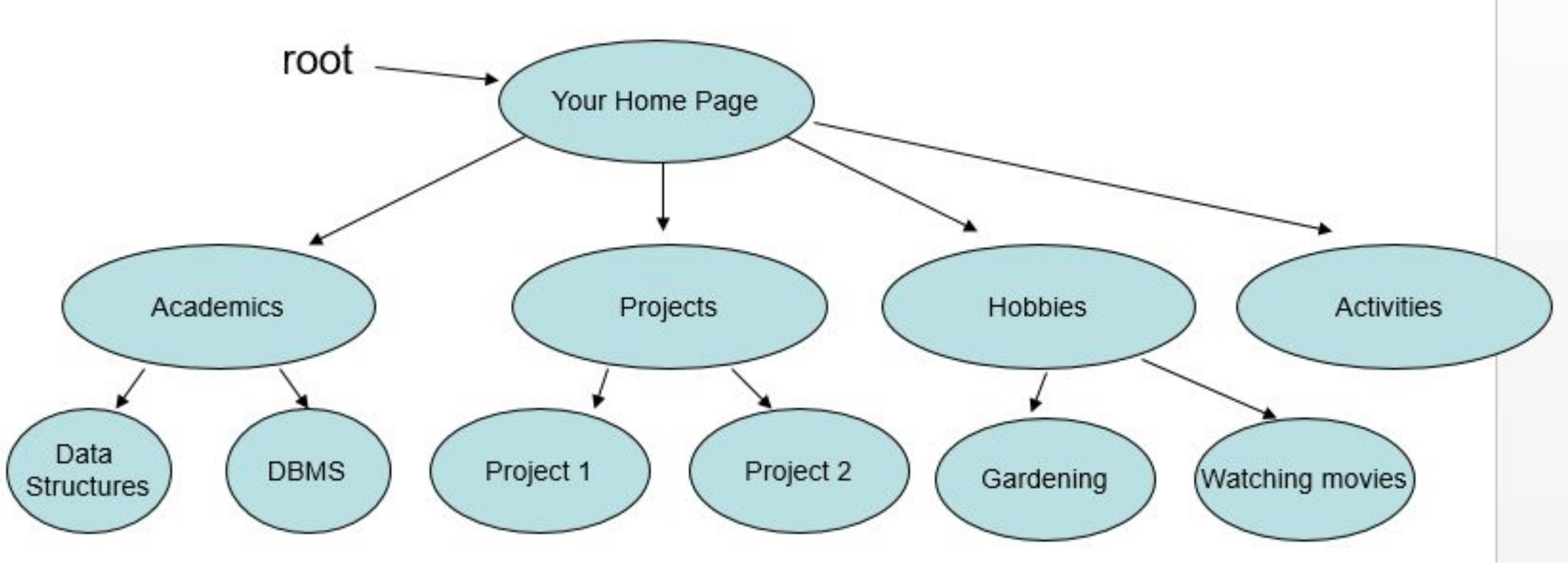
Non Linear Data Structures

Trees

- Hierarchical Collection
- Partitioning of set into disjoint sets
- Examples
  - File Directory Structure
  - Organization Chart
  - Moves in a game
  - Classification Hierarchies

