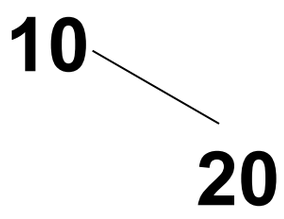
Esercizi su ABR

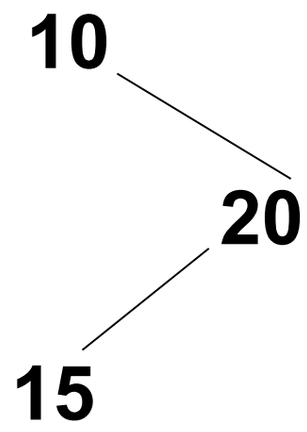
Costruzione di un ABR bilanciato nel numero dei nodi, a partire da un array ordinato.
Visita inorder iterativa di un ABR.
Calcolo del rango di una chiave in un ABR.
Selezione di un nodo in un ABR di un dato rango.
Stampa delle chiavi in un intervallo.

Costruzione ABR bilanciato

Dato un vettore ordinato A di interi si definisca un algoritmo che costruisce un ABR (BST) bilanciato nel numero dei nodi in O(n) passi, che ha gli elementi dell'array come chiavi.

Un ABR è bilanciato nel numero dei nodi se i due sottoalberi di ogni nodo hanno un numero di elementi che differisce in valore assoluto al più di 1.

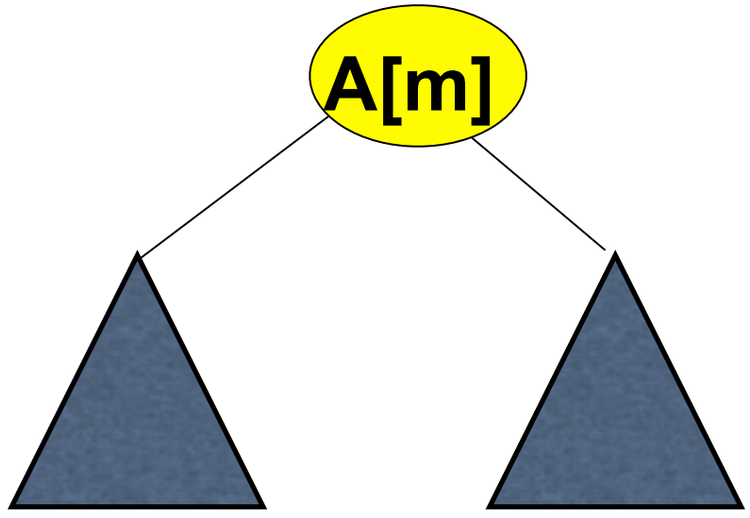




Costruzione ABR sol

Si può pensare di prendere come radice l'elemento intermedio A[m] con m = n/2 e allora se n = 2p, si inseriscono p-1 elementi sottoalbero sinistro e p nel sottoalbero destro, se invece n=2p+1 si inseriscono p elementi sia nel sottoalbero sinistro che nel destro.
Quindi i due sottoalberi in ogni caso avranno un numero di elementi che differisce in valore assoluto al più di 1.

Il sottalbero sinistro conterrà gli elementi da A[1] a A[m-1], che sono più piccoli di A[m]



Il sottalbero destro conterrà gli elementi da A[m+1] a A[n] che sono più grandi di A[m]

Soluzione

ABRBil(A, i, j)

Input: un array di interi A e due interi i e j.

precond: A è un vettore ordinato e 0≤i≤j≤A.length

output: un ABR bilanciato nel numero dei nodi costruito sugli elementi di A

In ogni chiamata la porzione di array su cui si lavora è quella compresa tra A[i] e A[j-1].

```
if (i ≤ j-1)  then
    m =  (i+j)/2;
    T = new(A[m])
    new crea un nuovo nodo  puntato da T con chiave A[m] e
    puntatori ai figli e al padre nil
    T.left = ABRBil(A,i,m)
    if (T.left)  then T.left.p = T
    T.right = ABRBil(A,m+1,j)
    if (T.right)  then T.right.p = T
    return T
else return NIL
```

$T(n) = 2T(n/2)+\Theta(1)$

La chiamata iniziale è ABRBil(A, 0, n).

Visita Inorder ABR

Un modo alternativo per eseguire una visita inorder di un albero binario di ricerca (non di un albero qualunque!).

