



Alberi di ricerca binari



Fulvio Corno, Matteo Sonza Reorda
Dip. Automatica e Informatica
Politecnico di Torino



Introduzione

Gli alberi di ricerca binari (Binary Search Tree, o BST) sono una struttura di dati che supporta in modo efficiente le operazioni SEARCH, MINIMUM, MAXIMUM, PREDECESSOR, SUCCESSOR, INSERT, DELETE.

Sono utili come implementazione di dizionari o di code prioritarie.



Obiettivi

I BST sono definiti in modo tale che le operazioni abbiano una complessità proporzionale all'altezza h dell'albero.

Per un albero completo e bilanciato con n nodi, la complessità è quindi $\Theta(\log n)$ nel caso peggiore.

Per un albero totalmente sbilanciato, invece, si ricade nel caso peggiore $O(n)$.

Per un albero casuale ci si aspetta $\Theta(\log n)$.

A.A. 2001/2002

APA-bst

3



Definizione

Albero binario di ricerca:

- **Albero**: struttura gerarchica con una radice. Esiste un solo percorso dalla radice a ciascun nodo. Tale percorso definisce delle relazioni padre-figlio
- **Binario**: ogni nodo ha al più 2 figli (*left* e *right*) e (salvo la radice) esattamente un padre (p)
- **Ricerca**: i nodi hanno un campo chiave *key*, e sono **ordinati** in base ad esso

A.A. 2001/2002

APA-bst

4

Relazione di ordinamento (I)

Per ciascun nodo x vale che:

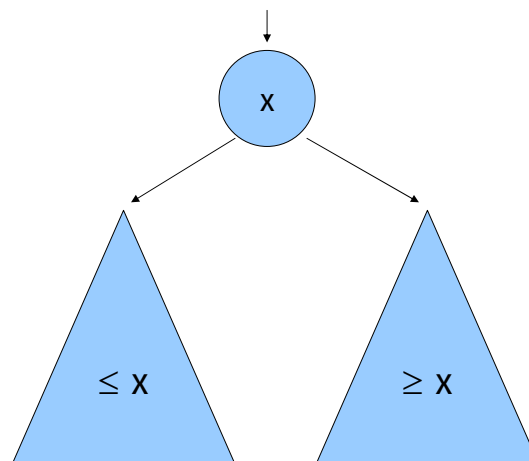
- Per tutti i nodi y nel sottoalbero sinistro di x , $\text{key}[y] \leq \text{key}[x]$
- Per tutti i nodi y nel sottoalbero destro di x , $\text{key}[y] \geq \text{key}[x]$

A.A. 2001/2002

APA-bst

5

Relazione di ordinamento (II)

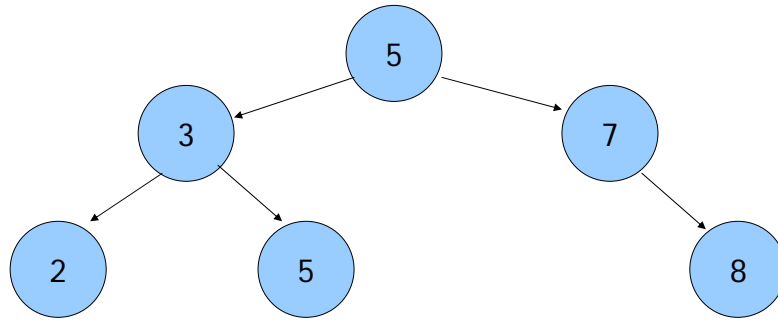


A.A. 2001/2002

APA-bst

6

Esempio

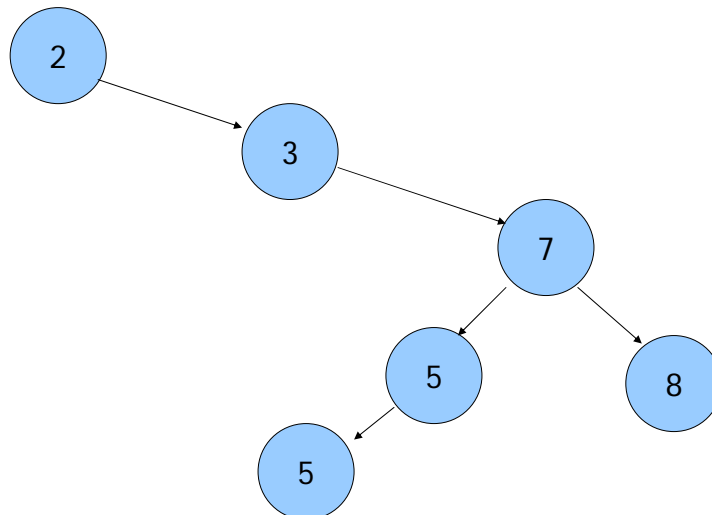


A.A. 2001/2002

APA-bst

7

Esempio (meno efficiente)



A.A. 2001/2002

APA-bst

8



Attraversamenti

Dato un BST, è possibile definire delle operazioni di attraversamento, ossia di visita di tutti i nodi, secondo 3 ordini diversi:

- **Preorder**: prima il nodo, poi i due sottoalberi
- **Inorder**: prima il sottoalbero sinistro, poi il nodo, poi il sottoalbero destro
- **Postorder**: prima i due sottoalberi, poi il nodo

A.A. 2001/2002

APA-bst

9



Attraversamento Preorder

Preorder-Tree-Walk(x)

- 1 **if** $x \neq \text{NIL}$
- 2 **then** print key[x]
- 3 Preorder-Tree-Walk(left[x])
- 4 Preorder-Tree-Walk(right[x])

A.A. 2001/2002

APA-bst

10



Attraversamento Inorder

Inorder-Tree-Walk(x)

```
1  if x ≠ NIL
2      then Inorder-Tree-Walk(left[x])
3      print key[x]
4      Inorder-Tree-Walk(right[x])
```

A.A. 2001/2002

APA-bst

11



Attraversamento Postorder

Postorder-Tree-Walk(x)

```
1  if x ≠ NIL
2      then Postorder-Tree-Walk(left[x])
3      Postorder-Tree-Walk(right[x])
4      print key[x]
```

A.A. 2001/2002

APA-bst

12

Osservazioni

Nel caso di un BST T , l'attraversamento **Inorder** (ottenuto chiamando `Inorder-Tree-Walk(root[T])`) stampa gli elementi nell'ordine crescente del campo **key**.

