

Root:

One root of a binary is designated as the root of the binary tree. Normally it is the starting point of a binary tree and in diagrams we write it at the top.

Child of a node:

A node directly connected to another node when moving away from the root is called a child. The node on the left is called the left child and the node on the right is called the right node.

Parent node:

Given a node, its parent is the node for which the given node is a child.

Siblings:

Nodes with the same parent are called siblings.

Descendant

A node reachable by repeated proceeding from parent to child.

Ancestor

A node reachable by repeated proceeding from child to parent.

Leaf

A node with no children.

Internal node

A node with at least one child.

Path

The path from a node A to a node B is the sequence of nodes from A to B.

Path length

The number edges in the path from A to B is called the path length from A to B.

Depth of a node

The depth of a node is the path length of the node from the root.

Height of a node

The longest path of the node from a leaf.

Height of a binary tree

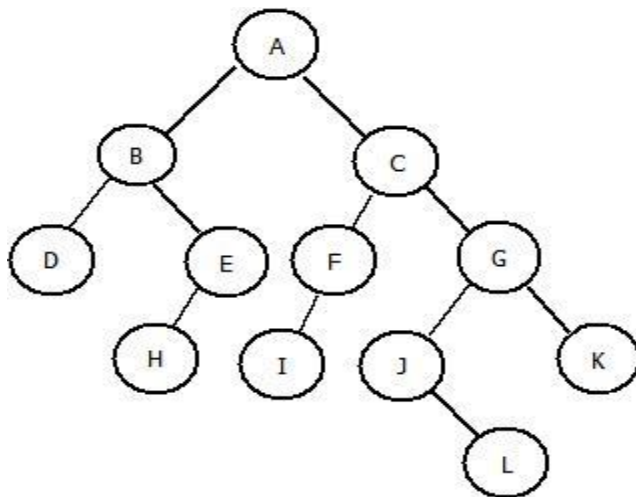
The height of the root is called the height of the binary tree.

Level of a node

Level of root is zero and the level of any other node is one more than the level of its parent.

Example:

Let us label nodes as shown:



Here are some examples for terms defined earlier:

A is the root.

B and C are children of A. B is the left child and C is the right child.

I is the left child of F.

D, H, I, L, and K are leaf nodes.

F and G are siblings.

J and K are siblings.

G is the parent of K.

A path from C to L is CGJL.

Path length of C to L is 3.
 Depth of E is 2.
 Height of C is 3.
 Height of B is 2.
 Height of the binary tree is 4.
 Level of A is 0.
 Level C is 1.
 Level of L is 4.

3. Expression tree

Given an infix expression, we represent it in the form of a binary tree, called an expression tree. An expression tree is an example of the use of a binary tree.

Manually creating an expression tree

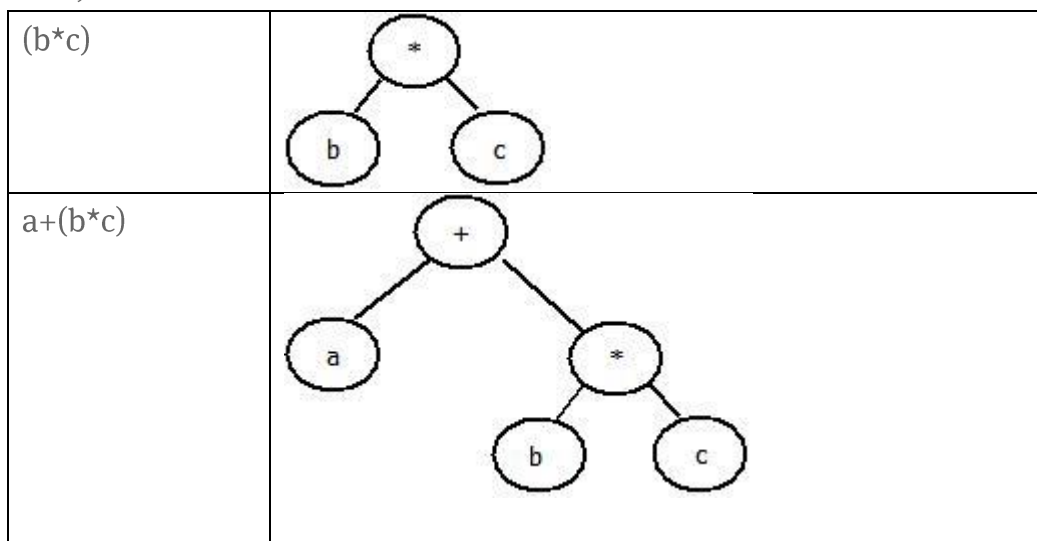
Consider an infix expression: $(a+b*c)*(c+d)$

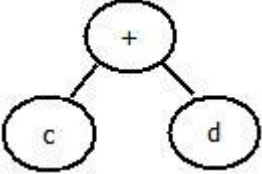
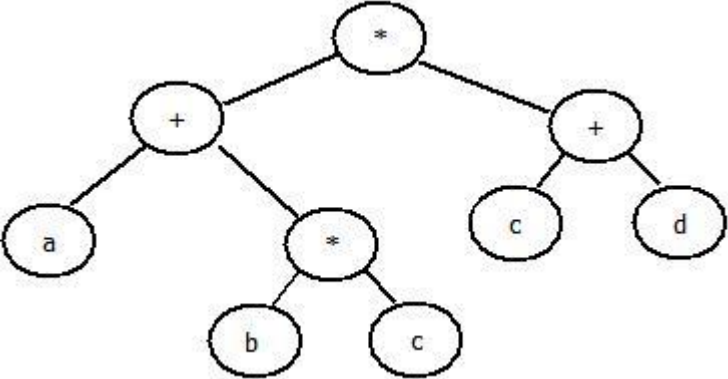
Completely parenthesis the expression: $((a+(b*c))*(c+d))$

Notice that each quantity in a pair of parentheses can be treated as an infix subexpression (meaning each pair of parentheses has two operands and one operator).

