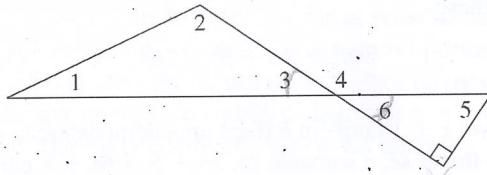


38. A soma de quaisquer três inteiros consecutivos é par.
39. O produto de um inteiro e seu quadrado é par.
40. A soma de um inteiro e seu cubo é par.
41. Qualquer inteiro positivo pode ser escrito como a soma dos quadrados de dois inteiros.
42. Para um inteiro positivo x , $x + \frac{1}{x} \geq 2$.
43. Para todo número primo n , $n + 4$ é primo.
44. Para um inteiro positivo n , $n > 2$, $n^2 - 1$ não é primo.
45. Para todo inteiro positivo n , $n^2 + n + 1$ é primo.
46. Para todo inteiro positivo n , $2^n + 1$ é primo.
47. Para n um inteiro par, $n > 2$, $2^n - 1$ não é primo.
48. O produto de dois números racionais é racional.
49. A soma de dois números racionais é racional.
50. O produto de dois números irracionais é irracional.
51. A soma de um número racional com um número irracional é irracional.

Para os Exercícios 52 a 54, use a figura abaixo e os seguintes fatos de geometria:

- ❖ A soma dos ângulos internos de um triângulo é 180° .
- ❖ Ângulos opostos formados pela interseção de duas retas são iguais.
- ❖ Um ângulo onde as duas semi-retas que o formam estão contidas na mesma reta, mas têm sentidos opostos, mede 180° .
- ❖ Um ângulo reto mede 90° .



52. Prove que a medida do ângulo 4 é a soma das medidas dos ângulos 1 e 2.
53. Prove que a medida do ângulo 5 mais a medida do ângulo 3 é 90° .
54. Se os ângulos 1 e 5 são iguais, então o ângulo 2 é reto.
55. Prove que a soma dos inteiros de 1 a 100 é 5050. (Sugestão: Ao invés de somar todos esses números, tente fazer a mesma observação engenhosa que o matemático alemão Karl Frederick Gauss (1777-1855) fez quando criança na escola: agrupe os números em pares, usando 1 e 100, 2 e 99, etc.)

SEÇÃO 2.2 INDUÇÃO

PRIMEIRO PRINCÍPIO DE INDUÇÃO

Existe uma última técnica de demonstração particularmente útil em ciência da computação. Para ilustrar como ela funciona, imagine que você está subindo uma escada infinitamente alta. Como você sabe se será capaz de chegar a um degrau arbitrariamente alto? Suponha que você faça as seguintes hipóteses sobre sua capacidade de subir:

1. Você consegue alcançar o primeiro degrau.
2. Uma vez chegando a um degrau, você sempre é capaz de chegar ao próximo. (Note que essa asserção é um condicional.)

Se a proposição 1 e o condicional 2 são ambos verdadeiros, então, pela proposição 1, você consegue chegar no primeiro degrau e, portanto, pela proposição 2, consegue chegar no segundo; novamente pela proposição 2, você consegue chegar no terceiro degrau; mais uma vez pela proposição 2, você consegue chegar no quarto degrau; e assim por diante. Você pode subir tão alto quanto quiser. Ambas as hipóteses são necessárias. Se apenas a primeira proposição fosse verdadeira, você não teria nenhuma garantia de passar do primeiro degrau e, se apenas a segunda fosse verdadeira, você poderia não ser capaz de começar nunca. Vamos supor que os degraus da escada estejam numerados pelos inteiros positivos — 1, 2, 3, etc. Agora pense sobre uma propriedade específica que um número possa ter. Ao invés de “chegar a um degrau arbi-

trariamente alto”, podemos falar sobre um inteiro positivo arbitrário tendo essa propriedade. Vamos usar a notação $P(n)$ para dizer que o inteiro positivo n tem a propriedade P . Como usar a mesma técnica que usamos para subir a escada para provar que, qualquer que seja o inteiro positivo n , temos $P(n)$? As duas proposições que precisamos provar são

1. $P(1)$ (1 tem a propriedade P)
2. Para qualquer inteiro positivo k , $P(k) \rightarrow P(k + 1)$ (Se qualquer número tem a propriedade P , o próximo também tem.)

Se pudermos provar ambas as proposições 1 e 2, então $P(n)$ é válida para qualquer inteiro positivo n , da mesma forma que você poderia subir até um degrau arbitrário da escada.

O fundamento para argumentos desse tipo é o primeiro princípio de indução matemática.

Primeiro Princípio de Indução Matemática

1. $P(1)$ é verdade
 2. $(\forall k)[P(k) \text{ verdade} \rightarrow P(k + 1) \text{ verdade}]$
- $\left. \begin{array}{l} \\ \end{array} \right\} \rightarrow P(n) \text{ é verdade para todo inteiro positivo } n$

LEMBRETE:

Para provar que alguma coisa é verdade para todo inteiro $n \geq$ que algum valor, pense em indução.

O primeiro princípio de indução matemática é um condicional. A conclusão é uma proposição da forma “ $P(n)$ é verdade para todo inteiro positivo n ”. Portanto, sempre que quisermos provar que alguma coisa é verdade para todo inteiro positivo n , é bastante provável que a indução matemática seja uma técnica apropriada.

Para mostrar que a conclusão desse condicional é verdadeira, precisamos provar que as duas hipóteses, 1 e 2, são verdadeiras. Para provar a proposição 1, basta mostrar que o número 1 tem a propriedade P , geralmente uma tarefa trivial. A proposição 2 é um condicional que tem que ser válido para todo k . Para provar esse condicional, suponha que $P(k)$ é verdade para um inteiro positivo k e mostre, baseado nessa hipótese, que $P(k + 1)$ é verdade. Você deve se convencer de que supor que o número k tem a propriedade P não é a mesma coisa que supor o que queremos provar (essa é uma confusão comum na primeira vez que se encontra uma demonstração desse tipo). Essa é, simplesmente, a maneira de proceder para obter uma demonstração direta do condicional $P(k) \rightarrow P(k + 1)$.

Ao fazer uma demonstração por indução, o estabelecimento da veracidade da proposição 1 é chamado de **base da indução ou passo básico** da demonstração por indução. O estabelecimento da veracidade de $P(k) \rightarrow P(k + 1)$ é o **passo indutivo**. Quando supomos que $P(k)$ é verdade para provar o passo indutivo, $P(k)$ é chamada de **hipótese de indução**.

Todos os métodos de demonstração de que falamos neste capítulo são técnicas para o raciocínio dedutivo — maneiras de provar uma conjectura que talvez tenha sido formulada por um raciocínio indutivo. A indução matemática também é uma técnica *dedutiva* e não um método de raciocínio *indutivo* (não se confunda com a terminologia usada). Nas outras técnicas de demonstração, podemos começar com uma hipótese e juntar diversos fatos até que “tropeçamos” na conclusão. De fato, mesmo que a nossa conjectura esteja ligeiramente incorreta, podemos ver qual deve ser a conclusão correta ao fazer a demonstração. Na indução matemática, no entanto, precisamos saber, desde o início, qual é a forma exata da propriedade $P(n)$ que queremos estabelecer. A indução matemática, portanto, não é uma técnica de demonstração exploratória — pode apenas confirmar uma conjectura correta.

DEMONSTRAÇÕES POR INDUÇÃO MATEMÁTICA

Suponha que um ancestral Silva casou e teve dois filhos. Vamos chamar esses dois filhos de geração 1. Suponha, agora, que cada um desses filhos teve dois filhos; então, a geração 2 contém quatro descendentes. Isso continua de geração em geração. A árvore genealógica da família Silva, portanto, tem a forma ilustrada na Fig. 2.2. (Ela é exatamente igual à Fig. 1.1b, onde obtivemos todos os valores lógicos possíveis para n letras de proposição.)

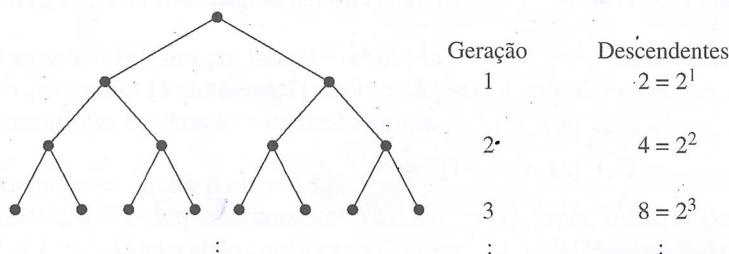


Fig. 2.2

Parece que a geração n contém 2^n descendentes. Mais formalmente, se denotarmos por $P(n)$ o número de descendentes em cada geração, nossa conjectura é que

$$P(n) = 2^n$$

Podemos usar indução para *provar* que nossa conjectura para $P(n)$ está correta.

O passo básico é estabelecer $P(1)$, que é a equação

$$P(1) = 2^1 = 2$$

Isso é verdade, pois nos foi dito que Silva teve dois filhos. Vamos supor, agora, que nossa conjectura está correta para uma geração arbitrária k , $k \geq 1$, isto é, vamos supor que

$$P(k) = 2^k$$

e tentar mostrar que

$$P(k+1) = 2^{k+1}$$

Nessa família, cada descendente tem dois filhos, de modo que o número de descendentes na geração $k+1$ será o dobro do número de descendentes na geração k , ou seja, $P(k+1) = 2P(k)$. Pela hipótese de indução, $P(k) = 2^k$, logo

$$P(k+1) = 2P(k) = 2(2^k) = 2^{k+1}$$

e, de fato,

$$P(k+1) = 2^{k+1}$$

Isso completa nossa demonstração. Agora que estamos tranqüilos sobre o clã dos Silva, podemos aplicar o método de demonstração por indução a problemas menos óbvios.

Prove que a equação

♦ EXEMPLO 14

$$1 + 3 + 5 + \dots + (2n - 1) = n^2 \quad (1)$$

é verdadeira para qualquer inteiro positivo n . A propriedade $P(n)$ aqui é que a Eq. (1) é válida. O termo à esquerda do sinal de igualdade é a soma de todos os inteiros ímpares de 1 até $2n - 1$. Embora possamos verificar a veracidade dessa equação para qualquer valor particular de n substituindo esse valor na equação, não podemos substituir n por todos os inteiros positivos que existem. Assim, uma demonstração por exaustão não funciona. Uma demonstração por indução é apropriada.

O passo básico é estabelecer $P(1)$, que é a Eq. (1) quando n tem o valor 1, ou seja,

$$P(1): \quad 1 = 1^2$$

Isso é certamente verdade. Para a hipótese de indução, vamos supor $P(k)$ para um inteiro positivo arbitrário k , que é a Eq. (1) quando n tem o valor k , isso é,

$$P(k): \quad 1 + 3 + 5 + \dots + (2k - 1) = k^2 \quad (2)$$

(Note que $P(k)$ *não* é a equação $(2k - 1) = k^2$, que só é verdade para $k = 1$.) Usando a hipótese de indução, queremos mostrar $P(k+1)$, que é a Eq. (1) quando n assume o valor $k+1$, ou seja,

$$P(k+1): \quad 1 + 3 + 5 + \dots + [2(k+1) - 1] = ? \quad (3)$$

(O ponto de interrogação em cima do sinal de igualdade é para nos lembrar de que é esse fato que queremos provar, ao invés de ser alguma coisa que já sabemos.)

A chave de uma demonstração por indução é encontrar um modo de relacionar o que queremos saber — $P(k+1)$, Eq. (3) — e o que supusemos — $P(k)$, Eq. (2). O lado esquerdo de $P(k+1)$ pode ser reescrito mostrando-se a penúltima parcela:

$$1 + 3 + 5 + \dots + (2k - 1) + [2(k+1) - 1]$$

Essa expressão contém o termo à esquerda do sinal de igualdade na Eq. (2). Como estamos supondo que $P(k)$ é válida, podemos substituir esse termo pelo termo à direita do sinal de igualdade na Eq. (2). Obtemos, então,

$$\begin{aligned} & 1 + 3 + 5 + \dots + [2(k+1) - 1] \\ &= 1 + 3 + 5 + \dots + (2k - 1) + [2(k+1) - 1] \\ &= k^2 + [2(k+1) - 1] \\ &= k^2 + [2k + 2 - 1] \\ &= k^2 + 2k + 1 \\ &= (k+1)^2 \end{aligned}$$

Portanto,

$$1 + 3 + 5 + \dots + [2(k+1) - 1] = (k+1)^2$$

o que mostra a validade de $P(k+1)$ provando, assim, que a Eq. (1) é verdadeira para qualquer inteiro positivo n . \diamond

A Tabela 2.3 resume os três passos necessários para uma demonstração usando o primeiro princípio de indução.

TABELA 2.3

Demonstração usando o primeiro princípio de indução

Passo 1	Prove a base da indução.
Passo 2	Suponha $P(k)$.
Passo 3	Prove $P(k+1)$.

EXEMPLO 15 Prove que

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

para todo $n \geq 1$.

Novamente, indução é apropriada. $P(1)$ é a equação

$$1 + 2 = 2^{1+1} - 1 \quad \text{ou} \quad 3 = 2^2 - 1$$

que é verdadeira. Vamos considerar $P(k)$

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

como a hipótese de indução e tentar estabelecer $P(k+1)$:

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

Novamente, reescrevendo a soma à esquerda do sinal de igualdade de $P(k+1)$, vemos como a hipótese de indução pode ser usada:

$$\begin{aligned} 1 + 2 + 2^2 + \dots + 2^{k+1} \\ &= 1 + 2 + 2^2 + \dots + 2^k + 2^{k+1} \\ &= 2^{k+1} - 1 + 2^{k+1} \quad (\text{pela hipótese de indução } P(k)) \\ &= 2(2^{k+1}) - 1 \\ &= 2^{k+1+1} - 1 \end{aligned}$$

Portanto,

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

o que mostra $P(k+1)$, concluindo a demonstração. \diamond

PROBLEMA PRÁTICO 7

Prove que, para qualquer inteiro positivo n ,

$$1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2}$$

Nem todas as demonstrações por indução envolvem somas. Outras identidades algébricas sobre os inteiros positivos podem ser demonstradas por indução, assim como proposições não algébricas, como o número de descendentes na geração n da família Silva.

EXEMPLO 16 Prove que, para qualquer inteiro positivo n , $2^n > n$.

$P(1)$ é a proposição $2^1 > 1$, que certamente é verdade. Vamos supor, agora, $P(k)$, $2^k > k$, e tentar concluir $P(k+1)$, $2^{k+1} > k+1$. Começando com a expressão, em $P(k+1)$, à esquerda da desigualdade, observa-

mos que $2^{k+1} = 2^k \cdot 2$. Usando a hipótese de indução $2^k > k$ e multiplicando os dois membros dessa desigualdade por 2, obtemos $2^k \cdot 2 > k \cdot 2$. Completando o argumento,

$$2^{k+1} = 2^k \cdot 2 > k \cdot 2 = k + k \geq k + 1$$

Ou seja,

$$2^{k+1} > k + 1$$

Prove que, para qualquer inteiro positivo n , o número $2^{2n} - 1$ é divisível por 3.

O passo básico é mostrar $P(1)$, isto é, que $2^{2(1)} - 1 = 4 - 1 = 3$ é divisível por 3. Isso é evidente.

Vamos supor que $2^{2k} - 1$ é divisível por 3, o que significa que $2^{2k} - 1 = 3m$ para algum inteiro m , ou seja, $2^{2k} = 3m + 1$. Queremos mostrar que $2^{2(k+1)} - 1$ é divisível por 3.

$$\begin{aligned} 2^{2(k+1)} - 1 &= 2^{2k+2} - 1 \\ &= 2^2 \cdot 2^{2k} - 1 \\ &= 2^2(3m + 1) - 1 \quad (\text{pela hipótese de indução}) \\ &= 12m + 4 - 1 \\ &= 12m + 3 \\ &= 3(4m + 1) \quad \text{onde } 4m + 1 \text{ é inteiro} \end{aligned}$$

Portanto, $2^{2(k+1)} - 1$ é divisível por 3.