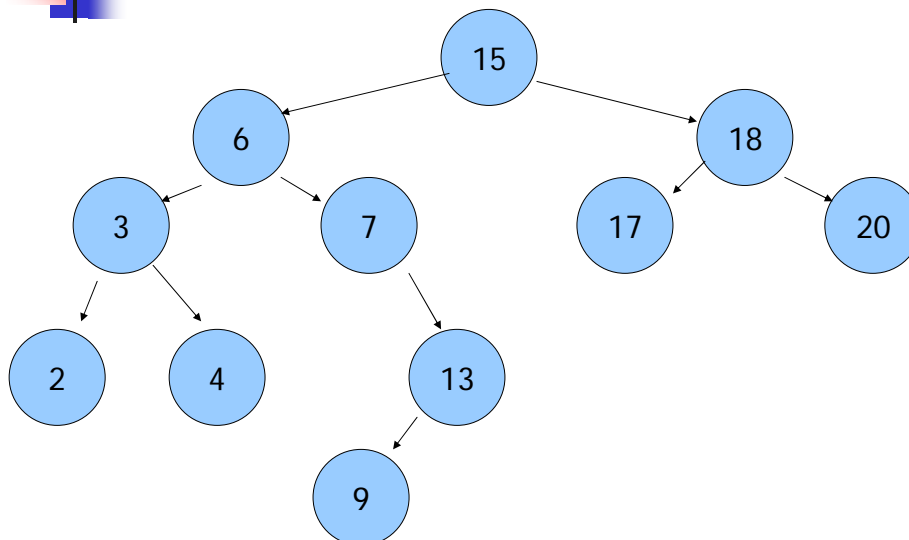
Tutti gli attraversamenti hanno complessità $\Theta(n)$, in quanto ciascun nodo viene considerato esattamente una volta.

A.A. 2001/2002

APA-bst

13

Esempio



A.A. 2001/2002

APA-bst

14

Esercizio

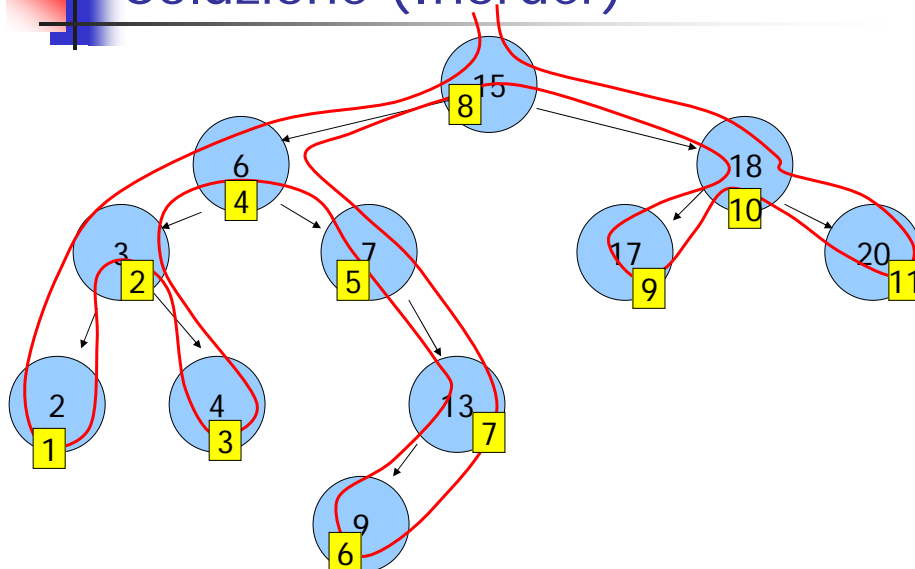
Si fornisca il risultato della visita in Preorder, Inorder, Postorder sul BST visto in precedenza.

A.A. 2001/2002

APA-bst

15

Soluzione (Inorder)

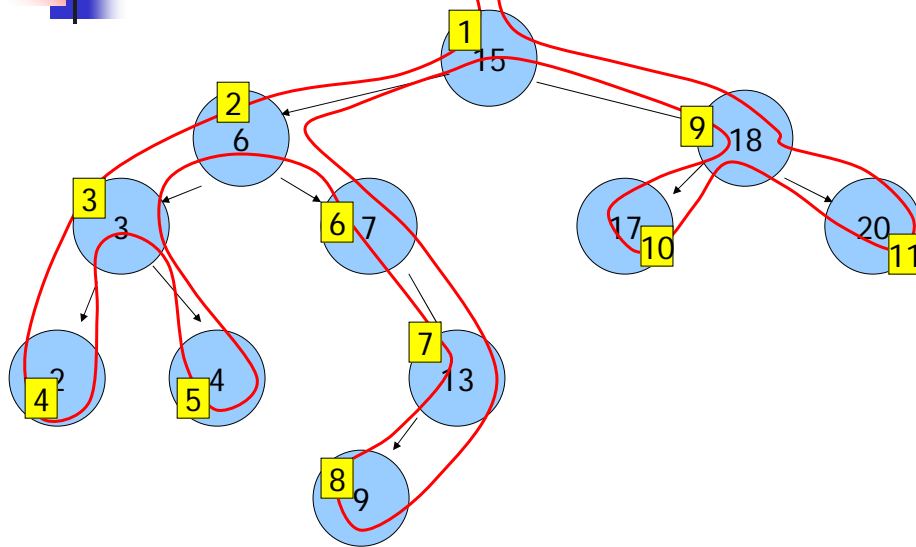


A.A. 2001/2002

APA-bst

16

Soluzione (Preorder)

