

Trees

Q: Why study trees?

A: Many advance ADTs are implemented using tree-based data structures.

Recursive Definition of (Rooted) Tree:

Let T be a set with $n \geq 0$ elements.

- (i) If $n = 0$, T is an *empty tree*,
- (ii) If $n > 0$, then there exists a distinct element r , called the *root*, such that $T - \{r\}$ can be partitioned into k disjoint subsets T_1, T_2, \dots, T_k , $k \geq 0$, such that each of these subsets also forms a tree.

Basic Concepts:

Elements of T are *nodes* in the tree T .

Each subset T_i is a *subtree of r* .

The roots of the subtrees of r are *children of r* .

The root r is the *parent* of the roots of its subtrees.

Nodes with the same parent are *siblings*.

Node, except the root, with no children is *leaf*.

Degree of a node is the number of children of the node.

A *path of length k* (in a tree) from node x to node y is a sequence of $k+1$ nodes $x = n_0, n_1, n_2, \dots, n_k = y$ such that n_i is the parent of n_{i+1} for all $i = 0, 1, \dots, k-1$.

If there is a path from x to y , then x is an *ancestor* of y and y is a *descendant* of x .

For any given node x , the *depth (level)* of x is the length of the (unique) path from the root to x , and the *height* of x is the length of a longest path from x to any leaf.

The height of a tree is the height of its root.

The depth of a tree is the maximum depth of its nodes.

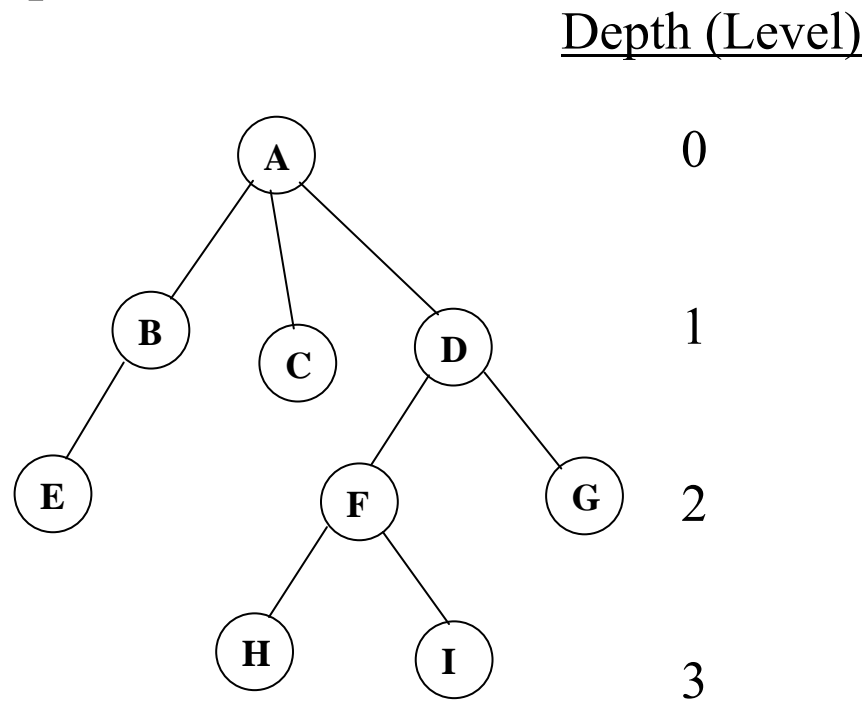
Remarks:

1. There is a unique path from the root to each node in the tree.
2. There is a path of zero length from any node to itself.
3. The depth (level) of the root of a tree is 0.
4. The height of a leaf is 0.
5. Height of tree = Depth of tree.
6. For convenience, the height of an empty tree is defined to be -1 .

Warning:

Some authors define the depth, or level, of the root to be 1.

