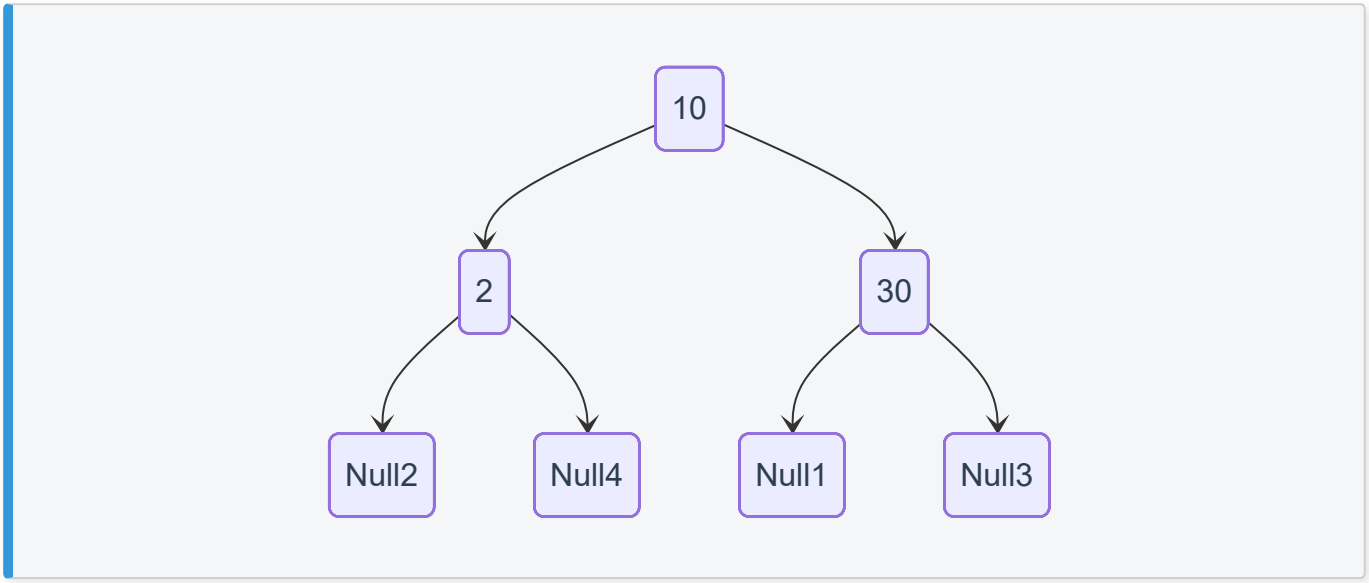
Esercizi

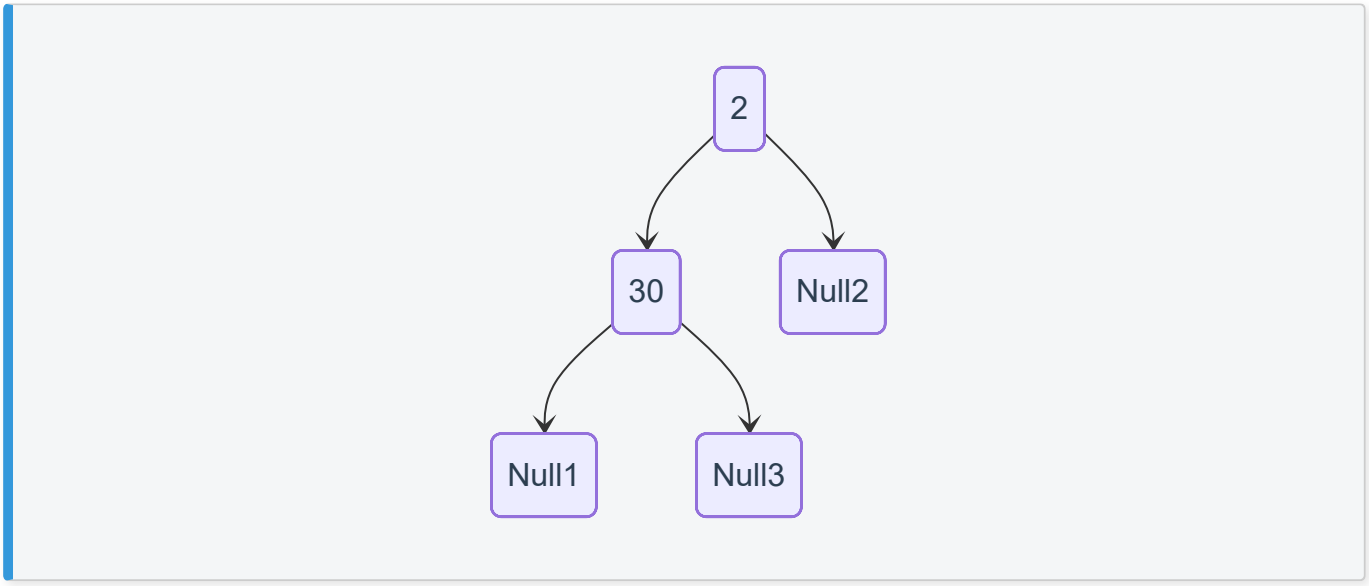
RBT

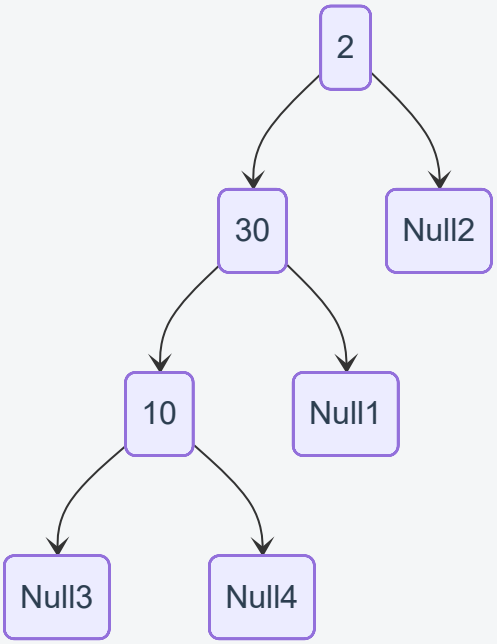
ES:
Se T è un RBT e una chiave K viene cancellata