Balanced Trees

Data Structures and Algorithms

ArfanShahzadTech

WhatsApp-Contact Us
0345-5922495

Arfan Shahzad

{ arfanskp@gmail.com }

Data Structures and Algorithms

Course Contents:

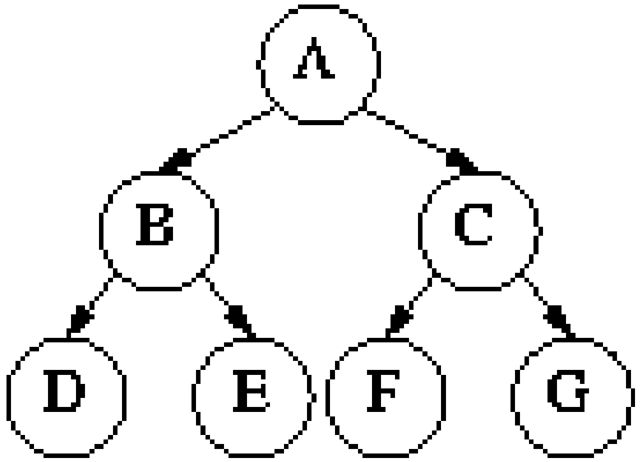
Abstract data types, complexity analysis, Big Oh notation, Stacks (linked lists and array implementations), Recursion and analyzing recursive algorithms, divide and conquer algorithms, Sorting algorithms (selection, insertion, merge, quick, bubble, heap, shell, radix, bucket), queue, dequeuer, priority queues (linked and array implementations of queues), linked list & its various types, sorted linked list, searching an unsorted array, binary search for sorted arrays, hashing and indexing, open addressing and chaining, trees and tree traversals, binary search trees, heaps, M-way tress, balanced trees, graphs, breadth-first and depth-first traversal, topological order, shortest path, adjacency matrix and adjacency list implementations, memory management and garbage collection

Balanced Trees

- We have seen that the efficiency of many important operations on trees is related to the Height of the tree - for example searching, inserting, and deleting in a BST are all O(Height).
- In general, the relation between Height (H) and the number of nodes (N) in a tree can vary from H = N (degenerate tree) to H = log(N).

Balanced Trees cont…

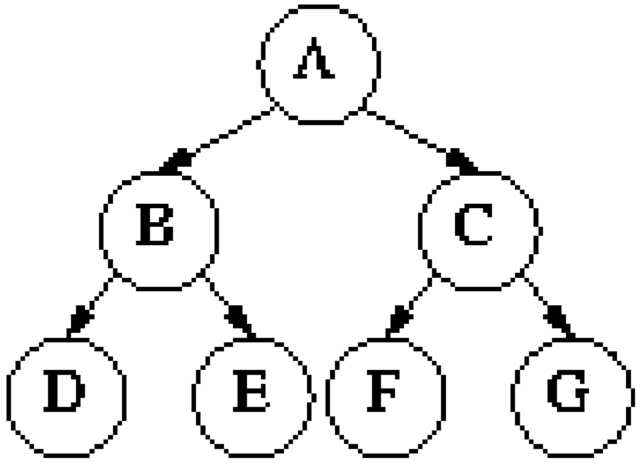
- For efficiency's sake, we would like to guarantee that H was O(logN).
- One way to do this is to force our trees to be height-balanced.
- A tree is *perfectly* height-balanced if the left and right subtrees of any node are the same height. e.g.



Balanced Trees cont…

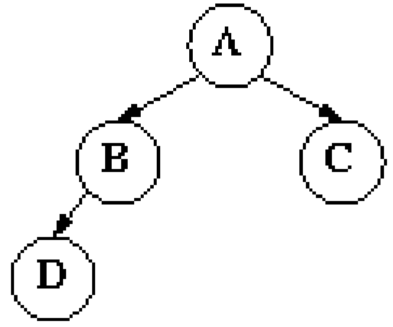
- It is clear that at every level there are twice as many nodes as at the previous level, so we do indeed get H = O(logN).
- However, perfect height balance is very rare: it is only possible if there are exactly 2^(H+1)-1 nodes!

$$\boxed{2^{h+1}-1}$$



Balanced Trees cont...

- As a practical alternative, we use trees that are `almost' perfectly height balanced.
- We will say that a tree is height-balanced if the heights of the left and right subtree's of each node are within 1. The following tree fits this definition:

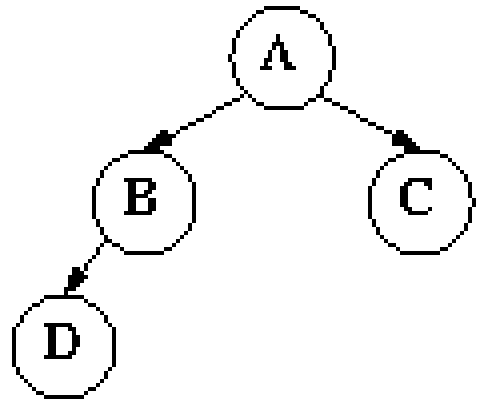


- We will say this tree is height-balanced.

Balanced Trees cont…

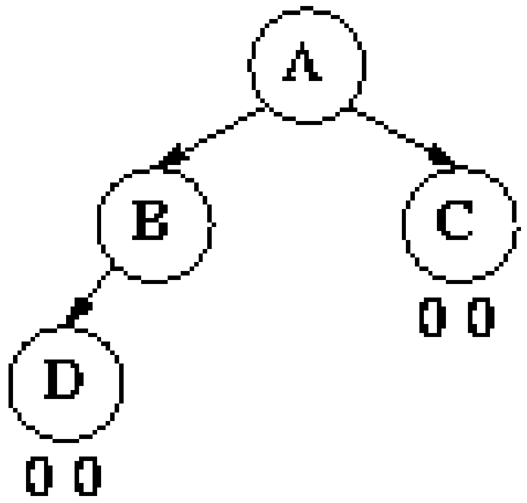
- How can we tell if a tree is height-balanced?
- We have to check every node.
- The fastest way is to start at the leaves and work your way up.
- When you reach a node, you will know the heights of its two subtrees; from that you can tell whether it is height-balanced or not.

Balanced Trees cont…



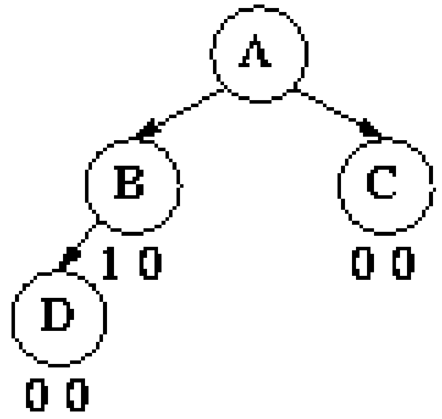
- For example, in the tree above:

- C and D are leaves, and their subtrees are all height 0 so C and D are both perfectly balanced.



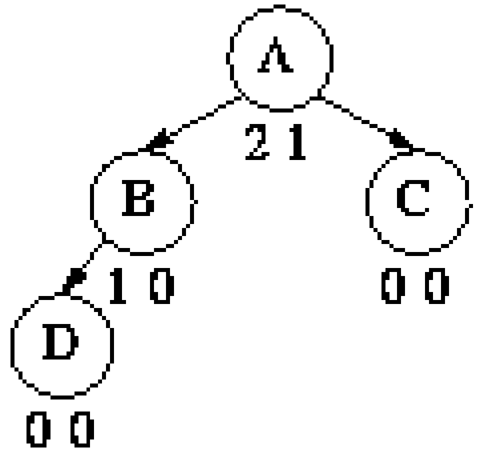
Balanced Trees cont…

- Having finished D we can compute the heights of B's subtrees.
- B is not perfectly balanced, but the heights of of its subtrees differ only by 1, so B is regarded as height-balanced.



Balanced Trees cont…

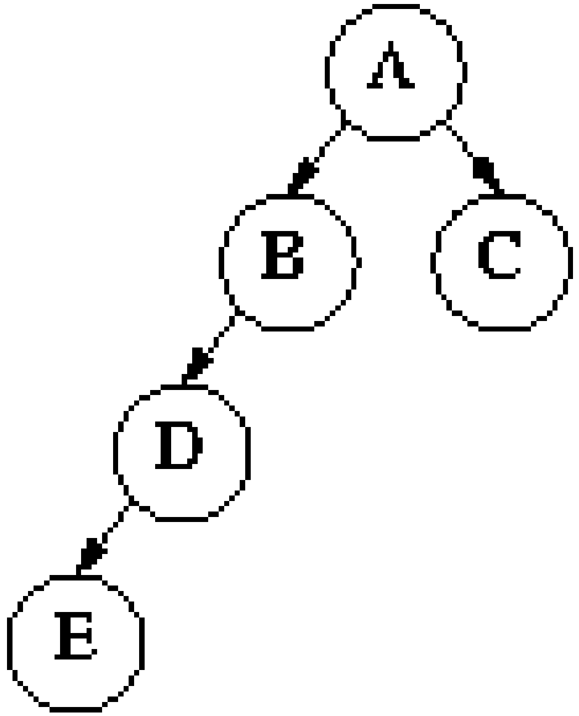
- Now we can compute the balance of A.
- Like B, A's two subtrees also differ by 1 in height.
- We have now looked at every node; every one is height-balanced, so the tree as a whole is considered to be height-balanced.



Balanced Trees cont…

- What about this tree - is it height-balanced?

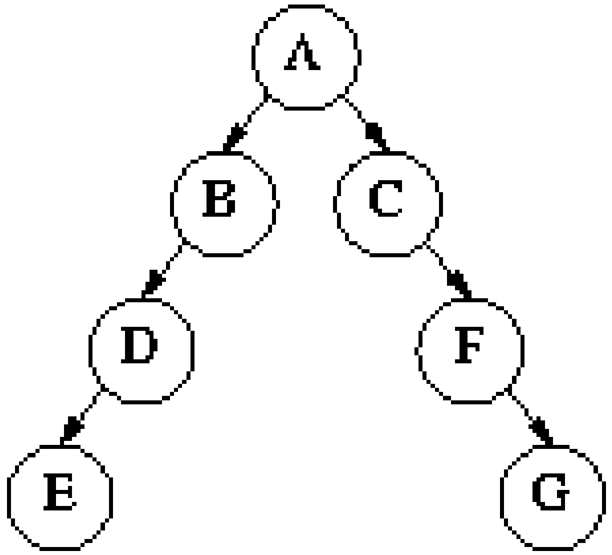
- **Answer is No**



Balanced Trees cont…

- Finally, what about this one?

- **Answer is No: check node b, and c**



Balanced Trees cont…

- Tree is said to be height balanced if balance factor of each node is in between **-1 to 1**, otherwise, the tree will be unbalanced and need to be balanced.
- Balance Factor (k) = height (left(k)) - height (right(k))

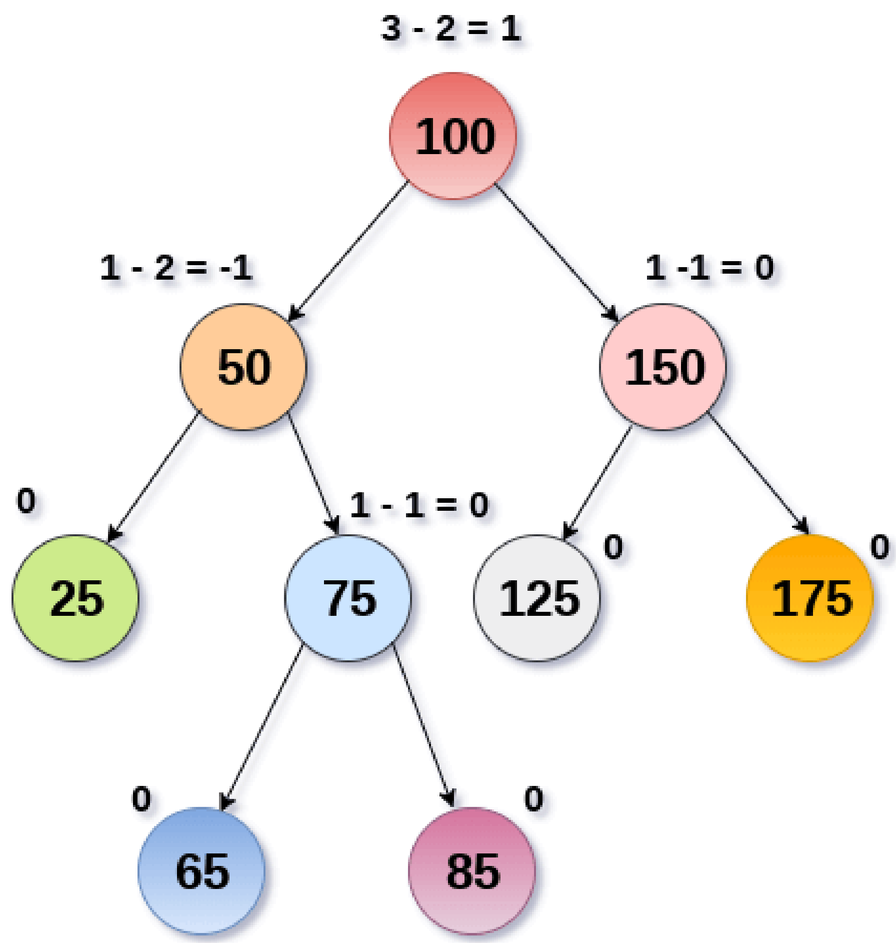
Balanced Trees cont…

- Tree is said to be height balanced if balance factor of each node is in between **-1 to 1**, otherwise, the tree will be unbalanced and need to be balanced.
- Balance Factor (k) = height (left(k)) - height (right(k))

Balanced Trees cont…

The balance factor associated with each node in this figure is between -1 and +1.

Therefore, it is an example of balance tree.



Balanced Trees cont…

- The following are the types of balanced trees:

1. **AVL Tree**
2. **Red Black Tree**
3. **Splay Tree**
4. **Treap**
5. **B Tree**
6. **B+ Tree**

Balanced Trees cont…
Tree (AVL Tree)

- AVL Tree is invented by GM Adelson, Velsky and Landis in 1962.
- The tree is named AVL in honour of its inventors.
- AVL Tree can be defined as ***height balanced*** binary search tree in which each node is associated with a balance factor which is calculated by subtracting the ***height*** of its ***right sub-tree*** from that of its ***left sub-tree***.

Balanced Trees cont…
Tree (AVL Tree – Operations on AVL)

- Due to the fact that, AVL tree is also a binary search tree; therefore, all the operations are performed in the same way as they are performed in a ***binary search tree***.
- Searching and traversing do not lead to the violation in property of AVL tree.

Balanced Trees cont…
Tree (AVL Tree – Operations on AVL)

- However, ***insertion*** and ***deletion*** are the operations which can violate the property of balance factor, and therefore, they need to be addressed here carefully.
- The tree can be balanced again after insertion or deletion by applying various techniques of ***rotations***.

Balanced Trees cont…

Tree (AVL Tree – Operations on AVL - Rotations)

- We perform rotation in AVL tree only in case if Balance Factor is other than -1, 0, and 1.
- There are basically four types of rotations which are as follows:

Balanced Trees cont…

Tree (AVL Tree – Operations on AVL - Rotations)

- **LL rotation:** Inserted node is in the *left subtree* of *left subtree* of A (left skewed).
- **RR rotation:** Inserted node is in the *right subtree* of *right subtree* of A (right skewed).
- **LR rotation:** Inserted node is in the *right subtree* of *left subtree* of A (having shape <).
- **RL rotation:** Inserted node is in the *left subtree* of *right subtree* of A (having shape >).

Balanced Trees cont…

Tree (AVL Tree – Operations on AVL - Rotations)

- The first two rotations **LL** and **RR** are ***single rotations*** and the next two rotations **LR** and **RL** are ***double rotations***.
- For a tree to be **unbalanced**, minimum ***height must be at least 2***, Let us understand each rotation.

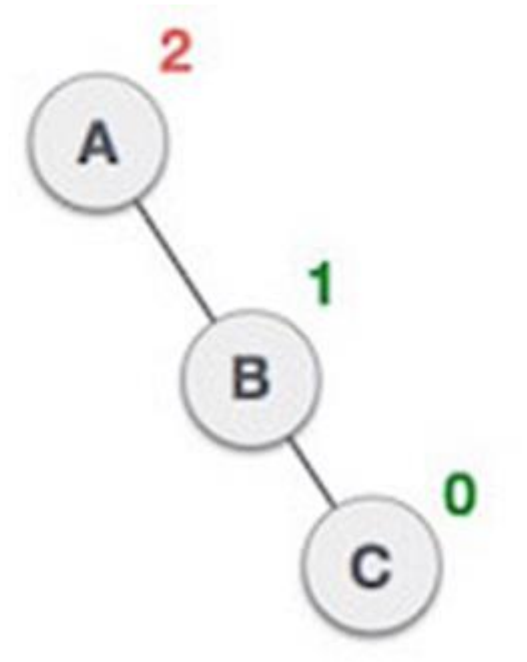
Balanced Trees cont…

Tree (AVL Tree – Operations on AVL - Rotations-RR)

- When BST becomes unbalanced, due to a node is inserted into the ***right subtree*** of the ***right subtree*** of A, then we perform **RR rotation**.
- RR rotation, is an **anticlockwise** rotation, which is applied on the **edge below** a node having ***balance factor -2***.

Balanced Trees cont…

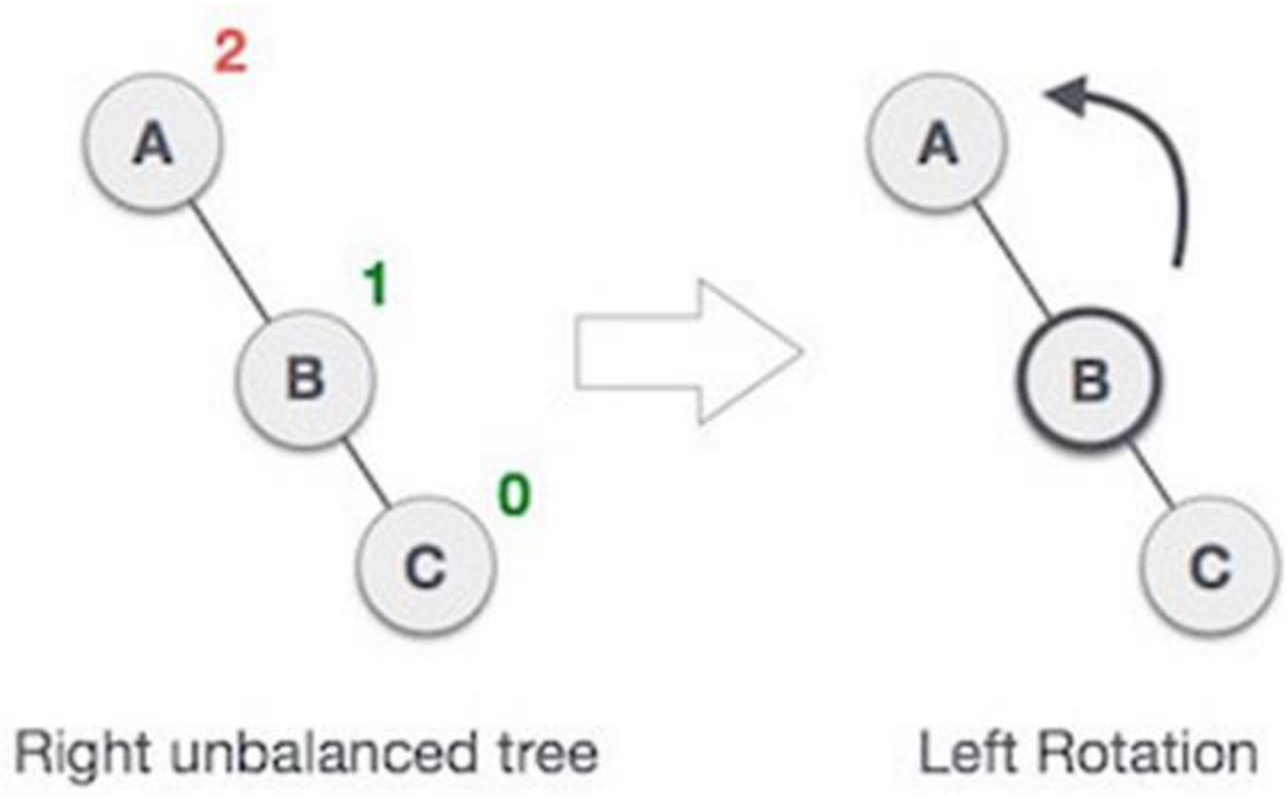
Tree (AVL Tree – Operations on AVL - Rotations-RR)



Right unbalanced tree

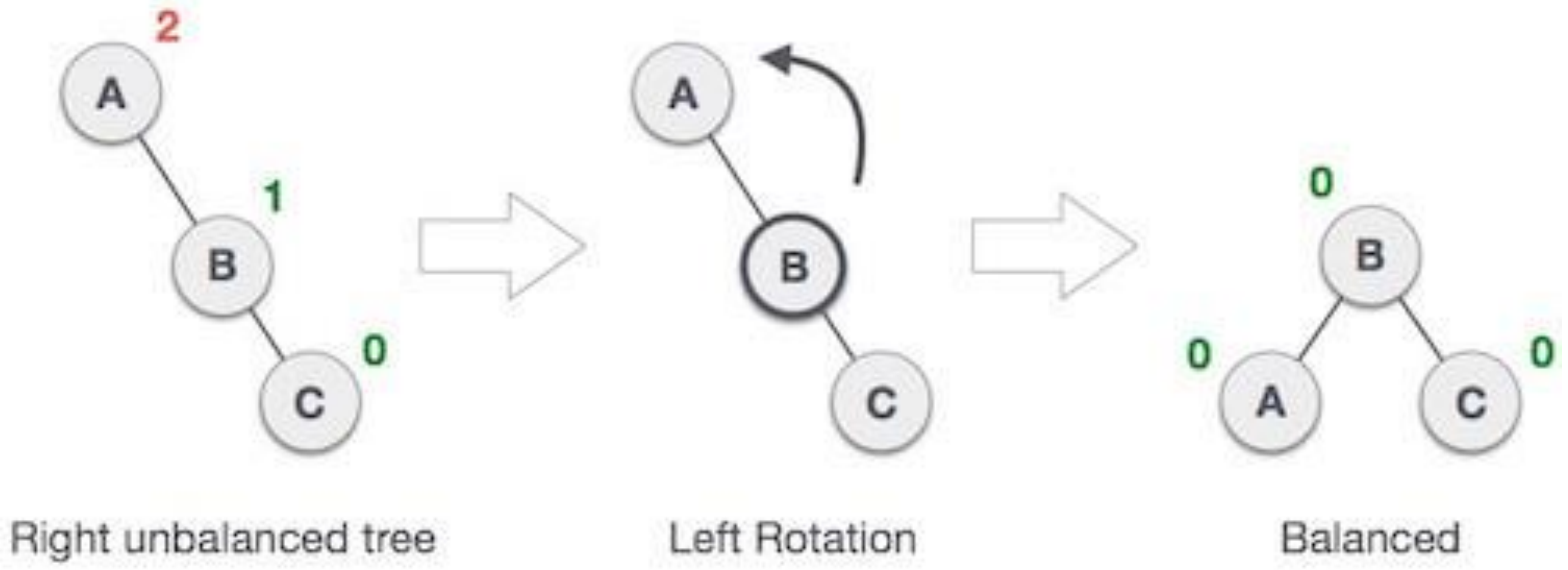
Balanced Trees cont…

Tree (AVL Tree – Operations on AVL - Rotations-RR)



Balanced Trees cont…

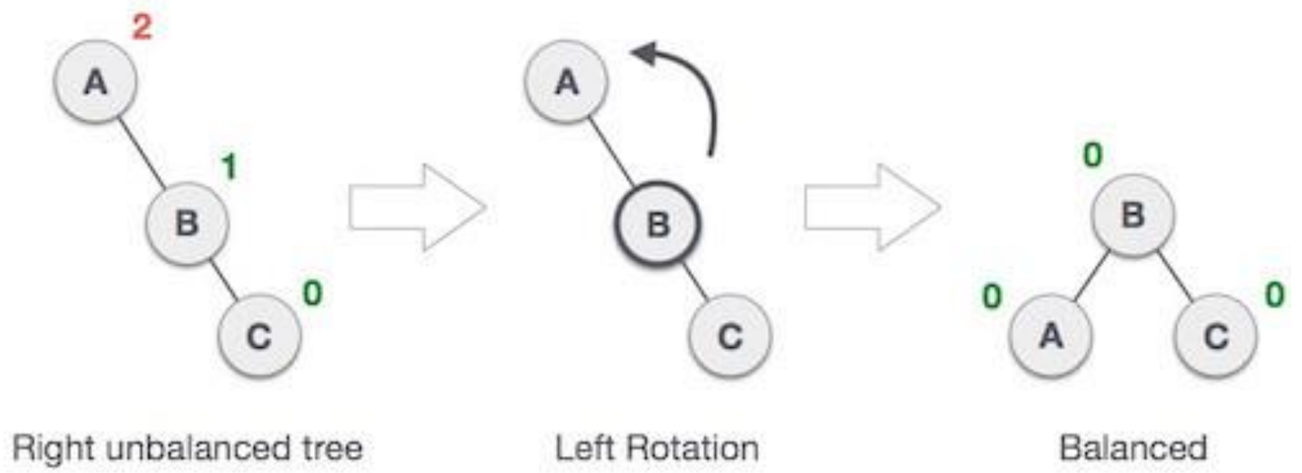
Tree (AVL Tree – Operations on AVL - Rotations-RR)



Balanced Trees cont…

Tree (AVL Tree – Operations on AVL - Rotations-RR)

- In this example, node A has balance factor -2 because a node C is inserted in the ***right subtree*** of A's ***right subtree***.
- We perform the RR rotation on the edge below A.

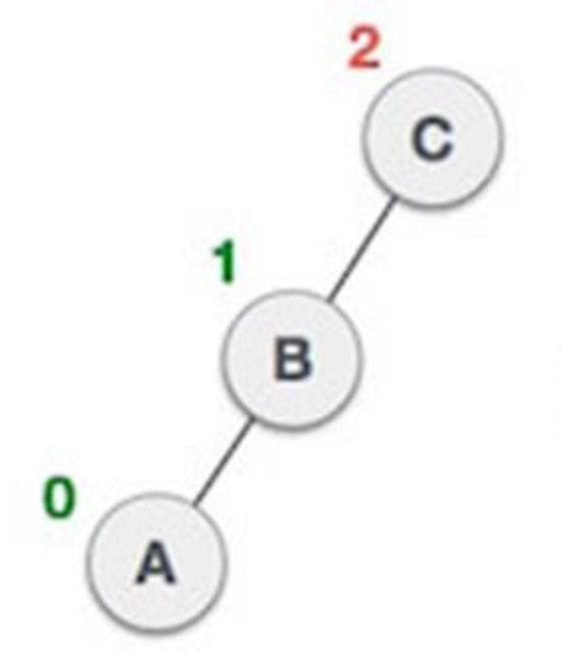


Balanced Trees cont…
Tree (AVL Tree – Operations on AVL - Rotations-LL)

- When BST becomes unbalanced, due to a node is inserted into the ***left subtree*** of the ***left subtree*** of C, then we perform **LL rotation**.
- LL rotation is **clockwise rotation**, which is applied on the **edge below** a node having ***balance factor 2***.

