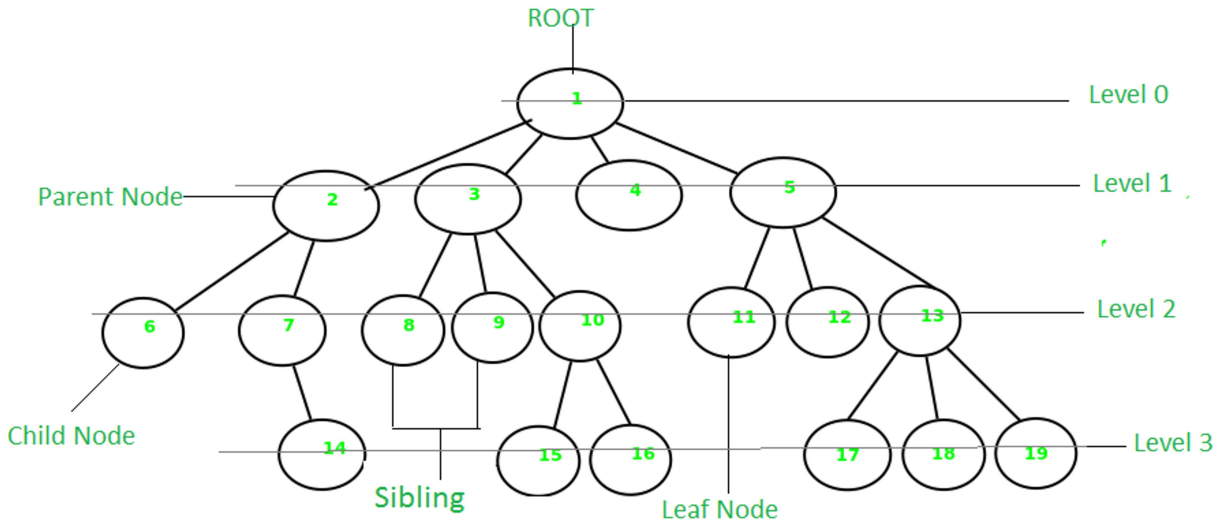
Saturday, January 22, 2022 4:11 PM

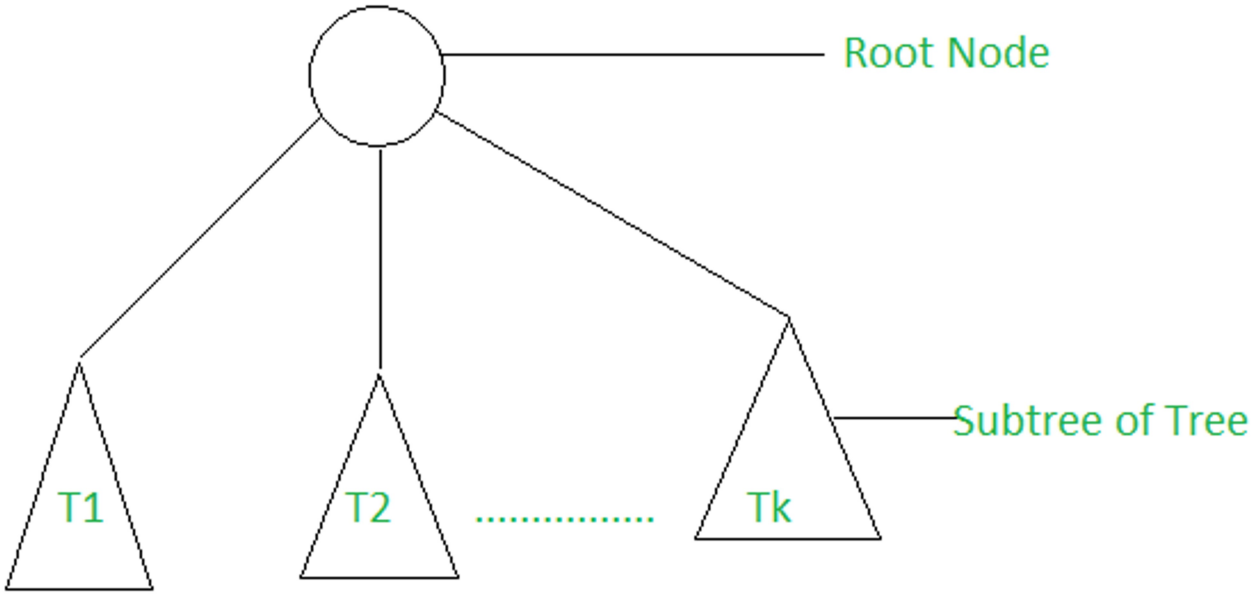
Introduction to Tree Data Structure

- Difficulty Level : Easy
- Last Updated : 12 Jan, 2022

A tree is non-linear and a hierarchical data structure consisting of a collection of nodes such that each node of the tree stores a value, a list of references to nodes (the "children").



Recursive Definition: : A tree consists of a root, and zero or more subtrees T_1, T_2, … , T_k such that there is an edge from the root of the tree to the root of each subtree.



Basic Terminology In Tree Data Structure:

- **Parent Node:** The node which is a predecessor of a node is called the parent node of that

node. {**2}** is the parent node of {**6, 7}**.

- **Child Node:** The node which is the immediate successor of a node is called the child node of that node. Examples: {**6, 7}** are the child nodes of {**2}**.

- **Root Node:** The topmost node of a tree or the node which does not have any parent node is called the root node. {**1}** is the root node of the tree. A non-empty tree must contain exactly one root node and exactly one path from the root to all other nodes of the tree.

- **Degree of a Node:** The total count of subtrees attached to that node is called the degree of the node. The degree of a leaf node must be **0**. The degree of a tree is the degree of its root. The degree of the node {**3}** is **3**.

