Data Structure & Algorithms

Lec 09: Introduction to Trees & Graphs

Fall 2017 - University of Windsor
Dr. Sherif Saad

Agenda

1. Introduction to Graphs and Trees
2. Trees Data Structure
3. Binary Trees
4. Binary Trees as ADTs

Learning Outcome

By the end of this class, you should be able to

- Explain the difference between trees and graphs
- Identify if a given problem requires a tree or a graph data structure
- Describe the properties of trees and binary trees
- Explain the different methods to traverse a binary tree structure

Nonlinear Data Structures

The elements in nonlinear data structures are not organized sequentially.

An element in a nonlinear data structure could be directly connected to more than one elements to represent special relationships between them.

Because of the nonlinear structure, it might be difficult to implement nonlinear data structures using contiguous memory allocation.

Non-linear data structures are graphs and trees.

Graphs and Trees

Graphs and trees are fundamental data structures and key building blocks in many applications.

Few Examples of Graphs and Trees Applications:

- Databases
- Network Security (attack trees, attack graphs)
- Search Engines
- Social Media
- Navigations (find shortest routes)
- Biochemistry
- Compilers

Graphs and Trees

Graphs and Trees are efficient methods to represent relationships between pairs of elements.

A Tree is restricted form of a graph minimally connected graph and no cycles, having only one path between any two nodes.

Graph is a more general concept (structure) than tree. That can represent any mathematical relation (Cartesian product of two sets).

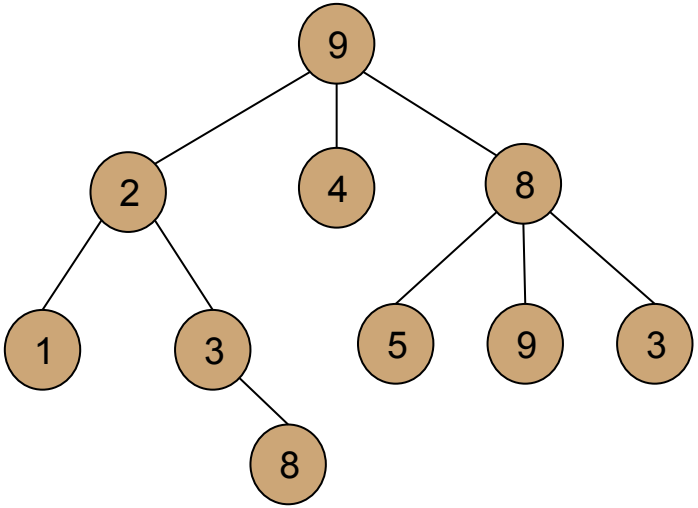
Graphs and trees are very efficient data structures in solving many problems.

When should we use a graph or tree

1. Does the problem require storing and maintaining a set or elements?
2. How many different relationships exist between the elements?
3. Is there any nonlinear relationship between the elements?
4. Does the problem require you to work with grid, network or similar structure?
5. Does the problem require you to find connections, paths, etc?

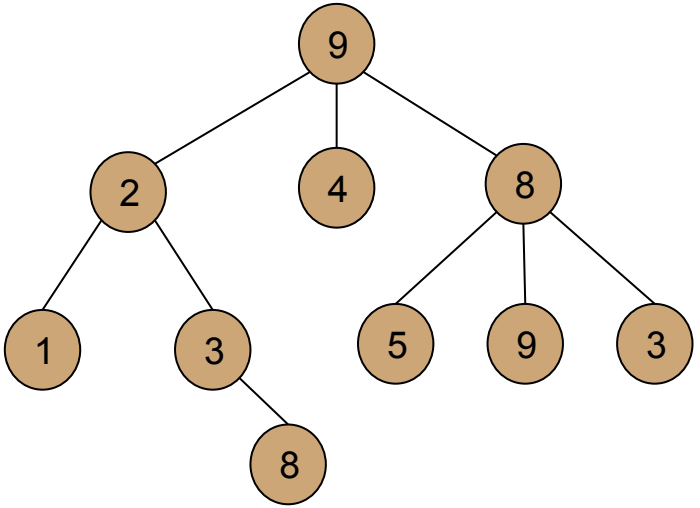
Trees Data Structure

1. A tree is a nonlinear data structure where each element point to one or more other elements in the tree.
2. Elements stored in a tree in a hierarchical structure.
3. There is only one path between any two nodes.
4. There is no cycle (loop) in a tree structure. The tree is not connected if any single edge removed from it

