Data Structures and Algorithms

W4- Lecture 1,2

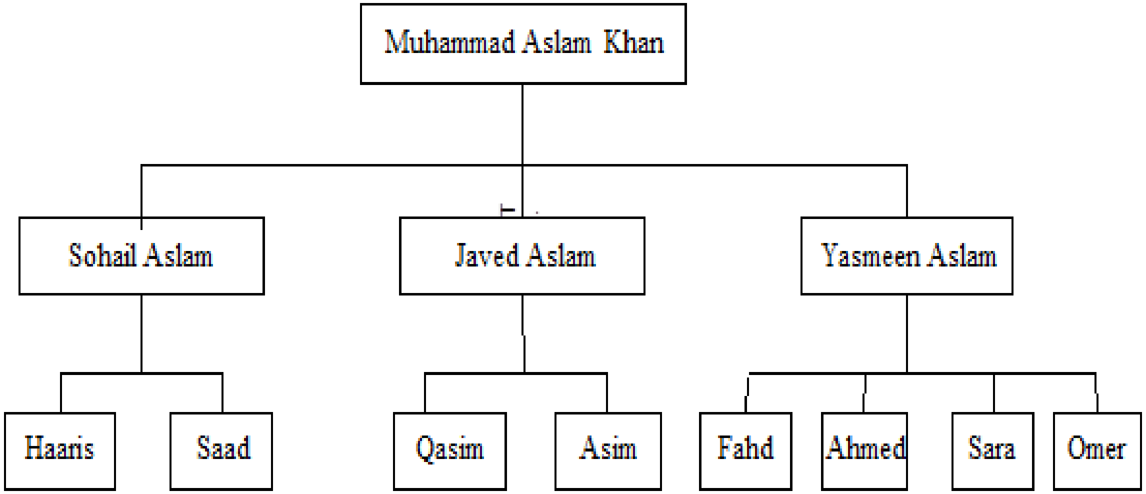
Trees

Engr. Bushra Tahir
Department of Electrical Engineering
Iqra National University

Introduction

- The data structures that we have discussed in previous lectures are linear data structures.
- There are a number of applications where linear data structures are not appropriate. In such cases, there is need of some non-linear data structure.
- Some examples will show us that why non-linear data structures are important. Tree is one of the non-linear data structures.

Genealogy tree



Example

- The thing to be noted in this genealogical tree is that it is not a linear structure
- We develop the tree top-down, in which the father of the family is on the top with their children downward.
- In the tree of a company, the CEO of the company is on the top, followed downwardly by the managers and assistant managers.

Why tree data structure?

- In some applications, the searching in linear data structures is very tedious.
- Suppose we want to search a name in a telephone directory having 100000 entries. If this directory is in a linked list manner, we will have to traverse the list from the starting position.
- We have to traverse on average half of the directory if we find a name. We may not find the required name in the entire directory despite traversing the whole list.
- Thus it would be better to use a data structure so that the search operation does not take a lot of time. Taking into account such applications, we will now talk about a special tree data structure, known as binary tree.

Binary Tree

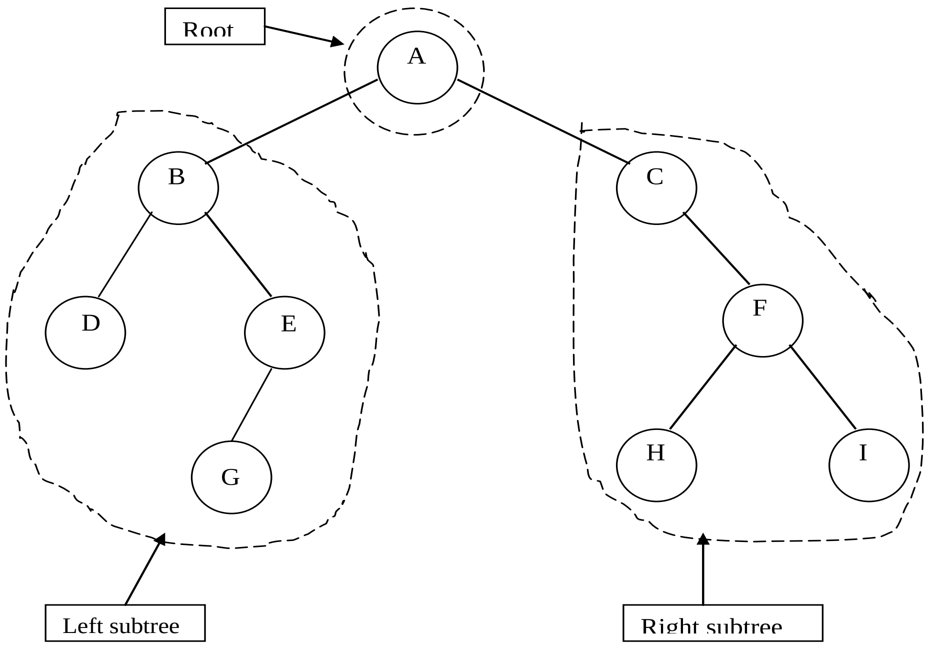
- A binary tree is a finite set of elements that is either empty or is partitioned into three disjoint subsets. The first subset contains a single element called the root of the tree. The other two subsets are themselves binary trees called the left and right sub-trees”.

OR

- In computer science, a **binary tree** is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child.
- The definition of tree is of recursive nature.

Binary Tree

- Each element of a binary tree is called a node of the tree.
- Following figure shows a binary tree.

