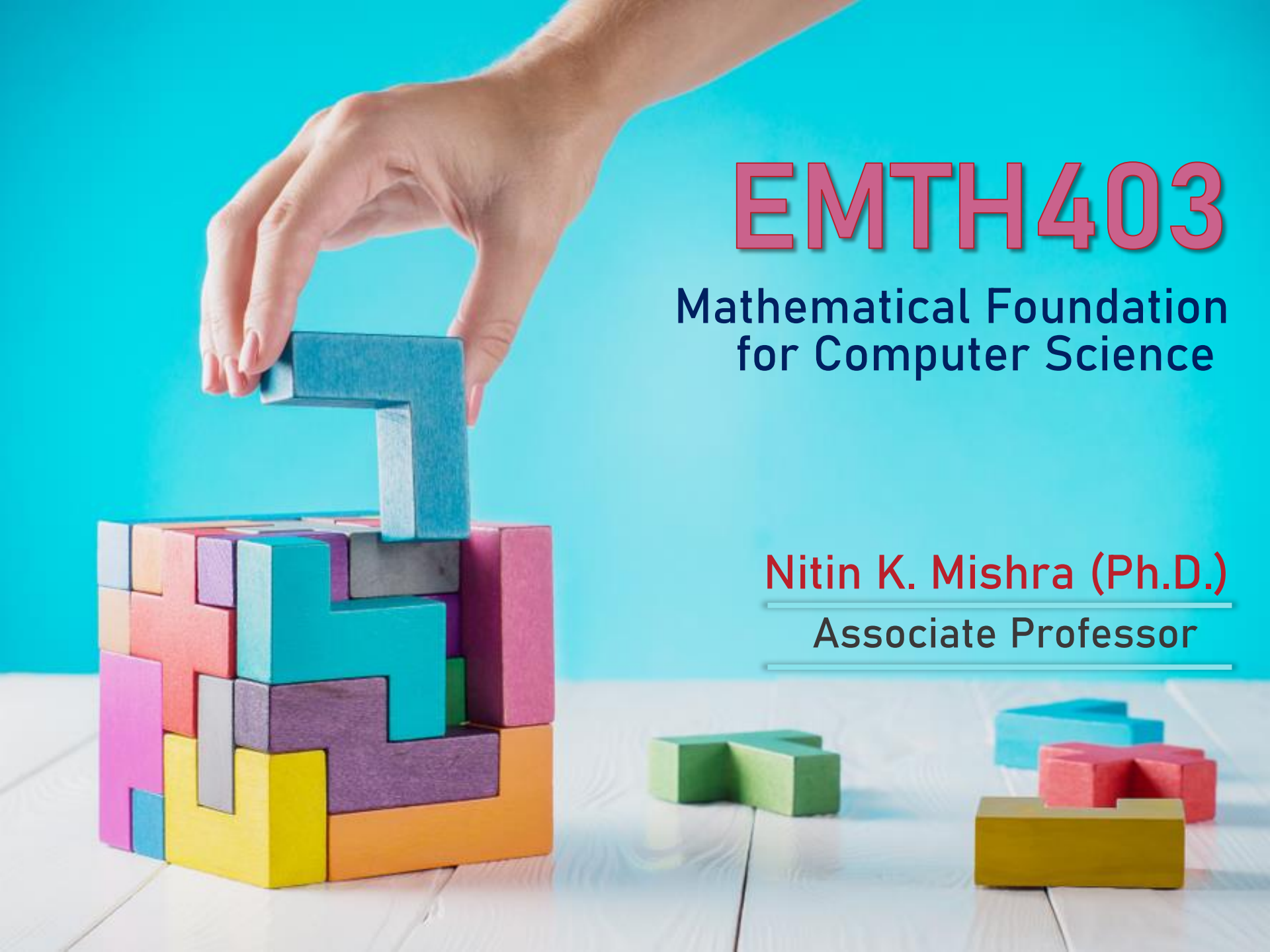


EMTH403

Mathematical Foundation for Computer Science

Nitin K. Mishra (Ph.D.)

Associate Professor

Lecture Outcomes

After this lecture, you will be able to

- Understand what Spanning Trees
- Understand what is Kruskal's Algorithm.

Tree traversals

A binary tree is defined recursively: it consists of a root, a left subtree, and a right subtree

To traverse (or walk) the binary tree is to visit each node in the binary tree exactly once

Tree traversals are naturally recursive

Since a binary tree has three "parts," there are six possible ways to traverse the binary tree:

Tree traversals

Since a binary tree has three “parts,” there are six possible ways to traverse the binary tree:

root, left, right

left, root, right

left, right, root

root, right, left

right, root, left

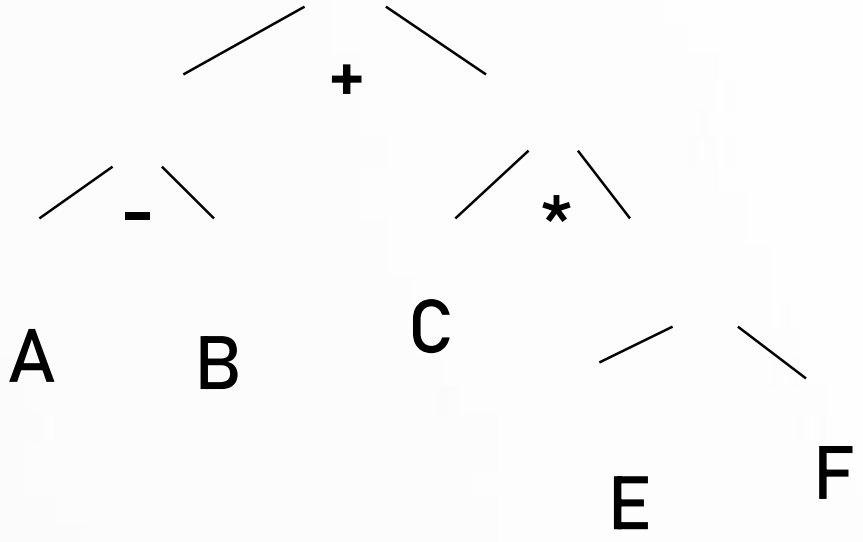
right, left, root

More binary trees examples

1. Binary tree for representing arithmetic expressions. The underlying hierarchical relationship is that of an arithmetic operator and its two operands.

More binary trees examples

2. Arithmetic expression in an infix form: (A - B) + C * (E / F)

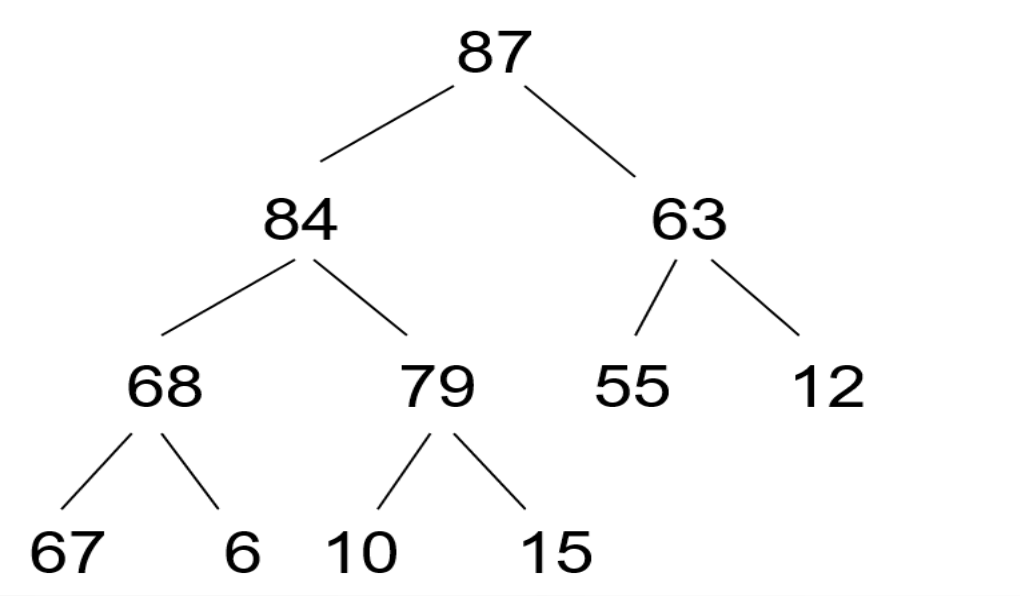


More binary trees examples

Note that a post-order traversal of this tree (i.e. visiting the left subtree first, right subtree next, and finally the root) returns the postfix form of the arithmetic expression, while the pre-order traversal (root is visited first, then the left subtree, then the right subtree) returns the prefix form of the arithmetic expression.

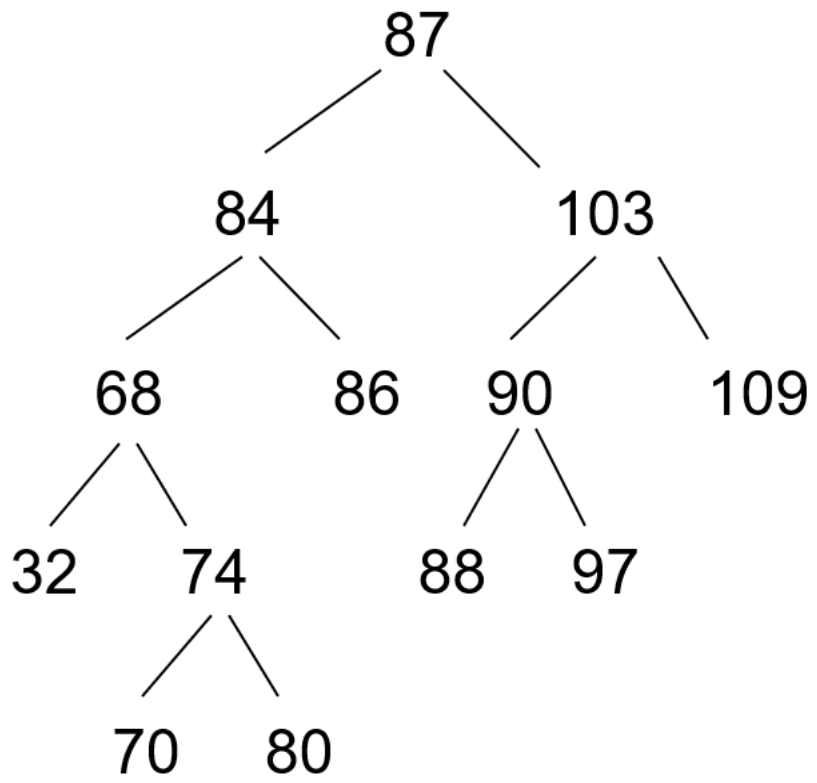
More binary trees examples

3. Binary tree with a heap property. The underlying hierarchical relationship suggests that the datum in each node is greater than or equal to the data in its left and right subtrees.