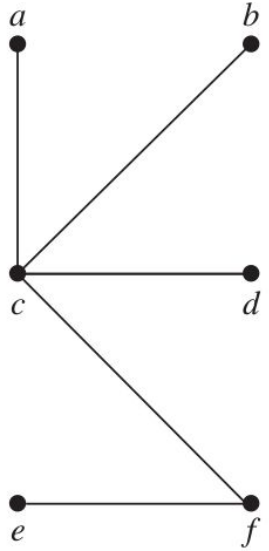


Trees Data Structure

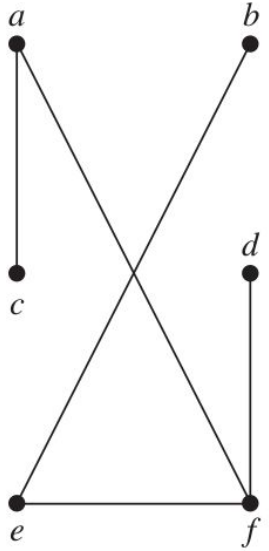
5. In Graph Theory: Tree is undirected connected graph where any two vertices are connected by exactly one path.
6. A graph is connected when there is a path between every pair of vertices (node).
7. The edges (links) in undirected graph have no orientation. The edges do not specify directions



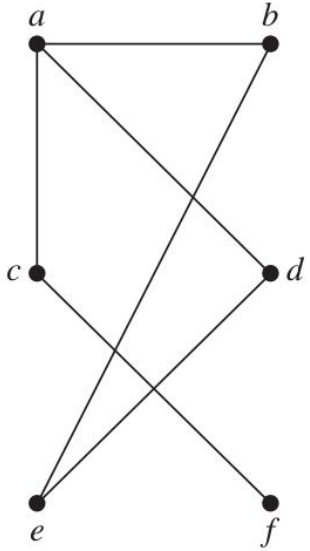
Tree or not a Tree



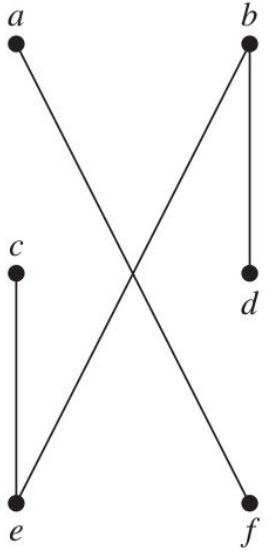
G_1



G_2



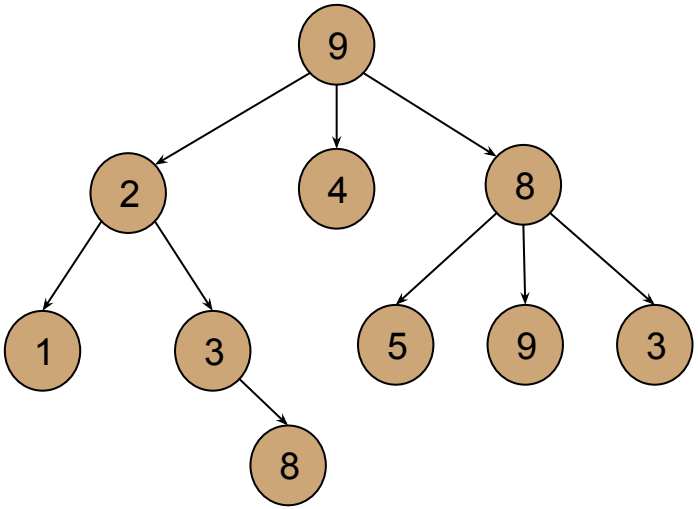
G_3



G_4

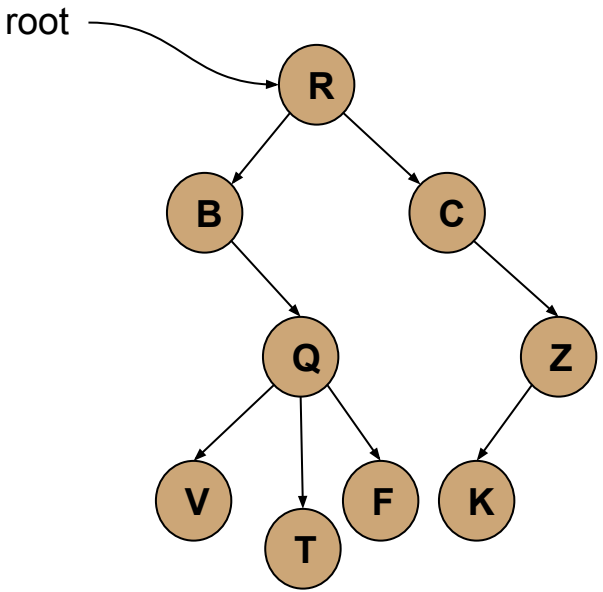
Trees Data Structure

1. In data structure we use the term tree to refer to rooted trees.
2. A rooted tree may be directed, called a directed rooted tree. In a directed rooted tree either all the edges point away from the root or all the edges point towards the root (in-tree |parent pointer tree).
3. The size of the tree is the number of nodes in the tree.



