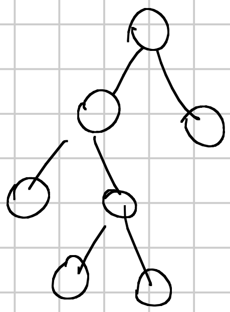
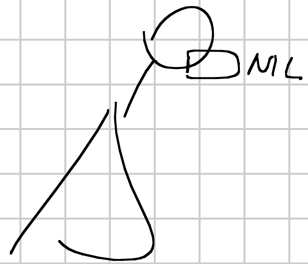
ESERCITAZIONE

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





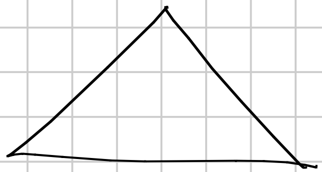
$$\boxed{T(n) = O(n)}$$

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).
```

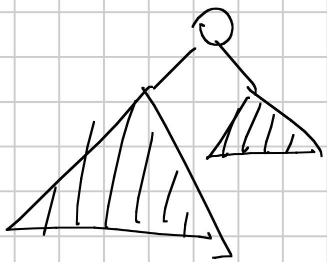
⟺ return (Completo(u.left) && Completo(u.right))

2 Algoritmo per verificare se un albero binario è completamente bilanciato



pieno su tutti i livelli

Un AB è CB $\iff$ è completo e le foglie hanno tutte la stessa profondità



AB di radice u è CB $\iff$

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)
```

∀ nodo, spendo → $O(n)$

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

↓

si può portare a $O(n \log n)$

Titolo nota 26/02/2016

CB2 (u) // < booleano: isCB , intero: altezza >

```
if ( u == NIL ) return < TRUE , -1 >;
< bilsx , hsx > = CB2 ( u.left );
< bildx , hdx > = CB2 ( u.right );
h = 1 + max{ hsx, hdx };
bil = bilsx && bildx && ( hsx == hdx );
return < bil , h >;
```

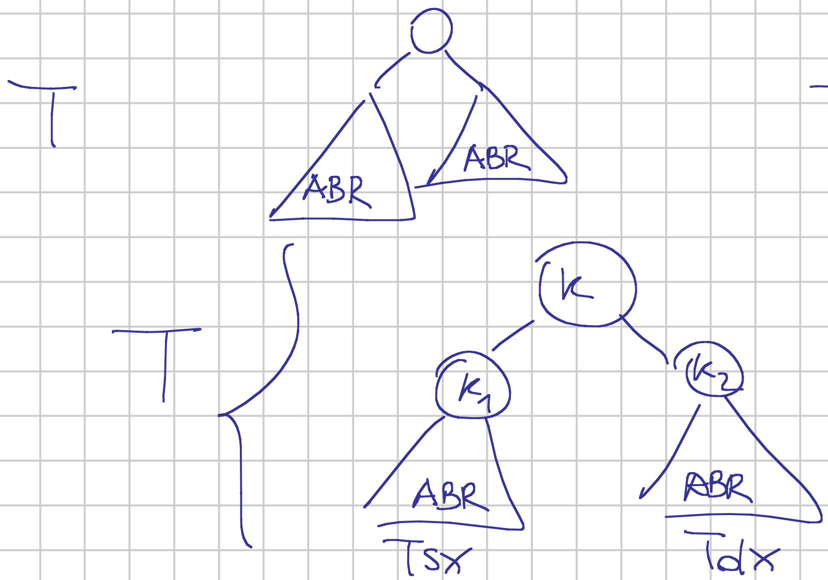
chiusura $\Theta(1)$

chiamate ricorsive

combinazione $\Theta(1)$

$$\boxed{T(n) = \Theta(n)}$$

(3) Progettare un algoritmo che verifichi se un AB i cui nodi contengono chiavi intere è un ABR.