Para o primeiro passo em uma demonstração por indução, pode ser apropriado começar em 0, ou em 2 ou 3, ao invés de 1. O mesmo princípio se aplica, independentemente do degrau onde você começa a subir na escada.

Prove que $n^2 > 3n$ para $n \geq 4$.

Devemos usar indução aqui, começando com a base da indução em $P(4)$. (Testando os valores $n = 1, 2$ e 3, pode-se mostrar que a desigualdade não é válida para esses valores.) $P(4)$ é a desigualdade $4^2 > 3(4)$, ou seja, $16 > 12$, que é verdadeira. A hipótese de indução é que $k^2 > 3k$, onde $k \geq 4$, e queremos mostrar que $(k+1)^2 > 3(k+1)$.

$$\begin{aligned} (k+1)^2 &= k^2 + 2k + 1 \\ &> 3k + 2k + 1 \quad (\text{pela hipótese de indução}) \\ &\geq 3k + 8 + 1 \quad (\text{pois } k \geq 4) \\ &> 3k + 3 \\ &= 3(k+1) \end{aligned}$$

Prove que $2^{n+1} < 3^n$ para todo $n > 1$.

Demonstrações supostamente por indução, mas que não o são de fato, também são possíveis. Quando demonstramos $P(k+1)$ sem usar $P(k)$, fizemos uma demonstração direta de $P(k+1)$, onde $k+1$ é arbitrário. Não é que a demonstração esteja errada, é só que ela deve ser reescrita como uma demonstração direta de $P(n)$ para qualquer n , e não como uma demonstração por indução.

Pode ser conveniente uma demonstração por indução mesmo em aplicações não tão óbvias como nos exemplos anteriores. Isso acontece, geralmente, quando alguma quantidade na proposição a ser demonstrada toma valores inteiros não-negativos arbitrários.

Uma linguagem de programação pode ser projetada com as seguintes convenções em relação à multiplicação: um único fator não necessita de parênteses mas o produto “ a vezes b ” precisa ser escrito na forma $(a)b$. Assim, o produto

$$a \cdot b \cdot c \cdot d \cdot e \cdot f \cdot g$$

poderia ser escrito nessa linguagem como

$$((((((a)b)c)d)e)f)g$$

EXEMPLO 17

EXEMPLO 18

PROBLEMA PRÁTICO 8

EXEMPLO 19

ou como, por exemplo,

$$((a)b)((c)d)(e)f)$$

dependendo da ordem em que se vai executar o produto. O resultado é o mesmo em qualquer dos casos.

Queremos mostrar que qualquer produto pode ser escrito com um número par de parênteses. A demonstração é por indução no número de fatores. Para um único fator, temos 0 parêntese, um número par. Suponha que, para qualquer produto com k fatores, temos um número par de parênteses. Agora considere um produto P com $k + 1$ fatores. P pode ser considerado como um produto de r vezes s , onde r tem k fatores e s é um único fator. Pela hipótese de indução, r tem um número par de parênteses. Podemos, então, escrever r vezes s como $(r)s$. Isso adiciona mais 2 parênteses ao número par de parênteses em r , dando a P um número par de parênteses. \diamond

EXEMPLO 20

Um problema de “ladrilhagem” fornece uma boa ilustração de indução em um contexto geométrico. Um *ângulo de ferro* é uma peça em forma de L cobrindo três quadrados em um tabuleiro quadriculado, como o de xadrez (veja a Fig. 2.3a). O problema é mostrar que, para qualquer inteiro positivo n , se removermos um quadrado de um tabuleiro originalmente com $2^n \times 2^n$ quadrados, ele pode ser ladrilhado — completamente recoberto — com ângulos de ferro.

A base da indução é $n = 1$, o que nos dá um tabuleiro com 2×2 quadrados. A Fig. 2.3b mostra a solução nesse caso se for removido o canto direito superior. A remoção de qualquer dos outros três quadrados funciona da mesma maneira. Suponha agora que qualquer tabuleiro $2^k \times 2^k$ com um quadrado removido pode ser ladrilhado usando-se ângulos de ferro. Vamos considerar um tabuleiro $2^{k+1} \times 2^{k+1}$. Precisamos mostrar que ele pode ser ladrilhado quando se remove um quadrado. Para relacionar o caso $k + 1$ com a hipótese de indução, divida o tabuleiro $2^{k+1} \times 2^{k+1}$ em quatro partes iguais. Cada parte é um tabuleiro $2^k \times 2^k$ e um deles vai ter um quadrado faltando (Fig. 2.3c). Pela hipótese de indução, esse tabuleiro pode ser ladrilhado. Retire um canto de cada uma das outras três partes como na Fig. 2.3d. Pela hipótese de indução, essas três partes com os quadrados removidos podem ser ladrilhadas. Como um único ângulo de ferro pode ser usado para cobrir esses três quadrados retirados, temos que o tabuleiro original $2^{k+1} \times 2^{k+1}$ com um dos quadrados removidos pode ser ladrilhado.

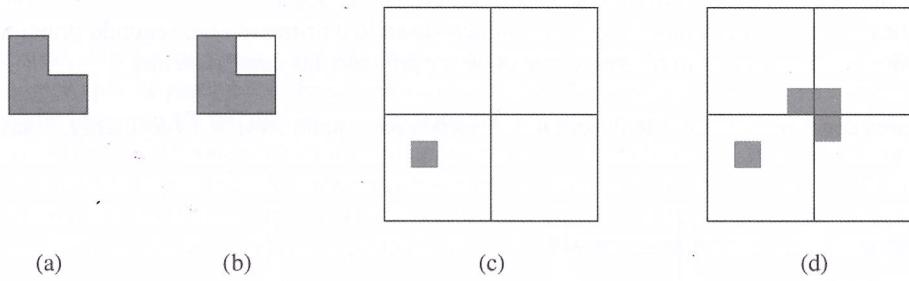


Fig. 2.3

O SEGUNDO PRINCÍPIO DE INDUÇÃO

Além do primeiro princípio de indução, que temos usado,

1. $P(1)$ é verdade
 2. $(\forall k)[P(k) \text{ verdade} \rightarrow P(k + 1) \text{ verdade}]$
- $\left. \begin{array}{l} 1. \\ 2. \end{array} \right\} \rightarrow P(n) \text{ verdade para todo inteiro positivo } n$

existe um segundo princípio de indução.

Segundo Princípio de Indução Matemática

- 1'. $P(1)$ é verdade
 - 2'. $(\forall r)[P(r) \text{ verdade para todo } r, 1 \leq r \leq k \rightarrow P(k + 1) \text{ verdade}]$
- $\left. \begin{array}{l} 1'. \\ 2'. \end{array} \right\} \rightarrow P(n) \text{ verdade para todo inteiro positivo } n$

Esses dois princípios de indução diferem nas proposições 2 e 2'. Na proposição 2, precisamos ser capazes de provar, para um inteiro positivo arbitrário k , que $P(k + 1)$ é verdadeira baseados apenas na hipótese de

que $P(k)$ é verdadeira. Na proposição 2', podemos supor que $P(r)$ é verdadeira para todos os inteiros r entre 1 e um inteiro positivo arbitrário k para provar $P(k+1)$. Isso parece nos dar muito mais “munição”, de modo que pode acontecer, algumas vezes, de sermos capazes de provar o condicional em 2' sem conseguir provar o condicional em 2.

O que nos permite deduzir $(\forall n)P(n)$ em cada caso? Veremos que os dois princípios, ou seja, os dois métodos de demonstração, são equivalentes. Em outras palavras, se aceitamos como válido o primeiro princípio, então o segundo também é válido, e reciprocamente. Para provar a equivalência entre os dois princípios, vamos considerar um outro princípio, que parece tão óbvio que não necessita discussão.

Princípio da Boa Ordenação Toda coleção de inteiros positivos que contém algum elemento tem um menor elemento.

Veremos que os seguintes condicionais são verdadeiros:

segundo princípio de indução \rightarrow primeiro princípio de indução

primeiro princípio de indução \rightarrow princípio da boa ordenação

princípio da boa ordenação \rightarrow segundo princípio de indução

Como consequência, todos os três princípios são equivalentes e aceitar qualquer um deles como verdadeiro significa aceitar os outros dois também.

Para provar que o segundo princípio de indução implica o primeiro, suponha que aceitamos o segundo princípio como sendo um argumento válido. Queremos mostrar, então, que o primeiro princípio é válido, isto é, que podemos concluir $P(n)$ para todo n das proposições 1 e 2. Se a proposição 1 é verdadeira, a proposição 1' também é. Se a proposição 2 é verdadeira, a proposição 2' também é, pois podemos dizer que concluímos $P(k+1)$ de $P(r)$ para todo r entre 1 e k , embora tenhamos usado apenas a condição $P(k)$. (De maneira mais precisa, a proposição 2' necessita que provemos que $P(1) \wedge P(2) \wedge \dots \wedge P(k) \rightarrow P(k+1)$; mas $P(1) \wedge P(2) \wedge \dots \wedge P(k) \rightarrow P(k)$ e, pela proposição 2, $P(k) \rightarrow P(k+1)$, logo $P(1) \wedge P(2) \wedge \dots \wedge P(k) \rightarrow P(k+1)$.) Pelo segundo princípio de indução podemos concluir $P(n)$ para todo n . As demonstrações de que o primeiro princípio de indução implica o princípio da boa ordenação, que por sua vez implica o segundo princípio de indução, são deixadas como exercício na Seção 3.1.

Para distinguir entre uma demonstração por indução usando o primeiro ou o segundo princípio, vamos considerar um exemplo um tanto pitoresco que pode ser provado das duas maneiras.

Prove que uma cerca reta com n esteios tem $n - 1$ seções para qualquer $n \geq 1$ (veja a Fig. 2.4a).

◆ EXEMPLO 21

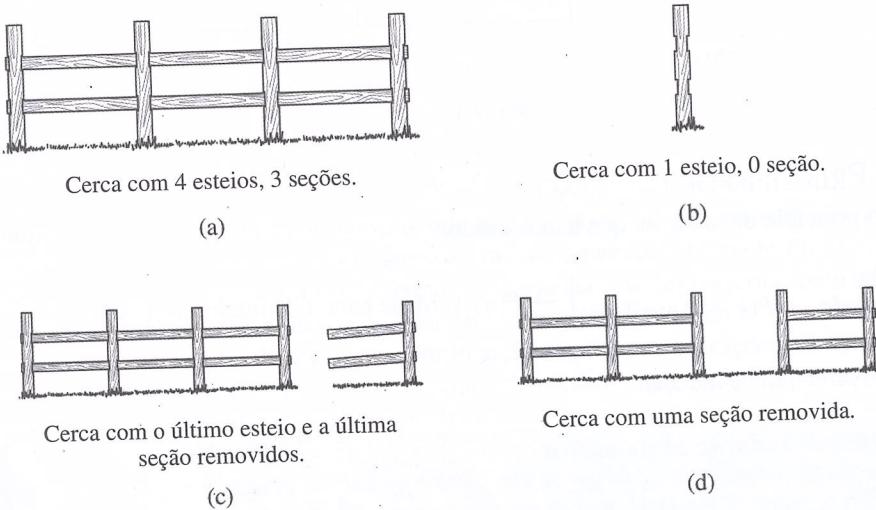


Fig. 2.4

Seja $P(n)$ a proposição que uma cerca com n esteios tem $n - 1$ seções; vamos provar que $P(n)$ é verdadeira para todo $n \geq 1$.

Vamos começar com o primeiro princípio de indução. Para o passo básico, $P(1)$ diz que uma cerca com apenas 1 esteio tem 0 seção, o que é claramente verdade (veja a Fig. 2.4b). Suponha que $P(k)$ é verdadeira:

uma cerca com k esteios tem $k - 1$ seções

e tente provar $P(k + 1)$:

(?) uma cerca com $k + 1$ esteios tem k seções.

Dada uma cerca com $k + 1$ esteios, como podemos relacioná-la a uma cerca com k esteios de modo a usar a hipótese de indução? Podemos cortar fora o último esteio e a última seção (Fig. 2.4c). A cerca resultante tem k esteios e, pela hipótese de indução, tem $k - 1$ seções. Portanto, a cerca original tinha k seções.

Vamos agora provar o mesmo resultado usando o segundo princípio de indução. O passo básico é igual ao do caso anterior. Para a hipótese de indução, supomos que

para todo r , $1 \leq r \leq k$, uma cerca com r esteios tem $r - 1$ seções

e tentamos provar $P(k + 1)$:

(?) uma cerca com $k + 1$ esteios tem k seções

Para uma cerca com $k + 1$ esteios, divida a cerca em duas partes removendo uma seção (Fig. 2.4d). As duas partes da cerca têm r_1 e r_2 esteios, onde $1 \leq r_1 \leq k$, $1 \leq r_2 \leq k$ e $r_1 + r_2 = k + 1$. Pela hipótese de indução, as duas partes têm, respectivamente, $r_1 - 1$ e $r_2 - 1$ seções, logo a cerca original tinha

$$(r_1 - 1) + (r_2 - 1) + 1 \text{ seções}$$

(O 1 extra é pela seção que foi removida.) A aritmética nos diz, então, que tínhamos

$$r_1 + r_2 - 1 = (k + 1) - 1 = k \text{ seções}$$

Isso prova que uma cerca com $k + 1$ esteios tem k seções, o que verifica a veracidade de $P(k + 1)$, completando a demonstração pelo segundo princípio de indução. ♦♦

O Exemplo 21 permite usar qualquer uma das formas de uma demonstração por indução, já que podemos diminuir a cerca retirando-se uma extremidade ou uma seção central. O problema no Exemplo 19 é semelhante.

EXEMPLO 22

Vamos mostrar, de novo, que qualquer produto de fatores pode ser escrito nessa linguagem de programação com um número par de parênteses, dessa vez usando o segundo princípio de indução. O passo básico é o mesmo que no Exemplo 19: um único fator tem 0 parêntese, um número par. Suponha que qualquer produto de r fatores, $1 \leq r \leq k$, pode ser escrito com um número par de parênteses. Considere agora um produto P com $k + 1$ fatores. Então P pode ser escrito como um produto $(S)T$ de dois fatores, S e T , onde S tem r_1 fatores e T tem r_2 fatores. Temos $1 \leq r_1 \leq k$, $1 \leq r_2 \leq k$ e $r_1 + r_2 = k + 1$. Pela hipótese de indução, S e T têm, cada um, um número par de parênteses e, portanto, $P = (S)T$ também tem um número par de parênteses. ♦♦

A maior parte dos problemas não funciona bem com uma das formas de indução: os problemas da cerca e da linguagem de programação são um tanto ou quanto artificiais. Em geral, usamos a segunda forma quando o problema se divide, naturalmente, no meio ao invés de crescer em uma das pontas.

EXEMPLO 23

Prove que, para todo $n \geq 2$, n é um número primo ou é um produto de números primos.

Vamos adiar a decisão sobre se usamos o primeiro ou o segundo princípio de indução; o passo básico é o mesmo nos dois casos e não precisamos começar com 1. É claro que devemos começar aqui com 2. $P(2)$ é a proposição que 2 é um número primo ou um produto de primos. Como 2 é primo, $P(2)$ é verdadeira. Pulando adiante, para qualquer dos dois princípios precisaremos analisar o caso $k + 1$. Se $k + 1$ for primo, estamos feitos. Se $k + 1$ não for primo, então é um número composto e pode ser escrito na forma $k + 1 = ab$. Dividimos $k + 1$ em dois fatores e talvez nenhum deles tenha o valor k , de modo que uma hipótese apenas sobre $P(k)$ não é suficiente. Usaremos, então, o segundo princípio de indução.

