Emirhan Uzun
171044019

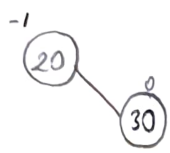
HOMEWORK #7 QUESTION 1

AVL Tree Insertion

Insert 20

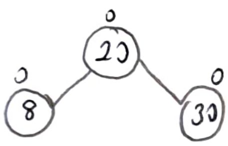
- Add to root

Insert 30



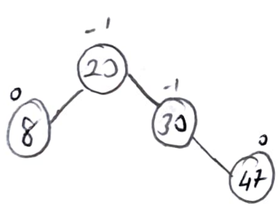
- 30>20
- Add root's right child

Insert 8



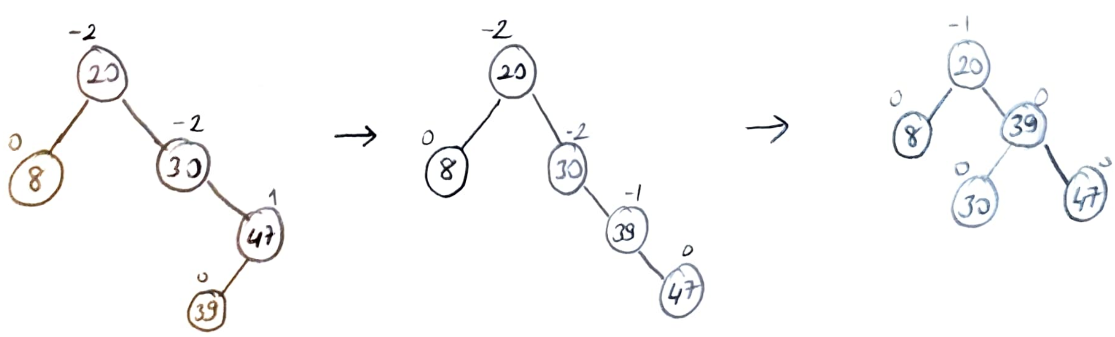
- 8<20
- Add root's left child

Insert 47



- Add 30's right child
- All nodes still between [-1,1]

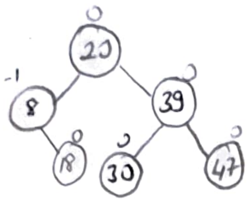
Insert 39



This tree is not balanced and heavy for RL rotation. So I have to rotate right first, and then rotate left for balance ([-1,1])

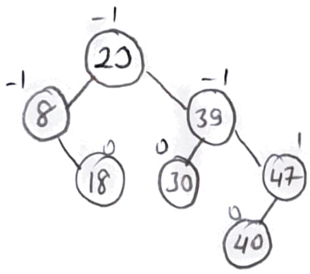
Emirhan Uzun
171044019

Insert 18



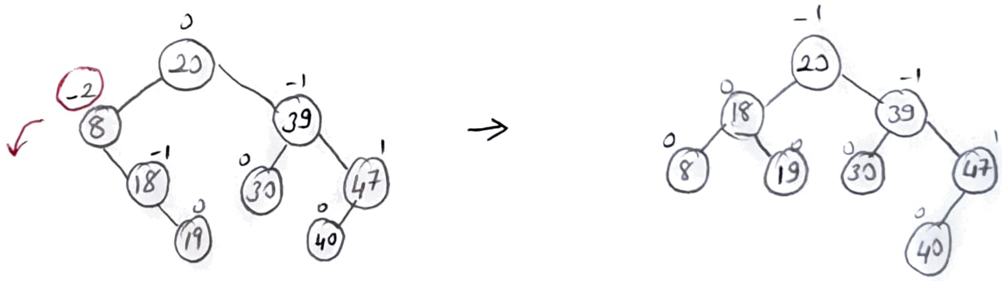
- All nodes heavy is between [-1,1]. So it is not need to rotate.

Insert 40



- Tree is balanced.

Insert 19

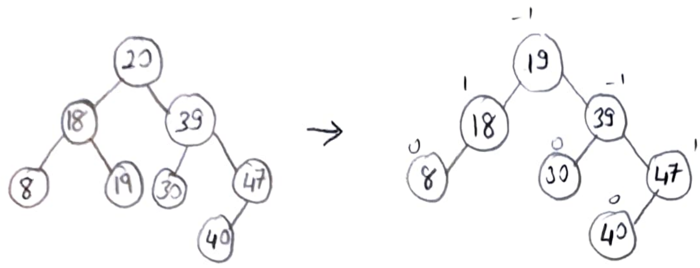


Node 8's heavy is not between [-1,1]. The heavy is right-right rotation. So I have to rotate left rotation.

