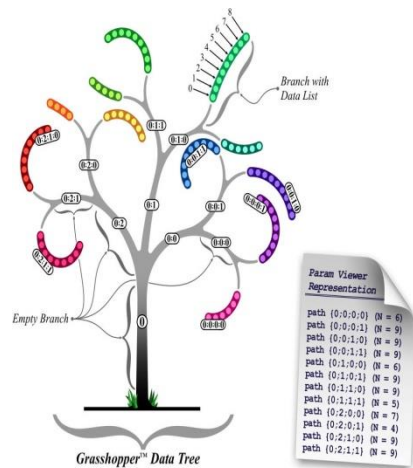


Data Structures

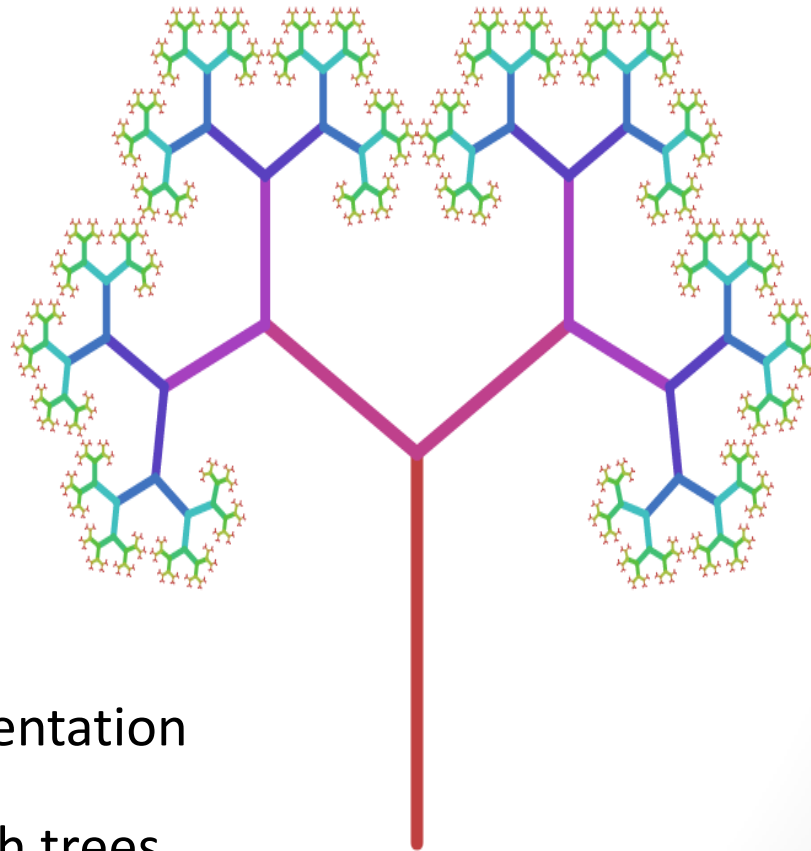


Chapter 4 Trees 1



Trees

- What is a Tree?
- Tree terminology
- Why trees?
- What is a general tree?
- Constructing trees
- Binary trees
- Binary Search tree implementation
- Application of Binary Search trees

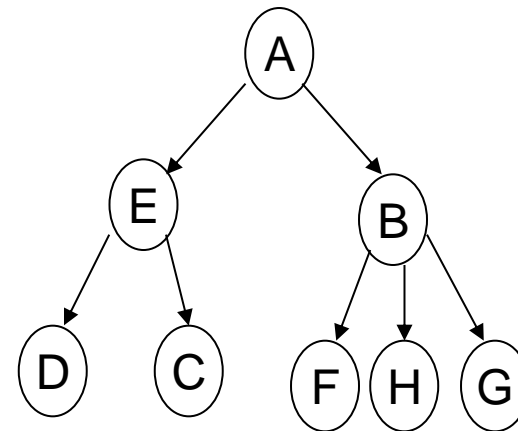


What is a Tree?

- A tree, is a finite set of nodes together with a finite set of directed edges that define parent-child relationships. Each directed edge connects a parent to its child. Example:

Nodes={A,B,C,D,E,f,G,H}

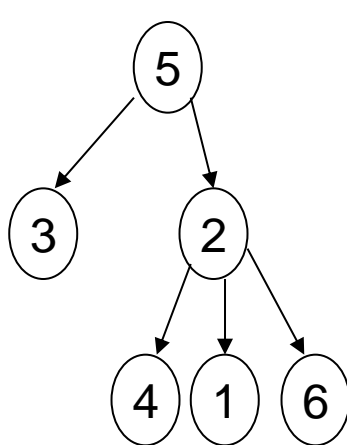
Edges={(A,B),(A,E),(B,F),(B,G),(B,H),
(E,C),(E,D)}



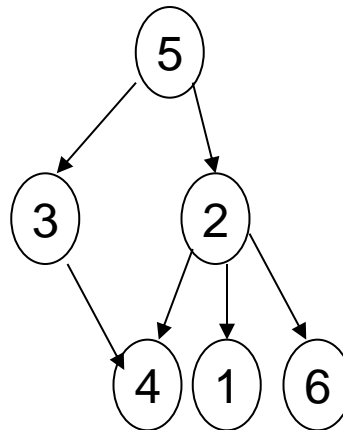
- A directed **path** from node m_1 to node m_k is a list of nodes m_1, m_2, \dots, m_k such that each is the parent of the next node in the list. The length of such a path is **$k - 1$** .
- Example: **A, E, C** is a directed path of length 2.

What is a Tree?

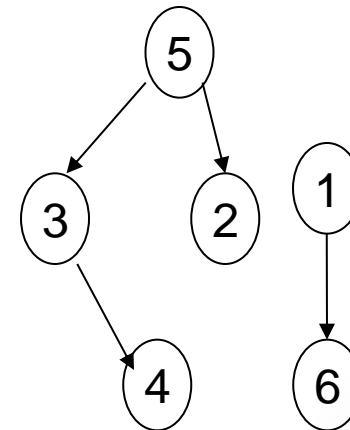
- A tree satisfies the following properties:
 - It has **one designated node**, called the **root**, that has **no parent**.
 - Every node, except the root, **has exactly one parent**.
 - A node may **have zero** or more children.
 - There is a **unique directed** path from the root to each node.



