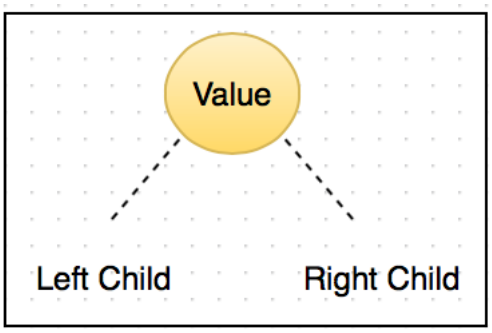
Nevetha Balasubramanian (800899379)
Raghunath Viswanath Nallapeta (800909795)
Sriharish Ranganathan (800901667)

ITCS 6114 Algorithms and Data Structures Fall 2015 Programming Project 3: Design of Demonstration Algorithms

Binary Search Tree

A binary search tree (BST) is a node-based data structure in which each node has no more than two child nodes. Each child must either be a leaf node or the root of another binary search tree. The left sub-tree contains only nodes with keys less than the parent node; the right sub-tree contains only nodes with keys greater than the parent node.

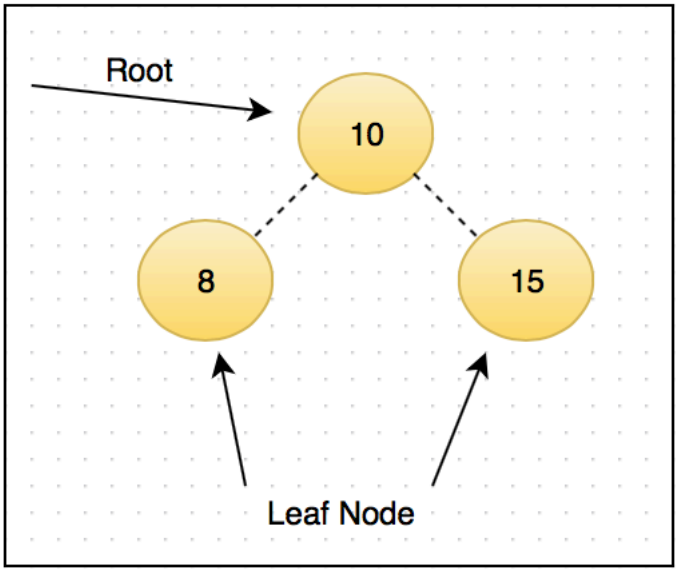
Each node in BST has holds a value and has links for left child and right child as shown below.



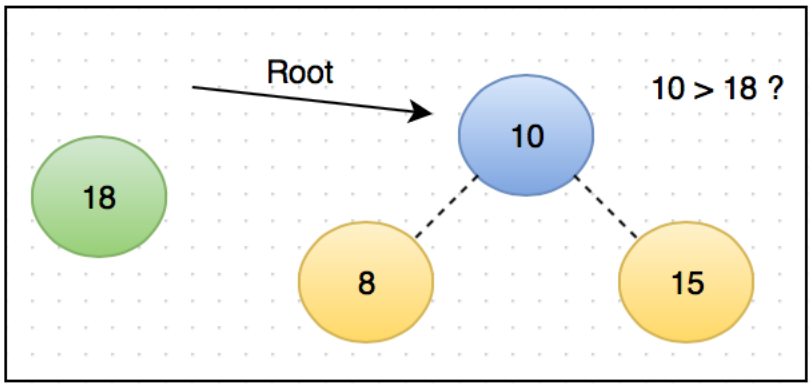
Different operations can be done on the Binary Search tree. They are : insertion, deletion and traversal through the tree. Lets look each operation in detail.

