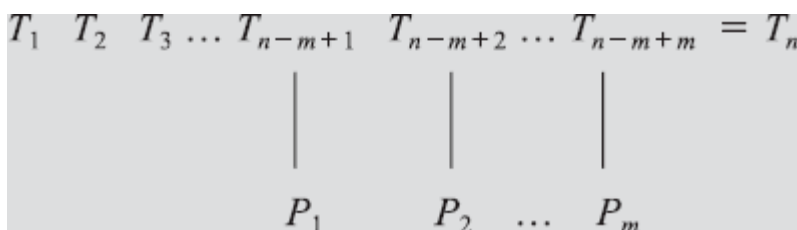
O algoritmo mais intuitivo (embora não seja o mais eficiente) para a busca de um padrão em uma cadeia compara o padrão (uma cadeia de comprimento m) com o texto (uma cadeia de comprimento n com $n \geq m$), começando com o primeiro caractere do texto e seguindo o padrão.



Se todos os m caracteres forem iguais, o padrão terá sido encontrado. Se em algum ponto o texto e o padrão diferirem, o padrão é deslizado um caractere para a direita sobre o texto e o processo de comparação começa novamente.



O último segmento do texto onde o padrão pode ser encontrado é formado pelos últimos m caracteres do texto. Esse segmento começa em T_{n-m+1} , como ilustrado a seguir.



Por exemplo, se o texto tiver 23 caracteres de comprimento e o padrão tiver 5 caracteres, então o último trecho do texto que poderá conter o padrão é $T_{19}T_{20}T_{21}T_{22}T_{23}$.

A unidade de trabalho nesse algoritmo é uma comparação entre um caractere no texto e um caractere no padrão. O melhor caso ocorre quando o padrão está nos primeiros m caracteres do texto, o que requer m comparações. O pior caso ocorre quando o texto não contém o padrão e a “janela” de padrão tiver que ser deslizada até T_{n-m+1} . Para cada um desses últimos casos, o padrão não é encontrado nos $n-m+1$ primeiros caracteres do texto, mas o pior caso ocorre quando o padrão é quase encontrado, ou seja, só o último caractere é diferente. Por exemplo, considere o texto e o padrão a seguir:

Texto: *TTTTTTTTTTTT*

Padrão: *TTTTTS*

Esse exemplo necessita de m comparações (os primeiros $m-1$ caracteres são iguais, só falham na m -ésima comparação) em cada um dos $n-m+1$ caracteres iniciais, fazendo com que o total de comparações seja igual a $m(n-m+1)$. ■

Análise Usando Relações de Recorrência

Nesta seção, vamos analisar algoritmos recursivos. Como a maior parte da atividade de um algoritmo recursivo acontece “fora das vistas”, nas diversas chamadas que podem ocorrer, uma análise usando uma técnica de contagem direta como no Exemplo 27 não vai funcionar. A análise de algoritmos recursivos envolve, muitas vezes, a resolução de uma relação de recorrência.

EXEMPLO 29

Vamos reescrever o algoritmo de busca sequencial em uma versão recorrente em vez de iterativa (que repete a mesma ação muitas vezes em um laço). O caso básico verifica se a lista já terminou e, se não tiver terminado, busca o elemento x na primeira posição da lista. Se encontrou, tudo bem, caso contrário, o algoritmo é chamado de novo no resto da lista. Um pseudocódigo para a função recursiva que busca na lista de $L[l]$ até $L[n]$ é dado a seguir; a função é chamada inicialmente com $i = 1$.

```

BuscaSequencialRecursiva(lista L; inteiros i, n; tipo item x)
//busca pelo item x na lista L de L[i] até L[n]
    se i > n então
        escreva("não encontrado")
    senão
        se L[i] = x então
            escreva("encontrado")
        senão
            BuscaSequencialRecursiva(L, i + 1, n, x)
    fim do se
fim da função BuscaSequencialRecursiva

```

A Figura 3.3 fornece uma representação visual do algoritmo de busca sequencial recursiva. Cada vez que o algoritmo é chamado, a nova lista onde será feita a busca tem apenas um elemento a menos que a lista anterior, de modo que, no pior caso, o algoritmo vai ter que trabalhar muito.