tree



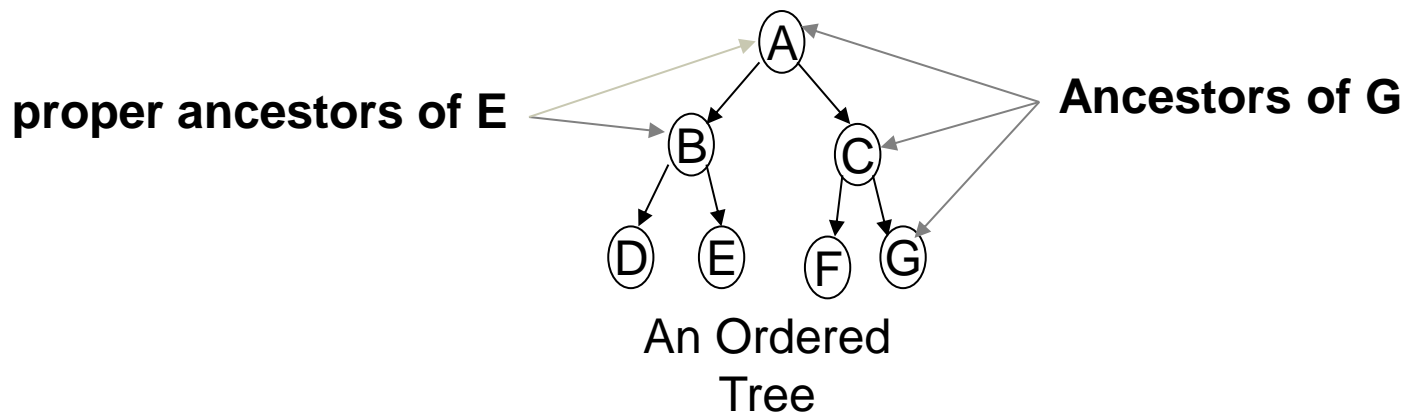
Not a tree



Not a tree

Tree Terminology

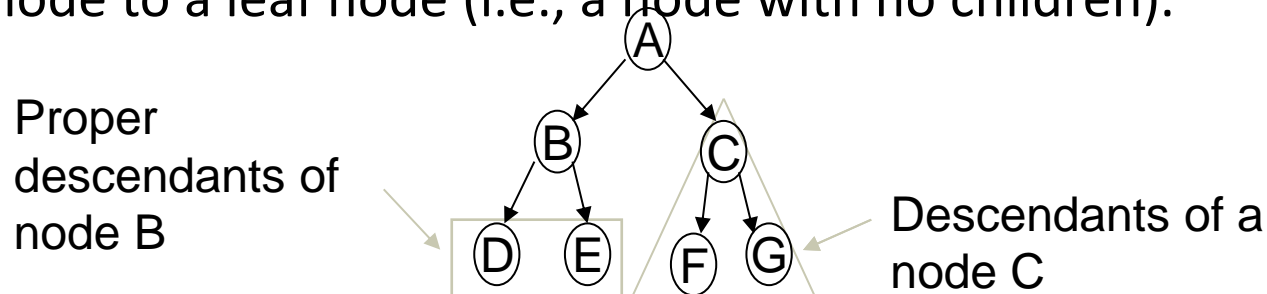
- Ordered tree: A tree in which the children of each node **are linearly ordered** (usually from left to right).



- **Ancestor** of a node v : Any node, **including v itself**, on the path from the root to the node.
- **Proper ancestor** of a node v : Any node, **excluding v** , on the path from the root to the node.

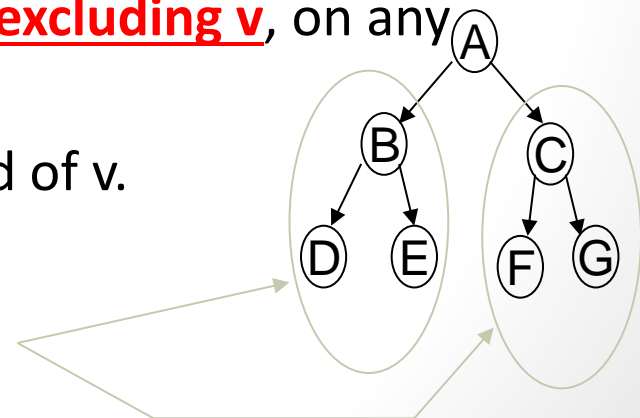
Tree Terminology (Contd.)

- **Descendant** of a node v : Any node, including v itself, on any path from the node to a leaf node (i.e., a node with no children).

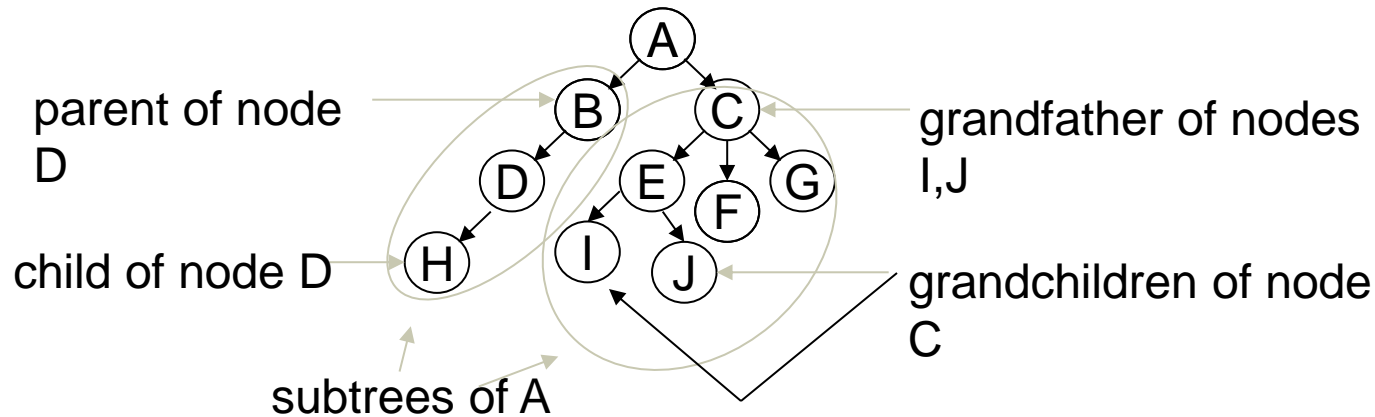


- **Proper descendant** of a node v : Any node, excluding v , on any path from the node to a leaf node.
- **Subtree** of a node v : A tree rooted at a child of v .

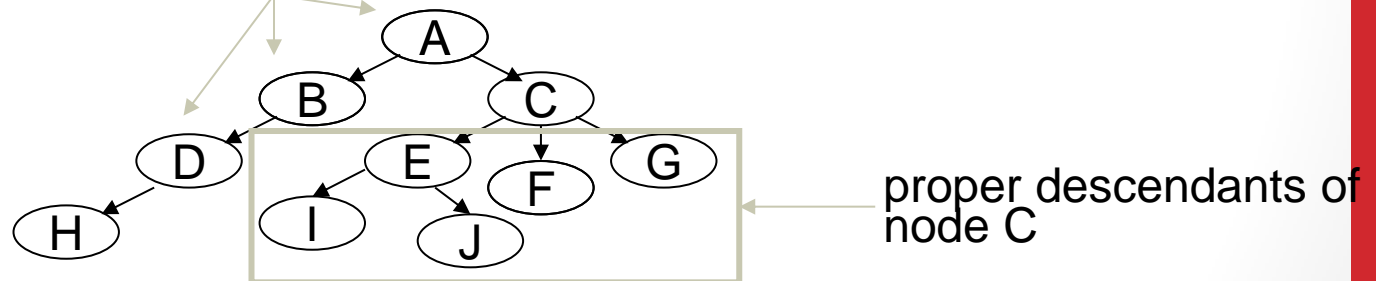
subtrees of node A



Tree Terminology (Contd.)



