Università di Napoli Federico II – Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Corso di Algoritmi e Strutture Dati

Alberi binari di ricerca

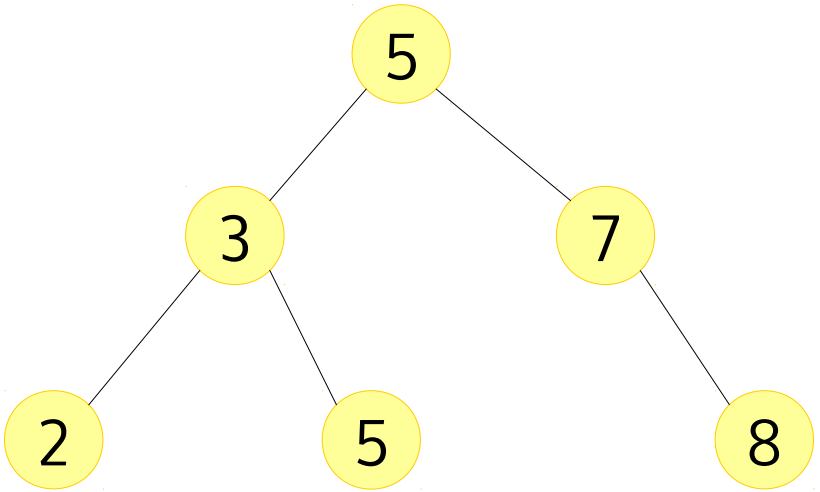
Alberi binari di ricerca

- Gli alberi di ricerca sono strutture dati che supportano molte delle operazioni definite su insiemi dinamici
  - Search, Minimum, Maximum, Predecessor, Successor, Insert, Delete
  - Possono essere usati sia come dizionario che coda a priorità
- Le operazioni base richiedono un tempo proporzionale all'altezza
  - lg n se l'albero è completo
  - n nel caso degenere di lista concatenata
- L'altezza attesa di un albero costruito in maniera aleatoria è lg n
- ...ma non sempre è possibile costruire l'albero in questa maniera
- Alcuni tipi particolari di alberi di ricerca presentano nel caso peggiore un tempo di esecuzione “buono”

Alberi binari di ricerca

- Un albero binario di ricerca soddisfa la seguente proprietà

  Sia x un nodo dell'albero. Se y è nel sottoalbero di sinistra di x, allora key[y]≤key[x]. Se y è nel sottoalbero di destra di x, allora key[y]≥key[x]



Alberi binari di ricerca

- La proprietà di un albero di ricerca consente di stampare tutte le chiavi in ordine crescente mediante una *visita in ordine* dell'albero

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

- Dimostriamo che impiega un tempo $\Theta(n)$

Visita in ordine di un albero

- Per n=0, richiede un tempo costante c: T(0)=c
- Per n>0, T(n)=T(k)+T(n-k-1)+d
- Proviamo per sostituzione T(n)=(c+d)n+c (e quindi T(n)=Θ(n))
  - Per n=0, T(0)=c
  - Passo induttivo:

    T(n) = T(k)+T(n-k-1)+d

    = (c+d)k + c + (c+d)(n-k-1) + c + d

    = (c+d)n + c – (c+d) + c + d

    = (c+d)n + c

Ricerca di un elemento

- Dato il puntatore alla radice e una chiave, restituisce il puntatore all'elemento con la data chiave, se esiste, e NIL altrimenti

```
Tree-Search (x,k)
if x=NIL or key[x]=k
    then return x
if k < key[x]
    then return Tree-Search (left[x],k)
    else return Tree-Search (right[x],k)
```

- I nodi incontrati nella ricerca formano un percorso verso il basso dalla radice
  - Il tempo di esecuzione è O(h) dove h è l'altezza dell'albero

Ricerca di un elemento

- Versione iterativa (tipicamente più efficiente)

```
Iterative-Tree-Search (x,k)
while x≠NIL and key[x]≠k
    do if k < key[x]
        then x ← left[x]
        else x ← right[x]
return x
```

Minimo e massimo

- L'elemento minimo può essere trovato percorrendo, a partire dalla radice, sempre il sottoalbero di sinistra

```
Tree-Minimum (x)
while left[x]≠NIL
    do x ← left[x]
return x
```