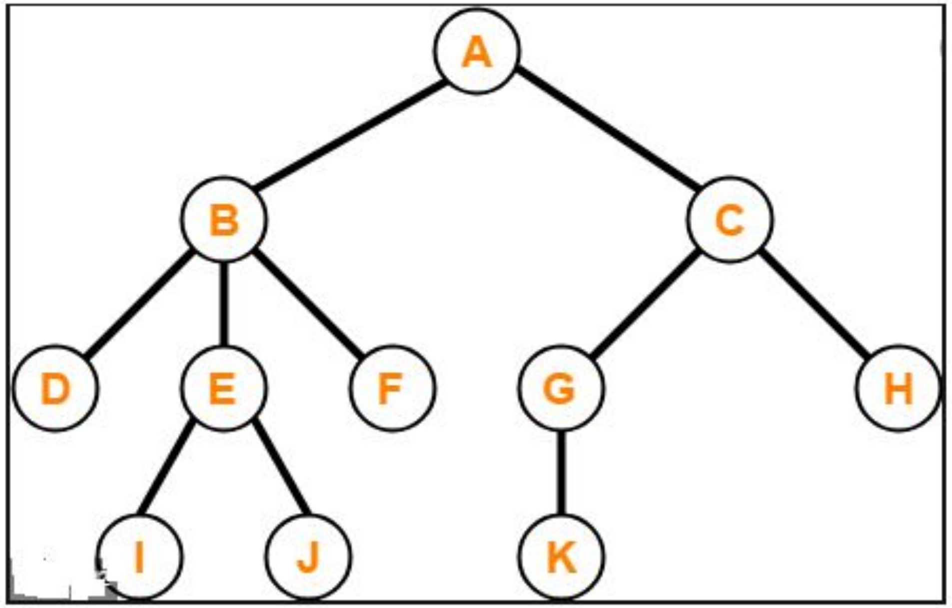
Tree Representation



Definition

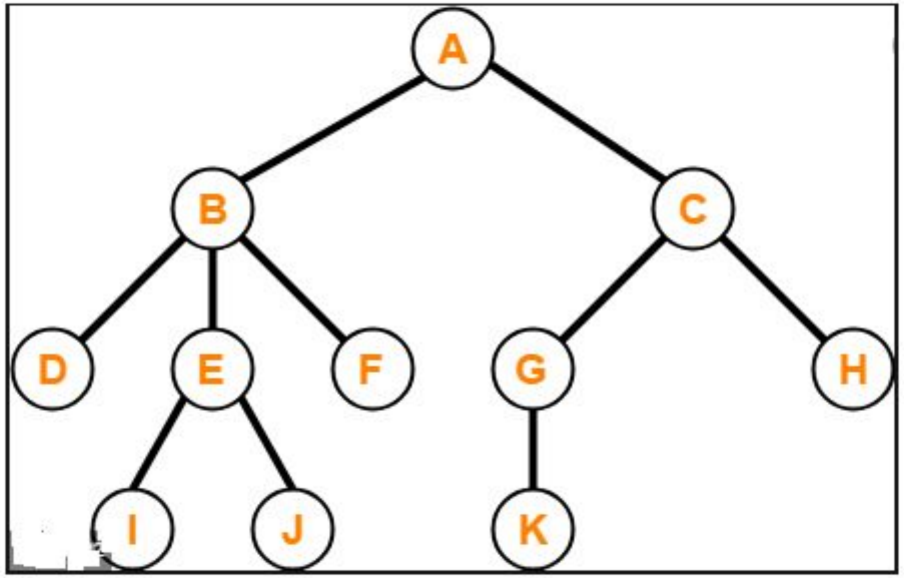
- A tree is a set of nodes that is
  a. an empty set of nodes, or
  b. has one node called the root from which zero or more trees (subtrees) descend.

- A tree is a set of nodes that is
  a. an empty set of nodes, or
  b. one node is designated as Root and remaining elements are partitioned into disjoint sets each of which are trees (subtrees)



{ A, B, C, D, E, F, G, H, I, J, K }
{A}, { B, D, E, F, I, J } { C, G, H, K }
{A}, { B } { D } { E, I, J } { F }{ C } {G, H} { K }
{A}, { B } { D } { E} { I } { J } { F }{ C } {G } {H} { K }

Terminologies



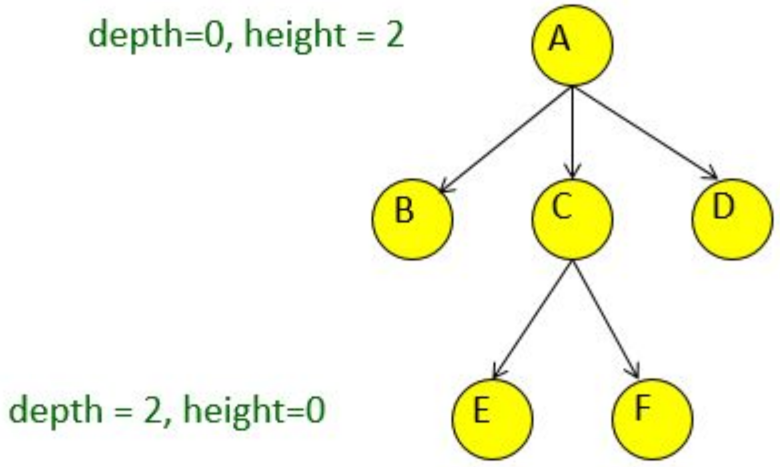
- A tree is a collection of elements (nodes)
- Each node may have 0 or more successors
  - (Unlike a list, which has 0 or 1 successor)
- Each node has exactly one predecessor
  - Except the starting / top node, called the root
- Links from node to its successors are called branches
- Successors of a node are called its children
- Predecessor of a node is called its parent
- Nodes with same parent are siblings
- Nodes with no children are called leaves

Quiz

- A tree with N nodes always has ___ edges
- Two nodes in a tree have at most how many paths between them?

