Binary Search Tree

A binary tree with following property (Sometimes a rooted B.T)

"For each node 'n' with value (i.e., data present in the node or value of the node or nodes data) v :

(i) Value property of all the nodes in the left subtree of node 'n' is smaller than that of value property of node n.

(ii) Value property of all the nodes in the right subtree of node 'n' is greater than that of value property of node 'n'

(iii) No duplicates allowed.*

Operations :-

Traversals :- Like Binary trees

(Inorder, Pre-order, Postorder, etc., traversals)

Search :-

Search a key (k).

(i.e., searching for a node with value property equals to k)

In the following algorithm

- BSTree represents root
- BSTree.data is representating the data or value property of the root node.
- BSTree.left (BSTree.right) indicates root of left subtree (root of right subtree) of the BSTree

```
Search (BSTree , k )
    if (BSTree is empty)
        return false ;
    endif
    if (BSTree.data == k)
        return true ;
    endif
    if (BSTree.data > k)
        return Search(BSTree.left , k) ;
    else if (BSTree.data < k )
        return Search (BSTree.right , k);
    endif
endif.
```

Homework :- Write an iterative algorithm for the search operation.

Searching Minimum.

```
Minimum ( BSTree )
        if ( BSTree.left == NULL)
                return ( BSTree.data );
        else
                return  Minimum ( BSTree.left );
        endif.
end Minimum.
```

Time Complexity : (Homework)

(discussed in class)

Searching node with Maximum Value Properly.

```
Maximum ( BSTree )
        if ( BSTree.right == NULL )
                return (BSTree.data)
        else
                return Maximum (BSTree.right)
        endif
end Maximum.
```

Discuss time complexity of above algorithm.

(Discussed in class)

Adding a new node in Binary Search Tree

```
AddNode ( BSTree , v)        // New node's value properly

    if (BSTree == NULL)
            BSTree = CreateTreeNode(v);
            return
    endif
    if ( BSTree.data == v)
            return;
    endif
    if ( BSTree.data > v)
        if(BSTree.left == NULL)
                BSTree.left = CreateTreeNode(v);
        else
                AddNode(BSTree.left, v);
        endif
    else if ( BSTree.data < v)
```

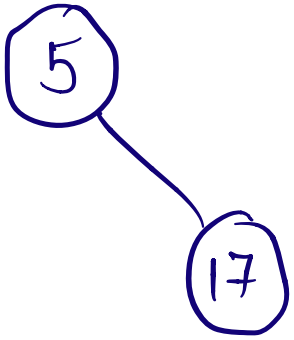
```
            if (BSTree.right == NULL)
                    BSTree.right = CreateTreeNode(v);
            else
                    AddNode(BSTree.right, v);
            endif
    endif
```

Example :- Assuming Empty tree at the start Construct a Binary Search Tree after the Following set of operations in sequence.

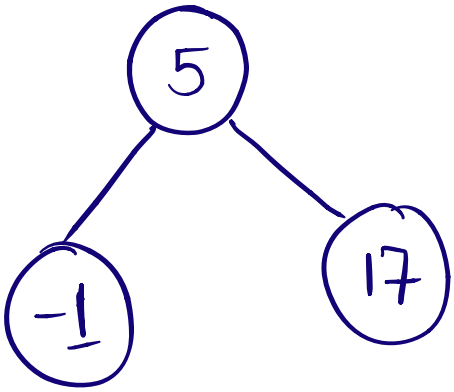
1) bst.AddNode(5);
2) bst.AddNode(17);
3) bst.AddNode(-1);
4) bst.AddNode(3);
5) bst.AddNode(12);
6) bst.AddNode(5);

Status After Step 1.

