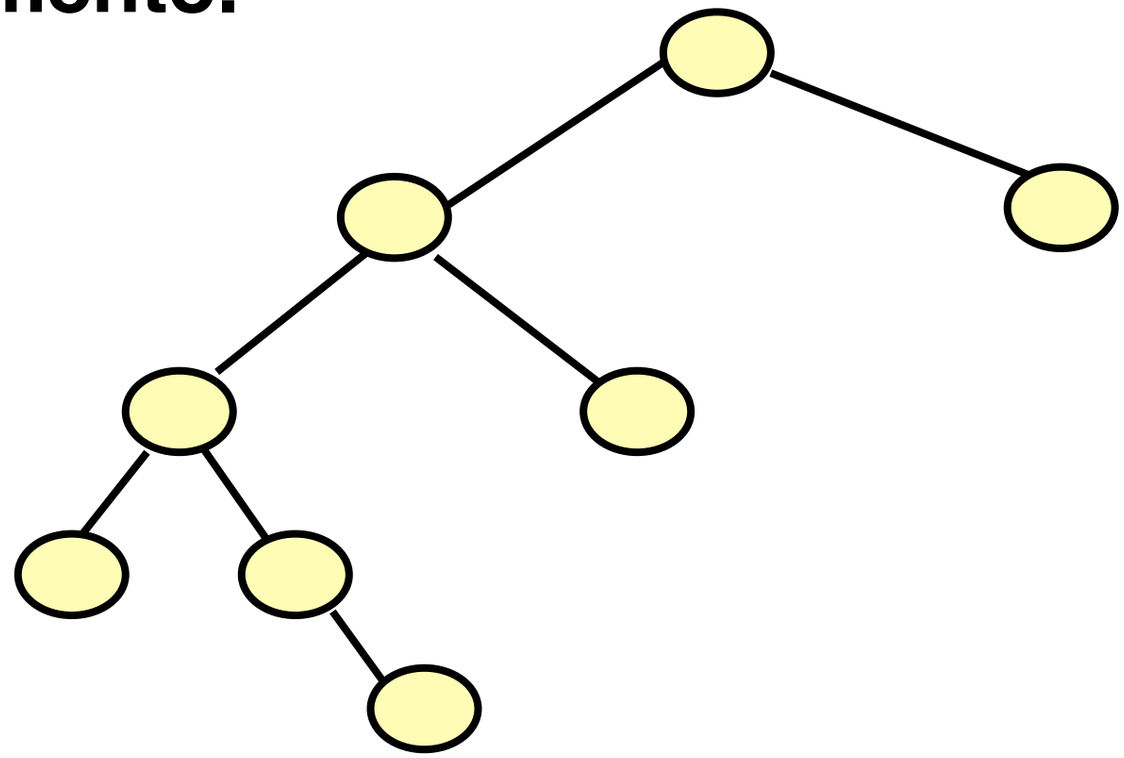
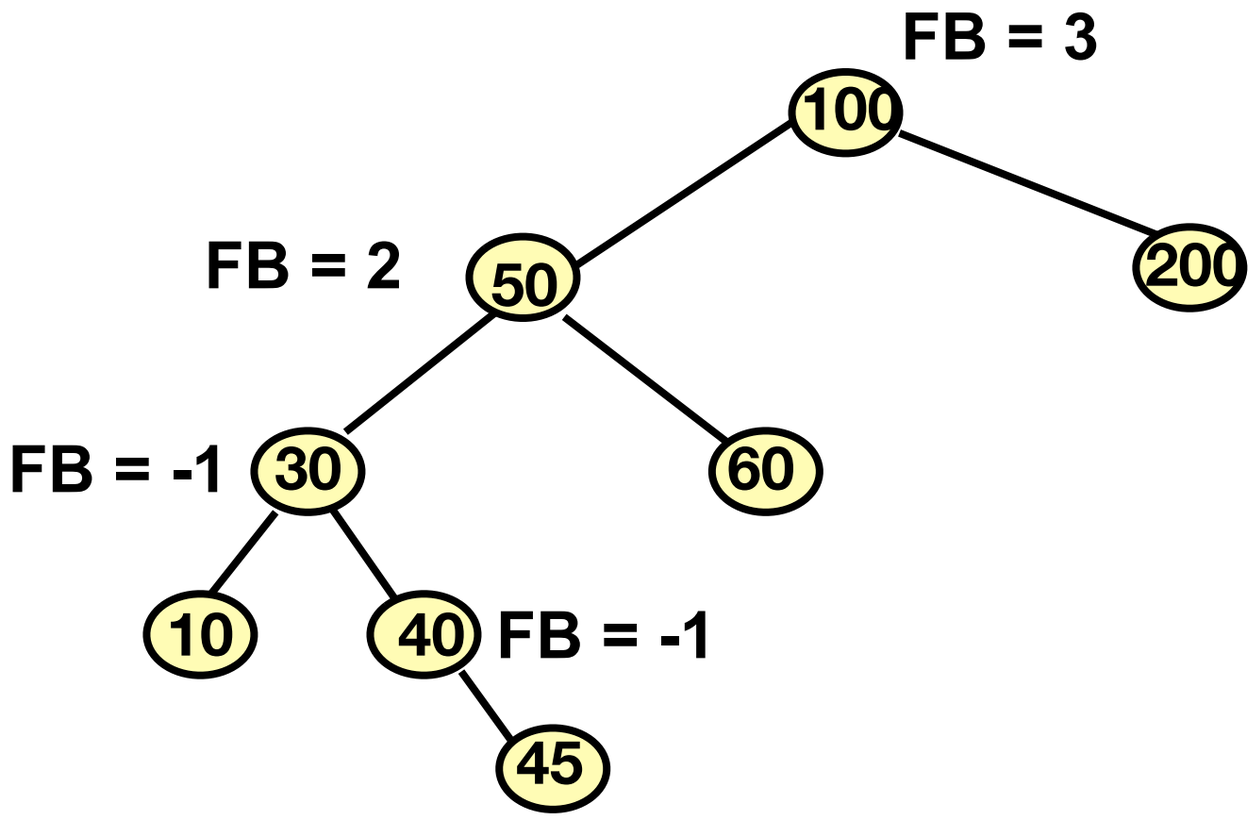
Esercizio 0

