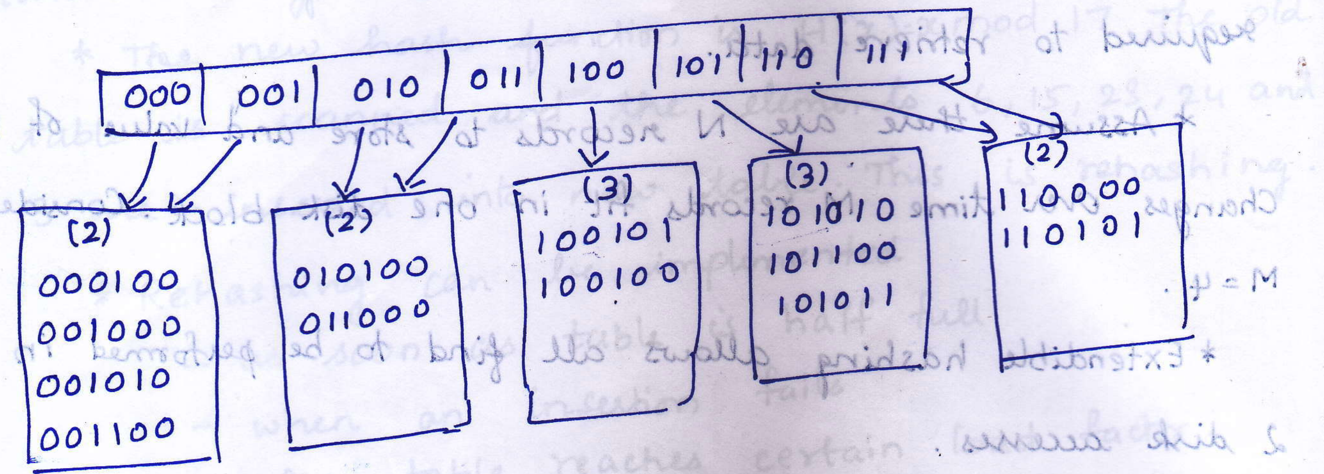
* Suppose we want to insert key 100100. This would go into 3rd leaf but it is already full. Thus, split the leaf into 2 leaves, which are determined by first 3 bits. So increase directory size by 3.



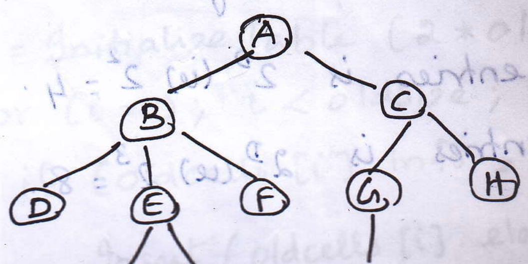
UNIT - ~~IV~~ III

Tree - Tree ADT - Tree Traversals - Binary Tree ADT - expression tree - Applications of trees - Binary Search Tree ADT - Threaded Binary Trees - AVL Trees - B-Tree - B+-Tree - Heap - Applications of heap.

TREE ADT:

* Tree is a non-linear data structure that organizes data in hierarchical order. Tree data structure is a collection of nodes (data) organized in hierarchical structure.

* Every individual element is called as Node. In tree, if N nodes are present then it can have maximum of N-1 number of links.



Terminology :

Root : The first node in a tree is called root node. Every tree must have a root node. which is the origin.

In the eg. A is root.

Edge : The connecting link between any 2 nodes is called edge. A tree with N nodes contains maximum of N-1 number of edges.

Parent : The node that is predecessor of any node is called as parent node. It is also defined as the node with child / children.

In the eg. A, B, C, E & G are parent nodes.

Child : The node that is descendent of any node is called as child node. A parent node can have any number of child nodes. In a tree, all the nodes except root are child nodes.

In the eg, B & C are children of A

G & H are children of C

K is child of G.

Siblings : The nodes that are child to same parent are called as siblings.

In the eg, B & C, D, E & F, G & H, I & J are siblings.

Leaf : The node that does not have a child is called as leaf node. Leaf nodes are also called as external nodes. External nodes are nodes with no child.

In the eg, D, I, J, F, K & H are leaf nodes.

External nodes are also called as Terminal nodes.