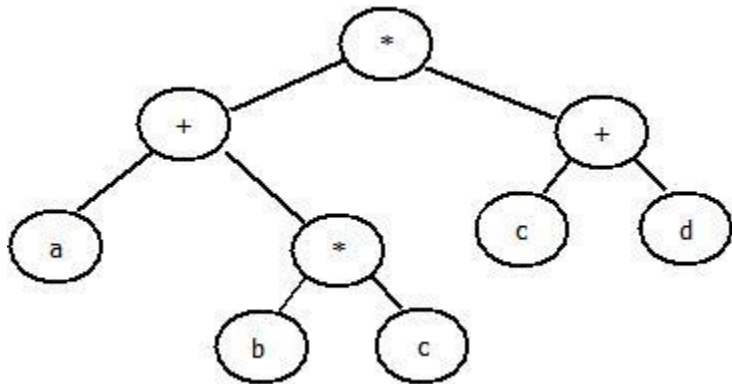
For each subexpression, create a part of binary tree with three nodes: root of this is the operator, left child is the left operand and right child is the right operand.

Thus, we have



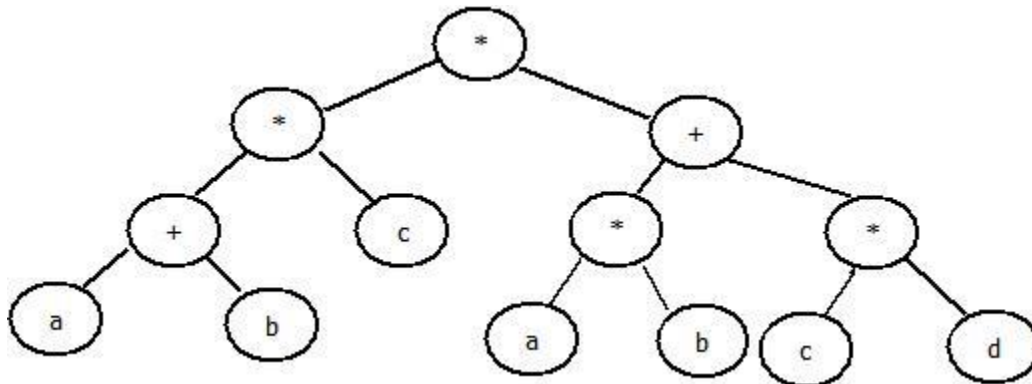
$(c+d)$	
$((a+(b*c))*(c+d))$	

The expression tree for $((a+(b*c))*(c+d))$ is,



Example:

Expression tree for $(a+b*c)*(a*b+c*d)$ is



4. Tree traversals

Given a binary tree, there are algorithms which require us to visit each node of the tree. We will discuss three ordered traversals of a given binary tree: **Preorder** traversal, **inorder** traversal and **postorder** traversal. We always start from the root. The definitions below are given using recursion. First try to understand the recursive definitions and then I will suggest an informal manual way of doing them.

Preorder traversal:

- Visit node
- Traverse left subtree in preorder
- Traverse right subtree in preorder

Inorder traversal:

- Visit node
- Traverse left subtree in inorder
- Traverse right subtree in inorder

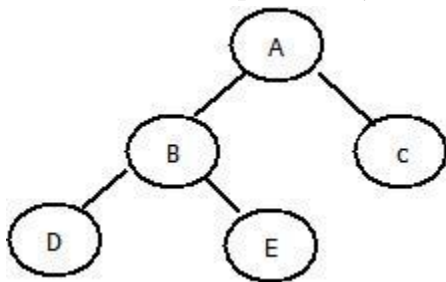
Postorder traversal:

- Visit node
- Traverse left subtree in inorder
- Traverse right subtree in inorder

Examples:

Because the definitions involve recursion, it is not easy to use a big tree for examples.

Consider the simple binary tree:



Preorder traversal: Let us follow recursive definition. For now, visit a node means output the visited node.

Recursive definition:

- Visit node
- Traverse left subtree in preorder
- Traverse right subtree in preorder

Let us start from the root (we always do), and then follow recursively, we get

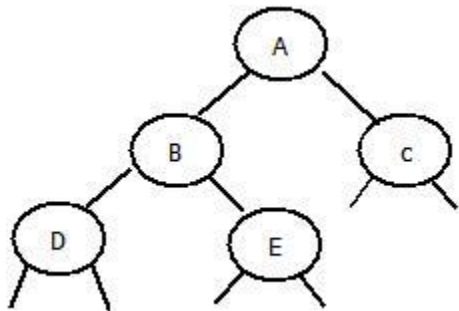
A
B
D
E
C

So preorder traversal results in A B D E C.

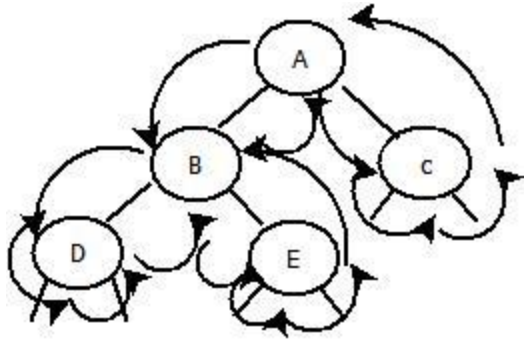
I know, if you are not very comfortable with recursion, this not trivial. Especially if the tree is big.

Here is a simple informal way of getting results for the three traversal methods.

First, draw dummy child-branches for all nodes which do not have both the children:



Now, draw a path in counter-clockwise along the tree starting from the root back to the root:



Notice your path drawn touches each node exactly three times.

Now, for preorder traversal, write nodes the first time you touch it.

For inorder traversal, write nodes the second time you touch it.

For postorder traversal, write nodes the last time you touch it.

Example: In the above drawing,

Preorder: (the first time you touch the node): A B D E C

Inorder: (the second time you touch the node): D B E A C

Postorder: (the last time you touch the node): D E B C A

5. Properties binary trees

➤ **The largest number of nodes on level i of a binary tree is 2^i .**

Recall the definition of level: Level of root is zero and the level of any other node is one more than the level of its parent. (Some books may define the level of root as 1. But we will keep this definition).

Proof:

The maximum number of nodes on any level $2 \times$ the maximum of number of nodes on the previous level.

