19115045 Discrete Mathematics

Kunal Sachdeva Submission Date:

3rd Semester CSE 14/12/2020

Term Paper

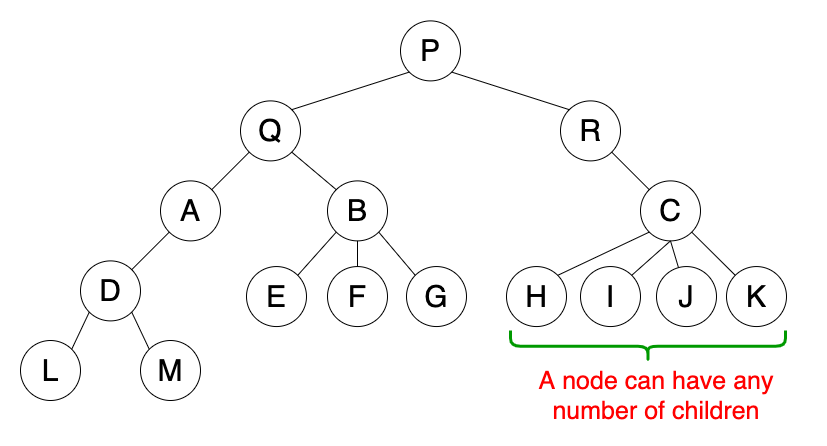
Explain Search Tree and Rooted Binary Tree

Tree is a hierarchical data structure which stores the information naturally in the form of hierarchy unlike linear data structures like, Linked List, Stack, etc. A tree contains nodes(data) and connections(edges) which should not form a cycle.

Types of Trees in Data Structure:

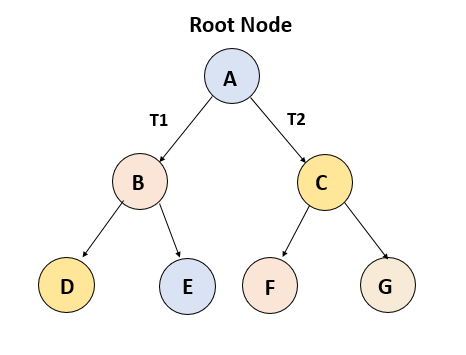
1. General Tree:

If no constraint is placed on the hierarchy of the tree, a tree is called a general tree. Every node may have infinite numbers of children in General Tree. The tree is the super-set of all other trees.



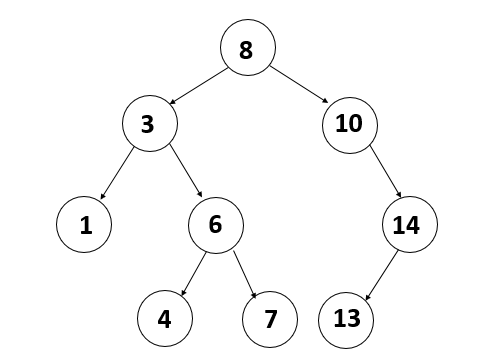
2. Binary Tree:

The binary tree is the kind of tree in which most two children can be found for each parent. The kids are known as the left kid and right kid. This is more popular than most other trees. When certain constraints and characteristics are applied in a Binary tree, a number of others such as AVL tree, BST (Binary Search Tree), RBT tree, etc. are also used.



3. Binary Search Tree:

Binary Search Tree (BST) is a binary tree extension with several optional restrictions. The left child value of a node should in BST be less than or equal to the parent value and the right child value should always be greater than or equal to the parent’s value. This Binary Search Tree property makes it ideal for search operations since we can accurately determine at each node whether the value is in the left or right sub-tree. This is why the Search Tree is named.



The binary Search tree is a binary tree which satisfies the following property −

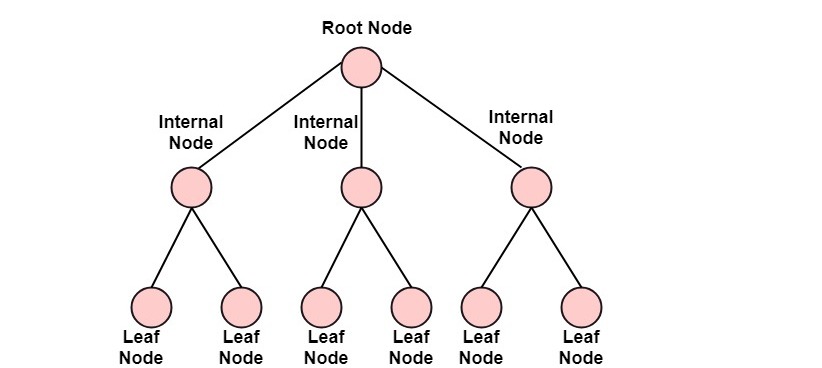
* A in left sub-tree of vertex V, Value(A) ≤ Value (V)
* B in right sub-tree of vertex V, Value(B) ≥ Value (V)

So, the value of all the vertices of the left sub-tree of an internal node ***V*** are less than or equal to ***V*** and the value of all the vertices of the right sub-tree of the internal node ***V*** are greater than or equal to ***V***. The number of links from the root node to the deepest node is the height of the Binary Search Tree.

4. Rooted Trees:

If a directed tree has exactly one node or vertex called root who’s incoming

degrees are 0 and all other vertices have incoming degree one, then the tree is

called rooted tree. 

●A search tree is a tree data structure used for locating specific keys from within a set. In order for a tree to function as a search tree, the key for each node must be greater than any keys in subtrees on the left, and less than any keys in subtrees on the right.

●The advantage of search trees is their efficient search time given the tree is reasonably balanced, which is to say the leaves at either end are of comparable depths. Various search tree data structures exist, several of which also allow efficient insertion and deletion of elements, which operations then have to maintain tree balance.

