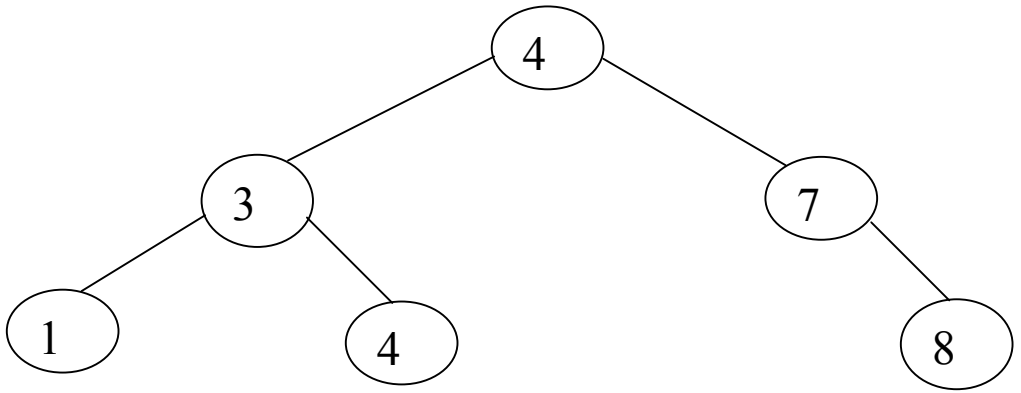
Albero binario di ricerca

Struttura di dati che può essere usata sia come dizionario, sia come coda con priorità

Proprietà: sia x un nodo di un (sotto)albero binario di ricerca

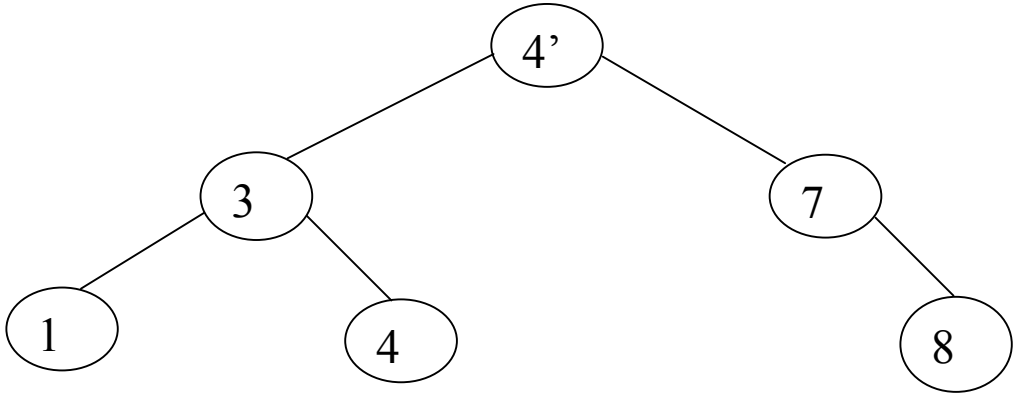
- Se y è un nodo del sottoalbero sx di x, allora key[y] ≤ key[x]
- Se y è un nodo del sottoalbero dx di x, allora key[y] ≥ key[x]

