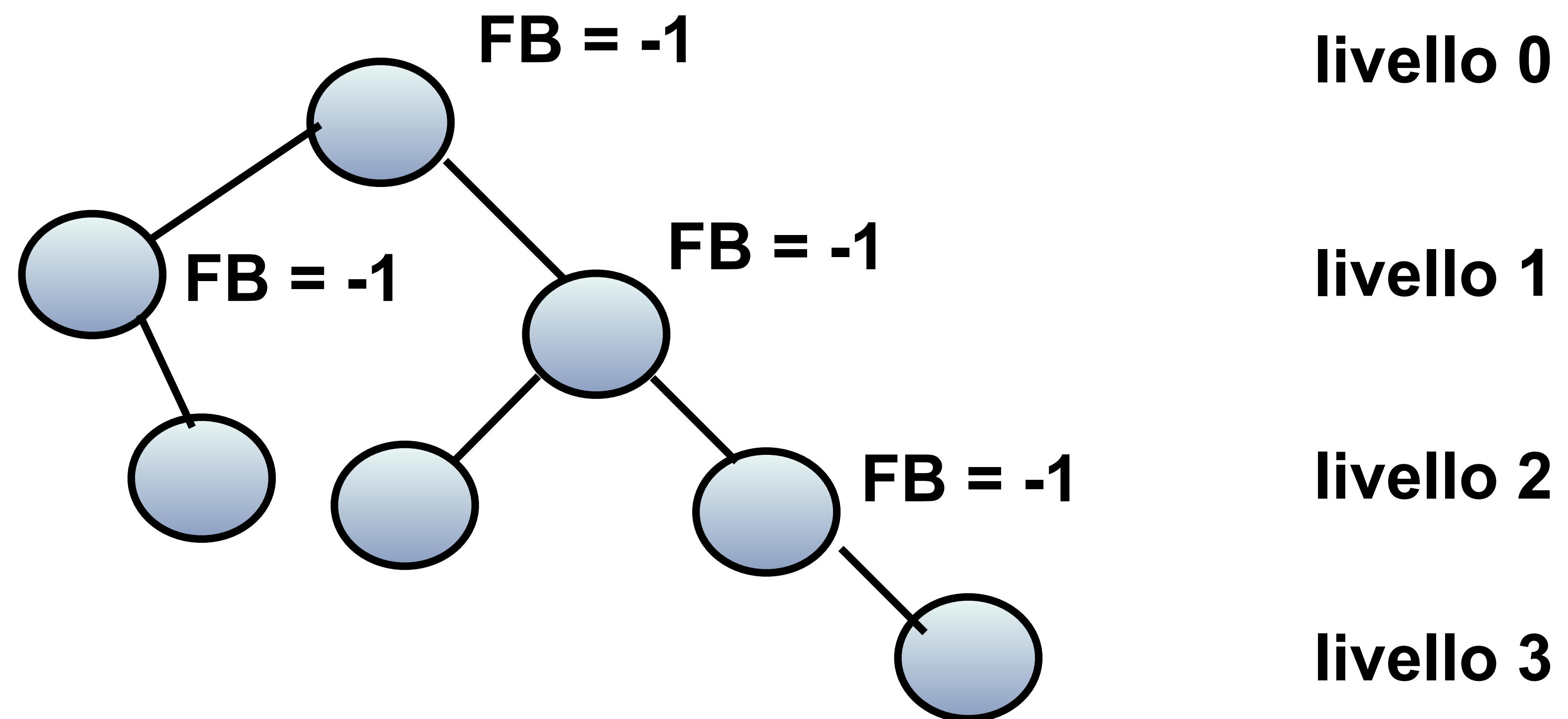
Esercizio 1

E' vero che in un AVL il minimo si trova in una foglia o nel penultimo livello?



L'altezza è 3, il minimo si trova nel livello 1

Cancellazione alternativa

Si modifichi l'algoritmo di rimozione di un nodo in un ABR trattando in modo alternativo il caso del nodo con due figli e lasciando inalterati i casi base di un nodo con un solo figlio o 0 figli.
Nel caso di un nodo x con due figli lo si faccia "scendere" nell'albero utilizzando delle rotazioni fino a rientrare in uno dei casi base. A questo punto il nodo viene rimosso come un qualsiasi nodo con un solo figlio o senza figli.
Confrontate le due implementazioni della rimozione di un nodo da un ABR dal punto di vista della complessità asintotica.

Cancellazione alternativa

L'algoritmo verifica in un ciclo che il nodo da cancellare abbia il figlio destro e se è così fa una rotazione a sinistra sul figlio destro, provocando una "discesa" del nodo da cancellare nel sotto albero sinistro. All'uscita dal ciclo si rimuove come nodo con un solo figlio.

Asintoticamente i due metodi sono equivalenti.
Nel caso della cancellazione "classica" i puntatori modificati sono solo quelli necessari per sostituire il nodo da cancellare con il nodo di chiave minima nel suo sotto albero destro, le altre operazioni si riducono a confrontare i puntatori con NULL per raggiungere il minimo.
Nella nuova versione si rischia di scendere di h passi, se h è l'altezza dell'albero facendo h rotazioni, ciascuna delle quali prevede l'aggiornamento di due puntatori ai figli e quindi due ai padri.

Esercizi bilanciamento in altezza

Si costruisca e si analizzi un algoritmo che verifica se un dato albero binario T è bilanciato in altezza.

L'albero è rappresentato in memoria con puntatori ai figli.

Verifica AVL - l'idea

L'idea è di modificare la visita inorder.

Ogni chiamata della funzione calcola e restituisce l'altezza del sottoalbero visitato.

