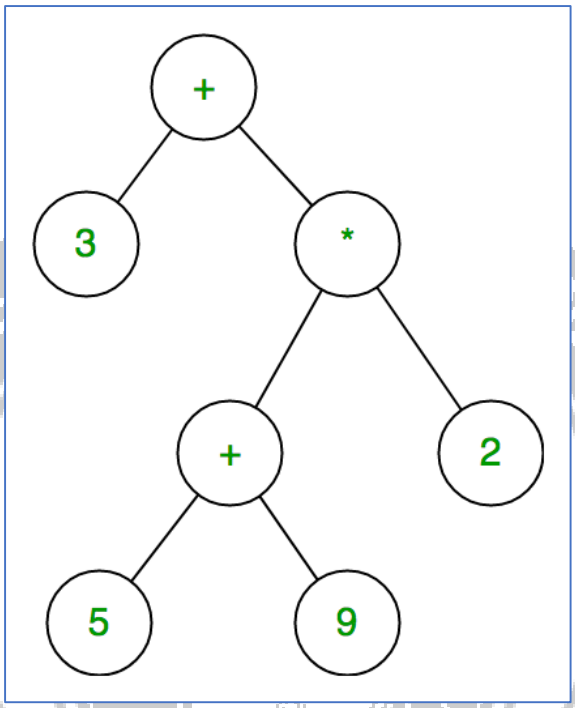
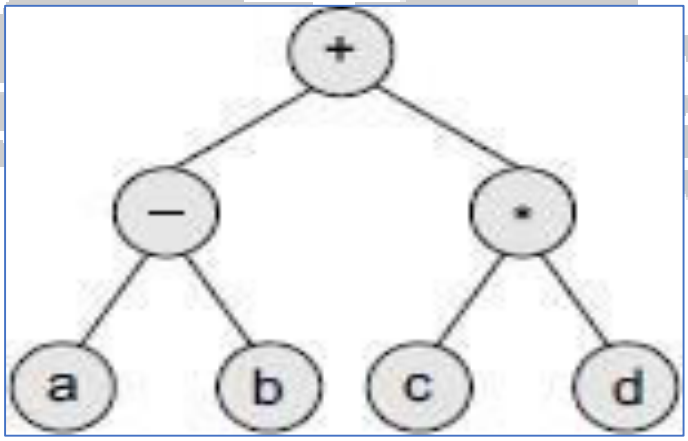
Expression Trees

The expression tree is a binary tree in which each internal node corresponds to the operator and each leaf node corresponds to the operand so for example expression tree for 3 + ((5+9)*2) would be:



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

Exp = (a – b) + (c * d)



Evaluating the expression represented by an expression tree:

Let t be the expression tree

If t is not null then

If t.value is operand then

Return t.value

A = solve(t.left)

B = solve(t.right)

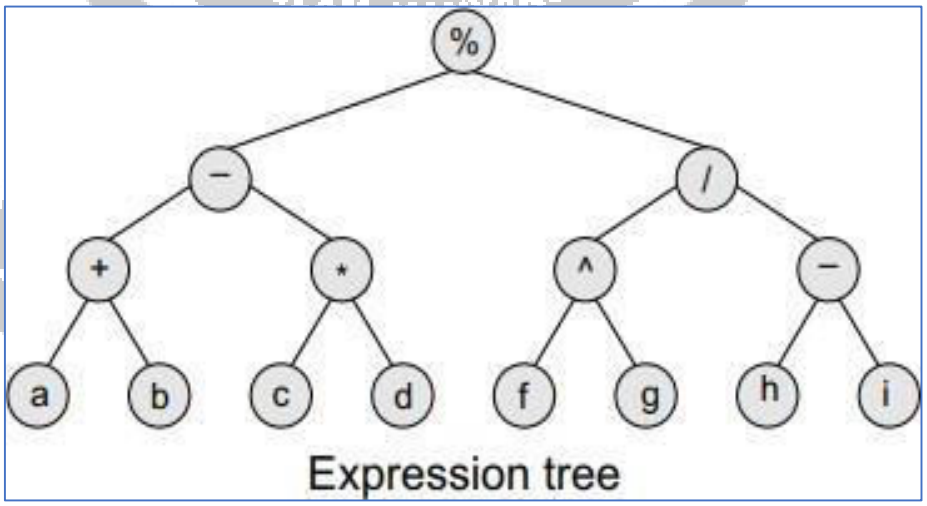
// calculate applies operator 't.value'

// on A and B, and returns value

Return calculate(A, B, t.value)

Example 1. Given an expression, Exp = ((a + b) – (c * d)) % ((e ^f) / (g – h)), construct the corresponding binary tree.