Level # i	Maximum number of nodes	2^i
0	1	2^0
1	2	2^1
2	4	2^2
...		
i	$2 \times 2^{i-1}$	2^i

➤ **The largest number of nodes in a binary tree of height h is $2^{h+1} - 1$**

Recall the definition of height of a binary tree: The longest path (number of edges) of the root from a leaf.

If the height of a binary tree is given to be h , then the level of the leaf farthest from the root is h .

Level #	Maximum number of nodes on this level
0	1
1	2^1
2	2^2
3	2^3
.....	
h	2^h

The maximum number of nodes in the binary tree of height $h = 1 + 2^1 + 2^2 + 2^3 + \dots + 2^h$

This is a geometric series. Remember the formula for the sum,

$$= a_1 + (a_1r) + (a_1r^2) + (a_1r^3) + \dots + (a_1r^{n-1})$$

$$= a_1 \left(\frac{1-r^n}{1-r} \right)$$

Now a_1 is 1, n is $h+1$, and $r=2$.

(From Week 1 notes).

$$\text{So the sum of the series} = 1 \left(\frac{1-2^{h+1}}{1-2} \right)$$

$$= 2^{h+1} - 1$$

Examples:

The maximum number of nodes in a binary tree of height 3 is $2^4 - 1 = 15$.

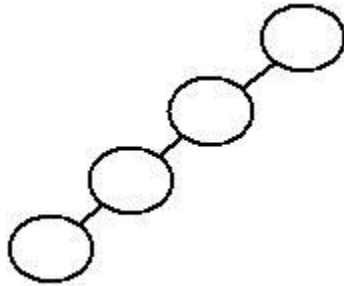
The maximum number of nodes in a binary tree of height 2 is 7.

The maximum number of nodes in a binary tree of height 5 is 31.

➤ **The minimum number of nodes in a binary tree height h is $h+1$.**

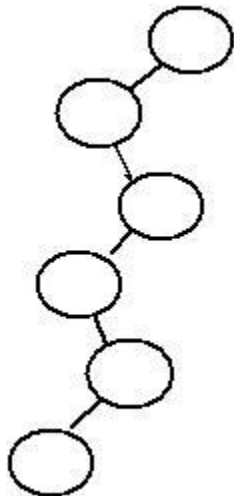
The number of nodes in a binary tree of height h will be minimum if each level has only one node. This means, there will be exactly $h+1$ nodes.

Example:



This is a binary tree of height 3 and the minimum number of nodes 4.

Example:



This a binary tree of height 5 and the minimum number of nodes is 6.

➤ **Minimum height of a binary tree with n nodes is**

(This is an important property.)

Let the height of a binary tree be h .

Maximum number of nodes the tree can have is $2^{h+1} - 1$. To have maximum number of nodes in a binary tree, we must have maximum number of nodes on each level of the tree.

The maximum number of nodes you can have in the tree up to height $h-1$ is $2^h - 1$.

Thus, the number of nodes n in a binary tree of height h must satisfy the following relationship:

$$2^h - 1 < n \leq 2^{h+1} - 1$$

Simplifying,

$$2^h < n+1 \leq 2^{h+1}$$

Taking log to the base 2,

$$h < \log_2(n+1) \leq h+1$$

That is, The value of height h (an integer) is such that $\log_2(n+1)$ is more than h but less than or equal to $h+1$.

If for example, the value of $\log_2(n+1)$ is 5.2, think of an integer h such that, $h < 5.2 \leq h+1$. Clearly $h = 5$, because $5 < 5.2 \leq 6$.

If $h < 12.86 \leq h+1$, then clearly $h = 12$, because $12 < 12.86 \leq 13$.

Now consider,

$h < 7 \leq h+1$. Now $h = 6$, because $6 < 7 \leq 7$.

To find the minimum height of a binary tree with n nodes, first calculate $\log_2(n+1)$. If this value is a whole number, the answer is $\log_2(n+1) - 1$, otherwise it is the floor of the number $\log_2(n+1)$, denoted by $\lfloor \log_2(n+1) \rfloor$.

Note:

Floor of a given number is the largest integer less than or equal to the given number. The notation for floor is $\lfloor \rfloor$. Java provides a method, `Math.floor(x)`, which returns the floor of x .

Examples:

$$\lfloor 5.6 \rfloor = 5$$

$$\lfloor 18.9 \rfloor = 18$$

$$\lfloor 5 \rfloor = 5$$

Examples:

Minimum height of a binary tree with 30 nodes = $\lfloor \log_2(30+1) \rfloor = \lfloor 4.954 \rfloor = 5$.

Minimum height of a binary tree with 100 nodes = $\lfloor \log_2(100+1) \rfloor = \lfloor 6.65821 \rfloor = 7$

Minimum height of a binary tree with 1000 nodes = $\lfloor \log_2(1000+1) \rfloor = \lfloor 9.9672 \rfloor = 10$

Minimum height of a binary tree with 63 nodes = $\log_2(63+1) = 6$. $h = 5$.

Note: I used log to base 2 calculator from here: <http://www.miniwebtool.com/log-base-2-calculator/>

➤ **A binary tree with n nodes has $n+1$ empty branches**

(Empty branch means no child branch.)

Proof:

Given: binary tree has n nodes.

Let E be number of empty branches in the binary tree.

We need to prove, $E = n + 1$.

Let B be total number of non-empty branches.

In a binary tree, every node has two branches (empty or non-empty). So the total number of branches (empty and non-empty) in the tree = $2n$.

So we have $B + E = 2n$ (1)

We can easily see that, $n = B + 1$... (2)

Substituting (2) in (1), we get,

$$n - 1 + E = 2n$$

Thus, $E = n + 1$.

Note: If you don't see (2) $n = B + 1$ easily, do this.

Observe that every node in a binary tree has a branch going from the node to its parent, except the root. Treat each of these objects like a lollipop with a stick. So the binary tree consists of a bunch of lollipops and one extra node (the root). Thus the number of nodes n and the number of branches (non-empty) are related as follows: $n = B + 1$.