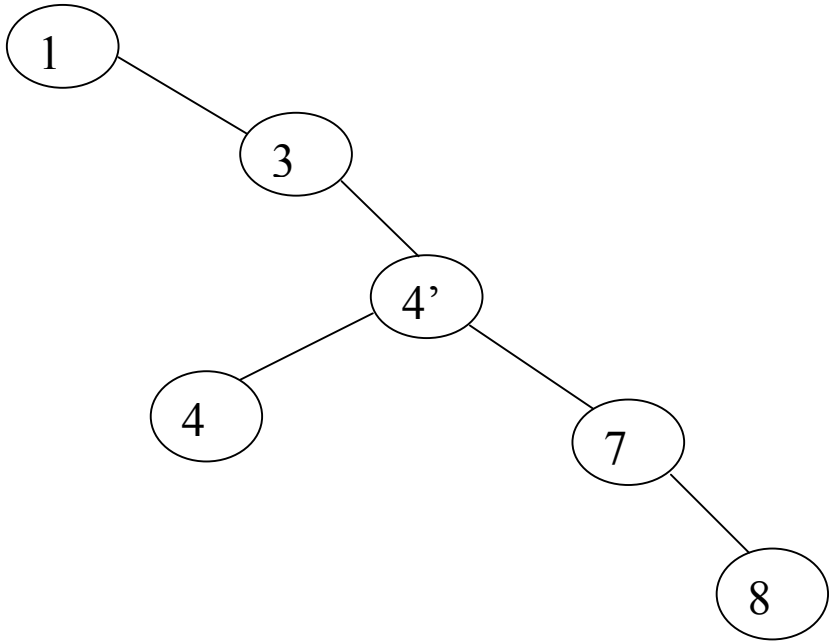
Esempio



Alberi binari di ricerca: visita

Due alberi binari di ricerca che rappresentano lo stesso insieme di valori





Visita in ordine		
simmetrico	1, 3, 4, 4’, 7, 8	1, 3, 4, 4’, 7, 8
anticipato	4’, 3, 1, 4, 7, 8	1, 3, 4’, 4, 7, 8
differito	1, 4, 3, 8, 7, 4’	4, 8, 7, 4’, 3, 1

Albero binario di ricerca: visita in ordine simmetrico

La chiave della radice di un (sotto)albero è elencata dopo i valori del suo sottoalbero sx e prima di quelli del suo sottoalbero dx

```
INORDER-TREE-WALK(x)
1 if x ≠ NIL
2    then    INORDER-TREE-WALK(left[x])
3            stampa key[x]
4            INORDER-TREE-WALK(right[x])
```

Per elencare tutti gli elementi di un albero binario di ricerca T di n nodi si invoca INORDER-TREE-WALK(root[T]), che impiega un tempo $T(n) = \Theta(n)$ perché la procedura è chiamata ricorsivamente esattamente due volte per ciascun nodo

Albero binario di ricerca: ricerca

(conforme alla successiva procedura di inserimento)

```
TREE-SEARCH(x, k)
1 if x = NIL o k = key[x]
2    then     return x
3    ▷ restituisce un puntatore a un nodo
     con chiave k, se ∃, NIL altrimenti
4 if k < key[x]
5    then     return TREE-SEARCH(left[x], k)
6    else     return TREE-SEARCH(right[x], k)
```

I nodi incontrati nella ricerca ricorsiva formano un cammino che discende dalla radice dell'albero → T(n) = O(h), dove h è l'altezza dell'albero

Albero binario di ricerca: ricerca (cont.)

Nella maggior parte dei calcolatori la seguente versione iterativa della procedura è più efficiente

ITERATIVE-TREE-SEARCH(x, k)

```
1 while x ≠ NIL e k ≠ key[x]
2    do  if k < key[x]
3              then     x ← left[x]
4              else     x ← right[x]
5 return x
```

Albero binario di ricerca: minimo e massimo

TREE-MINIMUM(x)
1 **while** left[x] ≠ NIL
2 **do** x ← left[x]
3 **return** x

TREE-MAXIMUM(x)
1 **while** right[x] ≠ NIL
2 **do** x ← right[x]
3 **return** x

Entrambe le procedure seguono cammini discendenti nell'albero → T(n) = O(h)

Albero binario di ricerca: successore e predecessore

Il successore di un nodo x di un albero binario di ricerca è il nodo che segue secondo l'ordinamento determinato dalla visita simmetrica (se tutte le chiavi sono distinte è il nodo con la più piccola chiave > key[x])

TREE-SUCCESSOR(x)

```
1 if right[x] ≠ NIL
2    then    return TREE-MINIMUM(right[x])
3 ▷ se il sottoalbero dx di x è vuoto, il successore
   di x è il suo antenato più prossimo il cui figlio sx
   è anch'esso antenato di x
4 y ← p[x]
5 while y ≠ NIL e x = right[y]
6    do  x ← y
7        y ← p[y]
8 return y
```