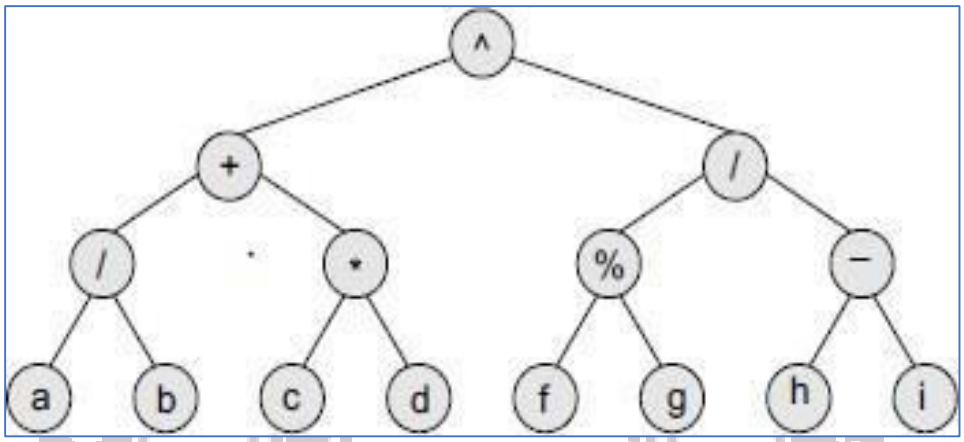
Solution



Expression tree

Example 2. Given the binary tree, write down the expression that it represents.

Solution



Construction of Expression Tree:

Now for constructing an expression tree we use a stack. We loop through input expression and do the following for every character.

- If a character is an operand push that into the stack
- If a character is an operator pop two values from the stack make them its child and push the current node again.

In the end, the only element of the stack will be the root of an expression tree.

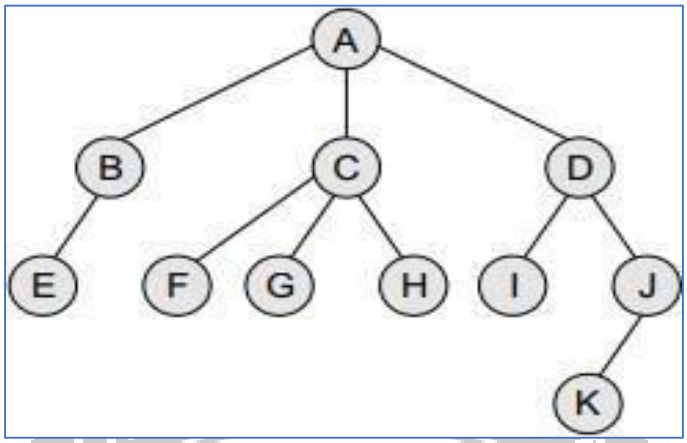
Creating a Binary Tree from a General Tree

The rules for converting a general tree to a binary tree are given below. Note that a general tree is converted into a binary tree and not a binary search tree.

Rule 1: Root of the binary tree = Root of the general tree

Rule 2: Left child of a node = Leftmost child of the node in the binary tree in the general tree

Rule 3: Right child of a node in the binary tree = Right sibling of the node in the general tree



Convert the given general tree into a binary tree

Now let us build the binary tree.

Step 1: Node A is the root of the general tree, so it will also be the root of the binary tree.

Step 2: Left child of node A is the leftmost child of node A in the general tree and right child of node A is the right sibling of the node A in the general tree. Since node A has no right sibling in the general tree, it has no right child in the binary tree.

Step 3: Now process node B. Left child of B is E and its right child is C (right sibling in general tree). Step 4: Now process node C. Left child of C is F (leftmost child) and its right child is D (right sibling in generaltree).

Step 5: Now process node D. Left child of D is I (leftmost child). There will be no right child of D because it has no right sibling in the general tree.

Step 6: Now process node I. There will be no left child of I in the binary tree because I has no left child in the general tree. However, I has a right sibling J, so it will be added as the right child of I.

Step 7: Now process node J. Left child of J is K (leftmost child). There will be no right child of J because it has no right sibling in the general tree.

Step 8: Now process all the unprocessed nodes (E, F, G, H, K) in the same fashion, so the resultant binary tree can be given as follows.