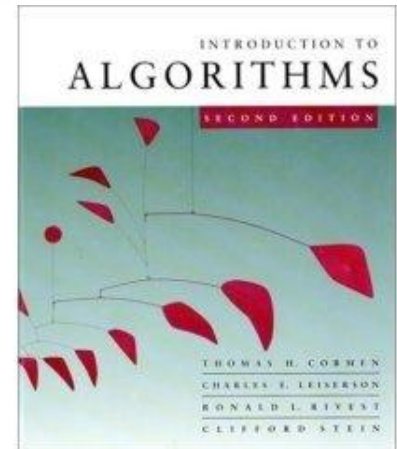




Alberi Binari di Ricerca

Algoritmi, Strutture Dati e Calcolo Parallelo

- ❑ Questo materiale è tratto dalle trasparenze del corso Introduction to Algorithms (2005-fall-6046) tenuto dal Prof. Leiserson all'MIT (<http://people.csail.mit.edu/cel/>)
- ❑ E dalle trasparenze del corso "Algoritmi e Strutture Dati" del prof. Alberto Montresor dell'Università di Trento
- ❑ T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein Introduction to Algorithms, Second Edition, The MIT Press, Cambridge, Massachusetts London, England McGraw-Hill Book Company
- ❑ Queste trasparenze sono disponibili sui siti <http://webpace.elet.polimi.it/lanzi>
<http://www.slideshare.net/pierluca.lanzi>



Nella lezione precedente...

- ❑ Insiemi dinamici che implementano le funzionalità
 - ▶ Item search(Key key)
 - ▶ void insert(Key key, Item item)
 - ▶ void delete(Key key)

- ❑ Implementazione
 - ▶ Array ordinato: ricerca $O(\log n)$, inserimento/cancellazione $O(n)$
 - ▶ Lista non ordinata: Ricerca/cancellazione $O(n)$, inserimento $O(1)$

- ❑ Tabelle Hash
 - ▶ Funzione hash mappa una chiave in uno slot della tabella
 - ▶ Concatenamento, indirizzamento aperto
 - ▶ Accesso $\Theta(1)$ nel caso medio

Alberi binari di ricerca

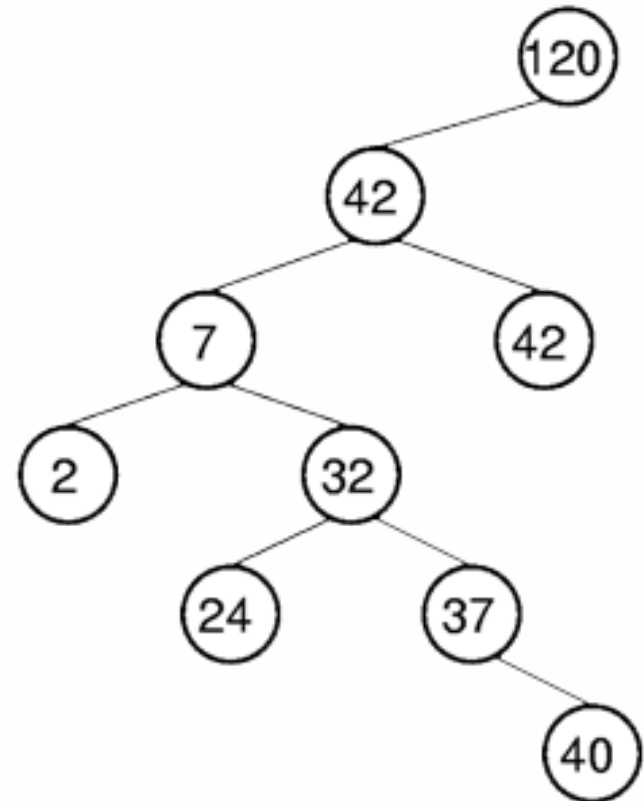
Alberi binari di ricerca

Strutture dati che supportano operazioni di ricerca, minimo, massimo, prev, next, inserimento, e cancellazione

