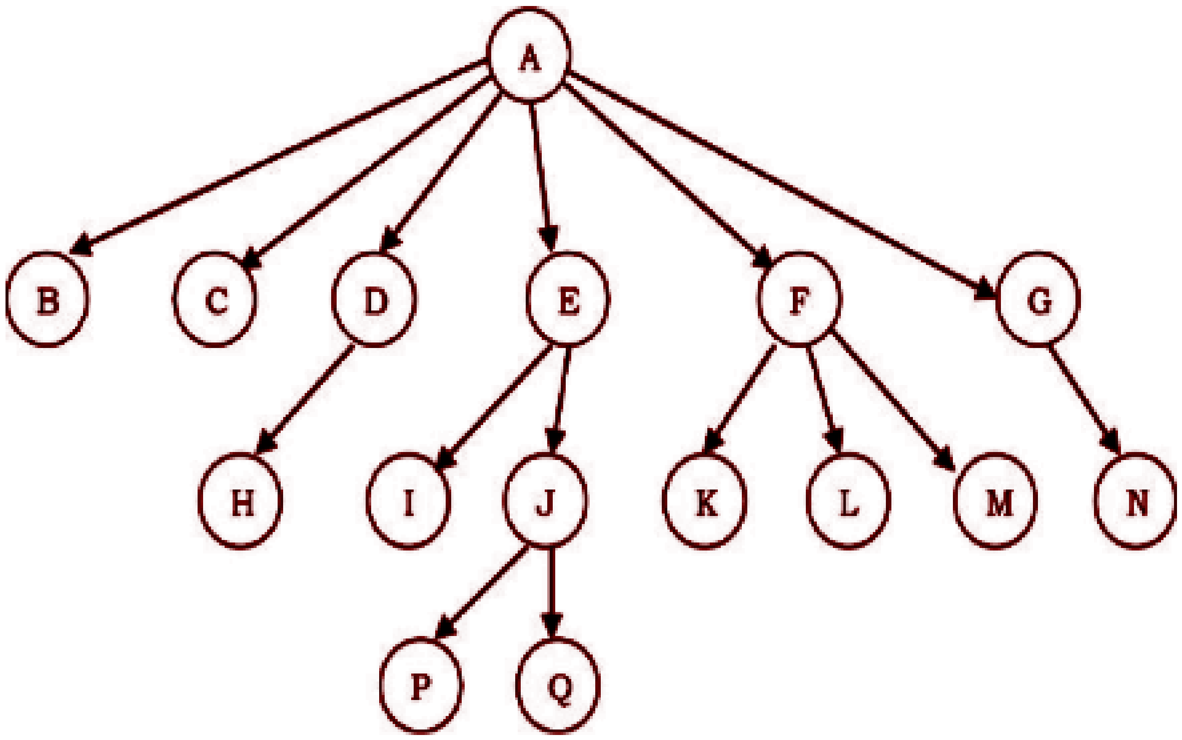
Trees

Introduction

- In the previous modules, we discussed binary trees where each node can have a maximum of two children and these can be represented easily with two pointers i.e right child and left child.
- But suppose, we have a tree with many children for each node.
- If we do not know how many children a node can have, how do we represent such a tree?
- **For example**, consider the tree shown below.



Generic Tree Node

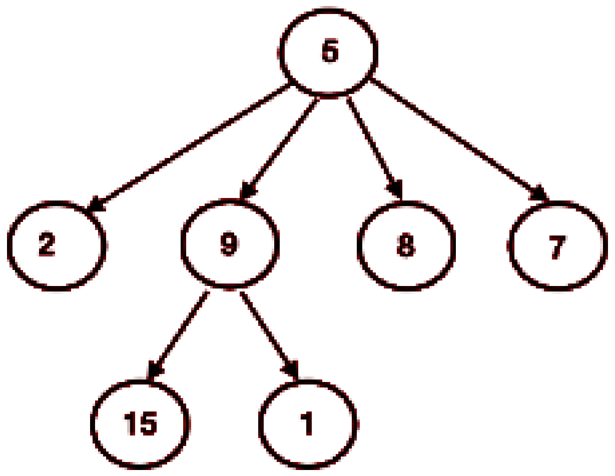
- Implementation of a generic tree node in Java is quite simple.
- The node class will contain two attributes:
    - The node **data**
    - Since there can be a variable number of children nodes, thus the second attribute will be a list of its children nodes. Each child is an instance of the same node class. This is a general **n-nary tree**.
- Consider the given implementation of the **Tree Node** class:

```
class TreeNode <T>{
    T data;
    ArrayList<TreeNode<T>> children;
    public GenericTreeNode(data){
        this.data = data //Node data
        children = new ArrayList<>() //List of children nodes
    }
}
```

