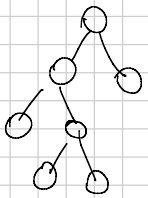
Titolo nota 21/04/2016

ESERCITAZIONE

(1) Algoritmo che verifica se un albero binario è completo.



$T(n) = O(n)$

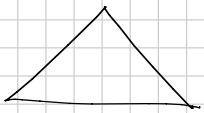
USITA

```
Completo(u)
if (u==NIL || (u.left==NIL && u.right==NIL)) return TRUE;
if ((u.left==NIL && u.right≠NIL) || (u.left≠NIL && u.right==NIL)) return FALSE;
• if (!Completo(u.left)) return FALSE;
• return Completo(u.right);
```

⟺ return (Completo(u.left) && Completo(u.right));

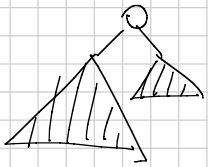
Titolo nota 26/02/2016

(2) Algoritmo per verificare se un albero binario è completamente bilanciato



pieno su tutti i livelli

Un AB è CB ⟺ è completo e le foglie hanno tutte la stessa profondità



AB di radice u è CB ⟺
1. i sottoalberi di radice u.left e u.right sono CB
2. e hanno la stessa altezza

```
CB1(u)
if (u==NIL) return TRUE;
if (!CB1(u.left)) return FALSE;
if (!CB1(u.right)) return FALSE;
// i due sottoalberi sono completamente bilanciati   (1)
return (Altezza(u.left) == Altezza(u.right));        (2)
```

$\forall$ nodo, spendo $\rightarrow$ $O(n)$

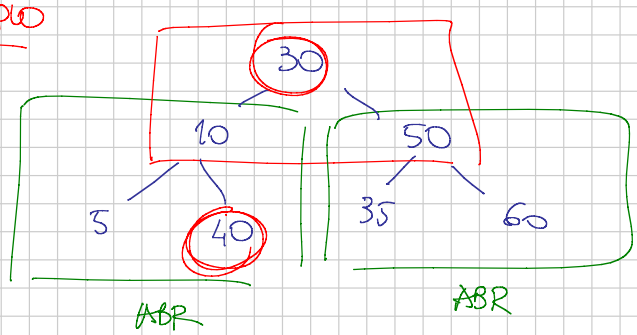
$T(n) = O(n^2)$

si può portare a $O(n \log n)$


```
CB2(u) // < booleano: isCB, intero: altezza >
if (u==NIL) return <TRUE, -1>;          } chiamata   Θ(1)
<bilsx, hsx> = CB2(u.left);             } due volte ricorsive
<bildx, hdx> = CB2(u.right);            ] costi costanti
h = 1 + max{hsx, hdx};                  ]   Θ(1)
bil = bilsx && bildx && (hsx == hdx);   ]
return <bil, h>;
```

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

CONTROESEMPIO



NON è ABR

T è ABR $\iff$ T_{SX} e T_{DX} sono ABR

$MAX(T_{SX}) < k(\text{chiave radice}) < MIN(T_{DX})$

ABR1(u)

```
if (u==NIL) return TRUE;
if (u.left == NIL && u.right == NIL) return TRUE;
if ( ! ABR1(u.left) || ! ABR1(u.right)) return FALSE;
// u non è una foglia
maxS = TREE-MAXIMUM(u.left);
minD = TREE-MINIMUM(u.right);
if (maxS == NIL) return (u.key < minD.key);
if (minD == NIL) return (maxS.key < u.key);
else return (maxS.key < u.key && u.key < minD.key);
```

- $T(n) = O(n \cdot h) = O(n^2)$

Titolo nota 26/02/2016

Versione più efficiente

```
ABR2(u) // < booleano: è un ABR, intero: chiave minima, intero: chiave massima >;
if (u==NIL) return < TRUE, +∞, -∞ >;
// if (u.left==NIL && u.right==NIL) return < TRUE, u.key, u.key >;
< abrSx, minSx, maxSx > = ABR2(u.left);
< abrDx, minDx, maxDx > = ABR2(u.right);
abr = abrSx && abrDx && maxSx < u.key && u.key < minDx;
min = Math.min(minSx, minDx, u.key);
max = Math.max(maxSx, maxDx, u.key);
return <abr, min, max>
```

