```
#include <bits/stdc++.h>
using namespace std;

class Node {
  public:
    int data;
    Node* left;
    Node* right;

    Node(int _data) {
      this->data = _data;
      this->left = NULL;
      this->right = NULL;
    }
};

bool searchInBSTItr(Node *root, int val) {
  while(root) {
    if(root->data == val)
      return true;
    else if(root->data > val)
      root = root->left;
    else
      root = root->right;
  }
  return false;
}

bool searchInBSTRec(Node* root, int val) {
  while(root) {
    if(root->data == val)
      return true;
    else if(root->data > val)
      return searchInBSTRec(root->left,val);
    else
      return searchInBSTRec(root->right,val);
  }
  return false;
}
```

```
Node* maxVal(Node* root) {
   while(root->right) {
      root = root->right;
   }
   return root;
}

Node* minVal(Node* root) {
   while(root->left) {
      root = root->left;
   }
   return root;
}

Node* deleteFromBST(Node* root, int val) {

   if(root == NULL)
      return root;

   if(root->data == val) {
      // 0 child (Leaf Node)
      if(root->left == NULL && root->right == NULL) {
         delete root;
         return NULL;
      }

      // 1 child -> Left child
      if(root->left && !root->right) {
         Node* temp = root->left;
         delete root;
         return temp;
      }

      // 1 child -> Right child
      if(!root->left && root->right) {
         Node* temp = root->right;
         delete root;
         return temp;
      }
      // 2 child
      if(root->left && root->right) {
         int mini = minVal(root->right)->data;
         root->data = mini;
         root->right = deleteFromBST(root->right,mini);
         return root;
      }
   }

   else if(root->data > val)
      root->left = deleteFromBST(root->left,val);
   else
      root->right = deleteFromBST(root->right,val);

   return root;
}

void postorder(Node* root) {
   if(root == NULL)
      return;
   postorder(root->left);
   postorder(root->right);
   cout<<root->data<<" ";
}

void inorder(Node* root) {
   if(root == NULL)
      return;
   inorder(root->left);
   cout<<root->data<<" ";
   inorder(root->right);
}

void preorder(Node* root) {
   if(root == NULL)
      return;
   cout<<root->data<<" ";
   preorder(root->left);
   preorder(root->right);
}
```

```
void levelOrderTraversal(Node* root) {
   queue<Node*> q;
   q.push(root);
   q.push(NULL);

   while(!q.empty()) {
      Node* temp = q.front();
      q.pop();
      if(temp == NULL) {
         // Makes sure every level is printed in a new line
         cout<<endl;
         if(!q.empty())
            q.push(NULL);
      }
      else {
         cout<<temp->data<<" ";
         if(temp->left)
            q.push(temp->left);
         if(temp->right)
            q.push(temp->right);
      }
   }
}

Node* insertIntoBST(Node* root, int data) {
   if(root == NULL) {
      root = new Node(data);
      return root;
   }

   if(data > root->data)
      // Insert into Right Side
      root->right = insertIntoBST(root->right,data);
   else
      // Insert into Left Side
      root->left = insertIntoBST(root->left,data);

   return root;
}
```

```
void takeInput(Node* &root) {
    int data;
    cin>>data;
    while(data != -1) {
        // Take input until -1 is encountered
        root = insertIntoBST(root,data);
        cin>>data;
    }
}

int main() {
    // vector<int> arr = {15,12,3,7,4,8,21,20,19,25,64,77,99,-1};
    Node* root = NULL;
    // for(int i : arr) {
    //     root = insertIntoBST(root,i);
    // }
    cout<<"Enter data to create BST"<<endl;
    takeInput(root);

    cout<<"Level order traversal of BST"<<endl;
    levelOrderTraversal(root);
    cout<<endl;

    cout<<"Preorder traversal of BST"<<endl;
    preorder(root);
    cout<<endl;

    cout<<"Inorder traversal of BST"<<endl;
    inorder(root);
    cout<<endl;

    cout<<"Postorder traversal of BST"<<endl;
    postorder(root);
    cout<<endl;

    cout<<"Minimum Value of BST"<<endl;
    cout<<minVal(root)->data<<endl;

    cout<<"Maximum Value of BST"<<endl;
    cout<<maxVal(root)->data<<endl;

    cout<<"Deleting 0 Child Node (99) from BST"<<endl;
    root = deleteFromBST(root,99);
    cout<<"Inorder traversal of BST"<<endl;
    inorder(root);
    cout<<endl;

    cout<<"Deleting 1 Child Node (3) from BST"<<endl;
    root = deleteFromBST(root,3);

    cout<<"Inorder traversal of BST"<<endl;
    inorder(root);
    cout<<endl;

    cout<<"Deleting 2 Child Node (15) from BST"<<endl;
    root = deleteFromBST(root,15);

    cout<<"Inorder traversal of BST"<<endl;
    inorder(root);
    cout<<endl;

    cout<<"Searching a key = 21"<<endl;
    cout<<searchInBSTRec(root,21);
    cout<<endl;

    cout<<"Searching a key = 99"<<endl;
    cout<<searchInBSTRec(root,99);
    cout<<endl;

    cout<<"Searching a key = 8"<<endl;
    cout<<searchInBSTItr(root,8);
    cout<<endl;

    cout<<"Searching a key = 100"<<endl;
    cout<<searchInBSTItr(root,100);

    return 0;
}
```

OUTPUT:

```
Enter data to create BST

15 12 3 7 4 8 21 20 19 25 64 77 99 -1

Level order traversal of BST

15

12 21

3 20 25

7 19 64

4 8 77

99


Preorder traversal of BST

15 12 3 7 4 8 21 20 19 25 64 77 99

Inorder traversal of BST

3 4 7 8 12 15 19 20 21 25 64 77 99

Postorder traversal of BST

4 8 7 3 12 19 20 99 77 64 25 21 15

Minimum Value of BST

3

Maximum Value of BST

99

Deleting 0 Child Node (99) from BST

Inorder traversal of BST

3 4 7 8 12 15 19 20 21 25 64 77

Deleting 1 Child Node (3) from BST

Inorder traversal of BST

4 7 8 12 15 19 20 21 25 64 77

Deleting 2 Child Node (15) from BST

Inorder traversal of BST

4 7 8 12 19 20 21 25 64 77

Searching a key = 21

1

Searching a key = 99

0

Searching a key = 8

1

Searching a key = 100

0
```