Insertion algorithm in detail

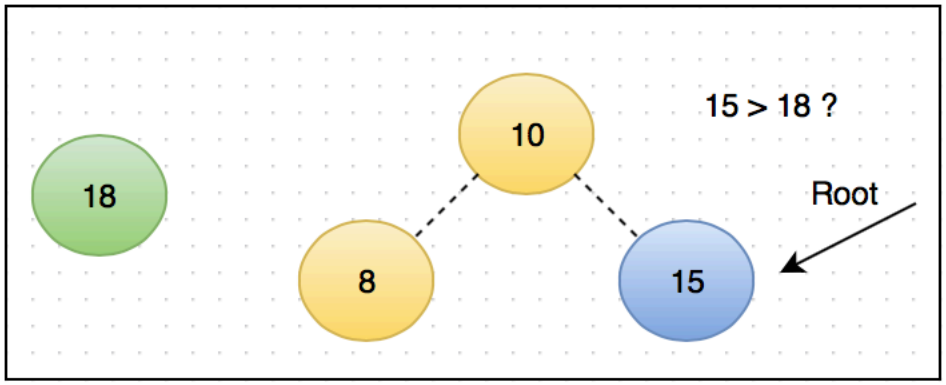
Consider below BST to which a node having value 18 has to be added. Root is a pointer pointing to the root node of the tree which is useful in traversing through the nodes of the tree.



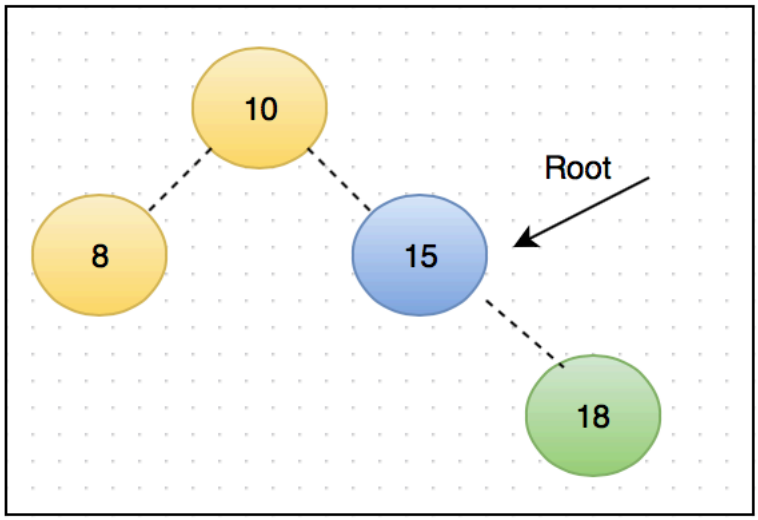
Step 1: First the node value which is being inserted (in our case 18) is compared with the root node value. If the root node value is greater then left child is considered for next comparison or if lesser then right child of root is considered. In this case root value 10 is compared with node value 18 which has to be inserted.



Step 2: Root node value(10) is less than 18 so in the next step root pointer is moved to right child of root that is root = root->right. Now the root is pointed to node 15 as shown in below diagram.



Step 3: Current node value that is 15 is less than 18 which has to be inserted, so now the pointer is moved to right of node 15 but there no child exists. And at this place new node 18 is added as right child of node 15. Below is the tree structure after insertion. At any case the right child values should always be greater than its parent and left child node values should be less than its parent.

