Capítulo 10

Estruturas de dados elementares

Neste capítulo, examinaremos a representação de conjuntos dinâmicos por estruturas de dados simples que usam ponteiros. Embora muitas estruturas de dados complexas possam ser modeladas com a utilização de ponteiros, apresentaremos apenas as estruturas rudimentares: pilhas, filas, listas ligadas e árvores enraizadas. Também discutiremos um método pelo qual objetos e ponteiros podem ser sintetizados a partir de arranjos.

10.1 Pilhas e filas

As pilhas e filas são conjuntos dinâmicos nos quais o elemento removido do conjunto pela operação DELETE é especificado previamente. Em uma ***pilha***, o elemento eliminado do conjunto é o mais recentemente inserido: a pilha implementa uma norma de ***último a entrar, primeiro a sair***, ou ***LIFO*** (last-in, first-out). De modo semelhante, em uma ***fila***, o elemento eliminado é sempre o que esteve no conjunto pelo tempo mais longo: a fila implementa uma norma de ***primeiro a entrar, primeiro a sair***, ou ***FIFO*** (first-in, first-out). Existem vários modos eficientes de implementar pilhas e filas em um computador. Nesta seção, mostraremos como usar um arranjo simples para implementar cada uma dessas estruturas.

Pilhas

A operação INSERT sobre uma pilha é chamada com freqüência PUSH, e a operação DELETE, que não toma um argumento de elemento, é freqüentemente chamada POP. Esses nomes são alusões a pilhas físicas, como as pilhas de pratos usados em restaurantes. A ordem em que os pratos são retirados da pilha é o oposto da ordem em que eles são colocados sobre a pilha e, como conseqüência, apenas o prato do topo está acessível.

Como mostra a Figura 10.1, podemos implementar uma pilha de no máximo n elementos com um arranjo $S[1..n]$. O arranjo tem um atributo $topo[S]$ que realiza a indexação do elemento inserido mais recentemente. A pilha consiste nos elementos $S[1 .. topo[S]]$, onde $S[1]$ é o elemento na parte inferior da pilha e $S[topo[S]]$ é o elemento na parte superior (ou no topo).

Quando $topo[S] = 0$, a pilha não contém nenhum elemento e está ***vazia***. É possível testar se a pilha está vazia, através da operação de consulta STACK-EMPTY. Se uma pilha vazia sofre uma operação de extração, dizemos que a pilha tem um ***estouro negativo***, que é normalmente um erro. Se $topo[S]$ excede n, a pilha tem um ***estouro positivo***. (Em nossa implementação de pseudocódigo, não nos preocuparemos com o estouro de pilhas.)

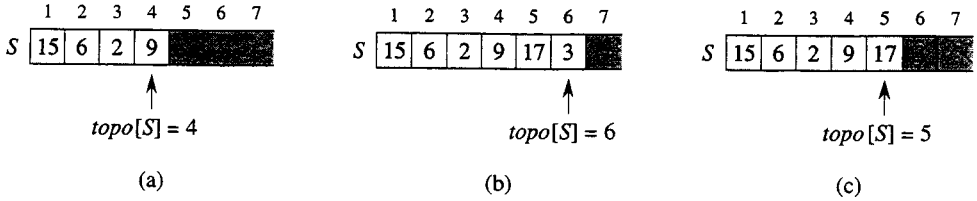

FIGURA 10.1 Uma implementação de arranjo de uma pilha S. Os elementos da pilha só aparecem nas posições levemente sombreadas. (**a**) A pilha S tem 4 elementos. O elemento do topo é 9. (**b**) A pilha S após as chamadas PUSH(S, 17) e PUSH(S, 3). (**c**) A pilha S após a chamada POP(S) retornou o elemento 3, que é o elemento mais recentemente inserido na pilha. Embora o elemento 3 ainda apareça no arranjo, ele não está mais na pilha; o elemento do topo é o elemento 17

Cada uma das operações sobre pilhas pode ser implementada com algumas linhas de código.

```
STACK-EMPTY(S)
1 if topo[S] = 0
2    then return TRUE
3    else return FALSE

PUSH(S, x)
1 topo[S] ← topo[S] + 1
2 S[topo[S]] ← x

POP(S)
1 if STACK-EMPTY(S)
2    then error "underflow"
3    else topo[S] ← topo[S] – 1
4         return S[topo[S] + 1]
```

A Figura 10.1 mostra os efeitos das operações de modificação PUSH (EMPILHAR) e POP (DESEMPILHAR). Cada uma das três operações sobre pilhas demora o tempo $O(1)$.