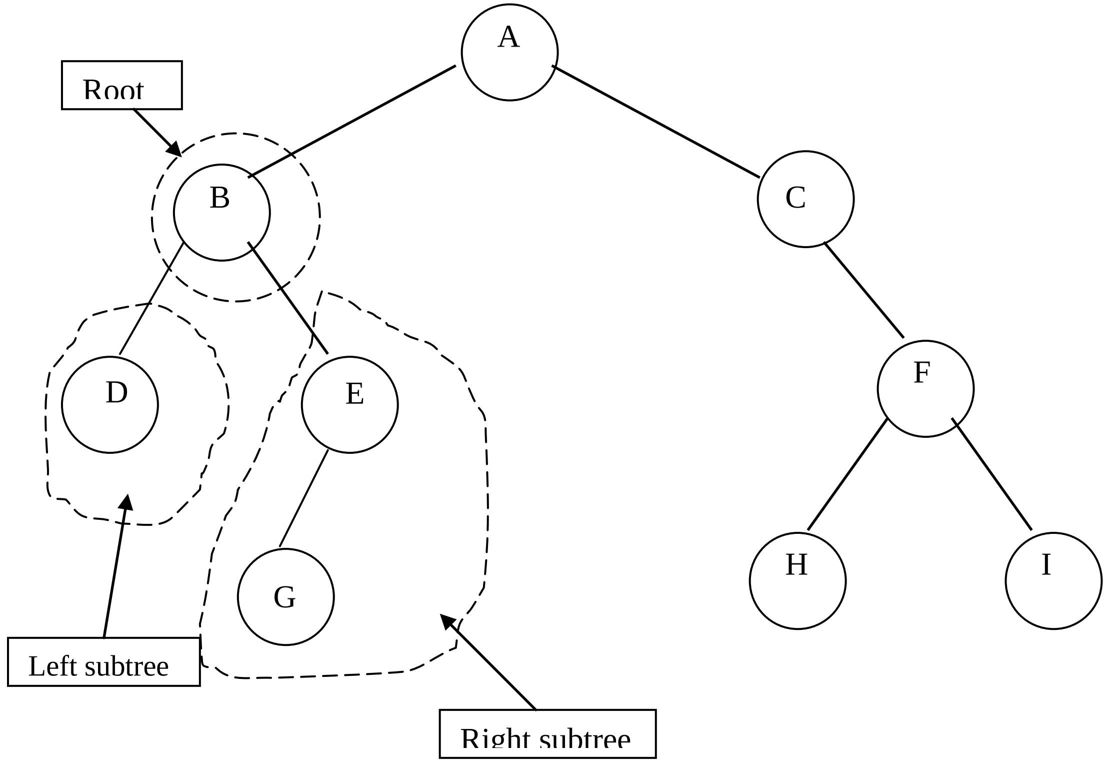


Sub Classing

- The same process of sub classing the tree can be done at the second level.
- The left and right subtrees can also be divided into three parts similarly.
- Now consider the left subtree of node A. This left subtree is also a tree.
- The node B is the root of this tree and its left subtree is the node D. There is only one node in this left subtree. The right subtree of the node B consists of two nodes i.e. E and G.

Sub Classing

- The following figure shows the analysis of the tree with root B.

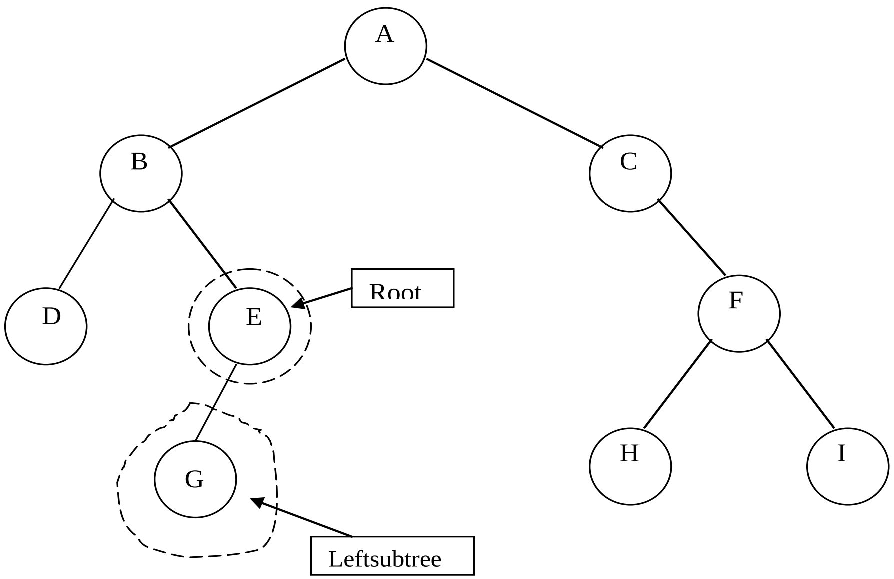


Sub Classing

- On going one step down and considering right subtree of node B, we see that E is the root and its left subtree is the single node G.
- There is no right subtree of node E or we can say that its right subtree is empty.

