Trees

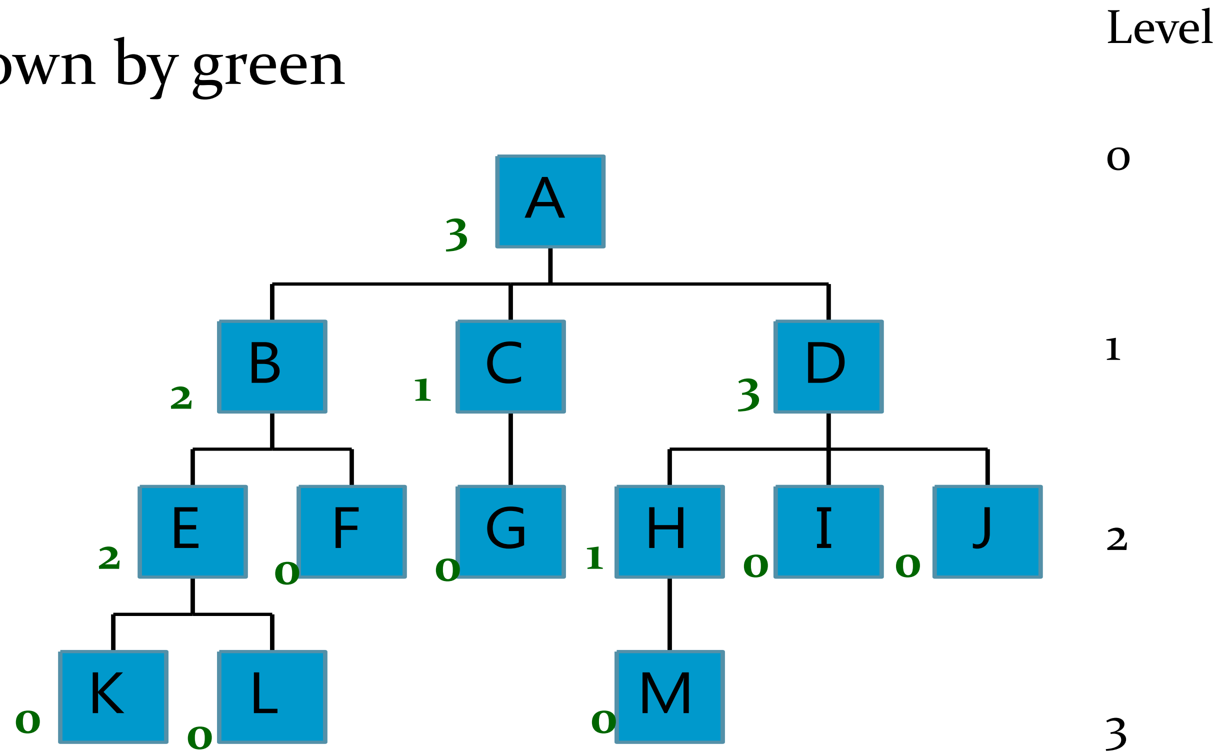
- Binary trees: introduction (complete and extended binary trees),
- Memory representation (sequential, linked)
- Binary tree traversal: pre-order, in-order and post-order (traversal algorithms using stacks)

Definition of Tree

- A tree is a finite set of zero or more nodes such that:
- There is a specially designated node called the root.
- The remaining nodes are partitioned into n>=0 disjoint sets T1, ..., Tn, where each of these sets is a tree.
- We call T1, ..., Tn the subtrees of the root.

Level and Depth

node (13)
degree of a node(shown by green number)
leaf (terminal)
nonterminal
parent
children
sibling
ancestor
descendant
level of a node
height(depth) of a tree (4)