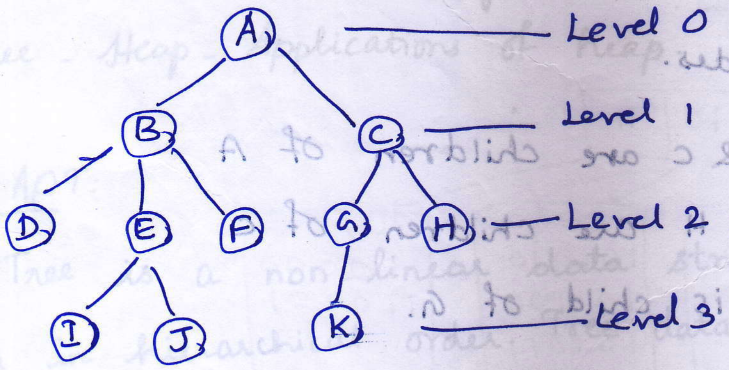
Internal nodes: The node with atleast 1 child is called as Internal node. Nodes other than leaf nodes are called internal nodes. They are also called as Non-terminal nodes.

In the eg, A, B, C, E & G are Internal nodes.

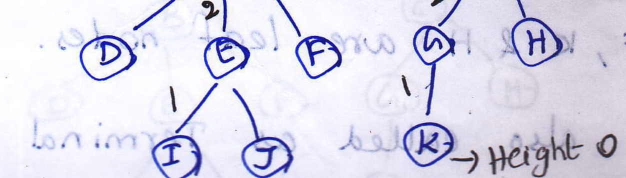
Degree: The degree of a node is the the total number of children of that node. The highest degree of a node among all nodes in a tree is called degree of tree.

In the eg. degree of B is 3.
degree of A, C is 2.
degree of F is 0.

Level: In a tree, each step from top to bottom is called as a level and level count starts from 0 & it incremented by one at each step. Root node is at level 0 & its child at level 1 & so on.

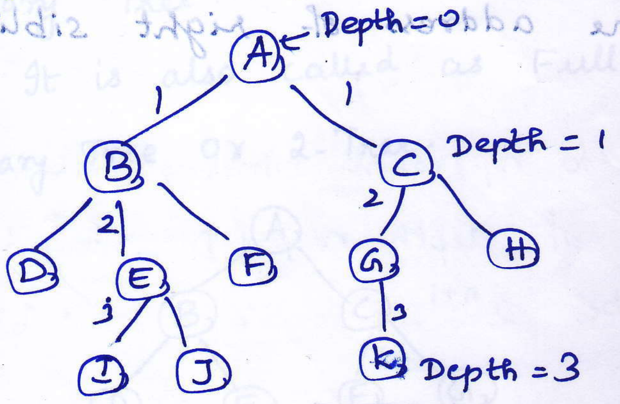


Height: Height of any node is the total no. of edges from leaf to that node in the longest path. Height of tree is the height of root node. Height of all leaf nodes is 0.



Depth: The depth of a node is the total no. of edges from root to that node. Depth of tree is total no. of edges from root to leaf node in the longest path.

Depth of root node is 0.

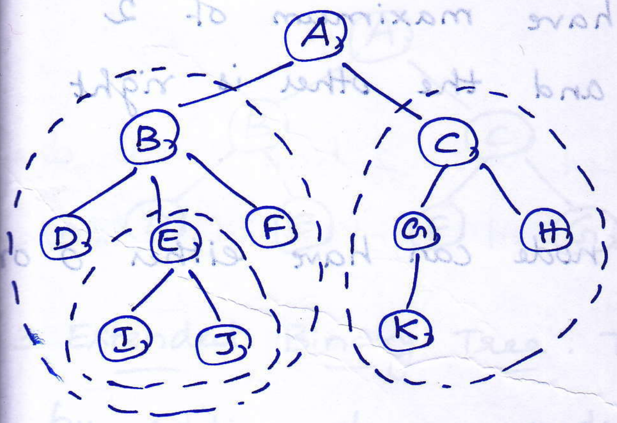


Path: The sequence of nodes and edges from one node to another node is called path.

In the eg. path from A to J is A - B - E - J.

Length of path is total no. of nodes in that path. For above eg. Length is 4.

SubTree: Every child from a node forms a subtree recursively to its parent.