Graphical Representation of Tree:



A is the *root* of tree T,

D is the *parent* of F and G,

F and G are children of D,

Degree of F is 2,

B, C, D are *siblings*,

C, E, G, H, I are *leaves*,

D is an *ancestor* of I and I is a *descendant* of D,

(A,D,F,H) is a *path* of length 3,

H is of *height* 0 but *depth* 3.

Height (depth) of the tree is 3.

Some Important Classes of Trees:

1. k-ary tree, $k \geq 2$:

Each node has at most k children, $k \geq 2$.

2. Binary tree:

An ordered k-ary tree when $k = 2$.

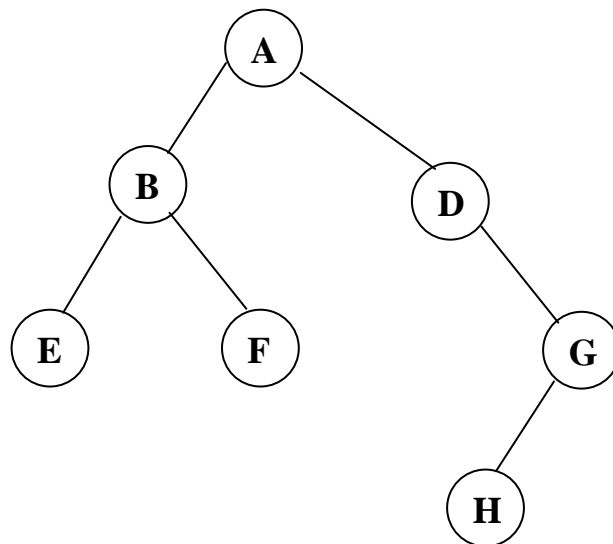
Recursive Definition of Binary Tree:

Let T be a set with $n \geq 0$ elements. T is a binary tree iff

(i) T is empty, or

(ii) if T is not empty, T has a root r such that $T - \{r\}$ can be partitioned into two disjoint binary trees T_L and T_R , called the *left subtree* and *right subtree* of r .

Example: A binary tree.



3. Complete binary tree:

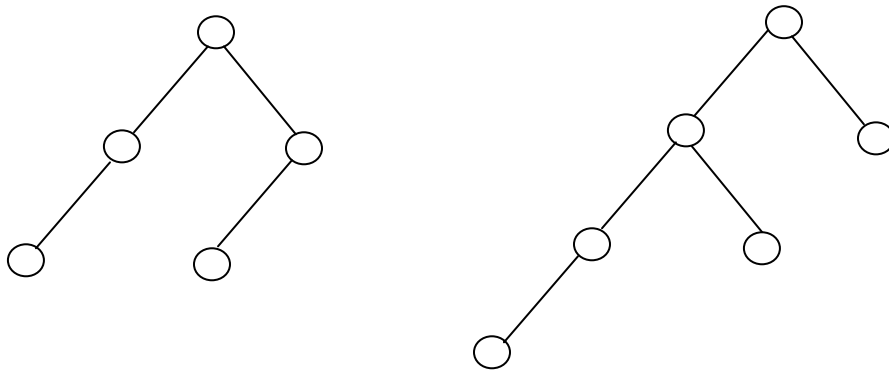
A complete binary tree T with height h is a binary tree such that

- (1) Each node on levels 0 to $h-2$ must have exactly two children,
- (2) Leaves can only be found on the two highest levels (levels $h-1$ and h), and
- (2) Leaves on level h are left-justified.

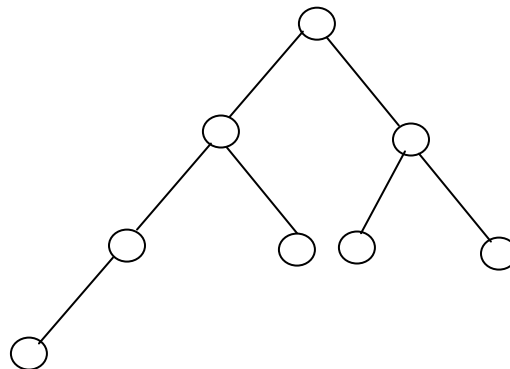
Warning: Some authors do not require Condition (3).

