GIT Department of Computer Engineering

CSE 222/505 – Spring 2020

Homework #07 Part 1 Report

Abdullah ÇELİK

171044002

Building the following data structures by sequence of integers 20, 30, 08, 47, 39, 18, 40, 02. Then removing all the items one by one in the same order (first in, first out).

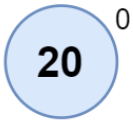
AVL Tree

Note: *The number on the top right of the node is its balance. The formula is* $h_R - h_L$. h_R *and* h_L *are the heights of the left and right subtrees, respectively.*

Adding

- Step 1 – Add integer 20

  Perform standart BinarySearchTree add.
  After operation, tree:

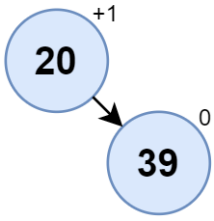
  

  Tree is balanced

  So, rotation isn't necessary.

- Step 2 – Add integer 30

  Perform standart BinarySearchTree add.
  After operation, tree:

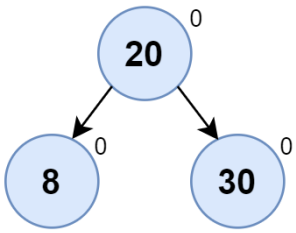
  

  Tree is balanced

  So, rotation isn't necessary.

- Step 3 – Add integer 08

  Perform standart BinarySearchTree add.
  After operation, tree:

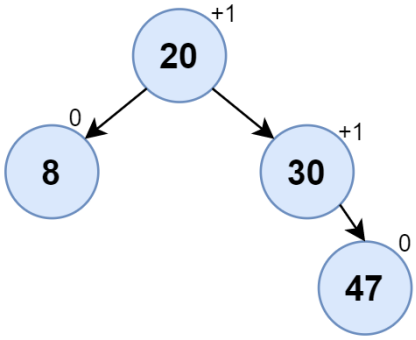


Tree is balanced

So, rotation isn't necessary.

- Step 4 – Add integer 47

  Perform standart BinarySearchTree add.
  After operation, tree:

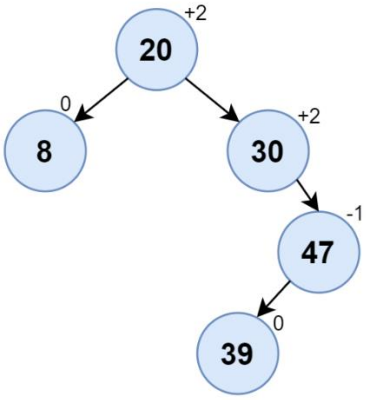


Tree is balanced

So, rotation isn't necessary.

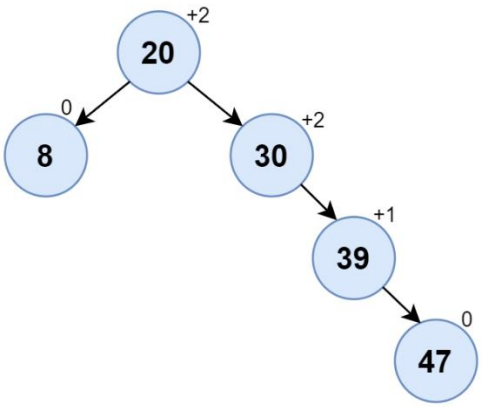
- Step 5 – Add integer 39

  Perform standart BinarySearchTree add.
  After operation, tree:



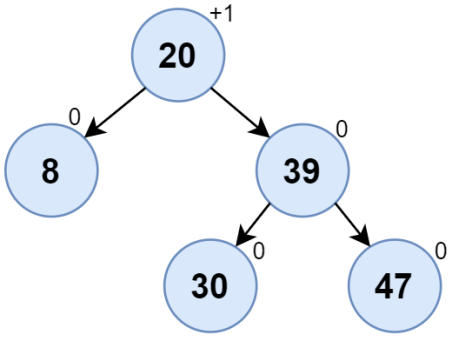
Tree is unbalanced

There are right-left case. So firstly, rotate right around child(47).



Tree is unbalanced

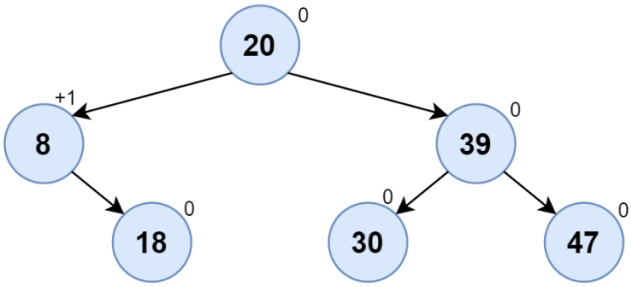
Secondly, rotate left around parent(30).



Tree is balanced

- Step 6 – Add integer 18

  Perform standart BinarySearchTree add.
  After operation, tree:



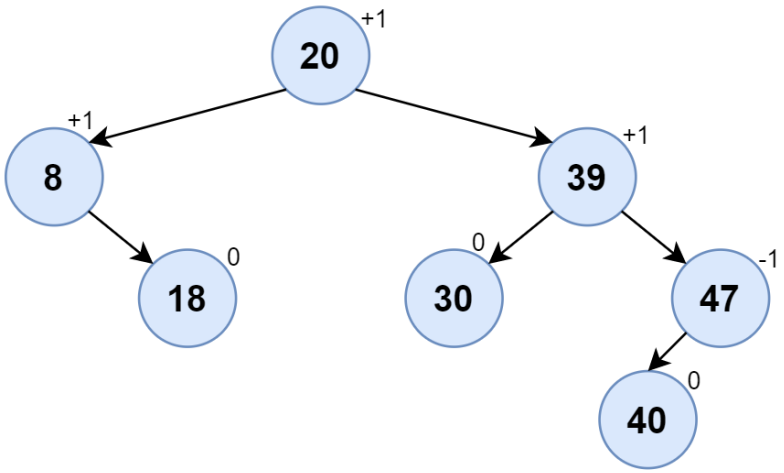
Tree is balanced

So, rotation isn't necessary.

- Step 7 – Add integer 40

  Perform standart BinarySearchTree add.

After operation, tree:

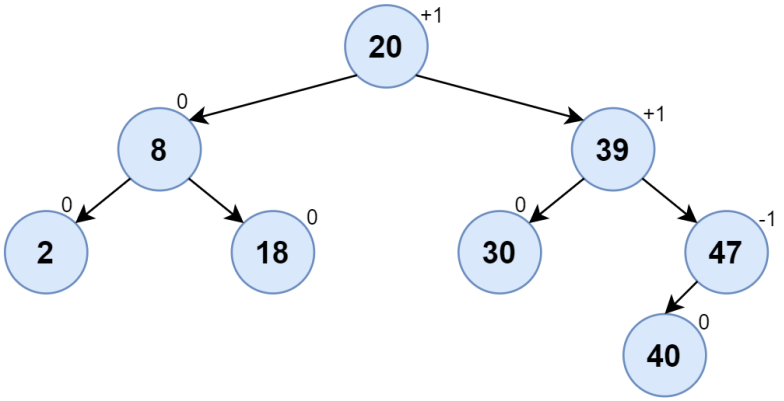


Tree is balanced

So, rotation isn't necessary.

- Step 8 – Add integer 02

  Perform standart BinarySearchTree add.
  After operation, tree:



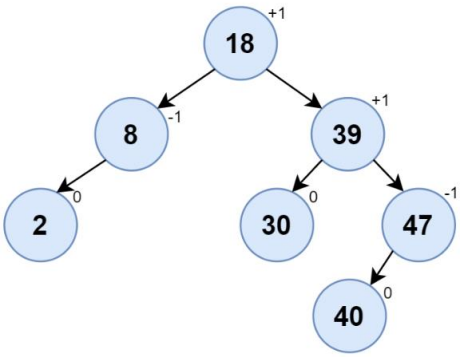
Tree is balanced

So, rotation isn't necessary.

Removing

- Step 1 – Remove 20

  Perform standart BinarySearchTree delete.
  After operation, tree:

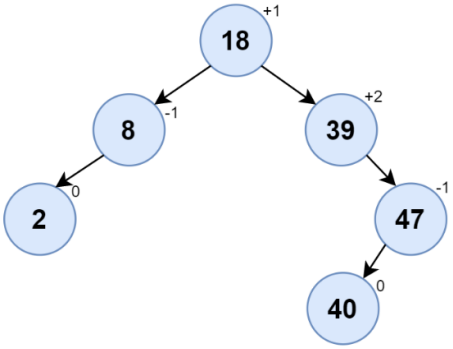


Tree is balanced

So, rotation isn't necessray.

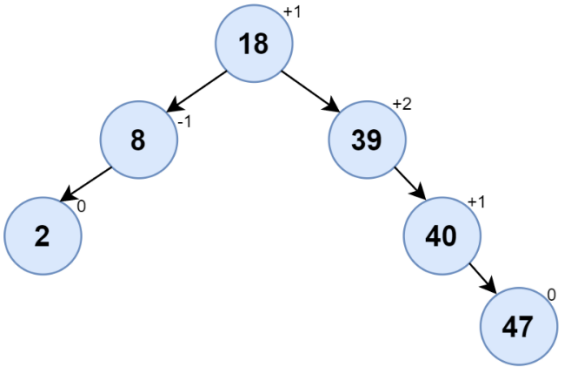
- Step 2 – Remove 30

  Perform standart BinarySearchTree delete.
  After operation, tree:



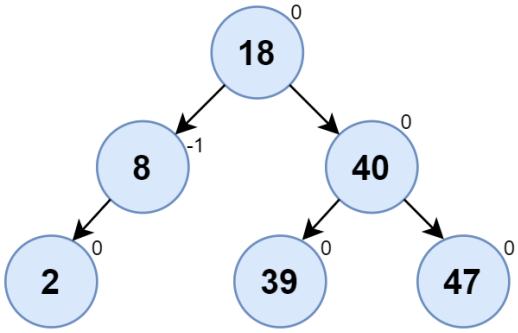
Tree is unbalanced

There is right-left case. So firstly, rotate right around child(47).



Tree is unbalanced

Secondly, rotate left around parent(39)

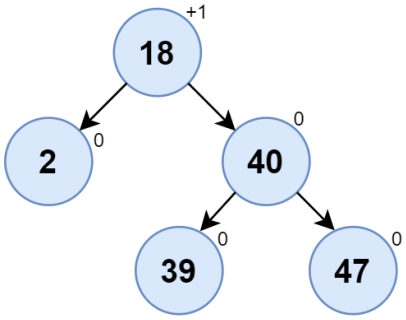


Tree is balanced

.

- Step 3 – Remove 08

  Perform standart BinarySearchTree delete.
  After operation, tree:

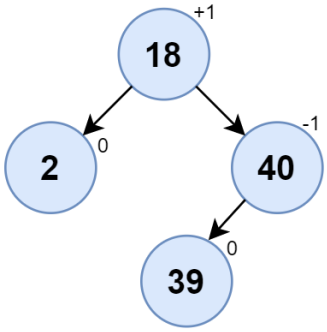


Tree is balanced

So, rotation isn't necessary.