➤ **If a binary tree has M nodes with exactly two children and has L leaves, then**

$$L = M + 1.$$

Proof:

M – Number of nodes with exactly two branches

L – Number of leaves

Let,

n – Total number of nodes in the tree

O – Number of nodes with one branch

We have,

Summing up all three kinds of nodes – two branches, one branch and no branches, we get

$$n = M + O + L \quad (1)$$

From the previous property, we know the number of empty branches = $n + 1$.

$$n + 1 = 2L + O \quad (2)$$

Substitute for n from (2) in (1):

$$2L + O - 1 = M + O + L$$

Simplifying,

$$L = M + 1.$$

6. Implementation of a binary tree

As you now, given a set of data, you can draw several kinds of binary trees. So there is no unique binary tree for a given set of values, unless certain conditions are placed.

As an example, let us create an expression tree.

We will now discuss a Java implementation of a binary tree representation a given postfix expression. There is no easy way of creating an expression tree directly for a given infix expression. Of course, we can always convert an infix expression into postfix then use the following method to create an expression tree for the postfix expression.

Constructing a binary expression tree in Java from a given postfix expression:

Before we start discussion on Java representation of expression trees, a brief review of objects in Java. Please remember, an object variable in Java is actually pointer to the object.

As an example, consider a class, sample with two fields, int x , and int y :

```

Class sample{
    int x;
    int y;
    sample(int a, int b){
        x = a;
        y = b;
    }
}

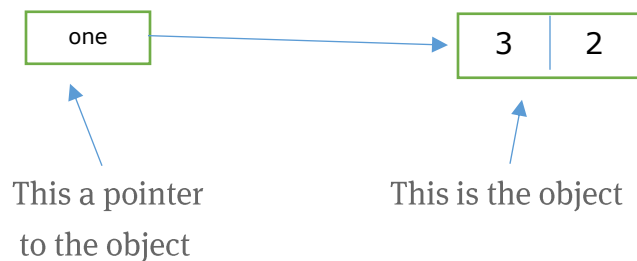
```

Consider creating an object one of sample as follows:

```
sample one = new sample(3,2);
```

Now, we say, the variable sample is an object of class sample.

In Java, object names are actually pointer, pointing to the object. In a diagram, it looks like this:



Now, let us start discussion on implementation of expression trees in Java. We implement a binary tree node in Java as a class, with three fields, as shown below:

```

class TreeNode {
    TreeNode left;
    char element;
    TreeNode right;

    TreeNode(TreeNode leftBranch, char token, TreeNode rightBranch) {
        left = leftBranch;
    }
}

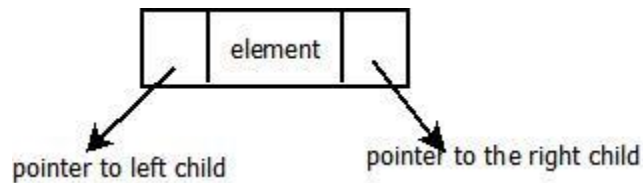
```

```

    element = token;
    right = rightBranch;
}
}

```

In a diagram a node looks like this:



If a child does not exist, we will denote it by **null**.

The algorithm to construct a binary expression tree from a given postfix expression is very simple.

Given: a postfix expression

We will use one stack of Objects.

Algoritihm:

Without any loss of generality, let us assume: Operands are one characters: a – z, and operators are +, -, *, and /.

Scan the input postfix expression left to right, one character at a time. We call the character being scanned a token,

- If the token is an operand, create a one node binary tree, with the token and with child pointers null. Push the tree on to the stack.

Some explanation on this step:

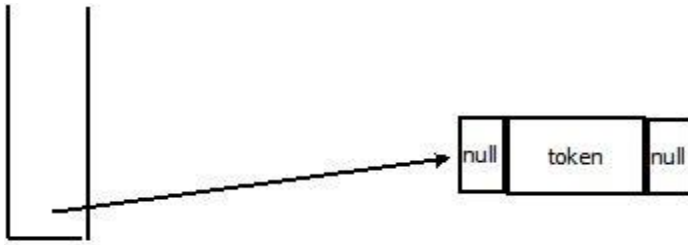
Creating a one node binary tree means the following:



This can be visualized as



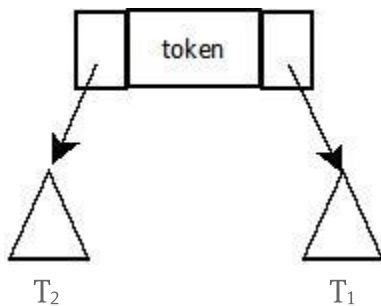
Push the tree on to the stack means, push the pointer to the tree on to stack:



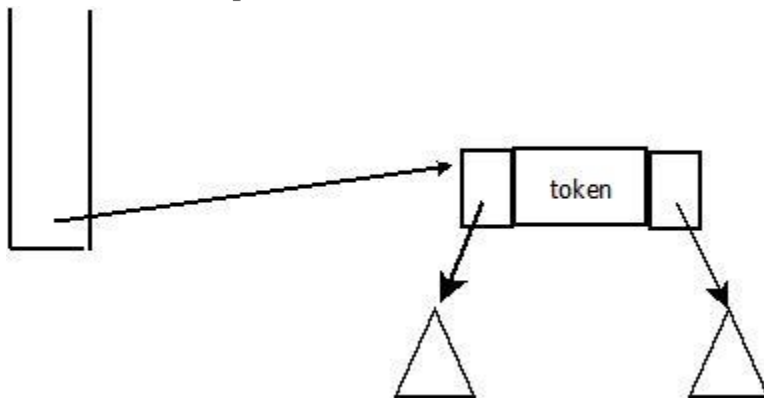
- If the token is operator, pop the stack twice. Denote the first popped element T_1 and the second T_2 . Create a new tree such that the root is the token (an operator), the left subtree is T_2 and the right subtree is T_1 . Push the pointer to this new tree on to the stack.

Some explanation on this step:

The new tree looks like this:



After this tree is pushed,



When the end of the input expression is reached (when there is no more character to scan), the final result in the stack. The value in the stack points to the expression tree.

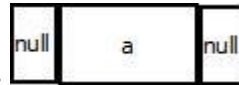
A complete example:

Before we write Java code to create an expression tree, let us manually do one example.