Applicazioni: dizionari, code di priorità

L'idea è portare la ricerca binaria in un albero binario

- ❑ Definizione: albero binario in cui ogni nodo x contiene un insieme di dati $data[x]$ associati ad una chiave $key[x]$ da un dominio totalmente ordinato
- ❑ Le chiavi dei nodi del sottoalbero sinistro $left[x]$ sono $\leq key[x]$
- ❑ Le chiavi dei nodi del sottoalbero destro $right[x]$ sono $\geq key[x]$

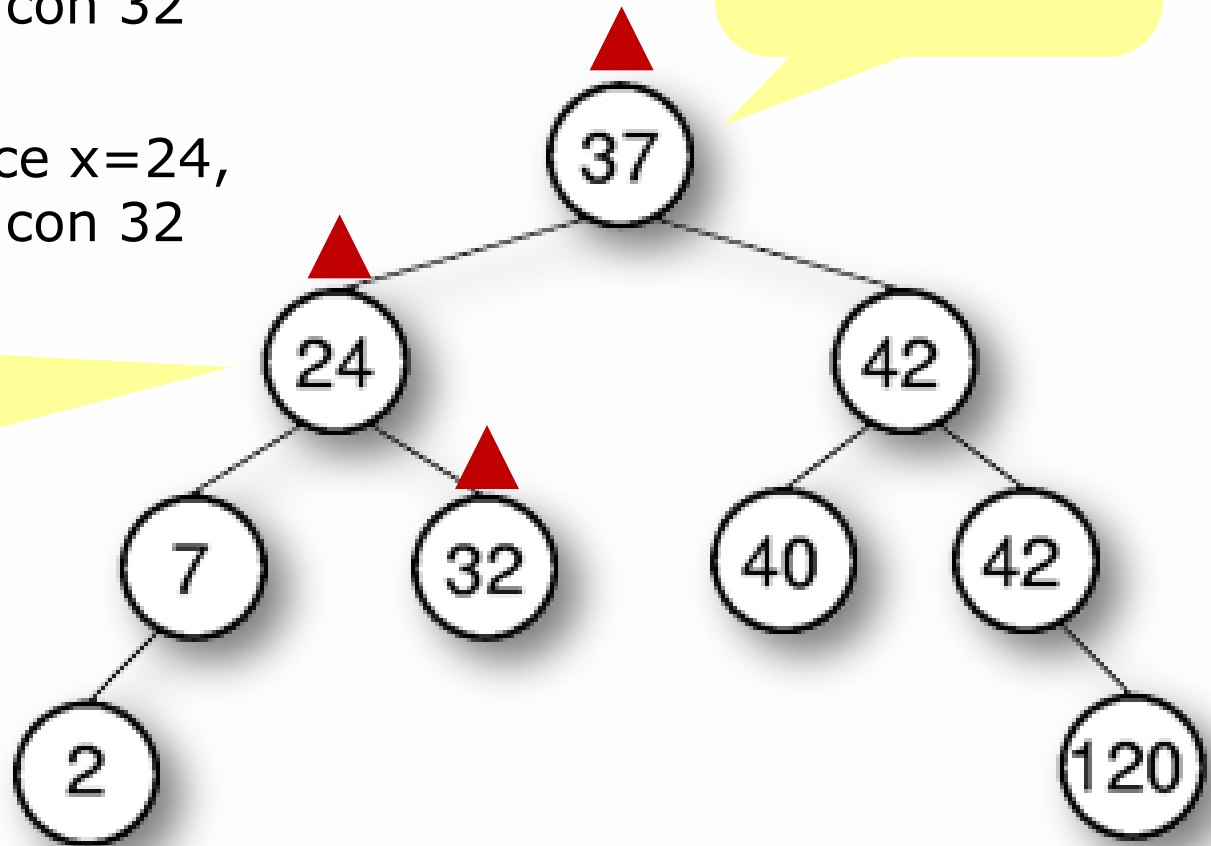


Alberi Binari di Ricerca: Ricerca di un Elemento

- ❑ Ricerca dell'elemento 32
- ❑ Considera la radice $x=37$, confronta $key[x]$ con 32
- ❑ Considera la radice $x=24$, confronta $key[x]$ con 32