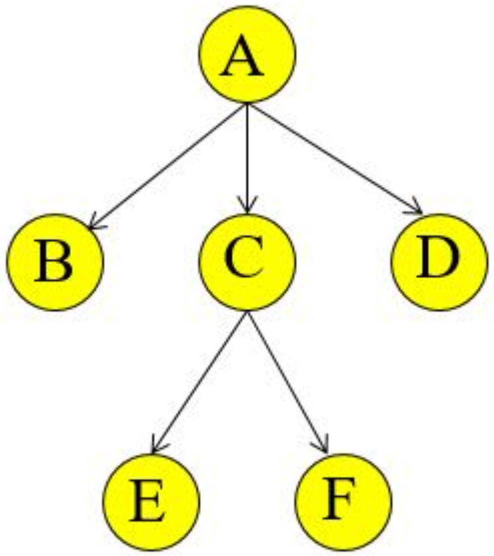
Tree

- Length of a path = number of edges
- Depth of a node N = length of path from root to N
- Height of node N = length of longest path from N to a leaf
- Height of tree = height of root



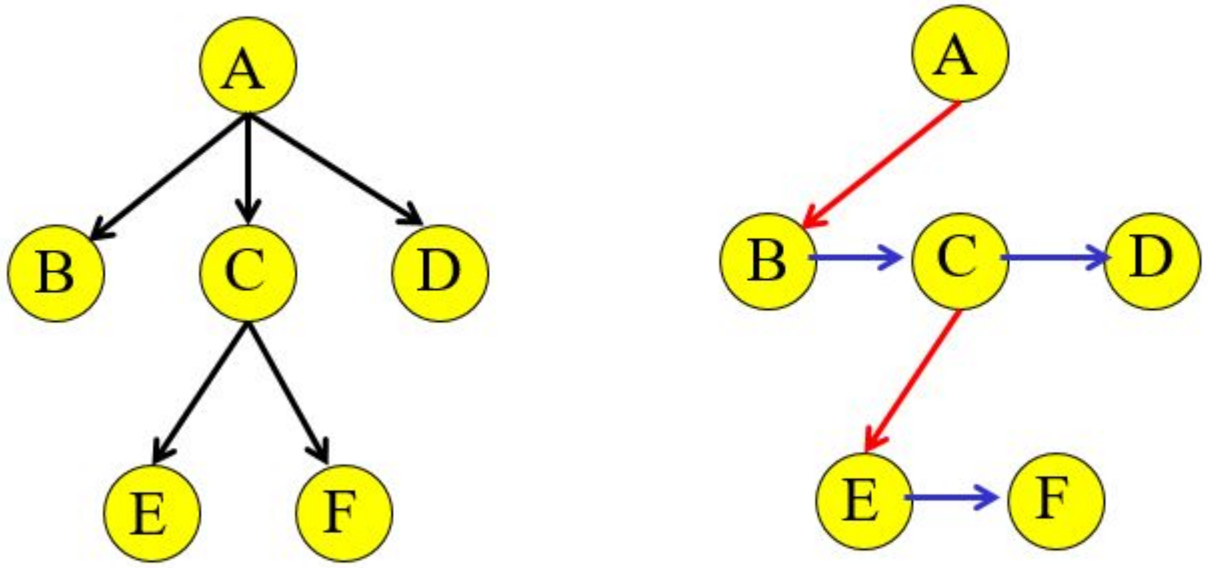
Implementation