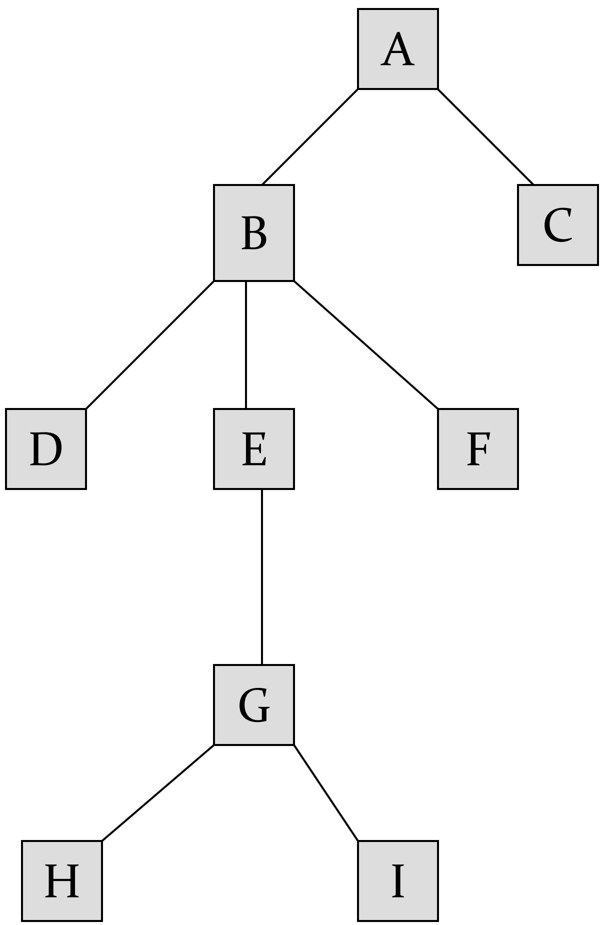
Terminology

- The degree of a node is the number of subtrees of the node
  - The degree of A is 3; the degree of C is 1.
- The node with degree 0 is a leaf or terminal node.
- A node that has subtrees is the *parent* of the roots of the subtrees.
- The roots of these subtrees are the *children* of the node.
- Children of the same parent are *siblings*.
- The ancestors of a node are all the nodes along the path from the root to the node.

Terminology

- The depth of a node is the number of edges from the node to the tree's root node. A root node will have a depth of 0.
- The height of a node is the number of edges on the longest path from node to a leaf node. A leaf node will have a height of 0.

Tree Properties



Property	Value
Number of nodes	9
Height	5
Root Node	A
Leaves	5
Interior nodes	A,B,E,G
Number of levels	4
Ancestors of H	G,E,B,A
Descendants of B	D,E,F,G,H,I
Siblings of E	D,F
degree of node A	2