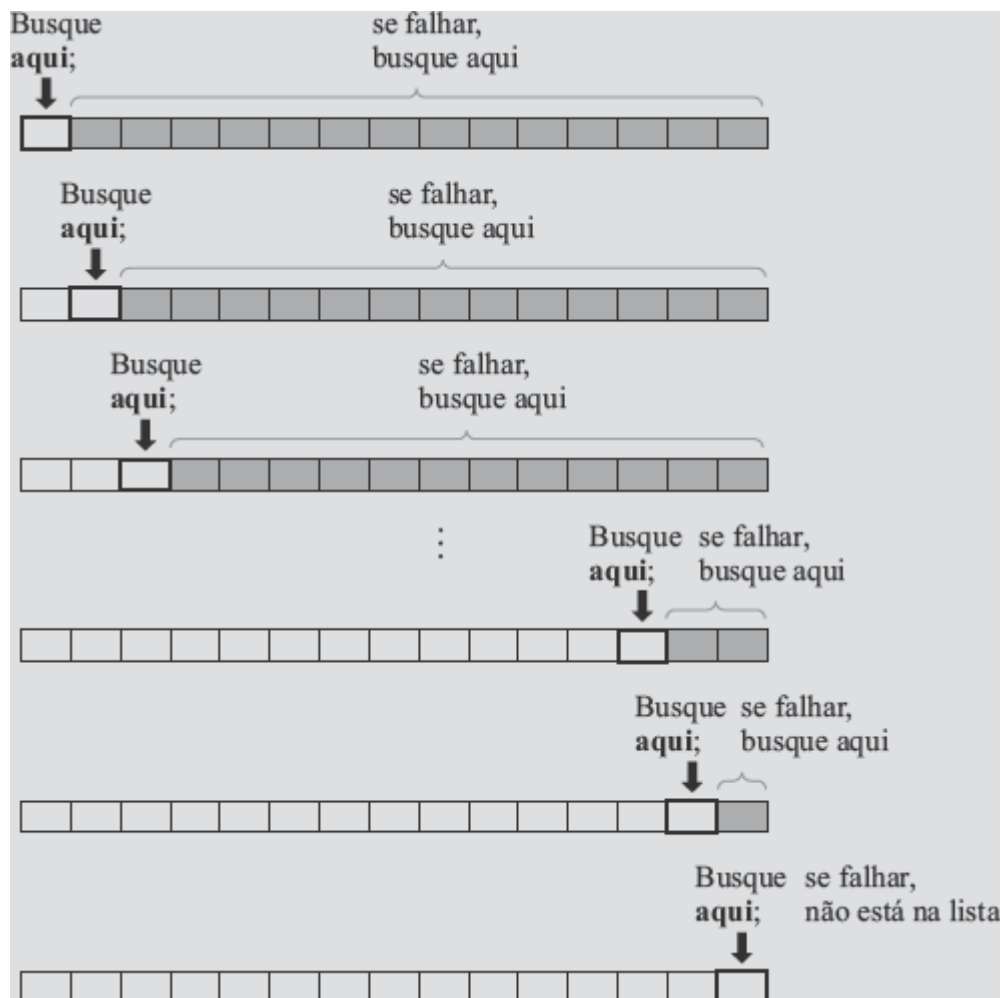


Figura 3.3

Vamos representar por $C(n)$ o número máximo de comparações necessárias para uma lista com n elementos. Essa é uma expressão simbólica para uma resposta que estamos supondo que ainda não sabemos, mas que iremos descobrir ao resolver a relação de recorrência. Com essa notação, $C(n - 1)$ representa, simbolicamente, o número máximo de comparações necessárias para buscar no resto da lista depois da primeira posição. A relação de recorrência é

$$C(1) = 1$$

(1 comparação para buscar em uma lista com 1 elemento)

$$C(n) = 1 + C(n - 1) \text{ para } n \geq 2$$

(1 comparação com o primeiro elemento, depois quantas comparações forem necessárias para o resto da lista)

Essa é uma relação de recorrência linear de primeira ordem com coeficientes constantes. Pela Equação (8) na Seção 3.2, a solução é

$$C(n) = (1)^{n-1}(1) + \sum_{i=2}^n (1)^{n-i}(1) = 1 + (n-1) = n$$

Isso está de acordo com a nossa análise anterior do pior caso. ●

EXEMPLO 30

Agora vamos analisar o pior caso do algoritmo de busca binária. Lembre-se de que o algoritmo de busca binária é um algoritmo recursivo que age em uma lista ordenada em ordem crescente. Ele primeiro compara o item procurado com o valor do meio da lista. Se essa comparação falhar, então o processo será repetido na metade direita ou na metade esquerda da lista, dependendo se o valor procurado é maior ou menor do que o valor do meio. A Figura 3.4 ilustra um caminho possível para o pior caso.

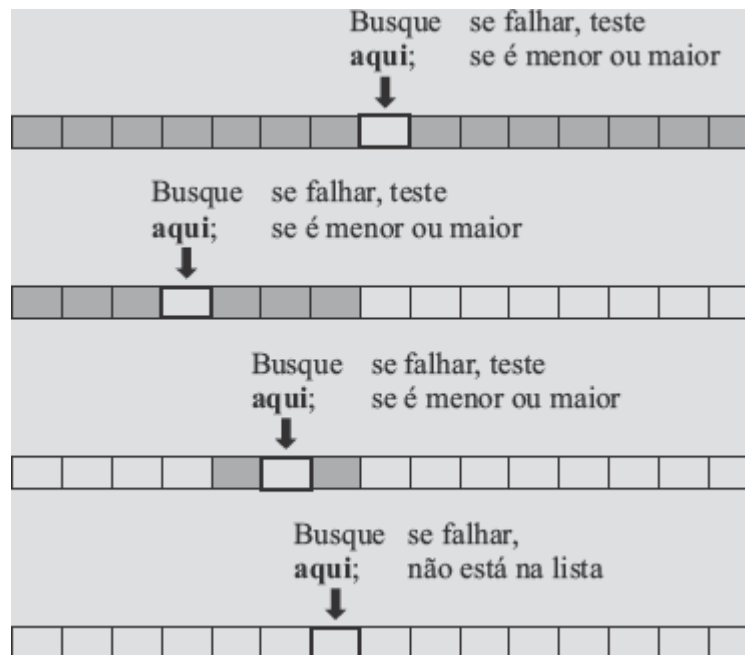


Figura 3.4

O algoritmo de busca binária é do tipo **dividir para conquistar**, quando o problema é decomposto de maneira recorrente em subproblemas significativamente menores. Se a lista original tiver n elementos, então a metade da lista terá, no máximo, $n/2$ elementos. (No Exemplo 14, em que 10 é o valor do meio, a “metade” direita da lista tem 4 elementos, mas a “metade” esquerda tem apenas 3.) Cortar a lista pela metade é muito mais rápido do que reduzi-la de um elemento, como na busca sequencial, de modo que esperamos que o caso pior da busca binária necessite de menos trabalho.