Vamos começar de novo e supor que, para todo r , $2 \leq r \leq k$, $P(r)$ é verdadeira — r é primo ou um produto de primos. Considere agora o número $k + 1$. Se $k + 1$ for primo, terminamos. Se $k + 1$ não for primo, então é um número composto e pode ser escrito na forma $k + 1 = ab$, onde $1 < a < k + 1$ e $1 < b < k + 1$. (Essa é uma fatoração não trivial, de modo que nenhum fator pode ser igual a 1 ou a $k + 1$.) Portanto, $2 \leq a \leq k$ e $2 \leq b \leq k$. A hipótese de indução pode ser aplicada a a e a b , logo cada um deles ou é um primo, ou é um produto de primos. Portanto, $k + 1$ é um produto de primos. Isso verifica $P(k + 1)$ e completa a demonstração pelo segundo princípio de indução. ♦♦

LEMBRETE:

Use o segundo princípio de indução quando o caso $k + 1$ depende de resultados anteriores a k .

A demonstração no Exemplo 23 é uma *demonstração de existência* em vez de uma *demonstração construtiva*. Saber que todo número que não é primo tem uma fatoração como um produto de primos não torna fácil *encontrar*, uma tal fatoração. Alguns sistemas de criptografia para transmitir informação de modo seguro na internet dependem da dificuldade de decompor números grandes em seus fatores primos (veja o Exercício 53 da Seção 4.4).

Prove que qualquer quantia, para franquia postal, maior ou igual a 8 centavos pode ser conseguida usando-se apenas selos de 3 e 5 centavos.

A proposição $P(n)$ agora é que precisamos apenas de selos de 3 e 5 centavos para obter n centavos em selos e queremos provar que $P(n)$ é verdadeira para todo $n \geq 8$. A base da indução é estabelecer $P(8)$, o que é feito pela equação

$$8 = 3 + 5$$

Por razões que ficarão claras em alguns instantes, vamos estabelecer, também, dois casos adicionais, $P(9)$ e $P(10)$, pelas equações

$$9 = 3 + 3 + 3$$

$$10 = 5 + 5$$

Vamos supor, agora, que $P(r)$ é verdadeira para qualquer r , $8 \leq r \leq k$, e considerar $P(k+1)$. Podemos supor que $k+1$ é pelo menos 11, já que provamos $P(r)$ para $r = 8, 9$ e 10 . Se $k+1 \geq 11$, então $(k+1) - 3 = k-2 \geq 8$ e, pela hipótese de indução, $P(k-2)$ é verdade. Portanto, $k-2$ pode ser escrito como uma soma de números iguais a 3 e a 5; adicionando-se mais um 3, obtemos $k+1$ como uma soma de números iguais a 3 e a 5. Isso prova a veracidade de $P(k+1)$ e completa a demonstração. ♦♦♦

- a. Por que os casos adicionais $P(9)$ e $P(10)$ foram demonstrados separadamente no Exemplo 24?
- b. Por que não poderíamos usar o primeiro princípio de indução na demonstração do Exemplo 24?

PROBLEMA PRÁTICO 9

SEÇÃO 2.2 REVISÃO

TÉCNICAS

- W ♦ Utilização do primeiro princípio de indução em demonstrações.
- W ♦ Utilização do segundo princípio de indução em demonstrações.

IDÉIAS PRINCIPAIS

- ♦ A indução matemática é uma técnica para provar propriedades de inteiros positivos.
- ♦ Uma demonstração por indução não precisa começar com 1.
- ♦ A indução pode ser usada para provar proposições sobre quantidades cujos valores são inteiros não-negativos arbitrários.
- ♦ O primeiro e o segundo princípios de indução provam a mesma conclusão, mas uma das abordagens pode ser mais fácil de usar em uma determinada situação.

EXERCÍCIOS 2.2

Nos Exercícios 1 a 22, use indução matemática para provar que as proposições dadas são verdadeiras para todo inteiro positivo n .

- ★1. $2 + 6 + 10 + \dots + (4n - 2) = 2n^2$
- 2. $2 + 4 + 6 + \dots + 2n = n(n + 1)$
- ★3. $1 + 5 + 9 + \dots + (4n - 3) = n(2n - 1)$
- 4. $1 + 3 + 6 + \dots + \frac{n(n+1)}{2} = \frac{n(n+1)(n+2)}{6}$
- ★5. $4 + 10 + 16 + \dots + (6n - 2) = n(3n + 1)$
- 6. $5 + 10 + 15 + \dots + 5n = \frac{5n(n+1)}{2}$
- 7. $1^2 + 2^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6}$

8. $1^3 + 2^3 + \dots + n^3 = \frac{n^2(n+1)^2}{4}$

★9. $1^2 + 3^2 + \dots + (2n-1)^2 = \frac{n(2n-1)(2n+1)}{3}$

10. $1^4 + 2^4 + \dots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$

11. $1 \cdot 3 + 2 \cdot 4 + 3 \cdot 5 + \dots + n(n+2) = \frac{n(n+1)(2n+7)}{6}$

12. $1 + a + a^2 + \dots + a^{n-1} = \frac{a^n - 1}{a - 1}$ para $a \neq 0, a \neq 1$

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

14. $\frac{1}{1 \cdot 3} + \frac{1}{3 \cdot 5} + \frac{1}{5 \cdot 7} + \dots + \frac{1}{(2n-1)(2n+1)} = \frac{n}{2n+1}$

★15. $1^2 - 2^2 + 3^2 - 4^2 + \dots + (-1)^{n+1}n^2 = \frac{(-1)^{n+1}(n)(n+1)}{2}$

16. $2 + 6 + 18 + \dots + 2 \cdot 3^{n-1} = 3^n - 1$

17. $2^2 + 4^2 + \dots + (2n)^2 = \frac{2n(n+1)(2n+1)}{3}$

18. $1 \cdot 2^1 + 2 \cdot 2^2 + 3 \cdot 2^3 + \dots + n \cdot 2^n = (n-1)2^{n+1} + 2$

19. $1 \cdot 2 + 2 \cdot 3 + 3 \cdot 4 + \dots + n(n+1) = \frac{n(n+1)(n+2)}{3}$

20. $1 \cdot 2 \cdot 3 + 2 \cdot 3 \cdot 4 + \dots + n(n+1)(n+2) = \frac{n(n+1)(n+2)(n+3)}{4}$

21. $\frac{1}{1 \cdot 4} + \frac{1}{4 \cdot 7} + \frac{1}{7 \cdot 10} + \dots + \frac{1}{(3n-2)(3n+1)} = \frac{n}{3n+1}$

22. $1 \cdot 1! + 2 \cdot 2! + 3 \cdot 3! + \dots + n \cdot n! = (n+1)! - 1$, onde $n!$ é o produto dos inteiros positivos de 1 a n .

★23. Uma *progressão geométrica* é uma seqüência de termos com um termo inicial a tal que cada termo a seguir é obtido multiplicando-se o termo anterior por uma *mesma razão* r . Prove a fórmula para a soma dos n primeiros termos de uma progressão geométrica ($n \geq 1$):

$$a + ar + ar^2 + \dots + ar^{n-1} = \frac{a - ar^n}{1 - r}$$

24. Uma *progressão aritmética* é uma seqüência de termos com um termo inicial a tal que cada termo a seguir é obtido somando-se uma *mesma parcela* d ao termo anterior. Prove a fórmula para a soma dos n primeiros termos de uma progressão aritmética ($n \geq 1$):

$$a + (a+d) + (a+2d) + \dots + [a+(n-1)d] = \frac{n}{2}[2a + (n-1)d]$$

25. Usando os Exercícios 23 e 24, encontre uma expressão para os valores das seguintes somas:

- a. $2 + 2 \cdot 5 + 2 \cdot 5^2 + \dots + 2 \cdot 5^9$
- b. $4 \cdot 7 + 4 \cdot 7^2 + 4 \cdot 7^3 + \dots + 4 \cdot 7^{12}$
- c. $1 + 7 + 13 + \dots + 49$
- d. $12 + 17 + 22 + 27 + \dots + 92$

26. Prove que

$$(-2)^0 + (-2)^1 + (-2)^2 + \dots + (-2)^n = \frac{1 - 2^{n+1}}{3}$$

para todo inteiro positivo ímpar n .

27. Prove que $n^2 \geq 2n + 3$ para $n \geq 3$.

★28. Prove que $n^2 > n + 1$ para $n \geq 2$.

29. Prove que $n^2 > 5n + 10$ para $n > 6$.

30. Prove que $2^n > n^2$ para $n \geq 5$.

Nos Exercícios 31 a 35, $n!$ é o produto dos inteiros positivos de 1 a n .

31. Prove que $n! > n^2$ para $n \geq 4$.
32. Prove que $n! > 3^n$ para $n \geq 7$.
- ★33. Prove que $2^n < n!$ para $n \geq 4$.
34. Prove que $2^{n-1} \leq n!$ para $n \geq 1$.
35. Prove que $n! < n^n$ para $n \geq 2$.
36. Prove que $(1+x)^n > 1+x^n$ para $n > 1$, $x > 0$.
37. Prove que $(a/b)^{n+1} < (a/b)^n$ para $n \geq 1$ e $0 < a < b$.
- ★38. Prove que $1+2+\dots+n < n^2$ para $n > 1$.
39. a. Tente usar indução para provar que

$$1 + \frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{2^n} < 2 \text{ para } n \geq 1$$

O que não funciona?

b. Prrove que

$$1 + \frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{2^n} = 2 - \frac{1}{2^n} \text{ para } n \geq 1.$$

mostrando, assim, que

$$1 + \frac{1}{2} + \frac{1}{4} + \dots + \frac{1}{2^n} < 2 \text{ para } n \geq 1$$

40. Prove que $1 + 1/2 + 1/3 + \dots + 1/2^n \geq 1 + n/2$ para $n \geq 1$. (Note que os denominadores aumentam de 1 em 1, não por potências de 2.)

Nos Exercícios de 41 a 52, prove que a proposição é verdadeira para todo inteiro positivo n .

- ★41. $2^{3n} - 1$ é divisível por 7.
42. $3^{2n} + 7$ é divisível por 8.
43. $7^n - 2^n$ é divisível por 5.
44. $13^n - 6^n$ é divisível por 7.
- ★45. $2^n + (-1)^{n+1}$ é divisível por 3.
46. $2^{5n+1} + 5^{n+2}$ é divisível por 27.
47. $3^{4n+2} + 5^{2n+1}$ é divisível por 14.
48. $7^{2n} + 16n - 1$ é divisível por 64.
- ★49. $10^n + 3 \cdot 4^{n+2} + 5$ é divisível por 9.
50. $n^3 - n$ é divisível por 3.
51. $n^3 + 2n$ é divisível por 3.
52. $x^n - 1$ é divisível por $x - 1$ para $x \neq 1$.
- ★53. Prove o Teorema de De Moivre:

$$(\cos \theta + i \sin \theta)^n = \cos n\theta + i \sin n\theta$$

para todo $n \geq 1$. Sugestão: lembre-se das fórmulas para a soma de ângulos em trigonometria:

$$\begin{aligned}\cos(\alpha + \beta) &= \cos \alpha \cos \beta - \sin \alpha \sin \beta \\ \sin(\alpha + \beta) &= \sin \alpha \cos \beta + \cos \alpha \sin \beta\end{aligned}$$

54. Prove que

$$\sin \theta + \sin 3\theta + \dots + \sin(2n-1)\theta = \frac{\sin^2 n\theta}{\sin \theta}$$

para todo $n \geq 1$ e para todo θ tal que $\sin \theta \neq 0$.

- ★55. Use indução para provar que o produto de três inteiros positivos consecutivos quaisquer é divisível por 3.

56. Suponha que a exponenciação é definida pela equação

$$x^j \cdot x = x^{j+1}$$

para qualquer $j \geq 1$. Use indução para provar que $x^n \cdot x^m = x^{n+m}$ para $n \geq 1$ e $m \geq 1$. (Sugestão: faça a indução para m fixo e n arbitrário.)

57. De acordo com o Exemplo 20, é possível usar ângulos de ferro para ladrilhar um tabuleiro 4×4 com o canto direito superior removido. Desenhe uma tal ladrilhagem.
58. O Exemplo 20 não cobre o caso de tabuleiros com tamanhos diferentes de potências de 2. Verifique se é possível ladrilhar um tabuleiro 3×3 .
59. Considere uma coleção de n retas infinitas tal que duas retas nessa coleção nunca são paralelas e três retas nunca têm um ponto comum de interseção. Mostre que, para $n \geq 1$, as retas dividem o plano em $(n^2 + n + 2)/2$ regiões separadas.
60. Uma cadeia formada por algarismos binários, 0 e 1, vai ser convertida em uma cadeia de paridade por adicionando-se um bit³ de paridade ao final da cadeia. (Para uma explicação sobre o uso de bits de paridade, veja o Exemplo 17 no Cap. 8.) O bit de paridade é inicialmente 0. Quando um 0 é processado, o bit de paridade não se altera. Quando um 1 é processado, o bit de paridade muda de 0 para 1 ou de 1 para 0. Prove que o número de algarismos iguais a 1 na cadeia final, incluindo o bit de paridade, é sempre par. (*Sugestão:* considere vários casos.)
- ★61. O que está errado na seguinte “demonstração” por indução matemática? Vamos provar que, para qualquer inteiro positivo n , n é igual a 1 mais n . Suponha que $P(k)$ é verdadeira.

$$k = k + 1$$

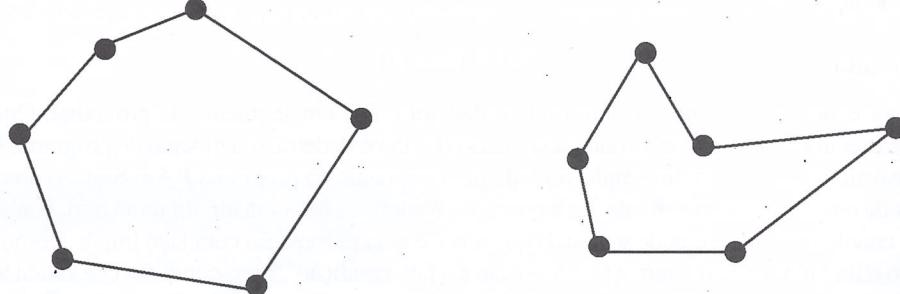
Somando 1 aos dois lados dessa equação, obtemos

$$k + 1 = k + 2$$

Portanto,

$P(k + 1)$ é verdadeira

62. O que está errado na seguinte “demonstração” por indução matemática? Vamos provar que todos os computadores são montados pelo mesmo fabricante. Em particular, vamos provar que em qualquer coleção de n computadores, onde n é um inteiro positivo, todos os computadores foram montados pelo mesmo fabricante. Vamos primeiro provar $P(1)$, um processo trivial, já que em qualquer coleção com apenas um computador existe apenas um fabricante. Vamos supor $P(k)$, isto é, em qualquer coleção de k computadores, todos os computadores foram montados pelo mesmo fabricante. Para provar $P(k + 1)$, vamos considerar uma coleção qualquer de $k + 1$ computadores. Vamos retirar um desses $k + 1$ computadores da coleção (vamos chamá-lo de HAL). Pela hipótese, todos os k computadores restantes foram montados pelo mesmo fabricante. Vamos trocar o HAL de lugar com um desses computadores. No novo grupo de k computadores, todos têm o mesmo fabricante.. Assim, o fabricante de HAL é o mesmo que produziu todos os outros computadores e todos os $k + 1$ computadores têm o mesmo fabricante.
63. Uma tribo obscura tem uma linguagem com apenas três palavras básicas, *mono*, *nono* e *sono*. Novas palavras são compostas agregando-se essas palavras em qualquer ordem, como em *sonononomononono*. Qualquer justaposição desse tipo é válida.
- Use o primeiro princípio de indução (sobre o número de palavras básicas em cada palavra composta) para provar que qualquer palavra nessa linguagem tem um número par de letras iguais a *o*.
 - Use o segundo princípio de indução (sobre o número de palavras básicas em cada palavra composta) para provar que qualquer palavra nessa linguagem tem um número par de letras iguais a *o*.
64. Um *polígono simples fechado* consiste em n pontos no plano ligados em pares por n segmentos de reta; cada ponto é a extremidade de exatamente dois segmentos de reta. A seguir são dados dois exemplos.