Emirhan Uzun
171044019

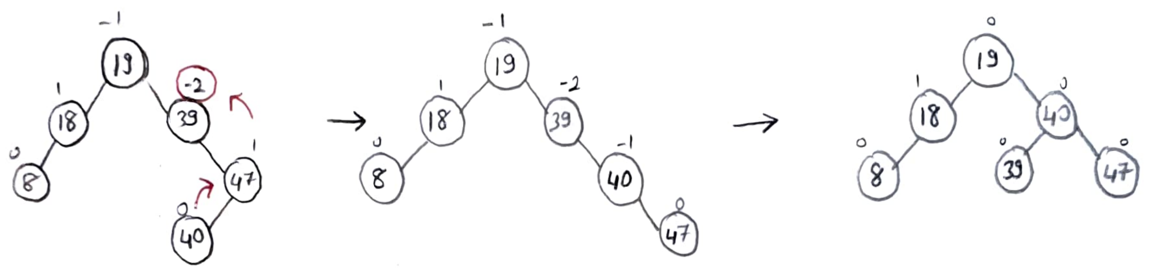
AVL Tree Deletion

Remove 20



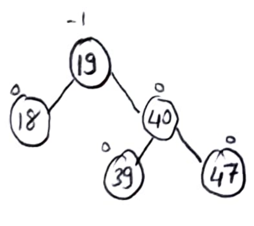
20 has 2 child
So I can't just remove it.
Change with biggest element of
left subtree. Tree is balanced

Remove 30



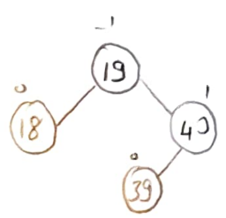
Firstly, I can remove 30 directly because it is a leaf node. After that, the tree's heavy is in LR (Left-Right) rotation. So I should rotate first right and then left. Tree is balanced.

Remove 8



8 is a leaf node, I can remove it.
Also tree is still balanced.

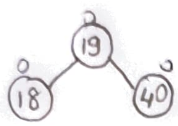
Remove 47



47 is a leaf node, I can remove it.
Tree is still balanced.

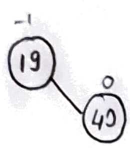
Emirhan Uzun
171044019

Remove 39



39 is a leaf node
Tree is balanced

Remove 18



18 is a leaf node
Tree is balanced.

Remove 40

40 is a leaf node
Tree is balanced.

Remove 19

X

19 is a leaf node
Tree is empty

PS: I used $|h_R - h_L|$ to find heavies of trees.

Emirhan Uzun
171044019

Red-Black Tree Insertion

Insert 20

The new inserted node always be red
But root must be black
Change color.

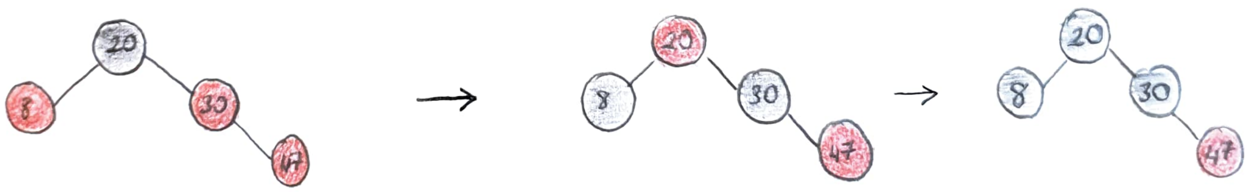
Insert 30

New node is red
Tree is balanced

Insert 8

New node is red
Tree is balanced

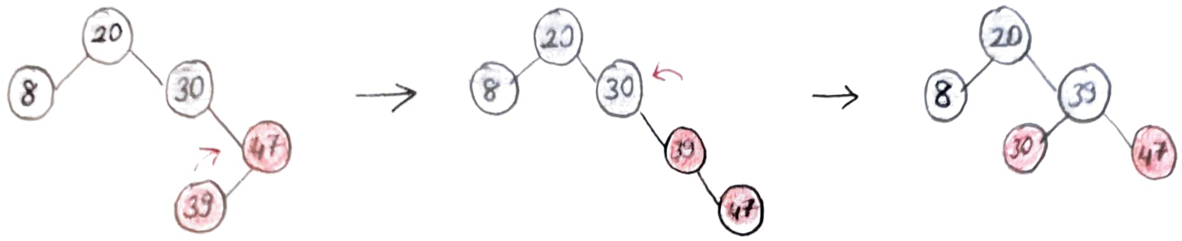
Insert 47



New node is red. But red's cannot be red child. So I look the uncle of new node which is 8. If it is red, then recolor parent, uncle and grandparent. After that root must be black. Recolor again.

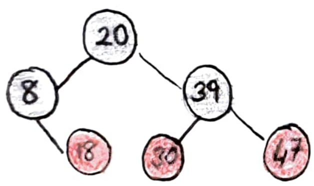
Emirhan Uzun
171044019

Insert 39



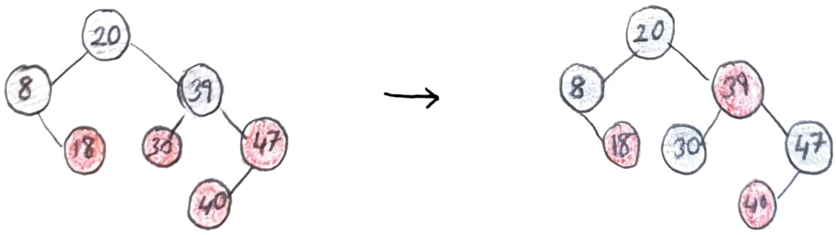
Firstly the new node is red. The red node can't be red's child. And I look uncle of this new node. The uncle doesn't exist. So we rotate opposite of heavy. The heavy is RL rotation. So I have to rotate first right and then left. Now the black node number from all path is equal. Tree is balanced.

Insert 18



The new node is red
Tree is balanced.