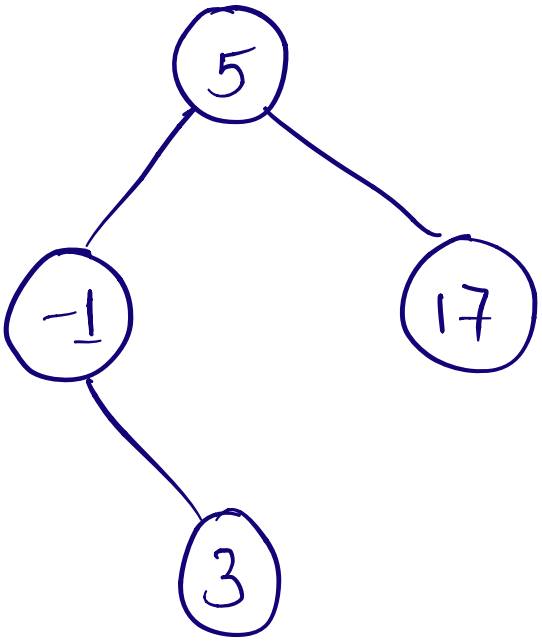
Status After Step 2



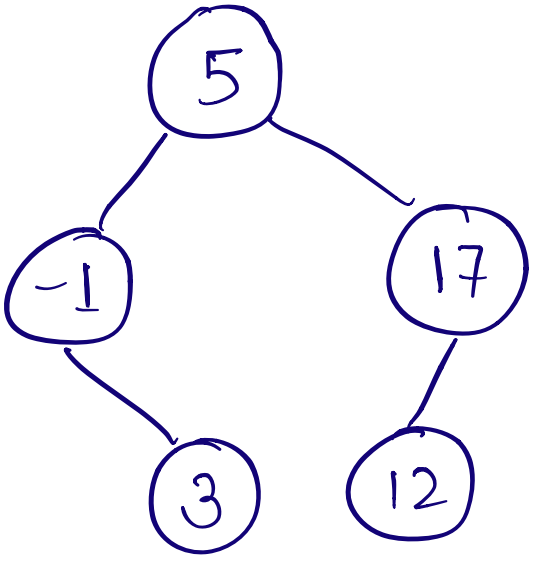
Status After Step 3



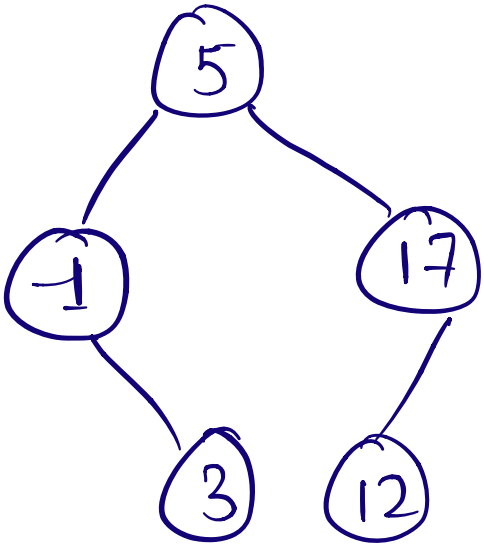
Status After Step 4



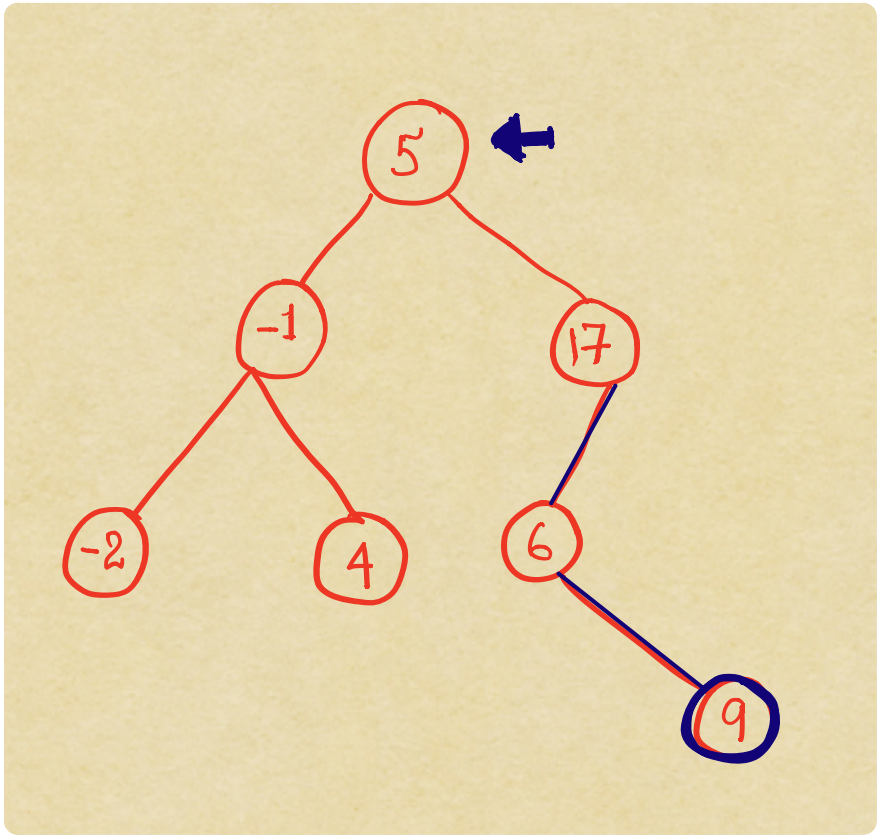
Status After Step 5



Status After Step 6



(Duplicates Not Allowed)



Successor

Succ(x) : successor of node x

: The node y, whose data or value properly (y.data) is the successor of data or value properly in sorted order of this tree.