- Step 4 – Remove 47

  Perform standart BinarySearchTree delete.
  After operation, tree:

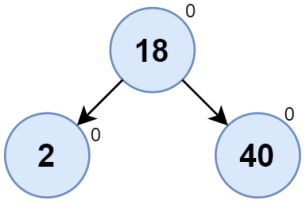


Tree is balanced

So, rotation isn't necessary.

- Step 5 – Remove 39

  Perform standart BinarySearchTree delete.
  After operation, tree:

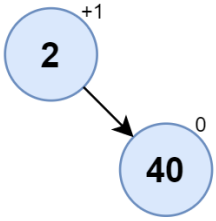


Tree is balanced

So, rotation isn't necessary.

- Step 6 – Remove 18

  Perform standart BinarySearchTree delete.
  After operation, tree:

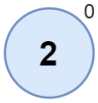


Tree is balanced

So, rotation isn't necessary.

- Step 7 – Remove 40

  Perform standart BinarySearchTree delete.
  After operation, tree:



Tree is balanced

So, rotation isn't necessary.

- Step 8 – Remove 02

  Perform standart BinarySearchTree delete.
  After operation, tree:

Red-Black Tree

Adding

- Step 1 – Adding integer 20

  Perform standart BinarySearchTree add. Adding node is always red.
  After adding, tree:

  Tree is unbalanced

  Because of invariant 2(in the book). Invariant 2 is that the root is always black.
  Recolor root to black.

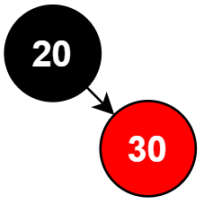
  

  Tree is balanced

  Tree maintains invariants. So, recoloring or rotation aren't necessary.

- Step 2 – Adding integer 30

  Perform standart BinarySearchTree add. Adding node is always red.
  After adding, tree:

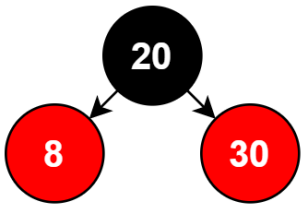
  

  Tree is balanced

  Tree maintains invariants. So, recoloring or rotation aren't necessary.

- Step 3 – Adding integer 8

  Perform standart BinarySearchTree add. Adding node is always red.
  After adding, tree:

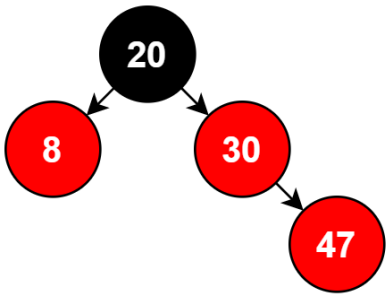


Tree is balanced

Tree maintains invariants. So, recoloring or rotation aren't necessary.

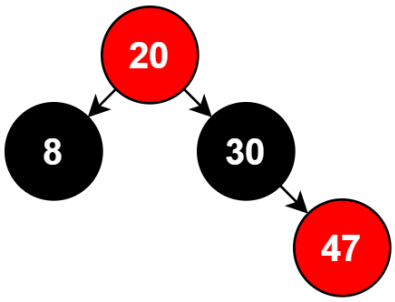
- Step 4 – Adding integer 47

  Perform standart BinarySearchTree add. Adding node is always red.
  After adding tree:



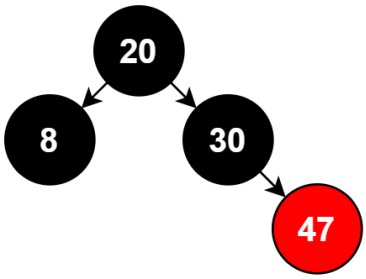
Tree is unbalanced

Because of invariant 3(in the book). Invariant 3 is that a red node always has black children. (A null reference is considered to refer to a black node)
Since 47's parent's sibling(8) is red, we should recolor parent and grandparent. We should paint the grandparent(20) of 47 to black, and the grandparent's childs(8,30) to red.



Tree is unbalanced

Because of invariant 2. Invariat 2 is that the root is always black.
Recolor root to black.

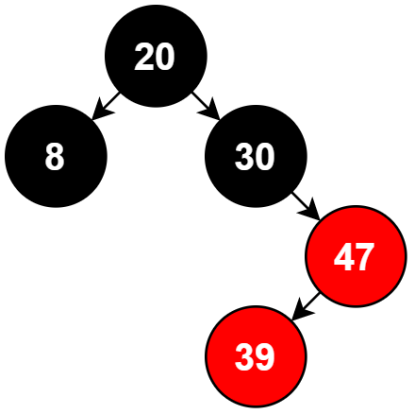


Tree is balanced

Tree maintains invariants. So, recoloring or rotation aren't necessary.

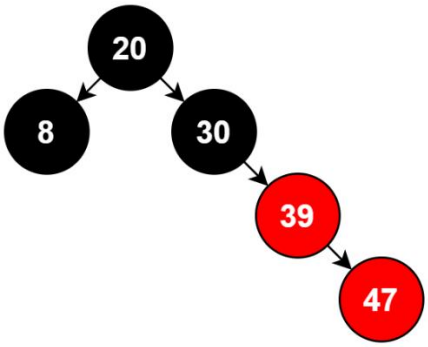
- Step 5 – Adding integer 39

  Perform standart BinarySearchTree add. Adding node is always red.
  After adding, tree:



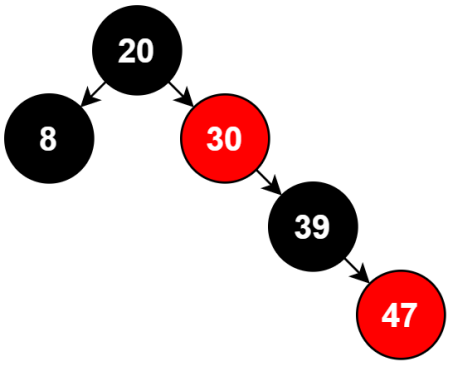
Tree is unbalanced

Because of invariant 3(in the book). Invariant 3 is that a red node always has black children. (A null reference is considered to refer to a black node)
Since node 39 parent's sibling(null) is black but node 39 isn't on the same side of its parent as the parent is to the grandparent, we should three operation. First operation is rotating node 39's parent to right. Second operation is recoloring. Third operation is rotating node 39's new parent to left. After rotating 47 to right, tree:



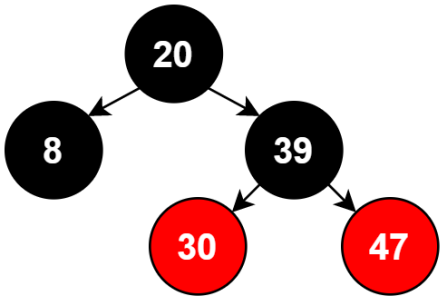
Tree is unbalanced

After recoloring, tree:



Tree is unbalanced

After rotation 30 to left, tree:

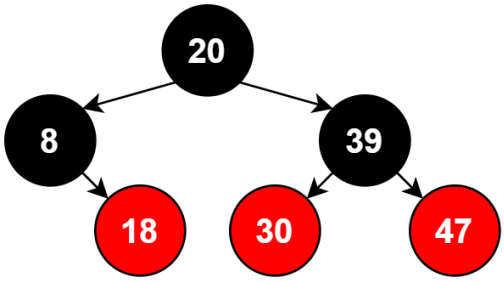


Tree is balanced

Tree maintains invariants. So, recoloring or rotation aren't necessary.

- Step 6 – Adding integer 18

  Perform standart BinarySearchTree add. Adding node is always red.
  After adding, tree:

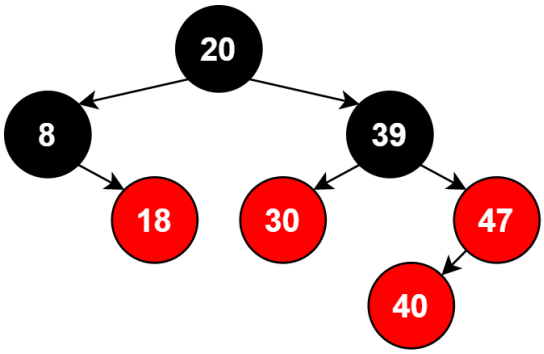


Tree is balanced

Tree maintains invariants. So, recoloring or rotation aren't necessary.

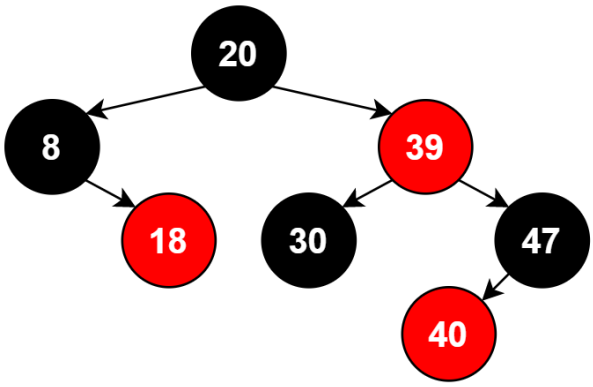
- Step 7 – Adding integer 40

  Perform standart BinarySearchTree add. Adding node is always red.
  After adding, tree:



Tree is unbalanced

Because of invariant 3(in the book). Invariant 3 is that a red node always has black children. (A null reference is considered to refer to a black node)
Since node 40's parent's sibling is red, we should recoloring. Recolor 40's grandparent red and its childs to black.
After recoloring, tree:

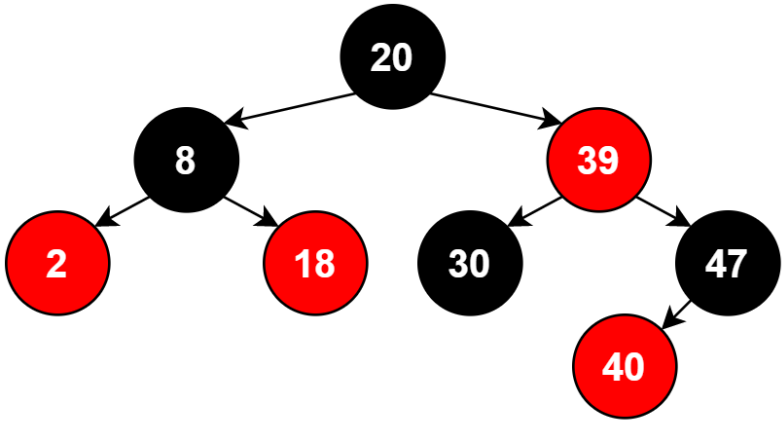


Tree is balanced

Tree maintains invariants. So, recoloring or rotation aren't necessary.

