# 

# Department of Computing

# School of Electrical Engineering and Computer Science

**CS - 250: Data Structure and Algorithms**

**Class: BSCS 10AB**

**Lab 09 : Binary Search Tree**

**Date: 07th December, 2021**

**Time: 10:00 am – 12:50 pm   
&  
 02:00 pm – 4:50 pm**

# Mashal Ashfaque

**337203**

# Lab 09: Implementation of Binary Search Trees – Part A

**Introduction:**

This lab is based on the implementation of Binary Search tree and its functions.

**Objectives**

The objectives of this lab are the following:

* Become familiar with implementation of binary search trees
* Write simple applications using binary search tree

**Tools/Software Requirement**

Visual Studio 2012 or gcc or g++

**Description**

In computer science, a binary search tree (BST), which may sometimes also be called an ordered or sorted binary tree, is a node-based binary tree data structure which has the following properties:

* The left sub-tree of a node contains only nodes with keys less than the node's key.
* The right sub-tree of a node contains only nodes with keys greater than the node's key.
* Both the left and right sub-trees must also be binary search trees.

In this lab, you will expand implement binary search tree, study some statistical properties of BST and write a simple application using the BST. We will assume that duplicate insertions are not allowed.

Here is a template of how your class/structure looks like.

class BST\_Node{

Template data;

BST\_Node \*LeftChild;

BST\_Node \*RightChild;

};

**Lab Task**

**Tasks**

Implement the following operations of Binary Search Tree ADT

1. **bool IsEmpty();**

It checks whether the tree is empty or not. It returns true value, the tree is empty; false otherwise.

1. **void Search(template value)**

It searches a value in a BST. It makes use of two pointer variables loc and ploc of type BT\_Node as explained in the class. If the searched value is found, loc should points to the node in which the searched value is found, and ploc to its parent node. If the value is not found, loc should contain NULL value, and ploc should point the logical parent node of the searched value. The following table represents the four possible combinations of values in loc and ploc and their interpretation:

|  |  |  |
| --- | --- | --- |
| **Loc** | **Ploc** | **Interpretation** |
| NULL | null | Value not found. It should be inserted as the root node implying the BST is currently empty. |
| non-null | Value not found. Ploc points to the logical parent node of the searched value. |
| Non-null | null | Value found in the root node of the BST. |
| non-null | Value found in a node other than the root. Loc points to the node in which the searched value is found; ploc points to its parent node. |

1. **Void InsertWithoutDuplication(template value)**

This function calls the above mentioned Search() function to insert a new value in a BST.

* If the searched value already exists in the tree, its duplicate should not be inserted; exit the function by displaying a relevant message.
* If the search value is not found i.e. loc=NULL, the new value should be inserted using ploc.

1. Implement the following tree traversal methods
   1. **PreOrder traversal**
   2. **InOrder traversal**
   3. **PostOrder traversal**
2. Implement a function that prints the **smallest** value of a BST.
3. Implement a function that prints the **largest** value of a BST.
4. Implement a function that traverses a tree and prints only its leaf nodes.
5. Implement a function that counts and returns the number of leaf nodes in a binary tree.
6. Implement a function that counts and returns the number of internal nodes in a binary tree.

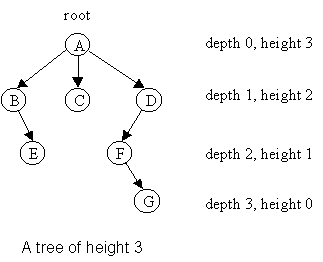


Figure 1: Height and Depth of Nods in Binary Tree (Source: <https://www.google.com/url?sa=i&url=https%3A%2F%2Fcondor.depaul.edu%2Fntomuro%2Fcourses%2F416%2Fnotes%2Flecture3-fall02.html&psig=AOvVaw2LmCcRw9mH1VJncXqqHfOT&ust=1638900929767000&source=images&cd=vfe&v>)