TREE REPRESENTATION:

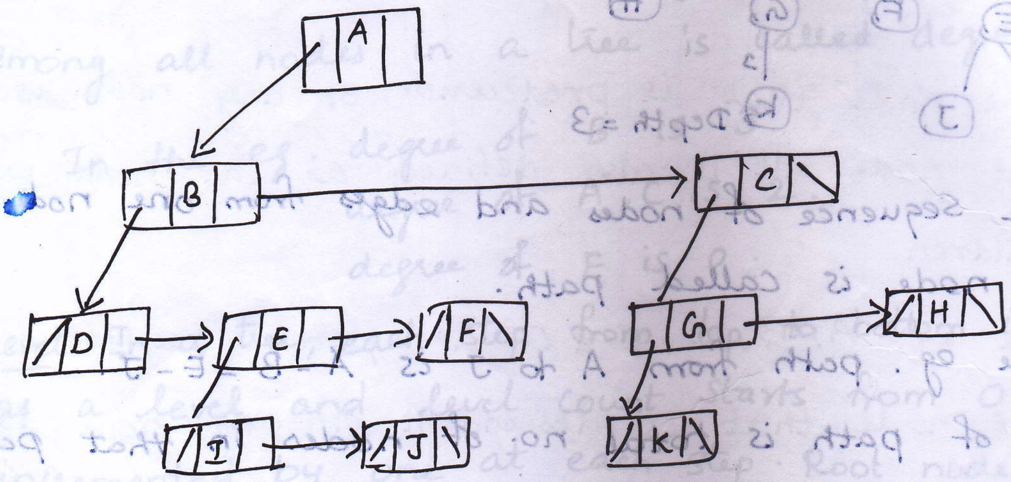
Left Child - Right Sibling Representation:

* It uses a node with 3 fields namely Data field, Left child reference field and Right Sibling reference field.

Data	
Left	Right

* Every node's data field stores the actual value of that node. If that node has left child, then left reference field stores address of that left child otherwise NULL. If that node has right sibling, then right reference field stores the address of right sibling node otherwise NULL.

Eg:

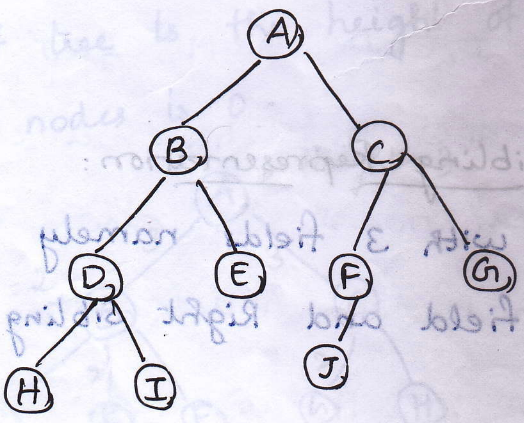


BINARY TREE.

* In a tree, a node can have any number of children. Binary tree is a special type of tree in which every node can have maximum of 2 children. One is left child and the other is right child.

* In a binary tree, every node can have either 0 or 1 child or 2 children.

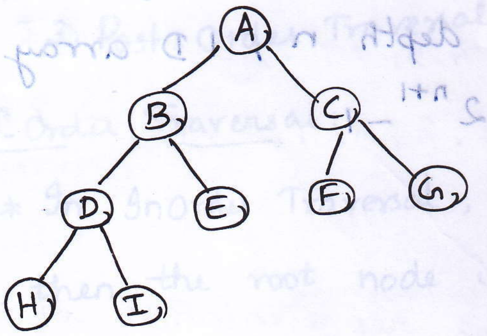
eg.



Types of Binary Tree :

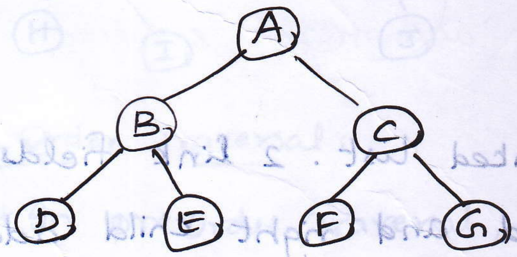
Strictly Binary Tree : A binary tree in which every node has either two or zero children is called Strictly Binary Tree.

It is also called as Full Binary Tree or Proper Binary Tree or 2-Tree.