- Step 8 – Adding integer 2

  Perform standart BinarySearchTree add. Adding node is always red.
  After adding, tree:



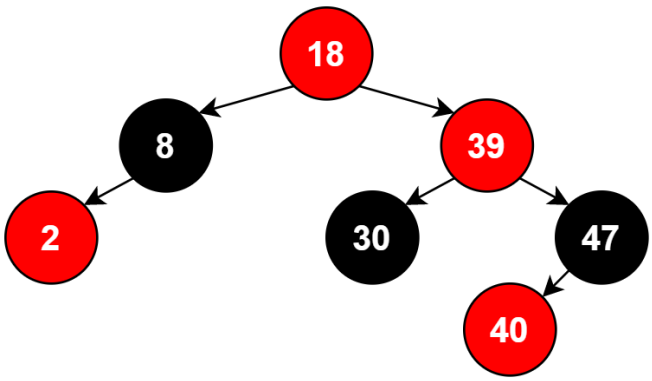
Tree is balanced

Tree maintains invariants. So, recoloring or rotation aren't necessary.

Removing

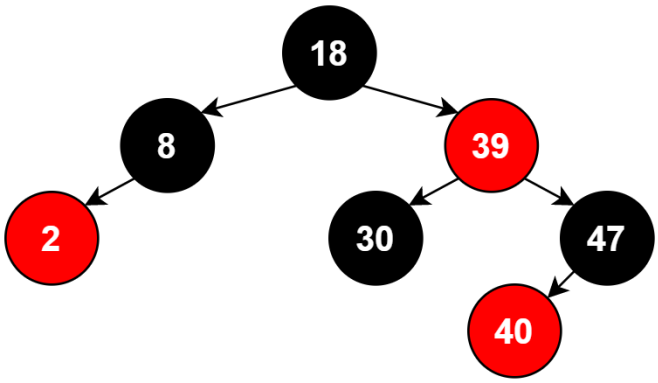
- Step 1 – Removing integer 20

  Perform standard BinarySearchTree delete.
  After operation, tree:



Tree is unbalanced

Because of invariant 2. Invariant 2 is that the root is always black. Since root is red, recolor it to black. After operation, tree:

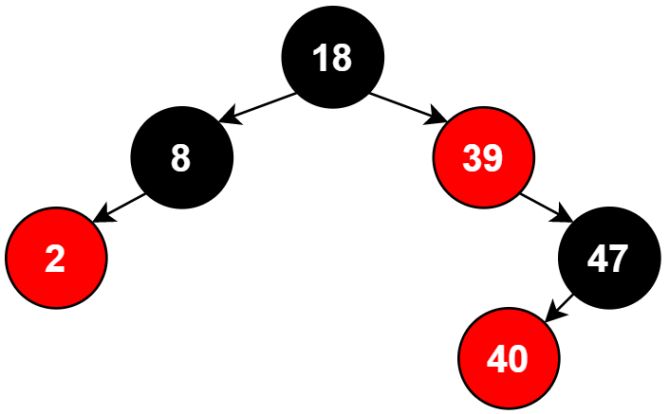


Tree is balanced

Tree maintains invariants. So, recoloring or rotation aren't necessary.

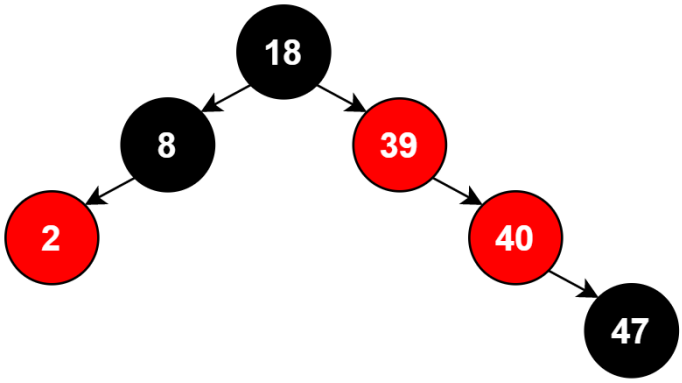
- Step 2 – Removing integer 30

  Perform standart BinarySearchTree delete. Since node 30 is black, fixup required.
  After operation, tree:



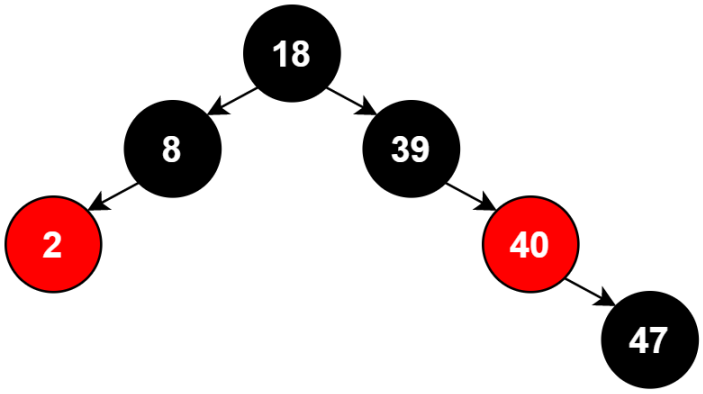
Tree is unbalanced

Because of invariant 4. Invariant 4 is that the number of black nodes in any path form the root to a leaft is the same. Firstly, rotate local root's right(47) to right, Secondly recolor local root(39) to black. Then rotate local root(39) to left.
After rotating root's right(47) to right, tree:



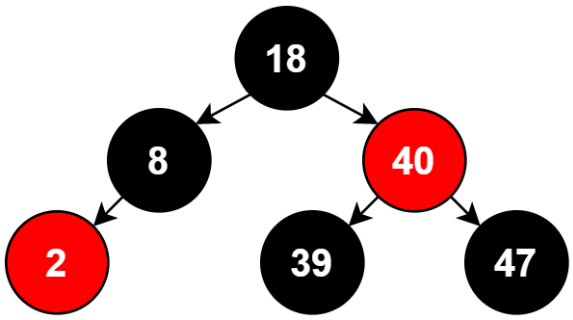
Tree is unbalanced

After recoloring local root(39) to black, tree:



Tree is unbalanced

After rotating local root(39) to left, tree:

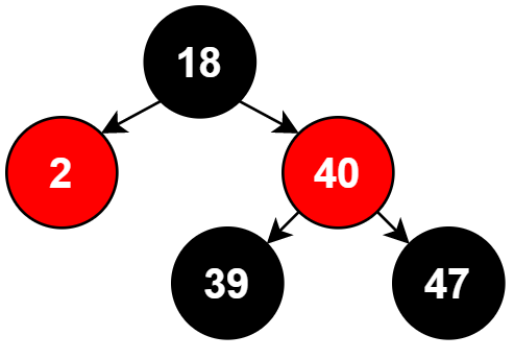


Tree is balanced

Tree maintains invariants. So, recoloring or rotation aren't necessary.

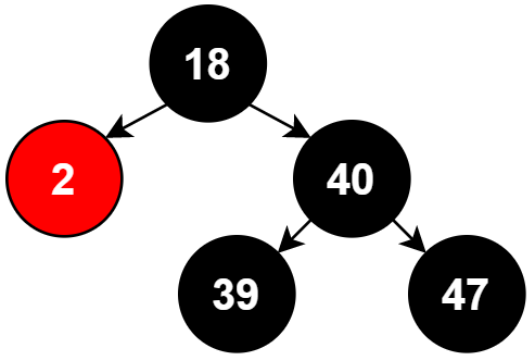
- Step 3 – Removing integer 8

Perform standart BinarySearchTree delete. Since node 8 is black, fixup required.
After operation, tree:



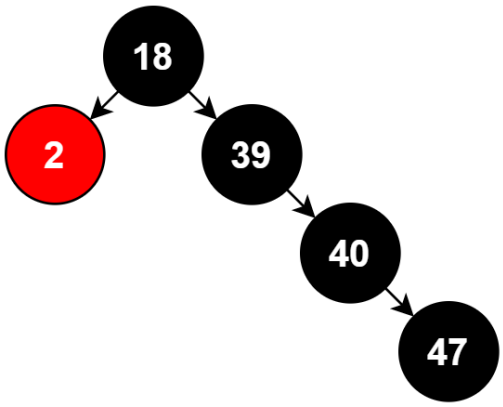
Tree is unbalanced

Because of invariant 4. Invariant 4 is that the number of black nodes in any path form the root to a leaft is the same. Firstly, recolor local root's right(40) to black. Secondly, rotate local root's right(40) to right. Thirdly, recolor local root(18) to red. Then rotate local root to left twice.
After recoloring local root's right(40) to black, tree:



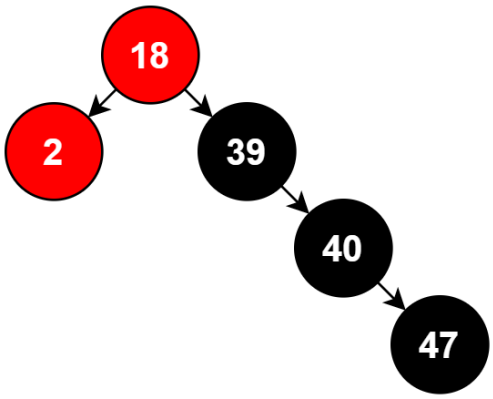
Tree is unbalanced

After rotating local root's right(40) to right, tree:



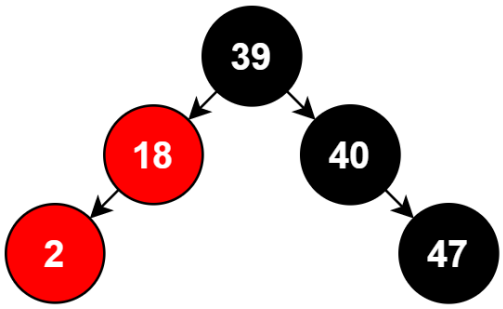
Tree is unbalanced

After recoloring local root(18) to red, tree:



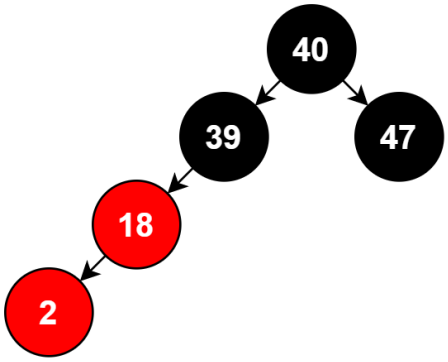
Tree is unbalanced

After rotating local root to left, tree:



Tree is unbalanced

After rotating local root to left, tree:

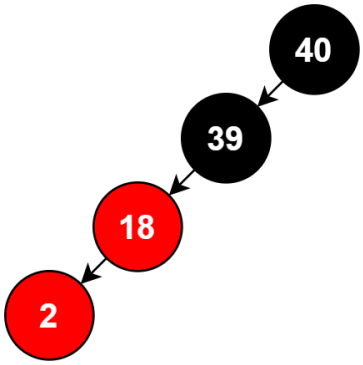


Tree is balanced

Tree maintains invariants. So, recoloring or rotation aren't necessary.