Trees Data Structure

1. The root node has no parents (R is the root node)
2. An edge is a link from a parent node to a child node
3. A node with no children is called a leaf node. (e.g V, T F, and K are leaf nodes)
4. A node with a parent and at least one child is called inner node. (B , C, Q, and Z are inner nodes)



Tree Data Structure

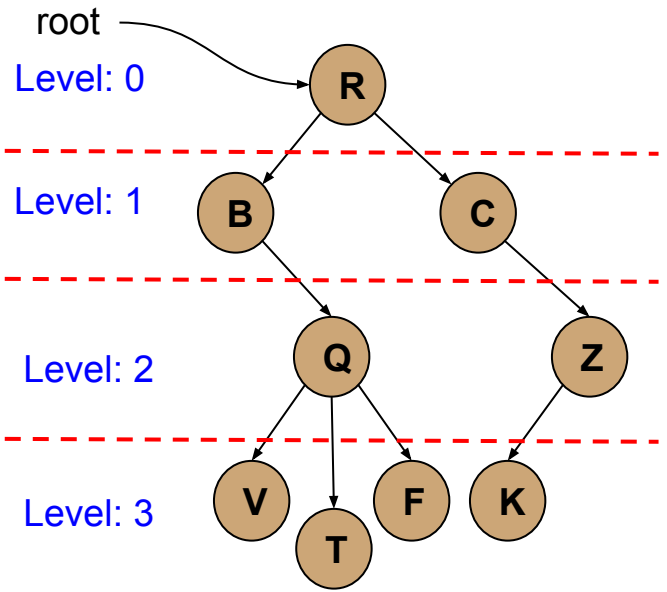
Trees Data Structure

The depth of a node is the number of edges from the root node to that node. (The depth of node Q is 2. The depth of R is 0)

The set of node at the same depth is called the level of the tree. (at level 1 we have B and C)

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

The height of the tree is the height of the root node (longest path from the root to the deepest leaf node)

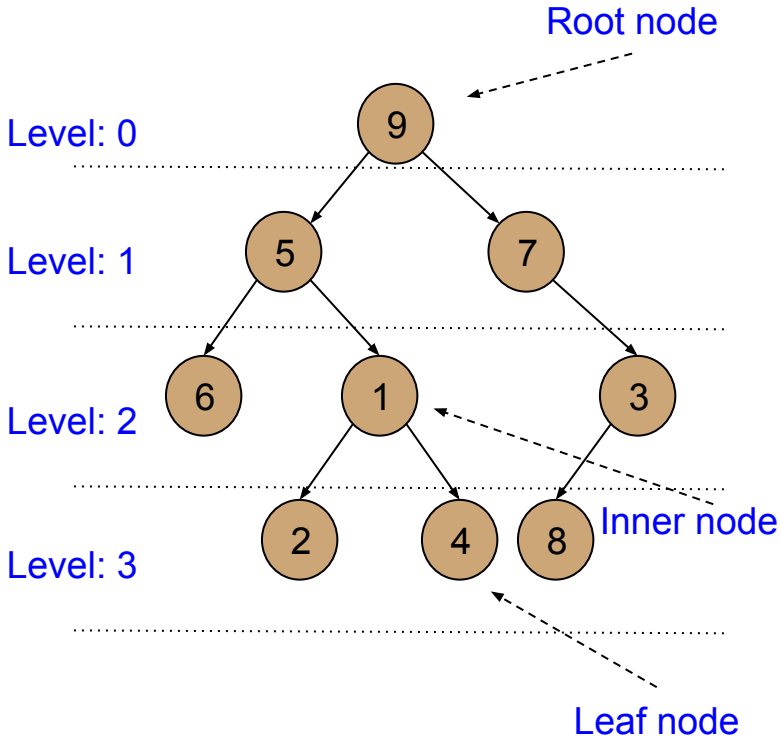


Tree Data Structure

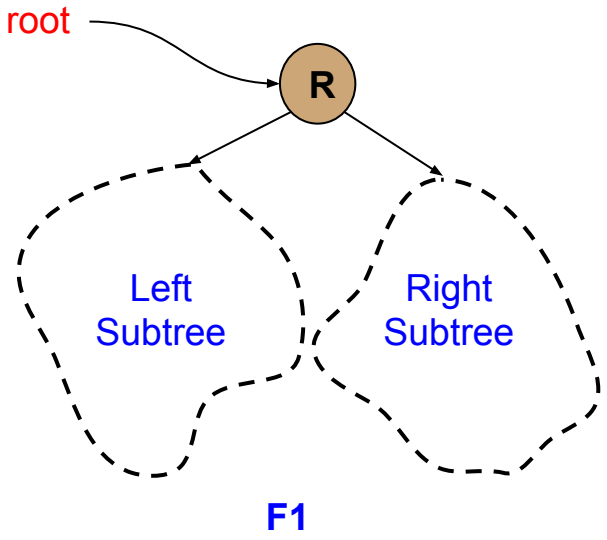
Binary Trees Data Structure

What is a binary tree?

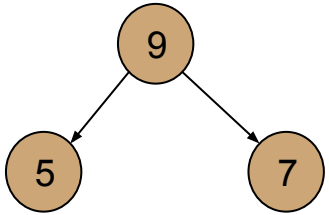
- Is a tree data structure
- Each node (vertex) has at most two children (left & right)
- An empty tree is a valid binary tree.
- A binary tree has a root node and two disjoint binary trees called the left and the right subtrees of the root.



