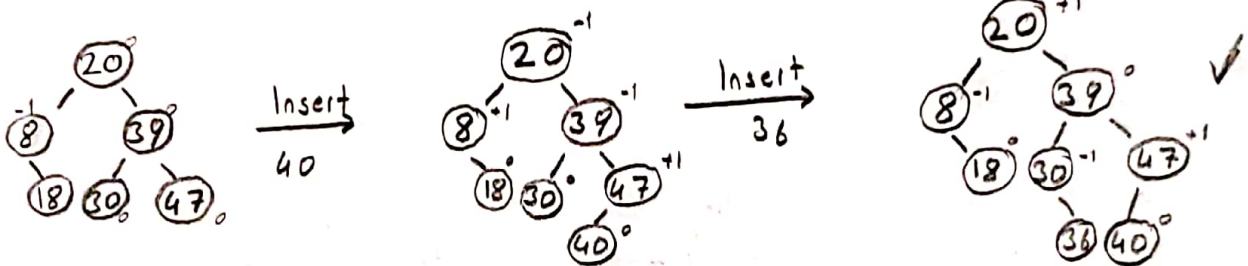
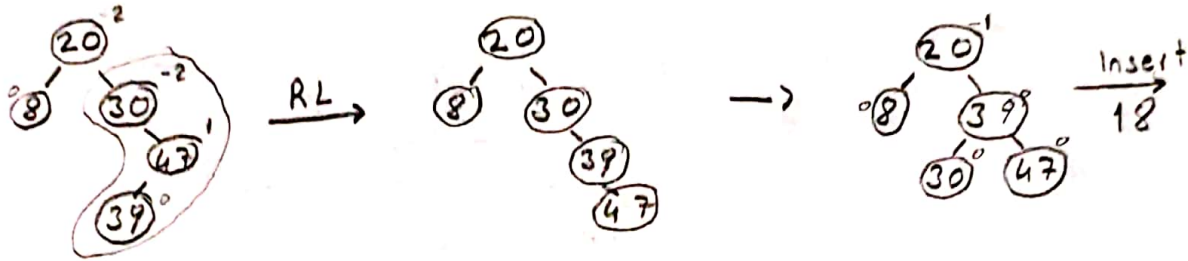
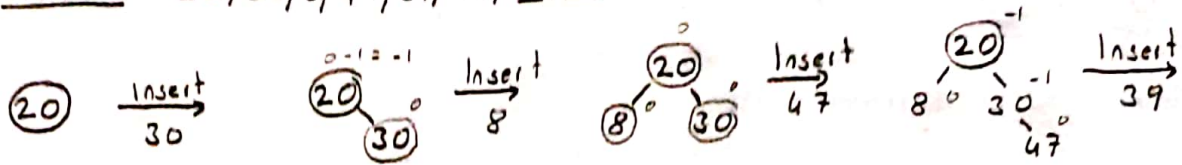


