# 

# Department of Computer Science

**CS 250: Data Structures and Algorithms**

**Binary Search Tree**

**Date: August 03, 2023**

# Instructor: Dr. Syed Imran Ali

# Lab : 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**

You are required to upload the lab tasks on LMS and the name of that tasks must be in this format

FullName\_reg#\_task#.cpp

Remember to comment your code properly. Inappropriate or no comment will result in the deduction of marks.

**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. **InOrer 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.

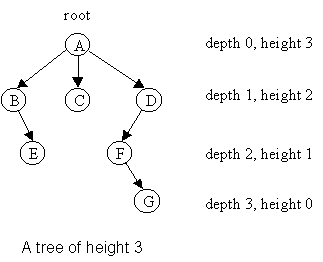


Figure : Height and Depth of Nods in Binary Tree (Source: <https://www.google.com/url?sa=i&url=https%3A%2F%2Fcondor.depaul.edu%2Fntomuro%2Fcourses%2F416%2Fnotes%2Flecture3-fall02.html&psig=AOvVaw2LmCcRw9mH1VJncXqqHfOT&ust=1638900929767000&source=images&cd=vfe&v>)

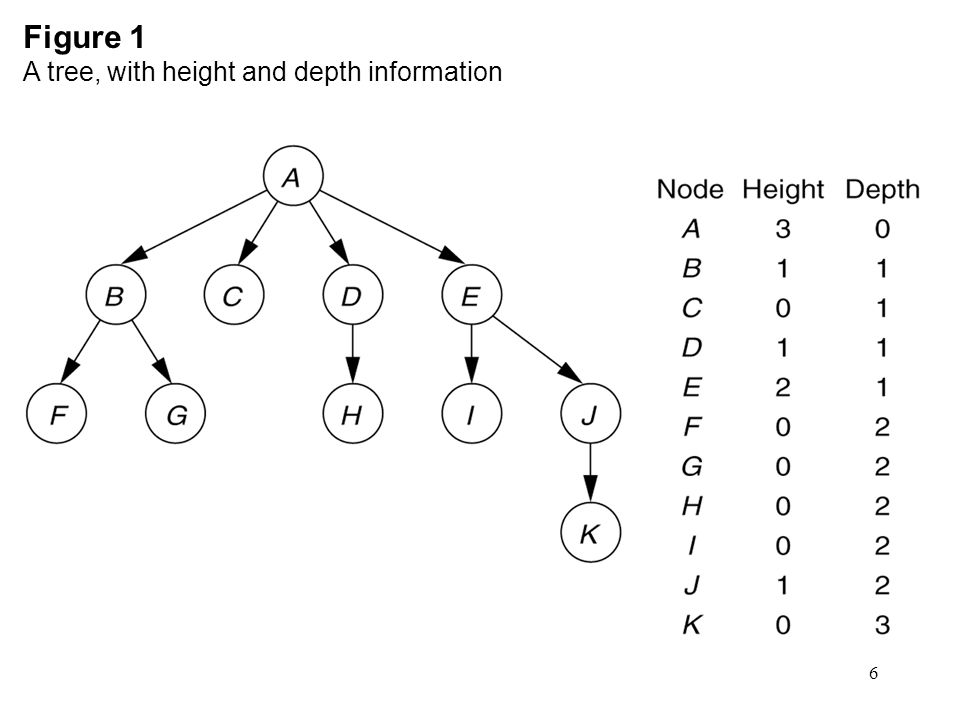


Figure : Height and Depth of Nodes in a BT (Source: https://images.slideplayer.com/42/11530377/slides/slide\_6.jpg)

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**

Students are required to upload the lab task on LMS before the deadline.