AVL Tree

Saturday, 21 August 2021 7:06 PM

Adelson -Velsky and Landis → self Balancing BST

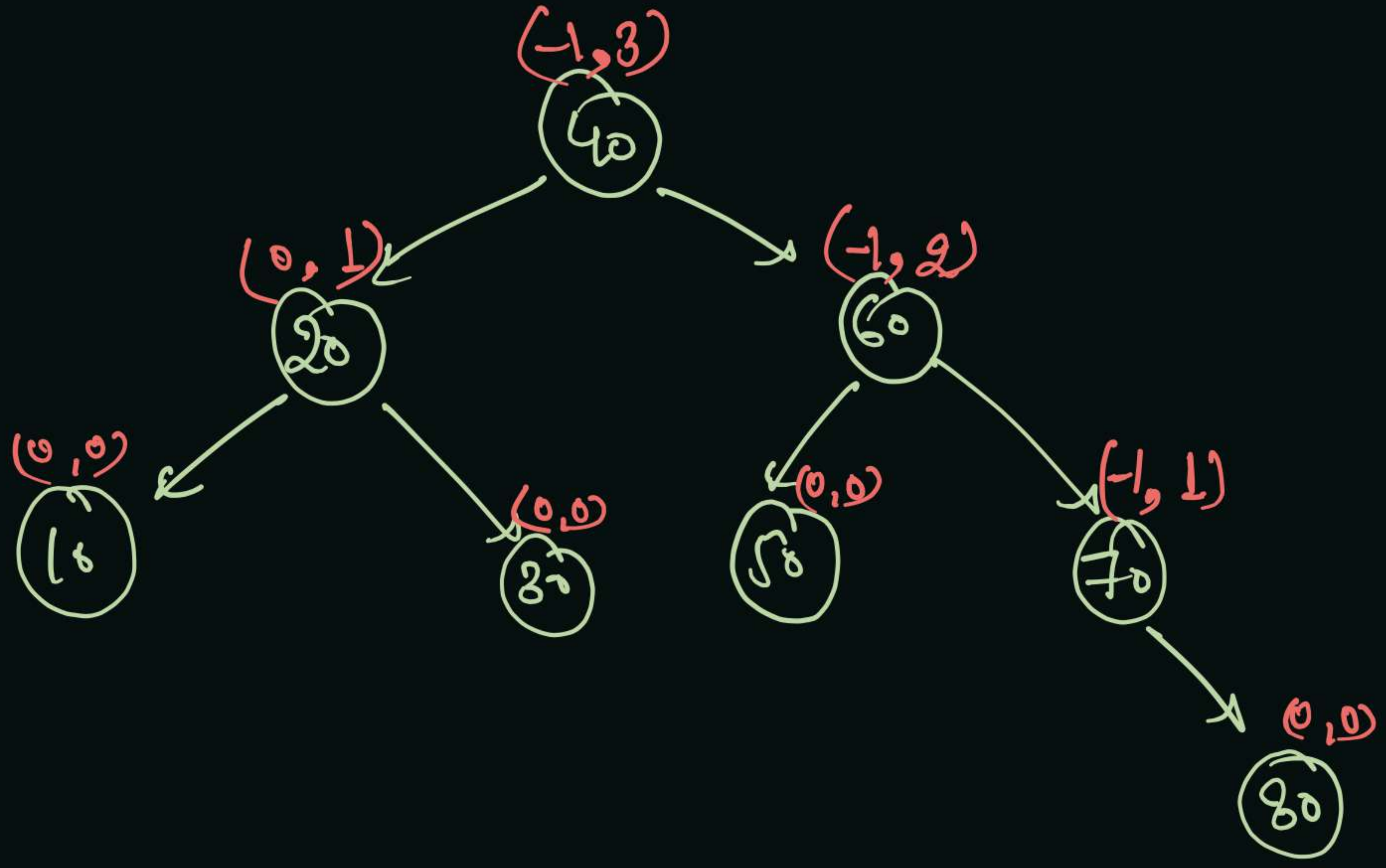
BST → data → 10, 20, 30, 40, 50, 60, 70, 80
(indices: 0, 1, 2, 3, 4, 5, 6, 7)

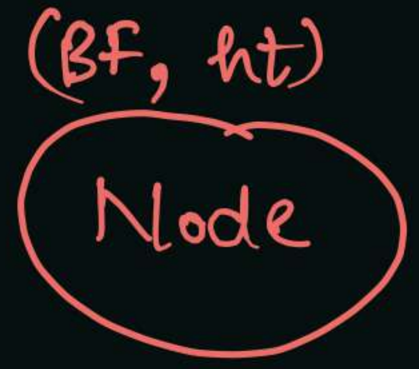
construction → Balanced

Balancing factor = |lh - rh| → absolute

lh → left height, rh → right height

0 ≤ Balancing factor ≤ 1

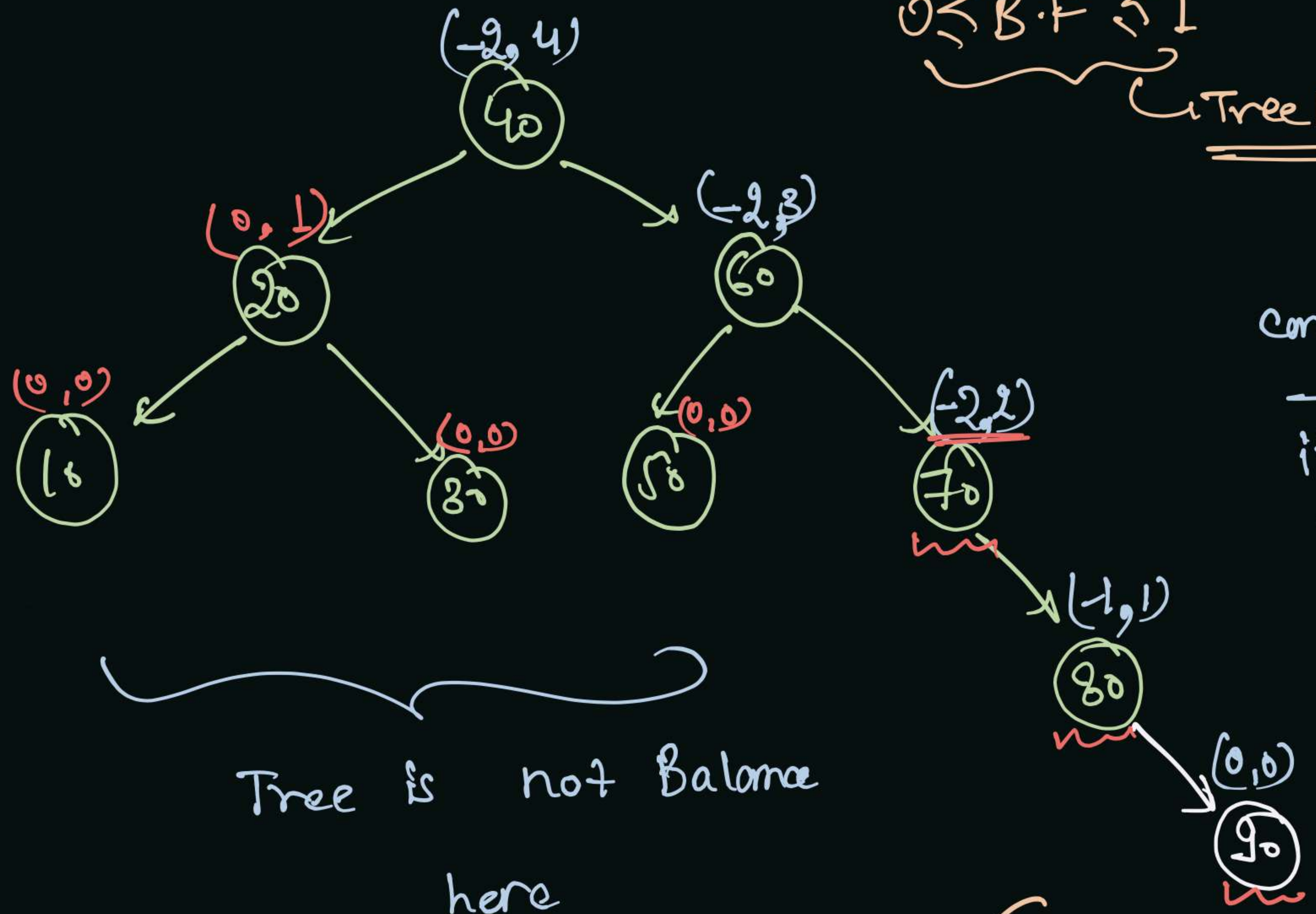




Balancing factor is fine on Every node.