Pointer-Based Implementation: Node with value and pointers to children



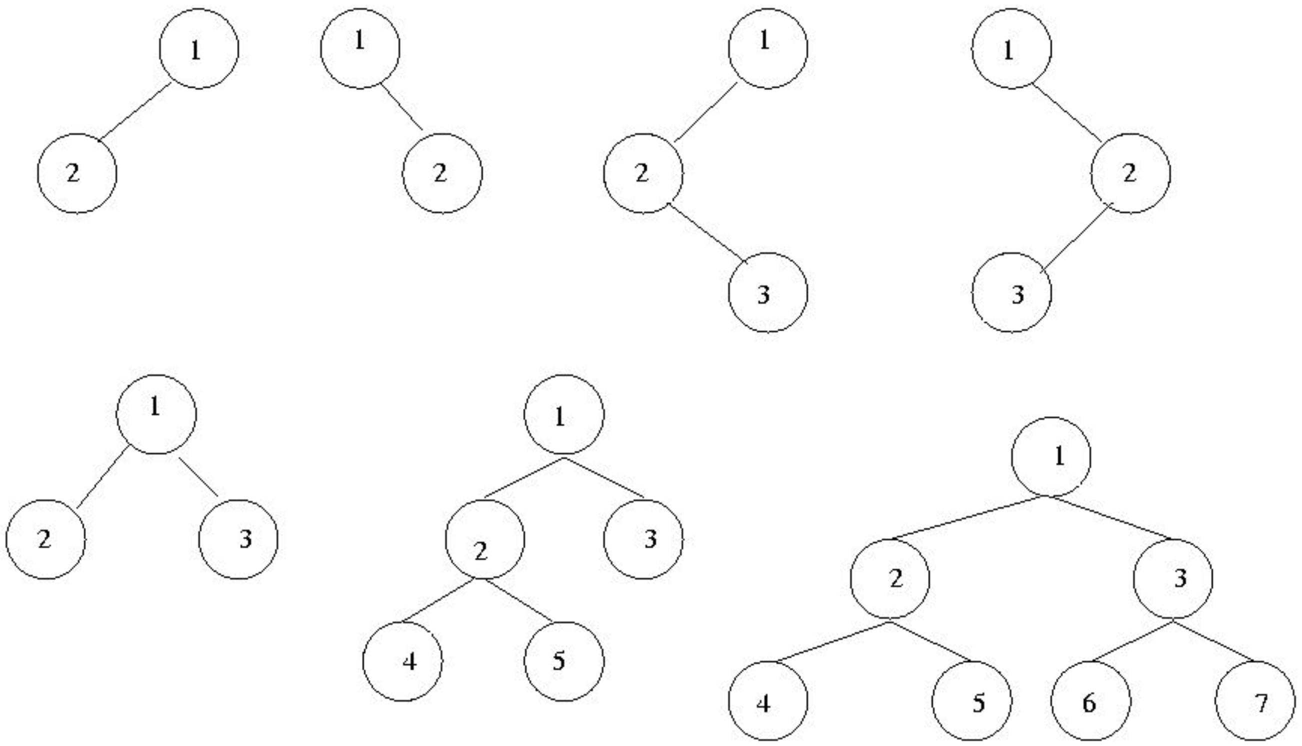
Child Sibling Representation

- Each node has 2 pointers: one to its first child and one to next sibling



Binary Trees

- Trees with number of children limited to maximum of 2
- Root. Left Subtree and Right Subtree