Sub Classing

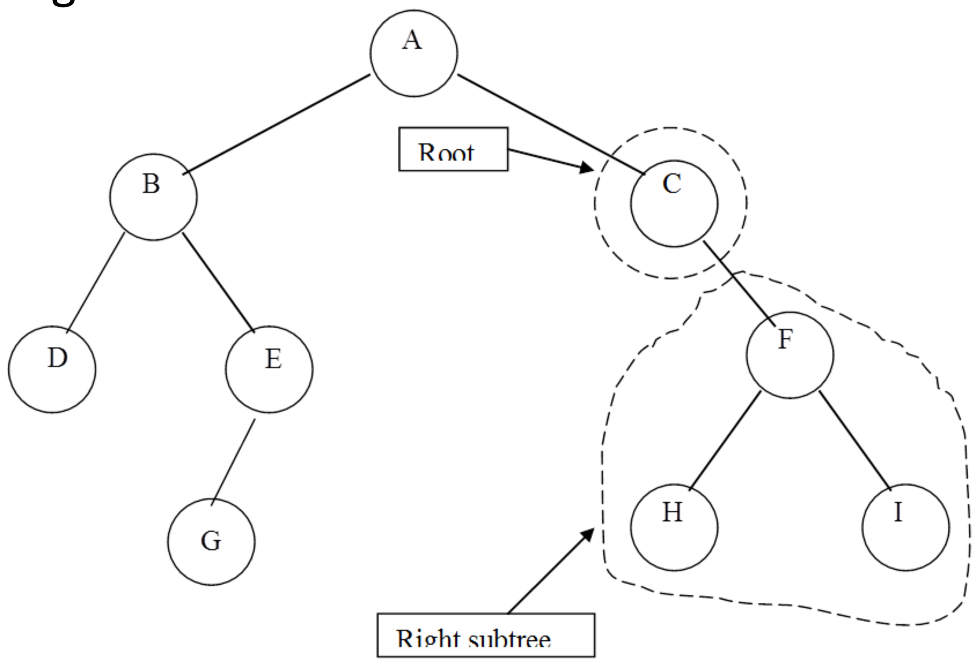
- This is shown in the following figure.



Now the left sub tree of E is the single node G. This node G can be considered as the root node with empty left and right sub trees.

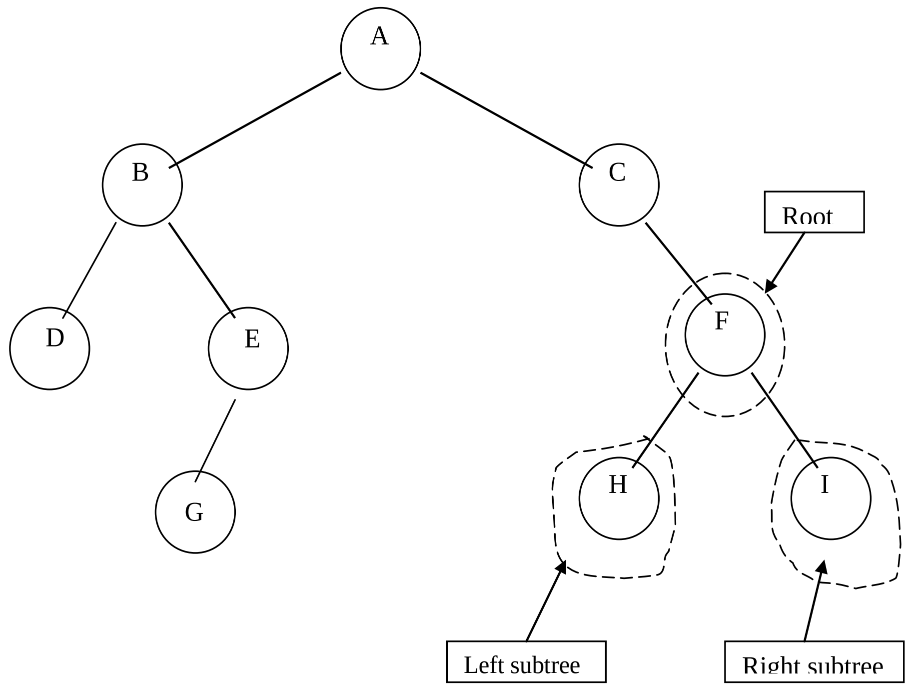
Sub Classing

- Now if we carry out the same process on the right subtree of root node A, the node C will become the root of this right subtree of A.
- The left subtree of this root node C is empty. The right subtree of C is made up of three nodes F, H and I.
- This is shown in the figure below.



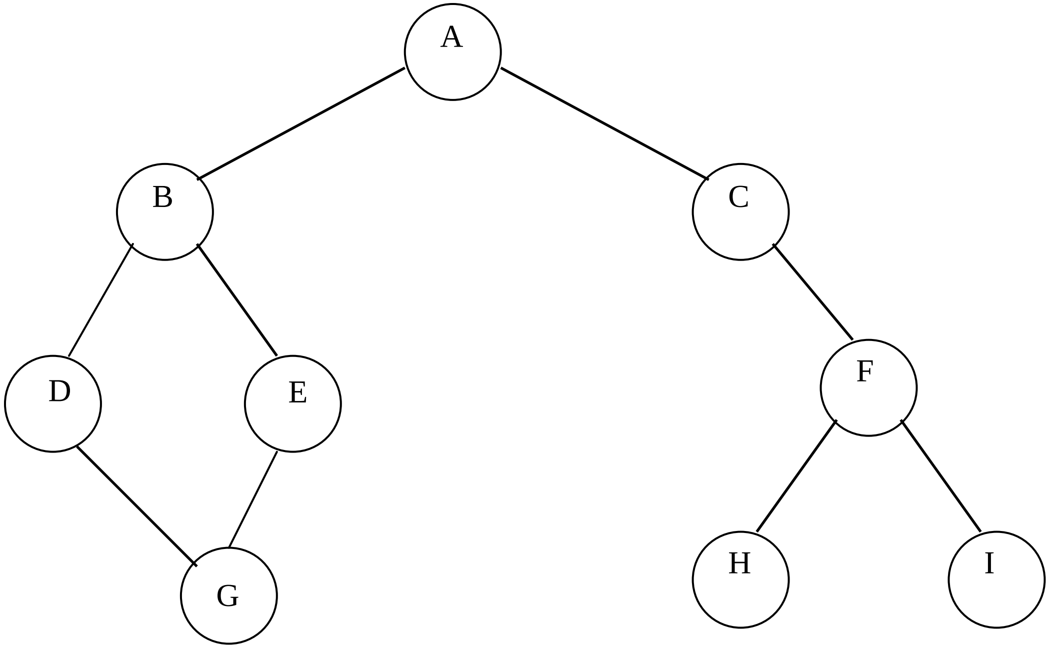
Sub Classing

- Now the right subtree of the node C will be considered as a tree. The root of this tree is the node F. The left and right subtrees of this root F are the nodes H and I respectively.
- The following figure depicts this.