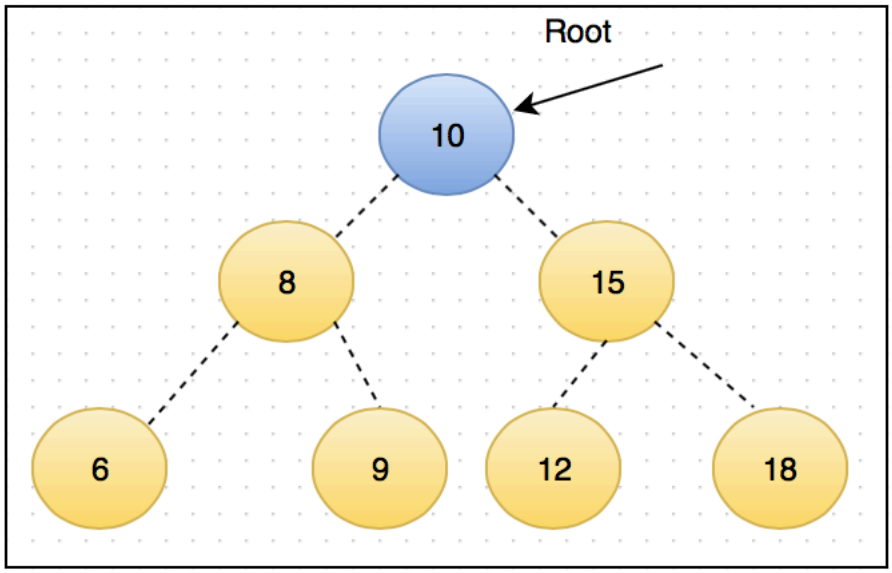


Deletion algorithm in detail

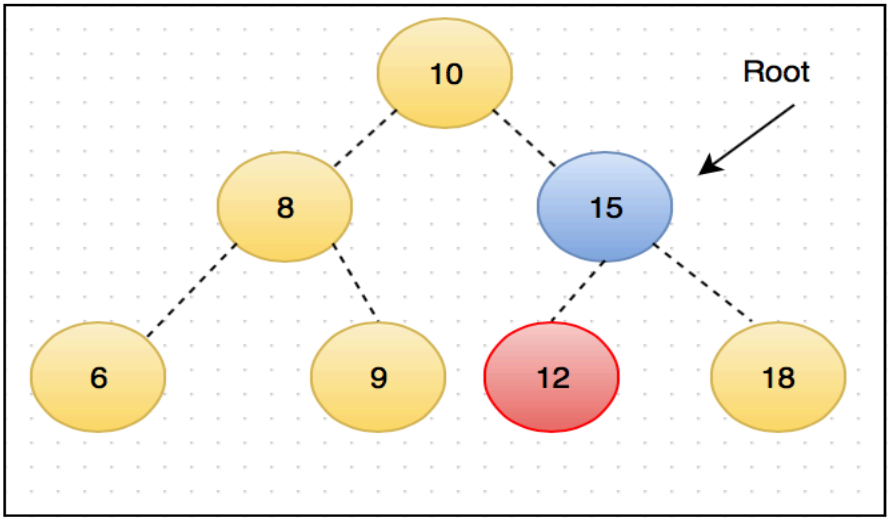
There are 3 cases which has to be dealt while deleting a node. Below are the cases explained

Case 1: Deleting a child node

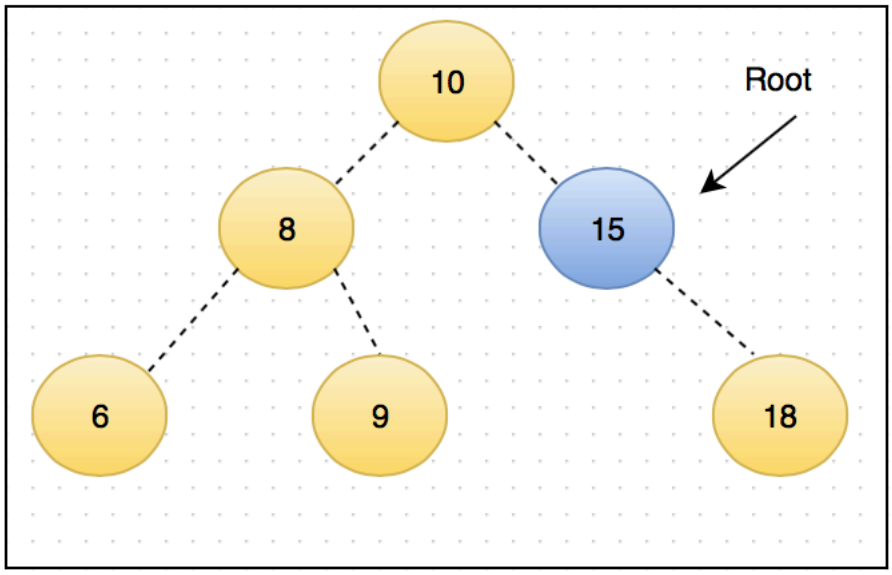
Step 1: Consider below BST and node 12 has to be deleted. 12 is greater than root node value so the root pointer has to be moved to right that is root = root->right.



