**TREES**

**What is a tree?**

A *tree* is a data structure similar to a linked list but instead of each node pointing simply to the

next node in a linear fashion, each node points to a number of nodes. Tree is an example of a nonlinear

data structure. A *tree* structure is a way of representing the hierarchical nature of a structure

in a graphical form.

In trees ADT (Abstract Data Type), the order of the elements is not important. If we need ordering

information, linear data structures like linked lists, stacks, queues, etc. can be used.

AThe *depth* of a node is the length of the path from the root to the node (depth of *G* is

2, *A* – *C* – *G*).

• The *height* of a node is the length of the path from that node to the deepest node. The

height of a tree is the length of the path from the root to the deepest node in the tree.

A (rooted) tree with only one node (the root) has a height of zero. In the previous

example, the height of *B* is 2 (*B* – *F* – *J*).

• *Height of the tree* is the maximum height among all the nodes in the tree and *depth of*

*the tree* is the maximum depth among all the nodes in the tree. For a given tree,

depth and height returns the same value. But for individual nodes we may get

different results.

• The size of a node is the number of descendants it has including itself (the size of the

subtree *C* is 3).

• If every node in a tree has only one child (except leaf nodes) then we call such trees

*skew trees*. If every node has only left child then we call them *left skew trees*.

Similarly, if every node has only right child then we call them *right skew trees.*

DAThe *depth* of a node is the length of the path from the root to the node (depth of *G* is

2, *A* – *C* – *G*).

• The *height* of a node is the length of the path from that node to the deepest node. The

height of a tree is the length of the path from the root to the deepest node in the tree.

A (rooted) tree with only one node (the root) has a height of zero. In the previous

example, the height of *B* is 2 (*B* – *F* – *J*).

• *Height of the tree* is the maximum height among all the nodes in the tree and *depth of*

*the tree* is the maximum depth among all the nodes in the tree. For a given tree,

depth and height returns the same value. But for individual nodes we may get

different results.

• The size of a node is the number of descendants it has including itself (the size of the

subtree *C* is 3).

• If every node in a tree has only one child (except leaf nodes) then we call such trees

*skew trees*. If every node has only left child then we call them *left skew trees*.

Similarly, if every node has only right child then we call them *right skew trees.*

BAThe *depth* of a node is the length of the path from the root to the node (depth of *G* is

2, *A* – *C* – *G*).

• The *height* of a node is the length of the path from that node to the deepest node. The

height of a tree is the length of the path from the root to the deepest node in the tree.

A (rooted) tree with only one node (the root) has a height of zero. In the previous

example, the height of *B* is 2 (*B* – *F* – *J*).

• *Height of the tree* is the maximum height among all the nodes in the tree and *depth of*

*the tree* is the maximum depth among all the nodes in the tree. For a given tree,

depth and height returns the same value. But for individual nodes we may get

different results.

• The size of a node is the number of descendants it has including itself (the size of the

subtree *C* is 3).

• If every node in a tree has only one child (except leaf nodes) then we call such trees

*skew trees*. If every node has only left child then we call them *left skew trees*.

Similarly, if every node has only right child then we call them *right skew trees.*

CAThe *depth* of a node is the length of the path from the root to the node (depth of *G* is

2, *A* – *C* – *G*).

• The *height* of a node is the length of the path from that node to the deepest node. The

height of a tree is the length of the path from the root to the deepest node in the tree.

A (rooted) tree with only one node (the root) has a height of zero. In the previous

example, the height of *B* is 2 (*B* – *F* – *J*).

• *Height of the tree* is the maximum height among all the nodes in the tree and *depth of*

*the tree* is the maximum depth among all the nodes in the tree. For a given tree,

depth and height returns the same value. But for individual nodes we may get

different results.

• The size of a node is the number of descendants it has including itself (the size of the

subtree *C* is 3).

• If every node in a tree has only one child (except leaf nodes) then we call such trees

*skew trees*. If every node has only left child then we call them *left skew trees*.

Similarly, if every node has only right child then we call them *right skew trees.*

HAThe *depth* of a node is the length of the path from the root to the node (depth of *G* is

2, *A* – *C* – *G*).