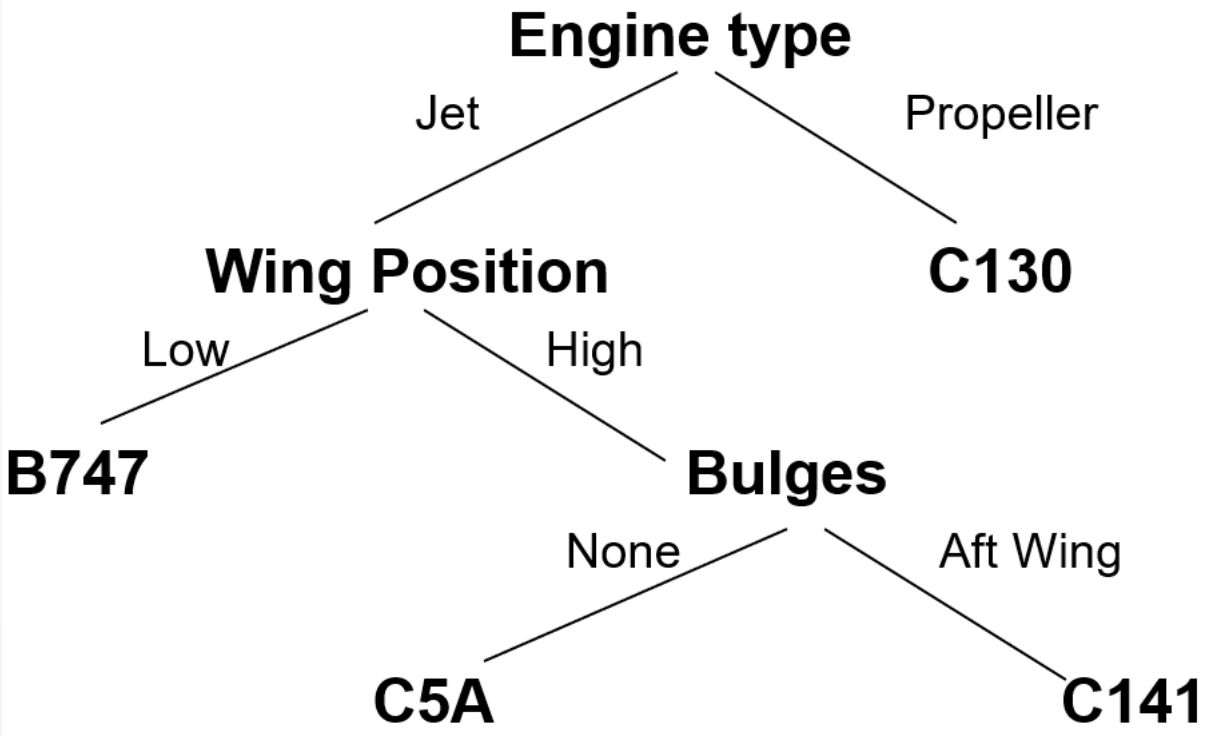
More binary trees examples

4. Binary tree with an ordering property. The underlying hierarchical relationship suggests that the datum in each node is greater than the data in its left subtree, and less than or equal to the data in its right subtrees.



More binary trees examples

Decision trees.



Preoder, Inorder, Postorder

In Preorder, the root

is visited before (pre)

the subtrees traversals

In Inorder, the root is

visited in-between left

and right subtree traversal

In Preorder, the root

is visited after (pre)

the subtrees traversals

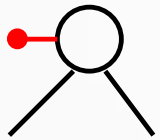
Preorder Traversal:
1. Visit the root
2. Traverse left subtree
3. Traverse right subtree

Inorder Traversal:
1. Traverse left subtree
2. Visit the root
3. Traverse right subtree

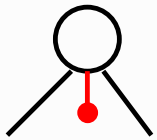
Postorder Traversal:
1. Traverse left subtree
2. Traverse right subtree
3. Visit the root

Tree traversals using “flags”

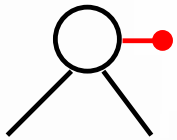
The order in which the nodes are visited during a tree traversal can be easily determined by imagining there is a “flag” attached to each node, as follows:



preorder

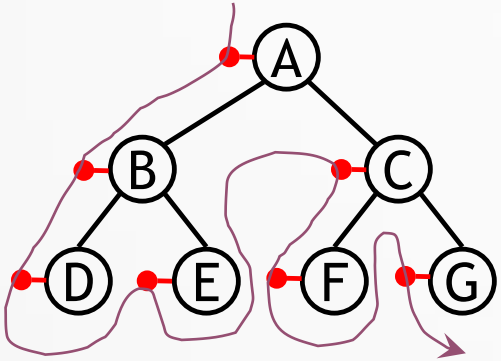


inorder

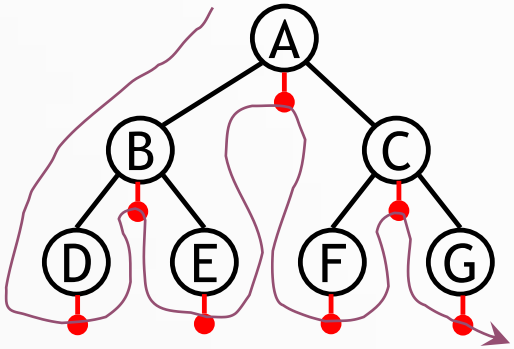


postorder

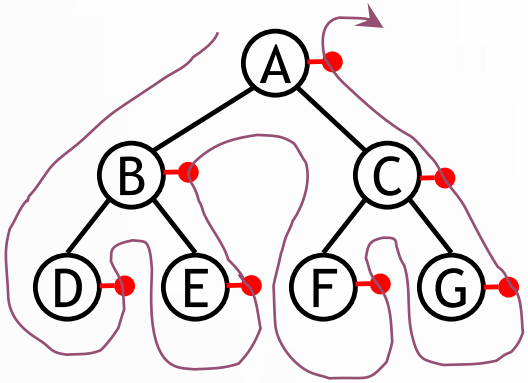
- To traverse the tree, collect the flags:



A B D E C F G



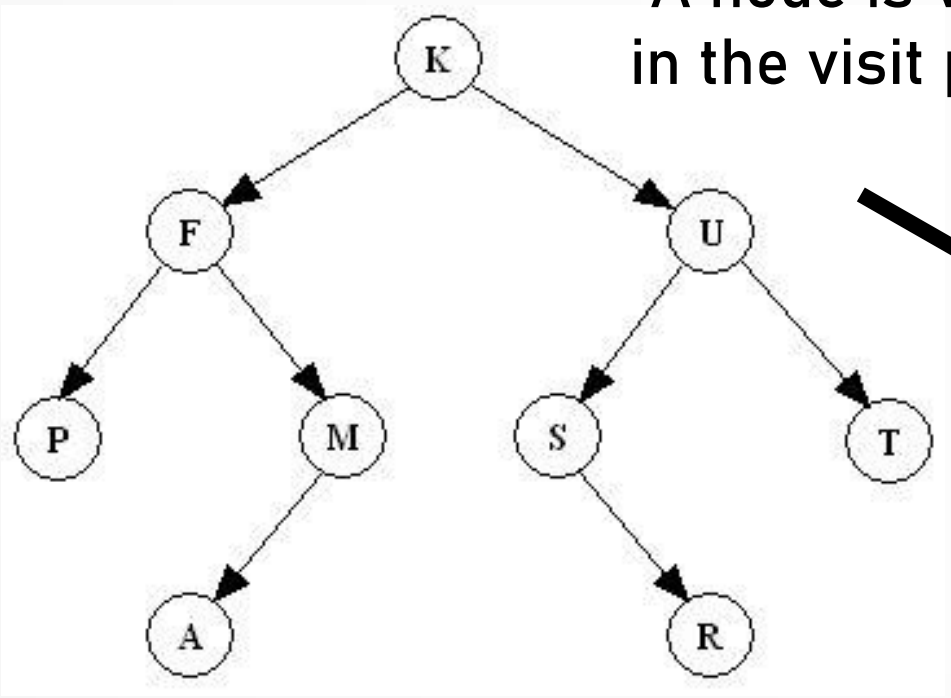
D B E A F C G

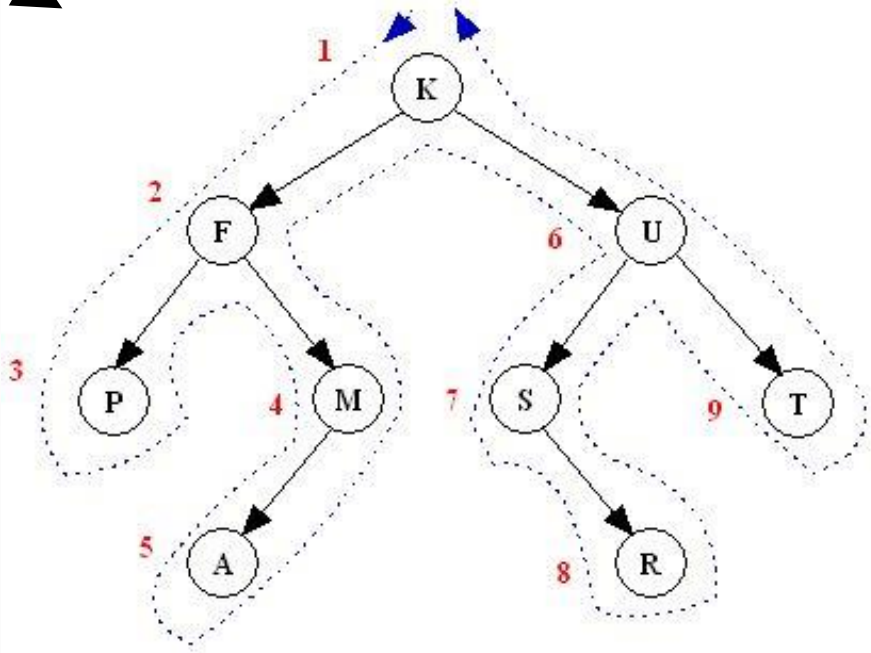


D E B F G C A

Preorder Depth-first Traversal - N-L-R

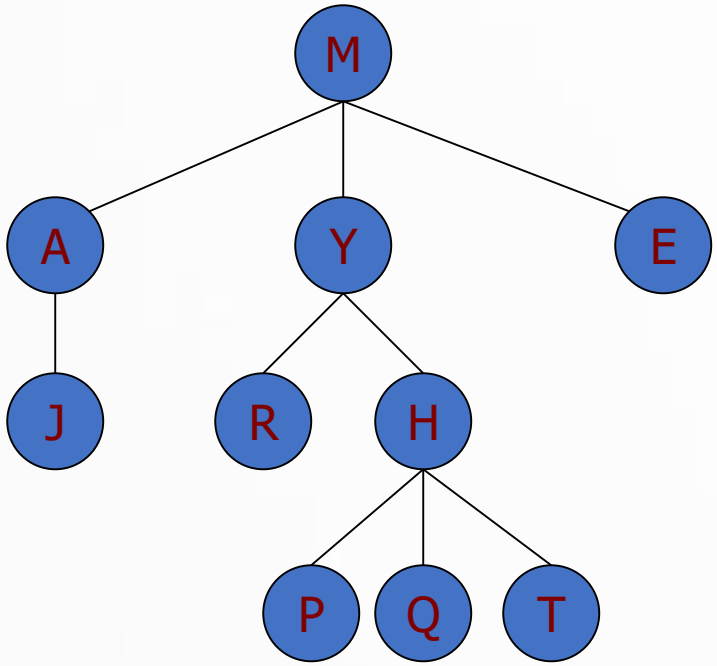
"A node is visited when passing on its left in the visit path"





K F P M A U S R T

Preorder Depth-first Traversal - N-L-R

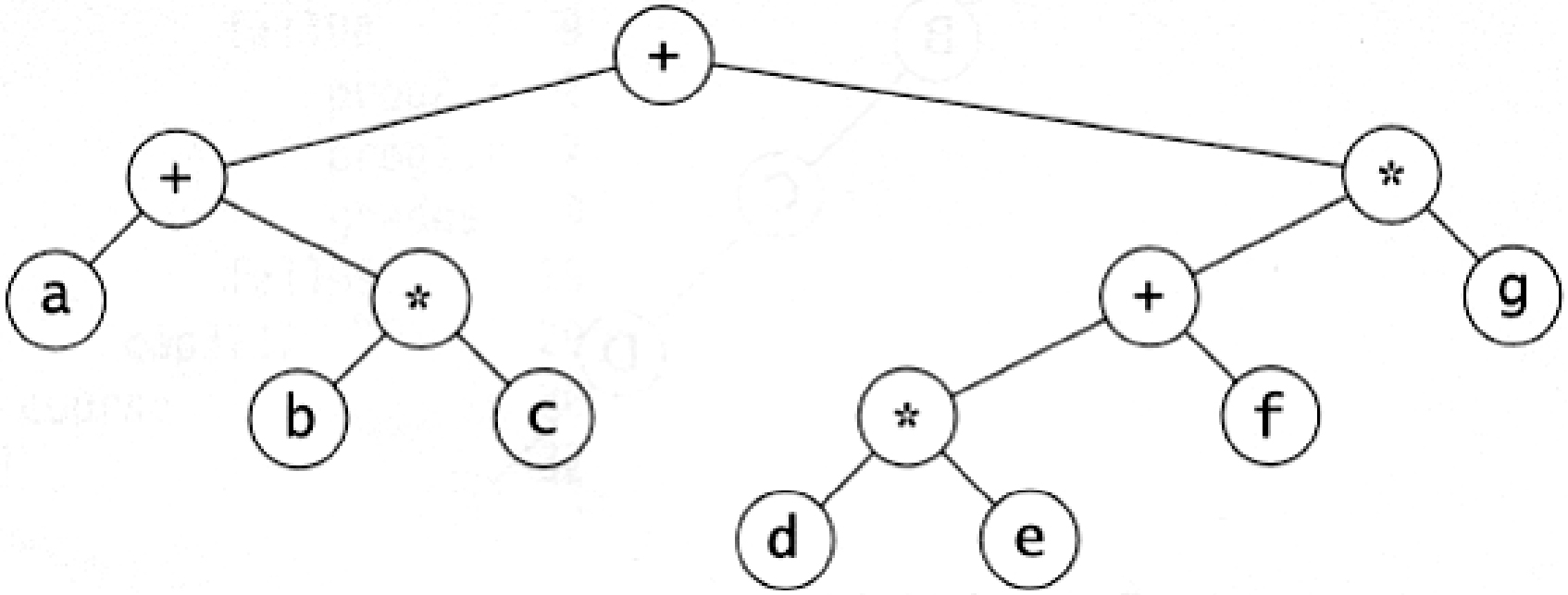


M A J Y R H P Q T E

Preorder Depth-first Traversal - N-L-R

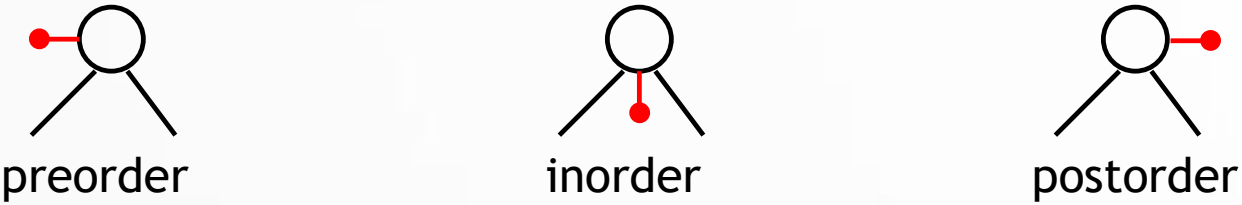
Preorder traversal node, left, right prefix expression?

Expression tree for ++a*bc*+*defg

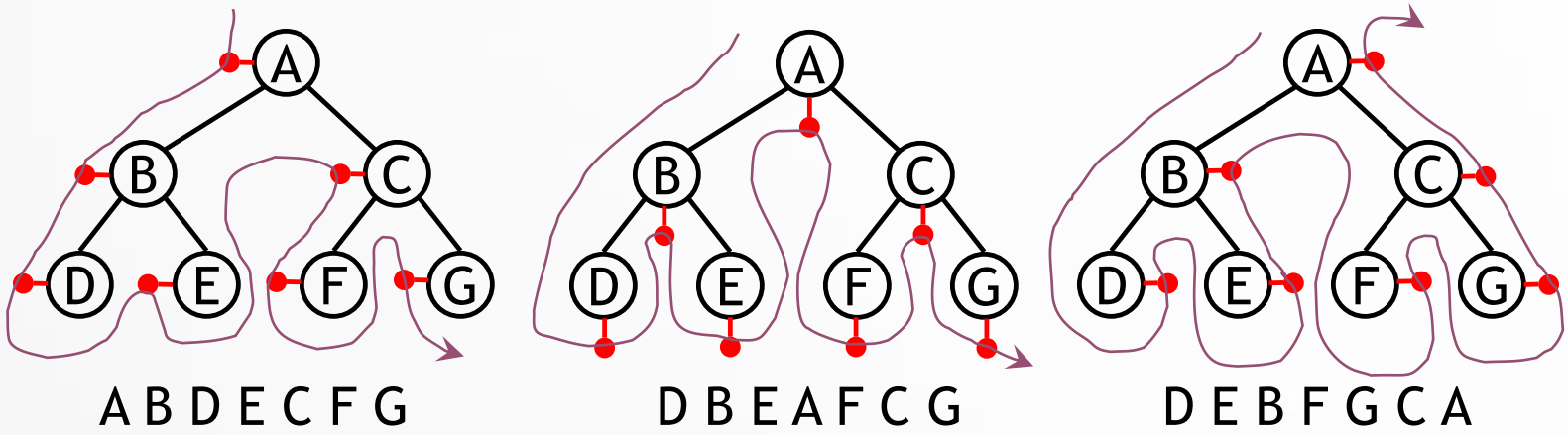


Tree traversals using “flags”

