ADITYA COLLEGE OF ENGINEERING & TECHNOLOGY

DATA STRUCTURES

UNIT IV : TREES

PART - 1

Branch: I-II IT

T. Srinivasulu

Dept. of Information Technology

Aditya College of Engineering & Technology

Surampalem.

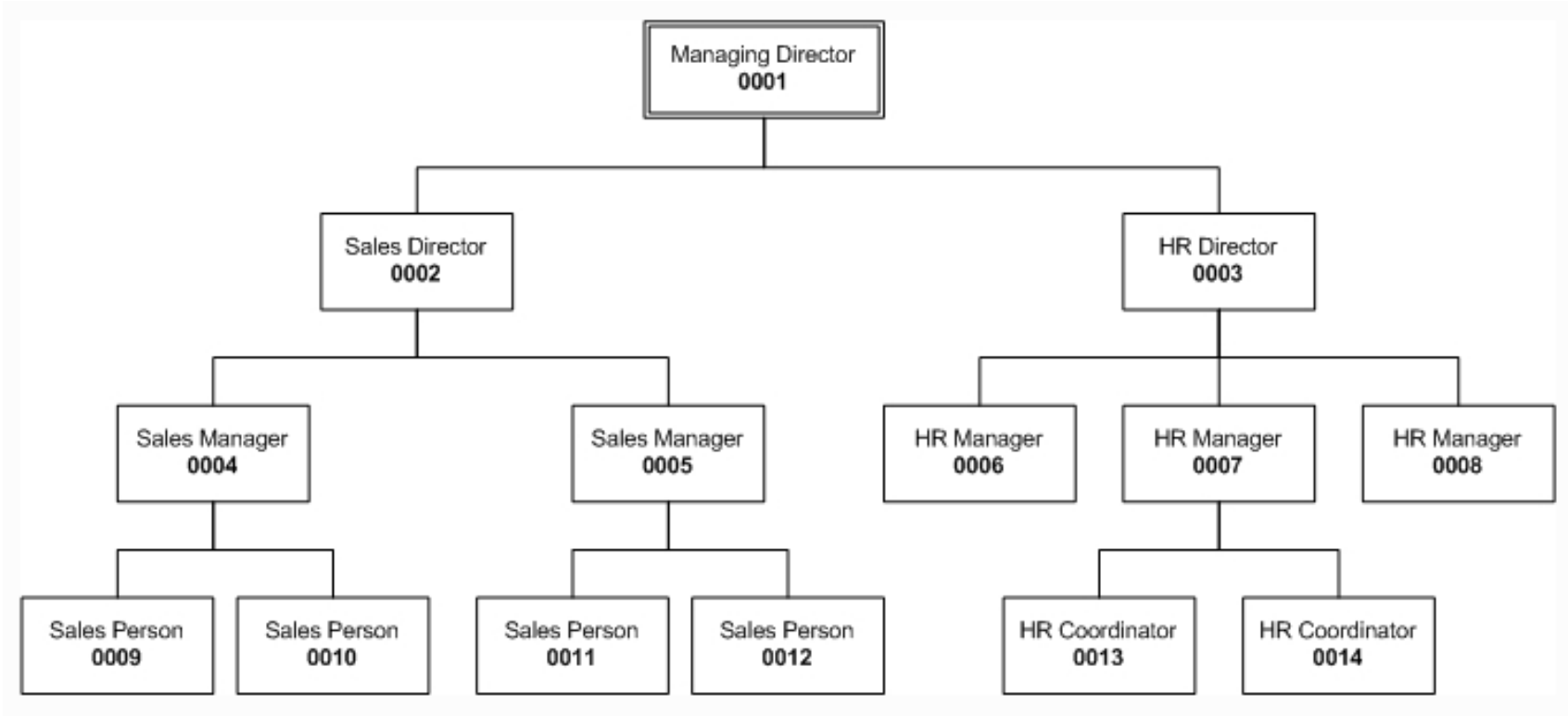
Contents

Trees:

- Trees: Basic Terminology in Trees,
- Binary Trees-Properties,
- Representation of Binary Trees using Arrays and Linked lists.
- Binary Search Trees- Basic Concepts,
- BST Operations: Insertion, Deletion, Tree Traversals
- Applications-Expression Trees,
- Heap Sort,
- Balanced Binary Trees- AVL Trees, Insertion, Deletion and Rotations.

Introduction to trees

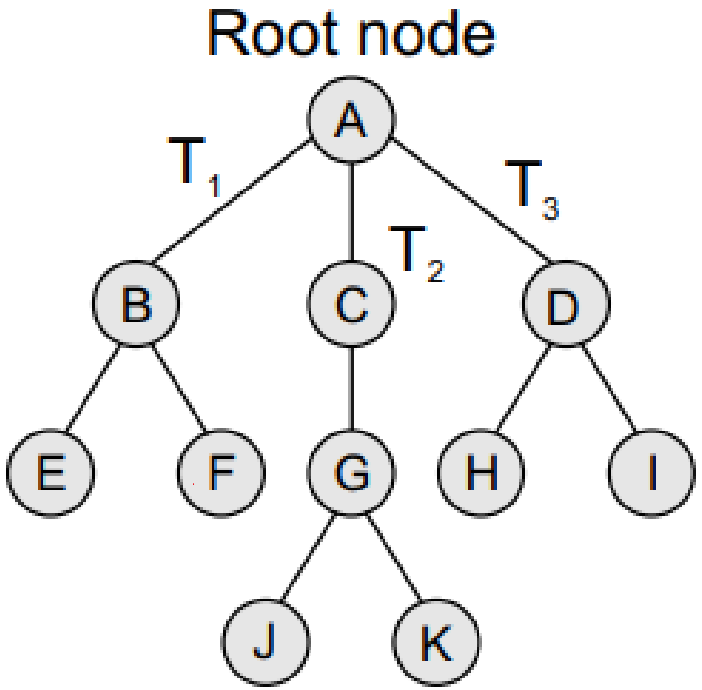
- In linear data structure data is organized in sequential order and in non-linear data structure data is organized in random order. Tree is a very popular non-linear data structure used in wide range of applications.
- A tree structure means that the data are organized in a hierarchical manner. Hierarchical data is represented for ancestor-descendent, superior-subordinate, whole – part etc.



- A tree is recursively **defined** as a set of one or more nodes where one node is designated as the root of the tree and all the remaining nodes can be partitioned into non-empty sets each of which is a sub-tree of the root.
- Below Figure shows a tree where node A is the root node; nodes B, C, and D are children of the root node and form sub-trees of the tree rooted at node A.

Basic Terminology

- **Root node** The root node R is the topmost node in the tree. If R = NULL, then it means the tree is empty.
- **Sub-trees** If the root node R is not NULL, then the trees T1 , T2 , and T3 are called the sub-trees of R.
- **Leaf node** A node that has no children is called the leaf node or the terminal node.
- **Path** A sequence of consecutive edges is called a path. For example, in Fig. the path from the root node A to node I is given as: A, D, and I.
- **Ancestor node** An ancestor of a node is any predecessor node on the path from root to that node. The root node does not have any ancestors. In the tree given in Fig. nodes A, C, and G are the ancestors of node K.



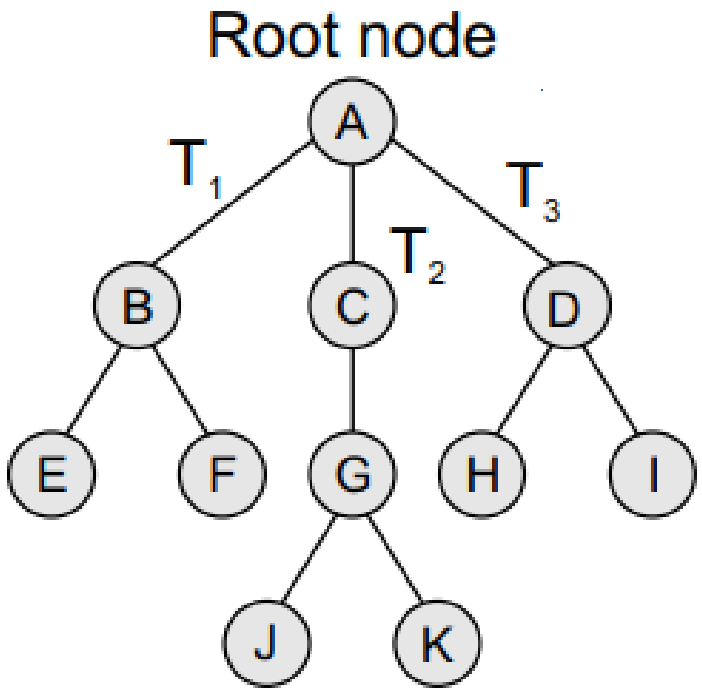
- **Descendant node** A descendant node is any successor node on any path from the node to a leaf node. Leaf nodes do not have any descendants. In the tree given in Fig, nodes C, G, J, and K are the descendants of node A.

- **Level number** Every node in the tree is assigned a level number in such a way that the root node is at level 0, children of the root node are at level number 1. Thus, every node is at one level higher than its parent. So, all child nodes have a level number given by parent's level number + 1.

- **Degree** Degree of a node is equal to the number of children that a node has. The degree of a leaf node is zero.

- **In-degree** In-degree of a node is the number of edges arriving at that node.

- **Out-degree** Out-degree of a node is the number of edges leaving that node.