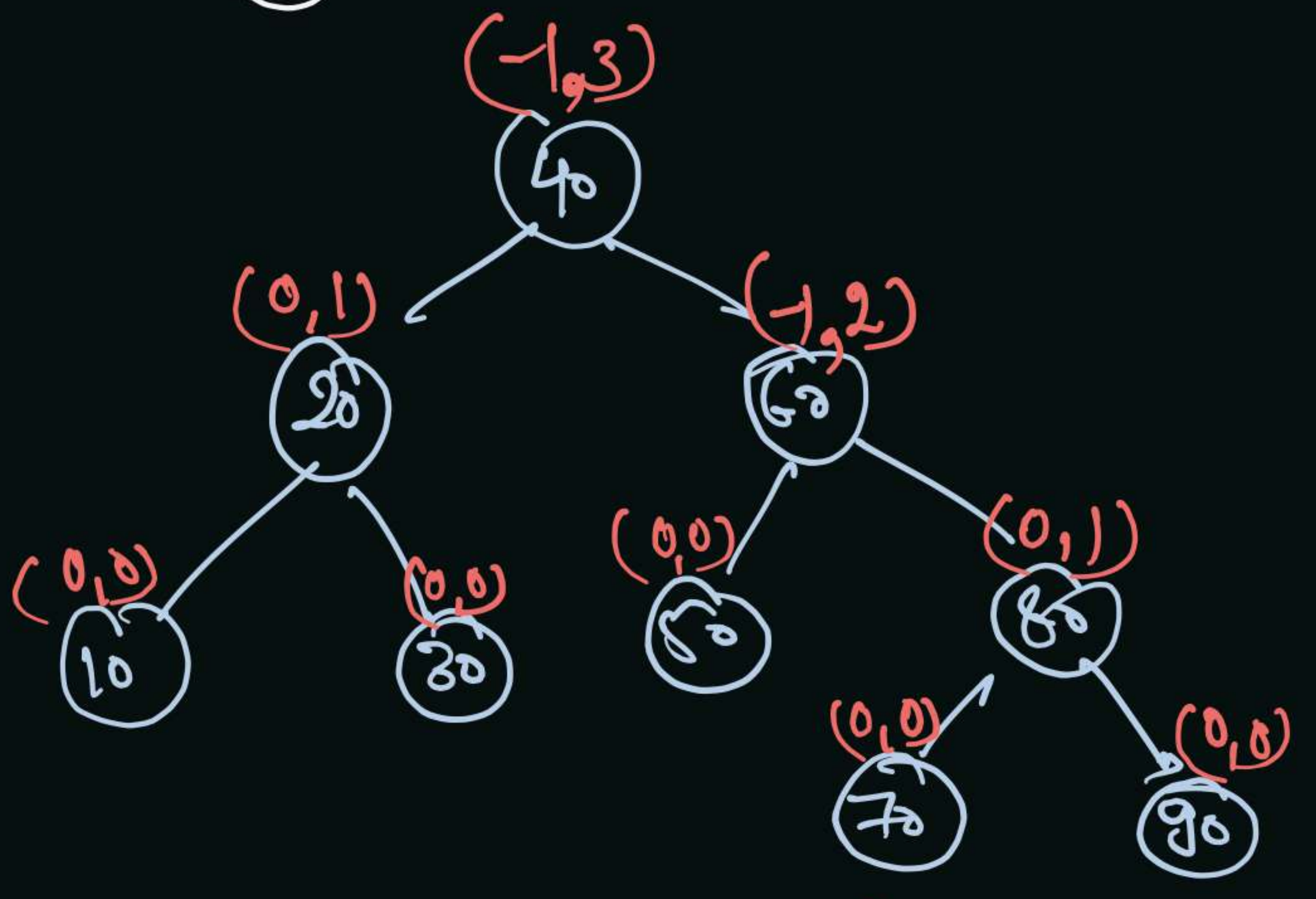
ddd new data → 90

$0 \le B.F \le 1$ ← Tree Balance



Tree is not Balance here

can we make → it balanced Tree again with some dataset



Tree is again Balanced

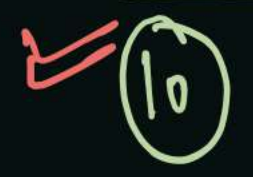
Changes Required to make it balance again

why we are doing it ?? Significance of AVL

Why we are doing it? significance of AVL.

operation on BST.

Root



Initial BST

complexity for searching in BST.

↪ O(h)

h= logn? how height is logn ??

add - 20

add - 30

add-40

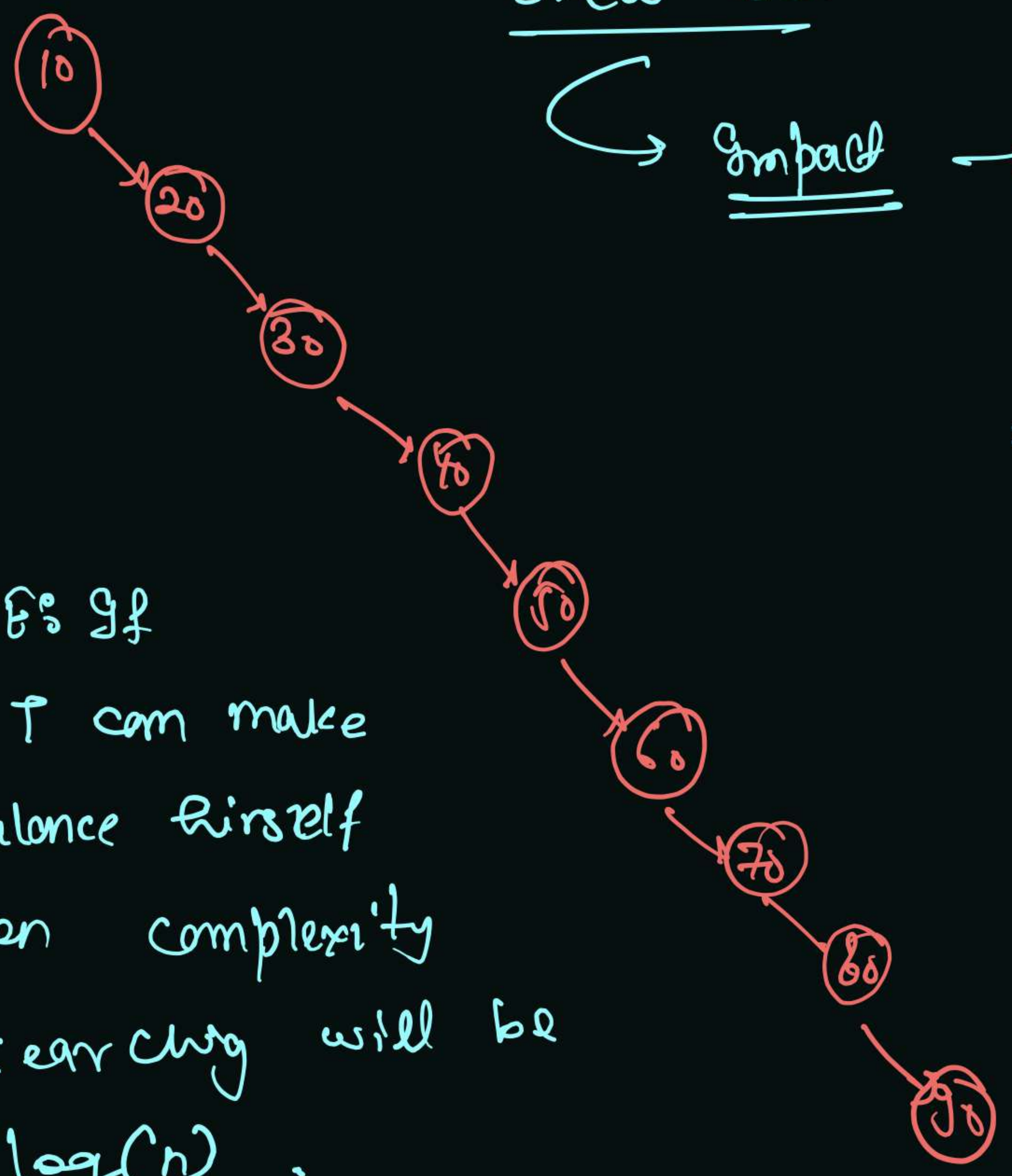
add- 50

add - 60

add- 70

add- 80

ada - 90



NOTES If BST can make balance hirself then complexity for searching will be log(n) .

Skew BST

↪ Impact → searching

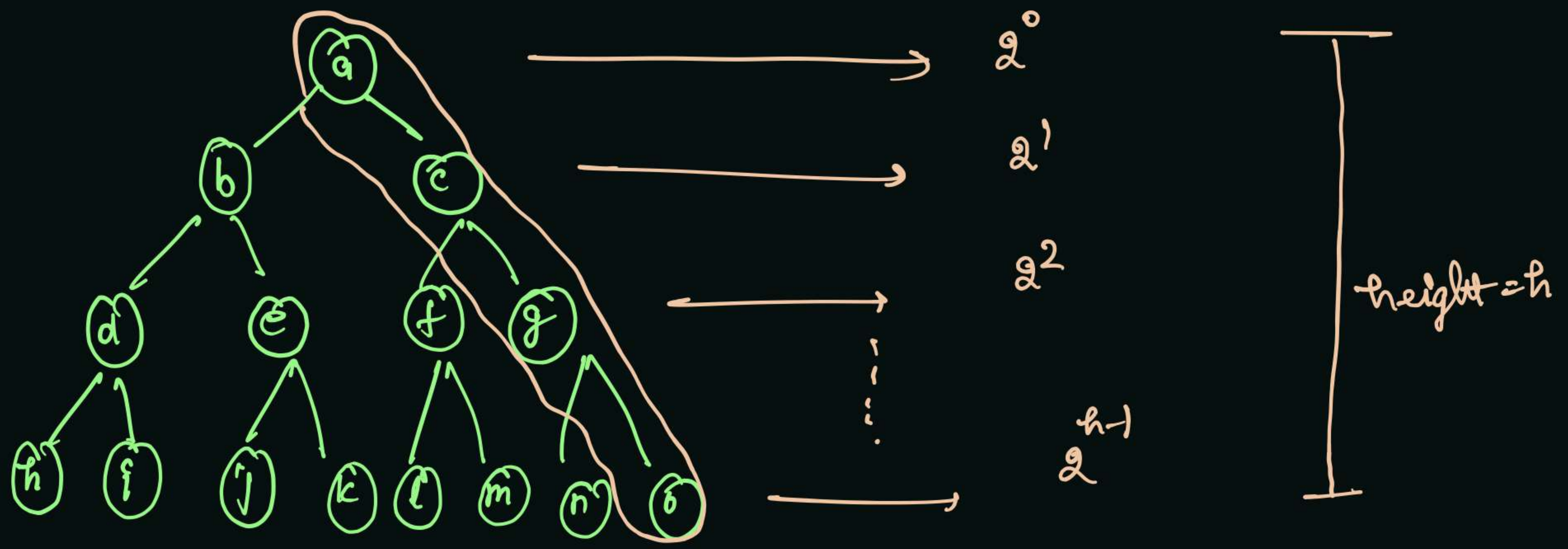
↓

O(h) time comple-

worst cas

linear search-

Explamation of height is BST. (Balanced BST) Total no. of nodes = n



we traverse single Segment in BST so searching in BST will be of complexity - $\underline{\log n}$

