Rango di un nodo in un ABR

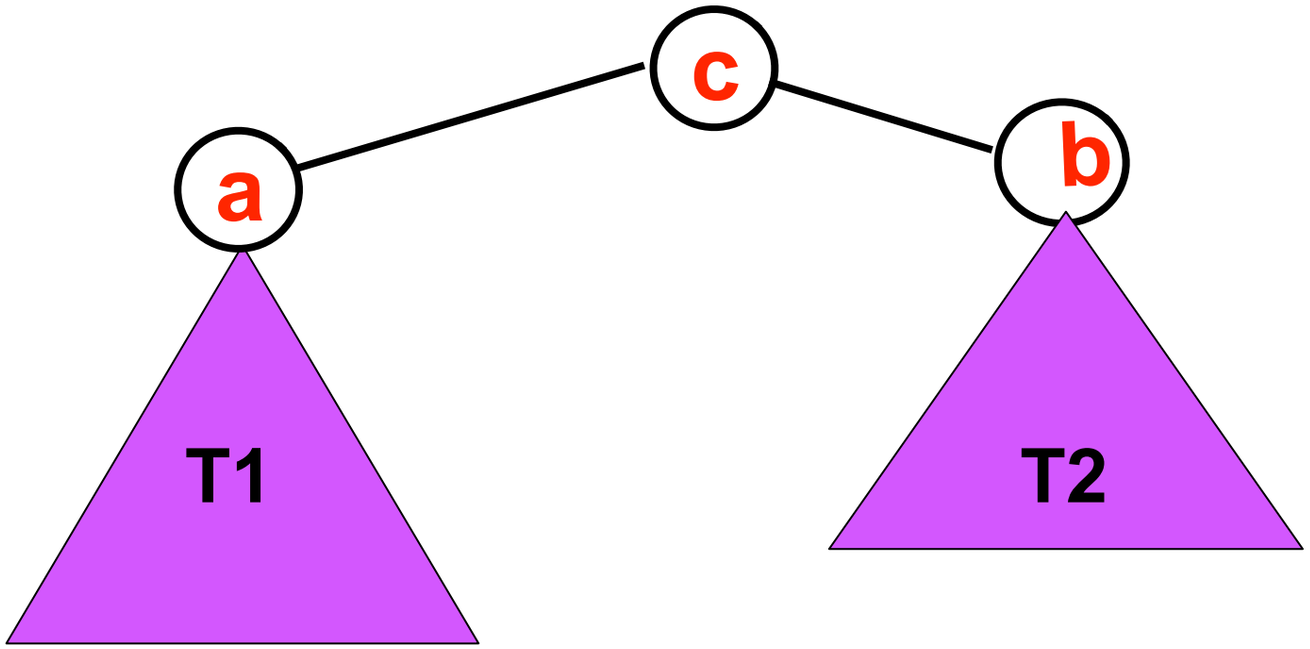
Dato un ABR T non nullo e un puntatore a un suo nodo x si scriva un algoritmo che fornisce in output il rango della chiave di x in T, cioè il numero degli elementi di chiave minore a quella di x in T.

Si supponga che T sia rappresentato in memoria con anche il puntatore al padre.

Poiché abbiamo l'ipotesi che le chiavi sono tutti diverse, ci limitiamo a calcolare il numero dei nodi con chiave minore.

Rango sol ricorsiva

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



Se T ha due sotto alberi e la chiave di x è uguale a quella della radice, allora il rango è il numero dei nodi nel sotto albero sinistro.

Se la chiave di x è minore di quella della radice, allora x è nel sotto albero sinistro e posso chiamare la funzione su quel sotto albero e il risultato vale per T.

Se invece la chiave di x è maggiore di quella della radice bisogna sommare al numero di nodi minori di x nel sotto albero destro, ottenuto con una chiamata della funzione su quel sotto albero, il numero dei nodi nel sotto albero sinistro più la radice.

Rango sol ricorsiva, psudocodice

```
Rango(T,x)
input: il puntatore alla radice T di un albero binario e a un suo nodo x
output: il numero dei nodi in T di chiave minore di quella di x
        if T == NIL then return  0
        if T.key == x.key then return   nNodi(T.left)
        if x.key < T.key then return  Rango(T.left,x)
       else  return   nNodi(T.left) + 1 + Rango(T.right,x)
        %x.key > T.key
```

```
nNodi(T)
input: un albero binario T
output: il numero dei nodi di T
if T==NIL then return 0
return nNodi(T.left) + nNodi(T.right) +1
Tempo di esecuzione in θ(n), se n è il numero dei nodi di T
```

Rango sol ricorsiva, analisi

```
Rango(T,x)
        if T == NIL then return  0
        if T.key == x.key then return   nNodi(T.left)
        if x.key < T.key then return  Rango(T.left,x)
         else  return   nNodi(T.left) + 1 + Rango(T.right,x)
          %x.key > T.key
```

Tempo di esecuzione nel caso peggiore O(n).

Rango sol 2

Dato un ABR T non nullo e un puntatore a un suo nodo x si può calcolare il numero dei precedenti di x, applicando la funzione predecessore al nodo x e incrementando una variabile ogni volta che il nodo ottenuto non è nullo.