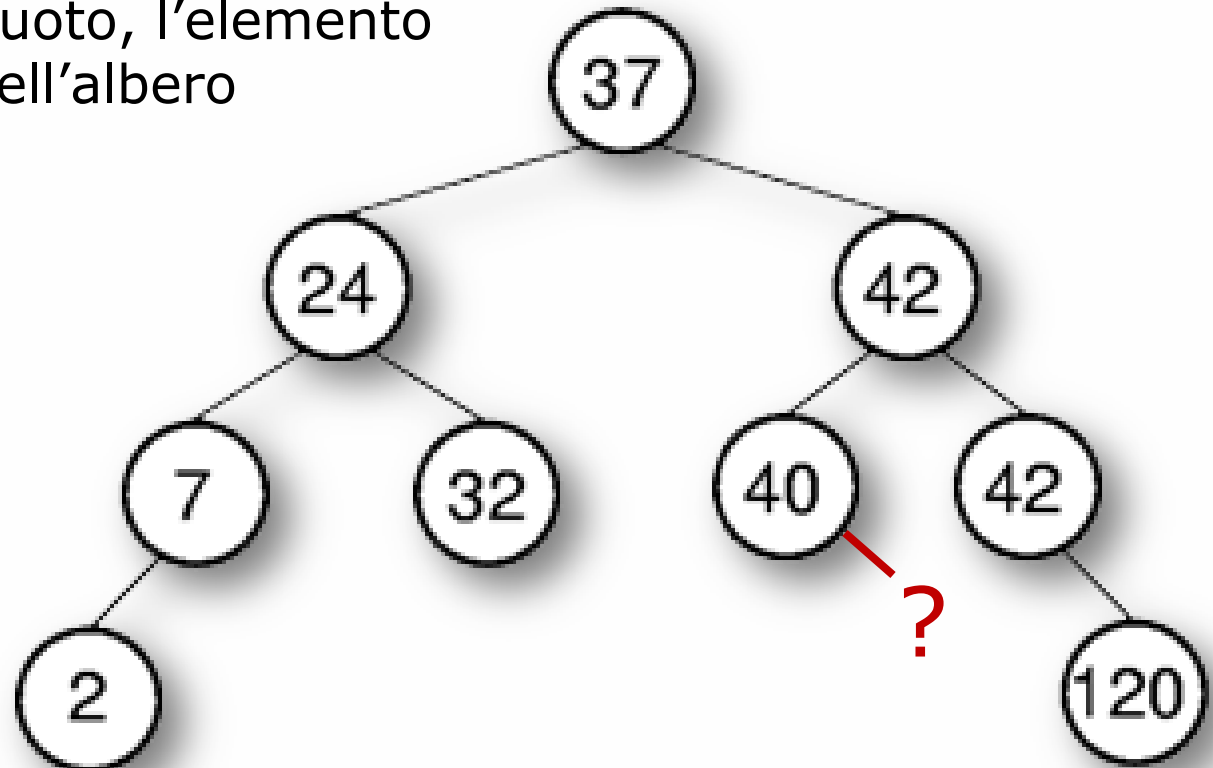
$32 \leq 37$
Passa al sotto
albero sinistro

$24 \leq 32$
Passa al sotto
albero destro



Alberi Binari di Ricerca: Ricerca di un Elemento

- ❑ Ricerca dell'elemento 41
- ❑ La ricerca si ferma nel sottoalbero destro di 40
- ❑ Il sottoalbero è vuoto, l'elemento non è presente nell'albero



```
Template <class Elem>
class ABR {
private:
    ABR parent;
    ABR left, right;
    Key key;
public:
    ABR(Key key, Data data) {
        key = key;
        this.data = data;
    }
    // key(), data(), left(), right(), parent()
    // definiti come getters dei rispettivi campi
    // ritornano i valori associati al nodo corrente
}
```