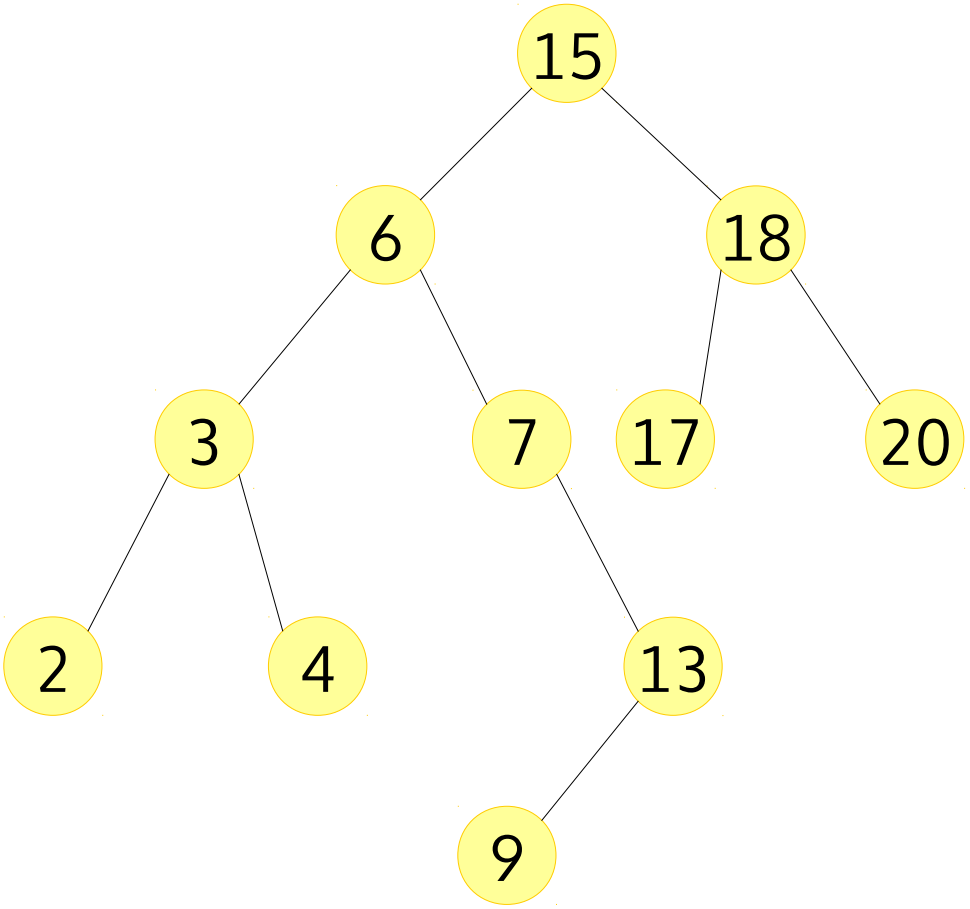
- L'elemento massimo può essere trovato percorrendo, a partire dalla radice, sempre il sottoalbero di destra

```
Tree-Maximum (x)
while right[x]≠NIL
    do x ← right[x]
return x
```

- Il tempo di esecuzione è O(h)