Given postfix expression: $ab+cd+*$

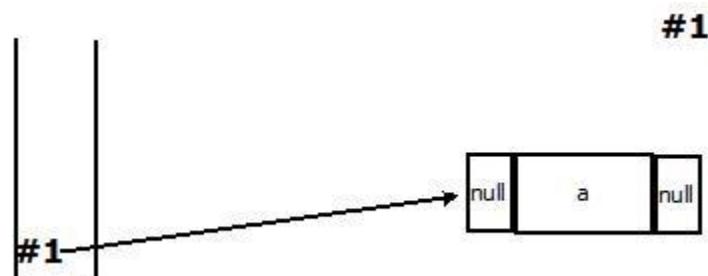
We scan left to right,

Token = 'a',



We create a tree with the token b:

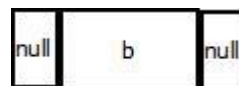
Push it on to the stack.



I have labeled the tree #1, just to indicate the tree is #1 and on stack #1 indicates the pointer to #1 tree (this is just to help up keep track of things).

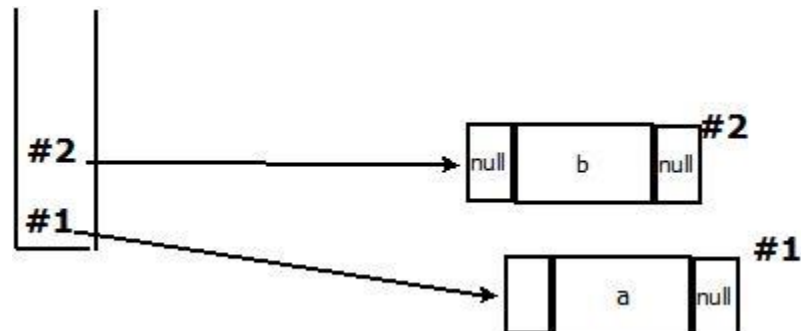
Continue scanning.

Token = b



We create a tree with the token b:

Push it on to the stack.



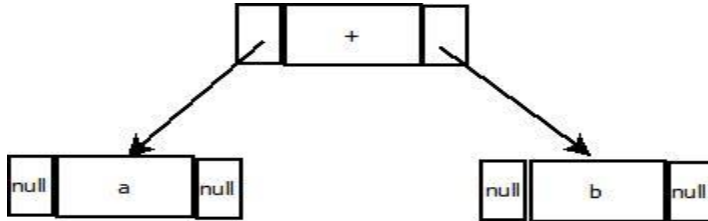
Continue scanning.

Token = +

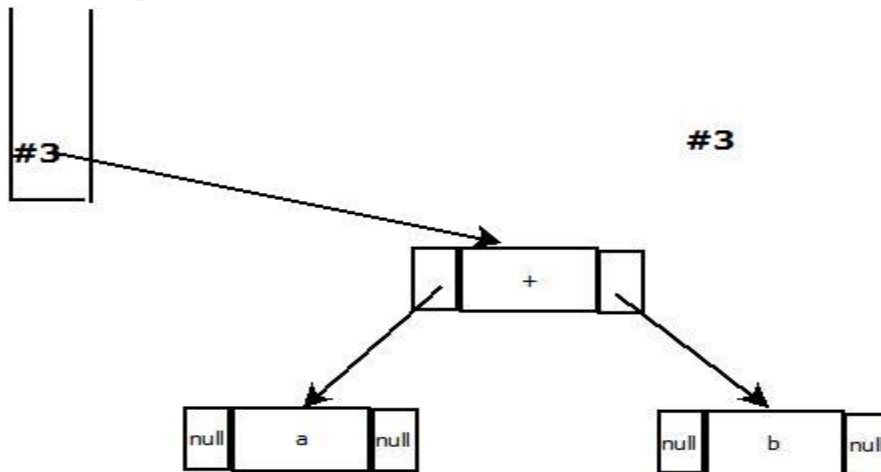
Pop the stack twice. Denote the first popped element T_1 and the second T_2 .

Now, T_1 = tree #2 and T_2 = tree #1

Create a new tree such that the root is the token (the operator +), the left subtree is T_2 and the right subtree is T_1 .



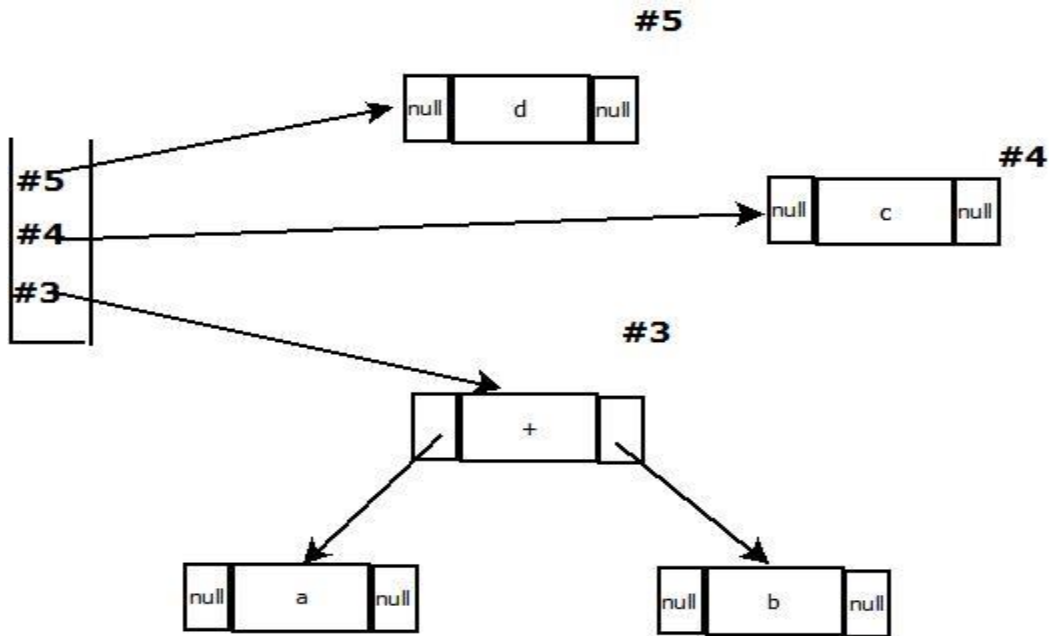
Push the pointer to this new tree on to the stack.



Remember we have popped #1 and #2.