Pseudo-codice

```
TREE-SEARCH (x, k)
1  if x= NIL or k = key[x]
2      then return x
3  if k < key[x]
4  then return
5      TREE-SEARCH(left[x], k)
6  else return
7      TREE-SEARCH(right[x], k)
```

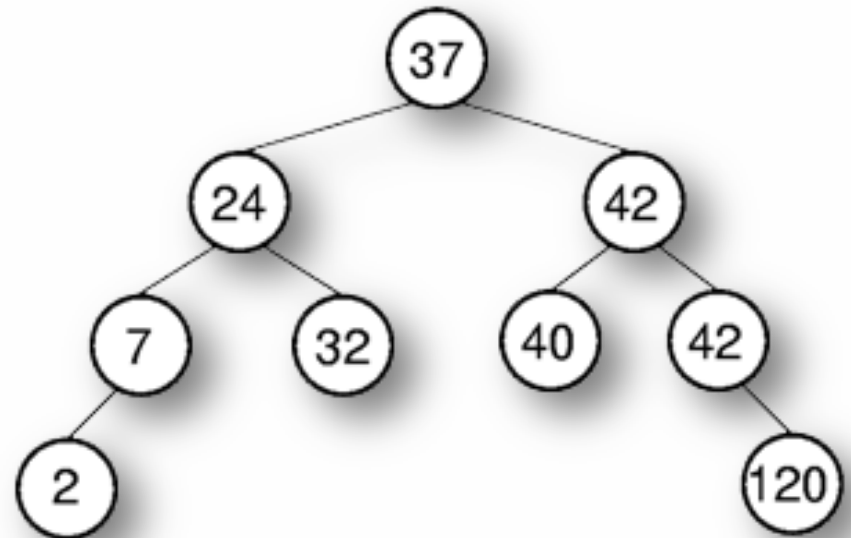
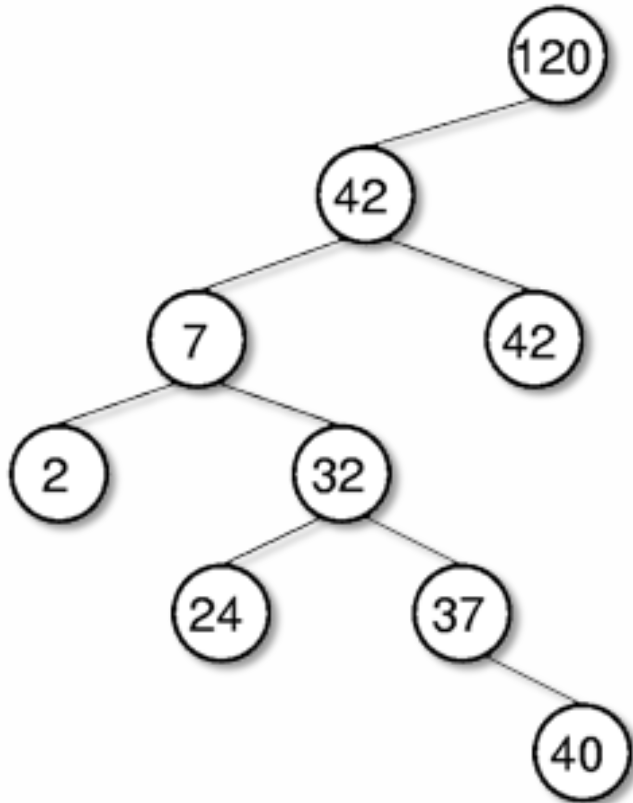
C++

```
ABR search(ABR T, Key k)
{
    if T = nil or k = T.key
        then return T
    else if k < T.key then
        return
        search(T.left, k)
    else return
        search(T.right, k)
}
```

La ricerca può essere riscritta in forma iterativa

```
ITERATIVE-TREE-SEARCH(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
```

Minimo, Massimo, Elemento Successivo e Precedente



Minimo & Massimo

Minimo

TREE-MINIMUM (x)