Adding Nodes to a Generic Tree

We can add nodes to a generic tree by simply using the **list.add()** function, to add children nodes to parent nodes. This is shown using the given example:

Suppose we have to construct the given Generic Tree:



Consider the given **Java code:**

```
// Create nodes
TreeNode<Integer> n1= TreeNode<>(5);
TreeNode<Integer> n2 =TreeNode<>(2);
TreeNode<Integer> n3 =TreeNode<>(9);
TreeNode<Integer> n4 =TreeNode<>(8);
TreeNode<Integer> n5 =TreeNode<>(7);
TreeNode<Integer> n6 =TreeNode<>(15);
TreeNode<Integer> n7 =TreeNode<>(1);

// Add children for node 5 (n1)
n1.children.add(n2);
n1.children.add(n3);
n1.children.add(n4);
n1.children.add(n5);

// Add children for node 9 (n3)
n3.children.add(n6);
n3.children.add(n7);
```

Print a Generic Tree (Recursively)

In order to print a given generic tree, we will recursively traverse the entire tree.

The steps shall be as follows:

- If the root is **null**, i.e. the tree is an empty tree, then return **null**.
- For every child node at a given node, we call the function recursively.

Go through the given Python code for better understanding:

```
public void printTree(TreeNode root){
      //Not a base case but an edge case
      if (root == null)
            return;

      System.out.println(root.data); //Print current node's data
      for (TreeNode child : root.children)
            printTree(child); //Recursively call the function for children
}
```

Take Generic Tree Input (Recursively)

Go through the given Java code for better understanding:

```
public TreeNode takeTreeInput(){
      System.out.println("Enter root Data");
      int rootData  = s.nextInt(); //TAKE USER INPUT
      if (rootData == -1) //Stop taking inputs
            return null;

      TreeNode<Integer> root = TreeNode<>(rootData);

      System.out.println("Enter number of children for  "+ rootData);
      childrenCount = s.nextInt(); //Get input for no. of child nodes
      while(childrenCount > 0){
            TreeNode child = takeTreeInput(); //Input for all childs
            root.children.add(child); //Add child
            childrenCount--;
      }
      return root;
}
```

Take input level-wise

For taking input level-wise, we will use **queue data structure.** Follow the comments in the code below:

```
public TreeNode<Integer> takeTreeInputLevelwise(){
      System.out.println("Enter root Data");
      int rootData  = s.nextInt(); //TAKE USER INPUT
      TreeNode<Integer> root = new TreeNode<int>(rootData);

      Queue<TreeNode<Integer>> pendingNodes = new Queue<>();
      pendingNodes.push(root);     // Root data pushed into queue at first

      while(pendingNodes.size() != 0){ //Runs until the queue is not empty
            TreeNode<Integer> front = pendingNodes.front();  //stores front
            pendingNodes.pop();// deleted that front node stored previously
            System.out.println("Enter num of children of "+front.data);
            int numChild  = s.nextInt();// get the number of child nodes
            for (int i=0; i<numChild; i++) {  // iterated over each
                                             //child node to input it
                  System.out.println("Enter "+i+"th child of "+front.data);
                  int childData = s.nextInt();
                  TreeNode<Integer> child = new TreeNode<>(childData);
                  front.children.add(child);   //Each child node is pushed
                              //into the queue as well as the list of child
                              //nodes as it is taken input so that next
                             // time we can take its children as input while
                            //we kept moving in the level-wise fashion
                  pendingNodes.push(child);
            }
      }
      return root;  // Finally returns the root node
}
```

Similarly, we can also print the child nodes using a queue itself. Now, try doing the same yourselves and for solution refer to the solution tab of the respective question.