VisitaItInorderABR(u):

1. determinare il minimo, x, di u, visitarlo e

2. calcolare il successivo di x, y, visitarlo e

3. ripetere il punto 2 dopo aver copiato y in x e fino a che il successivo è nullo.

Il limite inferiore è Ω(n), dove n è il numero dei nodi, perchè tutti i nodi sono visitati.

Dimostreremo che VisitaItInorderABR ha complessità O(n) e non O(nh) come si potrebbe pensare tenendo condo che calcolare il successivo di un nodo è in O(h), dove h è l'altezza dell'albero su cui si opera.

Il successivo

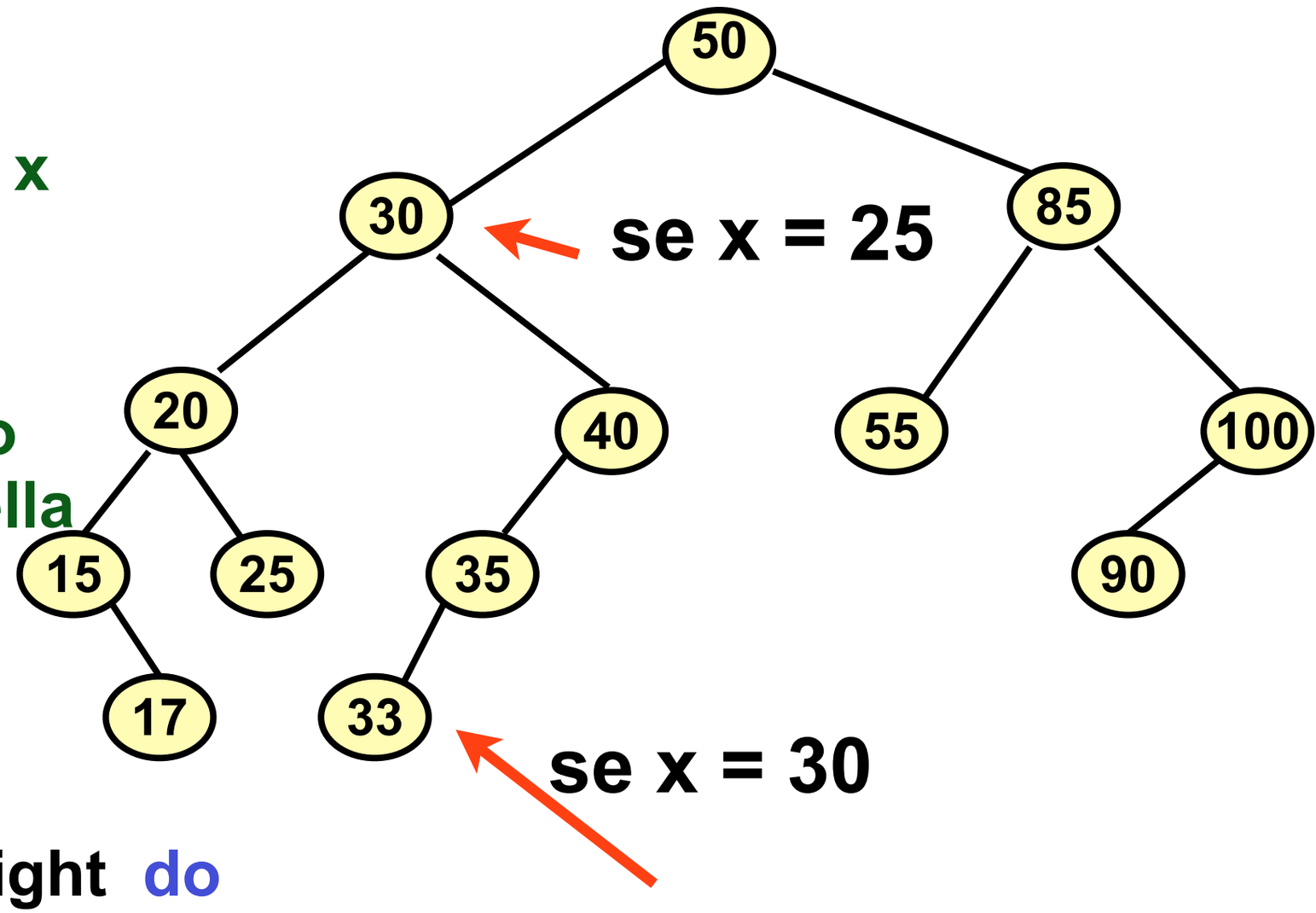
Successor(T,x)

input: T punta alla radice e x a un nodo di un ABR

prec: T e x non sono nulli

output: il puntatore al nodo di chiave successiva a quella di x in T

```
if x.right ≠ nil then
  return Minimum(x.right)
  y = x.p
     while y ≠ nil and x == y.right  do
          x = y
          y = x.p
     return y
```