Successore di un nodo

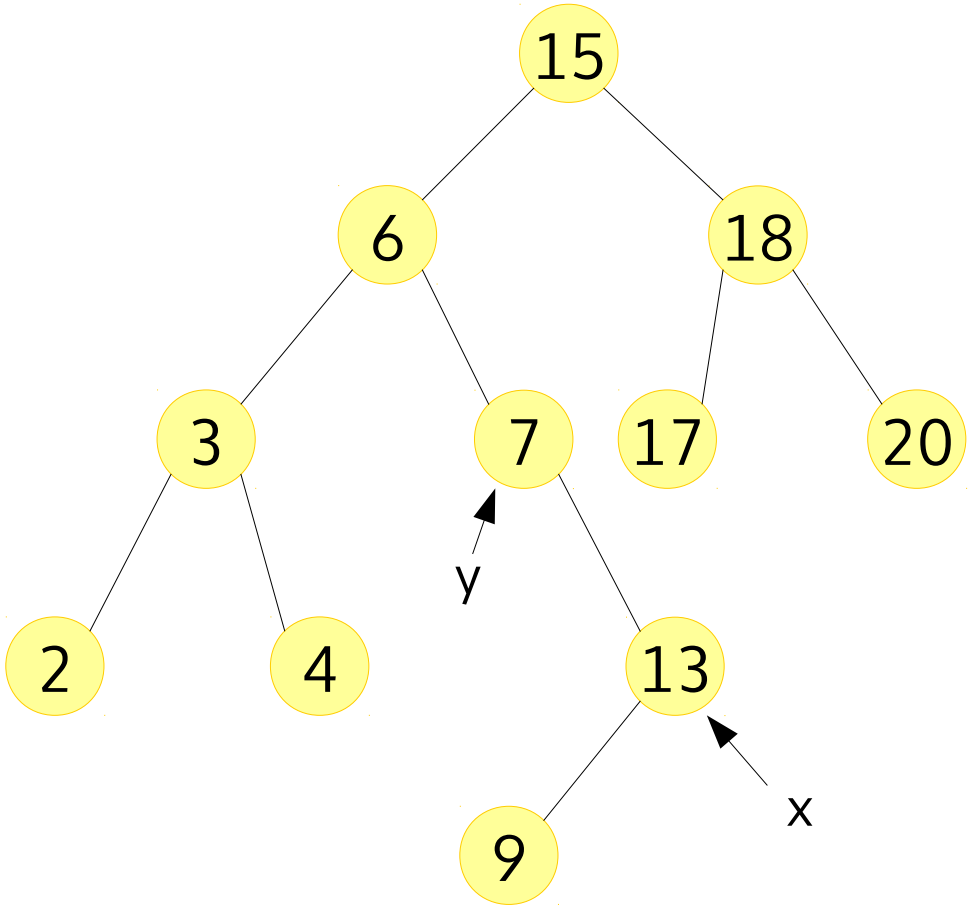
- Dato un nodo, il suo successore è quello che lo segue nella visita in ordine dell'albero



- Se il nodo ha un figlio di destra, il successore è il minimo del sottoalbero di destra
  - Successore di 6 è 7
- Altrimenti, è il più "basso" antenato di x il cui figlio di sinistra è un antenato di x
  - Successore di 13 è 15

Successore di un nodo

- Dato un nodo, il suo successore è quello che lo segue nella visita in ordine dell'albero

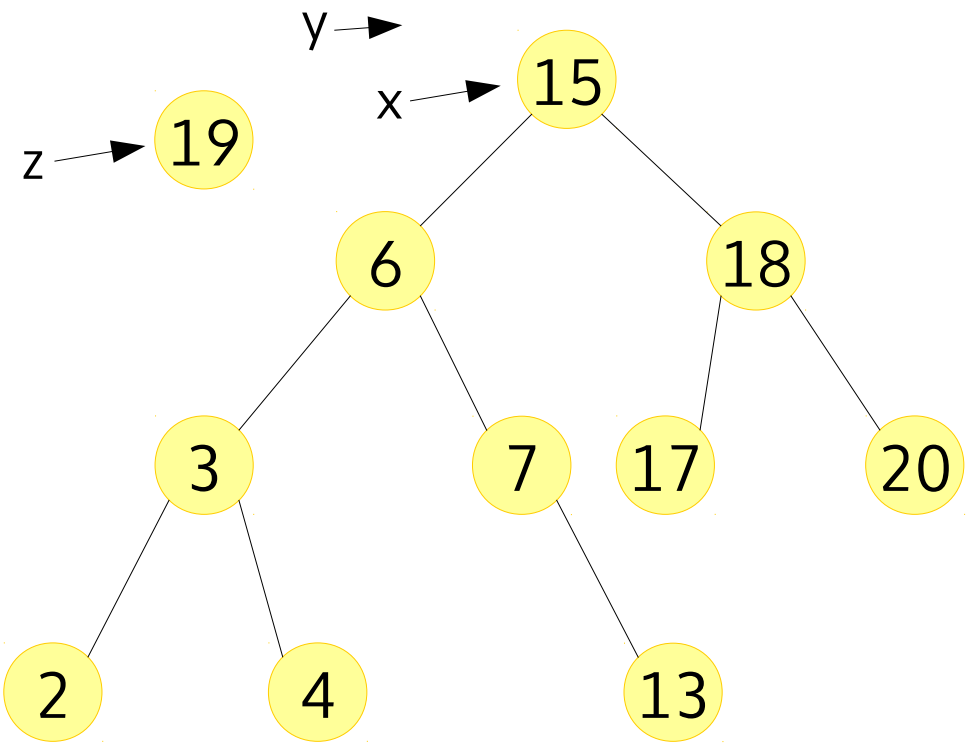



```
Tree-Successor (x)
if right[x]≠NIL
    then return Tree-Minimum(right[x])
y ← p[x]
while y≠NIL and x=right[y]
    do x ← y
    y ← p[y]
return y
```