→ AVL Tree

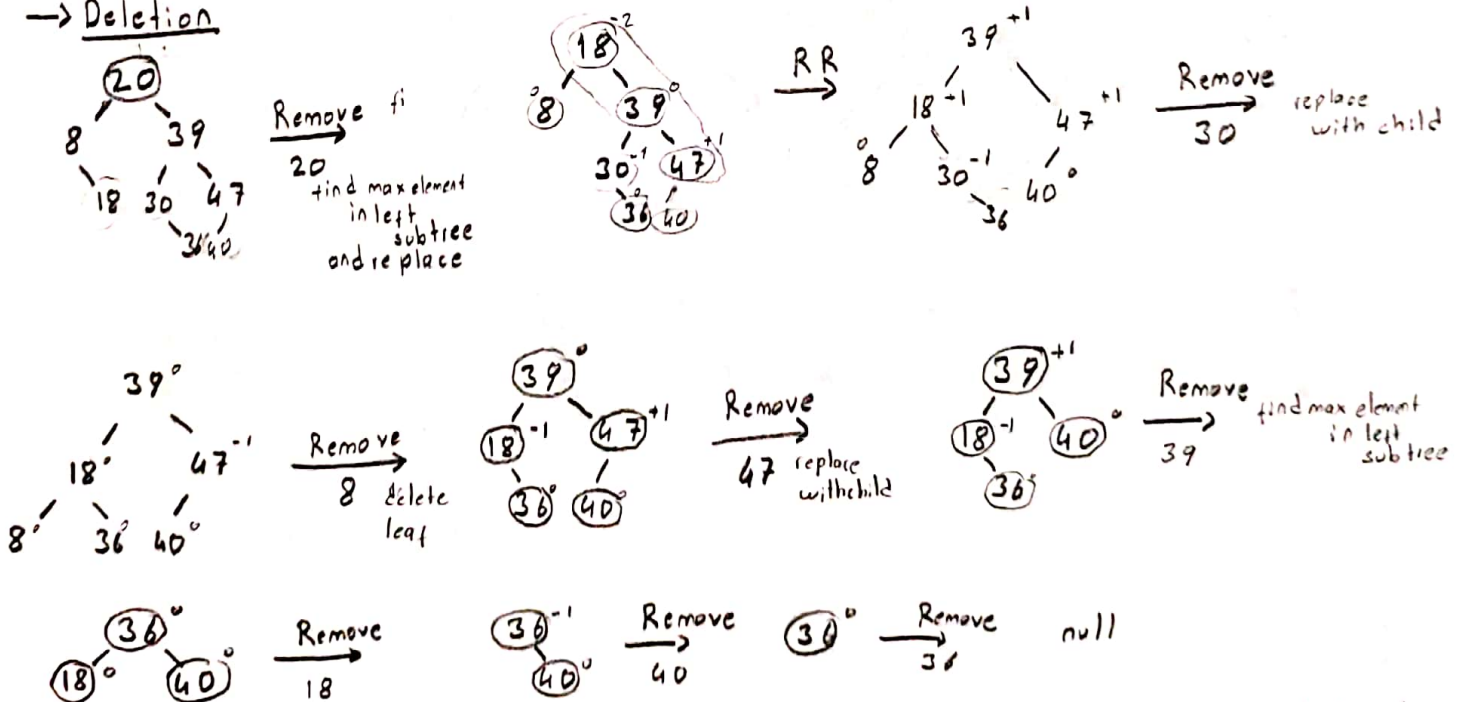
Baran Hasan BozDUMAN 171044036

→ Insertion 20, 30, 8, 47, 39, 18, 40, 36



When we're inserting an element to AVL tree we check for each step and decide whether the tree is balanced or not. If difference between right and left height is $\{-1, 0, +1\}$ for each node the tree is balanced ($|\text{balance factor}| = |\text{height left subtree} - \text{height right subtree}| \leq 1$) otherwise it needs some rotation operations

→ Deletion



When we're removing an element from AVL tree. The operations like BST delete operations but after each deletion operation we check the tree is balanced or not if it is unbalanced we make some rotation operations to obtain a balanced tree

→ Red-Black Tree

Rules

- Every node is red or black.
- Root is always black.
- New insertions are always red.
- Every path from root-leaf has same number of black nodes.
- No path can have two consecutive Red nodes.
- Nulls are black.

