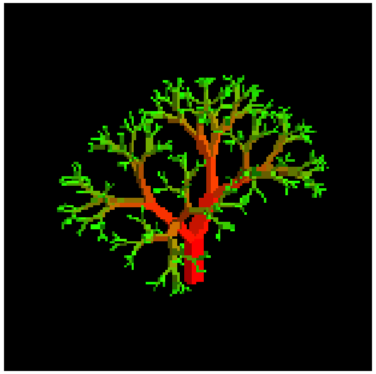
UNIT - IV

TREES

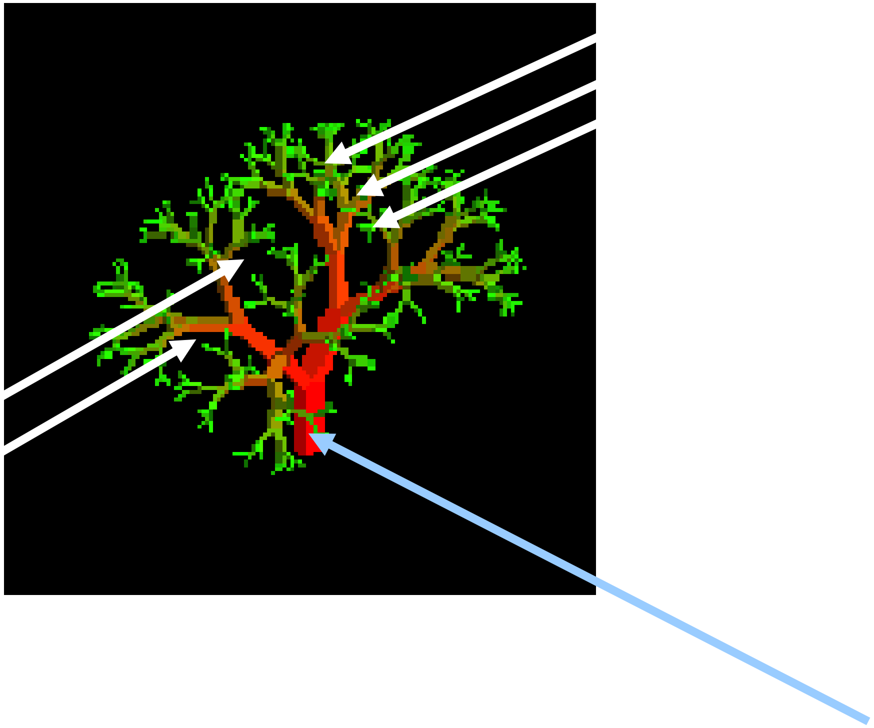
Tree Definition – Tree terminologies – General tree – Binary Tree – Tree traversal – Expression tree – Binary Search Tree – AVL Tree – Binary Heap.