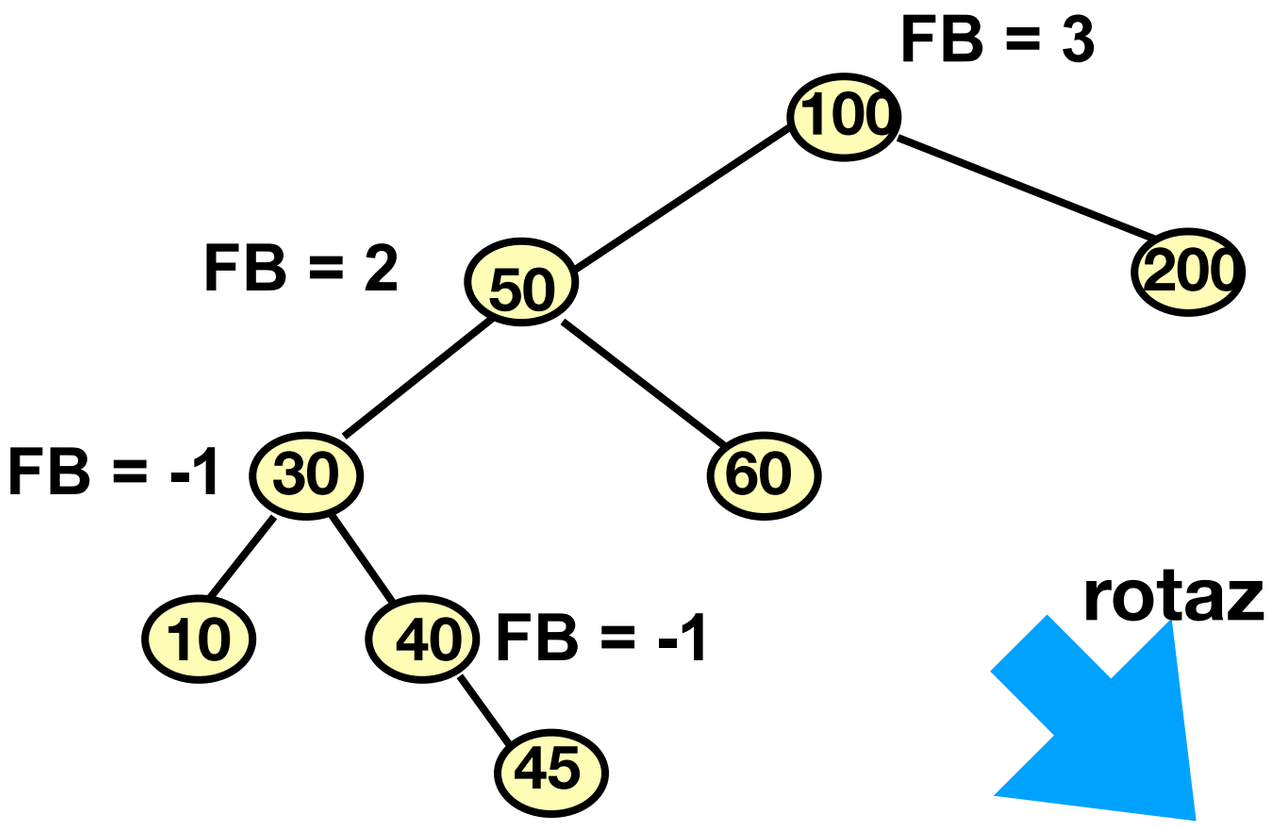
Dato l'albero sottostante T, si aggiungano valori delle chiavi tali da renderlo un ABR. Si calcolino i fattori di bilanciamento. Si individuino le rotazioni che lo trasformano in un AVL, dicendo per ogni rotazione su quale nodo si fa perno e di che rotazione si tratta, infine si disegni l'albero ottenuto con i fattori di bilanciamento.