- Il tempo di esecuzione è O(h)

Inserimento di un elemento

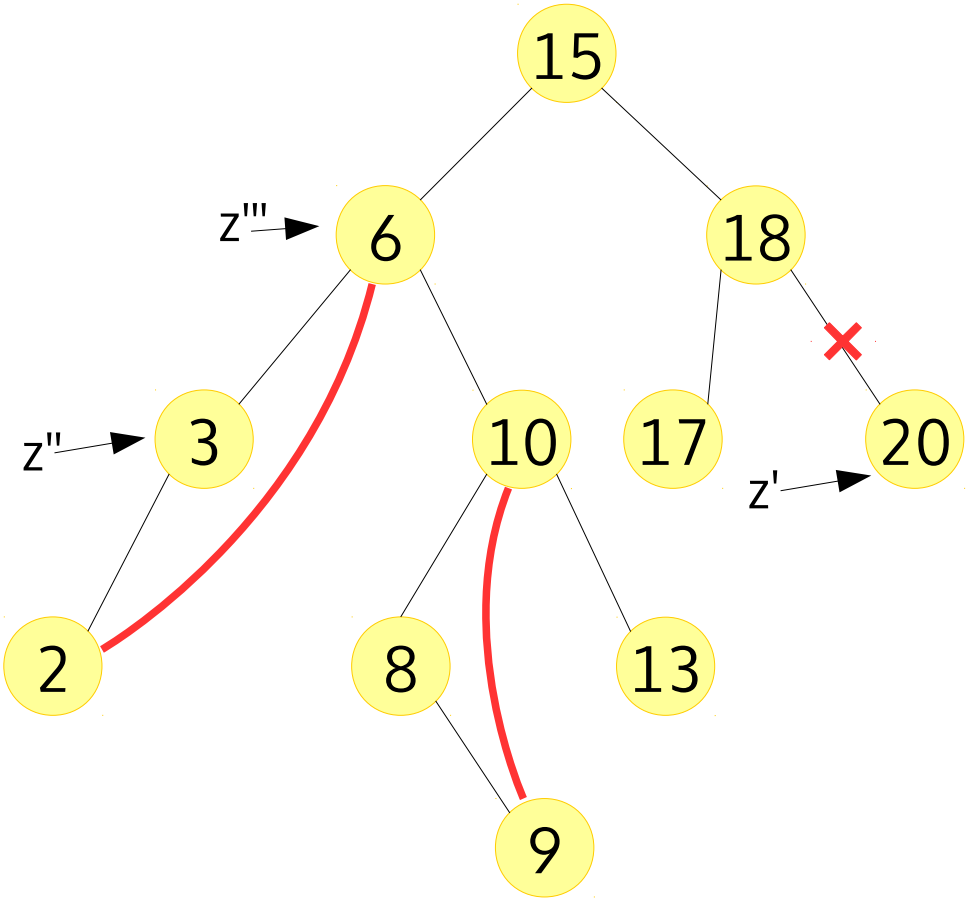
- Rispettando la proprietà di un albero di ricerca, l'elemento da inserire attraversa l'albero verso il basso a partire dalla radice




```
Tree-Insert (T,z)
y ← NIL
x ← root[T]
while x≠NIL
    do y ← x
    if key[z] < key[x]
        then x ← left[x]
        else x ← right[x]
p[z] ← y
if y = NIL
    then root[T] ← z  // albero vuoto
    else if key[z] < key[y]
        then left[y] ← z
        else right[y] ← z
```