Balanced Trees cont…

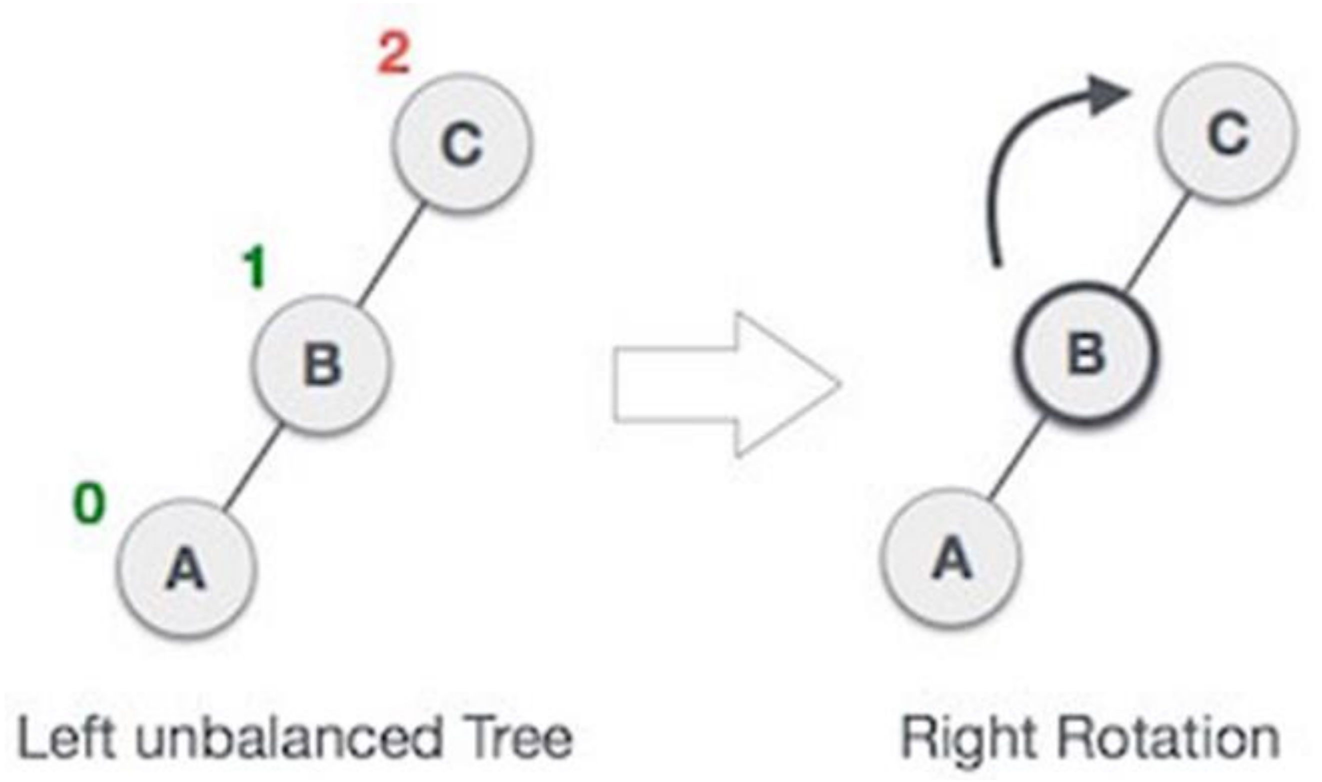
Tree (AVL Tree – Operations on AVL - Rotations-LL)



Left unbalanced Tree

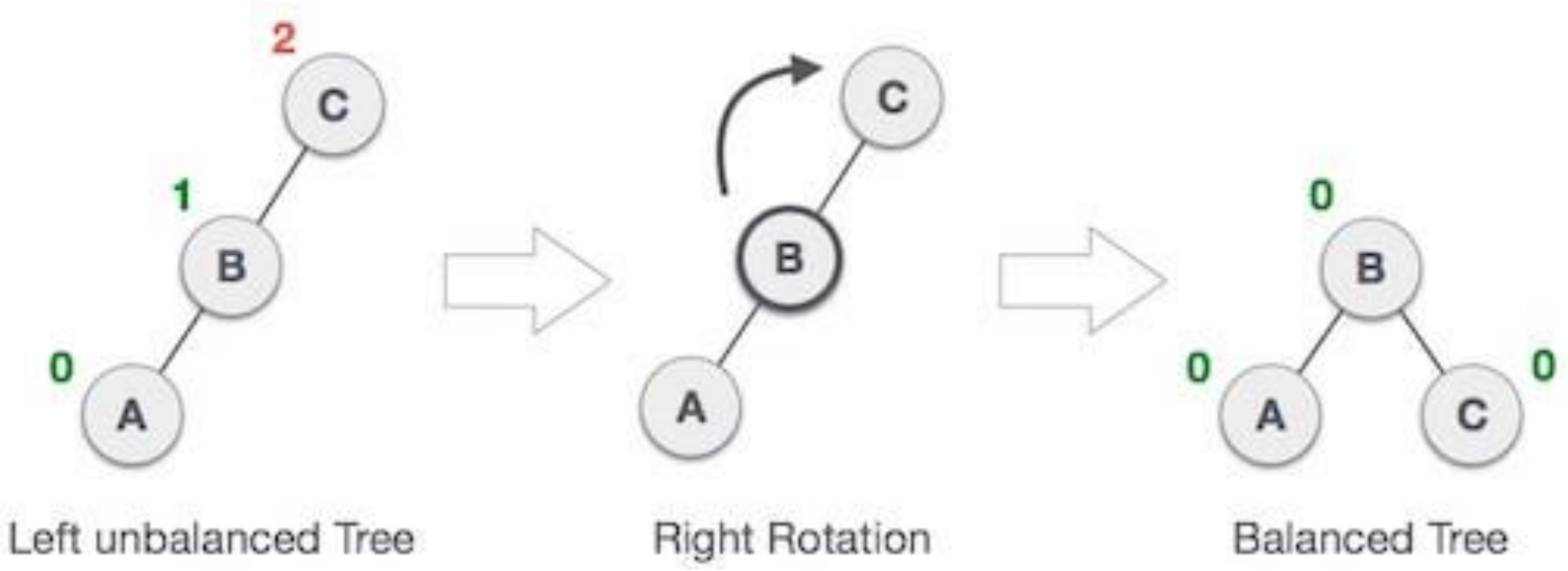
Balanced Trees cont…

Tree (AVL Tree – Operations on AVL - Rotations-LL)



Balanced Trees cont…

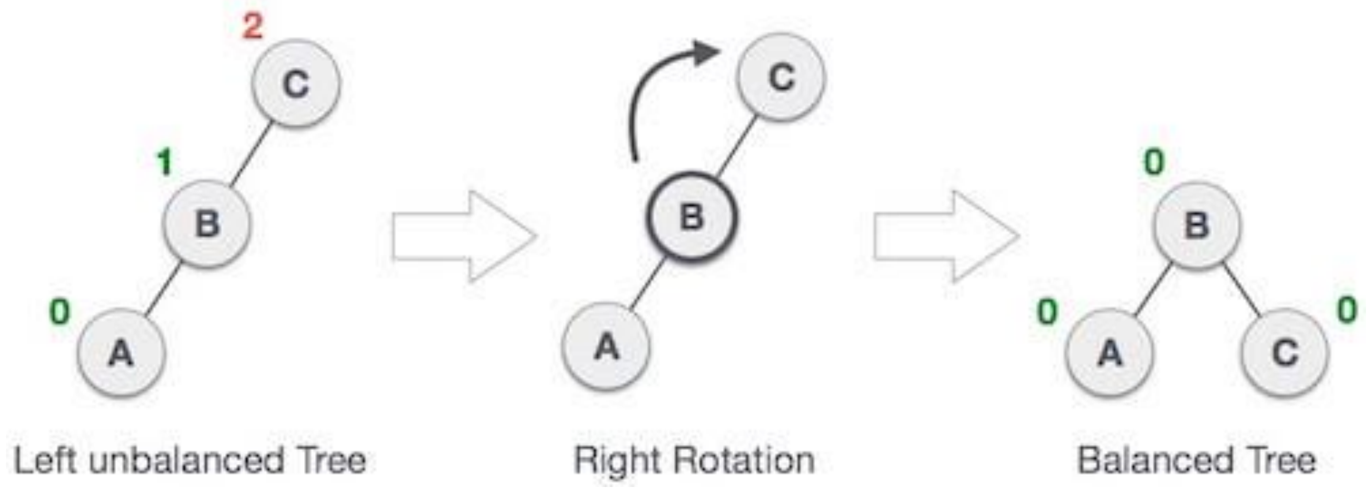
Tree (AVL Tree – Operations on AVL - Rotations-LL)



Balanced Trees cont…

Tree (AVL Tree – Operations on AVL - Rotations-LL)

- In above example, node C has balance factor 2 because a node A is inserted in the ***left subtree*** of C's ***left subtree***.
- We perform the LL rotation on the edge below A.



Balanced Trees cont…

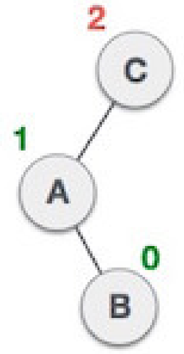
Tree (AVL Tree – Operations on AVL – Rotations-LR)

- Double rotations are bit tough than single rotation which has already explained above.
- LR rotation = **RR rotation** + **LL rotation**, i.e., first **RR rotation** is performed on ***subtree*** and then **LL rotation** is performed on ***full tree***, by full tree we mean the **first node** from the **path of inserted node** whose balance factor is other than -1, 0, or 1.

Balanced Trees cont…

Tree (AVL Tree – Operations on AVL – Rotations-LR)

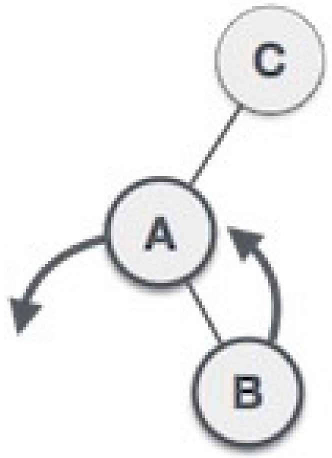
- A node B has been inserted into the ***right subtree*** of A the ***left subtree*** of C, because of which **C** has become an **unbalanced node** having balance factor 2.
- This case is **LR rotation** where: Inserted node is in the ***right subtree*** of ***left subtree*** of C



Balanced Trees cont…

Tree (AVL Tree – Operations on AVL – Rotations-LR)

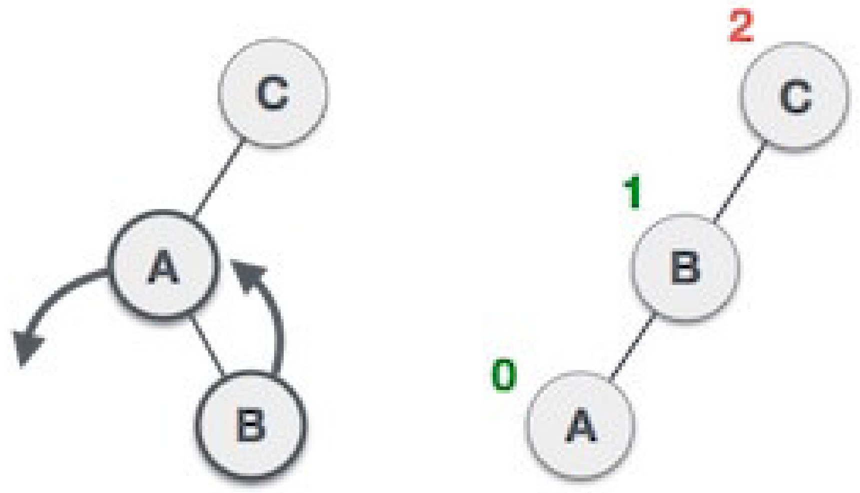
- As LR rotation = **RR + LL rotation**, hence RR (anticlockwise) on subtree rooted at A is performed first.
- By doing **RR rotation**, node **A**, has become the left subtree of **B**.



Balanced Trees cont…

Tree (AVL Tree – Operations on AVL – Rotations-LR)

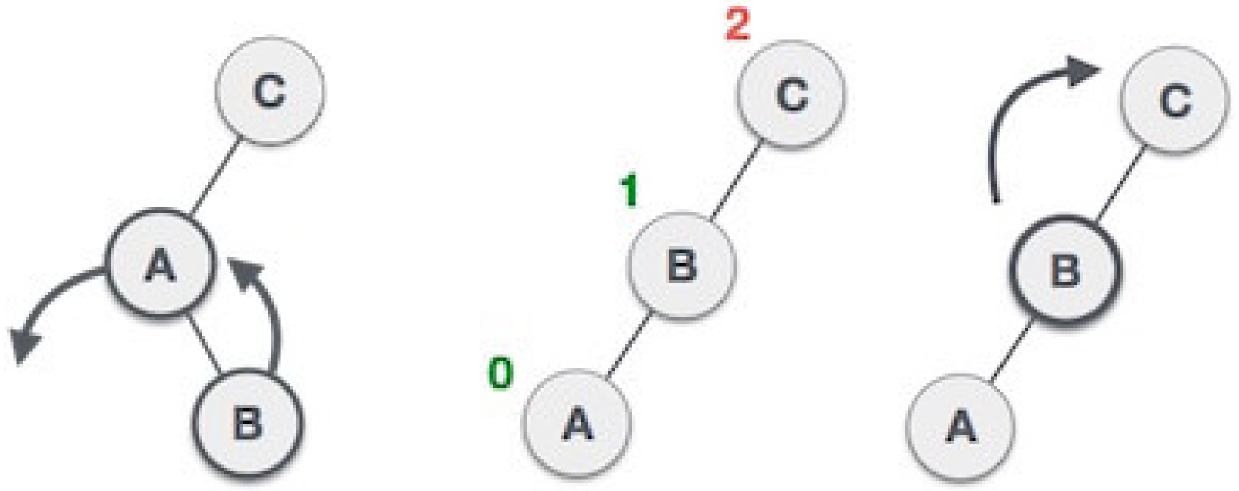
- After performing **RR rotation**, node C is still ***unbalanced***, i.e., having balance factor 2, as inserted node A is in the left of left of **C**



Balanced Trees cont…

Tree (AVL Tree – Operations on AVL – Rotations-LR)

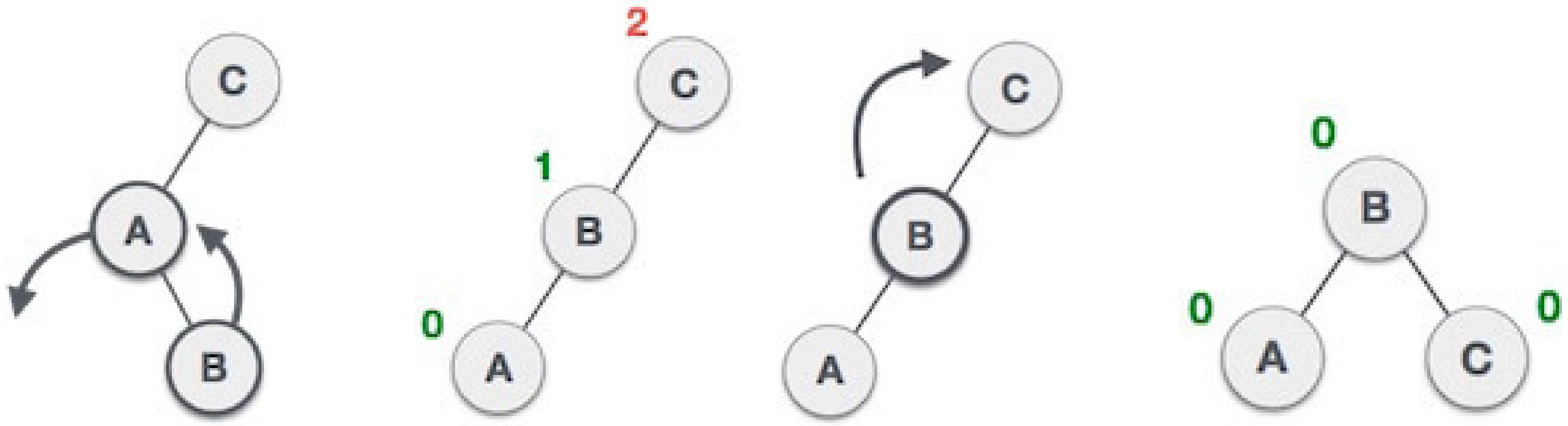
- Now we perform **<u>LL clockwise rotation</u>** on full tree, i.e. on node C.
- Node **C** has now become the ***right subtree*** of ***node B***, A is ***left subtree*** of B



Balanced Trees cont…

Tree (AVL Tree – Operations on AVL – Rotations-LR)

- Balance factor of each node is reasonable now (either -1, 0, or 1), i.e. BST is balanced now.



Balanced Trees cont…

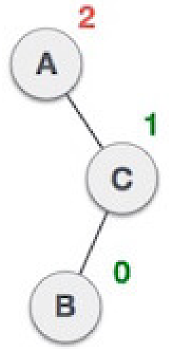
Tree (AVL Tree – Operations on AVL – Rotations-RL)

- As already discussed, that double rotations are bit tougher than single rotation which has already explained above.
- RL rotation = **LL rotation** + **RR rotation**, i.e., first **LL rotation** is performed on ***subtree*** and then **RR rotation** is performed on ***full tree***, by full tree we mean the **first node** from the ***path of inserted node*** whose balance factor is other than -1, 0, or 1.

Balanced Trees cont…

Tree (AVL Tree – Operations on AVL – Rotations-RL)

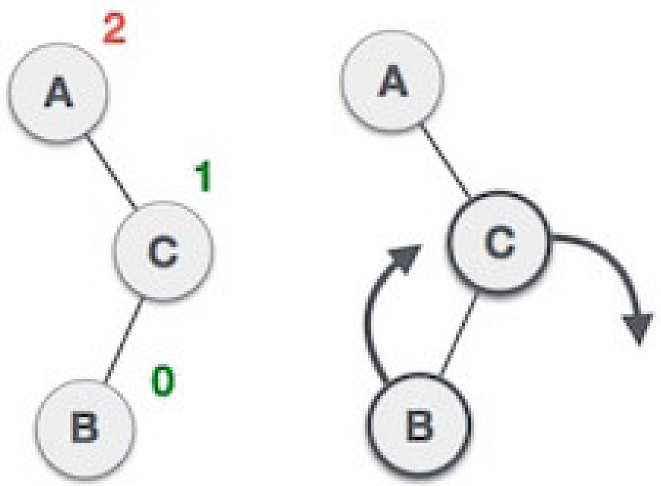
- A node **B** has been inserted into the ***left subtree*** of **C** the ***right subtree*** of **A**, because of which **A** has become an **unbalanced node** having balance factor - 2.
- This case is **RL rotation** where: Inserted node is in the left subtree of right subtree of A.



Balanced Trees cont…

Tree (AVL Tree – Operations on AVL – Rotations-RL)

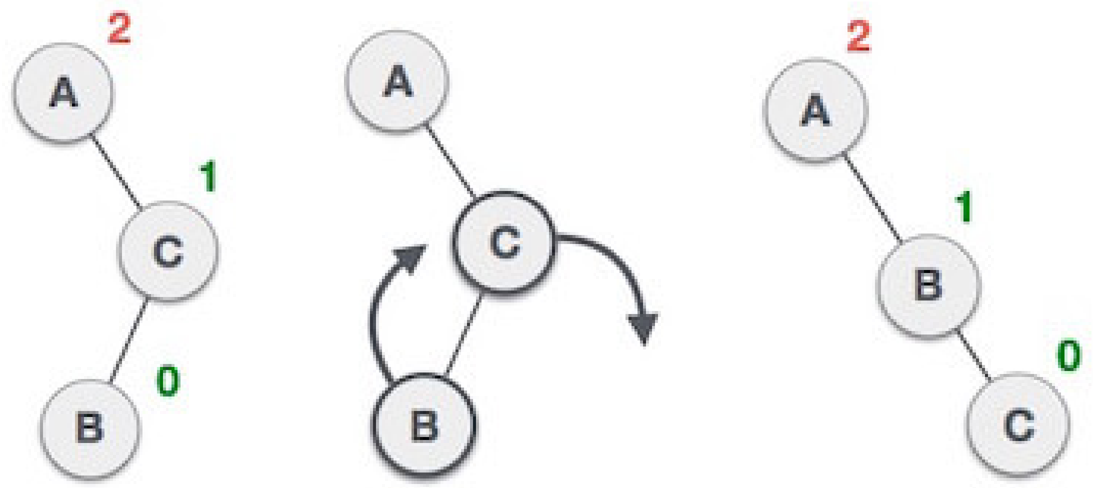
- As **RL rotation** = **LL rotation** + **RR rotation**, hence, LL (clockwise) on subtree rooted at **C** is performed first.
- By doing **LL rotation**, node **C** has become the right subtree of **B**.



Balanced Trees cont…

Tree (AVL Tree – Operations on AVL – Rotations-RL)

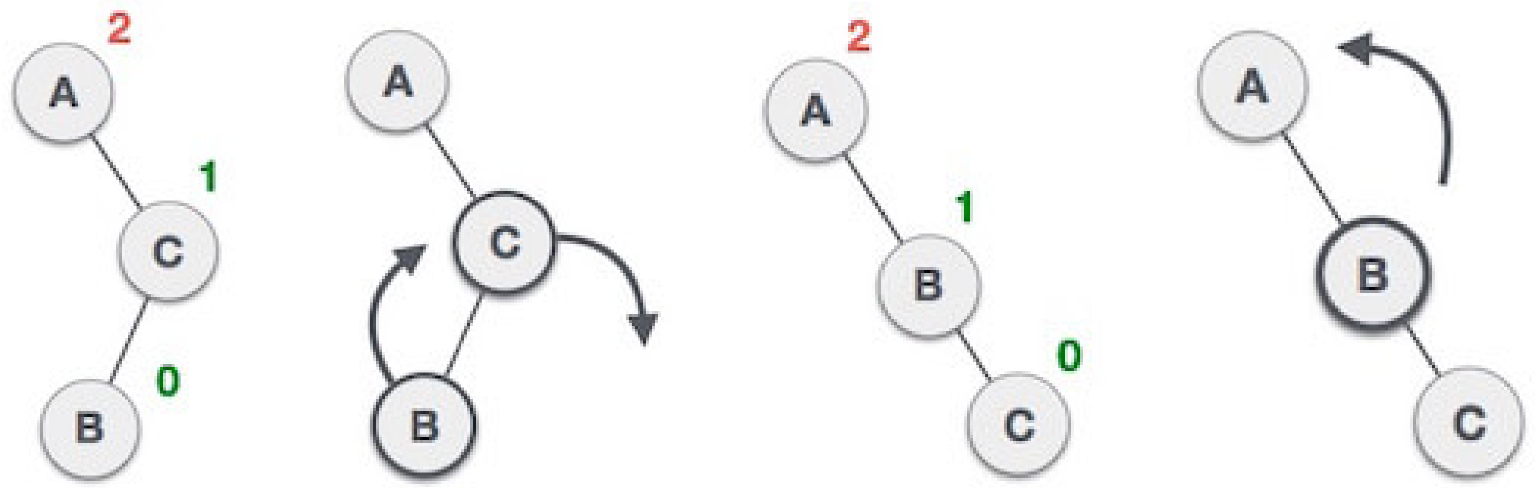
- After performing **LL rotation**, node **A** is still unbalanced, i.e. having balance factor -2, which is because of the right-subtree of the right-subtree node A.



Balanced Trees cont…

Tree (AVL Tree – Operations on AVL – Rotations-RL)

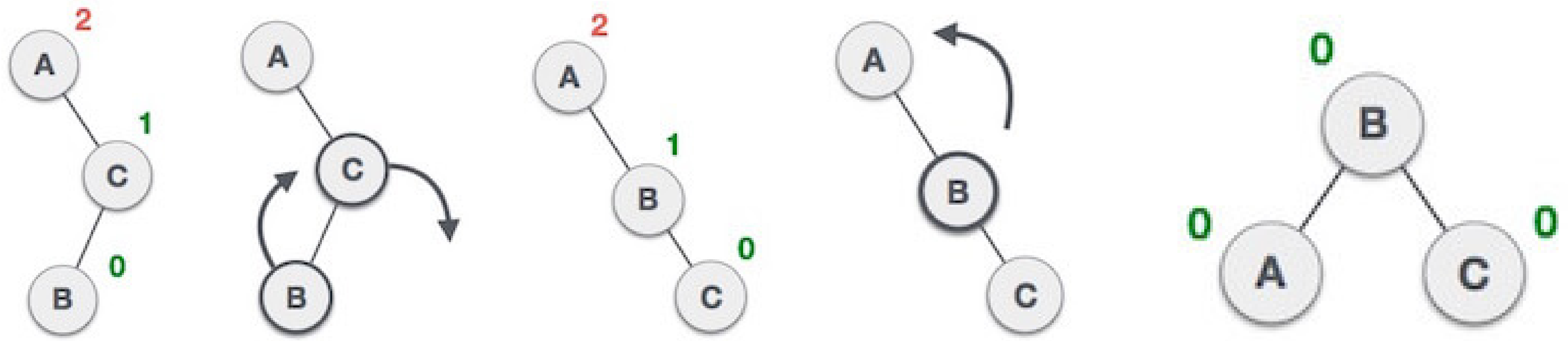
- Now we perform **RR rotation** (anticlockwise rotation) on full tree, i.e. on node A. Node **C** has now become the right subtree of node B, and node A has become the left subtree of B.



Balanced Trees cont…

Tree (AVL Tree – Operations on AVL – Rotations-RL)

- Balance factor of each node is now either -1, 0, or 1, i.e., BST is balanced now.



Balanced Trees cont…
Tree (AVL Tree – Construction of AVL Tree)

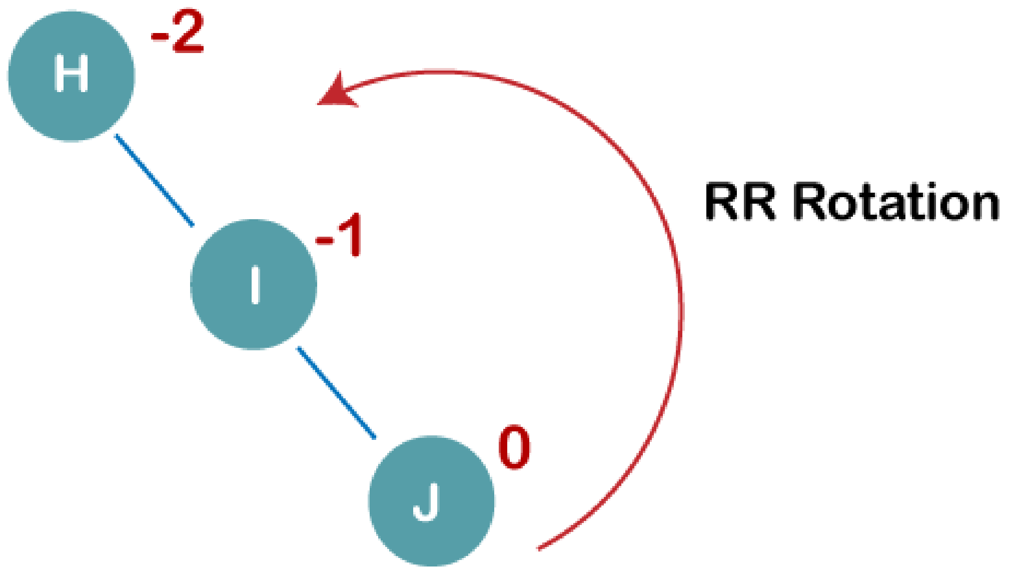
- Construct an AVL tree having the following elements:
- **H, I, J, B, A, E, C, F, D**

Balanced Trees cont…

Tree (AVL Tree – Construction of AVL Tree)

H, I, J, B, A, E, C, F, D   A B C D E F H I J

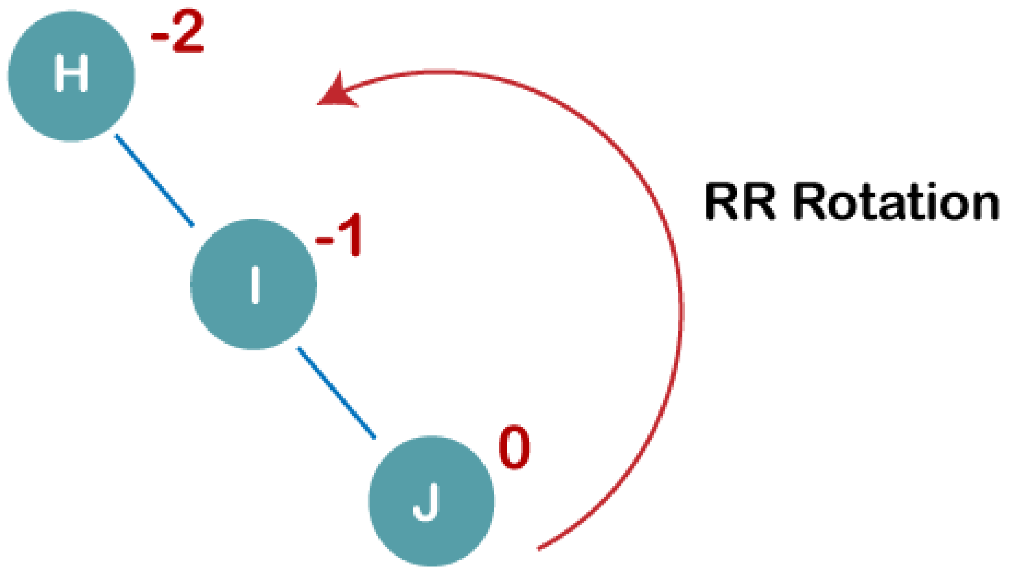
- Insert H, I, J



Balanced Trees cont…
Tree (AVL Tree – Construction of AVL Tree)

H, I, J, B, A, E, C, F, D A B C D E F H I J

- **Insert H, I, J**
- Node H, in the BST is unbalanced as the Balance Factor of H is -2.
- BST is right-skewed, we will perform ***RR Rotation*** on node H.

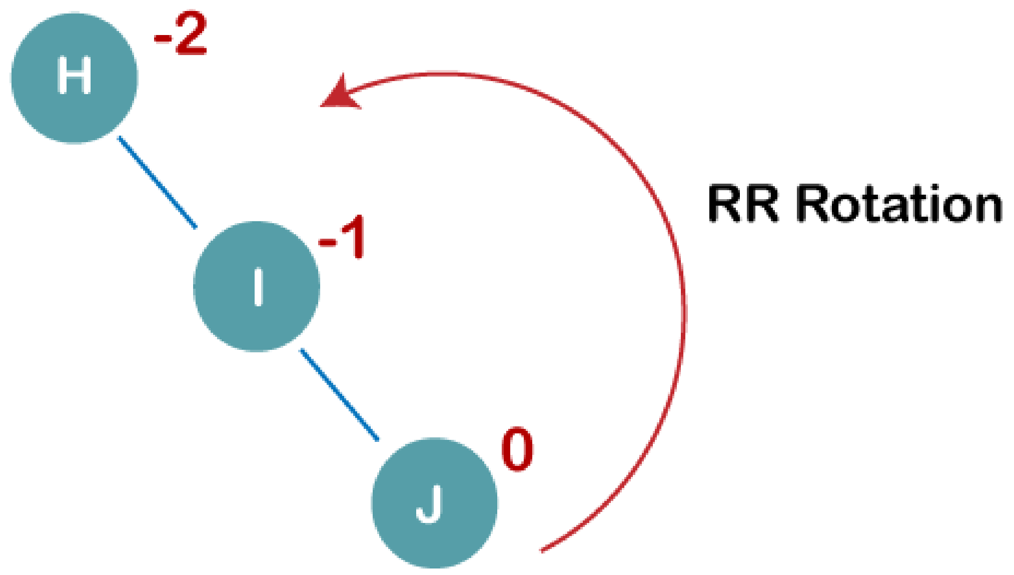


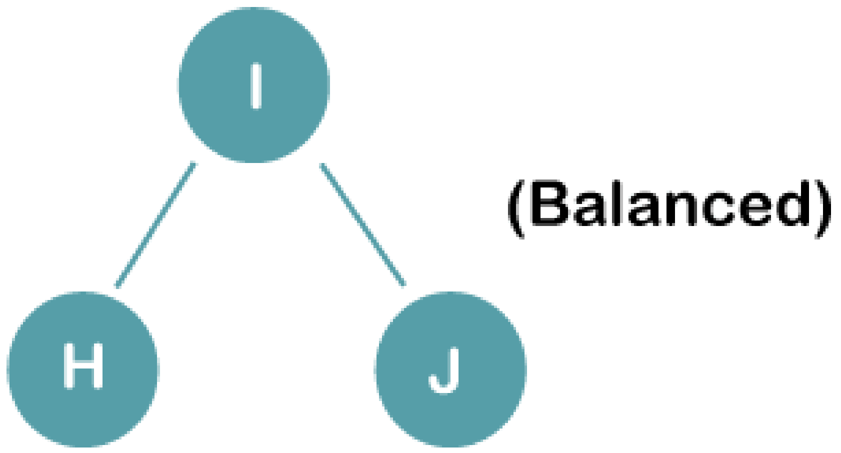
Balanced Trees cont...

