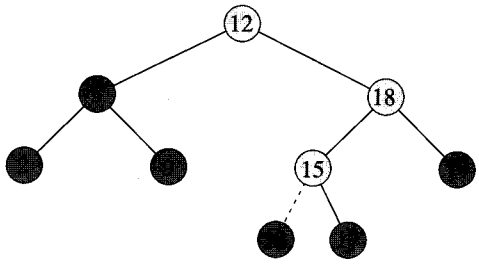

Figura 12.3 L'elemento con la chiave 13 viene inserito in un albero binario di ricerca. I nodi più chiari indicano il cammino semplice dalla radice verso la posizione dove l'elemento viene inserito. La linea tratteggiata indica il collegamento che è stato aggiunto per inserire l'elemento.

```
TREE-INSERT(T, z)
 1  y = NIL
 2  x = T.root
 3  while x ≠ NIL
 4      y = x
 5      if z.key < x.key
 6          x = x.left
 7      else x = x.right
 8  z.p = y
 9  if y == NIL
10      T.root = z          // L'albero T era vuoto
11  elseif z.key < y.key
12      y.left = z
13  else y.right = z
```

Cancellazione

La procedura per cancellare un nodo z da un albero binario di ricerca considera tre casi base ma, come vedremo, uno solo di essi è complicato.

- Se z non ha figli, modifichiamo suo padre $p[z]$ per sostituire z con NIL come suo figlio.
- Se il nodo z ha un solo figlio, eleviamo questo figlio in modo da occupare la posizione di z nell'albero, modificando il padre di z per sostituire z con il figlio di z.
- Se il nodo z ha due figli, troviamo il successore y di z – che deve trovarsi nel sottoalbero destro di z – e facciamo in modo che y assuma la posizione di z nell'albero. La parte restante del sottoalbero destro originale diventa il nuovo sottoalbero destro di y e il sottoalbero sinistro di z diventa il nuovo sottoalbero sinistro di y. Questo è il caso complicato perché, come vedremo, occorre considerare il caso in cui y è figlio destro di z.

La procedura per cancellare un dato nodo z da un albero binario di ricerca T richiede come argomenti i puntatori a T e z. La procedura organizza i casi in modo leggermente diverso dai tre appena descritti e considera i quattro casi mostrati nella Figura 12.4.

- Se z non ha un figlio sinistro (parte (a) della figura), sostituiamo z con il suo figlio destro, che può essere NIL oppure no. Se il figlio destro di z è NIL, si ha il caso in cui z non ha figli. Se il figlio destro di z non è NIL, si ha il caso in cui z ha un solo figlio, che è il suo figlio destro.

- Se z ha un solo figlio, che è il suo figlio sinistro (parte (b) della figura), sostituiamo z con il suo figlio sinistro.
- Altrimenti, z ha un figlio sinistro e un figlio destro. Troviamo il successore y di z, che si trova nel sottoalbero destro di z e non ha un figlio sinistro (vedere l'Esercizio 12.2-5). Vogliamo staccare y dalla sua posizione corrente per metterlo al posto di z nell'albero.
  - Se y è il figlio destro di z (parte (c)), sostituiamo z con y, lasciando a y soltanto il figlio destro.
  - Altrimenti, y si trova nel sottoalbero destro di z, ma non è il figlio destro di z (parte (d)). In questo caso, sostituiamo prima y con il suo figlio destro; poi sostituiamo z con y.

Per poter spostare i sottoalberi all'interno dell'albero binario di ricerca, abbiamo definito una subroutine TRANSPLANT, che sostituisce un sottoalbero, come figlio di suo padre, con un altro sottoalbero. Quando TRANSPLANT sostituisce il sottoalbero con radice nel nodo u con il sottoalbero con radice nel nodo v, il padre del nodo u diventa il padre del nodo v, e il padre di u alla fine ha v come figlio.

```
TRANSPLANT(T, u, v)
1  if u.p == NIL
2      T.root = v
3  elseif u == u.p.left
4      u.p.left = v
5  else u.p.right = v
6  if v ≠ NIL
7      v.p = u.p
```

Le righe 1–2 gestiscono il caso in cui u è la radice di T. Altrimenti, u è un figlio sinistro o un figlio destro di suo padre. Le righe 3–4 aggiornano $u.p.left$ se u è un figlio sinistro; la riga 5 aggiorna $u.p.right$ se u è un figlio destro. Poiché v può essere NIL, le righe 6–7 aggiornano $v.p$ se v non è NIL. Si noti che TRANSPLANT non aggiorna $v.left$ e $v.right$; fare ciò, o non farlo, è compito della procedura che chiama TRANSPLANT.