Denote por $C(n)$ o número máximo de comparações necessárias para efetuar uma busca binária em uma lista de n elementos. Então $C\left(\frac{n}{2}\right)$ representa o número máximo de comparações necessárias para uma lista com metade dos elementos. Já que vamos continuar dividindo a lista ao meio, é conveniente considerar apenas os casos em que obtemos um valor inteiro cada vez que dividimos pela metade, de modo que vamos supor que $n = 2^m$ para algum $m \geq 0$. A relação de recorrência para $C(n)$ é

$$C(1) = 1$$

(1 comparação para buscar em uma lista com 1 elemento)

$$C(n) = 1 + C\left(\frac{n}{2}\right) \text{ para } n \geq 2, n = 2^m$$

(1 comparação com o elemento do meio, depois quantas comparações forem necessárias para metade da lista)

Essa relação de recorrência foi resolvida na seção anterior (Exemplos 24 e 25). A solução é

$$C(n) = 1 + \log n$$

Pelo exemplo anterior, o número máximo de comparações necessárias em uma busca binária de uma lista ordenada com n elementos, com $n = 2^m$, é $1 + \log n$. No Exemplo 14, n era 8, e foram necessárias quatro comparações ($1 + \log 8$) no pior caso (x não pertencente à lista). Uma busca sequencial necessitaria de oito comparações. Como

$$1 + \log n < n \text{ para } n = 2^m, n \geq 4$$

a busca binária é, quase sempre, mais eficiente do que a busca sequencial. No entanto, o algoritmo de busca sequencial tem uma grande vantagem — se a lista em questão *não estiver ordenada*, o algoritmo de busca sequencial funciona, mas o de busca binária não. Se vamos primeiro ordenar a lista e depois usar o algoritmo de busca binária, precisamos, então, considerar o número de operações necessárias para ordenar a lista. Os Exercícios de 13 a 34 ao final desta seção pedem que você conte as operações necessárias para ordenar uma lista por diversos algoritmos diferentes.

PROBLEMA PRÁTICO 16

Complete a tabela a seguir para o número de comparações necessárias no pior caso para a busca sequencial e a busca binária em uma lista do tamanho indicado.

n	Busca Sequencial	Busca Binária
64		
1024		
32768		

■

Embora tenhamos calculado o trabalho para uma busca binária em uma lista de tamanho n , em que n é uma potência de 2, isso nos dá um intervalo para o trabalho necessário para valores de n que caem entre potências de 2. Essa é a razão pela qual limitar n a potências de 2 em nossa análise não é particularmente importante.

Cota Superior (Algoritmo de Euclides)

O algoritmo de Euclides, como apresentado na Seção 2.3, usa um laço de enquanto para efetuar divisões sucessivas de modo a encontrar $\text{mdc}(a, b)$ para inteiros positivos a e b , $a > b$. Para analisar o algoritmo de Euclides, precisamos decidir primeiro quais as operações que contaremos. Como esse algoritmo efetua repetidas divisões, vamos considerar a operação de divisão a nossa unidade básica. Como $a > b$, podemos usar a como medida do tamanho dos valores de entrada (que, em geral, denotamos por n). Queremos encontrar $E(a)$, que denota a quantidade de operações (o número de divisões) necessária para encontrar $\text{mdc}(a, b)$ no pior caso.

Uma versão recursiva do algoritmo de Euclides também pode ser escrita (veja o Exercício 81, Seção 3.1); a chave para a versão recursiva é reconhecer que o cálculo do $\text{mdc}(a, b)$ envolve encontrar o $\text{mdc}(b, r)$, em que r é o resto da divisão de a por b . Acabamos de ver um caso no qual as operações de um algoritmo recursivo (a busca binária) podiam ser expressas convenientemente como uma relação de recorrência em que o tamanho dos valores de entrada é reduzido pela metade após cada operação. Uma relação de recorrência expressaria $E(a)$ em termos de E com valores menores. Mas quais são esses valores menores? Para encontrar o $\text{mdc}(a, b)$, encontramos o $\text{mdc}(b, r)$, de modo que é claro que os valores de entrada estão ficando menores, mas, de que maneira? Considere o Exemplo 27 da Seção 2.3 onde, para encontrar o $\text{mdc}(420, 66)$, foram efetuadas as seguintes divisões:

$$\begin{array}{r} 420 \overline{)66} \\ \underline{-396} \\ 24 \end{array} \quad \begin{array}{r} 66 \overline{)24} \\ \underline{-48} \\ 18 \end{array} \quad \begin{array}{r} 24 \overline{)18} \\ \underline{-18} \\ 6 \end{array} \quad \begin{array}{r} 18 \overline{)6} \\ \underline{-18} \\ 0 \end{array}$$

Aqui os valores que são divididos sucessivamente são 420, 66, 24 e 18. A mudança de 420 para 66 é muito maior do que dividir pela metade, enquanto a mudança de 24 para 18 é menor.

De fato, não encontraremos uma relação de recorrência ou uma expressão exata para $E(a)$. Mas podemos, pelo menos, encontrar uma *cota superior* para $E(a)$. Uma **cota superior** é um valor superior para a quantidade de operações efetuadas

por um algoritmo; o algoritmo pode *não* precisar de *mais operações* do que a cota superior, mas pode não precisar de tantas.

Para encontrar essa cota superior, vamos mostrar que, se $i > j$ e se i for dividido por j com resto r , então $r < i/2$. Existem dois casos:

1. Se $j \leq i/2$, então $r < i/2$, pois $r < j$.
2. Se $j > i/2$, então $i = 1 * j + (i - j)$; em outras palavras, o quociente é 1 e r é $i - j$, que é $< i/2$.