chiamata $\Theta(1)$

chiamate ricorsive

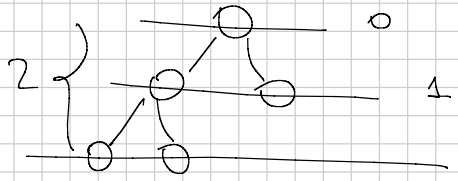
ricombinazione $\Theta(1)$

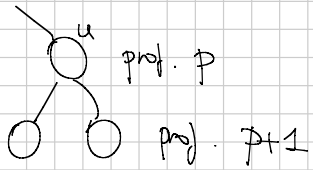
$T(n) = \Theta(n)$

Titolo nota 26/02/2016

(4) Dato un albero binario progettare un algoritmo che stampi chiave e profondità di ciascun nodo.

↳ DISTANZA dalla RADICE (# archi)

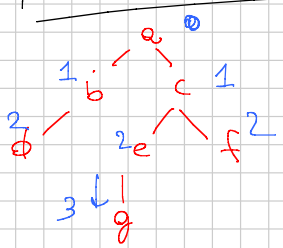




```
StampaProf. (u, p)   // p è la prof. del nodo u
if (u ≠ NIL) {
  else if (u.left == NIL && u.right == NIL)
        print " u.key, p ";
  else { print " u.key, p ";
         StampaProf (u.left, p+1);
         StampaProf (u.right, p+1);
  }
}
```

p: parametro di ingresso per propagare informazioni che dipendono dagli antenati

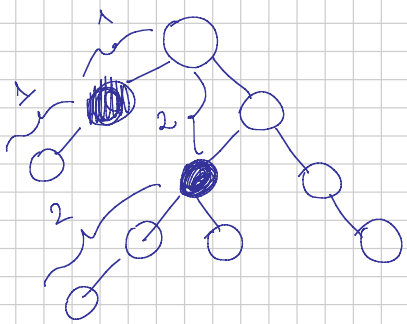
I^a chiamata: StampaProf (T.root, 0)



a, 0
b, 1
d, 2
c, 1
e, 2
g, 3
f, 2

(5) Progettare un algoritmo ricorsivo che stampi le chiavi di tutti i NODI CARDINE presenti in un Albero Binario

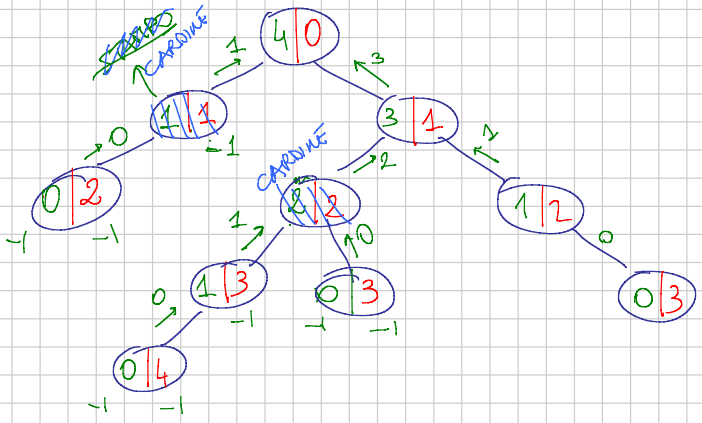
u è un nodo cardine ⟺ la profondità di u è uguale all'altezza di u (ossia altezza del sottoalbero radicato in u)



```
Cardine (u, p)   // p è la profondità di u, 1ª chiamata Cardine ("T.root, 0)
if (u == NIL) return -1;
altSx = Cardine (u.left, p+1);
altDx = Cardine (u.right, p+1);
alt = 1 + max{altSx, altDx};
if (p == alt) print u.key;
return alt;
```

// il codice calcola l'altezza e simultaneamente stampa le chiavi dei nodi Cardine

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

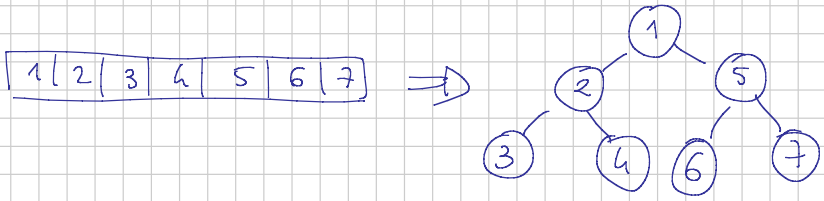


verde: altezza
rosso: profondità

Titolo nota 26/02/2016

(8) array a di n interi

Costruire ricorsivamente in tempo $\Theta(n)$ un albero binario bilanciato tale che $a[i]$ sia i-esimo campo u.key in ordine di visita anticipata.

