Binary Search Tree

Online Tutorial of Academic Support Club at IUT

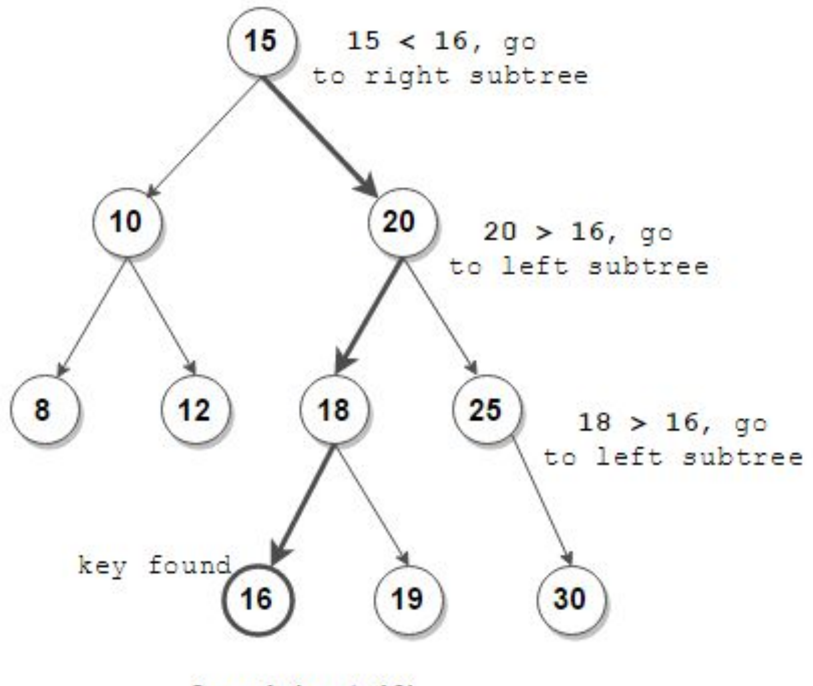
author: Shohrukh Yakhyoev

Main operations

- Search
- Insertion
- Deletion

Next we will analyze these operations one by one.

Search



15 < 16, go to right subtree
20 > 16, go to left subtree
18 > 16, go to left subtree
key found
15
10
20
8
12
18
25
16
19
30

Search (root, 16)

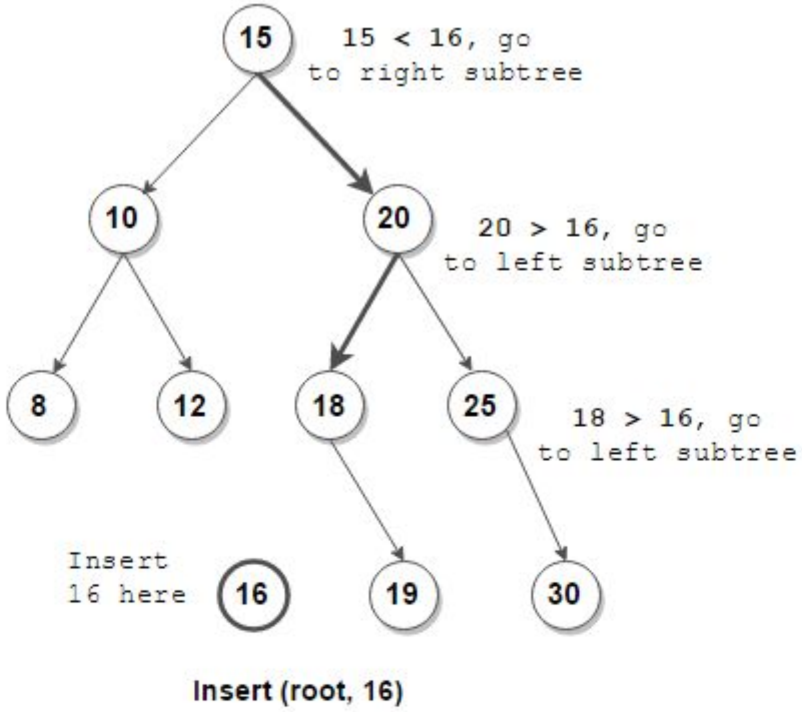
Algorithm is simple:

You will **stop** either when **node is found** or **node is null** (means key isn't found).

If target is bigger than value:
go to the right subtree.

else if target is less than value:
go to the left subtree.

Insertion


15
15 < 16, go to right subtree
10
20
20 > 16, go to left subtree
8
12
18
25
18 > 16, go to left subtree
Insert 16 here
16
19
30

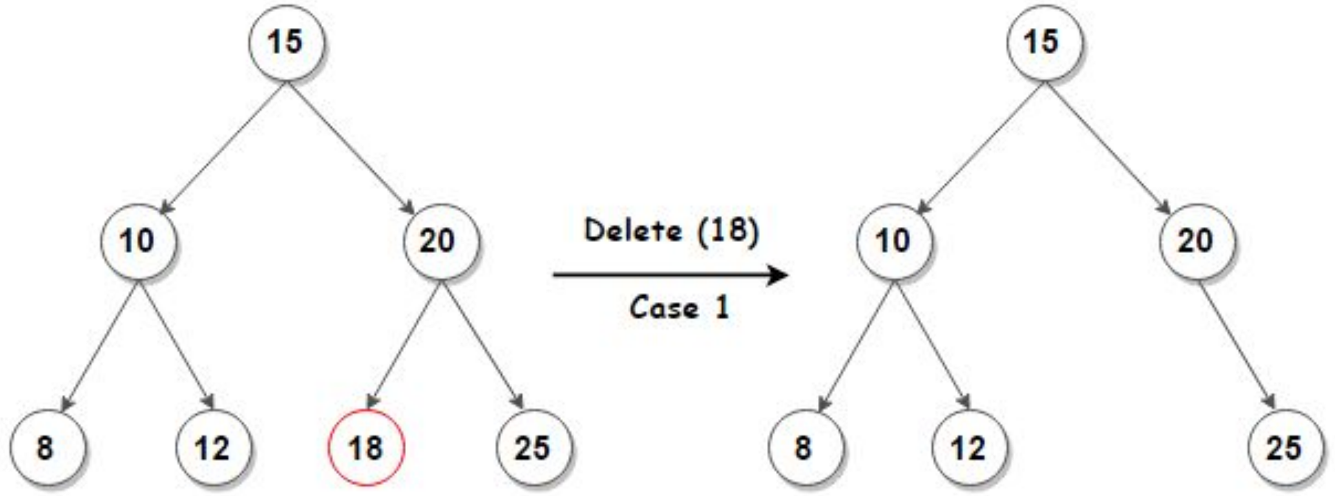
Insert (root, 16)

Algorithm is almost identical as in searching:

Firstly, you search null node using the same logic as in search func().

Once you find null node, insert new value there.

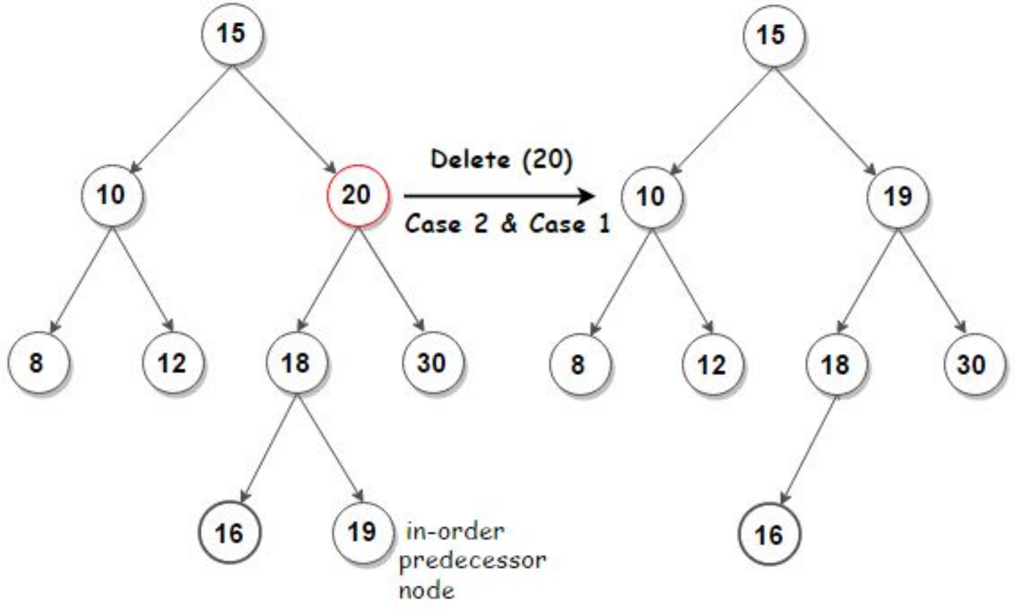
Deletion. Case 1: node with no children



Set parent's left or right child to null.

In this ex: 18 is left child of its parent. So logic to delete 18: **parent→left = null**

Deletion. Case 2: node with two child

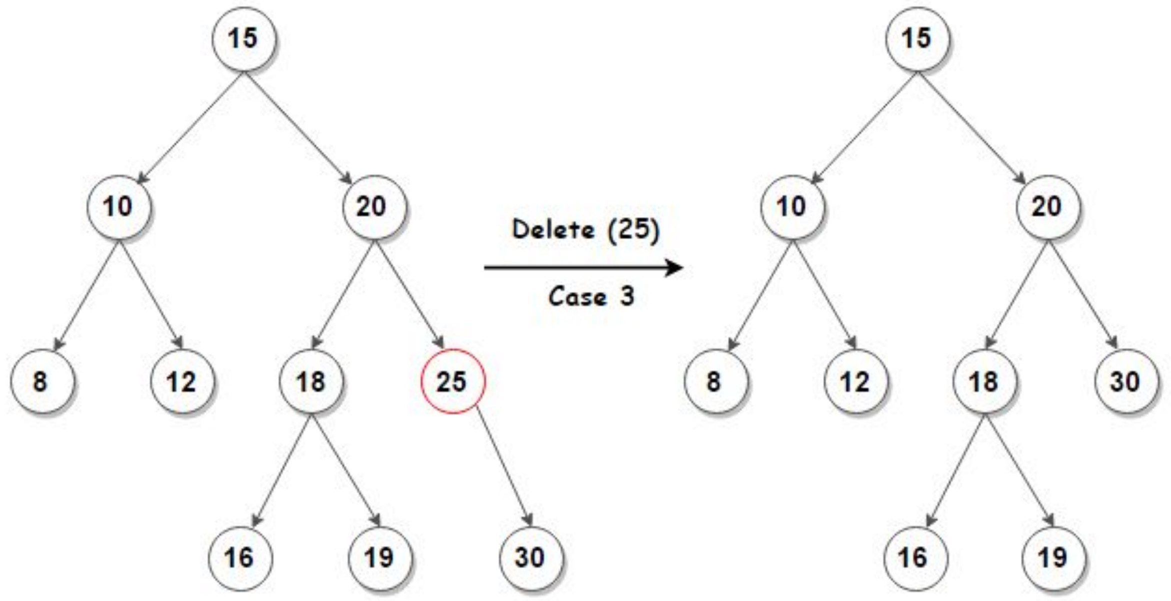


Step 1: Find inorder predecessor of node

Step 2: Delete inorder predecessor

Step 3: Replace value of node with value of in-order predecessor

Deletion. Case 3: node with one child



Set parent's left or right child to node's child.

In this ex: 25 is right child of its parent & has right child. So code: **parent→right = node→right**

Useful Links

- Reading
    - Overview of Trees

- Video
    - Overview of BST

References

- Photos illustrating BST: techiedelight.com