Total no. of nodes $= \overset{1}{2^0} + 2^1 + 2^2 + 2^3 + \cdots + 2^{h-1}$

GP → $a + ar + ar^2 + ar^3 + \cdots + ar^{n-1}$

$$\text{Sum} = \frac{a(r^n - 1)}{r-1} \qquad \underline{r > 1}$$

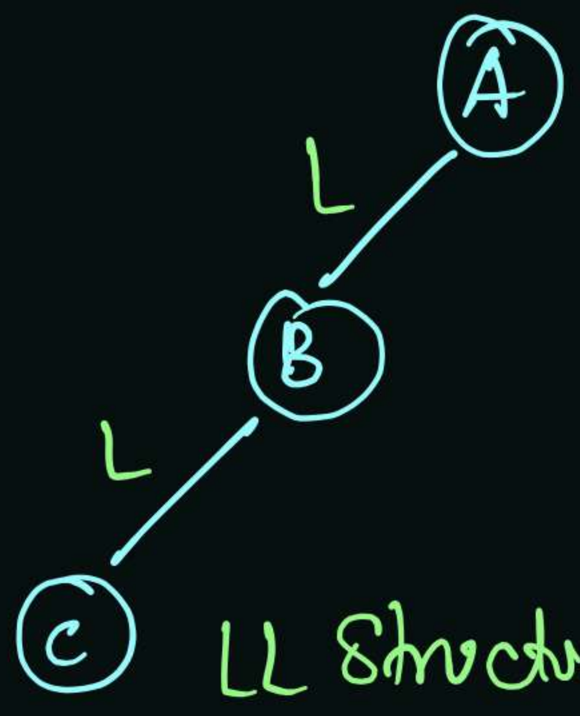
$a = 1$
$r = 2$
$n = h.$

$$n = \frac{1(2^h - 1)}{2-1} \Rightarrow n = 2^h - 1$$

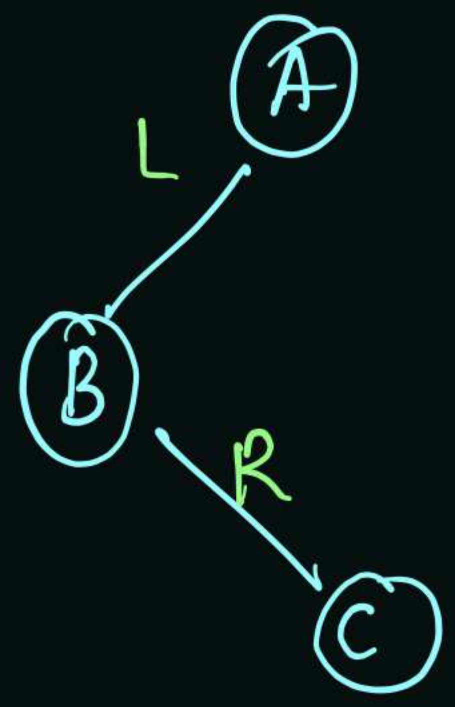
$$\Rightarrow n+1 = 2^h$$ take $\log_2$ both side

$$\log_2(n+1) = \log_2 2^h \Rightarrow h = \log(n+1) \rightarrow \text{order} \Rightarrow \boxed{h = \log(n)} \leftarrow$$

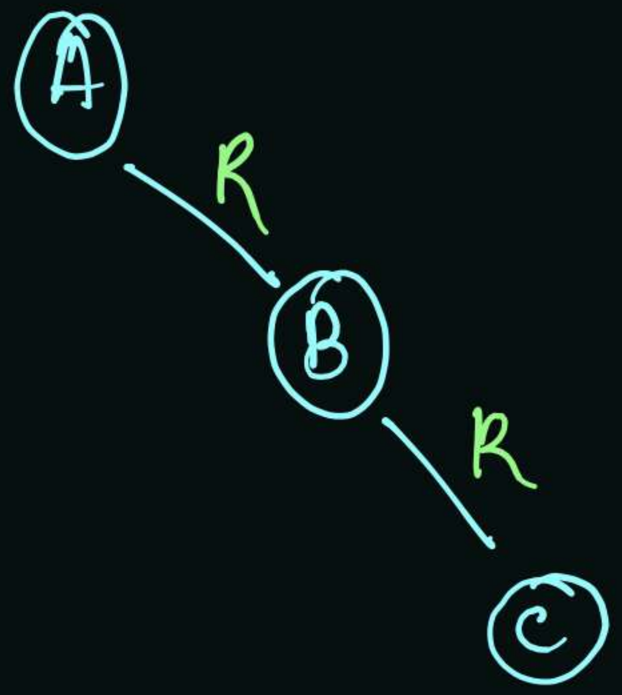
Structures of BST which are responsible for unbalance tree:-