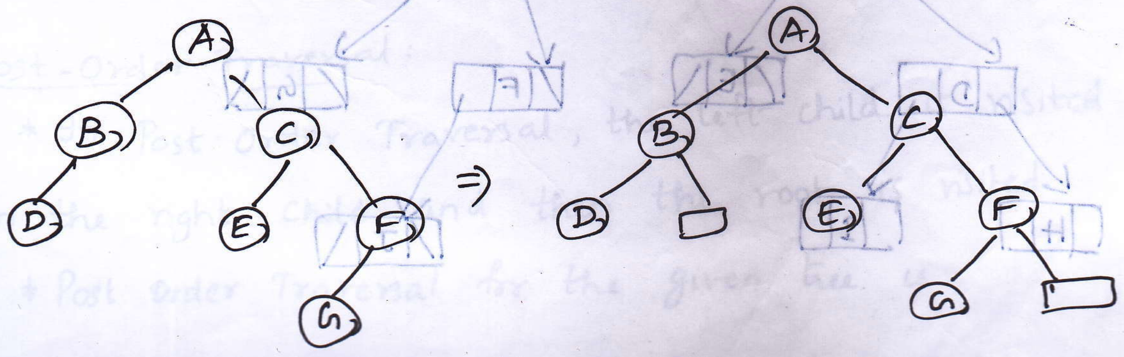


2. Complete Binary Tree : A binary tree in which every internal node has exactly 2 children and all the leaf nodes are at the same level is called Complete Binary Tree.

It is also called as Perfect Binary Tree.



3. Extended Binary Tree : The full binary tree obtained by adding dummy nodes to a binary tree is called as Extended Binary Tree.



Binary Tree Representation :

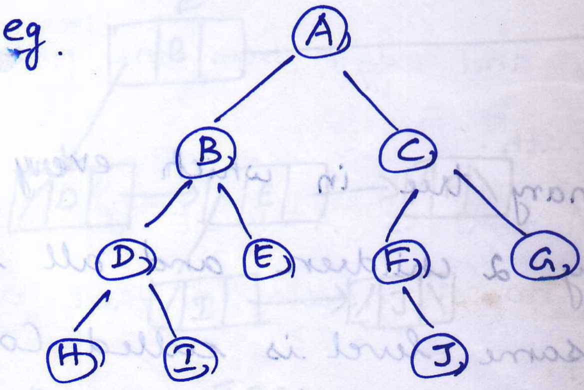
* A Binary Tree can be represented using 2 method

1. Array Representation.
2. Linked List Representation.

Array Representation :

* It uses 1D array to represent binary tree. To represent binary tree of depth n, 1D array maximum size should be $2^{n+1}-1$

eg.



Depth = 3 . $\therefore$ Array Size = $2^{3+1}-1 = 2^4-1 = 16-1 = 15$

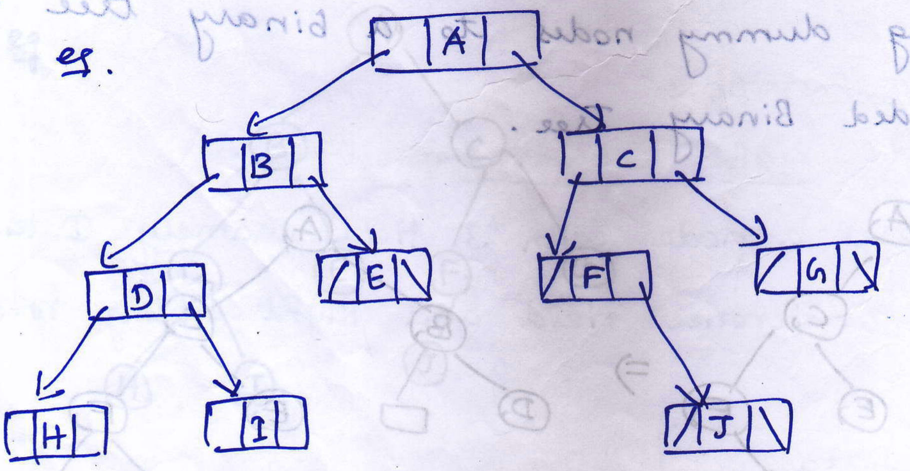
A	B	C	D	E	F	G	H	I	-	-	-	J	-	-

Linked List Representation :

* It uses double linked list. 2 link fields are used to store left child and right child address.