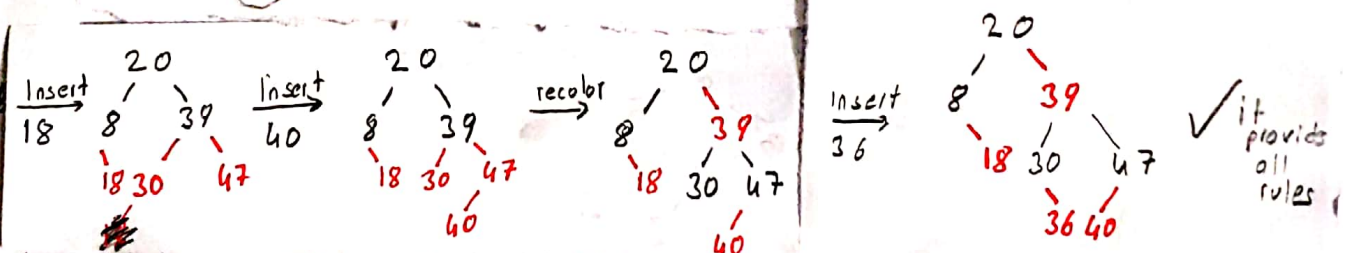
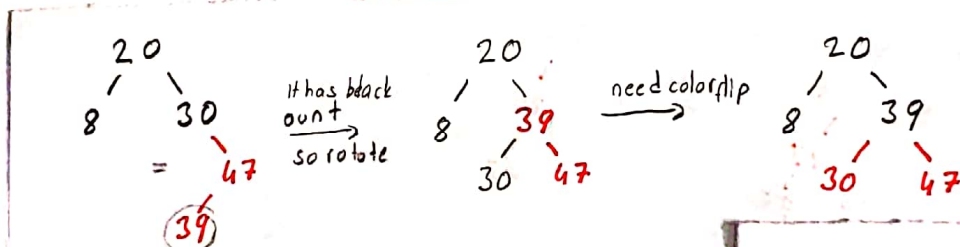
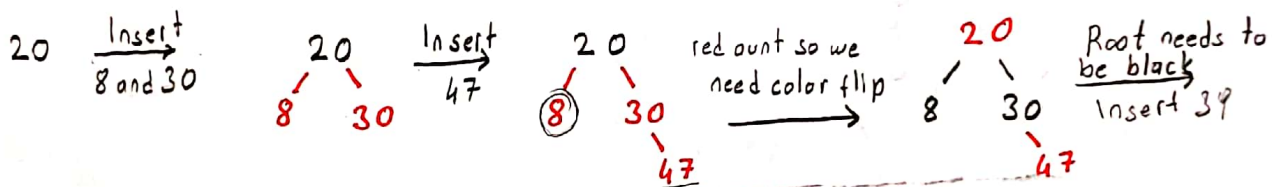
We should check if it providing the rules or not after each operation.

To fix tree

Black Aunt → Rotate

Red Aunt → Color flip

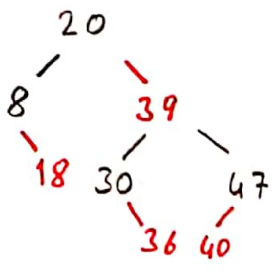
→ Insertion



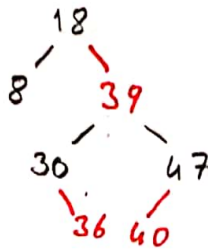
→ Deletion

Deletion Rules

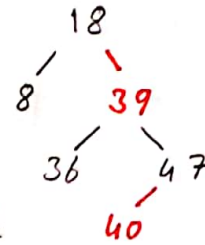
- If the node we deleted is red and its replacement is red or null, we are done
- If the node we deleted is red and its replacement is black, color the replacement red and proceed to the appropriate case.
- If the node we deleted is black and its replacement is red color the replacement black.
- If the node we deleted is black and its replacement is null or black, proceed to the appropriate case



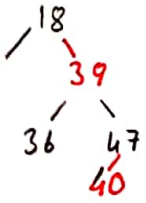
remove
20
replace with
left subtree
biggest and
color to black



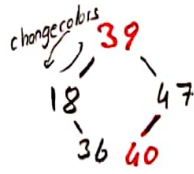
Remove
30
replace with
child and
color to black



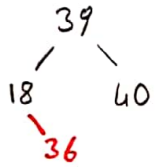
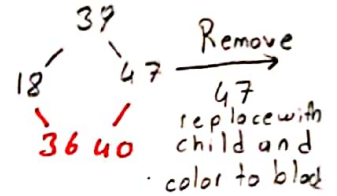
Remove
8



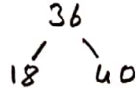
There is a
null leaf so
the number of
black is not
equal
shift to left



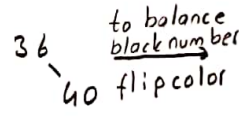
flip
color
to balance
black number



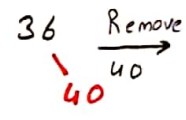
Remove
39
replace with
left subtree biggest
and color to black