No algoritmo de Euclides, o resto r em qualquer etapa torna-se o dividendo (o número que está sendo dividido) dois passos adiante. Logo, os dividendos sucessivos são, pelo menos, divididos por dois a cada duas divisões. O valor a pode ser dividido por $2 \log a$ vezes; portanto, são feitas, no máximo, $2 \log n$ divisões. Assim,

$$E(a) \leq 2 \log a \quad (1)$$

O valor de $2 \log a$ para $a = 420$ é quase 18, ao passo que foram necessárias apenas 4 divisões para se obter o $\text{mdc}(420, 66)$. É claro que essa cota superior é bastante grosseira, algo como dizer que todos os alunos em sala têm menos de três metros e meio. Uma cota superior mais fina (ou seja, mais baixa) é obtida nos Exercícios 37 a 40 ao final desta seção.

De Árvores ... e Panquecas

De Árvores ...

O mapeamento da “árvore da vida” evolucionária é objeto de pesquisa desde Charles Darwin. Até recentemente, essa pesquisa buscava semelhanças entre espécies com base em propriedades estruturais como esqueletos, mas, atualmente, os cientistas buscam semelhanças no DNA e outras evidências genéticas. Esse campo de pesquisa, chamado de filogenética, pode envolver o alinhamento de sequências moleculares de muitos milhares de moléculas, e o trabalho se transformou em um problema computacional enorme. Pesquisadores na Universidade do Texas desenvolveram um programa computacional chamado SATé (e outro novo, melhorado, SATé-II) — sigla do inglês para Estimativas Simultâneas para o Alinhamento e a Árvore —, que usa um algoritmo do tipo dividir para conquistar. Conjuntos imensos de dados são divididos em conjuntos pequenos de dados, são encontrados alinhamentos para os conjuntos pequenos de dados e depois os resultados são combinados para determinar um alinhamento total (e uma possível árvore) para o conjunto completo de dados. O alinhamento total resultante não é infalível, e o programa repete esse processo muitas vezes, criando alinhamentos e árvores novos. Um método estatístico de “verossimilhança máxima” seleciona o melhor resultado comparando com respostas conhecidas. Já foi demonstrado que essa abordagem produz resultados comparáveis a outros métodos, mais lentos, ou produz resultados mais precisos no mesmo intervalo de tempo.

Ferramentas de alinhamento de sequências e construção de árvores evolucionárias têm aplicações em áreas diferentes do traçado do caminho histórico da evolução. Por exemplo, os Centros para Controle de Doenças (CDC na sigla em inglês) usam essas ferramentas para detectar como um vírus novo emergente difere de vírus anteriores com o objetivo de planejar o contra-ataque melhor.

<http://www.tacc.utexas.edu/news/feature-stories/2012/tree-of-life>

<http://www.ncbi.nlm.nih.gov/pubmed/22139466>

... e Panquecas

Um problema apresentado na revista *American Mathematical Monthly* em 1975 por Jacob Goodman tratava de um garçom em uma lanchonete onde o cozinheiro produzia pilhas de panquecas* de tamanhos variados. O garçom, ao levar a pilha para o cliente, tentou arrumar as panquecas por ordem de tamanho, com a maior por baixo. A única ação possível era colocar uma espátula em algum ponto e virar toda a pilha acima daquele ponto. A pergunta é: qual é o número máximo das viradas necessárias para qualquer pilha com n panquecas? Esse número, P_n , é chamado de *n -ésimo número de panqueca*.

Aqui está um algoritmo simples para arrumar as panquecas. Coloque a espátula debaixo da panqueca maior e vire. Isso coloca a panqueca maior no topo da pilha. Agora coloque a espátula debaixo da panqueca inferior da seção ainda não ordenada (neste caso, a panqueca debaixo de todas) e vire. Isso coloca a panqueca maior debaixo de todas, onde deveria estar. Repita com o resto das panquecas. Assim, cada panqueca precisa de duas viradas, o que daria um total de $2n$ viradas. Mas as duas últimas panquecas precisam de, no máximo, uma virada: se elas já estiverem na ordem certa, não há necessidade de virada; se não estiverem, só precisam de uma virada. Logo, esse algoritmo precisa, no máximo, de $2(n - 2) + 1 = 2n - 3$ viradas no pior caso, o que significa que $P_n \leq 2n - 3$. Existem outros algoritmos que necessitam de menos viradas no pior caso?

Um professor na Universidade de Harvard apresentou esse problema para sua turma; diversos dias depois, um estudante do segundo ano que cursava sua disciplina procurou o professor para mostrar um algoritmo melhor. Esse algoritmo, que necessita, no máximo, de $(5n + 5)/3$ viradas, foi publicado na revista *Discrete Mathematics* em 1979. Os autores foram William Gates (o estudante) e Christos Papadimitriou.

SEÇÃO 3.3 REVISÃO

TÉCNICA

- Efetuar a análise de um algoritmo no pior caso diretamente da descrição do algoritmo ou de uma relação de recorrência.

IDEIAS PRINCIPAIS

- A análise de um algoritmo estima o número de operações básicas que o algoritmo efetua, o qual depende do tamanho dos dados de entrada.
- Muitas vezes, a análise de algoritmos recursivos leva a relações de recorrência.
- Na falta de uma expressão exata para o número de operações efetuadas por um algoritmo, pode ser possível encontrar uma cota superior.