Trees



Nature Lover's View Of A Tree

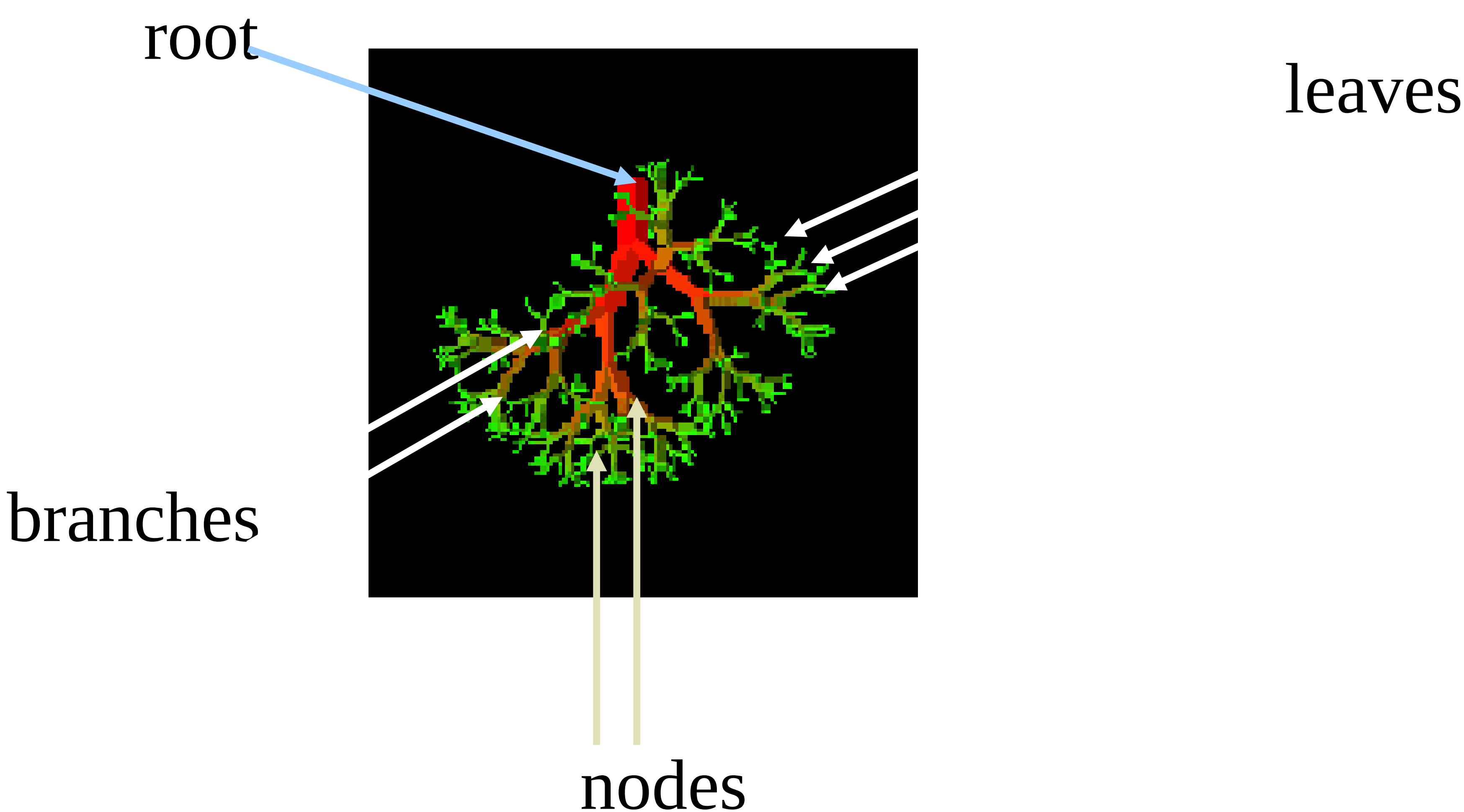
leaves



branches

root

Computer Scientist's View

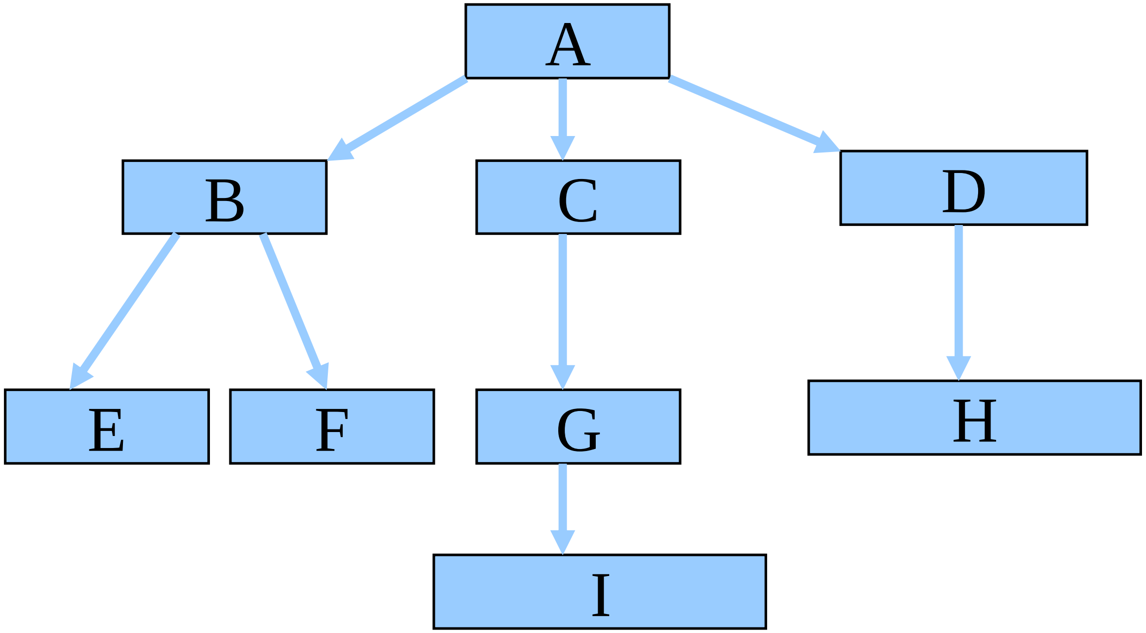


Linear Lists and Trees

- **Linear lists** are useful for **serially ordered data**.
  - Days of week.
  - Months in a year.
  - Students in this class.
- **Trees** are useful for **hierarchically ordered data**.
  - Employees of a corporation.
    - President, vice presidents, managers, and so on.
  - Java's classes.
    - Object is at the top of the hierarchy.
    - Subclasses of Object are next, and so on.

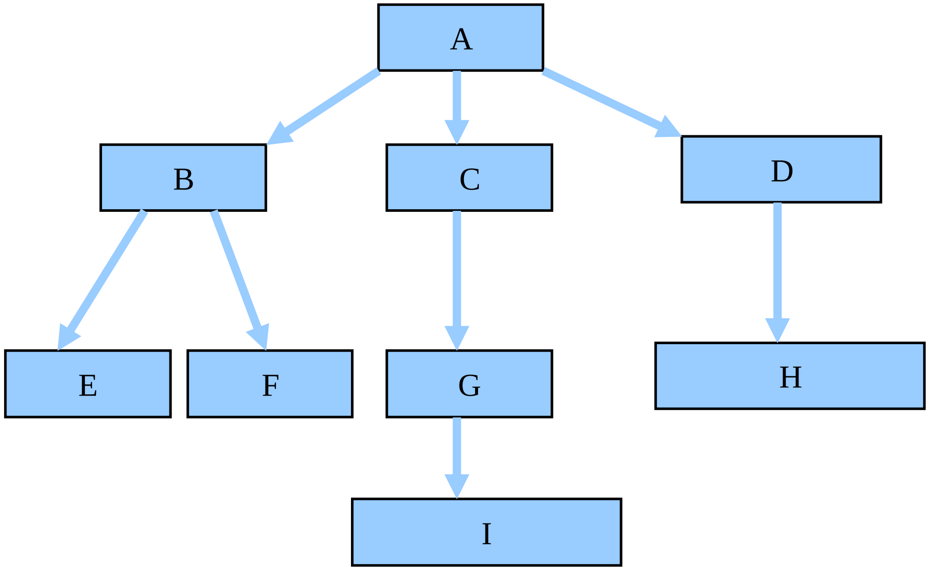
Definition

- A **tree** is a collection of nodes
- One of these node is called the root.
- The remaining nodes, if any, are partitioned into trees, which are called the **sub trees**



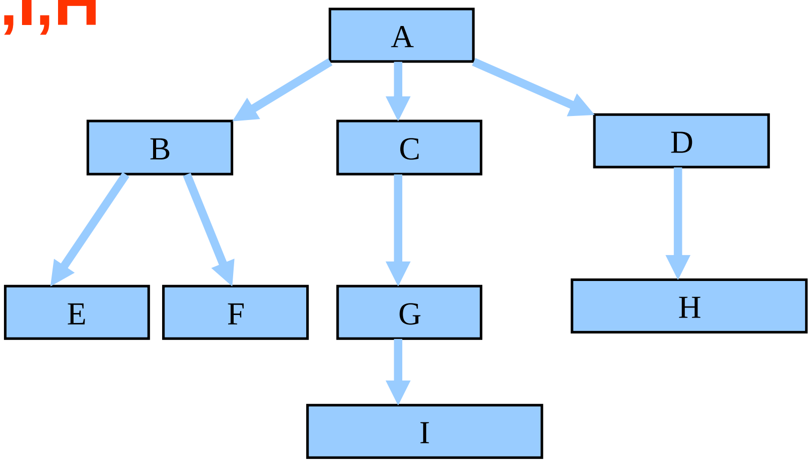
TREE TERMINOLOGIES

- Root
- Leaves or Terminal Nodes
- Siblings
- Grandparent
- Grandchild
- Path
- Length
- Node Degree
- Tree Degree
- Depth
- Height
- Ancestor & Descendent



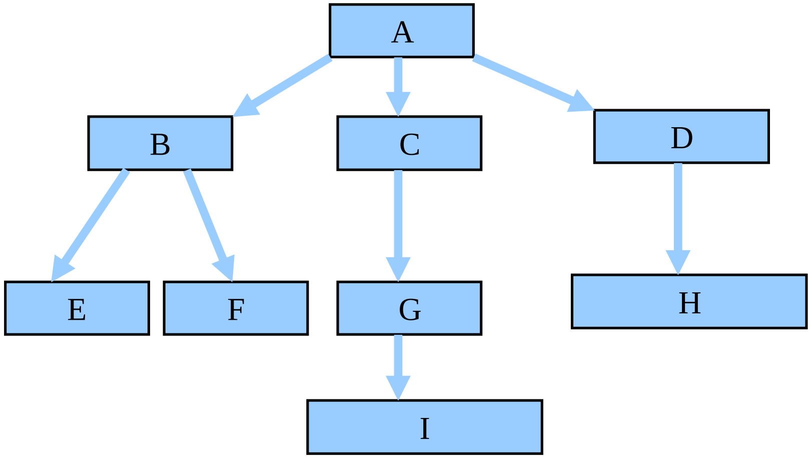
TREE TERMINOLOGIES

- **Root - A**
  - Nodes doesn't have a parent
- **Leaves or Terminal Nodes - E,F,I,H**
  - Nodes with no children
- **Siblings - B,C,D**
  - Nodes with same parent
- **Grandparent - A,C**
- **Grandchild - E,F,G,H,I**
- **Path**
  - A path from node n_1 to n_k is defined as a sequence of nodes n_1, n_2,……. n_k such that n_i is the parent of n_{i+1} for $1 \leq i < k$