Binary Trees Data Structure

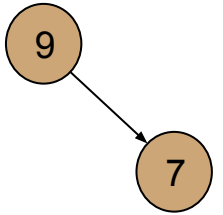


F1

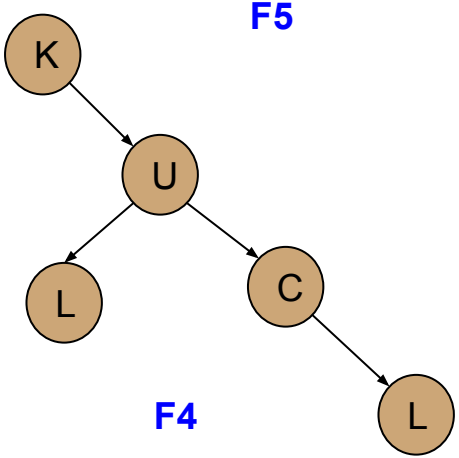


F2

F3



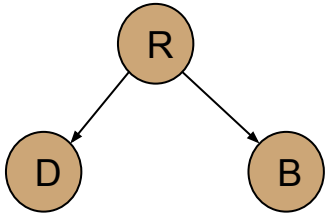
F5



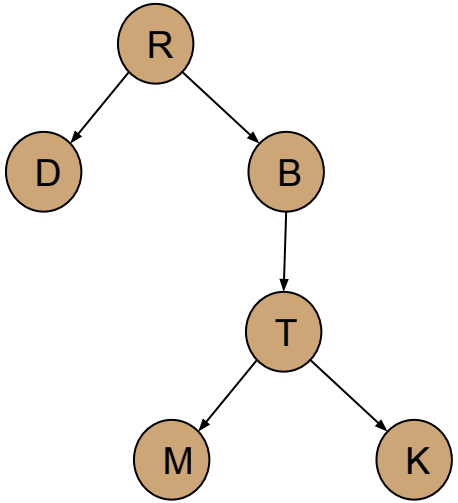
F4

Types of Binary Trees

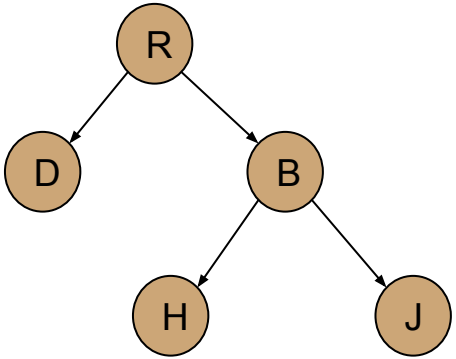
Strict Binary Tree: each node has 0 children or exactly two children



F1



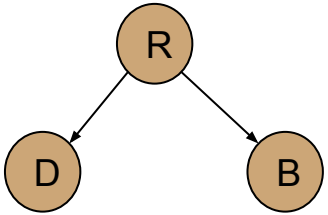
F2



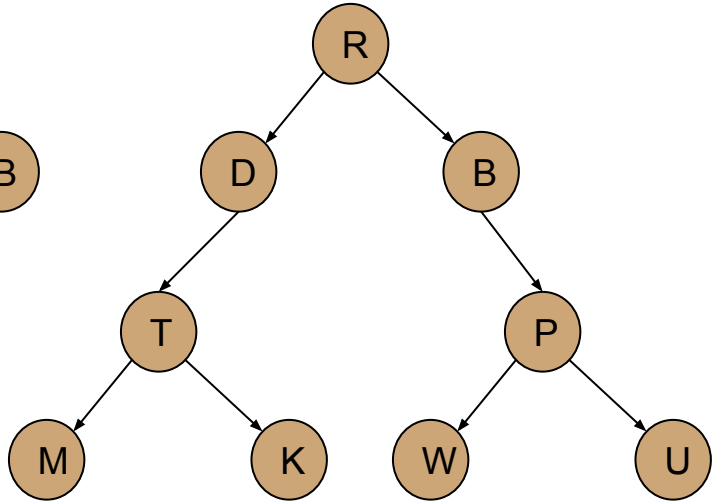
F3

Types of Binary Trees

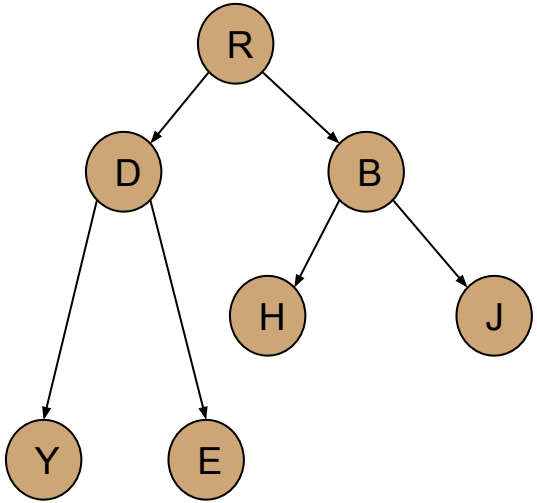
Full Binary Tree: all non leaf nodes have exactly two children and on the same level