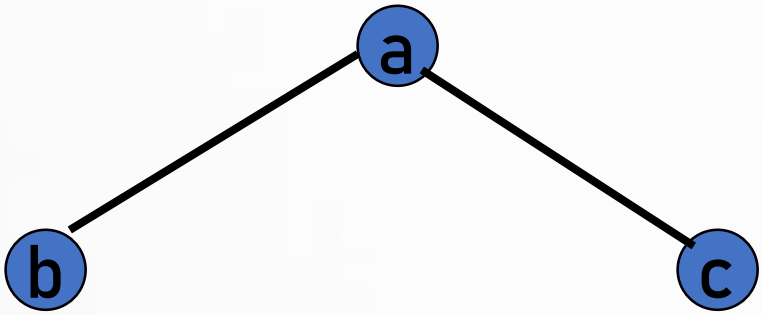
The order in which the nodes are visited during a tree traversal can be easily determined by imagining there is a “flag” attached to each node, as follows:



- To traverse the tree, collect the flags:

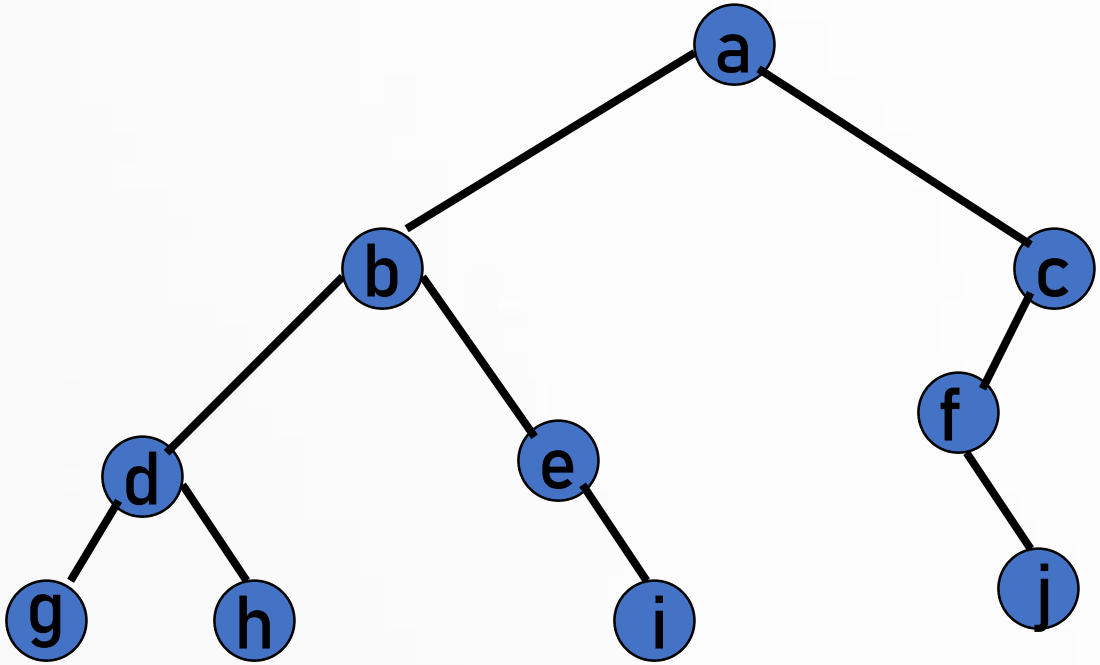


In-order Traversal: LNR



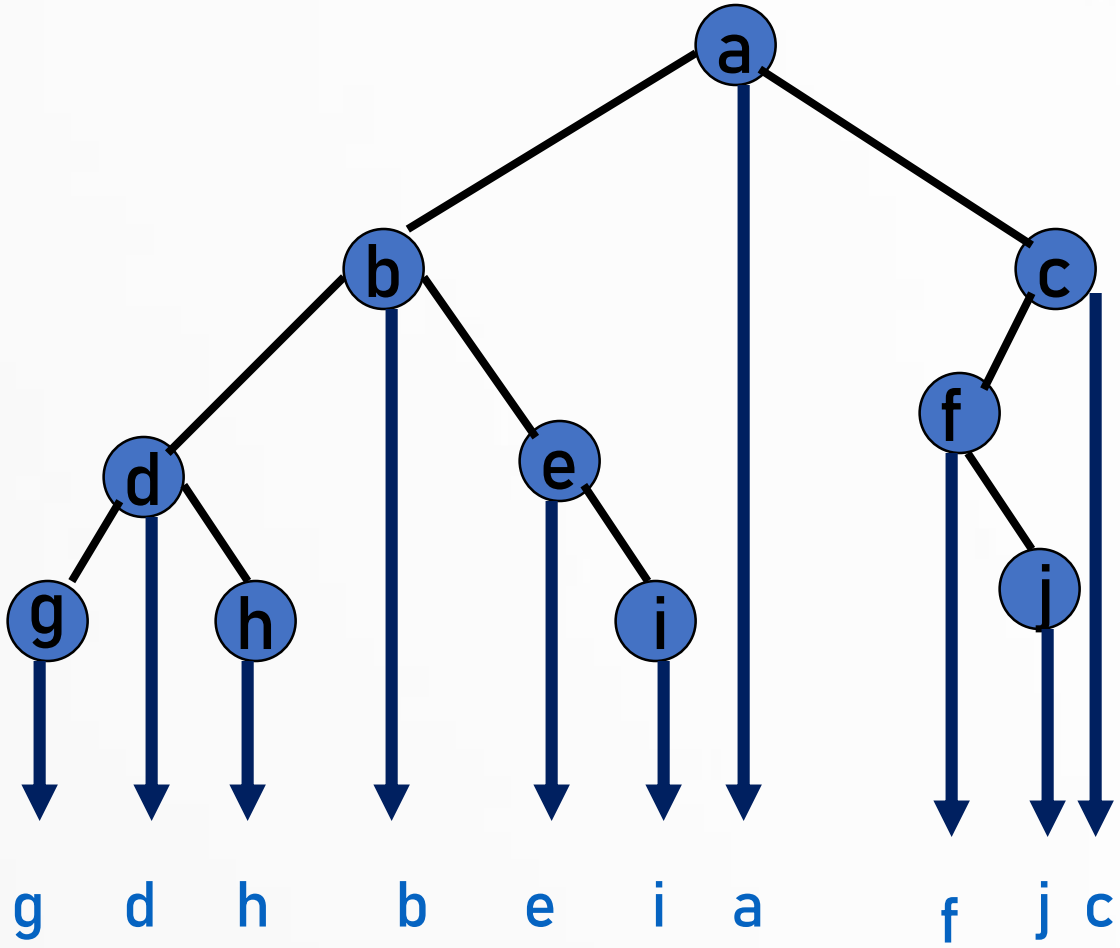
b a c

In-order Traversal: LNR

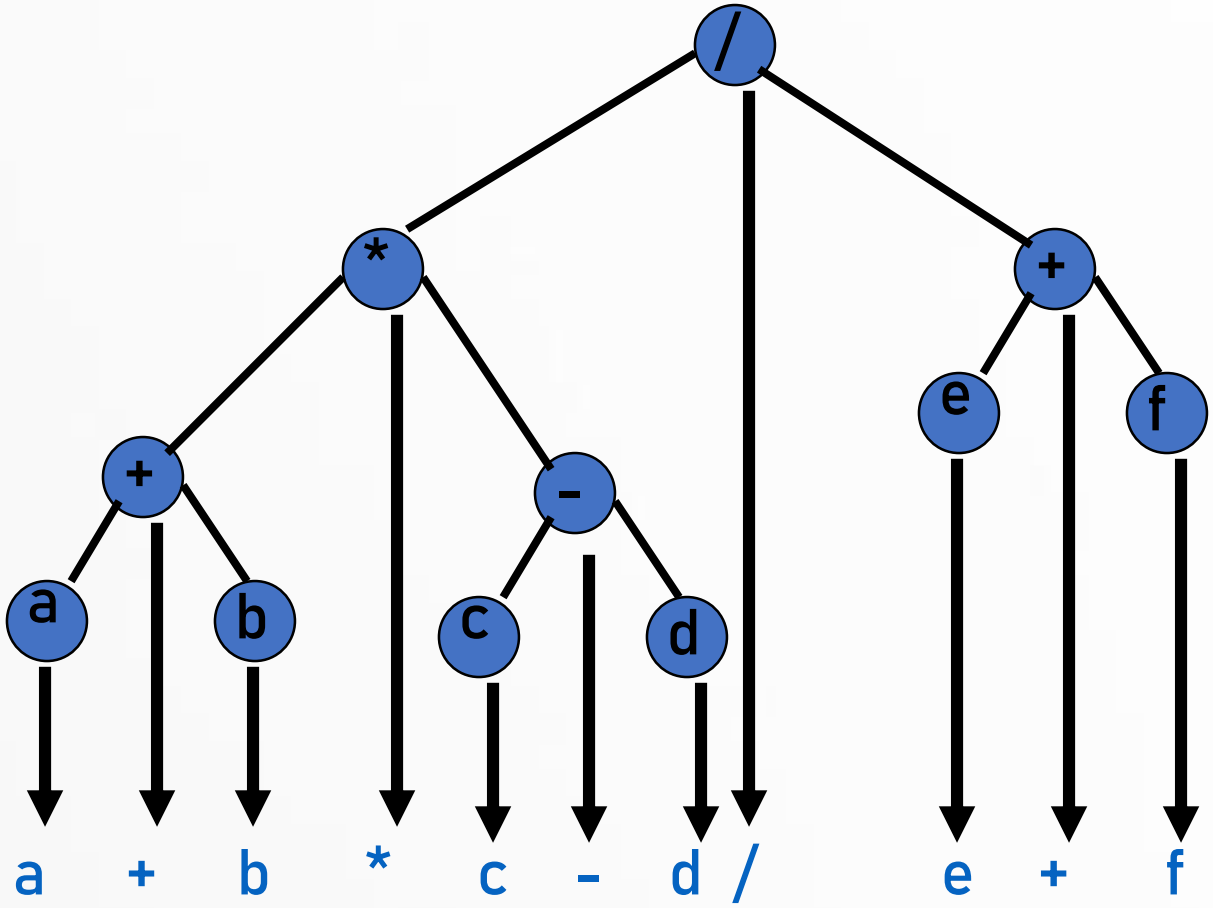


g d h b e i a f j c