T(n) = O(h) perché si segue un cammino che discende o uno che risale l'albero

TREE-PREDECESSOR(x) è simmetrica

Albero binario di ricerca: inserimento

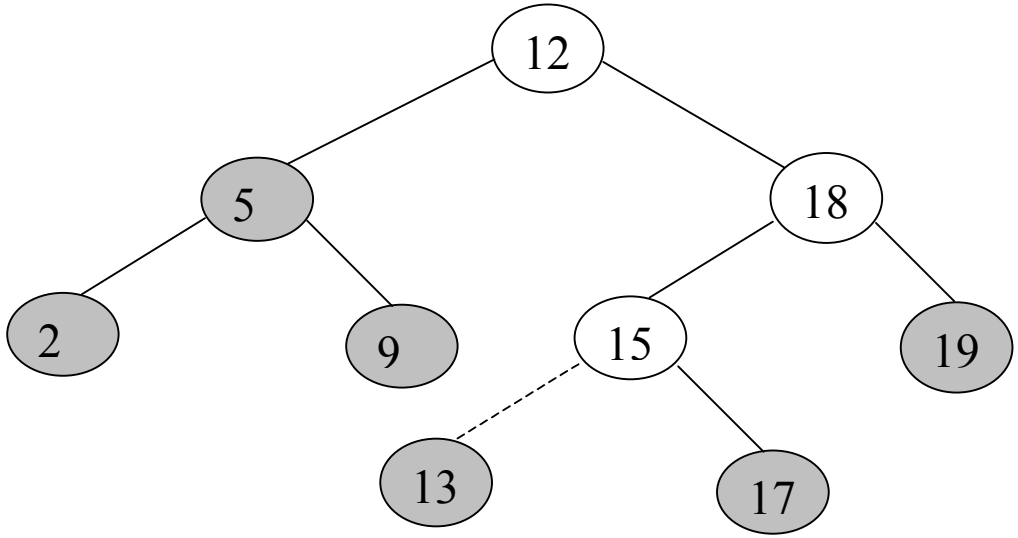
TREE-INSERT(T, z) T(n) = O(h)

```
0 ▷ z è tale che left[z] = right[z] = NIL
1 y ← NIL
2 x ← root[T]
3 while x ≠ NIL
4    do  y ← x
5        if key[z] < key[x]
6             then     x ← left[x]
7             else     x ← right[x]
8 p[z] ← y
9 if y = NIL
10  then     root[T] ← z
11   else if  key[z] < key[y]
12            then     left[y] ← z
13            else     right[y] ← z
```

Albero binario di ricerca: inserimento (cont.)

Esempio

Inserimento del nodo con chiave 13: i nodi chiari sono quelli percorsi da TREE-INSERT



