**Binary Search Tree**

**Binary Search Tree** is a node-based binary tree data structure which has the following properties:

* The left subtree of a node contains only nodes with keys lesser than or equal to the node's key.
* The right subtree of a node contains only nodes with keys greater than the node's key.
* The left and right subtree each must also be a binary search tree.  
  There must be no duplicate nodes.

***Sample Binary Search Tree***:  
200px-Binary_search_tree.svg  
  
The above properties of Binary Search Tree provide an ordering among keys so that the operations like search, minimum and maximum can be done fast in comparison to normal Binary Trees. If there is no ordering, then we may have to compare every key to search a given key.

**Search In BST Recursive**

#include <iostream>

using namespace std;

struct Node{

    int key;

    struct Node \*left;

    struct Node \*right;

    Node(int k){

        key=k;

        left=right=NULL;

    }

};

bool search(Node \*root, int x){

    if(root==NULL)    //it work for empty as well as not present element

        return false;

    if(root->key==x)

        return true;

    else if(root->key>x){

        return search(root->left,x);

    }else{

        return search(root->right,x);

    }

}

int main(){

    Node \*root=new Node(15);

    root->left=new Node(5);

    root->left->left=new Node(3);

    root->right=new Node(20);

    root->right->left=new Node(18);

    root->right->left->left=new Node(16);

    root->right->right=new Node(80);

    int x=16;

    if(search(root,x))

        cout<<"Found";

    else

        cout<<"Not Found";

}

**OUTPUT :**

**Found**

**Search In BST Iterative**

#include <iostream>

using namespace std;

struct Node{

    int key;

    struct Node \*left;

    struct Node \*right;

    Node(int k){

        key=k;

        left=right=NULL;

    }

};

bool search(Node \*root ,int x){

    while (root!=NULL)

    {

        if(root->key==x)

            return true;

        else if(root->key<x)

            root=root->right;

        else

            root=root->left;

    }

    return false;

}

int main(){

    Node \*root=new Node(15);

    root->left=new Node(5);

    root->left->left=new Node(3);

    root->right=new Node(20);

    root->right->left=new Node(18);

    root->right->left->left=new Node(16);

    root->right->right=new Node(80);

    int x=16;

    if(search(root,x))

        cout<<"Found";

    else

        cout<<"Not Found";

}

**OUTPUT : Found**

**BST Insert Iterative**

how to insert a node in a BST, the basic idea is that we first search for the key and if the key is not present in the BST then we insert the key with the leaf node. Else if the key is already present in the BST then we do nothing

#include<iostream>

using namespace std;

//here we need to maintain parent pointer for link

struct Node{

    int key;

    struct Node \*left;

    struct Node \*right;

    Node(int k){

        key=k;

        left=right=NULL;

    }

};

Node \*insert(Node \*root, int x){

    Node \*temp=new Node(x);

    Node \*parent=NULL, \*curr=root;

    while (curr!=NULL)

    {

        parent=curr;

        if(curr->key>x)

            curr=curr->left;

        else if(curr->key<x)

            curr=curr->right;

        else

            return root;

    }

    if(parent==NULL) // parent is null when root is empty its means current is empty

        return temp;  //this condition is execute if and only if root is empty

    if(parent->key>x)  //this condition is execute when while loop is execute at least one time

        parent->left=temp;  // that means root is not empty

    else

        parent->right=temp;

    return root;

}

void inorder(Node \*root){

    if(root!=NULL){

        inorder(root->left);

        cout<<root->key<<" ";

        inorder(root->right);

    }

}

int main(){

    Node \*root=new Node(10);

    root->left=new Node(5);

    root->right=new Node(15);

    root->right->left=new Node(12);

    root->right->right=new Node(18);

    int x=20;

    root=insert(root,x);

    inorder(root);

}

**OUTPUT :**

**5 10 12 15 18 20**

**BST Insert Recursive**

#include<iostream>

using namespace std;

struct Node{

    int key;

    struct Node \*left;

    struct Node \*right;

    Node(int k){

        key=k;

        left=right=NULL;

    }

};

Node \*insert(Node \*root,int x){

    if(root==NULL)

        return new Node(x);

    if(root->key<x)

        root->right=insert(root->right,x);

    else if(root->key>x)

        root->left=insert(root->left,x);

    return root;