TREE TERMINOLOGIES

- **Length**
  - Number of edges on the path
  - Ex: Path **A-C-G-I**
    Length **3**



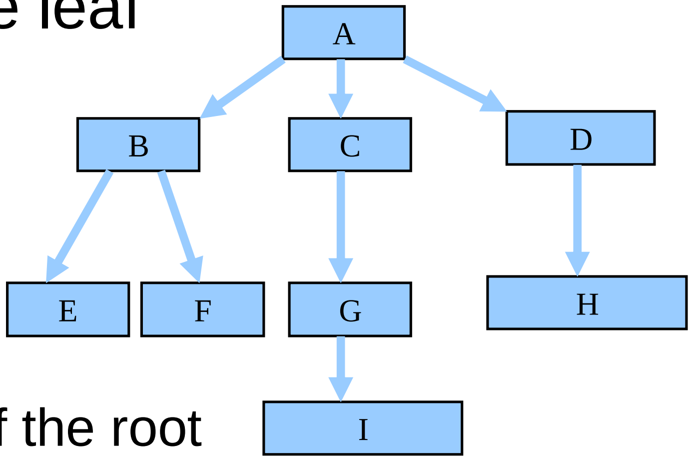
- **Node Degree** **A → 3**
  - Number of sub trees of a node
- **Tree Degree - 3**
  - Maximum degree of any node in the tree
- **Depth**
  - For any node n, the depth of the node n is the length of the unique path from root to n

    Root - **0**

    G - **2**

TREE TERMINOLOGIES

- **Height**
  - For any node n, the height of the node n is the length of the longest path from n to the leaf
    - A - **3**
    - G - **1**



NOTE:

- Height of the tree is equal to the height of the root
- Depth of the tree is equal to the height of the tree

- **Ancestor & Descendent**
  - There is a path from n1 to n2 then n1 is an ancestor of n2 and n2 is a descendent of n1
    - **A** Ancestor
    - **B** Descendent

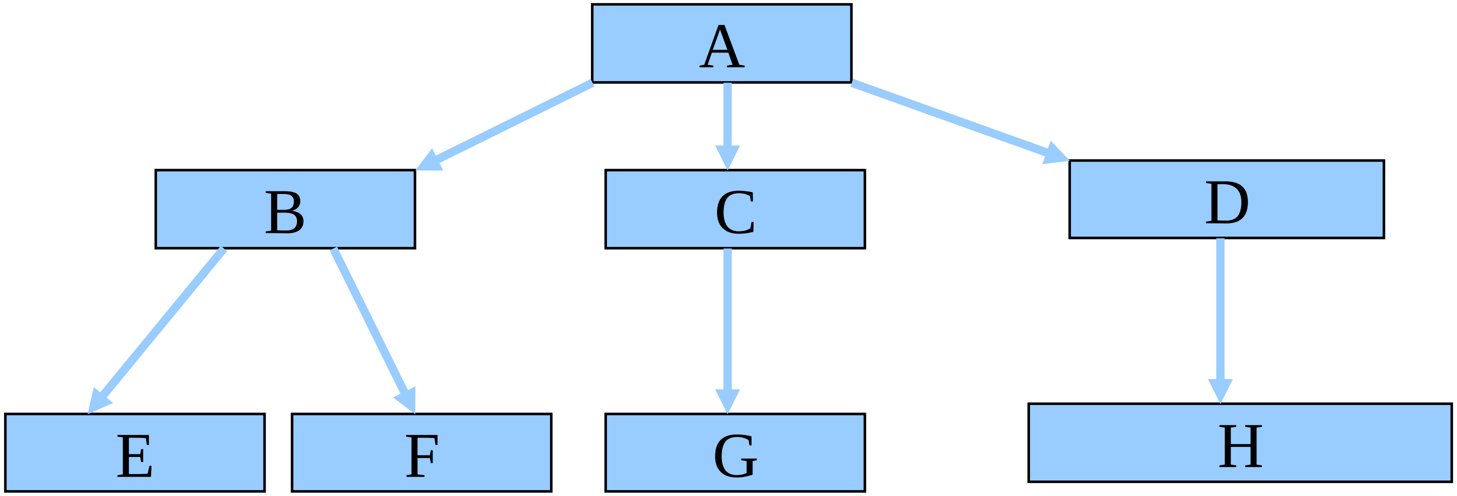
TREE TYPES

- General Tree
- Binary Tree
- Expression Tree
- Binary Search Tree
- Threaded Binary Tree
- AVL Tree
- Splay Tree
- B Tree
- B+ Tree
- Binary Heap

General Tree

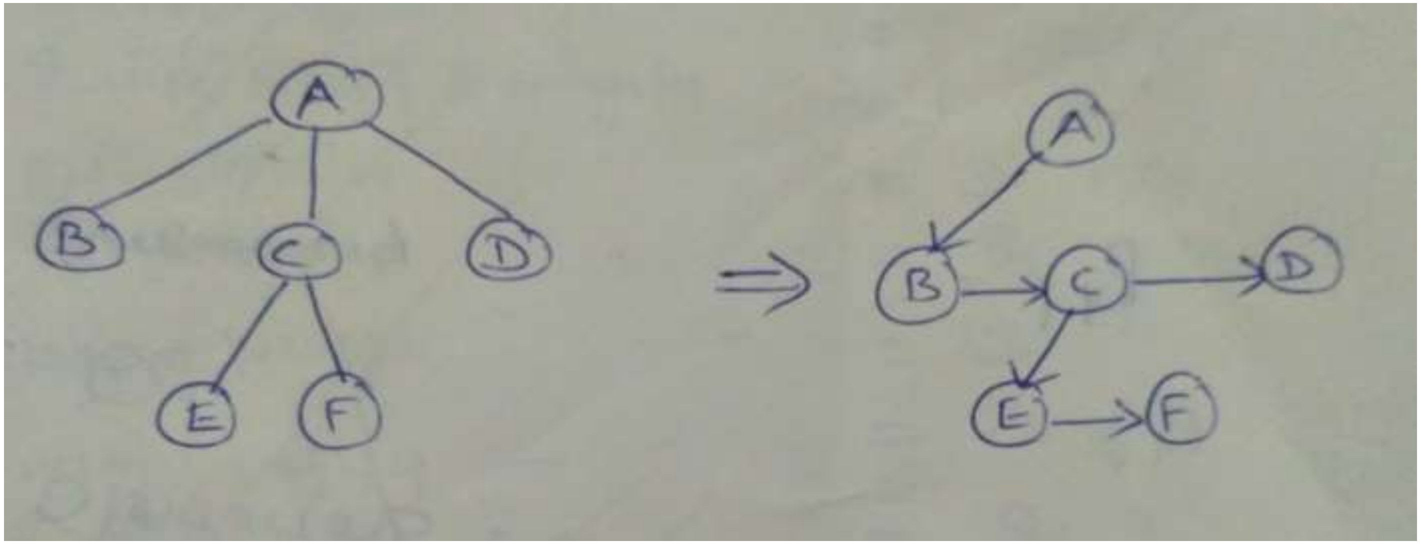
A **general tree** is a tree in which each node can have either zero or many child nodes