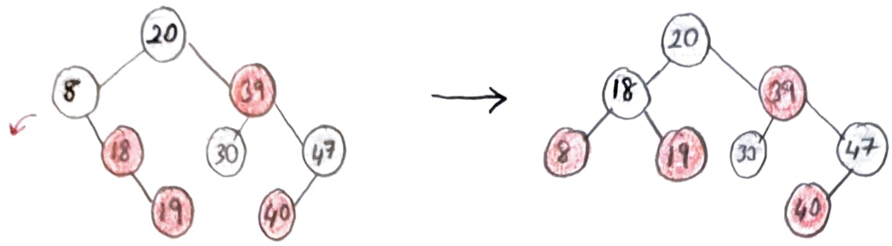
Insert 40



New node is red
I look its uncle (30)
It is red so I change uncle, parent and grandparent's color.
Tree is balanced.

Emirhan Uzun
171044019

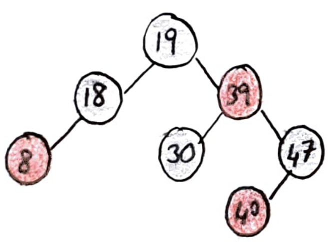
Insert 19



New node is red. Red node can't be red node's child. The heavy is right right (RR) rotation. So I have to rotate left rotation. Now all path from root to leaf, black nodes are equal. Tree is balanced

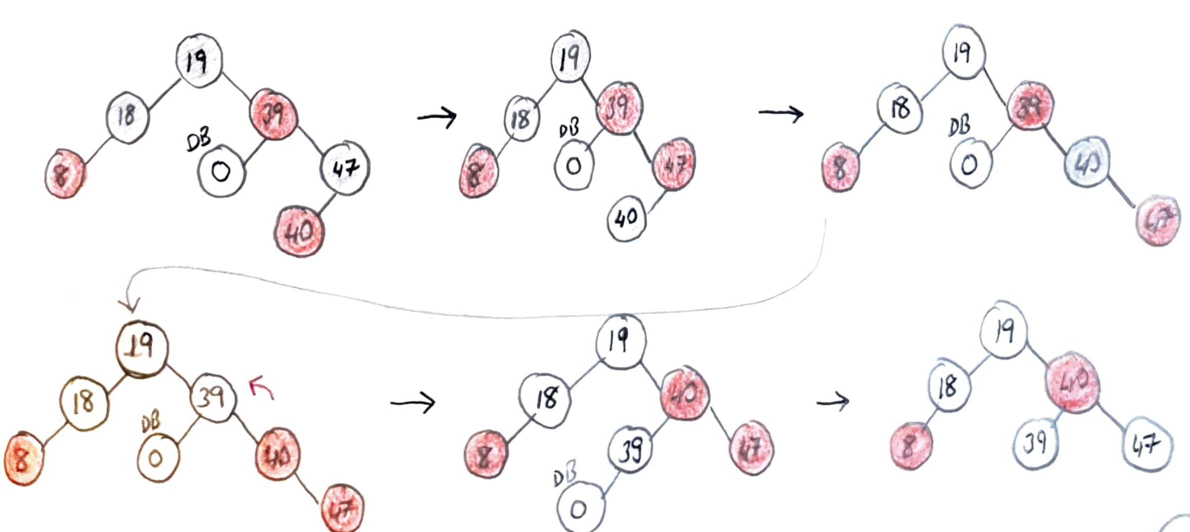
Red-Black Tree Deletion

Remove 20



Find the root 20. It has 2 child so find the biggest element of left subtree. Change the root value of that number. Remove the biggest element node in left sub. If it is red, just delete it.

Remove 30

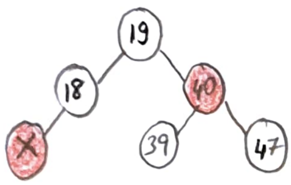


Emirhan Uzun
171044019

Firstly we find the 30 If it is black, it will be double black after remove. We look its sibling. If its sibling are black and its child which is near to DB is red, then swap colors of DB's sibling and sibling child. Rotate sibling in opposite direction to DB.

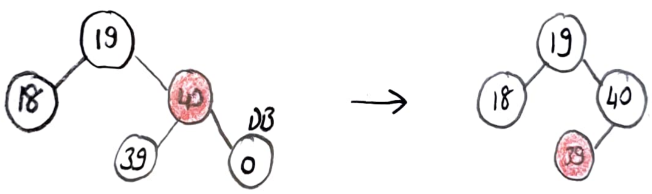
After that swap colors of parent and sibling. Rotate parent in DB's direction. Remove DB and change color of red child to black

Remove 8



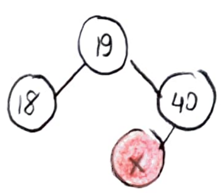
If the removed node is red, just delete it.

Remove 47



If the removed node is black, we look its sibling. If DB's sibling is black and its children are also black, we add black to DB's parent and sibling will be red

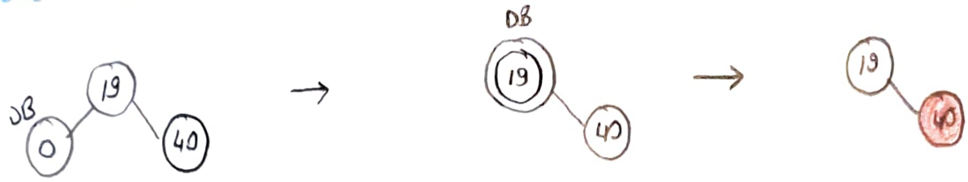
Remove 39



If the removed node is red, just delete it.