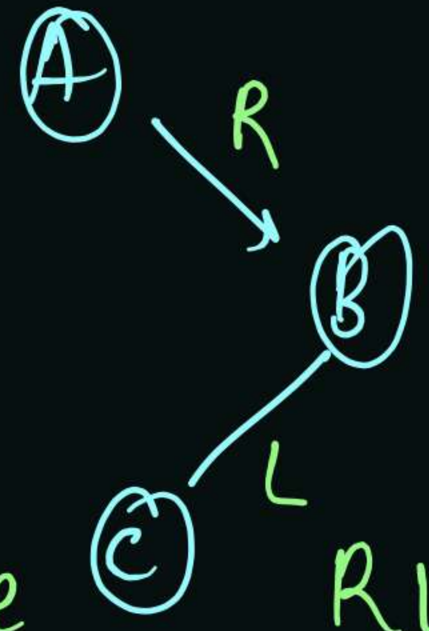
LL Structure



LR Structure

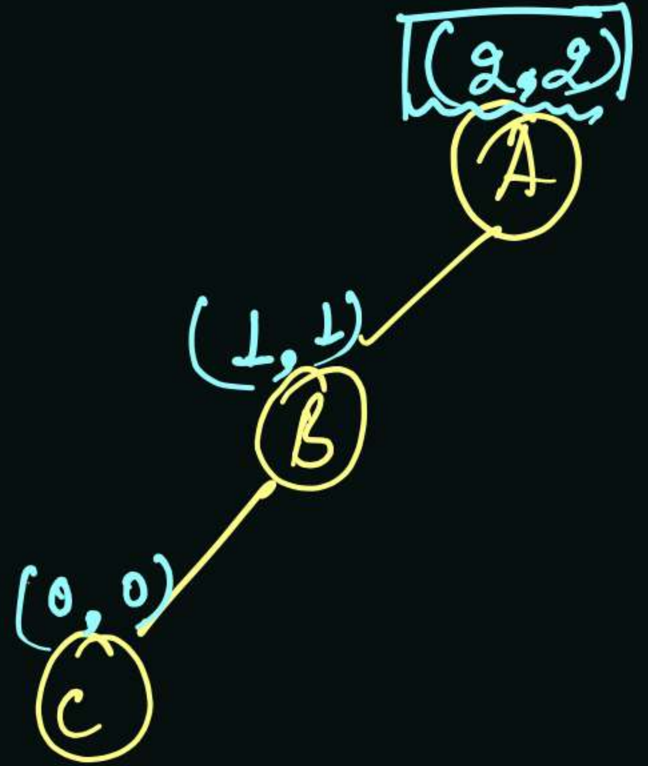


RR Structure



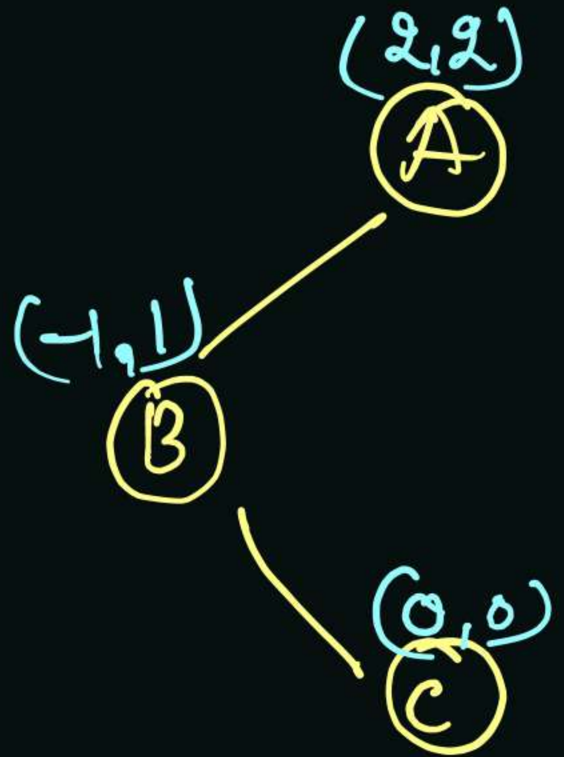
RL structure

LL Structure



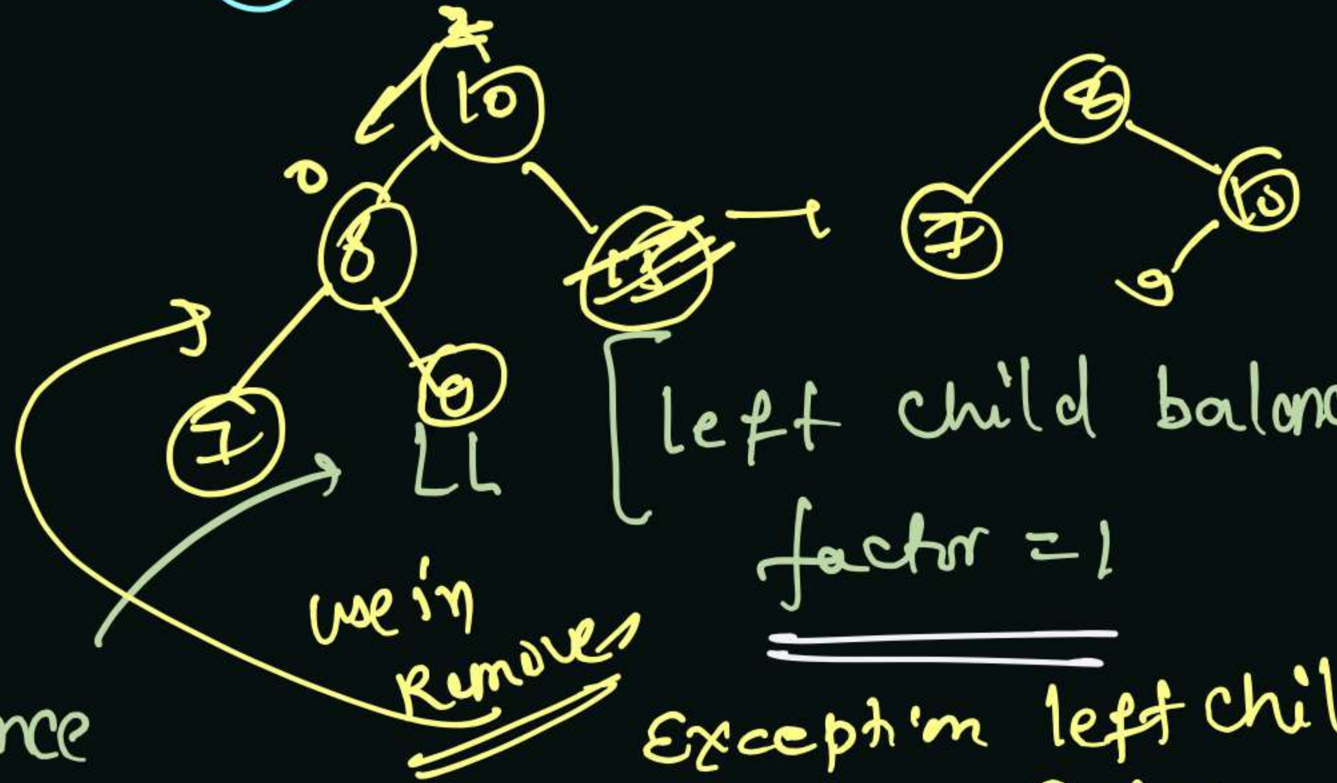
(B.F, height)

LR Structure



(B.F. height)

If Balance factor = 2

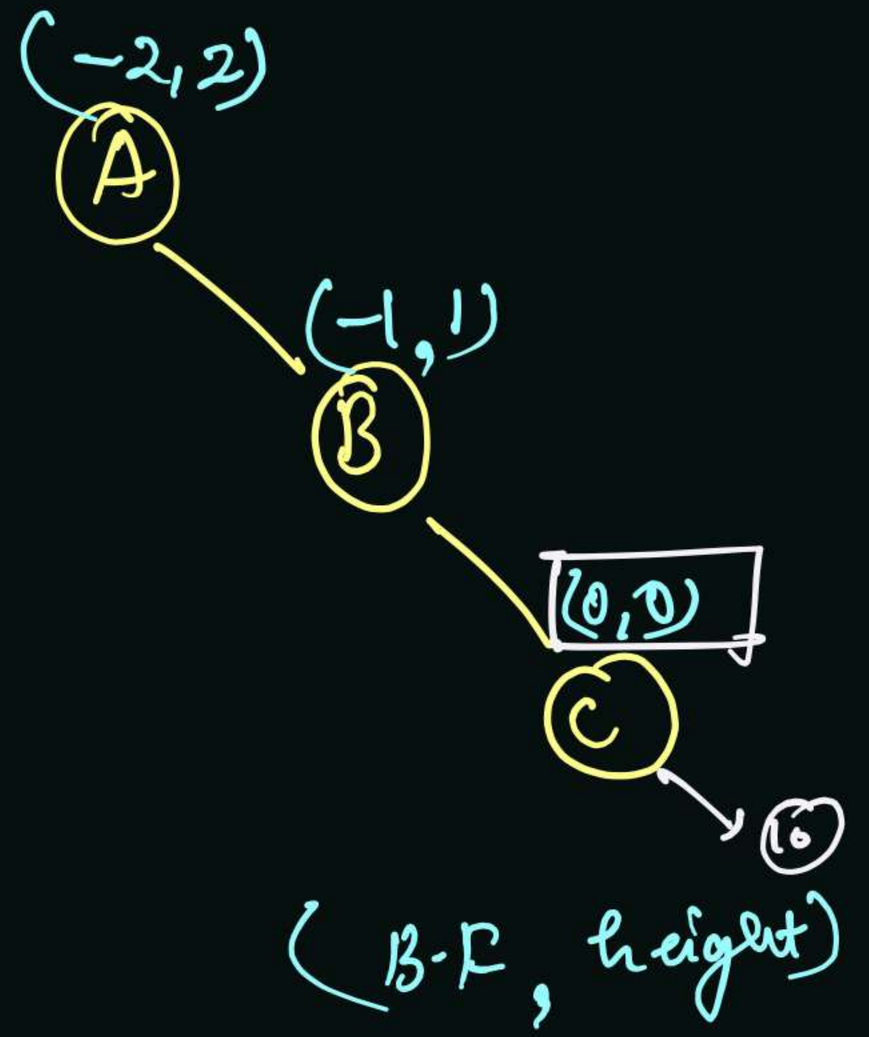


[left child balancy factor = 1]