Continue scanning....

After scanning and processing the next two tokens, c and d, we get the following picture:



Continue scanning.

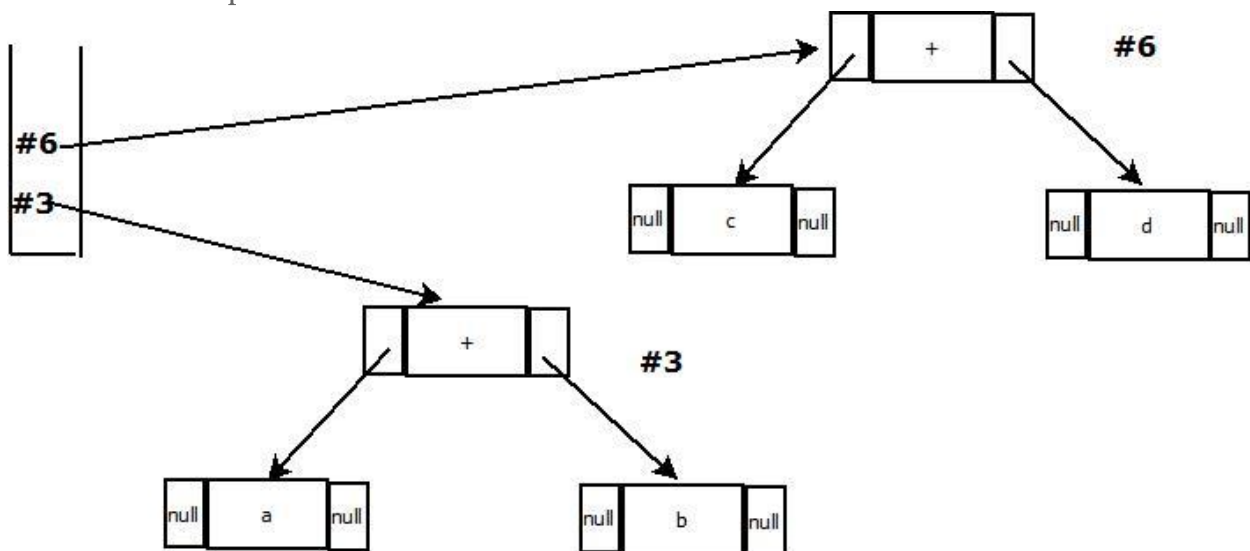
Token = +

Pop the stack twice. Denote the first popped element T_1 and the second T_2 .

Now, T_1 = tree #5 and T_2 = tree #4

Create a new tree such that the root is the token (the operator +), the left subtree is #4 and the right subtree is #5. Push the pointer to this new tree on to the stack.

Here is the new picture:



Continue scanning.

Continue scanning.

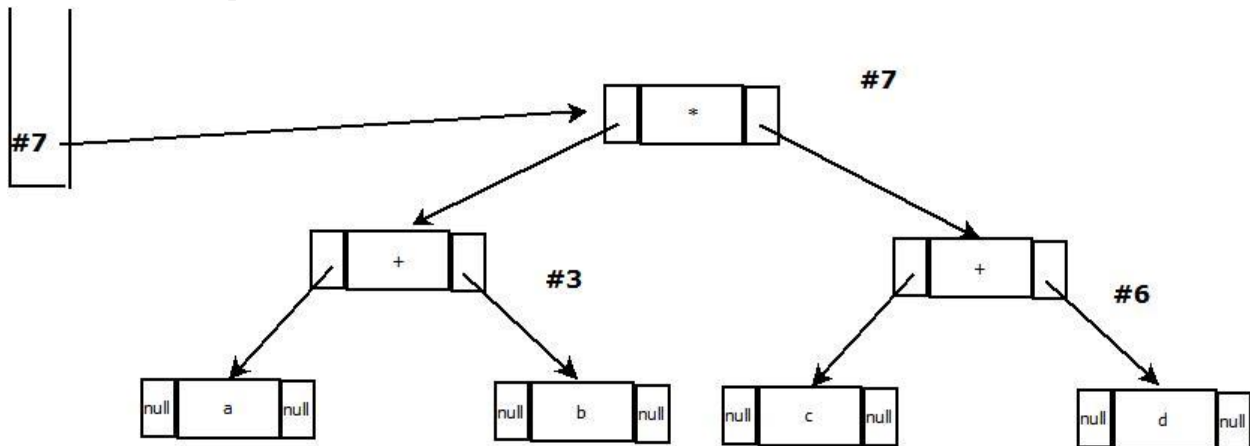
Token = *

Pop the stack twice. Denote the first popped element T_1 and the second T_2 .

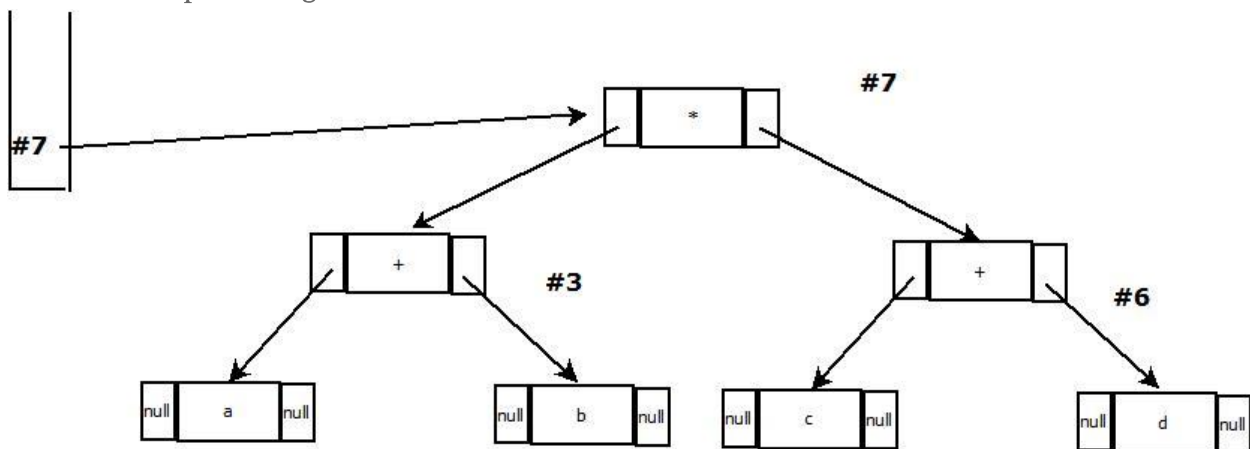
Now, T_1 = tree #6 and T_2 = tree #3

Create a new tree such that the root is the token (the operator *), the left subtree is #3 and the right subtree is #6. Push the pointer to this new tree on to the stack.