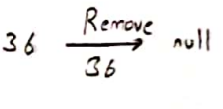
Remove
18



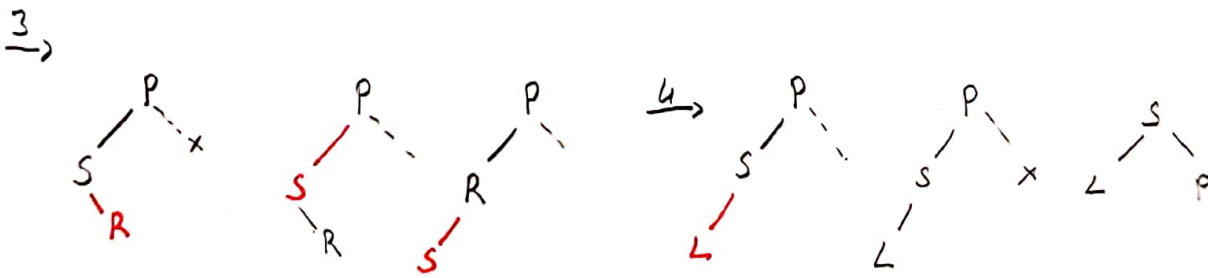
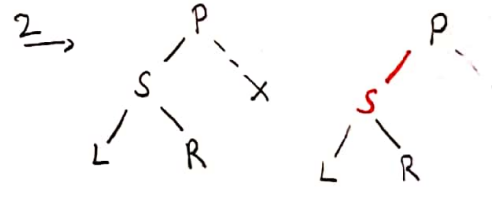
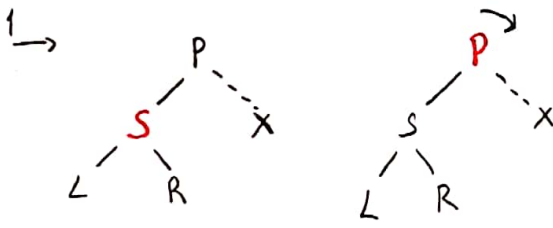
to balance
black number
flip color



Remove
40
Replace with
child and
color to black



Cases

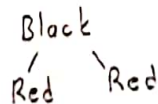


After color flip:



you need to change red
to black if its the root

After Rotation



→ 2-3 Tree

→ Insertion

20-30-8-47-39-18-40-36

Insert
20

(20)

Insert
30

we add it next to 20

(20, 30)

since we try to add before 20 and there is no more room we split them

(20, 30)
(8)

Insert
47

47 is bigger than 30 and there is a room for it

(20, 30, 47)

Insert
39

(20, 30, 47)
(8)

(20, 39)
(8, 30, 47)

39 the 39 is between 30 and 47 so we can add it next to 20 and split the 30 and 47

Insert
18
between 8-20

(20, 39)
(8, 18, 30, 47)

18 is between 8 and 20 and there is a room for 18 in next to 8

Insert
40

(20, 39)
(8, 18, 30, 40, 47)

we add it next to 47 and it provides 39 > 40 > 47

Insert
36

(20, 39)
(8, 18, 30, 36, 40, 47)

we decide the numbers and where they suppose to be if the given number are between them of a nodes number we add it into a node which is connected to middle.

→ Deletion

(20, 39)
(8, 18, 30, 36, 40, 47)

Remove
20

we put 18 to place of 20 and it's providing the condition again

(18, 39)
(8, 30, 36, 40, 47)

Remove
30

we just delete it

(18, 39)
(8, 36, 40, 47)

Remove
8

(39)
(18, 36, 40, 47)

we need to shift 18 down and merge with 36 to provide condition

(39)
(18, 36, 40, 47)

Remove
47

we just delete it

(39)
(18, 36, 40)

Remove
39

we replace with 36 which is the bigger of left

(36)
(18, 40)

Remove
18

(36, 40)

Remove
40

(36)

Remove
36

null

After remove 18 we need to merge them

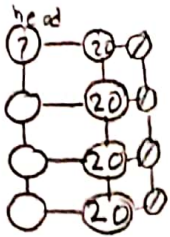
The tree should provide below situation after we make any changes

(x, y)
(<x, >y)