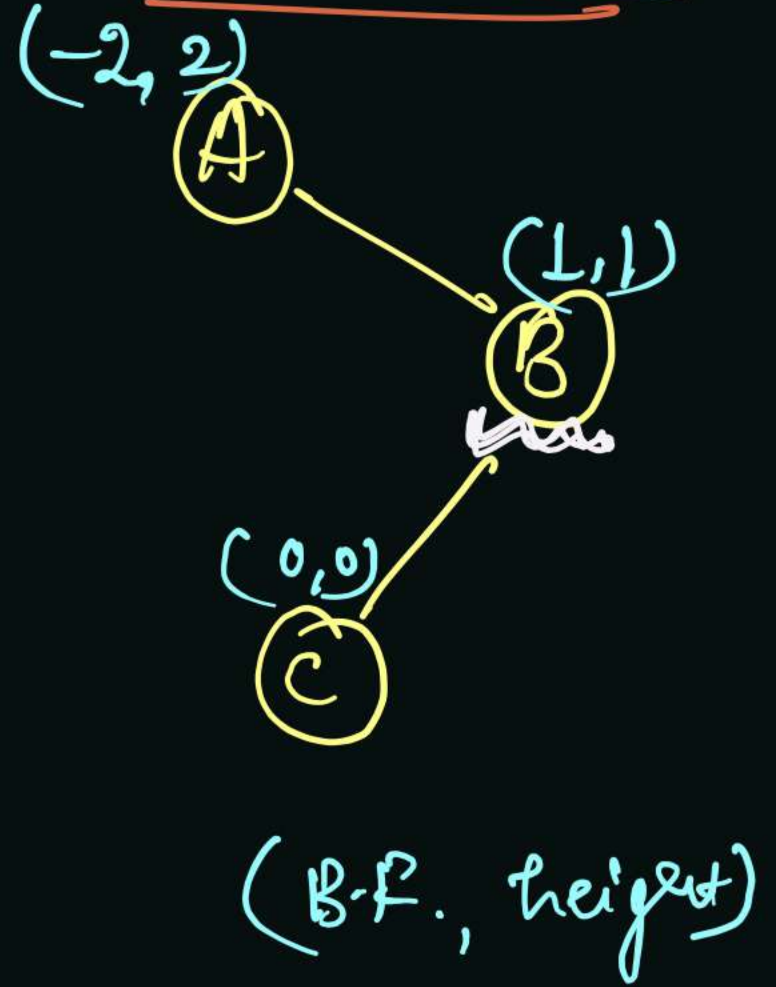
Exception left child balancy factor = 0

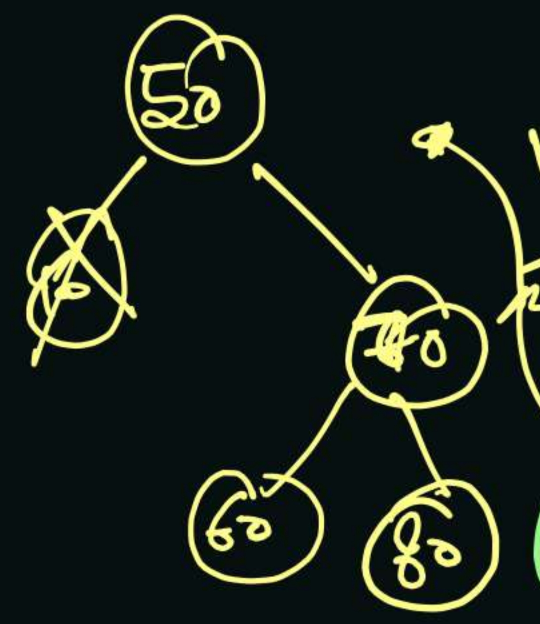
LR [left child balancy factor = -1]

RR Structure



RL Structure





use in Remove

If Balancing factor == -2

RR [if Right child have Balancing factor = -1
right child B.factor == 0]

RL [if Right child have B. Factor = 1]

Way of Implementation.

Node → int data
Node left
Node right

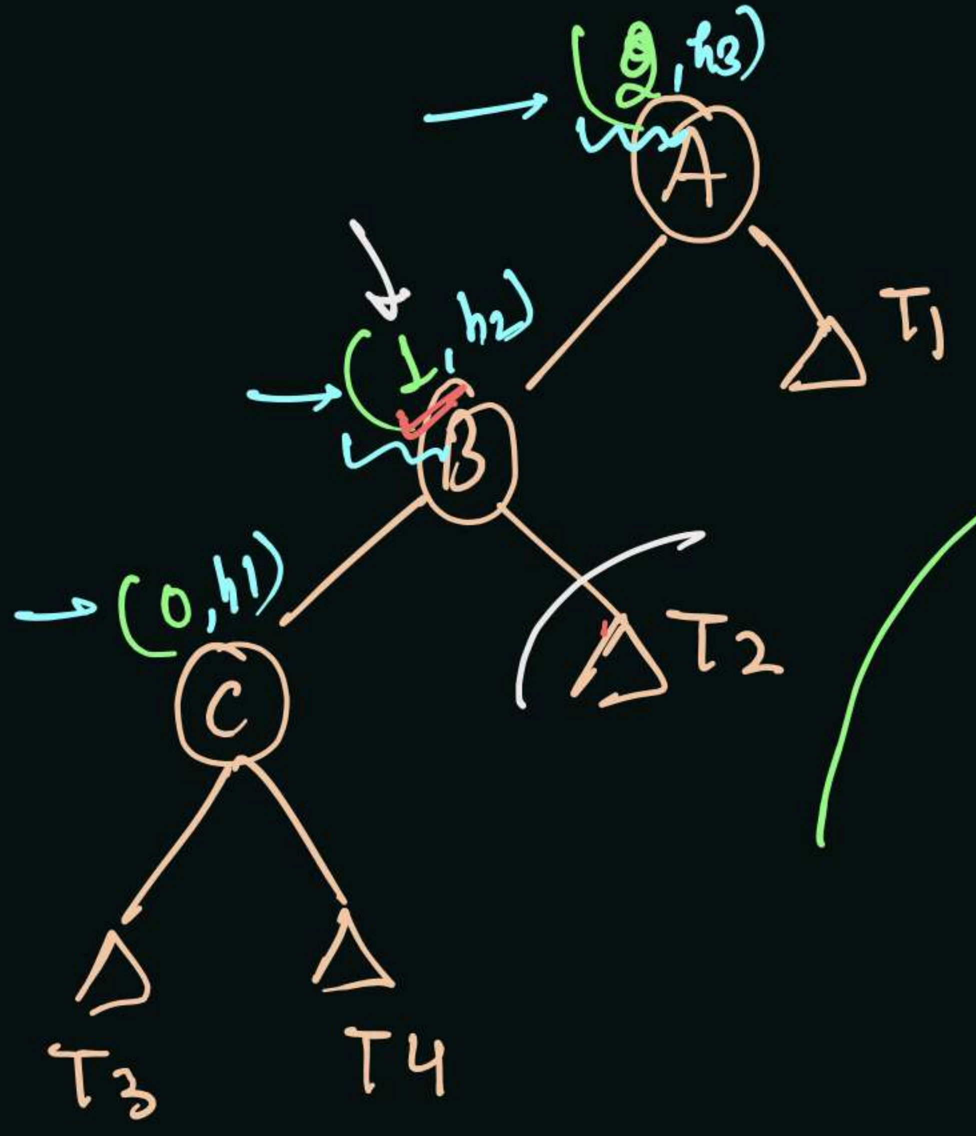
int balance
int height

→ // getRotation //

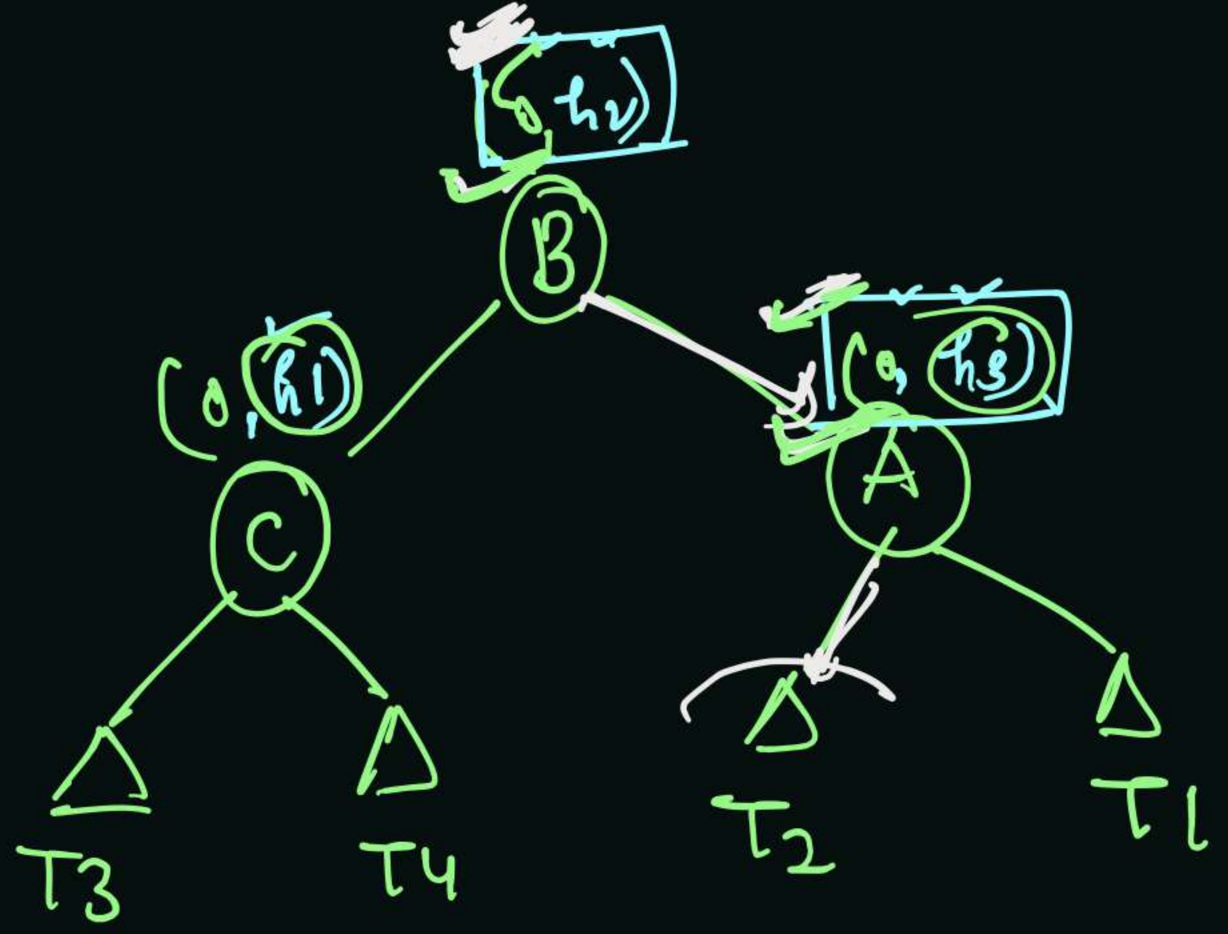
→ update height And balance

→ solve for structure

LL Structure → Right Rotation



Right Rotation on A node.



Root is changing After Rotation so Return new Root for calling function.