Disponendo della procedura TRANSPLANT, ecco la procedura che cancella il nodo z dall'albero binario di ricerca T:

```
TREE-DELETE(T, z)
 1  if z.left == NIL
 2      TRANSPLANT(T, z, z.right)
 3  elseif z.right == NIL
 4      TRANSPLANT(T, z, z.left)
 5  else y = TREE-MINIMUM(z.right)
 6      if y.p ≠ z
 7          TRANSPLANT(T, y, y.right)
 8          y.right = z.right
 9          y.right.p = y
10      TRANSPLANT(T, z, y)
11      y.left = z.left
12      y.left.p = y
```

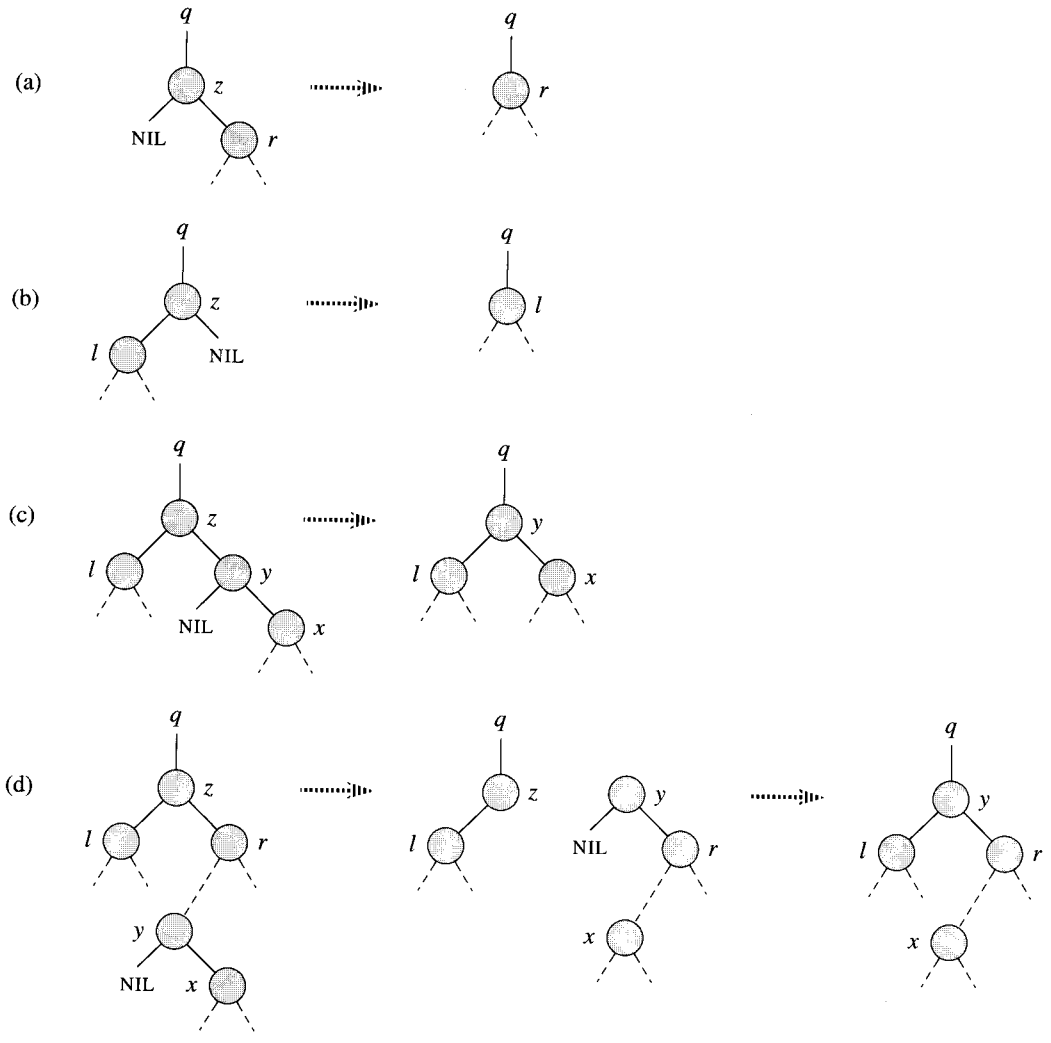

Figura 12.4 Cancellare un nodo z da un albero binario di ricerca. Il nodo z può essere la radice, un figlio sinistro del nodo q o un figlio destro di q. **(a)** Il nodo z non ha un figlio sinistro. Sostituiamo z con il suo figlio destro r, che può essere NIL oppure no. **(b)** Il nodo z ha un figlio sinistro l, ma non ha un figlio destro. Sostituiamo z con l. **(c)** Il nodo z ha due figli; il figlio sinistro è il nodo l, il figlio destro è il suo successore y, e il figlio destro di y è il nodo x. Sostituiamo z con y, modificando il figlio sinistro di y in modo che diventi l, ma lasciando x come figlio destro di y. **(d)** Il nodo z ha due figli (il figlio sinistro l e il figlio destro r), e il suo successore $y \neq r$ si trova nel sottoalbero con radice in r. Sostituiamo y con il suo figlio destro x e impostiamo y come padre di r. Poi, impostiamo y come figlio di q e padre di l.

La procedura TREE-DELETE esegue i quattro casi nel seguente modo. Le righe 1–2 gestiscono il caso in cui il nodo z non ha un figlio sinistro; le righe 3–4 gestiscono il caso in cui z ha un figlio sinistro, ma non un figlio destro. Le righe 5–12 trattano gli altri due casi, in cui z ha due figli. La riga 5 trova il nodo y, che è il successore di z. Poiché z ha un sottoalbero destro non vuoto, il suo successore deve essere il nodo di quel sottoalbero con la chiave più piccola; da qui la chiamata della procedura TREE-MINIMUM($z.right$). Come detto in precedenza, y non ha un figlio sinistro. Stacchiamo y dalla sua posizione corrente e mettiamolo al posto di z. Se y è il figlio destro di z, le righe 10–12 sostituiscono z, come figlio di suo

padre, con y; poi, sostituiscono il figlio sinistro di y con il figlio sinistro di z. Se y non è il figlio sinistro di z, le righe 7–9 sostituiscono y, come figlio di suo padre, con il figlio destro di y e cambiano il figlio destro di z nel figlio destro di y; le righe 10–12 sostituiscono z, come figlio di suo padre, con y e, infine, rimpiazzano il figlio sinistro di y con il figlio sinistro di z.

Ogni riga di TREE-DELETE, incluse le chiamate di TRANSPLANT, richiede un tempo costante, tranne la chiamata di TREE-MINIMUM nella riga 5. Quindi, la procedura TREE-DELETE viene eseguita nel tempo $O(h)$ in un albero di altezza h. In sintesi, abbiamo dimostrato il seguente teorema.

Teorema 12.3
Le operazioni sugli insiemi dinamici INSERT e DELETE possono essere svolte nel tempo $O(h)$ in un albero binario di ricerca di altezza h. ■

Esercizi

12.3-1
Scrivete una versione ricorsiva della procedura TREE-INSERT.

12.3-2
Supponete che un albero binario di ricerca sia costruito inserendo ripetutamente alcuni valori distinti nell'albero. Dimostrate che il numero di nodi esaminati durante la ricerca di un valore è pari a uno più il numero di nodi esaminati quando il valore fu inserito per la prima volta nell'albero.

12.3-3
Possiamo ordinare un dato insieme di n numeri costruendo prima un albero binario di ricerca che contiene questi numeri (usando ripetutamente TREE-INSERT per inserire i numeri uno alla volta) e stampando poi i numeri con un attraversamento simmetrico dell'albero. Quali sono i tempi di esecuzione nel caso peggiore e nel caso migliore per questo algoritmo di ordinamento?

12.3-4
L'operazione di cancellazione è "commutativa" nel senso che cancellare prima x e poi y da un albero binario di ricerca equivale a cancellare prima y e poi x? Spiegate perché sì oppure indicate un controesempio.

12.3-5
Supponete che ciascun nodo x, anziché includere l'attributo $x.p$, che punta al padre di x, includa $x.succ$, che punta al successore di x. Scrivete lo pseudocodice di SEARCH, INSERT e DELETE per un albero binario di ricerca T utilizzando questa rappresentazione. Queste procedure devono operare nel tempo $O(h)$, dove h è l'altezza dell'albero T (*suggerimento:* potreste implementare una subroutine che restituisce il padre di un nodo.)

12.3-6
Quando, nella procedura TREE-DELETE, il nodo z ha due figli possiamo scegliere il predecessore invece del successore come nodo y. Quali altre modifiche occorre apportare a TREE-DELETE se facciamo così? Alcuni ritengono che, con una strategia che scelga casualmente e con equità tra predecessore e successore, è possibile ottenere prestazioni empiriche migliori. Come dovrebbe essere modificata la procedura TREE-DELETE per implementare tale strategia?