Prima di uscire dalla chiamata, quindi dopo la visita del sottoalbero destro, si controlla se le altezze dei sottoalberi differiscono in valore assoluto al più di 1. In caso la verifica fallisca restituiamo un valore che non può essere l'altezza di un albero binario, per esempio -2.

Bisogna allora anche controllare che i valori delle altezze restituite siano diversi da -2 prima di calcolare la nuova altezza altrimenti perderemmo l'informazione che un sottoalbero viola la proprietà.

La complessità è quella della visita inorder, quindi O(n)

Verifica AVL soluzione

La funzione descritta di seguito prende in input (la radice di) un albero binario di ricerca T e restituisce l'altezza dell'albero in input se l'albero è bilanciato in altezza e -2 altrimenti.

```
algoritmo verificaAVL(T)
if T = NULL return -1
h1 = verificaAVL(figlio sinistro di T)
if h1 = -2 return -2     *non è un AVL*
h2 = verificaAVL(figlio destro di T)
if h2 ≠ -2 e |h1 - h2| ≤ 1 then return max(h1,h2) +1 else return -2
```

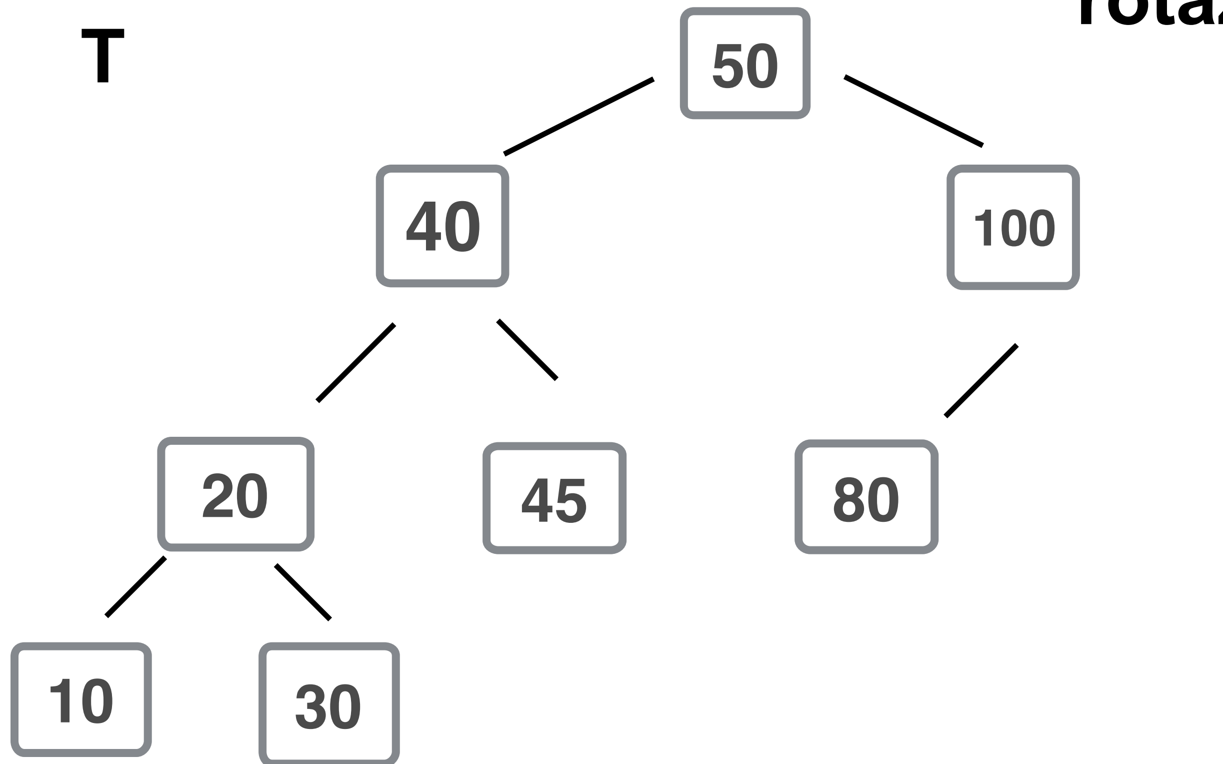
SPLIT di un ABR

Sia T un ABR e x un suo nodo. Ogni nodo di è rappresentato in memoria con puntatore al padre. Si scriva un algoritmo che, preso in input T e x, restituisce due ABR T1 e T2, l'uno contenente le chiavi minori di x e l'altro le maggiori.