Emirhan Uzun
171044019

Remove 18



If removed node is black, it becomes double black.
If DB's sibling is black, add black to DB's parent. So parent becomes black and sibling becomes red. In root, we can remove DB and it will be just black.

Remove 40

If node is red, just delete it

Remove 19

The tree is empty.

Emirhan Uzun
171044019

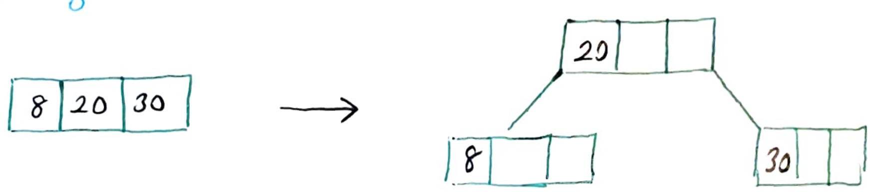
2-3 Tree Insertion

Insert 20

Insert 30

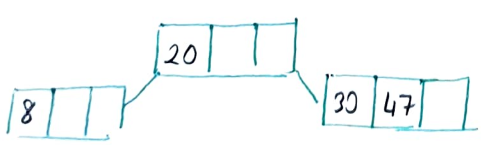
We have to add ascending order.

Insert 8



A node can't store three values. The middle value propagates up to parent. This node splits into two new 2-nodes

Insert 47



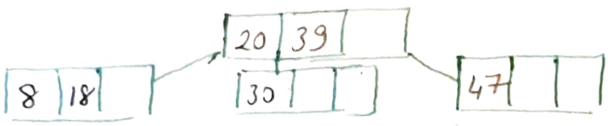
47>20 go right
47>30 put after 30

Insert 39

A node can't store 3 values. Middle value go up and this node splits into two new 2-nodes

