# ALBERI BINARI DI RICERCA

Un *albero binario di ricerca* (binary search tree) è un tipo particolare di albero binario, in cui esiste una ***relazione d'ordine* fra gli elementi** dell'albero stesso.

Si tratta dunque di un albero *ordinato*.

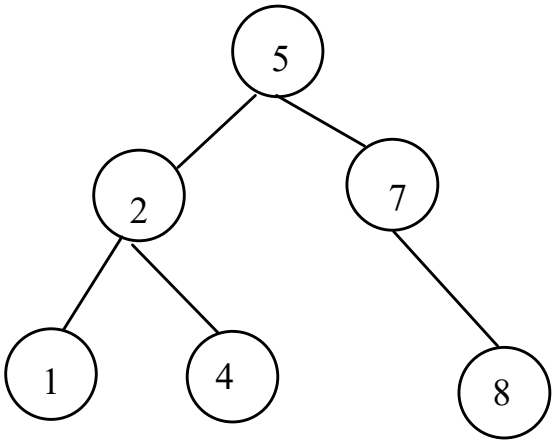
Gli alberi binari di ricerca sono usati per memorizzare grandi quantità di dati *su cui si esegue spesso un'operazione di ricerca*.

In un albero binario di ricerca, ogni nodo N ha la seguente proprietà:

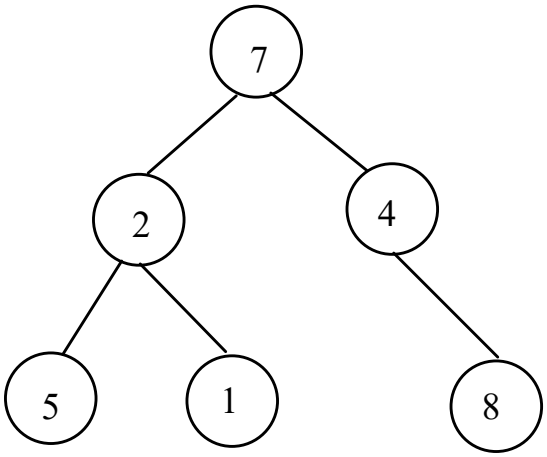
- tutti i nodi del sottoalbero sinistro di N hanno un valore *minore o uguale* a quello di N
- tutti i nodi del sottoalbero destro di N hanno un valore *maggiore* di quello di N.

ESEMPIO

Albero binario di ricerca




Albero binario



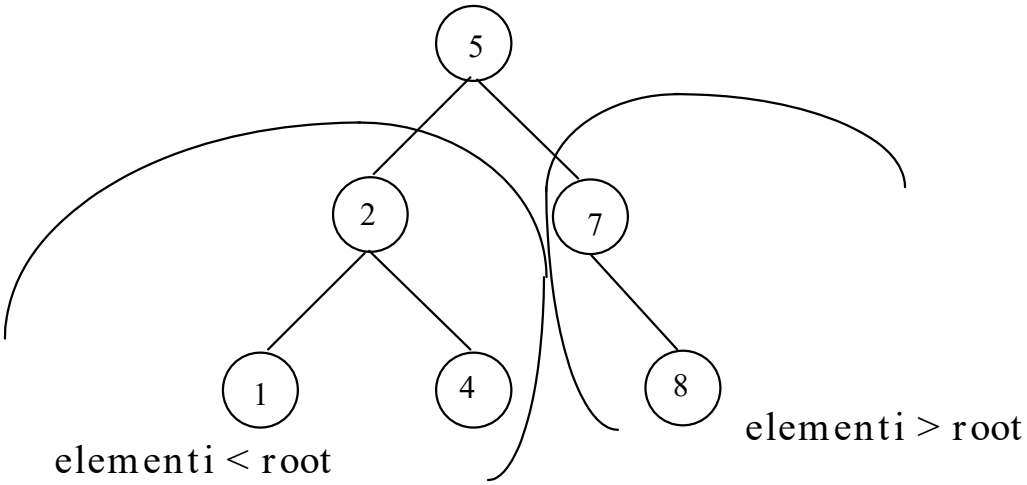
# ALBERI BINARI DI RICERCA (II)

## INSERIMENTO DI UN NUOVO ELEMENTO

Inserire un nuovo elemento in un albero binario di ricerca comporta la necessità di *rispettare la relazione d'ordine*, in modo che l'albero risultante sia ancora ordinato.

Questo però è molto semplice. Infatti

- *se l'elemento da inserire è minore (o uguale) al valore del nodo attuale,* l'elemento va nel sottoalbero di sinistra
- *altrimenti,* l'elemento va nel sottoalbero di destra.




Come nel caso delle liste, l'inserimento (non-primitivo) procede *per (ri)costruzione*, creando un nuovo albero uguale a quello dato, con *in più* l'elemento richiesto.

**L'ordine di inserimento degli elementi è rilevante:** inserendo gli elementi in ordine diverso, l'albero risulta diverso.