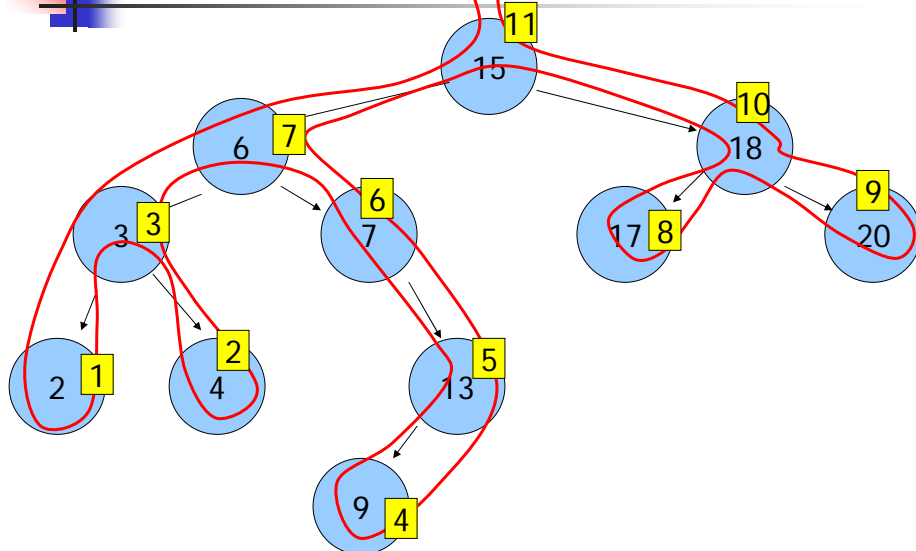


A.A. 2001/2002

APA-bst

17

Soluzione (Postorder)



A.A. 2001/2002

APA-bst

18



Esercizio proposto

Si definiscano le strutture dati per la rappresentazione di un BST.

Si definiscano i prototipi e si implementino le funzioni di visita dell'albero.

A.A. 2001/2002

APA-bst

19



Operazioni di ricerca nei BST

I BST sono particolarmente ottimizzati per le funzioni di ricerca: *Search*, *Minimum/Maximum*, *Predecessor/Successor*.

La loro complessità è $O(h)$, in funzione dell'altezza h dell'albero.

A.A. 2001/2002

APA-bst

20

Tree-Search

Tree-Search(x, k)

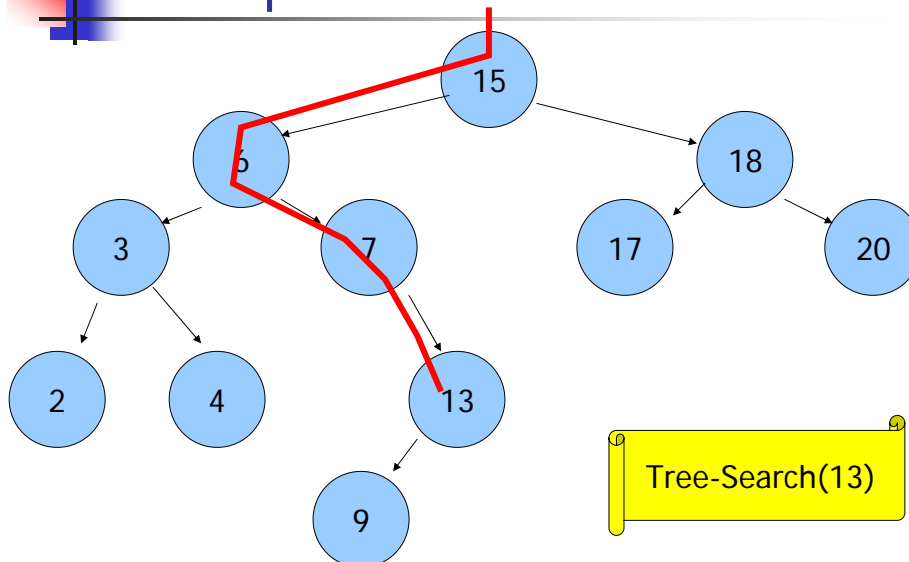
```
1  if  $x = \text{NIL}$  or  $k = \text{key}[x]$ 
2      then return  $x$ 
3  if  $k < \text{key}[x]$ 
4      then return Tree-Search(left[ $x$ ],  $k$ )
5      else return Tree-Search(right[ $x$ ],  $k$ )
```