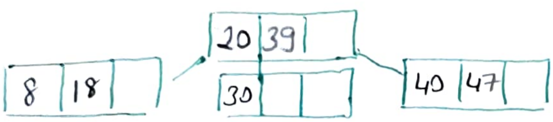
Emirhan Uzun
171044019

Insert 18



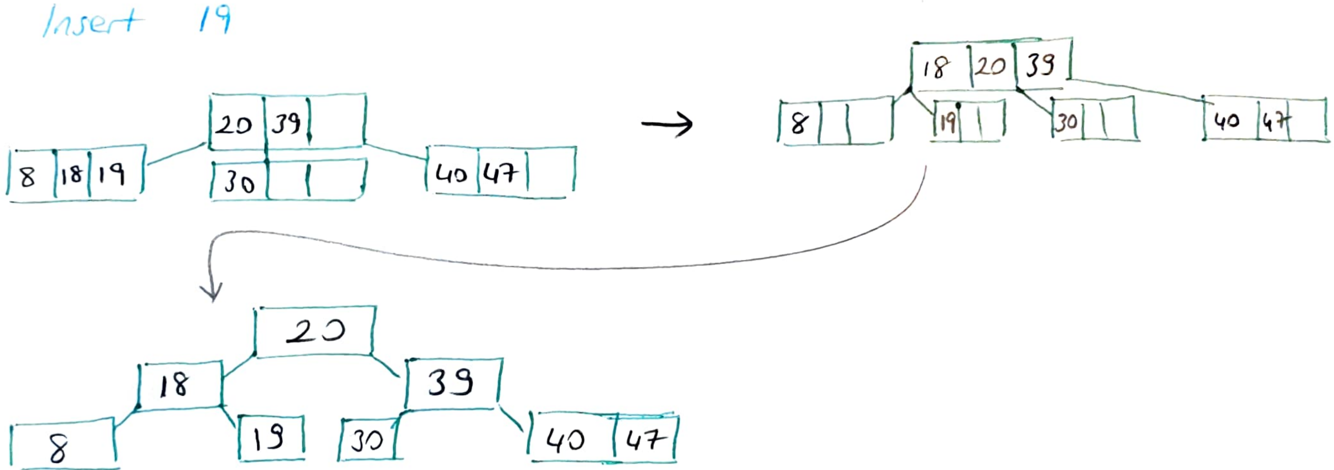
18<20 go left
18>8 put right of 8

Insert 40



40>39 go right
40<47 put left of 47

Insert 19

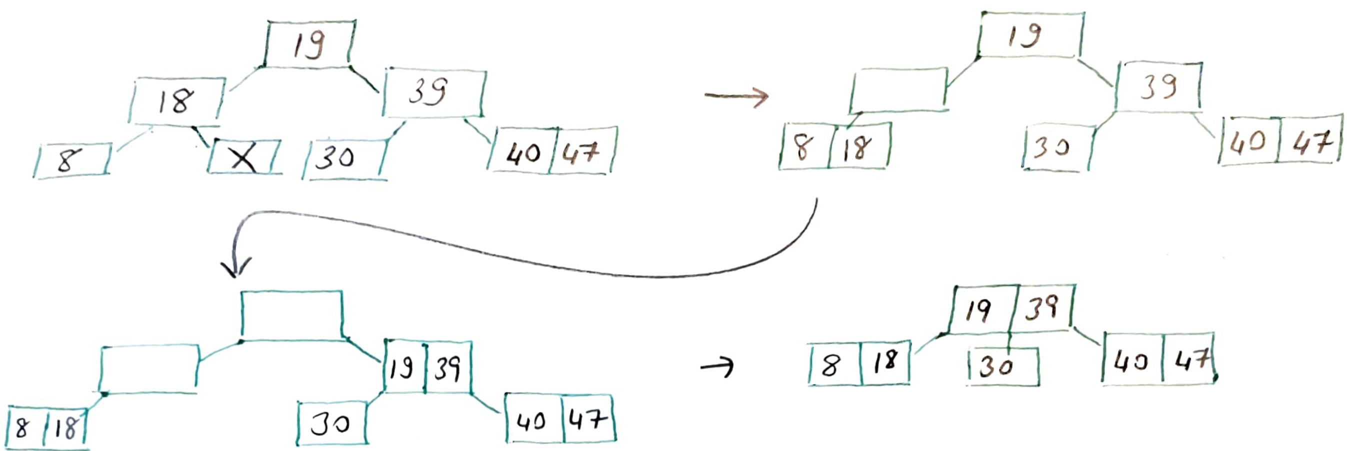


Firstly, the middle value 18 go up. But the node can't store 3 values. So in this node (18,20,39), middle value 20 go up and split into two new 2-nodes.

Emirhan Uzun
171044019

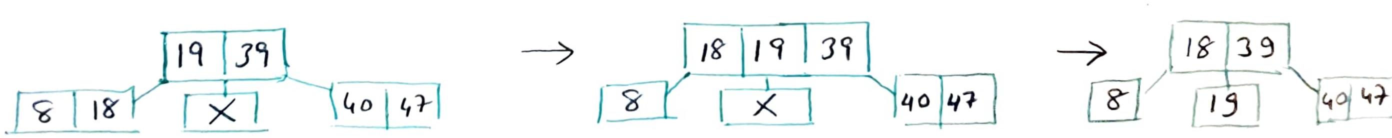
2-3 Tree Deletion

Remove 20



First, we remove the value 20. So we get biggest element in left subtree which is 19. Now, we delete node 19 and merge the sibling and parent of this node. After that we merge the root and its sibling. Now, the tree's height is equal and tree is balanced.

Remove 30



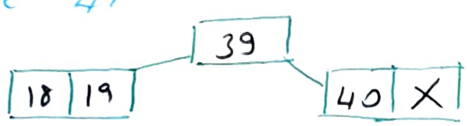
First, we find and remove 30. After that, we have too few value for node which contains 30. So we take a value from left subling and put node, middle value of parent which is 19

Emirhan Uzun
171044019

Remove 8

We find node and remove value. Then, we merge the nodes. The smallest element in root, goes down the near of 18

Remove 47



No more operations

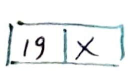
Remove 39

We removed root value. So the biggest element in left subtree becomes root.

Remove 18

We remove the node value and then merge the values.

Remove 40



Just delete the value. The node is not empty so it is not need to more operations

Remove 19

Tree is empty

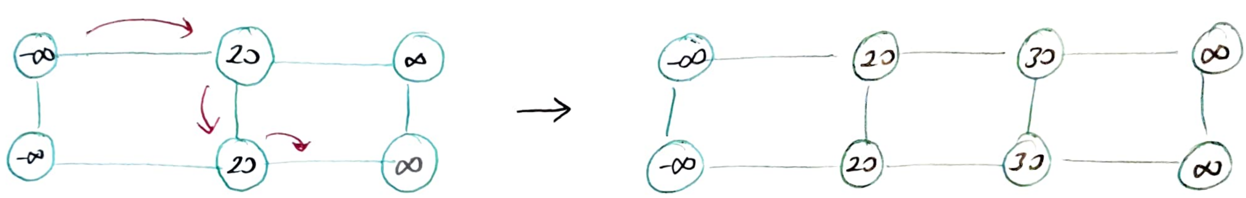
Emirhan Uzun
171044019

Skip List Insertion

Insert 20

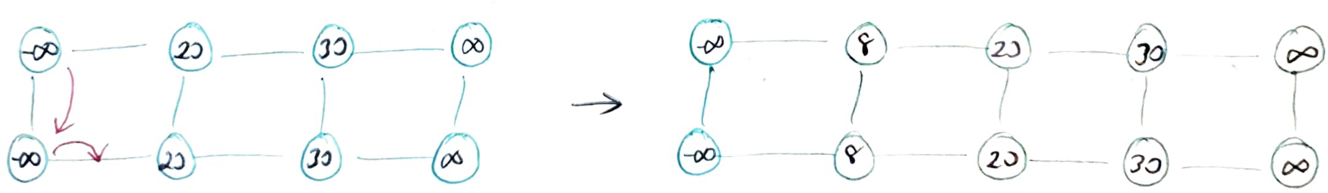
$20 > -\infty$ and we are at first level. So add the value and added new level (it is optional)

Insert 30



We search to value. $20 < 30$ so go right. After that, $\infty > 30$ so moving down. Again $\infty > 30$ and no more level to move down, insert here. If you would like, you can add one level high (it is optional)