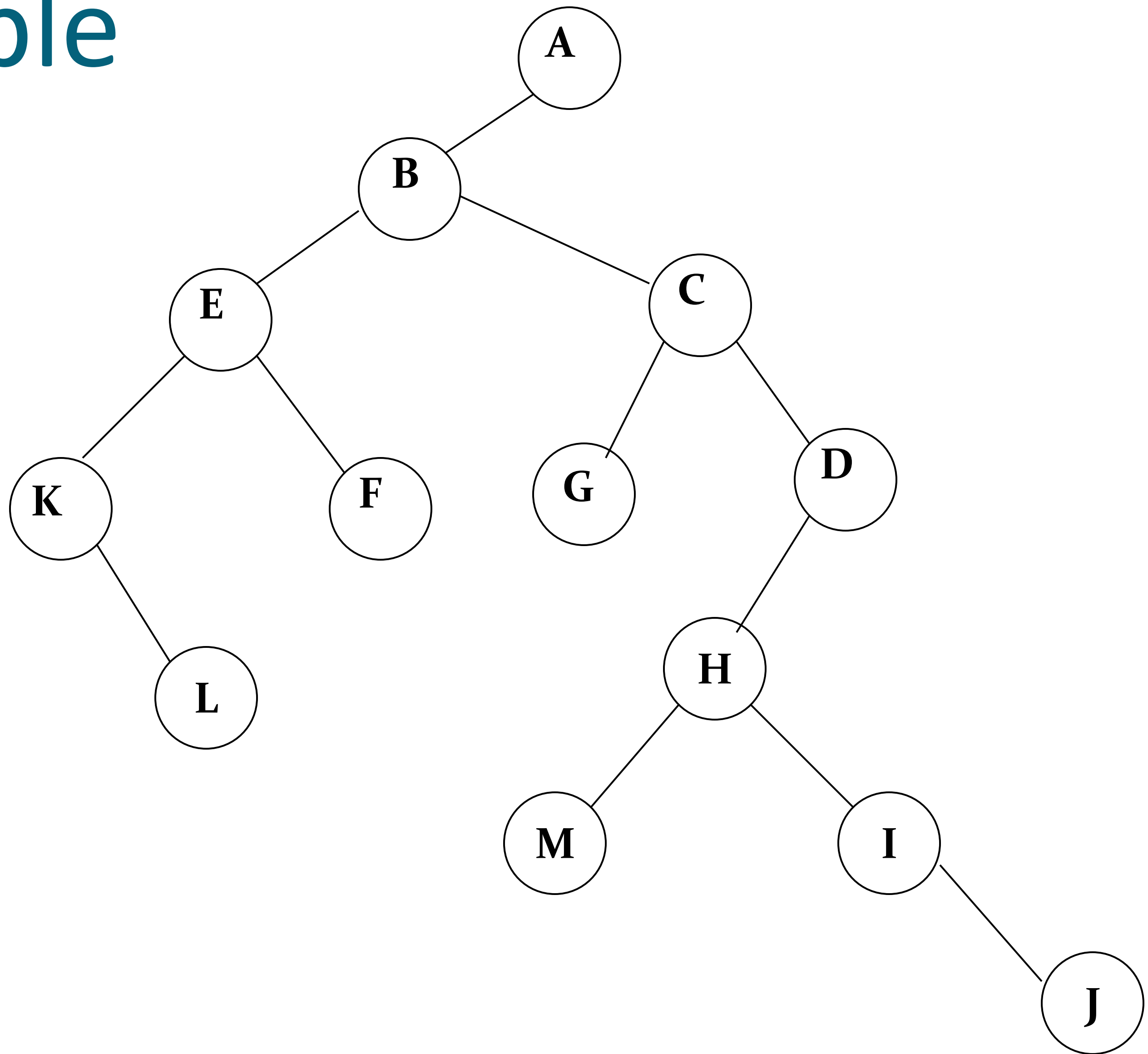
Height of A: 4 and depth of A: 0

CODERINDEED

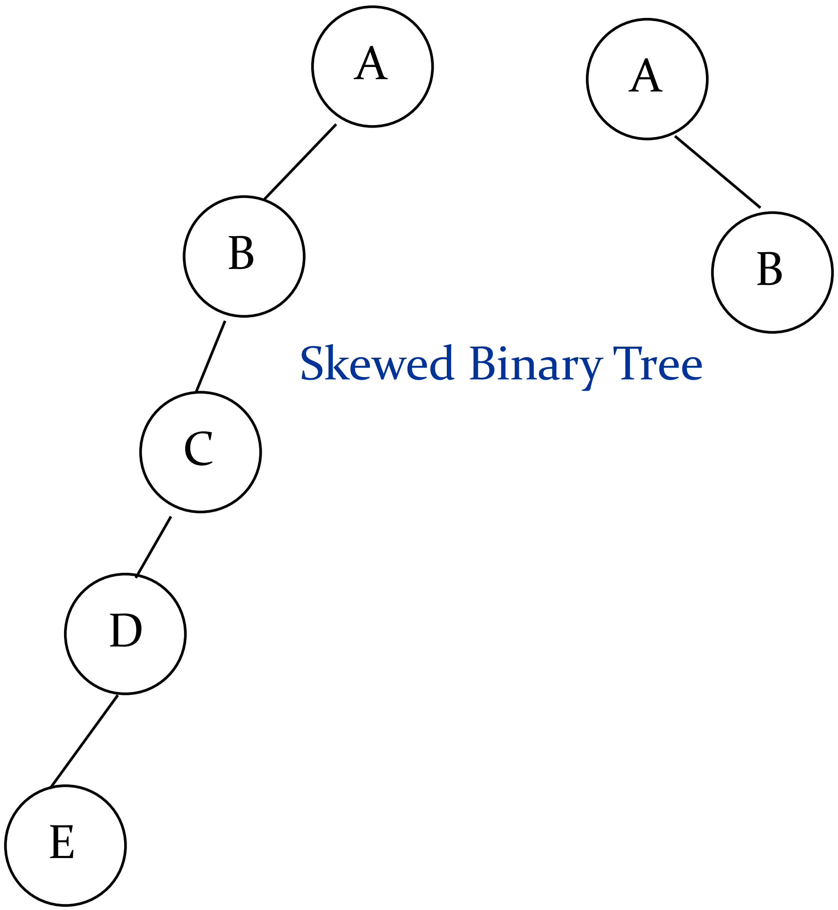
Binary Trees

- A special class of trees: max degree for each node is 2
- Recursive definition: A binary tree is a finite set of nodes that is either empty or consists of a root and two disjoint binary trees called *the left subtree* and *the right subtree*.