Complessità O(h)

Visita Inorder ABR

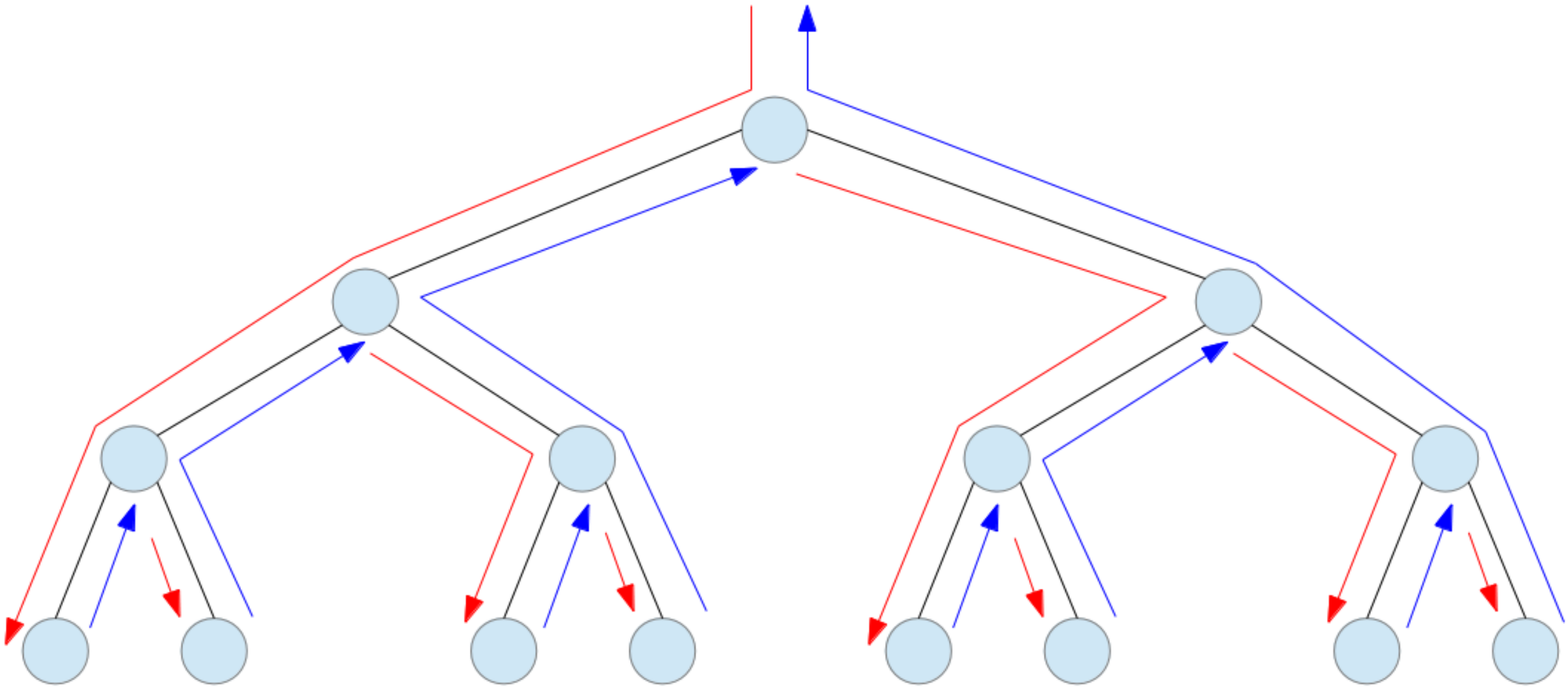
```
Inorder(T)
input: T punta alla radice di un albero binario
prec: T è  ABR non nullo
output:la sequenza delle chiavi di T in ordine crescente
x = Minimum(T) %x è il puntatore al nodo di chiave minima in T
repeat
    print x.key
    x = Successor(x)
until x ≠ nil
```

Visita Inorder ABR

Si può dimostrare che ogni arco (u,v), con v=u.left o v=u.right, viene attraversato al più due volte.