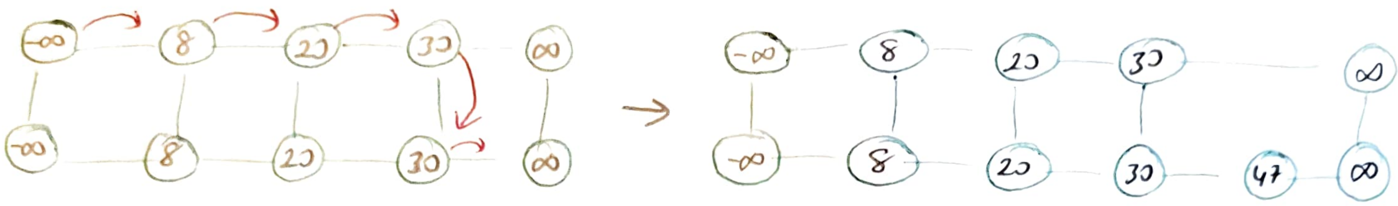
insert 8



$8 < 20$ move down. $8 < 20$ again but no more level so insert here. Add above level(s). (It's optional)

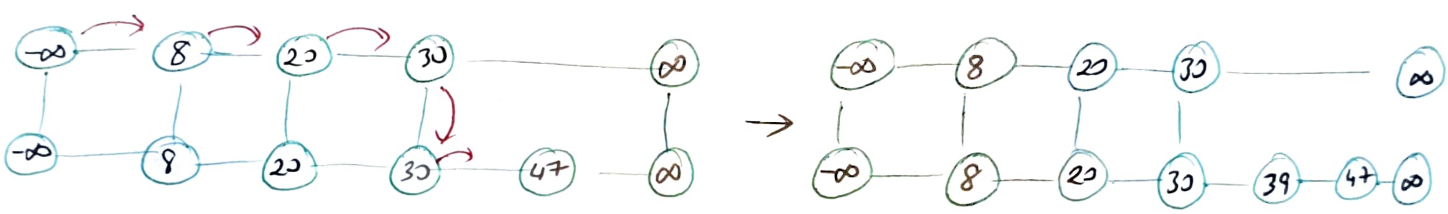
Emirhan Uzun
171044019

Insert 47



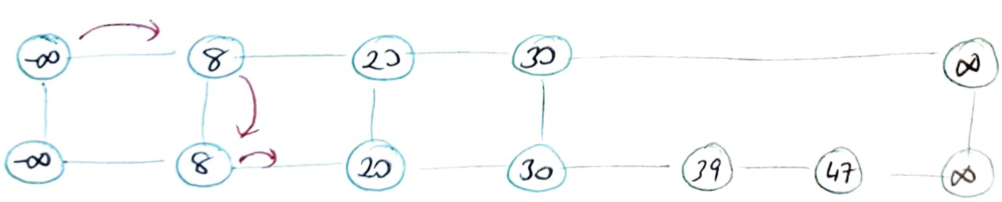
$8 < 47$ move right. $20 < 47$ move right. $30 < 47$ move right. $\infty > 47$ so move down. Again $47 < \infty$ and no more levels. So insert here

Insert 39

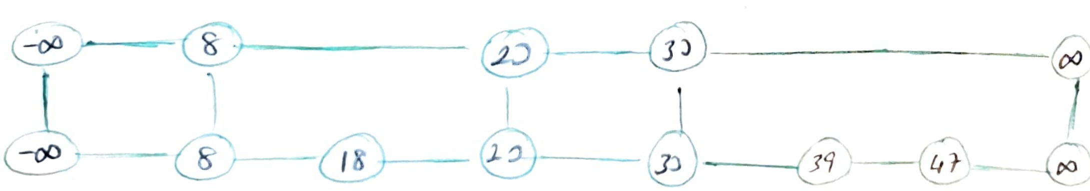


$8 < 39$ move right. $20 < 39$ move right. $30 < 39$ move right. $\infty > 39$ move down and compare again. $39 < 47$ and no more levels. Insert here.

Insert 18

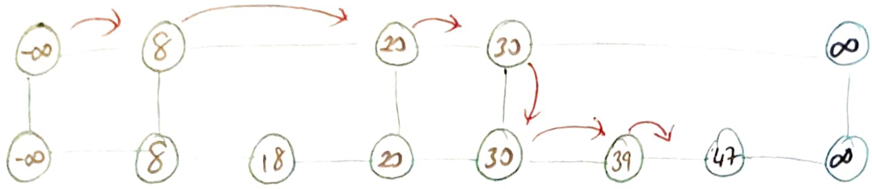


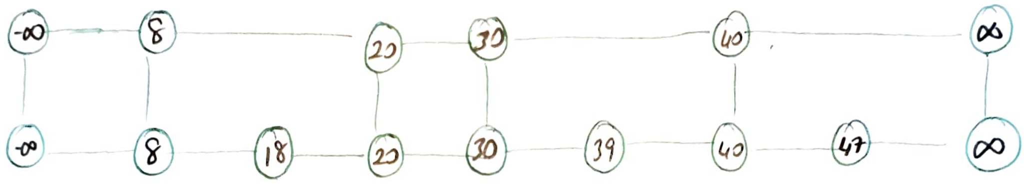
$8 < 18$ move right
$18 < 20$ move down.
$18 < 20$ but no more levels. Insert here.