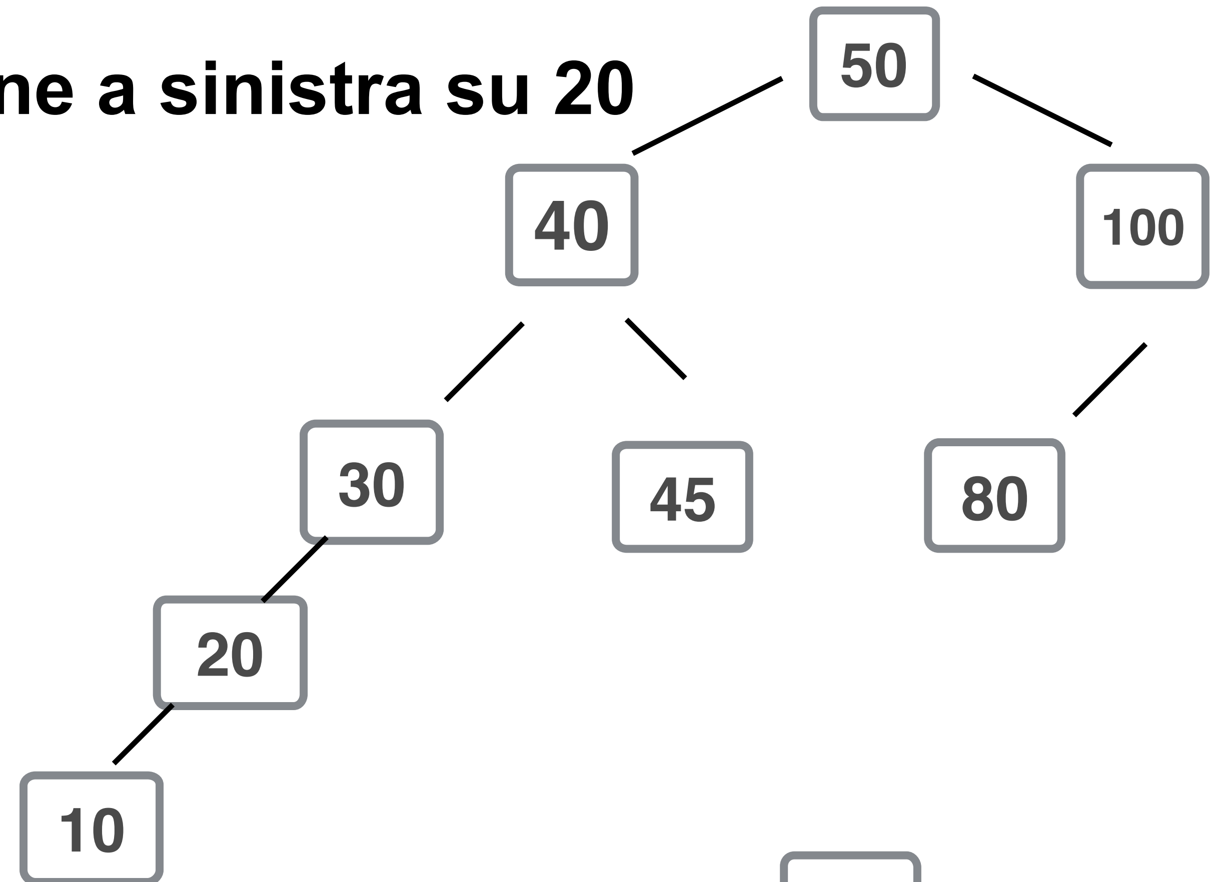
Esempio: x = 30

Verso la soluzione

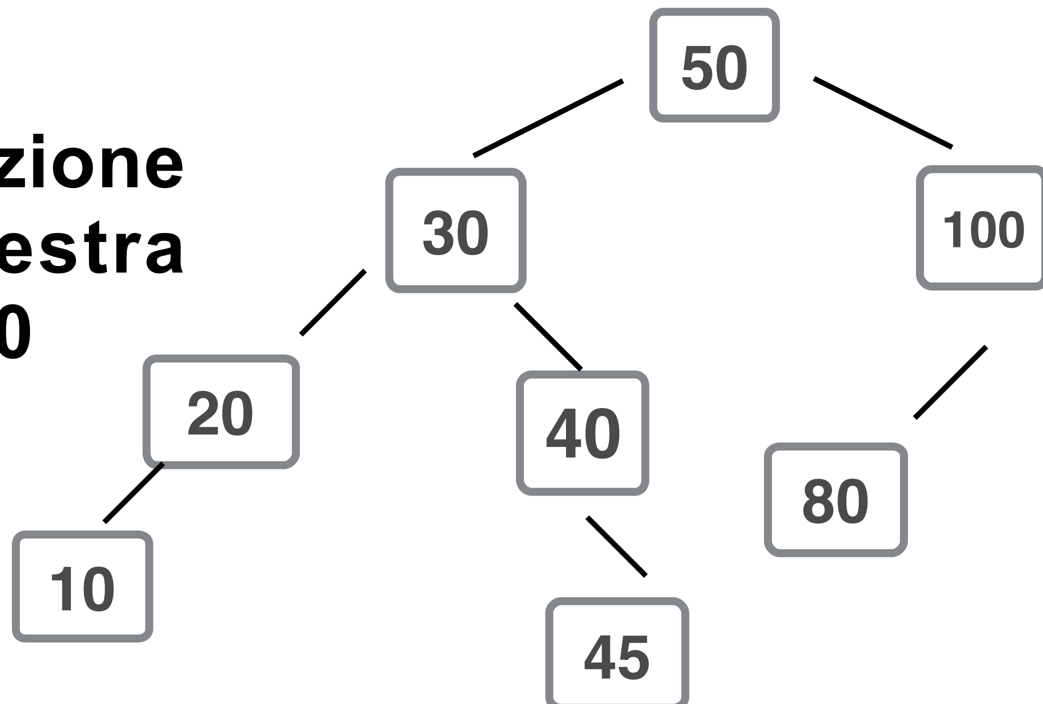
Esempio: x = 30



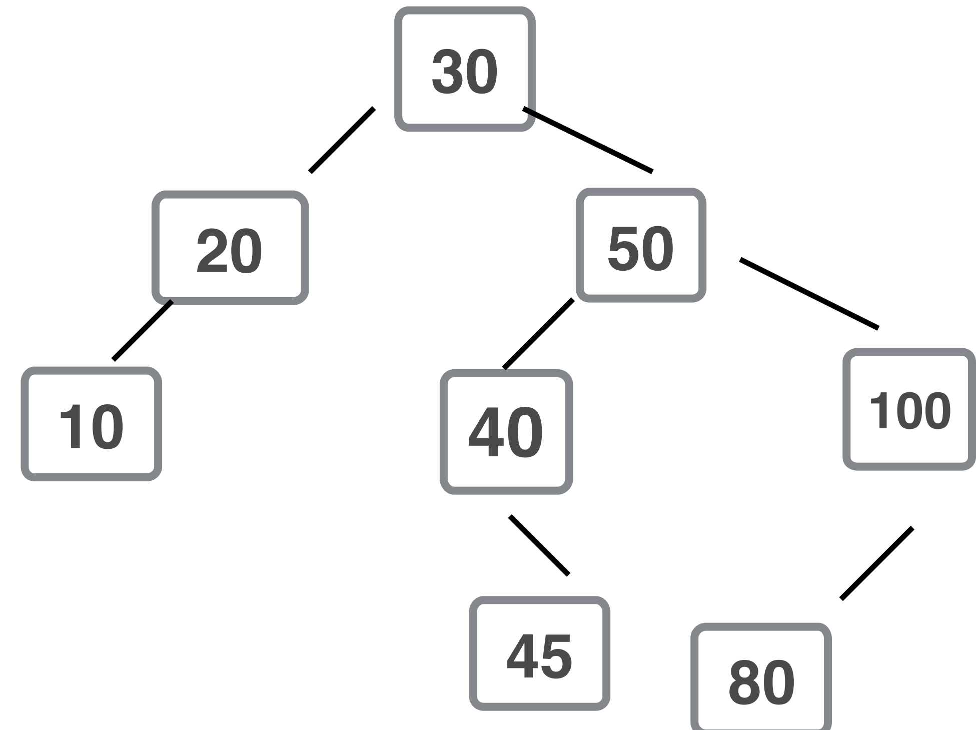
rotazione a sinistra su 20



rotazione a destra su 40

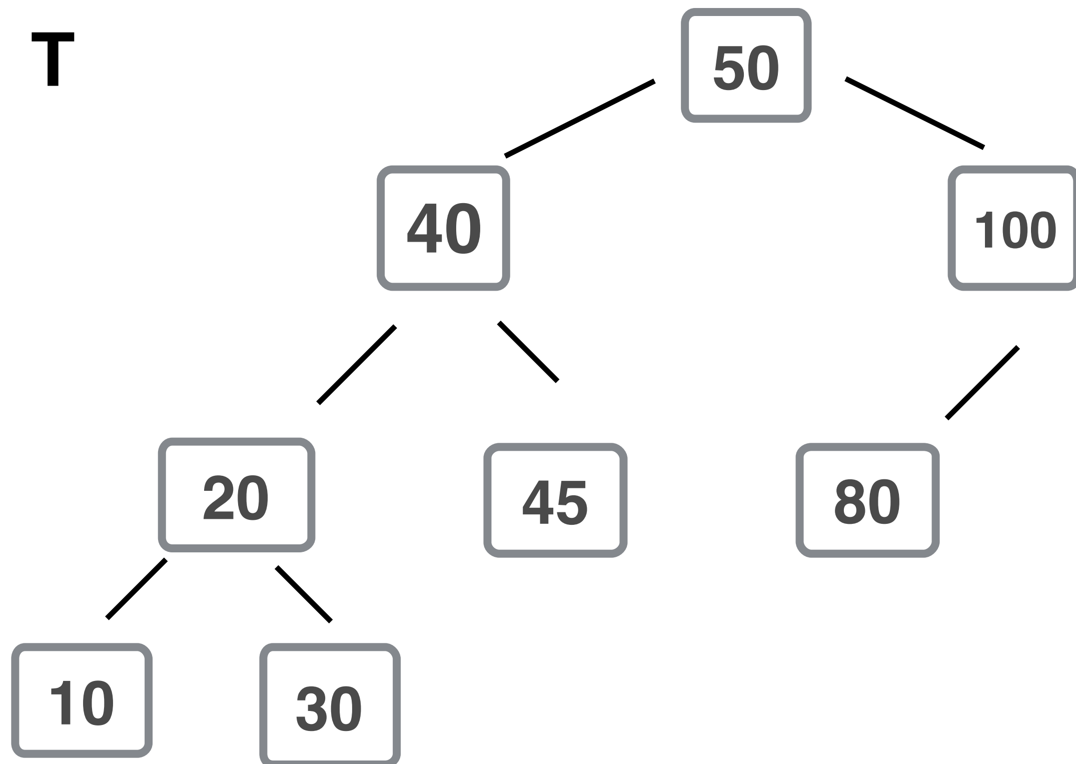


rotazione a destra su 50

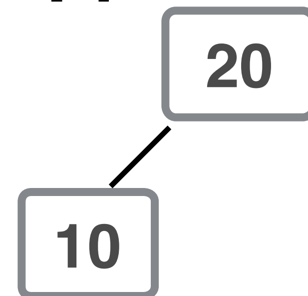


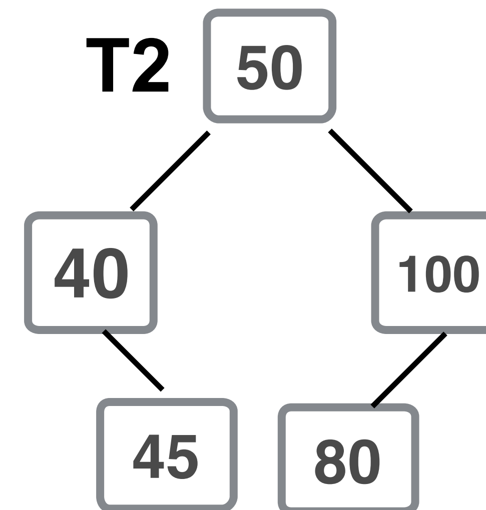
Fine esempio

Esempio: x = 30



T1





SPLIT pseudocodice

Si usano le funzioni di rotazione

LeftRot(y)

input: y è un nodo di un albero binario

Prec: y è un nodo di un ABR non vuoto che ha il figlio destro

postc: il figlio destro, x, di y diventa suo padre e ne prende il posto come figlio;

conserva il suo figlio destro, e prende y come figlio sinistro mentre il figlio sinistro di x diventa figlio destro di y.