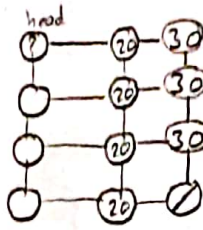
→ Skip List

→ Insertion 20-30-8-47-39-18-40-36

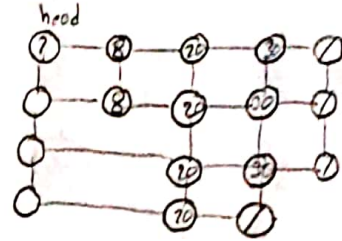
Insert
20



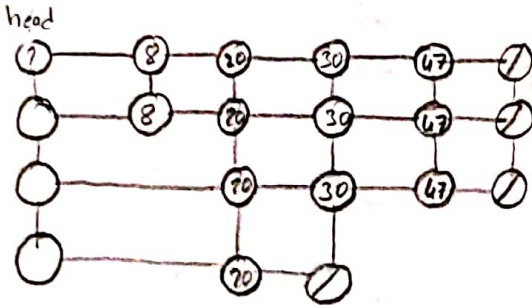
Insert
30



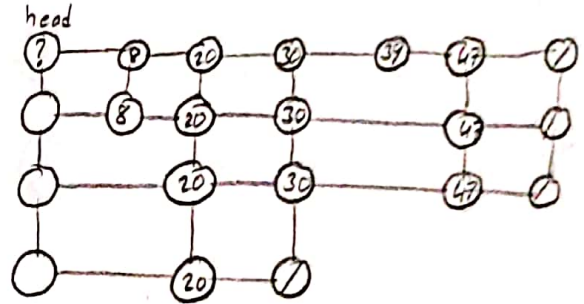
Insert
8



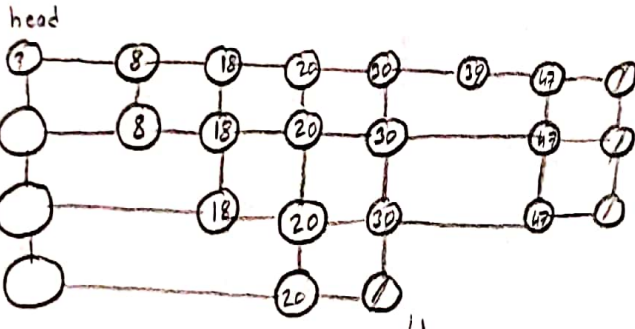
Insert
47



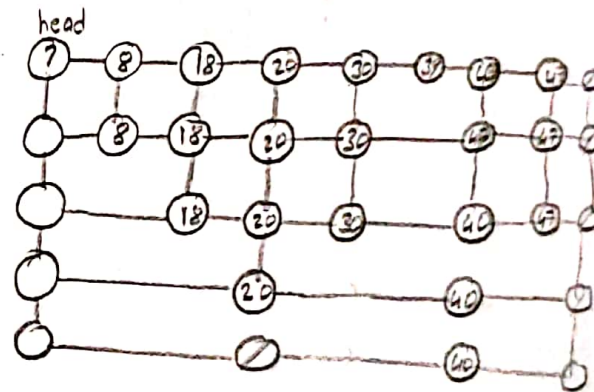
Insert
39



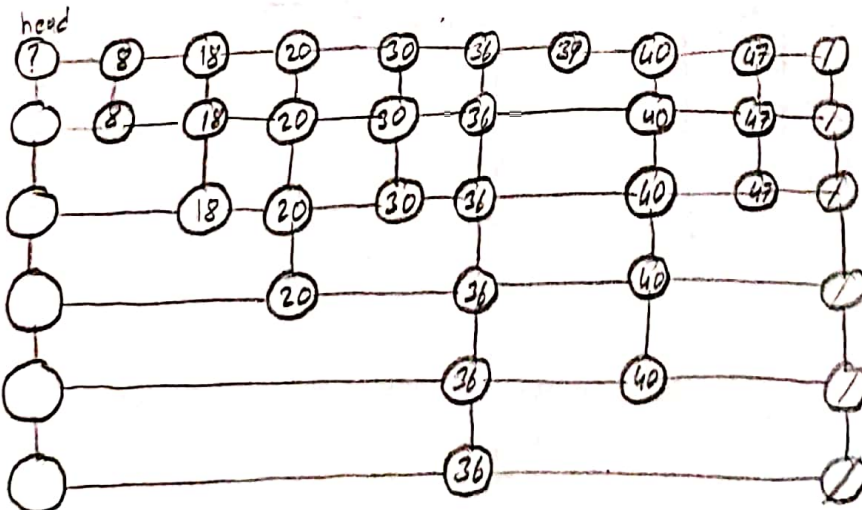