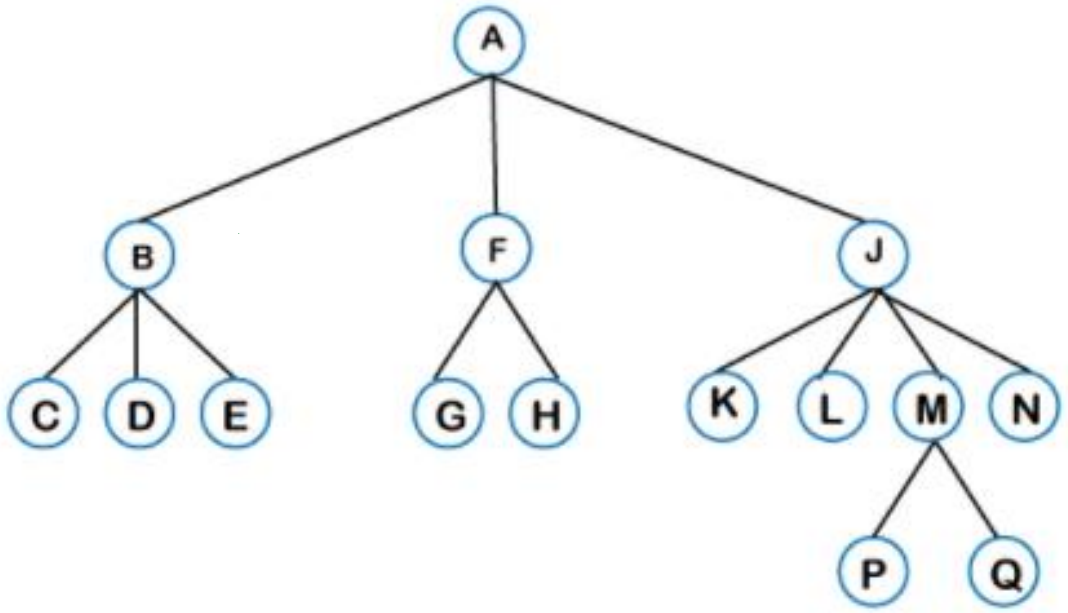
TYPES OF TREES

Trees are of following 6 types:

1. General trees
2. Binary trees
3. Binary search trees
4. Forests
5. Expression trees
6. Tournament trees

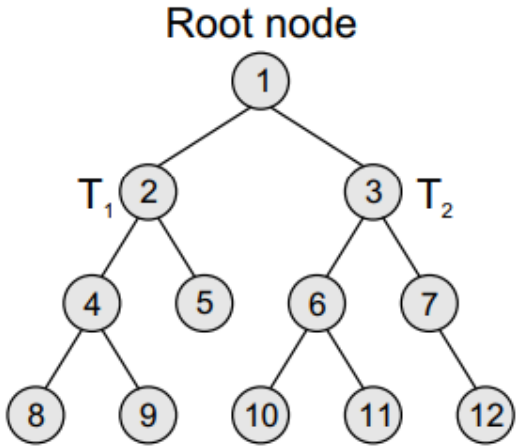
General Trees

- General trees are data structures that store elements **hierarchically**.
- The top node of a tree is the root node and each node, except the root, has a parent.
- A node in a general tree (except the leaf nodes) may have zero or more sub-trees.
- General trees which have 3 sub-trees per node are called ternary trees.
- However, the number of sub-trees for any node may be variable. For example, a node can have 1 sub-tree, whereas some other node can have 3 sub-trees.



Binary Trees

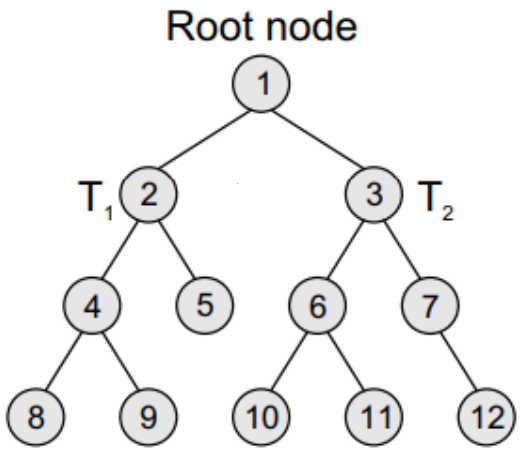
- A binary tree is a data structure that is defined as a collection of elements called **nodes**.
- In a binary tree, the topmost element is called the **root node**, and each node has 0, 1, or at the most 2 children.
- A node that has zero children is called a **leaf node** or a **terminal node**.
- Every node contains a **data element**, a **left pointer** which points to the **left child**, and a **right pointer** which points to the **right child**.
- The root element is pointed by a '**root**' pointer. If root = NULL, then it means the tree is empty.
- Figure, shows a binary tree. In the figure, R is the root node and the two trees T1 and T2 are called the left and right sub-trees of R. T1 is said to be the left successor of R. Likewise, T2 is called the right successor of R.




```
struct node
{
  int data;
  struct node * left;
  struct node * right;
};
```

Terminology Binary Trees

- **Parent** If N is any node in T that has left successor S1 and right successor S2, then N is called the **parent** of S1 and S2. Correspondingly, S1 and S2 are called the left child and the right child of N. Every node other than the root node has a parent.
- **Level number** Every node in the binary tree is assigned a level number. The root node is defined to be at level 0. The left and the right child of the root node have a level number 1. Similarly, every node is at one level higher than its parents. So all child nodes are defined to have level number as parent's level number + 1.
- **Degree** of a node It is equal to the number of children that a node has. The degree of a leaf node is zero. For example, in the tree, degree of node 4 is 2, degree of node 5 is zero and degree of node 7 is 1.
- **Sibling** All nodes that are at the same level and share the same parent are called siblings (brothers). For example, nodes 2 and 3; nodes 4 and 5; nodes 6 and 7; nodes 8 and 9; and nodes 10 and 11 are siblings.
- **Leaf node** A node that has no children is called a leaf node or a terminal node. The leaf nodes in the tree are: 8, 9, 5, 10, 11, and 12.



Terminology Binary Trees

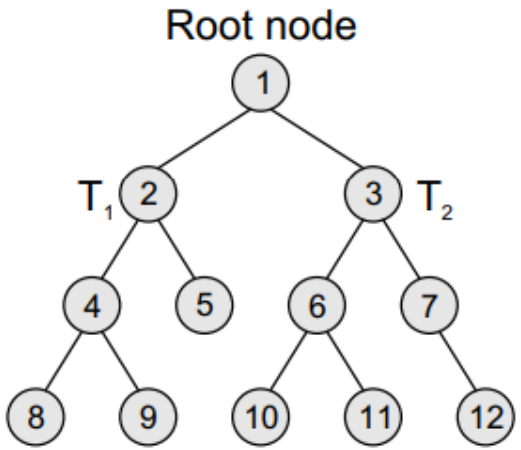
- **Edge** It is the line connecting a node N to any of its successors. A binary tree of n nodes has exactly n – 1 edges because every node except the root node is connected to its parent via an edge.

- **Path** A sequence of consecutive edges. For example, the path from the root node to the node 8 is given as: 1, 2, 4, and 8.

- **Depth** The depth of a node N is given as the length of the path from the root R to the node N. The depth of the root node is zero.

- **Height of a tree** It is the total number of nodes on the path from the root node to the deepest node in the tree. A tree with only a root node has a height of 1.
- The height of a binary tree with n nodes is at least log2 (n+1) and at most n.
- A binary tree of height h has at least h nodes and at most $2^h - 1$ nodes.

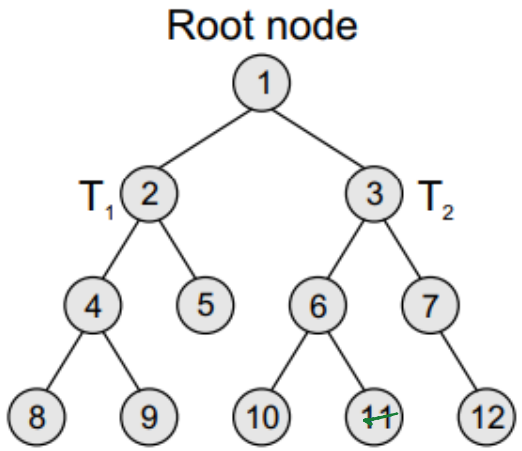
- **In-degree/out-degree** of a node It is the number of edges arriving at a node. The root node is the only node that has an in-degree equal to zero. Similarly, out-degree of a node is the number of edges leaving that node.



Properties of Binary Trees:

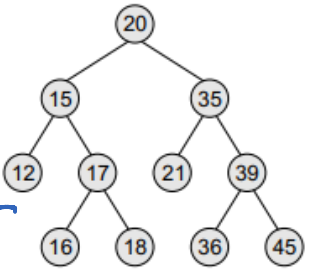
Basic properties of a binary tree:

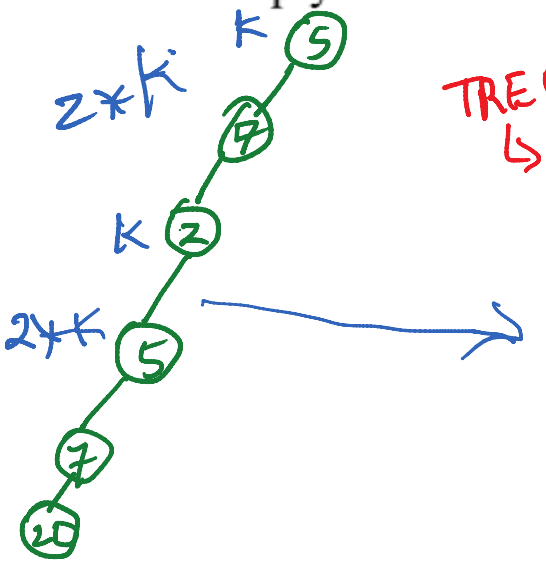
- A binary tree can have a maximum of 2^l nodes at level l if the level of the root is zero.
- When each node of a binary tree has one or two children, the number of leaf nodes (nodes with no children) is one more than the number of nodes that have two children.
- There exists a maximum of $(2^h - 1)$ nodes in a binary tree if its height is h, and the height of a leaf node is one.
- If there exist L leaf nodes in a binary tree, then it has at least $L+1$ levels.
- A binary tree of n nodes has $\log_2(n+1)$ minimum number of levels or minimum height.
- The minimum and the maximum height of a binary tree having n nodes are $\log_2 n$ and n, respectively.



Sequential representation of binary trees

- Sequential representation of trees is done using **single** or **one-dimensional arrays**.
- Though it is the simplest technique for memory representation, it is inefficient as it requires a lot of memory space.
- A sequential binary tree follows the following rules:
  1. A one-dimensional array, called TREE, is used to store the elements of tree.
  2. The root of the tree will be stored in the first location. That is, TREE[1] will store the data of the root element.
  3. The children of a node stored in location K will be stored in locations (2 × K) and (2 × K+1).
  4. The maximum size of the array TREE is given as $(2^h - 1)$, where h is the height of the tree.
  5. An empty tree or sub-tree is specified using NULL. If TREE[1] = NULL, then the tree is empty.