e poi reinserita in T. Al termine T è come era all'inizio? **NO**

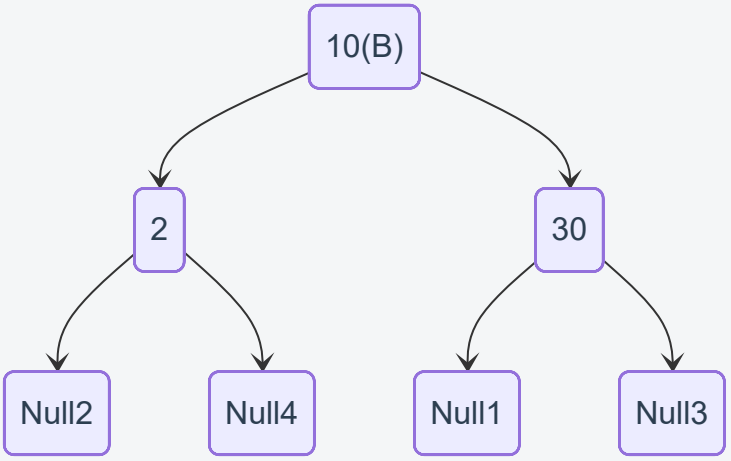
In un BST? **NO**

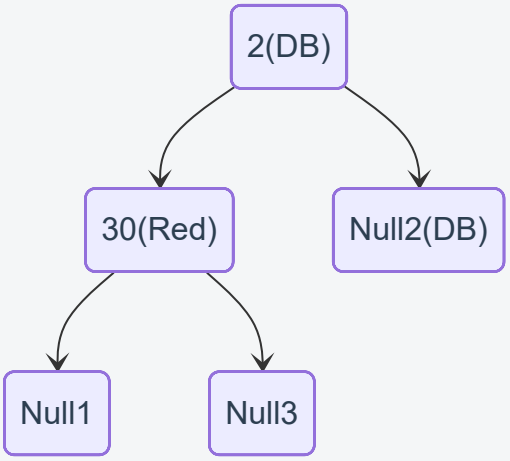


Preced:

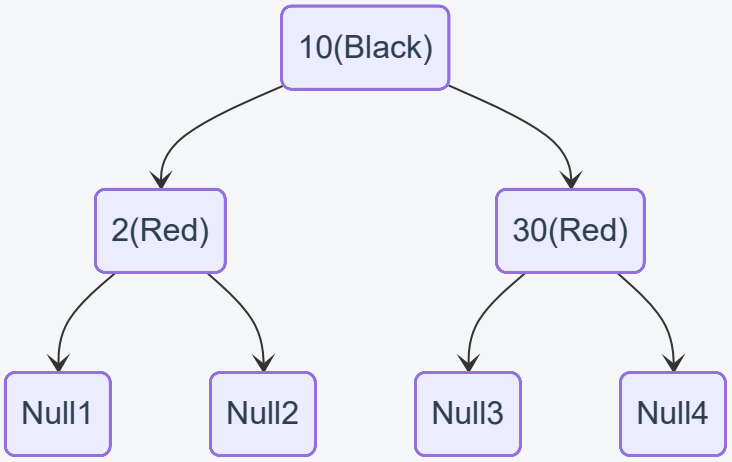



2
30
Null2
10
Null1
Null3
Null4


10(B)
2
30
Null2
Null4
Null1
Null3


2(DB)
30(Red)
Null2(DB)
Null1
Null3

(serie di iterazioni)

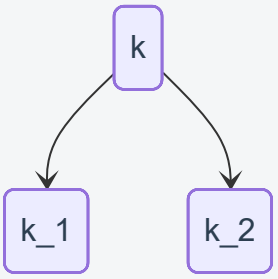


ES:

[][][][][][][][][][][k][][][][][][][][][][] A un vettore no ripetizioni (lunghezza n)

Costruite T BST bilanciato contentente le chiavi di A

k (posizionato in n/2)




```
Costruisci BST(A){
    T<-new.BST()
    x<-new.node
    T.root<-x
    RicCostruisciBST(A, x, 1, A-length)
}
```

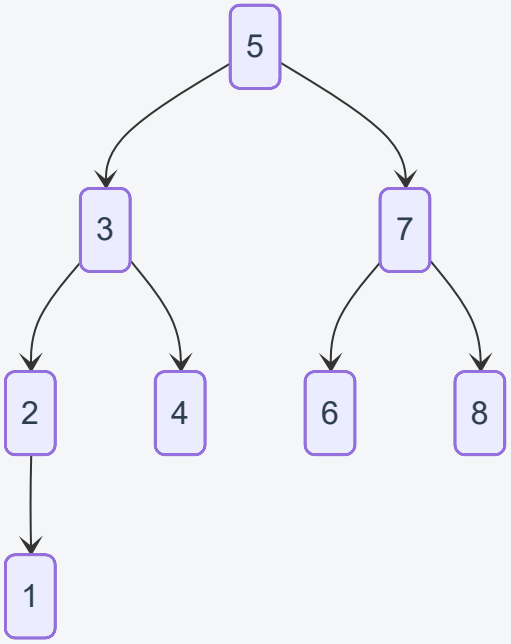
```
RicCostruisciBST(A, x, p, q){   //  q>=p
    r<- (q+p)/2
    x.key<-A[r]
    if (p<=r-1){
```

```
        y<-newNode()
        x.left<-y
        y.parent<-x
        RicCostruisciBST(A, y, p, r-1)
    }
    if (r+1<=q){
        z<-newNode()
        x.right<-z
        z.parent<-x
        RicCostruisciBST(A, z, r+1, q)
    }
}
```

$$T(n) = \begin{cases} \theta(1) & n \leq b1 \\ 2T(\frac{n}{2}) + \theta(1) & n > 1 \end{cases}$$

$n = q - p + 1$
$T(n) = \theta(n)\ 2^m = n$

[1][2][3][4][5][6][7][8]



T) h=O(logn) T) $\exists \overline{\mathrm{n}} \exists c < 0 \forall m \geq \overline{\mathrm{n}} h \leq c \cdot log_n$

Base: $\overline{\mathrm{n}} = 2$
$h = 1 \leq c \cdot \log_2 \overline{\mathrm{n}}$

Passo Induttivo:

Hp) se l<n vale che h_l vale che $h_l \leq c \cdot \log_2 l$

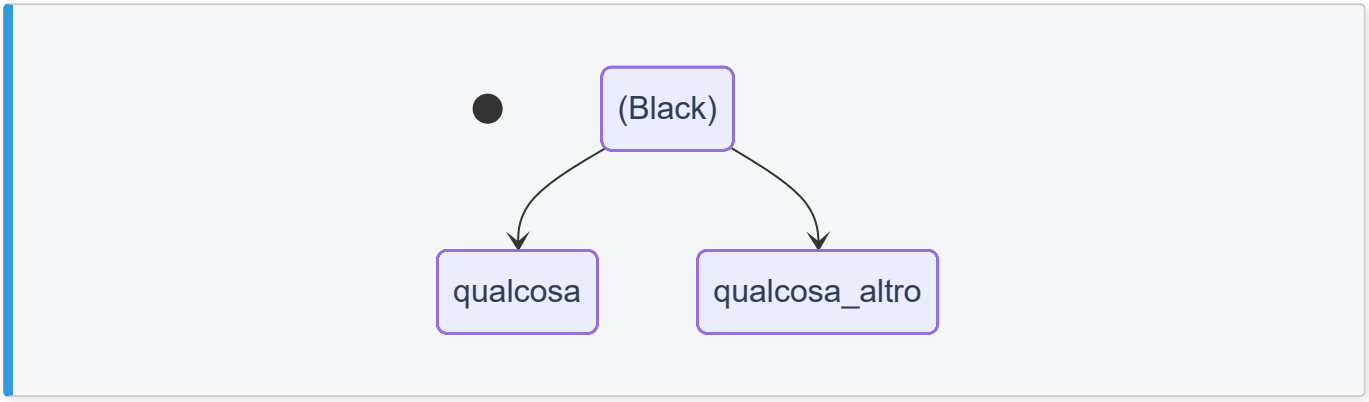
$$h_n = 1 + \log_n \leq 1 + c \cdot log_2 \frac{n}{2}$$

Es

Si consideri una definizione alternativa di RBT in cui solo per la radice viene richiesto che lungo ogni cammino fino ad una foglia ci sia lo stesso numero di nodi BLACK

Esistono degli alberi che soddisfano questa definizione e non quella originale? **NO**

Le due dif sono equivalenti:

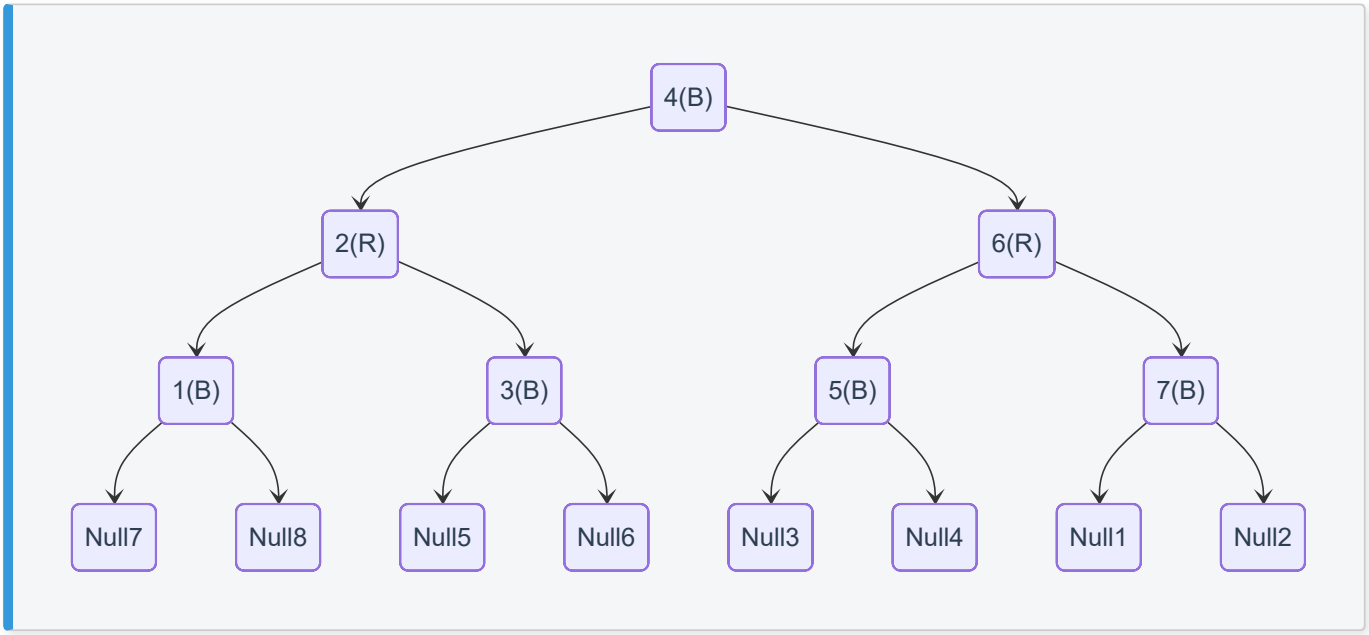


T che soddisfa Def 2 se p.a. T non soddisfa Def 1 allora $\exists x$ con altezza nera non ben definita