// 1. make a proper connection

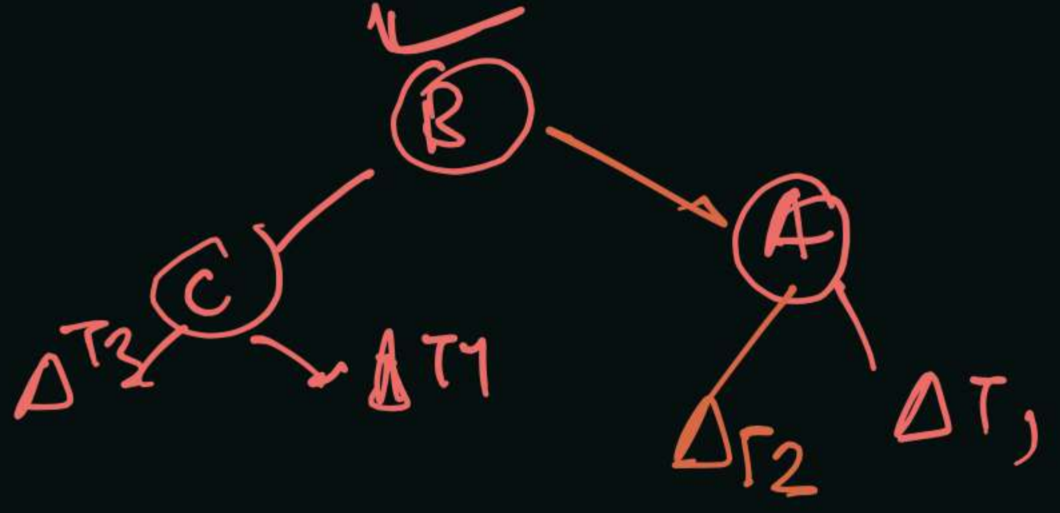
order matters

// 2. correct height and balance for A then B

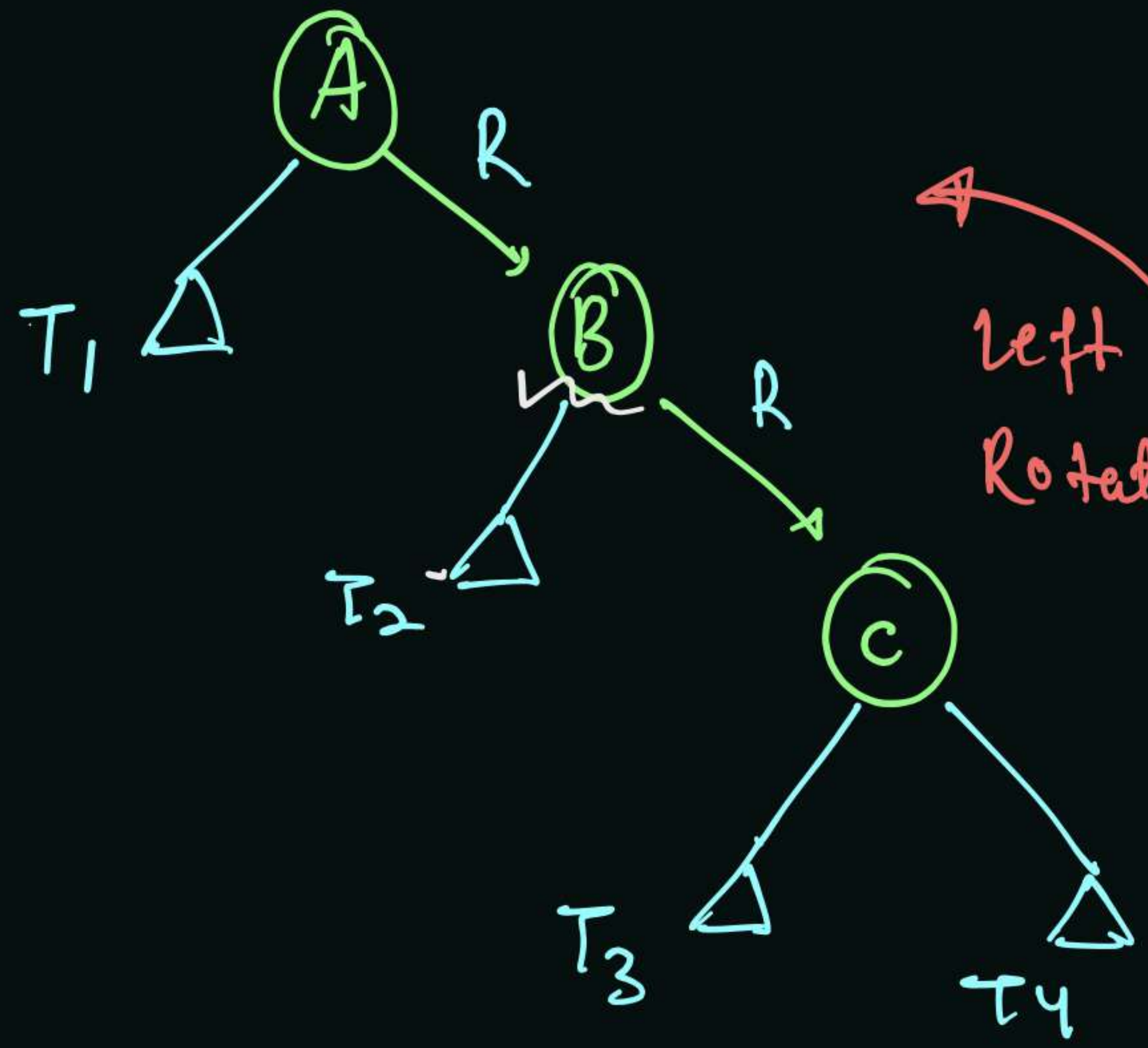
```
Node B = A.left
A.left = B.right
B.right = A
return B; // new Root
```