• The *height* of a node is the length of the path from that node to the deepest node. The

height of a tree is the length of the path from the root to the deepest node in the tree.

A (rooted) tree with only one node (the root) has a height of zero. In the previous

example, the height of *B* is 2 (*B* – *F* – *J*).

• *Height of the tree* is the maximum height among all the nodes in the tree and *depth of*

*the tree* is the maximum depth among all the nodes in the tree. For a given tree,

depth and height returns the same value. But for individual nodes we may get

different results.

• The size of a node is the number of descendants it has including itself (the size of the

subtree *C* is 3).

• If every node in a tree has only one child (except leaf nodes) then we call such trees

*skew trees*. If every node has only left child then we call them *left skew trees*.

Similarly, if every node has only right child then we call them *right skew trees.*

IAThe *depth* of a node is the length of the path from the root to the node (depth of *G* is

2, *A* – *C* – *G*).

• The *height* of a node is the length of the path from that node to the deepest node. The

height of a tree is the length of the path from the root to the deepest node in the tree.

A (rooted) tree with only one node (the root) has a height of zero. In the previous

example, the height of *B* is 2 (*B* – *F* – *J*).

• *Height of the tree* is the maximum height among all the nodes in the tree and *depth of*

*the tree* is the maximum depth among all the nodes in the tree. For a given tree,

depth and height returns the same value. But for individual nodes we may get

different results.

• The size of a node is the number of descendants it has including itself (the size of the

subtree *C* is 3).

• If every node in a tree has only one child (except leaf nodes) then we call such trees

*skew trees*. If every node has only left child then we call them *left skew trees*.

Similarly, if every node has only right child then we call them *right skew trees.*

GAThe *depth* of a node is the length of the path from the root to the node (depth of *G* is

2, *A* – *C* – *G*).

• The *height* of a node is the length of the path from that node to the deepest node. The

height of a tree is the length of the path from the root to the deepest node in the tree.

A (rooted) tree with only one node (the root) has a height of zero. In the previous

example, the height of *B* is 2 (*B* – *F* – *J*).

• *Height of the tree* is the maximum height among all the nodes in the tree and *depth of*

*the tree* is the maximum depth among all the nodes in the tree. For a given tree,

depth and height returns the same value. But for individual nodes we may get

different results.

• The size of a node is the number of descendants it has including itself (the size of the

subtree *C* is 3).

• If every node in a tree has only one child (except leaf nodes) then we call such trees

*skew trees*. If every node has only left child then we call them *left skew trees*.

Similarly, if every node has only right child then we call them *right skew trees.*

FAThe *depth* of a node is the length of the path from the root to the node (depth of *G* is

2, *A* – *C* – *G*).

• The *height* of a node is the length of the path from that node to the deepest node. The

height of a tree is the length of the path from the root to the deepest node in the tree.

A (rooted) tree with only one node (the root) has a height of zero. In the previous

example, the height of *B* is 2 (*B* – *F* – *J*).

• *Height of the tree* is the maximum height among all the nodes in the tree and *depth of*

*the tree* is the maximum depth among all the nodes in the tree. For a given tree,

depth and height returns the same value. But for individual nodes we may get

different results.

• The size of a node is the number of descendants it has including itself (the size of the

subtree *C* is 3).

• If every node in a tree has only one child (except leaf nodes) then we call such trees

*skew trees*. If every node has only left child then we call them *left skew trees*.

Similarly, if every node has only right child then we call them *right skew trees.*

EAEEThe *depth* of a node is the length of the path from the root to the node (depth of *G* is

2, *A* – *C* – *G*).

• The *height* of a node is the length of the path from that node to the deepest node. The

height of a tree is the length of the path from the root to the deepest node in the tree.

A (rooted) tree with only one node (the root) has a height of zero. In the previous

example, the height of *B* is 2 (*B* – *F* – *J*).

• *Height of the tree* is the maximum height among all the nodes in the tree and *depth of*

*the tree* is the maximum depth among all the nodes in the tree. For a given tree,

depth and height returns the same value. But for individual nodes we may get

different results.