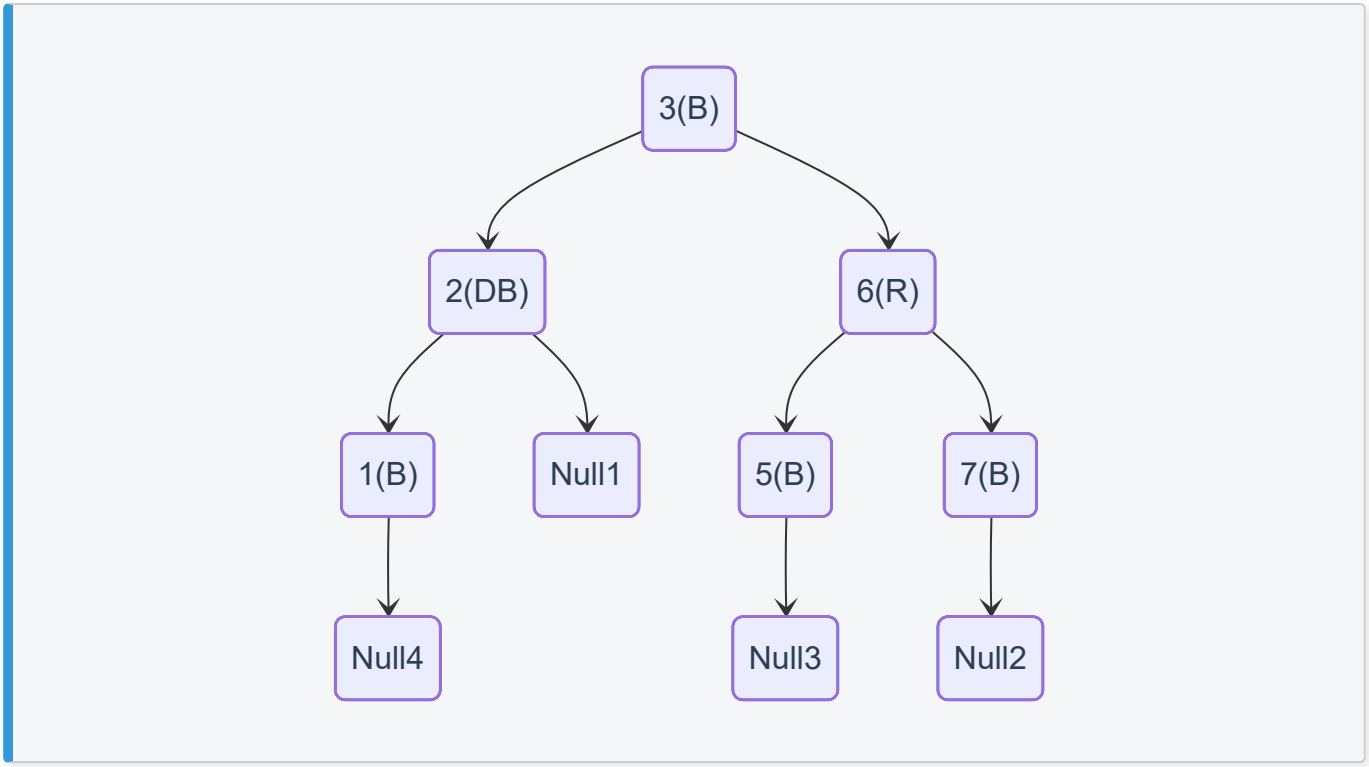
$n \neq m \implies l + m \implies$ ASSURDO: T non soddisfa Def 2

ES1 12/06/2023:

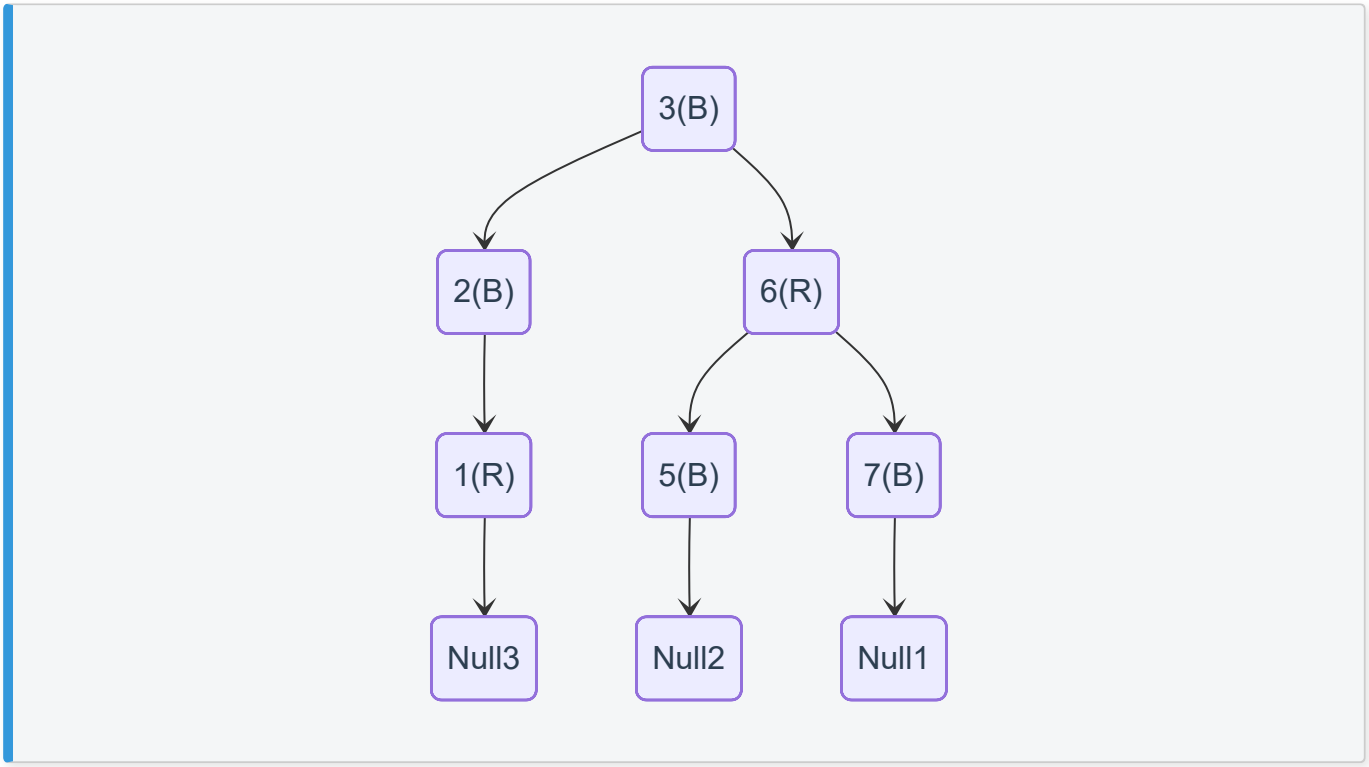
T Red-Black Tree con chiavi 1,2,3,..,7 tutti black
Cancellare la chiave della radice



Dopo la cancellazione:



poi:



In Red Blac Tree contenente $n = n^h - 1$ chiari e eventualmente tutti nodi black quante rotazioni e ricolorazioni costa la cancellazione della chiave della radice?

Ad ogni passo:

- 0 rotazioni

- 1 ricolorazione

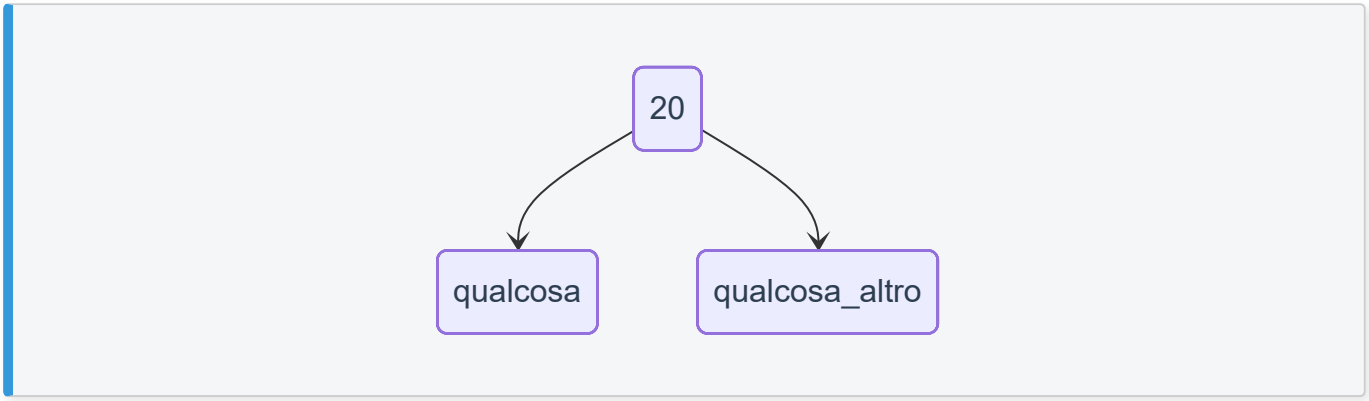
$\theta(\log n)$ Ricolorazioni $\implies$ $\theta(h)$