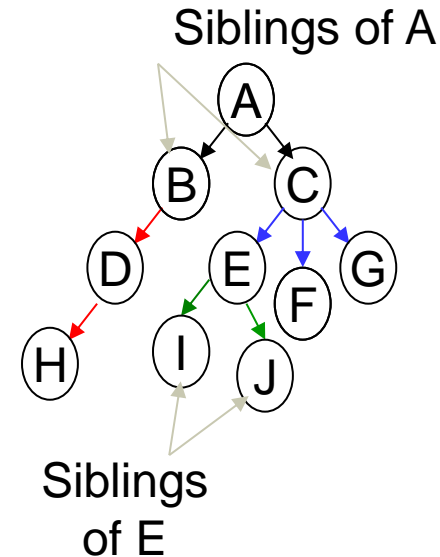
proper ancestors of node H



Tree Terminology

Tree with size of 10

- Degree: The number of subtrees of a node
 - Each of node D and B has degree **1**.
 - Each of node A and E has degree **2**.
 - Node C has degree **3**.
 - Each of node F,G,H,I,J has degree **0**.

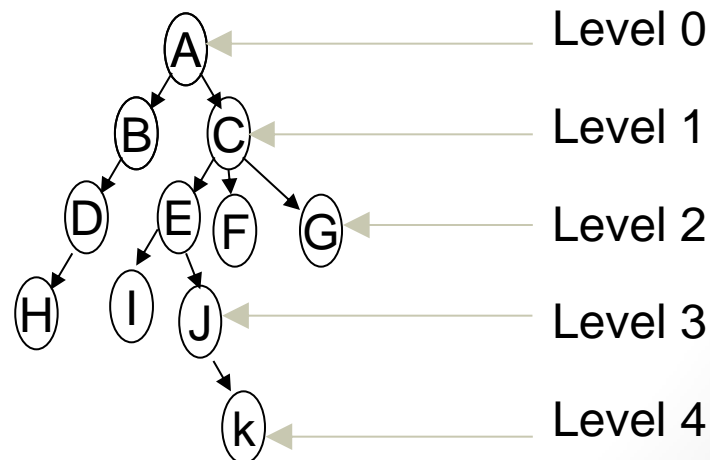


- **Leaf:** A node with degree 0.
- **Internal** or interior node: a node with degree greater than 0.
- **Siblings:** Nodes that have the same parent.
- **Size:** The number of nodes in a tree.

Tree Terminology (Contd.)

- **Level (or depth)** of a node v : The length of the path from the **root** to v .
- **Height** of a node v : The length of the **longest path from v to a leaf** node.
 - The **height of a tree** is the height of its **root node**.
 - By definition the height of an empty tree is **-1**.

- The height of the tree is 4.
- The height of node C is 3.

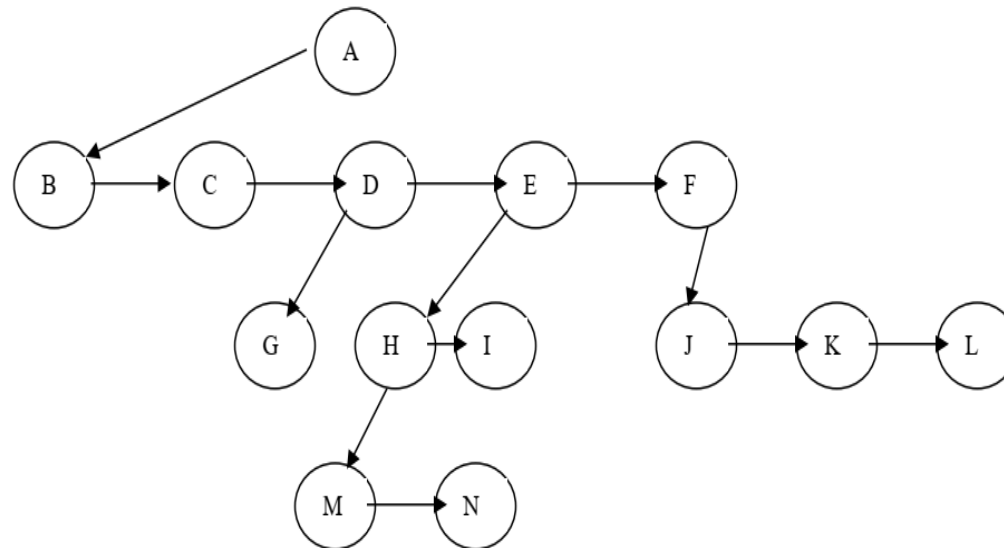


Why Trees?

- Trees are very important data structures in computing.
- They are suitable for:
 - Hierarchical structure representation, e.g.,
 - File directory.
 - Organizational structure of an institution.
 - Class inheritance tree.
 - Problem representation, e.g.,
 - Expression tree.
 - Decision tree.
 - Efficient algorithmic solutions, e.g.,
 - Search trees.
 - Efficient priority queues via heaps.

General Trees and its Implementation

- In a general tree, there is **no limit to the number** of children that a node can have.
- Representing a general tree by **linked lists**:
 - Each node **has a linked list of the subtrees** of that node.
 - Each element of the linked list is a subtree of the current node