Ex:

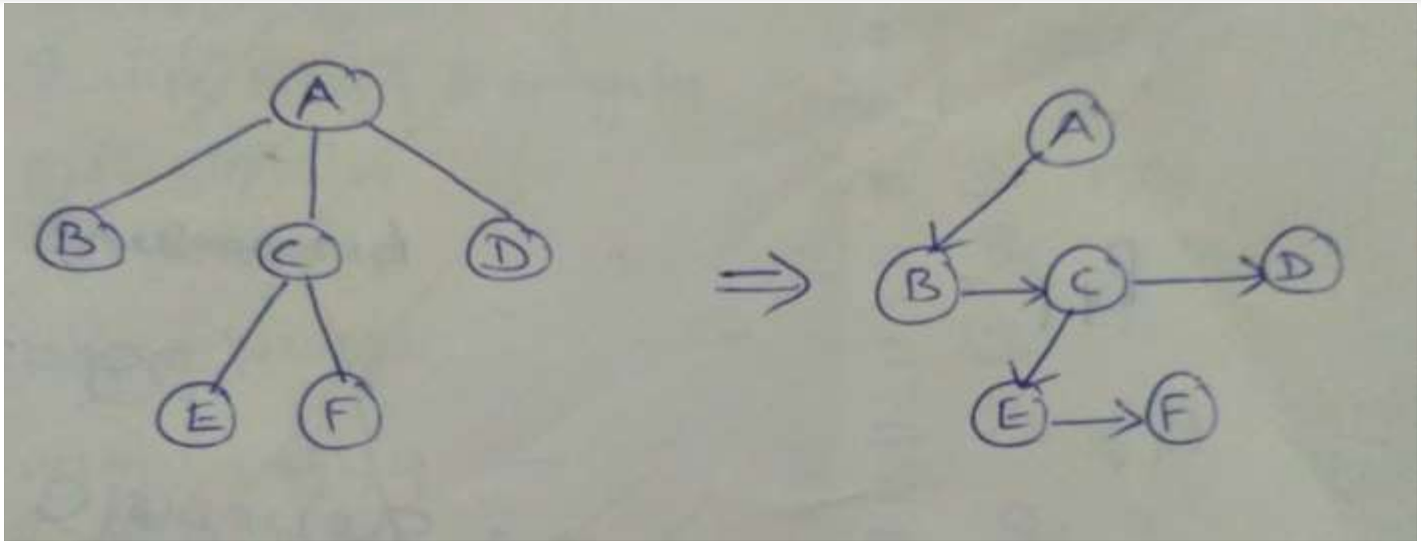


General Tree

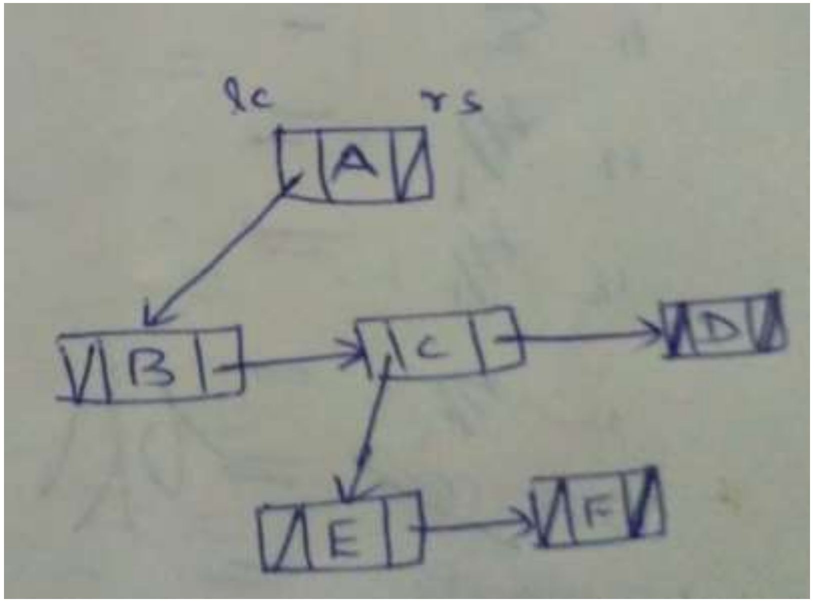
- We cannot aware of number of children, so no need to assign address for the child
- It can be implemented by using "**Left Child Right Sibling Data Structure**"
- From the root node, the left child has been directly linked, the remaining sibling nodes of the left child have been connected to the left child of the root