F1



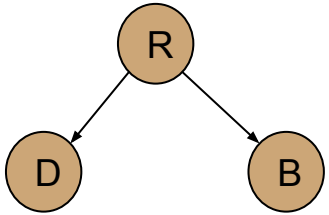
F2



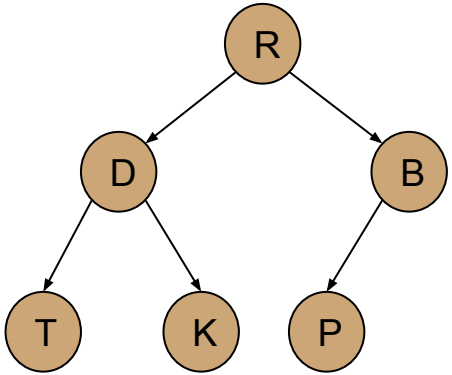
F3

Types of Binary Trees

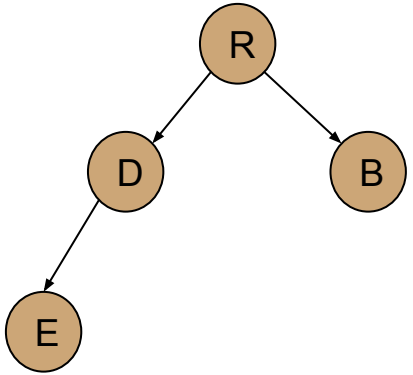
Complete Binary Tree: is a binary tree in which every level, except possibly the last, is filled, and all nodes are as far left as possible.



F1



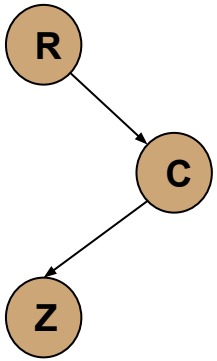
F2



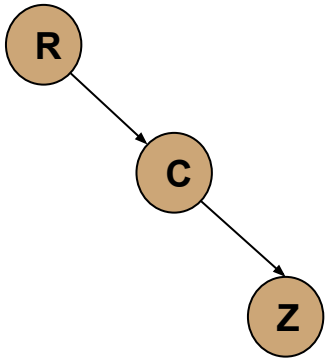
F3

Types of Binary Trees

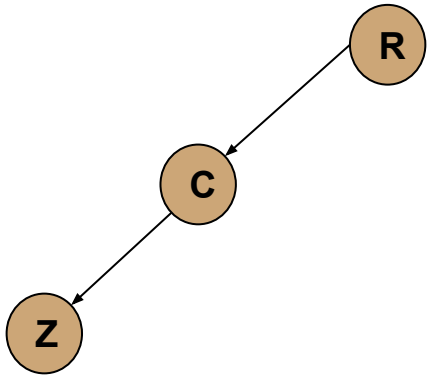
Skew Binary Tree: every node in the tree has one child only



F1



F2

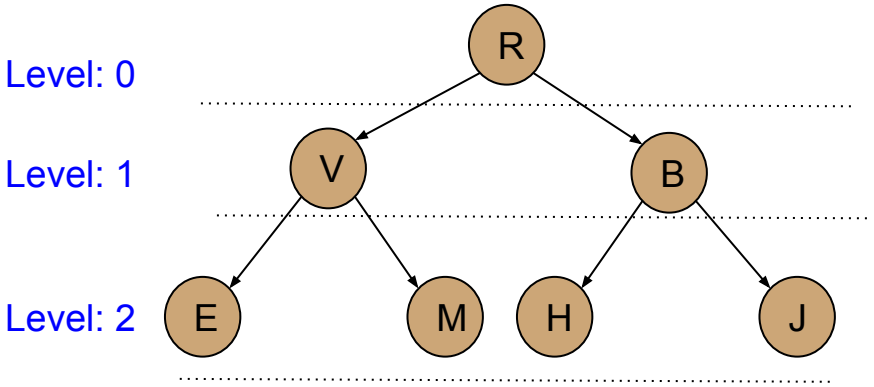


F3

Properties of Binary Trees

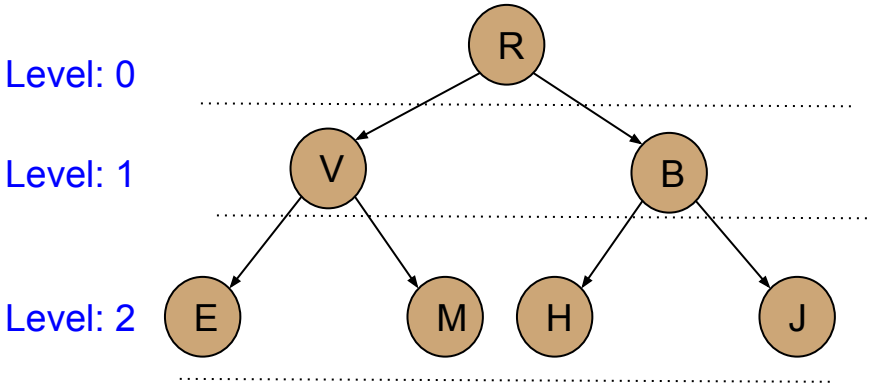
Could you guess the relation between the height of the tree and the number of nodes in a full binary tree?