Tree (AVL Tree – Construction of AVL Tree)

H, I, J, B, A, E, C, F, D **A B C D E F H I J**

- The resultant Balance tree after RR rotation is:



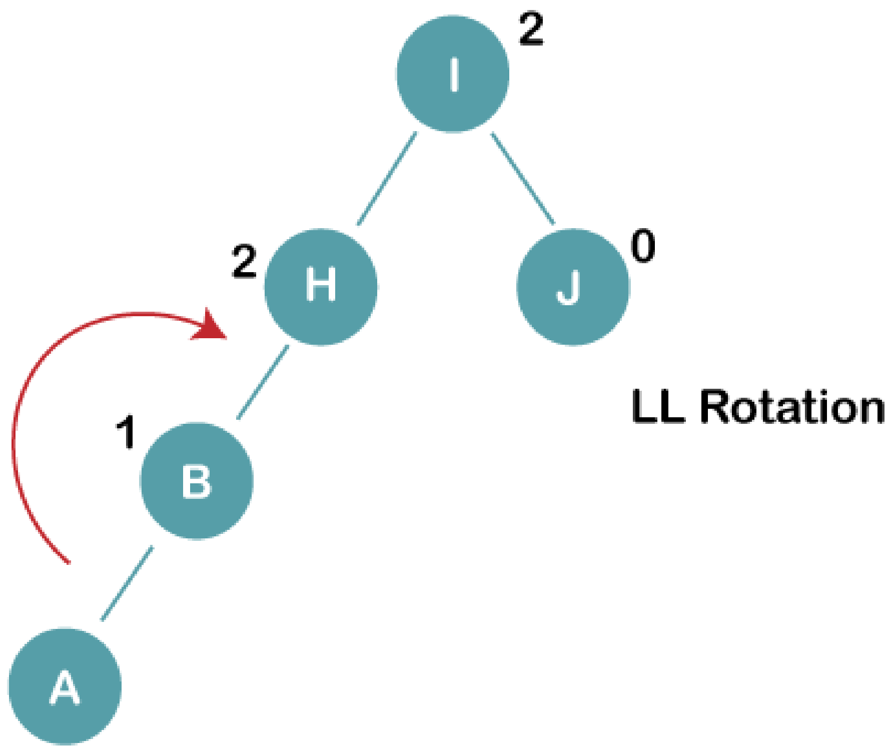


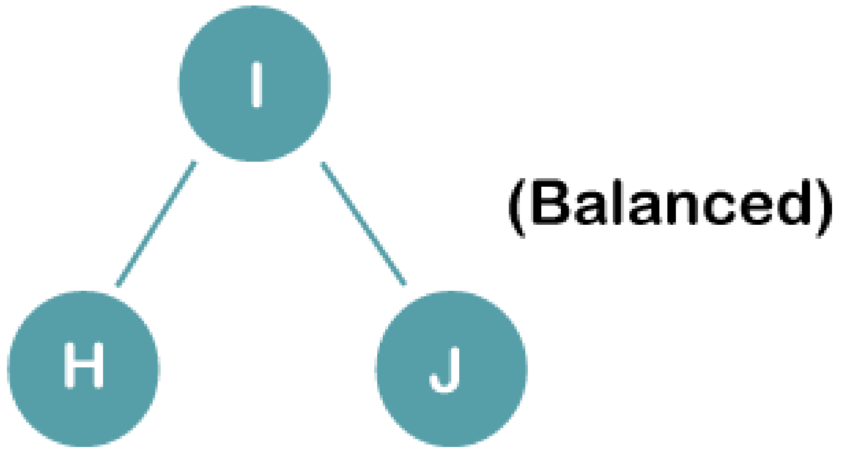
Balanced Trees cont…

Tree (AVL Tree – Construction of AVL Tree)

H, I, J, B, A, E, C, F, D A B C D E F H I J

- **Insert B, A**





ArfanShahzadTech

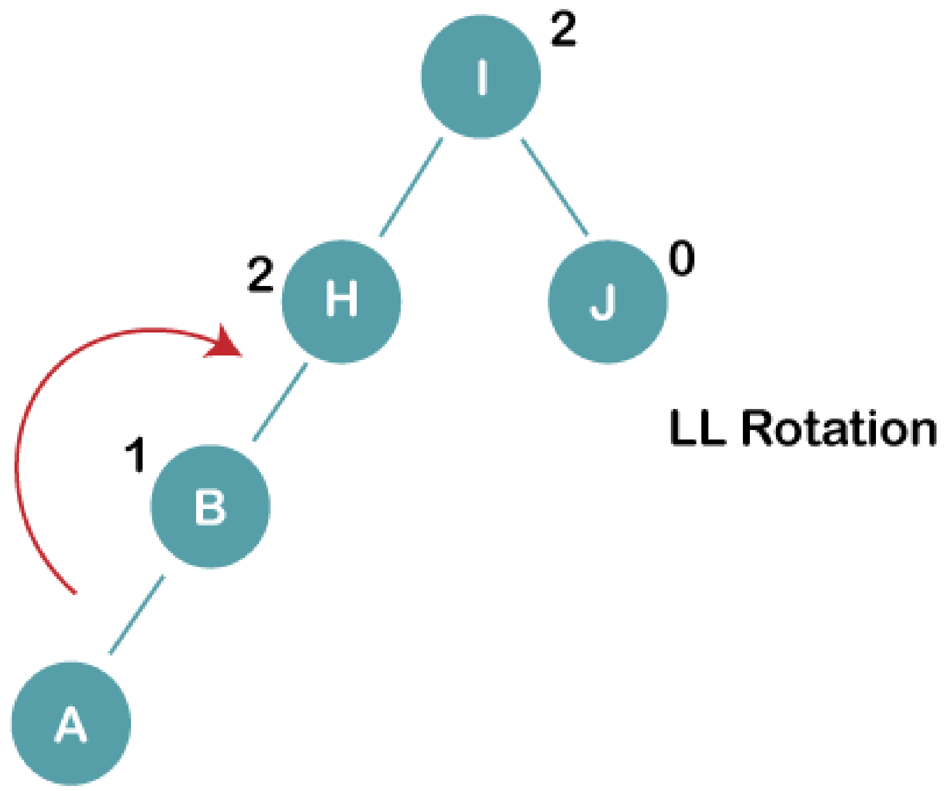
WhatsApp-Contact Us 0345-5922495

Balanced Trees cont...

Tree (AVL Tree – Construction of AVL Tree)

H, I, J, B, A, E, C, F, D A B C D E F H I J

- On inserting A, the BST becomes unbalanced as the Balance Factor of H and I is 2.
- Since the BST from H is left-skewed, we will perform LL Rotation on node H.

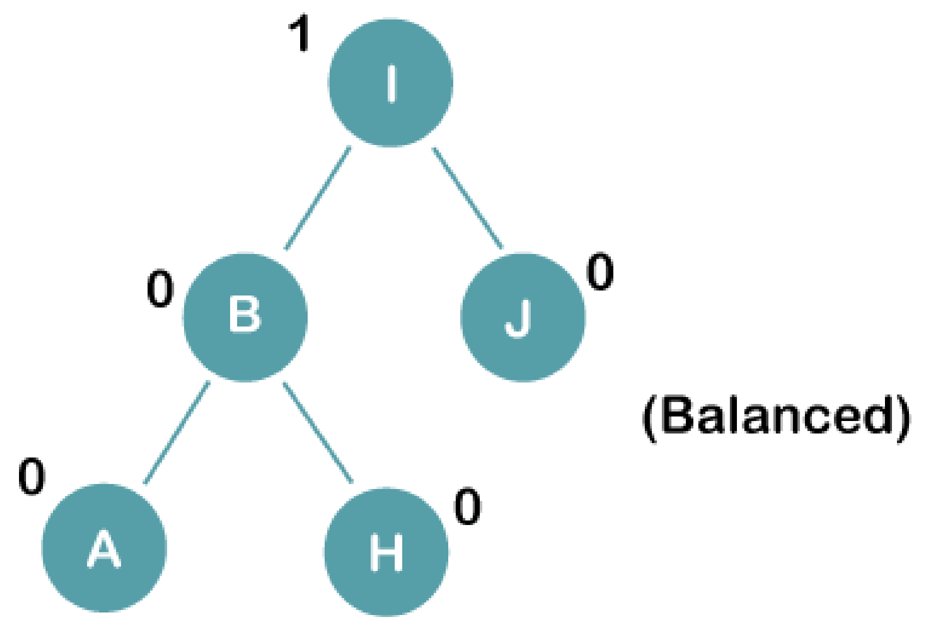


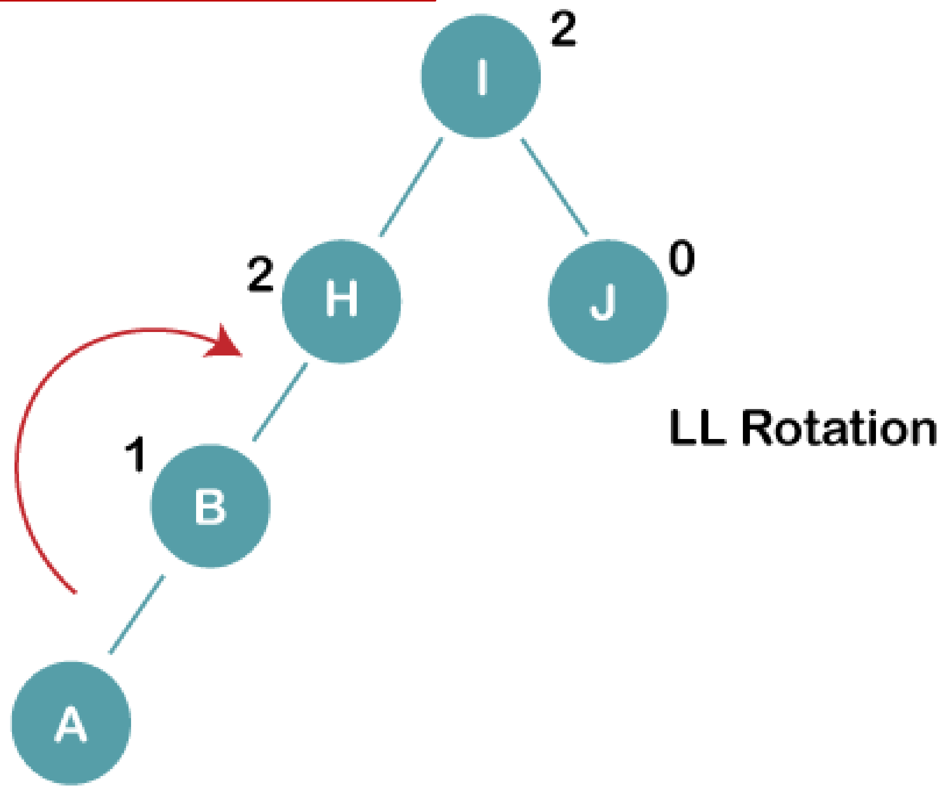
Balanced Trees cont…

Tree (AVL Tree – Construction of AVL Tree)

H, I, J, B, A, E, C, F, D **A B C D E F H I J**

- The resultant balance tree is:



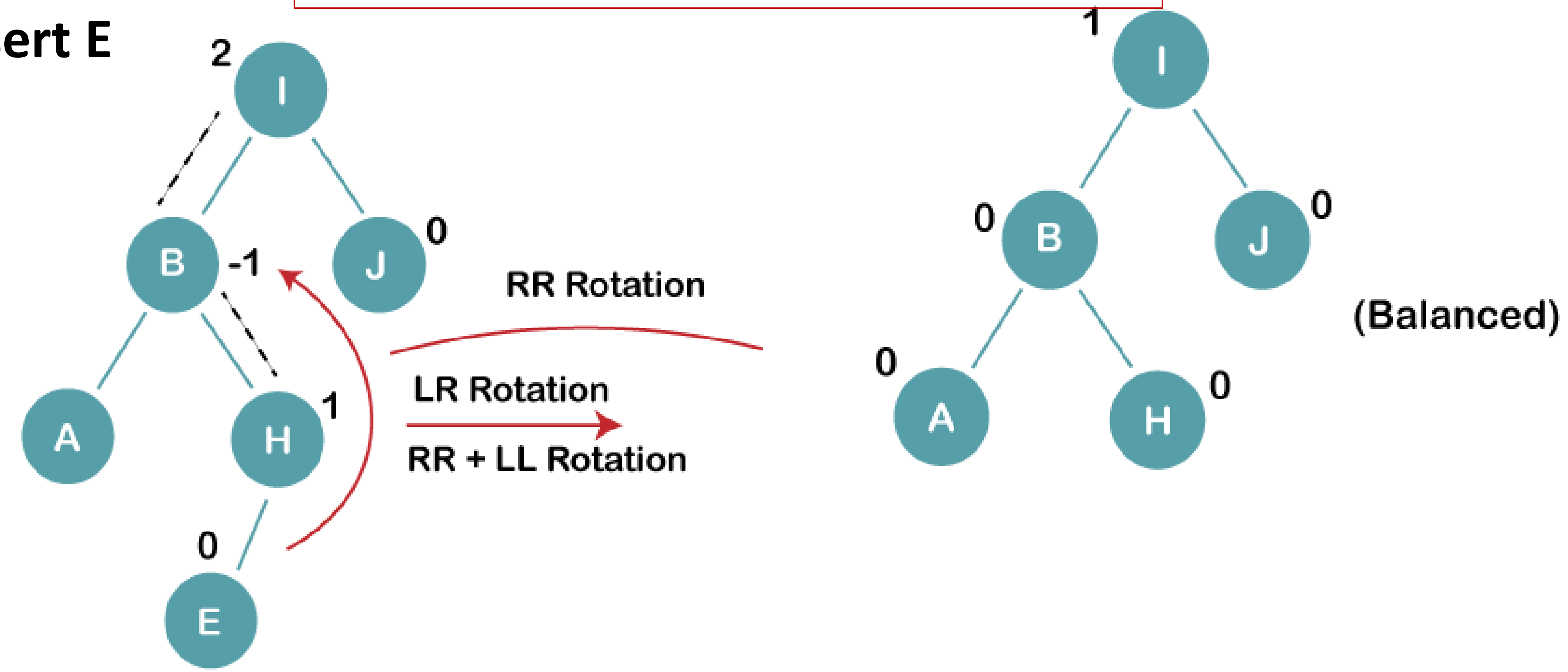


Balanced Trees cont…

Tree (AVL Tree – Construction of AVL Tree)

H, I, J, B, A, E, C, F, D   A B C D E F H I J

- **Insert E**

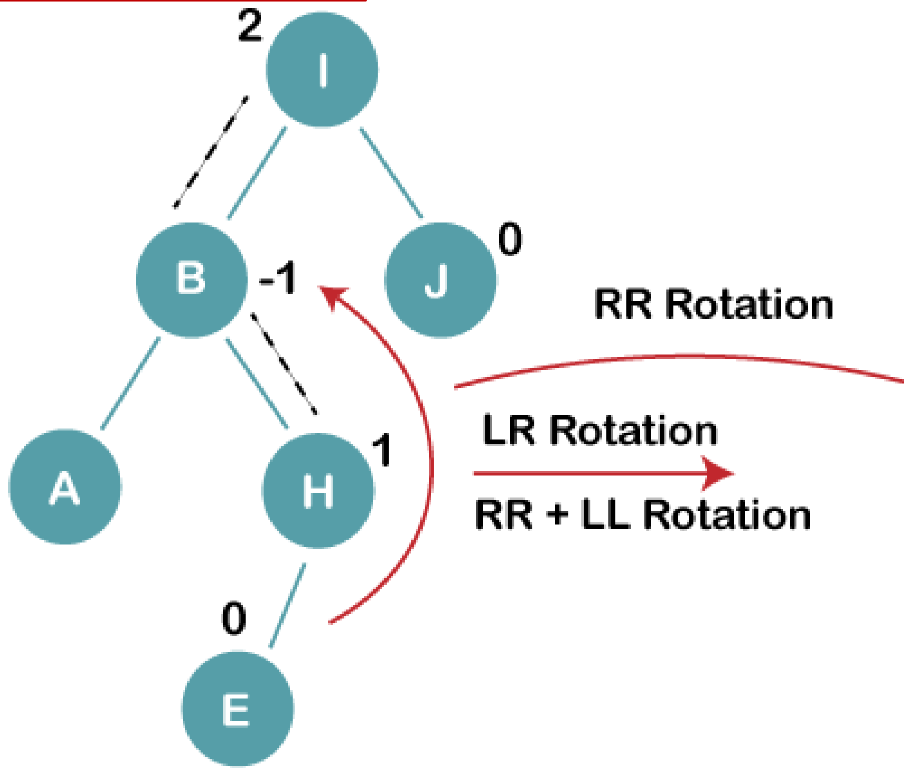


Balanced Trees cont…

Tree (AVL Tree – Construction of AVL Tree)

H, I, J, B, A, E, C, F, D   A B C D E F H I J

- On inserting E, BST becomes unbalanced as the Balance Factor of I is 2.
- Since if we travel from E to I we find that it is inserted in the left subtree of right subtree of I, we will perform LR Rotation on node I.
- LR = RR + LL rotation

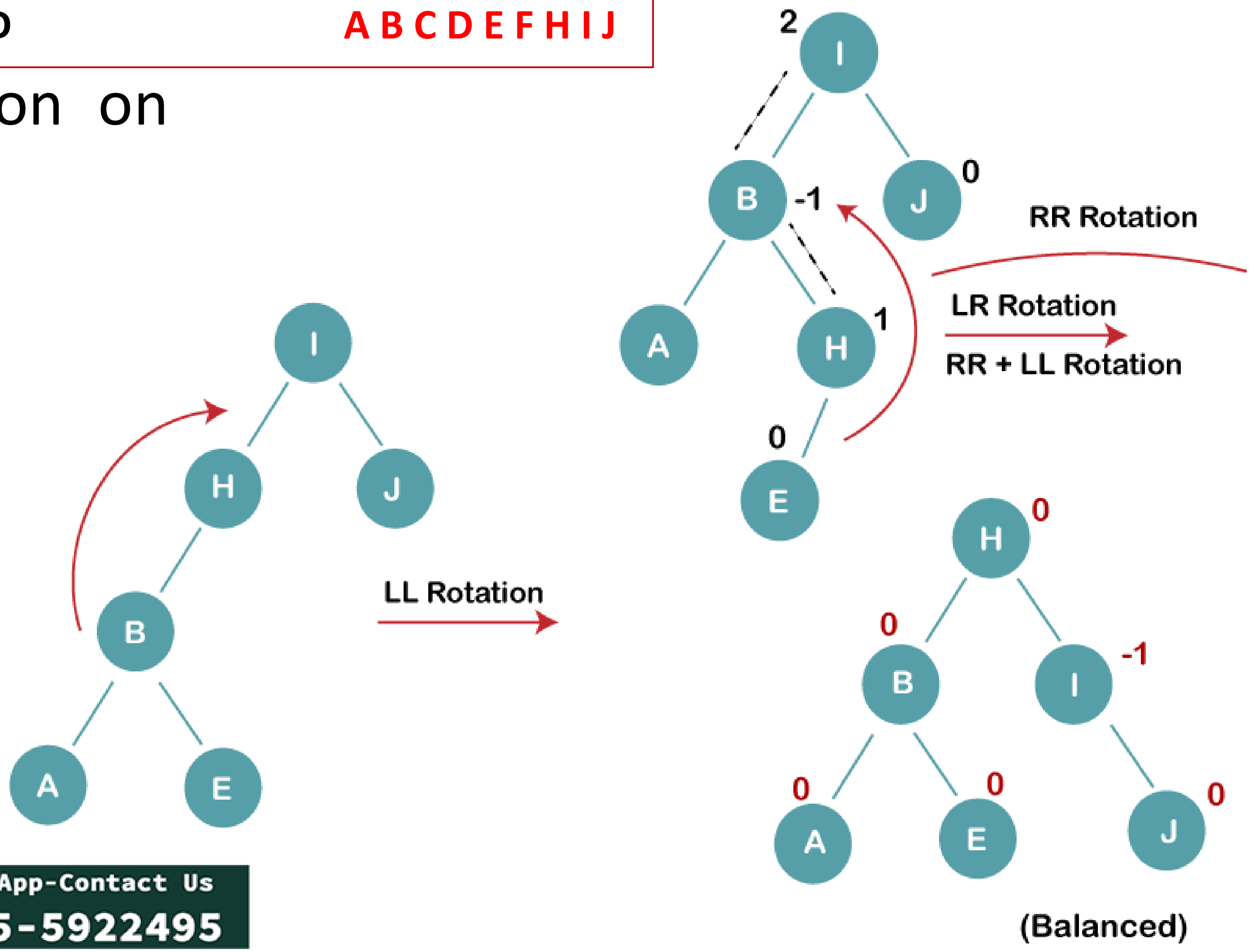


Balanced Trees cont…

Tree (AVL Tree – Construction of AVL Tree)

H, I, J, B, A, E, C, F, D   A B C D E F H I J

- We first perform RR rotation on node B, the result is:
- Then we will perform LL rotation on the node I, the result is:

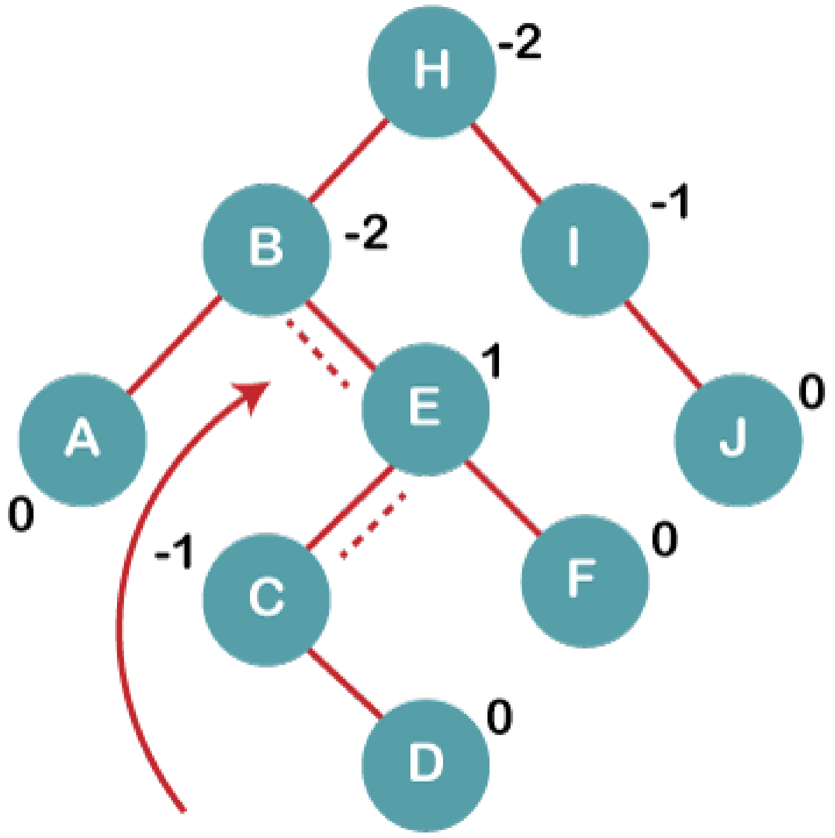


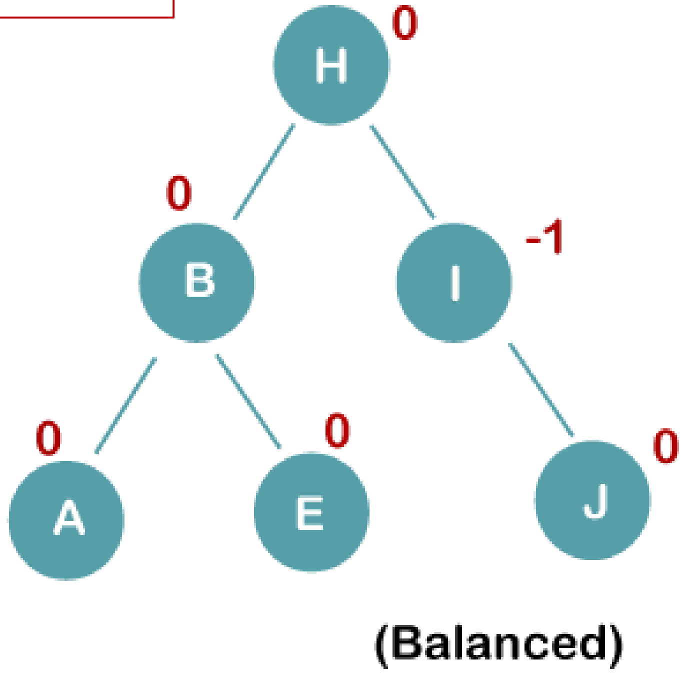
Balanced Trees cont…

Tree (AVL Tree – Construction of AVL Tree)

H, I, J, B, A, E, C, F, D   A B C D E F H I J

- Insert C, F, D





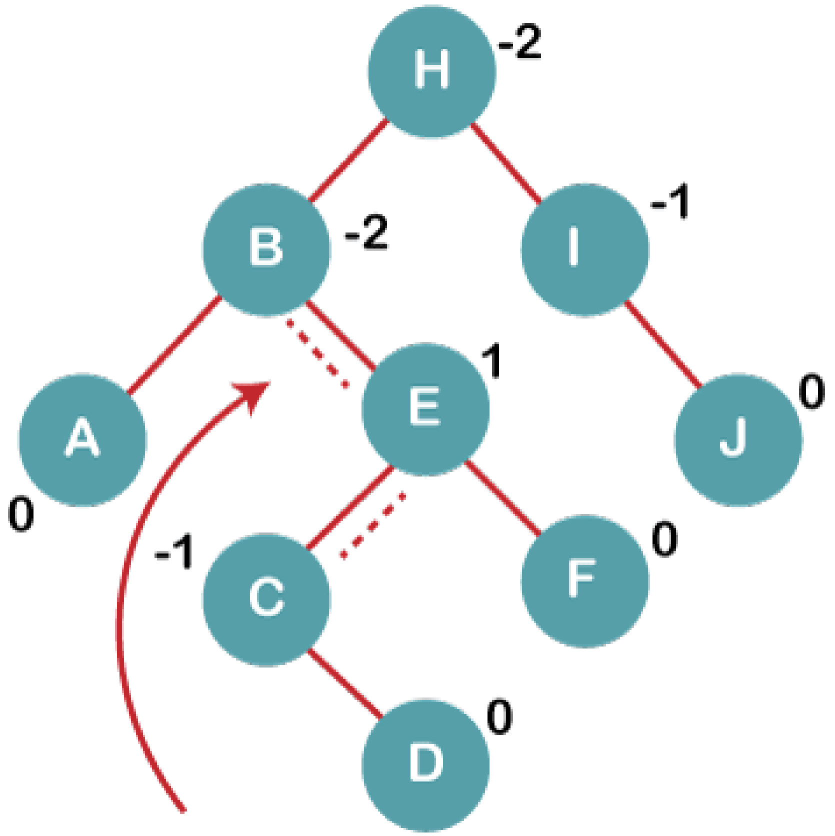
ArfanShahzadTech

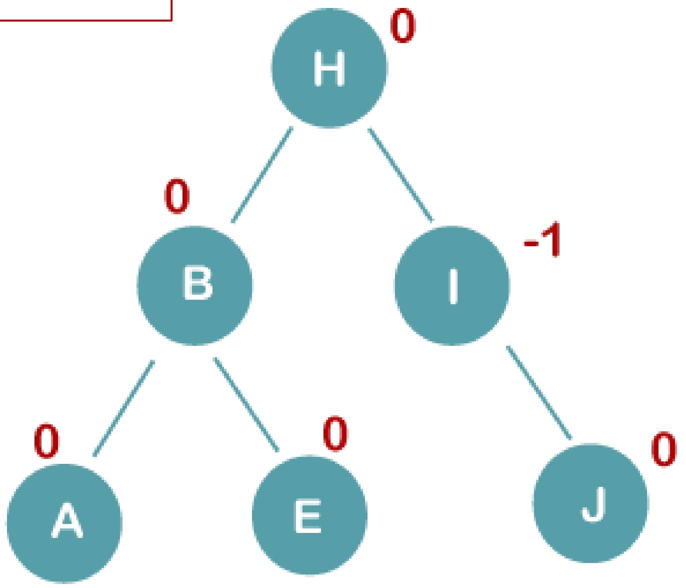
WhatsApp-Contact Us
0345-5922495

Balanced Trees cont…
Tree (AVL Tree – Construction of AVL Tree)

H, I, J, B, A, E, C, F, D   A B C D E F H I J

- Insert C, F, D





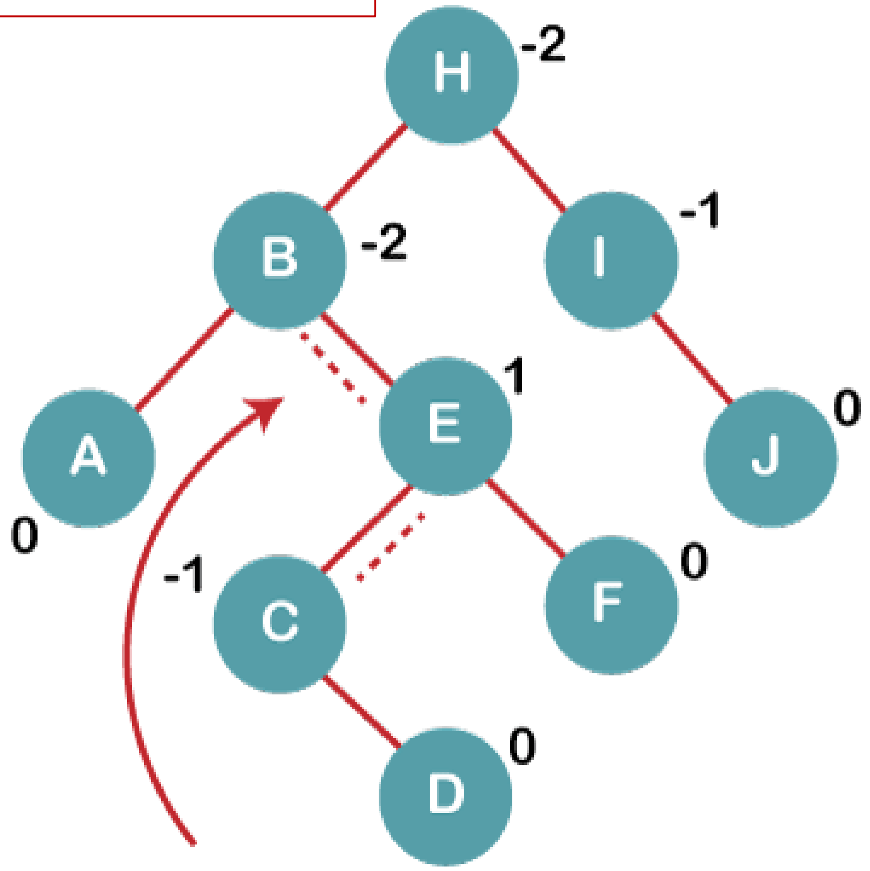
(Balanced)

Balanced Trees cont…

Tree (AVL Tree – Construction of AVL Tree)

H, I, J, B, A, E, C, F, D **A B C D E F H I J**

- On inserting C, F, D, BST becomes unbalanced as the Balance Factor of B and H is -2.
- Since if we travel from D to B we find that it is inserted in the right subtree of left subtree of B.
- We will perform RL Rotation on node I.
- RL = LL + RR rotation.