Here is the new picture:



End of the input string is reached. The result is:



This is the binary tree representation of the postfix expression: $ab+cd+^*$

Now a Java program to implement this algorithm to create a binary expression tree for a given postfix expression.

7. A Java program to create a binary expression tree for a given postfix expression.

Once we write a program to construct the binary tree, there is no easy way of verifying the program has done a correct job. No easy methods to display the tree in a visibly nice format. So I have included a preorder() method to display the preorder traversal of the tree. This just helps verify that the tree is constructed correctly.

Now Java code:

```
import java.util.*;
public class expressionTrees {
    static String inputPostfix;
    static TreeNode root;
    /**
     * Constructs an binary expression tree, using the given Postfix expression
     * This is the most important part ... study carefully
     */

    static public void createExpressionTree() {
        Stack nodes = new Stack(); //This is Java built-in Stack. Stack of Objects
        for (int i = 0; i < inputPostfix.length(); i++) {
            char token = inputPostfix.charAt(i);
            if(token == ' ') continue; // Spaces in input expression are ignored
            if( token >='a' && token <='z'){
                TreeNode newNode = new TreeNode(null,token,null);
                nodes.push(newNode);
            }
            else{
                TreeNode rightNode = (TreeNode)nodes.pop();
                TreeNode leftNode = (TreeNode)nodes.pop();
                TreeNode newNode = new TreeNode(leftNode,token,rightNode);
                nodes.push(newNode);
            }
        }
    }
}
```

```

    root = (TreeNode)nodes.pop();
}

static void preOrder(TreeNode node) {
    if (node != null) {
        System.out.print(node.element+" ");
        preOrder(node.left);
        preOrder(node.right);
    }
}

public static void main(String[] args) {
    inputPostfix = "a b + c + d +";
    createExpressionTree();
    preOrder(root);
}
}

//This is the TreeNode class. Just three fields and a constructor
class TreeNode {
    TreeNode left;
    char element;
    TreeNode right;
    //Constructor
    TreeNode(TreeNode leftBranch, char token, TreeNode rightBranch) {
        left = leftBranch;
        element = token;
        right = rightBranch;
    }
}

```

8. Computing the height of a binary tree

Given a binary tree, we will write a Java method to compute the height of the tree. First, what is the meaning of “given a binary tree”, in general it means a pointer to the root node is given. You have to start from there.

Recall, the height of a binary tree is the longest path of the root from a leaf. In other words, identify the leaf which is farthest from the root and then count the number of edges between that leaf and the root. The number of edges between that leaf and the root is the height of the binary tree. There are other slightly different definitions, let us keep this definition.

Given the pointer to the root, the following method finds the height of the tree.

```
static int height(TreeNode node){
    if(node == null) return -1;
    else{
        int tempLeft = height(node.left);
        int tempRight = height(node.right);
        if (tempLeft < tempRight) return tempRight+1;
        else return tempLeft+1;
    }
}
```

When you call this method from your main(), pass the pointer to the root of the tree. This method uses recursion: Find the height of the left subtree (recursively), find the height of the right subtree (recursively), find the larger of these two and add one to get the height of the binary tree.

I hope you like recursion. Otherwise, you have start linking it. Programming/problem solving life is beautiful with recursion.

9. Displaying the elements of a binary tree on a given level

Given a binary tree and a level (an integer), we will display (print) the elements of the binary tree on that level.

Recall the definition of the level: Level of root is zero and the level of any other node is one more than the level of its parent. Some books may start the level of root as 1. But we will keep our definition.

The following recursive method inputs the root of a binary tree and a level, and prints the elements on that level.

```
static void printElementsOnALevel (TreeNode node ,int level)
{
    if (node == null) return;
    if (level == 0) System.out.print(node.element + " ");
    else if (level > 0)
    {
        printElementsOnALevel(node.left, level-1);
        printElementsOnALevel(node.right, level-1);
    }
}
```

Note:

Here is an interesting way of using this method.
Consider the following code,

```
for (int i=0; i<=height(root); i++) {
    System.out.println();
    printElementsOnALevel(root,i);
}
```

This code displays the elements on each level (starting from level zero through the height of the binary tree). Thus displaying the whole binary tree without edges. Comes in handy if you want to visualize a binary tree.

10. A complete example

I have written two Java classes: `BTNode.java` and `BTMethods.java`.
The **BTNode** is a simple class, which implements a binary node class:


```

class BTreeNode {
    BTreeNode left;
    char element;
    BTreeNode right;
    //Constructor
    BTreeNode(BTreeNode leftBranch, char token, BTreeNode rightBranch) {
        left = leftBranch;
        element = token;
        right = rightBranch;
    }
}

```

The node holds a character.

The **BTMethods.java** class is just a class (without main()), which contains the following:

static String inputString static BTreeNode root;	Two fields. Field inputString holds the string which will be converted into a binary tree.
BTMethods(){}	One constructor
static public void createBinaryTree()	Method This method creates a binary tree. Data elements for the binary tree are characters in the variable inputString.
static void preOrder(BTreeNode node)	Method Prints the elements of the binary tree in preorder.
static int height(BTreeNode node)	Method Returns the height of the tree
static void displayTree(BTreeNode node)	Method Prints elements in the order of level.

You can download these two classes, BTNode.class and BTMethods.class using the following links.

Note that they are Java source code files, they are class files.

<https://drive.google.com/file/d/0BwSKJqThXLLzc2d5Q1hVVjZuOW8/view?usp=sharing>

<https://drive.google.com/file/d/0BwSKJqThXLLzbTFzVDZwbTdkU2s/view?usp=sharing>

We will write a simple main program to use these two classes.

Download the two class files and keep them in the same folder as the main programs you will write.