Left Child	Data	Right Child

eg.



TREE TRAVERSALS.

* Tree traversal specifies the order in which all the nodes of the binary tree must be visited & displayed.

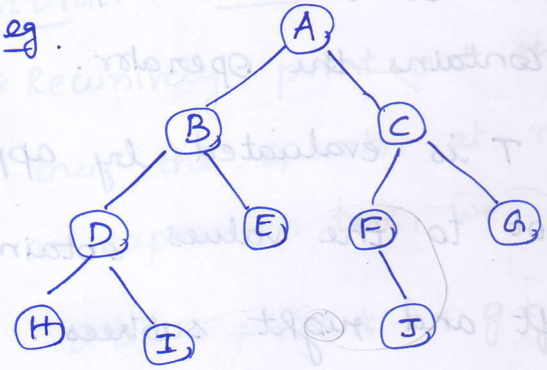
* There are three types of binary tree traversal

1. In-Order Traversal
2. Pre-Order Traversal
3. Post-Order Traversal.

1. In-Order Traversal:

* In InOrder Traversal, the left child node is visited first, then the root node is visited and at last right child node is visited. The traversal is performed recursively for all nodes in the tree.

eg.



Inorder Traversal for given tree is.

H - D - I - B - E - A - F - J - C - G

2. Pre-Order Traversal:

* In preOrder Traversal, the root node is visited first, then its left child and then its right child. The traversal is recursively applied on all subtrees.

* PreOrder Traversal for given tree is

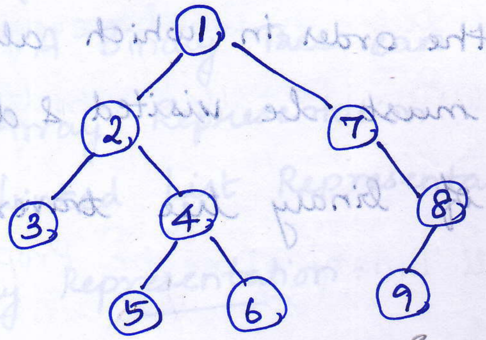
A - B - D - H - I - E - C - F - J - G

3. Post-Order Traversal:

* In Post Order Traversal, the left child is visited first then the right child and then the root is visited.

* Post order Traversal for the given tree is

Example:



InOrder Traversal : L Ⓡ R 3, 2, 5, 4, 6, 1, 7, 9, 8 → L Ⓡ R.

PreOrder Traversal : Ⓡ L R 1, 2, 3, 4, 5, 6, 7, 8, 9 → L Ⓡ Ⓡ

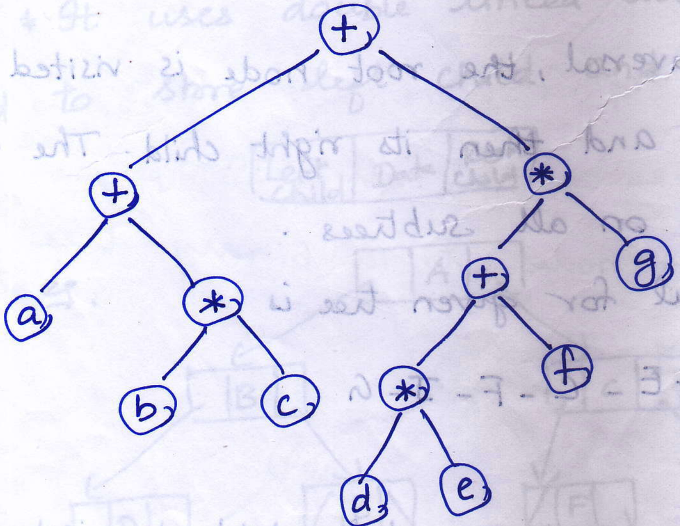
PostOrder Traversal : L R Ⓡ 3, 5, 6, 4, 2, [illegible], 9, 8, 7, 1 → Ⓡ L R

EXPRESSION TREE:

* An expression tree is a representation of expressions arranged in a tree data structure. Leaves of an expression tree contains the operands and the internal node contains the operator.

* An expression tree T is evaluated by applying the operator at the root to the values obtained by recursively evaluating left and right subtrees.