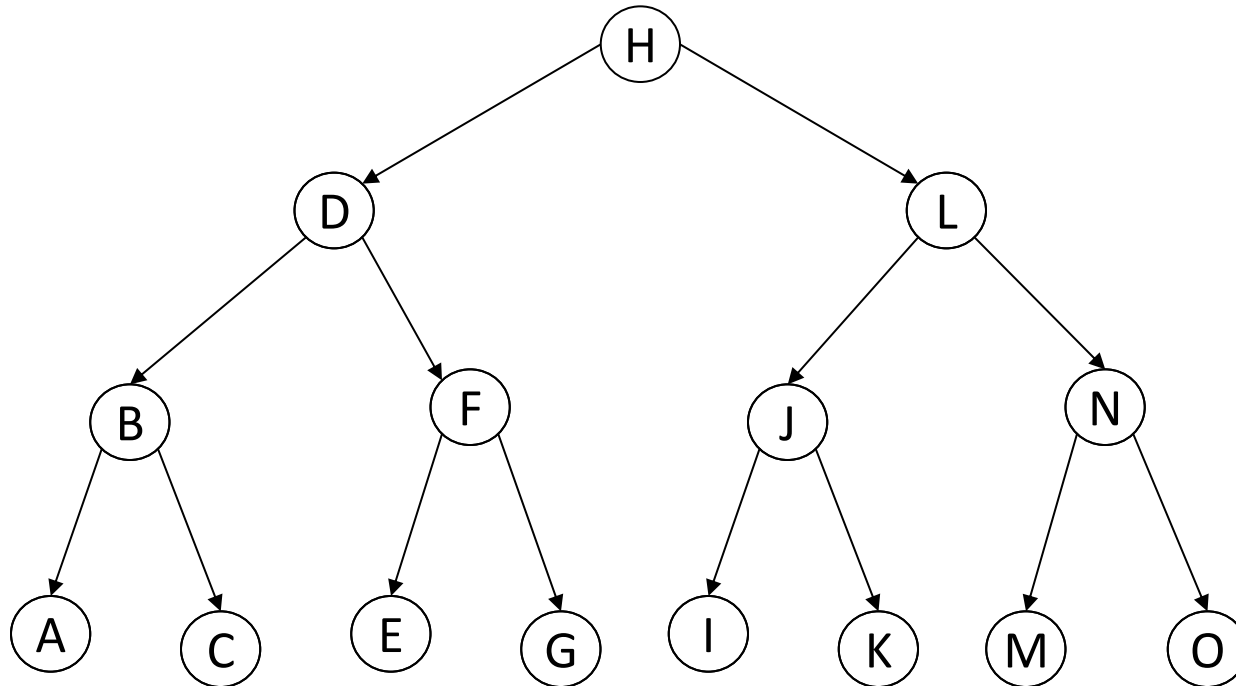


Tree Traversal

- Visiting (e.g. printing) all nodes.
- **Three methods**
 - Pre-order : root-Left-Right
 - In-order : Left-root-Right
 - Post-order : Left-Right-root

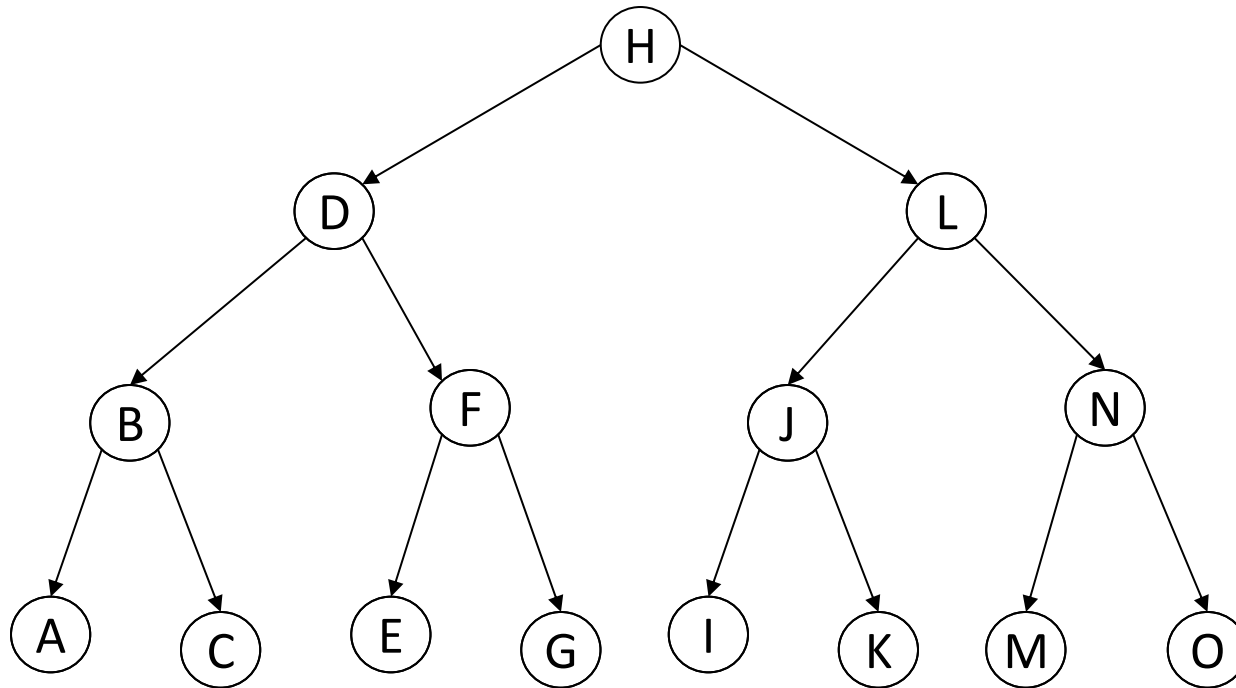
Tree Traversal

Preorder root-L-R



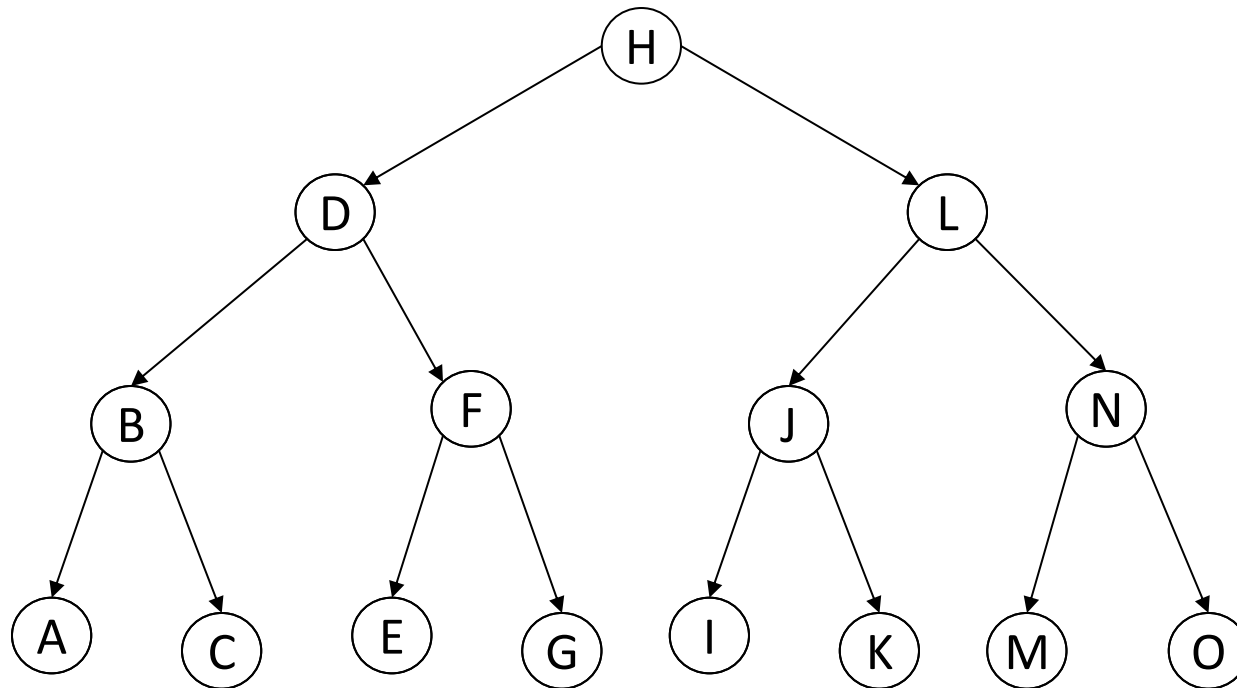
H	D	B	A	C	F	E	G	L	J	I	K	N	M	O
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