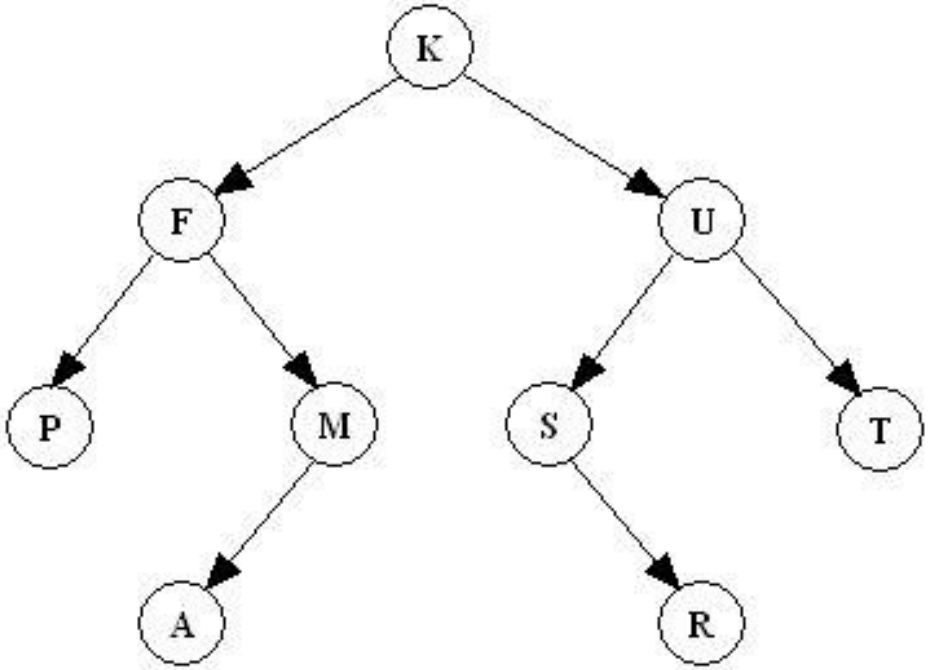
In-order Traversal: LNR



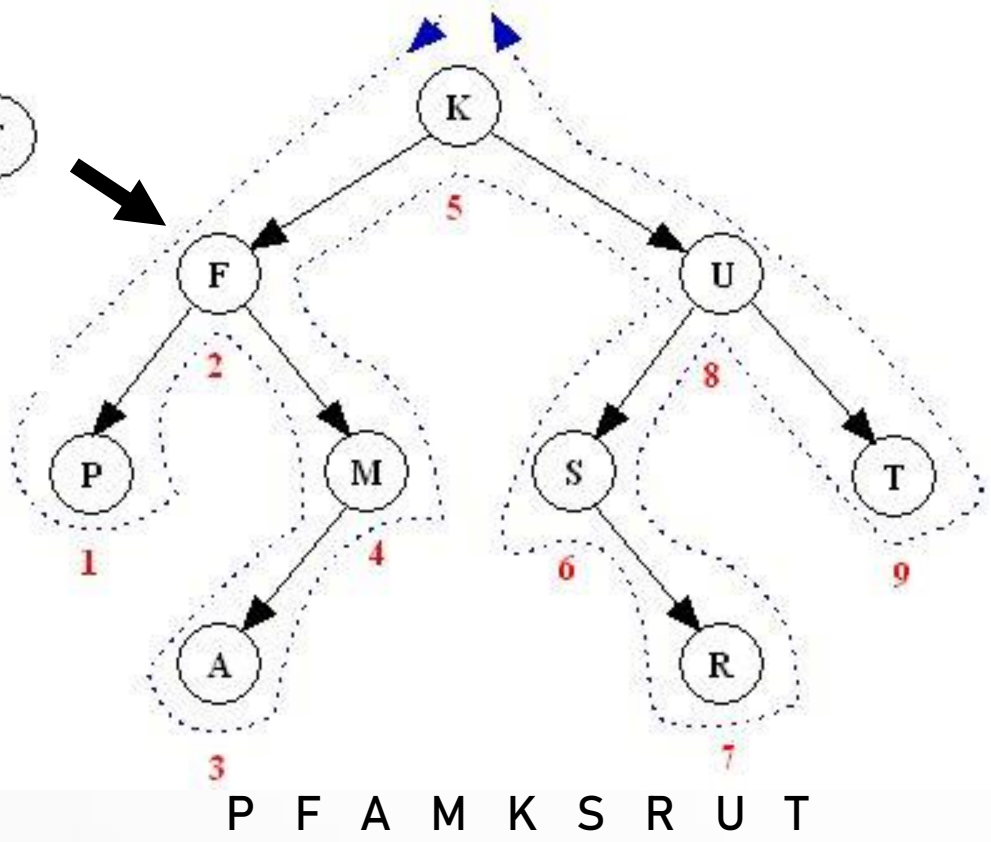
In-order Traversal: LNR



In-order Traversal: LNR



d when passing below it in the visit path"



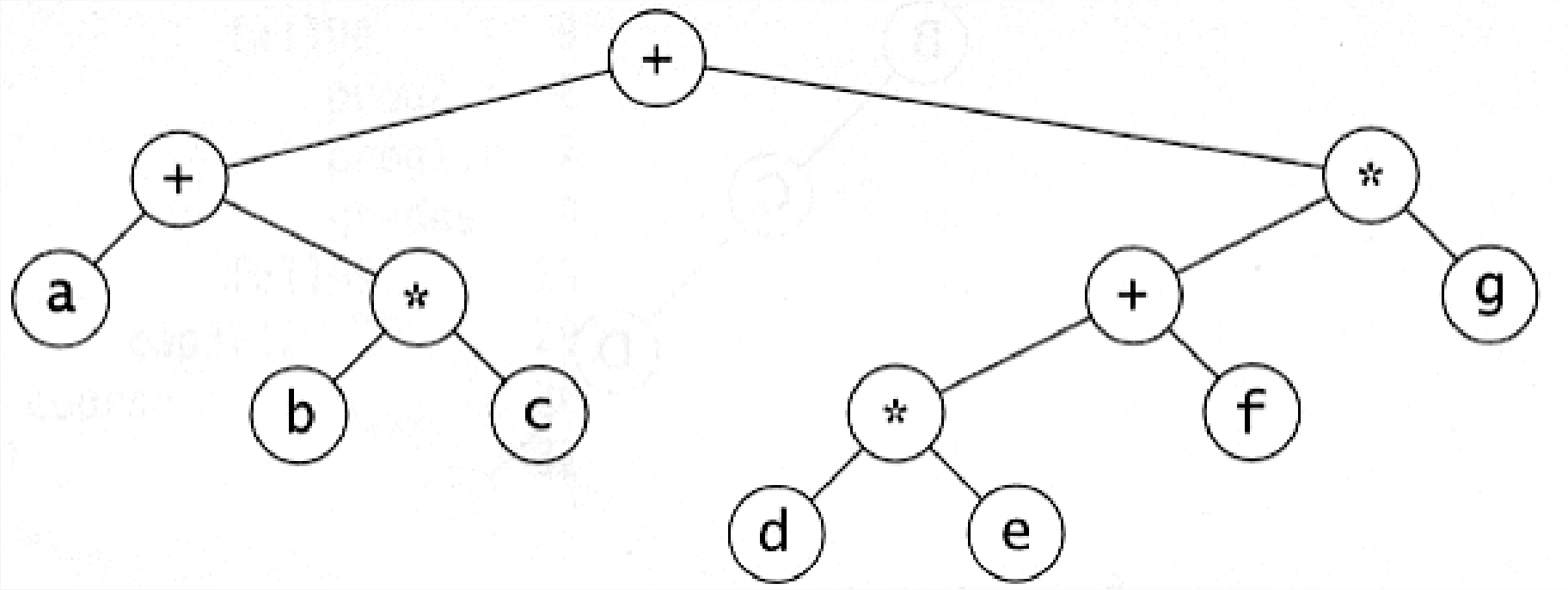
Note: An inorder traversal can pass through a node without visiting it at that moment.

P F A M K S R U T

In-order Traversal: LNR

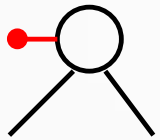
Inorder traversal left, node, right. infix expression?