³Utilizaremos, como é usual, a nomenclatura inglesa *bit* para denotar um algarismo binário. (A palavra *bit* é uma abreviação para *binary digit*, que significa *algarismo binário*.) (N.T.)

- a. Use o primeiro princípio de indução para provar que a soma dos ângulos internos de um polígono simples fechado com n lados é $(n - 2)180^\circ$ para todo $n \geq 3$.
 - b. Use o segundo princípio de indução para provar que a soma dos ângulos internos de um polígono simples fechado com n lados é $(n - 2)180^\circ$ para todo $n \geq 3$.
- *65. Considere fbf's proposicionais contendo apenas os conectivos \wedge , \vee e \rightarrow (sem a negação) e tais que, ao usar um conectivo para juntar duas fbf's, temos que usar parênteses. Conte cada letra de proposição, conectivo ou parêntese como um símbolo. Por exemplo, $((A) \wedge (B)) \vee ((C) \wedge (D))$ é uma fbf desse tipo, com 19 símbolos. Prove que qualquer fbf desse tipo tem um número ímpar de símbolos.
- *66. Prove que qualquer quantia em selos maior ou igual a 2 centavos pode ser obtida usando-se apenas selos de 2 e 3 centavos.
67. Prove que qualquer quantia em selos maior ou igual a 12 centavos pode ser obtida usando-se apenas selos de 4 e 5 centavos.
68. Prove que qualquer quantia em selos maior ou igual a 14 centavos pode ser obtida usando-se apenas selos de 3 e 8 centavos.
- *69. Prove que qualquer quantia em selos maior ou igual a 64 centavos pode ser obtida usando-se apenas selos de 5 e 17 centavos.
70. O caixa automático em seu banco usa apenas notas de R\$20,00 e de R\$50,00 para dar dinheiro. Prove que você pode obter, além de R\$20,00, qualquer quantia múltipla de R\$10,00 que seja maior ou igual a R\$40,00.
71. Em qualquer grupo de k pessoas, $k \geq 1$, cada pessoa cumprimenta, com aperto de mão, todas as outras pessoas. Encontre uma fórmula para o número de apertos de mão e prove-a usando indução.

SEÇÃO 2.3 MAIS SOBRE DEMONSTRAÇÃO DE CORREÇÃO

Explicamos, na Seção 1.6, como usar um sistema de lógica formal para demonstrar matematicamente a correção de um programa. Asserções ou predicados envolvendo as variáveis do programa são inseridas no início, no final e em pontos intermediários entre as proposições do programa. Então, demonstrar a correção de qualquer proposição particular s_i envolve provar que o condicional representado pela tripla de Hoare

$$\{Q\} s_i \{R\} \quad (1)$$

é verdadeiro. Aqui, Q e R são asserções conhecidas como a pré-condição e a pós-condição, respectivamente, para a proposição. Podemos demonstrar a correção do programa se todos esses condicionais forem verdadeiros.

Discussimos, no Cap. 1, regras de inferência que fornecem condições sob as quais o condicional (1) é verdadeiro quando s_i é uma atribuição e quando s_i é um condicional. Vamos agora usar regras de inferência que nos dão condições sob as quais (1) é verdadeiro quando s_i é um laço. Adiamos a análise de proposições em laço até agora porque utiliza-se indução matemática na aplicação dessa regra de inferência.

A REGRa DO LAÇO

Suponha que s_i é uma proposição com laço da forma

enquanto condição B faça

P

fim do enquanto

onde B é uma condição que pode ser falsa ou verdadeira e P é um segmento de programa. Quando essa proposição é executada, verifica-se a condição B . Caso B seja verdadeira, o segmento de programa P é executado e B é reavaliada. Se B continua sendo verdadeira, o segmento de programa P é executado novamente, B é reavaliada de novo, e assim por diante. Se a condição B torna-se falsa em algum momento, o laço termina.

A forma condicional (1) que pode ser usada quando s_i é uma proposição com laço impõe (como o axioma de atribuição o fez) uma relação entre a pré-condição e a pós-condição. A pré-condição Q é válida antes de se entrar no laço; parece estranho mas uma das condições é que Q continue válido depois que o laço termina (o que significa que devemos procurar um Q' que queremos que seja verdadeiro quando o laço terminar). Além disso, B' — a condição para o laço parar — também tem que ser verdadeira. Assim, (1) vai ter a forma

$$\{Q\} s_i \{Q \wedge B'\} \quad (2)$$

♦ EXEMPLO 25 Considere a função em pseudocódigo a seguir, que é suposta de dar como resposta o valor $x * y$ para inteiros não-negativos x e y .

Produto(inteiro não-negativo x ; inteiro não-negativo y)

Variáveis locais:

inteiros i, j

$i = 0$

$j = 0$

enquanto $i \neq x$ **faça**

$j = j + y$

$i = i + 1$

fim do enquanto

// j agora tem o valor $x * y$

retorne j

fim da função Produto

Essa função contém um laço; a condição B para que o laço continue a ser executado é que $i \neq x$. A condição B' que pára a execução do laço é que $i = x$. Ao final do laço, afirma-se, no comentário, que j tem o valor $x * y$. Dado que $i = x$ quando o laço termina, a asserção $j = i * y$ também teria que ser verdadeira. Assim, ao término do laço, se a função faz o que se afirma, a asserção

$$j = i * y \wedge i = x$$

é verdadeira. Isso é da forma $Q \wedge B'$, se considerarmos a asserção

$$j = i * y$$

como Q . Para colocar na mesma forma que (2), a asserção $j = i * y$ teria que ser verdadeira antes do laço. Isso ocorre, de fato, já que imediatamente antes do laço temos que $i = j = 0$:

Parece que temos um candidato para Q , nesse exemplo, para o condicional (2) mas ainda não temos a regra de inferência que nos permite dizer quando (2) é um condicional verdadeiro. (Lembre-se de que descobrimos Q supondo que o código para a função funciona corretamente.)

A asserção Q tem que ser verdadeira antes de começar o laço. Se o condicional (2) é verdadeiro, Q tem que ser verdadeira após o término do laço. Já que podemos não saber exatamente quando o laço vai terminar, Q precisa permanecer verdadeira a cada iteração, o que inclui a iteração final. Q representa um predicado, ou uma relação, entre os valores das variáveis do programa. Se essa relação entre os valores das variáveis do programa é válida antes da execução de uma iteração do laço e após a execução da iteração, então a relação entre essas variáveis não é afetada pela ação da iteração do laço, embora os valores propriamente ditos possam ser modificados. Tal relação é chamada de um **invariante do laço**.

A **regra de inferência para laços** permite que a veracidade de (2) seja inferida de um condicional dizendo que Q é um invariante do laço, isto é, que se Q é verdadeira e se a condição B é verdadeira, de modo que é executada outra iteração do laço, então Q permanece verdadeira após essa iteração. A regra está formalmente enunciada na Tabela 2.4.

Para usar essa regra de inferência, precisamos encontrar um invariante do laço Q que seja *útil* — um que afirme o que queremos e o que esperamos que aconteça — e depois provar o condicional

$$\{Q \wedge B\} P \{Q\}$$

TABELA 2.4

De	Podemos deduzir	Nome da regra	Restrições sobre o uso
$\{Q \wedge B\} P \{Q\}$	$\{Q\} S_i \{Q \wedge B'\}$	laço	S_i tem a forma enquanto condição B faça P fim do enquanto

E é aqui que a indução entra em cena. Vamos denotar por $Q(n)$ a proposição que um invariante do laço Q proposto seja verdadeiro após n iterações do laço. Como não sabemos quantas iterações serão executadas no laço (isto é, por quanto tempo a condição B permanece verdadeira), queremos mostrar que $Q(n)$ é verdadeira para todo $n \geq 0$. (O valor $n = 0$ corresponde à asserção antes do início do laço, antes de qualquer iteração.)

Considere, novamente, a função em pseudocódigo do Exemplo 25. Naquele exemplo, conjecturamos que Q é a relação

$$j = i * y$$

Para usar a regra de inferência do laço, precisamos provar que Q é um invariante do laço.

As quantidades x e y permanecem constantes durante todo o processo mas os valores de i e j variam dentro do laço. Vamos denotar por i_n e j_n , respectivamente, os valores de i e j após n iterações do laço. Então, Q_n é a proposição $j_n = i_n * y$.

Vamos provar por indução que $Q(n)$ é válida para todo $n \geq 0$. $Q(0)$ é a proposição

$$j_0 = i_0 * y$$

que, como observamos no Exemplo 25, é verdadeira, pois antes de qualquer iteração, ao chegar pela primeira vez no laço, são atribuídos a ambos, i e j , o valor 0. (Formalmente, o axioma de atribuição poderia ser usado para provar que essas condições sobre i e j são válidas nesse instante.)

$$\text{Suponha } Q(k): j_k = i_k * y$$

$$\text{Prove } Q(k+1): j_{k+1} = i_{k+1} * y$$

Entre o instante em que j e i têm os valores j_k e i_k e o instante em que j e i têm os valores j_{k+1} e i_{k+1} , ocorre uma iteração do laço. Nessa iteração, j muda adicionando-se y a seu valor anterior e i muda adicionando-se 1. Ou seja,

$$j_{k+1} = j_k + y \quad (3)$$

$$i_{k+1} = i_k + 1 \quad (4)$$

Então

$$\begin{aligned} j_{k+1} &= j_k + y && (\text{por (3)}) \\ &= i_k * y + y && (\text{pela hipótese de indução}) \\ &= (i_k + 1)y \\ &= i_{k+1} * y && (\text{por (4)}) \end{aligned}$$

Acabamos de provar que Q é um invariante do laço.

A regra de inferência do laço nos permite inferir que, ao sair do laço, a condição $Q \wedge B'$ é válida, o que, nesse caso, torna-se

$$j = i * y \wedge i = x$$

Portanto, nesse instante, a proposição

$$j = x * y$$

é verdadeira, o que é exatamente o que é esperado que a função calcule.

O Exemplo 26 ilustra o fato de que invariantes do laço dizem algo mais forte sobre o programa do que o que queremos, de fato, mostrar; o que queremos mostrar é o caso particular do invariante ao final do laço. Encontrar o invariante do laço apropriado necessita uma análise de trás para a frente, partindo da conclusão desejada, como no Exemplo 25.

Não provamos, na realidade, que o laço nesse exemplo termina. O que provamos foi uma **correção parcial** — o programa produzirá a resposta correta se terminar. Como x é um inteiro não-negativo e i é um inteiro que começa em 0 e vai aumentando de 1 em 1 em cada iteração, sabemos que $i = x$ vai ter que ser verdade em algum instante.

Mostre que a função a seguir dá como resposta o valor $x + y$ para inteiros não-negativos x e y , provando o invariante do laço $Q: j = x + i$ e calculando Q ao final do laço.

EXEMPLO 26

PROBLEMA PRÁTICO 10

Soma(inteiro não-negativo x ; inteiro não-negativo y)

Variáveis locais:

inteiros i, j

$i = 0$

$j = x$

enquanto $i \neq y$ **faça**

$j = j + 1$

$i = i + 1$

fim do enquanto

//agora j tem o valor $x + y$

retorne j

fim da função Soma

As duas funções, a do Exemplo 25 e a do Problema Prático 10, não são muito realistas; afinal de contas, se quiséssemos calcular $x * y$ ou $x + y$ poderíamos, sem dúvida, usar um único comando. No entanto, as mesmas técnicas podem ser aplicadas a cálculos mais significativos, como o algoritmo de Euclides.

O ALGORITMO DE EUCLIDES

O **algoritmo de Euclides** foi elaborado pelo matemático grego Euclides há mais de 2300 anos, o que o torna um dos algoritmos mais antigos conhecidos. Esse algoritmo encontra o maior divisor comum entre dois inteiros não-negativos a e b , no qual ambos não são nulos ao mesmo tempo. O **máximo divisor comum** de a e b , denotado por $\text{mdc}(a, b)$, é o maior inteiro que divide (sem resto) ambos a e b . Por exemplo, $\text{mdc}(12, 18)$ é 6, $\text{mdc}(420, 66) = 6$ e $\text{mdc}(18, 0) = 18$ (esse último porque qualquer número diferente de 0 divide 0).

O algoritmo de Euclides funciona por meio de uma sucessão de divisões. Para encontrar $\text{mdc}(a, b)$, supondo que $a \geq b$, divida, primeiro, a por b , obtendo um quociente e um resto. Formalmente, nesse instante temos $a = q_1b + r_1$, onde $0 \leq r_1 < b$. A seguir, divida o divisor, b , pelo resto r_1 , obtendo $b = q_2r_2 + r_2$, onde $0 \leq r_2 < r_1$. Novamente, divida o divisor, r_1 , pelo resto r_2 , obtendo $r_1 = q_3r_3 + r_3$, onde $0 \leq r_3 < r_2$. É claro que temos aqui um processo em laço. Esse processo termina quando encontramos um resto 0; o máximo divisor comum é o último divisor utilizado.

EXEMPLO 27

Para encontrar $\text{mdc}(420, 66)$ são efetuadas as seguintes divisões:

$$\begin{array}{r} 420 \mid 66 \\ -396 \quad 6 \\ \hline 24 \end{array} \quad \begin{array}{r} 66 \mid 24 \\ -48 \quad 2 \\ \hline 18 \end{array} \quad \begin{array}{r} 24 \mid 18 \\ -18 \quad 1 \\ \hline 6 \end{array} \quad \begin{array}{r} 18 \mid 6 \\ -18 \quad 3 \\ \hline 0 \end{array}$$

A resposta é 6, o divisor utilizado quando temos resto 0.

A seguir apresentamos uma versão em pseudocódigo do algoritmo na forma de uma função que retorna o valor $\text{mdc}(a, b)$.

ALGORITMO ALGORITMO DE EUCLIDES

MDC(inteiro não-negativo a ; inteiro não-negativo b)

// $a \geq b$, um dos dois diferentes de zero

Variáveis locais:

inteiros i, j

$i = a$

$j = b$

enquanto $j \neq 0$ **faça**

calcule $i = qj + r$, $0 \leq r < j$

$i = j$

$j = r$

fim do enquanto

//agora i tem o valor $\text{mdc}(a, b)$

retorne i

fim da função MDC

Queremos mostrar que essa função está correta mas precisamos, primeiro, de um fato adicional, a saber,

$$(\forall \text{ inteiros } a, b, q, r)[(a = qb + r) \rightarrow (\text{mdc}(a, b) = \text{mdc}(b, r))] \quad (5)$$

Para provar (5), vamos supor que $a = qb + r$ e que c divide ambos a e b , de modo que $a = q_1c$ e $b = q_2c$. Então

$$r = a - qb = q_1c - q_2c = c(q_1 - q_2)$$

de modo que c também divide r . Portanto, qualquer inteiro que divida a e b também divide b e r . Suponha, agora, que d divide ambos b e r , de modo que $b = q_3d$ e $r = q_4d$. Então

$$a = qb + r = qq_3d + q_4d = d(qq_3 + q_4)$$

logo d também divide a . Portanto, qualquer inteiro que divida b e r também divide a e b . Como $(a, b) \in (b, r)$ têm precisamente os mesmos divisores, eles têm que ter o mesmo máximo divisor comum.

❖ EXEMPLO 28

Prove que o algoritmo de Euclides está correto.