- **Leaf Node or External Node:** The nodes which do not have any child nodes are called leaf nodes. {**6, 14, 8, 9, 15, 16, 4, 11, 12, 17, 18, 19}** are the leaf nodes of the tree.

- **Ancestor of a Node:** Any predecessor nodes on the path of the root to that node are called Ancestors of that node. {**1, 2}** are the parent nodes of the node {**7}**

- **Descendant:** Any successor node on the path from the leaf node to that node. {**7, 14}** are the descendants of the node. {**2}**.

- **Sibling:** Children of the same parent node are called siblings. {**8, 9, 10}** are called siblings.

- **Depth of a node:** The count of edges from the root to the node. Depth of node {**14}** is **3**.

- **Height of a node**: The number of edges on the longest path from that node to a leaf. Height of node {**3}** is **2**.

- **Height of a tree:** The height of a tree is the height of the root node i.e the count of edges from the root to the deepest node. The height of the above tree is **3**.

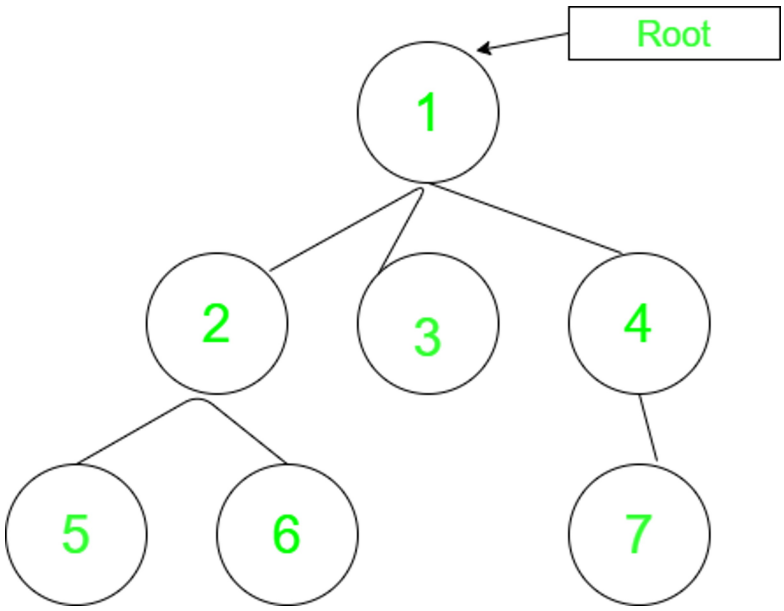
- **Level of a node:** The count of edges on the path from the root node to that node. The root node has level **0**.

- **Internal node:** A node with at least one child is called Internal Node.