Inorder : -2, -1, 4, 5, 6, 9, 17

Pred(4) ↙ ↘ Succ(5)

Let x be the node whose successor we are interested in.

Scenario 1:

If x has right subtree.

succ(x) = Minimum(x → right)

(Why ??)

Scenario 2:

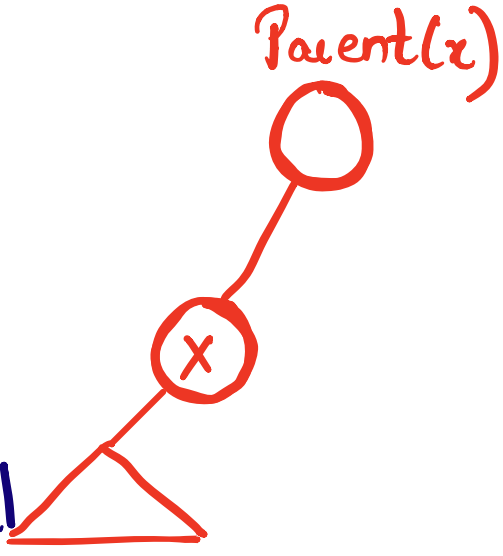
If x has no right subtree. and x is the left child of the parent(x)

then

the succ(x) = parent(x)

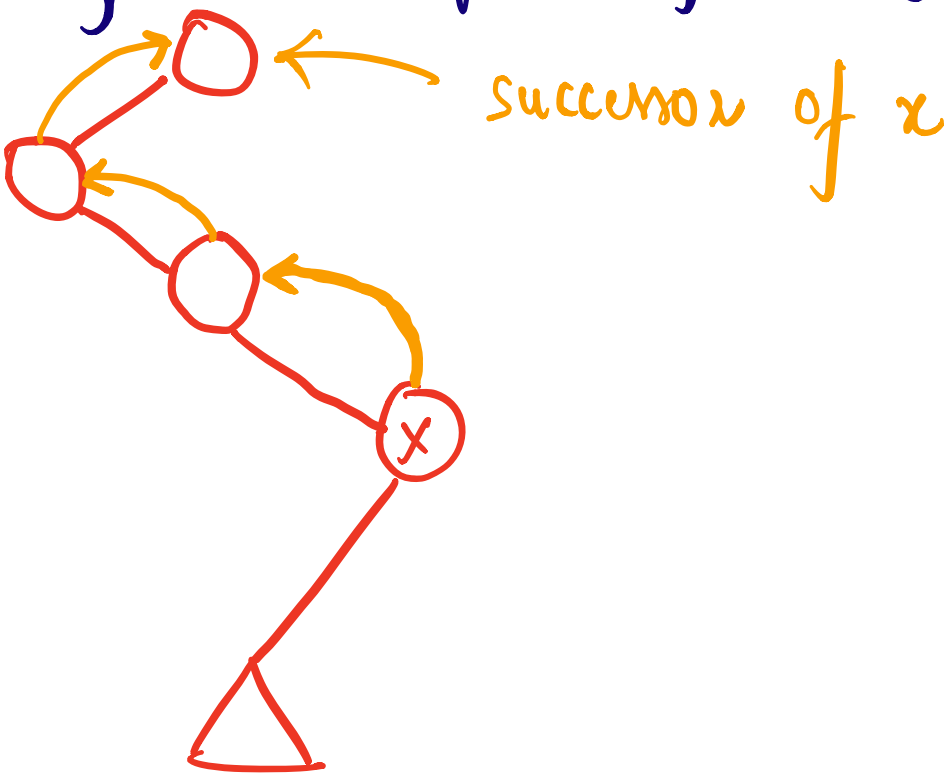
(why ??)