Usando a função MDC, vamos provar o invariante de laço Q : $\text{mdc}(i, j) = \text{mdc}(a, b)$ e calcular Q quando o laço terminar. Usaremos indução para provar $Q(n)$: $\text{mdc}(i_n, j_n) = \text{mdc}(a, b)$ para todo $n \geq 0$. $Q(0)$ é a proposição

$$\text{mdc}(i_0, j_0) = \text{mdc}(a, b)$$

que é verdadeira, já que quando chegamos no laço pela primeira vez i e j têm os valores a e b , respectivamente.

$$\text{Suponha } Q(k): \text{mdc}(i_k, j_k) = \text{mdc}(a, b)$$

$$\text{Prove } Q(k+1): \text{mdc}(i_{k+1}, j_{k+1}) = \text{mdc}(a, b)$$

Pelos comandos de atribuição dentro do laço, sabemos que

$$i_{k+1} = j_k$$

$$j_{k+1} = r_k$$

Então,

$$\begin{aligned} \text{mdc}(i_{k+1}, j_{k+1}) &= \text{mdc}(j_k, r_k) \\ &= \text{mdc}(i_k, j_k) \quad \text{por (5)} \\ &= \text{mdc}(a, b) \quad \text{pela hipótese de indução} \end{aligned}$$

e, portanto, Q é um invariante do laço. Quando o laço termina, $\text{mdc}(i, j) = \text{mdc}(a, b)$ e $j = 0$, logo $\text{mdc}(i, 0) = \text{mdc}(a, b)$. Mas $\text{mdc}(i, 0) = i$, de modo que $i = \text{mdc}(a, b)$. Portanto a função MDC está correta. ❖

SEÇÃO 2.3 REVISÃO

TÉCNICAS

- W ❖ Verificação da correção de um segmento de programa que inclui uma proposição com laço.
- ❖ Cálculo do $\text{mdc}(a, b)$ usando o algoritmo de Euclides.

IDÉIAS PRINCIPAIS

- ❖ Podemos usar um invariante do laço, demonstrando por indução no número de iterações, para provar a correção de um laço em um programa.
- ❖ Pode-se demonstrar que o algoritmo de Euclides clássico para encontrar o máximo divisor comum de dois números inteiros não-negativos está correto.

EXERCÍCIOS 2.3

Nos Exercícios 1 a 4, prove que o pseudocódigo do segmento de programa está correto provando o invariante do laço Q e calculando Q depois do laço terminar.

1. Função que retorna o valor de x^2 para $x \geq 1$

Quadrado(inteiro positivo x)

Variáveis locais:

inteiros i, j

$$i = 1$$

$$j = 1$$

enquanto $i \neq x$ **faça**

$$j = j + 2i + 1$$

$$i = i + 1$$

fim do enquanto

//agora j tem o valor x^2

retorne j

fim da função Quadrado

$$Q: j = i^2$$

2. Função que retorna o valor de $x!$ para $x \geq 1$

Fatorial(inteiro positivo x)

Variáveis locais:

inteiros i, j

$$i = 2$$

$$j = 1$$

enquanto $i \neq x + 1$ **faça**

$$j = j * i$$

$$i = i + 1$$

fim do enquanto

//agora j tem o valor $x!$

retorne j

fim da função Fatorial

$$Q: j = (i - 1)!$$

3. Função que retorna o valor de x^y para $x, y \geq 1$

Potência(inteiro positivo x ; inteiro positivo y)

Variáveis locais:

inteiros i, j

$$i = 1$$

$$j = x$$

enquanto $i \neq y$ **faça**

$$j = j * x$$

$$i = i + 1$$

fim do enquanto

//agora j tem o valor x^y

retorne j

fim da função Potência

$$Q: j = x^i$$

4. Função para calcular e escrever o quociente q e o resto r quando x é dividido por y , $x \geq 0, y \geq 1$

Divide(inteiro não-negativo x ; inteiro positivo y)

Variáveis locais:

inteiros não-negativos q, r

$$q = 0$$

$$r = x$$

enquanto $r \geq y$ **faça**

$$\begin{aligned} q &= q + 1 \\ r &= r - y \end{aligned}$$

fim do enquanto

//agora q e r são o quociente e o resto
escreva("O quociente é" q "e o resto é" r)

fim da função Divide

$$Q: x = q * y + r$$

Para os Exercícios 5 a 8, use o algoritmo de Euclides para encontrar o máximo divisor comum dos números dados.

- 5. (2420, 70)
- ★6. (735, 90)
- 7. (1326, 252)
- 8. (1018215, 2695)

Nos Exercícios 9 a 14, prove que o segmento de programa está correto encontrando e demonstrando o invariante do laço apropriado Q e calculando Q depois do laço terminar.

9. Função que retorna o valor de $x - y$ para $x, y \geq 0$

Diferença(inteiro não-negativo x ; inteiro não-negativo y)

Variáveis locais:

inteiros i, j

$$i = 0$$

$$j = x$$

enquanto $i \neq y$ **faça**

$$j = j - 1$$

$$i = i + 1$$

fim do enquanto

//agora j tem o valor $x - y$

retorne j

fim da função Diferença

- ★10. Função que retorna o valor de $x * y^n$ para $n \geq 0$

Cálculo(inteiro x ; inteiro y ; inteiro não-negativo n)

Variáveis locais:

inteiros i, j

$$i = 0$$

$$j = x$$

enquanto $i \neq n$ **faça**

$$j = j * y$$

$$i = i + 1$$

fim do enquanto

//agora j tem o valor $x * y^n$

retorne j

fim da função Cálculo

11. Função que retorna o valor de $(x + 1)^2$ para $x \geq 1$

QuadradoDoIncremento(inteiro positivo x)

Variáveis locais:

inteiros i, j

$$i = 1$$

$$j = 4$$

enquanto $i \neq x$ **faça**

$$j = j + 2i + 3$$

$$i = i + 1$$

fim do enquanto

//j agora tem o valor $(x + 1)^2$

retorne j

fim da função QuadradoDoIncremento

12. Função que retorna o valor de 2^n para $n \geq 1$

PotênciasDe2(inteiro positivo n)

Variáveis locais:

inteiros i, j

i = 1

j = 2

enquanto i ≠ n **faça**

j = j * 2

i = i + 1

fim do enquanto

//agora j tem o valor 2^n

retorne j

fim da função PotênciasDe2

13. Função que retorna o valor de $x * n!$ para $n \geq 1$

Outra(inteiro x; inteiro positivo n)

Variáveis locais:

inteiros i, j

i = 1

j = x

enquanto i ≠ n **faça**

j = j * (i + 1)

i = i + 1

fim do enquanto

//agora j tem o valor x * n!

retorne j

fim da função Outra

14. Função que retorna o valor do polinômio

$$a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$$

para um valor dado de x.

Polinômio(real $a_n; \dots; a_0$; real x)

Variáveis locais:

inteiros i, j

i = n

j = a

enquanto i ≠ 0 **faça**

j = j * x + a_{i-1}

i := i - 1

fim do enquanto

//agora j tem o valor do polinômio em x

retorne j

fim da função Polinômio

15. São dadas a seguir quatro funções que deveriam retornar o valor $a[1] + a[2] + \dots + a[n]$ para $n \geq 1$ (a soma dos n primeiros elementos de um vetor de inteiros). Para as que não produzem resultados corretos, explique o que está errado. Para as que produzem resultados corretos, demonstre a correção.

- a. SomaA(inteiros n, a[1], a[2], ..., a[n])

Variáveis locais:

inteiros i, j

$i = 0$
 $j = 0$

enquanto $i \leq n$ **faça**

$i = i + 1$
 $j = j + a[i]$

fim do enquanto

//agora j tem valor $a[1] + a[2] + \dots + a[n]$
retorne j

fim da função SomaA

b. SomaB(inteiros $n, a[1], a[2], \dots, a[n]$)

Variáveis locais:

inteiros i, j

$i = 1$
 $j = 0$

enquanto $i \leq n$ **faça**

$j = j + a[i]$
 $i = i + 1$

fim do enquanto

//agora j tem valor $a[1] + a[2] + \dots + a[n]$
retorne j

fim da função SomaB

c. SomaC(inteiros $n, a[1], a[2], \dots, a[n]$)

Variáveis locais:

inteiros i, j

$i = 0$
 $j = 0$

enquanto $i \leq n$ **faça**

$j = j + a[i]$
 $i = i + 1$

fim do enquanto

//agora j tem valor $a[1] + a[2] + \dots + a[n]$
retorne j

fim da função SomaC

d. SomaD(inteiros $n, a[1], a[2], \dots, a[n]$)

Variáveis locais:

inteiros i, j

$i = 1$
 $j = a[1]$

enquanto $i \leq n$ **faça**

$j = j + a[i + 1]$
 $i = i + 1$

fim do enquanto

//agora j tem valor $a[1] + a[2] + \dots + a[n]$
retorne j

fim da função SomaD

SEÇÃO 2.4 RECURSIVIDADE E RELAÇÕES DE RECORRÊNCIA

DEFINIÇÕES RECORRENTES

Uma definição onde o item sendo definido aparece como parte da definição é chamada de uma **definição recorrente** ou **definição por recorrência** ou ainda **definição por indução**. 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, onde alguns casos simples do item sendo definido são dados explicitamente.
2. Um passo de indução ou recorrência, onde novos casos do item 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 nos permite construir novos casos, a partir desses simples, e depois construir ainda outros casos a partir desses novos, e assim por diante. (O nome “definição por indução” é devido à analogia com 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, onde a veracidade de $P(k + 1)$ é estabelecida a partir da veracidade de P em valores anteriores.)

Recorrência é uma idéia importante que pode ser usada para definir seqüências 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.

Seqüências Definidas por Recorrência

Uma **seqüência** S é uma lista de objetos que são numerados em determinada ordem; existe um primeiro objeto, um segundo, e assim por diante. $S(k)$ denota o k -ésimo objeto na seqüência. Uma seqüência é definida por recorrência nomeando-se, explicitamente, o primeiro valor (ou alguns poucos primeiros valores) na seqüência e depois definindo valores subsequentes na seqüência em termos de valores anteriores.

♦ EXEMPLO 29

A seqüência S é 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 seqüência S é

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

Uma regra como a da proposição 2 no Exemplo 29, que define um valor de uma seqüência em termos de um ou mais valores anteriores, é chamada uma **relação de recorrência**.

PROBLEMA PRÁTICO 11

A seqüência 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 seqüência T .

♦ EXEMPLO 30

A famosa **seqüência de Fibonacci**, introduzida no século XIII por um comerciante e matemático italiano, é uma seqüência de números definida por recorrência por

$$F(1) = 1$$

$$F(2) = 1$$

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

Aqui são dados os dois primeiros valores da seqüência e a relação de recorrência define o n -ésimo valor 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.

Escreva os oito primeiros valores da seqüência de Fibonacci.

Prove que, na seqüência 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 $F(n-2)$, deve-se 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 12)

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

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 $k+1$, onde $k+1 \geq 3$. 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 seqüência de Fibonacci, temos

$$F(k+5) = F(k+3) + F(k+4) \quad (F \text{ em qualquer valor é a soma de } F \text{ nos dois valores anteriores})$$

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,

$$\begin{aligned} F(k+5) &= F(k+3) + F(k+4) \\ &= [3F(k+1) - F(k-1)] + [3F(k+2) - F(k)] \\ &= 3[F(k+1) + F(k+2)] - [F(k-1) + F(k)] \\ &= 3F(k+3) - F(k+1) \quad (\text{usando a relação de recorrência novamente}) \end{aligned}$$

Isso completa a demonstração por indução.

A fórmula

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

do Exemplo 31 também pode ser provada diretamente, sem indução, usando apenas a relação de recorrência na definição dos números de Fibonacci. A relação de recorrência

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

PROBLEMA PRÁTICO 12

EXEMPLO 31

EXEMPLO 32

pode ser reescrita na forma

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

Logo,

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

PROBLEMA PRÁTICO 13

Na demonstração por indução do Exemplo 31, por que é necessário provar o caso $n = 2$ como um caso particular?

Conjuntos Definidos por Recorrência

Os objetos em uma seqüência são ordenados — existe 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 33

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 fbf's, então $(P \wedge Q)$, $(P \vee Q)$, $(P \rightarrow Q)$, (P') e $(P \leftrightarrow Q)$ também o são.

Muitas vezes omitimos os parênteses quando isso não causa confusão; assim, podemos escrever $(P \vee Q)$ como $P \vee Q$, ou (P') como P' . Começando com letras de proposição e usando, repetidamente, a regra 2, podemos construir todas as fbf's proposicionais. Por exemplo, A , B e C são fbf's pela regra 1. Pela regra 2,

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

são, ambas, fbf's. 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 14

Mostre como construir a fbf $((A \vee (B')) \rightarrow C)$ da definição no Exemplo 33.

PROBLEMA PRÁTICO 15

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 freqüência em ciência da computação. Computadores guardam os dados como **cadeias binárias**, cadeias do alfabeto que consiste apenas em 0 e 1; compiladores vêem proposições ou comandos em programas como cadeias de *tokens*, 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, pode ser definida de forma

recorrente (veja o Exemplo 34). Muitos conjuntos de cadeias com propriedades particulares também têm definições recorrentes.

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 induutivo dessa definição. Note que, para qualquer cadeia x , $x\lambda = \lambda x = x$.

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

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.

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 é um identificador, a concatenação de A e qualquer letra ou dígito também o é.

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 pelos 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 seqüência de escolhas como

$<\text{identificador}>$	pode ser	$<\text{identificador}> <\text{dígito}>$
	que pode ser	$<\text{identificador}>2$
	que pode ser	$<\text{identificador}><\text{letra}>2$
	que pode ser	$<\text{identificador}>e2$
	que pode ser	$<\text{letra}>e2$
	que pode ser	$me2$

Operações Definidas por Recorrência

Certas operações em objetos podem ser definidas de forma recorrente, como nos Exemplos 36 e 37.

Uma definição recorrente da operação de exponenciação a^n de um número real não-nulo a , onde n é um inteiro não-negativo, é

1. $a^0 = 1$
2. $a^n = (a^{n-1})a$ para $n \geq 1$

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

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

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

EXEMPLO 34

PROBLEMA PRÁTICO 16

PROBLEMA PRÁTICO 17

EXEMPLO 35

EXEMPLO 36

EXEMPLO 37

PROBLEMA PRÁTICO 18

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. $A_1 \vee A_2$ definida como na Seção 1.1
 2. $A_1 \vee \dots \vee A_n = (A_1 \vee \dots \vee A_{n-1}) \vee A_n$ 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 em parênteses é desnecessário porque todos esses agrupamentos são equivalentes à expressão geral de n letras de proposição. Em forma simbólica, para qualquer n com $n \geq 3$ e qualquer p com $1 \leq p \leq n - 1$,

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

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 $1 \leq p \leq k - 1$,

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

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

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

Algoritmos Definidos por Recorrência

O Exemplo 29 dá uma definição recorrente para uma seqüência S . Suponha que queremos escrever um programa de computador para calcular $S(n)$ para algum inteiro positivo n . Podemos usar uma entre duas abordagens. Se queremos 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 29, até chegar, finalmente, em $S(12)$. Sem dúvida, essa abordagem envolve iteração em alguma espécie de laço. A seguir, vamos dar uma função em pseudocódigo S que usa esse algoritmo iterativo. A base, com $n = 1$, é obtida na primeira cláusula da proposição **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 seqüência 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 n)
//função que calcula iterativamente o valor S(n)
//para a seqüência S do Exemplo 29
```

```
Variáveis locais:
inteiro i //índice do laço
ValorCorrente //valor corrente da função S
se n = 1 então
    retorne 2
senão
    i = 2
    ValorCorrente = 2
    enquanto i <= n faça
        ValorCorrente = 2*ValorCorrente
        i = i + 1
    fim do enquanto
    //agora ValorCorrente tem o valor S(n)
    retorne ValorCorrente
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 recorrente* ou *recursivo*, escrito, novamente, como uma função em pseudocódigo.