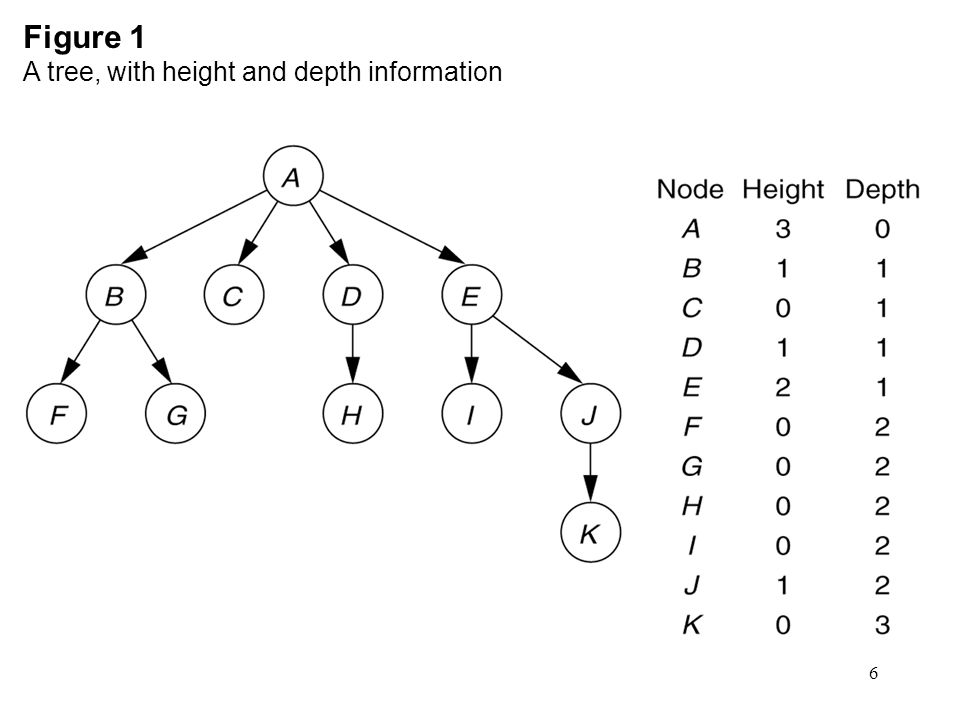


Figure 2: Height and Depth of Nodes in a BT (Source: https://images.slideplayer.com/42/11530377/slides/slide\_6.jpg)

1. Implement a function to calculate the **height** of a BST.

You ought to first identify the base case and recursive case definitions.

**Recursive Case:** To calculate the height of a node x, one needs to first calculate the height of its left sub-tree and right-subtree. The height of node x is 1+Max(height of x’s left subtree, height of x’s right sub-tree).

**Base Case:** Height of an empty tree is -1.

What is height of a leaf node? The answer is one. How? It is 1+max((height of x’s left subtree, height of x’s right sub-tree). For a leaf node, the height of its left sub-tree and its right sub-tree is -1.

1. Implement a function that calculates the **depth** of a BST**.**

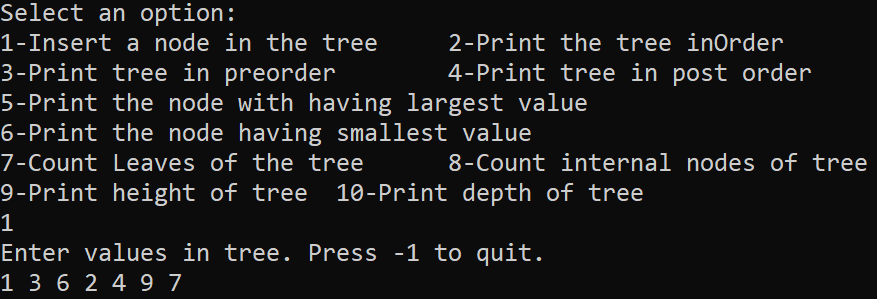
You ought to first identify the base case and recursive case definitions. Here are some hints:

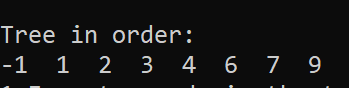
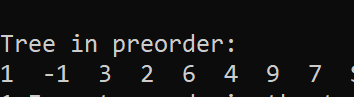
* Depth of the root node of a binary tree is 0.
* Depth of a node **x** in a binary tree is 1 plus the depth of its parent node.
* Pass the depth of the node or that of its parent for which the depth function is called.
* The **depth of a binary tree** rooted at node **y** is the number of nodes from the root y down to the furthest leaf node which may be in its left or right sub-tree.

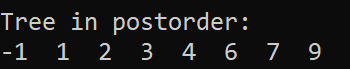
**Deliverables:**

Compile a single word document by filling in the solution part and submit this Word file on LMS. The name of word document should follow this format. i.e. **YourFullName(reg)\_Lab#.** This lab grading policy is as follows: The lab is graded between 0 to 10 marks. The submitted solution can get a maximum of 5 marks. At the end of each lab or in the next lab, there will be a viva related to the tasks. The viva has a weightage of 5 marks. Insert the solution/answer in this document. You must show the implementation of the tasks in the designing tool, along with your complete Word document to get your work graded. You must also submit this Word document on the LMS. In case of any problems discuss it by emailing it to [aftab.farooq@seecs.edu.pk](mailto:aftab.farooq@seecs.edu.pk).

**Note:** Students are required to upload the lab on LMS before deadline.

Use proper indentation and comments. Lack of comments and indentation will result in deduction of marks.