A.A. 2001/2002

APA-bst

21

Esempio



A.A. 2001/2002

APA-bst

22

Tree-Search iterativa

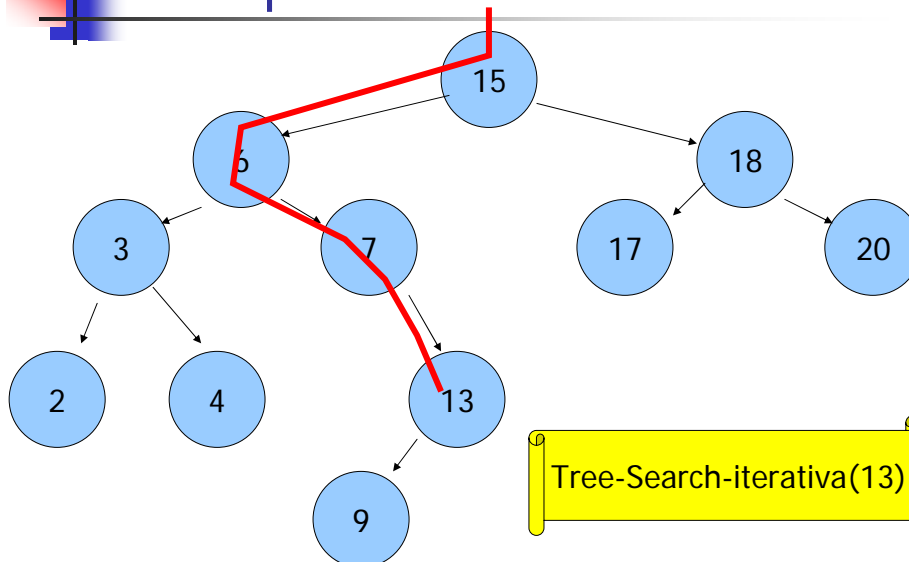
```
Tree-Search-iterativa(x, k)
1  while x ≠ NIL and k ≠ key[x]
2      do if k < key[x]
3          then x ← left[x]
4          else x ← right[x]
5  return x
```

A.A. 2001/2002

APA-bst

23

Esempio



A.A. 2001/2002

APA-bst

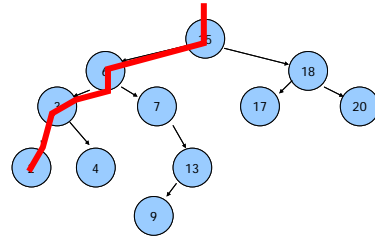
24

Minimo e massimo (versioni iterative)

Tree-Minimum(x)

```

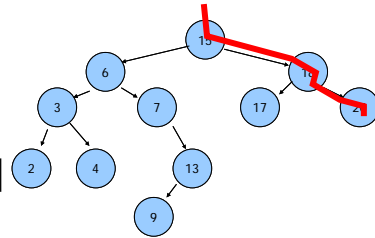
1  while left[x] ≠ NIL
2      do x ← left[x]
3  return x
    
```



Tree-Maximum(x)

```

1  while right[x] ≠ NIL
2      do x ← right[x]
3  return x
    
```



A.A. 2001/2002

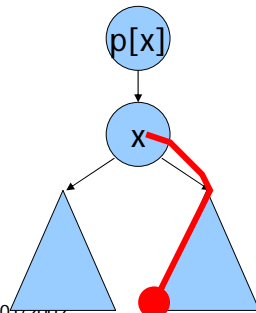
APA-bst

25

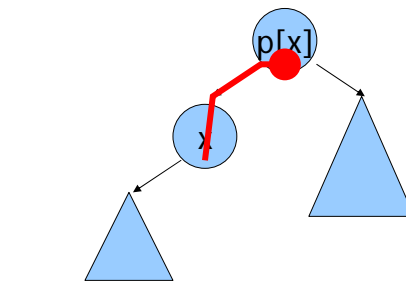
Successore

Dato un nodo, determinare il nodo immediatamente successivo. Vi sono 2 casi:

Il minimo del sottoalbero di destra



A.A. 2001/2002



Il primo padre di cui il nodo è nel sottoalbero sinistro

APA-bst

26

Successore

Tree-Successor(x)

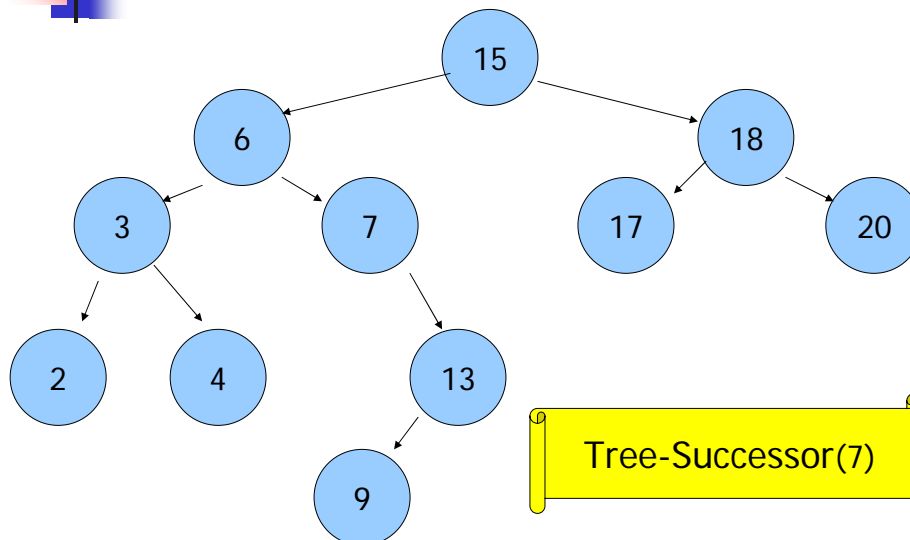
```
1  if right[x] ≠ NIL
2      then return Tree-Minimum(right[x])
3  y ← p[x]
4  while y ≠ NIL and x = right[y]
5      do x ← y
6      y ← p[y]
7  return y
```