A non-tree structure

- We make (draw) a tree by joining different nodes with lines. But we cannot join any nodes whichever we want to each other. Consider the following figure.

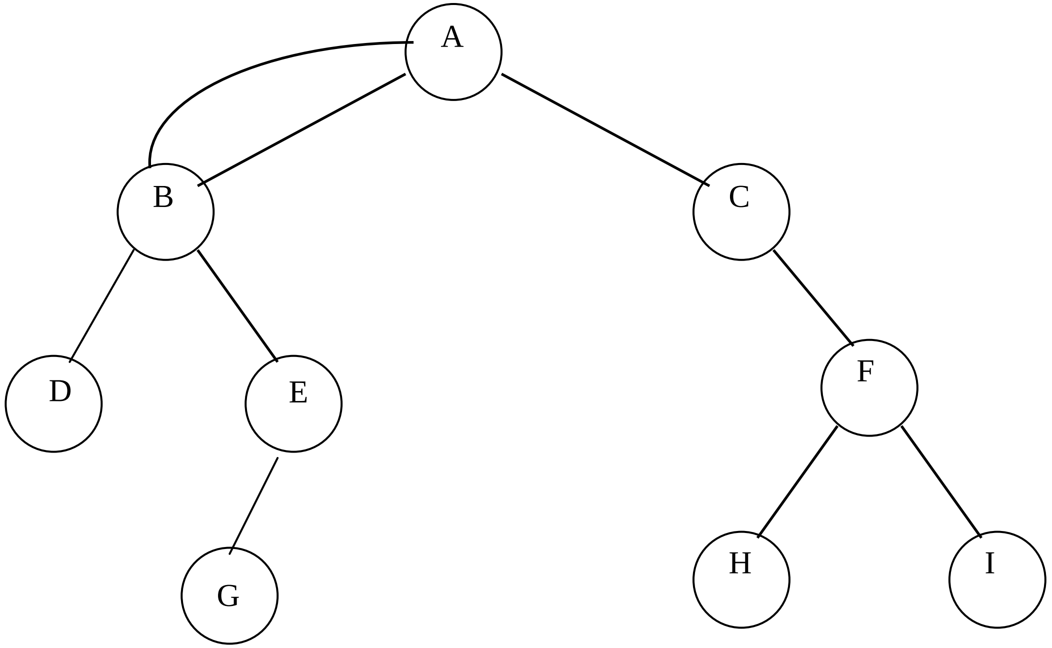


A non-tree structure

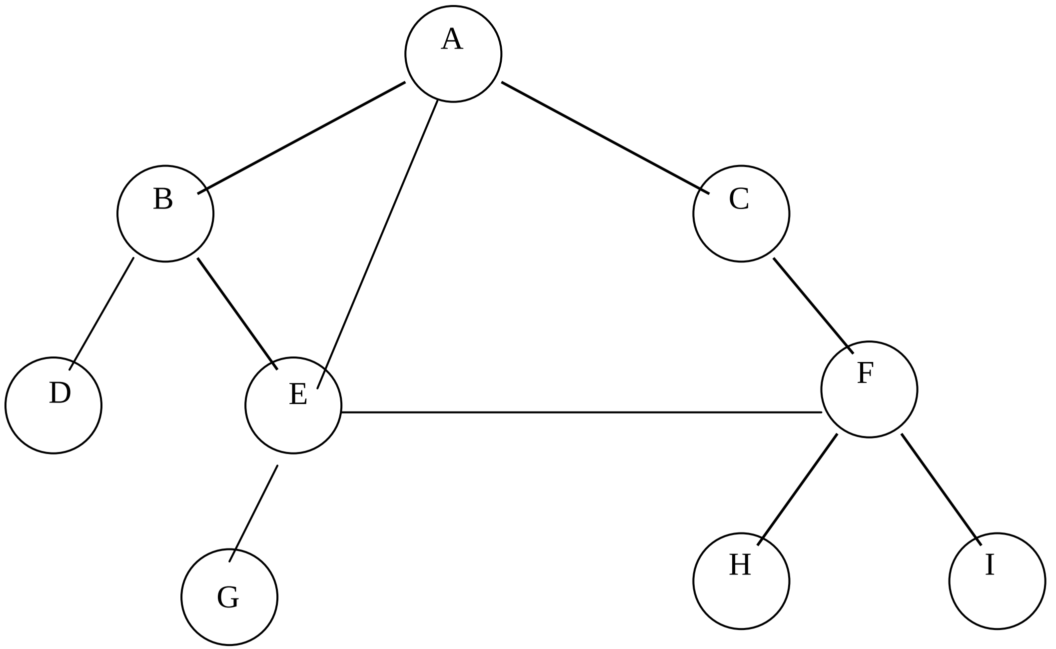
- It is the same tree, made earlier with a little modification.
- In this figure we join the node G with D. Now this is not a tree. It is due to the reason that in a tree there is always one path to go (from root) to a node. But in this figure, there are two paths (tracks) to go to node G.
- One path is from node A to B, then B to D and D to G. That means the path is A-B-D-G. We can also reach to node G by the other path that is the path A-B-E-G.
- If we have such a figure, then it is not a tree.

A non-tree structure

- Similarly if we put an extra link between the nodes A and B, as in the figure below, then it is also no more a tree. This is a multiple graph as there are multiple (more than 1) links between node A and B.