Poichè gli archi sono n-1, si ottiene una complessità asintotica O(n).

(per esercizio fate una prova induttiva dell'affermazione: se un albero binario ha n nodi allora ha n-1 archi.)



In rosso le “discese” da padre a figlio per il calcolo del successivo con l’individuazione del minimo nel sottoalbero destro e in blu le risalite alla ricerca del successivo come l’antenato più vicino che ha la chiave nel suo sotto albero sinistro.

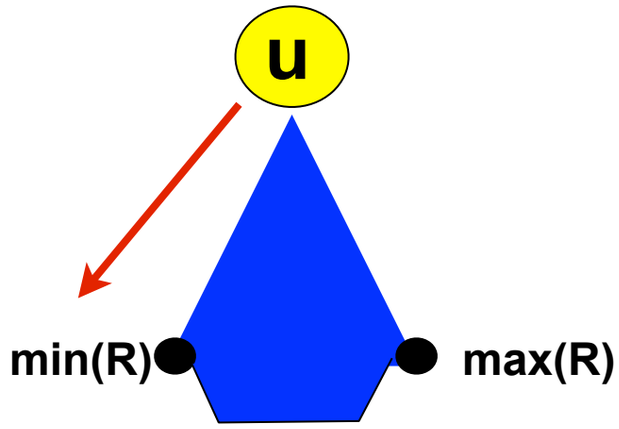
Visita Inorder ABR

VisitaItInorderABR u:

1. determinare il minimo, x, di u, visitarlo e

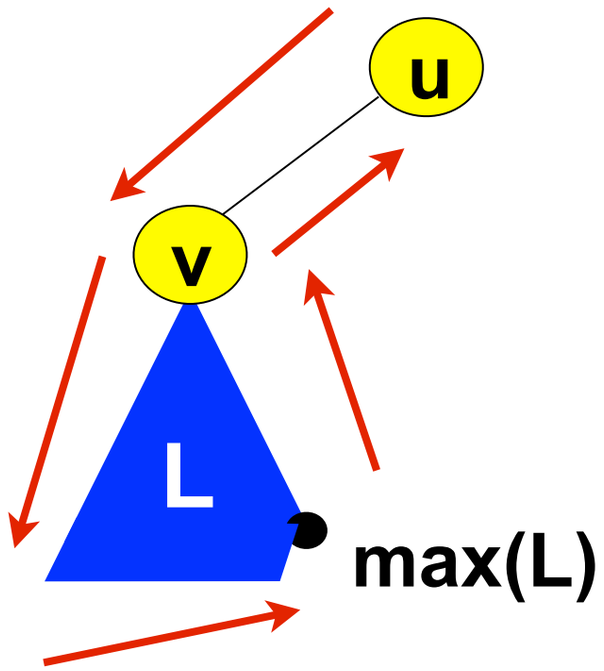
2. calcolare il successivo di x, y, visitarlo e

3. copiare y in x e ripetere il punto 2 fino a che il successivo è nullo.



Partendo dalla radice si attraverseranno gli archi del cammino più a sinistra nella chiamata di MINIMUM, questi archi potranno essere riattraversati quando si cercherà il successivo di un nodo che non ha figlio destro