A.A. 2001/2002

APA-bst

27

Esempio

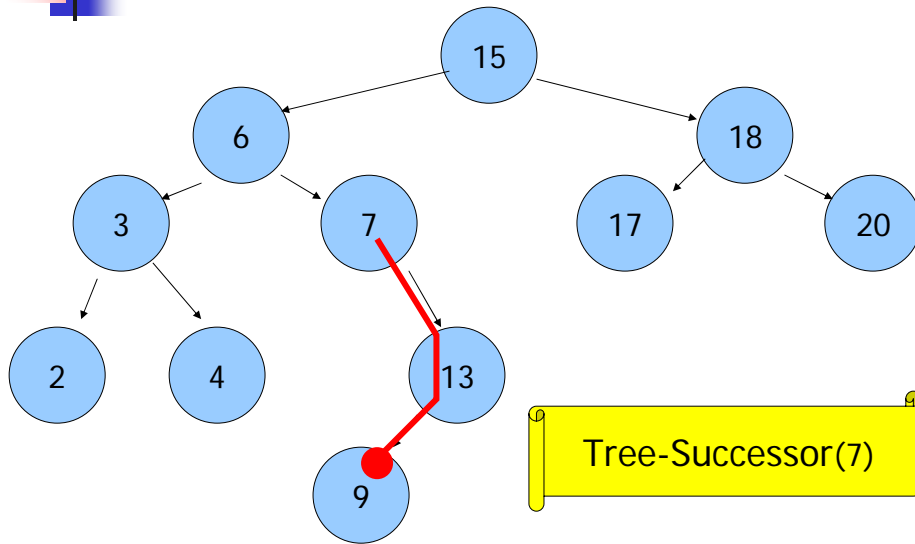


A.A. 2001/2002

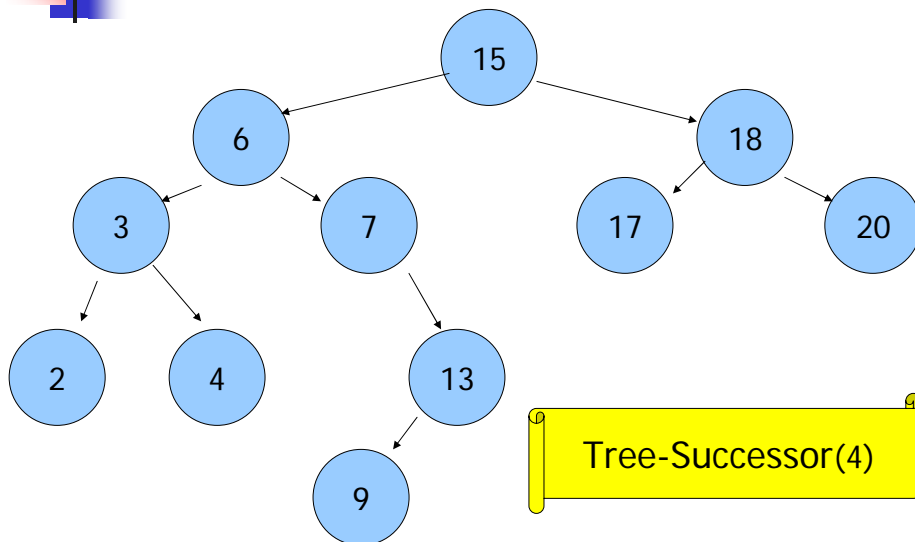
APA-bst

28

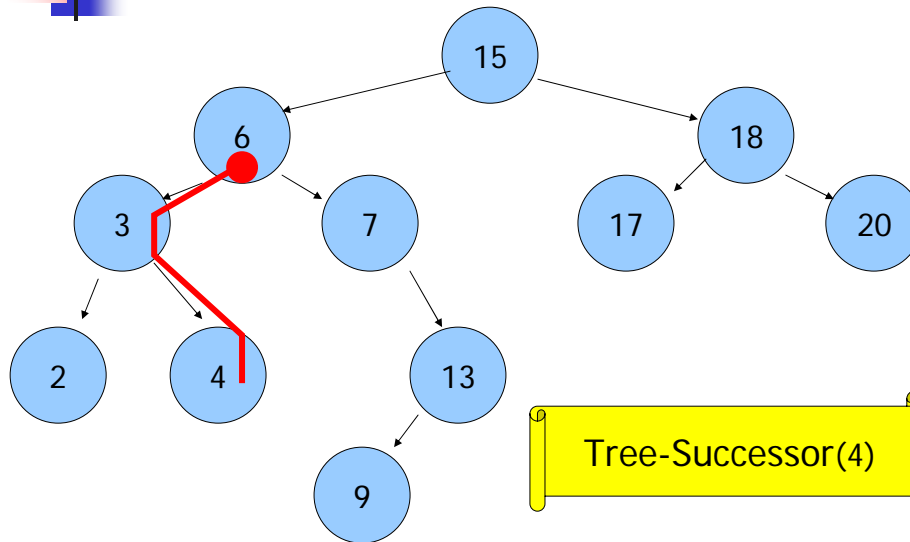
Esempio



Esempio



Esempio



Predecessore

Tree-Predecessor(x)

```
1  if left[x] ≠ NIL
2      then return Tree-Maximum(left[x])
3  y ← p[x]
4  while y ≠ NIL and x = left[y]
5      do x ← y
6      y ← p[y]
7  return y
```



Complessità

La complessità di tutte le procedure di ricerca è $O(h)$.

A.A. 2001/2002

APA-bst

33



Esercizio proposto

Si implementino in C le procedure di ricerca nel BST precedentemente definito.

A.A. 2001/2002

APA-bst

34



Inserimento e cancellazione

Queste operazioni richiedono di modificare la struttura dati, aggiungendo o togliendo nodi, **mantenendo la proprietà di ordinamento** propria del BST.

A.A. 2001/2002

APA-bst

35



Inserimento

Dato un BST, inserire un nodo z di chiave v :

- Si crea un nuovo nodo z , con $\text{left}[z]=\text{right}[z]=\text{NIL}$
- Si trova la posizione in cui inserirlo, simulando la ricerca di $\text{key}[z]$
- Si aggiornano i puntatori

Il nuovo nodo è **sempre** inserito come una **nuova foglia**.