EXERCÍCIOS 3.3

1. Modifique o algoritmo do Exemplo 27 de modo que, além de eliminar a menor nota do aluno nos testes, conta duas vezes a maior nota nos testes (como a versão antiga, seu algoritmo novo não deve fazer outras operações além de soma e subtração).
2. Qual é o número total de operações aritméticas efetuadas pelo algoritmo do Exercício 1?
3. O algoritmo a seguir soma todos os elementos de uma matriz quadrada A $n \times n$. Analise esse algoritmo, em que a unidade de trabalho é a operação de soma.

soma = 0

para $i = 1$ até n **faça**

para $j = 1$ até n **faça**

soma = *soma* + $A[i, j]$

fim do para

fim do para

escreva (“A soma total dos elementos na matriz é”, *soma*)

4. O algoritmo a seguir soma todos os elementos na parte “triangular superior” de uma matriz quadrada A $n \times n$. Analise esse algoritmo, em que a unidade de trabalho é a operação de soma.

soma = 0

para $k = 1$ até n **faça**

para $j = k$ até n **faça**

soma = *soma* + $A[k, j]$

fim do para

fim do para

escreva (“A soma total dos elementos na parte triangular superior da matriz é”, *soma*)

5. Analise o algoritmo a seguir, em que a unidade de trabalho é o comando que gera a saída. Suponha que $n = 2^m$ para algum inteiro positivo m .

inteiros j, k

para $k = 1$ até n **faça**

$j = n$;

enquanto $j \geq 2$ **faça**

 escreva

$$j \cdot j = j/2$$

fim do enquanto

fim do para

6. Analise o algoritmo a seguir, em que a unidade de trabalho é o comando que gera a saída. (*Sugestão:* Um dos exercícios na Seção 2.2 pode ajudar.)

inteiro i

real d, x ;

para $i = 1$ até n **faça**

$d = 1.0/i$;

$x = i$;

enquanto $x > 0$ **faça**

escreva x

$x = x - d$;

fim do enquanto

fim do para

Os Exercícios 7 e 8 envolvem $n! = n(n-1)(n-2) \cdot \dots \cdot 1$.

7. a. Escreva o corpo de uma função iterativa para calcular $n!$ para $n \geq 1$.
b. Analise essa função, em que a unidade de trabalho é a operação de multiplicação.
8. a. Escreva uma função recursiva para calcular $n!$ para $n \geq 1$.
b. Escreva uma relação de recorrência para o trabalho executado por essa função, em que a unidade de trabalho é a operação de multiplicação.
c. Resolva a relação de recorrência do item (b).
d. Compare sua resposta no item (c) com seu resultado no Exercício 7b.

Os Exercícios 9 e 10 envolvem o cálculo de um polinômio $a_n x^n + a_{n-1} x^{n-1} + \dots + a_0$ em um valor específico de x .

9. Um algoritmo direto para calcular um polinômio é dado pela seguinte função:

Poli(real a_n , real a_{n-1} , ..., real a_0 , real c , inteiro n)

//calcula o polinômio $a_n x^n + a_{n-1} x^{n-1} + \dots + a_0$ em $x = c$

Variáveis locais:

inteiro i

real $soma = a_0$

real $produto = 1$

para $i = 1$ até n **faça**

$produto = produto * c$

$soma = soma + a_i * produto$

fim do para

retorne $soma$

fim da função *Poli*

- a. Siga esse algoritmo para calcular o valor de $2x^3 - 7x^2 + 5x - 14$ para $x = 4$.
 - b. O algoritmo envolve tanto somas quanto multiplicações; analise esse algoritmo, em que essas operações são as unidades de trabalho.
10. Uma alternativa ao algoritmo de cálculo de um polinômio dado no Exercício 9 é um algoritmo chamado de *método de Horner*. O método de Horner baseia-se em uma expressão diferente para um polinômio, por exemplo,

$$2x^3 - 7x^2 + 5x - 14 = -14 + x(5 + x(-7 + x(2)))$$

Horner(real a_n , real a_{n-1} , ..., real a_0 , real c , inteiro n)

//calcula o polinômio $a_n x^n + a_{n-1} x^{n-1} + \dots + a_0$ em $x = c$

//usando o método de Horner

Variáveis locais:

inteiro i

real $resultado = a_n$

para $i = 1$ até n **faça**

$resultado = resultado * c + a_{n-i}$

fim do para

retorna $resultado$

fim da função Horner

- a. Siga esse algoritmo para calcular o valor de $2x^3 - 7x^2 + 5x - 14$ para $x = 4$.
 - b. Analise esse algoritmo, em que as operações de somas e multiplicações são as unidades de trabalho.
 - c. Ao calcular um polinômio de grau $n = 98$ em algum valor de x , quantas operações foram economizadas ao se usar o método de Horner em vez do método do Exercício 9?
11. Para o algoritmo do Exemplo 27, conte o número total de atribuições e de comparações feitas no melhor caso (trabalho mínimo) e no pior caso (trabalho máximo); descreva cada um desses casos.
12. a. Escreva uma função para converter uma cadeia binária $b_n b_{n-1} \dots b_1 b_0$ em seu equivalente decimal.
- b. Teste sua função na cadeia binária 10011.
- c. Descreva o pior caso para esse algoritmo e encontre o número de multiplicações e somas executadas neste caso.
- d. Descreva o melhor caso para este algoritmo e encontre o número de multiplicações e somas executadas neste caso.

Os Exercícios 13 e 14 estão relacionados com um algoritmo de ordenação chamado de *OrdenaçãoPorBolhas*.

13. O algoritmo *OrdenaçãoPorBolhas* funciona percorrendo uma lista diversas vezes; em cada passagem pela lista, elementos adjacentes fora de ordem são permutados. Ao final da primeira passagem, o elemento máximo foi “como uma bolha” para o final da lista e não participa das passagens subsequentes. O algoritmo a seguir é chamado inicialmente com $j = n$.