Insert
18



Insert
40



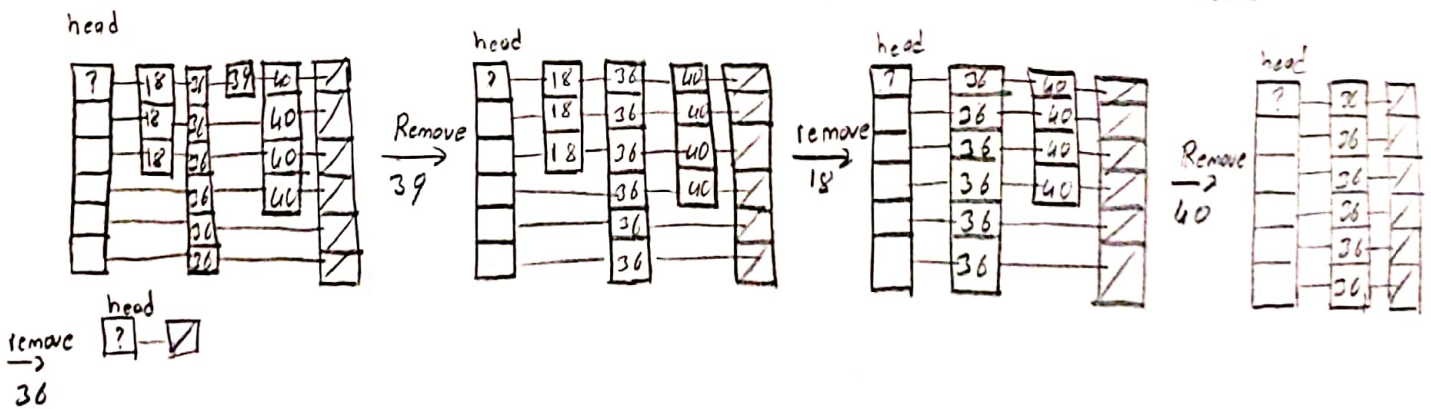
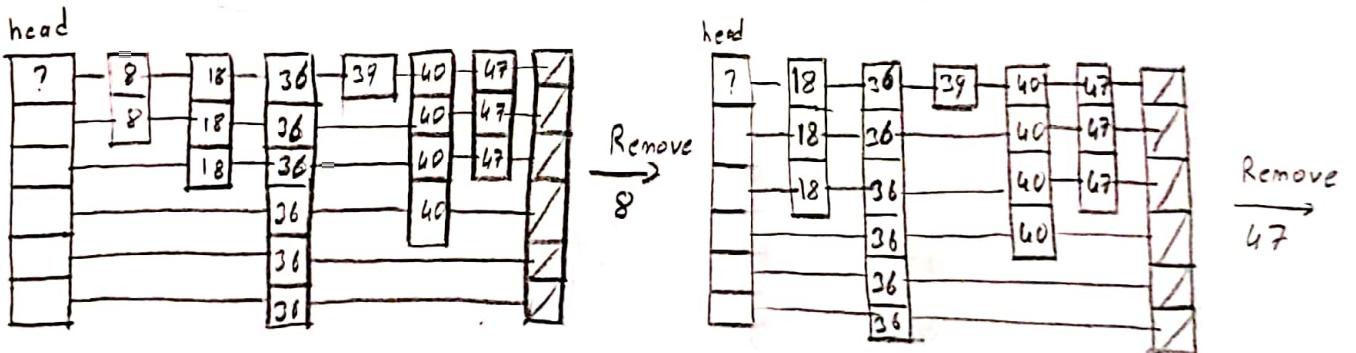
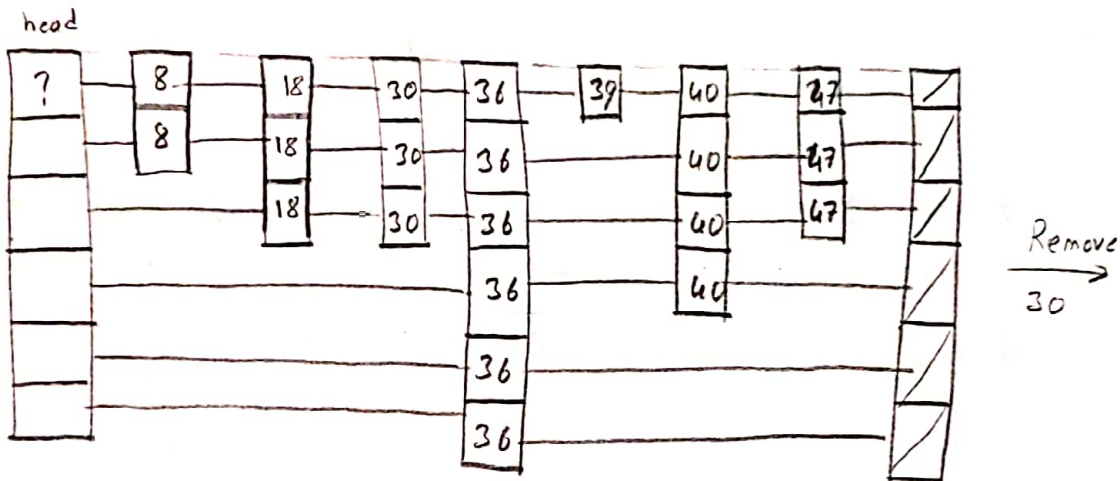
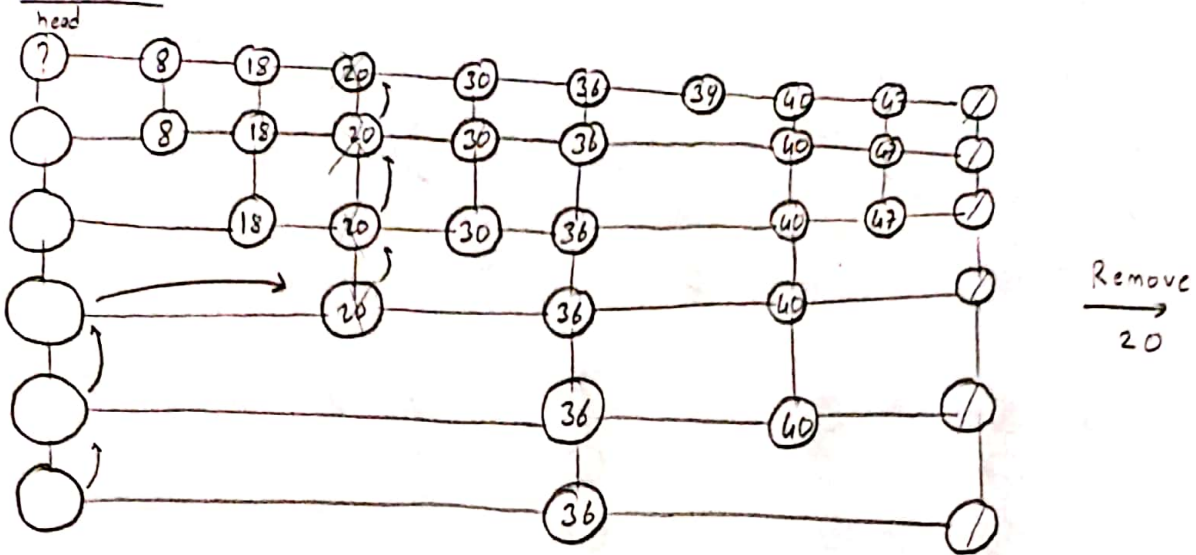
Insert
36



