HeapIntersect (heap1, heap2)

$n > m$

$T(n,m) = O((n+m)\log n) = O(n \log n)$

```
H = nuovo heap inizialmente vuoto
H.heapsize = 0;
while ( heap1.heapsize > 0  &&  heap2.heapsize > 0 ) {
    if ( Heap_Maximum(heap1) == HeapMaximum(heap2) ) {
        e = Heap_Extract_Max (heap1);
        Max-Heap-Insert (H, e);
        Heap_Extract_Max (heap2);
    }
    else if ( Heap_Maximum (heap1) > Heap_Maximum (heap2) ) {
        HeapExtractMax (heap1);
    }
    else HeapExtractMax (heap2);
}
return H;
```

$O(n+m)$

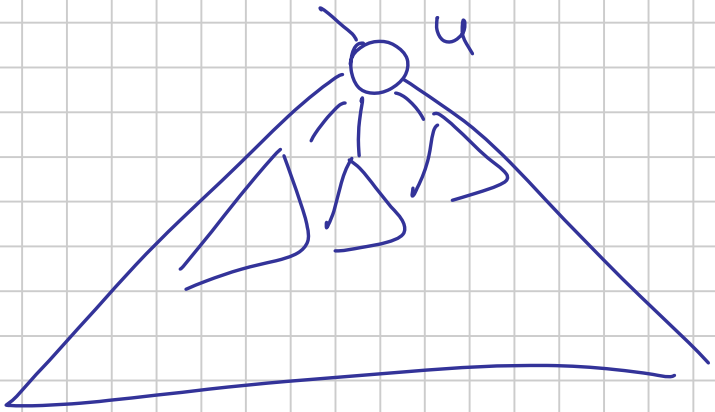
$O(\log n)$ ·
- $O(\log m)$
- $O(\log m)$
- $O(\log n)$

oppure
- $O(\log m)$
- $O(\log n)$

Titolo nota 26/02/2016

ALBERI (BINARI)

- Struttura dati composta da NODI
- l'albero associa ad ogni nodo (eccetto uno, chiamato RADICE) un nodo PADRE (unico)
- ogni nodo può avere uno o più nodi figli (due negli alberi binari)
- nodi senza figli → FOGLIE
- ~~[illegible]~~ l'unico nodo senza PADRE è la RADICE
- un albero di n nodi contiene esattamente $n-1$ archi (riferimenti al nodo padre)



Sottoalbero radicato in u

$\forall v \in$ sottoalbero di radice u,

- v è un DISCENDENTE di u
- u è un ANTENATO di v

ALBERO BINARIO (definizione ricorsiva)

Un albero binario è una struttura definita su un insieme di nodi che

- non contiene alcun nodo (ALBERO VUOTO), oppure
- contiene un nodo detto RADICE, un albero binario detto sottoalbero SX della radice, e un albero binario detto sottoalbero DX della radice
- se il sottoalbero SX (DX) non è vuoto, la sua radice è chiamata figlio SX (DX) della radice

Rappresentazione in memoria

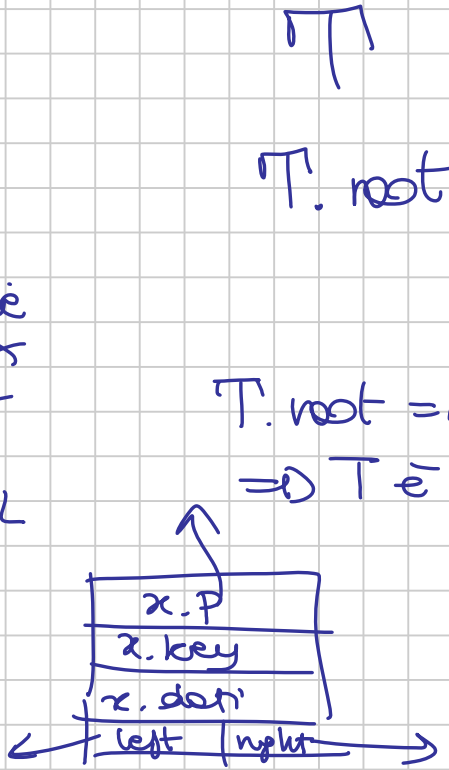
struttura concatenata

ogni nodo x è un oggetto che contiene:

$x.key$	chiave
$x.dati$	dati satellite
$x.p$	puntatore al nodo padre
$x.left$	puntatore al figlio SX
$x.right$	puntatore al figlio DX

x è una foglia $\iff$ $x.left == x.right == NIL$

x è la radice dell'albero T $\iff$ $x.p == NIL$



$T.root$ = puntatore alla radice di T

$T.root = NIL$ $\Rightarrow$ T è l'albero vuoto

ALBERI BINARI : VISITE

Visita: esame sistematico di tutti i nodi dell'albero

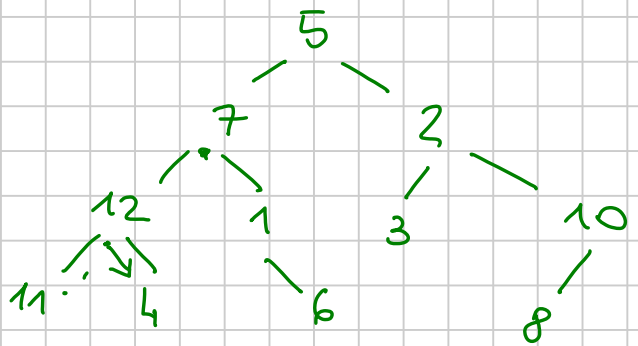
3 visite

(1) VISITA ANTICIPATA (Previsita)

la visita di un nodo precede la visita ricorsiva dei suoi sottoalberi.

ANTICIPATA(u) // u è un nodo di T

```
if (u ≠ NIL) {
    * esamina u
    Anticipata(u.left)
    Anticipata(u.right)
}
```

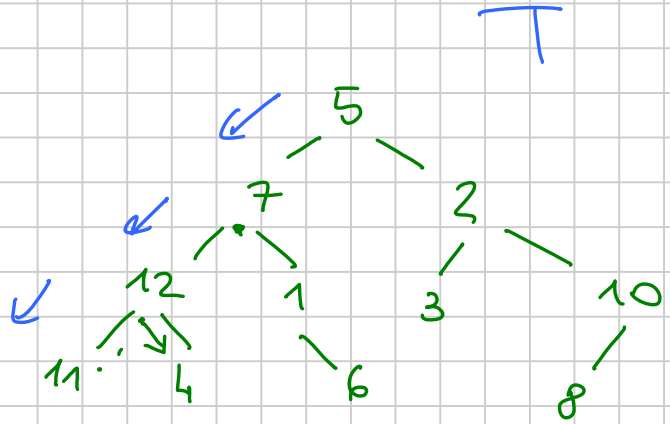


VISITA ANTICIPATA = 5, 7, 12, 11, 4, 1, 6, 2, 3, 10, 8

(2) VISITA SIMMETRICA o INVISITA

la visita di un nodo è intermedia alla visita ricorsiva dei suoi sottoalberi

```
Simmetrica (u)
if (u ≠ NIL) {
    Simmetrica (u.left);
    * esamina u
    Simmetrica (u.right);
}
```

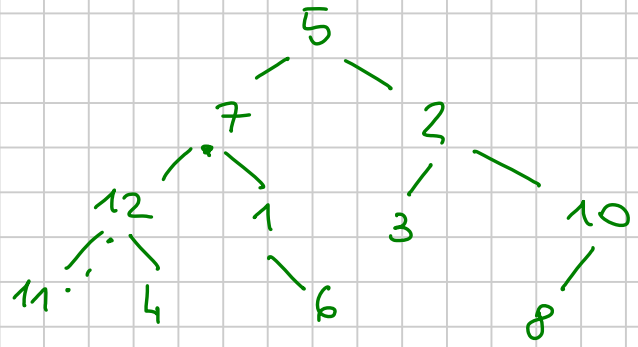


VISITA: 11, 12, 4, 7, 1, 6, 5, 3, 2, 8, 10

③ VISITA POSTICIPATA o POST VISITA

L'esame di un nodo segue la visita ricorsiva dei suoi sottoalberi

```
Posticipata (u)
if (u ≠ NIL) {
    Posticipata (u.left);
    Posticipata (u.right);
    * esamina u;
}
```

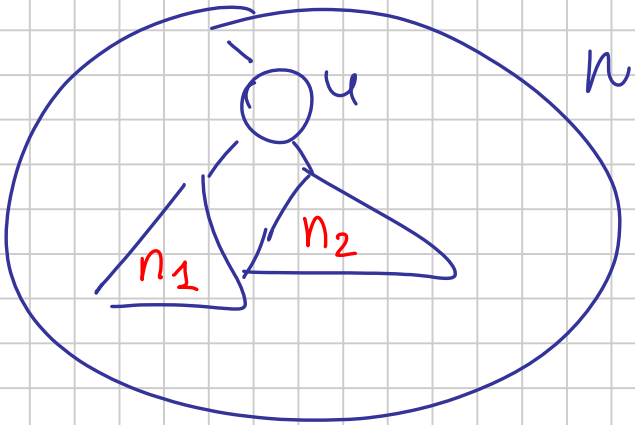


VISITA: 11, 4, 12, 6, 1, 7, 3, 8, 10, 2, 5

Analisi

- $T(n) = \Omega(n)$
- $\boxed{T(n) = O(n)}$

n = # nodi nell'albero T



$$T(n) = \begin{cases} c & n = 0 \\ T(n_1) + T(n_2) + d \end{cases}$$

c: costante

d: costo di "esame u"