What is the height of the tree on the right?



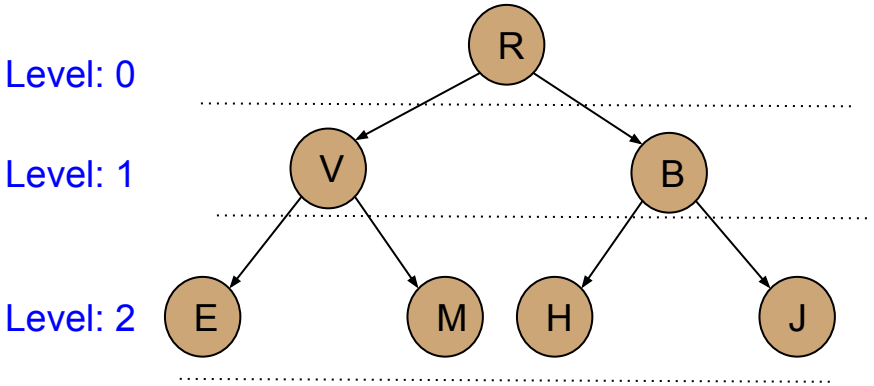
Properties of Binary Trees

1. The number of nodes ***n*** in a full binary tree is $2^{h+1}-1$ where ***h*** is the height of the tree.
2. The number of leaf nodes in full binary tree is 2^{h}
3. The number of nodes in a complete binary tree is between 2^{h} (minimum) and $2^{h+1} - 1$ (maximum)



Properties of Binary Trees

1. The maximum number of nodes at level L of a binary tree is 2^L
2. In a Binary Tree with n nodes, minimum possible height or minimum number of levels is $Log_2(n+1)$
3. The number of leaf nodes is always one more than nodes with two children.



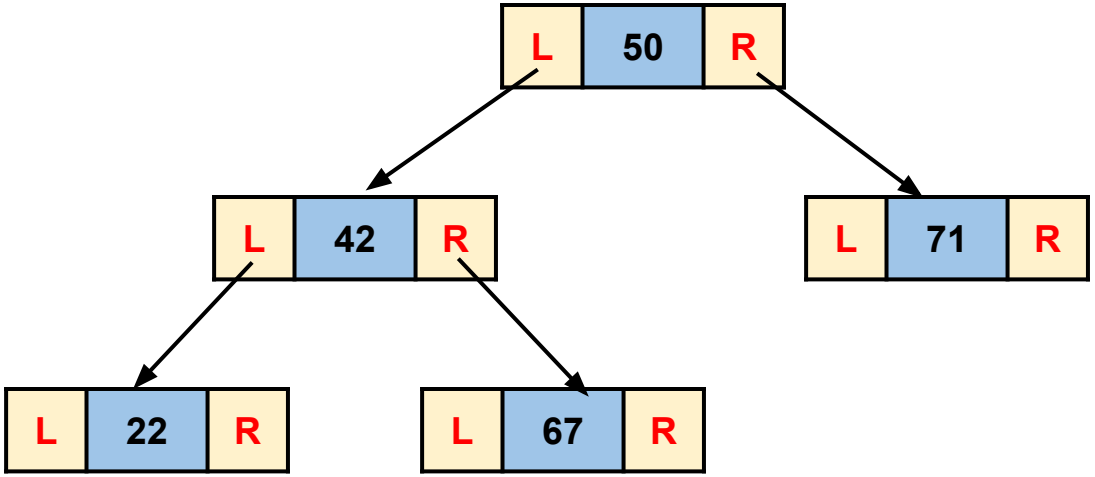
Binary Trees as ADTs

A logical data storage in a hierarchical structure with the following operations:

1. Inserting an element into a tree
2. Deleting an element from a tree
3. Search for an element
4. Traversing the tree
5. Finding the size of the tree
6. Finding the height of the tree
7. Finding the level with the maximum number of nodes.
8. Finding the least common ancestor (LCA) for any given pair of nodes.