Attraversamenti arco (u,v), v figlio sinistro



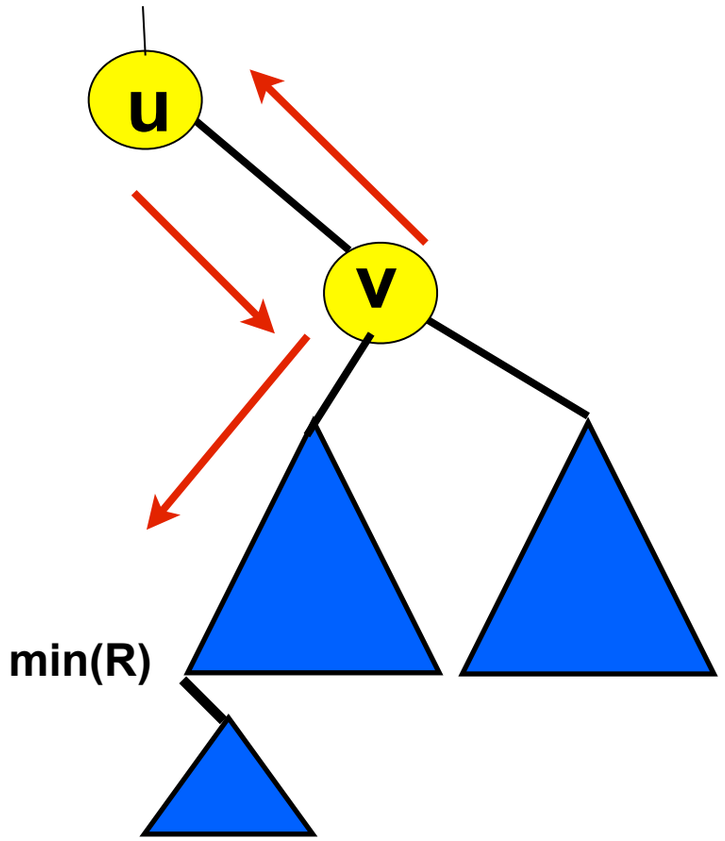
Caso 1. v è figlio sinistro di u. Poiché le chiavi dei nodi nel sotto albero sinistro di u sono minori di u tutto il sotto albero sinistro di u è visitato prima di u e questo comporta che l'arco (u,v) è attraversato una volta da padre a figlio.

SUCCESSOR(max(L)) = u

Quando si cerca il successivo del massimo si deve tornare a u, quindi si riattraversa l'arco (u,v) da figlio a padre.

Il successivo di u si trova o nel suo sottoalbero destro o tra i suoi antenati, quindi l'arco (u,v) non verrà più attraversato.

Attraversamenti arco (u,v), v figlio destro



Caso 2. v è figlio destro di u.
Quando si cerca il successivo di u, si deve prendere il minimo nel sottoalbero radicato in v, R, quindi bisogna scendere lungo l'arco (u,v), da padre a figlio.

Il successivo del minimo di R può trovarsi nel suo sotto albero destro o tra gli antenati di u, quindi l'arco (u,v) verrà attraversato da figlio a padre.

Rango di un nodo in un ABR

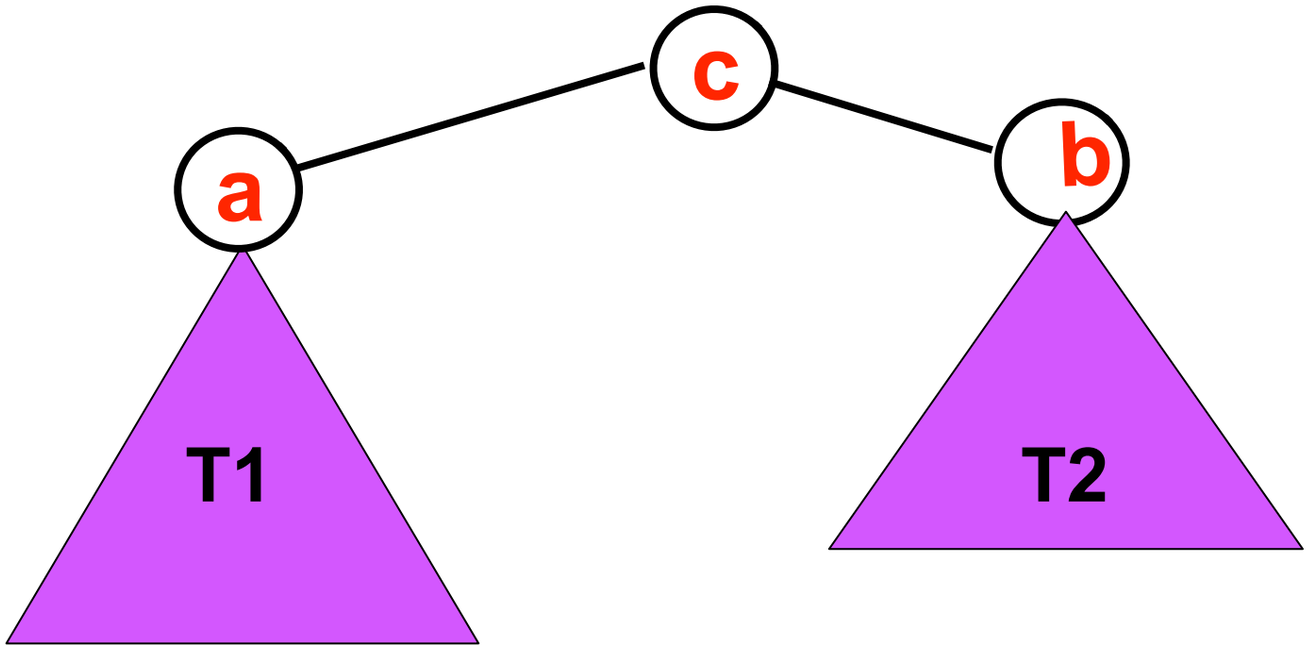
Dato un ABR T non nullo e un puntatore a un suo nodo x si scriva un algoritmo che fornisce in output il rango della chiave di x in T, cioè il numero degli elementi di chiave minore a quella di x in T.