update height and Balance A

LL LL B

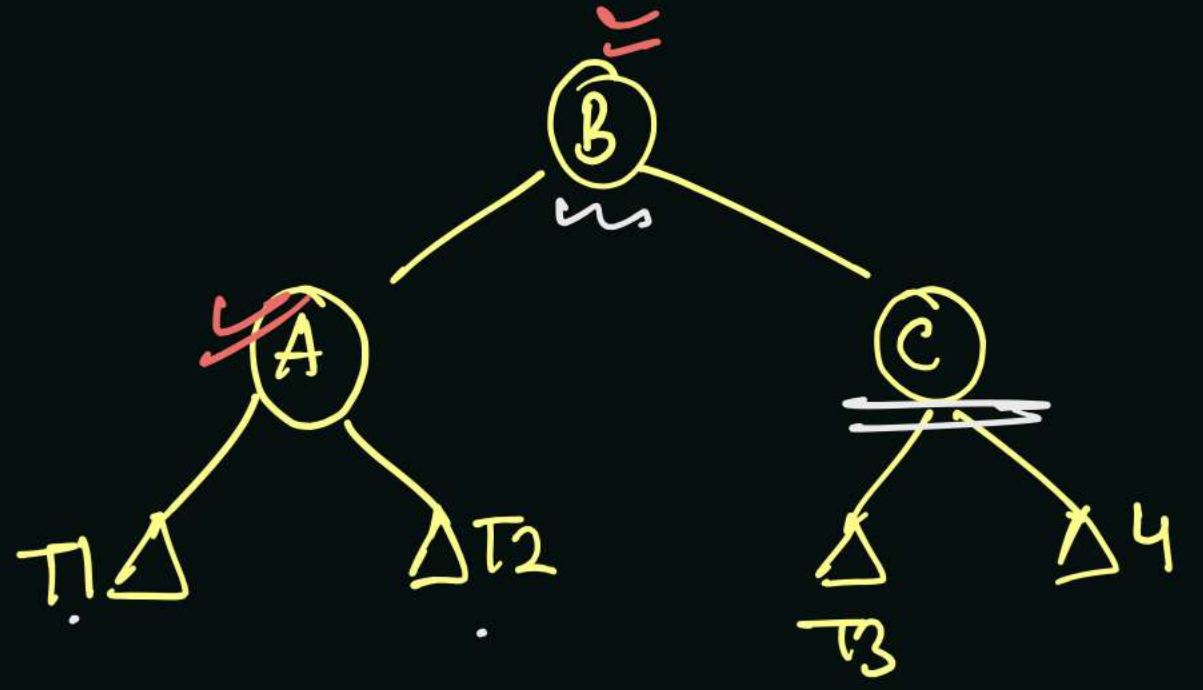


RR Structure



Left Rotation

Left Rotation on A node



Node B = A.right

A.right = B.left

B.left = A

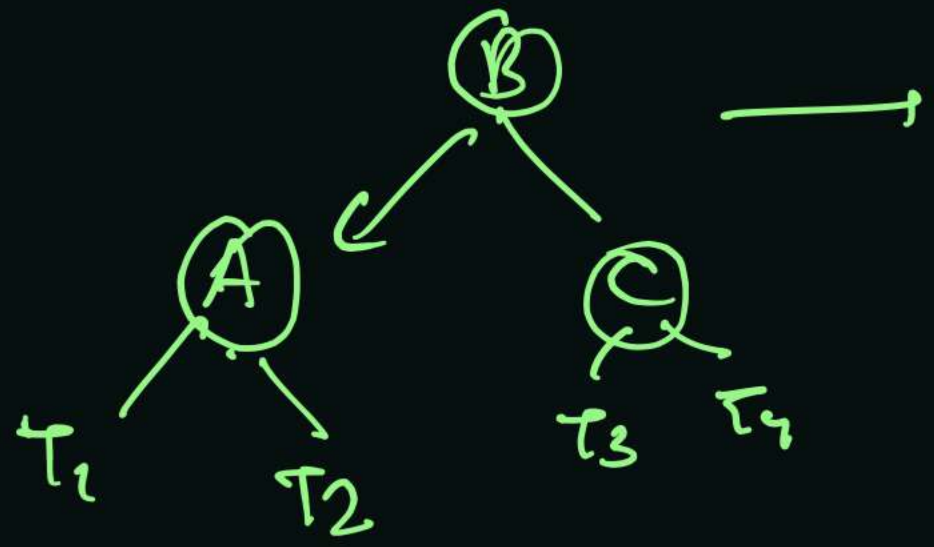
return B

// Step1 → Make a proper connect

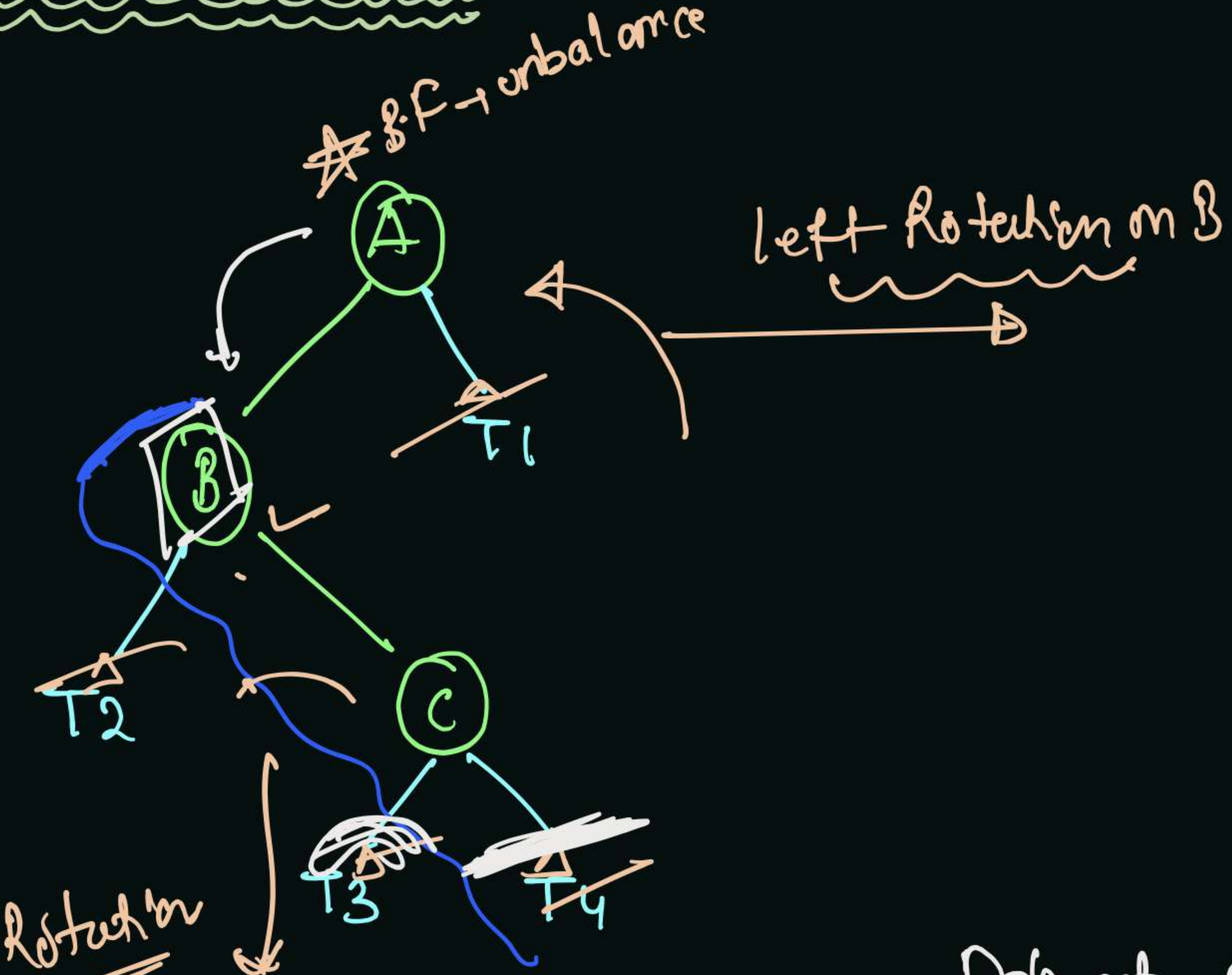
// Step2. → Maintain height and Balancing factor