RightRot(y)

input: y è un nodo di un albero binario

Prec: y è un nodo di un ABR non vuoto, che ha il figlio sinistro

postc: il figlio sinistro, x, di y diventa suo padre e ne prende il posto come figlio; conserva il suo figlio sinistro,

e prende y come figlio destro mentre il figlio destro di x diventa figlio sinistro di y.

N.B. x e y puntano agli stessi nodi dopo la rotazione

SPLIT pseudocodice cont.

```
Split(T,x)
    Input: un puntatore T e a un nodo x
    prec: T è un ABR non nullo e x è un nodo di T
    postc: restituisce la coppia di puntatori radici di due ABR, l'uno con le
    chiavi minori di quella di x e l'altro con le maggiori.
   y = x.p
   while y ≠ nil do
     y è il padre di x
      if (y.left ≠ Null  and y.left == x) then RightRot(y)
      if (y.right ≠ Null  and y.right == x) then LeftRot(y)
      y = x.p
 return (T.left,T.right)
```

Il più vicino antenato

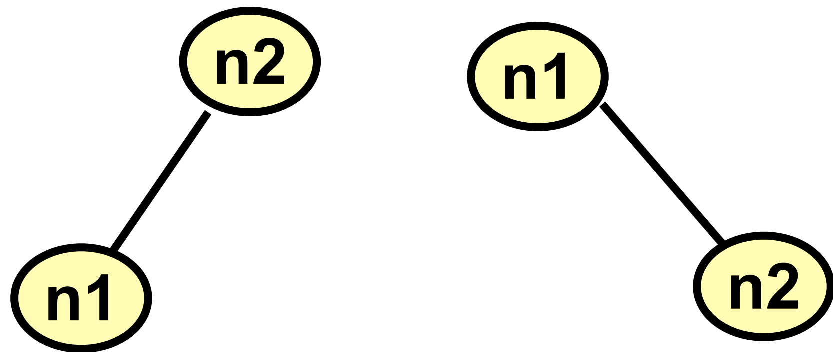
Presi due valori n1 ed n2 ed un ABR T, rappresentato in memoria con puntatori ai figli sinistro e destro, si scriva un algoritmo che trovi il più vicino antenato dei due nodi aventi chiavi n1 ed n2.

Si può assumere che le due chiavi siano presenti in T. L'algoritmo progettato deve restituire il puntatore all'antenato comune più vicino.

Si dimostri la correttezza e si analizzi la complessità dell’algoritmo proposto, che dovrebbe essere O(h), dove h denota l'altezza dell'albero.

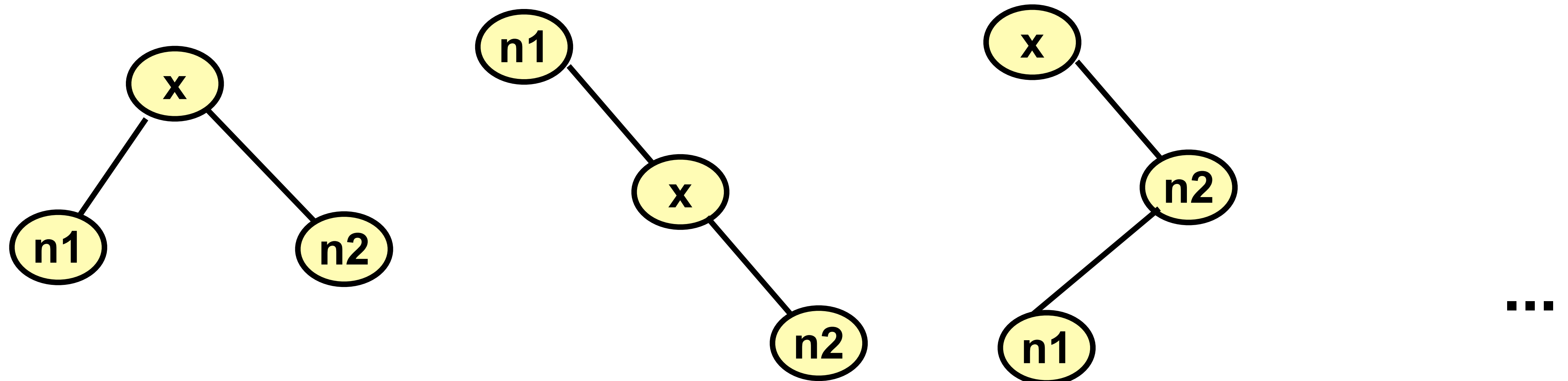
il più vicino antenato: analisi

Per problemi su alberi, l'analisi parte sempre dai più piccoli alberi.
supponiamo che n1<n2, per semplicità, allora i soli alberi con due nodi sono:



Nel primo il nodo di chiave n1 è il l'antenato comune, nel secondo quello di chiave n2

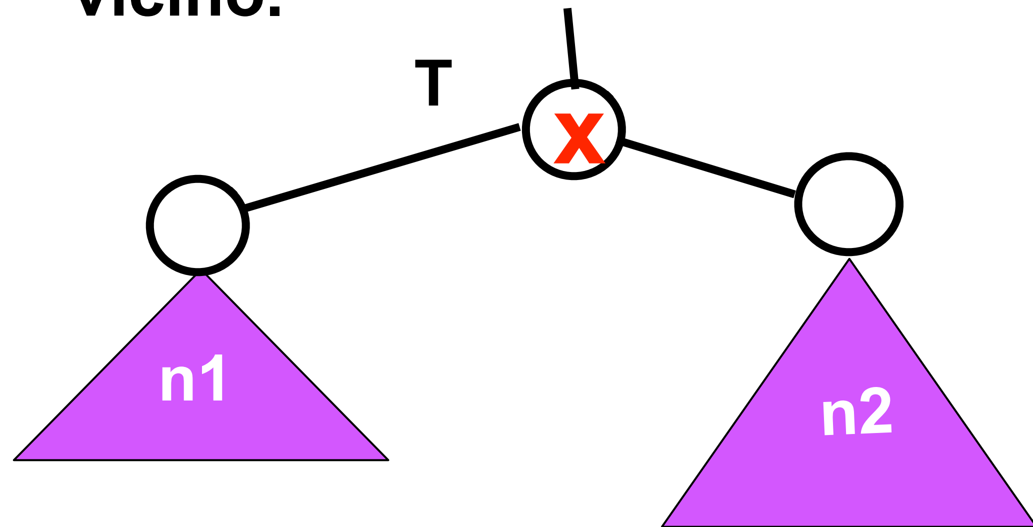
Con tre nodi si può avere:



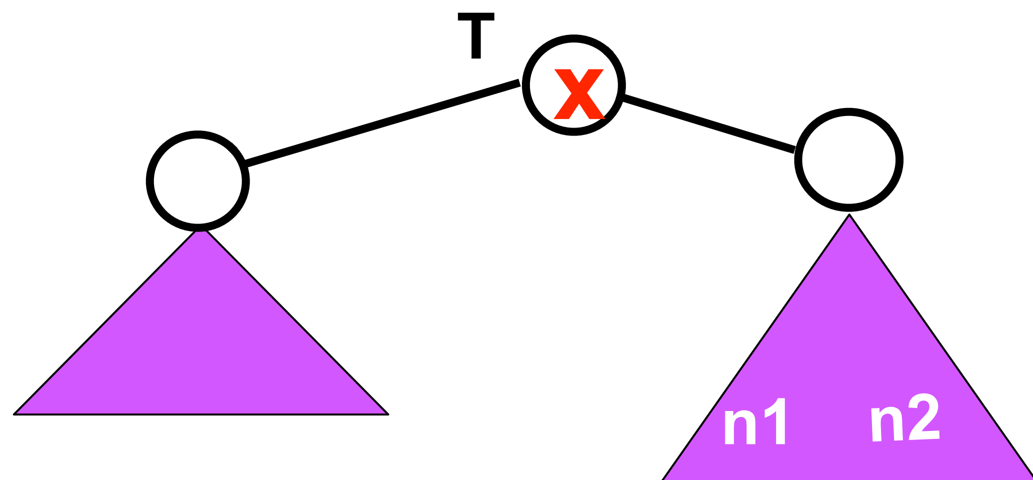
Qui il nodo di chiave x è l'antenato comune, nel secondo quello di chiave n1, nel terzo quello di chiave n2, etc.

il più vicino antenato: analisi fase 2

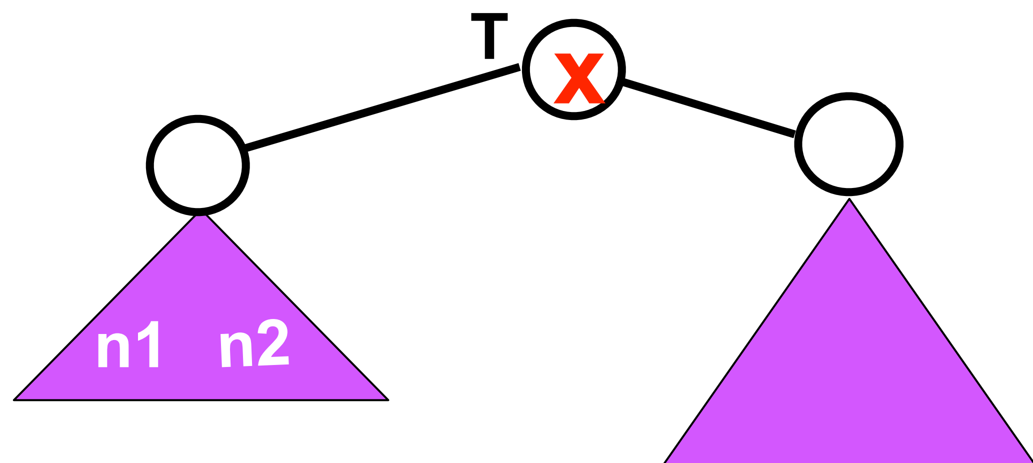
Generalizzando si vede che se x è un nodo il cui sottoalbero sinistro contiene n1 e il cui sottoalbero destro contiene n2, allora x è l'antenato più vicino.



Il nodo T di chiave x può essere figlio sinistro o destro. Osserviamo che questo caso è caratterizzato dal fatto che n1<x<n2.
Chiaramente il nodo di chiave x è l'antenato comune più vicino, visto che ogni antenato di n1 nel sottoalbero destro di T non lo è di n2 , e viceversa.
Quindi in questo caso l'algoritmo deve restituire T