A.A. 2001/2002

APA-bst

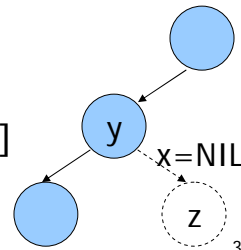
36

Tree-Insert (I)

Tree-Insert(T, z)

```
1   $y \leftarrow \text{NIL}$ 
2   $x \leftarrow \text{root}[T]$ 
3  while  $x \neq \text{NIL}$ 
4      do  $y \leftarrow x$ 
5      if  $\text{key}[z] < \text{key}[x]$ 
6          then  $x \leftarrow \text{left}[x]$ 
7          else  $x \leftarrow \text{right}[x]$ 
```

Ricerca key[z]
nell'albero



A.A. 2001/2002

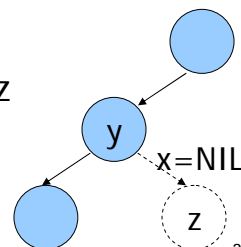
APA-bst

37

Tree-Insert (II)

```
8   $p[z] \leftarrow y$ 
9  if  $y = \text{NIL}$ 
10     then  $\text{root}[T] \leftarrow z$ 
11     else if  $\text{key}[z] < \text{key}[y]$ 
12         then  $\text{left}[y] \leftarrow z$ 
13         else  $\text{right}[y] \leftarrow z$ 
```

Inserisce z
come figlio di y

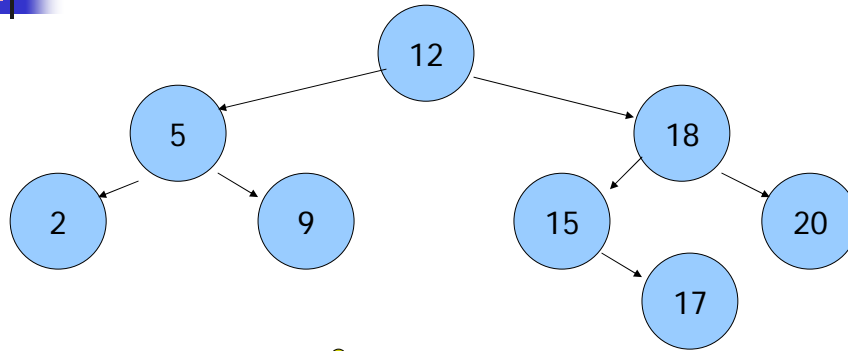


A.A. 2001/2002

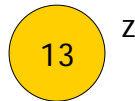
APA-bst

38

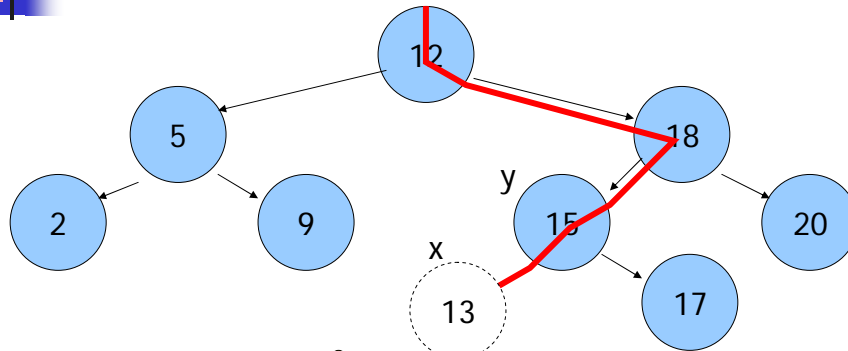
Esempio



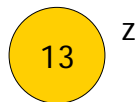
Tree-Insert(13)



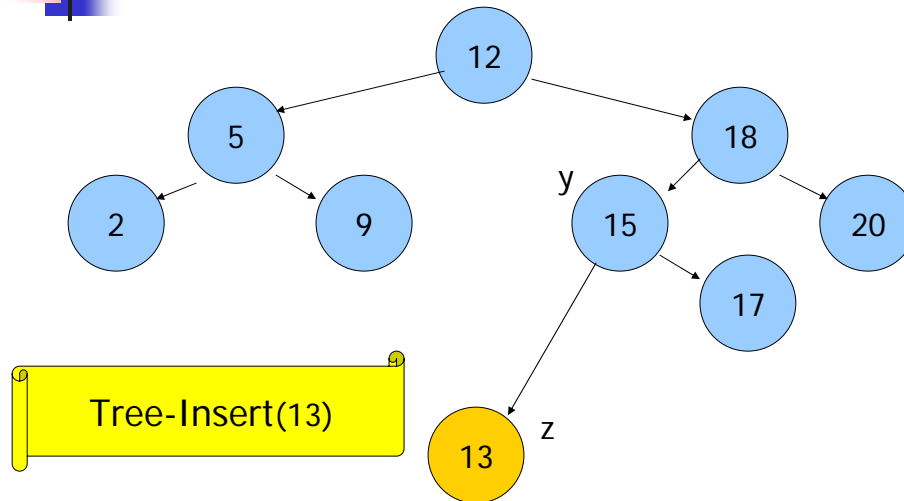
Esempio



Tree-Insert(13)



Esempio



A.A. 2001/2002

APA-bst

41

Esercizio proposto

Si implementi la funzione di inserimento.
Si testi il programma completo leggendo un insieme di elementi da file, e compiendo una serie di ricerche su di essi.