Eg.



(1) InOrder Traversal :

* Recursively print out the left subtree, then the root & then right subtree. On applying this traversal on an expression tree, we get an infix expression.

a + b * c + d * e + f * g

(2) Pre Order Traversal :

* Print out the operator at root first, then recursively print out left and right subtrees. On applying this traversal on an expression tree, we get a prefix expression.

+ + a * b c * + * d e f g

(3) PostOrder Traversal :

* Recursively print out the left subtree, right subtree and then the operator at root. On applying this traversal on an expression tree, we get a postfix expression.

a b c * + d e * f + g * +

Constructing an Expression Tree :

Algorithm to convert a postfix expression into an expression tree.

Steps:

(i) Read out expression one symbol at a time.

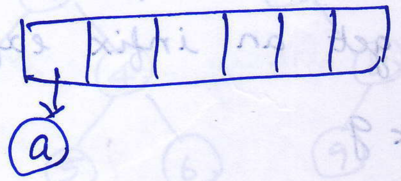
ir) If symbol is an operand, create a one-node tree and push a pointer to it on stack

ii) If symbol is an operator, pop pointers to 2 trees T_1 (popped first) & T_2 from stack. Form a new tree whose root is operator and left & right children

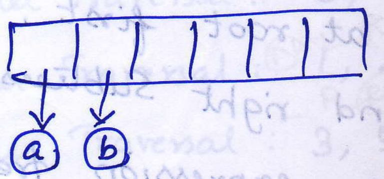
Eg:

a b + c d e + * *

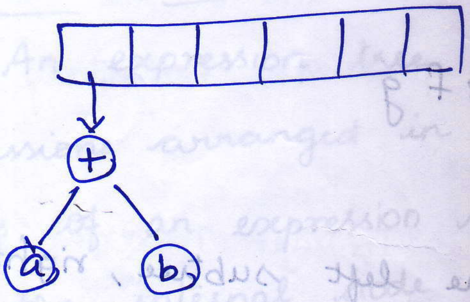
(*) 1st symbol - a - operand.



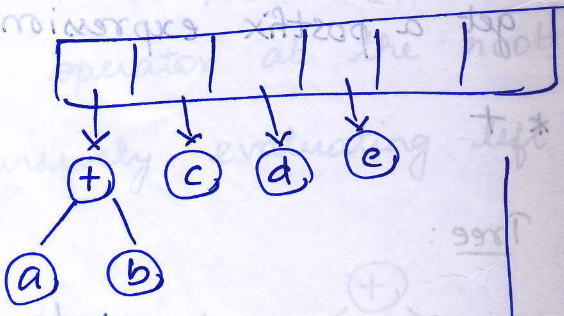
(*) b - operand



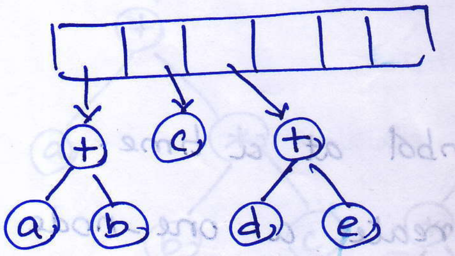
(*) + - operator



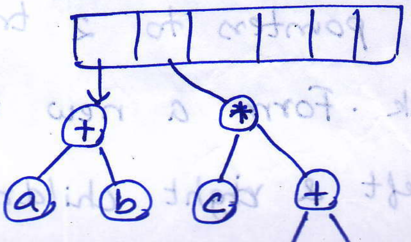
(*) c, d, e - operands.



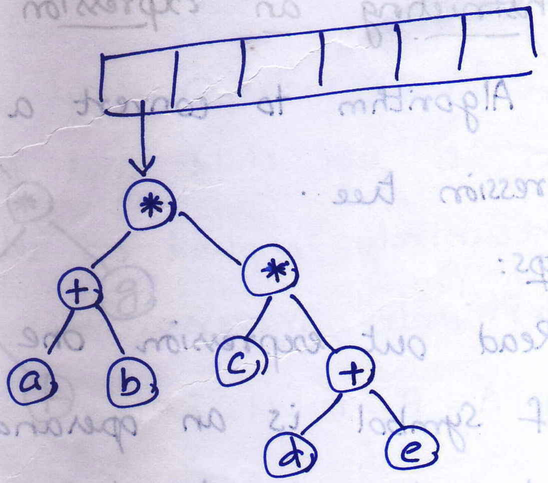
(*) + - operator.