TREE

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	20	15	35	12	17	21	39			16	18			36	45				

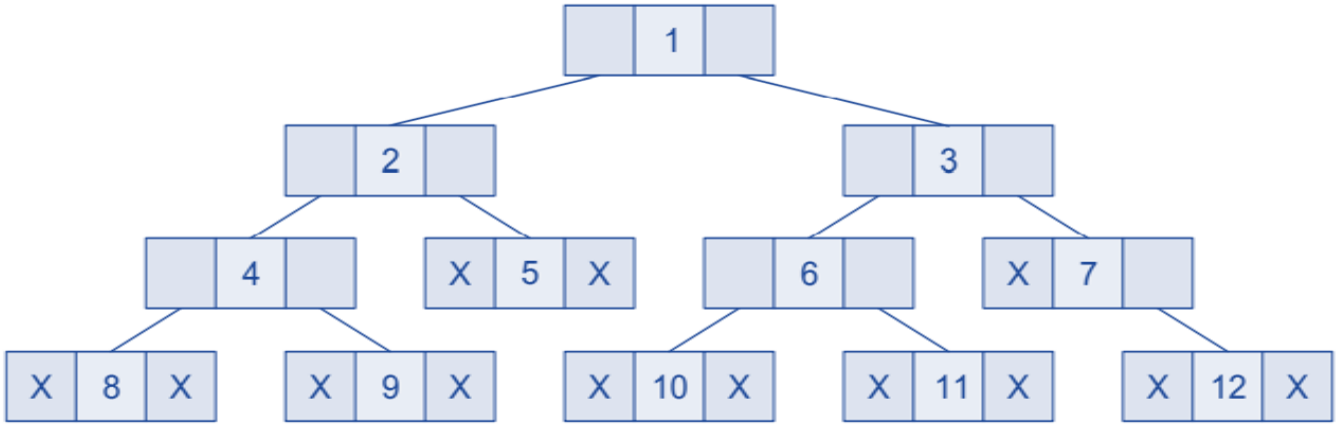
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	5	7		2				5								7			
								K								K			

Linked representation of binary trees

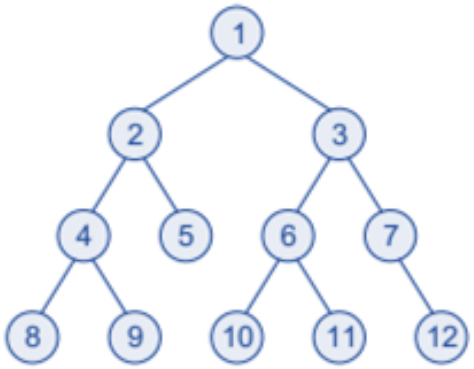
- In the linked representation of a binary tree, every node will have **three parts**: the data element, a pointer to the left node, and a pointer to the right node.

```
struct node
{
      struct node *left;
      int data;
      struct node *right;
};
```

- Every binary tree has a pointer ROOT, which points to the root element (topmost element) of the tree. If ROOT = NULL, then the tree is empty.



Representation of Binary tree in main memory using linked list.



ROOT
3

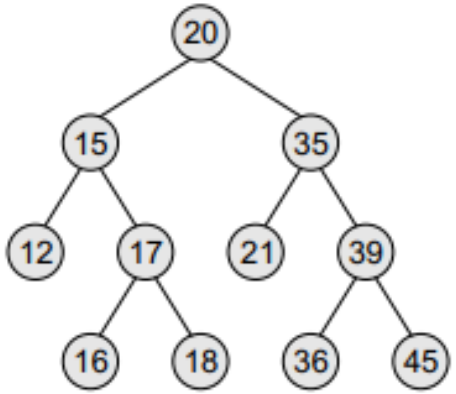
15
AVAIL

	LEFT	DATA	RIGHT
1	–1	8	–1
2	–1	10	–1
3	5	1	8
4			
5	9	2	14
6			
7			
8	20	3	11
9	1	4	12
10			
11	–1	7	18
12	–1	9	–1
13			
14	–1	5	–1
15			
16	–1	11	–1
17			
18	–1	12	–1
19			
20	2	6	16

Creating a binary tree using linked list

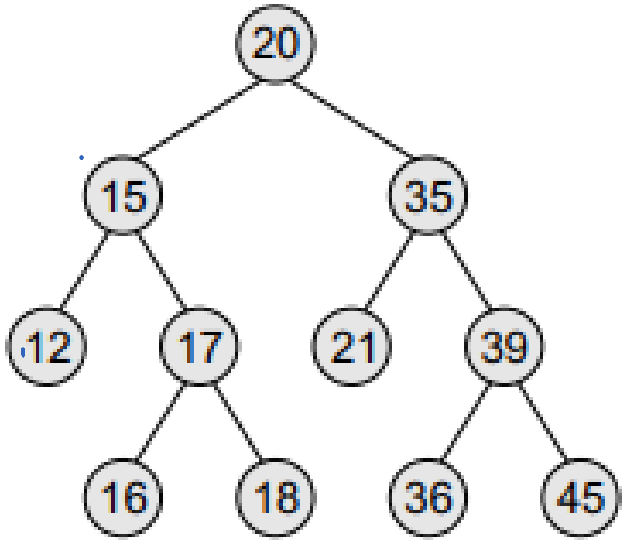
```
struct node *create()
{
        struct node *p;
        int x;
        printf("Enter data(-1 for no data):");
        scanf("%d",&x);
        if(x==-1)
        return NULL;
        p=(struct node*)malloc(sizeof(struct node));
        p->data=x;
        printf("Enter left child of %d:\n",x);
        p->left=create();

        printf("Enter right child of %d:\n",x);
        p->right=create();
        return p;
}
```



Creating a binary tree using linked list

```
void display(struct node *root)
{
  if(root!=NULL)
  {
        display(root->left);
        printf("\n %5d",root->data);
        display (root->right);
  }
}
```



TRAVERSING A BINARY TREE

- Traversing a binary tree is the process of visiting each node in the tree exactly once in a systematic way.
- Unlike linear data structures in which the elements are traversed sequentially, tree is a non-linear data structure in which the elements can be traversed in many different ways.
- There are different algorithms for tree traversals. These algorithms differ in the order in which the nodes are visited.

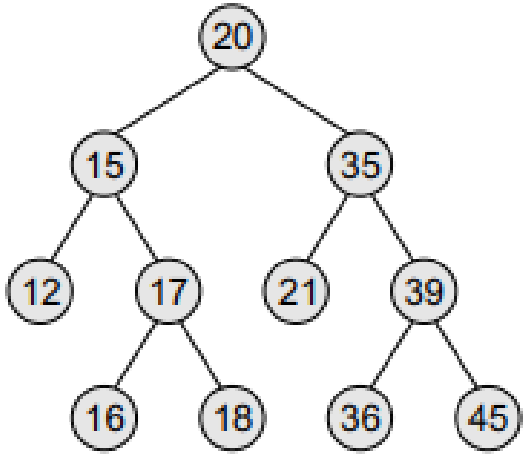
In-order Traversal

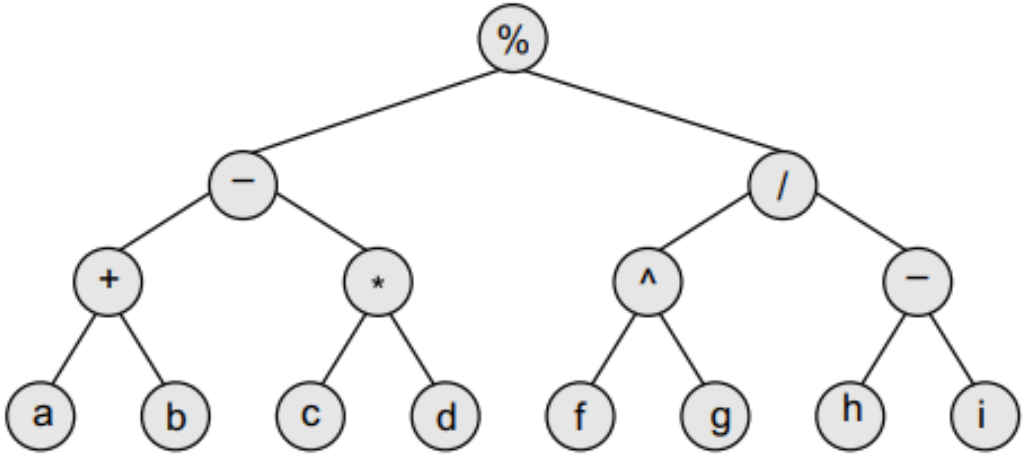
- To traverse a non-empty binary tree in in-order, the following operations are performed recursively at each node. The algorithm works by:
    1. Traversing the left sub-tree,
    2. Visiting the root node, and finally
    3. Traversing the right sub-tree.
- the left sub-tree is always traversed before the root node and the right sub-tree.
- The word ‘in’ in the in-order specifies that the root node is accessed in between the left and the right sub-trees.
- In-order algorithm is also known as the **LNR traversal algorithm (Left-Node-Right).**

The algorithm for in-order traversal:

```
Step 1: Repeat Steps 2 to 4 while TREE != NULL
Step 2:            INORDER(TREE -> LEFT)
Step 3:            Write TREE -> DATA
Step 4:            INORDER(TREE -> RIGHT)
       [END OF LOOP]
Step 5: END
```

Example:





12, 15, 16, 17, 18, 20, 21, 35, 36, 39, 45

((a + b) − (c * d)) % ((f ^ g) / (h − i))

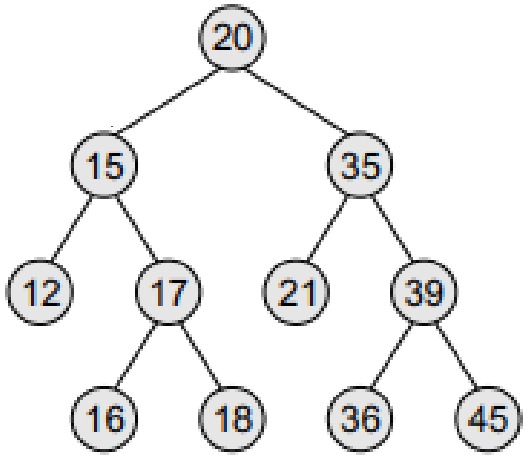
Pre-order Traversal

- To traverse a non-empty binary tree in pre-order, the following operations are performed recursively at each node.
- The algorithm works by:
    1. Visiting the root node,
    2. Traversing the left sub-tree, and finally
    3. Traversing the right sub-tree.
- Root node first, the left sub-tree next, and then the right sub-tree.
- Pre-order traversal is also called as **depth-first traversal**.
- In this algorithm, the left sub-tree is always traversed before the right sub-tree.
- The word ‘pre’ in the pre-order specifies that the root node is accessed prior to any other nodes in the left and right sub-trees.
- Pre-order algorithm is also known as the **NLR traversal algorithm (Node-Left-Right).**
- **Pre-order** traversal algorithms are used to extract a **prefix notation** from an expression tree.