General Tree

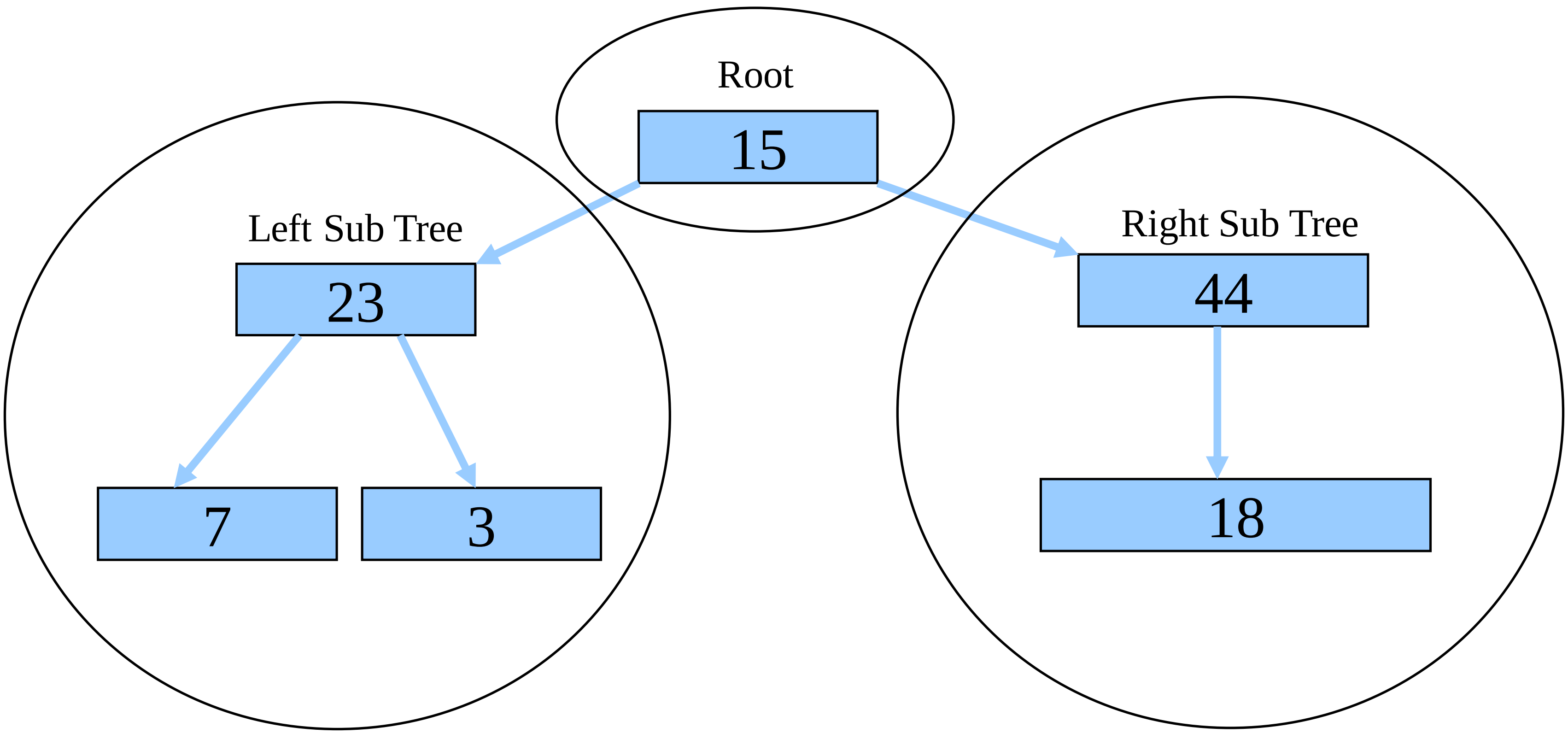


struct node
{
int data;
struct node *lchild;
struct node *rsibling;
};



Binary Tree

- A tree whose elements have **at most 2 children** (i.e., 0 or 1 or 2 child) is called a **binary tree.**
- Since each element in a binary tree can have only 2 children, we typically name them the left and right child.



Binary Tree Implementation / Representation

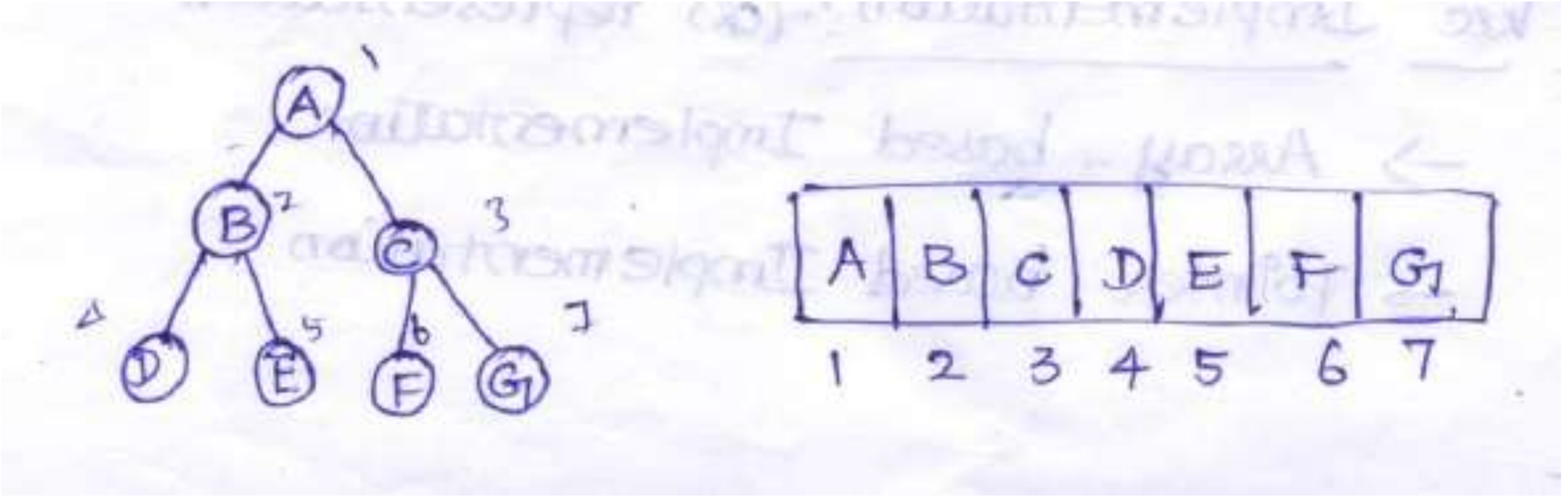
Binary Tree can Implemented in two ways

- Array Based Implementation (Linear Representation)
- Pointer Based Implementation (Linked Representation)

Binary Tree Implementation / Representation

Array Based Implementation (Linear Representation)

- Only Complete Binary Tree can be implemented by array



Binary Tree Implementation / Representation

Pointer Based Implementation (Linked Representation)

A tree is represented by a pointer to the topmost node in tree. If the tree is empty, then value of root is NULL.

A Tree node contains following parts.

1. Data

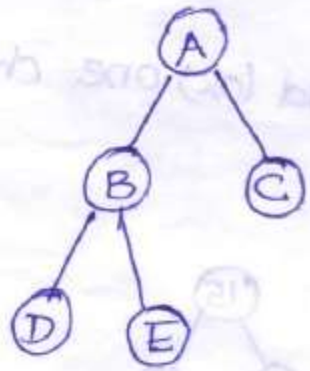
2. Pointer to left child

3. Pointer to right child

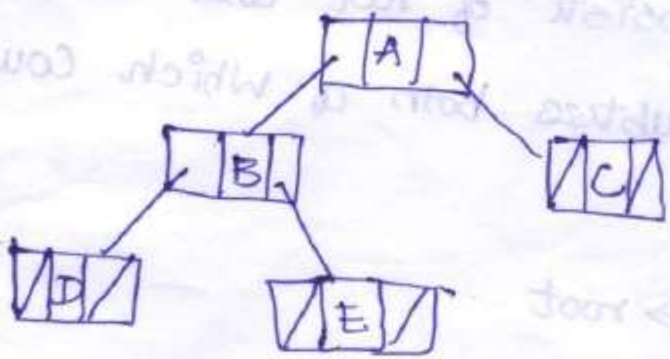
```
struct node
{
  int data;
  struct node *left;
  struct node *right;
};
```

Binary Tree Implementation / Representation

For ex :