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

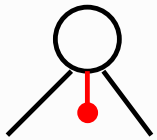


Tree traversals using “flags”

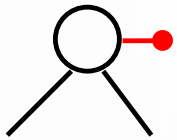
The order in which the nodes are visited during a tree traversal can be easily determined by imagining there is a “flag” attached to each node, as follows:



preorder

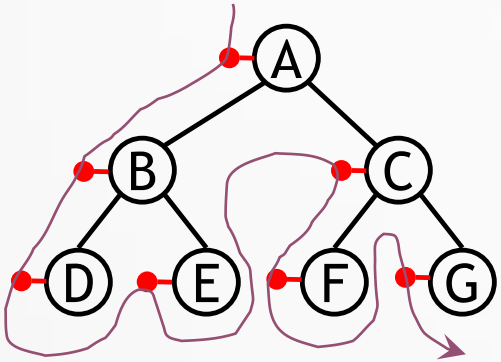


inorder

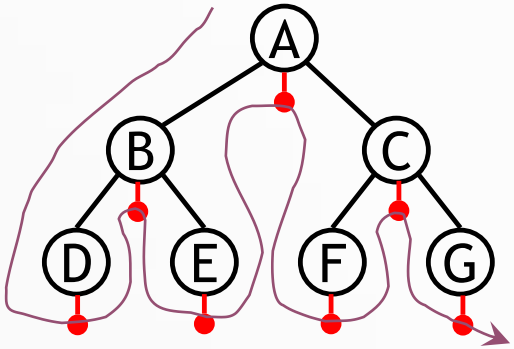


postorder

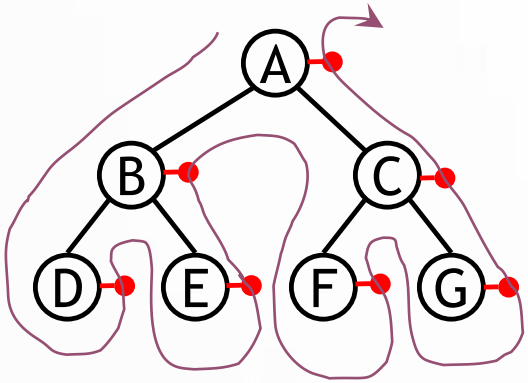
- To traverse the tree, collect the flags:



A B D E C F G

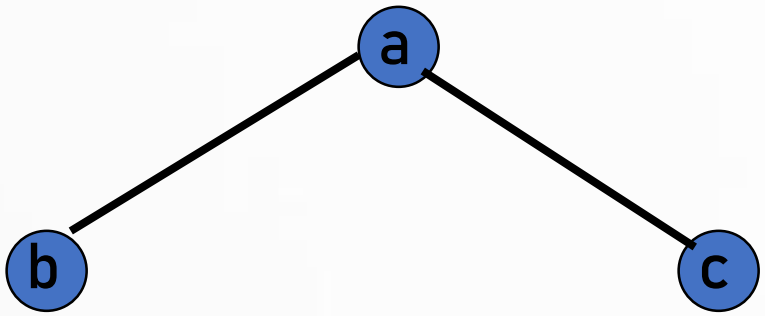


D B E A F C G



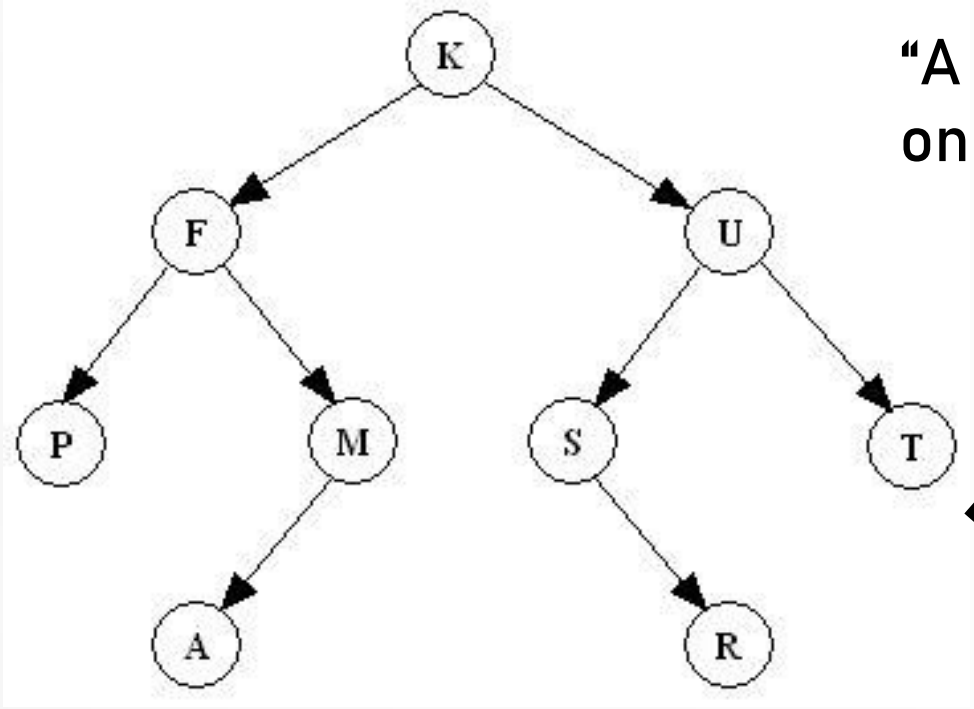
D E B F G C A

Post-order Traversal: LRV

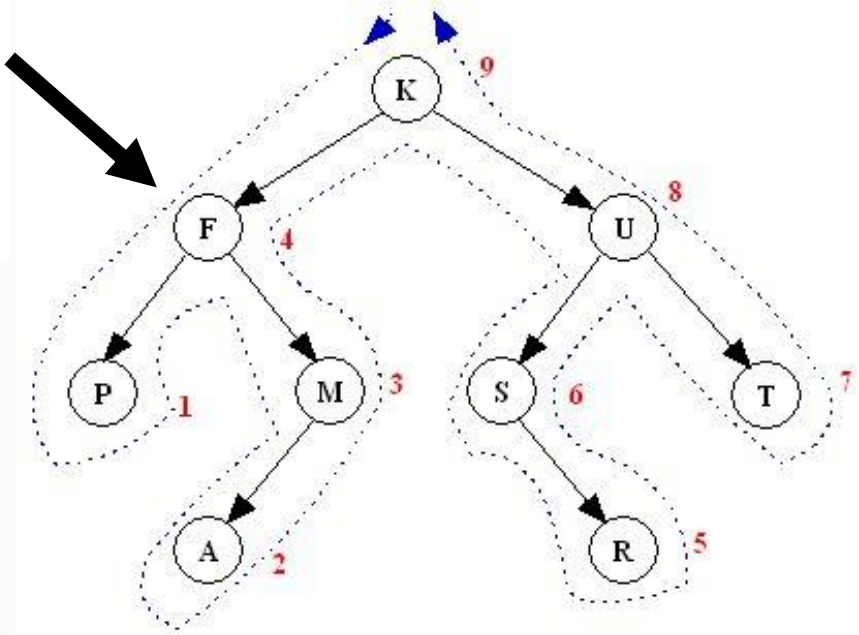


b c a

Post-order Traversal: LRV



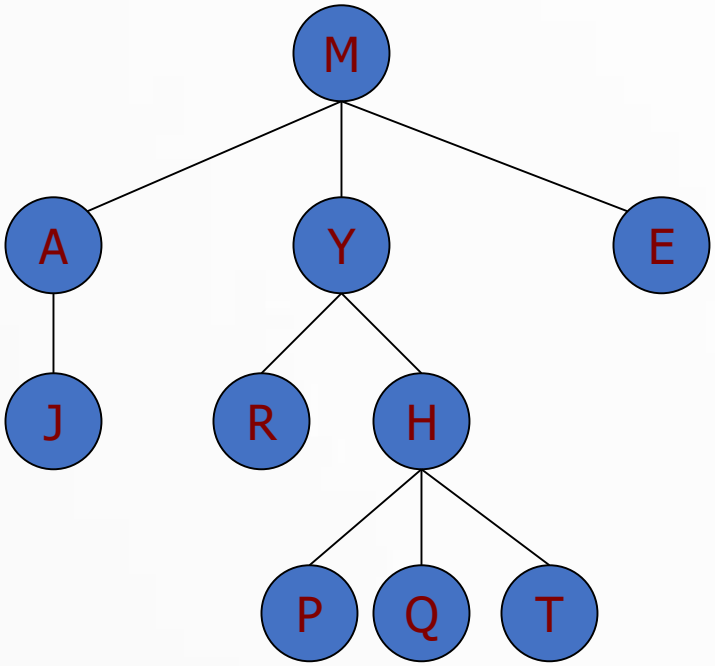
"A node is visited when passing on its right in the visit path"



P A M F R S T U K

Note: An postorder traversal can pass through a node without visiting it at that moment.