ALGORITMO

```

S(inteiro n)
//função que calcula o valor S(n) de forma recorrente
//para a seqüência S do Exemplo 29
    se n = 1 então
        retorno 2
    senão
        retorno 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 onde 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*, isto é, 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 $n = 1$.

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 algoritmos iterativo e recorrente ao executar a mesma tarefa? Nesse exemplo, a versão recorrente é certamente mais curta, já que não precisa gerenciar um cálculo em laço. A descrição da execução do algoritmo recorrente 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 recorrentes 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 recorrentes podem necessitar de mais cálculos e executar mais lentamente do que os não-recorrentes (veja o Exercício 7 no segmento 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 soluções não-recorrentes muito complexas. Uma solução recorrente é bem adequada para o problema de calcular os valores de uma seqüência definida de maneira recorrente. Muitas linguagens de programação aceitam recorrência.

Escreva o corpo de uma função recorrente para calcular $T(n)$ para a seqüência T definida no Problema Prático 11.

PROBLEMA
PRÁTICO 19

No Exemplo 37, 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.

EXEMPLO 38

ALGORITMO

```

Produto(inteiro  $m$ ; inteiro  $n$ )
//função que calcula de forma recorrente o produto de  $m$  e  $n$ 
se  $n = 1$  então
    retorne  $m$ 
senão
    retorne Produto( $m, n - 1$ ) +  $m$ 
fim do se
fim da função Produto

```

LEMBRETE:

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

Um algoritmo recorrente 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 recorrente pode ser útil, mesmo que o problema original não tenha sido enunciado de forma recorrente.

Para nos convencer de que um determinado algoritmo recorrente 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 recorrente. Ao invés 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).

EXEMPLO 39

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” deveria vir depois de “Teixeira, José”.) O algoritmo de **ordenação por seleção** — um algoritmo simples mas particularmente eficaz — é descrito em pseudocódigo abaixo.

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 ordena toda a lista). A parte recorrente 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 $L[j]$, depois do que o máximo ocorre na posição j , a última posição na parte da lista 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 tem o valor 1, a parte da lista sendo considerada tem apenas um elemento, que tem que estar no lugar certo. Nesse instante a lista toda está ordenada.

ALGORITMO ORDENAÇÃO POR SELEÇÃO

```

OrdenaçãoPorSeleção(lista  $L$ ; inteiro  $j$ )
//algoritmo recorrente 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  $j$ 
    permute  $L[i]$  e  $L[j]$ 
    OrdenaçãoPorSeleção( $L, j - 1$ )
fim do se
fim da função OrdenaçãoPorSeleção

```

EXEMPLO 40

Agora que ordenamos nossa lista, uma 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 recorrente descrito em pseudocódigo a seguir:

ALGORITMO BUSCA BINÁRIA

```

BuscaBinária(lista L; inteiro i; inteiro j; tipo item x)
//procura na lista ordenada L, de L[i] a L[j], pelo item x
    se i > j então
        escreva("não encontrado")
    senão
        encontre o índice k do item do meio na lista L[i] – L[j]
        se x = item do meio então
            escreva("encontrado")
        senão
            se x < item do meio 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 **se** principal é o passo básico que diz que x não pode ser encontrado em uma lista vazia, uma na qual 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 tem um número ímpar de itens, existe, de fato, um item do meio; se a seção contém um número par de itens, basta 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.

Vamos aplicar o algoritmo de busca binária à lista

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

na qual 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, isto é, 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.

Em uma busca binária da lista no Exemplo 41, nomeie os elementos que são comparados com x se x tem o valor 8.

EXEMPLO 41

PROBLEMA PRÁTICO 20

Vimos uma série de definições recorrentes. A Tabela 2.5 resume suas características.

TABELA 2.5

Definições Recorrentes

O que está sendo definido	Características
Seqüência recorrente	O primeiro ou os dois primeiros valores da seqüência são conhecidos; os outros elementos na seqüência são definidos em termos dos anteriores.
Conjunto recorrente	Alguns elementos específicos do conjunto são conhecidos; outros elementos no conjunto são construídos a partir dos elementos que já sabemos que pertencem ao conjunto.
Operação recorrente	Um caso “pequeno” da operação fornece um valor específico; outros casos são definidos a partir de casos menores.
Algoritmo recorrente	Para o menor valor do(s) argumento(s), o comportamento do algoritmo é conhecido; para valores maiores, o algoritmo chama a si mesmo com valores menores do(s) argumento(s).

RESOLVENDO RELAÇÕES DE RECORRÊNCIA

Desenvolvemos dois algoritmos, um iterativo, o outro recorrente, para calcular um valor $S(n)$ para a seqüência S do Exemplo 29. No entanto, existe uma maneira ainda mais fácil para 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 — explicitamente ou implicitamente por recorrência — todos os valores menores de S antes. Uma equação como (3), na qual 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.

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 idéia da forma geral. Finalmente essa conjectura é verificada por indução matemática.

EXEMPLO 42

Considere novamente a condição básica e a relação de recorrência para a seqüência S do Exemplo 29:

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

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

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 tem 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 Eq. (4). Vamos supor que $S(k) = 2^k$. Então

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

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

Encontre uma solução em forma fechada para a relação de recorrência, sujeita à condição básica, para a sequência T :

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

(Sugestão: expanda, conjecture e verifique.) ◊

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

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

onde 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

$$S(n) = cS(n - 1) + g(n) \tag{6}$$

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

Vamos encontrar a fórmula para a solução da Eq. (6), a relação de recorrência linear de primeira ordem genérica 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 42. Usando a Eq. (6) repetidamente e simplificando, obtemos

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

PROBLEMA PRÁTICO 21

Após k expansões, a forma geral parece ser

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

Se o primeiro termo da seqüência é conhecido, então a expansão termina quando $n - k = 1$, ou $k = n - 1$, quando temos

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

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 A para uma discussão da notação de somatório.) Assim, por exemplo,

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

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

Com a notação de somatório a Eq. (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 42, para verificar que essa fórmula é a solução da relação de recorrência (6) (veja o Exercício 90).

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, no entanto, ainda não é 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.

Encontramos, portanto, uma solução geral 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 Eq. (6) é a notação usual para uma função de n ; só estudaremos funções formalmente no Cap. 4, mas você pode pensar em $g(n)$ como sendo 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 Eq. (8) se $g(n) = 2n$.

♦ EXEMPLO 43 A seqüência $S(n)$ do Exemplo 42,

$$S(1) = 2$$

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

é uma relação de recorrência linear homogênea de primeira ordem com coeficientes constantes. Em outras palavras, ela coincide com a Eq. (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. ♦

Temos agora duas maneiras alternativas para resolver uma relação de recorrência linear de primeira ordem com coeficientes constantes. A Tabela 2.6 resume essas abordagens.

TABELA 2.6

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"> Use a relação de recorrência repetidamente até poder adivinhar o padrão. Decida qual será o padrão quando $n - k = 1$. Verifique a fórmula resultante por indução.
Fórmula da solução	<ol style="list-style-type: none"> Coloque sua relação de recorrência na forma $S(n) = cS(n - 1) + g(n)$ para encontrar c e $g(n)$. Use c, $g(n)$ e $S(1)$ na fórmula $S(n) = c^{n-1}S(1) + \sum_{i=2}^n c^{n-i}g(i)$ <ol style="list-style-type: none"> Calcule o somatório resultante para obter a expressão final.

Encontre uma solução em forma fechada para a relação de recorrência

EXEMPLO 44

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

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

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

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, expandimos

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

Essa forma geral parece ser

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

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

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

LEMBRETE:

Ao expandir, certifique-se de que colocou todas as partes da refeita da relação de recorrência, como o $+3$ nesse exemplo.

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

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

$$\begin{aligned} S(k + 1) &= 2S(k) + 3 \\ &= 2(2^{k+1} + 3[2^{k-1} - 1]) + 3 \\ &= 2^{k+2} + 3 \cdot 2^k - 6 + 3 \\ &= 2^{k+2} + 3[2^k - 1] \end{aligned}$$

pela relação de recorrência

pela hipótese de indução

multiplicando

PROBLEMA PRÁTICO 22

Refaça o Problema Prático 21 usando a Eq. (8).

EXEMPLO 45

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 Eq. (6), $S(n) = cS(n - 1) + g(n)$, vemos que $c = 1$ e $g(n) = n + 1$. Vamos substituir na Eq. (8), $S(n) = c^{n-1}S(1) +$

$$\sum_{i=2}^n C^{n-1}g(i), \text{ onde } g(i) \text{ vai ser } i + 1.$$

$$\begin{aligned} T(n) &= (1)^{n-1}(2) + \sum_{i=2}^n (1)^{n-i}(i + 1) \\ &= 2 + \sum_{i=2}^n (i + 1) \\ &= 2 + (3 + 4 + \dots + (n + 1)) \\ &= \frac{n(n + 1)}{2} + m \end{aligned}$$

EXEMPLO 46

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 (Fig. 2.5). 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

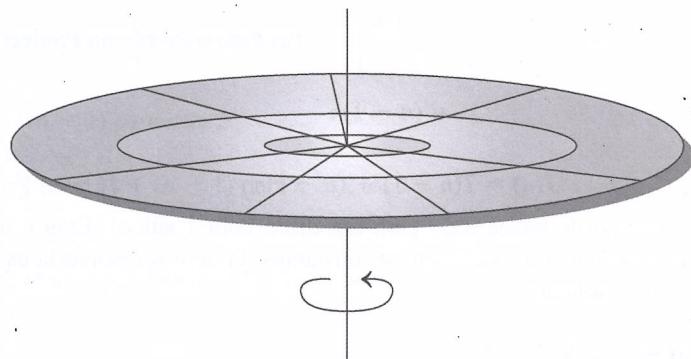


Fig. 2.5

⁴Esse exemplo baseia-se em um trabalho em "Research Problem for Undergraduate Students which Spans Hardware Issues, Mathematical Methods and Programming: Average Seek Time of a Computer Disk", de autoria de Jan Plaza, <http://faculty.plattsburgh.edu/jan.plaza/teaching/papers/seektme.html>

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, onde 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; de fato, encontrar o número médio $A(n)$ de unidades. As hipóteses são que existem n sulcos, a cabeça de leitura está sobre algum sulco i e que é igualmente provável que tenha que se movimentar para qualquer sulco j .

A Tabela 2.7 mostra o número de unidades ao se ir de um sulco para outro, onde 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 2.7

Sulco Inicial \ Sulco Desejado	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 2.7 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

$$T(n) = T(n - 1) + (\text{o total da última linha mais a última coluna})$$

e que a última linha mais a última coluna contribuem com

$$\begin{aligned} 2[1 + 2 + 3 + \dots + (n - 1)] &= 2\left[\frac{(n - 1)n}{2}\right] \quad (\text{usando o Problema Prático 7}) \\ &= (n - 1)n \end{aligned}$$

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 Eq. (8), onde $c = 1$ e $g(n) = (n - 1)n$. A solução é

$$\begin{aligned} T(n) &= 0 + \sum_{i=2}^n (i - 1)i \\ &= 1 \cdot 2 + 2 \cdot 3 + 3 \cdot 4 + \dots + (n - 1)n \\ &= \frac{(n - 1)n(n + 1)}{3} \quad (\text{do Exercício 19, Seção 2.2}) \end{aligned}$$

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 é, de fato, um pouco menor do que $n/3$. 

SEÇÃO 2.4 REVISÃO

TÉCNICAS

- ❖ Geração de valores em uma seqüência definida por recorrência.
- ❖ Demonstração de propriedades da seqüência de Fibonacci.
- W ❖ Reconhecimento de objetos em uma coleção definida por recorrência.
- ❖ Obtenção de definições recorrentes para um conjunto particular de objetos.
- ❖ Obtenção de definições recorrentes para determinadas operações em objetos.
- ❖ Obtenção de algoritmos recorrentes para gerar seqüências definidas por recorrência.
- W ❖ Resolução de relações de recorrência lineares de primeira ordem com coeficientes constantes através da fórmula da solução.
- W ❖ Resolução de relações de recorrência pelo método de expandir, conjecturar e verificar.

IDÉIAS PRINCIPAIS

- ❖ Podem ser dadas definições recorrentes para seqüências de objetos, conjuntos de objetos e operações em objetos, nos quais a condição básica é conhecida e novas informações dependem de informações já conhecidas.
- ❖ Algoritmos recorrentes fornecem um modo natural de resolver determinados problemas chamando a mesma tarefa para versões menores do problema.
- ❖ Certas relações de recorrência têm solução em forma fechada.

EXERCÍCIOS 2.4

Para os Exercícios 1 a 10, escreva os cinco primeiros valores da seqüência.

★1. $S(1) = 10$

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

2. $A(1) = 2$

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

3. $B(1) = 1$

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

★4. $S(1) = 1$

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

5. $T(1) = 1$

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

6. $P(1) = 1$

$P(n) = n^2P(n - 1) + (n - 1)$ para $n \geq 2$

★7. $M(1) = 2$

$M(2) = 2$

$M(n) = 2M(n - 1) + M(n - 2)$ para $n > 2$

8. $D(1) = 3$

$D(2) = 5$

$D(n) = (n - 1)D(n - 1) + (n - 2)D(n - 2)$ para $n > 2$

9. $W(1) = 2$

$W(2) = 3$

10. $W(n) = W(n - 1)W(n - 2)$ para $n > 2$

$T(1) = 1$

$T(2) = 2$

$T(3) = 3$

$T(n) = T(n - 1) + 2T(n - 2) + 3T(n - 3)$ para $n > 3$

Nos Exercícios 11 a 15, prove a propriedade dada dos números de Fibonacci diretamente da definição.

★11. $F(n + 1) + F(n - 2) = 2F(n)$ para $n \geq 3$

12. $F(n) = 5F(n - 4) + 3F(n - 5)$ para $n \geq 6$

13. $[F(n + 1)]^2 = [F(n)]^2 + F(n - 1)F(n + 2)$ para $n \geq 2$

14. $F(n + 3) = 2F(n + 1) + F(n)$ para $n \geq 1$

15. $F(n + 6) = 4F(n + 3) + F(n)$ para $n \geq 1$

Nos Exercícios 16 a 19, prove a propriedade dada dos números de Fibonacci para todo $n \geq 1$. (Sugestão: o primeiro princípio de indução vai funcionar.)

★16. $F(1) + F(2) + \dots + F(n) = F(n + 2) - 1$

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

18. $F(1) + F(3) + \dots + F(2n - 1) = F(2n)$

19. $[F(1)]^2 + [F(2)]^2 + \dots + [F(n)]^2 = F(n)F(n + 1)$

Nos Exercícios 20 a 23, prove a propriedade dada dos números de Fibonacci usando o segundo princípio de indução.

★20. Exercício 14

21. Exercício 15

22. $F(n) < 2^n$ para $n \geq 1$

23. $F(n) > \left(\frac{3}{2}\right)^{n-1}$ para $n \geq 6$

24. 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 a parte (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 seqüência de Fibonacci.

25. Uma seqüência 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 $n \geq 4$.

26. Uma seqüência 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 $T(n) \leq (5/2)^n$ para $n \geq 0$.

27. A seqüência de Lucas é definida por

$$L(1) = 1$$

$$L(2) = 3$$

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

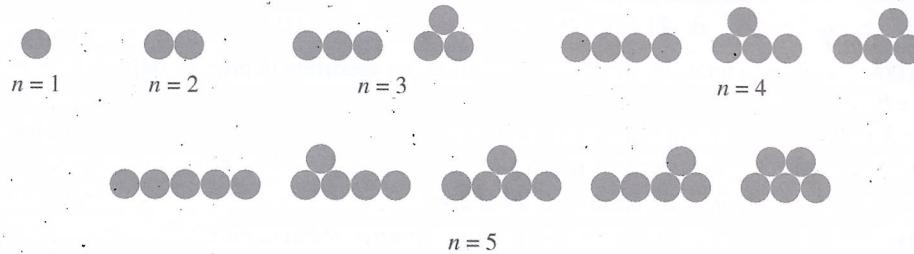
- Escreva os cinco primeiros termos da seqüência.
- Prove que $L(n) = F(n) + F(n - 2)$ para $n > 2$, onde F é a seqüência de Fibonacci.⁵

Para os Exercícios 28 a 31, decida se as seqüências descritas são subseqüências da seqüência de Fibonacci, isto é, se seus elementos são alguns ou todos os elementos, na ordem correta, da seqüência de Fibonacci.⁵

28. A seqüência $A(n)$, onde $A(n) = (n - 1)2^{n-2} + 1$, $n \geq 1$. Os quatro primeiros valores são 1, 2, 5 e 13, os quais — até agora — formam uma subseqüência da seqüência de Fibonacci.