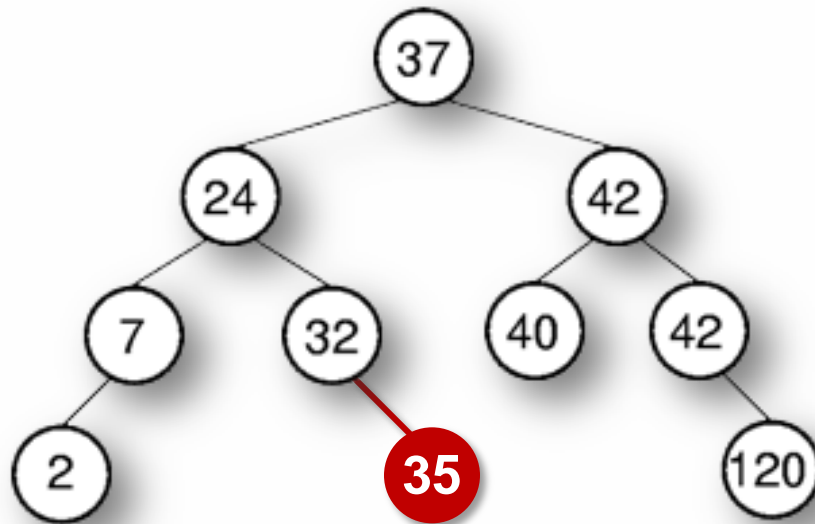
```
1 while left[x] ≠ NIL
2     do x ← left[x]
3 return x
```

Massimo

TREE-MAXIMUM(x)

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

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



- ❑ Supponiamo di dover inserire l'elemento 35
- ❑ Dove dovrebbe essere inserito affinché l'albero rimanga un albero binario di ricerca ?
- ❑ Devo cercare la chiave!

Inserimento di una nuova chiave
Ricerca Posizione + Inserimento della foglia

```
TREE-INSERT(T, z)
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 // Tree T was empty
11     else if key[z] < key[y]
12         then left[y] ← z
13         else right[y] ← z
```