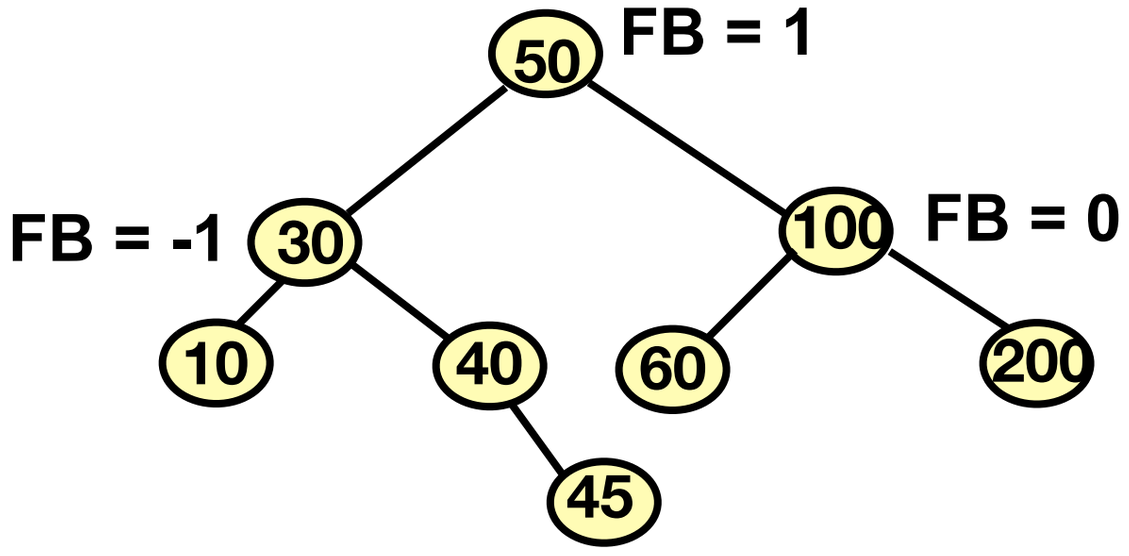
Esercizio 0



Esercizio 0 - sol



rotazione a destra sulla radice

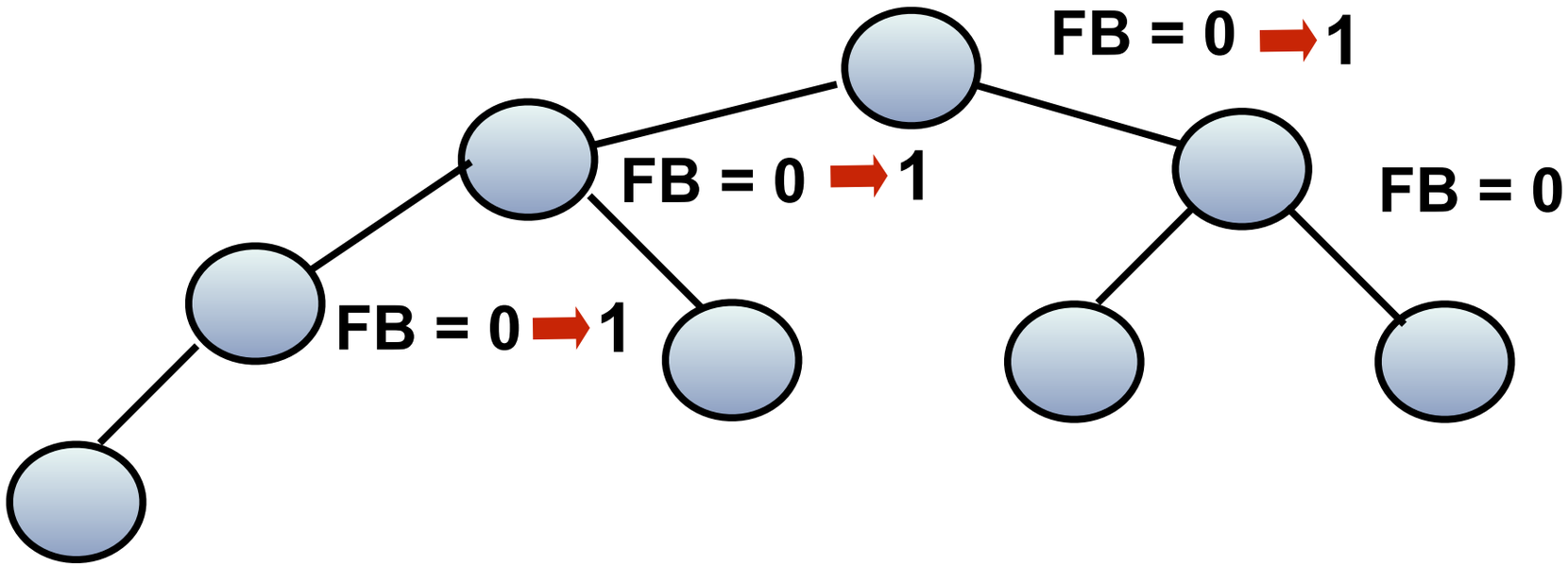


Esercizio 1

Quale classe di AVL può comportare a seguito di inserimento qualunque un aggiornamento dei FB dal padre del nodo inserito alla radice, senza rotazioni?

Esercizio 1- sol

La classe degli alberi completi, cioè quelli in cui ogni nodo ha due o zero figli e le foglie sullo stesso livello. Aggiungendo un nodo si sbilancia il cammino foglia-radice, in modo tollerato nella definizione di AVL.

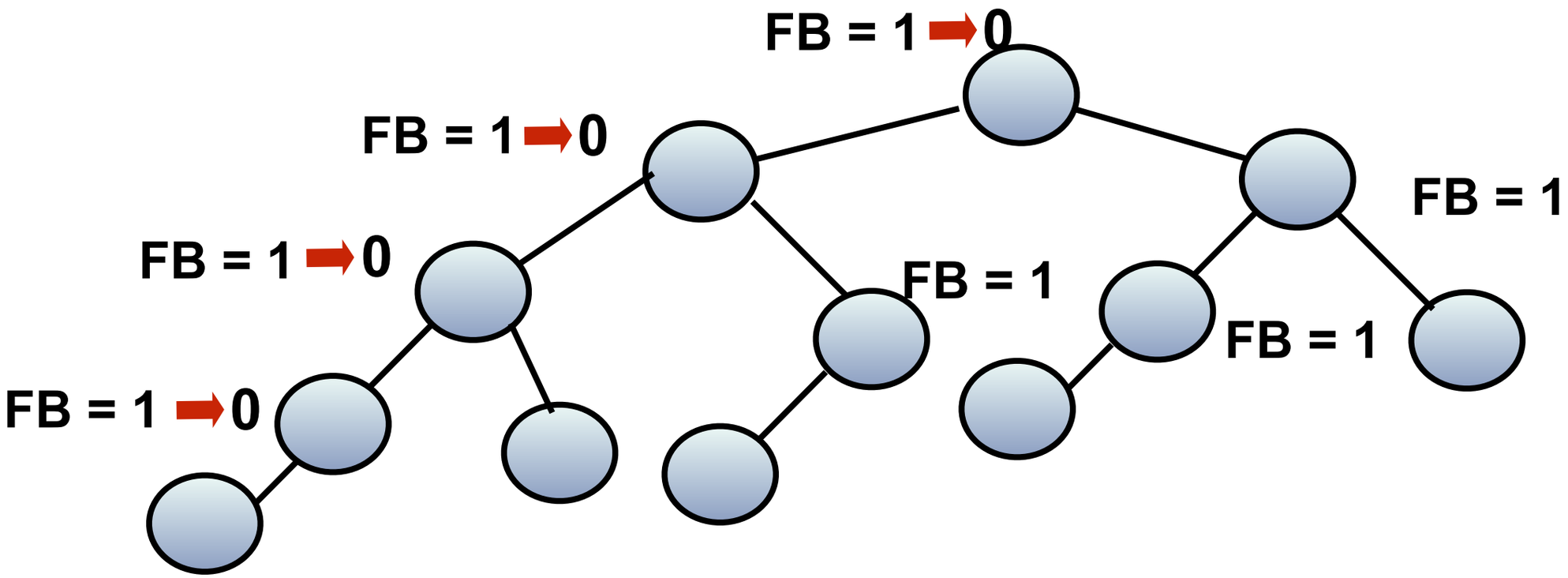


Esercizio 2

Quale classe di AVL può comportare a seguito di cancellazione un aggiornamento dei FB dal padre del nodo cancellato alla radice, senza rotazioni?
Si deve indicare quale nodo cancellato produce l'effetto indicato.

Esercizio 2 - sol

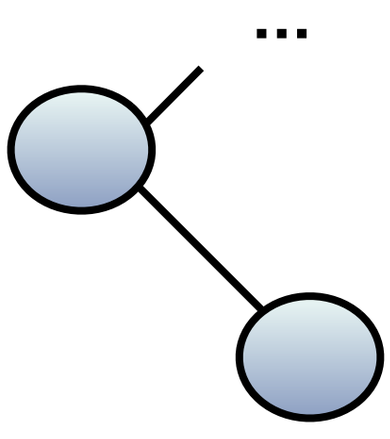
Gli alberi di Fibonacci, infatti se cancelliamo la foglia di profondità massima dovremo aggiornare a 0 tutti i FB lungo il cammino fino alla radice.



Esercizio 3

E’ vero che in un AVL il minimo si trova in una foglia o nel penultimo livello? Si motivi la risposta.