↳ Recall Inordu Traversal



Scenario 3:

If x has no right subtree and x is not the left child of the parent of x.



Keep moving up from x, until you find a parent which branches from the left.

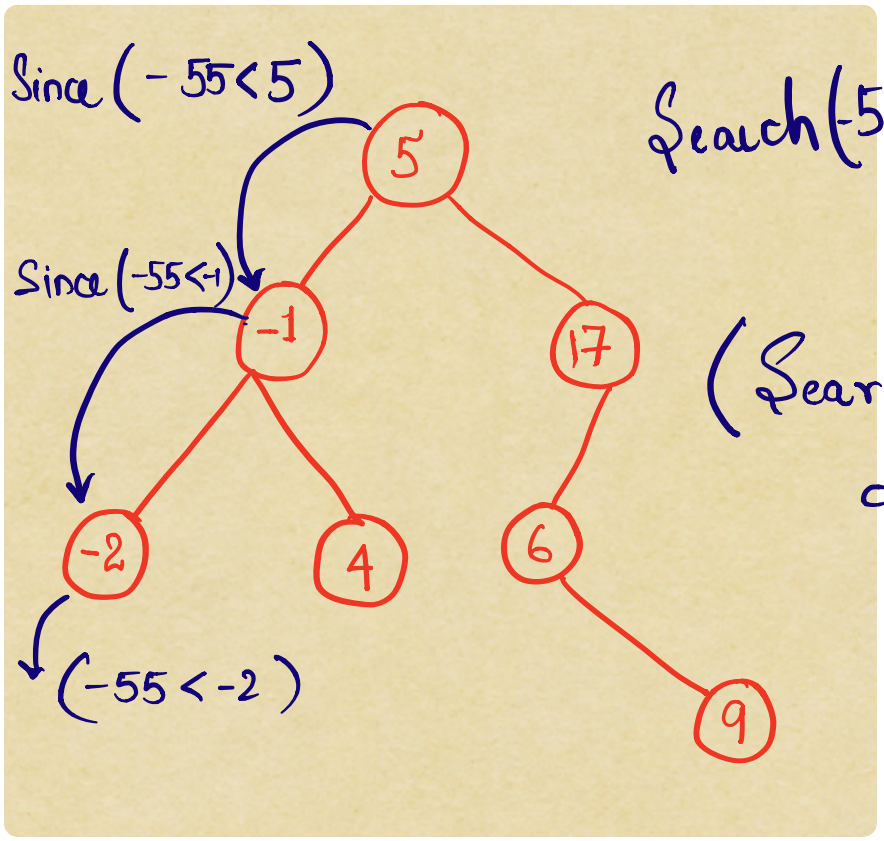
```
Successor( node)
    if ( node.right != NULL)
        return Minimum(node.right)
    endif
    temp ← node.parent
    while (temp != NULL &&
                temp.right == node)
        node ← temp;
        temp ← temp.parent;
    endwhile
    return (temp.data);
```

(Similarly design the algorithm for Predecessor)

Deletion of node with value property v.

Deletion of a node with value property v without disturbing the value and structure property of binary search tree.