A.A. 2001/2002

APA-bst

42

Cancellazione

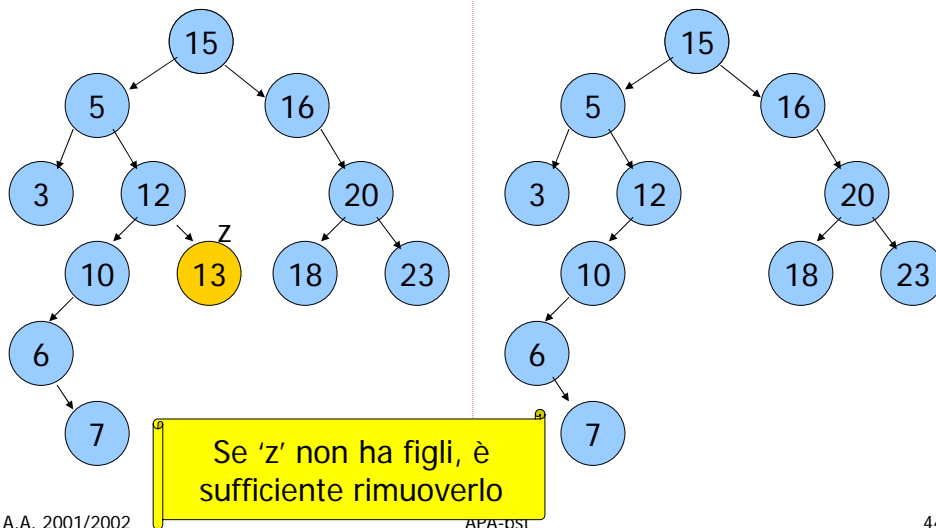
La cancellazione è l'operazione più complessa sui BST, in quanto il nodo da cancellare potrebbe avere 0, 1 o 2 figli, e occorre "ricollegare" i nodi rimasti in assenza di quello cancellato.

A.A. 2001/2002

APA-bst

43

Casi possibili: 0 figli

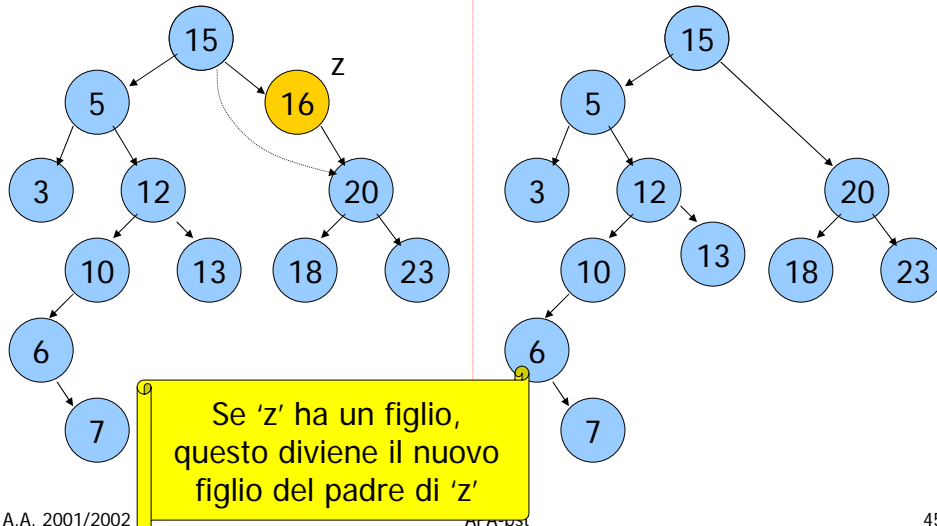


A.A. 2001/2002

APA-DST

44

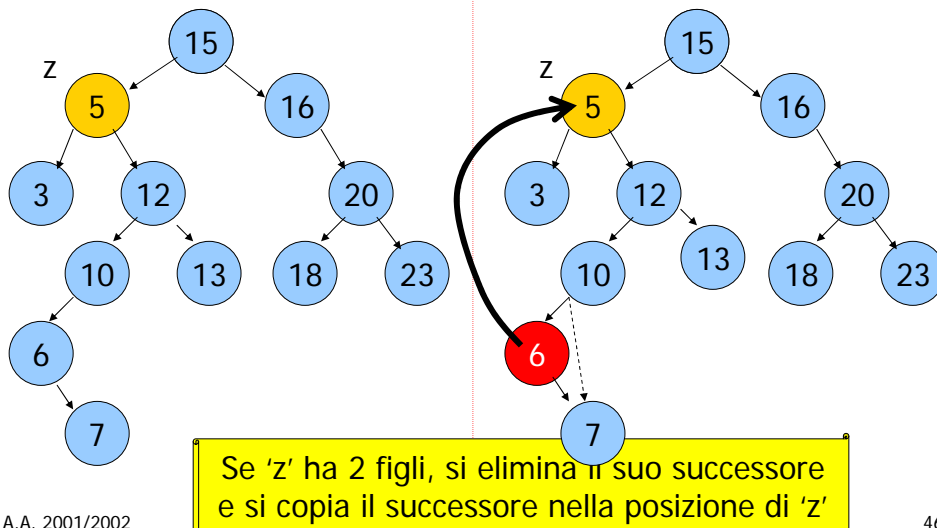
Casi possibili: 1 figlio



A.A. 2001/2002

45

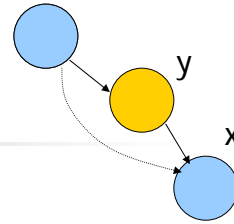
Casi possibili: 2 figli (I)



A.A. 2001/2002

46

Tree-Delete (I)



Tree-Delete(T, z)

1 **if** left[z]=NIL or right[z]=NIL

2 **then** $y \leftarrow z$

3 **else** $y \leftarrow$ Tree-Successor(z)