Update height And Balce. A

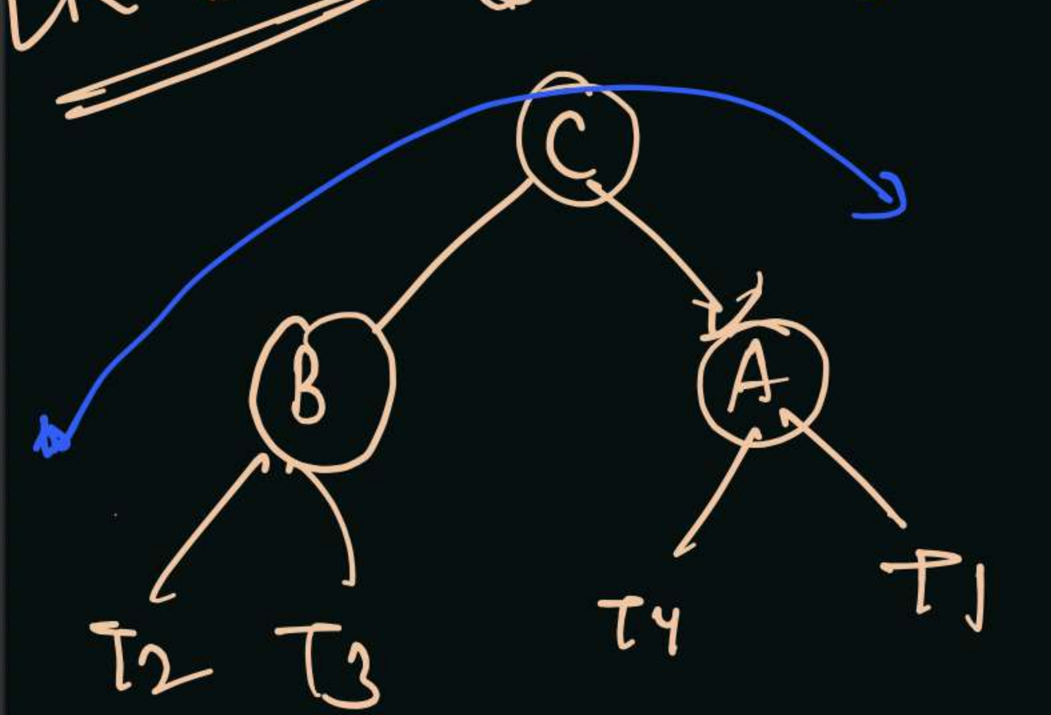
" " " B



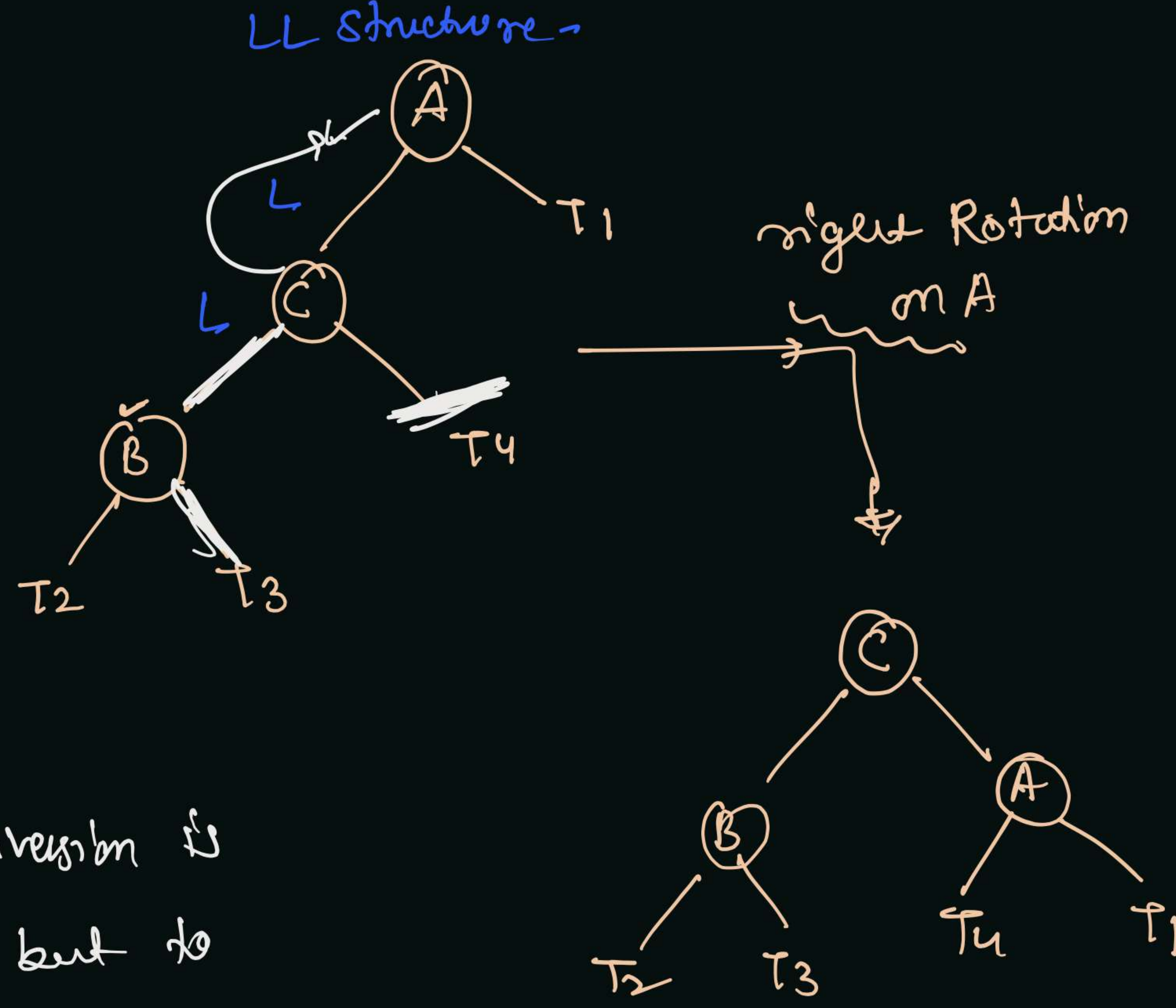
LR Structure



LR Rotation

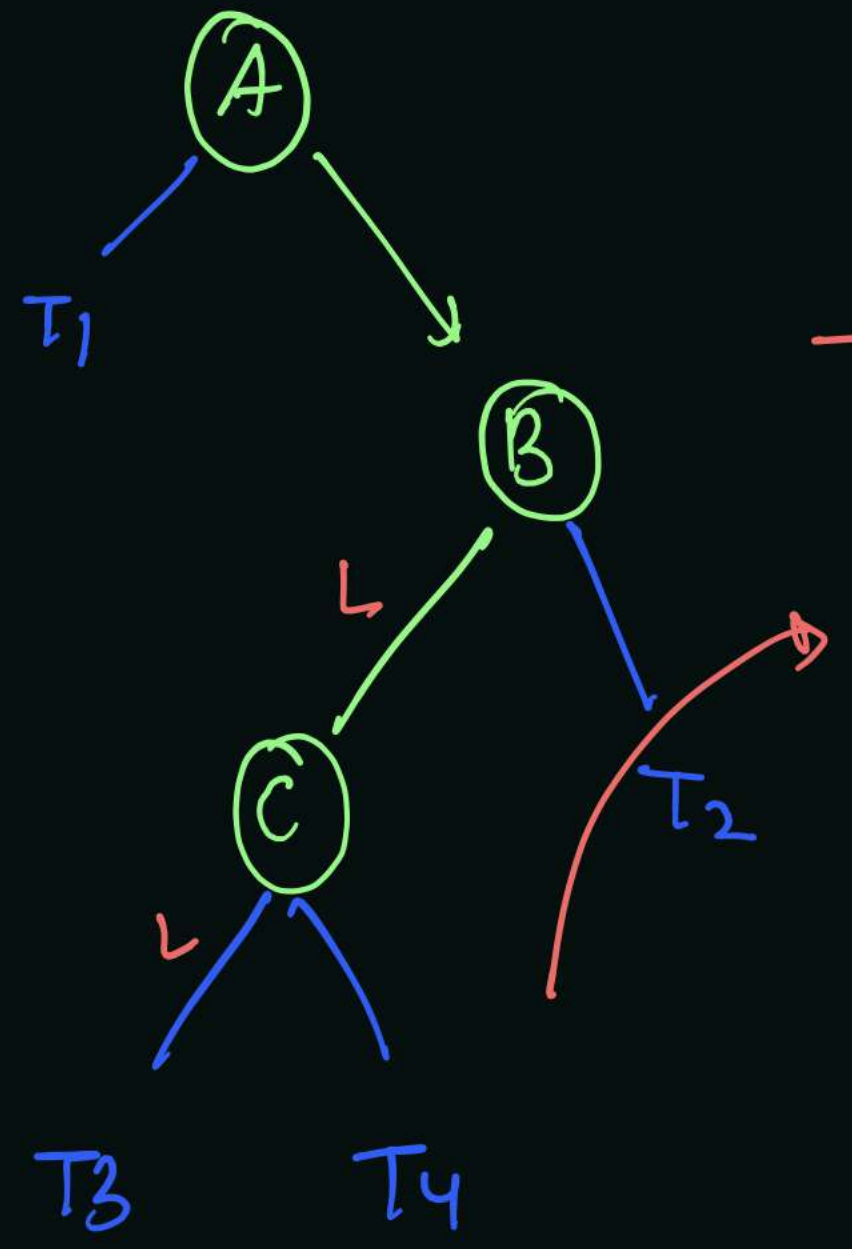


LL Structure-

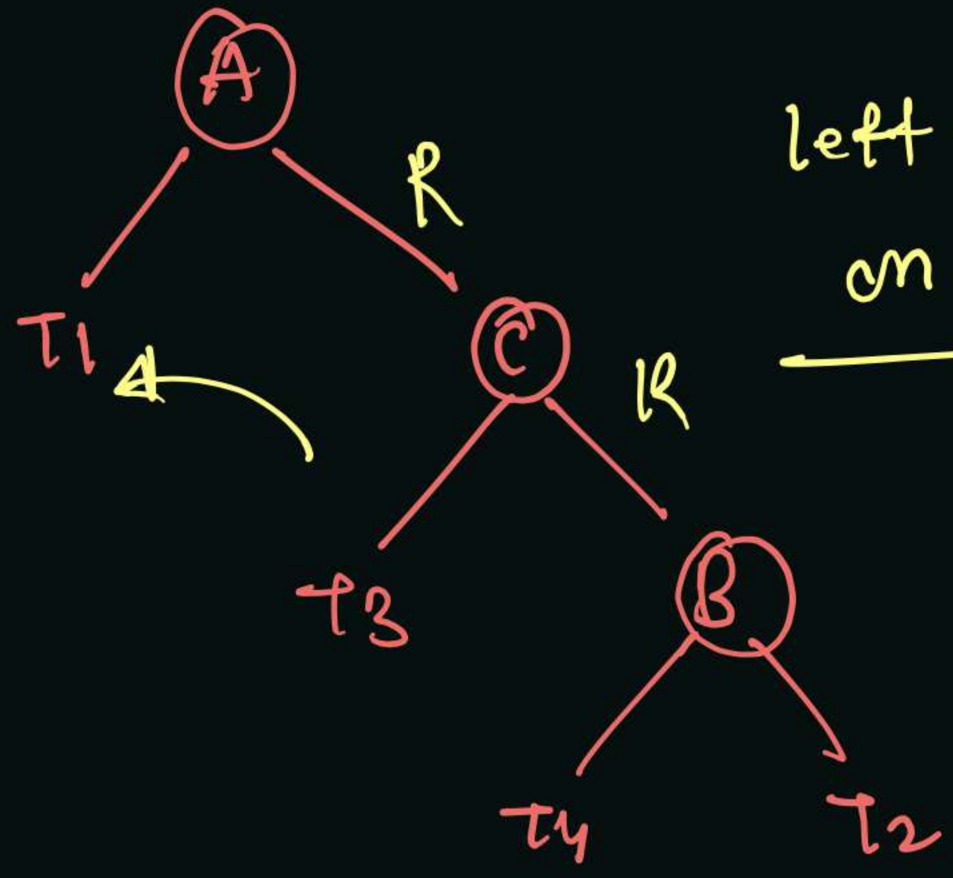


Direct conversion is possible, but to reuse code.

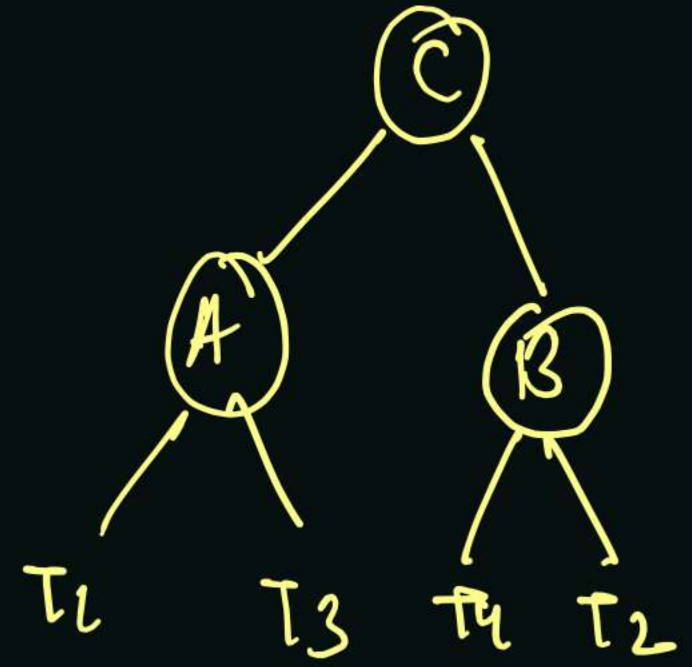
RL Structre



Right rotation on B →



left Rotation on A →



node.right = Right Rotation on node.right

return leftRotation on node.

Display →

```
A   ← C →   B
T1  ← A →  T3
.   ← T1 →  .
.   ← T3 →  .
T4  ← B →  T2
.   ← T4 →  .
.   ← T2 →  .
```