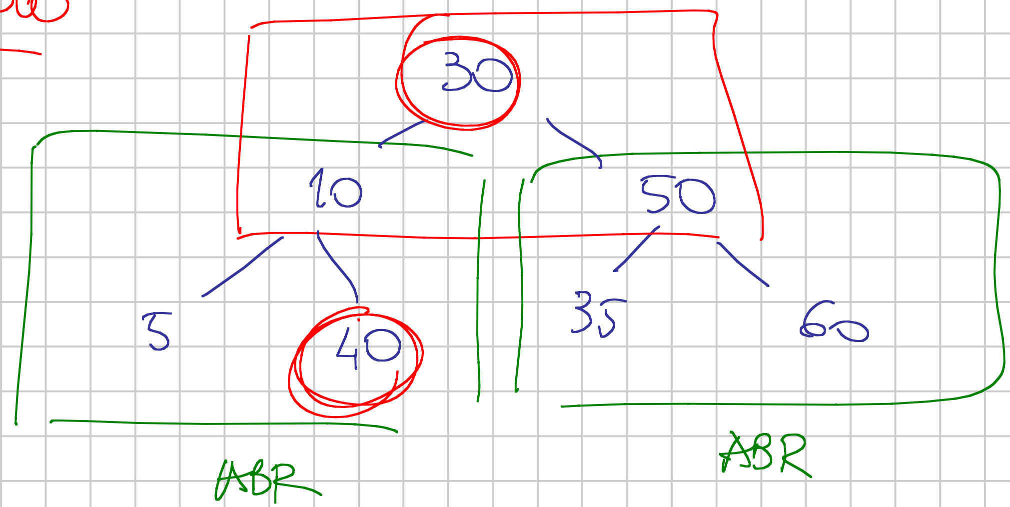
T è ABR?

~~T è ABR $\Longleftrightarrow$ Tsx e Tdx sono ABR e $k_1 < k < k_2$~~ $\Longrightarrow$

NO

condizione necessaria, ma non sufficiente.

CONTROESEMPIO



NON è ABR

T è ABR $\iff$ Tsx e Tdx sono ABR

MAX(Tsx) < k (chiave radice) < MIN (Tdx)

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>
```

chiusura $\Theta(1)$

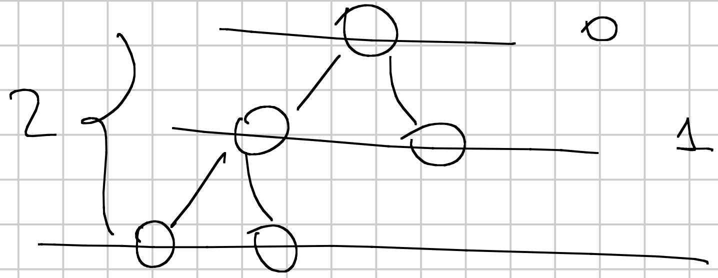
chiamate ricorsive

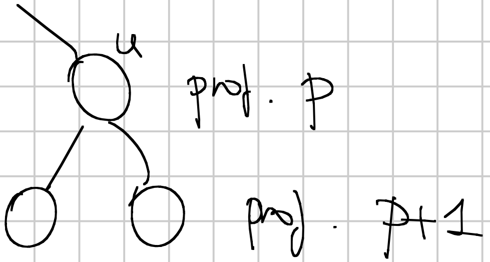
ricombinazione $\Theta(1)$

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

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

↳ DISTANZA dalla RADICE (# archi)



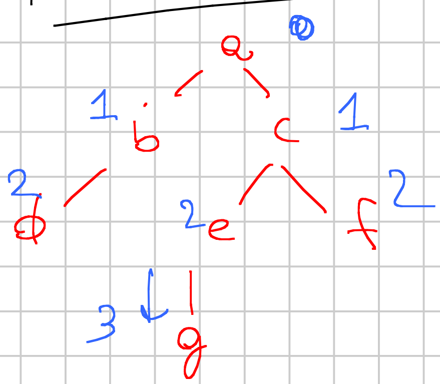