Consider the following tree for demonstration

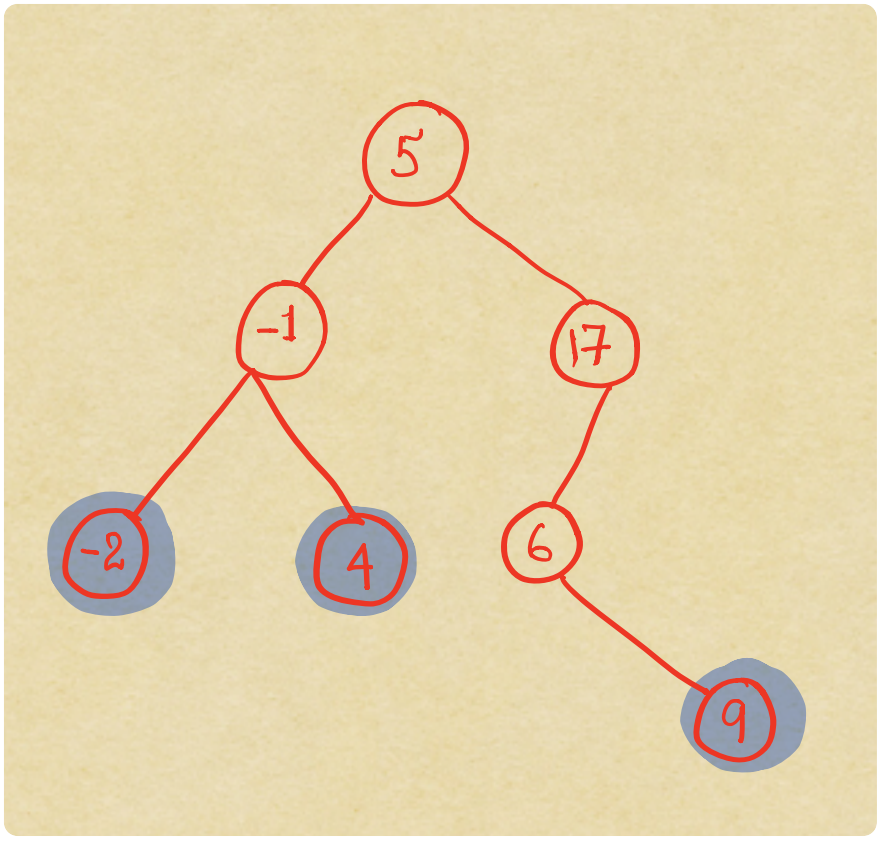


(Search and delete)

In this case Search returns false.

delete(-55) : No node with value property -55 is there so nothing will be deleted

"If the node to be deleted is a leaf node"