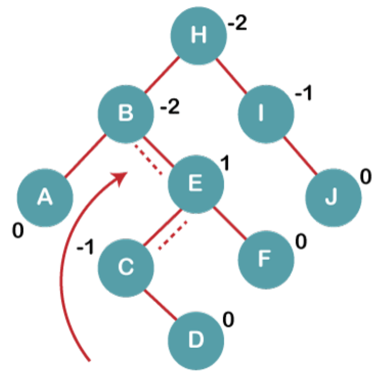


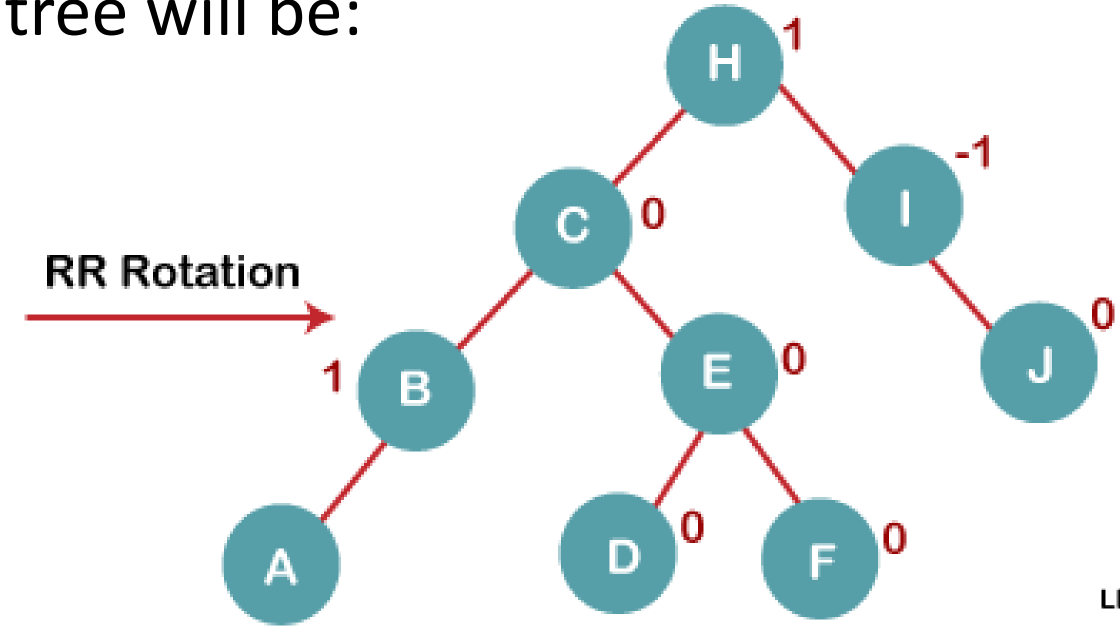
Balanced Trees cont…

Tree (AVL Tree – Construction of AVL Tree)

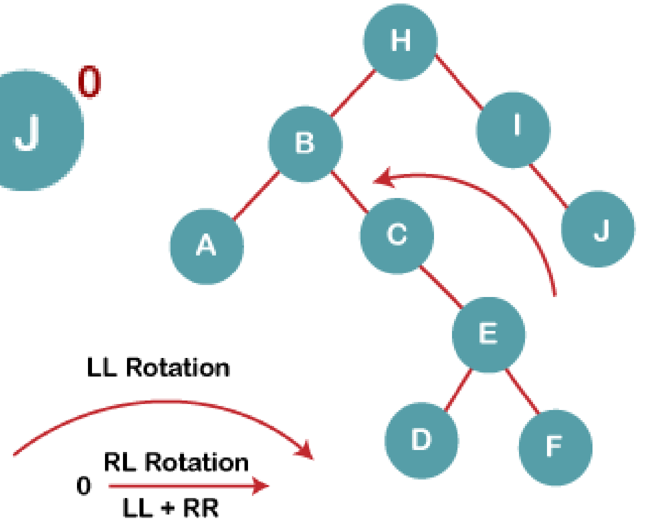
H, I, J, B, A, E, C, F, D   A B C D E F H I J

- We first perform LL rotation on node E, the resultant tree will be:
- We then perform RR rotation on node B, resultant balanced tree will be:





(Balanced)



Balanced Trees cont…
Tree (Red Black Tree)

- The **Red-Black tree** is a ***binary search tree***.
- In a binary search tree, the values of the nodes in the ***left subtree*** should be ***<u>less than</u>*** the value of the **root node**, and the values of the nodes in the ***right subtree*** should be ***<u>greater than</u>*** the value of the ***root node***.

Balanced Trees cont…
Tree (Red Black Tree)

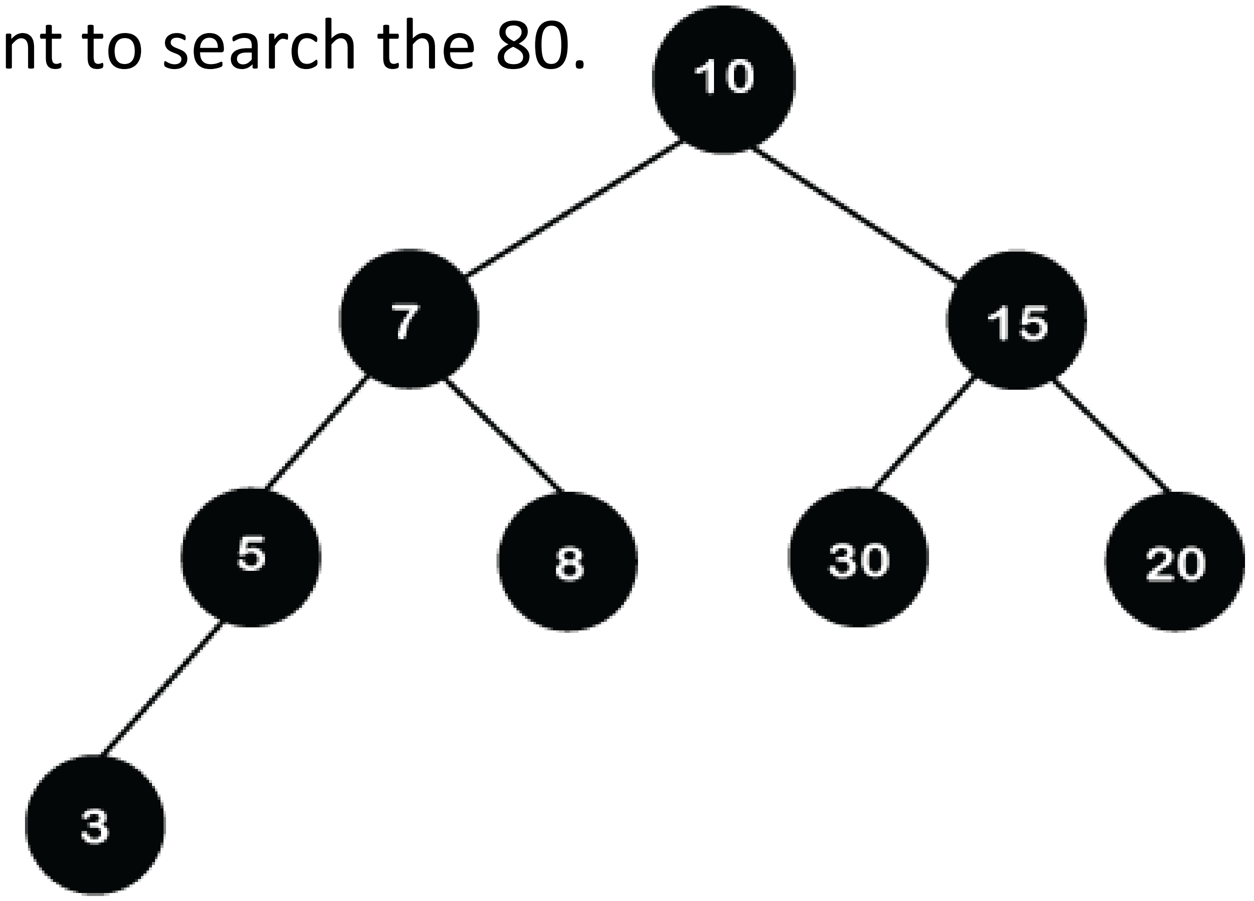
- Each node in the Red-black tree **contains** an ***extra bit*** that represents a **color** to ensure that the tree is balanced during any operations performed on the tree like ***insertion***, ***deletion***, etc.

Balanced Trees cont…
Tree (Red Black Tree)

- In a binary search tree, the ***searching***, ***insertion*** and ***deletion*** take **O($\log_2 n$)** time in the **average case**, **O(1)** in the **best case** and **O(n)** in the **worst case**.
- Let's understand the different scenarios of a binary search tree.

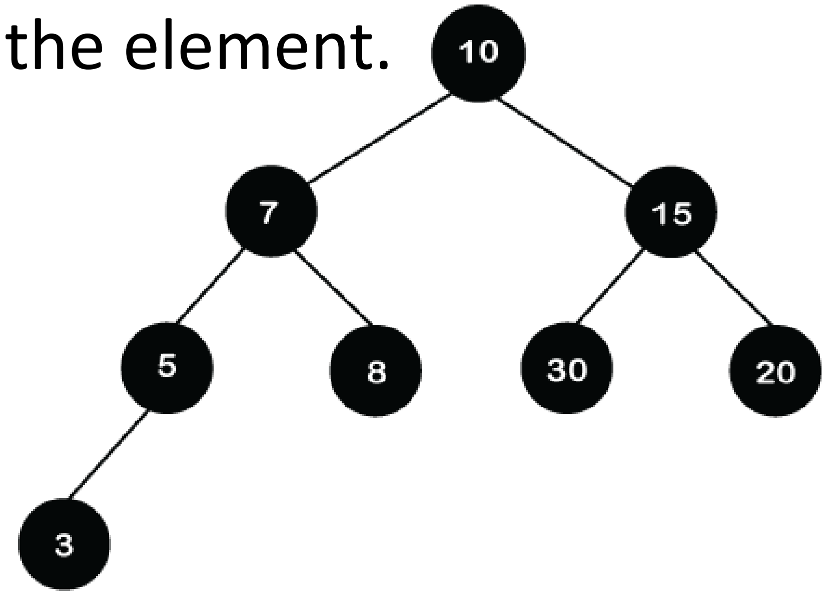
Balanced Trees cont…
Tree (Red Black Tree)

- In the above tree, if we want to search the 80.



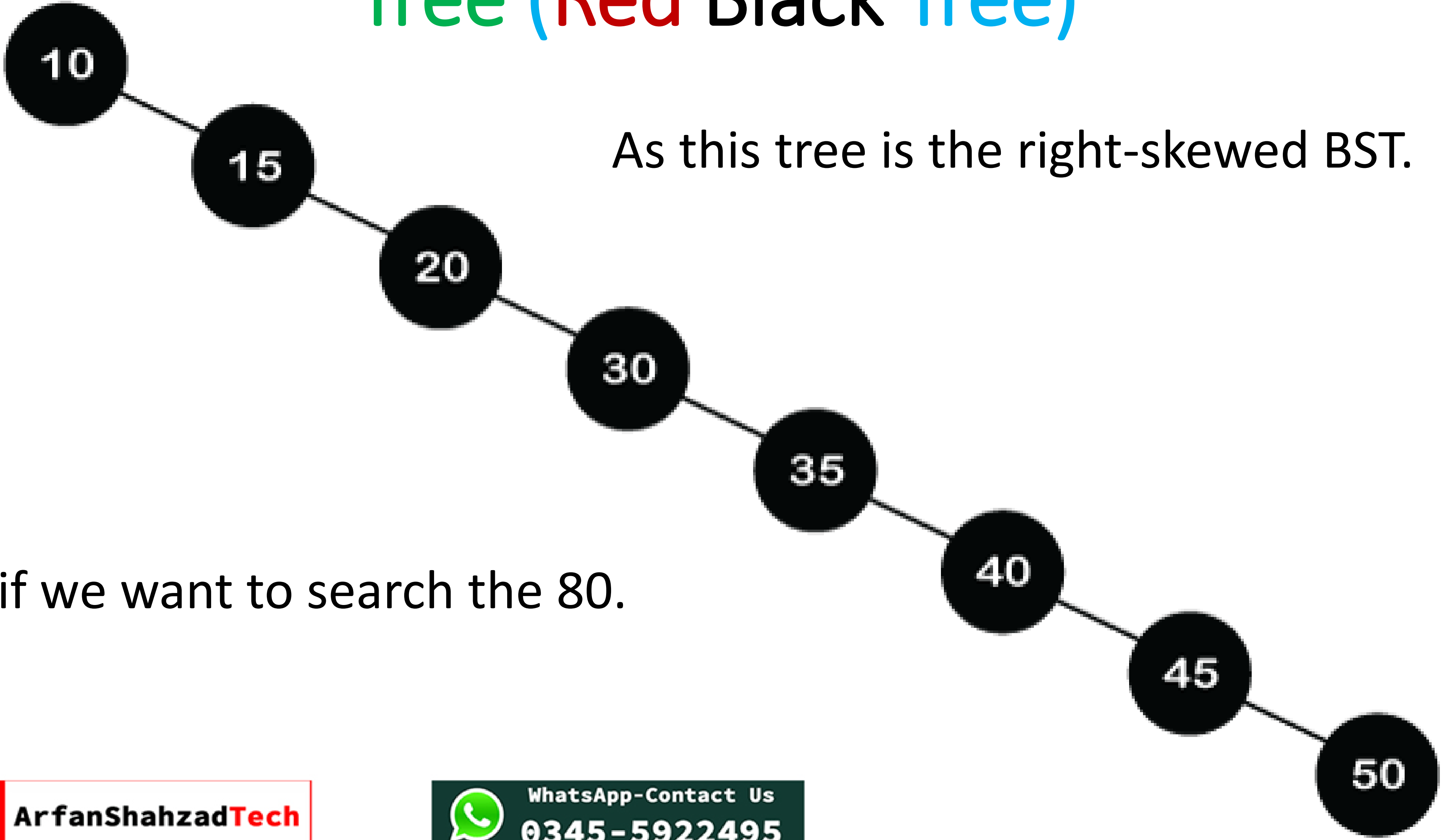
Balanced Trees cont...
Tree (Red Black Tree)

- Therefore, it will show that the element is not found in the tree.
- After each operation, the search is divided into half.
- The above BST will take **O(log n)** time to search the element.



Balanced Trees cont...

Tree (Red Black Tree)

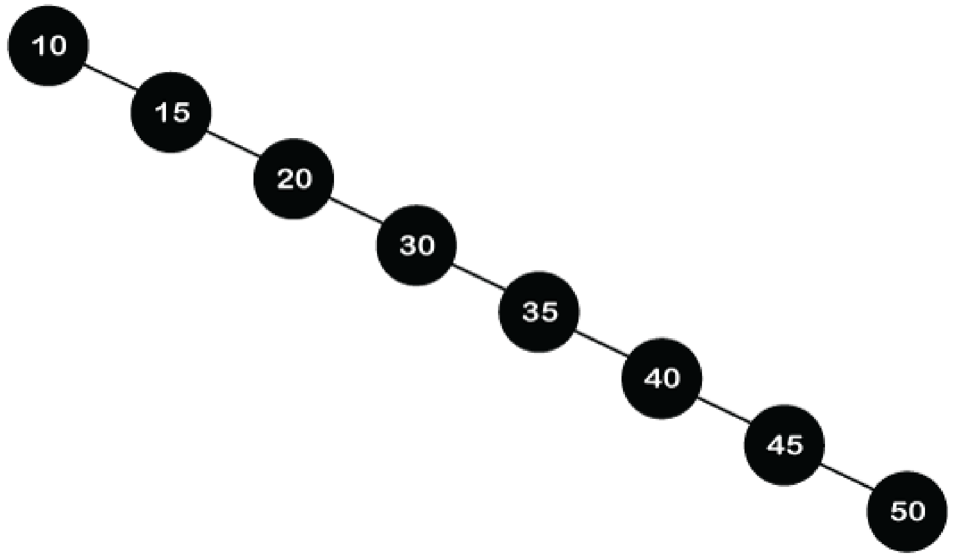


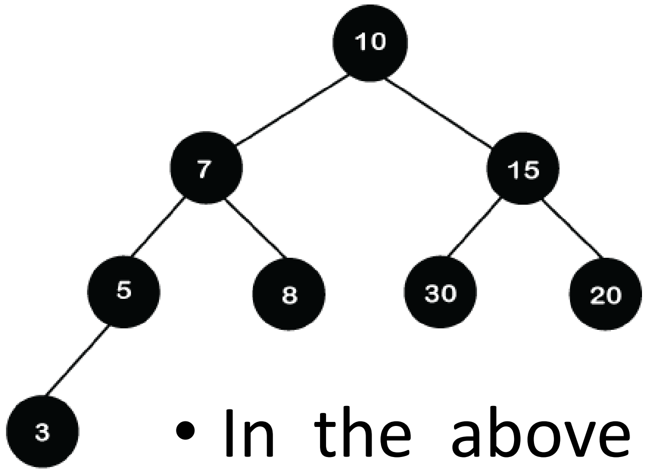
As this tree is the right-skewed BST.

So, if we want to search the 80.

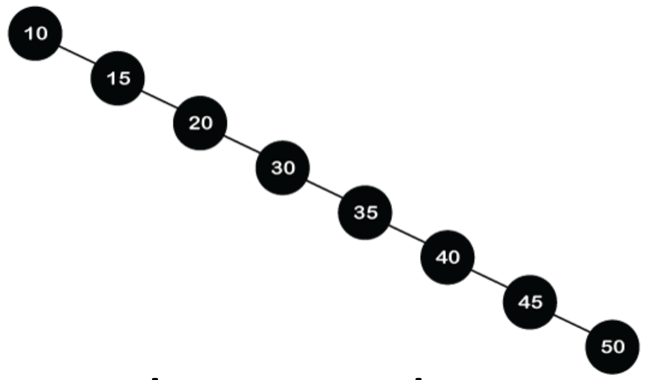
Balanced Trees cont…
Tree (Red Black Tree)

- As 80 is not found in the tree.
- So, this right-skewed BST will take **O(n)** time to search the element.





Balanced Trees cont...
Tree (Red Black Tree)



- In the above BST, the first one is the balanced BST, whereas the second one is the unbalanced BST.
- We conclude from the above two binary search trees that a **balanced** tree takes ***less time*** than an **unbalanced** tree for performing any operation on the tree.

Balanced Trees cont…
Tree (Red Black Tree)

- Therefore, we need a balanced tree, and the **Red**-**Black** tree is a self-balanced binary search tree.
- Now, the question arises that why do we require a **Red**-**Black** tree if AVL is also a ***height-balanced tree***.

Balanced Trees cont…
Tree (Red Black Tree)

- The Red-Black tree is used because the **AVL tree** requires ***many rotations*** when the tree is large, whereas the Red-Black tree requires a maximum of two rotations to balance the tree.
- The main difference between the AVL tree and the Red-Black tree is that the ***AVL tree is strictly balanced***, while the Red-Black tree is not completely height-balanced.

Balanced Trees cont…
Tree (Red Black Tree)

- So, the AVL tree is more balanced than the Red-Black tree, but the Red-Black tree guarantees $O(\log_2 n)$ time for all operations like ***insertion***, ***deletion***, and ***searching***.
- Insertion is easier in the AVL tree as the AVL tree is strictly balanced, whereas deletion and searching are easier in the **Red**-**Black** tree as the Red-Black tree requires fewer rotations.

Balanced Trees cont…
Tree (Red Black Tree)

- As the name suggests that the node is either colored in Red or Black color.
- Sometimes no rotation is required, and only recoloring is needed to balance the tree.

Balanced Trees cont…
Tree (Red Black Tree – Properties)

- Properties of Red-Black tree are given:

1. It is a ***self-balancing Binary Search tree***.

- Here, self-balancing means that it balances the tree itself by either doing the rotations or recoloring the nodes.

Balanced Trees cont…
Tree (Red Black Tree – Properties)

2. This tree Balanced Trees is named as a **Red-Black** tree as each node is either Red or Black.

- Every node stores one extra information “bit” that represents color of the node.
- For example, **0 bit** denotes the **black color** while **1 bit** denotes the **red color**.
- Other information stored by the node is similar to the binary tree, i.e., data part, left pointer and right pointer.

Balanced Trees cont…
Tree (Red Black Tree – Properties)

3. In the Red-Black tree, the **root node** is ***always black*** in color.

4. If any node is Red, then its children should be in Black color. In other words, we can say that there should be no **red-red parent-child** relationship.

5. Every path from a node to any of its descendant's should have same number of black nodes.

Balanced Trees cont…

Tree (Red Black Tree – Is AVL tree can be Red Black tree?)

- Yes, every AVL tree can be a Red-Black tree if we color each node either by Red or Black color.
- But every Red-Black tree is not an AVL because the AVL tree is strictly height-balanced while the Red-Black tree is not completely height-balanced.

Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

1. If the **<u>tree is empty</u>**, then we ***create a new node*** as a **<u>root node</u>** with the ***color black***.

2. If the ***tree is not empty***, then we ***create a new node*** as a <u>leaf node</u> with a ***color red***.

3. If the parent of a new node is black, then exit (do nothing).

Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

4. If the parent of a new node is Red, then we have to check the color of the parent's sibling of a new node.

a) If the color is **Black**, then we perform ***rotations*** and ***recoloring***.

b) If the color is **Red** then we recolor the node. We will also check whether the **parents' parent of a new node is the root node or not**; if it is **not a root node**, we will ***recolor*** and ***recheck*** the node.

Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

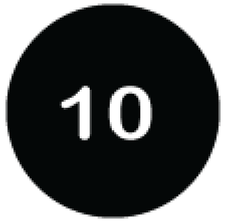
- Let's understand the insertion in the Red-Black tree.
- 10, 18, 7, 15, 16, 30

Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- **Step 1:** Initially, the tree is empty, so we create a new node having value 10.
- This is the first node of the tree, so it would be the root node of the tree.
- As we already discussed, that root node must be black in color, which is shown below:

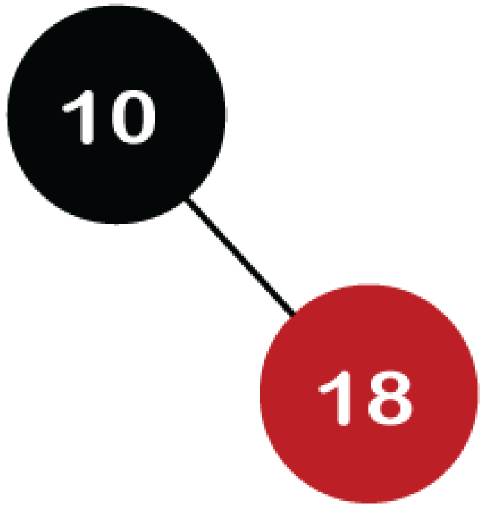


Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- **Step 2:** The next node is 18.
- As 18 is greater than 10 so it will come at the right of 10 as shown below.

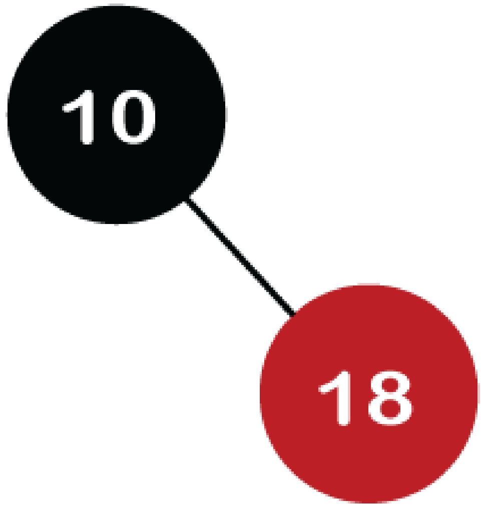


Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- **Second rule:** if the tree is not empty then the newly created node will have the Red color, therefore, node 18 has a Red color.



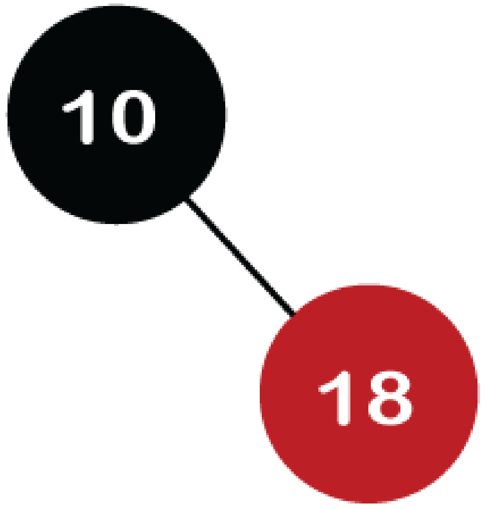
- **Third rule:** the parent of the new node is black or not, the parent of the node is black in color; therefore, it is a Red-Black tree.

Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- **Step 3:** Now, we create the new node having value 7 with Red color.
- As 7 is less than 10, so it will come at the left of 10 as shown below.

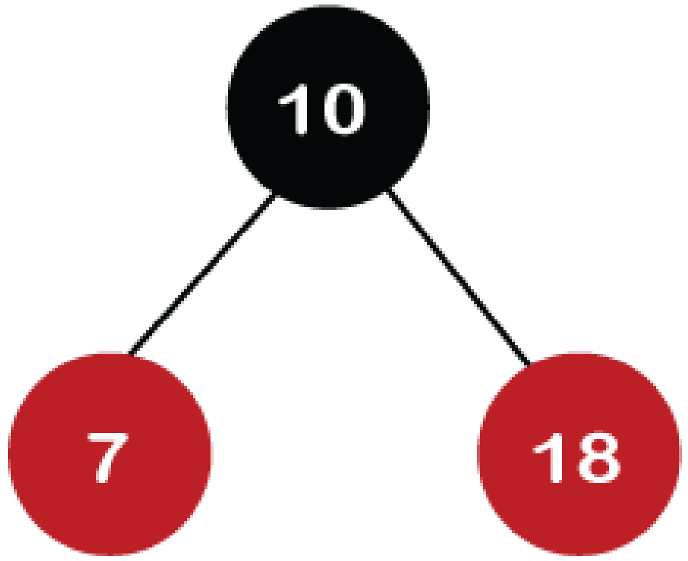


Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- **Step 3:** Now, we create the new node having value 7 with Red color.
- As 7 is less than 10, so it will come at the left of 10 as shown below.

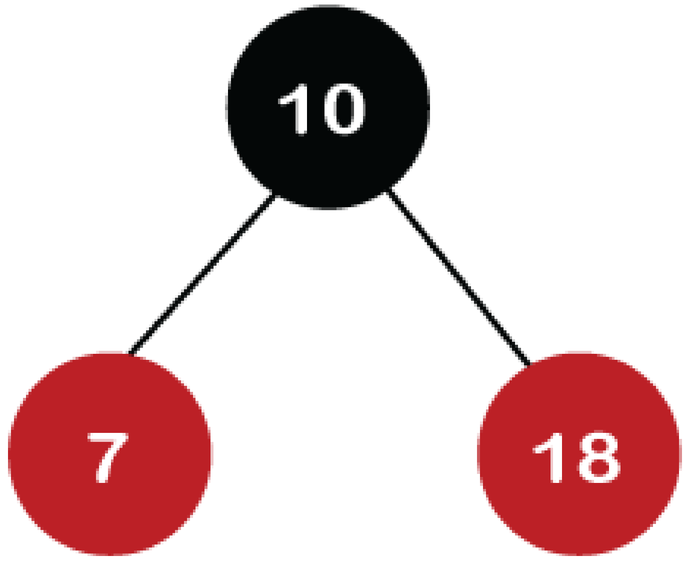


Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- **Third rule:** the parent of the new node is black or not?
- As we can observe, the parent of the node 7 is black in color, and it obeys the Red-Black tree's properties.

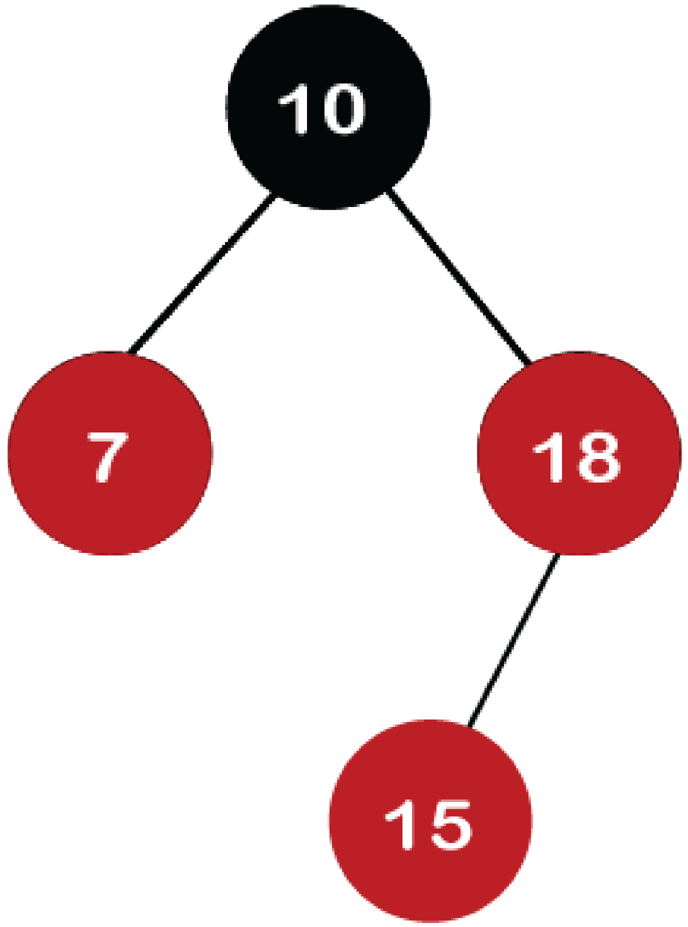


Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- **Step 4:** The next element is 15, and 15 is greater than 10, but less than 18, so the new node will be created at the left of node 18.
- The node 15 would be Red in color as the tree is not empty.

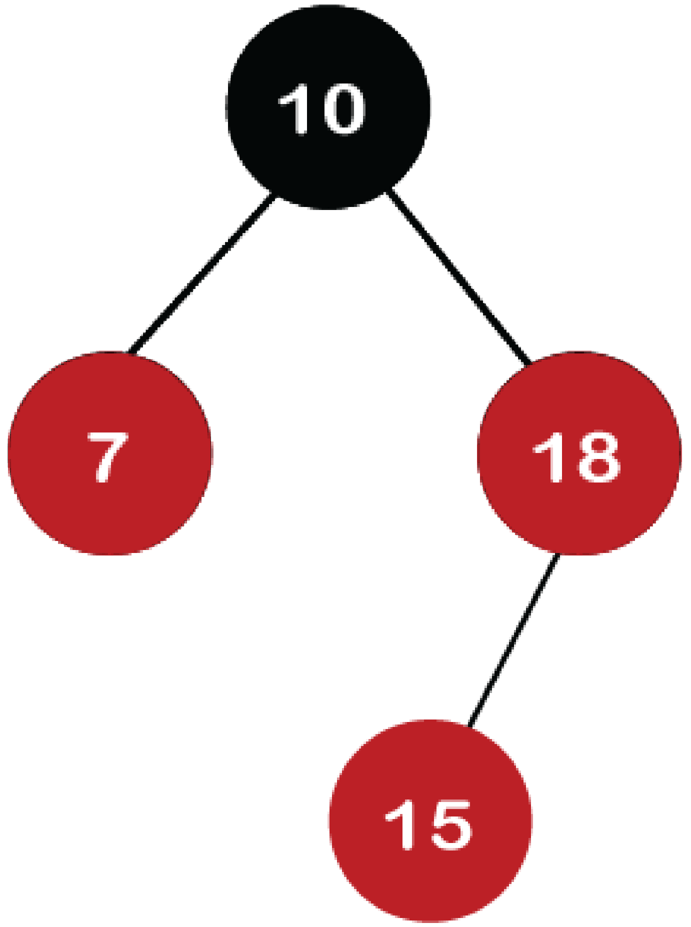


Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- The tree violates the property of the Red-Black tree as it has Red-red parent-child relationship.
- Now we have to apply some rule to make a Red-Black tree.
- The rule 4 says that ***if the new node's parent is Red, then we have to check the color of the parent's sibling of a new node.***

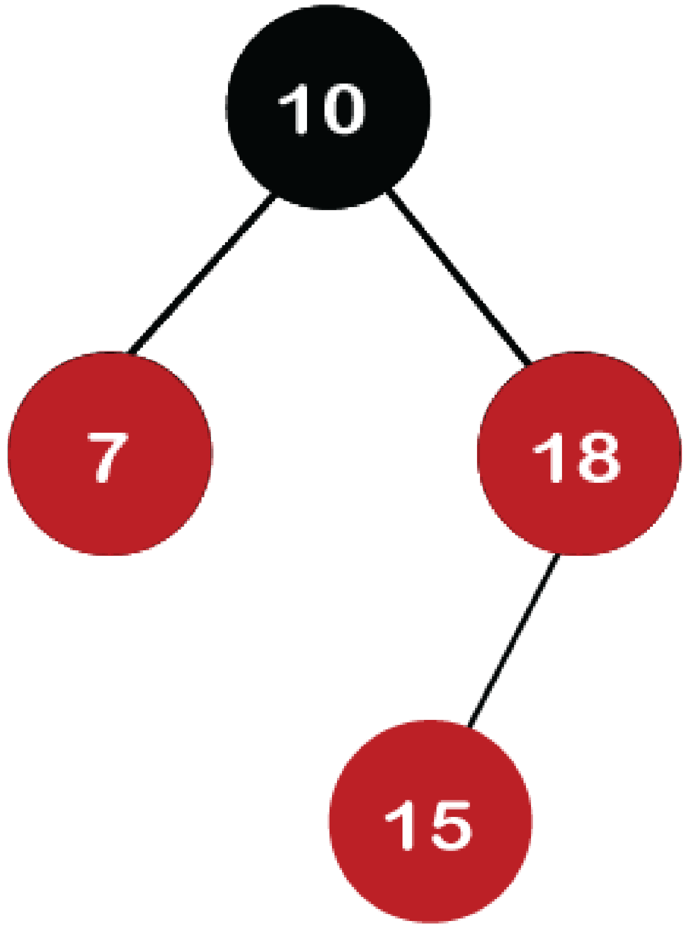


Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- The new node is node 15; the parent of the new node is node 18 and the sibling of the parent node is node 7.
- As the color of the parent's sibling is Red in color, so we apply the rule 4a.