supponiamo $d = \Theta(1)$

Risolviamo la ricorrenza con il metodo di sostituzione

IPOTESI $T(n) \leq (c+d) \cdot n + c$

$n = 1 + n_1 + n_2$

$n_1 < n$

$n_2 < n$

Dimostrazione per induzione

CASO BASE

$n = 0$ $\quad T(0) \leq c$ ✓ $\quad (T(0) = c)$

$n > 0$

$$T(n) = T(n_1) + T(n_2) + d \overset{\text{i.i.}}{\leq} \underbrace{\left[(c+d) \cdot n_1 + c\right]}_{T(n_1)} +$$

$$+ \underbrace{\left[(c+d)\, n_2 + c\right]}_{T(n_2)} + d = (c+d)\, n_1 + (c+d)\, n_2 + (c+d) + c$$

$$= (c+d)(n_1 + n_2 + 1) + c = (c+d) \cdot n + c$$ ✓

se d è costante $\Rightarrow$

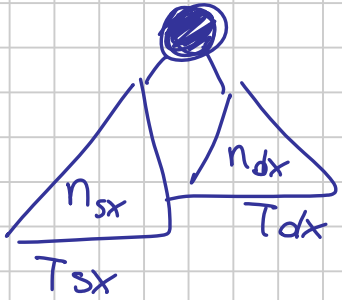
$$T(n) = O(n)$$

supponiamo che $T(n) = \Omega(n)$

$\Downarrow$

$$T(n) = \Theta(n)$$

ESEMPI: Calcolo della dimensione (# nodi) di un AB



$n = ?$

$n = 1 + n_{sx} + n_{dx}$

D & I sugli alberi

```
Dim(u)
if (u == NIL) return 0;
nsx = Dim(u.left);
ndx = Dim(u.right);
return 1 + nsx + ndx;
```

chiusura

Ricorsione

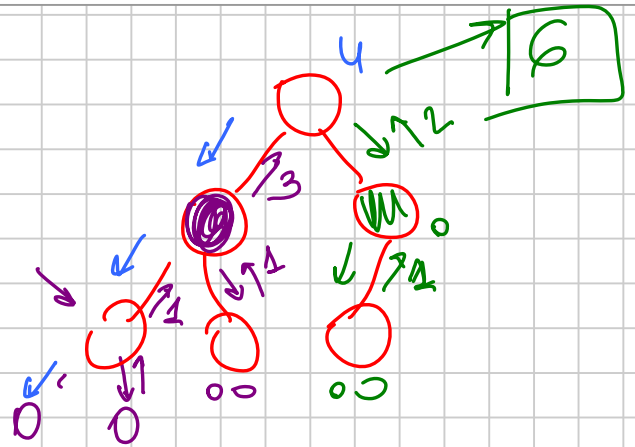
combinazione

"post visita"

$T(n) = \Theta(n)$

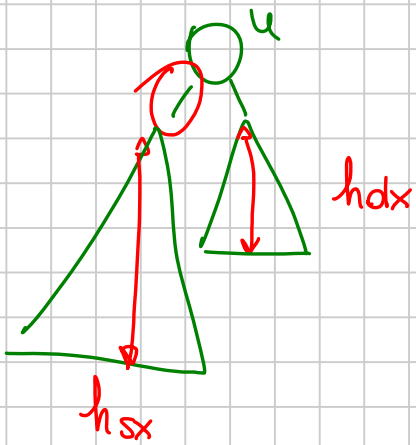
Titolo nota

26/02/2016



Calcolo dell'altezza

```
Altezza(u)
if (u == NIL) return -1;
hsx = Altezza(u.left);
hdx = Altezza(u.right);
return 1 + max{hsx, hdx}
```



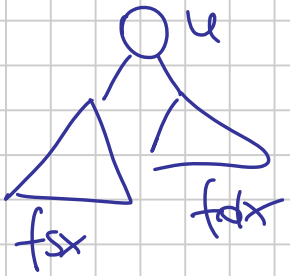
$$h = \max\{h_{sx}, h_{dx}\} + 1$$

$$T(n) = \Theta(n)$$

Calcolo del # di foglie

(ERRATO)

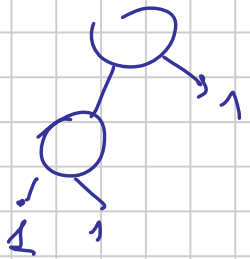
```
Foglie (u)
if (u == NIL) return 0;
fsx = Foglie (u.left);
fdx = Foglie (u.right);
return fsx + fdx;
```



$f = ? = fsx + fdx$

⟶ restituisce sempre 0 !!!