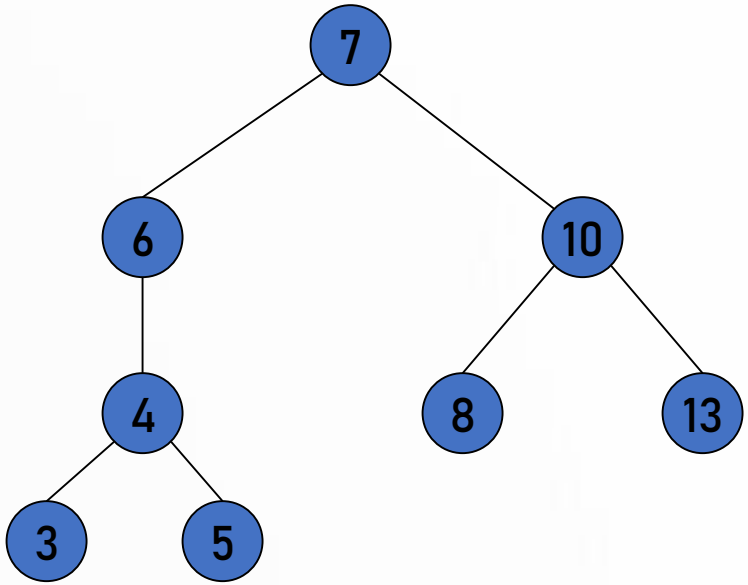
Post-order Traversal: LRV



J A R P Q T H Y E M

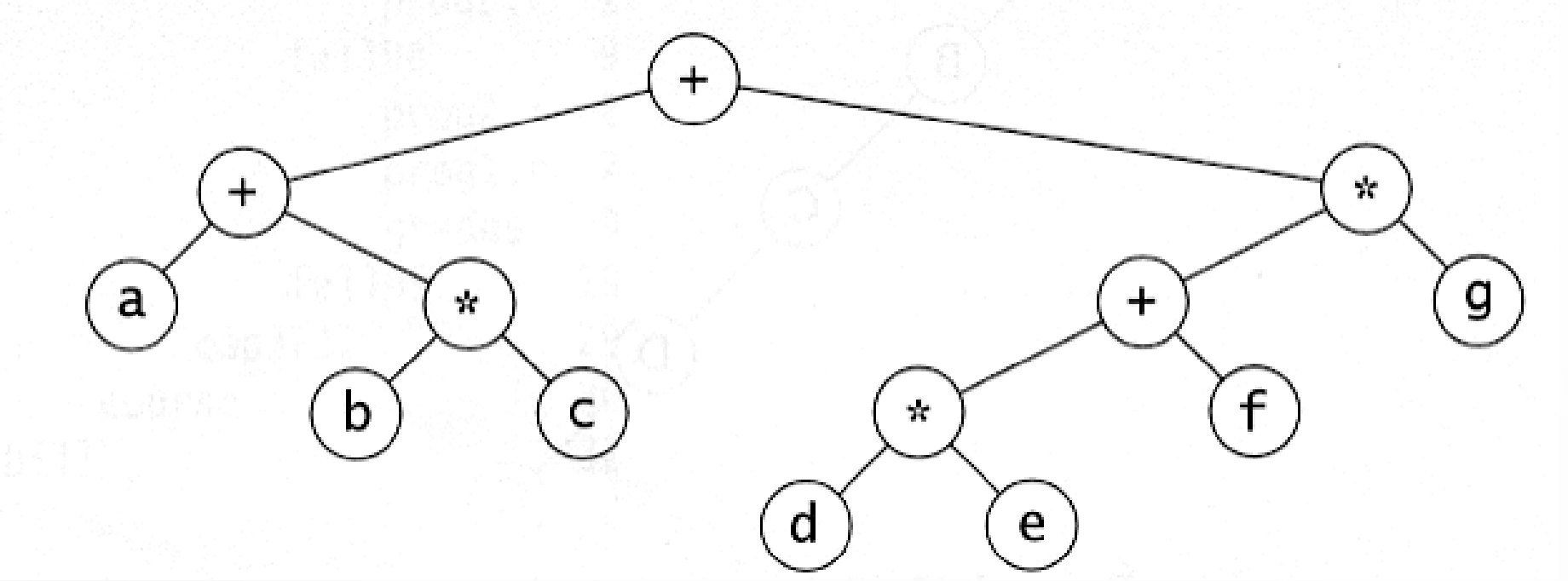
Post-order Traversal: LRV

- Do a post-order traversal of the left subtree
- Followed by a post-order traversal of the right subtree
- Visit the node
- For the sample tree
  - 3, 5, 4, 6, 8, 13, 10, 7



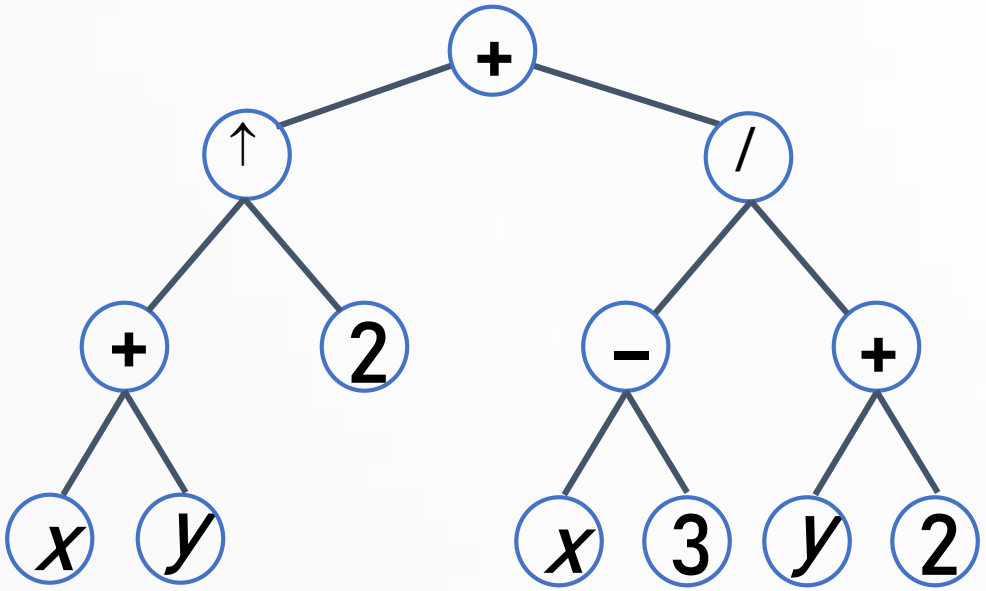
Post-order Traversal: LRV

- Postorder traversal
  - left, right, node
  - postfix expression?
    - (a+b*c)+((d*e+f)*g)



Infix Notation

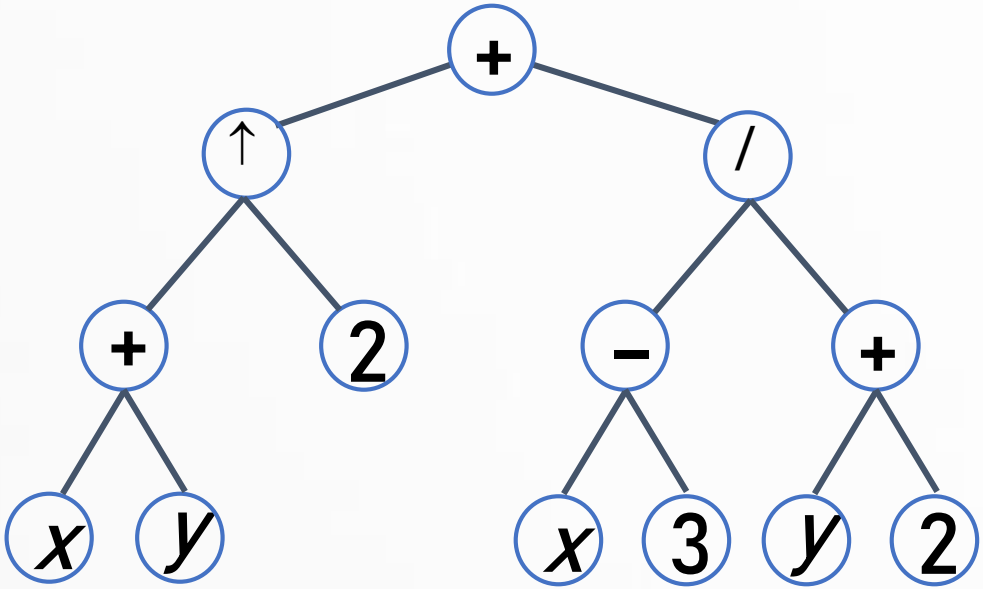
- Traverse in order (LVR) adding parentheses for each operation



$$(((x + y) \uparrow 2) + ((x - 3) / (y + 2)))$$

Prefix Notation

- Traverse in preorder (VLR)