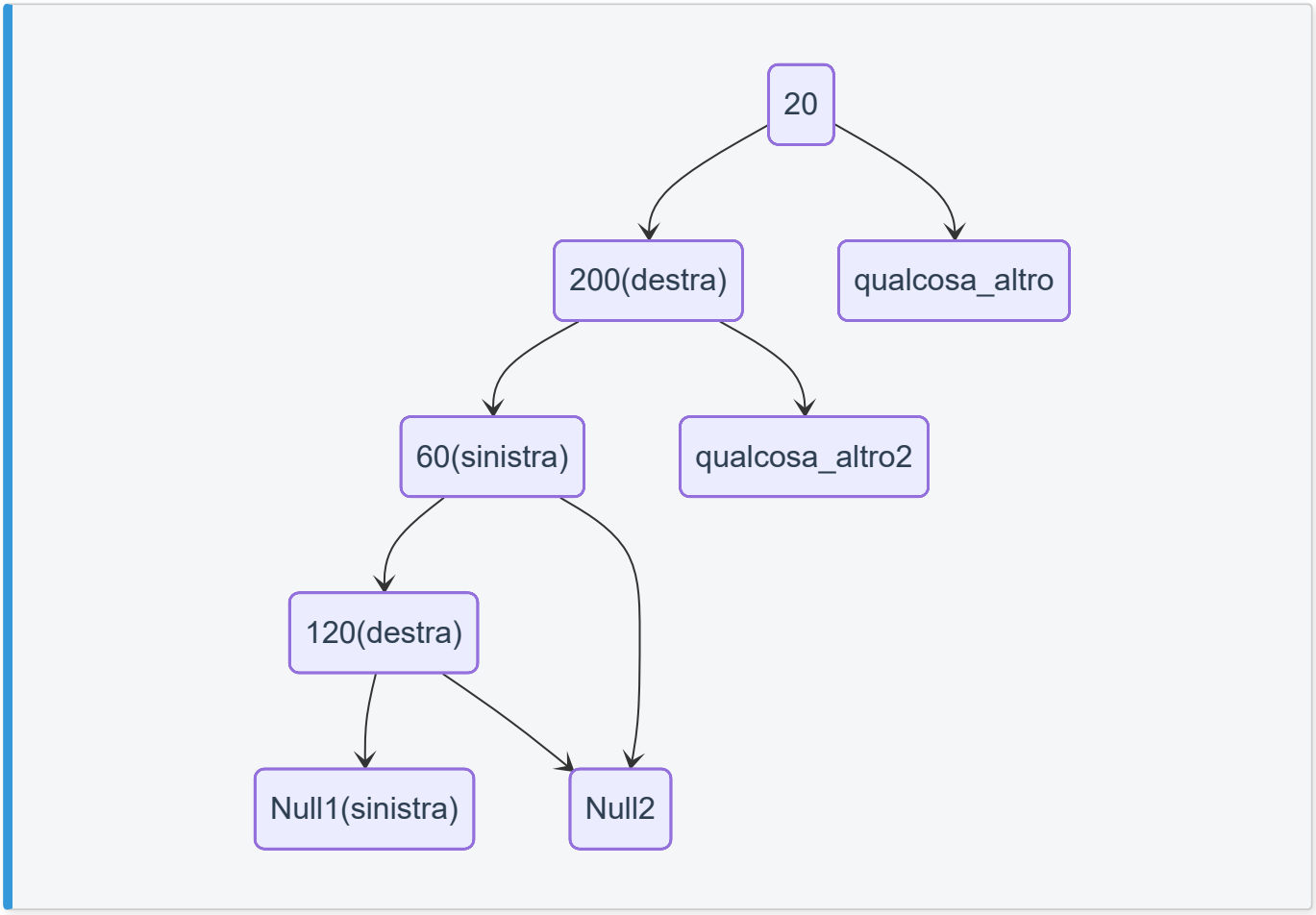
ES1 17/08/2019:

T RBT $k \notin T$

Trovare il sucessore di K in T Senza modificare T

se avessimo k>=80





Mi memorizzo 120 come chiave più piccola trovata finora (120<200)

Se si trova un sottoalbero a destra si continua, altrimenti si risale finché non si trova un sucessore

```
RBT_succ.ext(T,k){
    x<-T.root
    succ<--NIL
    while (x!=NIL){
        if (x.key < k){
            x<-x.right
        }else
            succ<-x
            x<-x.left
    }
}
```

27/02/2025

ES:

A vettore lung n

a) Heap.Sort in place? **SI:** Sfrutta BeildHeap e Heapfy. Deapyfy si può definire come un ciclo while.

b) Quicksort è stabile? **NO**: esepio:

A=[2*,2,1] -->(Pertition(A,1,3))--> A[1,2,2*] -->(Pertition(A,2,3))--> A[1,2,2*]

c) Nessuno tra InsertionSort, MergeSort, HeapSort è stabile? **NO**: InsertionSort e MergeSort sono stabili

d) InsertionSort è $\Omega(n)^2$ nel caso peggiore? **SI**

InsertionSort nel caso peggiore è $\theta(n^2)$ (vettore ordinato in modo decrescente)

e) InsertionSort è in place? **SI**: InsertionSort usa solo variabili ausiliarie per gli indici di cicli (ciclo for e ciclo while) per memorizzare temporaneamente un elemento del vettore A(i,j,key).

f) HeapSort nel caso peggiore è $\Omega(n^2)$? **NO**: HeapSort nel caso peggiore è $O(n \log n)$

ES 2:

A matrice mxk $n = m \cdot k$ Ogni riga di A ha tutti elementi << di quelli della rige successive

a) Descrivere formalmente

			l	z	
i →			*		
v					
j →				*	

$\forall i, j, l, z (i < j \rightarrow A[i, l] \leq A[j, z])$

Ordino righe e le stampo:

```
Matrix_Row_Sort(A,m,k){
    for(i<-1 to m){
        B = a[i][1:k]
        HeapSort(B)
        for(j<-i to k){
            Print(B[j])
        }
    }
}
```

Questo algoritmo ha costo $\theta(m \cdot k \log k) \leftarrow \theta(m \log k)$

Se la matrice è ordinata sulle colonne:

in ogni istante tutti i merge lavorano su ogni colonna, ottenendo un costo di $(H)(m \cdot k) \cdot \theta(\log k)$

b) se k è costante rispetto a n:

Se k $= \theta(n) \implies O(n)$

Se k $= 0(log_n)\ O(m \cdot log_n \cdot log_{log_n})$

Es.3:

$$T(n) = \begin{cases} \theta(1) & \text{se } n \leq 1 \\ 2T\left(\frac{n}{4}\right) + \theta(\sqrt{n}) & \text{se } n > 1 \end{cases}$$

semplificato:

$$T(n) = \begin{cases} a & n \leq 1 \\ 2T(\frac{n}{4}) + b(\sqrt{n}) & n > 1 \end{cases}$$

$$T(n) = \sum_{i=0}^{log_n n=1} b\sqrt{n} + a \cdot 2^{\log_4 n} = b \cdot \sqrt{n} \cdot log_4 n + a \cdot n^{\log_4 2} = \theta(\sqrt{n} \cdot \log n)$$

```
Ricorsive(n){
    if(n<1){
        Ricorsiva(n/4)
        Ricorsiva(n/4)
        for(i<-1 to sqrt(n)){
            print("ciao")
    }
}
}
```

Es5:

T tabelle di Hesh |T|=m open addressing scansione lineare (peso 1)

key e [1. M]

a) h,(key,i)=(key + i) mod m b) Si cancella usando DEL. Durante la ricerca ci si ferma al primo NIL mentre proseguo se trovo DEL

```
Search(T, x, h){
    i<-0
    j<-h(x.key,i)
    count<-0
    while(i<m && t[j] != x && count < 2){
        if (T[i] == NIL) count<-1
        i <- i+1
        j<-h(x.key, i)
    }
    if(T[i]==x){
        return j
    }else{
        return "non trovato"
    }
}
```

nel caso peggiore: $\theta(m)$ nel caso medio con hasking uniforme: $\theta(1)$ nel caso migliore: $\theta(1)$

Quindi con h sono minore lineare