## ESEMPIO

L'albero sopra illustrato può essere costruito inserendo gli elementi nell'ordine 5, 2, 1, 4, 7, 8 (o anche 5, 2, 7, 4, 8, 1).

Un albero diverso può essere ottenuto, ad esempio, inserendo gli elementi nell'ordine 5, 4, 1, 2, 8, 7.

# ALBERI BINARI DI RICERCA (III)

INSERIMENTO DI UN NUOVO ELEMENTO (segue)

Occorre quindi *costruire comunque un nuovo albero*, e precisamente:

- un nuovo albero fatto *dal solo elemento da inserire*, se l'albero dato è vuoto;
- un nuovo albero avente *lo stesso sottoalbero di sinistra e un nuovo sottoalbero di destra*, se l'elemento va inserito nel sottoalbero di destra;
- un nuovo albero avente *un nuovo sottoalbero di sinistra e lo stesso sottoalbero di destra*, se l'elemento va inserito nel sottoalbero di sinistra.

Il "*nuovo sottoalbero*" (di sinistra o di destra) è quello che risulta dall'inserimento dell'elemento, rispettivamente, nel precedente sottoalbero di sinistra o di destra
→ ***algoritmo ricorsivo***:

```
tree insOrdTree(element e, tree t) {
 if (isEmptyTree(t))
   return consTree(e,emptyTree(),emptyTree());
 else
 if ( isLess(e,root(t)) || isEqual(e,root(t)) )
  return consTree(root(t),
                  insOrdTree(e,left(t)),
                  right(t));
 else
  return consTree(root(t),
                  left(t),
                  insOrdTree(e,right(t)));
}
```

NB: in questo modo *lo heap si riempie di schifezze*!

Servirebbe un ***garbage collector***.

# ALBERI BINARI DI RICERCA (IV)

## VERIFICA DI PRESENZA DI UN ELEMENTO TRAMITE RICERCA BINARIA

Come in ogni ricerca binaria, l'idea à di fare un confronto e, in base all'esito, *restringere la ricerca* a "una parte" (idealmente la metà) della struttura dati. In questo caso:

- si confronta l'elemento da cercare con il valore del nodo attuale: se è uguale, l'elemento è stato trovato;
- altrimenti, si prosegue la ricerca *o nel sottoalbero di sinistra o nel sottoalbero di destra*, secondo se l'elemento sia minore o maggiore del valore del nodo considerato.

```
boolean memberTreeOrd(element e, tree t) {
   if (isEmptyTree(t)) return false;
   else
   if (isEqual(e,root(t))) return true;
   else
   if (isLess(e,root(t)))
         return memberTreeOrd(e, left(t));
   else return memberTreeOrd(e, right(t));
}
```

## VALUTAZIONE DI COMPLESSITA'

Il numero di confronti è (nel caso peggiore, di elemento assente) proporzionale alla profondità dell'albero.

L'ideale sarebbe avere un albero ***bilanciato***, ossia avente lo stesso numero di nodi nei due sottoalberi (così da dimezzare ogni volta l'ampiezza del problema).

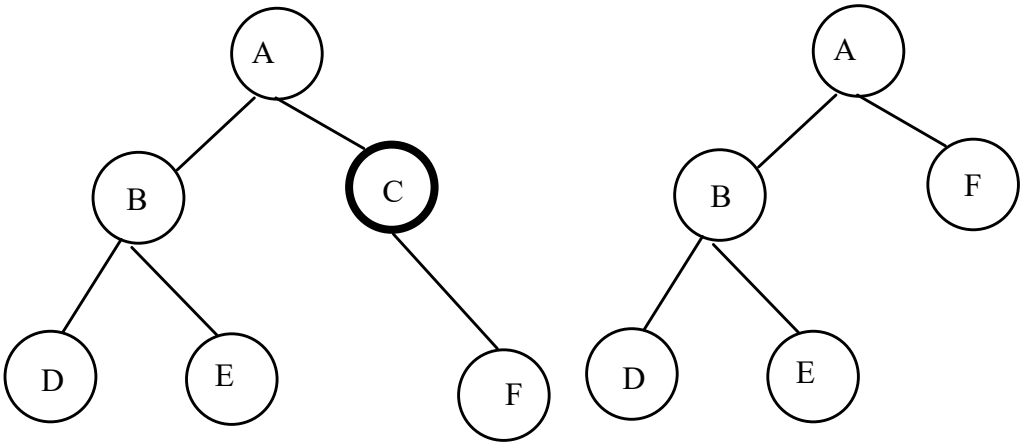
Esistono opportuni algoritmi per mantenere / rendere bilanciato un albero.