****

**Code:**

#include <iostream>

using namespace std;

class bstnode{

public:

int data;

bstnode\* LeftChild;

bstnode\* RightChild;

bstnode()

{

LeftChild = NULL;

RightChild = NULL;

}

};

class bst

{public:

bstnode\* root;

bstnode\* loc;

bstnode\* preloc;

bst()

{

root = NULL;

loc = NULL;

preloc = NULL;

}

bool IsEmpty(); //returns true if it's empty

void search(int value);

void InsertWithoutDuplication(int value);

};

bool bst::IsEmpty()

{

return (root== NULL);

}

void bst::search(int value) {

if (!IsEmpty())

{

loc = root;

preloc = NULL;

while (loc != NULL && loc->data != value)

{

if (loc->data > value)

{

preloc = loc;

loc = loc->LeftChild;

}

else

{

preloc = loc;

loc = loc->RightChild;

}

}

}

}

void bst::InsertWithoutDuplication(int value)

{

search(value);

if (loc == NULL)

{

bstnode\* newnode = new bstnode();

newnode->data = value;

if (preloc == NULL)

{

root = newnode;

}

else if (value > preloc->data)

{

preloc->RightChild = newnode;

}

else if (value < preloc->data)

{

preloc->LeftChild = newnode;

}

}

}

void preorder(bstnode\* root)

{

if (root != NULL)

{

cout << root->data<< " ";

preorder(root->LeftChild);

preorder(root->RightChild);

}

}

void inorder(bstnode\* root)

{

if (root != NULL)

{

inorder(root->LeftChild);

cout << root->data << " ";

inorder(root->RightChild);

}

}

void postorder(bstnode\* root)

{

if (root != NULL)

{

postorder(root->LeftChild);

cout << root->data << " ";

postorder(root->RightChild);

}

}

void printsmallest(bstnode\* root)

{

if(root!=NULL)

{

if (root->LeftChild != NULL)

{

printsmallest(root->LeftChild);

}

else cout << root->data;

}

}

void printlargest(bstnode\* root)

{

if (root != NULL)

{

if (root->RightChild != NULL)

{

printlargest(root->RightChild);

}

else cout << root->data;

}

}

void printleafnodes(bstnode\* loc)

{

if (loc->LeftChild == NULL && loc->RightChild == NULL)

{

cout << loc->data << " ";

}

else

{

if (loc->LeftChild != NULL)

printleafnodes(loc->LeftChild);

if (loc->RightChild != NULL)

printleafnodes(loc->RightChild);

}

}

int countleaves(bstnode\* root)

{

static int count = 0;

if (root->LeftChild == NULL && root->RightChild == NULL)

{

count++;

return count;

}

else

{

if (root->LeftChild != NULL)

countleaves(root->LeftChild);

if (root->RightChild != NULL)

countleaves(root->RightChild);

}

}

//Task 9

int countinternal(bstnode\* root)

{

//base case

if (root==NULL || (root->LeftChild == NULL && root->RightChild == NULL))

{

return 0;

}

else

{

return 1 + countinternal(root->LeftChild) + countinternal(root->RightChild);

}

}

//Task 10

int height(bstnode\* root)

{

if (root == NULL)

return -1;

else

return 1 + max(height(root->LeftChild), height(root->RightChild));

}

//task11

int DEPTH(bstnode\* root, int depth)

{

if (root == NULL)

return (depth - 1);

//find the depth of each subtree

int a = DEPTH(root->LeftChild, depth + 1);

int b = DEPTH(root->RightChild, depth + 1);

// Choose the larger one and add the root to it.

if (a > b)

return a;

else

return b;

}

int main()

{

bst mytree;

//menu

while (1) {

cout << "Select an option: \n1-Insert a node in the tree\t2-Print the tree inOrder\n3-Print tree in preorder\t4-Print tree in post order\n";

cout << "5-Print the node with having largest value\n6-Print the node having smallest value\n7-Count Leaves of the tree\t8-Count internal nodes of tree\n";

cout << "9-Print height of tree\t10-Print depth of tree" << endl;

int select;

{

cin >> select;

switch (select)

{

case 1: { cout << "Enter values in tree. Press -1 to quit.\n";

int val = 0;

while (val != -1)

{

cin >> val;

mytree.InsertWithoutDuplication(val);

}

break; }

case 2: {cout << "\nTree in order:\n";

inorder(mytree.root);

break; }

case 3: { cout << "\nTree in preorder:\n";

preorder(mytree.root);

break; }

case 4: { cout << "\nTree in postorder:\n";

postorder(mytree.root); break; }

case 5: { cout << "\nLargest value in tree:\t";

printlargest(mytree.root); break; }

case 6: { cout << "\nSmallest value in tree:\t";

printsmallest(mytree.root); break; }

case 7: { cout << "\nTotal Leaves in tree: "<< countleaves(mytree.root);

break; }

case 8: { cout << "\nTotal internal nodes in tree: "<< countinternal(mytree.root);

break; }

case 9: { cout << "\nHeight of tree:\t"<< height(mytree.root);

break; }

}

}

}

}