In-order L-root-R



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Post-order L-R-root



A	C	B	E	G	F	D	I	K	J	M	O	N	L	H
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

InOrder(root) visits nodes in the following order:

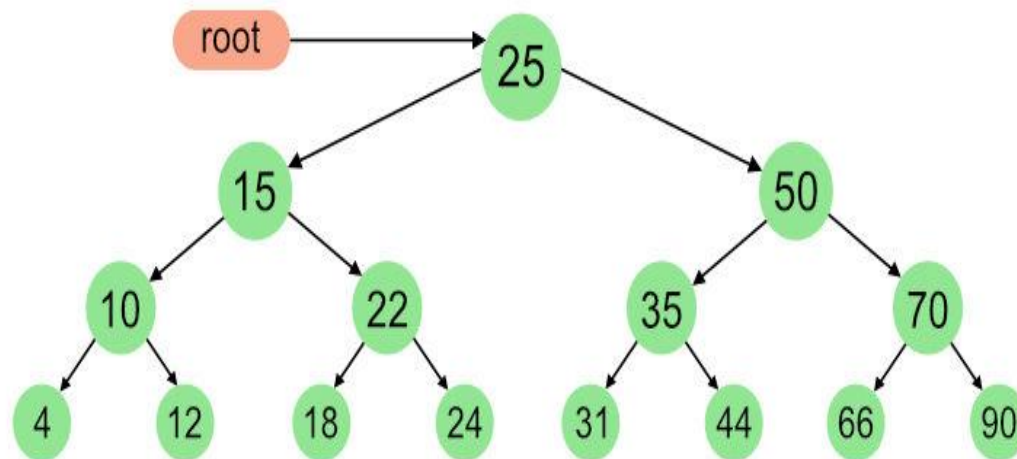
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

A Pre-order traversal visits nodes in the following order:

25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90

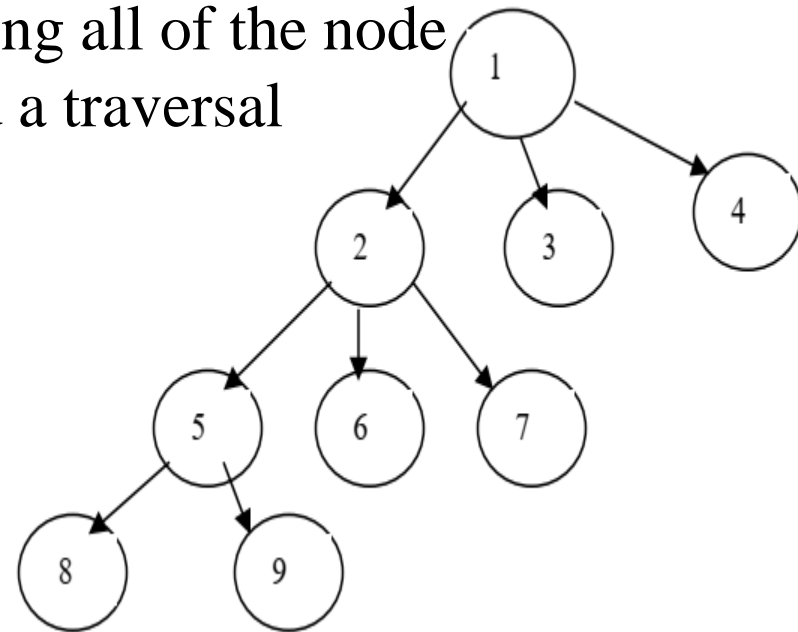
A Post-order traversal visits nodes in the following order:

4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25



Tree Traversal

- Any process for visiting all of the nodes in some order is called a traversal



1) Pre order

root , left , right

1, 2, 5, 8, 9, 6, 7, 3, 4

2) Post order

left , right , root

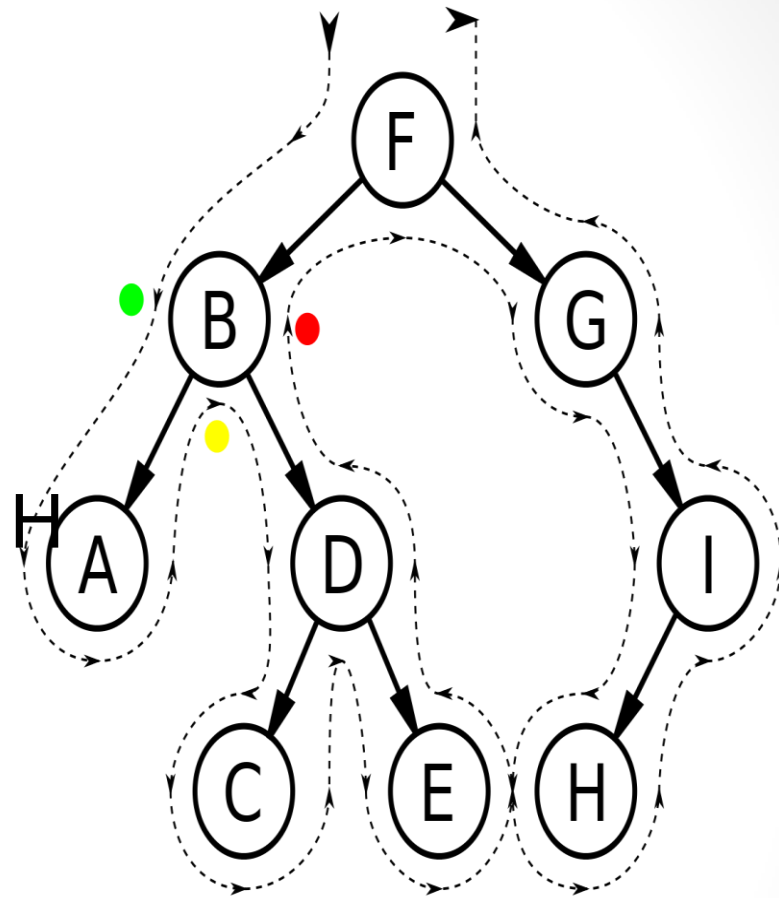
8, 9, 5, 6, 7, 2, 3, 4, 1

3) In order

left , root , right

8, 5, 9, 2, 6, 7, 1, 3, 4

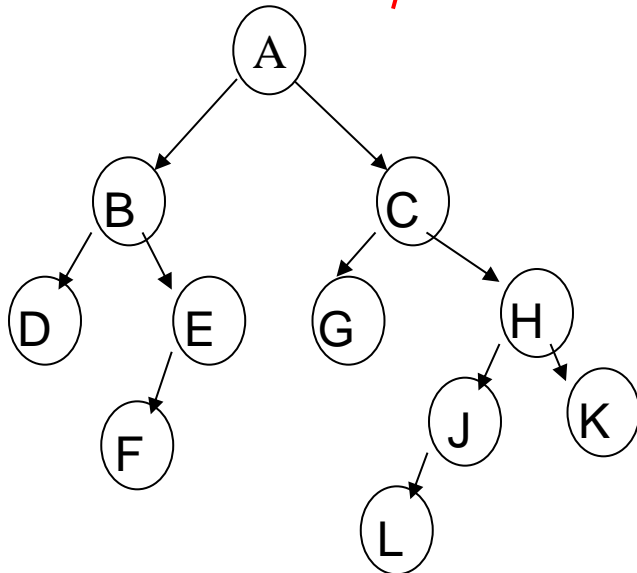
- 1) Pre order
root ,left, right
F, B, A,D, C, E,G,I, H
- 2) Post order
left , right , root
A, C, E, D, B, H, I,
G,F
- 3) In order
left , root , right
A, C, D, E, B, F, H, I,
G



Construct the binary tree whose traversal are given below

- In order : D B F E A G C L J H K
- Pre order : A B D E F C G H J L K

- In order : (D B F E) A (G C (L J H) K)
- Pre order : A B D E F C G H J L K



Exercise (very Important)

Construct the binary tree whose traversal are given below

Preorder

H	D	B	A	C	F	E	G	L	J	I	K	N	M	O
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

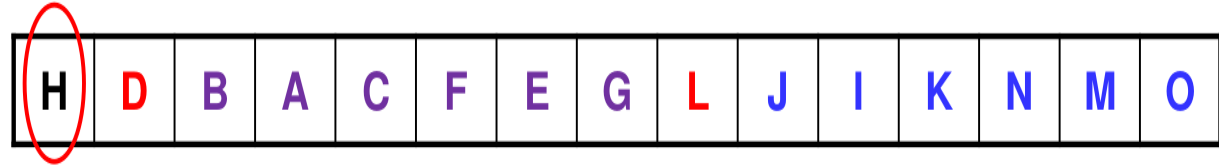
Postorder

A	C	B	E	G	F	D	I	K	J	M	O	N	L	H
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

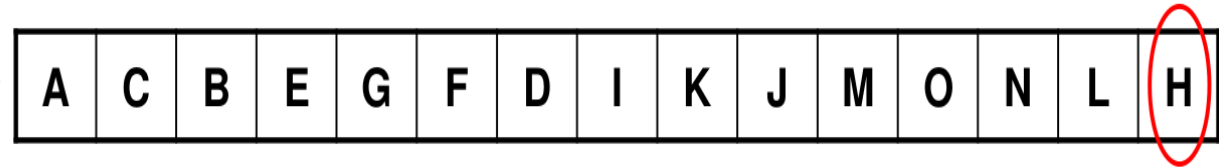
N1=predecessor of root in postorder (right)

N2=successor of root in preorder (left)

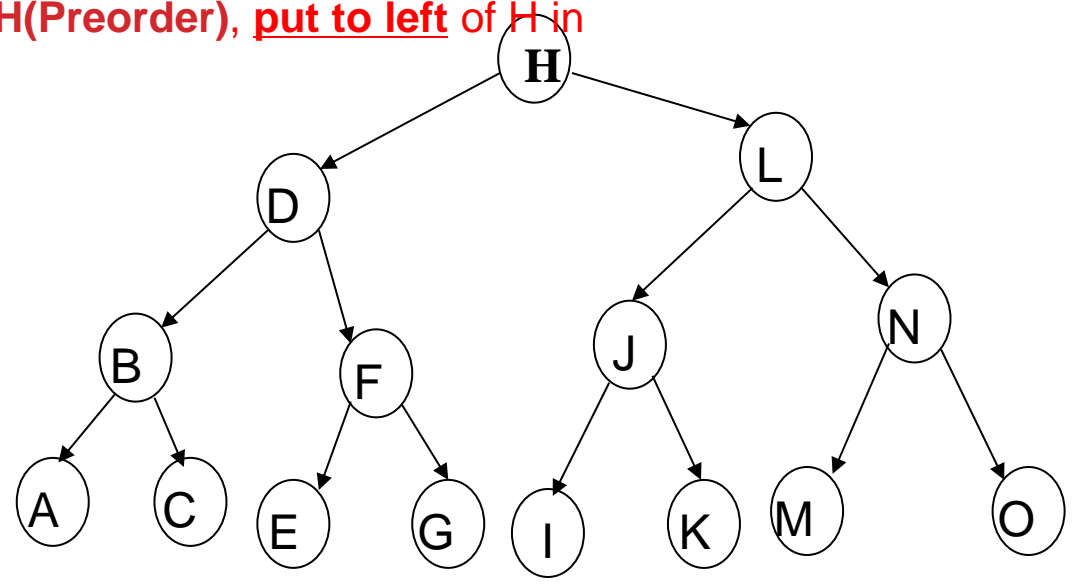
Preorder



Postorder



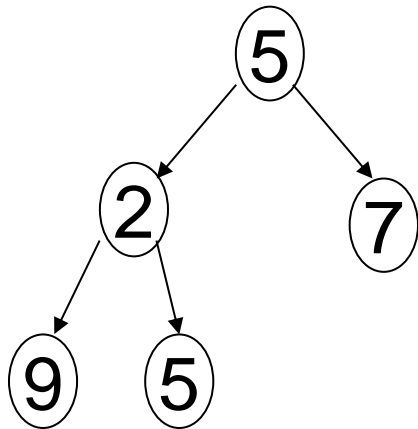
The left of node H (postorder) ,put on right of H in tree
 The right of node H(Preorder), put to left of H in tree



Binary Trees

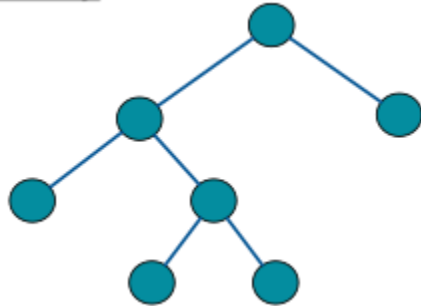
- A **binary tree** is a tree in which node can **have no more than two** children
- Thus, a binary tree is either:
 1. An empty tree, or
 2. A tree consisting of a root node and at most two non-empty binary subtrees.