}

void inorder(Node \*root){

    if(root!=NULL){

        inorder(root->left);

        cout<<root->key<<" ";

        inorder(root->right);

    }

}

int main(){

    Node \*root=new Node(10);

    root->left=new Node(5);

    root->right=new Node(15);

    root->right->left=new Node(12);

    root->right->right=new Node(18);

    int x=20;

    root=insert(root,x);

    inorder(root);

}

**OUTPUT :**

**5 10 12 15 18 20**

**BST Delete**

/\*how to delete a node from a Binary Search There are three posibilities:

1.Node to be deleted is a leaf node.

2.Node to be deleted has only one child.

3.Node to be deleted has two children.

we also learn, inorder successor is the closest higher value and inorder predecessor is the closest lower value./

\*/

#include <bits/stdc++.h>

using namespace std;

struct Node

{

  int key;

  struct Node \*left;

  struct Node \*right;

  Node(int k){

      key=k;

      left=right=NULL;

  }

};

Node \*getSuccessor(Node \*curr){

    curr=curr->right;

    while(curr!=NULL && curr->left!=NULL)

        curr=curr->left;

    return curr;

}

Node \*delNode(Node \*root, int x){

    if(root==NULL)

        return root;

    if(root->key>x)

        root->left=delNode(root->left,x);

    else if(root->key<x)

        root->right=delNode(root->right,x);

    else{

        if(root->left==NULL){

            Node \*temp=root->right;

            delete root;

            return temp;

        }

        else if(root->right==NULL){

            Node \*temp=root->left;

            delete root;

            return temp;

        }

        else{

            Node \*succ=getSuccessor(root);

            root->key=succ->key;

            root->right=delNode(root->right,succ->key);

        }

    }

    return root;

}

void inorder(Node \*root){

    if(root!=NULL){

        inorder(root->left);

        cout<<root->key<<" ";

        inorder(root->right);

    }

}

int main() {

    Node \*root=new Node(10);

    root->left=new Node(5);

    root->right=new Node(15);

    root->right->left=new Node(12);

    root->right->right=new Node(18);

    int x=15;

    root=delNode(root,x);

    inorder(root);

}

**OUTPUT :**

**5 10 12 18**

**Floor In BST**

#include <bits/stdc++.h>

using namespace std;

//floor means value lower (smaller) closer to or equal to given key

struct Node

{

  int key;

  struct Node \*left;

  struct Node \*right;

  Node(int k){

      key=k;

      left=right=NULL;

  }

};

Node \*floor(Node \*root, int x){

    Node \*res=NULL;

    while(root!=NULL){

        if(root->key==x)

            return root;

        else if(root->key>x)

            root=root->left;

        else{

            res=root;

            root=root->right;

        }

    }

    return res;

}

int main() {

    Node \*root=new Node(15);

    root->left=new Node(5);

    root->left->left=new Node(3);

    root->right=new Node(20);

    root->right->left=new Node(18);

    root->right->left->left=new Node(16);

    root->right->right=new Node(80);

    int x=17;

    cout<<"Floor: "<<(floor(root,17)->key);

}

**OUTPUT :**

**Floor: 16**

**Ceil In BST**

#include <bits/stdc++.h>

using namespace std;

//ceil means value greater (larger) closer to or  equal to given key

struct Node

{

  int key;

  struct Node \*left;

  struct Node \*right;

  Node(int k){

      key=k;

      left=right=NULL;

  }

};

Node \*ceil(Node \*root, int x){

    Node \*res=NULL;

    while(root!=NULL){

        if(root->key==x)

            return root;

        else if(root->key<x)

            root=root->right;

        else{

            res=root;

            root=root->left;

        }

    }

    return res;

}

int main() {

    Node \*root=new Node(15);

    root->left=new Node(5);

    root->left->left=new Node(3);

    root->right=new Node(20);

    root->right->left=new Node(18);

    root->right->left->left=new Node(16);

    root->right->right=new Node(80);

    int x=17;

    cout<<"Ceil: "<<(ceil(root,17)->key);

}

**OUTPUT : Ceil: 18**