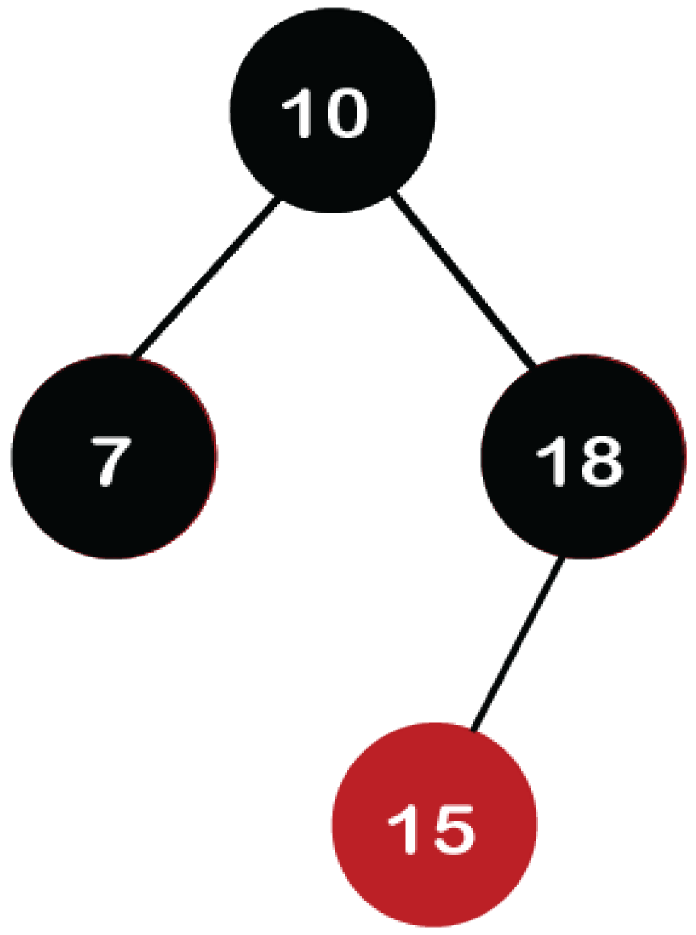


Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- The rule 4a says that we have to ***recolor both the parent and parent's sibling node***.
- So, both the nodes, i.e., 7 and 18, would be recolored as shown in the below figure.

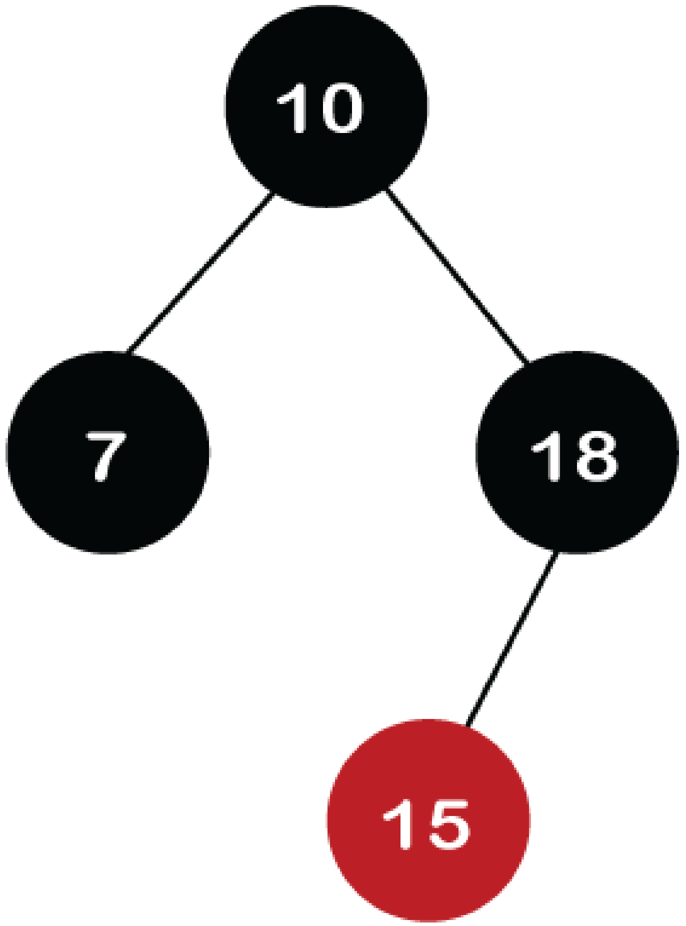


Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- We also have to check whether the parent's parent of the new node is the root node or not.
- As we can observe in the above figure, the parent's parent of a new node is the root node, so we do not need to recolor it.

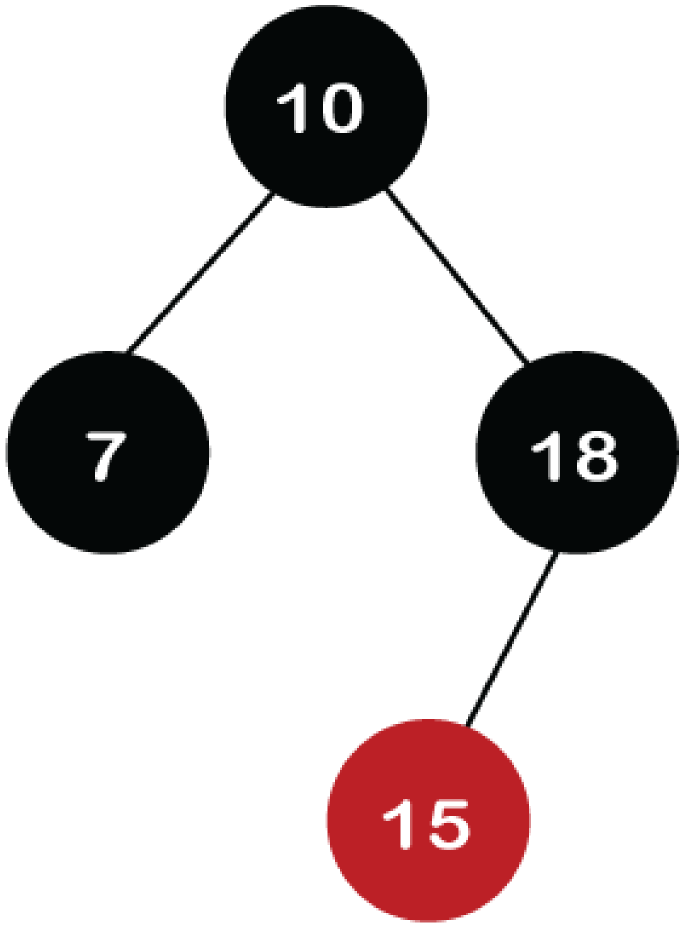


Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- **<u>Step 5:</u>** The next element is 16.
- As 16 is greater than 10 but less than 18 and greater than 15, so node 16 will come at the right of node 15.
- The tree is not empty; node 16 would be Red in color, as shown in the figure:

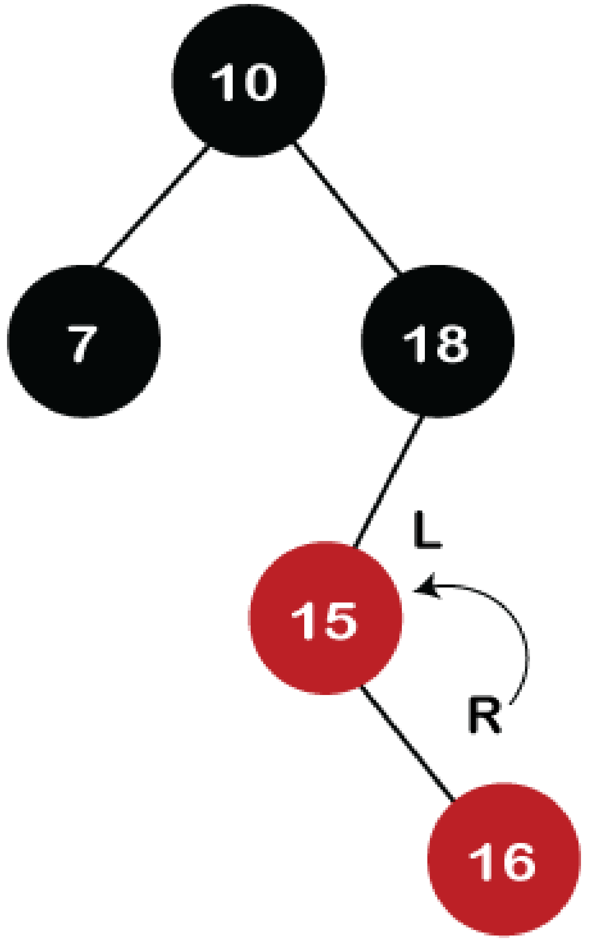


Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- **Step 5:** The next element is 16.
- As 16 is greater than 10 but less than 18 and greater than 15, so node 16 will come at the right of node 15.
- The tree is not empty; node 16 would be Red in color, as shown in the figure:

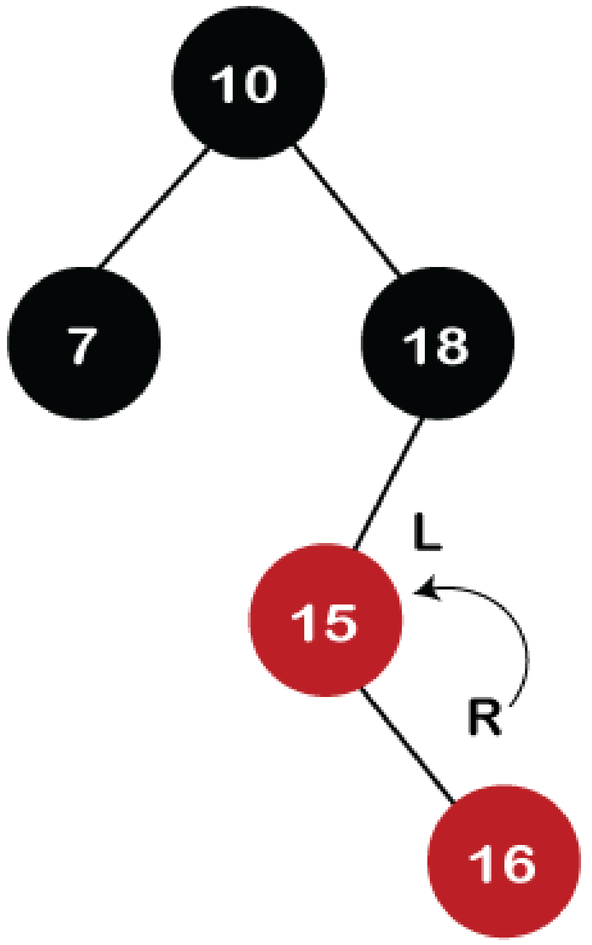


Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- In this figure, we can observe that it violates the property of the parent-child relationship as it has a red-red parent-child relationship.
- We have to apply some rules to make a Red-Black tree.

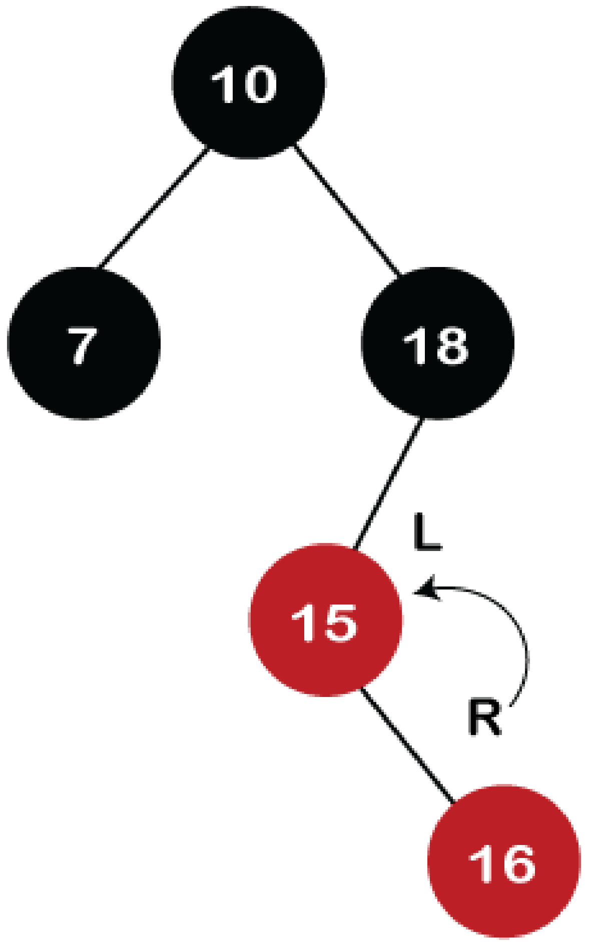


Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- Since the new node's parent is Red color, and the parent of the new node has no sibling, so rule **4a** will be applied.
- The rule **4a** says that some rotations and recoloring would be performed on the tree.

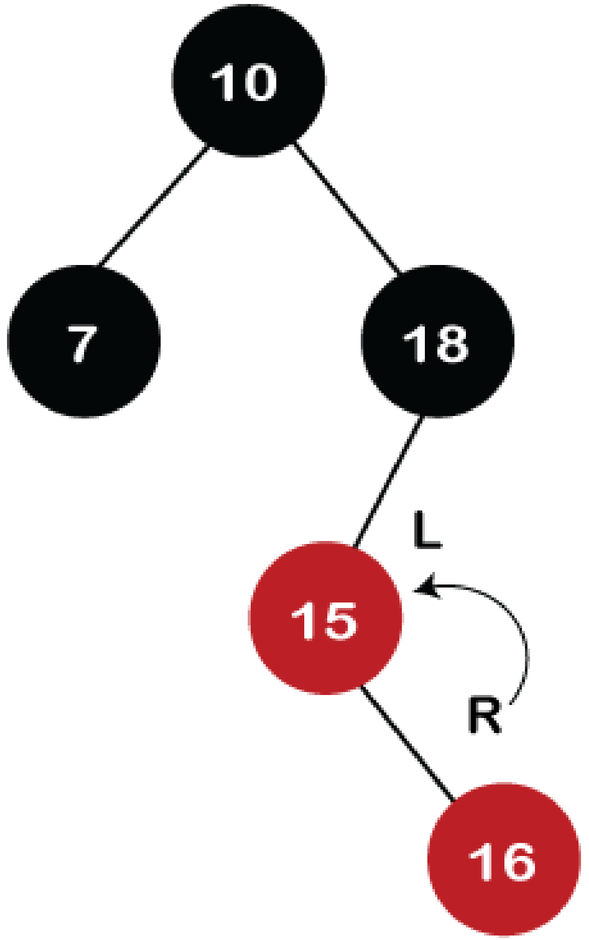


Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- Since node 16 is right of node 15 and the parent of node 15 is node 18.
- Node 15 is the left of node 18. Here we have **an LR** relationship, so we require to perform two rotations.
- First, we will perform left, and then we will perform the right rotation.

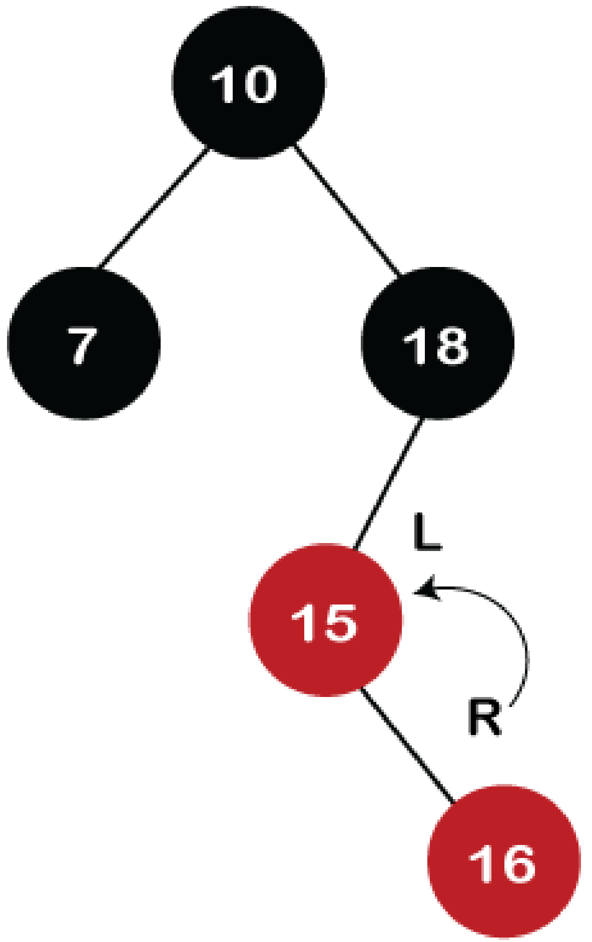


Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- The left rotation would be performed on nodes 15 and 16, where node 16 will move upward, and node 15 will move downward.
- Once the left rotation is performed, the tree looks like as shown in the figure:

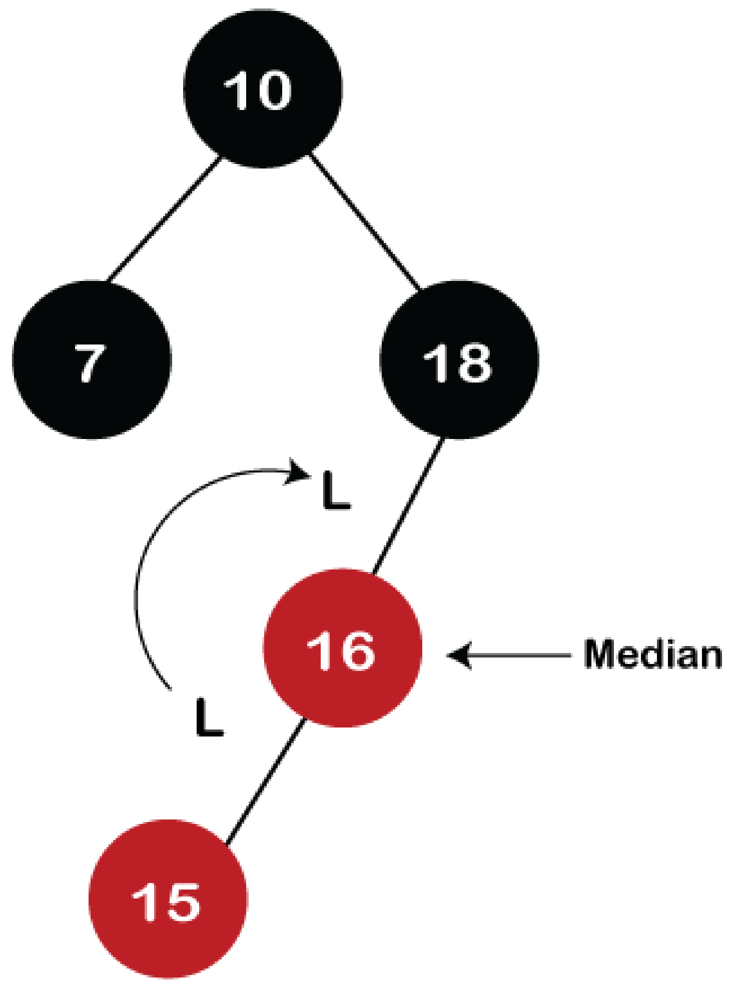


Balanced Trees cont...

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- The left rotation would be performed on nodes 15 and 16, where node 16 will move upward, and node 15 will move downward.
- Once the left rotation is performed, the tree looks like as shown in the figure:

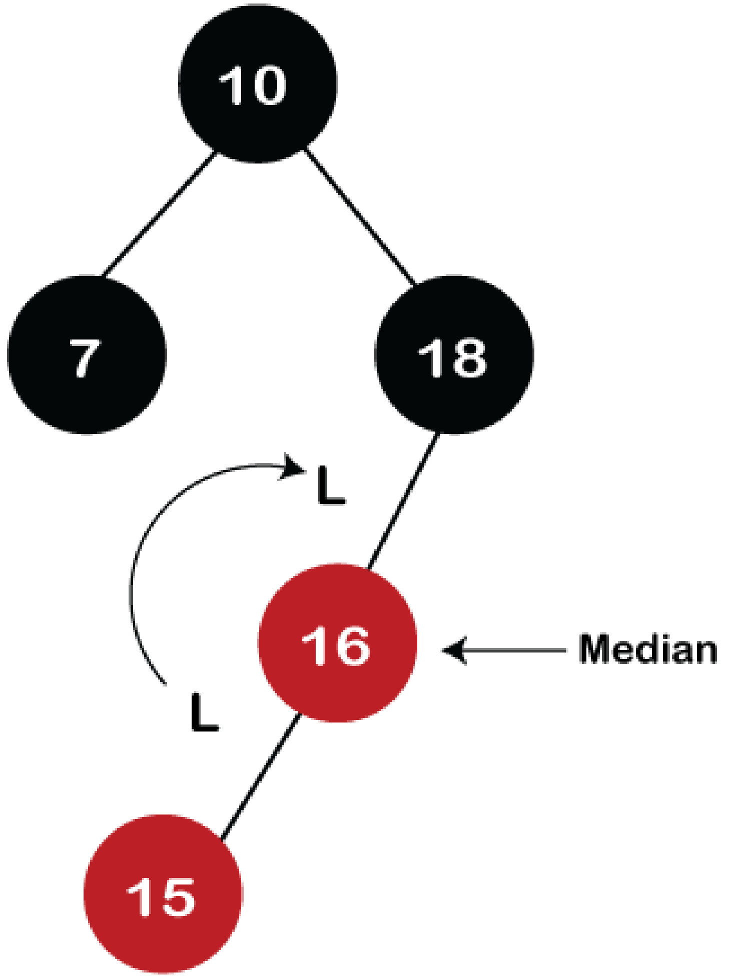


Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- In this figure, we can observe that there is **an LL** relationship.
- The above tree has a Red-red conflict, so we perform the right rotation.
- When we perform the right rotation, the median element would be the root node.

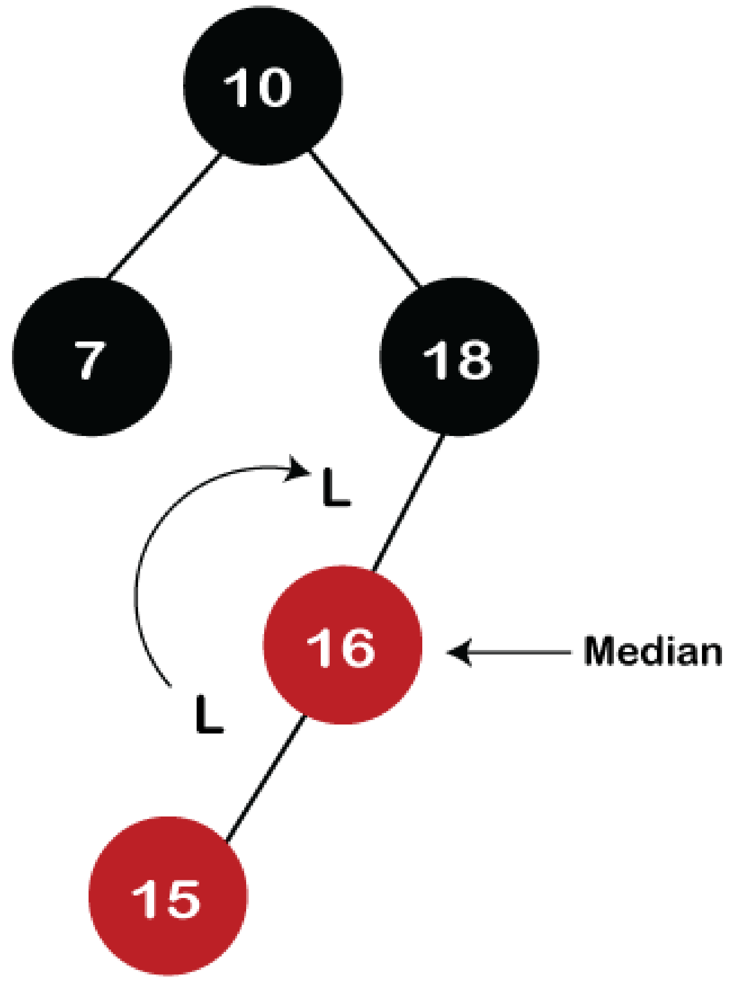


Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- Once the right rotation is performed, node 16 would become the root node, and nodes 15 and 18 would be the left child and right child, respectively, as shown in the figure.

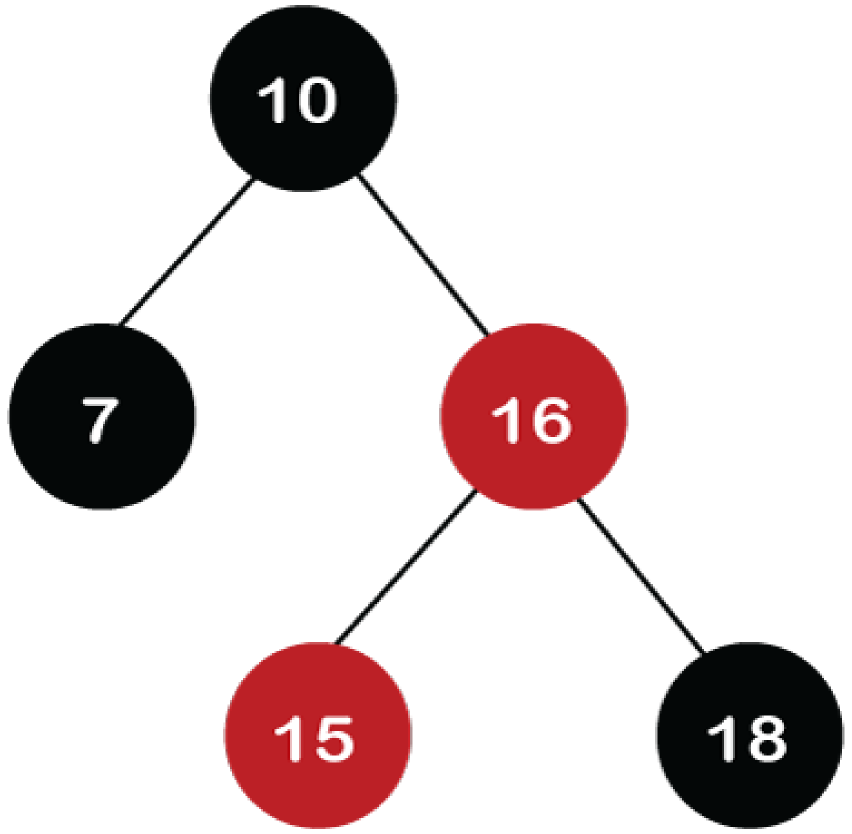


Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- Once the right rotation is performed, node 16 would become the root node, and nodes 15 and 18 would be the left child and right child, respectively, as shown in the figure.

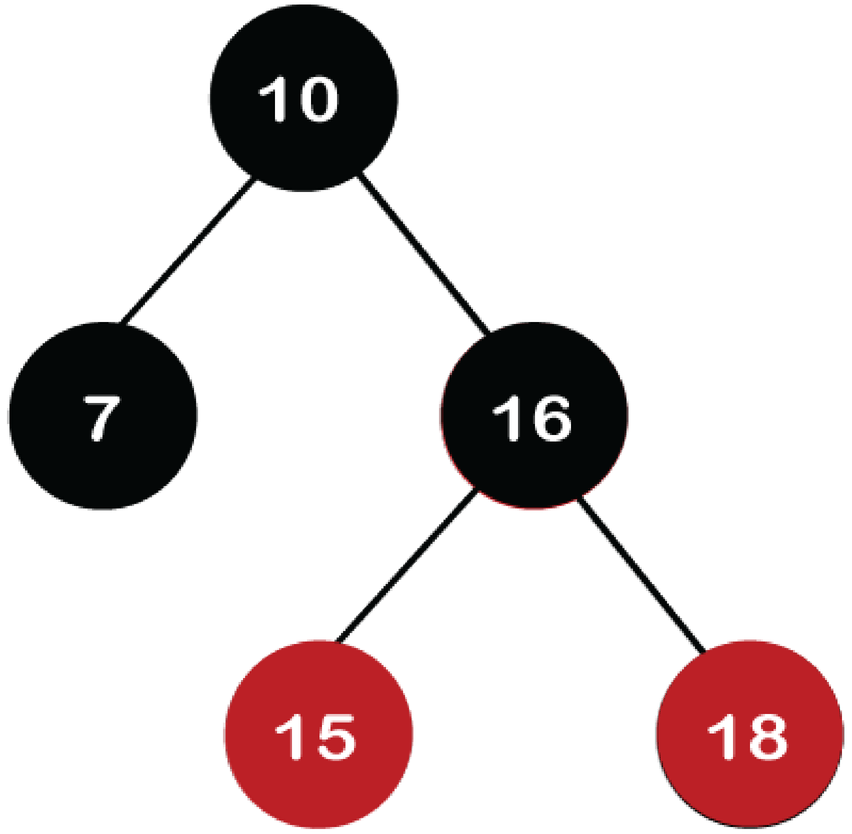


Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- After rotation, node 16 and node 18 would be recolored; the color of node 16 is red, so it will change to black, and the color of node 18 is black, so it will change to a red color as shown in the below figure:

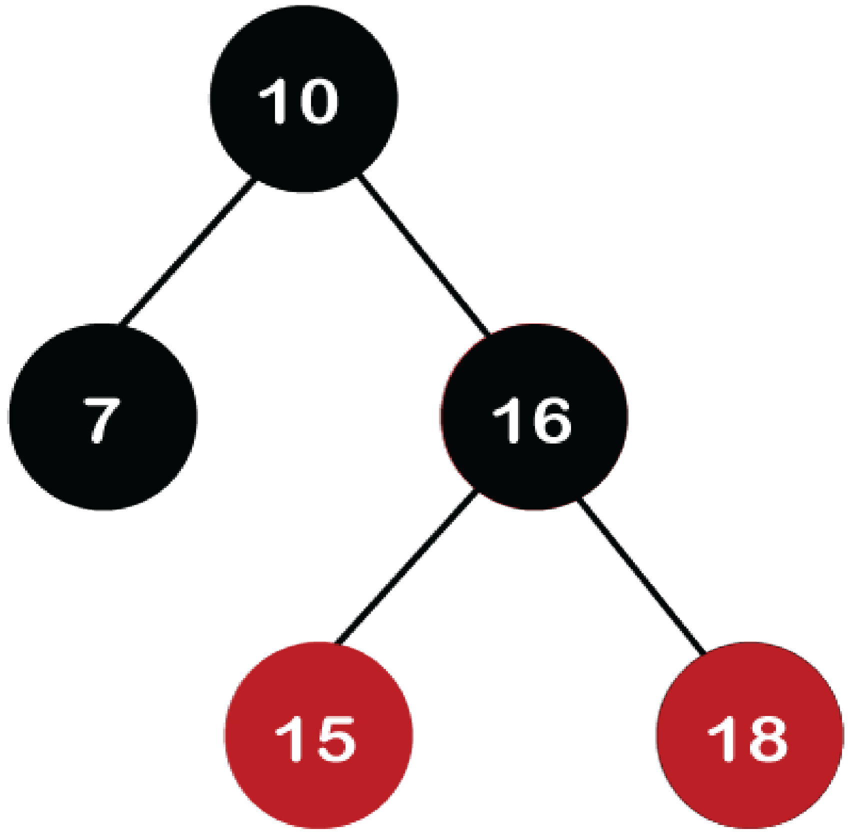


Balanced Trees cont...