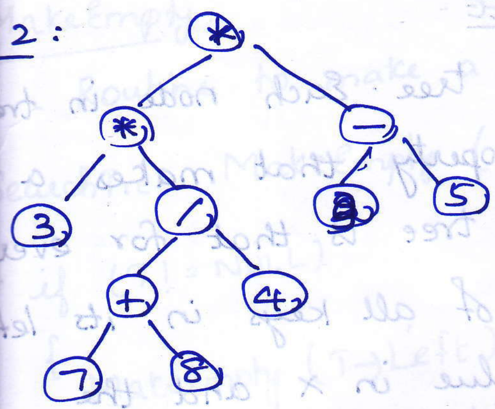
(*) * - operator



(*) * - operator.



2:



Infix Expression :

3 * 7+8 / 4 * 9 - 5 .

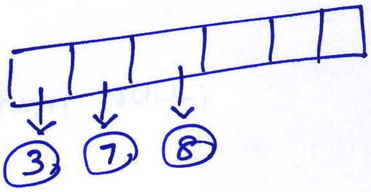
Prefix Expression :

* * 3 / + 7 8 4 − 9 5

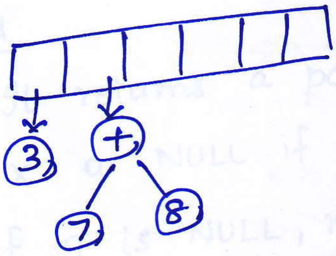
Postfix Expression :

3 7 8 + 4 / * 9 5 − * .

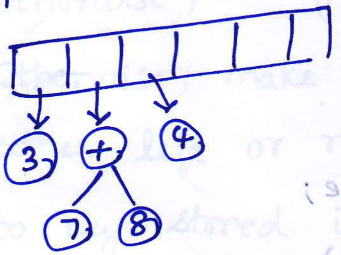
→ 3, 7, 8



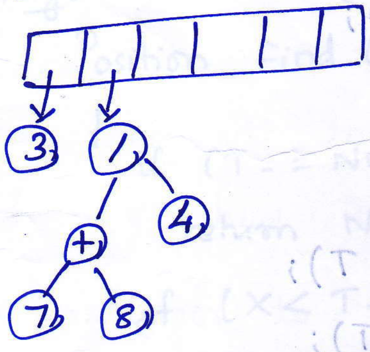
→ +



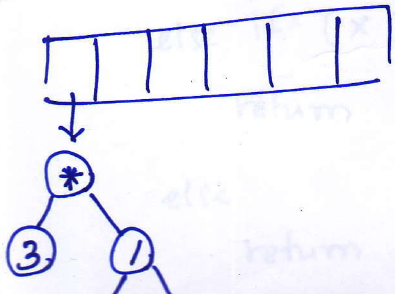
→ 4



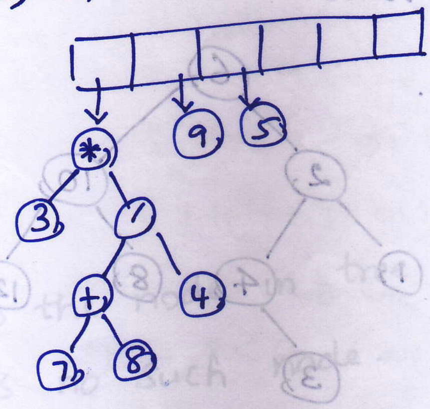
→ /



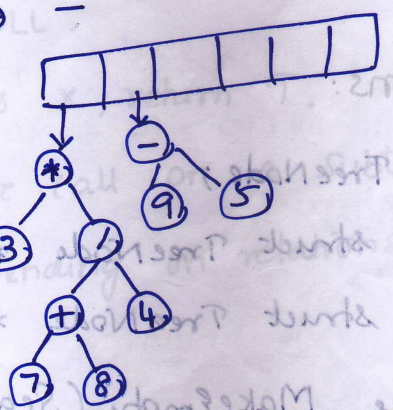
→ *



→ 9 , 5



→ −



→ *