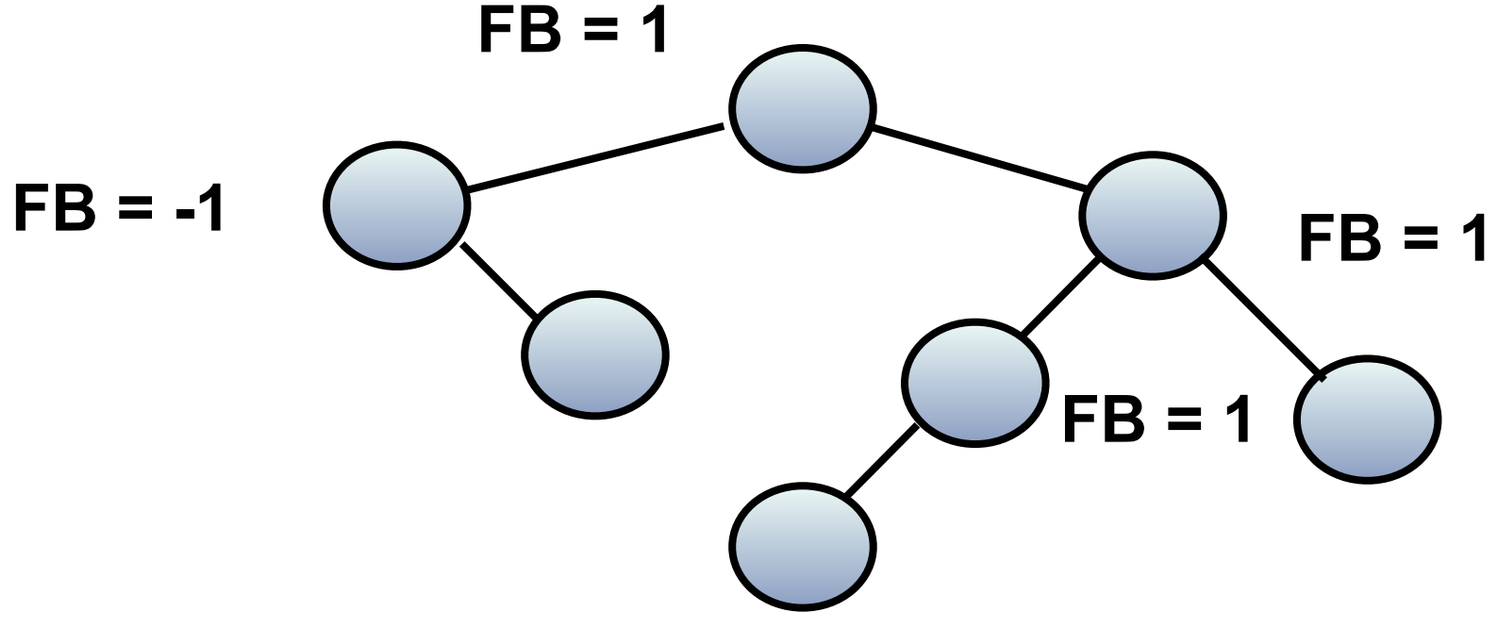
<u>Sembra</u> vero, e si potrebbe motivare così:
il minimo per definizione non ha il figlio sinistro, ma può avere un figlio destro. Quest’ultimo deve essere radice di un sotto albero di altezza tale da differire da quella del sotto albero sinistro, -1, al più di 1 in valore assoluto. Quindi o l’altezza del sotto albero destro è -1, oppure 0.
Ma manca il resto dell’albero!



Esercizio 3

E’ vero che in un AVL il minimo si trova in una foglia o nel penultimo livello? Si motivi la risposta.

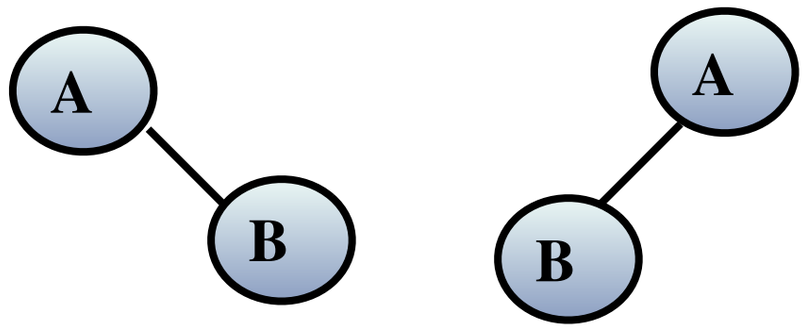
Si può riflettere sul fatto che il sotto albero destro potrebbe essere più alto del sinistro e quindi quel livello del minimo può non essere il penultimo.
Qui c’è un contro esempio:



Esercizio visite

Abbiamo visto che da una visita in inorder e una in preorder è possibile costruire univocamente un albero binario. Si può fare lo stesso partendo da una visita inorder e una in postorder.
Ma non è possibile a partire una preorder e una postorder. Si dia un esempio di due sequenze rappresentanti una visita in preorder e una in postorder da cui si possano costruire due alberi.

Esercizio visite - sol



In entrambi i casi preorder AB e una postorder BA

Se si dispone di un ABR è possibile ricostruire un albero univocamente a partire solo da una visita in postorder o in preorder?
Sì, banalmente si fa una visita inorder e si procede con l'algoritmo noto.

Esercizio 4

Si costruisca un algoritmo ricorsivo che, dato un AVL T, e un valore positivo k, trova il nodo più a destra di profondità k e con fattore di bilanciamento 1. La complessità è O(n).

Esercizio 4

E' una variante del calcolo dell'altezza, cui si devono aggiungere i controlli indicati nel testo. Inoltre si esplora prima il sotto albero destro rispetto al sinistro per soddisfare il requisito della individuazione del nodo più a destra con quella profondità e FB. Si visita l'albero in postorder quindi il tempo è in O(n).

```
algoritmo RicNodoAVL(T,k)
input: un puntatore alla radice di un albero binario T con chiavi
intere e un intero k
prec: T è un AVL e k è non negativo
output: il puntatore al nodo più a destra di profondità k e fb 1
if T = NULL return NULL
if k = 0
      if T.fb = 1 return T else return NULL

Q1= RicNodoAVL(T.right,k-1)
Q2 = RicNodoAVL(T.left,k-1)
if Q1≠ NULL then return Q1 else Q2
```

Esercizio 5

Si scriva un algoritmo per l'individuazione del mediano in un ABR, in O(n).
Il mediano in un insieme ordinato di n elementi è l'elemento che ha n/2 elementi minori nell'insieme.
Quindi se n=2m+1 o n = 2m il mediano è l'elemento che ha m elementi minori.

Si cerchi una soluzione che non preveda il calcolo del numero dei nodi, come primo passo.

Esercizio 5 - sol

Qui conviene eseguire due visite inorder, una da sinistra che parte dal minimo e una rovesciata che parte dal massimo. La visita avviene calcolando il successivo di ogni nodo nella visita da sinistra e il precedente per quella da destra. Detto x il nodo visitato dal minimo e z quello dal massimo, la chiave di x è sempre minore di quella di z, tranne quando le due visite si "sovrappongono".
Se il numero degli elementi è pari il mediano è alla stessa "distanza" dal minimo e dal massimo, le due visite si troveranno sullo stesso nodo e quindi le due chiavi sono uguali.
Se il numero è dispari il mediano è a una distanza dal minimo di uno maggiore rispetto a quella dal massimo, durante la visita da sinistra si visiterà un nodo già visitato da destra, e quindi di chiave maggiore di quello che verrà visitato da destra, che a sua volta è un nodo già visitato da sinistra.