# ALBERI BINARI: ELIMINAZIONE DI UN NODO

Si distinguono *due casi*:

**1) il nodo da eliminare *manca di un sottoalbero***

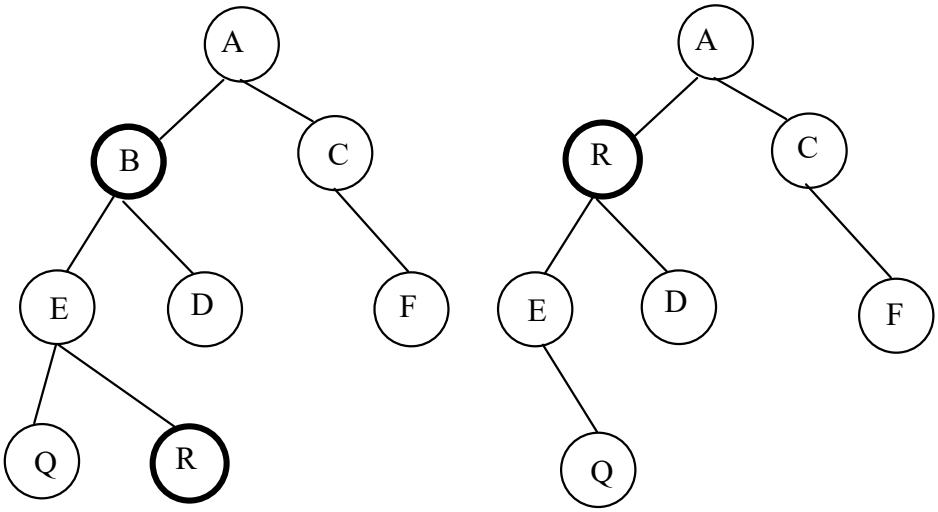
Basta inserire al suo posto l'altro sottoalbero.




**2) il nodo da eliminare *ha entrambi i sottoalberi***

Dipende se l'albero è o no un albero binario di ricerca.

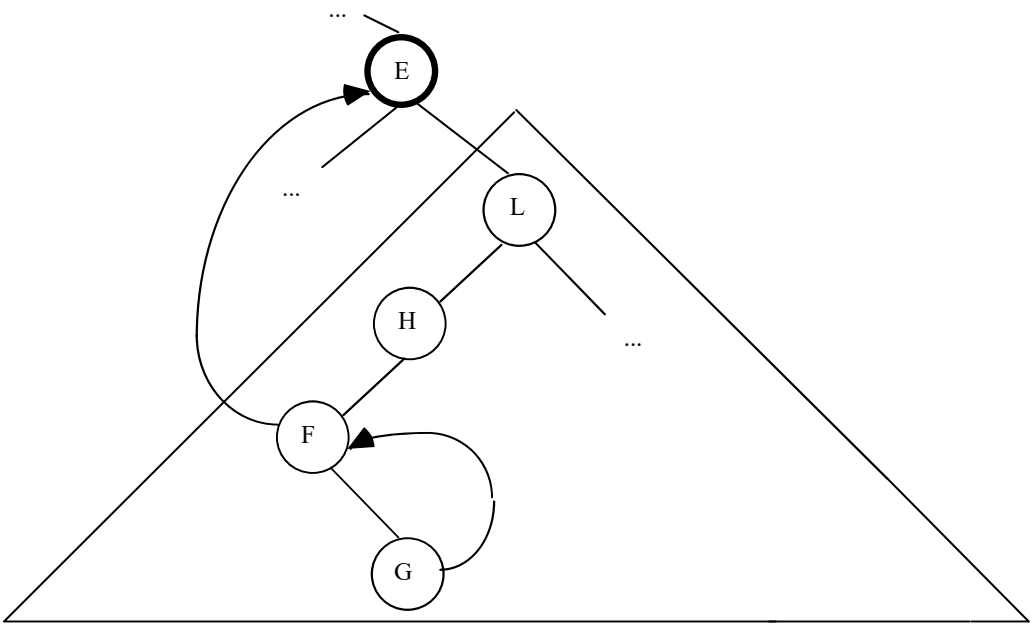
**Se l'albero *non* è un albero binario di ricerca** (ossia non è ordinato), basta *spostare una foglia* di un sottoalbero al posto del nodo soppresso.



**Se invece l'albero è un albero binario di ricerca** (e quindi *è ordinato*), bisogna adottare un algoritmo più sofisticato, che garantisca il mantenimento dell'ordine.

Detto N il nodo da eliminare:

- si sposta **succ(N)** al posto di **N**
- si sposta **right(succ(N))** al posto di **succ(N)**




***Il successore di un nodo N*** è il nodo M contenente il *minimo* valore *maggiore* di quello del nodo N.

Per come è definita la relazione d'ordine sugli alberi binari di ricerca, M è *il nodo più a sinistra del sottoalbero destro* di N.

(nell'esempio sopra, succ(E) = F)

In quanto minimo dell'insieme dei valori maggiori di N, tale valore può essere inserito al posto di N *mantenendo verificata la relazione d'ordine* su tutto l'albero.

Non è l'unico nodo con questa proprietà: in alternativa si sarebbe potuto anche prendere il *predecessore di N*, cioè *il nodo P più a destra del sottoalbero sinistro* di N.