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- **Step 6:** The next element is 30.
- Node 30 is inserted at the right of node 18.
- As the tree is not empty, so the color of node 30 would be red.

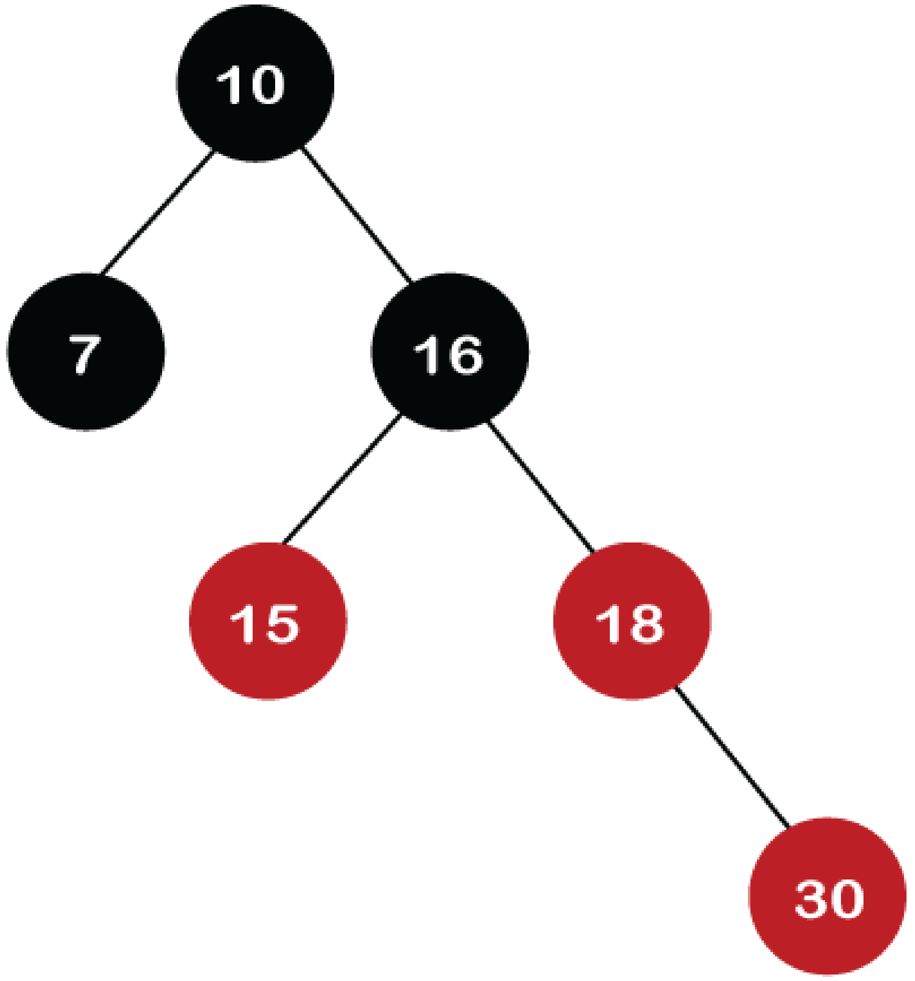


Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- **Step 6:** The next element is 30.
- Node 30 is inserted at the right of node 18.
- As the tree is not empty, so the color of node 30 would be red.

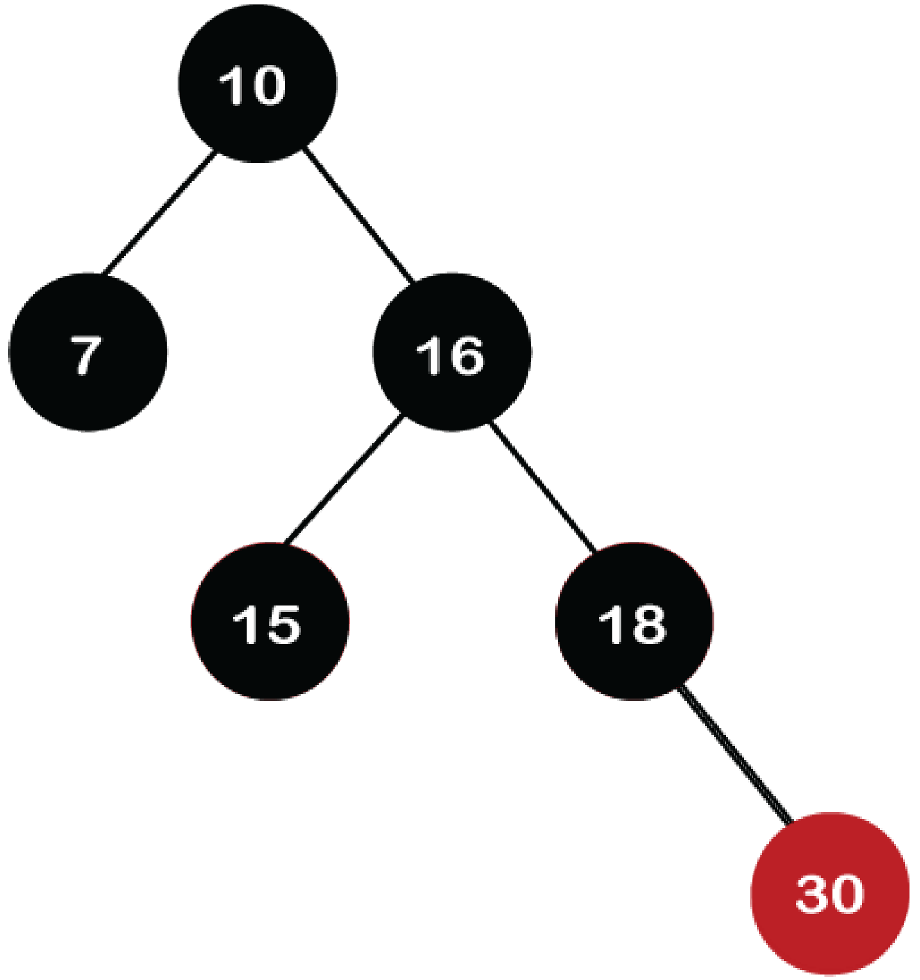


Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- The color of the parent and parent's sibling of a new node is Red, so rule 4b is applied.
- In rule 4b, we have to do only recoloring, i.e., no rotations are required.
- The color of both the parent (node 18) and parent's sibling (node 15) would become black, as shown in the below image.

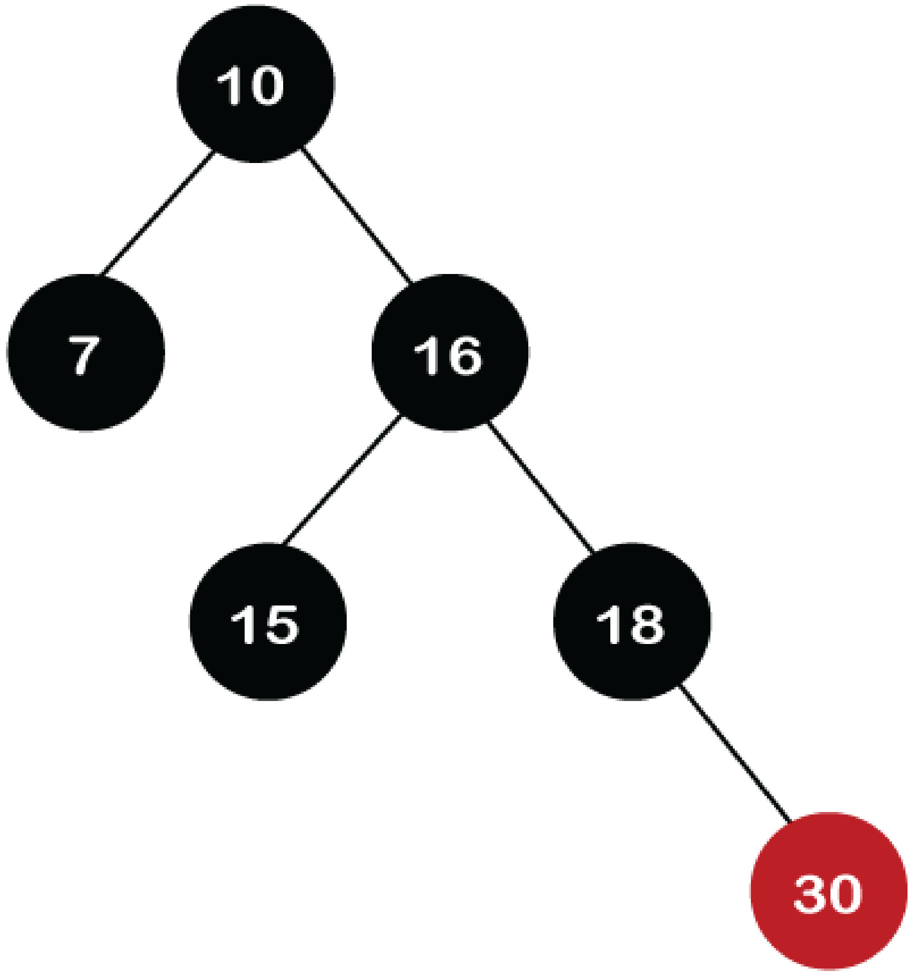


Balanced Trees cont…

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- We also have to check the parent's parent of the new node, whether it is a root node or not.
- The parent's parent of the new node, (i.e., node 30) is node 16 and node 16 is not a root node, so we will recolor the node 16 and changes to the Red color.
- The parent of node 16 is node 10, and it is not in Red color, so there is no Red-red conflict.

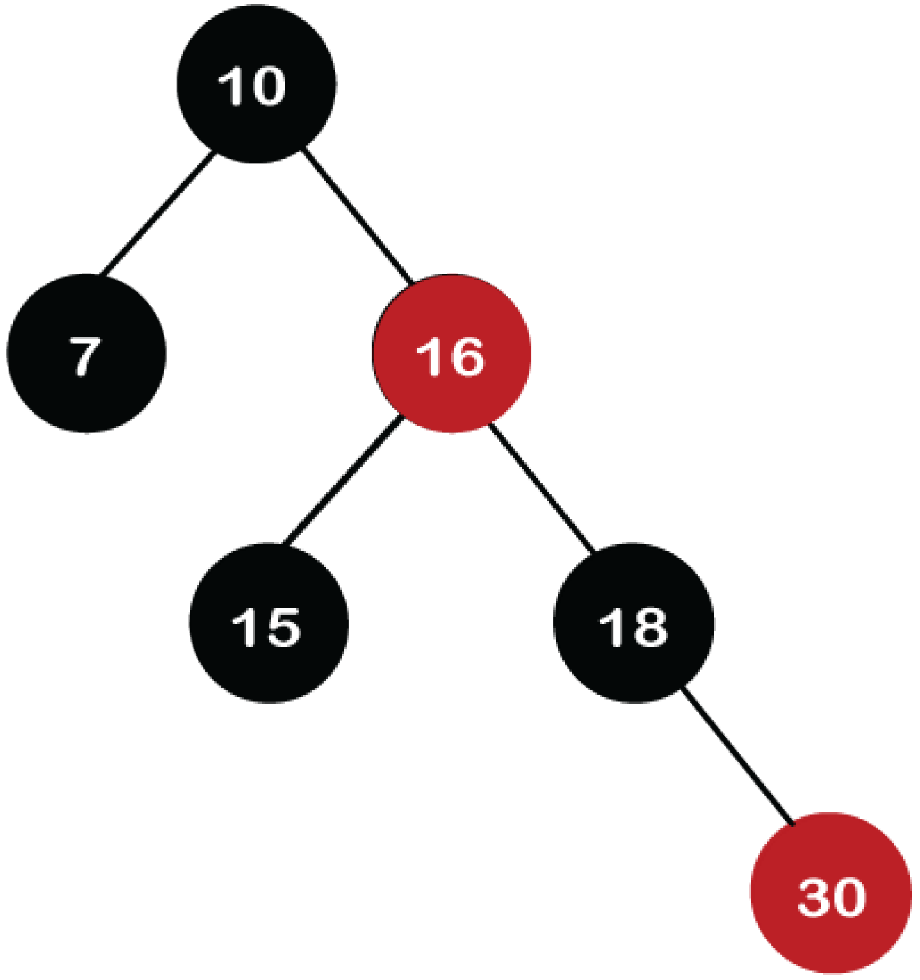


Balanced Trees cont...

Tree (Red Black Tree – Insertion in Red Black tree)

10, 18, 7, 15, 16, 30

- We also have to check the parent's parent of the new node, whether it is a root node or not.
- The parent's parent of the new node, (i.e., node 30) is node 16 and node 16 is not a root node, so we will recolor the node 16 and changes to the Red color.
- The parent of node 16 is node 10, and it is not in Red color, so there is no Red-red conflict.



Data Structure cont…
Tree (Splay Tree)

- Splay trees are the self-balancing or self-adjusted binary search trees.
- In other words, we can say that the splay trees are the variants of the binary search trees.
- The prerequisite for the splay trees, categorically is the binary search trees.

Data Structure cont…
Tree (Splay Tree)

- As we already know, the average case time complexity of a binary search tree is O(log n) and the time complexity in the worst case is O(n).
- In a binary search tree, the value of the left subtree is smaller than the root node, and the value of the right subtree is greater than the root node; in such case, the time complexity would be O(log n).
- If the binary tree is ***left-skewed*** or ***right-skewed***, then the time complexity would be O(n).

Data Structure cont…

Tree (Splay Tree)

- To limit the skewness, the AVL and Red-Black tree came into the existence, having O(log n) time complexity for all the operations in all the cases.
- We can also improve this time complexity by doing more practical implementations, so the new Tree data structure was designed, known as a Splay tree.

Data Structure cont…
Tree (Splay Tree)

- A splay tree is a self-balancing tree, but AVL and Red-Black trees are also self-balancing trees then, what makes the splay tree unique two trees.
- It has one extra property that makes it unique is splaying.
- A splay tree contains the same operations as a Binary search tree, i.e., Insertion, deletion and searching, but it also contains one more operation, i.e., splaying.
- So, all the operations in the splay tree are followed by splaying.

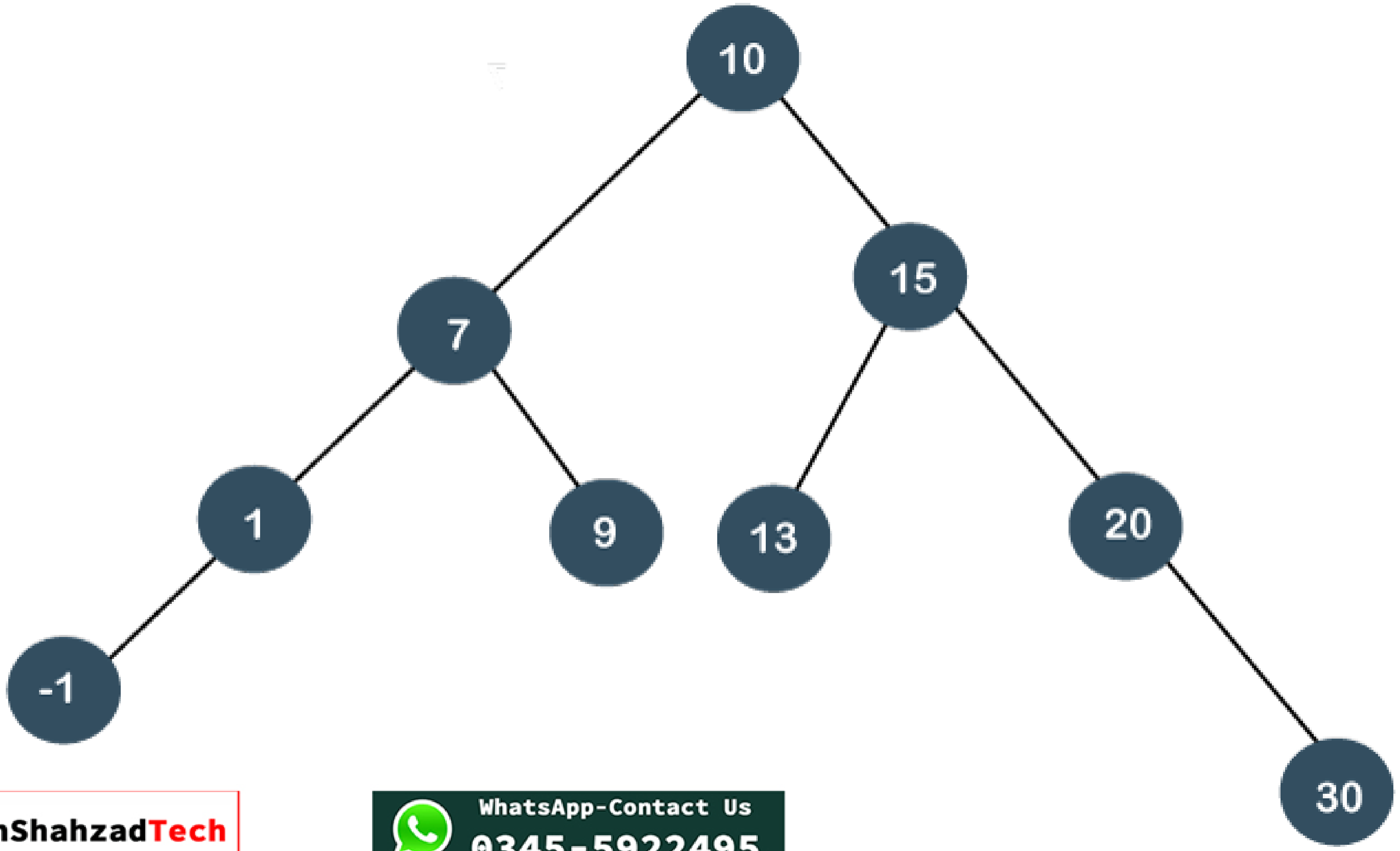
Data Structure cont…

Tree (Splay Tree)

- Splay trees are not strictly balanced, rather they are roughly balanced trees.
- Let's understand the search operation in the splay-tree.
- Suppose we want to search 7 element in the tree, which is shown below:

Data Structure cont…

Tree (Splay Tree - Search)



Data Structure cont…
Tree (Splay Tree - Search)

- After performing the search operation, we need to perform splaying.
- Here splaying means that the operation that we are performing on any element should become the root node after performing some rearrangements.
- The rearrangement of the tree will be done through the rotations.

Data Structure cont…
Tree (Splay Tree – Rotations)

- There are six types of rotations used for splaying:

1. Zig rotation (Right rotation)
2. Zag rotation (Left rotation)
3. Zig zag (Zig followed by zag)
4. Zag zig (Zag followed by zig)
5. Zig zig (two right rotations)
6. Zag zag (two left rotations)

Data Structure cont…
Tree (Splay Tree – Rotations – Factors)

- The following are the factors used for selecting a type of rotation:

1. Does the node which we are trying to rotate have a grandparent?
2. Is the node left or right child of the parent?
3. Is the node left or right child of the grandparent?

Data Structure cont…

Tree (Splay Tree – Rotations – Cases for the Rotations)

- **Case 1:** If the node ***does not*** have a **grand-parent**, and if it is the **right child** of the parent, then we carry out the ***left rotation***; otherwise, the ***right rotation*** is performed.
- **Case 2:** If the node has a **grandparent**, then based on the following scenarios; the rotation would be performed:

Data Structure cont…

Tree (Splay Tree – Rotations – Cases for the Rotations)

- **Scenario 1:** If the ***node is the right of the parent*** and the ***parent is also right of its parent*** (right skewed), then **(zig zig) right right** rotation is performed.
- **Scenario 2:** If the ***node is left of a parent***, but the ***parent is right of its parent*** (>), then **(zig zag) right left** rotation is performed .

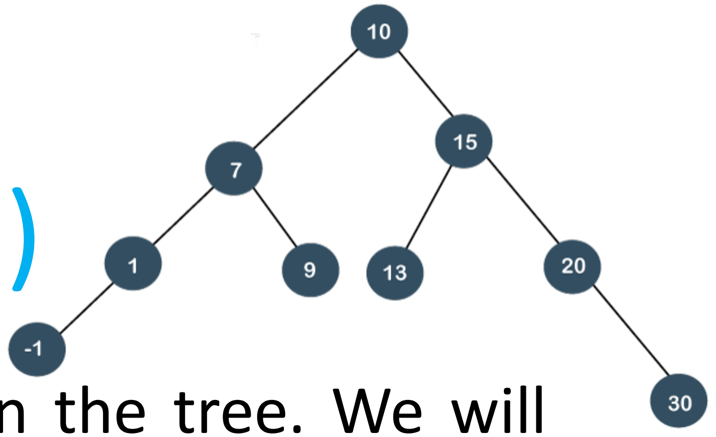
Data Structure cont…

Tree (Splay Tree – Rotations – Cases for the Rotations)

- **Scenario 3:** If the ***node is right of the parent*** and the ***parent is right of its parent***, then **(zig zig)** left left rotation is performed.
- **Scenario 4:** If the ***node is right of a parent***, but the ***parent is left of its parent*** (<), then **(zig zag) right-left** rotation is performed.

Data Structure cont…

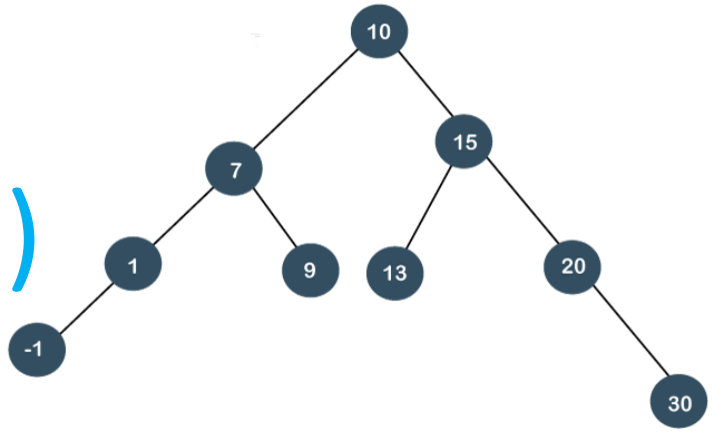
Tree (Splay Tree - Search)

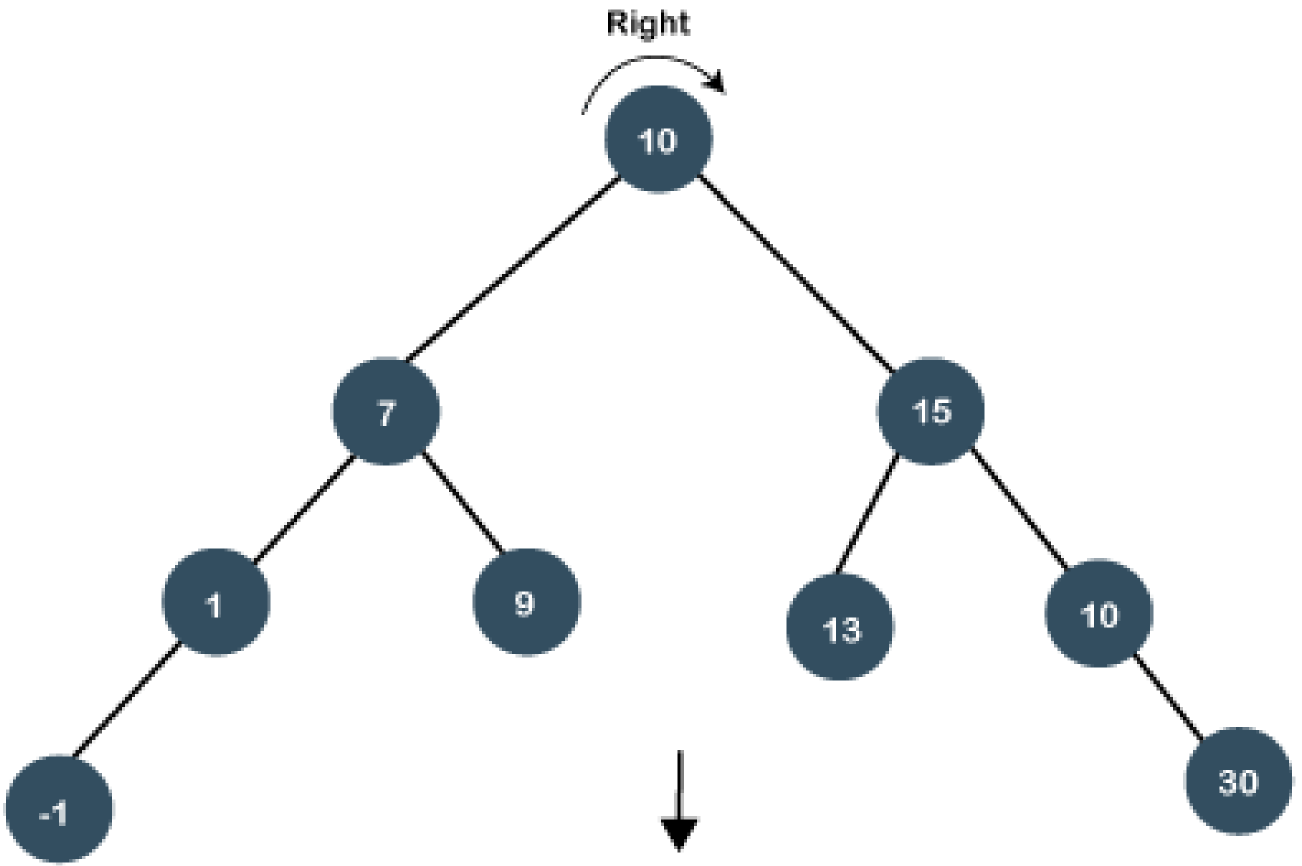


- In the above example, we have to search 7 element in the tree. We will follow the below steps:
- **Step 1:** First, we compare 7 with a root node. As 7 is less than 10, so it is a left child of the root node.
- **Step 2:** Once the element is found, we will perform splaying.
- The zig (right) rotation is performed so that 7 becomes the root node of the tree, as shown below:

Data Structure cont…

Tree (Splay Tree - Search)



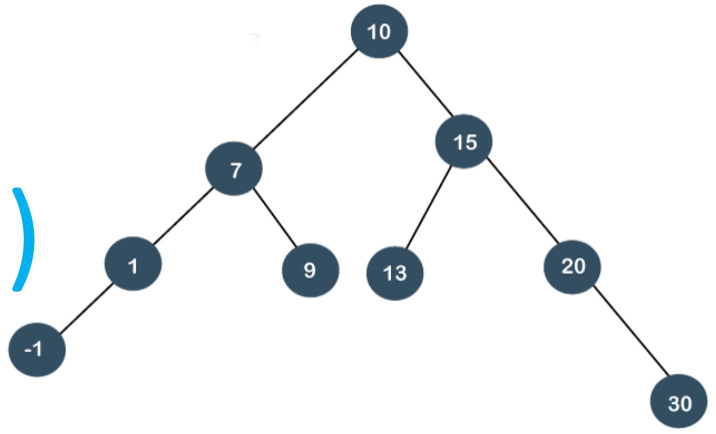


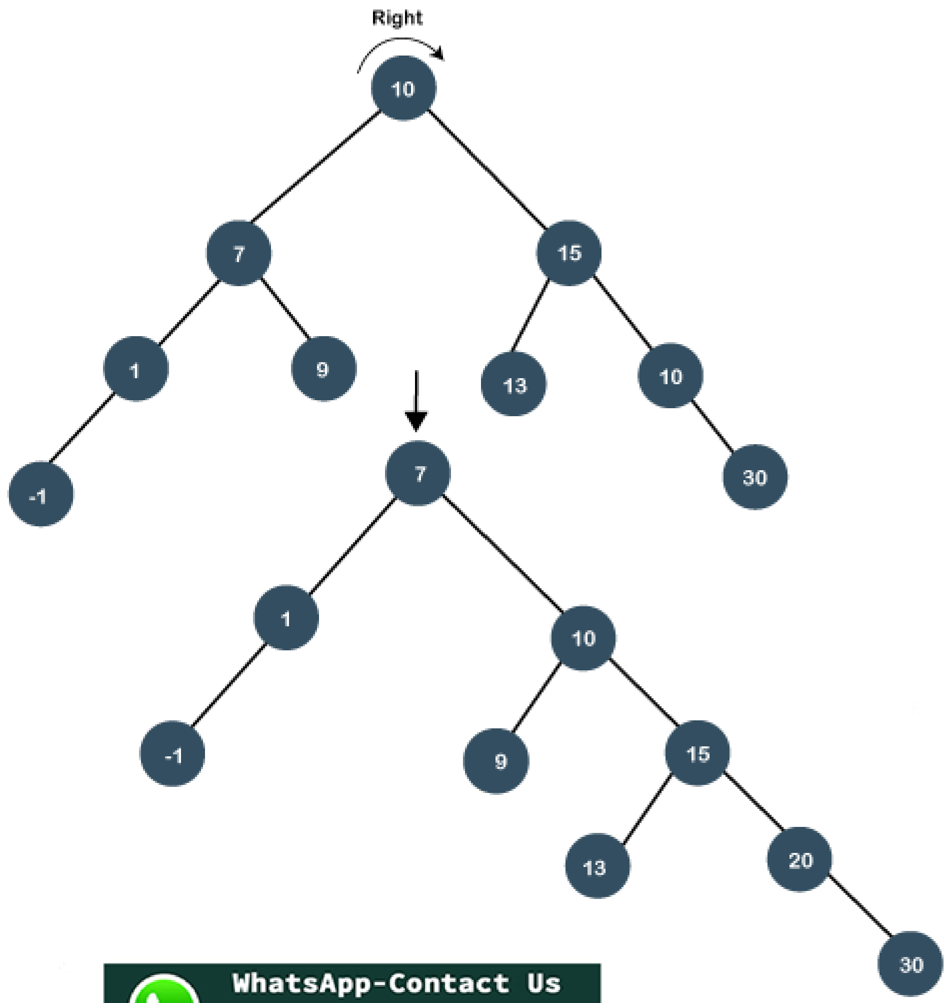
ArfanShahzadTech

WhatsApp-Contact Us
0345-5922495

Data Structure cont...