★29. A seqüência $B(n)$, onde $B(n) = 1 + (\text{a soma dos } n \text{ primeiros elementos da seqüência de Fibonacci})$, $n \geq 1$. Os quatro primeiros valores são 2, 3, 5 e 8; os quais — até agora — formam uma subseqüência da seqüência de Fibonacci.

30. A seqüência $C(n)$, onde $C(n)$ 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 e 5, os quais — até agora — formam uma subseqüência da seqüência de Fibonacci.



31. A seqüência $D(n)$, onde $D(n)$ 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 e 8, os quais — até agora — formam uma subseqüência da seqüência 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 $D(n + 1)$.)

32. O problema original colocado por Fibonacci tratava de pares de coelhos. Dois coelhos não cruzam até terem dois meses de idade. Depois disso, cada par de coelhos produz um novo par cada mês. Suponha que nenhum coelho morre. Denote por $C(n)$ o número de pares de coelhos ao final de n meses se você começa com um único par de coelhos. Mostre que $C(n)$ é a seqüência de Fibonacci.

33. A seqüência 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.

⁵Os Exercícios de 28 a 31 foram tirados da seção “Mathematical Recreations”, da autoria de Ian Stewart, no exemplar de maio de 1995 da revista *Scientific American*.

- ★34. Escreva uma definição recorrente para uma progressão geométrica com termo inicial a e razão r (veja o Exercício 23, Seção 2.2).
35. Escreva uma definição recorrente para uma progressão aritmética com termo inicial a e parcela a ser somada d (veja o Exercício 24, Seção 2.2).
- ★36. Em um experimento, uma 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 seqüência $A(n)$, o número de bactérias presentes no início do n -ésimo período de tempo.
 - No início de que intervalo terão 1.350.000 bactérias presentes?
37. Uma quantia de R\$500,00 é investida em uma aplicação que paga juros de 10% 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\$700,00?
38. Uma coleção T de números é definida por recorrência por:
- 1 pertence a T .
 - Se X pertence a T , então $X + 3$ e $2*X$ também pertencem.
- Quais dos números a seguir pertencem a T ?
- 6
 - 7
 - 19
 - 12
39. Uma coleção M de números é definida por recorrência por:
- 1 e 3 pertencem a M .
 - Se X e Y pertencem a M , então $X * Y$ também pertence.
- Quais dos números a seguir pertencem a M ?
- 6
 - 9
 - 16
 - 21
 - 26
 - 54
 - 72
 - 218
- ★40. Uma coleção S de cadeias de caracteres é definida por recorrência por:
- a e b pertencem a S .
 - Se X pertence a S , então Xb também pertence.
- Quais das cadeias a seguir pertencem a S ?
- a
 - ab
 - aba
 - $aaab$
 - $bbbb$
41. Uma coleção W de cadeias de símbolos é definida por recorrência por:
- a , b e c pertencem a W .
 - Se X pertence a W , então $a(X)c$ também pertence.
- Quais das cadeias a seguir pertencem a W ?
- $a(b)c$
 - $a(a(b)c)c$
 - $a(abc)c$
 - $a(a(a(a)c)c)c$
 - $a(aacc)c$
- ★42. Dê uma definição recorrente para o conjunto de todas as fbsfs predicadas unárias em x .
43. Dê uma definição recorrente para o conjunto de todas as fórmulas bem formuladas de aritmética inteira envolvendo números inteiros e as operações aritméticas $+$, $-$, $*$ e $/$.
44. Dê uma definição recorrente para o conjunto de todas as cadeias de parênteses bem balanceados.
45. Dê uma definição recorrente para o conjunto de todas as cadeias binárias contendo um número ímpar de elementos iguais a 0.
- ★46. Dê uma definição recorrente para x^R , a cadeia em ordem inversa da cadeia x .
47. Dê uma definição recorrente para $|x|$, o comprimento da cadeia x .
- ★48. Use a notação FBN para definir o conjunto dos inteiros positivos.
49. 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.
- ★50. Dê uma definição recorrente para a operação factorial $n!$ para $n \geq 1$.
51. Dê uma definição recorrente para a adição de dois inteiros não-negativos m e n .
52. 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 $a_1, \dots, a_n, n \geq 2$.
53. 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.
- ★54. 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 $n \geq 2$.

55. 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 56 a 61, escreva o corpo de uma função recorrente para calcular $S(n)$, onde S é a seqüência dada.

56. 1, 3, 9, 27, 81, ...

57. 2, 1, 1/2, 1/4, 1/8, ...

★58. 1, 2, 4, 7, 11, 16, 22,

59. 2, 4, 16, 256, ...

60. $a, b, a + b, a + 2b, 2a + 3b, 3a + 5b, \dots$ ★61. $p, p - q, p + q, p - 2q, p + 2q, p - 3q, \dots$ 62. Qual o valor retornado pela função recorrente Mistério para um valor de entrada n ?Mistério (inteiro n)se $n = 1$ então

retorne 1

senão

retorne Mistério($n - 1$) + 1

fim do se

fim da função Mistério

63. 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 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

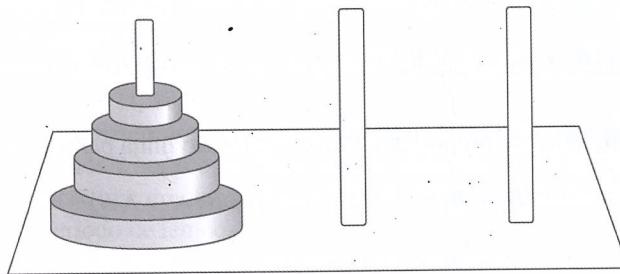
fim da função g

★64. Descreva informalmente um algoritmo recorrente que inverte a ordem dos elementos em uma lista de itens.

65. Descreva informalmente um algoritmo recorrente para calcular a soma dos dígitos de um inteiro positivo.

66. Descreva informalmente um algoritmo recorrente para calcular o máximo divisor comum de dois inteiros positivos a e b , onde $a \geq b$. (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.)

67. 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.



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

$4, 10, -6, 2, 5$

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

$9, 0, 2, 6, 4$

70. 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

71. 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

72. Demonstre a correção da função iterativa dada nesta seção para calcular $S(n)$ do Exemplo 29, onde $S(n) = 2^n$.

Nos Exercícios 73 a 78, resolva a relação de recorrência sujeita à condição básica.

★73. $S(1) = 5$

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

74. $F(1) = 2$

$F(n) = 2F(n - 1) + 2^n$ para $n \geq 2$

75. $T(1) = 1$

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

(*Sugestão:* veja Exemplo 15.)

★76. $A(1) = 1$

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

(*Sugestão:* veja Problema Prático 7.)

77. $S(1) = 1$

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

(*Sugestão:* veja Exemplo 14.)

78. $P(1) = 2$

$P(n) = 2P(n - 1) + n2^n$ for $n \geq 2$

(*Sugestão:* veja Problema Prático 7.)

Para os Exercícios 79 e 80, resolva a relação de recorrência sujeita à condição básica usando a abordagem de expandir, conjecturar e verificar. Note que a solução geral desenvolvida nesta seção não se aplica, já que a relação de recorrência não tem coeficientes constantes.

★79. $F(1) = 1$

$F(n) = nF(n - 1)$ para $n \geq 2$

80. $S(1) = 1$

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

81. Uma colônia de morcegos é contada a cada dois meses. As quatro primeiras contagens foram de 1.200, 1.800, 2.700 e 4.050. Se essa taxa de crescimento continuar, qual será a 12.^a contagem? (*Sugestão:* escreva e resolva uma relação de recorrência.)

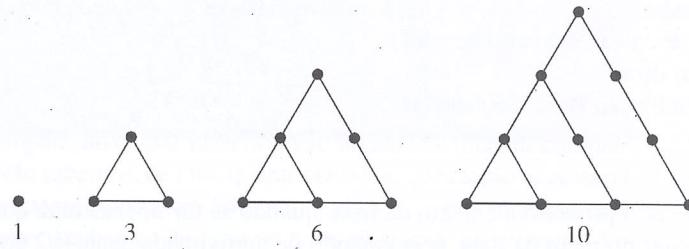
★82. No início deste capítulo, o empreiteiro afirmou:

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

Escreva e resolva uma relação de recorrência para verificar a afirmação do empreiteiro; note que ao final de 20 anos começa o vigésimo primeiro ano.

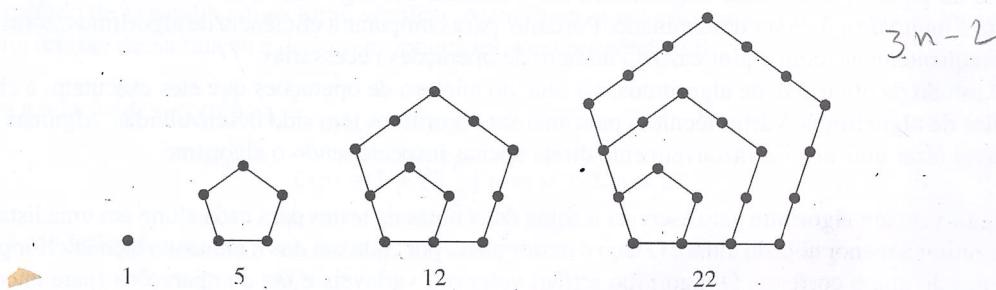
83. Investe-se R\$1.000,00 em uma aplicação que paga 8% de juros ao ano. Ao final de cada ano, aplique-se mais R\$100,00. Qual o total da aplicação ao final de 7 anos (isto é, no início do oitavo ano)? (*Sugestão:* escreva e resolva uma relação de recorrência. Consulte, também, o Exercício 23 da Seção 2.2 para a fórmula da soma de uma progressão geométrica.)
84. Um empréstimo de R\$5.000,00 paga juros de 12% ao ano. É feito um pagamento de R\$80,00 por mês. Quanto falta pagar ao final do 18.º mês (isto é, no início do 19.º mês)? (*Sugestão:* escreva e resolva uma relação de recorrência. Consulte, também, o Exercício 23 da Seção 2.2 para a fórmula da soma de uma progressão geométrica.)
85. 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. Após 9 anos, isto é, no início do décimo ano, qual a população aproximada de mariscos? (*Sugestão:* escreva e resolva uma relação de recorrência. Consulte, também, o Exercício 23 da Seção 2.2 para a fórmula da soma de uma progressão geométrica.)
86. Um vírus de computador que se prolifera por mensagens de correio eletrônico (*e-mail*) é colocado em 3 máquinas no primeiro dia. Cada dia, cada computador infectado no dia anterior infecta 5 novas máquinas. No segundo dia, é desenvolvido um programa para atacar o vírus e limpá-lo 1 computador. Cada dia após esse, são limpas 6 vezes mais máquinas do que foram limpas no dia anterior. Quantos dias vão se passar até os efeitos do vírus estarem completamente eliminados?
87. Esse problema está relacionado com o quebra-cabeças da Torre de Hanói (Exercício 67).
- Baseado no algoritmo recorrente do Exercício 67, 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. Consulte o Exercício 23 da Seção 2.2 para a fórmula da soma de uma progressão geométrica.
 - 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 terminar. Supondo que os monges movem um disco por segundo, calcule o número de anos necessários para completar a tarefa.

★88. Os primeiros membros da Associação de Pitágoras definiram *números poligonais* como sendo 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.)

89. Os primeiros *números pentagonais* (veja o Exercício 88) são 1, 5, 12 e 22:



Encontre e resolva uma relação de recorrência para o n -ésimo número pentagonal. (Sugestão: veja o Exercício 24 da Seção 2.2 para a fórmula da soma de uma progressão aritmética.)

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

SEÇÃO 2.5 ANÁLISE DE ALGORITMOS

Muitas vezes existe mais de um algoritmo para executar a mesma tarefa. Ao comparar algoritmos, pode-se usar diversos critérios para se decidir qual o “melhor”. Poderíamos perguntar, por exemplo, qual o mais fácil de entender ou qual o mais eficiente. Um modo de julgar a eficiência de um algoritmo é estimar o número de operações que ele tem que executar. Contamos apenas as operações básicas para a tarefa em questão e não operações de “manutenção”, que contribuem pouco para o “trabalho” total necessário.

Por exemplo, suponha que a tarefa é procurar, em uma lista ordenada contendo n itens, um item particular x . Como qualquer algoritmo possível parece necessitar a comparação de elementos na lista com x até se encontrar um elemento igual, a operação básica a ser contada é a comparação.

O **algoritmo de busca seqüencial** compara, simplesmente, com cada elemento da lista até se encontrar x ou acabar a lista. A seguir, fornecemos uma descrição em pseudocódigo para o algoritmo de busca seqüencial. O número de operações executadas por esse algoritmo vai ser máximo quando x é o último elemento da lista ou quando x não pertence à lista. Em qualquer desses casos, todos os elementos são comparados com x , logo faz-se n comparações.

```

ALGORITMO BUSCASEQÜENCIAL
BuscaSeqüencial(lista  $L$ ; inteiro  $n$ ; tipo item  $x$ )
//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 BuscaSeqüencial

```

É claro que x pode ser o primeiro elemento da lista, quando se faz apenas uma comparação; também pode acontecer de x estar no meio da lista, necessitando de aproximadamente $n/2$ comparações. Obviamente existem muitas possibilidades e ajudaria se pudéssemos ter alguma idéia da quantidade média de operações necessárias. Isso necessitaria de alguma descrição da lista média e da relação média entre x e os itens daquela lista. Os Exercícios 18 e 19 desta seção exploram alguns aspectos da análise para o caso médio do algoritmo de busca seqüencial. Para a maioria dos algoritmos, no entanto, o comportamento médio é muito difícil de ser determinado. Portanto, para comparar a eficiência de algoritmos, contentamo-nos freqüentemente com o pior caso do número de operações necessárias.

O estudo da eficiência de algoritmos, ou seja, do número de operações que eles executam, é chamado **análise de algoritmos**. Várias técnicas para analisar algoritmos têm sido desenvolvidas. Algumas vezes é possível fazer uma análise razoavelmente direta apenas inspecionando o algoritmo.