Example:

The following simple main program uses the two classes:

```
public class BTMain {  
    public static void main(String[] args) {  
        BTMethods one = new BTMethods();  
        one.createBinaryTree();  
        System.out.println("\nBinary tree traversed in preorder:");  
        one.preOrder(one.root);  
        System.out.println("\nHeight of the tree = "+one.height(one.root));  
        System.out.println("Binary tree displayed in the level order:");  
        one.displayTree(one.root);  
    }  
}
```

Note: Save this Java program in the same folder as the two classes you downloaded using the links above.

Output:

Binary tree traversed in preorder:

A B P D S F I V J W X U C Q E R H G Y K A P Q P T

Height of the tree = 7

Binary tree displayed in the level order:

A
B C
P D Q E
S F R H
I U G T
V J Y K
W X A P
P Q

Let us add a method, **numberOfNodes()**, to our main class. This method returns the number of nodes in the tree:

```
public class BTMain {  
    public static void main(String[] args) {  
        BTMethods one = new BTMethods();  
        one.createBinaryTree();  
        System.out.println("\nBinary tree traversed in preorder:");  
        one.preOrder(one.root);  
        System.out.println("\nHeight of the tree = "+one.height(one.root));  
        System.out.println("Binary tree displayed in the level order:");  
        one.displayTree(one.root);  
        System.out.println("\n Number of nodes in the binary tree:  
                                "+numberOfNodes(one.root));  
    }  
    static int numberOfNodes(BTNode root){  
        if(root == null)  
            return 0;  
        else  
            return (numberOfNodes(root.right)+ numberOfNodes(root.left)+1);  
    }  
}
```

Output:

Binary tree traversed in preorder:

A B P D S F I V J W X U C Q E R H G Y K A P Q P T

Height of the tree = 7

Binary tree displayed in the level order:

A

B C

P D Q E

S F R H

I U G T

V J Y K

W X A P

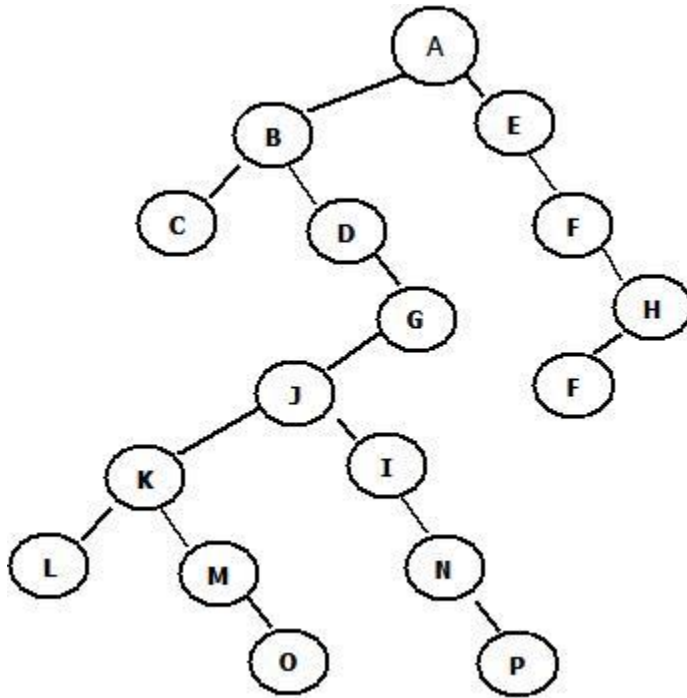
P Q

Number of nodes in the binary tree: 25

11. Exercises:

Exercises with Answers

1. Consider the following binary tree:



- (a) What is the height of the tree?
- (b) Write the elements on level 2
- (c) What is the path length from G to N?
- (d) How many leaves are in the tree?
- (e) What is the level of the node with element J?
- (f) Write the preorder traversal of the tree
- (g) Write the inorder traversal of the tree
- (h) Write the postorder traversal of the tree
- (i) How many empty branches are in the tree
- (j) How many nodes have exactly two children
- (k) Verify that this statement is true: A binary tree with n nodes has $n+1$ empty branches
- (l) Verify this statement is true: If a binary tree has M nodes with exactly two children and has L leaves, then $L = M + 1$.

2. (a) Given an example of a binary tree with four nodes where inorder and postorder travels give the same result. If such a tree does not exist, just say, "DOES NOT EXIST".

(b) Given an example of a binary tree with four nodes where inorder and preorder travels give the same result. If such a tree does not exist, just say, "DOES NOT EXIST".

(c) Given an example of a binary tree with four nodes where preorder and postorder travels give the same result. If such a tree does not exist, just say, "DOES NOT EXIST".
3. (a) What is the largest number of nodes you can have on level 3 in a binary tree?

(b) What is the largest number of nodes you can have on level 10 in a binary tree?

(c) What is the largest number of nodes you can have on level 0 in a binary tree?
4. (a) What is the largest number of nodes you can have in a binary tree of height 3?

(b) What is the largest number of nodes you can have in a binary tree of height 9?

(c) What is the largest number of nodes you can have in a binary tree of height 12?
5. (a) What is the minimum height of a binary tree you can have with 100 nodes?

(b) What is the minimum height of a binary tree you can have with 1024 nodes?

(c) What is the minimum height of a binary tree you can have with 64 nodes?
6. (a) How many empty branches does a binary tree with 100 nodes have?

(b) How many empty branches does a binary tree with 1024 nodes have?

(c) How many empty branches does a binary tree with 64 nodes have?
7. (a) A binary tree has 17 nodes with exactly two branches. How many leaves are in the tree?

(b) A binary tree has 100 nodes with exactly two branches. How many leaves are in the tree?

- (c) A binary tree has 64 nodes with exactly two branches. How many leaves are in the tree?
8. (a) True or False: The nodes with two branches in a binary tree appear in the same relative order in preorder, inorder, and postorder traversals.
- (b) True or False: The leaves in a binary tree appear in the same relative order in preorder, inorder, and postorder traversals.
9. A binary tree is such that,
- (1) no node has one child and (2) all leaf nodes are on the last level.
- If K is the number nodes on level i , how many nodes are in the tree before level i (answer in terms of K).
-