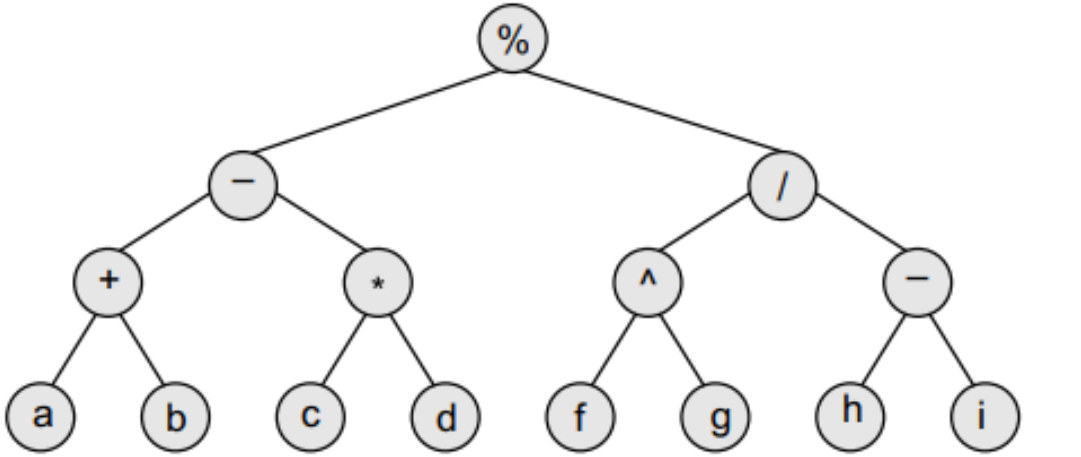
The algorithm for pre-order traversal:

```
Step 1: Repeat Steps 2 to 4 while TREE != NULL
Step 2:            Write TREE -> DATA
Step 3:            PREORDER(TREE -> LEFT)
Step 4:            PREORDER(TREE -> RIGHT)
       [END OF LOOP]
Step 5: END
```

Example:



20, 15, 12, 17, 16, 18, 35, 21, 39, 36, 45



% − + a b * c d / ^ f g − h i

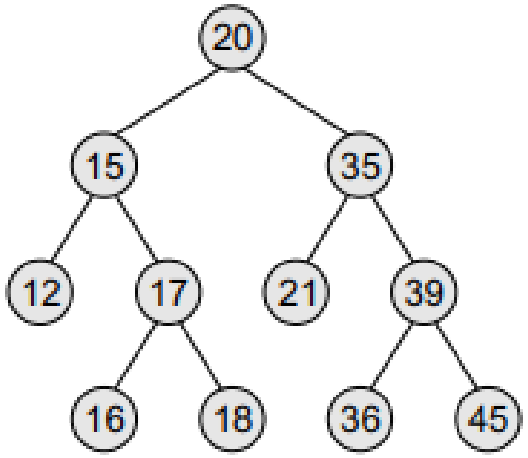
Post-order Traversal

- To traverse a non-empty binary tree in post-order, the following operations are performed recursively at each node.
- The algorithm works by:
    1. Traversing the left sub-tree,
    2. Traversing the right sub-tree, and finally
    3. Visiting the root node.
- Left sub-tree first, the right sub-tree next, and finally the root node.
- The left sub-tree is always traversed before the right sub-tree and the root node.
- The word ‘post’ in the post-order specifies that the root node is accessed after the left and the right sub-trees.
- Post-order algorithm is also known as the **LRN traversal algorithm (Left-Right-Node).**
- **Post-order** traversals are used to extract **postfix notation** from an expression tree.

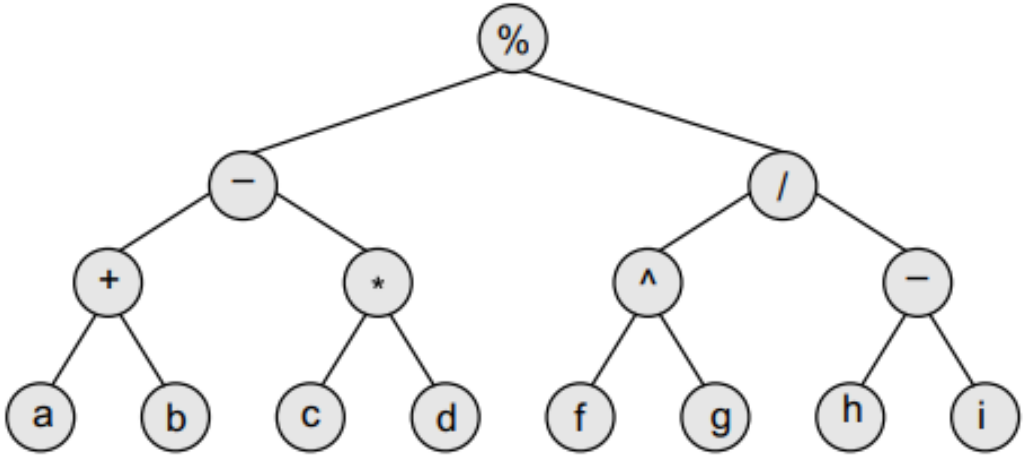
The algorithm for post-order traversal:

```
Step 1: Repeat Steps 2 to 4 while TREE != NULL
Step 2:            POSTORDER(TREE -> LEFT)
Step 3:            POSTORDER(TREE -> RIGHT)
Step 4:            Write TREE -> DATA
       [END OF LOOP]
Step 5: END
```

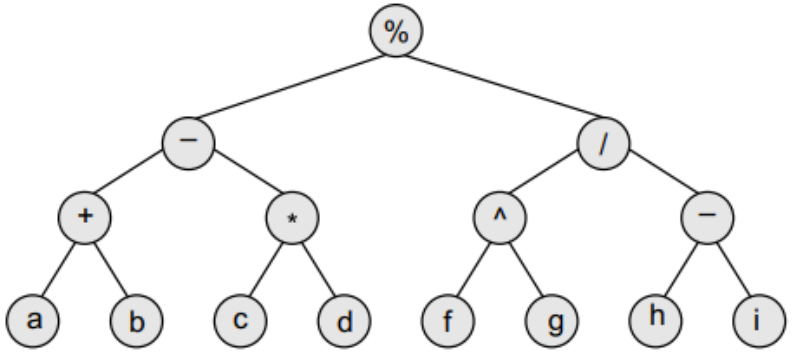
Example:

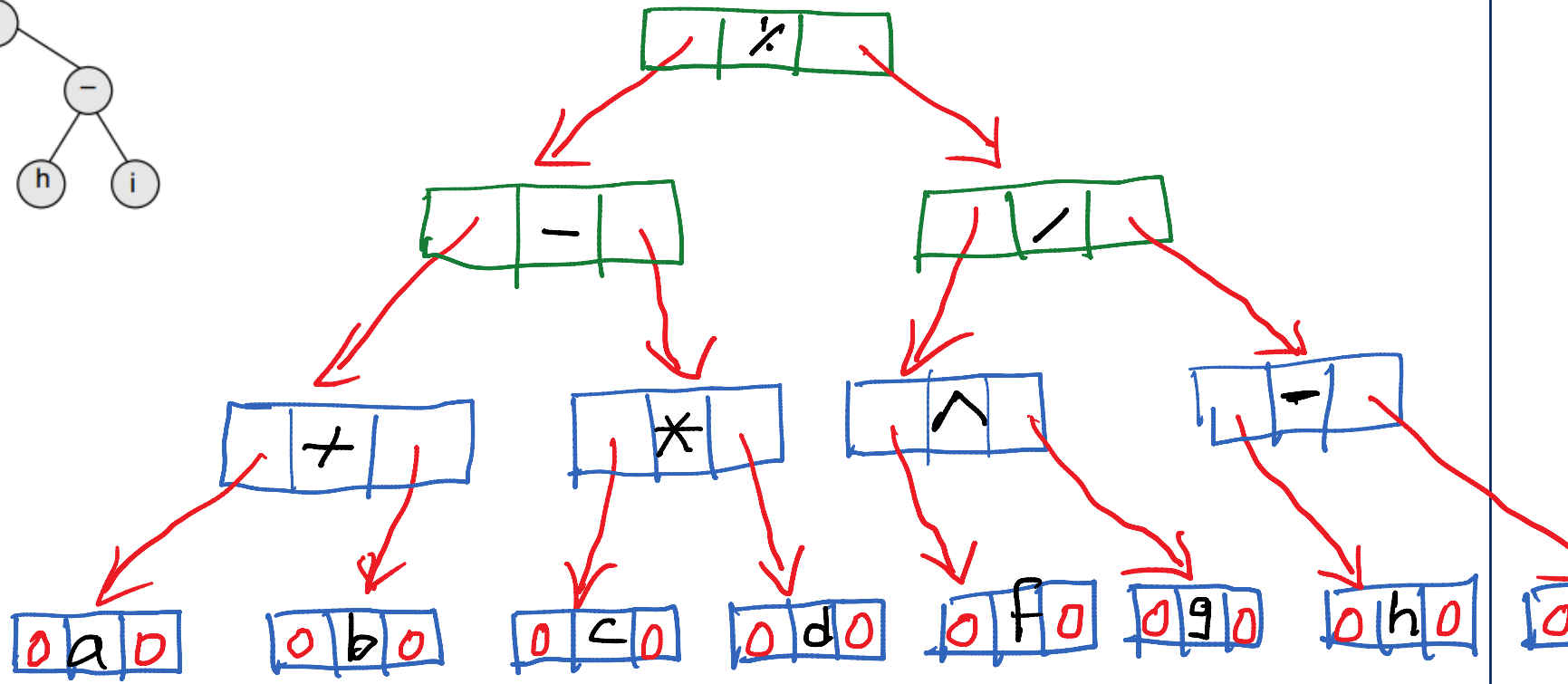


12, 16, 18, 17, 15, 21, 36, 45, 39, 35, 20,



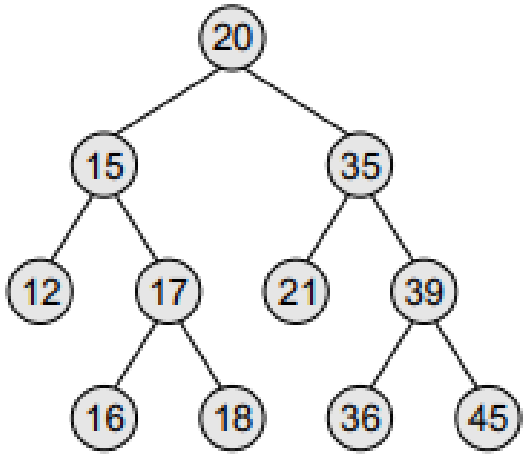
a b + c d * − f g ^ h i − / %


%
−
/
+
*
^
−
a
b
c
d
f
g
h
i

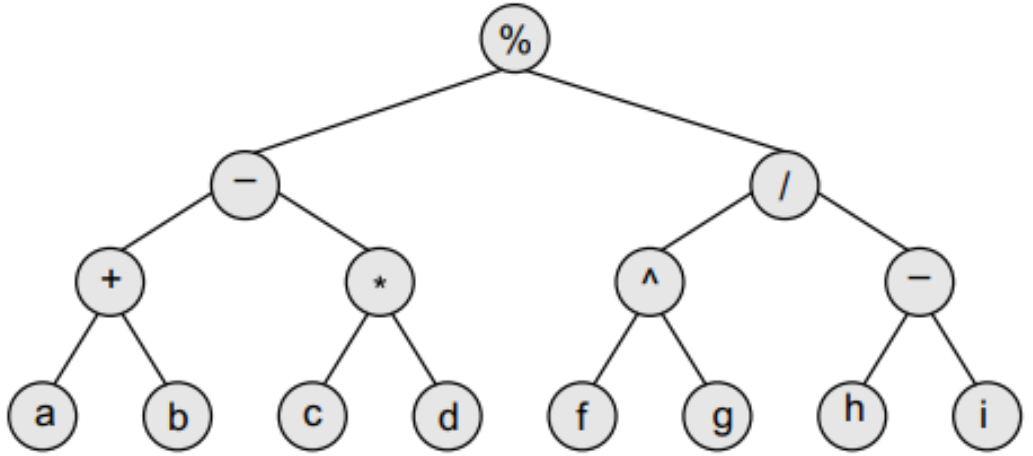


Level-order Traversal

- In level-order traversal, all the nodes at a level are accessed before going to the next level.
- This algorithm is also called as the breadth-first traversal algorithm.