step 2: Now the pointer is on node 15 and node to be deleted 12 is less than 15 so move to left of node 15 that is root = root-> left. The node 12 does not have any child so the node 12 is deleted.

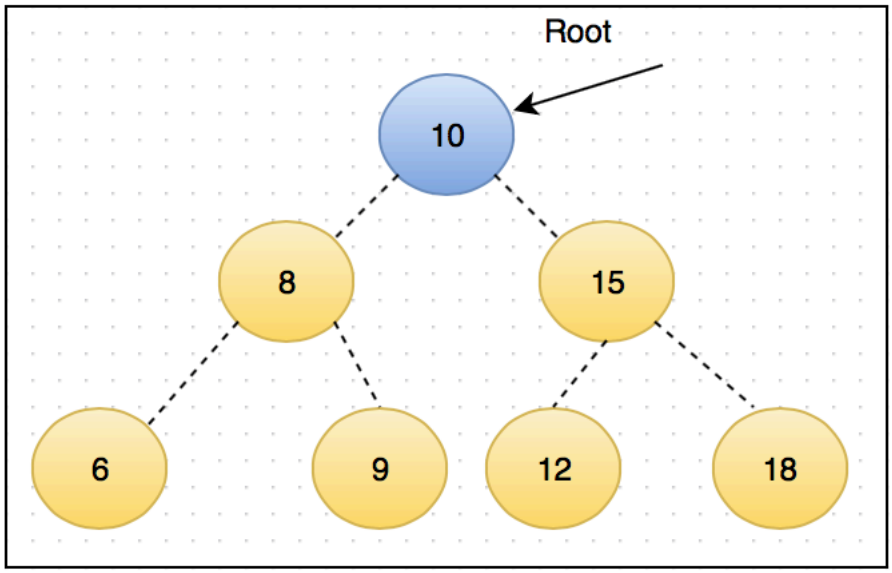


step 3: After deletion the tree looks like below.

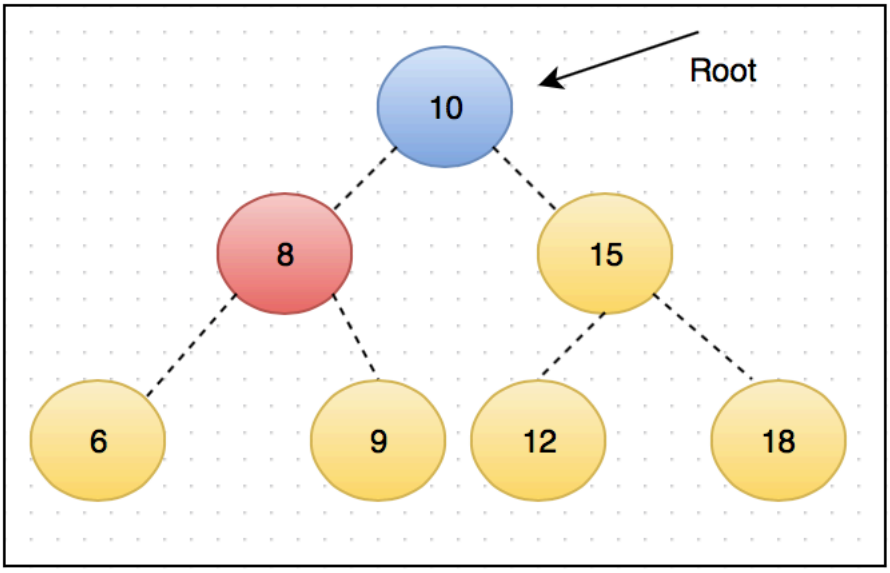


Case 2: Deletion of node which is having child nodes.