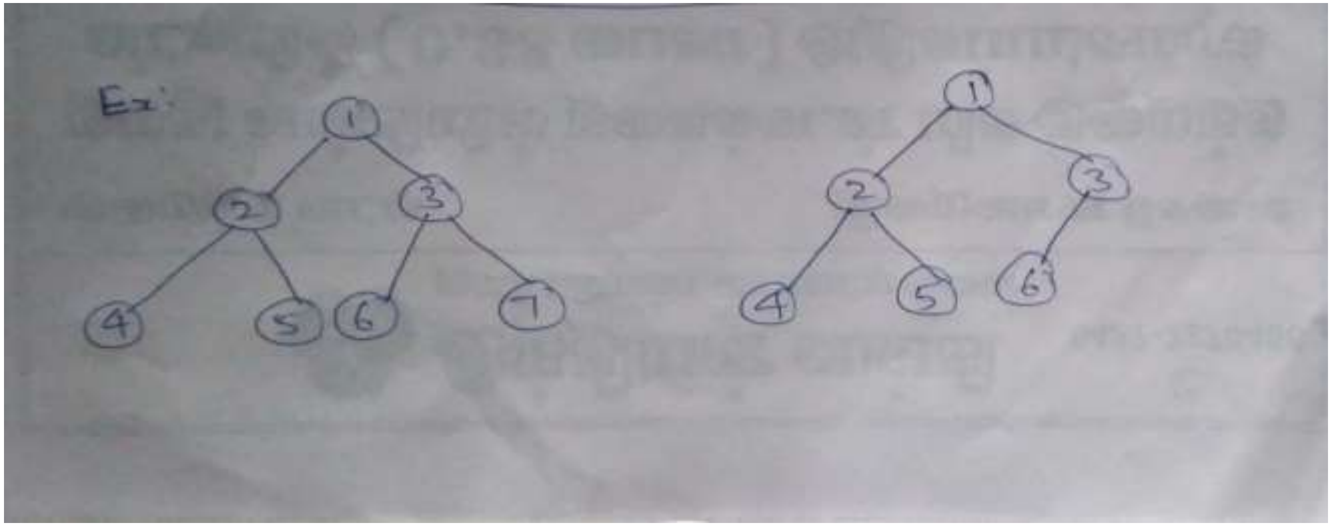


The above binary tree can be represented by the following structure:-



Complete Binary Tree

- A complete binary tree is a binary tree in which **every level, except the last**, is completely filled, and all nodes are **as far left as possible**.
- The position of root, left child and right child will be N,2N,2N+1 respectively
- A complete binary tree of height h has nodes between 2^h to 2^{h+1}-1



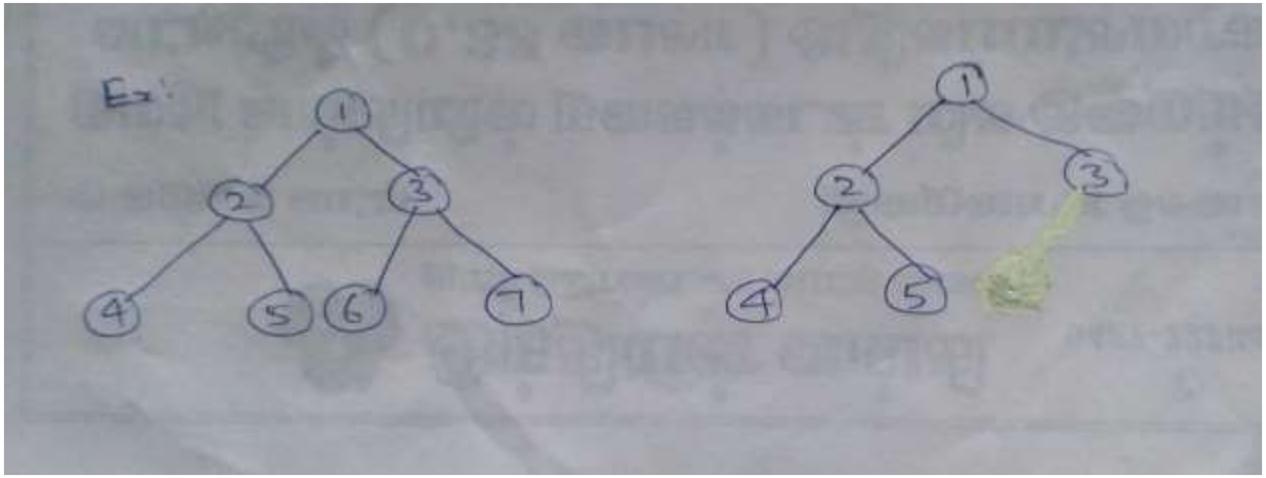
Full Binary Tree

Full Binary Tree or Proper Binary Tree or 2-Tree

- A full binary tree is a binary tree in which **every node other than the leaves has two children**.

or

- A full binary tree is a binary tree in which **every node has zero or two children**.



Traversing a Binary Tree

- Traversing a binary tree is the process of **visiting each node in the tree exactly once** in a systematic way.
- There are three different algorithms for tree traversals, which differ in the order in which the nodes are visited.
- These algorithms are:

✓ **Pre-order algorithm**

✓ **In-order algorithm**

✓ **Post-order algorithm**

In-order Algorithm

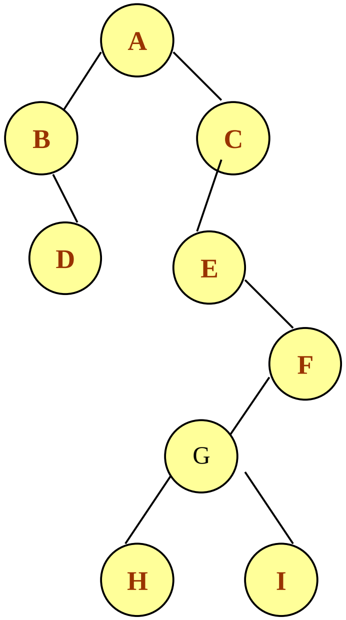
- To traverse a non-empty binary tree in **in-order**, the following operations are performed recursively at each node.
- The algorithm starts with the root node of the tree and continues by,

✓ Traversing the left subtree

✓ Visiting the root node

✓ Traversing the right subtree

```
void inorder(struct node *ptr )
{
   if(ptr != NULL)
   {
          Inorder(ptr -> left);
          printf("%d ",ptr -> data);
          inorder(ptr -> right);
   }
}
```



B, D, A, E, H, G, I, F and C

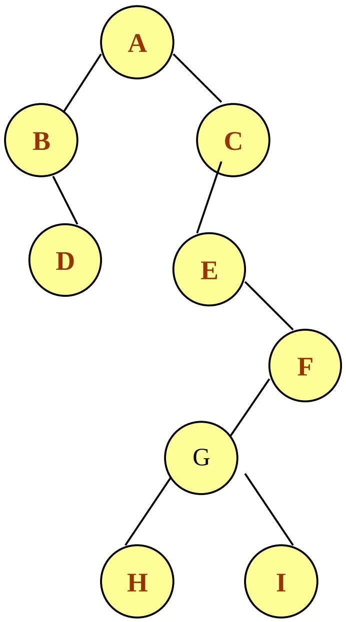
Pre-order Algorithm

- To traverse a non-empty binary tree in **pre-order**, the following operations are performed recursively at each node.
- The algorithm starts with the root node of the tree and continues by,