20, 35, 15, 12, 17, 21, 39,
45, 36, 18, 16



% − / + * ^ − a b c d f g
h i

Constructing a Binary Tree from Traversal Results

- We can construct a binary tree if we are given at least two traversal results.
- The first traversal must be the in-order traversal and the second can be either pre-order or post-order traversal.
- The in-order traversal result will be used to determine the left and the right child nodes, and the pre-order/post-order can be used to determine the root node.
- If we have the **in-order** traversal sequence and **pre-order** traversal sequence.
- Follow the steps given below to construct the tree:

Step 1: Use the pre-order sequence to determine the root node of the tree. The first element would be the root node.

Step 2: Elements on the left side of the root node in the in-order traversal sequence form the left sub-tree of the root node. Similarly, elements on the right side of the root node in the in-order traversal sequence form the right sub-tree of the root node.

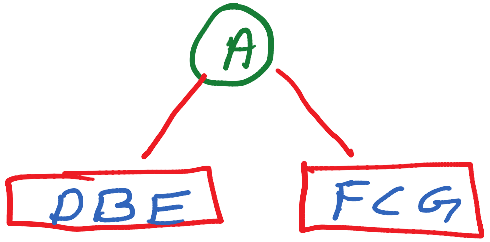
Step 3: Recursively select each element from pre-order traversal sequence and create its left and right sub-trees from the in-order traversal sequence.

Example:

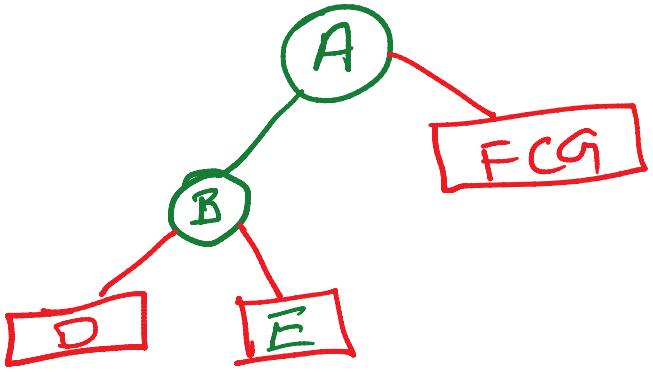
In–order Traversal: **D B E A F C G**

Pre–order Traversal: **A B D E C F G**

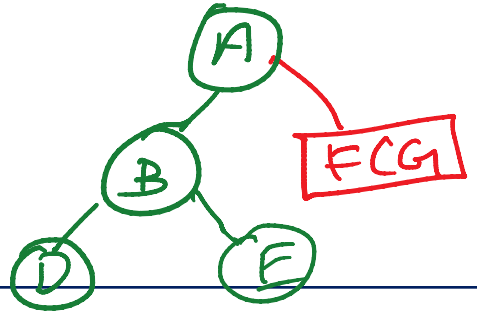
step 1 :-



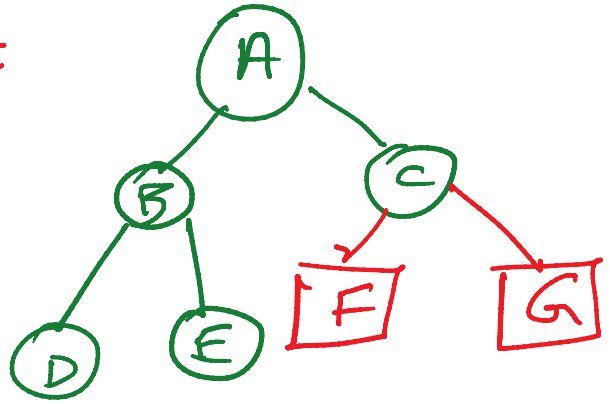
step2 :-



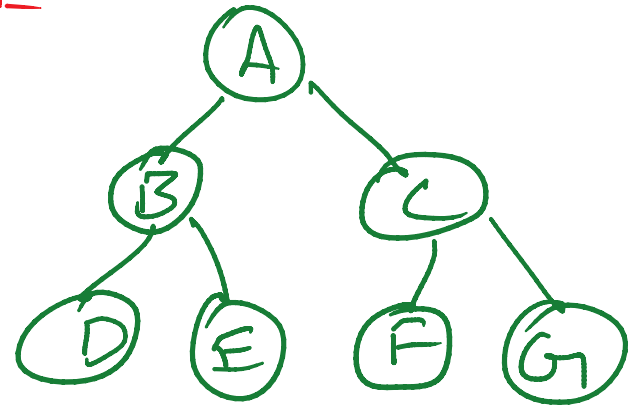
step3 :-



step4



step5 :-



Constructing a Binary tree from in-order and post-order traversals

- If we have the in-order traversal sequence and post-order traversal sequence.
- Follow the steps given below to construct the tree:

Step 1: Use the post-order sequence from right to left to determine the root node of the tree. The last element would be the root node.

Step 2: Elements on the left side of the root node in the in-order traversal sequence form the left sub-tree of the root node. Similarly, elements on the right side of the root node in the in-order traversal sequence form the right sub-tree of the root node.

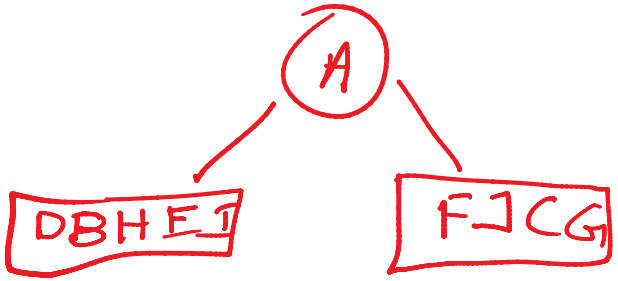
Step 3: Recursively select each element from post-order traversal sequence and create its left and right sub-trees from the in-order traversal sequence.

Example:

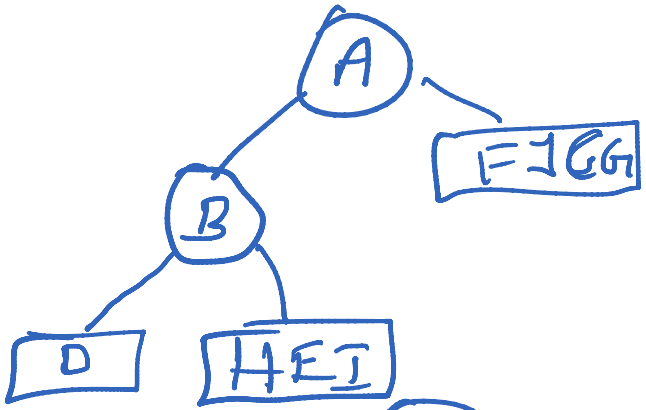
In–order Traversal: **D B H E I A F J C G**

Post order Traversal: **D H I E B J F G C A**

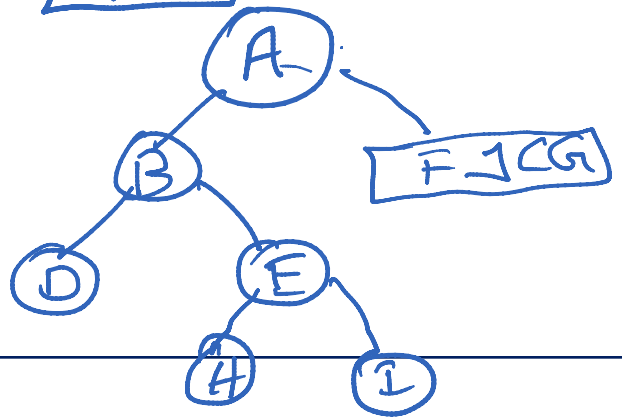
step1



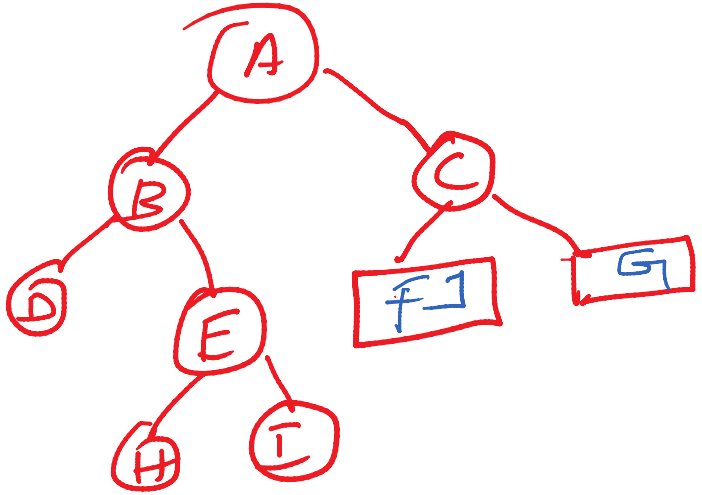
step2



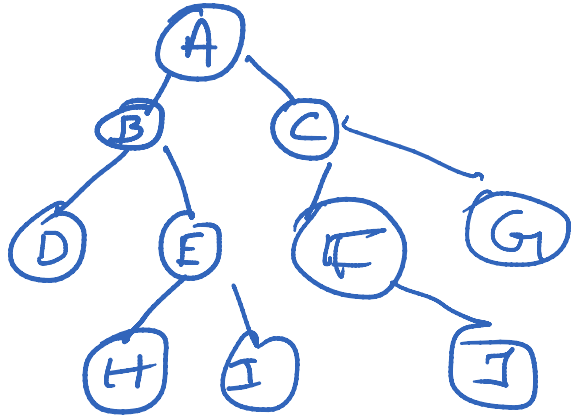
step3



step4

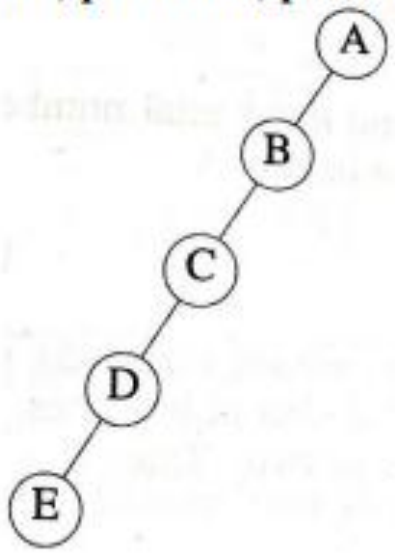


step5

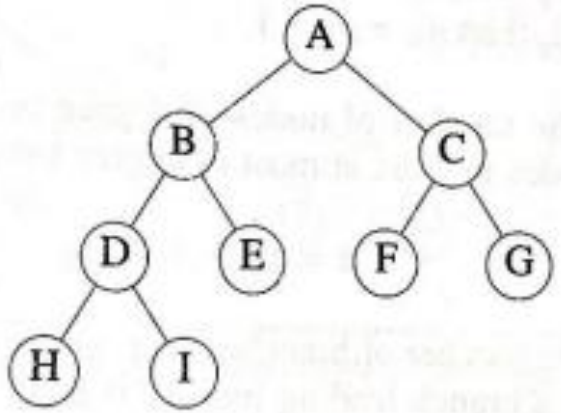


EXERCISE:

1. **Write out the inorder, preorder, postorder and level-order traversal for the binary trees**



(a)



(b)

2. **Construct a Binary tree by the following traversals**

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

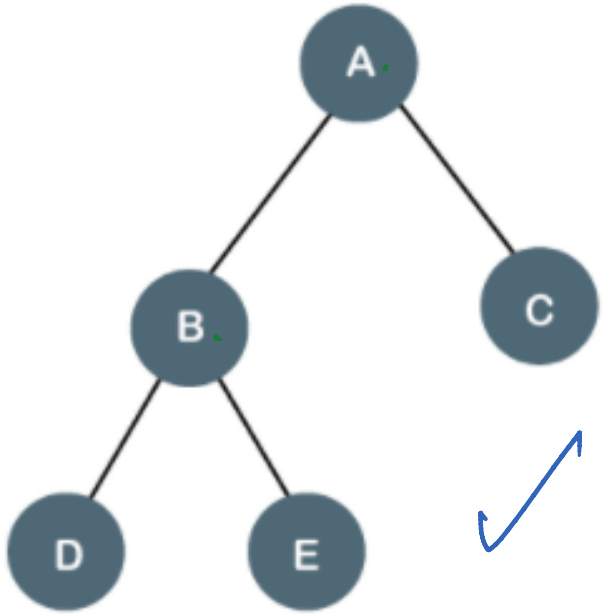
Types of Binary Tree

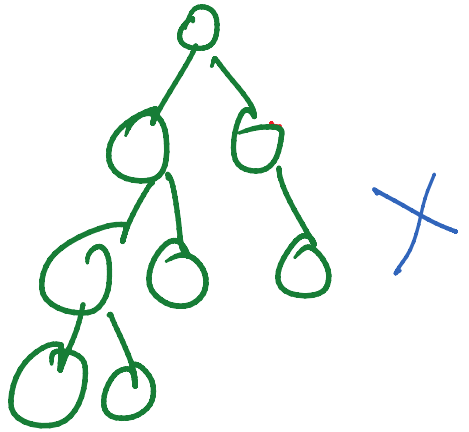
- A **binary tree** is a tree-type non-linear data structure with a maximum of two children for each parent. Every node in a binary tree has a left and right reference along with the data element. The node at the top of the hierarchy of a tree is called the root node. The nodes that hold other sub-nodes are the parent nodes.

- Types of Binary Trees
  1. Full Binary Tree
  2. Complete Binary Tree
  3. Perfect Binary Tree
  4. Balanced Binary Tree
  5. Degenerate Binary Tree

Full Binary Tree

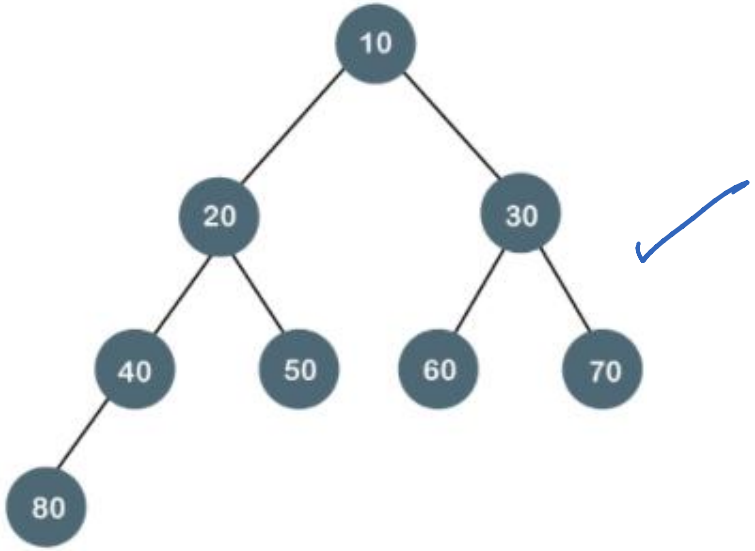
- The full binary tree is also known as a **strict binary tree**.
- The tree can only be considered as the full binary tree if each node must contain either 0 or 2 children.
- The full binary tree can also be defined as the tree in which each node must contain 2 children except the leaf nodes.

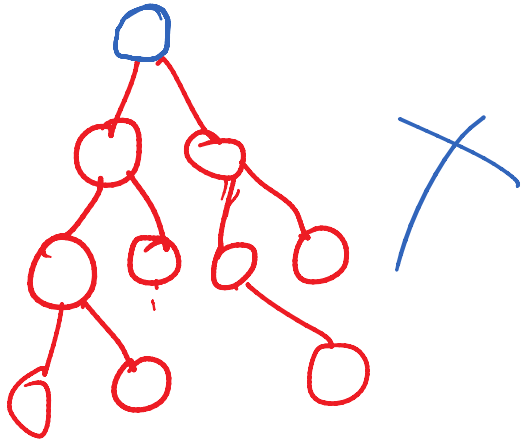




Complete Binary Tree

- The complete binary tree is a tree in which all the nodes are **completely filled except the last level**.
- In the last level, all the nodes must be as left as possible.
- In a complete binary tree, the nodes should be added from the left.

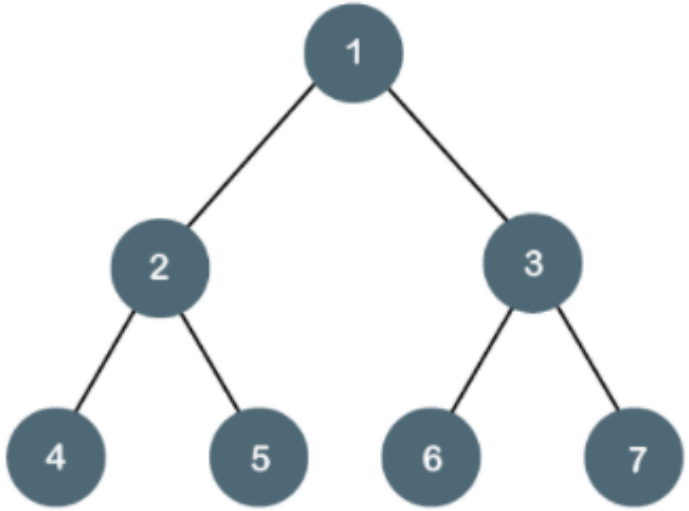


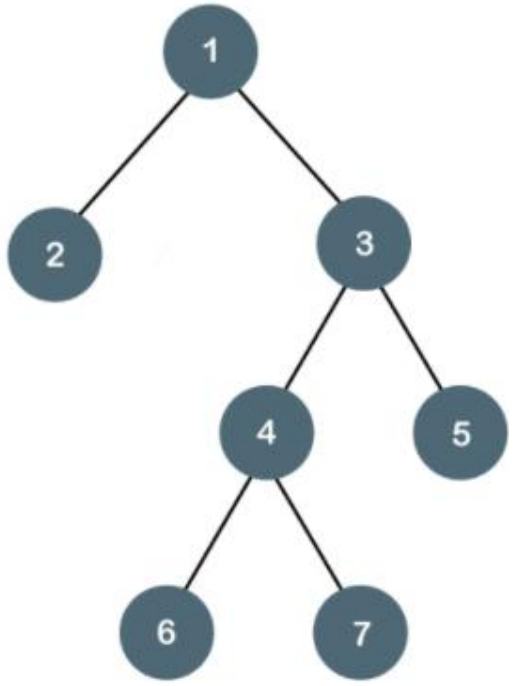


- The above tree is a complete binary tree because all the nodes are completely filled, and all the nodes in the last level are added at the left first.

Perfect Binary Tree

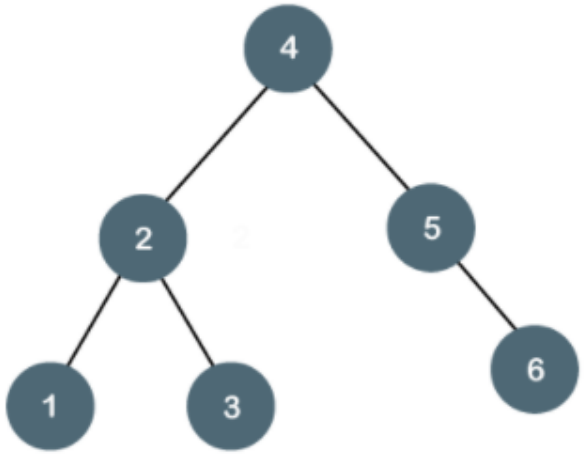
- A tree is a perfect binary tree if all the internal nodes have 2 children, and all the leaf nodes are at the same level.



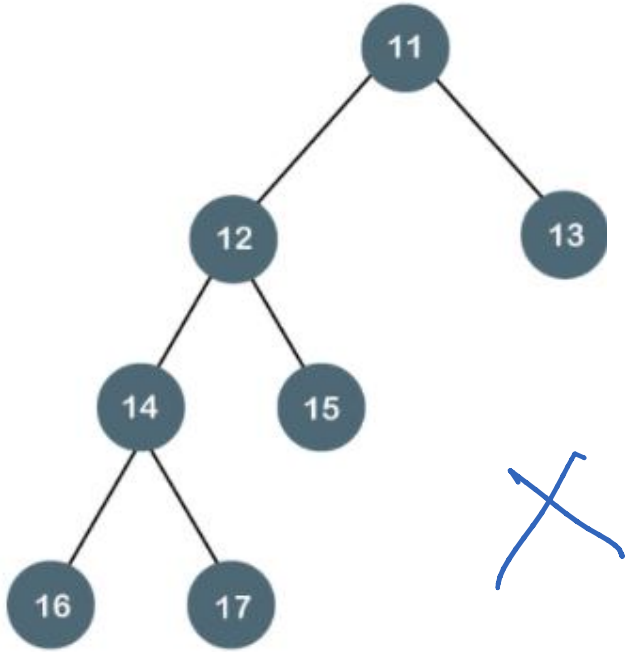


Balanced Binary Tree

- The balanced binary tree is a tree in which both the left and right trees differ by atmost 1. For example, **AVL** and **Red-Black** trees are balanced binary tree.