- **Neighbour of a Node:** Parent or child nodes of that node are called neighbors of that node.

- **Subtree**: Any node of the tree along with its descendants

  Few examples on Tree Data Structure: A code to demonstrate few of the above terminologies has been described below:



- C++
- Java
- Python3

```java
// java code for above approach
import java.io.*;
import java.util.*;

class GFG
{

  // Function to print the parent of each node
  public static void
    printParents(int node, Vector<Vector<Integer> > adj,
                 int parent)
  {

    // current node is Root, thus, has no parent
    if (parent == 0)
      System.out.println(node + "->Root");
    else
      System.out.println(node + "->" + parent);

    // Using DFS
    for (int i = 0; i < adj.get(node).size(); i++)
      if (adj.get(node).get(i) != parent)
        printParents(adj.get(node).get(i), adj,
                     node);
  }

  // Function to print the children of each node
  public static void
    printChildren(int Root, Vector<Vector<Integer> > adj)
  {

    // Queue for the BFS
    Queue<Integer> q = new LinkedList<>();
```

```
  // pushing the root
  q.add(Root);

  // visit array to keep track of nodes that have been
  // visited
  int vis[] = new int[adj.size()];

  Arrays.fill(vis, 0);

  // BFS
  while (q.size() != 0) {
    int node = q.peek();
    q.remove();
    vis[node] = 1;
    System.out.print(node + "-> ");

    for (int i = 0; i < adj.get(node).size(); i++) {
      if (vis[adj.get(node).get(i)] == 0) {
        System.out.print(adj.get(node).get(i)
                         + " ");
        q.add(adj.get(node).get(i));
      }
    }
    System.out.println();
  }
}

// Function to print the leaf nodes
public static void
  printLeafNodes(int Root, Vector<Vector<Integer> > adj)
{

  // Leaf nodes have only one edge and are not the
  // root
  for (int i = 1; i < adj.size(); i++)
    if (adj.get(i).size() == 1 && i != Root)
      System.out.print(i + " ");

  System.out.println();
}

// Function to print the degrees of each node
public static void
  printDegrees(int Root, Vector<Vector<Integer> > adj)
{
  for (int i = 1; i < adj.size(); i++) {
    System.out.print(i + ": ");

    // Root has noo parent, thus, its degree is
    // equal to the edges it is connected to
    if (i == Root)
      System.out.println(adj.get(i).size());
    else
      System.out.println(adj.get(i).size() - 1);
  }
}
```

```
  // Driver code
  public static void main(String[] args)
  {

    // Number of nodes
    int N = 7, Root = 1;

    // Adjacency list to store the tree
    Vector<Vector<Integer> > adj
      = new Vector<Vector<Integer> >();
    for (int i = 0; i < N + 1; i++) {
      adj.add(new Vector<Integer>());
    }

    // Creating the tree
    adj.get(1).add(2);
    adj.get(2).add(1);

    adj.get(1).add(3);
    adj.get(3).add(1);

    adj.get(1).add(4);
    adj.get(4).add(1);

    adj.get(2).add(5);
    adj.get(5).add(2);

    adj.get(2).add(6);
    adj.get(6).add(2);

    adj.get(4).add(7);
    adj.get(7).add(4);

    // Printing the parents of each node
    System.out.println("The parents of each node are:");
    printParents(Root, adj, 0);

    // Printing the children of each node
    System.out.println(
      "The children of each node are:");
    printChildren(Root, adj);

    // Printing the leaf nodes in the tree
    System.out.println(
      "The leaf nodes of the tree are:");
    printLeafNodes(Root, adj);

    // Printing the degrees of each node
    System.out.println("The degrees of each node are:");
    printDegrees(Root, adj);
  }
}

// This code is contributed by rj13to.
```

Output

```
The parents of each node are:
1->Root
2->1
5->2
6->2
3->1
4->1
7->4
The children of each node are:
1-> 2 3 4
2-> 5 6
3->
4-> 7
5->
6->
7->
The leaf nodes of the tree are:
3 5 6 7
The degrees of each node are:
1: 3
2: 2
3: 0
4: 1
5: 0
6: 0
7: 0
```

From <https://www.geeksforgeeks.org/introduction-to-tree-data-structure/>