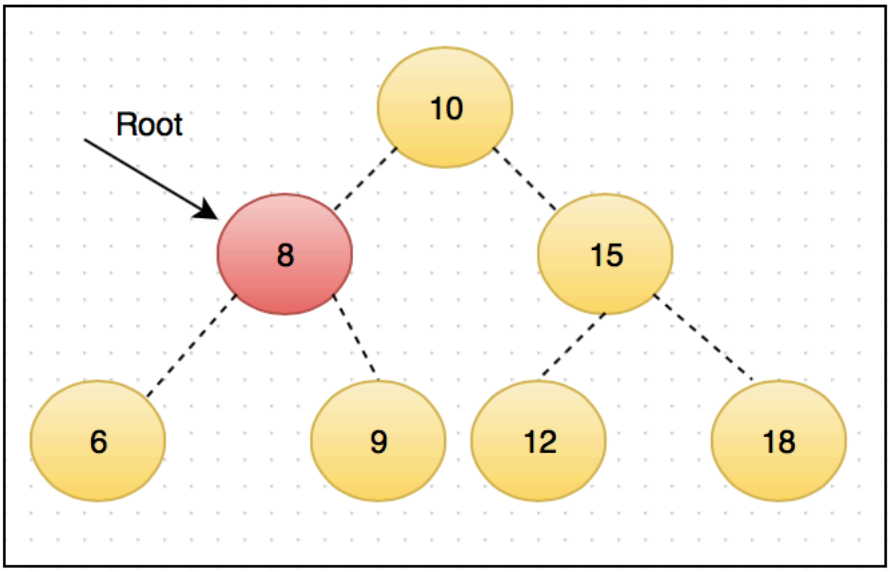
Step 1: Consider below BST and node 8 has to be deleted. As shown in figure below it has nodes 6 and 9 as children.



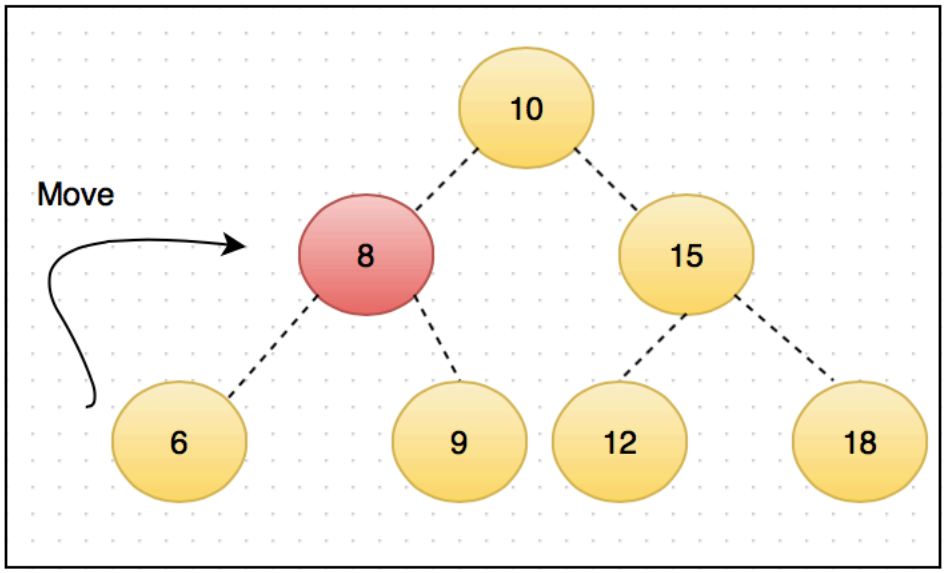
step 2: Root node value is greater than the node which has to be deleted, so move the pointer to its left child.