Albero binario di ricerca: cancellazione

TREE-DELETE(T, z) $T(n) = O(h)$

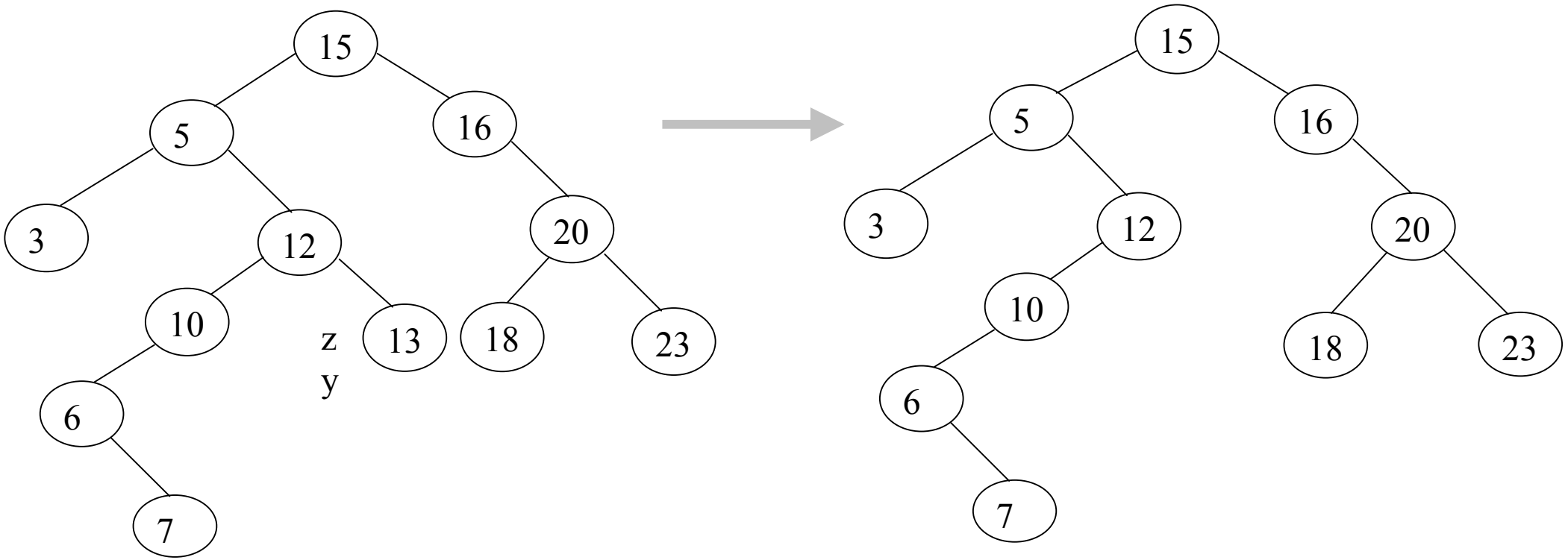
```
1   if left[z] = NIL o right[z] = NIL
2         then    y ← z
3         else    y ← TREE-SUCCESSOR(z)
4         ▷ y è il nodo da estrarre
5   if left[y] ≠ NIL
6         then    x ← left[y]
7         else    x ← right[y]
8         ▷x è il figlio non NIL di y o NIL se y non ha figli
9   if x ≠ NIL
10        then    p[x] ← p[y]
11  if p[y] = NIL
12        then    root[T] ← x
13        else if y = left[p[y]]
14              then    left[p[y]] ← x
15              else    right[p[y]] ← x
16  ▷ y è stato estratto
```

Albero binario di ricerca: cancellazione (cont.)

```
17  if y ≠ z
18        then      key[z] ← key[y]
19         ▷  se y ha altri campi, copia anche questi;
           se il nodo estratto è il successore di y,
           il contenuto di z viene sostituito con quello di y
20  return y
21  ▷ y viene restituito per essere riutilizzato nella lista libera
```