4 **if** left[y] \neq NIL

y: nodo da eliminare

5 **then** $x \leftarrow$ left[y]

6 **else** $x \leftarrow$ right[y]

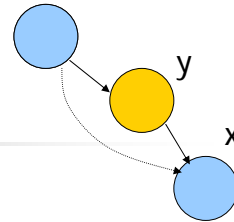
x: unico figlio di y

A.A. 2001/2002

APA-bst

47

Tree-Delete (II)



7 **if** $x \neq$ NIL

8 **then** $p[x] \leftarrow p[y]$

Aggiorna padre di x

9 **if** $p[y] =$ NIL

10 **then** root[T] = x

y è la radice? x
diviene radice

11 **else if** $y =$ left[$p[y]$]

12 **then** left[$p[y]$] \leftarrow x

13 **else** right[$p[y]$] \leftarrow x

Se no, collega x al
padre di y

A.A. 2001/2002

APA-bst

48

Tree-Delete (III)

```
14  if  $y \neq z$ 
15      then  $\text{key}[z] \leftarrow \text{key}[y]$ 
16           $\text{fields}[z] \leftarrow \text{fields}[y]$ 
17  return  $y$ 
```

Eventualmente, ricopia
le informazioni del
successore nel nodo
da eliminare

A.A. 2001/2002

APA-bst

49

Esercizio proposto (opzionale)

Si implementi la funzione di cancellazione di un nodo da un BST.

A.A. 2001/2002

APA-bst

50



Complessità

La complessità di tutte le procedure di modifica dell'albero (inserimento e cancellazione) è $O(h)$.

A.A. 2001/2002

APA-bst

51



Bilanciamento

Le operazioni hanno complessità $O(h)$:

- Un albero completamente bilanciato ha
 - $h = \log_2 n$
- Un albero completamente sbilanciato ha
 - $h = n$
- Le operazioni sui BST hanno complessità variabile tra $O(\log_2 n)$ e $O(n)$

A.A. 2001/2002

APA-bst

52

Esercizio

Si vuole costruire un BST contenente gli elementi interi da 0 a 9.

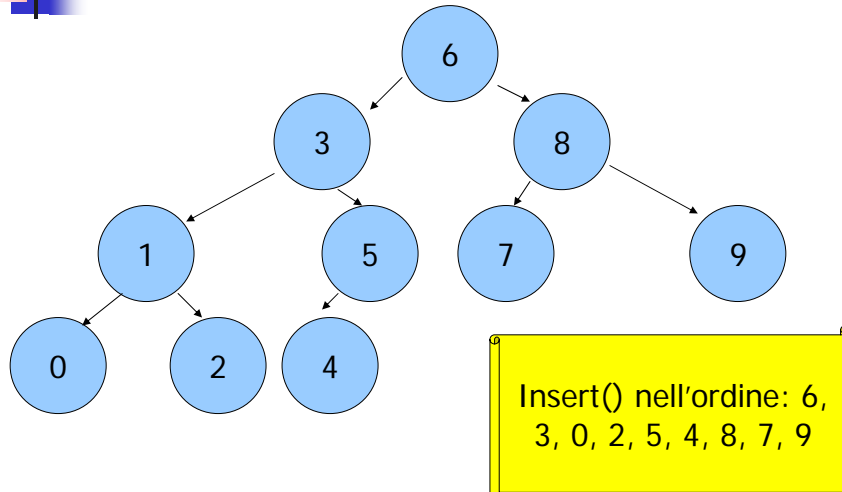
- Si fornisca almeno una sequenza di chiamate alla funzione Insert che crei un BST bilanciato
- Si fornisca almeno una sequenza di chiamate alla funzione Insert che crei un BST sbilanciato

A.A. 2001/2002

APA-bst

53

Soluzione (I)

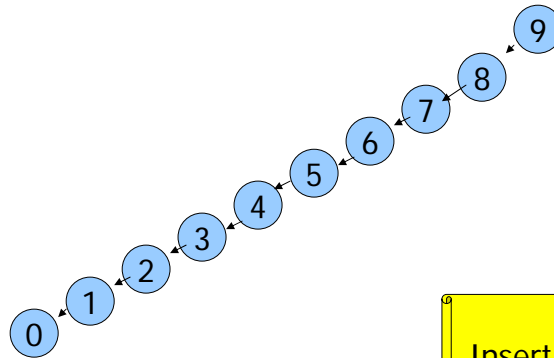


A.A. 2001/2002

APA-bst

54

Soluzione (II)



Insert() nell'ordine: 9,
8, 7, 6, 5, 4, 3, 2, 1, 0

A.A. 2001/2002

APA-bst

55

Esercizio proposto

Si sperimenti la costruzione di BST a partire da file diversamente ordinati, in modo da generare alberi bilanciati e sbilanciati, e si confrontino i tempi di esecuzione delle procedure di ricerca nei due casi.

A.A. 2001/2002

APA-bst

56



Alberi bilanciati

Le procedure Insert e Delete sono in grado di mantenere la proprietà di ordinamento dei BST, ma non garantiscono affatto il bilanciamento.

Esistono versioni più sofisticate degli alberi binari nelle quali viene garantito anche il bilanciamento.

Esempio: alberi red-black (per i quali si dimostra che: $h \leq 2 \log_2(n+1)$).

A.A. 2001/2002

APA-bst

57



Alberi red-black

Soddisfano le seguenti proprietà:

- Ogni nodo è *red* oppure *black*
- Le foglie (NIL) sono tutte *black*
- Se un nodo è *red*, i suoi figli sono *black* (ma non viceversa)
- Ogni cammino tra un nodo ed una foglia ad esso discendente contiene lo stesso numero di nodi *black*

A.A. 2001/2002

APA-bst

58