• The size of a node is the number of descendants it has including itself (the size of the

subtree *C* is 3).

• If every node in a tree has only one child (except leaf nodes) then we call such trees

*skew trees*. If every node has only left child then we call them *left skew trees*.

Similarly, if every node has only right child then we call them *right skew trees.*

JAThe *depth* of a node is the length of the path from the root to the node (depth of *G* is

2, *A* – *C* – *G*).

• The *height* of a node is the length of the path from that node to the deepest node. The

height of a tree is the length of the path from the root to the deepest node in the tree.

A (rooted) tree with only one node (the root) has a height of zero. In the previous

example, the height of *B* is 2 (*B* – *F* – *J*).

• *Height of the tree* is the maximum height among all the nodes in the tree and *depth of*

*the tree* is the maximum depth among all the nodes in the tree. For a given tree,

depth and height returns the same value. But for individual nodes we may get

different results.

• The size of a node is the number of descendants it has including itself (the size of the

subtree *C* is 3).

• If every node in a tree has only one child (except leaf nodes) then we call such trees

*skew trees*. If every node has only left child then we call them *left skew trees*.

Similarly, if every node has only right child then we call them *right skew trees.*

KAThe *depth* of a node is the length of the path from the root to the node (depth of *G* is

2, *A* – *C* – *G*).

• The *height* of a node is the length of the path from that node to the deepest node. The

height of a tree is the length of the path from the root to the deepest node in the tree.

A (rooted) tree with only one node (the root) has a height of zero. In the previous

example, the height of *B* is 2 (*B* – *F* – *J*).

• *Height of the tree* is the maximum height among all the nodes in the tree and *depth of*

*the tree* is the maximum depth among all the nodes in the tree. For a given tree,

depth and height returns the same value. But for individual nodes we may get

different results.

• The size of a node is the number of descendants it has including itself (the size of the

subtree *C* is 3).

• If every node in a tree has only one child (except leaf nodes) then we call such trees

*skew trees*. If every node has only left child then we call them *left skew trees*.

Similarly, if every node has only right child then we call them *right skew trees.*

* The root of a tree is the node with no parents. There can be at most one root node in

a tree (node A in the above example).

* An edge refers to the link from parent to child (all links in the figure).
* A node with no children is called leaf node (E,J,K,H and I).
* Children of same parent are called siblings (B,C,D are siblings of A, and E,F are the

siblings of B).

* A node p is an ancestor of node q if there exists a path from root to q and p appears

on the path. The node q is called a descendant of p. For example, A,C and G are the ancestors of if.

* The set of all nodes at a given depth is called the level of the tree (B, C and D are

the same level). The root node is at level zero.

* The depth of a node is the length of the path from the root to the node (depth of G is

2, A – C – G).

* The height of a node is the length of the path from that node to the deepest node. The

height of a tree is the length of the path from the root to the deepest node in the tree.

A (rooted) tree with only one node (the root) has a height of zero. In the previous example, the height of B is 2 (B – F – J).

* Height of the tree is the maximum height among all the nodes in the tree and depth of the tree is the maximum depth among all the nodes in the tree. For a given tree, depth and height returns the same value. But for individual nodes we may get different results.
* The size of a node is the number of descendants it has including itself (the size of the subtree C is 3).
* If every node in a tree has only one child (except leaf nodes) then we call such trees skew trees. If every node has only left child then we call them left skew trees. Similarly, if every node has only right child then we call them right skew trees.

Root Root Root

D

4

3

2

C

B

A

1

4

3

2

1

**Left Skew Tree Skew Tree Right Skew Tree**

**Binary Trees**

Binary tree is a tree where each node has zero child, one child or two children. Empty tree is

also a valid binary tree. We can visualize a binary tree as consisting of a root and two disjoint

binary trees, called the left and right subtrees of the root.

**Generic Binary Tree**

**Root**

Right

Subtree

Left

Subtree

**Example**

**Root**

3

4

1

2

**Types of Binary Trees**

1. **Strict Binary Tree:** A binary tree is called strict binary tree if each node has exactly two

children or no children.