Emirhan Uzun
171044019

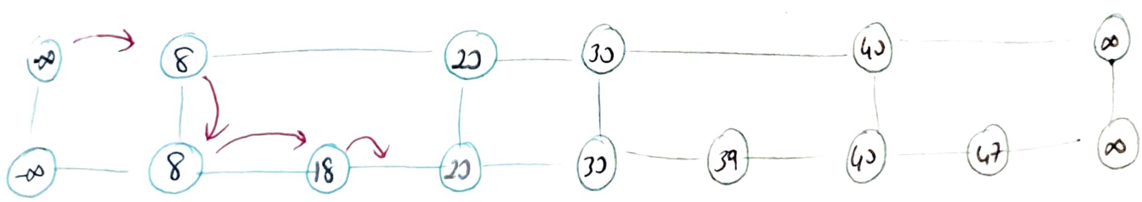
Insert 40

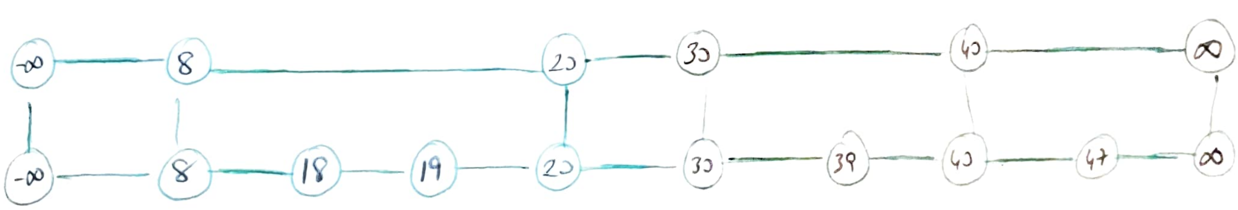




8<40 move →
20<40 move →
30<40 move →
40<∞ move ↓
39<40 move →
47<40 move ↓ but no more level insert here. If you want add above level (it is optional)

Insert 19





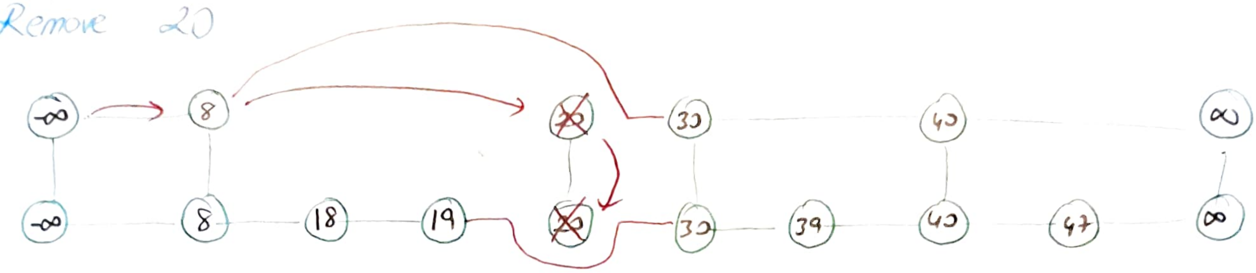
8<19 move →
20>19 move ↓

18<19 move →
19<20 but there is no more level. Insert here

Emirhan Uzun
171044019

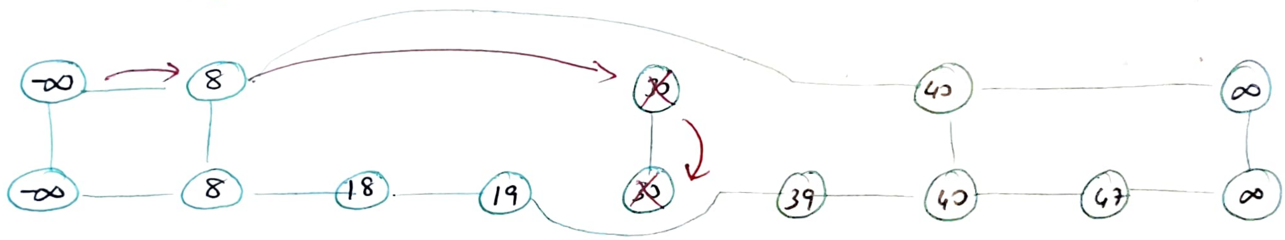
Skip List Deletion

Remove 20



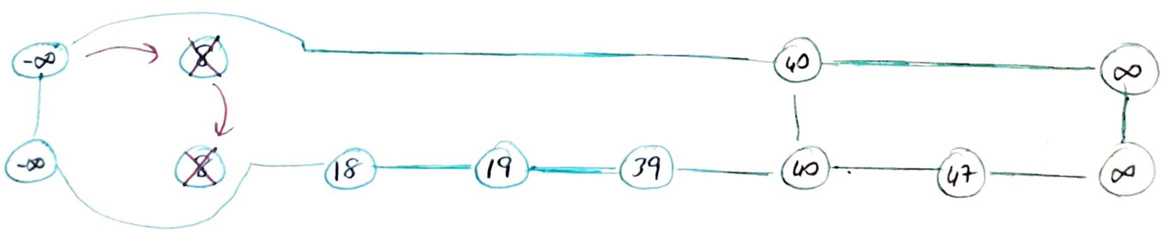
8 < 20 move right
20 = 20 move down. There is no more level. Delete each level this value

Remove 30



Find node lowest level and delete value from each node.
8 < 30 move right, 30 = 30 move down, 30 = 30 no more level.