Versione CORRETTA: chiamata sulla foglia.




```
Foglie (u)
  if (u == NIL) return 0;
  if (u.left == NIL && u.right == NIL) return 1;
  fsx = Foglie (u.left)
  fdx = Foglie (u.right)
  return fsx + fdx;
}
```

$T(n) = \Theta(n)$

Alberi con grado di ramificazione > 2 (eventualmente illimitato)

ALBERI CARDINALI

1) nodi di al più k figli, k costante

x.p
x.child 1, x.child 2, ---, x.child k

k riferimenti

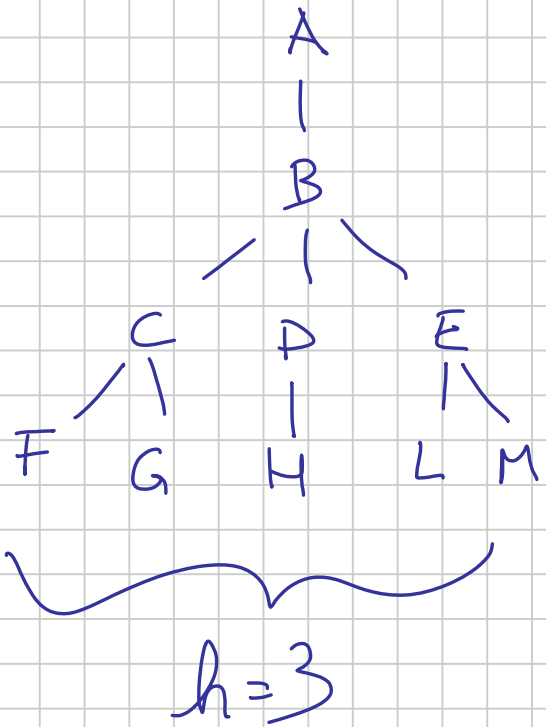
oppy di k riferimenti

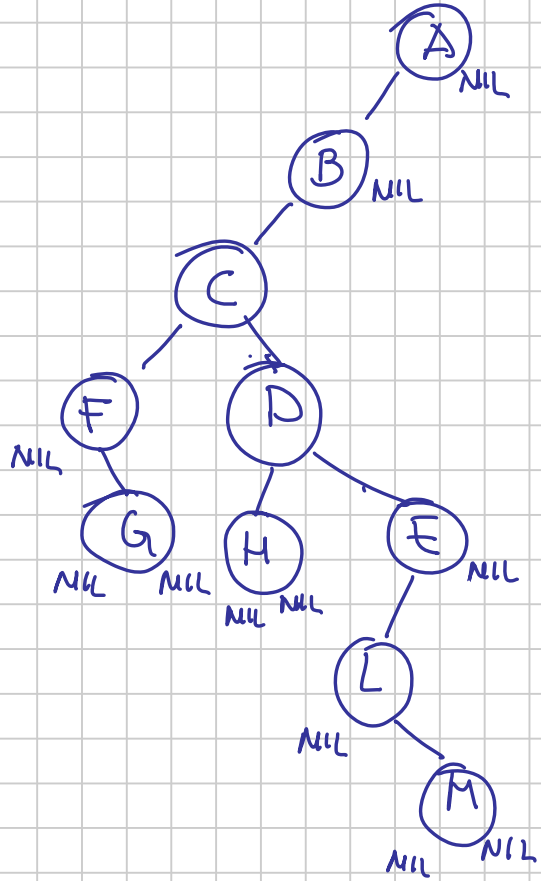
2) ALBERO ORDINATO: non si distinguono i figli → lista / # figli illimitato

Rappresentazione con la NOTAZIONE BINARIZZATA

tre riferimenti $\forall$ nodo x

- $x.p$ puntatore al padre di x
- $x.$ left-child puntatore al primo figlio di x
- $x.$ right-sibling puntatore al fratello di x (immediatamente successivo sulla destra)





foglie dell'albero ordinato sono tutti e soli i nodi con puntatore left-child uguale a NIL

$\rightsquigarrow h = 6$