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

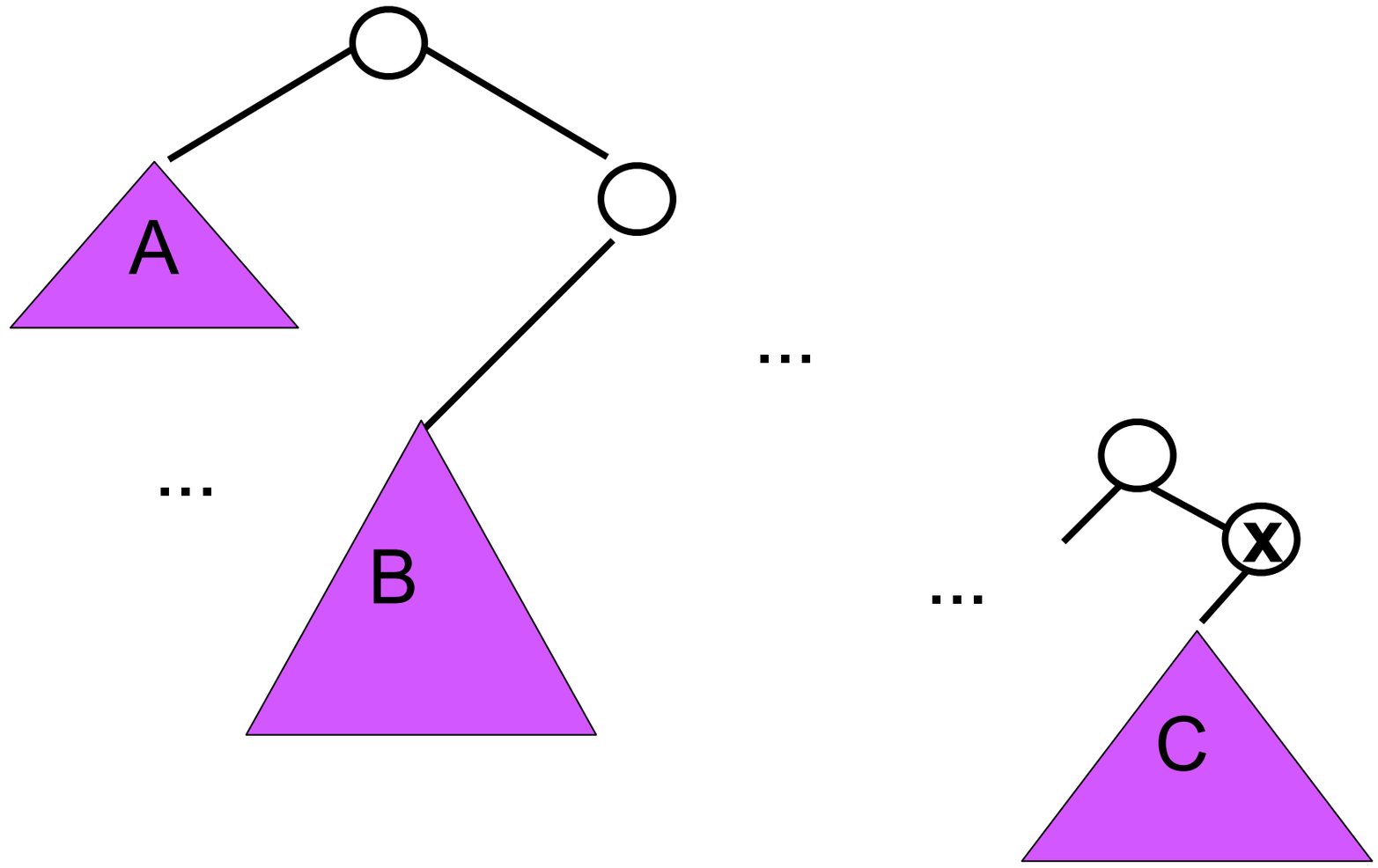
```
Rango(T,x)
        if T == NIL then return  0
        if T.key == x.key then return   nNodi(T.left)
        if x.key < T.key then return  Rango(T.left,x)
         else  return   nNodi(T.left) + 1 + Rango(T.right,x)
```