Esercizio 5

```
algoritmo Mediano(T)
input: un puntatore alla radice di un albero binario T con chiavi intere
prec: T è un ABR
output: il puntatore al nodo mediano
if T = NULL return NULL
z = MAXIMUM(T) /dà in output il puntatore al nodo di chiave massima
x = MINIMUM(T)/dà in output il puntatore al nodo di chiave minima
while x.key < z.key do
        x = SUCCESSOR(x)
        z = PREDECESSOR(z)
return x
```

Esercizio 6

Si imposti e si risolva la relazione di ricorrenza che esprime la complessità asintotica nel caso peggiore dell'algoritmo seguente.

```
Rip (A,lo,hi)
n = lo - hi
if (n ≤ 1) return 0
m = (lo+hi)/2
count = Rip(A,lo, m) + Rip(A,m+1, hi);
for i = lo to  m -1 do
     c1 = 0
   for j = lo  to m-1 do
          if A[i] == A[j]  then c1 = c1+1
     c2 = 0
   for j = m + 1 to hi -1  do
          if A[i] == A[j] then c2 = c2+1
                if (c1 == 1 && c2 == 1) then count=count+1
return count;
```

Esercizio 6 - sol

```
Rip (A,lo,hi)
n = lo - hi
if (n ≤ 1) return 0
m = (lo+hi)/2
count = Rip(A,m+1, hi)+ Rip(A,m+1, hi)
for i = lo to  m -1 do
      c1 = 0
   for j = lo  to m-1 do
            if A[i] == A[j]  then c1 = c1+1
      c2 = 0
   for j = m + 1 to hi -1  do
           if A[i] == A[j] then c2 = c2+1
                  if (c1 == 1 && c2 == 1) then count=count+1
return count;
```

La relazione di ricorrenza che ne esprime il tempo di esecuzione è

$T(n) = 2T(n/2) + \Theta(n^2/2)$

Esercizio 6 - sol 2

$T(n) = 2T(n/2) + \Theta(n^2/2)$, per risolverla innanzi tutto esplicitiamo le costanti e semplifichiamo:

$T(n) = d$ se $n \leq 1$

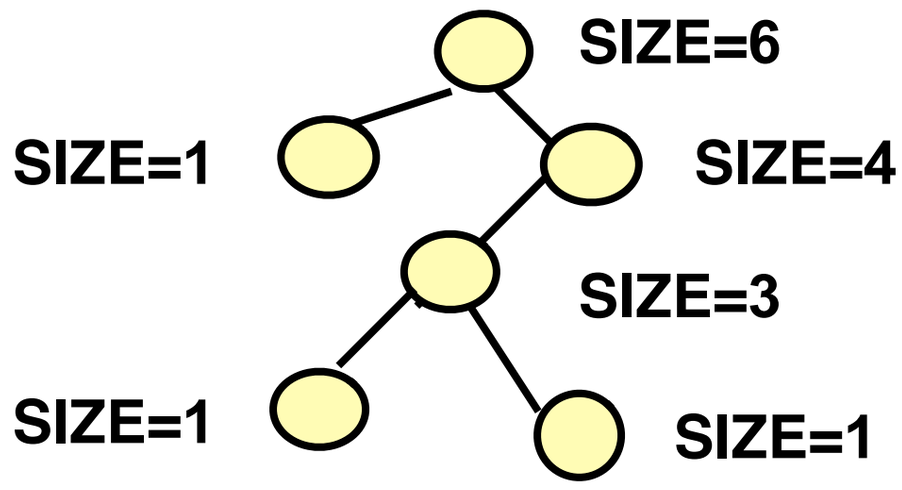
$T(n) = 2T(n/2) + cn^2$

Ricerca di un nodo di rango r

Dato un ABR T non nullo e un valore r si scriva un algoritmo che fornisce in output il riferimento (puntatore) al nodo di rango r in T, se presente, NIL altrimenti.

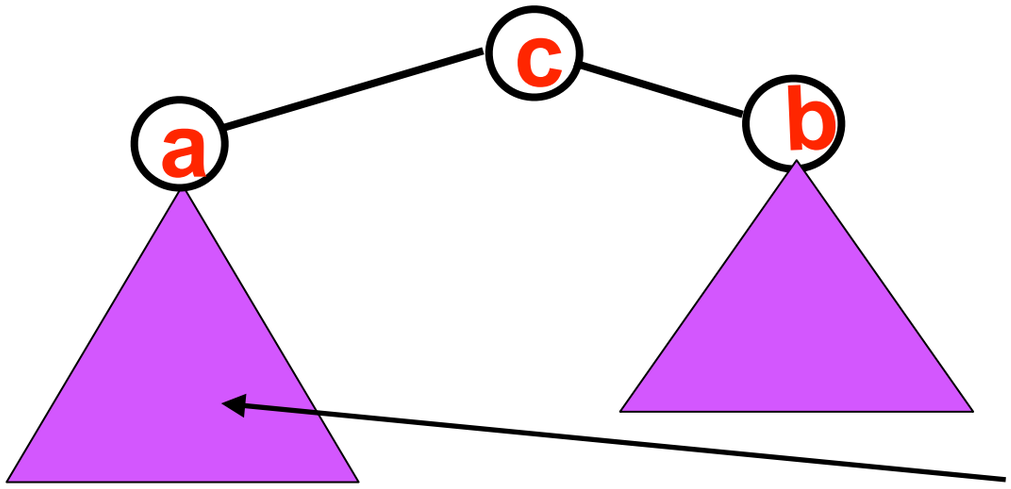
Il rango qui è il numero degli elementi minori, quindi se per esempio il numero dato in input è il numero dei nodi dell'albero meno 1 si dovrà dare in output il nodo di chiave massima.

Si supponga l'ABR sia una struttura dati aumentata, aggiungendo un campo size in ogni nodo che contiene il numero dei nodi nel sotto albero radicato in quel nodo.



Ricerca di un nodo di rango r

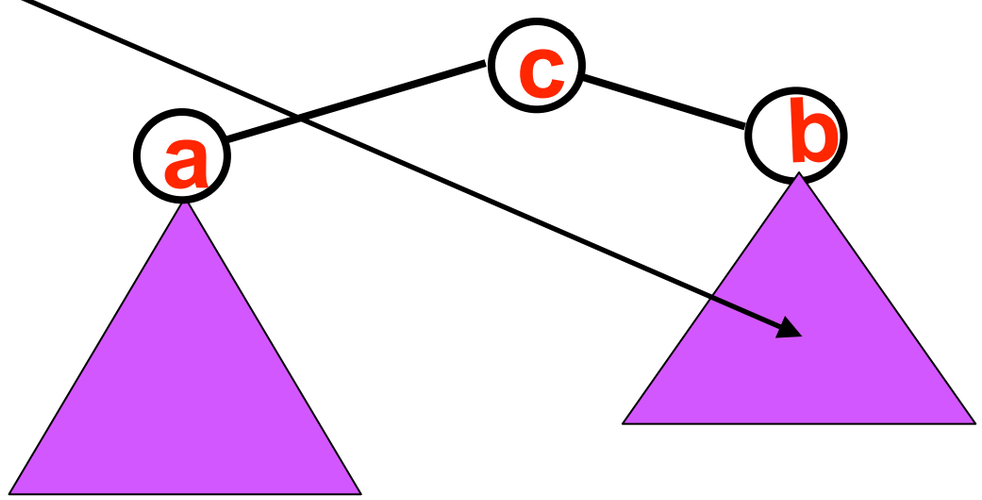
- Se T ha un solo nodo e r=0 allora la risposta è T



Se T ha due sotto alberi e il numero dei nodi nel sotto albero sinistro è uguale al rango r dato in input allora il nodo è T.