Filas

Chamamos a operação INSERT sobre uma fila de ENQUEUE (ENFILEIRAR), e também a operação DELETE de DEQUEUE (DESINFILEIRAR); como a operação sobre pilhas POP, DEQUEUE não tem nenhum argumento de elemento. A propriedade FIFO de uma fila faz com que ela opere como uma fileira de pessoas no posto de atendimento da previdência social. A fila tem um ***início*** (ou cabeça) e um ***fim*** (ou cauda). Quando um elemento é colocado na fila, ele ocupa seu lugar no fim da fila, como um aluno recém-chegado que ocupa um lugar no final da fileira. O elemento retirado da fila é sempre aquele que está no início da fila, como o aluno que se encontra no começo da fileira e que esperou por mais tempo. (Felizmente, não temos de nos preocupar com a possibilidade de elementos computacionais "furarem" a fila.)

A Figura 10.2 mostra um modo de implementar uma fila de no máximo $n - 1$ elementos usando um arranjo $Q[1 .. n]$. A fila tem um atributo $início[Q]$ que indexa ou aponta para seu início. O atributo $fim[Q]$ realiza a indexação da próxima posição na qual um elemento recém-chegado será inserido na fila. Os elementos na fila estão nas posições $início[Q]$, $início[Q] + 1$, ..., $fim[Q] - 1$, onde "retornamos", no sentido de que a posição 1 segue imediatamente a posição n em uma ordem circular. Quando $início[Q] = fim[Q]$, a fila está vazia. Inicialmente, temos $início[Q] = fim[Q] = 1$. Quando a fila está vazia, uma tentativa de retirar um elemento da fila provoca o estouro negativo da fila. Quando $início[Q] = fim[Q] + 1$, a fila está cheia, e uma tentativa de colocar um elemento na fila provoca o estouro positivo da fila.

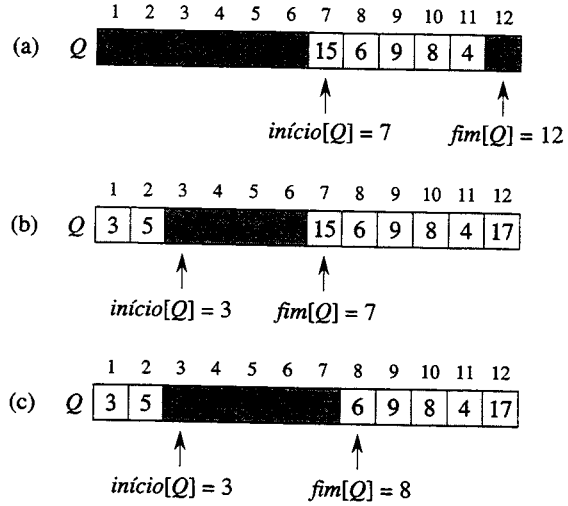

FIGURA 10.2 Uma fila implementada com a utilização de um arranjo $Q[1..12]$. Os elementos da fila aparecem apenas nas posições levemente sombreadas. (**a**) A fila tem 5 elementos, nas localizações $Q[7..11]$. (**b**) A configuração da fila depois das chamadas ENQUEUE(Q, 17), ENQUEUE(Q, 3) e ENQUEUE(Q, 5). (**c**) A configuração da fila depois da chamada DEQUEUE(Q) retorna o valor de chave 15 que se encontrava anteriormente no início da fila. O novo início tem a chave 6

Em nossos procedimentos ENQUEUE e DEQUEUE, a verificação de erros de estouro negativo (underflow) e estouro positivo (overflow) foi omitida. (O Exercício 10.1-4 lhe pede para fornecer o código que efetua a verificação dessas duas condições de erro.)

```
ENQUEUE(Q, x)
1 Q[fim[Q]] ← x
2 if fim[Q] = comprimento[Q]
3    then fim[Q] ← 1
4    else fim[Q] ← fim[Q] + 1
```

```
DEQUEUE(Q)
1 x ← Q[início[Q]]
2 if início[Q] = comprimento[Q]
3    then início[Q] ← 1
4    else início[Q] ← início[Q] + 1
5 return x
```

A Figura 10.2 mostra os efeitos das operações ENQUEUE e DEQUEUE. Cada operação demora o tempo $O(1)$.