Tree (Splay Tree - Search)





ArfanShahzadTech

WhatsApp-Contact Us
0345-5922495

Data Structure cont…
Tree (Treap)

- Treap data structure is a hybrid of a ***binary search tree*** and a ***heap***.
- We already have enough knowledge about binary search tree.
- However, we have to understand heap:

Data Structure cont…

Tree (Treap – Heap)

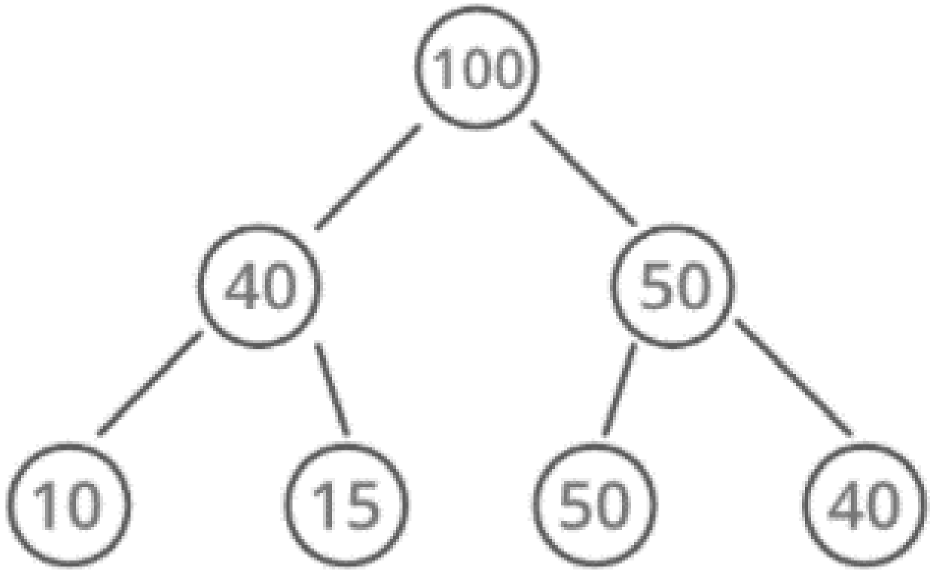
- A Heap is a special Tree-based data structure in which the tree is a complete binary tree.
- Generally, Heaps can be of two types:

1. Max-Heap
2. Min-Heap

Data Structure cont…

Tree (Treap – Heap)

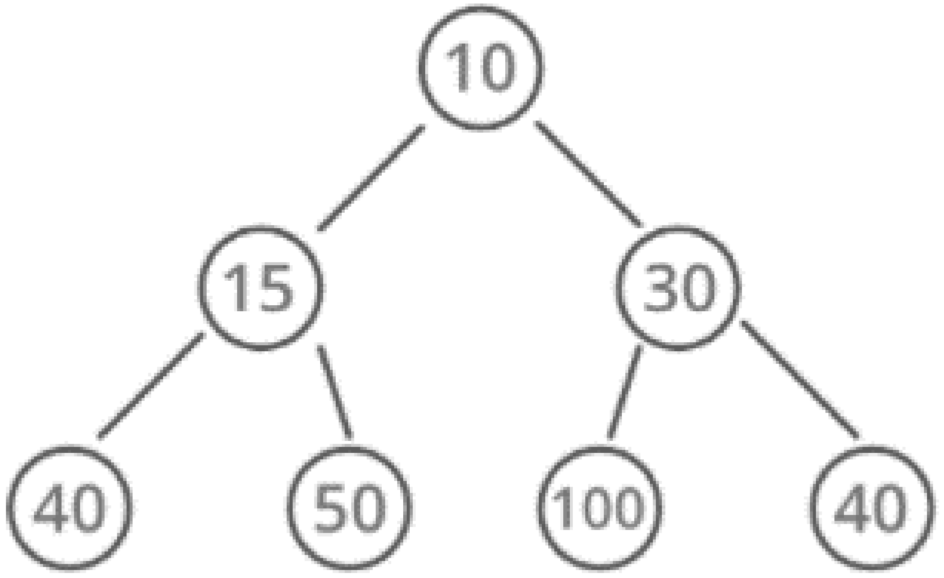
- **Max-Heap:** In a Max-Heap the key present at the root node must be greatest among the keys present at all of it’s children.
- The same property must be recursively true for all sub-trees in that Binary Tree.



Data Structure cont…

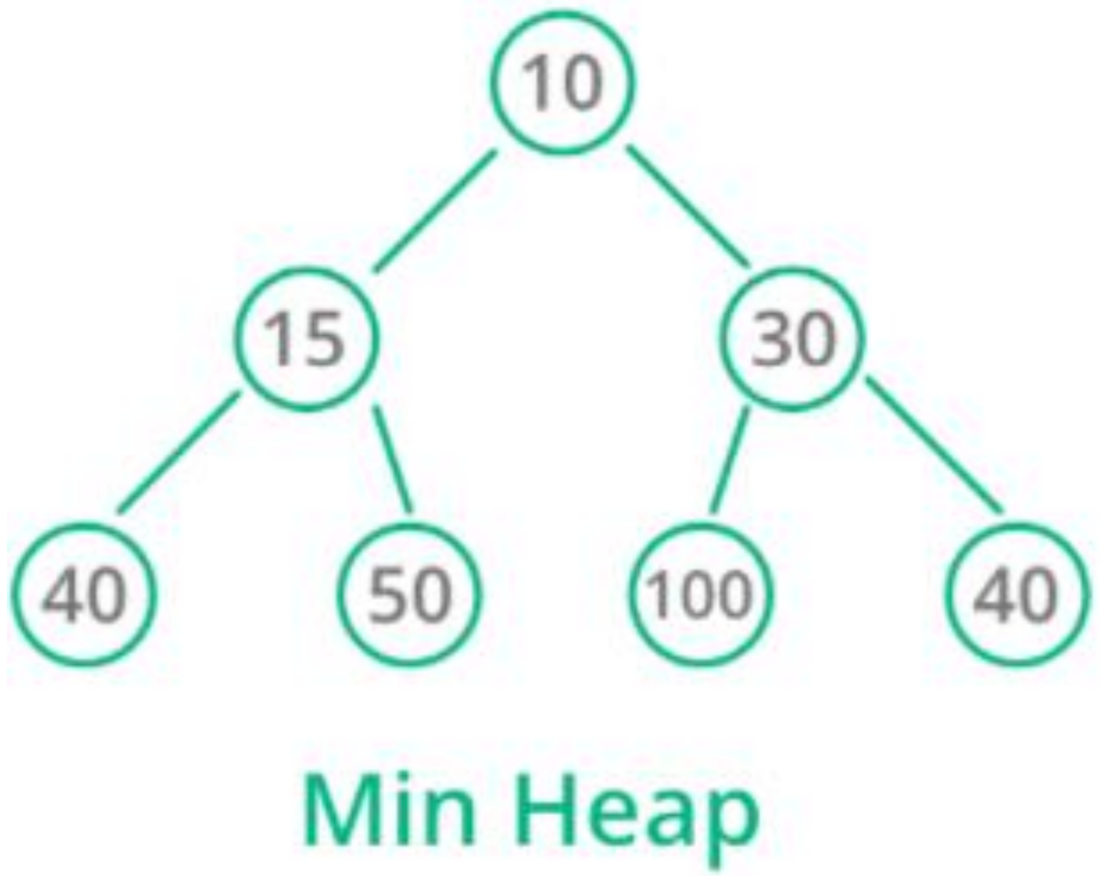
Tree (Treap – Heap)

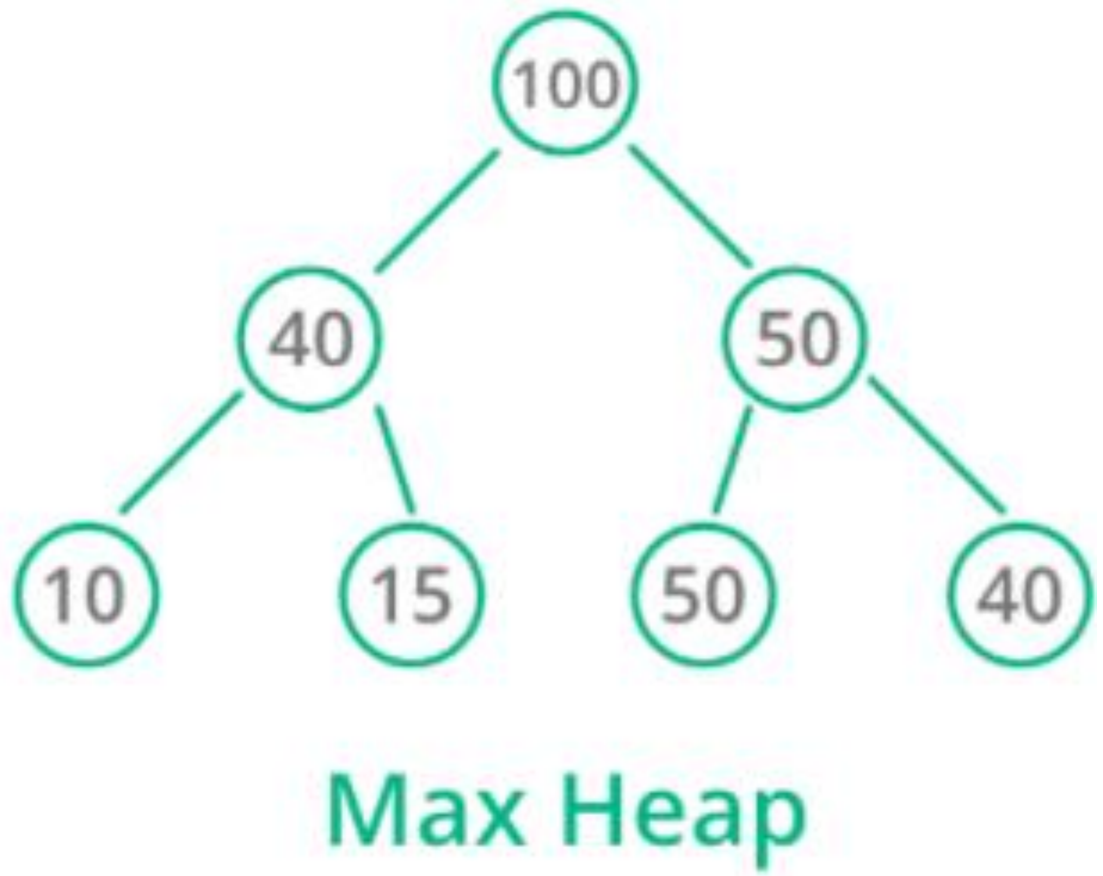
- **Min-Heap:** In a Min-Heap the key present at the root node must be minimum among the keys present at all of it's children.
- The same property must be recursively true for all sub-trees in that Binary Tree.



Data Structure cont…

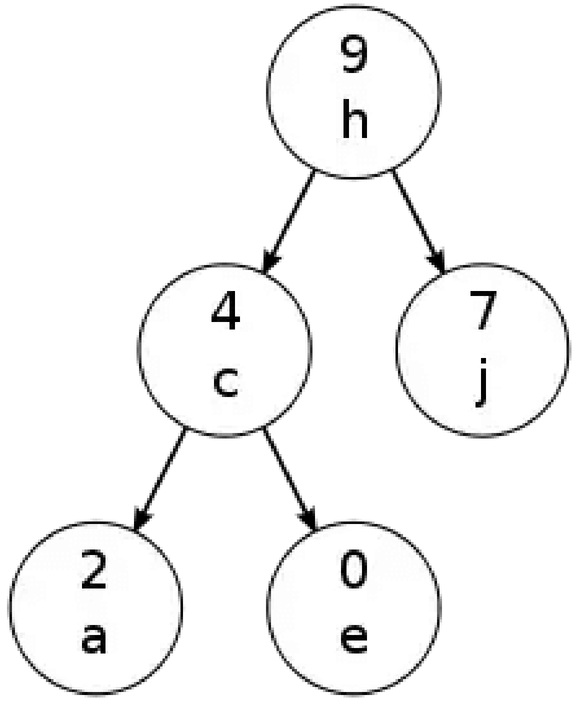
Tree (Treap – Heap)





Data Structure cont…
Tree (Treap)

- A treap data structure is a hybrid of heaps and binary search trees.
- When we create a heap, we are essentially creating an ordered binary tree that also satisfies the "heap" characteristic.

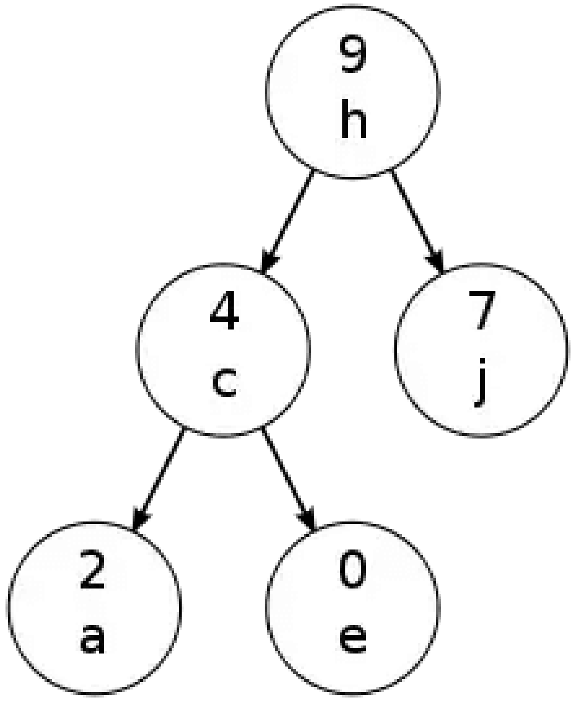


ArfanShahzadTech

WhatsApp-Contact Us
0345-5922495

Data Structure cont…
Tree (Treap)

- The numbers represent the heap arrangement of the data structure (in max-heap order), while the alphabets represent the tree part.
- So we have a tree and a heap now.



Data Structure cont…
Tree (B-Tree)

- B-Tree is a self-balancing search tree.
- In most of the other self-balancing search trees (e.g.: AVL, Red-Black and splay trees), it is assumed that everything is in main memory.
- B-trees were originally invented for storing data structures on disk, where locality is even more crucial than with memory.

Data Structure cont…
Tree (B-Tree)

- Disk access time is very high compared to the main memory access time.
- The main idea of using B-Trees is to reduce the number of disk accesses.
- Most of the tree operations (search, insert, delete) require O(h) disk accesses where h is the height of the tree.

Data Structure cont…

Tree (B-Tree)

- A B-tree of order *m* (maximum ***number of children <u>m</u>***) is a search tree in which each nonleaf node has **up to *<u>m</u>* children**.
- The actual elements of the collection are stored in the leaves of the tree, and the nonleaf nodes contain keys only for direction.
- Each leaf stores some ***number of elements***; the maximum number may be greater or (typically) ***less than <u>m</u>***.
- The data structure satisfies several properties:

Data Structure cont…

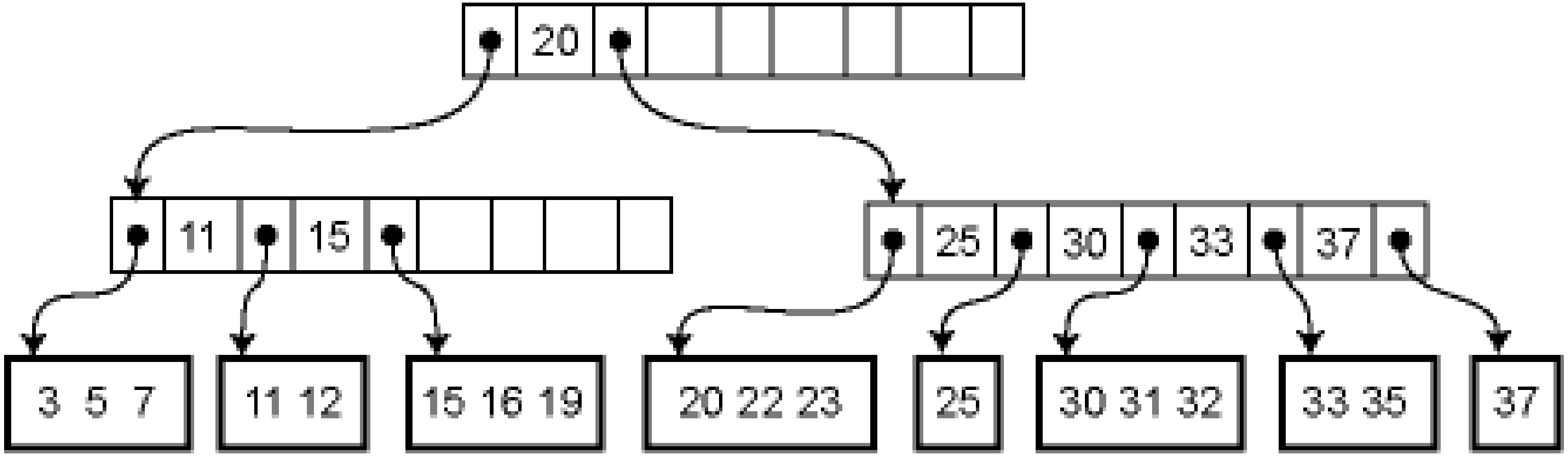
Tree (B-Tree)

1. Every **path** from the ***root to a leaf*** has the **same length**
2. If a node has ***n children***, it contains ***n−1 keys***.
3. Every node (except the root) has at least $\lceil m/2 \rceil$ child nodes (**half full**).
4. The elements stored in a given subtree all have keys that are between the keys in the parent node on either side of the subtree pointer.
5. The root has at least two children if it is not a leaf.

Data Structure cont…

Tree (B-Tree)

- For example, the following is an order-5 B-tree (m=5) where the leaves have enough space to store up to 3 data records:

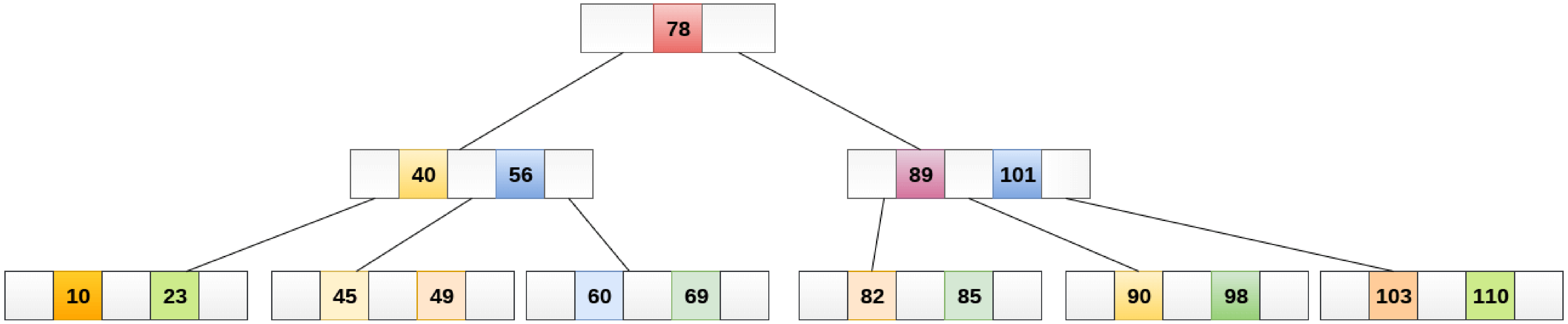


Data Structure cont…
Tree (B-Tree – Operations)

- While performing some operations on B Tree, any property of B Tree may violate such as number of minimum children a node can have.
- Let us discuss different operations on B-Tree.

Data Structure cont...
Tree (B-Tree – Operations – Search)

- Searching in B Trees is similar to that in Binary search tree.
- For example, if we search for an item 49 in the following B Tree.

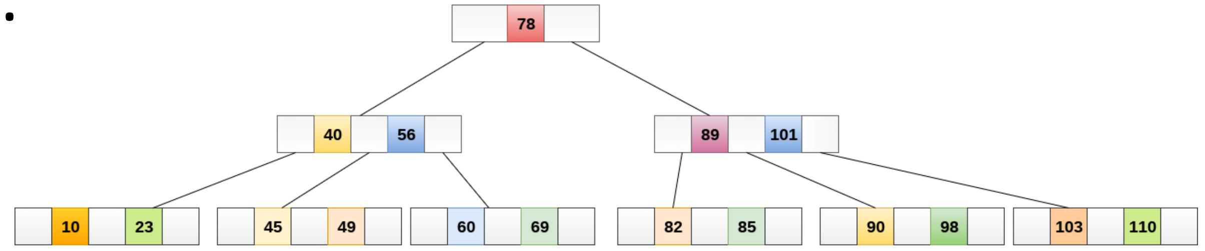


Data Structure cont…

Tree (B-Tree – Operations – Search)

- The process will something like following :

1. Compare item 49 with root node 78. since 49 < 78 hence, move to its left sub-tree.
2. Since, 40<49<56, traverse right sub-tree of 40.
3. 49>45, move to right. Compare 49.
4. match found, return.



Data Structure cont…
Tree (B-Tree – Operations – Search)

- Searching in a B tree depends upon the height of the tree.
- The search algorithm takes O(log n) time to search any element in a B tree.

Data Structure cont…
Tree (B-Tree – Operations – Insertion)

- Insertions are done at the leaf node level.
- The following algorithm needs to be followed in order to insert:

1. Traverse the B Tree in order to find the appropriate leaf node at which the node can be inserted.
2. If the leaf node contain less than ***m-1 keys*** then insert the element in the ***increasing order***.
3. Else, if the leaf node contains m-1 keys, then follow the following steps:

Data Structure cont…
Tree (B-Tree – Operations – Insertion)

a) Insert the new element in the increasing order of elements.

b) Split the node into the two nodes at the ***median***.

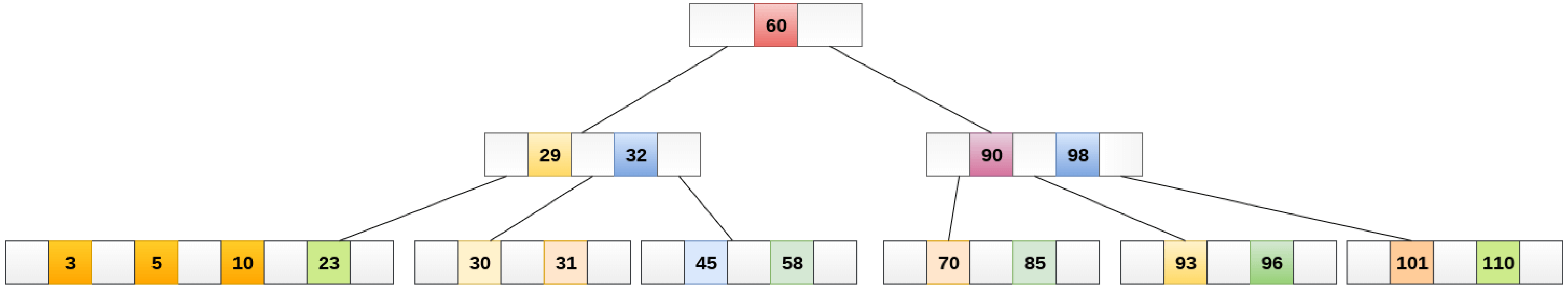
c) Push the median element up to its parent node.

d) If the parent node also contain m-1 number of keys, then split it too by following the same steps.

Data Structure cont…

Tree (B-Tree – Operations – Insertion)

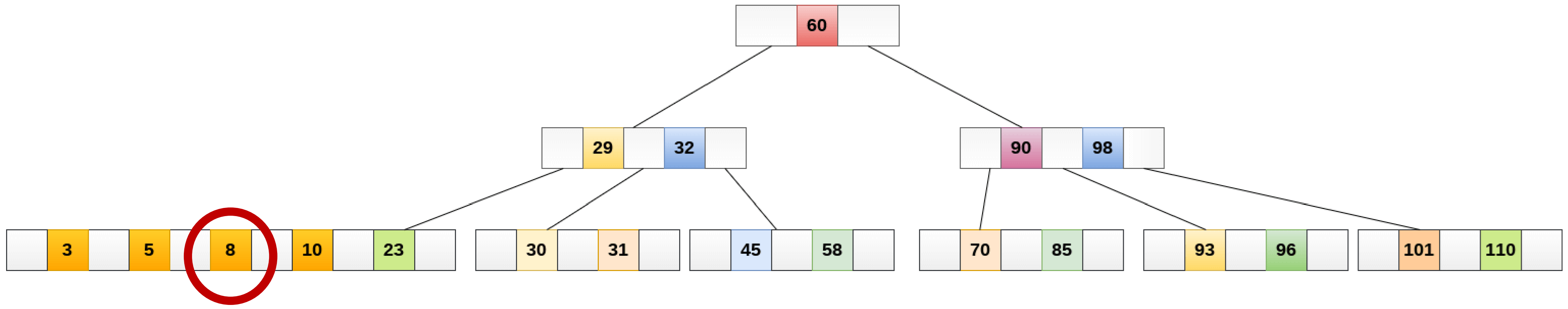
- Insert the node 8 into the B Tree of order 5 shown in the following image.



- 8 will be inserted to the right of 5, therefore insert 8.

Data Structure cont...

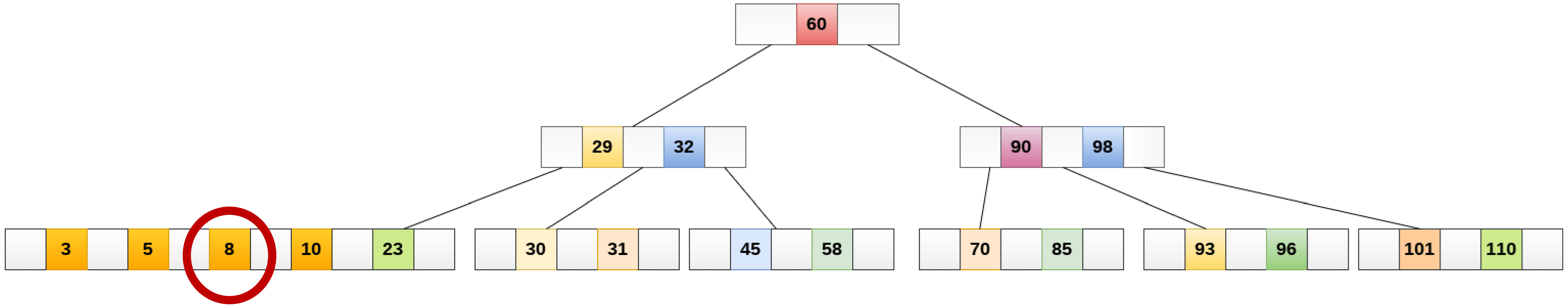
Tree (B-Tree – Operations – Insertion)

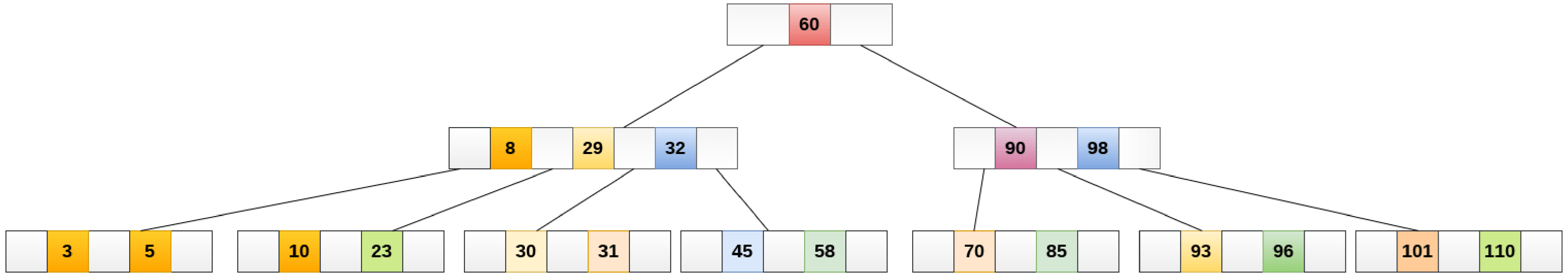


- The node, now contain 5 keys which is greater than (5 -1 = 4) keys.
- Therefore split the node from the median i.e. 8 and push it up to its parent node shown as follows:

Data Structure cont…

Tree (B-Tree – Operations – Insertion)





Data Structure cont…
Tree (B-Tree – Operations – Deletion)

- Deletion is also performed at the leaf nodes.
- The node which is to be deleted can either be a leaf node or an internal node.
- Following algorithm needs to be followed in order to delete a node from a B tree.

Data Structure cont…
Tree (B-Tree – Operations – Deletion)

1. Locate the leaf node.
2. If there are more than m/2 keys in the leaf node then delete the desired key from the node.
3. If the leaf node doesn't contain m/2 keys then complete the keys by taking the element from right or left sibling.

Data Structure cont…

Tree (B-Tree – Operations – Deletion)

a) If the left sibling contains more than m/2 elements then push its largest element up to its parent and move the intervening element down to the node where the key is deleted.

b) If the right sibling contains more than m/2 elements then push its smallest element up to the parent and move intervening element down to the node where the key is deleted.

Data Structure cont…

Tree (B-Tree – Operations – Deletion)

4. If neither of the sibling contain more than m/2 elements then create a new leaf node by joining two leaf nodes and the intervening element of the parent node.
5. If parent is left with less than m/2 nodes then, apply the above process on the parent too.

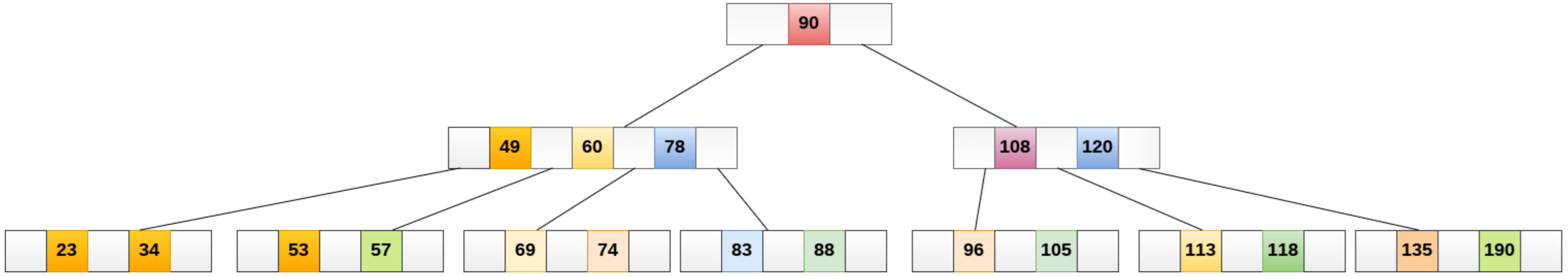
Data Structure cont…

Tree (B-Tree – Operations – Deletion)

- If the node which is to be deleted is an internal node, then replace the node with its in-order successor or predecessor.
- Since, successor or predecessor will always be on the leaf node hence, the process will be similar as the node is being deleted from the leaf node.