```
StampaProf. (u, p)   // p è la prof. del nodo u
if (u ≠ NIL) {
else if (u.left == NIL && u.right == NIL)
        print " u.key, p "
else {  print " u.key, p ";
        StampaProf (u.left, p+1);
        StampaProf (u.right, p+1);
     }
}
```

p: parametro di ingresso per propagare informazione che scende dagli antenati

I^a chiamata

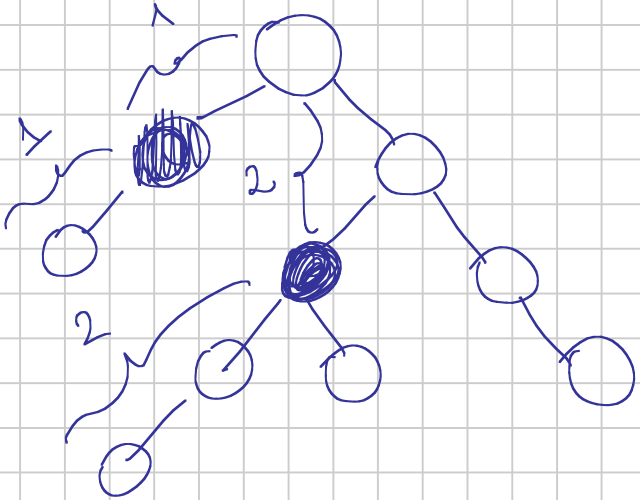
$\boxed{\text{StampaProf (T.root, 0)}}$



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 $\iff$ la profondità di u è uguale all' altezza di u (~~[illegible]~~ altezza del sottoalbero radicato in u)



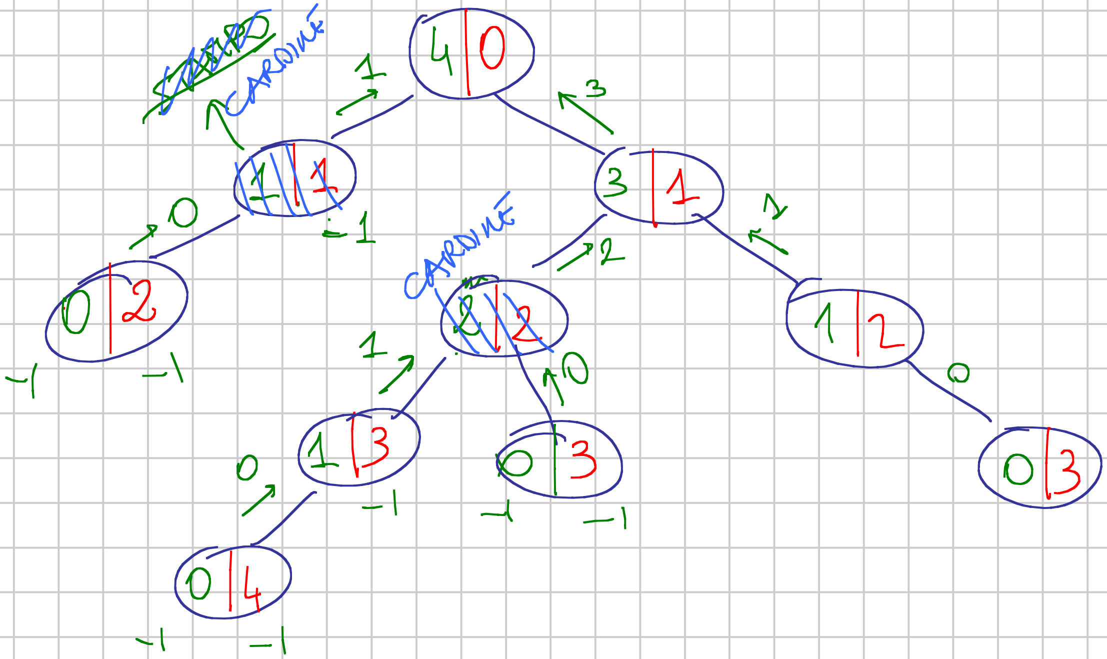
Titolo nota 26/02/2016