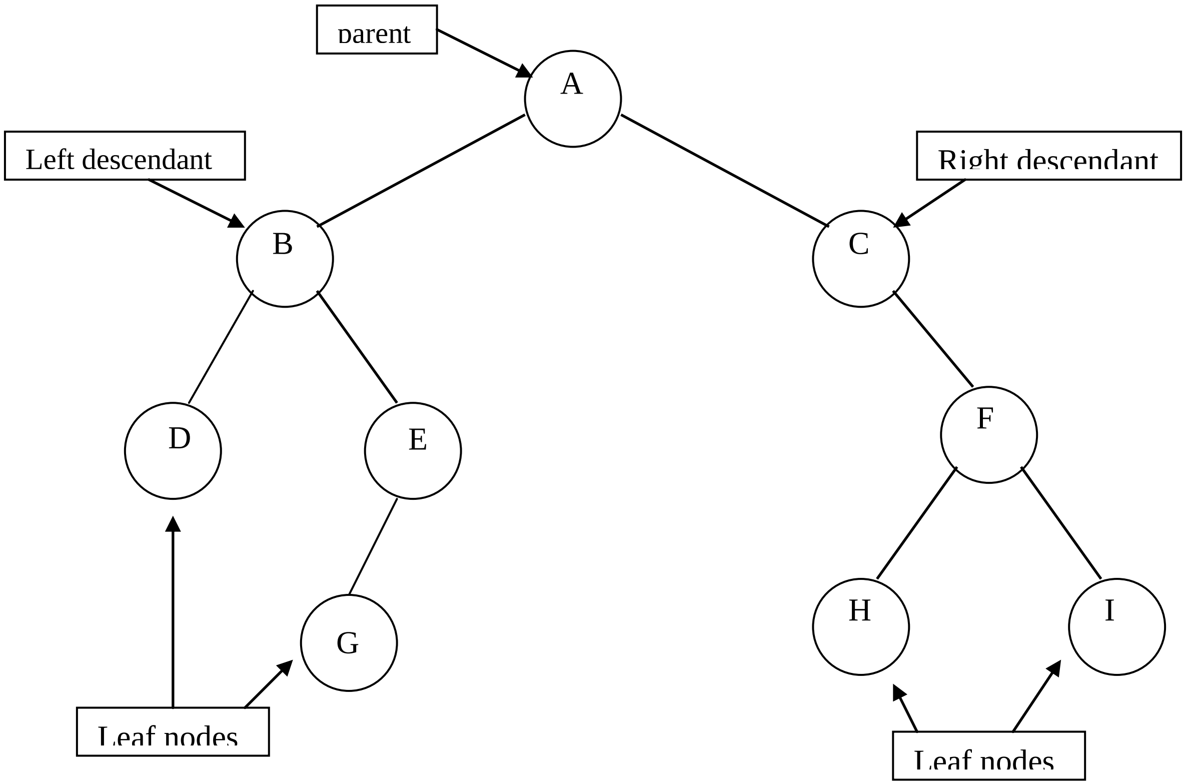


A non-tree structure



Terminologies of a binary tree

- Now let's discuss different terminologies of the binary tree. We will use these terminologies in our different algorithms. Consider the following figure.

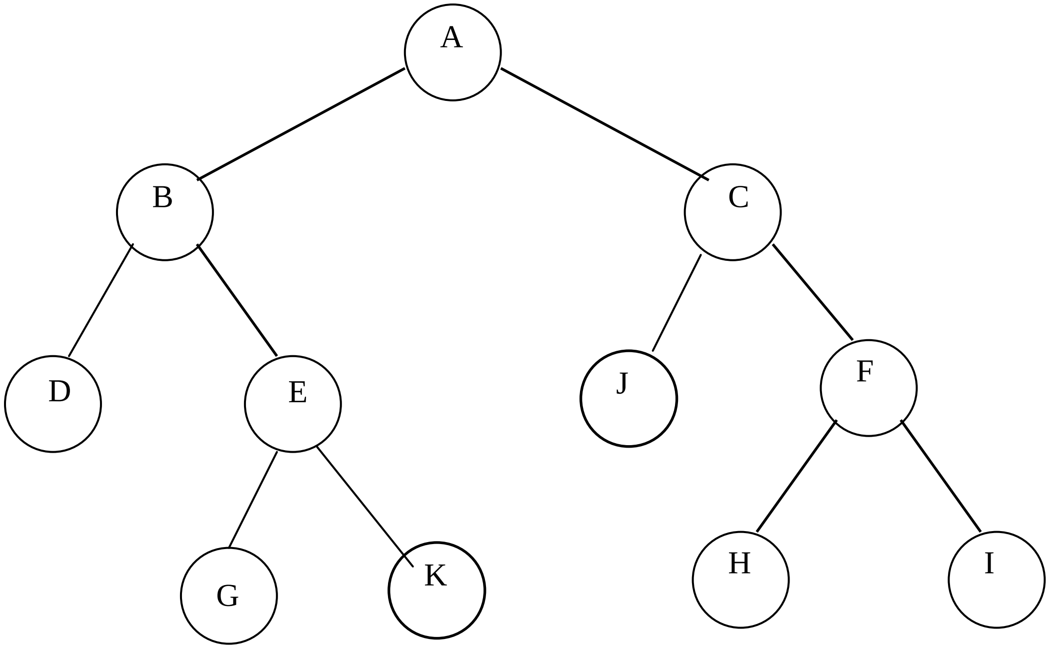


Terminologies of a binary tree

- A is the parent node with B and C as the left and right descendants respectively.
- We can use the words descendant and child interchangeably.
- Nodes D, G, H and I are said leaf nodes as there is no descendant of these nodes.

Strictly Binary Tree

- A binary tree is said to be a strictly binary tree if every non-leaf node in a binary tree has non-empty left and right subtrees.