Examples:

(a) Incomplete binary tree:



(b) Complete binary tree:

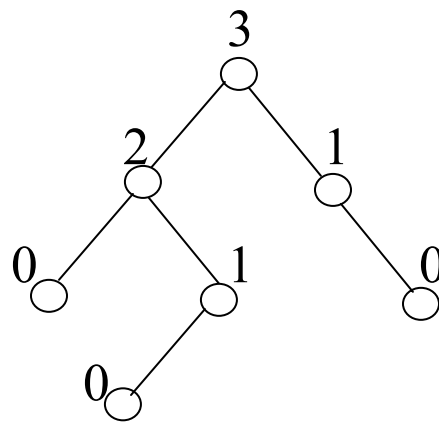


4. Full binary tree:

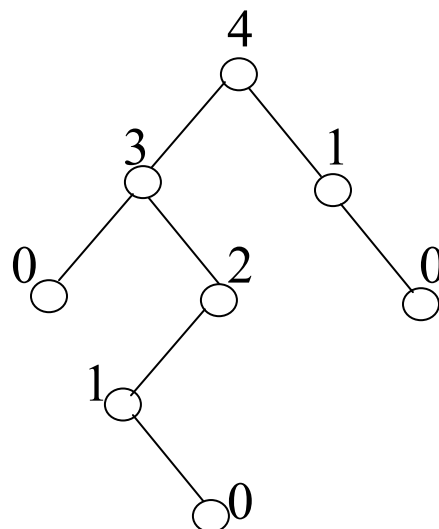
tree, the height of its left subtree differs by no more than one from the height of its right subtree. Hence, $|h(T_L(x)) - h(T_R(x))| \leq 1$, where $h(T_L(x))$, and $h(T_R(x))$, are the height of the left, and right, subtree rooted at x , respectively.

Examples:

Balanced binary tree showing height of nodes:



Not a balanced binary tree showing height of nodes:



Observation:

$\{\text{full trees}\} \subset \{\text{complete binary trees}\} \subset \{\text{balanced binary trees}\}$

When implementing an ADT using a (binary) tree, either non-leaf or leaf-nodes can be used to hold the data objects of the ADT and the performance of an ADT operation will usually depend on the height of the tree.

Nodes, leaves, and heights of binary trees:

Given a non-empty binary tree T.

1. T with height h , $h \geq 0$:
Min # nodes = $h+1$. Max # nodes = $2^{h+1} - 1$.
2. T with height h , $h \geq 0$:
Min # leaves = 1. Max # leaves = 2^h .
3. T with n nodes, $n \geq 1$:
Min height = $\lfloor \log_2 n \rfloor$. Max height = $n-1$.
4. T with n nodes, $n \geq 1$:
Min # leaves = 1. Max # leaves = $\lceil n/2 \rceil$.
5. T with m leaves, $m \geq 1$:
Min height = $\lceil \log_2 m \rceil$. Max height = ∞ .
6. T with m leaves, $m \geq 1$:
Min # nodes = $2m-1$. Max # nodes = ∞ .

Traversing a Binary Tree:

1. To explore the underlying hierarchical structures.
2. To retrieve information stored in nodes.
3. To store and restore the topological structure of a tree (relation).

Basic Binary Tree Traversal Algorithms:

1. *Preorder traversal:*

Traverse/retrieve root,
Traverse left subtree recursively in preorder,
Traverse right subtree recursively in preorder.

2. *Postorder traversal:*

Traverse left subtree recursively in postorder,
Traverse right subtree recursively in postorder,
Traverse/retrieve root.