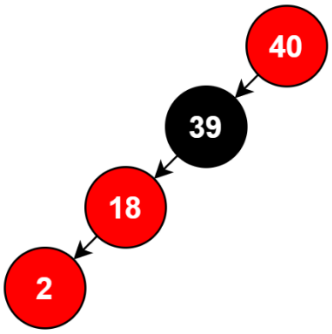
- Step 4 – Removing integer 47

  Perform standart BinarySearchTree delete. Since node 47 is black, fixup reqired.
  After operation, tree:



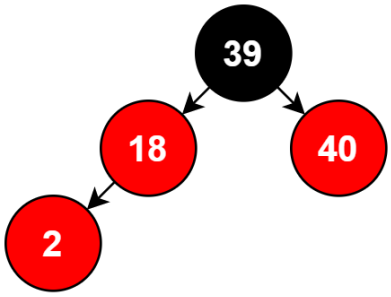
Tree is unbalanced

  Because of invariant 4. Invariant 4 is that the number of black nodes in any path form the root to a leaft is the same. Firstly, recolor local root(40) to red. Then rotate local root(40) to right.
  After recoloring local root(40) to red, tree:



Tree is unbalanced

  After rotating local root(40) to right, tree:

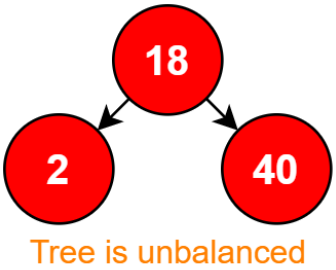


Tree is balanced

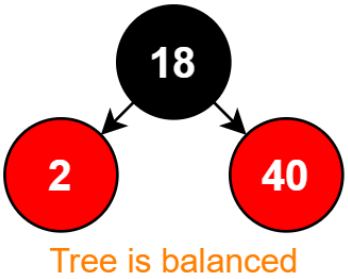
  Tree maintains invariants. So, recoloring or rotation aren't necessary.

- Step 5 – Removing integer 39

  Perform standart BinarySearchTree delete.
  After operation, tree:



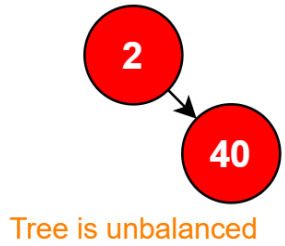
Because of invariant 2. Invariant 2 is that the root is always black. Since root is red, recolor it to black. After operation, tree:



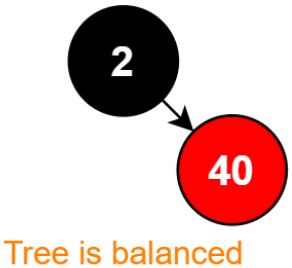
Tree maintains invariants. So, recoloring or rotation aren't necessary.

- Step 6 – Removing integer 18

  Perform standart BinarySearchTree delete.
  After operation, tree:



Because of invariant 2. Invariant 2 is that the root is always black. Since root is red, recolor it to black. After operation, tree:



Tree maintains invariants. So, recoloring or rotation aren't necessary.

- Step 7 – Removing integer 40

  Perform standart BinarySearchTree delete.
  After operation, tree:

Tree is balanced

Tree maintains invariants. So, recoloring or rotation aren't necessary.

- Step 8 – Removing integer 2

  Perform standart BinarySearchTree delete.
  After operation, tree:

2-3 stree

Adding

- Step 1- Adding integer 20

  Local root is empty. Add 20 to root.
  After adding, tree:

  Tree is balanced.

- Step 2- Adding integer 30

  Local root is 2-node. Add 30 to root right side of 20(20 < 30).
  After adding, tree:

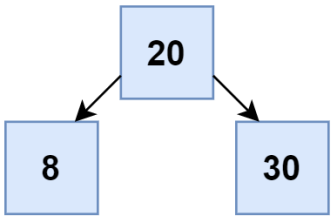
  Tree is balanced.

- Step 3- Adding integer 8

  Local root is 3-node. So, to illustrate add 8 to left side of 20(8 < 20) as virtual(color gray).
  After adding, tree:

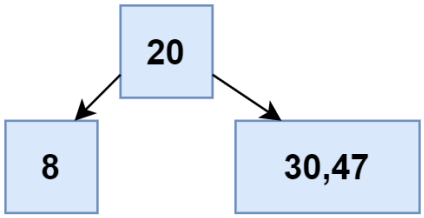
  Tree is unbalanced. We will split the local root. 20 is the middle because of between 8 and 30. So, 20 is their parent.
  After splitting, tree:



Tree is balanced.

- Step 4- Adding integer 47

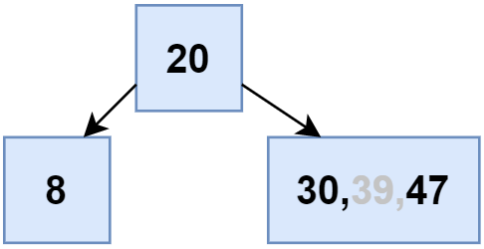
  20 isn't leaf node and 47 > 20. So go on right child. Right child is leaf node and 2-node. Add 47 to right side of 30(30 < 47).
  After adding, tree:



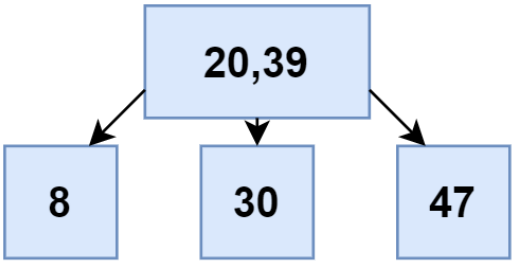
  Tree is balanced.

- Step 5- Adding integer 39

  20 isn't leaf node and 39 > 20. So go on right child. Right child is leaf node and 3-node. To illustrate, add 39 as virtual between 30 and 47(30 < 39 && 39 < 47).
  After adding, tree:



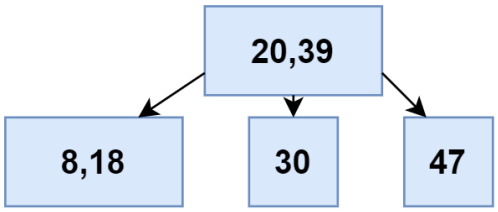
  Tree is unbalanced. We will split the local root. 39 is the middle because of between 30 and 47. So, 39 is their parent.
  After splitting, tree:



  Tree is balanced.

- Step 6- Adding integer 18

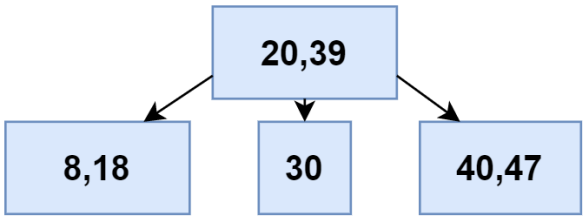
20 and 39 isn't leaf node and 18 < 20. So go on left child. Left child is leaf node and 2-node. Add 18 to right side of 8(8 < 18).
After adding, tree:



Tree is balanced.

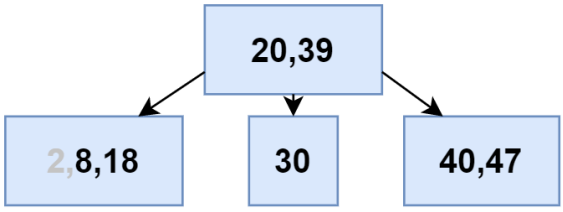
- Step 7- Adding integer 40

20 and 39 isn't leaf node and 40 > 39. So go on right child. Right child is leaf node and 2-node. Add 40 to left side of 47(40 < 47).
After adding, tree:

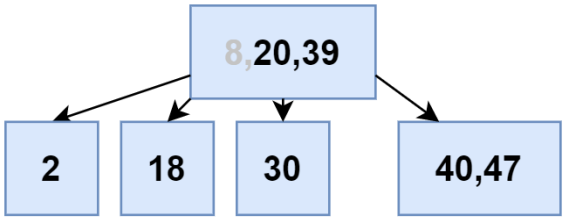


- Step 8- Adding integer 2

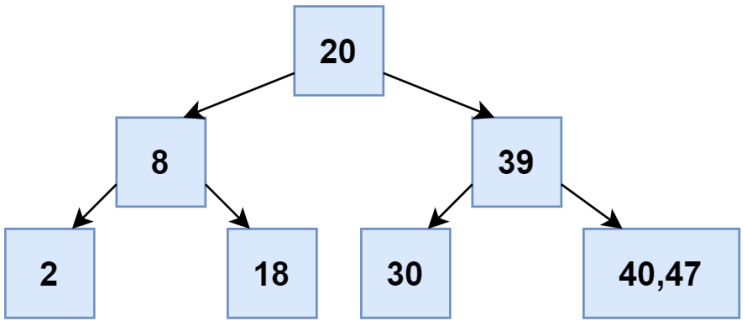
20 and 39 isn't leaf node and 8 < 20. So go on left child. Left child is leaf node but 3-node. To illustrate, add 8 as virtual to left side of 8(2 < 8).
After adding, tree:



Tree is unbalanced. We will split the local root. 8 is the middle because of between 2 and 18. So, 8 is their parent.
After splitting, tree:



Tree is unbalanced. We split the local root. 20 is the middle because of between 8 and 39. So, 20 is their parent.
After splitting, tree:

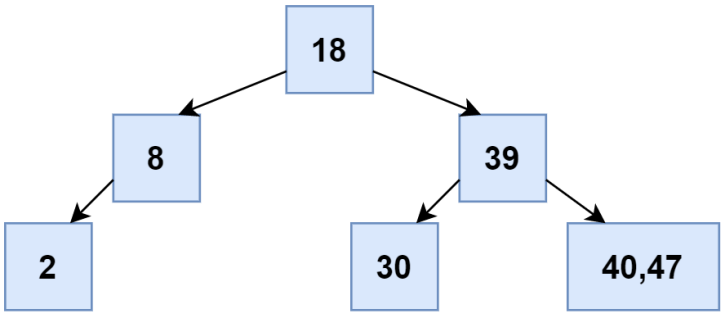


Tree is balanced.