Strictly Binary Tree

- We know that a leaf node has no left or right child. Thus nodes D, G, H and I are the leaf nodes. The non-leaf nodes in this tree are A, B, C, E and F.
- Now according to the definition of strictly binary tree, these non-leaf nodes (i.e. A, B, C, E and F) must have left and right subtrees (Childs).
- The node A has left child B and right child C. The node B also has its left and right children that are D and E respectively.
- The non-leaf node C has right child F but not a left one.

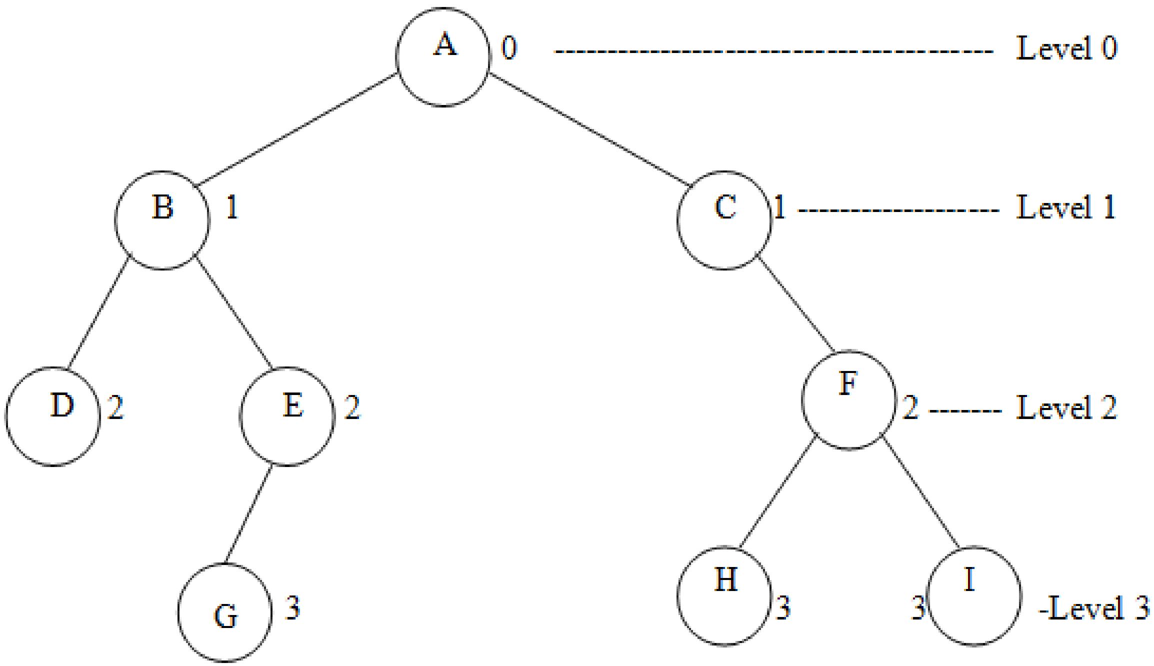
Strictly Binary Tree

- Now we add a left child of C that is node J.
- Here C also has its left and right children. The node F also has its left and right descendants, H and I respectively.
- Now the last non-leaf node E has its left child, G but no right one. We add a node K as the right child of the node E.
- Now all the non-leaf nodes (A, B, C, E and F) have their left and right children so according to the definition, it is a strictly binary tree.

Level

- The level of a node in a binary tree is defined as follows:
- Root has level 0,
- Level of any other node is one more than the level its parent (father).
- The *depth* of a binary tree is the maximum level of any leaf in the tree.

Level



Level

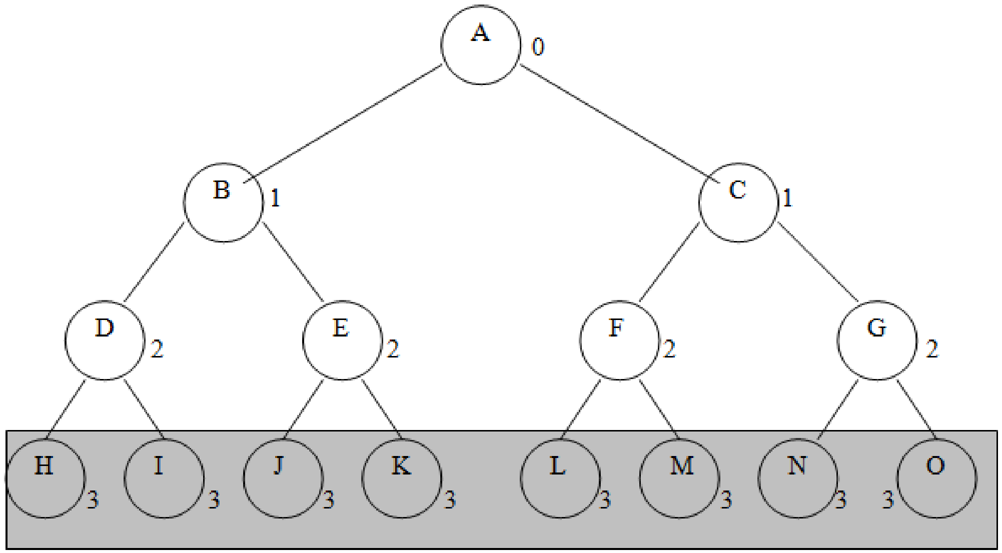
- The root node has level 0.
- The level of B and C will be 1 (i.e. level of A + 1).
- The levels of the tree increase, as the tree grows downward more. The number of nodes also increases.

Depth of Binary Tree

- By seeing the level of a tree, we can tell the depth of the tree.
- If we put level with each node of the binary tree, the depth of a binary tree is the maximum level.
- In the previous figure, the deepest node is at level 3 i.e. the maximum level is 3. So the depth of this binary tree is 3.