Se il numero dei nodi nel sotto albero sinistro è maggiore del rango r dato in input, il nodo deve trovarsi nel sotto albero sinistro e il suo rango nel sotto albero coincide con il rango in T.

Altrimenti, il nodo deve trovarsi a destra, se c'è, ma per trovarlo si deve cercare nel sotto albero destro un nodo il cui rango è r meno il numero numero dei nodi nel sotto albero sinistro più 1 per la radice.



Select

```
Select(T,r)
    Input: un albero binario T e valore r
    prec: 0≤r<nNodi(T)
    output: il riferimento al nodo di rango r
    if T == NIL then return  NIL
    n = T.left.size
    if n == r then return    T
    if n > r then return Select1(T.left,r)
             else return Select1(T.right,r-n-1)
```

Complessità O(lg n).

Esercizio 7

Sia T un albero AVL contenente n valori. Progettare un algoritmo che calcoli il numero minimo di nodi con fattore di bilanciamento +1 su un cammino radice-foglia. Si analizzino correttezza e complessità dell'algoritmo proposto.

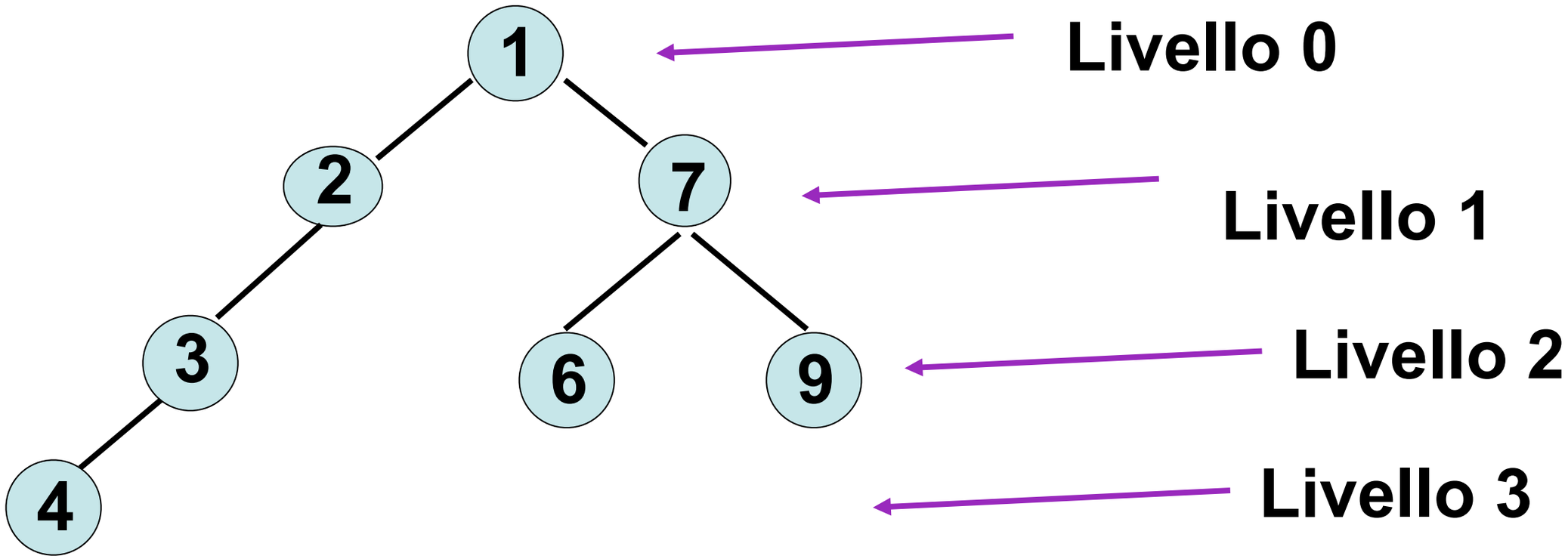
Esercizio 7 - sol

E' una variante del calcolo dell'altezza, cui si devono aggiungere i controlli indicati nel testo.

```
algoritmo MinFB1(T)
input: un puntatore alla radice di un albero binario T
prec: T è un AVL
output: il minimo numero di nodi con fb=1 in un cammino radice-foglia
if T = NULL o è una foglia return 0
n1 = RicNodoAVL(T.left)

n2= RicNodoAVL(T.right)
if T.fb = 1 then return min(n1,n2)+1 else return min(n1,n2)
```

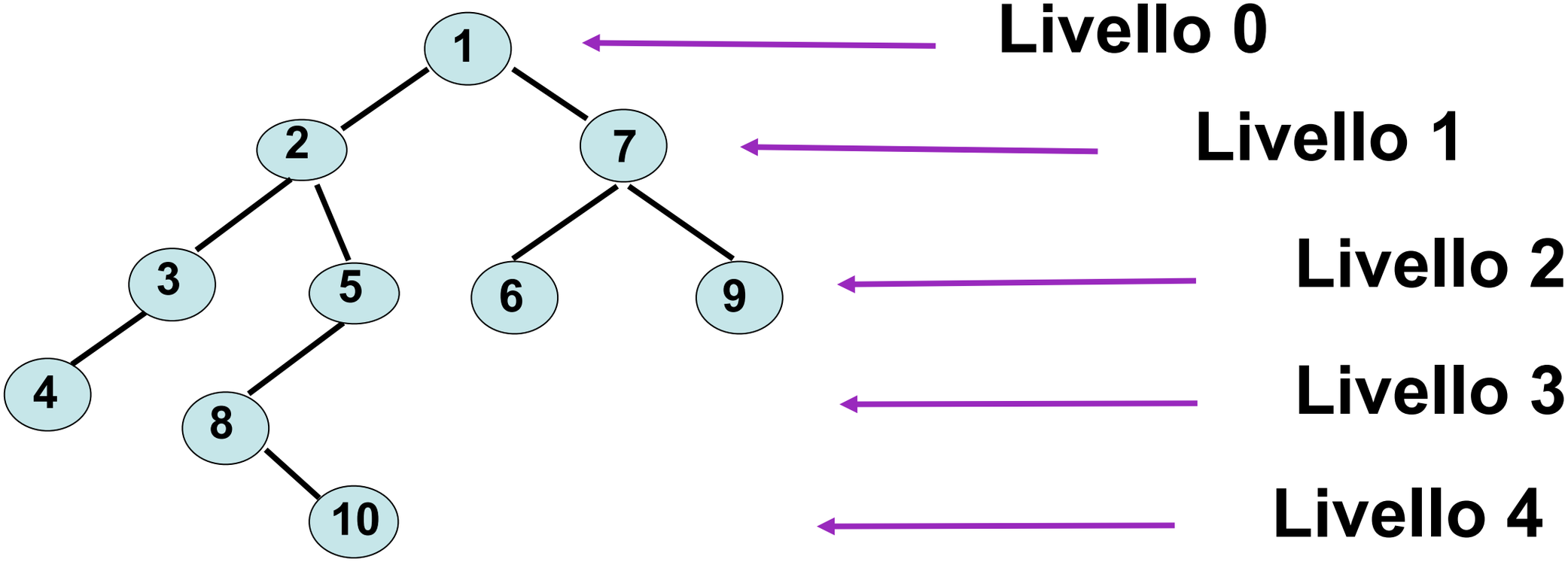
APPLICAZIONE DELLA CODA: VISITA PER LIVELLI DI UN ALBERO BINARIO



Si vuole visitare i nodi su ogni livello da sinistra a destra e in ordine discendente di livello.