Data Structure cont…
Tree (B-Tree – Operations – Deletion)

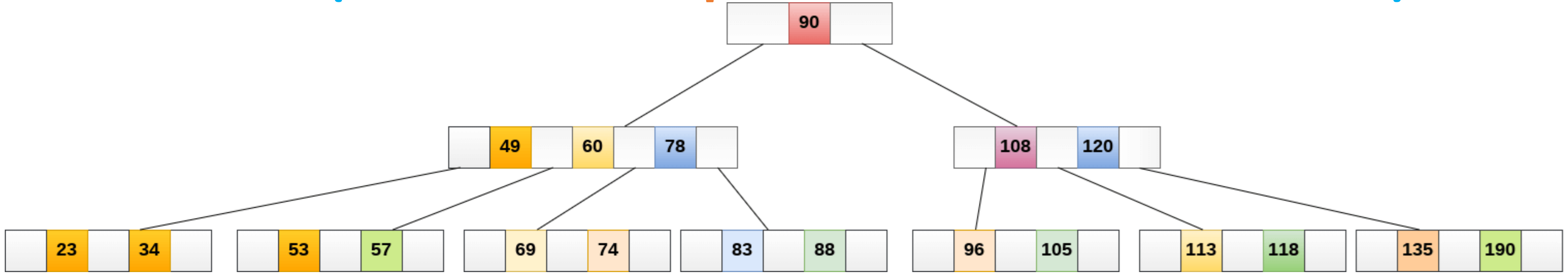
- Delete the node 53 from the B Tree of order 5 shown in the following figure:

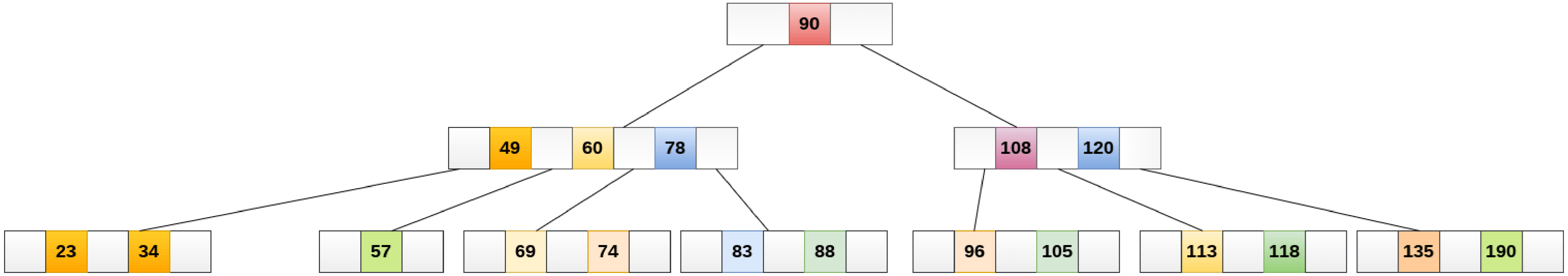


- 53 is present in the right child of element 49. Delete it.

Data Structure cont…

Tree (B-Tree – Operations – Deletion)



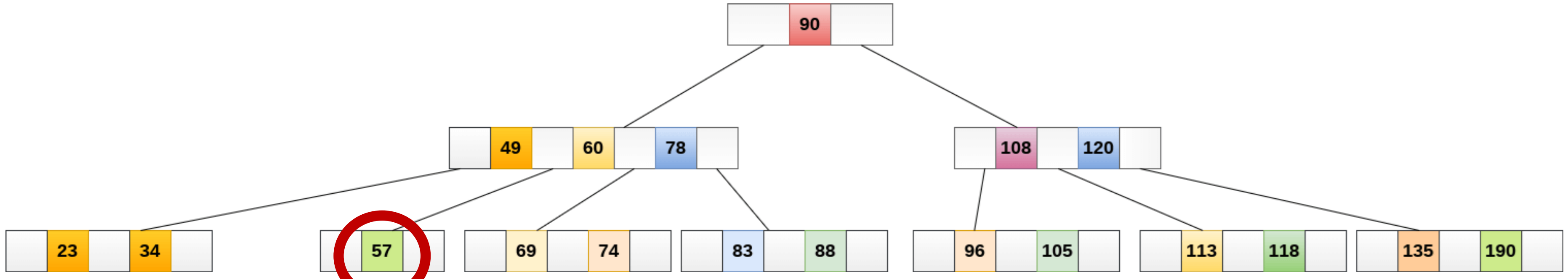


ArfanShahzadTech

WhatsApp-Contact Us
0345-5922495

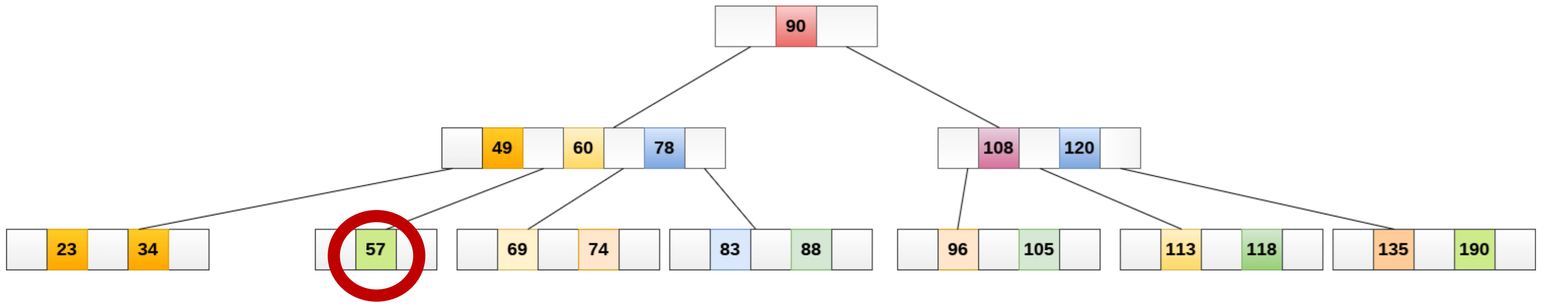
Data Structure cont…
Tree (B-Tree – Operations – Deletion)

- Now, 57 is the only element which is left in the node, the minimum number of elements that must be present in a B tree of order 5, is 2.
- it is less than that, the elements in its left and right sub-tree are also not sufficient so, merge it with left sibling and intervening element of parent i.e. 49.

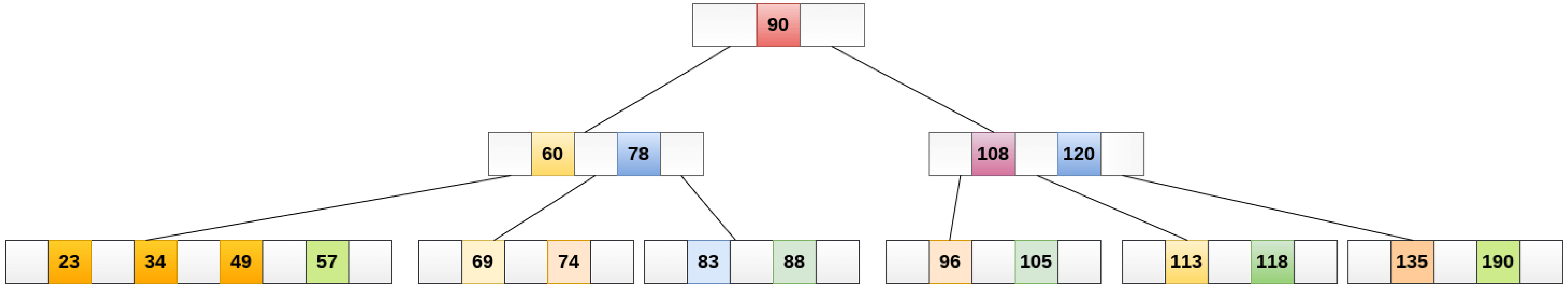


Data Structure cont…

Tree (B-Tree – Operations – Deletion)



- The final B tree is shown as follows:



ArfanShahzadTech

WhatsApp-Contact Us
0345-5922495

Data Structure cont...

Tree (B-Tree – Applications of B Tree)

- B tree is used to index the data and provides fast access to the actual data stored on the disks since, the access to value stored in a large database that is stored on a disk is a very time consuming process.
- Searching an un-indexed and unsorted database containing n key values needs O(n) running time in worst case.
- However, if we use B Tree to index this database, it will be searched in O(log n) time in worst case.

Data Structure cont…
Tree (B+ Tree)

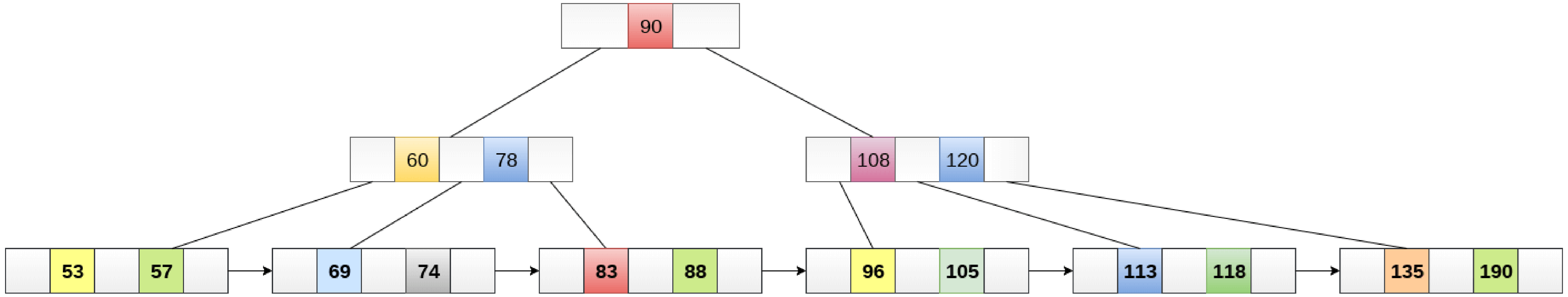
- B+ Tree is an extension of B Tree which allows efficient insertion, deletion and search operations.
- In B Tree, Keys and records both can be stored in the internal as well as leaf nodes.
- Whereas, in B+ tree, records (data) can only be stored on the leaf nodes while internal nodes can only store the key values.

Data Structure cont…
Tree (B+ Tree)

- The leaf nodes of a B+ tree are linked together in the form of a singly linked lists to make the search queries more efficient.
- B+ Tree are used to store the large amount of data which can not be stored in the main memory.
- Due to the fact that, size of main memory is always limited, the internal nodes (keys to access records) of the B+ tree are stored in the main memory whereas, leaf nodes are stored in the secondary memory.

Data Structure cont…
Tree (B+ Tree)

- The internal nodes of B+ tree are often called index nodes.
- A B+ tree of order 3 is shown in the following figure.



Data Structure cont…

Tree (B+ Tree – Advantages of B + Tree)

1. Records can be fetched in equal number of disk accesses.
2. Height of the tree remains balanced and less as compare to B tree.
3. We can access the data stored in a B+ tree sequentially/ directly.
4. Keys are used for indexing.
5. Faster search queries as the data is stored only on the leaf nodes.

Data Structure cont…

Tree (B+ Tree – B-Tree vs. B + Tree)

Sr.	B Tree	B+ Tree
1	Search keys can not be repeatedly stored.	Redundant search keys can be present.

Data Structure cont…

Tree (B+ Tree – B-Tree vs. B + Tree)

Sr.	B Tree	B+ Tree
1	Search keys can not be repeatedly stored.	Redundant search keys can be present.
2	Data can be stored in leaf nodes as well as internal nodes	Data can only be stored on the leaf nodes.

Data Structure cont…

Tree (B+ Tree – B-Tree vs. B + Tree)

Sr.	B Tree	B+ Tree
1	Search keys can not be repeatedly stored.	Redundant search keys can be present.
2	Data can be stored in leaf nodes as well as internal nodes	Data can only be stored on the leaf nodes.
3	Searching for some data is a slower process since data can be found on internal nodes as well as on the leaf nodes.	Searching is comparatively faster as data can only be found on the leaf nodes.

Data Structure cont…

Tree (B+ Tree – B-Tree vs. B + Tree)

Sr.	B Tree	B+ Tree
1	Search keys can not be repeatedly stored.	Redundant search keys can be present.
2	Data can be stored in leaf nodes as well as internal nodes	Data can only be stored on the leaf nodes.
3	Searching for some data is a slower process since data can be found on internal nodes as well as on the leaf nodes.	Searching is comparatively faster as data can only be found on the leaf nodes.
4	Leaf nodes can not be linked together.	Leaf nodes are linked together to make the search operations more efficient.

ArfanShahzadTech

WhatsApp-Contact Us
0345-5922495

Data Structure cont…

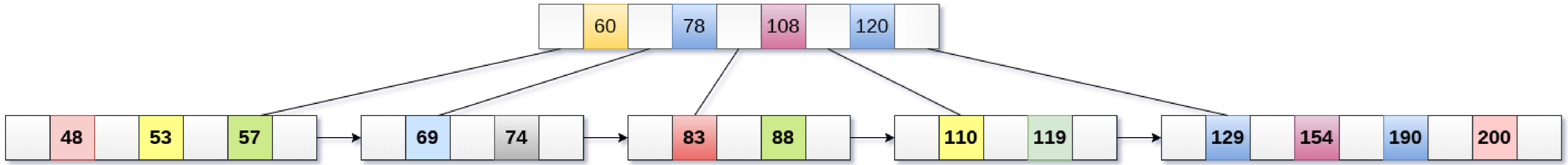
Tree (B+ Tree – Insertion in B + Tree)

- The process of insertion in B+ tree is given below:
- **Step 1:** Insert the new node as a leaf node
- **Step 2:** If the leaf doesn't have required space, split the node and copy the middle node to the next index node.
- **Step 3:** If the index node doesn't have required space, split the node and copy the middle element to the next index page.

Data Structure cont…

Tree (B+ Tree – Insertion in B + Tree)

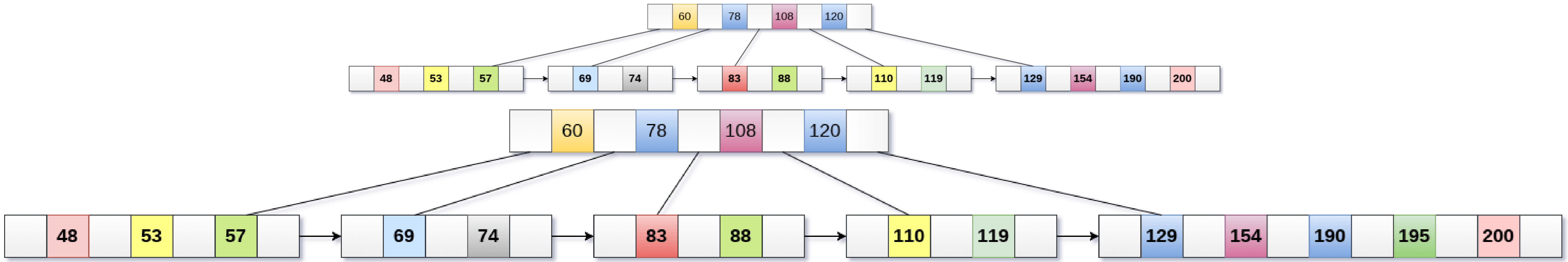
- Insert the value 195 into the B+ tree of order 5 shown in the following figure.



- 195 will be inserted in the right sub-tree of 120 after 190.
- Insert it at the desired position.

Data Structure cont…

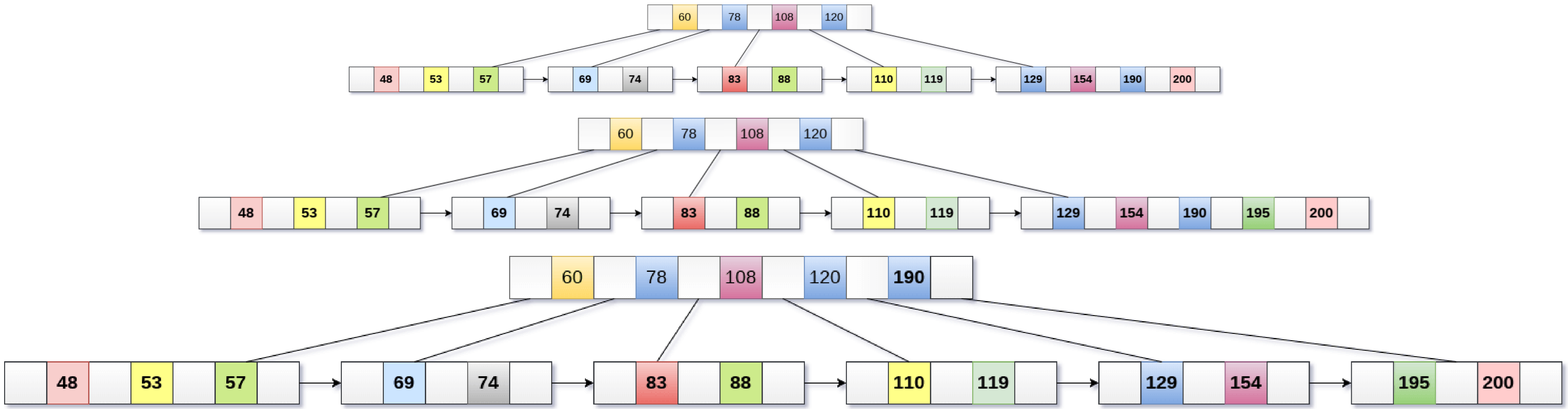
Tree (B+ Tree – Insertion in B + Tree)



- The node contains greater than the maximum number of elements i.e. 4, therefore split it and place the median node up to the parent.

Data Structure cont…

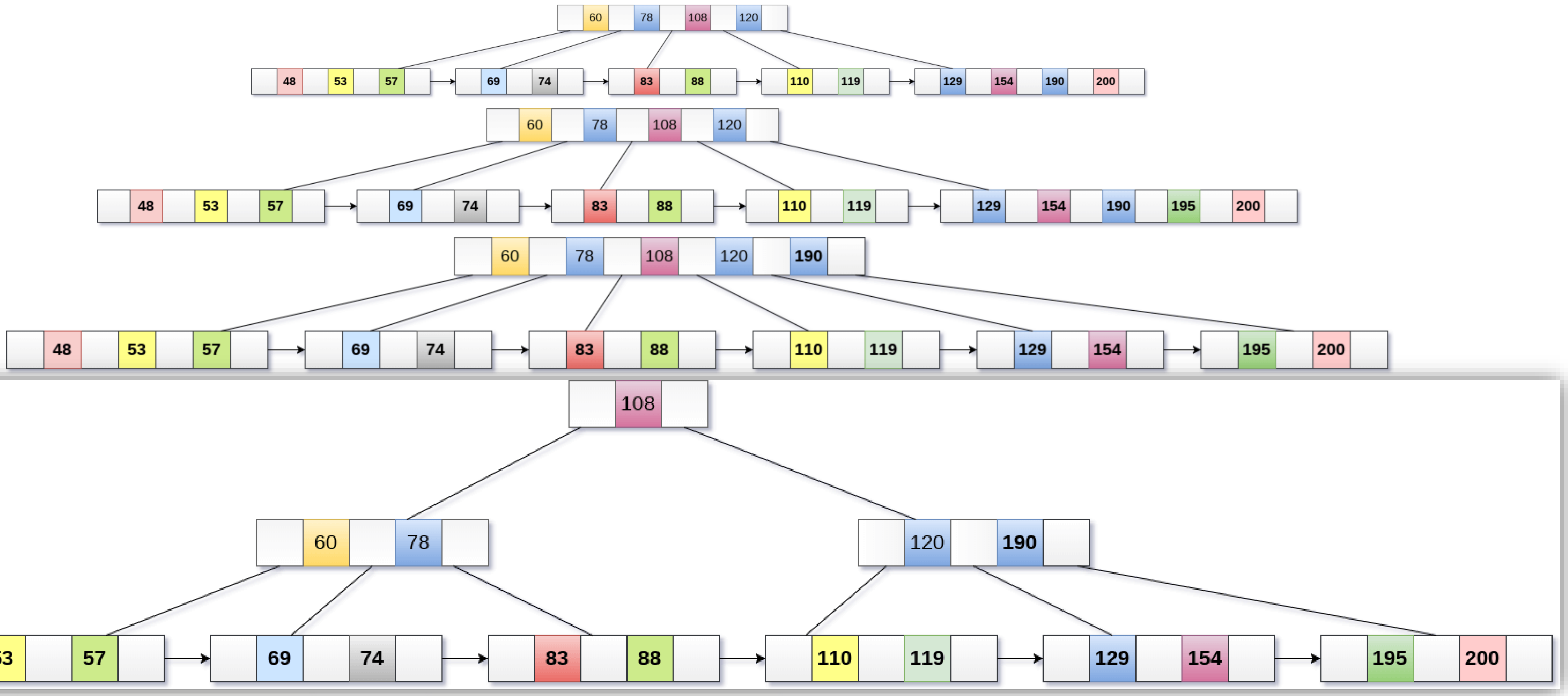
Tree (B+ Tree – Insertion in B + Tree)



- Now, the index node contains 6 children and 5 keys which violates the B+ tree properties, therefore we need to split it, shown as follows.

Data Structure cont…

Tree (B+ Tree – Insertion in B + Tree)



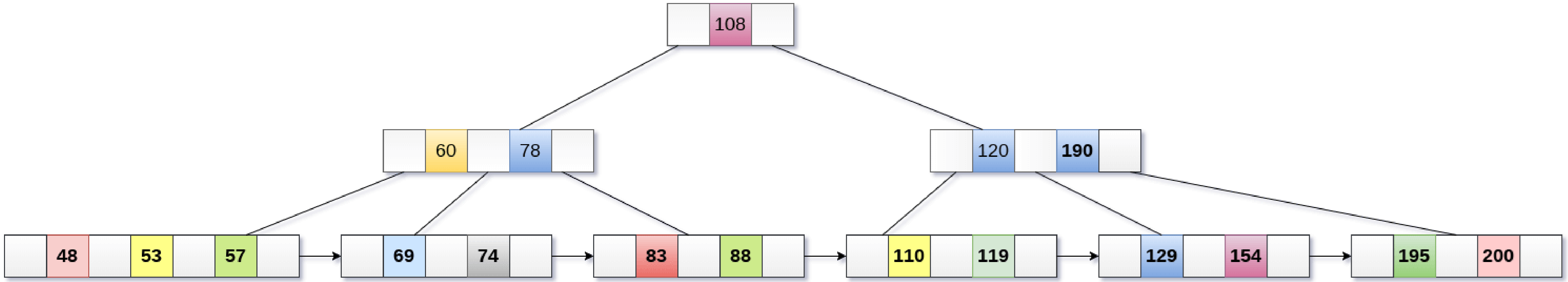
Data Structure cont…
Tree (B+ Tree – Deletion in B + Tree)

- The process of deletion in B+ tree is given below:
- **Step 1:** Delete the key and data from the leaves.
- **Step 2:** if the leaf node contains less than minimum number of elements, merge down the node with its sibling and delete the key in between them.
- **Step 3:** if the index node contains less than minimum number of elements, merge the node with the sibling and move down the key in between them.

Data Structure cont…

Tree (B+ Tree – Deletion in B + Tree)

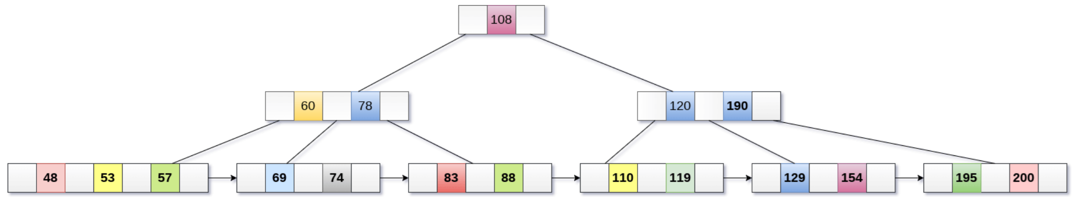
- Delete the key 200 from the B+ Tree shown in the following figure.

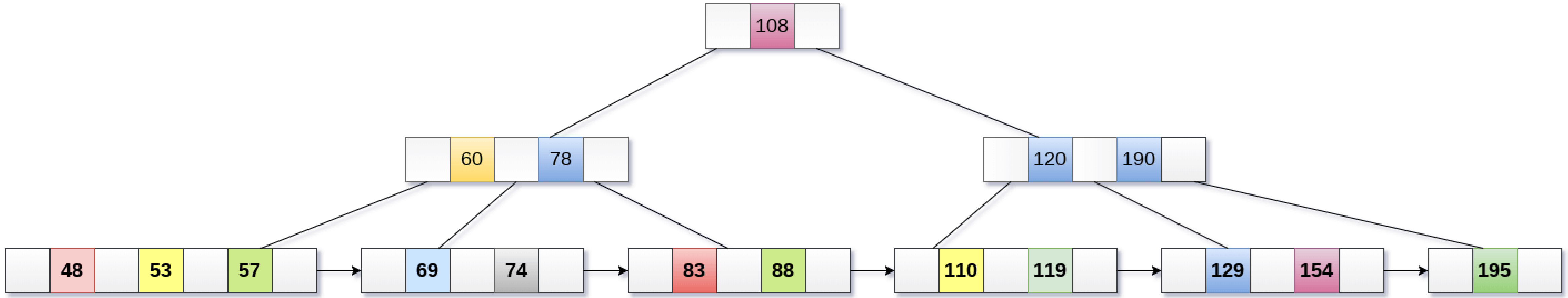


- 200 is present in the right sub-tree of 190, after 195. delete it.

Data Structure cont…

Tree (B+ Tree – Deletion in B + Tree)





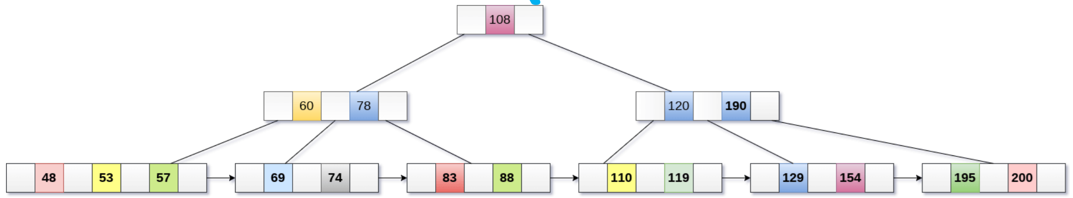
- Merge the two nodes by using 195, 190, 154 and 129.

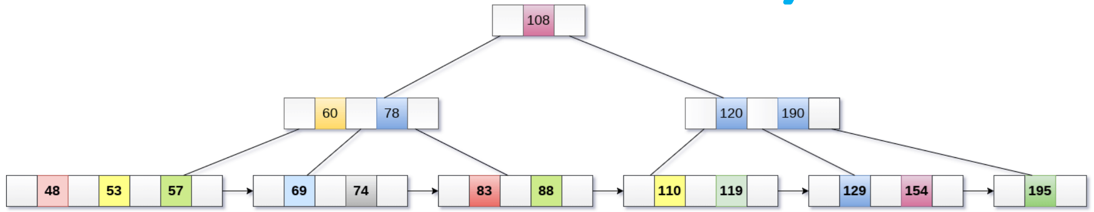
ArfanShahzadTech

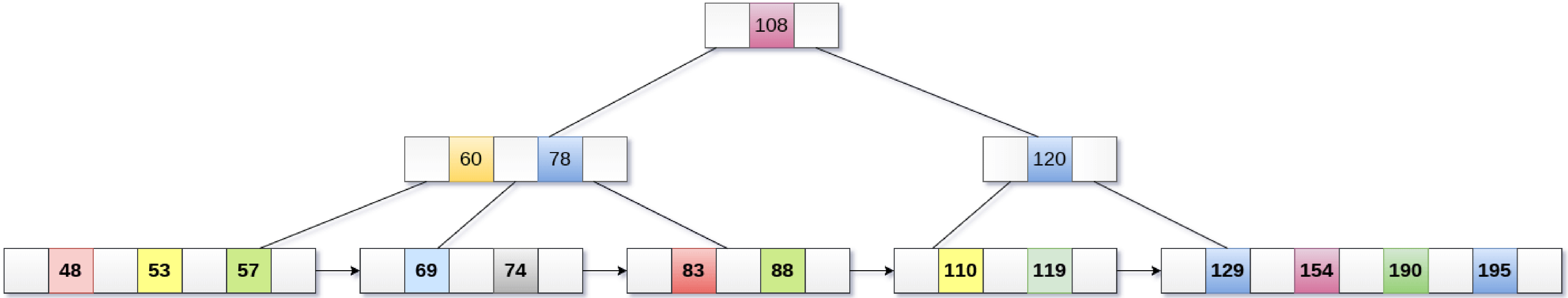
WhatsApp-Contact Us
0345-5922495

Data Structure cont…

Tree (B+ Tree – Deletion in B + Tree)







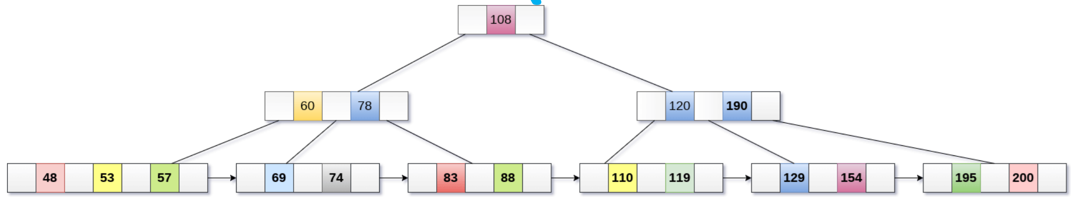
- Now, element 120 is the single element present in the node which is violating the B+ Tree properties.
- Therefore, we need to merge it by using 60, 78, 108 and 120.

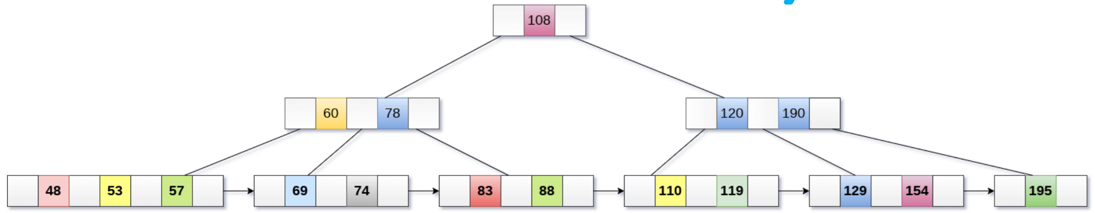
ArfanShahzadTech

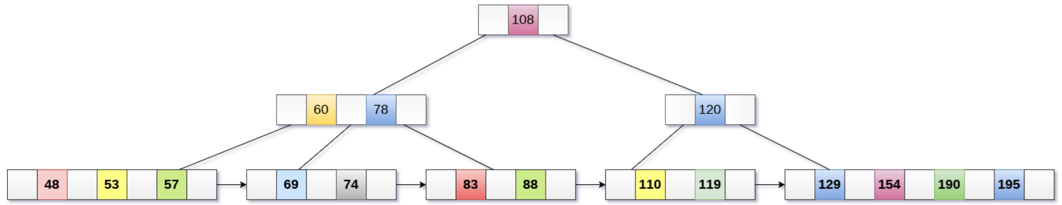
WhatsApp-Contact Us
0345-5922495

Data Structure cont…

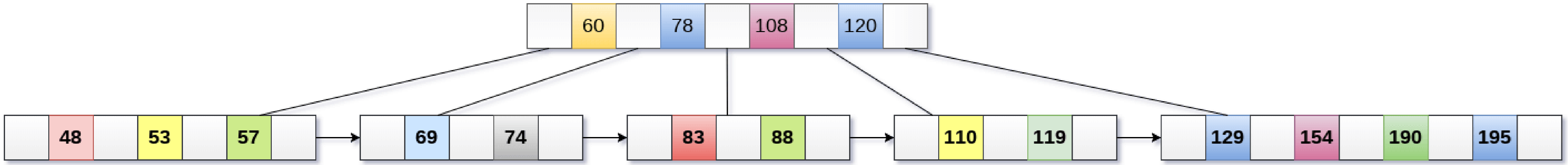
Tree (B+ Tree – Deletion in B + Tree)







- Now, the height of B+ tree will be decreased by 1.



ArfanShahzadTech

WhatsApp-Contact Us
0345-5922495