✓ Visiting the root node

✓ Traversing the left subtree

✓ Traversing the right subtree



A, B, D, C, E, F, G, H and I

```
void preorder(struct node *ptr )
{
   if(ptr != NULL)
   {
          printf("%d ",ptr -> data);
          preorder(ptr -> left);
          preorder(ptr -> right);
   }
}
```

Post-order Algorithm

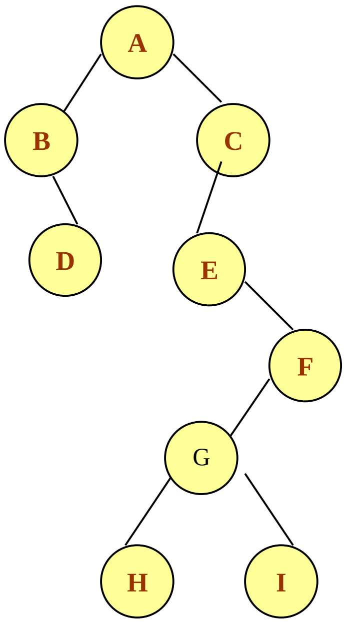
- To traverse a non-empty binary tree in **pre-order**, the following operations are performed recursively at each node.
- The algorithm starts with the root node of the tree and continues by,

✓ Traversing the left subtree

✓ Traversing the right subtree

✓ Visiting the root node

```
void postorder(struct node *ptr )
{
   if(ptr != NULL)
   {
          postorder(ptr -> left);
          postorder(ptr -> right);
          printf("%d ",ptr -> data);
   }
}
```

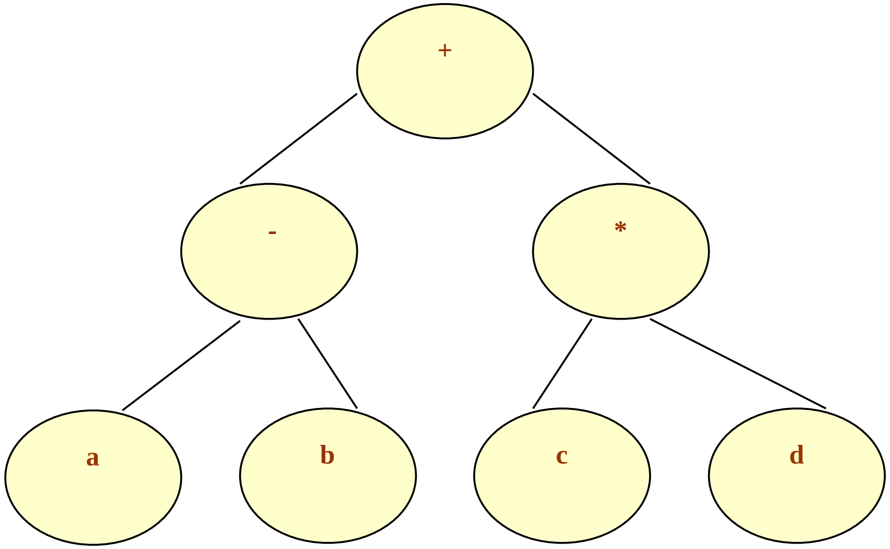


D, B, H, I, G, F, E, C and A

Expression Trees

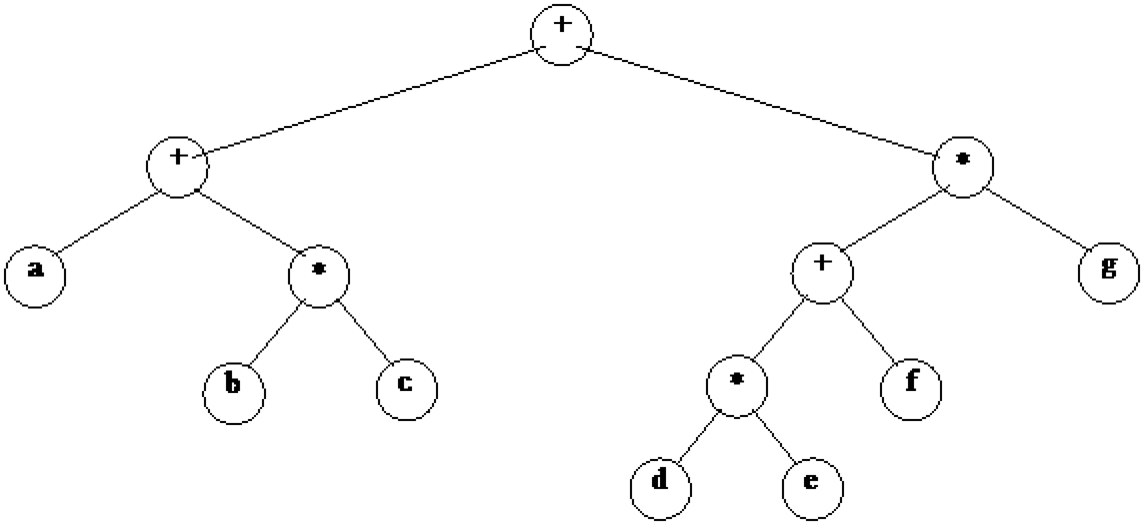
- Binary trees are widely used to store algebraic expressions. For example, consider the algebraic expression Exp given as:

  Exp = (a – b) + (c * d)
- This expression can be represented using a binary tree as shown in figure



Expression Trees

- (a+b*c)+((d*e+f)*g)