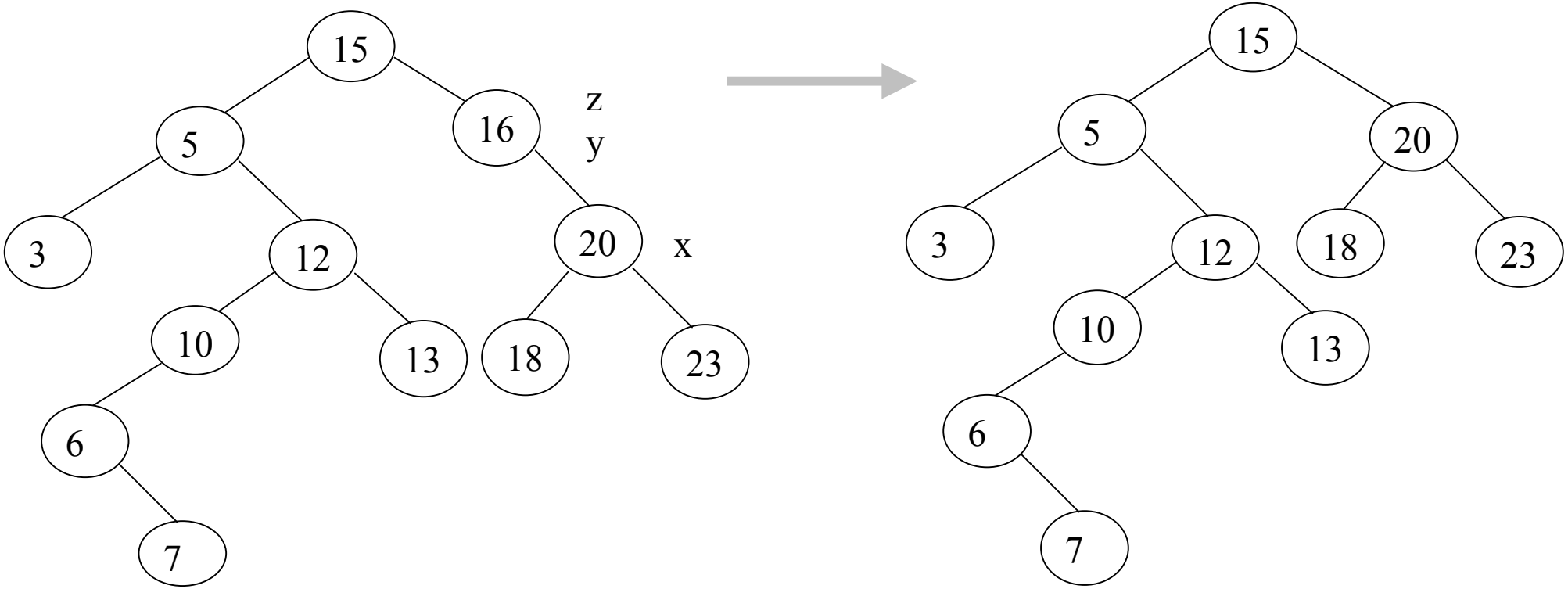
Albero binario di ricerca: cancellazione (cont.)

Esempio 1



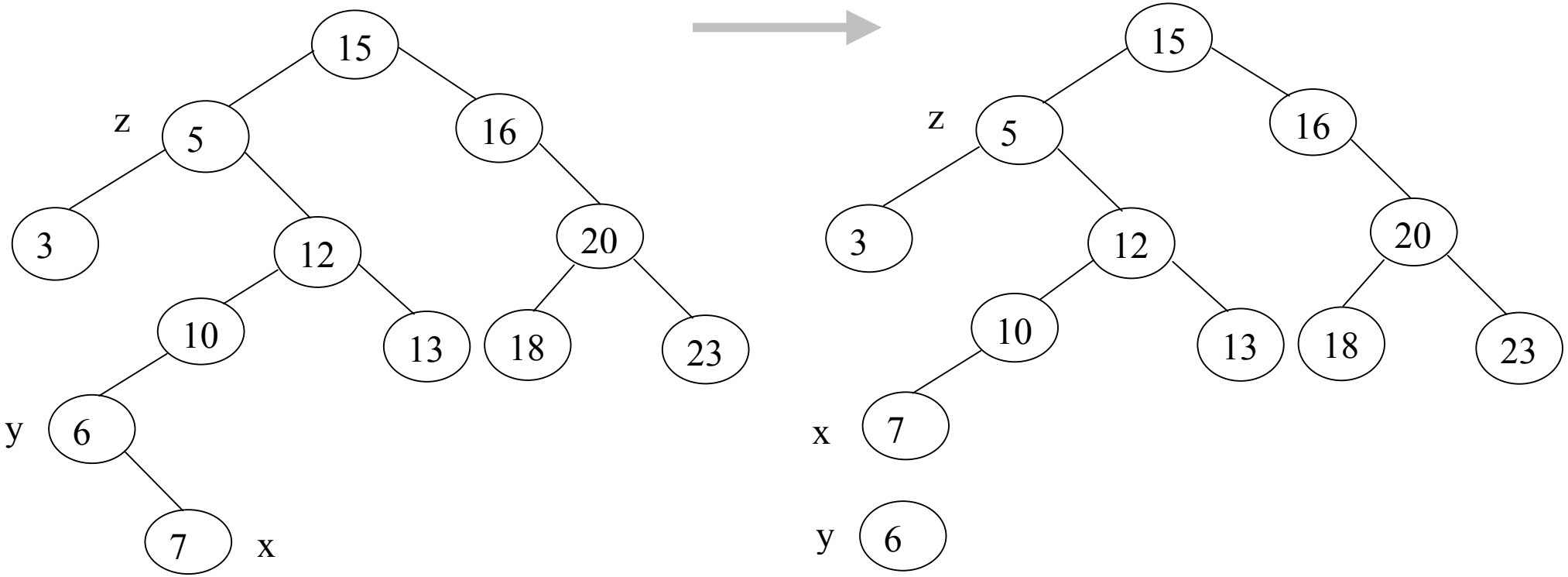
Albero binario di ricerca: cancellazione (cont.)