OrdenaçãoPorBolhas(lista L ; inteiro j)

//ordena recursivamente os itens de 1 até j na lista L em ordem crescente

se $j = 1$ **então**

a ordenação está completa, escreva a lista ordenada

senão

para $i = 1$ até $j - 1$ **faça**

se $L[i] > L[i + 1]$ **então**

permuta $L[i]$ e $L[i + 1]$

fim do se

fim do para

OrdenaçãoPorBolhas($L, j - 1$)

fim do se

fim da função *OrdenaçãoPorBolhas*

- a. Siga o algoritmo *OrdenaçãoPorBolhas* para ordenar a lista 5, 6, 3, 4, 8, 2.
 - b. Escreva uma relação de recorrência para o número de comparações feitas entre elementos da lista por esse algoritmo para ordenar uma lista com n elementos.
 - c. Resolva essa relação de recorrência.
14. No algoritmo *OrdenaçãoPorBolhas*, suponha que incluímos a permutação de elementos na lista como uma unidade de trabalho, além das comparações entre elementos da lista.
- a. Descreva o pior caso e encontre o número de comparações e permutações feitas neste caso.
 - b. Descreva o melhor caso e encontre o número de comparações e permutações feitas neste caso.
 - c. Suponha que, em média, permutações de elementos têm que ser feitas durante metade do tempo. Encontre o número de comparações e permutações feitas neste caso.

Os Exercícios 15 a 18 se referem ao algoritmo *OrdenaçãoPorSeleção* da Seção 3.1.

15. Em uma parte do algoritmo *OrdenaçãoPorSeleção*, é preciso encontrar o índice do item máximo em uma lista. Isso requer comparações entre elementos da lista. Em uma lista (não ordenada) com n elementos, quantas comparações são necessárias, no pior caso, para encontrar o elemento máximo? Quantas comparações são necessárias em média?
16. Definindo a operação básica como a comparação dos elementos na lista e ignorando o trabalho necessário para as operações de permutação dos elementos na lista, escreva uma relação de recorrência para a quantidade de operações executadas pela ordenação por seleção em uma lista com n elementos. (*Sugestão*: Use o resultado do Exercício 15.)
17. Resolva a relação de recorrência do Exercício 16.
18. Suponha que a permutação de $L[i]$ e $L[j]$ acontece mesmo quando $i = j$. Escreva uma expressão para o número total de comparações e permutações para ordenar uma lista de n elementos.

Os Exercícios 19 a 24 estão relacionados com um algoritmo recursivo de ordenação chamado *OrdenaçãoPorFusão*, que pode ser descrito da seguinte maneira: uma lista com um elemento já está ordenada, não há necessidade de fazer nada; se a lista tiver mais de um elemento, divida-a pela metade, ordene cada metade usando *OrdenaçãoPorFusão* (esta é a parte recursiva) e depois combine as duas listas em uma única lista ordenada.

19. A parte de fusão do algoritmo *OrdenaçãoPorFusão* requer que se comparem os elementos de cada uma das listas ordenadas para ver qual o próximo elemento na lista conjunta ordenada. Quando acabam os elementos de uma das listas, os elementos restantes na outra podem ser adicionados sem outras comparações. Dados os pares de listas a seguir, combine-as e conte o número de comparações feitas para fundi-las em uma única lista ordenada.
 - a. 6, 8, 9 e 1, 4, 5
 - b. 1, 5, 8 e 2, 3, 4
 - c. 0, 2, 3, 4, 7, 10 e 1, 8, 9
20. Sob que circunstâncias será necessário executar o número máximo de comparações ao se fundir duas listas ordenadas? Se as duas listas tiverem comprimento r e s , qual será o número máximo de comparações?
21. Escreva uma relação de recorrência para o número de comparações entre os elementos das listas efetuadas pelo algoritmo *OrdenaçãoPorFusão* no pior caso. Suponha que $n = 2^m$.
22. Resolva a relação de recorrência do Exercício 21.
23. Use os resultados dos Exercícios 18 e 22 para comparar o comportamento, no pior caso, dos algoritmos *OrdenaçãoPorSeleção* (contando comparações e permutações) e *OrdenaçãoPorFusão* (contando comparações) para $n = 4, 8, 16$ e 32 (use uma calculadora ou uma planilha).
24. Use os resultados dos Exercícios 14 e 22 para comparar o comportamento, no pior caso, dos algoritmos *OrdenaçãoPorBolhas* (contando comparações e permutações) e *OrdenaçãoPorFusão* (contando comparações) para $n = 4, 8, 16$ e 32 (use uma calculadora ou uma planilha).

Os Exercícios 25 a 34 estão relacionados com um algoritmo recursivo de ordenação chamado de *OrdenaçãoRápida*, que é descrito da seguinte maneira: uma lista com um elemento já está ordenada, não há necessidade de fazer nada. Caso contrário, considere o primeiro elemento da lista, chame-o de pivô, e percorra a lista original para criar duas listas novas, L_1 e L_2 . L_1 consiste em todos os elementos que são menores do que o elemento pivô, e L_2 consiste em todos os elementos que são maiores do que o elemento pivô. Coloque o elemento pivô entre L_1 e L_2 . Ordene as duas listas L_1 e L_2 usando *OrdenaçãoRápida* (essa é a parte recursiva). Finalmente todas as listas consistirão em listas de 1 elemento separadas por elementos pivôs anteriores, e, nesse ponto, a lista original estará ordenada. Isso é um pouco confuso, de modo que vamos ver um exemplo; os elementos pivôs estão entre colchetes.