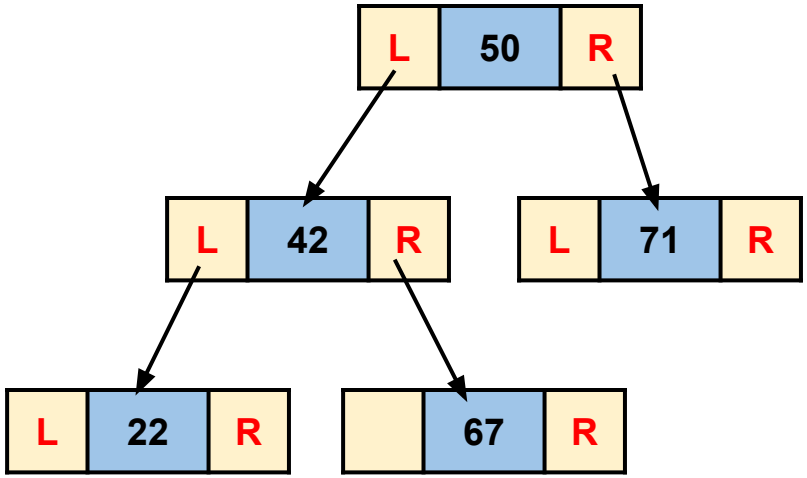
Implementing a Binary Tree

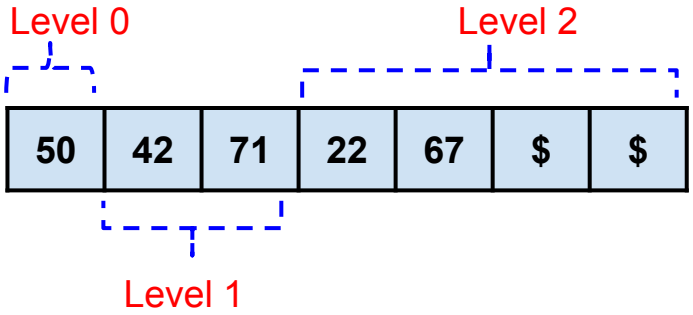
One common implementation of binary trees uses nodes. Where each nodes contains two pointers left and right pointer



Implementing a Binary Tree

We could also use one dimensional array to implement a binary tree





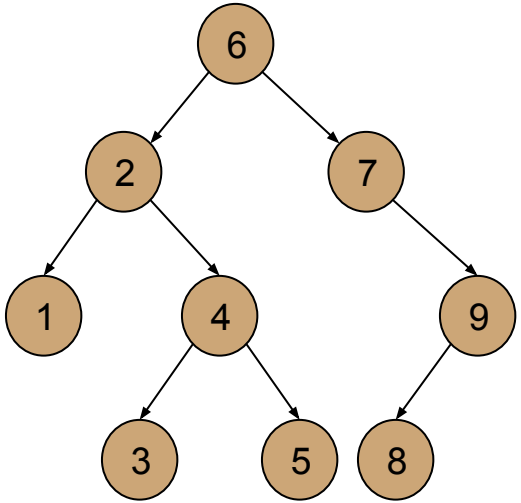
Binary Tree Node Implementation

```
 8 struct BinaryTreeNode {
 9
10     int data;
11     struct BinaryTreeNode * left;
12     struct BinaryTreeNode * right;
13
14 }
```

How could we do better?

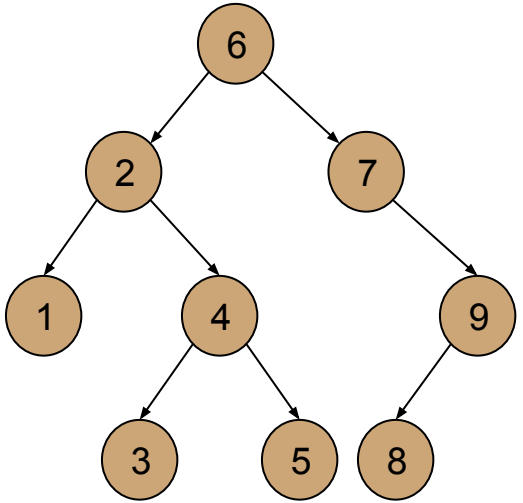
Inserting Node in Binary Tree

```
If root is NULL
   then create root node
return

If root exists then
   compare the data with node.data

   while until insertion position is located

      If data is greater than node.data
         goto right subtree
      else
         goto left subtree

   end while

   insert data

end If
```



Searching a Binary Tree

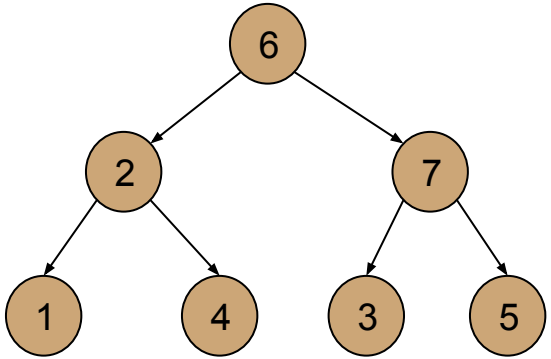
```
1  If root.data is equal to search.data
2     return root
3  else
4     while data not found
5
6        If data is greater than node.data
7           goto right subtree
8        else
9           goto left subtree
10
11       If data found
12          return node
13
14    endwhile
15
16    return data not found
17
18 end if
```



Traversing a Binary Tree

There are three ways to traverse (visit every node in) the tree:

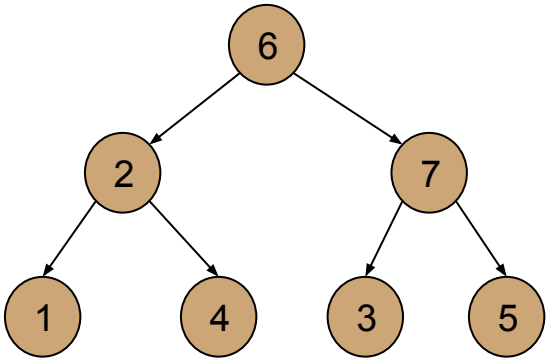
- In-order Traversal
- Pre-order Traversal
- Post-order Traversal



Traversing a Binary Tree

In-Order Traversal: the left subtree is visited first, then the root and later the right subtree.