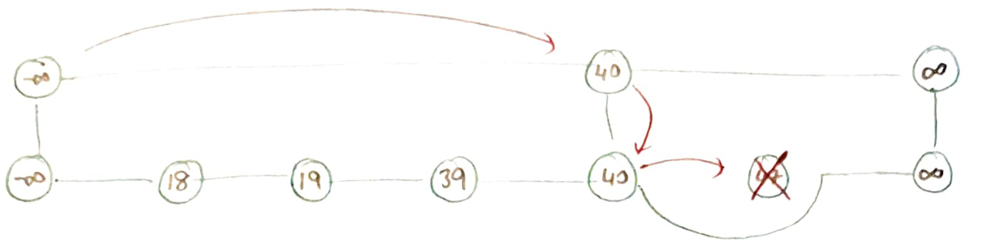
Remove 8



8 = 8 move down, 8 = 8 no more level, delete from each node

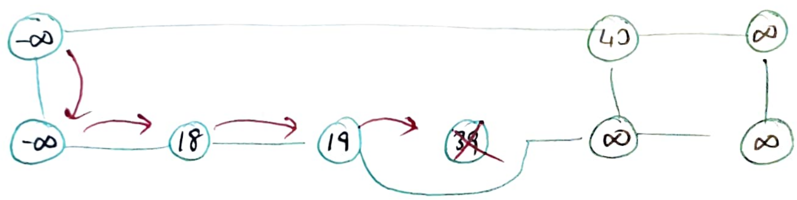
Emirhan Uzun
171044019

Remove 47



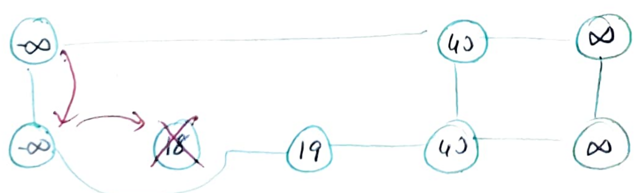
40<47 move right, ∞>47 move down, 40<47 move right,
47=47 delete from each level.

Remove 39



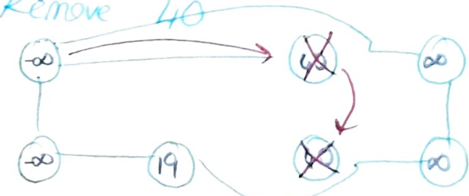
39<40 move down.
18<39 move right
19<39 move right
39=39 delete

Remove 18



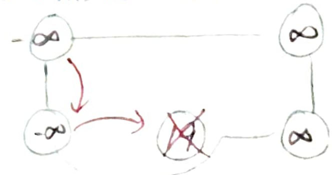
18<40 move down
18=18 delete

Remove 40



40=40 move down
Delete

Remove 19



19<∞ move down
19=19 delete

Emirhan Uzun
171044019

B-Tree Order 4 Insertion

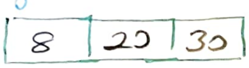
Insert 20

Add to root

Insert 30

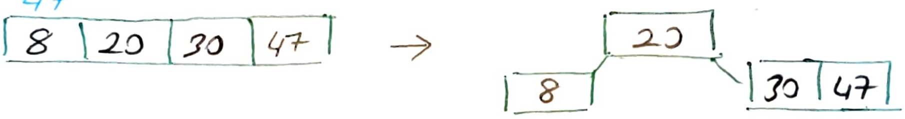
30>20 Put 20's right place

Insert 8



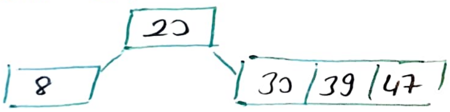
8<20 . Put left place of 20.

Insert 47



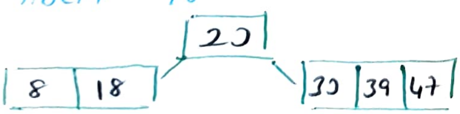
The node has too much keys. Split into 2 node and 20 move up to parent of them. (I choose 20 instead of 30)

Insert 39



39>20 go right
39>30 and 39<47 add middle of them

Insert 18

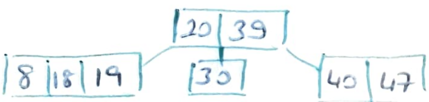


18<20 go left
18>8 add right place of 8

Insert 40

Too much value. Split and middle value move up.

Insert 19



19<20 go left
19>18 add right place of 18

Emirhan Uzun
171044019

B Tree Order 4 Deletion

Remove 20

The 20 was deleted and add to root biggest element of left subtree

Remove 30

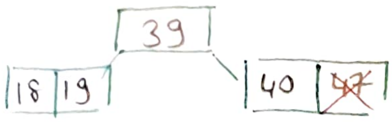
The value 30 is removed and this node has too few value (no element), so 19 which is biggest element after 30. Now the root has just 1 element, so biggest element in left subtree moves up to root.

Remove 8

The value 8 is removed and this node has no elements. Also its right sibling has too few keys. So, merge with parent and sibling.

Emirhan Uzun
171044019

Remove 47

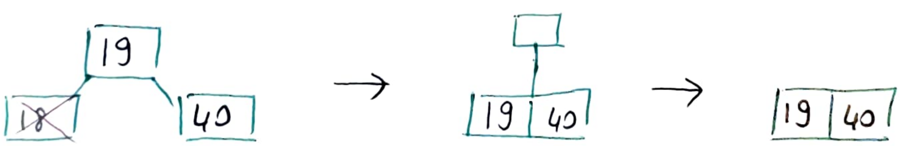


Just delete value. The node is not empty so it is not need to do more operations

Remove 39

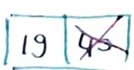
The root has no elements after removing. So get biggest element in left subtree and put to root.

Remove 18



The node has no elements and its sibling also has just 1 key. So merge these values and put to root

Remove 40



Just delete it. Because root is not empty and nothing needs to do.

Remove 19

Tree is empty.