Soluzione 2 rango

Rango1(T,x)

Input: un albero binario T e un suo nodo

prec:T è un ABR non vuoto e x ≠ NIL

postc: restituisce il rango di

```
 z = PREDECESSOR(x)
 k = 0
 while z ≠ NIL do
 z è il precedente di y e k
contiene il numero dei nodi di T
 con chiave a tale
che  a ≤ x.key, tra y e x.
        y = z
         k = k+1
         z = PREDECESSOR(y)
return   k
```

Tempo di esecuzione in O(k + h), dove k è il rango del nodo x e h è l'altezza dell'albero. Se x è il minimo si ha O(h). Il risultato si ottiene con un'analisi analoga a quella della visita inorder in cui si usa la funzione successor.

All'uscita dal ciclo z è NIL perchè si è calcolato il precedente del minimo y, e k contiene il numero dei nodi di T con chiave a tale che a ≤ x.key

Rango sol 3

Dato un ABR T non nullo e un puntatore a un suo nodo x si può calcolare il numero dei successivi del minimo minori di x, applicando la funzione che individua il successivo a partire dal minimo, incrementando una variabile inizializzata a 0 fino a che non si arriva al nodo x.

Rango 3 ABR

```
Rango3(T,x)
    Input: un albero binario T e un suo nodo
    prec: x ≠ NIL
    postc: restituisce il rango di della chiave di x
    k = 0
    if T == NIL return   0
        z = MINIMUM(T)
      while x.key > z.key do
  z è l' elemento di rango k in T
            z = SUCCESSOR(z)
            k = k+1
All'uscita dal ciclo vale y.key = x.key perchè x è un nodo
dell'albero e quindi k è il rango di x.
      return   k
```

Complessità O(k + h), dove k è il rango del nodo x e h è l'altezza dell'albero.

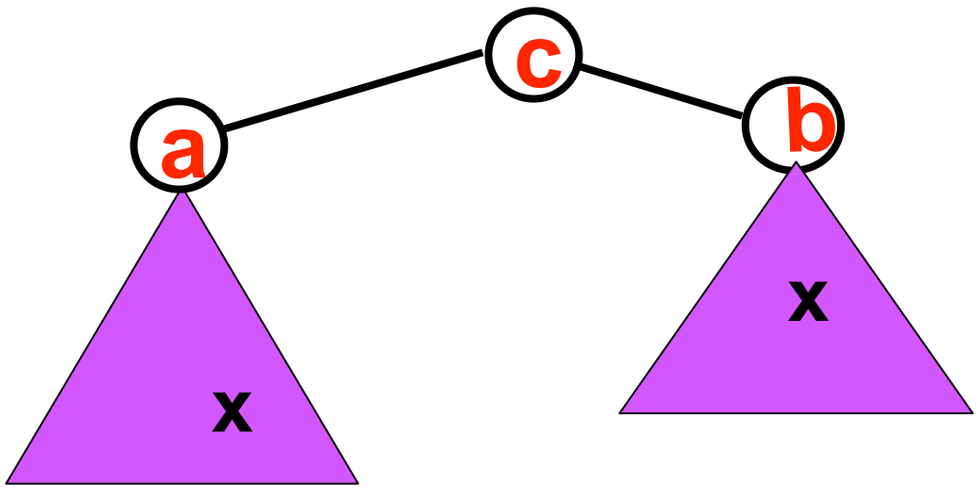
Ricerca di un nodo di rango r

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.

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 x deve trovarsi nel sotto albero sinistro e il suo rango nel sotto albero coincide con il rango in T.

Altrimenti, il nodo x 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

```
Select1(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 = nNodi(T.left)
    if n == r then return    T
    if n > r then return Select1(T.left,r)
            else return Select1(T.right,r-n-1)
```

Complessità $O(n^2)$, nel caso peggiore, perché se l'albero è un albero degenere a sinistra si calcola n-1 volte il numero dei nodi del sottoalbero sinistro.

Ma il numero dei nodi può essere calcolato risalendo nell'albero e confrontando di volta in volta il numero ottenuto con il rango per individuare il nodo cercato, ma la soluzione non è così lineare.

Ricerca di un nodo di rango r, versione iterativa

Si può sfruttare la visita inorder iterativa, partendo dal minimo e visitando i successivi fino a visitare l' r-esimo nodo o nil se r è maggiore o uguale al numero dei nodi.

Tempo di esecuzione O(r + h).

Ricerca di un nodo di rango r, versione iterativa

SelectIt(T,r)

Input: un albero binario T e un intero r

prec: T è un ABR e r ≥ 0

postc: restituisce il nodo la cui chiave ha rango r in T, se presente, nil altrimenti.

```
if T == NIL return   T
 k = 0
z = MINIMUM(T)
while k < r and z ≠ nil do
z è il nodo di rango k in T
     z = SUCCESSOR(z)
     k = k+1
All'uscita dal ciclo se vale k = r e z ≠ nil allora z  ha  rango r.
Se z = nil allora r è maggiore o uguale al numero dei nodi di
T e la risposta è nil
return  z
```

Esercizio AVL 1

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 AVL 1

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

Y = RicNodoAVL(T.right,k-1)
if Y = NULL then return RicNodoAVL(T.left,k-1)
return Y
```

Esercizio mediano in ABR

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 mediano in ABR

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 dispari 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 pari 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 mediano in ABR

```
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 AVL 2

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 AVL 2 - 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)
```