Example:

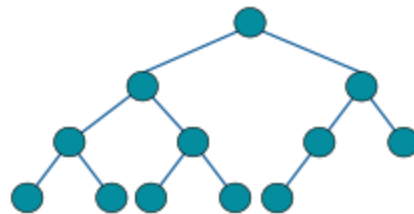


Types of Binary Trees

- A full binary if every node has 0 or 2 children.

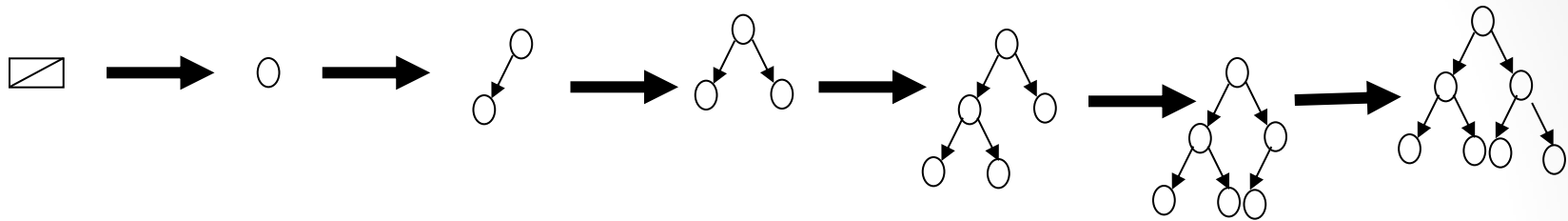


- A complete binary every level, *except possibly the last*, is completely filled, and all nodes in the last level are as far left as possible.

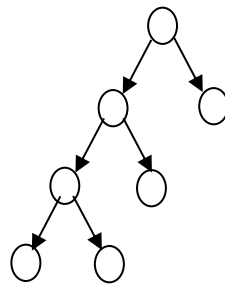


Binary Trees (Contd.)

- Example showing the growth of a complete binary tree:



- Full, but not complete: Huffman coding example

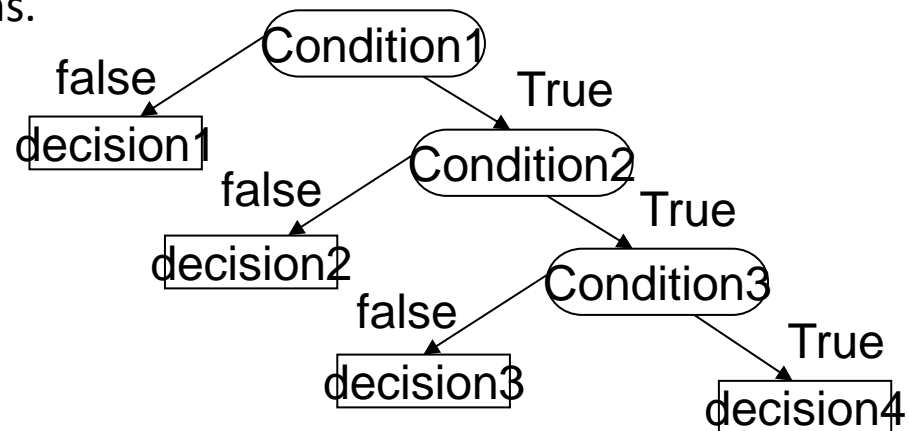


Application of Binary Trees

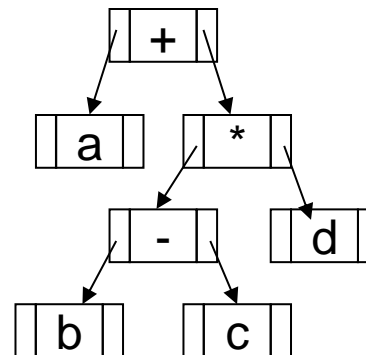
- Binary trees have many important uses. Two examples are:

1. Binary decision trees.

- Internal nodes are conditions. Leaf nodes denote decisions.



2. Expression Trees



Constructing an Expression Tree

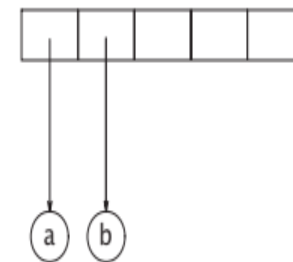
Since we already have an algorithm to convert infix to postfix, we can generate expression trees from the two common types of input

We read our **expression one symbol at a time**. If the symbol is an **operand**, we create a **one-node tree** and **push it onto a stack**. If the symbol is an operator, we pop two trees T_1 and T_2 from the stack (T_2 is popped first) and form a new tree whose root is the operator and whose left and right children are T_1 and T_2 , respectively. This new tree is then pushed onto the stack.

As an example,
suppose the input is

$a b + c d e + * *$

The first two symbols are operands, so we create one-node trees and push them onto a stack.

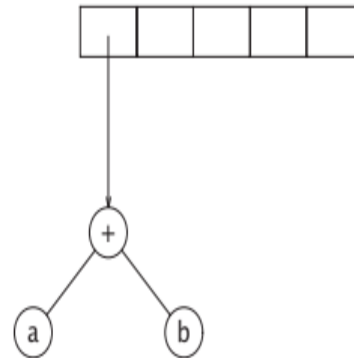


**As an example,
suppose the input is**

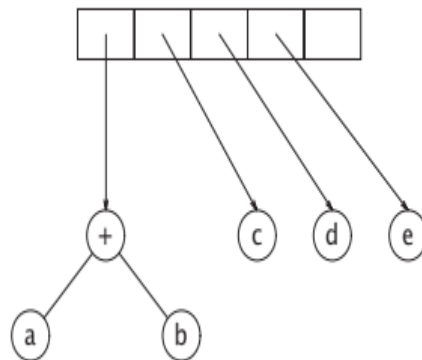
$a b + c d e + * *$

Next, a **+** is read, so two trees are popped, a new tree is formed, and it is pushed onto the stack

a b + c d e + * *

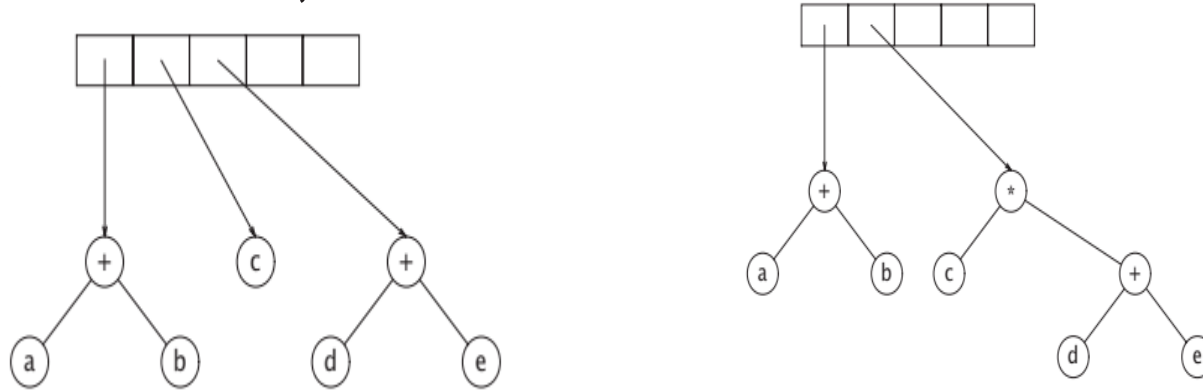


Next, c, d, and e are read, and for each a one-node tree is created and the corresponding nodes are pushed onto the stack