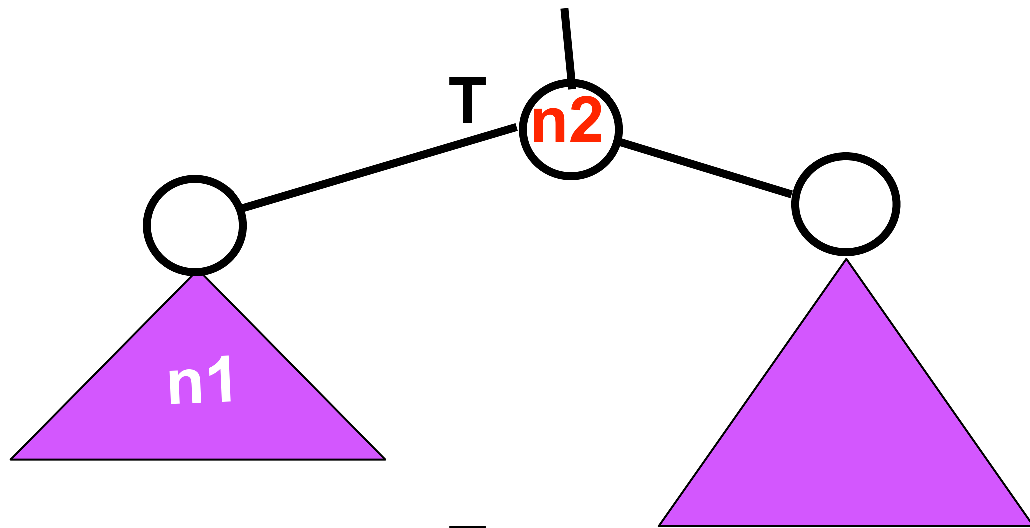
mentre questo caso è caratterizzato dal fatto che x<n1 e x<n2, e certamente il nodo T di chiave x non è l'antenato comune più vicino, perchè se non altro potrebbe esserlo il figlio destro, si deve quindi cercarlo nel sotto albero destro



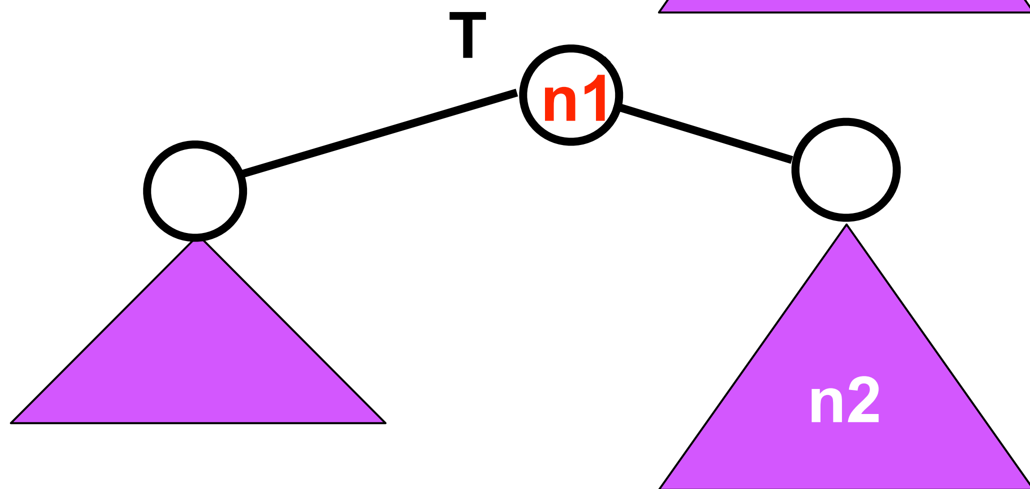
Questo è il caso simmetrico, per il quale valgono considerazioni analoghe e che è caratterizzato dal fatto che x>n1 e x>n2. Si deve dirigere la ricerca nel sotto albero sinistro.

il più vicino antenato: analisi 3

Sotto l'ipotesi che n1<n2 altri casi che si possono verificare sono



Il nodo T qui ha la chiave uguale a n2 e maggiore di n1, quindi è il nodo da restituire.



Mentre in questo caso la chiave di T è uguale a n1 e minore di n2, ed è il nodo da restituire

Quindi i casi base sono quando n1<x<n2, oppure x=n1< n2 o x= n2 e x>n1

if (n1 ≤ T.key **and** T.key ≤ n2) **then** **return** T

che comprende i tre casi in cui T è l'antenato. Altrimenti basterà dirigere la ricerca nel sottoalbero sinistro o nel destro a seconda dell'esito dei confronti:

Il più vicino antenato Pseudocodice

Antenato(T,n1,n2)
input: T è un albero binario (il riferimento alla radice), n1 e n2 due valori
prec: T è un ABR, n1 e n2 due chiavi prsenti in T e n1<n2
postc: restituisce il puntatore all'antenato comune più vicino (quello i cui discendenti non sono antenati comuni dei nodi chiave n1 e n2)

```
if (n1 ≤ T.key and T.key ≤ n2) then  return T
if (T.key < n1 and T.key < n2)  then  return Antenato(T.right,n1,n2)
if (T.key > n1 and T.key > n2)  then  return Antenato(T.left,n1,n2)
```

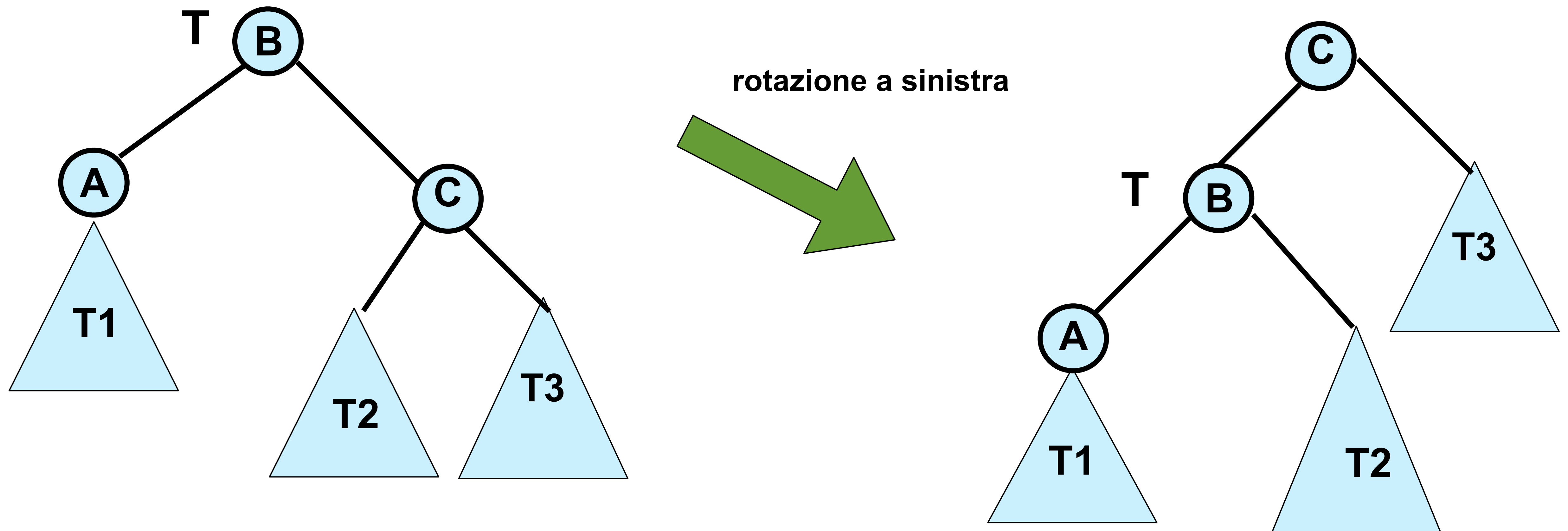
Correttezza: deriva dalle considerazioni fatte prima.