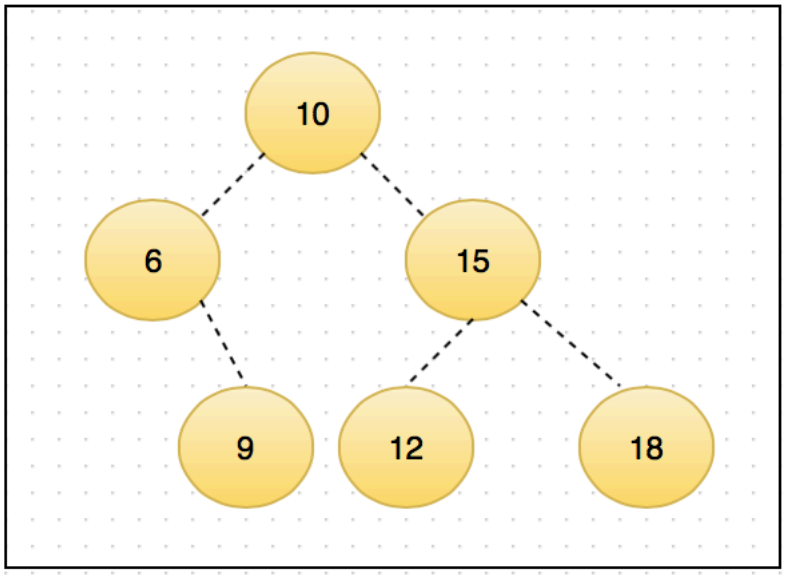
step 3: Now the pointer is pointed to node 8 which has to be deleted. The node 8 position has to be replaced with the left most child of the sub tree in which 8 acts as parent.



step 4: Node 8 has left child 6 and it further doesn't have any Childs, so the node 6 has to be moved to place of node 8 because at anytime the BST order has to be followed that is right child greater than parent and left lesser than parent.

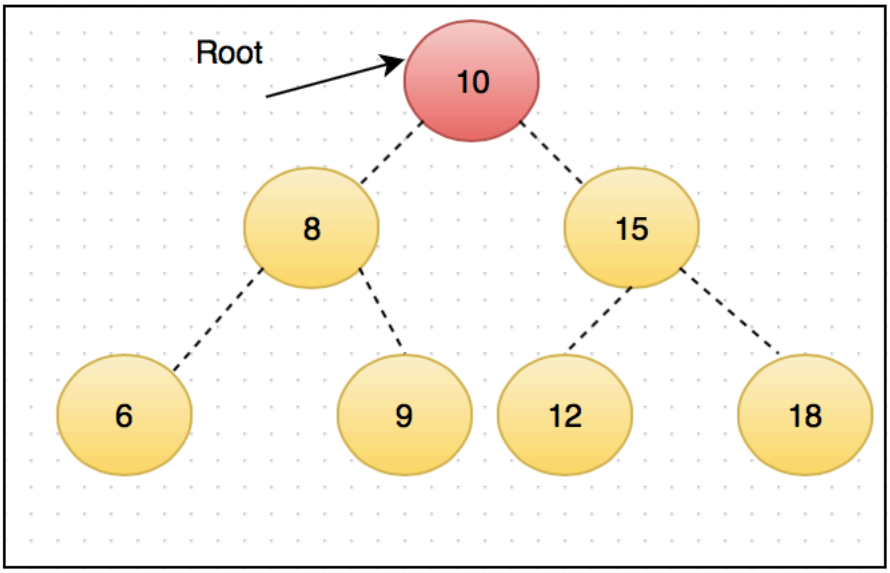


step 5: After replacing the nodes the BST looks like below.