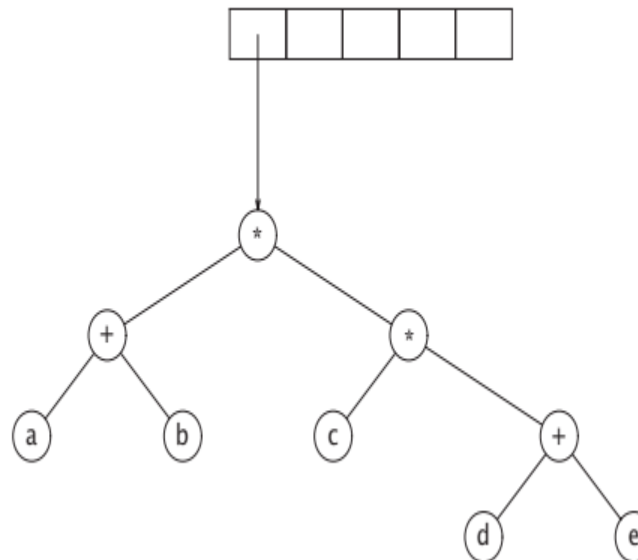
Now a + is read, so two trees are merged

a b + c d e + * *



Continuing, a * is read, so we pop two trees and form a new tree with a * as root.

Finally, the last symbol is read, two trees are merged, and the final tree is left on the stack

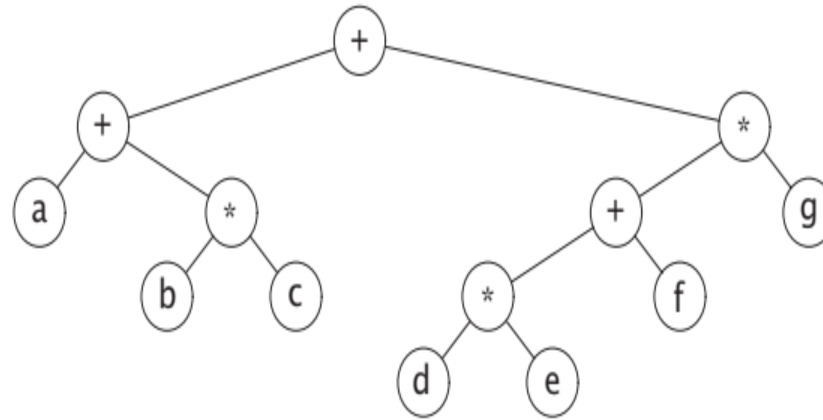


Construct the binary tree whose traversal are given below
(solved at class)

Post order a b c* + d e * f + g * **+Postfix**

Construct the binary tree whose traversal are given below
(solved at class)

Pre order + + a * b c * + * d e f **Prefix**
g



Pre order + + a * b c * + * d e f **Prefix**
g

Post order a b c * + d e * f + g * **Postfix**

In order (a + (b * c)) + (((d * e) + f) * g). **Infix**

Construct expression tree?

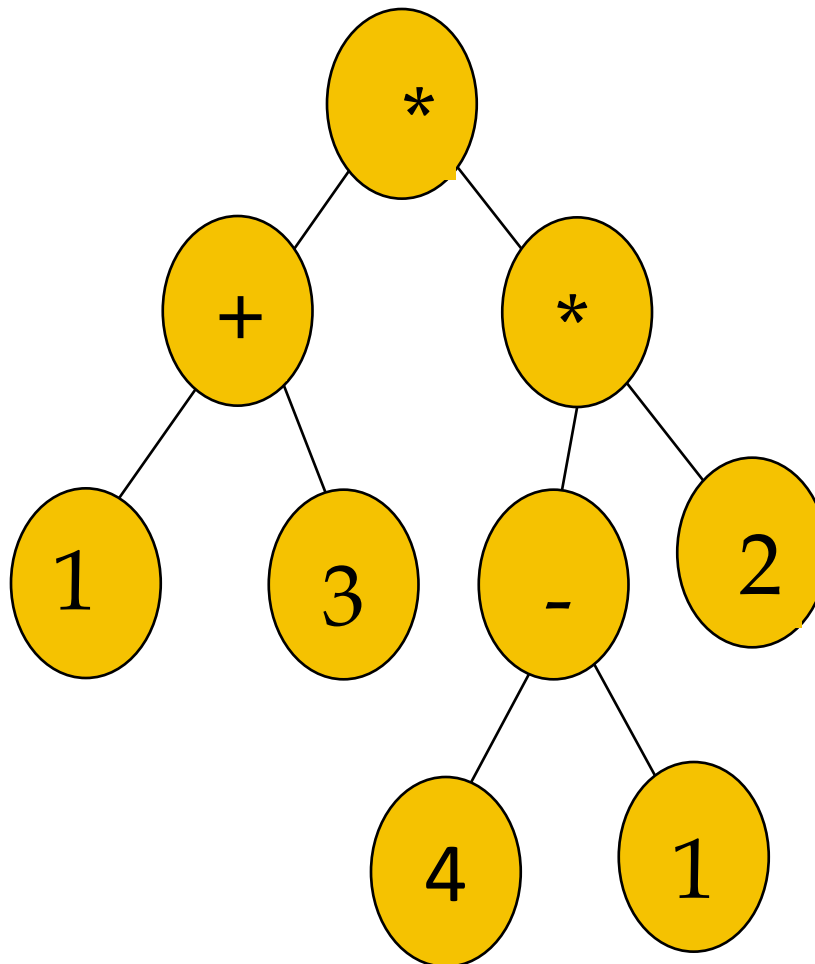
Excercise (Try at home)

A) preorder : * + 1 3 * - 4 1 2

B) postorder : 1 3 + 4 1 - 2 * *

Traversing an expression tree

Excercise (Try at home :answer)



preorder (1st touch)
* + 1 3 * - 4 1 2

postorder (last touch)
1 3 + 4 1 - 2 * *

inorder (2nd touch)
1 + 3 * 4 - 1 * 2