Complessità: Ogni chiamata compie un lavoro costante, inoltre si fa al più una chiamata su uno dei due figli quindi al più si eseguono h chiamate se h è l'altezza dell'albero.
Si può concludere che si ha una complessità asintotica O(h) .

Catena sinistra

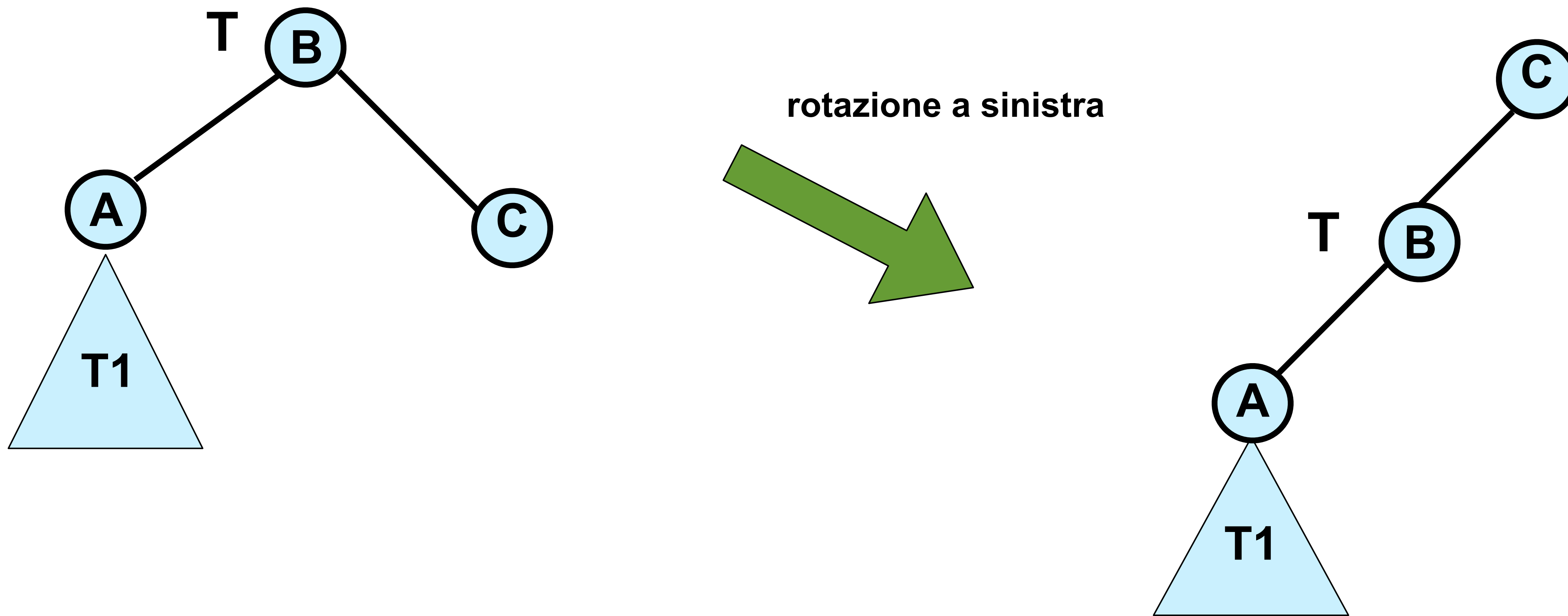
Si scriva una funzione ricorsiva per trasformare un ABR T qualunque in una catena sinistra (cioè un ABR in cui ogni nodo ha solo il figlio sinistro) utilizzando le rotazioni introdotte per ribilanciare gli AVL
Si dia una valutazione della complessità asintotica della funzione.

Catena sinistra



Ora per capire se è necessario agire su C, bisogna verificare se T3 è vuoto o no, se no bisogna chiamare la funzione su C.

Catena sinistra: l'ultima rotazione



Ora si può scendere al figlio sinistro di B

Esercizio 2: soluzione

CatenaSin(T)

Input: un ABR T

postc: trasforma T in una catena sinistra

```
if T == NIL return T
if T.right ≠ NIL then  Left-Rotation(T)
if T.p.right ≠ NIL then return CatenaSin(T.p)
if T.left ≠ NIL then  return CatenaSin(T.left)
```

Catena sinistra: analisi

Si deve fare una rotazione per ogni nodo che ha un figlio destro.
Se n è il numero dei nodi, si faranno al più O(n) rotazioni.