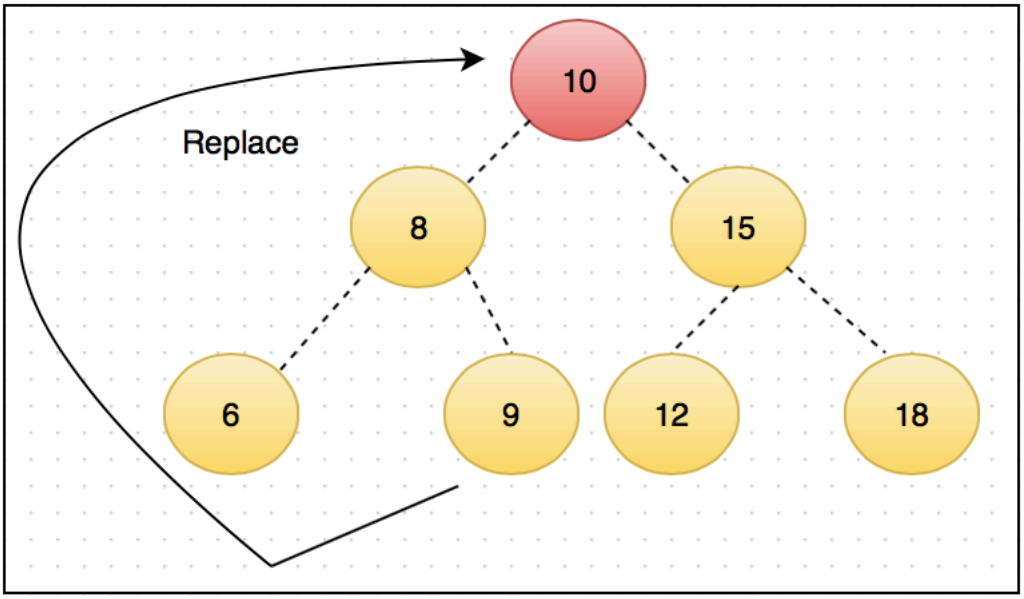


Case 3: Deletion of root node

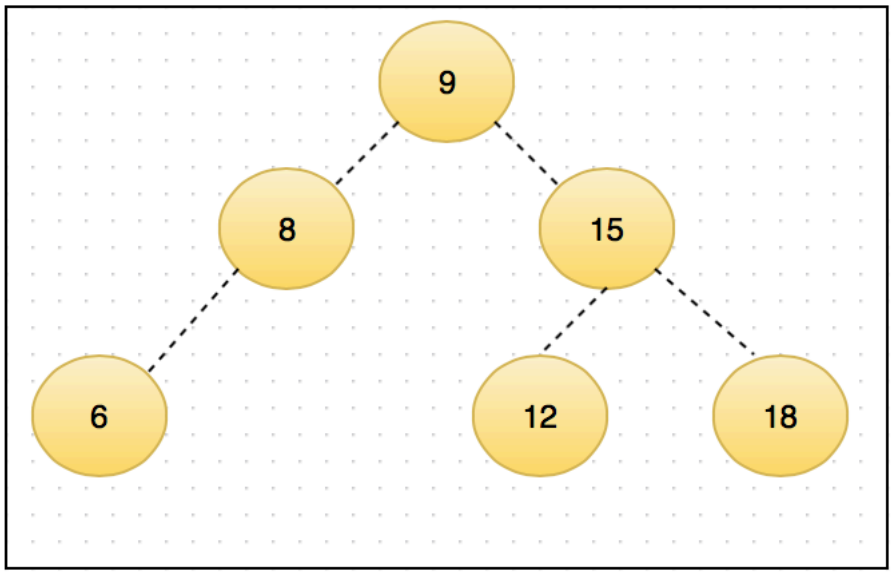
Step 1: Root node is the parent node of the tree. In the below example root node is 10. It has to be replace with the one of the nodes of its left sub tree which satisfies BST condition.