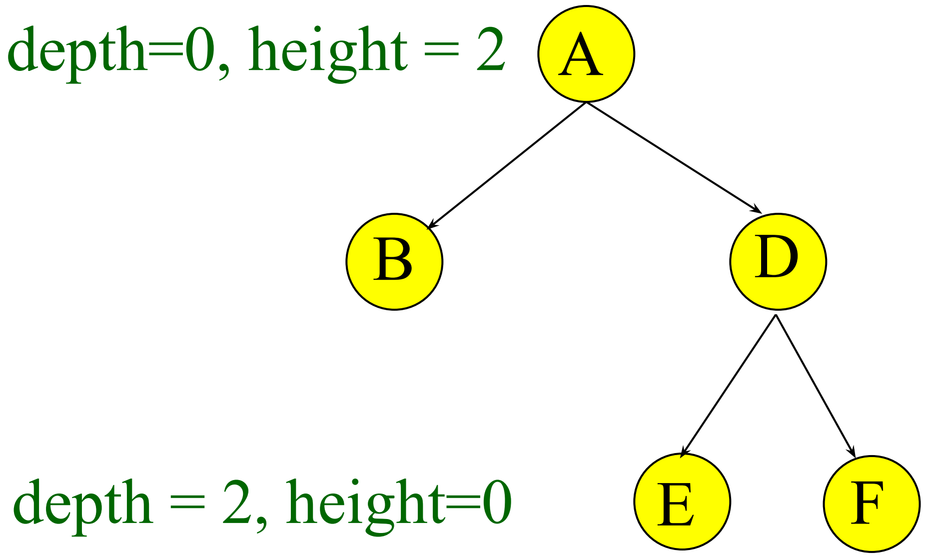


Types of Binary Trees

- Complete Binary Tree
- Skew Tree
- Strictly (or Full) Binary Tree

Tree Terminology

- Length of a path = number of edges
- Depth of a node N = length of path from root to N
- Height of node N = length of longest path from N to a leaf
- Depth and height of tree = height of root



Properties of Binary Trees

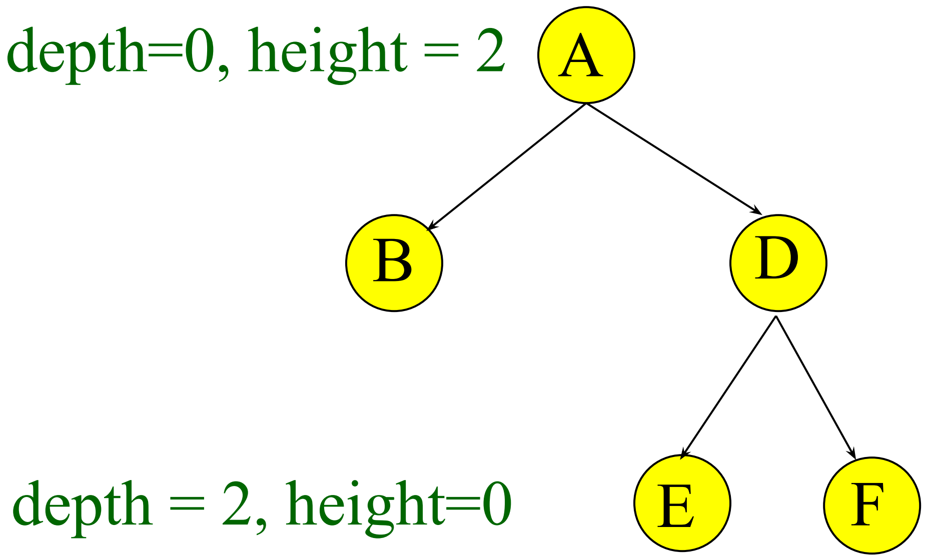
- Depth
  - Depth(tree) = MAX {depth(leaf)} = height(root)
  - max number of leaves = $2^{height(tree)}$
  - max number of nodes = $2^{depth(tree)+1} - 1$
  - max depth = n-1
- ***Subtree*** of a node:

  A tree whose root is a child of that node
- ***Level*** of a node:

  A measure of its distance from the root:

  Level of the root = 1

  Level of other nodes = 1 + level of parent

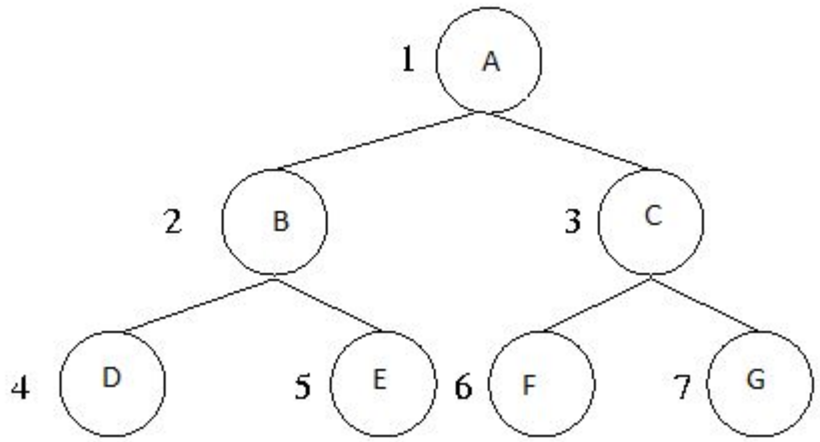


Representation of Binary Trees

- Sequential / Array Representation
- Linked List Representation

Sequential Representation

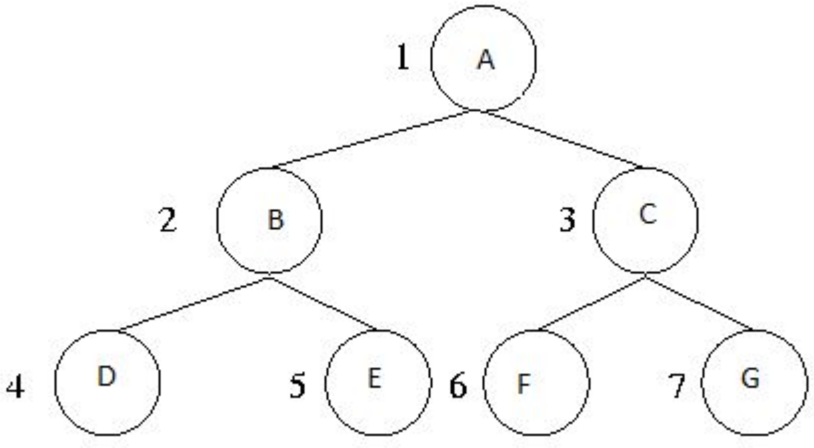
- Number the nodes level by level from left to right



1	A
2	B
3	C
4	D
5	E
6	F
7	G

Sequential Representation

- Number the nodes level by level from left to Right

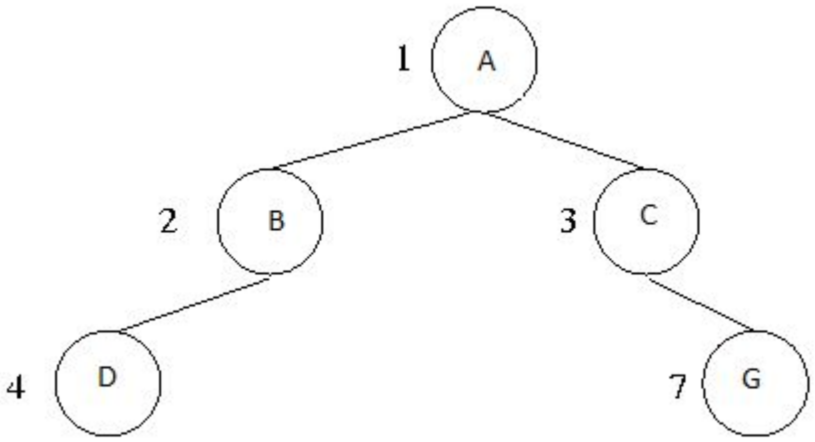


1	A
2	B
3	C
4	D
5	E
6	F
7	G

Parent of i $= \lfloor i/2 \rfloor$

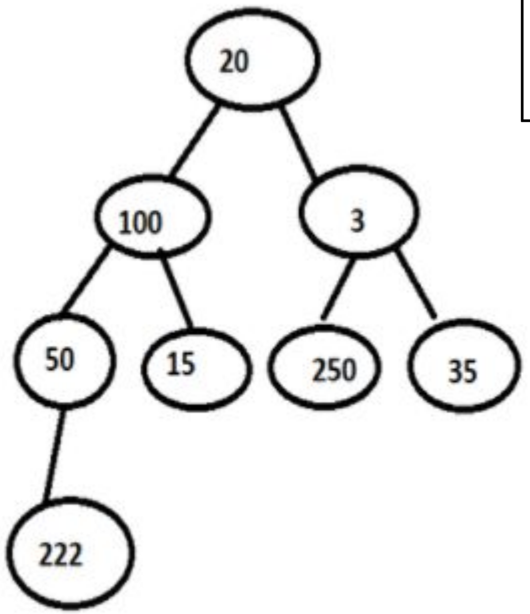
Left Child of p = 2p
Right Child of p = 2p + 1

