# Alberi

## Alberi radicati

Non c'è la possibilità di fare di cicli.

Può essere importante distinguere l'ordine dei figli, gli alberi hanno una natura prettamente ricorsiva.

Per indicare un albero radicato

T = (V, E, R)\
U = insieme dei nodi
E = insieme degli archi
R = radice

$R \in V$
$E \subseteq VxV$ con $u, v \in V$
$e \in E$ (u, v)

$\forall (u, v), (u', v) \in E \Rightarrow u = u'$ non posso avere due coppie in cui ho lo stesso figlio e due padri diversi

$\nexists\, e = (u, R) \in E$

Con gli alberi riusciamo a rappresentare una relazione più complessa tra gli elementi dell'insieme.

Esiste un solo cammino per raggiungere una foglia a partire dalla radice.
La distanza di u nodo dalla radice è la lunghezza in archi dalla radice a quel nodo.
Il livello h rappresenta l'altezza dell'albero (percorso più lungo).

TIPOLOGIA: modo in cui i nodi sono messi insieme.

### Alberi binari

Ogni albero ha al più 2 figli.

### D-Ari

Ogni nodo ha al più d-figli

Sono detti **completi** nel caso in cui abbiano 0 oppure 2 figli (binari completi) o 0 oppure d figli (d-ari completi).

### Alberi bilanciati

Hanno le foglie quasi tutte della stessa altezza

### Perfettamente bilanciati

Hanno le foglie tutte della stessa altezza

# Albero binario completo perfettamente bilanciato

**Dimostrabile per induzione**

n° foglie albero altezza h = $2 \cdot (\text{n}^\circ \text{ foglie h - 1}) = 2 \cdot 2^{h-1} = 2^h$

$$\begin{aligned} \text{n}^\circ \text{ nodi albero altezza h} &= \text{n}^\circ \text{ foglie albero altezza h - 1} + \text{n}^\circ \text{ foglie nuove} \\ &= 2^{h-1} + 2^h = 2^{h+1} - 1 \end{aligned}$$

$$log(n+1) - 1 = h$$
$$h \in \Theta(logn)$$

L'albero binario ha altezza minima che è logaritmica al numero dei nodi

# Alberi visite

### Visita in profondità

- Previsita

```
vistita(R), Preorder(I1), ..., Preorder(Ik)
```

es.: 2, 7, 5, 6, 8, 2 ...

- Postvisita

```
Postorder(I1), ..., Postorder(Ik), ..., Visita(R)
```

es.: (((6, 8, 2, 9, 1)5)((1, 7, 2)3)... 2)

- Invisita $i \geq 1$

```
Invisita(T1), ...., Invisita(Te), Visita(R), Invisita(Ti + 1), ..., Invisit
```

### Visita in ampiezza

Visitare livello per livello, non è una procedura ricorsiva ma iterativa

## Visita per livelli

```
BFSR(R)
   Q := NEW_QUEUE()
   ENQUEUE(Q, R)
   while not IS_EMPTY_QUEUE(Q)
      a := DEQUEUE(Q)
      print chiave in a
      for each v appartenente a  children(a)
         enqueue(Q, v)
```

Il costo per accedere ai nodi è lineare al numero di nodi, faccio un numero di operazioni costanti per ogni nodo.
$\rightarrow \Theta(n)$

## Implementazione alberi

Possiamo scegliere di usare i vettori oppure nodi e puntatori

- Vettore: quando l'albero è molto regolare
- Nodi + Puntatori
    - Albero binario
    - Alberi Generici

## Alberi binari di ricerca

- Insieme chiavi tot. ordinato
- Albero binario
- Proprietà: dato un nodo a e v nel nuovo sottoalbero, se la chiave di v è > alla chiave di a allora v starà nelo sottoalbero di destra, di sinistra altrimenti

Primitive

1. Visita -> stampare in ordine crescente le chiavi

```
INORDER_TREE_WALK(t)
   if t != NIL
      then
         INORDER_TREE_WALK(t.left)
         print t.key
         INORDER_TREE_WALK(t.right)
```

Nella chiamata principale T sarà equivalente alla radice dell'albero principale.

$d \in \Theta(1)$ senza ricorsione

$T(n) \leq 2T(n-1) + d \in O(2^n) \rightarrow$ al più

> E' un analisi corretta ma fatta male (grande stima per eccesso), cerchiamo di fare una stima più precisa.

Intuizione: $\Omega(n)$ - $O(n)$

$T(n) \leq 2dn + d$

Caso base n = 0
$T(n) \leq d$ VERO

**IPOTESI INDUTTIVA**

$\forall m : 0 \leq m \leq n-1$
$T(m) \leq dm + d$

$$\begin{aligned} T(n) &\leq T(k) + T(k-k+1) + d \\ &\leq (2dk + d) + (2d(n-k-1) + d) + d \end{aligned}$$

2. Ricerca chiave

```
SEARCH(t, k)
   if t = NIL or t.key = k
      then return t
   if k < t.key
      then SEARCH(t.left, k)
      else SEARCH(t.right, k)
```

3. Valore minimo

```
MIN_VAL(t)
   if t.left = NIL
      then return t.key
   else
      MIN_VAL(t.left)
```

4. Inserimento Key (senza duplicazioni)

```
INSERT(t, k)
   if t = NIL
      then
         t := new_node_tree()
         t.key := k
         t.left := NIL
         t.right := NIL
      else
         if k < t.key
            then t.left := INSERT(t.left, k)
         if k > t.key
            then t.right := INSERT(t.right, k)
```

5. Cancellazione Chiave

```
CANCEL(t,k)
   if NOT t = NIL
      then if t.key = k
         then //caso 3 - valore da eliminare trovato
         if t.left = NIL
            then return t.right // caso 3.1 o 3.2 - al max 1 figlio
            else if t.right = NIL
               then return t.left // caso 3.2 - c'e' figlio sinistro
               else t.key := min_val(t.right) //caso 3.3
                  t.right := cancel(t.right,t.key)
         else if t.key < k then t.left := cancel(t.left,k) //caso 1
                          else t.right := cancel(t.right,k)// caso 2
   return t //include il caso in cui k non sia presente in T (t=NIL)
```