Lista original: 6, 2, 1, 7, 9, 4, 8

Depois da 1ª passagem: 2, 1, 4, [6], 7, 9, 8

Depois da 2ª passagem: 1, [2], 4, [6], [7], 9, 8

Depois da 3ª passagem: 1, [2], 4, [6], [7], 8, [9] Ordenada

25. Ilustre a *OrdenaçãoRápida* como acima usando a lista 9, 8, 3, 13
26. Ilustre a *OrdenaçãoRápida* como acima usando a lista 8, 4, 10, 5, 9, 6, 14, 3, 1, 12, 11.
27. Quantas comparações entre elementos da lista são necessárias para a 1ª passagem de *OrdenaçãoRápida* na lista do exemplo?

28. Quantas comparações entre elementos da lista são necessárias para a 1ª passagem de *OrdenaçãoRápida* em uma lista com n elementos?
29. Suponha que, em cada passagem, cada elemento pivô divide sua sublista em duas sublistas de mesmo comprimento, cada uma com o tamanho aproximado de metade da sublista (o que, de fato, é muito difícil de obter). Escreva uma relação de recorrência para o número de comparações entre elementos da lista neste caso.
30. Resolva a relação de recorrência encontrada no Exercício 29.
31. Suponha que, em cada passagem, cada elemento pivô divide sua sublista (que tem k elementos) em uma lista vazia e uma lista de tamanho $k - 1$. Escreva uma relação de recorrência para o número de comparações entre elementos da lista neste caso.
32. Resolva a relação de recorrência encontrada no Exercício 31.
33. Ao contrário da situação descrita no Exercício 29, em que cada elemento pivô divide a sublista pela metade para a próxima passagem, a situação descrita no Exercício 31 pode ocorrer facilmente. Descreva uma característica da lista original que causaria essa situação.
34. O Exercício 29 descreve o melhor caso de *OrdenaçãoRápida* e o Exercício 31 descreve o pior caso de *OrdenaçãoRápida* em relação à comparação entre elementos da lista.
- Em relação ao número de comparações necessárias, qual dos algoritmos de ordenação (*OrdenaçãoPorSeleção*, *OrdenaçãoPorBolhas*, *OrdenaçãoPorFusão*) é comparável ao melhor caso de *OrdenaçãoRápida*?
 - Em relação ao número de comparações necessárias, qual dos algoritmos de ordenação (*OrdenaçãoPorSeleção*, *OrdenaçãoPorBolhas*, *OrdenaçãoPorFusão*) é comparável ao pior caso de *OrdenaçãoRápida*?

Os Exercícios 35 e 36 referem-se ao algoritmo *BuscaSequencial*. Não é difícil fazer uma análise do caso médio do algoritmo de busca sequencial sob determinadas hipóteses. Dada uma lista com n elementos e um item x a ser procurado, a operação básica é a comparação de elementos na lista com x ; portanto, uma análise deveria contar quantas vezes é efetuada tal operação “em média”. A definição de “média” depende de nossas hipóteses.

35. Suponha que x pertence à lista e é igualmente provável de ser encontrado em qualquer das n posições na lista. Preencha o restante da tabela a seguir, fornecendo o número de comparações em cada caso.

Posição em que Ocorre x	Número de Comparações
1	1
2	
3	
\vdots	
n	

Encontre o número médio de comparações somando os resultados na tabela e dividindo por n . (*Sugestão*: Veja o Problema Prático 7 na Seção 2.2 — dissemos que você deveria se lembrar disso!)

36. Encontre o número médio de comparações sob a hipótese de que x é igualmente provável de ser encontrado em qualquer das n posições na lista ou de não estar na lista.

Os Exercícios 37 a 40 procuram uma cota superior melhor para o número de divisões necessárias para o algoritmo de Euclides encontrar $\text{mdc}(a, b)$. Suponha que a e b são inteiros positivos e que $a > b$.

37. Suponha que são necessárias m divisões para encontrar o $\text{mdc}(a, b)$. Prove, por indução, que, para $m \geq 1$, $a \geq F(m + 2)$ e $b \geq F(m + 1)$, em que $F(n)$ é a sequência de Fibonacci. (*Sugestão*: Para encontrar o $\text{mdc}(a, b)$, o algoritmo calcula $\text{mdc}(b, r)$ depois da primeira divisão.)
38. Suponha que são necessárias m divisões para encontrar o $\text{mdc}(a, b)$, com $m \geq 4$. Prove que

$$\left(\frac{3}{2}\right)^{m+1} < F(m+2) \leq a$$

(Sugestão: Use o resultado do Exercício 37 acima e o do Exercício 26 da Seção 3.1.)

39. Suponha que são necessárias m divisões para encontrar o $\text{mdc}(a, b)$, com $m \geq 4$. Prove que $m < (\log_{1,5} a) - 1$.

(Sugestão: Use o resultado do Exercício 38.)

40. a. Calcule o $\text{mdc}(89, 55)$ e conte o número de divisões necessárias.
 b. Calcule uma cota superior para o número de divisões necessárias para calcular o $\text{mdc}(89, 55)$ usando a Equação (1).
 c. Calcule uma cota superior para o número de divisões necessárias para calcular o $\text{mdc}(89, 55)$ usando o resultado do Exercício 39.
 d. O matemático francês do século XVIII Gabriel Lamé provou que 5 vezes o número de dígitos decimais em b é uma cota superior para o número de divisões executadas pelo algoritmo de Euclides para encontrar $\text{mdc}(a, b)$, em que $a > b$. Calcule uma cota superior para o número de divisões necessárias para calcular o $\text{mdc}(89, 55)$ usando o teorema de Lamé.