So we can reach any number less than 'number of steps' as we reach like single or double linked list, we don't need to visit every node we skip some of them

level of number is determined by randomly but if we chose it according to number of elements in the list it would be more useful. According to book implementation it chose max level as m and the number of data between 1 to 2^{m-1} . I chose randomly above because I know the number of data the above 2^{m-1} is valid for data size which is bigger than 15.

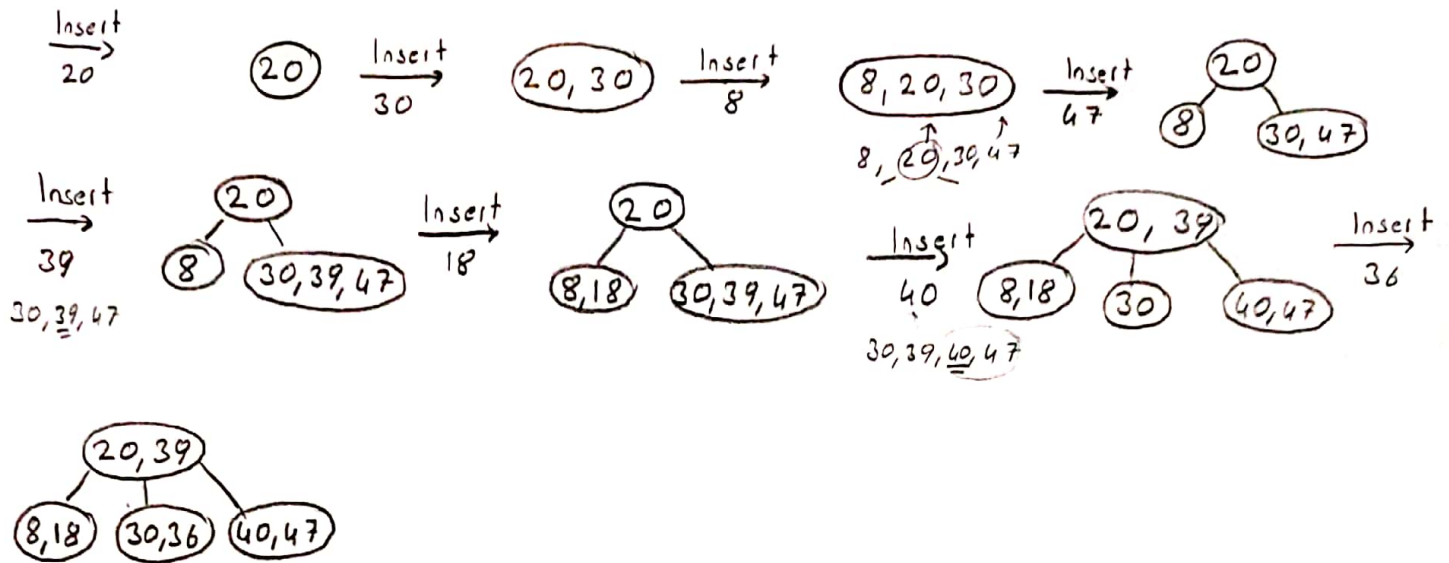
→ Deletion



Delete operations like deleting a node from a linked list first we find the datum then goes up until the starting point of datum which will be deleted and we delete one by one and connect the lists again.

→ B-Tree (degree 4)

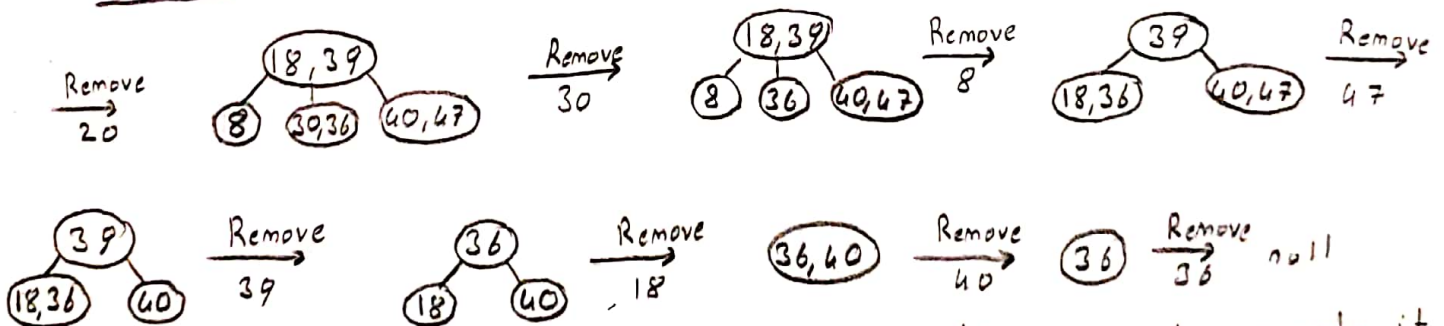
→ Insertion 20-30-8-47-39-18-40-36



→ Rules for B tree

- For m-degree tree a node must fill at least $\lceil m/2 \rceil$ of node, after that you can add new nodes.
- The rule is determined above except for the root root can have minimum 2 children
- All leaf must be at same level
- Creation process is from bottom to up if there is no room for given datum we shift one of them from bottom to up then we split it new nodes.

→ Deletion



→ Also for deletion the rule (if the item to be removed is in an interior node, it can't be deleted simply because that would damage the B-tree. To retain the B-tree property, the item must be replaced by its inorder predecessor (or its inorder successor) which is in a leaf) is also valid.

References: Lectures

Book

youtube/Abdul Bari

youtube/mycodeschool

youtube/Adam Gaweda

youtube/Jenny's lectures

youtube/Rob Edwards SDSU

online visualization