Construction of Expression Tree from Postfix Expression

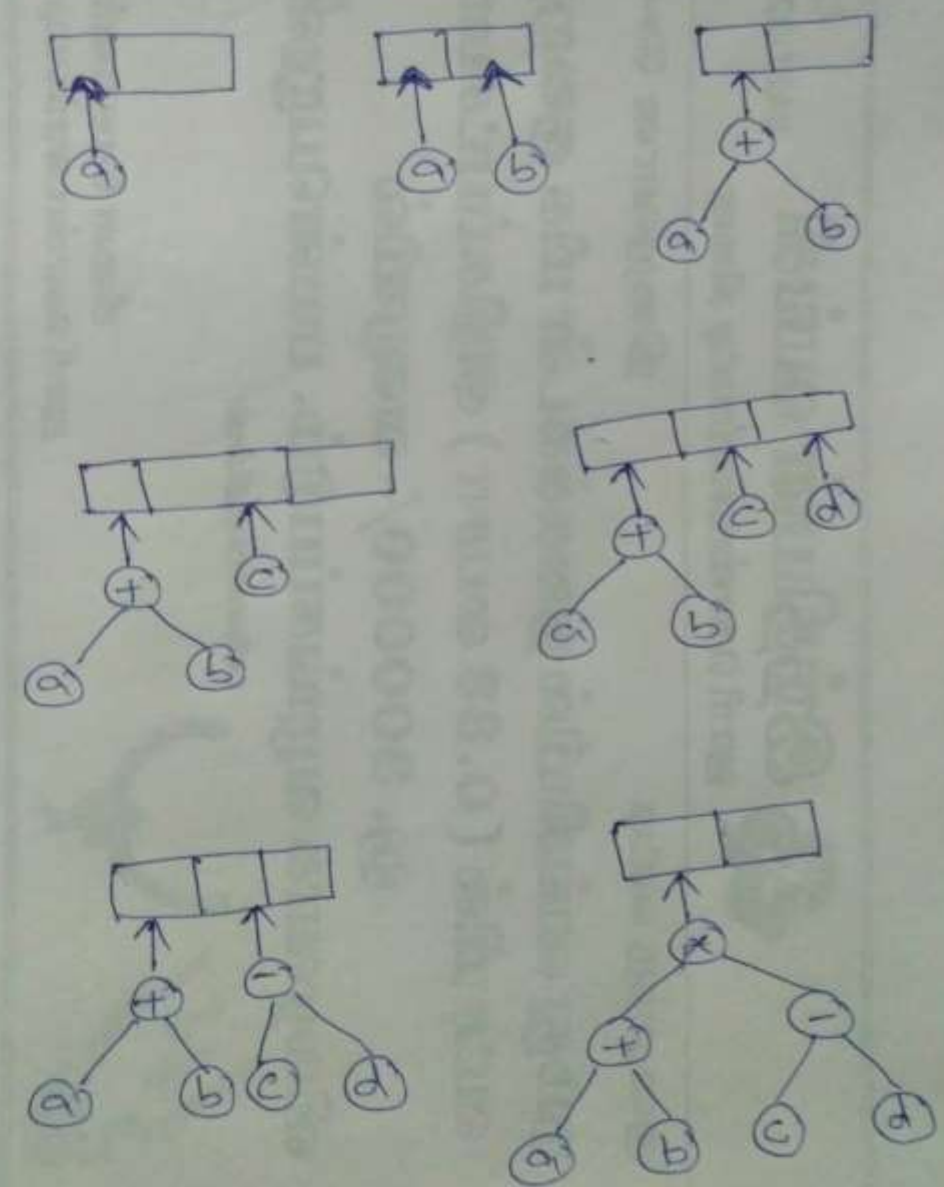
Step 1: Add # symbol into the end of the postfix expression

Step 2: Read the postfix expression one by one symbol

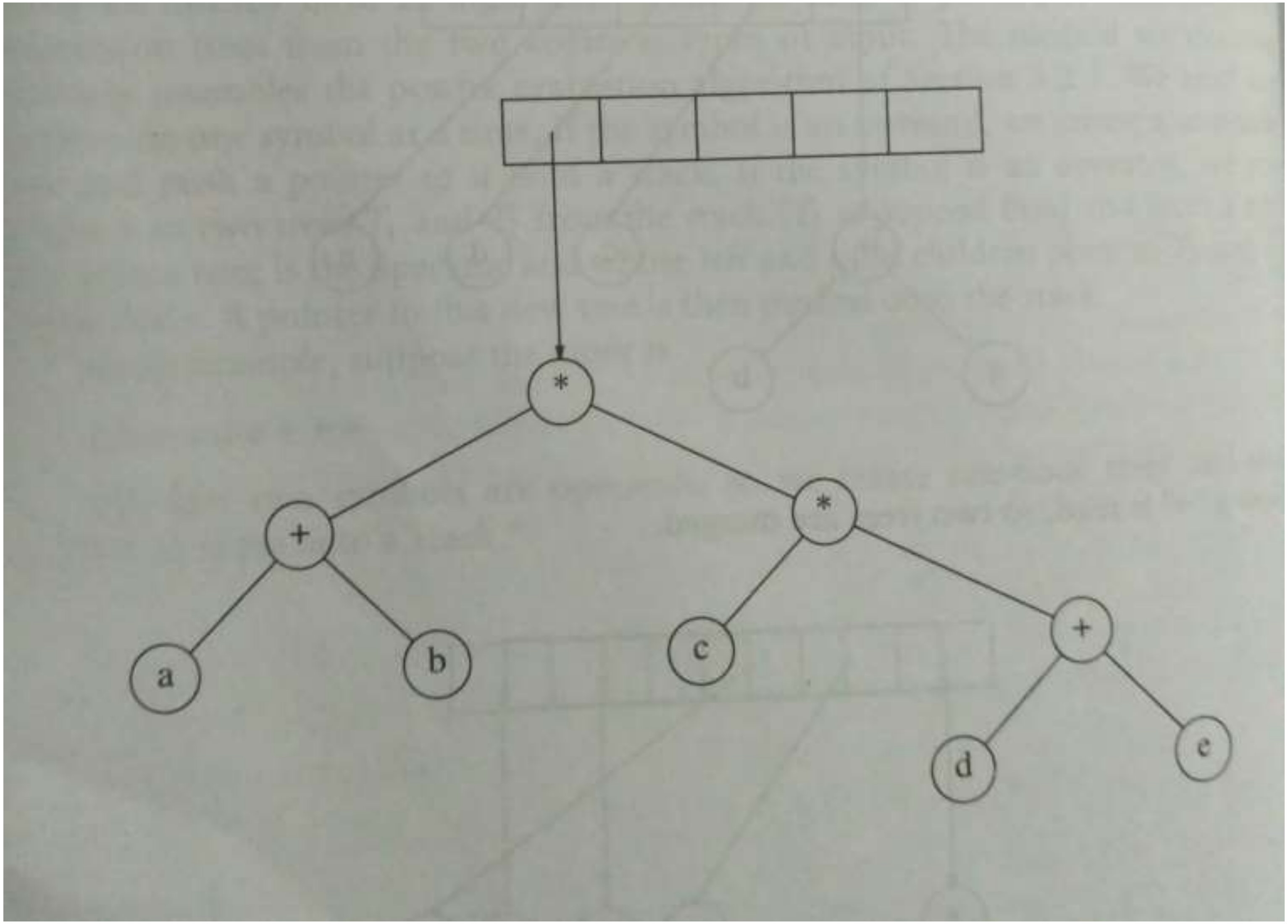
Step 2a: If symbol is an operand push that into stack

Step 2b: If symbol is an operator pop two values from stack make them its child and push current symbol again into the stack

Step 3: Repeat the same process again and again until unique symbol # comes ab+cd-*



ab+cde+**



Applications of Trees

- Trees are used to **store simple(int,char) as well as complex data(structure)**.
- Trees are often used for **implementing other types of data structures like hash tables, sets, and maps.**
- Red-black tree is used in **kernel scheduling to preempt massively multi-processor computer operating system use**.
- 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**.