Array Representation – Advantages and Limitations

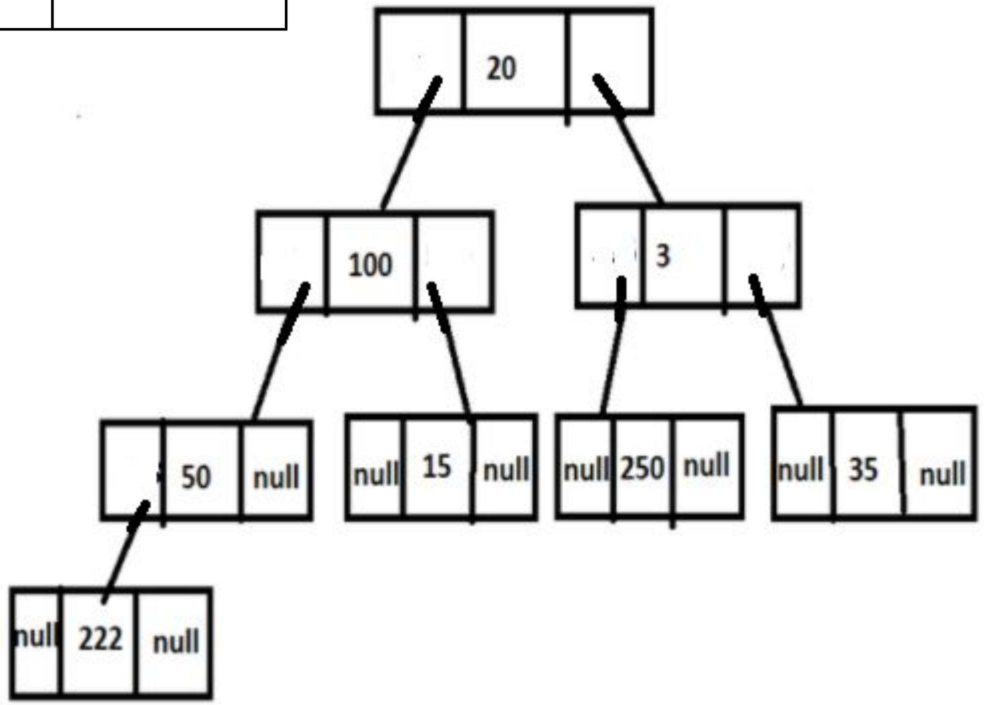


1	A
2	B
3	C
4	D
5	-
6	-
7	G

Linked Representation



Data	left	right



Binary Tree traversals

- Preorder: Visit root, traverse left, traverse right
- Inorder: Traverse left, visit root, traverse right
- Postorder: Traverse left, traverse right, visit root

Inorder Traversal

```
Algorithm inorder(t)
If ( t == NULL)
    return
inorder( t -> left)
print t -> data
inorder( t -> right)
return
```

Preorder Traversal

```
Algorithm preorder(t)
If ( t == NULL)
   return
print t→ data
preorder( t → left)
preorder( t → right)
return
```

Postorder Traversal

```
Algorithm postorder(t)
If ( t == NULL)
   return
postorder( t → left)
postorder( t → right)
print t → data
return
```

Compute the number of nodes in a Binary tree

```
Algorithm size(T)
 1. if (T == NULL)
          return 0
    else
          return   size(T →left) + 1 + size(T →right)
2. end
```

Count number of leaf nodes

Count the number of leaf nodes

```
Algorithm countLeaf(T)
if T== NULL
    return 0
If( T → left == NULL AND T → right == NULL)
    return 1
else
    return countLeaf( T → left) + countLeaf( T → right)
end
```

Depth of a Binary tree

Algorithm Depth(T)

1. if (T == NULL)

          return(0)

     else {

          lDepth = maxDepth(T →left)

          rDepth = maxDepth(T →right)

          if (lDepth > rDepth)

          return(lDepth+1)

          else

          return(rDepth+1)

2. end

Find level of a node with given value

Level of a node

```
Algorithm nodeLevel(T, elt, level)
[ initial call nodeLevel(T, elt, 0) ]
 if (T == NULL)
       return -1
 if (T → data == elt)
       return level
 l = nodeLevel(T→left, elt, level+1)
 if (l != 0)
        return l;
 else
       return level(T →right, elt, level+1)
end
```

Structurally Identical Trees

```
Algorithm identicalTree(T1, T2) {
1.   if (T1 == NULL AND T2 == NULL)            // [ If both trees are empty -> true ]
       return TRUE

    else if (T1 != NULL AND T2 != NULL)        // [If both trees are non-empty , compare them ]

         if (  T1 → data == T2 → data AND identicalTree(T1 → left, T2 → left)
                                          AND identicalTree(T1 → right, T2 → right)
                return TRUE

         else                       // one empty, one not -> false

                return   FALSE
2. end
```

Applications

- Expression Trees
- Binary Search Trees
- Dictionary
- Huffman Trees