●Search trees are often used to implement an associative array. The search tree algorithm uses the key from the key-value pair to find a location, and then the application stores the entire key–value pair at that particular location.

●The process of visiting each vertex of a tree in some specified order will be called searching the tree or performing a tree search. In some text, this process is called walking or traversing the tree.

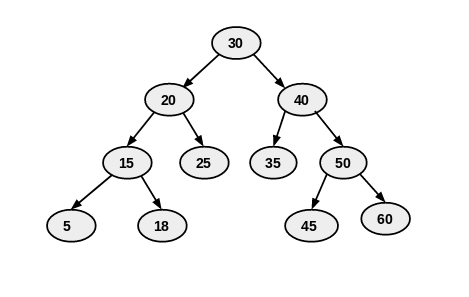
There are three methods for searching (Binary positional trees):

●Inorder Search

●Preorder Search

●Postorder Search

Sample tree:



1. Inorder Search:

As DFS suggests, we will first focus on the depth of the chosen Node and then go to the breadth at that level. Therefore, we will start from the root node of the tree and go deeper-and-deeper into the left subtree with recursive manner.

When we will reach to the left-most node with the above steps, then we will visit that current node and go to the left-most node of its right subtree (if exists).

Same steps should be followed in a recursive manner to complete the inorder search. Order of those steps will be like (in recursive function)

1. Go to left-subtree
2. Visit Node
3. Go to right-subtree

public void inorderSearch (TreeNode root)

{

if (root != null)

{

inorderSearch (root.left);

System.out.print(root.data + " ");

inorderSearch (root.right);

}

}

Inorder traversal of above sample tree:{5, 15, 18, 20, 25, 30, 35, 40, 45, 50, 60}

2. Preorder Search:

Preorder Search is another variant of DFS. Where atomic operations in a recursive function, are as same as Inorder traversal but with a different order.

Here, we visit the current node first and then goes to the left sub-tree. After covering every node of the left sub-tree, we will move towards the right sub-tree and visit in a similar fashion. Order of the steps will be like

1. Visit Node
2. Go to left-subtree
3. Go to right-subtree

public void preorderSearch (TreeNode root)

{

if (root != null)

{

System.out.print(root.data + " ");

preorderSearch (root.left);

preorderSearch (root.right);

}

}

Preorder traversal of above sample tree:{30, 20, 15, 5, 18, 25, 40, 35, 50, 45, 60}

3. Postorder Search:

Similar goes with Postorder Search. Where we visit the left subtree and the right subtree before visiting the current node in recursion.

So, the sequence of the steps will be…

1. Go to left-subtree
2. Go to right-subtree
3. Visit Node

public void postorderSearch(TreeNode root)

{

if (root != null)

{

postorderSearch (root.left);

postorderSearch (root.right);

System.out.print(root.data + " ");

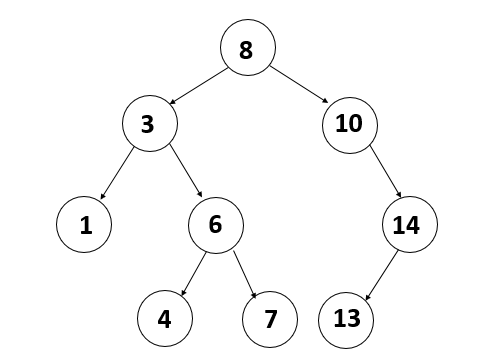
}

}

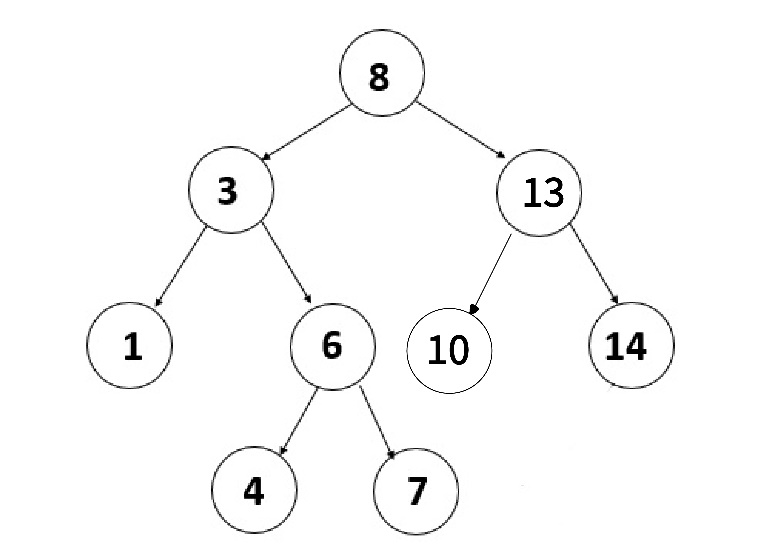
Postorder traversal of above sample tree:{5, 18, 15, 25, 20, 35, 45, 60, 50, 40, 30}

A rooted binary tree is a binary tree in which only the root is allowed to have degree 2. The remaining nodes have degree equal to either 1 or 3. The degree of a vertex is given by the number of edges incident or leaving from it.

In my above example of binary search tree (on page 2). There are 2 nodes (excluding root node) which have degree equal to 2 that is nodes with values 10 and 14.



A proper example for rooted binary tree will be if we modify this tree by creating a left node for the node with value 10 and giving the new node value equal to 10. Store 13 instead of 10 in the parent node of 10 and 14. Delete the child node of 14. These operations will remove the nodes with degree equal to 2. The new updated tree will look like this:



------------------------x------------------------------x-----------------------------x----------------------