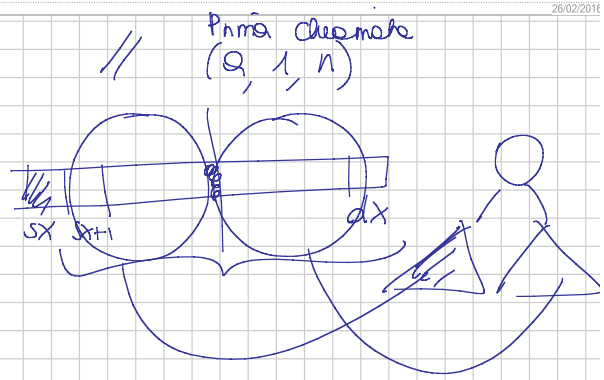


Titolo nota 26/02/2016

```
Anticipata (a, sx, dx)
if (sx > dx) return NIL;
u = nuovo Nodo()
u.key = a[sx];
cx = ((sx+1) + dx) / 2
u.left = Anticipata(a, sx+1, cx);
u.right = Anticipata(a, cx+1, dx);
return u;
```

$\Theta(1)$ | $2T(\frac{n}{2})$ | $\Theta(1)$

// Prima chiamata (a, 1, n)



$$T(n) = \begin{cases} \Theta(1) & n = 0 \\ 2T(\frac{n}{2}) + \Theta(1) & n > 0 \end{cases}$$

$T(n) = \Theta(n)$ teorema dell'esperto, caso 1.

Albero ordinale rappresentato in forma binaria.

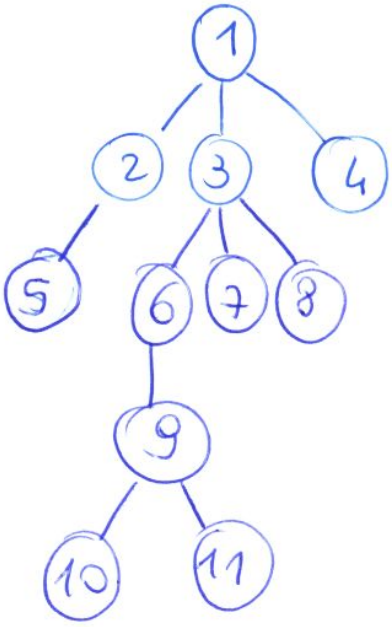
foglie

```
Foglie (u)
    if (u == NIL) return 0;
    if (u.left-child == NIL)
            return 1 + Foglie (u.right-sibling);
    else return Foglie (u.left-child) +
                Foglie (u.right-sibling)
```

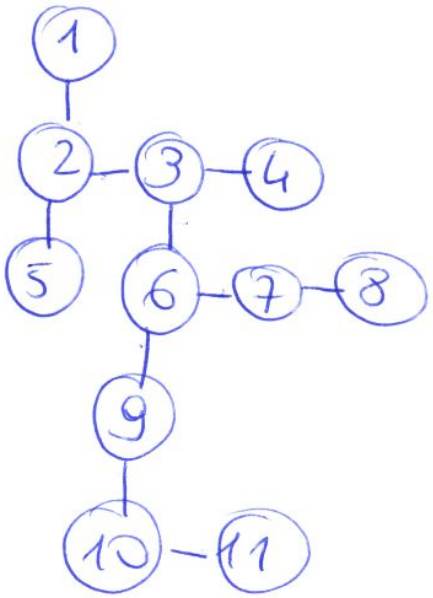
$T(n) = \Theta(n)$

Albero ordinale rappresentato in forma binarizzata

Altezza



albero ordinale, $h = 4$



forma binarizzata (albero di altezza 6)

```
Altezza (u)
if (u == NIL) return -1;
altFiglio = Altezza (u.left-child);
altFratello = Altezza (u.right-sibling);
altezza = max {1+altFiglio, altFratello}
return altezza;
```

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

Titolo nota 26/02/2016

(1) array S di n chiavi intere, non necessariamente distinte
trovare il numero r di chiavi distinte di S, con un'unica scansione.
Suggerimento: usare un dizionario D

```
ChiaviDistinte(S)   // dim S = n
D = nuovo dizionario
ctr = 0
for i = 1 to n {
    if( search(D, S[i]) == NULL) {
        ctr++;
        insert(D, S[i]);
    }
}
return ctr;
```