Exercícios

10.1-1

Usando a Figura 10.1 como modelo, ilustre o resultado de cada operação na seqüência PUSH(S, 4), PUSH(S, 1), PUSH(S, 3), POP(S), PUSH(S, 8) e POP(S) sobre uma pilha S inicialmente vazia armazenada no arranjo $S[1 .. 6]$.

10.1-2

Explique como implementar duas pilhas em um único arranjo $A[1 .. n]$ de tal modo que nenhuma das pilhas sofra um estouro positivo, a menos que o número total de elementos em ambas as pilhas juntas seja n. As operações PUSH e POP devem ser executadas no tempo $O(1)$.

10.1-3
Usando a Figura 10.2 como modelo, ilustre o resultado de cada operação na seqüência ENQUEUE(Q, 4), ENQUEUE(Q, 1), ENQUEUE(Q, 3), DEQUEUE(Q), ENQUEUE(Q, 8) e DEQUEUE(Q) sobre uma fila Q inicialmente vazia armazenada no arranjo $Q[1 .. 6]$.

10.1-4
Reescreva ENQUEUE e DEQUEUE para detectar o estouro negativo e o estouro positivo de uma fila.

10.1-5
Enquanto uma pilha permite a inserção e a eliminação de elementos em apenas uma extremidade e uma fila permite a inserção em uma extremidade e a eliminação na outra extremidade, uma ***deque*** (double-ended queue, ou fila de extremidade dupla) permite a inserção e a eliminação em ambas as extremidades. Escreva quatro procedimentos de tempo $O(1)$ para inserir elementos e eliminar elementos de ambas as extremidades de uma deque construída a partir de um arranjo.

10.1-6
Mostre como implementar uma fila usando duas pilhas. Analise o tempo de execução das operações sobre filas.

10.1-7
Mostre como implementar uma pilha usando duas filas. Analise o tempo de execução das operações sobre pilhas.

10.2 Listas ligadas

Uma ***lista ligada*** é uma estrutura de dados em que os objetos estão organizados em uma ordem linear. Entretanto, diferente de um arranjo, no qual a ordem linear é determinada pelos índices do arranjo, a ordem em uma lista ligada é determinada por um ponteiro em cada objeto. As listas ligadas fornecem uma representação simples e flexível para conjuntos dinâmicos, admitindo (embora não necessariamente de modo eficiente) todas as operações listadas na introdução à Parte III, seção "Operações sobre conjuntos dinâmicos".

Como mostra a Figura 10.3, cada elemento de uma ***lista duplamente ligada*** L é um objeto com um campo de *chave* e dois outros campos de ponteiros: *próximo* e *anterior*. O objeto também pode conter outros dados satélite. Sendo dado um elemento x na lista, *próximo*$[x]$ aponta para seu sucessor na lista ligada, e *anterior*$[x]$ aponta para seu predecessor. Se *anterior*$[x]$ = NIL, o elemento x não tem nenhum predecessor e portanto é o primeiro elemento, ou ***início***, da lista. Se *próximo*$[x]$ = NIL, o elemento x não tem nenhum sucessor e assim é o último elemento, ou ***fim***, da lista. Um atributo *início*$[L]$ aponta para o primeiro elemento da lista. Se *início*$[L]$ = NIL, a lista está vazia.

Uma lista pode ter uma entre várias formas. Ela pode ser simplesmente ligada ou duplamente ligada, pode ser ordenada ou não, e pode ser circular ou não. Se uma lista é ***simplesmente ligada***, omitimos o ponteiro *anterior* em cada elemento. Se uma lista é ***ordenada***, a ordem linear da lista corresponde à ordem linear de chaves armazenadas em elementos da lista; o elemento mínimo é o início da lista, e o elemento máximo é o fim. Se a lista é ***não ordenada***, os elementos podem aparecer em qualquer ordem. Em uma ***lista circular***, o ponteiro *anterior* do início da lista aponta para o fim, e o ponteiro *próximo* do fim da lista aponta para o início. Desse modo, a lista pode ser vista como um anel de elementos. No restante desta seção, supomos que as listas com as quais estamos trabalhando são listas não ordenadas e duplamente ligadas.