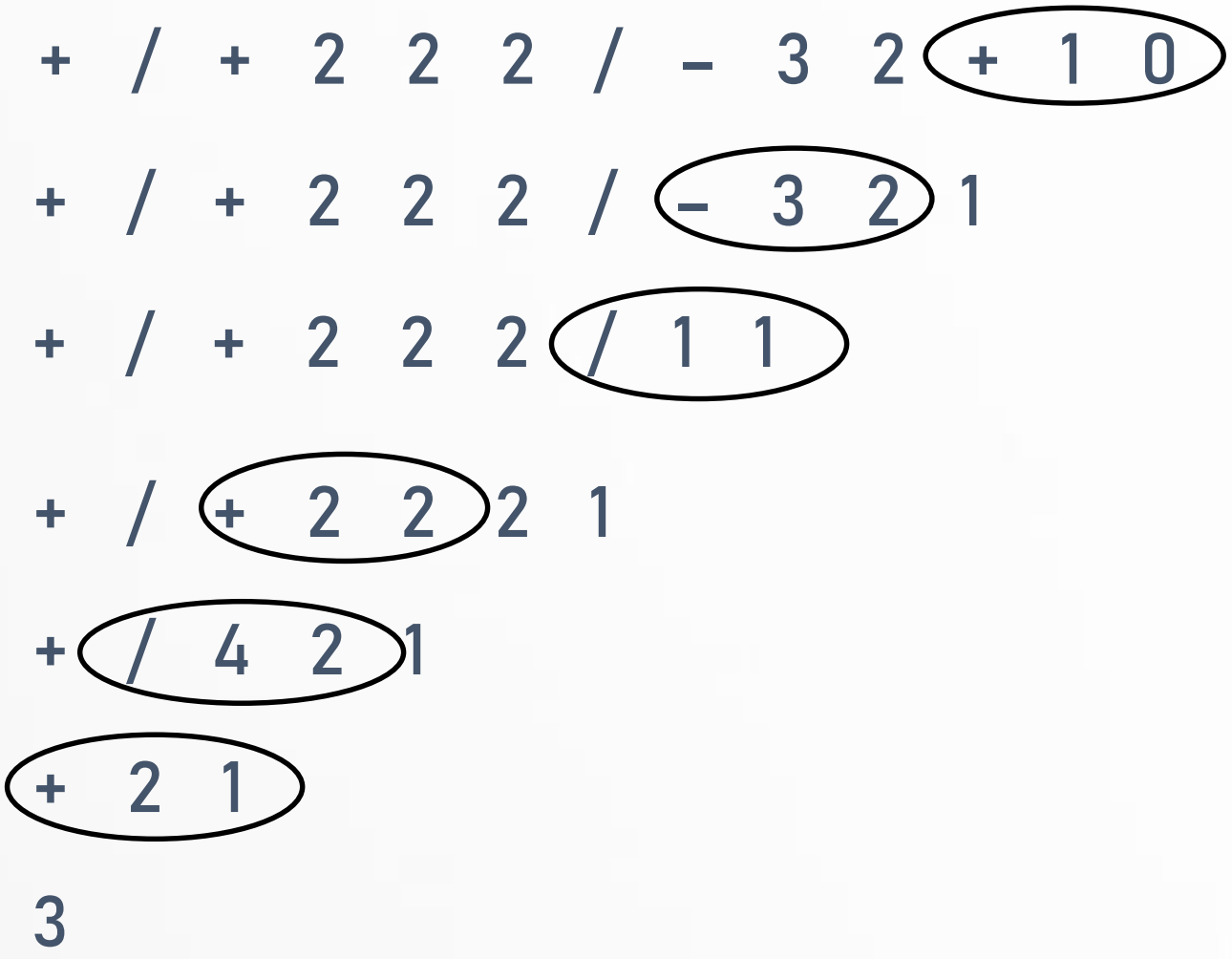


$+ \uparrow + x\ y\ 2\ /\ -\ x\ 3\ +\ y\ 2$

Evaluating Prefix Notation

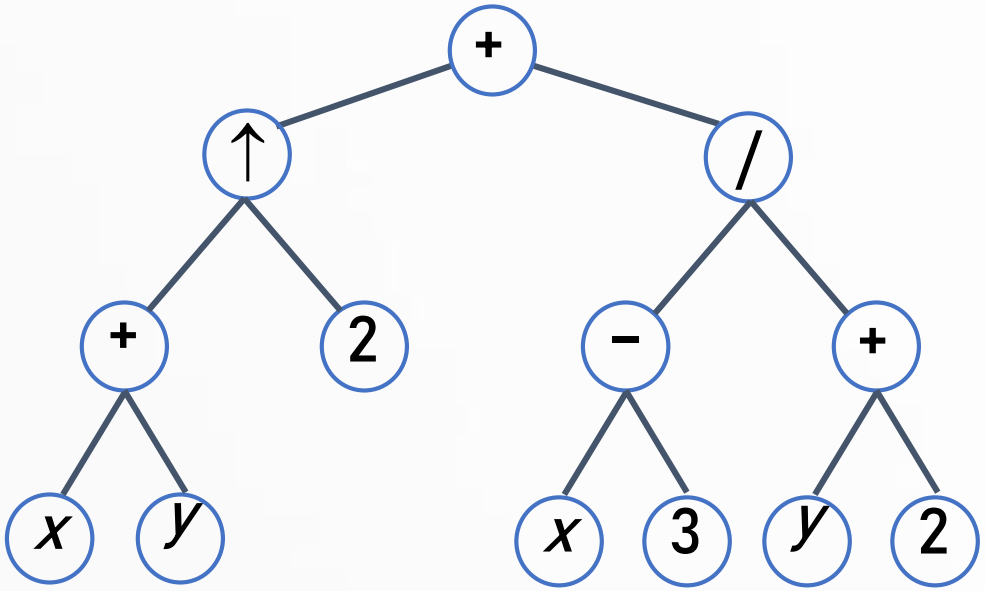
- In an prefix expression, a binary operator precedes its two operands
- The expression is evaluated right-left
- Look for the first operator from the right
- Evaluate the operator with the two operands immediately to its right

Evaluating Prefix Notation



Postfix Notation

- Traverse in postorder (LRV)



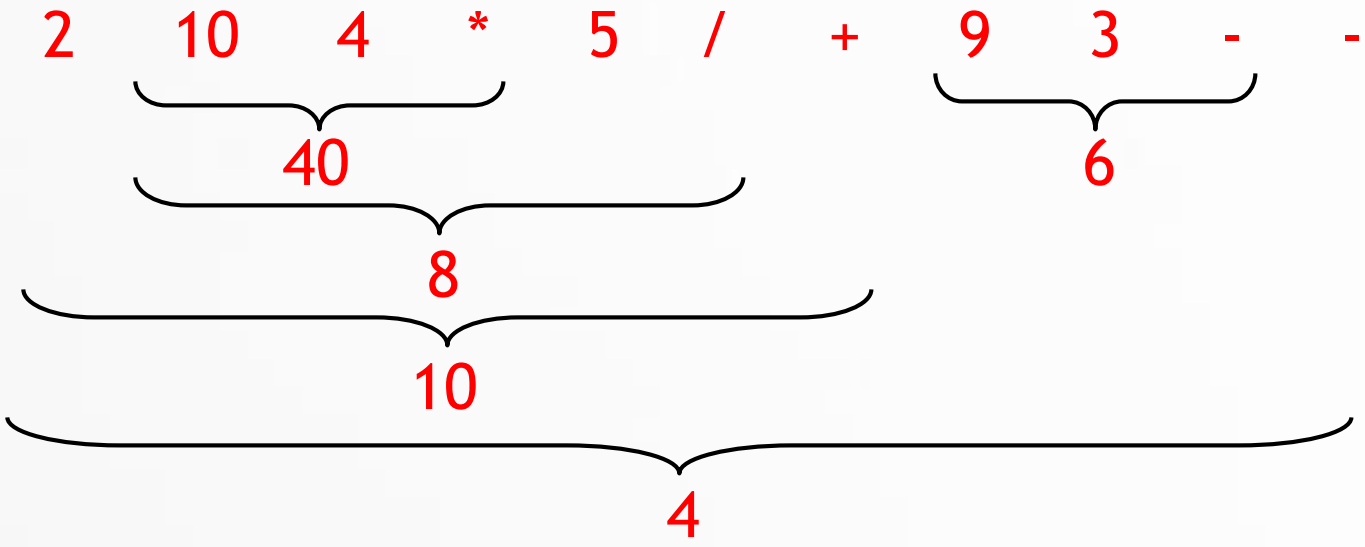
$$x\ y\ +\ 2\ \uparrow\ x\ 3\ -\ y\ 2\ +\ /\ +$$

Postfix Notation - LRV

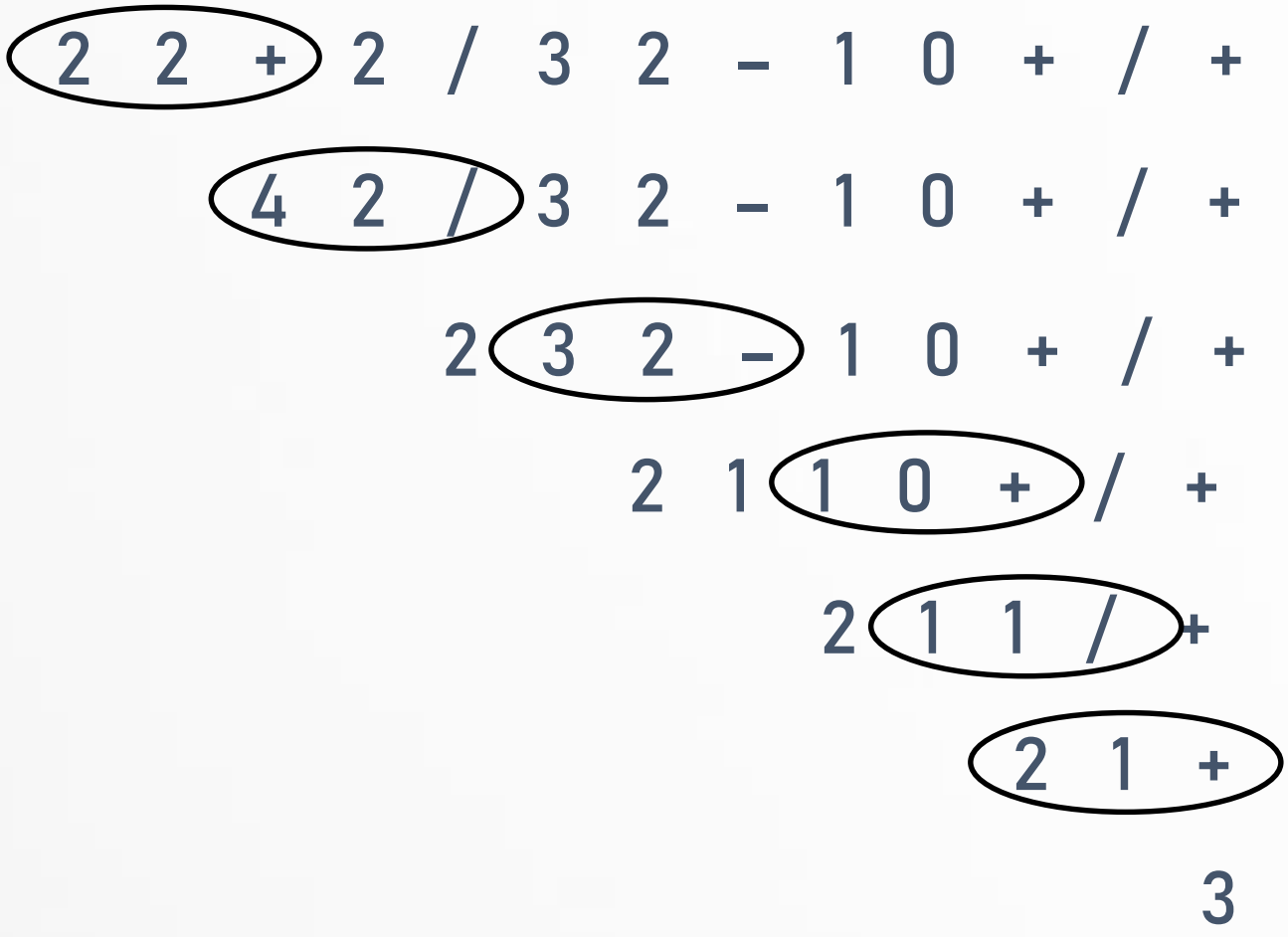
- In an postfix expression, a binary operator follows its two operands
- The expression is evaluated left-right
- Look for the first operator from the left
- Evaluate the operator with the two operands immediately to its left

Postfix Notation - LRV

- Going from left to right, if you see an operator, apply it to the previous two operands (numbers)
- Example:



Postfix Notation - LRV



That's all for now…