Complete Binary Tree

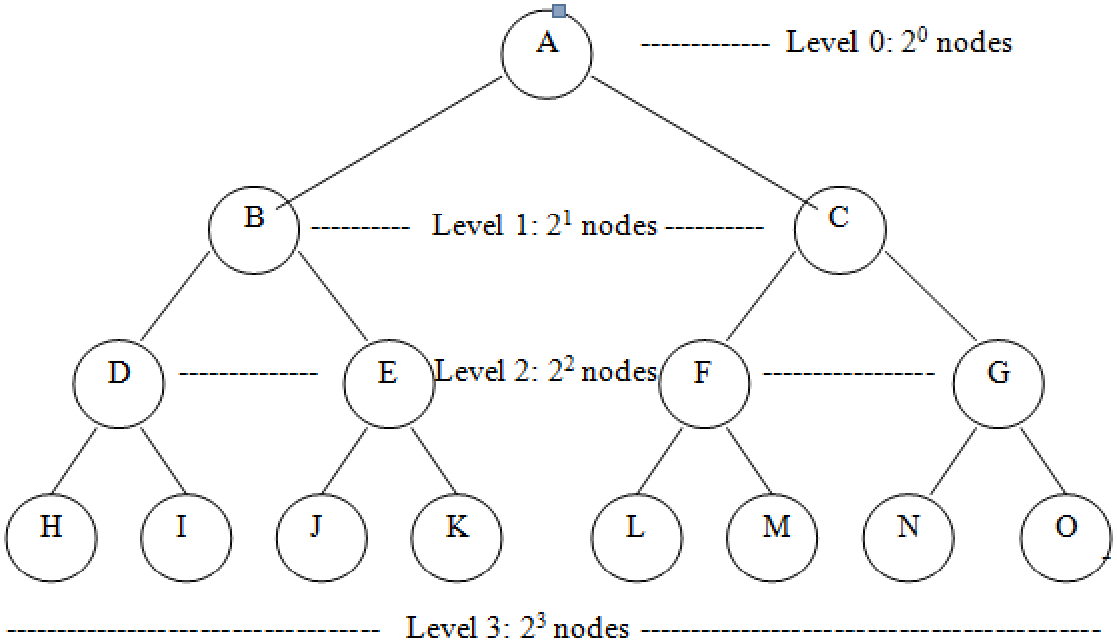
- “A complete binary tree of depth d is the strictly binary tree all of whose leaves are at level d”.



Complete Binary Tree

- The leaf nodes of the tree are at level 3 and are H, I, J, K, L, M, N and O. There is no such a leaf node that is at some level other than the depth level *d* i.e. 3.
- All the leaf nodes of this tree are at level 3 (which is the depth of the tree i.e. d). So this is a complete binary tree.

Level



In a complete binary tree at a particular level *k*, the number of nodes is equal to 2^k.

Number of Nodes

- This formula for the number of nodes is for a complete binary tree only. It is not necessary that every binary tree fulfill this criterion.
- We can calculate the total number of nodes in a complete binary tree of depth *d* by adding the number of nodes at each level. This can be written as the following summation.

$$2^0 + 2^1 + 2^2 + \ldots\ldots\ldots + 2^d = \sum_{j=0}^{d} 2^j = 2^{d+1} - 1$$

Number of Nodes

- Thus according to this summation, the total number of nodes in a complete binary tree of depth *d* will be 2*d*+1 – 1. Thus if there is a complete binary tree of depth 4, the total number of nodes in it will be calculated by putting the value of *d* equal to 4. It will be calculated as under.

  $2^{4+1} - 1 = 2^5 – 1 = 32 – 1 = 31$

- Thus the total number of nodes in the complete binary tree of depth 4 is 31.

Leaf and Non-leaf Nodes

- We know that the total number of nodes (leaf and non-leaf) of a complete binary tree of depth *d* is equal to 2*d*+1 – 1.
- In a complete binary tree, all the leaf nodes are at the depth level *d*. So the number of nodes at level *d* will be 2^d . These are the leaf nodes.
- The difference of total number of nodes and number of leaf nodes will give us the number of non-leaf (inner) nodes. It will be (2*d*+1 – 1) – 2^d i.e. $2^d – 1$.
- Thus we conclude that in a complete binary tree, there are 2^d leaf nodes and $2^d – 1$ non-leaf (inner) nodes.

Level of a Complete Binary Tree

- We can find the depth of a complete binary tree if we know the total number of nodes. If we have a complete binary tree with *n* total nodes, then by the equation of the total number of nodes we can write

$$\text{Total number of nodes} = 2^{d+1} - 1 = n$$

To find the value of d, we solve the above equation as under

$$2^{d+1} - 1 = n$$
$$2^{d+1} = n + 1$$
$$d + 1 = \log_2 (n + 1)$$
$$d = \log_2 (n + 1) - 1$$

Example

- After having *n* total nodes, we can find the depth *d* of the tree by the above equation.
- Suppose we have 100,000 nodes. It means that the value of *n* is 100,000, reflecting a depth i.e. *d* of the tree will be $\log_2 (100000 + 1) - 1$, which evaluates to 15. So the depth of the tree will be 15. In other words, the tree will be 15 levels deep

Operations on Binary Tree

We can define different operations on binary trees.

- If *p* is pointing to a node in an existing tree, then
- left(*p*) returns pointer to the left subtree
- right(*p*) returns pointer to right subtree
- parent(*p*) returns the father of *p*
- info(p) returns content of the node.

Tree Traversal

- Traversal is a process to visit all the nodes of a tree and may print their values too.
- Because, all nodes are connected via edges (links) we always start from the root (head) node.
- We cannot randomly access a node in a tree.