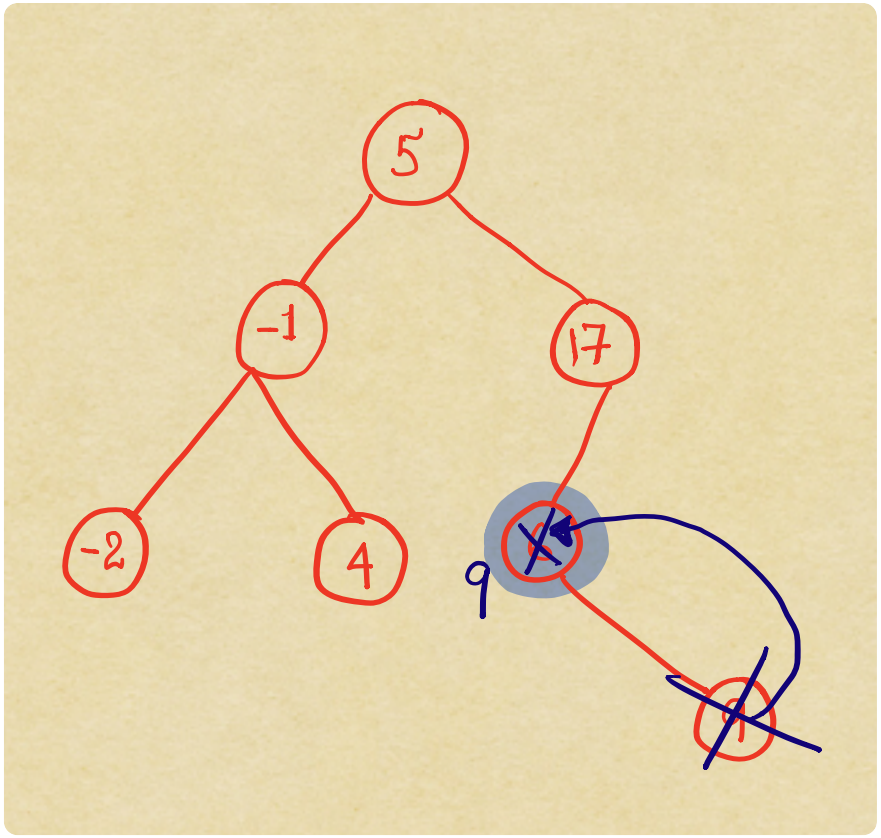


Locate or search the node with the value property, if it is a leaf node (i.e., node's left and right child is NULL)

then

Just Remove the Node from the BST.

If the node to be deleted has exactly one child.



Search the node with the specified value property.
if the node to be deleted has exactly one child
then promote the child

Eg :- node to be deleted is (6)

```
deleteNode ( BSTree , v )
    if ( BSTree == NULL )
            return
    endif
    if ( BSTree.data > v )
            if ( BSTree.left != NULL )
                    deleteNode ( BSTree.left , v ) ;
            endif
            return
    endif
    if ( BSTree.data < v )
            if ( BSTree.right != NULL )
                    deleteNode ( BSTree.right , v ) ;
            endif
            return
    endif
```

Searching or locating the node

```
if ( BSTree.parent == NULL)
    BSTree = NULL;
    return
endif
```

If node to delete is a root node.

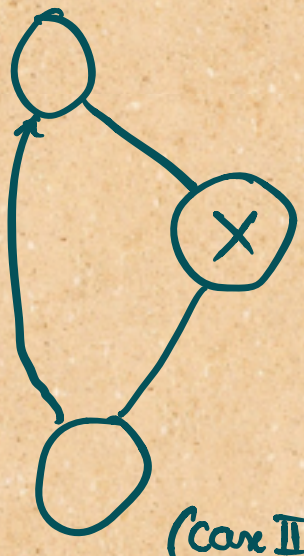
```
if (BSTree.left == NULL  &&
                BSTree.right = NULL)
    if (BSTree.parent.left == BSTree)
        BSTree.parent.left = NULL
    else
        BSTree.parent.right = NULL
    endif
    return
endif
```

If the node to delete is a leaf node

```
if ( BSTree.left != NULL  &&
                BSTree.right == NULL)
    BSTree.left.parent = BSTree.parent
    if ( BSTree.parent.left == BSTree)
        BSTree.parent.left = BSTree.left
    else
        BSTree.parent.right = BSTree.left
    endif
    return
endif
```

If the node to delete has one left child only

(Case I)

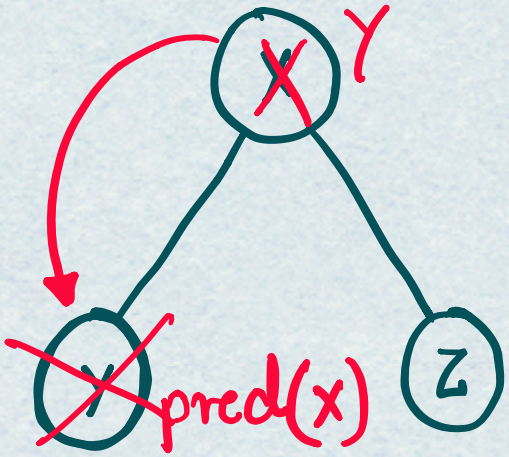


(Case II)

```
if ( BSTree.left == NULL    &&
                BSTree.right != NULL)
    BSTree.right.parent = BSTree.parent
    if ( BSTree.parent.left == BSTree)
            BSTree.parent.left = BSTree.right
    else
            BSTree.parent.right = BSTree.right
    endif
    return
endif
```

If node to delete has one right child only.

```
if ( BSTree.left != NULL  &&
     BSTree.right != NULL)
   pdata ← pred(BSTree) ;  (pred. of the node to be deleted)
   BSTree.data ← pdata
   delete ( BSTree.left , pdata)
   return
endif
```



If node to delete has both left and right child.

Complexities of

- Search
- Adding A node
- Deleting a node
- Maximum
- Minimum

} Covered in class