Titolo nota 26/02/2016

D = array ordinato

n = dim S
r = # chiavi distinte

n ricerche in $D \rightarrow O(\log r)$
r inserimenti in $D \rightarrow O(r)$
$|D| = r$

$$T(n, r) = O(n \log r + r^2)$$

D = AVL

n ricerche $\rightarrow O(\log r)$
r inserimenti $\rightarrow O(\log r)$

$$T(n, r) = (n \log r + r \cdot \log r)$$

Titolo nota 26/02/2016

(3) T = ABR.

Progettare un algoritmo che restituisca il numero di elementi di T di chiave $\leq k$.

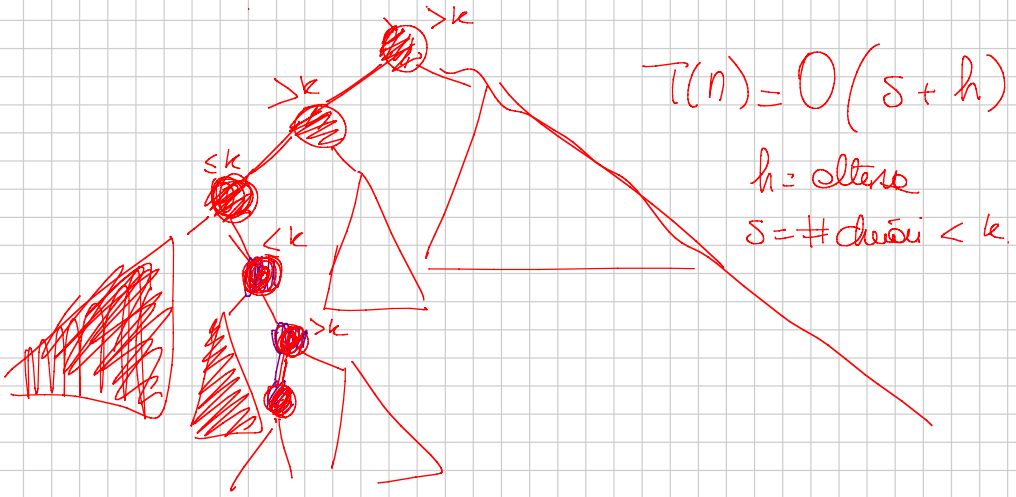
$T(n) = O(n)$

```
COUNTLE(u, k)
if (u == NIL) return 0;
if (u.key > k) return COUNTLE(u.left, k);
else return 1 + COUNTLE(u.left, k) + COUNTLE(u.right, k)
```

s = # chiavi $\leq k$

$S = \Theta(n)$ visito tutto T.

Titolo nota 26/02/2016



$T(n) = O(s + h)$

h = altezza

s = # chiavi $< k$

Titolo nota 26/02/2016

ESERCIZIO 4

T = albero binario completo (ogni nodo ha 2 figli, eccetto le foglie)

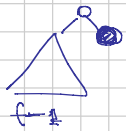
v nodo di T

$Imbalance(v) = |$ # foglie del sottoalbero sx di v − # foglie sottoalbero dx $|$

$$Imbalance(T) = \max_{v \in T} Imbalance(v)$$

(1) Valore massimo $Imbalance(T)$, dove T è un ABC di dim n.

foglie $T = f$



$Imbalance(T) = (f-1) - 1 = f - 2$

Titolo nota 26/02/2016

$$f = \frac{n+1}{2} = \left\lceil \frac{n}{2} \right\rceil$$ (si dimostra per induzione)

$$Imbalance(T) \leq \frac{n+1}{2} - 2 = \frac{n-3}{2}$$

(2) per esercizio

(3) Algoritmo per calcolare $Imbalance(T)$

```
Imbalance(u)  // due parametri di uscita  < maxImb, #foglie >
                                            dell'albero radicato in u
if (u==NIL) return <0, 0>;
if (u.left==NIL && u.right==NIL) return <0, 1>
<ImbSx, fSx> = Imbalance(u.left);
<ImbDx, fDx> = Imbalance(u.right);
f = fSx + fDx;
imb = max{ ImbSx, ImbDx, |fSx - fDx| }
return <imb, f>
```

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