Types

There are three ways which we use to traverse a tree

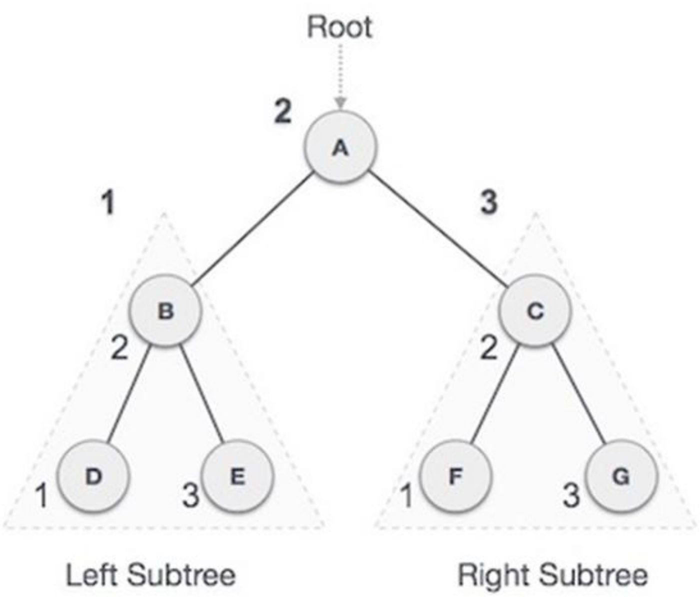
- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

Generally, we traverse a tree to search or locate a given item or key in the tree or to print all the values it contains.

In-order Traversal

- In this traversal method, the left subtree is visited first, then the root and later the right sub-tree. We should always remember that every node may represent a subtree itself.
- If a binary tree is traversed **in-order**, the output will produce sorted key values in an ascending order.

In-order Traversal



We start from **A**, and following in-order traversal, we move to its left subtree **B**. **B** is also traversed in-order. The process goes on until all the nodes are visited. The output of inorder traversal of this tree will be –

D → B → E → A → F → C → G

Algorithm

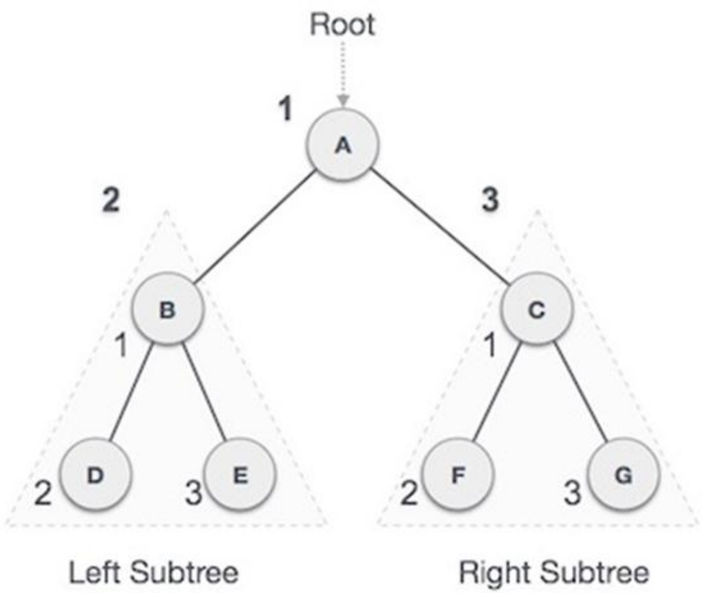
Until all nodes are traversed −

- **Step 1** − Recursively traverse left subtree.
- **Step 2** − Visit root node.
- **Step 3** − Recursively traverse right subtree.

Pre-order Traversal

- In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.

Pre-order Traversal



We start from **A**, and following pre-order traversal, we first visit **A** itself and then move to its left subtree **B**. **B** is also traversed pre-order. The process goes on until all the nodes are visited. The output of pre-order traversal of this tree will be –

A → B → D → E → C → F → G

Algorithm

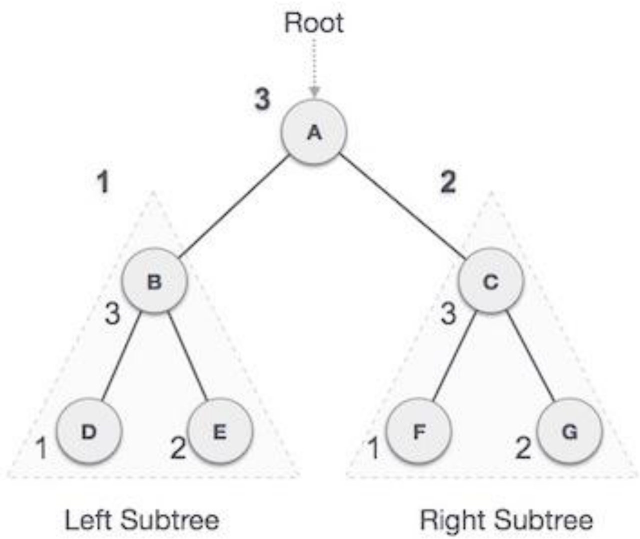
Until all nodes are traversed −

- **Step 1** − Visit root node.
- **Step 2** − Recursively traverse left subtree.
- **Step 3** − Recursively traverse right subtree.

Post-order Traversal

- In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.

Post-order Traversal



We start from **A**, and following Post-order traversal, we first visit the left subtree **B**. **B** is also traversed post-order. The process goes on until all the nodes are visited. The output of post-order traversal of this tree will be −

D → E → B → F → G → C → A

Post-order Traversal

- Until all nodes are traversed –
- **Step 1** – Recursively traverse left subtree.
- **Step 2** – Recursively traverse right subtree.
- **Step 3** – Visit root node.

Questions?