```
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);                    // il codice calcola l'altezza
altDx = Cardine(u.right, p+1);                      e simultaneamente stampa
alt = 1 + max{altSx, altDx};                        le chiavi dei nodi cardine
if (p == alt) print u.key;
return alt;
```

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

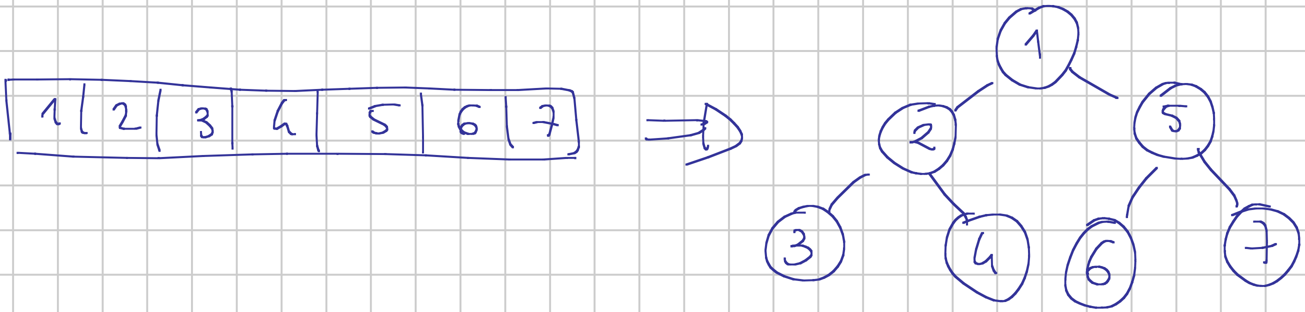
verde: altezza

rosso: profondità



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.



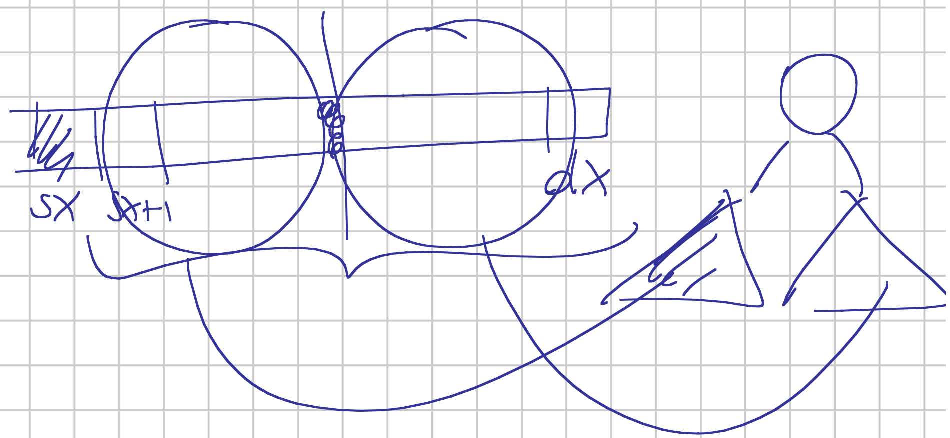
```
Anticipata(a, sx, dx)
if (sx > dx) return NIL;
u = nuovo Nodo()
u.key = a[sx];
cx = ((sx+1) + dx) / 2
u.left = Anticipata(a, sx+1, cx);
u.right = Anticipata(a, cx+1, dx);
return u;
```

$\Theta(1)$ (righe da `if` a `cx`)

$2T\left(\frac{n}{2}\right)$ (chiamate ricorsive)

$\Theta(1)$ (`return u;`)

Prima chiamata (a, 1, n)



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

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