Step 2: Left child of root node is 8 and its right child is 9 which is a leaf node. This has to be placed in the position of the node 10.



Step 3: After replacing the tree looks like below.



How does your demonstration algorithm improve the understanding of the algorithm?

Our demonstration algorithm helps to understand how actually a tree looks when a node is inserted or deleted. At each step after any operation tree structure is displayed which helps one to understand the modifications applied to a tree.

Below screen shots are the program output. The same example which is considered which was previously used for insertion and deletion of nodes.

```
---Binary Search Tree---
1. Insert
 2. Delete
 3. Find 4. Display Tree
 5. Exit

1
Enter value to insert into tree
10
Displaying the Tree
****..................................................****
                              10
****..................................................****
1. Insert
 2. Delete
 3. Find
 4. Display Tree
 5. Exit

1
Enter value to insert into tree
8
Displaying the Tree
****..................................................****
                              10
               8                              --
****..................................................****
1. Insert
 2. Delete
 3. Find
 4. Display Tree
 5. Exit

1
Enter value to insert into tree
15
Displaying the Tree
****..................................................****
                              10
               8                              15
****..................................................****
```

```
1. Insert
 2. Delete
 3. Find
 4. Display Tree
 5. Exit

1
Enter value to insert into tree
6
Displaying the Tree
****.....................................................****
                              10
                8                             15
        6               --              --              --
****.....................................................****
1. Insert
 2. Delete
 3. Find
 4. Display Tree
 5. Exit

1
Enter value to insert into tree
9
Displaying the Tree
****.....................................................****
                              10
                8                             15
        6               9               --              --
****.....................................................****
1. Insert
 2. Delete
 3. Find
 4. Display Tree
 5. Exit

1
Enter value to insert into tree
12
Displaying the Tree
****.....................................................****
                              10
                8                             15
        6               9               12              --
****.....................................................****
```

```
1. Insert
 2. Delete
 3. Find
 4. Display Tree
 5. Exit

1
Enter value to insert into tree
18
Displaying the Tree
****.....................................................****
                              10
               8                              15
        6              9              12              18
****.....................................................****
```

From this we can easily come to know which is parent of which and the leaf nodes.

Algorithm Analysis:

The time complexity of binary search tree for insertion, deletion or searching a node always have a average runtime of O(log n) because by comparing with the parent nodes either right or left traversal is decided and half of the nodes are eliminated by this. Worst case of all these operations is O(n) and this happens when the tree is unbalanced. An "unbalanced" tree is one which is lopsided, meaning that some of the leaf nodes have very high depths and some leaf nodes have very low depths. The most extreme case of an "unbalanced" tree is a linked list, with the root being one end of the linked list and the only leaf node being the other end.

Example with output of program:

```
****.....................................................****
1. Insert
 2. Delete
 3. Find
 4. Display Tree
 5. Exit

4
****.....................................................****
                              10
                8                             15
        6               9               12              18
****.....................................................****
1. Insert
 2. Delete
 3. Find
 4. Display Tree
 5. Exit

3
Enter value to find in tree
9
Found: [9]

1. Insert
 2. Delete
 3. Find
 4. Display Tree
 5. Exit

3
Enter value to find in tree
99
Could not find the Nodes
1. Insert
 2. Delete
 3. Find
 4. Display Tree
 5. Exit

1
Enter value to insert into tree
66
Displaying the Tree
****.....................................................****
                              10
                8                             15
        6               9               12              18
    --      --      --      --      --      --      --      66
****.....................................................****
```

```
1. Insert
 2. Delete
 3. Find
 4. Display Tree
 5. Exit

2
Enter value to delete from tree
66
Deleted 66
****.....................................................****
                              10
               8                              15
        6              9               12              18
****.....................................................****
```

External Reference:

http://algs4.cs.princeton.edu/32bst/

http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/binarySearchTree.htm

https://www.cs.usfca.edu/~galles/visualization/BST.html