Esempio 2

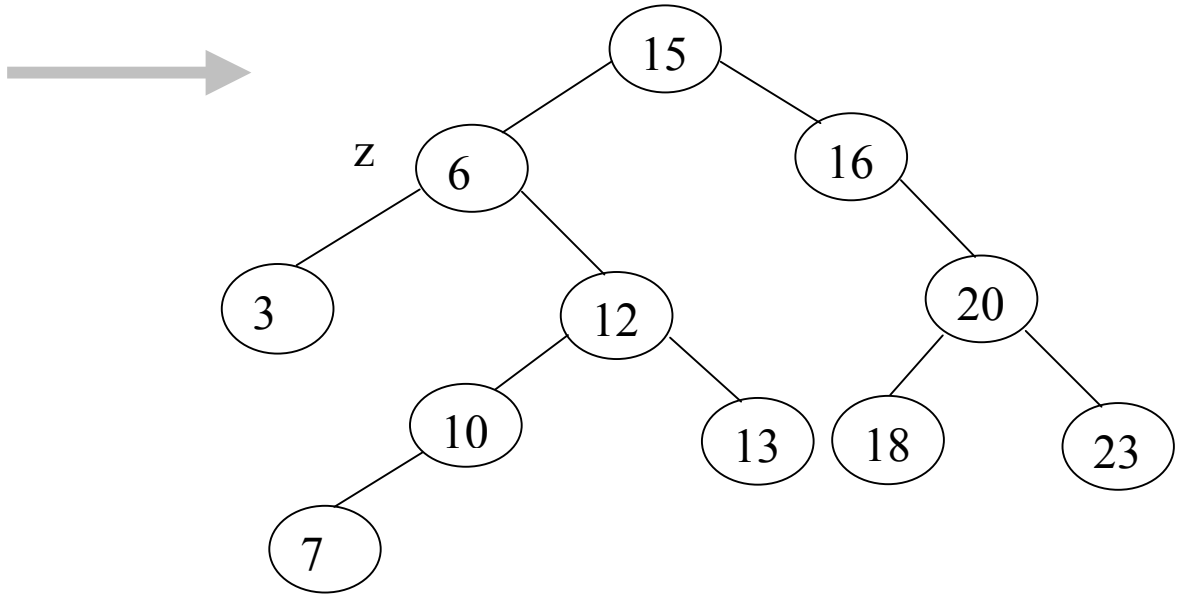


Albero binario di ricerca: cancellazione (cont.)

Esempio 3



Albero binario di ricerca: cancellazione (cont.)



Conclusioni

Le operazioni di base su un albero binario di ricerca richiedono un tempo proporzionale all'altezza dell'albero →

Albero binario completo: $T(n) = \Theta(lgn)$ nel caso peggiore

Albero lineare: $T(n) = \Theta(n)$ nel caso peggiore