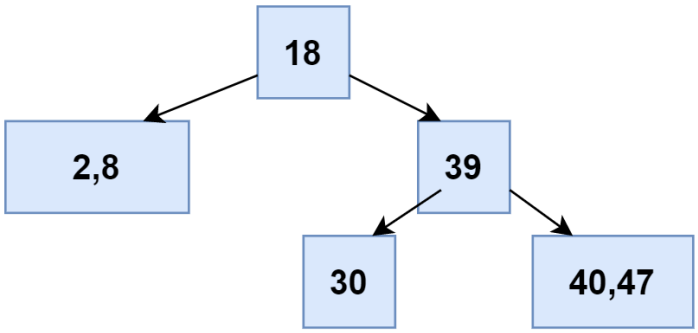
Removing

- Step 1 – Removing integer 20

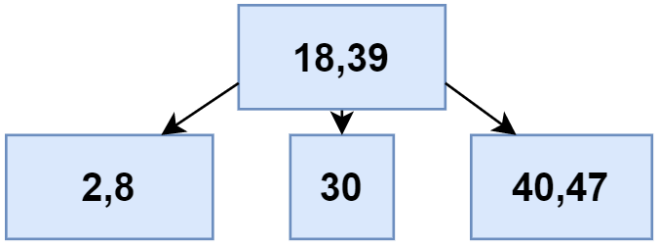
  20 isn't leaf local root. So, we remove it by swapping it with its inorder predecessor in a leaf node and deleting it from the leaf node.
  After removing, tree:



  Tree is unbalanced. Firstly, parent(8) and its left siblings(2) are merged.
  After merging, tree:



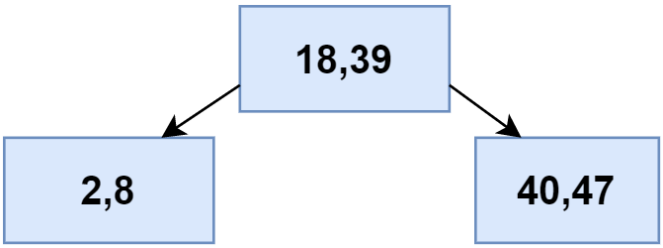
  Tree is unbalanced. This has the effect of deleting 8 from the next higher level. Therefore, the process repeats, and parent(18) and 18's right child(39) are merged.
  After merging, tree:



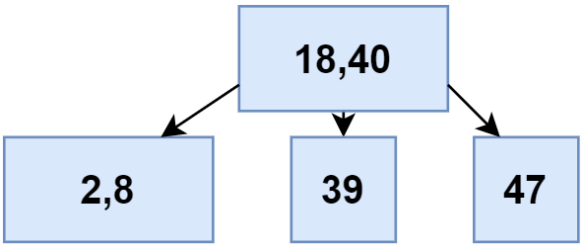
  Tree is balanced.

- Step 2 – Removing integer 30

  Perform search operation to delete 30. 30 > 18 and 30 < 39. Therefore go on middle child. 30 is a leaf node. Remove it.
  After removing, tree:



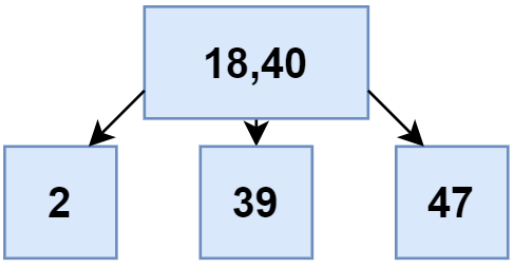
  Tree is unbalanced. We can merge 39 and its right child to from the virtual leaf node {39,40,47}. Item 40 moves up to the parent node. Item 39 is the new left child of 40.
  After operation, tree:



  Tree is balanced.

- Step 3 – Removing integer 8

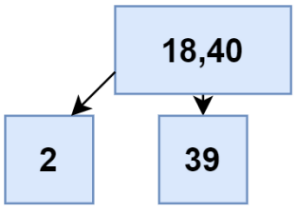
  Perform search operation to delete 8. 8 < 18. Therefore go on left child. 2,8 is a leaft node. Remove 8.
  After removing, tree:



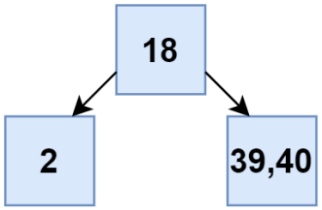
  Tree is balanced.

- Step 4 – Removing integer 47

  Perform search operation. 47 > 18 and 47 > 40. Therefore, go on right child. 47 is a leaf node. Remove 47.
  After removing, tree:



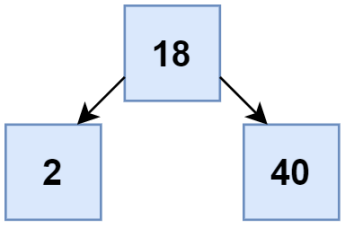
Tree is unbalanced. We can merge 39 and 40.
After merging, tree:



Tree is balanced.

- Step 5 – Removing integer 39

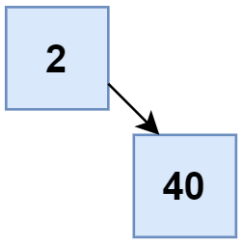
  Perform search operation to remove 39. 18 < 39. Therefore, go on right child. 39,40 is a leaft node. Remove 39.
  After removing, tree:



Tree is balanced.

- Step 6 – Removing integer 18

  18 isn't leaf local root. So, we remove it by swapping it with its inorder predecessor in a leaf node and deleting it from the leaf node.
  After removing, tree:



Tree is unbalanced. This has the effect of deleting 2 from the next higher level. Therefore, the process repeats, and parent(2) and 2's right child(40) are merged.
After merging, tree:

Tree is balanced.

- Step 7 – Removing integer 40

  2,40 is a leaf node. Therefore, remove 40.
  After removing, tree:

  Tree is balanced.

- Step 8 – Removing integer 2

  2 is a leaf node. Therefore, remove 2.
  After removing, tree:

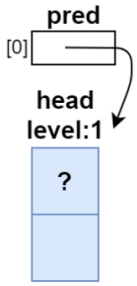
Skip List

We define a common method,search, which will return an array pred of references to the SLNodes at each level that were last examined in the search. Because array subscripts start at 0, pred[i] references the predecessor in the level-(i+1) list of the target. The level-(i+1) link for the node referenced by pred[i] is greater than or equal to the target or is null.

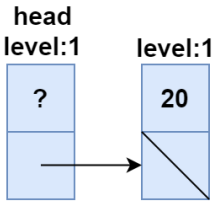
Adding

- Step 1 – Adding integer 20

  Before adding, determine pred array

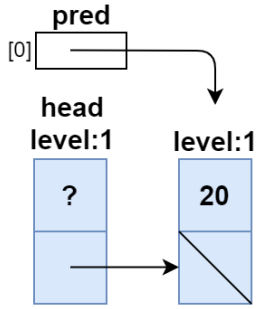
  

  After adding, list:

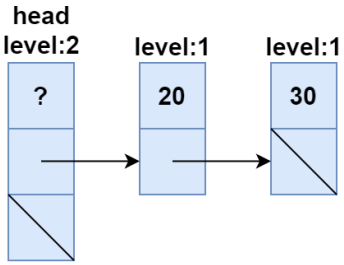
  

- Step 2 – Adding integer 30

Before adding, determine pred array

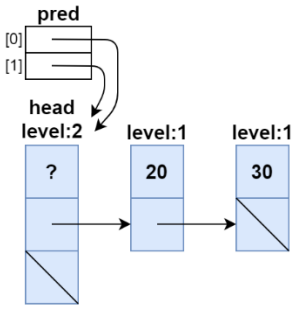


After adding, list:

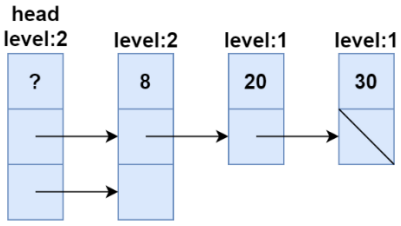


- Step 3 – Adding integer 8

Before adding, determine pred array

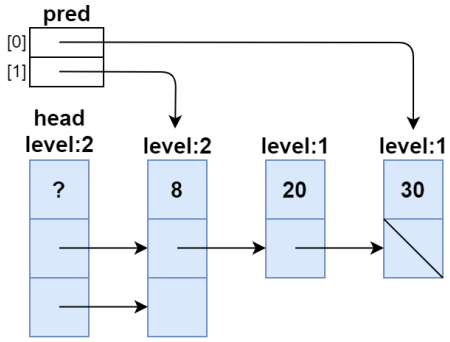


After adding, list:

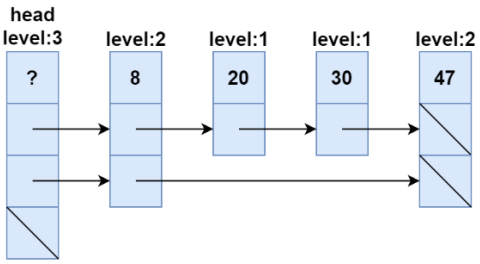


- Step 4 – Adding integer 47

Before adding, determine pred array

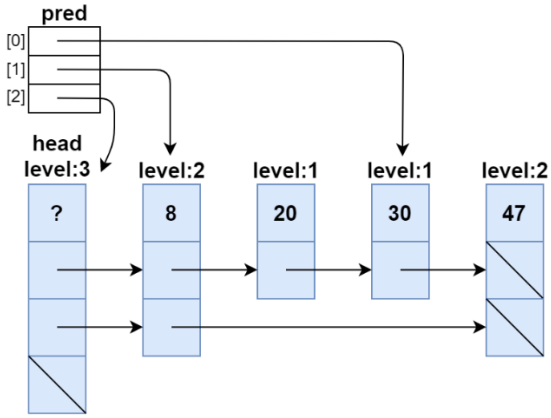


After adding, list:

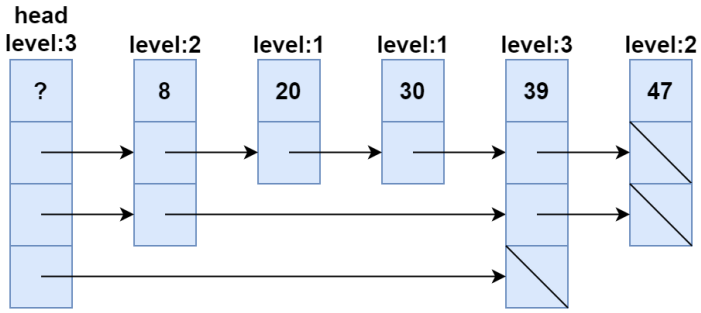


- Step 5 – Adding integer 39

  Before adding, determine pred array

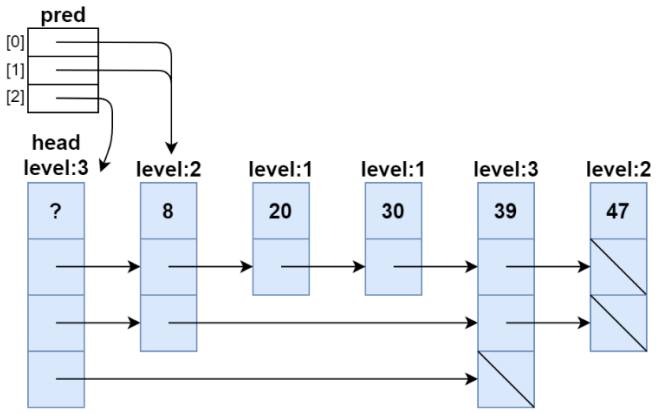


  After adding, list:

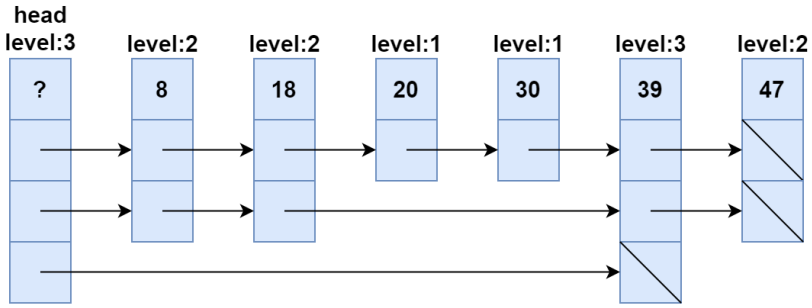


- Step 6 – Adding integer 18

  Before adding, determine pred array

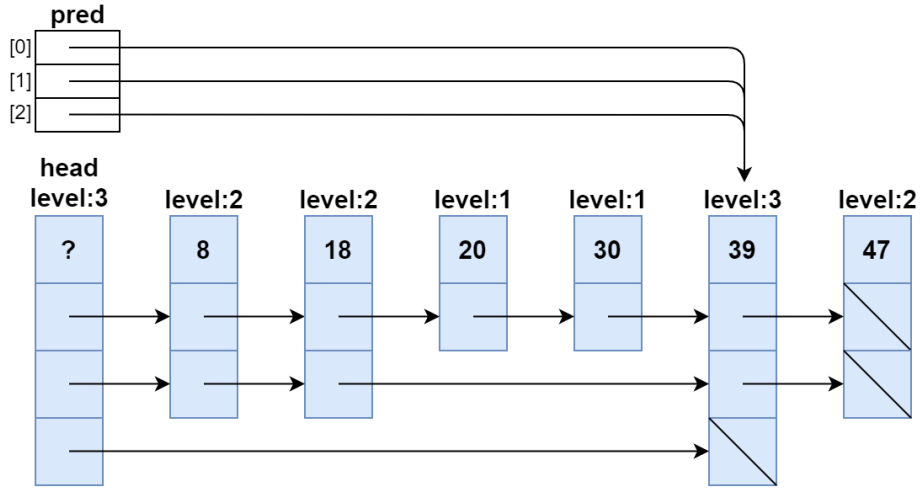


After adding, list:

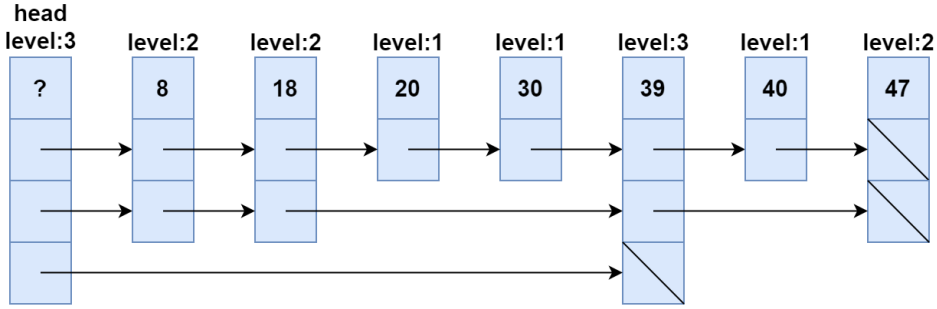


- Step 7 – Adding integer 40

Before adding, determine pred array

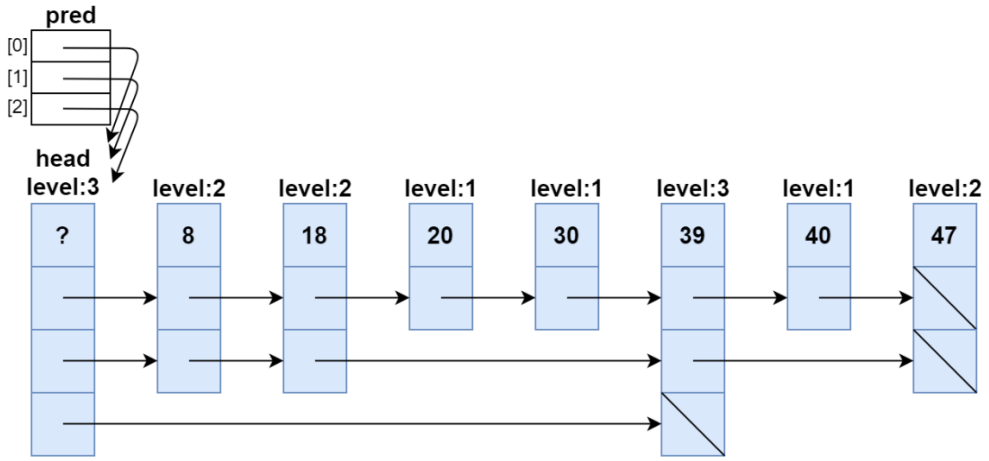


After adding, list:

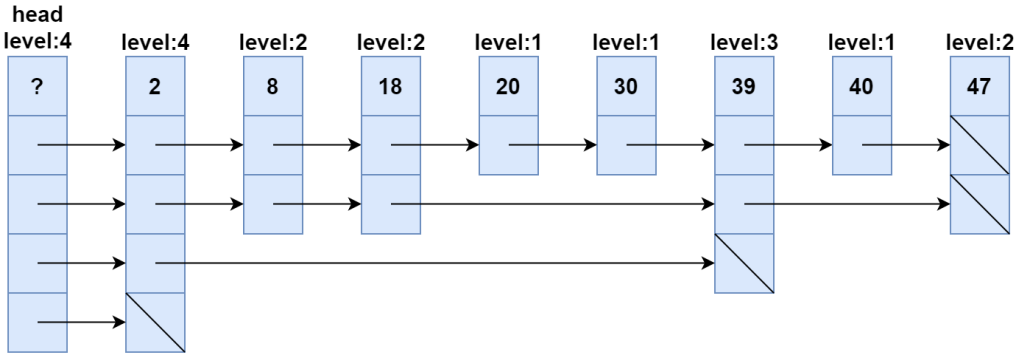


- Step 8 – Adding integer 2

  Before adding, determine pred array



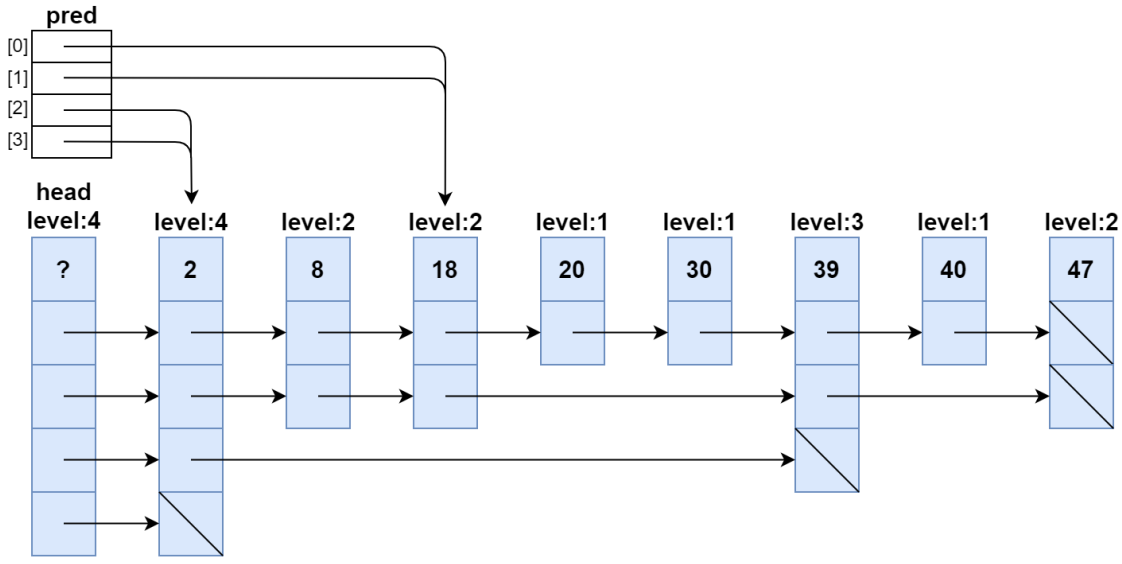
  After adding, list:



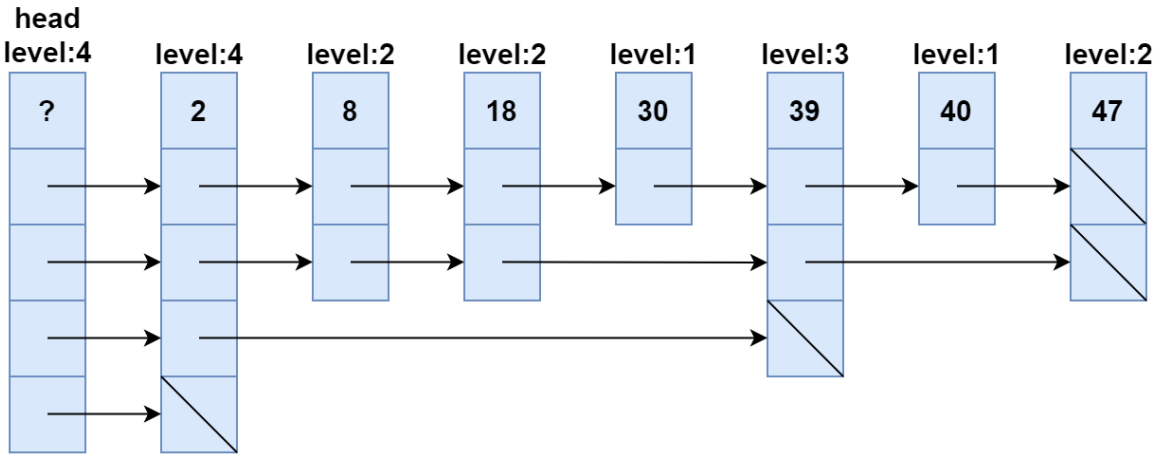
Removing

- Step 1 – Removing integer 20

  Before removing, determine pred array

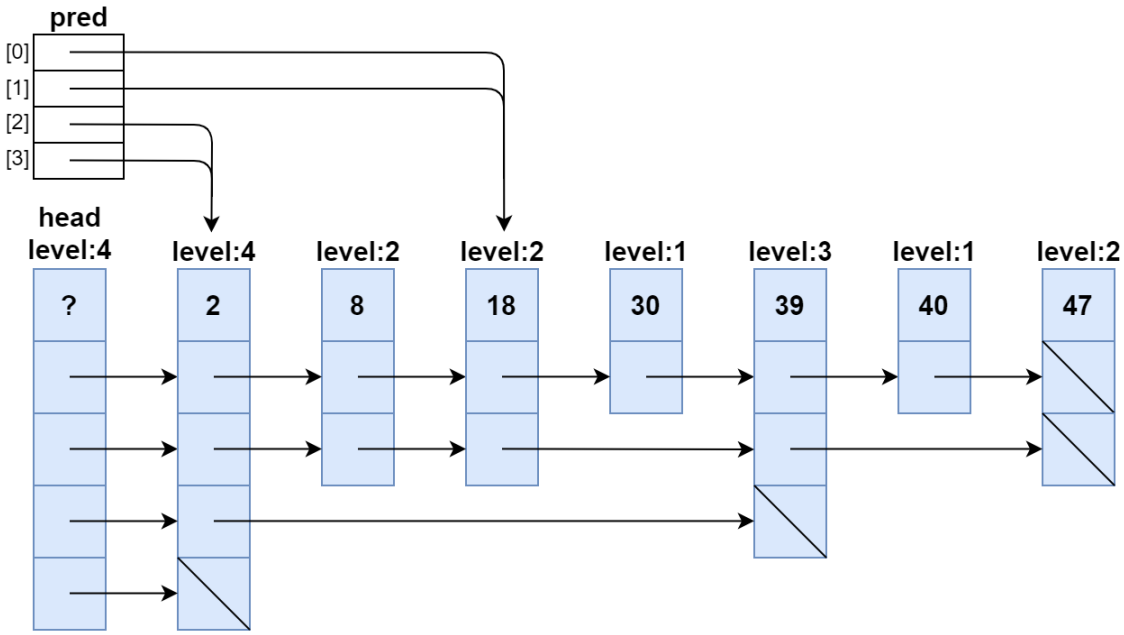


After removing, list:

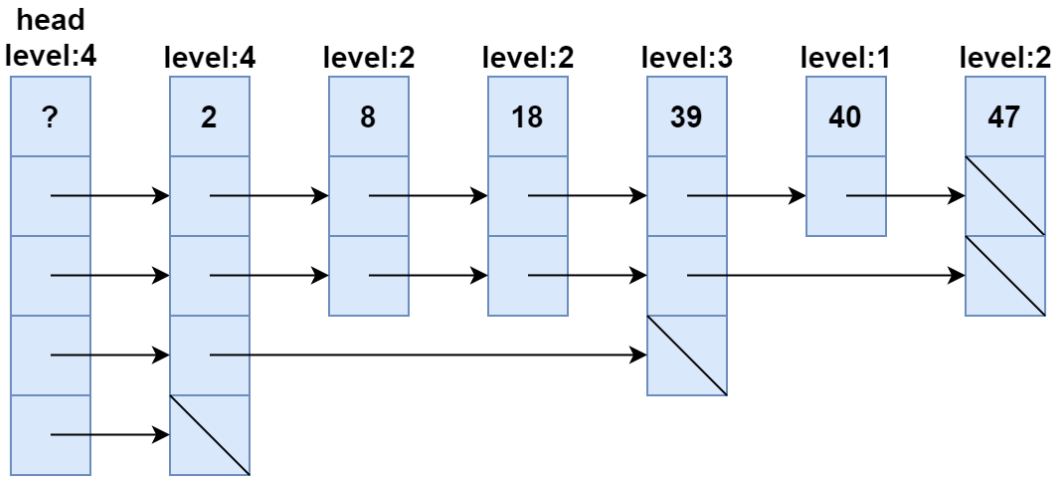


- Step 2 – Removing integer 30

Before removing, determine pred array

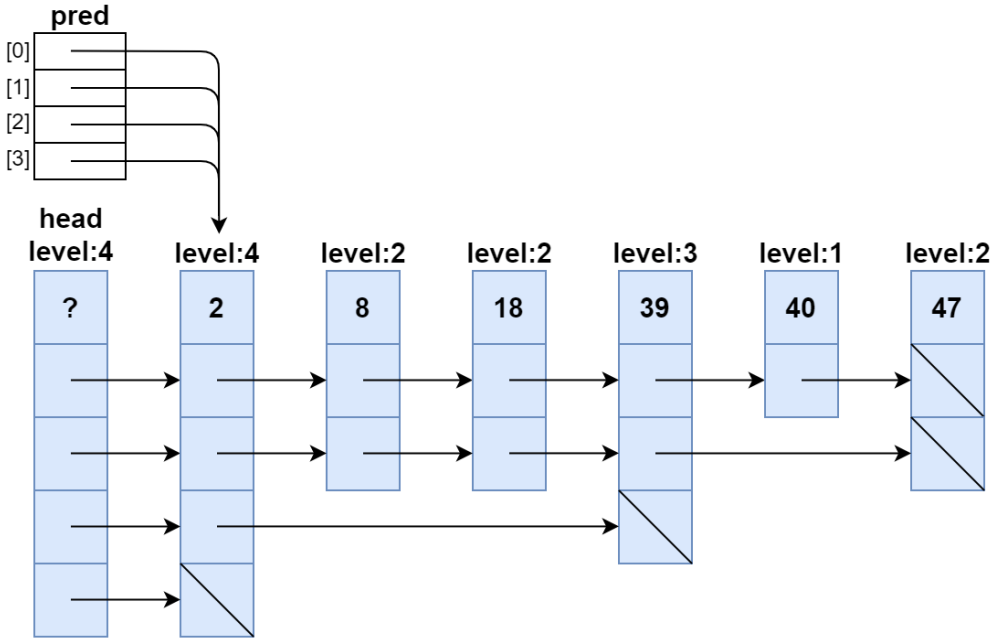


After removing, list:

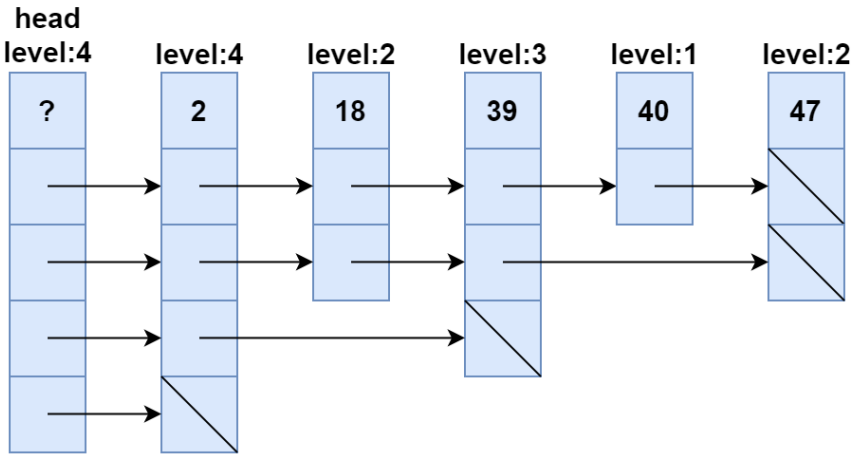


- Step 3 – Removing integer 8

  Before removing, determine pred array

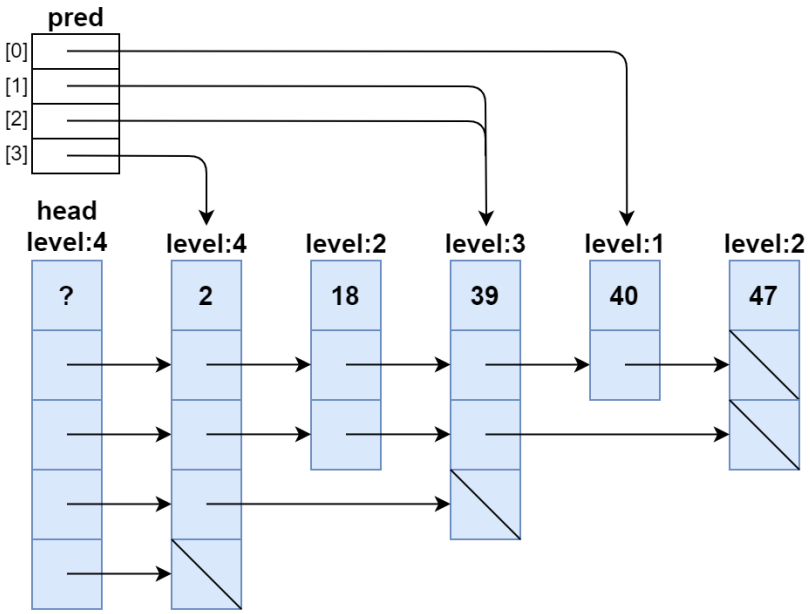


  After removing, list:

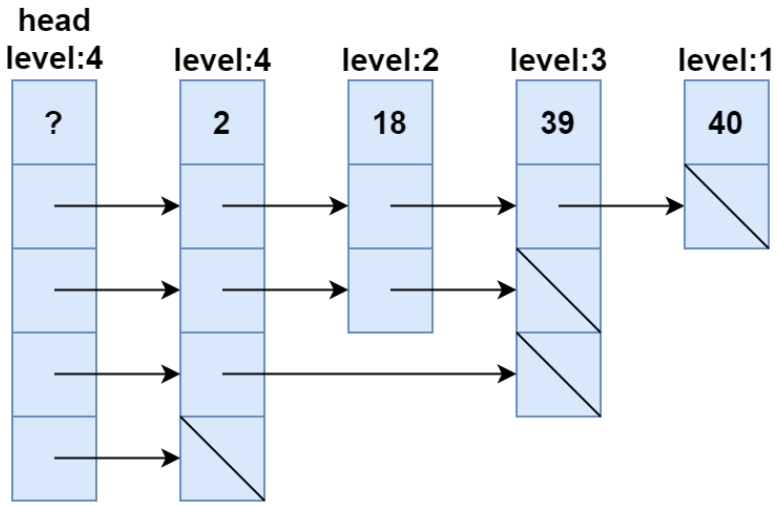


- Step 4 – Removing integer 47

  Before removing, determine pred array

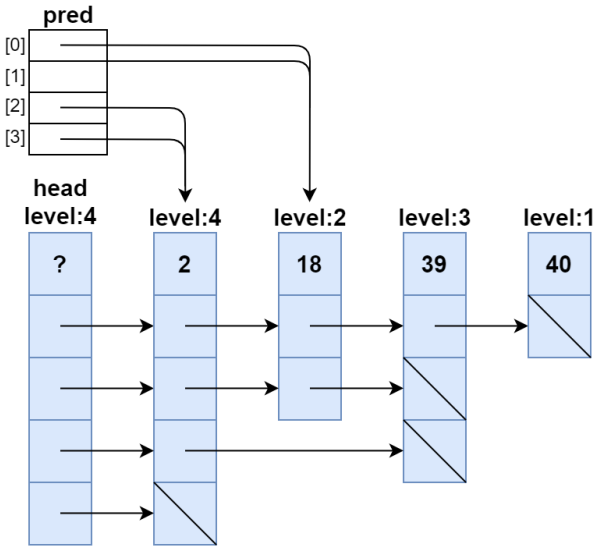


After removing, list:

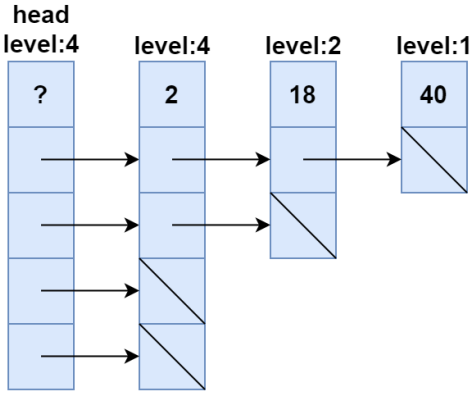


- Step 5 – Removing integer 39

  Before removing, determine pred array

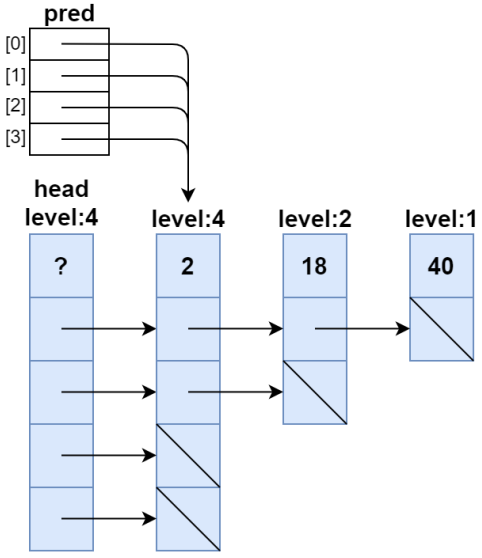


After removing, list:

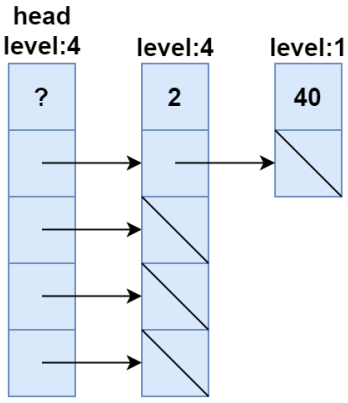


- Step 6 – Removing integer 18

  Before removing, determine pred array

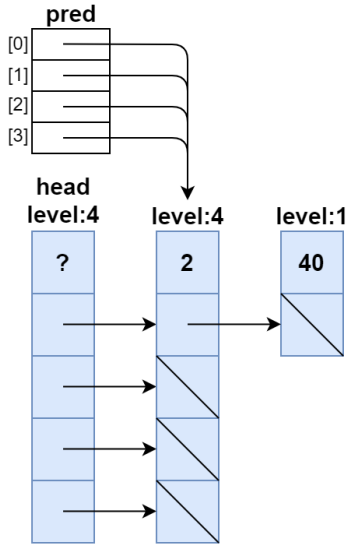


  After removing, list:

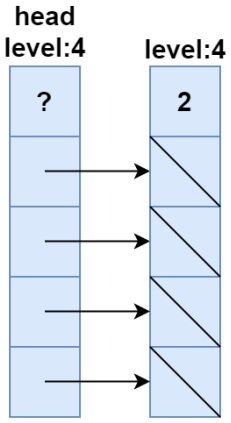


- Step 7 – Removing integer 40

  Before removing, determine pred array

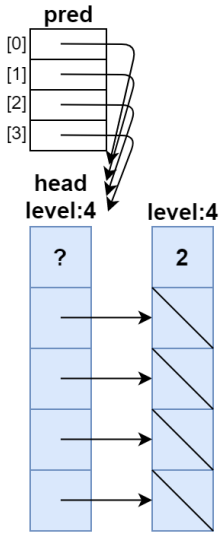


  After removing, list:

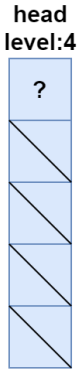


- Step 8 – Removing integer 2

  Before removing, determine pred array



After removing, list:



B-Tree with order 4

Adding

- Step 1 – Adding integer 20

  Root is empy. Create root and add 20 to root.
  After adding, tree:

  Tree is balanced.

- Step 2 – Adding integer 30

  Local root is a leaf node and size is smaller than order. Add 30 to local root.
  After adding, tree:

Tree is balanced.

- Step 3 – Adding integer 8

  Local root is a leaf node and size is smaller than order. Add 8 to local root.
  After adding, tree:

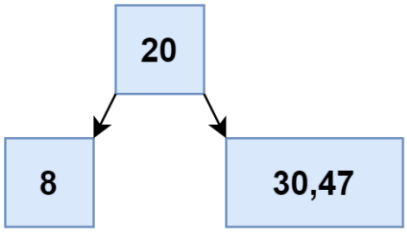
  Tree is balanced.

- Step 4 – Adding integer 47

  Local root is a leaf node but size is not smaller than order. To illustrate, add 47 as virtual to local root.
  After adding, tree:

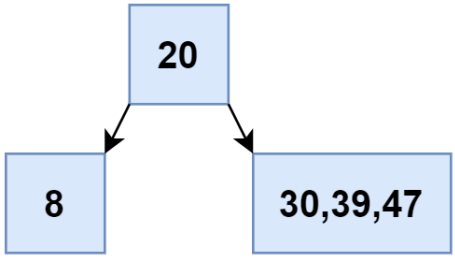
  Tree is unbalanced because of size. 20 splits the node because of middle element.
  After splitting, tree:

  Tree is balanced.

- Step 5 – Adding integer 39

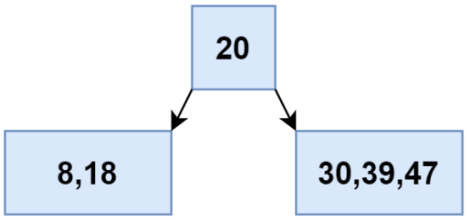
  Local root is not a leaf node. 39 > 29. So, go on right child. Right child is a leaf node and size is smaller than order. Add 39 simplfy.
  After adding, tree:

  Tree is balanced.

- Step 6 – Adding integer 18

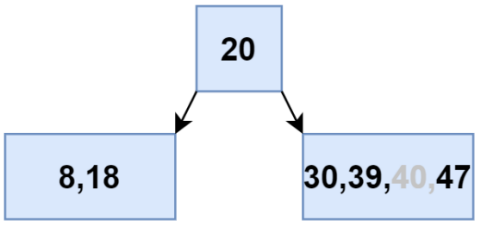
Local root is not a leaf node. 18 < 20. So, go on left child. Left child is a leaf node and size is smaller than order. Add 18 to left child.
After adding, tree:



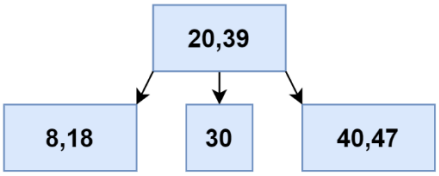
Tree is balanced

- Step 7 – Adding integer 40

Local root is not a leaf node. 40 > 30. So, go on right child. Right child is a leaf node but size is not smaller than order. To illustrate, add 40 as virtual to right child {30,39,40,47}. Then middle element(39) goes parent and splits node by two.
After adding, tree:



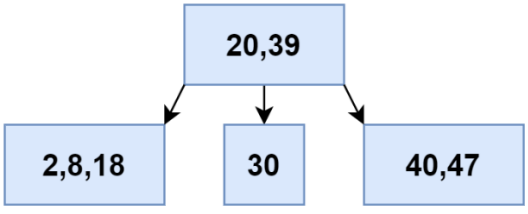
Tree is unbalanced.



Tree is balanced.

- Step 8 – Adding integer 2

Local root is not a leaf node. 2 < 20. So, go on left child. Left child is a leaf node and size is smaller than order. Add 2 to left child.
After adding, tree:

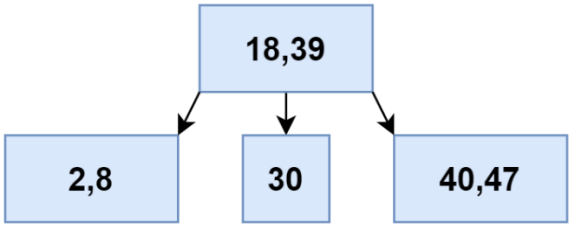


Tree is balanced.

Removing

- Step 1 – Removing integer 20

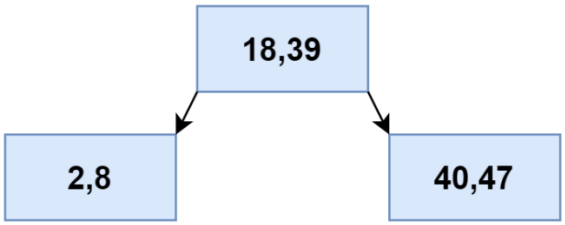
Perform search operation. Node with 20 isn't a leaf node. So we remove it by swapping its inorder predecessor(18) in a leaf node and deleting it from the leaf node.
After removing, tree:



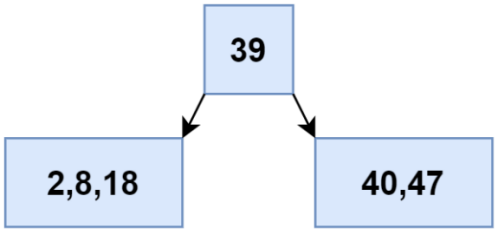
Tree is balanced.

- Step 2 – Removing integer 30

Perform search operation. 30 > 18 and 30 < 39. So, go on middle child. Node with 30 is a leaf node. Remove 30 simplfy.
After removing, tree:



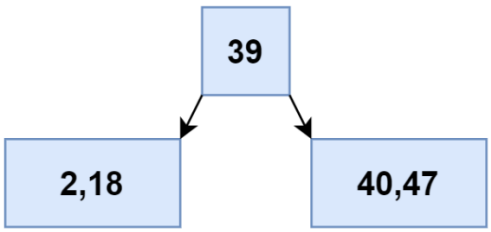
Tree is unbalanced from middle child's size. Therefore, we merge 30's parent, and its left sibling {2,8,18}.
After merging, tree:



Tree is balanced.

- Step 3 – Removing integer 8

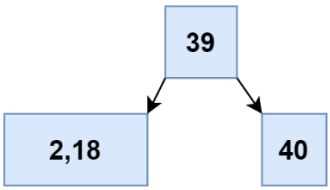
Perform search operation. 8 < 39. So, go on left child. Node with 8 is a leaf node. Remove 8 simplfy.
After removing, tree:



Tree is balanced.

- Step 4 – Removing integer 47

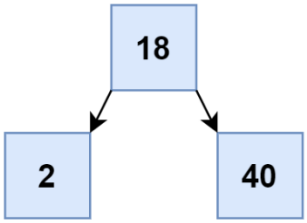
  Perform search operation. 47 > 39. So, go on right child. Node with 47 is a leaf node. Remove 47 simplyf.
  After removing, tree:



  Tree is balanced.

- Step 5 – Removing integer 39

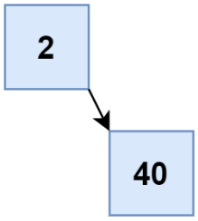
  Perform search operation. Node with 39 isn't a leaf node. So we remove it by swapping its inorder predecessor(18) in a leaf node and deleting it from the leaf node.
  After removing, tree:



  Tree is balanced.

- Step 6 – Removing integer 18

  Perform search operation. Node with 18 isn't a leaft node. So we remove it by swapping its inorder predecessor(18) in a leaf node and deleting it from the leaf node.
  After removing, tree:



  Tree is unbalanced. Because, after swapping and removing, node size became 0. Therefore, we merge it's parent(2) and right subling(40).
  After merging, tree:

Tree is balanced.

- Step 7 – Removing integer 40

  Perform search operation. Node with 40 is a leaf node. Remove 40 simplfy.
  After removing, tree:

  Tree is balanced.

- Step 8 – Removing integer 2

  Perform search operation. Node with 2 is a leaf node. Remove 2 simplfy.
  After removing, tree:
