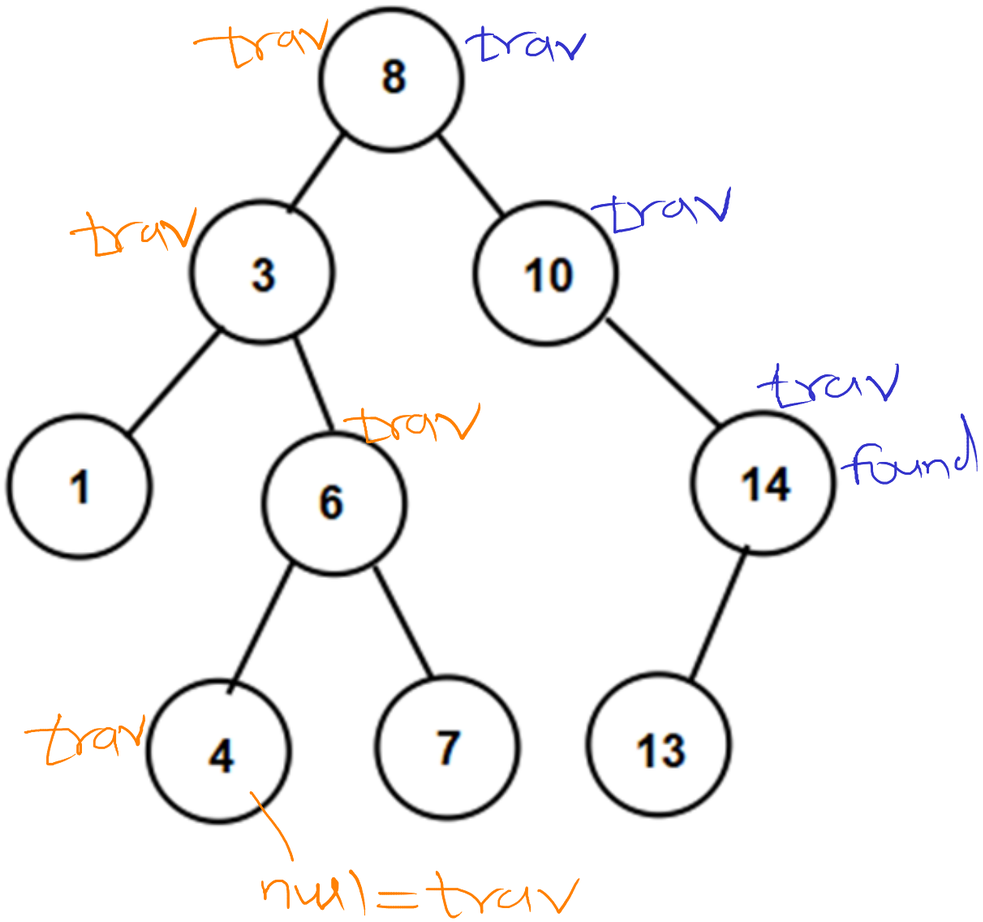
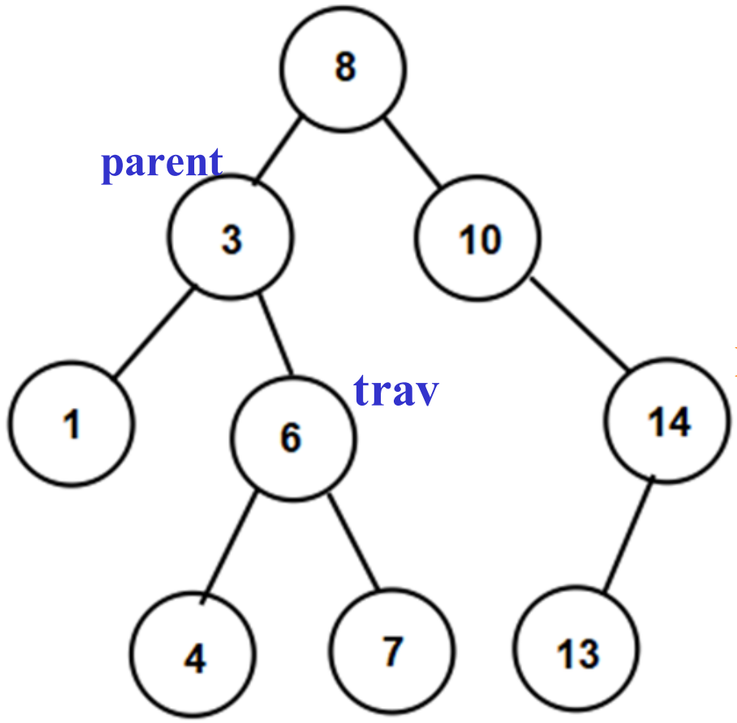
BST - Binary Search

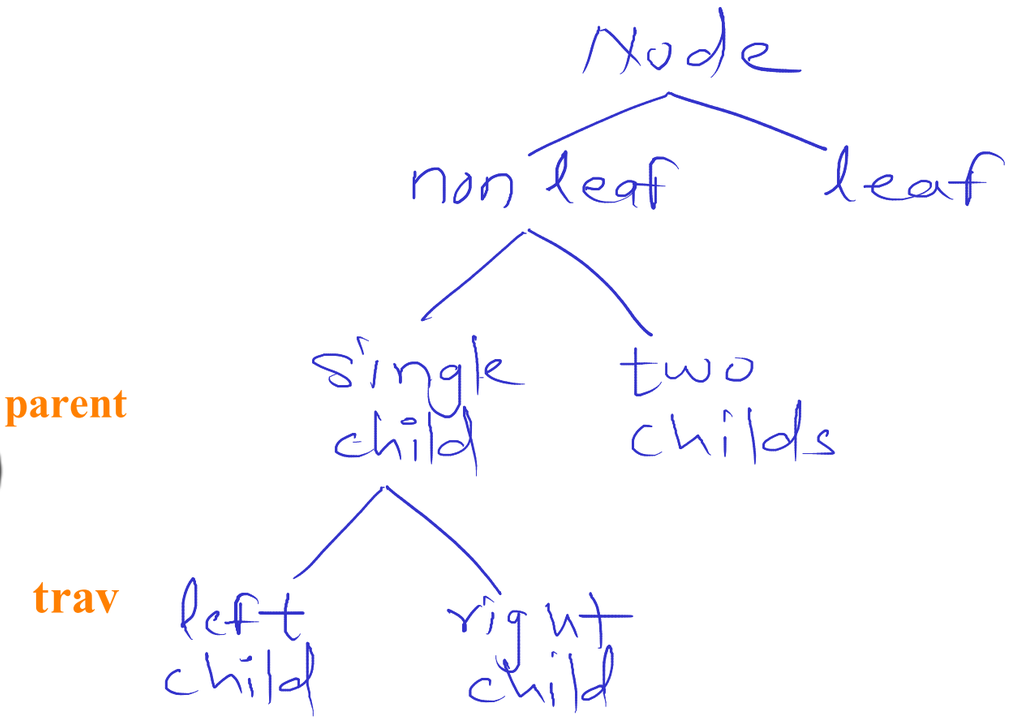


//1. start from root
//2. if key is equal to current data
//return current node
//3. if key is less than current data
// search key into left of current node
//4. if key is greater than current data
// search key into right of current node
//5. repeat step 2 to 4 till leaf nodes

Key = 14
Key = 5

BST - Delete Node

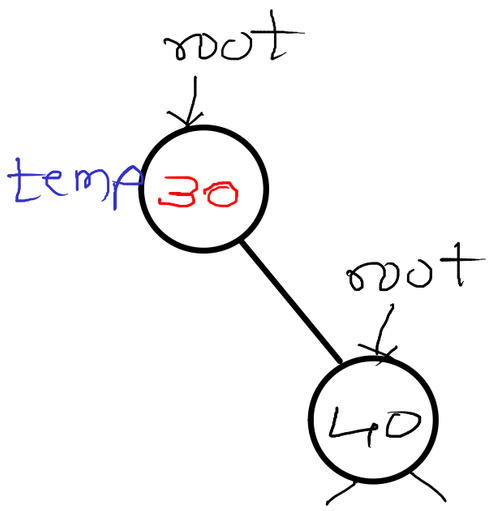


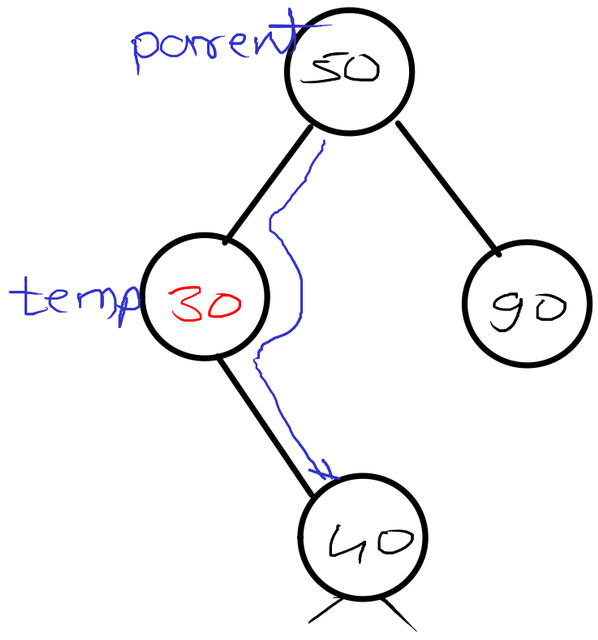


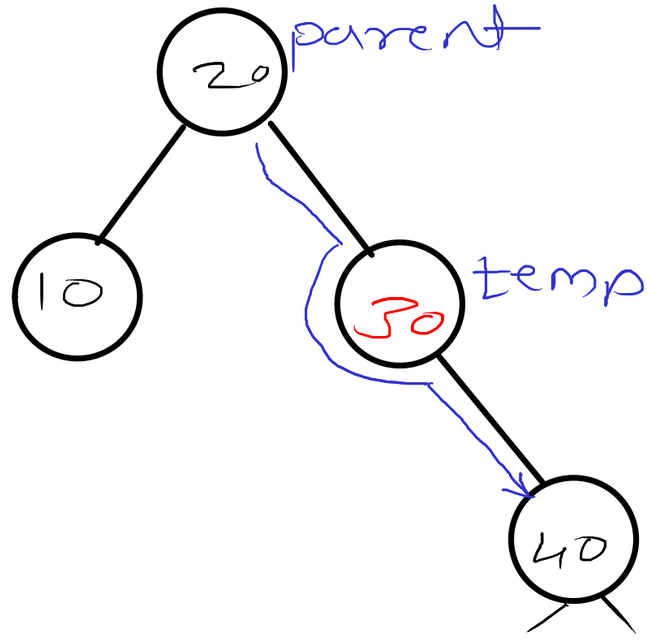
key = 6

left
right
root

BST - Delete node which has single child (right child)

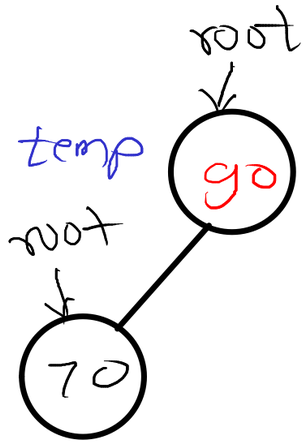


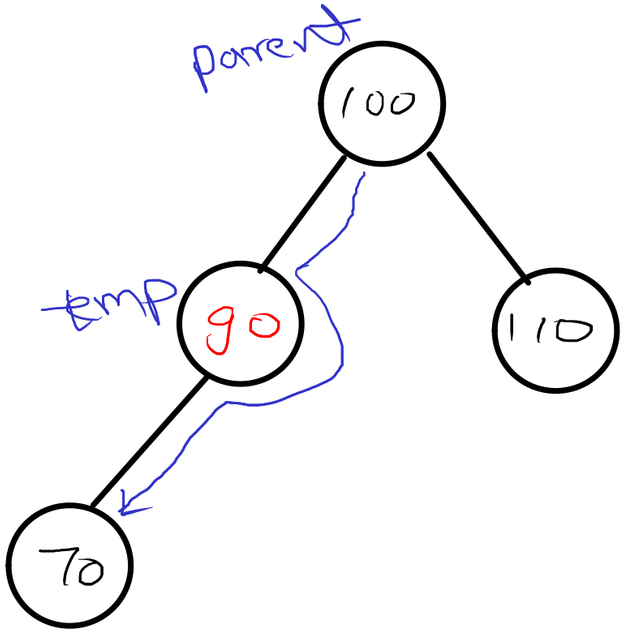


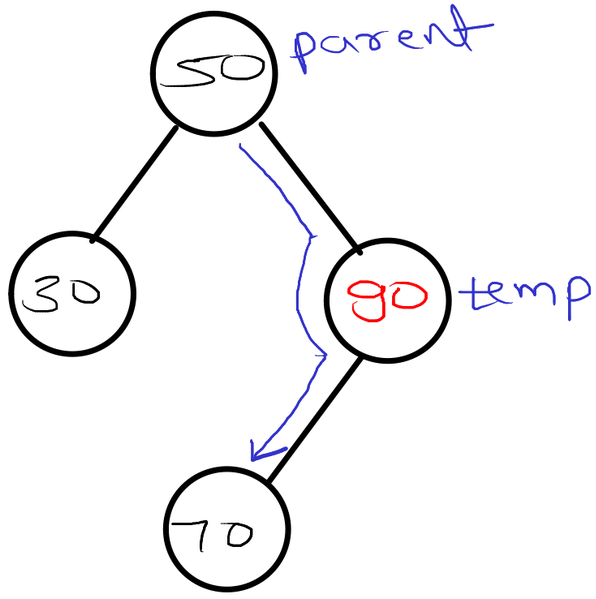



```
if(temp.left == null){
        if(temp == root)
                root = temp.right;
        else if(temp == parent.left)
                parent.left = temp.right;
        else
                parent.right = temp.right;
}
```

BST - Delete node which has single child (left child)

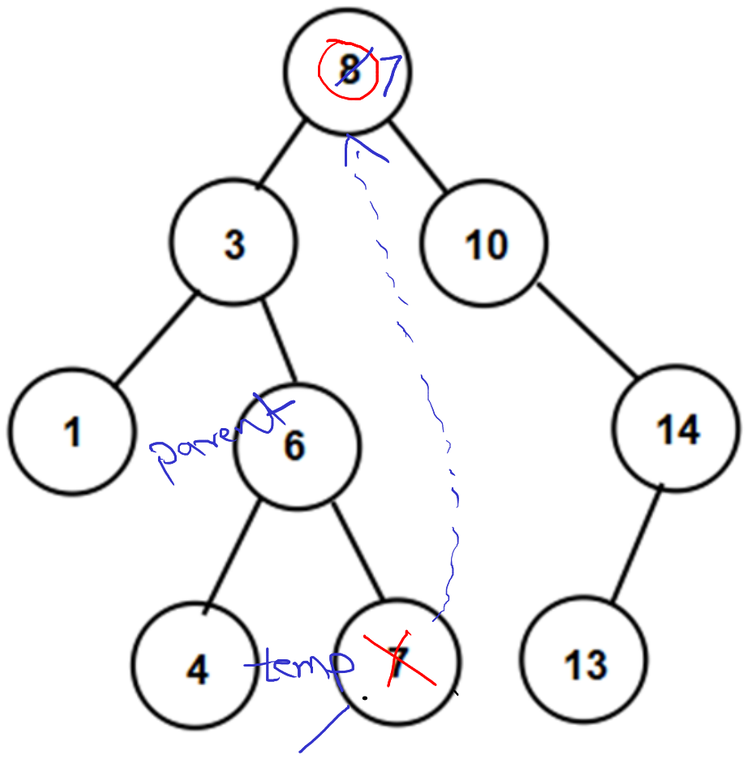







```
if(temp.right == null){
        if(temp == root)
                root = temp.left;
        else if(temp == parent.left)
                parent.left = temp.left;
        else
                parent.right = temp.left;
}
```

BST - Delete node which has two childs




```
if(temp.left != null && temp.right != null){
        trav = temp.left;
        parent = temp;
        while(trav.right != null){
                parent = trav;
                trav = trav.right;
        }
        temp.data = trav.data;
        temp = trav;
}
```

Inorder : 1 3 4 6 **7** 8 **10** 13 14

(left extreme right) — Inorder predecessor

Inorder successor — (right extreme left)