%x.key > T.key

Rango di x in T

Se x è il massimo

T=



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 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 2 ABR

```
Rango2(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 (z ≠ nil and 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. Se cerco il rango del minimo il tempo è O(h).

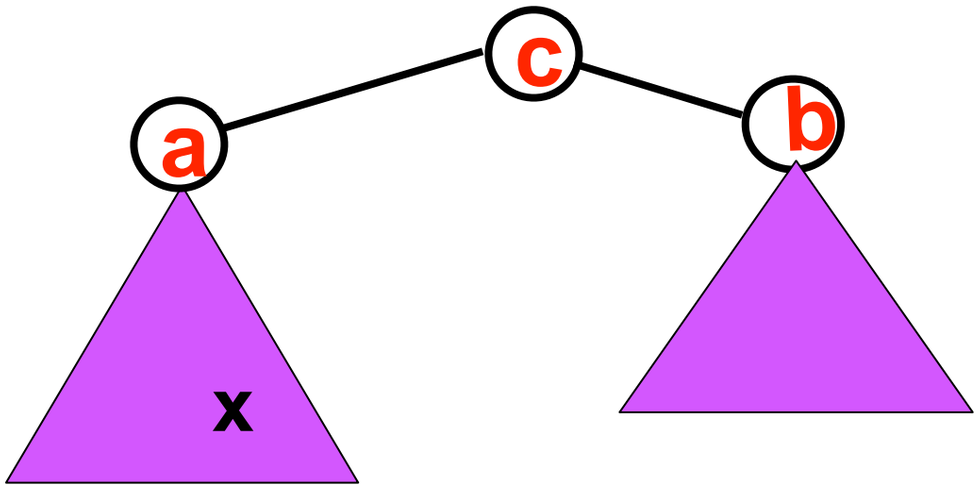
Ricerca di un nodo di rango r

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 1

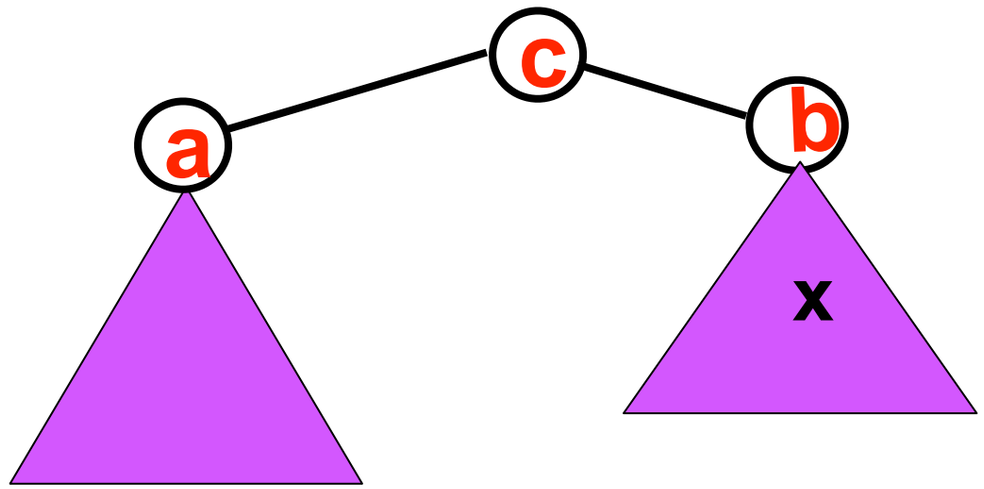
- **Se T ha un solo nodo e r=0 allora la risposta è T**



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.

Ricerca di un nodo di rango r 2



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<n=nNodi(T)

output: il riferimento al nodo di rango r

if T == NIL then return NIL

n_1 = nNodi(T.left)

if n_1 == r then return T

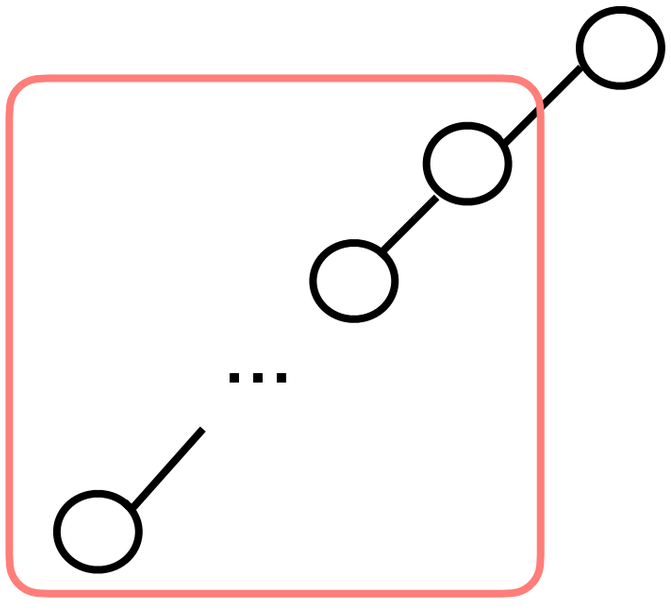
if n_1 > r then return Select1(T.left,r)

else return Select1(T.right,r-n_1-1)

Select: analisi