A seguir vem um algoritmo para escrever a soma de m notas de testes para cada aluno em uma lista depois de se retirar a menor nota do aluno. O laço exterior passa por cada um dos n alunos; o laço interior percorre as notas do aluno corrente. O algoritmo atribui valores a variáveis e faz comparações (para encontrar a

❖ EXEMPLO 47

menor nota de cada aluno). Também executa somas e subtrações, e contaremos o número dessas operações aritméticas.

```

para  $i = 1$  até  $n$  faca
    menor = aluno[ $i$ ].teste[1]
    soma = aluno[ $i$ ].teste[1]
    para  $j = 2$  até  $m$  faca
        soma = soma + aluno[ $i$ ].teste[ $j$ ]      //A
        se aluno[ $i$ ].teste[ $j$ ] < menor então
            menor = aluno[ $i$ ].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 com //S, que é executada uma vez em cada laço externo (para cada aluno), em um total de n vezes. A adição, no entanto, ocorre na linha marcada //A, no laço interno, que é executado $m - 1$ vezes para cada aluno, ou seja, para cada uma das n passagens do laço externo. O número total de adições, portanto, é $n(m - 1)$. O número total de operações aritméticas é $n + n(m - 1)$. O mesmo número de operações aritméticas é sempre feito; não existe caso melhor, nem pior, nem médio. ♦♦

Vamos analisar nesta seção algoritmos recorrentes. Como a maior parte da atividade de um algoritmo recorrente acontece “fora das vistas”, nas diversas chamadas que podem ocorrer, uma análise usando uma técnica de contagem direta como no Exemplo 47 não vai funcionar. A análise de algoritmos recorrentes envolve, muitas vezes, a resolução de uma relação de recorrência.

ANÁLISE USANDO RELAÇÕES DE RECORRÊNCIA (BUSCA BINÁRIA)

Como exemplo, vamos considerar um outro algoritmo (além da busca seqüencial) para se procurar em uma lista ordenada, o algoritmo de busca binária da Seção 2.4. Quantas comparações são necessárias, no pior caso, para o algoritmo de busca binária? Uma comparação é feita com o valor do meio da lista, depois o processo é repetido em metade da lista. Se a lista original tem n elementos, então a metade da lista tem, no máximo, $n/2$ elementos. (No Exemplo 41, por exemplo, onde 10 é o valor do meio, a “metade” da direita da lista tem 4 elementos, mas a “metade” da esquerda tem apenas 3.) Se vamos continuar cortando a lista pela metade, é conveniente considerar apenas o caso quando temos um número inteiro de elementos cada vez que dividimos ao meio a lista, de modo que vamos supor que $n = 2^m$ para algum $m \geq 0$. Se $C(n)$ denota o número máximo de comparações necessárias para uma lista de n elementos, então

$$C(n) = 1 + C\left(\frac{n}{2}\right)$$

Isso reflete uma comparação com o valor do meio seguida de quantas comparações forem necessárias para a metade da lista. Não sabemos, de fato, quantas comparações serão necessárias para metade da lista, assim como não sabemos quantas são necessárias para toda a lista, mas já temos uma notação para expressar esse valor simbolicamente. Acabamos de escrever uma relação de recorrência. A condição básica é que

$$C(1) = 1$$

já que é necessária apenas uma comparação em uma lista com um só elemento.

Como essa relação de recorrência não é de primeira ordem, para resolvê-la vamos começar do início com o método de expandir, conjecturar, verificar. Além disso, a solução vai envolver a função logaritmo; para uma revisão dessa função e de suas propriedades, veja o Apêndice B.

♦ EXEMPLO 48 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

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

e o termo geral parece ser

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

O processo pára quando $2^{k+1} = n$ ou $k = \log_2 n$. (Vamos omitir a base 2 a partir de agora — $\log n$ vai simbolizar $\log_2 n$.) Então

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

Vamos agora usar indução para mostrar que $C(n) = 1 + \log n$ para todo $n \geq 1$, $n = 2^m$. Essa é uma forma um pouco diferente de indução, já que todos os valores de interesse são potências de 2. Vamos considerar 1 como a base da indução, mas depois provaremos que, se a afirmação é verdadeira para k , então ela é verdadeira para $2k$. A proposição, então, será verdadeira para 1, 2, 4, 8, ..., isto é, para todas as potências inteiras não-negativas de 2, que é o que queremos.

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

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

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

Isso completa a demonstração por indução. ❖

Pelo Exemplo 48, 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 41, n era 8 e foram necessárias quatro comparações ($1 + \log 8$) no pior caso (x não pertencente à lista). Uma busca seqüencial 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 seqüencial. No entanto, o algoritmo de busca seqüencial tem uma grande vantagem — se a lista em questão *não está ordenada*, o algoritmo de busca seqüencial 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 10 a 17 ao final desta seção pedem que você conte as operações necessárias para ordenar uma lista por diversos algoritmos diferentes.

O algoritmo de busca binária é recorrente, o que significa que o algoritmo chama a si mesmo com valores de entrada menores. Nesse caso, a versão menor é muito menor — aproximadamente a metade do problema original. Isso fica claro na relação de recorrência, onde $C(n)$ depende de $C(n/2)$ e não de $C(n - 1)$. Algoritmos com relações de recorrência dessa forma, onde o problema é decomposto em subproblemas muito menores, são chamados algumas vezes de algoritmos do tipo **dividir para conquistar**.

A relação de recorrência para a busca binária é um caso particular da forma geral

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

onde c é uma constante e g pode ser uma expressão envolvendo n . Gostaríamos de encontrar uma solução em forma fechada para (1) sujeita à condição básica de que $S(1)$ é conhecido. Poderíamos então resolver

qualquer relação de recorrência da forma (1) substituindo na fórmula da solução, como fizemos na última seção para relações de recorrência de primeira ordem. Observe que (1) não é uma relação de recorrência de primeira ordem, já que o valor em n não depende do valor em $n - 1$. Poderíamos usar a abordagem de expandir, conjecturar e verificar, mas, ao invés disso, vamos fazer algumas transformações em (1) para convertê-la em uma relação de recorrência de primeira ordem com coeficientes constantes e depois usar o que já sabemos.

A Eq. (1) supõe que $n = 2^m$ com $n \geq 2$. Disso segue que $m = \log n$ e $m \geq 1$. Substituindo n por 2^m na Eq. (1) resulta em

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

Representando $S(2^m)$ por $T(m)$ na Eq. (2), obtemos

$$T(m) = cT(m-1) + g(2^m) \text{ para } m \geq 1 \quad (3)$$

A Eq. (3) é uma equação linear de primeira ordem com coeficientes constantes; da Eq. (8) na Seção 2.4 obtemos a solução

$$T(m) = c^{m-1}T(1) + \sum_{i=2}^m c^{m-i}g(2^i) \quad (4)$$

sujeita à condição básica de que $T(1)$ é conhecido. Como a Eq. (3) é válida para $m = 1$, sabemos que

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

Substituindo em (4), obtemos

$$T(m) = c^m T(0) + \sum_{i=1}^m c^{m-i}g(2^i) \quad (5)$$

Voltando à notação anterior, usando que $T(m) = S(2^m)$, (5) 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

$$\boxed{S(n) = c^{\log n} S(1) + \sum_{i=1}^{\log n} c^{(\log n)-i} g(2^i)} \quad (6)$$

A Eq. (6) é, então, a solução da relação de recorrência (1). Como anteriormente, para usar essa solução geral basta colocar sua relação de recorrência na forma (1) para determinar c e $g(n)$, depois substituir na Eq. (6). Ainda como antes, $g(n)$ dá uma receita para o que fazer com um argumento n ; na Eq. (6), o argumento é 2^i . Se você puder calcular o somatório, o resultado será uma solução em forma fechada.

♦ EXEMPLO 49

A relação de recorrência para o algoritmo de busca binária é

$$C(1) = 1$$

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

que é da forma da Eq. (1) acima com $c = 1$ e $g(n) = 1$. Como $g(n) = 1$, a função g é sempre igual a 1, independente de seu argumento. A solução, de acordo com a fórmula (6), é

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

o que está de acordo com nosso resultado anterior.

♦ EXEMPLO 50

Resolva a relação de recorrência

$$T(1) = 3$$

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

Elá é do mesmo tipo que a Eq. (1) com $c = 2$ e $g(n) = 2n$. Logo, $g(2^i) = 2(2^i)$. Substituindo na solução da Eq. (6), obtemos o resultado a seguir, onde usamos o fato de que $2^{\log n} = n$.

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

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 e note que $2^{\log n} = n$.)

PROBLEMA PRÁTICO 23

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 não-negativos a e b , $a \geq b$. Para analisar o algoritmo de Euclides, precisamos decidir primeiro que operações contaremos. Como esse algoritmo efetua repetidas divisões, vamos considerar a operação de divisão como nossa unidade básica. Dados a e b , suponha que $b \leq a = n$, de modo que n é uma medida do tamanho dos valores de entrada. Queremos encontrar $E(n)$, que denota a quantidade de operações (o número de divisões) necessária para encontrar $\text{mdc}(a, b)$ no pior caso.

Uma versão recorrente do algoritmo de Euclides também pode ser escrita (veja o Exercício 66, Seção 2.4); a chave para a versão recorrente é reconhecer que o cálculo do $\text{mdc}(a, b)$ envolve encontrar o $\text{mdc}(b, r)$, no qual r é o resto da divisão de a por b . Acabamos de ver um caso onde as operações de um algoritmo recorrente (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 $E(n)$ 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 no qual, para encontrar o $\text{mdc}(420, 66)$, foram efetuadas as seguintes divisões:

$$\begin{array}{r} 420 \mid 66 \\ -396 \quad 6 \\ \hline 24 \end{array} \quad \begin{array}{r} 66 \mid 24 \\ -48 \quad 2 \\ \hline 18 \end{array} \quad \begin{array}{r} 24 \mid 18 \\ -18 \quad 1 \\ \hline 6 \end{array} \quad \begin{array}{r} 18 \mid 6 \\ -18 \quad 3 \\ \hline 0 \end{array}$$

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 que a mudança de 24 para 18 é menor.

De fato, não encontraremos uma relação de recorrência ou uma expressão exata para $E(n)$. Mas podemos, pelo menos, encontrar uma *cota superior* para $E(n)$. Uma *cota superior* é um valor superior para a quantidade de operações efetuadas por um algoritmo; o algoritmo *não* pode 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 n pode ser dividido por dois $\log n$ vezes; portanto, são feitas, no máximo, $2 \log n$ divisões. Assim,

$$E(n) \leq 2 \log n \tag{7}$$

O valor de $2 \log n$ para $n = 420$ é quase 18, ao passo que foram necessárias apenas 4 divisões para se obter o mdc(420, 66). É claro que essa cota superior é bastante grosseira, algo como dizer que todos os aluños em sala têm menos de três metros é meio de altura. Uma cota superior mais fina (isto é, mais baixa) é obtida nos Exercícios 20 a 22 ao final desta seção.

SEÇÃO 2.5 REVISÃO

TÉCNICA

- W** ♦ Resolução de relações de recorrência obtidas de algoritmos do tipo dividir para conquistar usando-se uma fórmula para a solução.

IDÉIAS PRINCIPAIS

- ♦ A análise de um algoritmo estima o número de operações básicas que o algoritmo efetua.
 ♦ A análise de algoritmos recorrentes leva, com freqüência, 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 2.5

1. O algoritmo a seguir soma todos os elementos de uma matriz quadrada A $n \times n$. Encontre uma expressão em termos de n para o número de somas executadas.

```
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)
```

2. O algoritmo a seguir soma todos os elementos na parte “triangular superior” de uma matriz quadrada A $n \times n$. Encontre uma expressão em termos de n para o número de somas executadas.

```
soma = 0
para k = 1 até n faça
  para j = k até n faça
    soma = soma + A[j, k]
  fim do para
fim do para
escreva ("A soma total dos elementos na parte triangular superior da matriz é", soma)
```

3. Para o algoritmo do Exemplo 47, conte o número total de atribuições e comparações feitas no melhor dos casos (número mínimo de operações) e no pior dos casos (número máximo de operações); descreva cada um desses casos.
- ★4. Descreva uma versão recorrente do algoritmo de busca seqüencial para encontrar o item x em uma lista L contendo n itens.
5. Usando a versão recorrente do Exercício 4, escreva uma relação de recorrência para o número de comparações de x com os elementos da lista feitas pelo algoritmo de busca seqüencial no pior caso e resolva essa relação de recorrência. (Como antes, a resposta deve ser n .)

Nos Exercícios 6 a 9, resolva a relação de recorrência dada sujeita à condição básica. (Sugestão: veja o Exemplo 15 e observe que $2^{\log n} = n$.)

6. $T(1) = 3$
 $T(n) = T\left(\frac{n}{2}\right) + n$ para $n \geq 2$, $n = 2^m$

★7. $P(1) = 1$

se $i > m$ then
 não achou
 else
 if $L(i) = x$ then
 escreva achou
 else
 sequencial $[L, i+1, n]$

$$P(n) = 2P\left(\frac{n}{2}\right) + 3$$

8. $S(1) = 1$

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

9. $P(1) = 1$

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

Os Exercícios 10 a 12 se referem ao algoritmo *OrdenaçãoPorSeleção* da Seção 2.4.

- ★10. Em uma parte do algoritmo *OrdenaçãoPorSeleção*, é preciso encontrar o índice do maior item 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 maior elemento? Quantas comparações são necessárias em média?
11. Definindo a operação básica como sendo 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 10.)
12. Resolva a relação de recorrência do Exercício 11.

Os Exercícios 13 a 17 estão relacionados com um algoritmo 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 e depois combine as duas listas em uma única lista ordenada.

- ★13. A parte de fusão do algoritmo *OrdenaçãoPorFusão* precisa que se compare os elementos de cada uma das listas ordenadas para ver qual o próximo elemento na lista combinada e 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.
- 6, 8, 9 e 1, 4, 5
 - 1, 5, 8 e 2, 3, 4
 - 0, 2, 3, 4, 7, 10 e 1, 8, 9
14. Sob que circunstâncias teremos que executar o número máximo de comparações ao se fundir duas listas ordenadas? Se as duas listas têm comprimento r e s , qual é o número máximo de comparações?
- ★15. Escreva uma relação de recorrência para o número de comparações entre os elementos da lista efetuadas pelo algoritmo *OrdenaçãoPorFusão* no pior caso. Suponha que $n = 2^m$.
16. Resolva a relação de recorrência do Exercício 15.
17. Compare o comportamento, no pior caso, dos algoritmos *OrdenaçãoPorSeleção* e *OrdenaçãoPorFusão* para $n = 4, 8, 16$ e 32 (use uma calculadora).

Os Exercícios 18 e 19 referem-se ao algoritmo *BuscaSeqüencial*. Não é difícil fazer uma análise do caso médio do algoritmo de busca seqüencial sob determinadas hipóteses. Dada uma lista com n elementos e um item x a ser pesquisado, a operação básica é a comparação de elementos na lista com x ; portanto, uma análise deveria contar quantas vezes é efetuada uma tal operação “em média”. A definição de “média” depende de nossas hipóteses.

18. 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 Onde Está 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.)

19. 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 não estar na lista.

Os Exercícios 20 a 22 procuram uma cota superior melhor para o número de divisões necessárias para que o algoritmo de Euclides encontre o $\text{mdc}(a, b)$. Suponha que a e b são inteiros não-negativos e que $a > b$. (Se $a = b$, é necessária apenas uma divisão, de modo que esse caso corresponde ao menor número de operações; queremos uma cota superior para o número máximo de operações.)

20. 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 $b \geq F(m + 1)$, onde $F(n)$ é a seqüência de Fibonacci. (*Sugestão:* para encontrar $\text{mdc}(a, b)$, o algoritmo calcula $\text{mdc}(b, r)$ depois da primeira divisão.)
- ★21. Suponha que são necessárias m divisões para encontrar o $\text{mdc}(a, b)$, com $m \geq 4$, e que $a = n$. Prove que

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

(*Sugestão:* use o resultado do Exercício 20 desta seção e o do Exercício 23 da Seção 2.4.)

22. Suponha que são necessárias m divisões para encontrar o $\text{mdc}(a, b)$, com $m \geq 4$, e que $a = n$. Prove que

$$m < (\log_{1.5} n) - 1$$

(*Sugestão:* use o resultado do Exercício 21.)

23. 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 Eq. (7).
 c. Calcule uma cota superior para o número de divisões necessárias para calcular o $\text{mdc}(89, 55)$ usando o resultado do Exercício 22.

REVISÃO DO CAPÍTULO 2

Terminologia

- algoritmo de busca binária (Seção 2.4)
- algoritmo de busca seqüencial (Seção 2.5)
- algoritmo de Euclides (Seção 2.3)
- algoritmo de ordenação por seleção (Seção 2.4)
- algoritmo do tipo dividir para conquistar (Seção 2.5)
- análise de algoritmo (Seção 2.5)
- base da indução (Seção 2.2)
- cadeia binária (Seção 2.4)
- cadeia vazia (Seção 2.4)
- concatenação (Seção 2.4)
- contra-exemplo (Seção 2.1)
- contrapositiva (Seção 2.1)
- correção parcial (Seção 2.3)
- cota superior (Seção 2.5)
- definição por indução (Seção 2.4)
- definição por recorrência (Seção 2.4)
- demonstração direta (Seção 2.1)
- demonstração por absurdo (Seção 2.1)
- demonstração por casos (Seção 2.1)
- demonstração por contraposição (Seção 2.1)
- demonstração por exaustão (Seção 2.1)
- forma de Backus-Naur (FBN) (Seção 2.4)
- hipótese de indução (Seção 2.2)
- hipótese de indução (Seção 2.2)
- índice do somatório (Seção 2.4)
- invariante do laço (Seção 2.3)