Count total nodes in a tree

To count the total number of nodes in the tree, we will just traverse the tree recursively starting from the root node until we reach the leaf node by iterating over the vector of child nodes. As the size of the child nodes vector becomes 0, we will simply return. Kindly check the code below:

```
public void numNodes(TreeNode<Integer> root){
      if(root == null) {              // Edge case
          return 0;
      }
      int ans = 1;                     // To store total count
      for (int i = 0; i < root.children.size(); i++) {
            ans += numNodes(root.children[i]); // recursively storing count
                                            // of children's children nodes.
      }
      return ans;    // ultimately returning the final answer
}
```

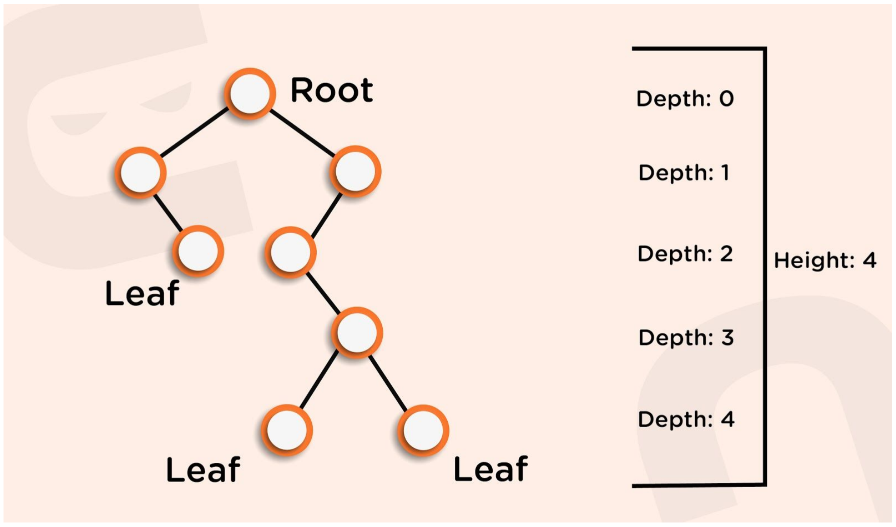
Height of the tree

Height of a tree is defined as the length of the path from the tree's root node to any of its leaf nodes. Just think what should be the height of a tree with just one node? Well, there are a couple of conventions; we can define the height of a tree with just one node to be either 1 or zero. We will be following the convention where the height of a null tree is zero and that with only one node is one. This has been left as an exercise for you, if need be you may follow the code provided in the solution tab of the topic corresponding to the same question.

Approach: Consider the height of the root node as 1 instead of 0. Now, traverse each child of the root node and recursively traverse over each one of them also and the one with the maximum height is added to the final answer along by adding 1 (this 1 is for the current node itself).

Depth of a node

Depth of a node is defined as it's distance from the root node. For example, the depth of the root node is 0, depth of a node directly connected to root node is 1 and so on. Now we will write the code to find the same... (Below is the pictorial representation of the depth of a node)



If you observe carefully, then the depth of the node is just equal to the level in which it resides. We have already figured out how to calculate the level of any node,using a similar approach we will find the depth of the node as well. Suppose, we want to find all the nodes at level 3, then from the root node we will tell its children to find the node that is at level 3 - 1 = 2, and similarly keep this up recursively until we reach the depth = 0. Look at the code below for better understanding...

```
public void printAtLevelK(TreeNode<Integer> root, int k){
      if(root == null) {                // Edge case
          return;
      }

      if(k == 0) {                // Base case: when the depth is 0
          System.out.println(root.data);
          return;
      }

      for(int i=0; i<root.children.size(); i++) { // Iterating over each
                                                  //child and
          printAtLevelK(root.children[i], k - 1);  // recursively calling
                                                 //with with 1 depth less
      }
}
```

Count Leaf nodes

To count the number of leaves, we can simply traverse the nodes recursively until we reach the leaf nodes (the size of the children vector becomes zero). Following recursion, this is very similar to finding the height of the tree. Try to code it yourself and for the solution refer to the solution tab of the same.

Traversals

Traversing the tree is the manner in which we move on the tree in order to access all its nodes. There are generally 4 types of traversals in a tree:

- Level order traversal
- Preorder traversal
- Inorder traversal
- Postorder traversal

We have already discussed level order traversal. Now let's discuss the other traversals.

In Preorder traversal, we visit the current node first(starting with root) and then traverse the left sub-tree. After covering all nodes there, we will move towards the right subtree and visit in a similar manner. Refer the code below:

```
public void preorder(TreeNode<Integer> root){
      if(root == null) {
          return;
      }
      System.out.println(root.data);
      for(int i = 0; i < root.children.size(); i++) {
          preorder(root.children[i]);
      }
}
```