Risultato visita: 1 27 369 4

VISITA PER LIVELLI DI UN ALBERO BINARIO



Risultato visita: 1 2 7 3 5 6 9 4 8 10

Osserviamo che se per esempio abbiamo visitato 6 il prossimo da visitare è suo fratello ma il successivo è il figlio del primo nodo sul suo livello.

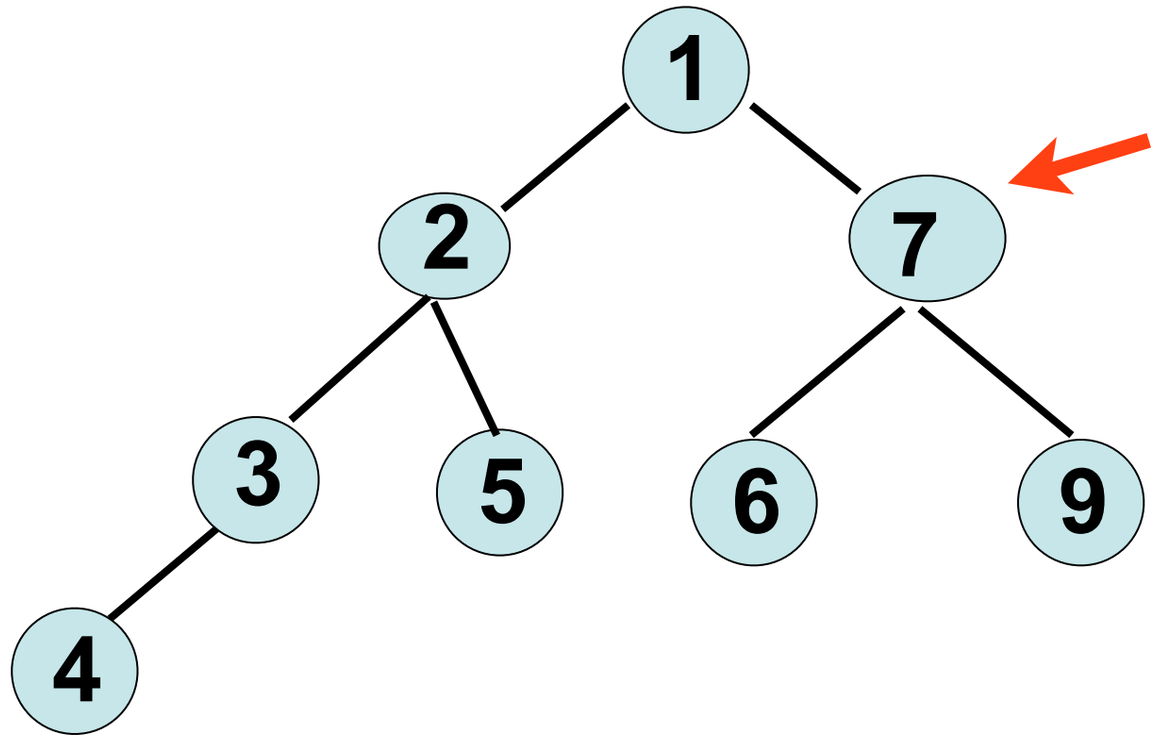
Allora se man mano che si visita un livello si tiene memoria degli eventuali figli, cioè dei nodi del livello successivo, il primo memorizzato è il primo da estrarre.

Quindi serve una coda!

Visita per livelli

Alg VisitaPerLivelli(T)
//T è un albero binario
postc: stampa la sequenza dei nodi di T per livelli da sinistra a destra.

In un generico passo, potremmo trovarci alla fine di un livello e nella necessità di stampare i nodi del livello successivo, partendo da sinistra.
Quindi bisogna che quei nodi siano stati accodati nell'ordine da sinistra verso destra, e l'unica possibilità di farlo è nel momento in cui si è stampato il loro padre. Inoltre un nodo stampato non serve più e quindi deve essere tolto dalla coda.