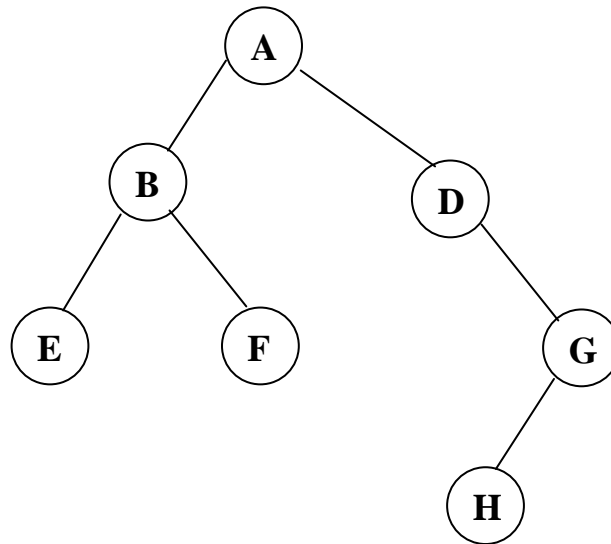
3. *Inorder traversal:*

Traverse left subtree recursively in inorder,
Traverse/retrieve root,
Traverse right subtree recursively in inorder.

4. *Level-order traversal:*

Starting at level 0, traverse the nodes on each level from left to right level by level.

Example: Binary tree traversals.



Preorder: A B E F D G H

Postorder: E F B H G D A

Inorder: E B F A D H G

Level-order: A B D E F G H

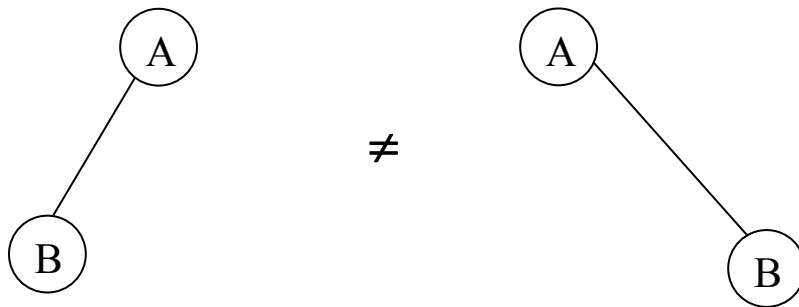
More on binary tree traversals:

Observe that for a given binary tree T, it is easy to traverse T.

Q: If the traversal of a binary tree T is given, can we reconstruct the binary tree T?

No!

Consider the following binary trees:



Observe that preorder, postorder, and level-order traversals for both trees are given by AB, BA, and AB, respectively. Hence, these two binary trees are indistinguishable if only one of the above tree traversals is given!

Q: How about inorder traversal?

Consider the following binary trees:



Again, these two binary trees are indistinguishable if only their inorder traversal is given!

Q: Can we reconstruct the binary tree T from its traversal(s)?

Must know the root, the left, and the right subtrees.

If we are given any one of the following pairs of traversals of T, T can be reconstructed if exists.

1. Preorder and inorder traversals.
2. Postorder and inorder traversals.
3. Level-order and inorder traversals.

Remark: Preorder and postorder, level order and preorder, and level-order and postorder traversals will not work!

HW. Can you reconstruct a binary tree T having the following pairs of traversals? Algorithms?

Preorder: B A D C E F G I H
Inorder: A F E G C D I B H

Preorder: A C D H I E B F G
Inorder: D I H C E A B G F

Preorder: A C D H I E B F G
LevelOrder: A C B D E F H G I

Postorder: A K J G B D H F I E C
Inorder: A D K G J B C F H E I

Consider given a pair of preorder and inorder traversals of a binary tree T.

Q: How do we reconstruct T if exists?

Scan preorder traversal from left to right to determine root followed by scanning inorder traversal to determine left and right subtree.

Example: Let preorder = ABDEC and inorder = BEDAC.