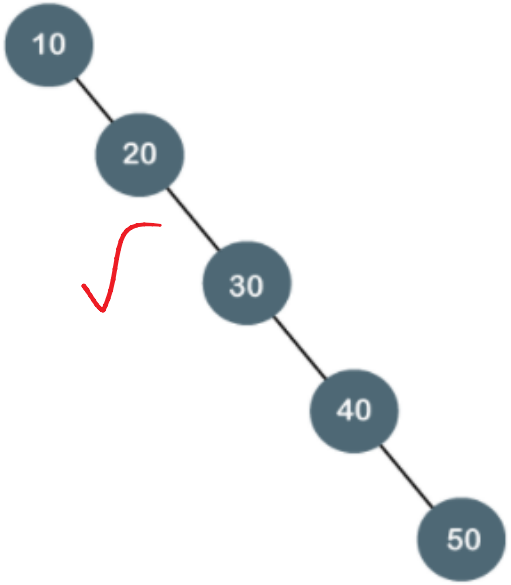
3 - 2 = 1 ✓



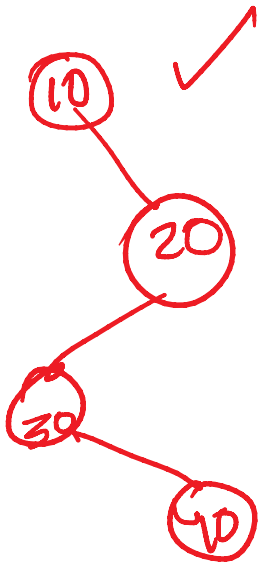
5 - 1 = 4 ✗

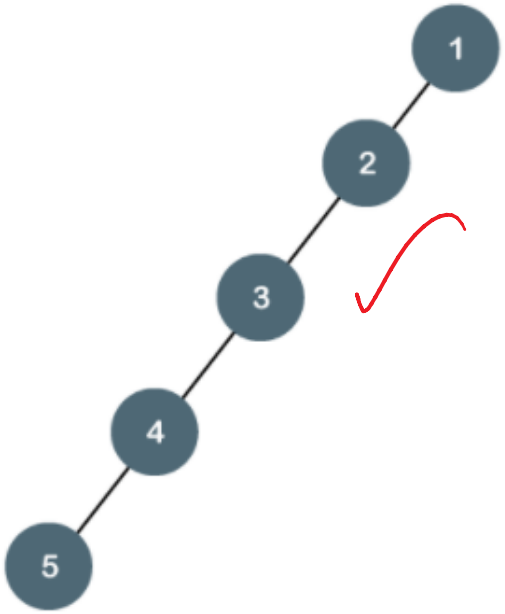
Degenerate Binary Tree

- A binary tree is said to be a degenerate binary tree or pathological binary tree if every internal node has only a single child.
- Such trees are similar to a linked list performance-wise.
- Here is an example of a degenerate binary tree:



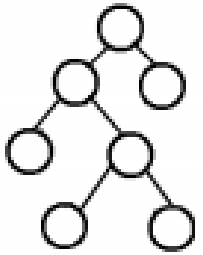
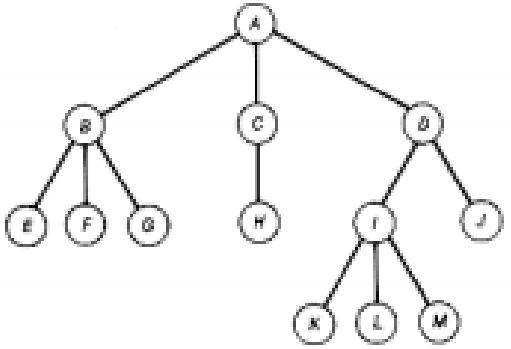
The above tree is also known as a right-skewed tree as all the nodes have a right child only.





The above tree is also known as a left-skewed tree as all the nodes have a left child only.

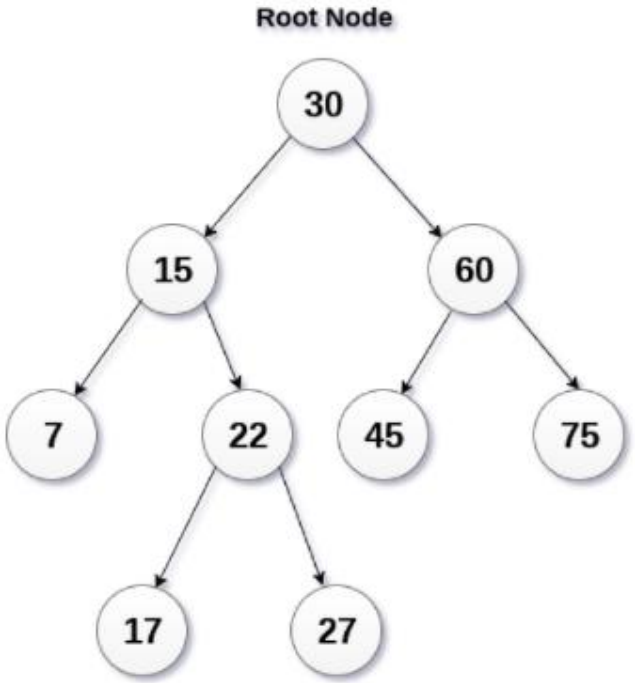
Differences between Binary Tree and Tree

Binary Tree	Tree
1. Each element in a binary tree has exactly two subtrees. 2. The subtrees of each element in a binary tree are ordered that is we distinguish between the left and right subtrees. 3. A binary tree can be empty. 4. 	1. Each element in a tree can have any number of subtrees. 2. The subtrees of each element in a tree are un-ordered. 3. A general tree cannot be empty. 4. 

- Binary trees are commonly used to implement **binary search trees**, **expression trees**, **tournament trees**, and **binary heaps**.

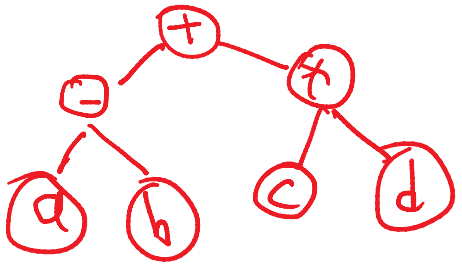
binary search trees

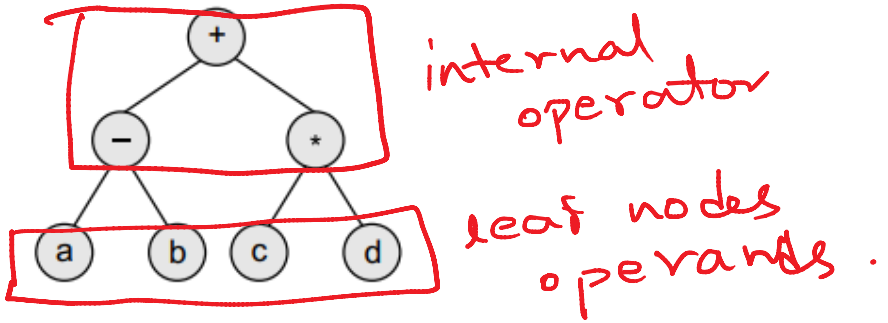
- Binary Search tree can be defined as a class of binary trees, in which the nodes are arranged in a **specific order**. This is also called ordered binary tree.
- In a binary search tree, the value of all the nodes in the left sub-tree is **less than** the value of the root.
- Similarly, value of all the nodes in the right sub-tree is **greater than** or equal to the value of the root.
- This rule will be recursively applied to all the left and right sub-trees of the root.



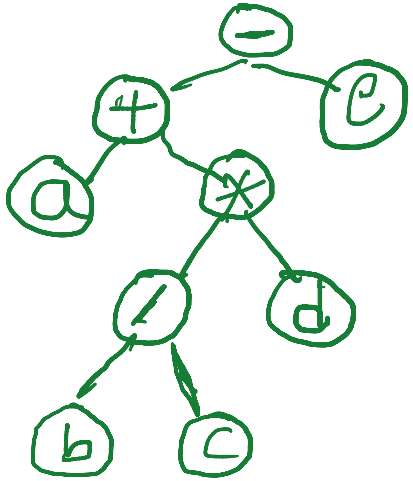
Expression trees

- Binary trees are widely used to store algebraic expressions.
- Expression trees are those in which the leaf nodes have operands, such as constants, variables to be operated, and internal nodes contain the operator on which the leaf node will be performed.
- For example, consider the algebraic expression given as:
- Exp = (a – b) + (c * d)





- Given the expression, Exp = **a + b / c * d – e**, construct the corresponding binary tree.