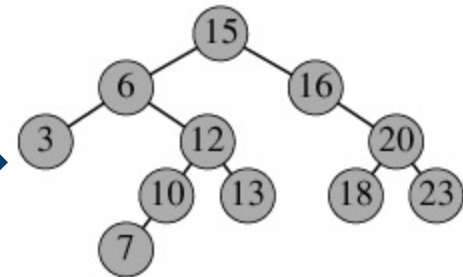
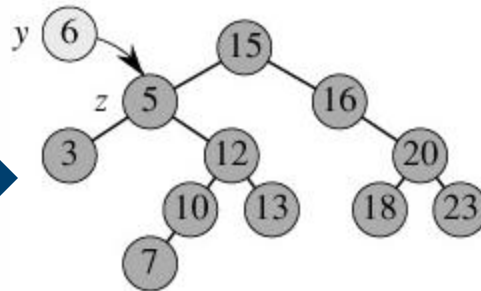
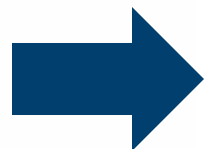
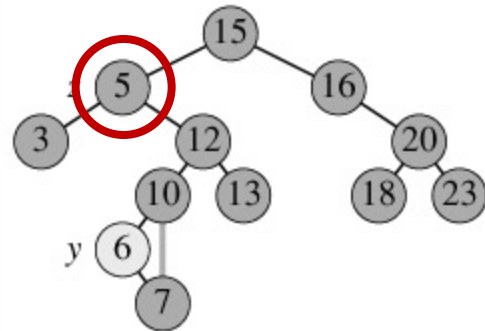
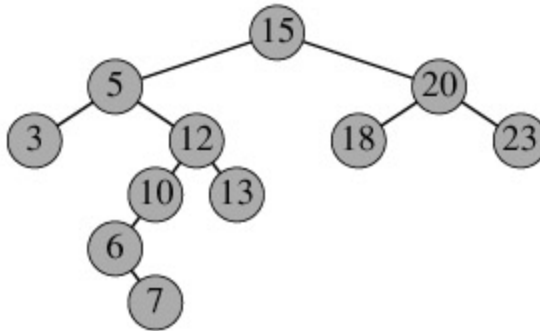
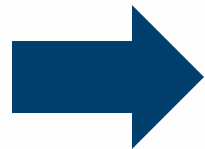
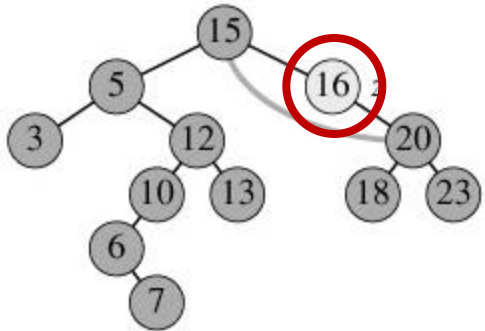
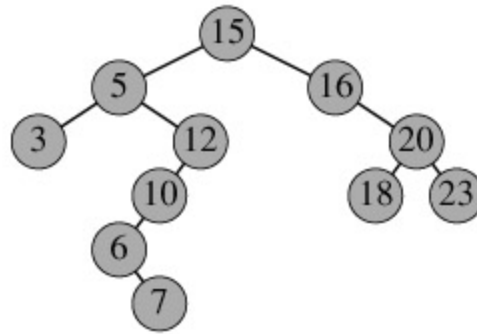
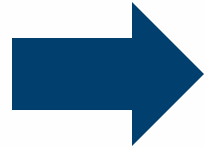
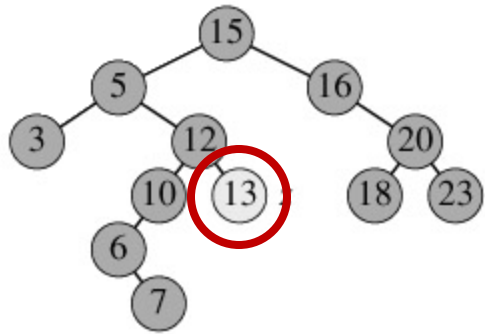
T albero di ricerca binaria

z nodo da inserire

key[z] = v

left[z] = NIL

right[z] = NIL



```
TREE-DELETE(T, z)
1  if left[z] = NIL or right[z] = NIL
2      then y ← z
3      else y ← TREE-SUCCESSOR(z)
4  if left[y] ≠ NIL
5      then x ← left[y]
6      else x ← right[y]
7  if x ≠ NIL
8      then p[x] ← p[y]
9  if p[y] = NIL
10     then root[T] ← x
11     else if y = left[p[y]]
12             then left[p[y]] ← x
13             else right[p[y]] ← x
14  if y ≠ z
15     then key[z] ← key[y]
16         copy y's satellite data into z
17  return y
```

Qual è la complessità delle operazioni di ricerca, cancellazione, inserimento?

Le operazioni di ricerca, inserimento, cancellazione sono $O(h)$ per un albero binario di ricerca di altezza h

Qual è il caso peggiore? Il caso migliore?

L'altezza attesa di un albero binario di ricerca con n nodi è $O(\lg n)$

Sommario

- ❑ Alberi Binari di Ricerca
 - ▶ Struttura dati ispirata alla ricerca binaria
 - ▶ Le chiavi dei nodi del sottoalbero sinistro $\text{left}[x]$ sono $\leq \text{key}[x]$
 - ▶ Le chiavi dei nodi del sottoalbero destro $\text{right}[x]$ sono $\geq \text{key}[x]$

- ❑ Operazioni: ricerca, minimo, massimo, prev, next, inserimento, e cancellazione

- ❑ Complessità
 - ▶ Ricerca, Inserimento, Cancellazione sono $O(h)$ per un albero di altezza h
 - ▶ Un albero binario di ricerca con n nodi generato in maniera casuale ha un'altezza $O(\lg n)$