5

2

4

3

1

Root

1. **Full Binary Tree:** A binary tree is called full binary tree if each node has exactly two children and all leaf nodes are at the same level.

7

6

5

3

4

2

1

Root

1. **Complete Binary Tree:** Before defining the complete binary tree, let us assume that the height of the binary tree is h. In complete binary trees, if we give numbering for the nodes by starting at the root (let us say the root node has 1) then we get a complete sequence from 1 to the number of nodes in the tree. While traversing we should give numbering for NULL pointers also. A binary tree is called complete binary tree if all leaf nodes are at height h or h – 1 and also without any missing number in the sequence.

Root

5

4

3

2

1

**Applications of Binary Trees**

Following are the some of the applications where binary trees play an important role:

* Expression trees are used in compilers.
* Huffman coding trees that are used in data compression algorithms.
* Binary Search Tree (BST), which supports search, insertion and deletion on a collection of items in O (logn) (average).
* Priority Queue (PQ), which supports search and deletion of minimum (or maximum) on a collection of items in logarithmic time (in worst case).

**Binary Tree Traversals**

The process of visiting all nodes of a tree is called tree traversal. It is when each node is processed only once but it may be visited more than once as seen in linear data structures (like linked lists, stacks, queues, etc.), the elements are visited in sequential order. But, in tree structures there are many different ways.

Tree traversal is like searching the tree, except that in traversal the goal is to move through the tree in a particular order. In addition, all nodes are processed in the traversal but searching stops when the required node is found.

**Traversal Possibilities**

Starting at the root of a binary tree, there are three main steps that can be performed and the order in which they are performed defines the traversal type. These steps are:

1. Performing an action on the current node (referred to as “visiting” the node and denoted with “D”),
2. Traversing to the left child node (denoted with “L”), and
3. Traversing to the right child node (denoted with “R”).

Based on the steps above there are 6 possibilities which are:

* **LDR**: Process left subtree, process the current node data and then process right subtree
* **LRD**: Process left subtree, process right subtree and then process the current node data.
* **DLR**: Process the current node data, process left subtree and then process right subtree.
* **DRL**: Process the current node data, process right subtree and then process left subtree.
* **RDL**: Process right subtree, process the current node data and then process left subtree
* **RLD**: Process right subtree, process left subtree and then process the current node data

The main steps of tree traversal are:

1. Preorder (DLR) Traversal
2. Inorder (LDR) Traversal
3. Postorder (LRD) Traversal

There is another traversal method which does not depend on the above orders and it is:

* Level Order Traversal: This method is inspired from Breadth First Traversal (BFS of Graph algorithms).

NOTE: The diagram below will be used to solve explain the 3 main steps of tree traversal.

2

7

6

4

5

3

1

Root

**PreOrder Traversal**

In the example above, 1 is processed first, then the left subtree, and this is followed by the right subtree. Therefore, processing must return to the right subtree after finishing the processing of the left subtree. To move to the right subtree after processing the left subtree, we must maintain the root information.

Preorder traversal is defined as follows:

* Visit the root.
* Traverse the left subtree in Preorder.
* Traverse the right subtree in Preorder.

The nodes of tree would be visited in the order: 1 2 4 5 3 6 7

**InOrder Traversal**

In Inorder Traversal the root is visited between the subtrees. Inorder traversal is defined as follows:

* Traverse the left subtree in Inorder.
* Visit the root.
* Traverse the right subtree in Inorder.

The nodes of tree would be visited in the order: 4 2 5 1 6 3 7

**PostOrder Traversal**

In postorder traversal, the root is visited after both subtrees. Postorder traversal is defined as follows:

* Traverse the left subtree in Postorder.
* Traverse the right subtree in Postorder.
* Visit the root.

The nodes of the tree would be visited in the order: 4 5 2 6 7 3 1

**Level Order Traversal**

Level order traversal is defined as follows:

* Visit the root.
* While traversing level ( keep all the elements at level ( + 1 in queue.
* Go to the next level and visit all the nodes at that level.

• Repeat this until all levels are completed.

The nodes of the tree are visited in the order: 1 2 3 4 5 6 7