Nodi nella coda prima della stampa di 7

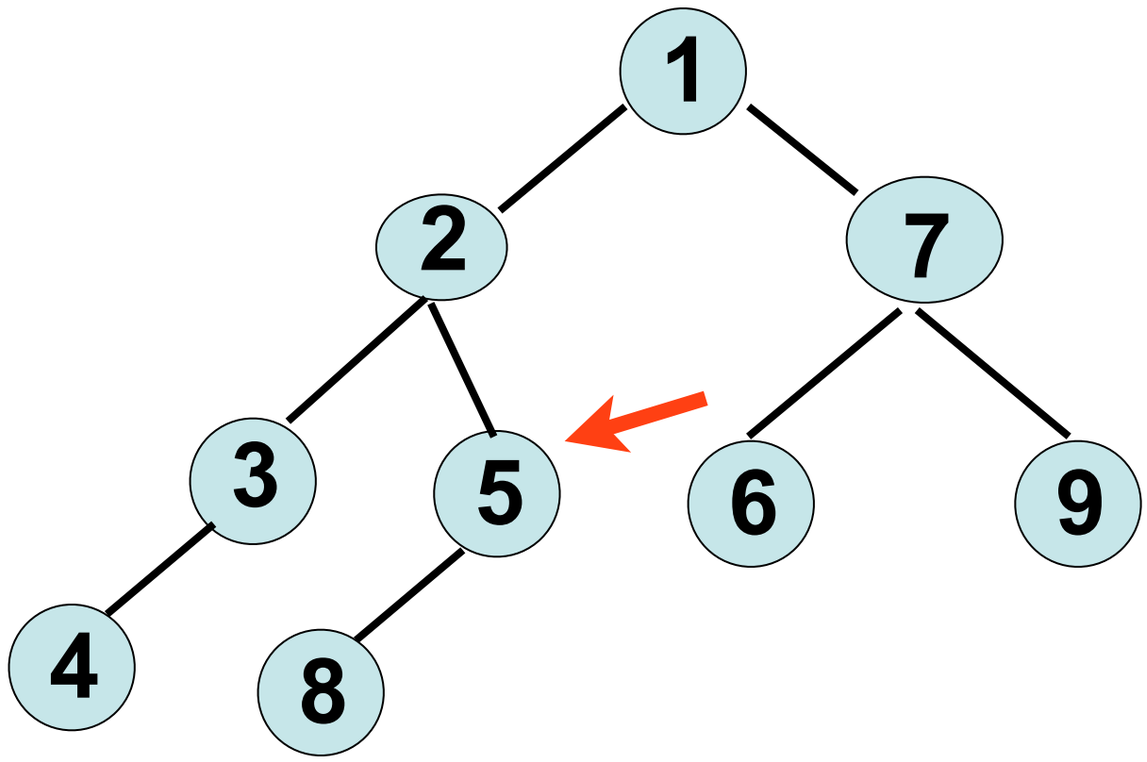
Q= | 7 | 3 | 5 |

Nodi nella coda dopo la stampa di 7

Q= | 3 | 5 | 6 | 9 |

Visita per livelli

Una situazione del tutto analoga si verifica se consideriamo un passo intermedio in cui ci troviamo su un generico nodo di un livello, infatti quel nodo va stampato e tolto dalla coda, che deve contenere i restanti nodi di quel livello, ancora da stampare, e anche i figli dei nodi già stampati.
Inotre bisogna accodare i figli del nodo tolto e stampato.



Nodi nella coda prima della stampa di 5

Q= | 5 | 6 | 9 | 4 |

Nodi nella coda dopo la stampa di 5

Q= | 6 | 9 | 4 | 8 |

Quindi la coda contiene i tutti i nodi di un livello oppure i nodi dal k-simo fino all'ultimo di un livello e tutti i nodi del livello successivo figli dei nodi alla sinistra del k-simo. In ogni caso i nodi sono nell'ordine da sinistra verso destra.

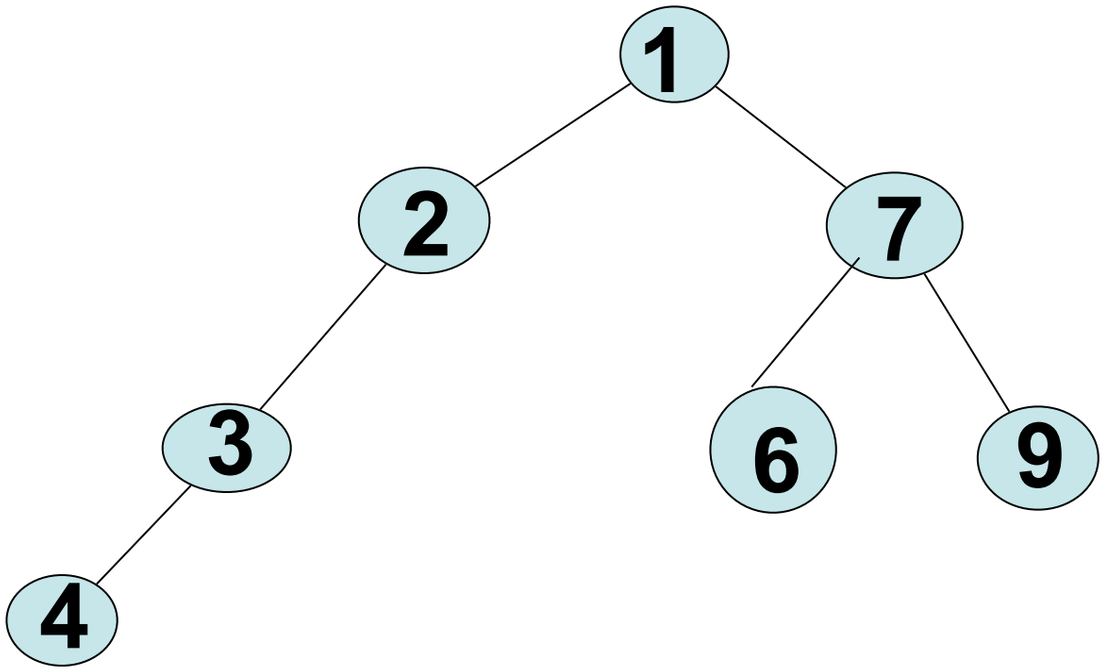
Visita per livelli: l'algoritmo

La coda contiene i tutti i nodi di un livello oppure i nodi dal k-simo fino all'ultimo di un livello e tutti i nodi del livello successivo figli dei nodi alla sinistra del k-simo. In ogni caso i nodi sono nell'ordine da sinistra verso destra.

```
Alg VisitaPerLivelli(T)
//T è un albero binario
postc: stampa la sequenza dei nodi di T
per livelli da sinistra a destra.
Q è una coda inizialmente vuota
accoda in Q il nodo radice, se l'albero è non vuoto
while la coda non è vuota do
        y = dequeue(Q)
        stampa y
       if il figlio sinistro,x, di y è presente then Enqueue(Q,x)
       if se il figlio destro,x, di y è presente then Enqueue(Q,x)
```

N.B. L'algoritmo è formulato in modo da prescindere dalla rappresentazione in memoria di T e di Q.

Visita per livelli: esempio di esecuzione



Contenuto coda all'inizio:

1

Nodo estratto e stampato: 1

2	7

Nodo estratto e stampato: 2

7	3

Nodo estratto e stampato: 7

3	6	9

Nodo estratto e stampato: 3

6	9	4

Nodo estratto e stampato: 6

9	4

Nodo estratto e stampato: 9

4

Nodo estratto e stampato: 4

Visita per livelli: analisi

```
Alg VisitaPerLivelli(T)
//T è un albero binario
postc: stampa la sequenza dei nodi di T
per livelli da sinistra a destra.
Q è una coda inizialmente vuota
accoda in Q il nodo radice, se l'albero è non vuoto
while la coda non è vuota do
        y = dequeue(Q)
        stampa y
if il figlio sinistro,x, è presente then Enqueue(Q,x)
if se il figlio destro,x,  è presente then Enqueue(Q,x)
```

Ogni nodo viene inserito ed estratto dalla coda una sola volta, quindi $T(n) = \Theta(n)$

analisi complessità

Sia dato un albero binario completo, T, cioè un albero in cui ogni nodo ha 2 o 0 figli e tutte le foglie sono sullo stesso livello, con n nodi. Impostare e risolvere la relazione di ricorrenza che esprime il tempo di esecuzione della seguente funzione, in funzione di n.

```
Funzione (T)
if (!T.left and !T.right) return T
T1=T;
while (T1.left) do T1=T1.left;
n1 = T1.key;
T1=T;
while (T1.right) do T1=T1.right;
n2=T1.key;
if (n1 < n2) return Funzione(T.left) else return Funzione(T.right);
```

analisi complessità

```
Funzione (T)
if (!T.left and !T.right) return T
T1=T;
while (T1.left) do T1=T1.left;
n1 = T1.key;
T1=T;
while (T1.right) do T1=T1.right;
n2=T1.key;
if (n1 < n2) return Funzione(T.left) else return Funzione(T.right);
```

Poiché l'albero è completo i due sotto alberi hanno circa n/2 elementi e l'altezza è lg n, quindi la relazione è
$T(n) = T(n/2) + \Theta(\lg n)$