CAPÍTULO 3 REVISÃO

TERMINOLOGIA

algoritmo de busca binária
 algoritmo de busca sequencial
 algoritmo de ordenação por seleção
 algoritmo do tipo dividir para conquistar
 análise de algoritmo
 cadeia binária
 cadeia vazia
 concatenação
 cota superior
 definição por indução
 definição por recorrência
 equação característica de uma relação de recorrência
 forma de Backus-Naur (FBN)
 índice do somatório
 indução estrutural
 notação de somatório
 palíndromo
 relação de recorrência
 relação de recorrência com coeficientes constantes
 relação de recorrência de primeira ordem
 relação de recorrência de segunda ordem
 relação de recorrência do tipo dividir para conquistar
 relação de recorrência homogênea
 relação de recorrência linear
 resolução de uma relação de recorrência
 sequência (sequência infinita)
 sequência de Fibonacci
 solução em forma fechada

AUTOTESTE

Responda se as afirmações a seguir são verdadeiras ou falsas sem consultar o capítulo.

Seção 3.1

1. Uma sequência definida por

$$S(1) = 7$$

$$S(n) = 3S(n-1) + 2 \quad \text{para } n \geq 2$$

contém o número 215.

2. Uma coleção T de números é definida por recorrência por

1. 6 e 8 pertencem a T .

2. Se X e Y pertencerem a T , então $X + 2Y$ também pertencerá.

Todo número par ≥ 18 pertence a T .

3. Algoritmos recorrentes são valiosos principalmente porque são mais eficientes do que algoritmos iterativos.

4. No algoritmo recursivo *OrdenaçãoPorSeleção*, a mudança de uma linha do algoritmo

“encontre o índice i do item mínimo em L entre 1 e j ”

fará com que a lista L seja ordenada em ordem decrescente.

5. Ao se aplicar o algoritmo de busca binária à lista

2, 5, 7, 10, 14, 20

em que $x = 8$ é o item procurado, x nunca é comparado a 5.

Seção 3.2

1. Uma solução em forma fechada de uma relação de recorrência é obtida aplicando-se indução matemática à relação de recorrência.

2. $S(n) = 2S(n-1) + 3S(n-2) + 5n$ é uma relação de recorrência linear de primeira ordem com coeficientes constantes.

3. $S(n) = c^{n-1}S(1) + \sum_{i=2}^n c^{n-i}g(i)$ é uma solução em forma fechada de qualquer relação de recorrência linear de primeira ordem com coeficientes constantes.

4. A solução da relação de recorrência $S(n) = c_1S(n-1) + c_2S(n-2)$ envolve a resolução da equação característica $t^2 - c_1t - c^2 = 0$.

5. Algoritmos do tipo dividir para conquistar levam a relações de recorrência que não são de primeira ordem.

Seção 3.3

1. A análise de um algoritmo encontra, em geral, a quantidade de operações efetuadas no pior caso porque é muito difícil analisar um caso médio.

2. No pior caso, o algoritmo que busca um padrão em um texto necessita de $n + m$ comparações, em que n é o tamanho do texto e m é o tamanho do padrão.

3. A busca binária é mais eficiente do que a busca sequencial em uma lista ordenada com mais de três elementos.

4. A versão recursiva do algoritmo de busca sequencial é um algoritmo do tipo dividir para conquistar.

5. Uma cota superior para o algoritmo de Euclides é um valor superior para a quantidade de divisões efetuadas para calcular o $\text{mdc}(a, b)$.

NO COMPUTADOR

Para os Exercícios 1 a 7, escreva um programa que produza a saída desejada a partir dos dados de entrada fornecidos.

1. *Entrada*: Cadeia binária

Saída: Mensagem indicando se a cadeia de entrada é um palíndromo (veja o Problema Prático 7)

Algoritmo: Use recorrência.

2. *Entrada:* Cadeia de caracteres x e um inteiro positivo n

Saída: Concatenação de n cópias de x

Algoritmo: Use recorrência.

(Algumas linguagens de programação já vêm com capacidade de manipulação de cadeias, como a concatenação.)

3. *Entrada:* Inteiro positivo n

Saída: O n -ésimo valor em uma sequência de Fibonacci usando

a. iteração

b. recorrência

Insira agora um contador em cada versão para indicar o número total de adições efetuadas. Execute cada versão para diversos valores de n e coloque em um único gráfico o número de adições em função de n para cada versão.

4. *Entrada:* Dois inteiros positivos a e b com $a > b$

Saída: $\text{mdc}(a, b)$ usando

a. a versão iterativa do algoritmo de Euclides

b. a versão recursiva do algoritmo de Euclides

5. *Entrada:* Lista não ordenada de 10 inteiros.

Saída: A lista de entrada ordenada em ordem crescente

Algoritmo: Use o algoritmo recursivo de ordenação por seleção do Exemplo 12.

6. *Entrada:* Lista ordenada de 10 inteiros e um inteiro x

Saída: Mensagem indicando se x pertence à lista

Algoritmo: Use o algoritmo de busca binária do Exemplo 13.

7. *Entrada:* Cadeia de texto, cadeia (padrão) a ser encontrada

Saída: Localização do início da cadeia (padrão) no texto, ou uma mensagem dizendo que a cadeia procurada não se encontra no texto

Algoritmo: Veja o Exemplo 28.

8. *Entrada:* O valor $(1 + \sqrt{5})/2$ conhecido como *razão áurea*, está relacionado com a sequência de Fibonacci por

$$\lim_{n \rightarrow \infty} \frac{F(n+1)}{F(n)} = \frac{1 + \sqrt{5}}{2}$$

Verifique esse limite calculando $F(n+1)/F(n)$ para $n = 10, 15, 25, 50, 100$ e comparando os resultados com a razão áurea.

9. Compare o número de operações efetuadas pelos algoritmos de busca sequencial e busca binária em uma lista ordenada com n elementos calculando n e $1 + \log n$ para valores de n de 1 a 100. Apresente os resultados em forma gráfica.