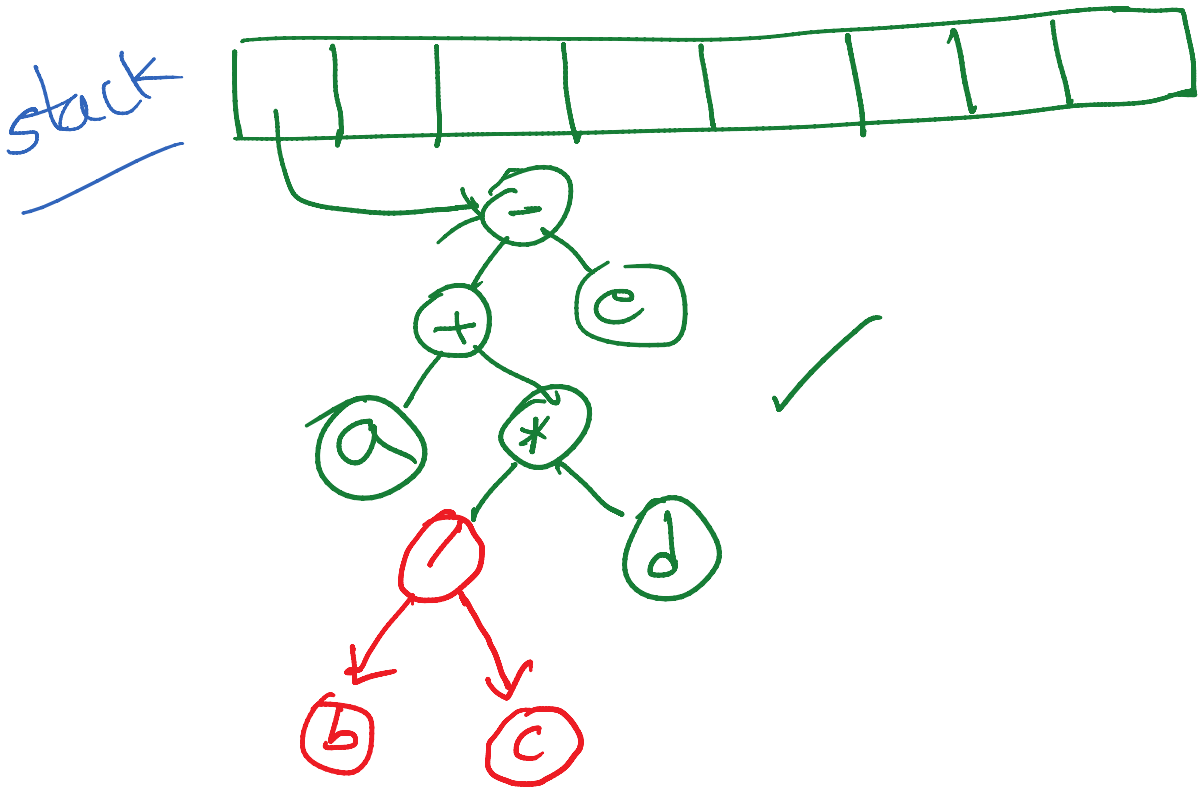
How to construct an expression tree?

- To construct an Expression Tree for the given expression, we generally use **Stack Data Structure**.
- Following are the step to construct an expression tree:

1. Read one symbol at a time from the postfix expression.
2. Check if the symbol is an operand or operator.
3. If the symbol is an operand, create a one node tree and push a pointer onto a stack
4. If the symbol is an operator, pop two pointers from the stack namely T1 & T2 and form a new tree with root as the operator, T1 & T2 as a left and right child
5. A pointer to this new tree is pushed onto the stack

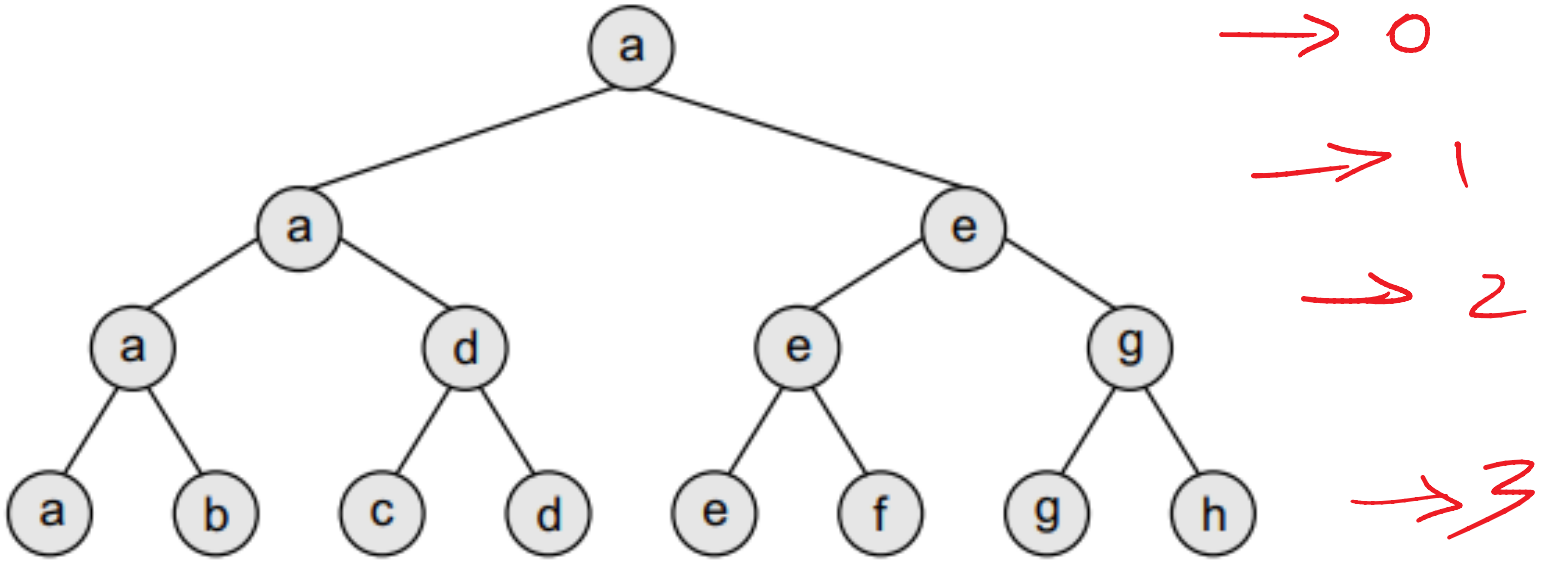
Example: Exp = **a + b / c * d – e**

postfix :— abc/d*+e-



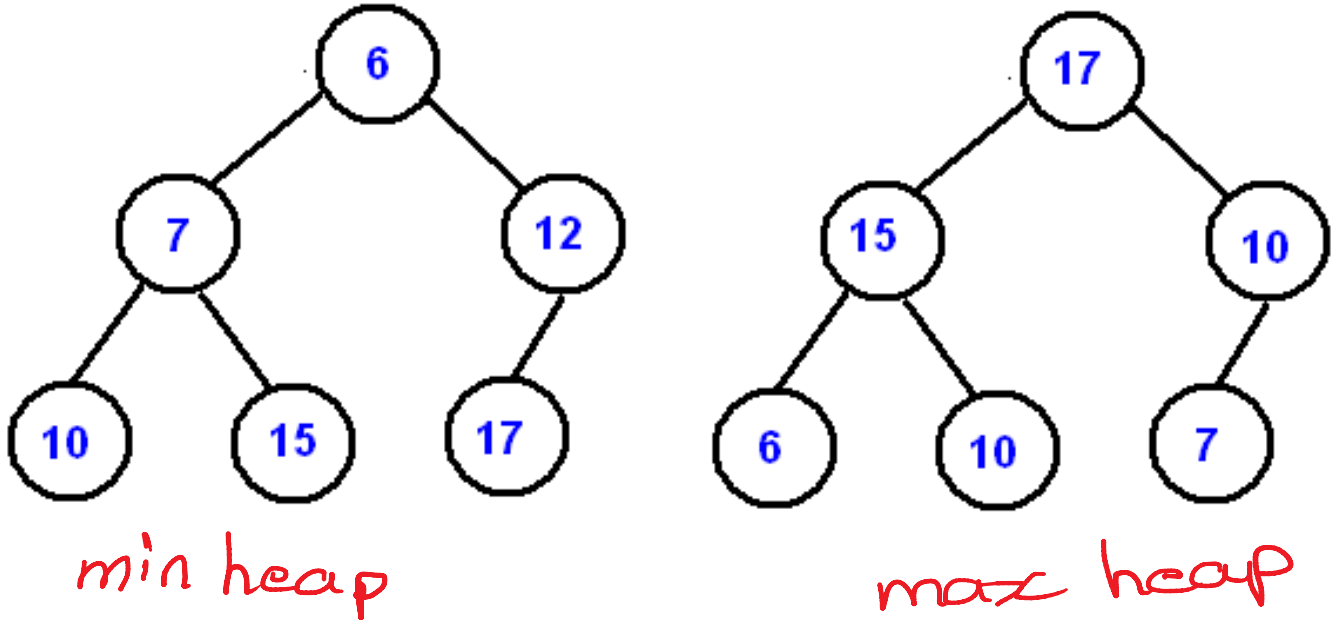
Tournament trees

- In a tournament tree (also called a **selection tree**), each **external node** represents a **player** and each **internal node** represents the **winner** of the match played between the players represented by its children nodes.
- These tournament trees are also called **winner trees** because they are being used to record the winner at each level.
- We can also have a **loser tree** that records the loser at each level.



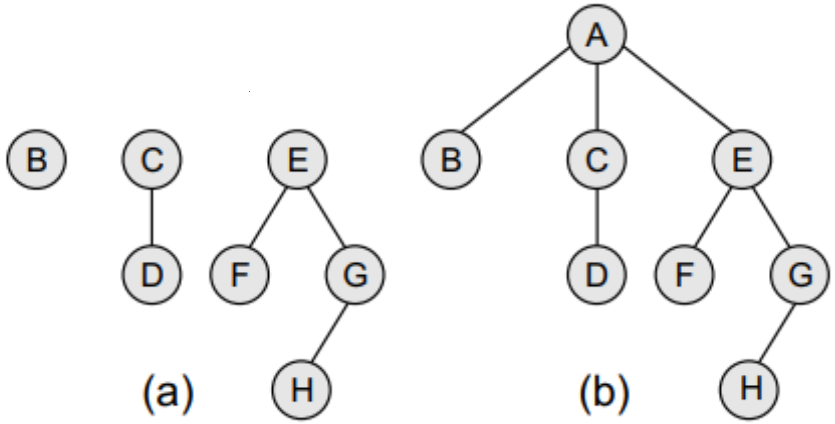
Binary Heaps

- A binary heap is a complete binary tree which satisfies the **heap ordering property**. The ordering can be one of two types:
- **the min-heap property**: the value of each node is greater than or equal to the value of its parent, with the minimum-value element at the root.
- **the max-heap property**: the value of each node is less than or equal to the value of its parent, with the maximum-value element at the root.



Forests

- A forest is a disjoint union of trees. A set of disjoint trees (or forest) is obtained by deleting the root and the edges connecting the root node to nodes at level 1.
- Every node of a tree is the root of some sub-tree. Therefore, all the sub-trees immediately below a node form a forest.
- A forest can also be defined as an ordered set of zero or more general trees.
- While a general tree must have a root, a forest on the other hand may be empty because by definition it is a set, and sets can be empty.
- We can convert a forest into a tree by adding a single node as the root node of the tree.



Applications of Trees

- Trees are used to store simple as well as complex data. Here simple means an int value, char value and complex data (structure).
- Trees are often used for implementing other types of data structures like hash tables, sets, and maps.
- A self-balancing tree, Red-black tree is used in kernel scheduling to preempt massively multi-processor computer operating system use.
- Another variation of tree, B-trees are used to store tree structures on disc. They are used to index a large number of records.
- B-trees are also used for secondary indexes in databases, where the index facilitates a select operation to answer some range criteria.
- Trees are used for compiler construction.
- Trees are also used in database design.
- Trees are used in file system directories.
- Trees are also widely used for information storage and retrieval in symbol tables.