```
Select1(T,r)
    Input: un albero binario T e valore r
    prec: 0≤r<n=nNodi(T)
    output: il riferimento al nodo di rango r
    if T == NIL then return  NIL
    n1 = nNodi(T.left)
    if n1 == r then return   T
   if n > r then return Select1(T.left,r)
            else return Select1(T.right,r-n1-1)
```

T=



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

Ricerca di un nodo di rango r, versione iterativa

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.

Tempo di esecuzione O(r + h).

Select(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

```
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
```

Intervallo chiavi

Dato un ABR T e due chiavi a e b (non necessariamente presenti nell'albero) si scriva un algoritmo che fornisce in output, ordinate, le chiavi x dell'albero tali che a≤x≤b.

Intervallo chiavi

Dato un ABR T e due chiavi a e b (non necessariamente presenti nell'albero) si scriva un algoritmo che fornisce in output, ordinate, le chiavi x dell'albero tali che a≤x≤b.

```
StampaTra(T,a,b)
    Input: un puntatore T e due chiavi
    prec:T è un ABR e a≤b, min(T)≤a≤b≤max(T)
    output: stampa le chiavi di T comprese tra a e b
    if T ≠ nil then
            StampaTra(x.left,a,b)
            if T.key ≥ a and T.key ≤ b then print x.key
            StampaTra(x.right,a,b)
```

Soluzione semplice, ma inefficiente, infatti visita sempre tutti i nodi con una visita inorder, con un tempo di esecuzione O(n), se n è il numero dei nodi.

Intervallo chiavi sol

Soluzione alternativa, che fa riferimento alla visita inorder alternativa che abbiamo visto. L'idea è cercare a in T, facendo in modo di ottenere il puntatore al nodo di chiave a o al padre della foglia in cui a sarebbe inserita nell'albero; quindi si stampano tutte le chiavi dei nodi successivi, di chiave minore o uguale a b.

```
StampaTraEff(T,a,b)
    Input: un puntatore T e due chiavi
      prec:T è un ABR e a≤b, min(T)≤a≤b≤max(T)
      output: stampa le chiavi di T comprese tra a e b
 if T ≠ nil then y = Tree-Search-Mod(T,a)
     y è un nodo di chiave a o il nodo che avrebbe a come figlio
  while y ≠ nil and y.key ≤ b do
       if   y.key ≥ a then print(y.key)
       y = SUCCESSOR(y)
return
```

Complessità O(m + h), se m è il numero dei nodi con chiavi comprese tra a e b e h è l'altezza dell'albero.