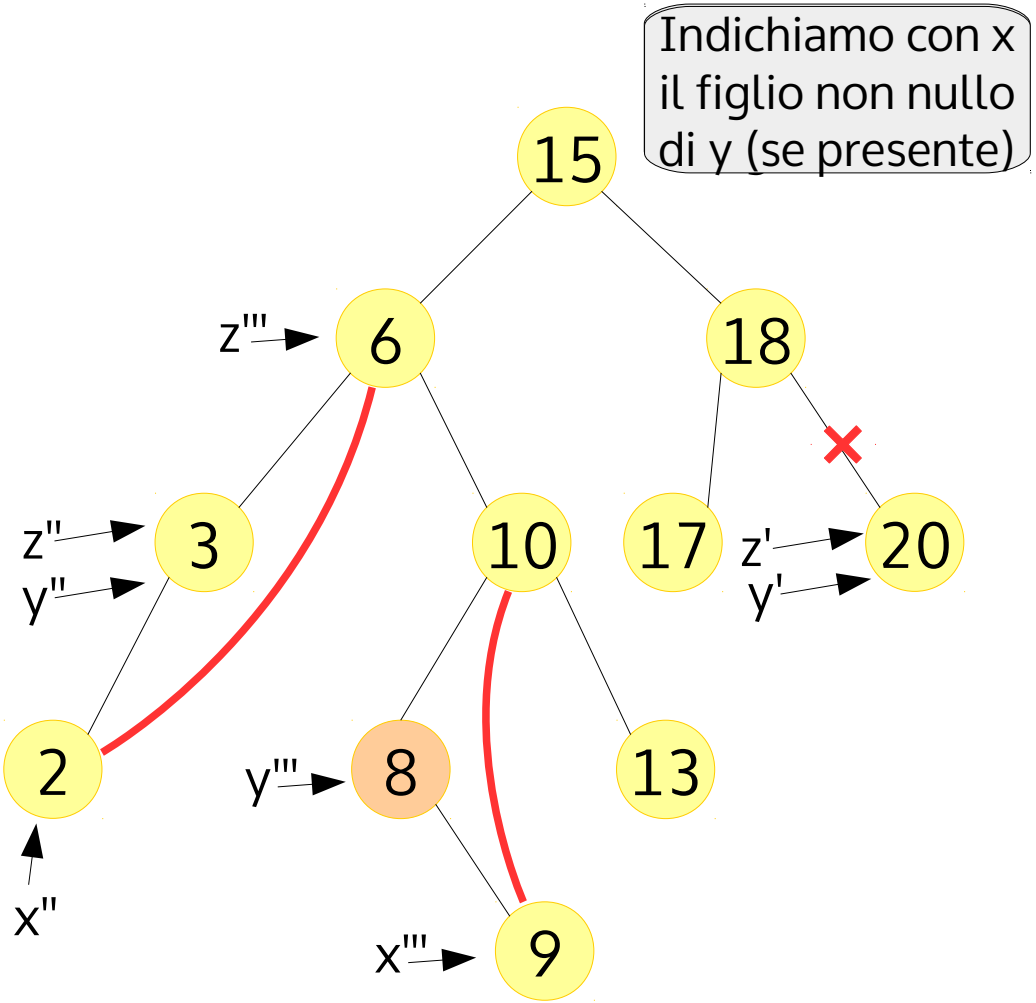
- Il tempo di esecuzione è O(h)

Eliminazione di un elemento



- Consideriamo 3 diversi casi
  - Il nodo non ha figli
  - Il nodo ha un solo figlio
  - Il nodo ha due figli
    - Cerchiamo il successore y, che non ha il figlio di sinistra
    - Tagliamo fuori y
    - Scambiamo chiave e dati satellite di z con quelli di y
- In tutti i casi c'è un nodo da tagliare fuori
  - z oppure il successore di z
  - ha al più un solo figlio

Eliminazione di un elemento



- Il tempo di esecuzione è O(h)

```
Tree-Delete (T,z)
if left[z]=NIL or right[z]=NIL
    then y ← z
    else y ← Tree-Successor (z)
if left[y] ≠ NIL
    then x ← left[y]
    else x ← right[y]
if x ≠ NIL          // y non e' foglia
    then p[x] ← p[y]
if p[y] = NIL    // y è radice
    then root[T] ← x
    else if y = left[p[y]]
        then left[p[y]] ← x
        else right[p[y]] ← x
if y ≠ z  // terzo caso
    then key[z] ← key[y]
    Copy y's satellite data into z
return y
```