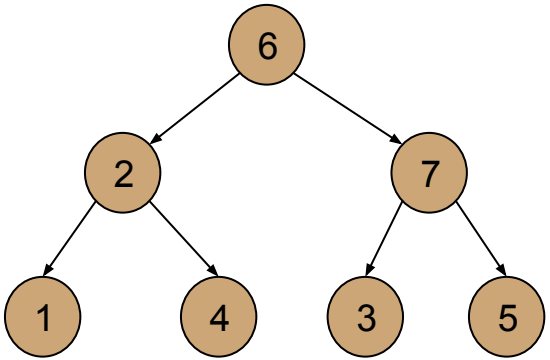
Every subtree is a binary tree and every node may represent a subtree itself



Traversing a Binary Tree

In-Order Traversal: the left subtree is visited first, then the root and later the right subtree.

1, 2, 4, 6, 3, 7, 5

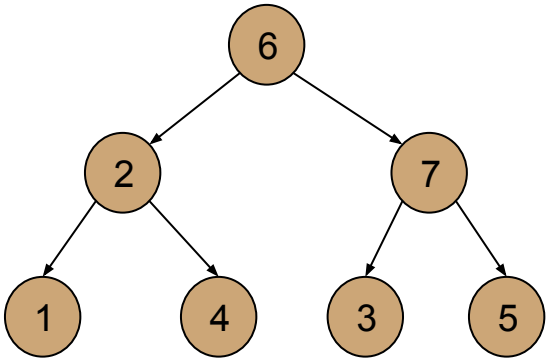


Every subtree is a binary tree and every node may represent a subtree itself

Traversing a Binary Tree

Pre-Order Traversal: the root node is visited first, then the left subtree and the right subtree

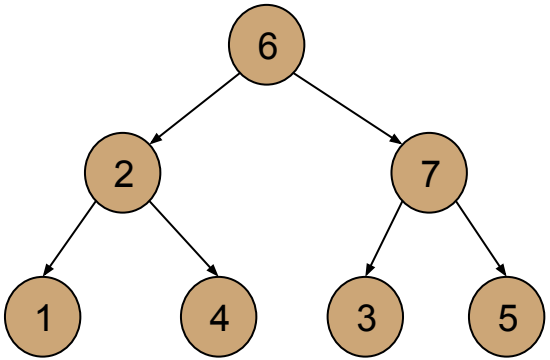
Every subtree is a binary tree and every node may represent a subtree itself



Traversing a Binary Tree

Pre-Order Traversal: the root node is visited first, then the left subtree and the right subtree

6, 2, 1, 4, 7, 3, 5

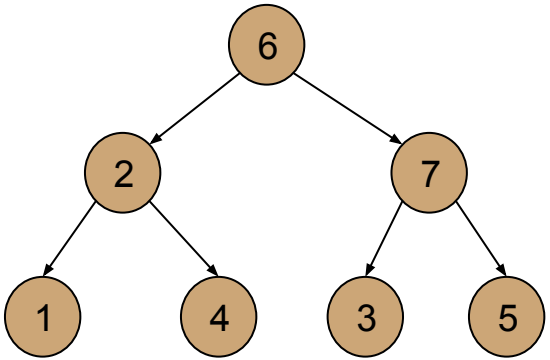


Every subtree is a binary tree and every node may represent a subtree itself

Traversing a Binary Tree

Post-Order Traversal: the left subtree is visited first, then the right subtree and finally the root

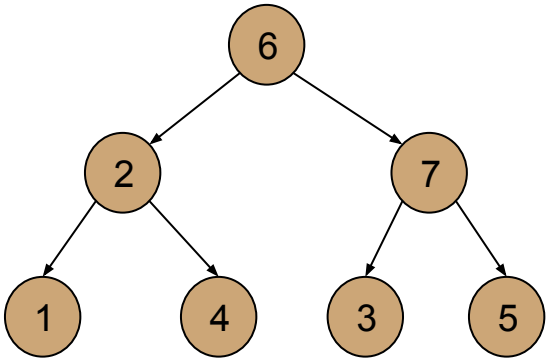
Every subtree is a binary tree and every node may represent a subtree itself



Traversing a Binary Tree

Pre-Order Traversal: the left subtree is visited first, then the right subtree and finally the root

1, 4, 2, 3, 5, 7, 6



Every subtree is a binary tree and every node may represent a subtree itself

Self-Assessment

Which of the following statements hold true for binary trees?

1. The number of nodes on the last level is always equal to the sum of the number of nodes on all other levels plus 1
2. In a full binary tree with n nodes about half of the nodes on the last level.
3. In a complete binary tree the number of total nodes on each level almost doubles as we move down the tree.