Example

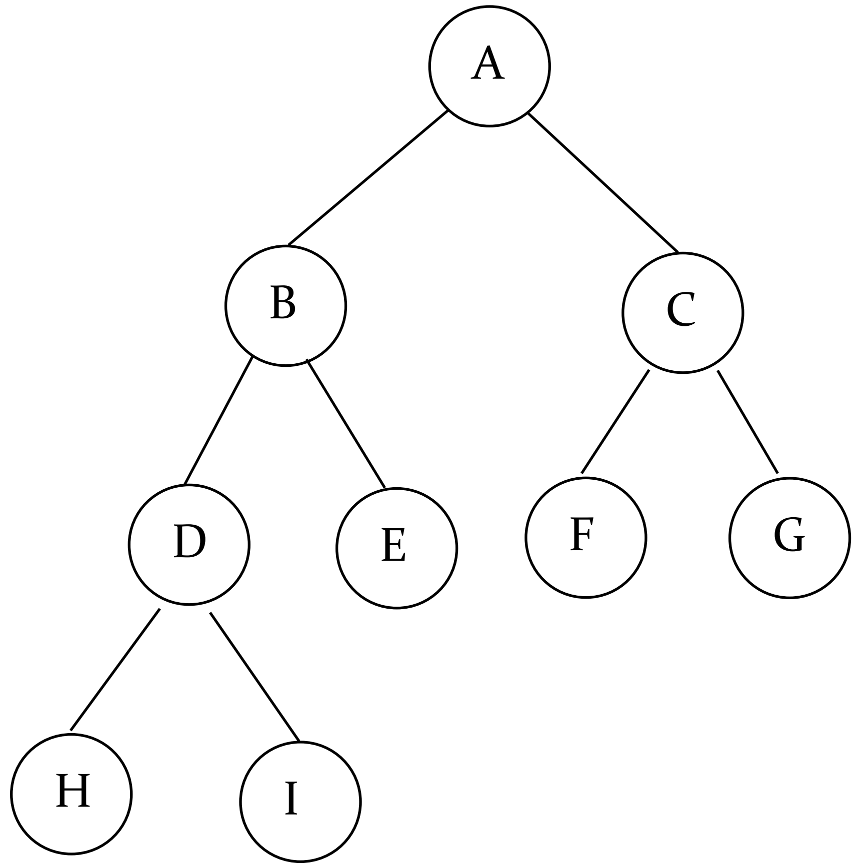


Samples of Trees



Skewed Binary Tree

Complete Binary Tree



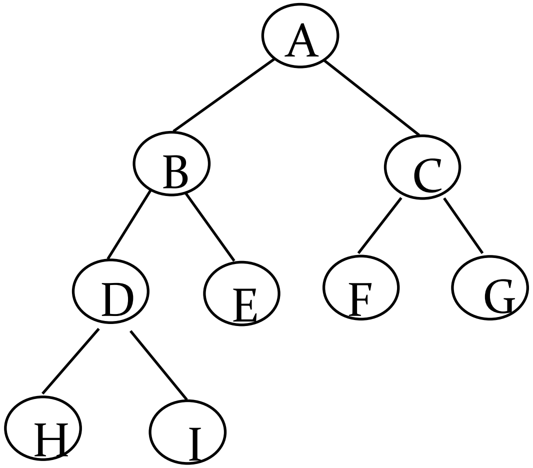
Maximum Number of Nodes in BT

- The maximum number of nodes on level i of a binary tree is 2^i, $i>=0$.

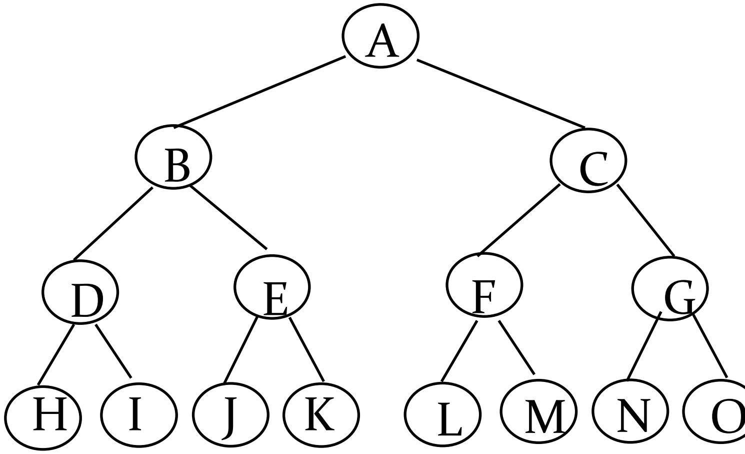
- The maximum number of nodes in a binary tree
  of depth k is 2^k-1, $k>=1$.

Full BT vs. Complete BT

- A full binary tree of depth k is a binary tree of depth k having 2^k-1 nodes, $k>=1$.
- A binary tree with n nodes and depth k is complete *iff* its nodes correspond to the nodes numbered from 1 to n in the full binary tree of depth k.



Complete binary tree



Full binary tree of depth 4

CODERINDEED

Complete Binary Tree

- If a complete binary tree with n nodes (depth= $\lfloor \log n + 1 \rfloor$)

is represented sequentially,

then for any node with index i, $1 <= i <= n$, we have:

  - *parent*(i) is at $\lfloor i/2 \rfloor$ if $i != 1$. If $i=1$, i is at the root and has no parent.
  - *leftChild*(i) is at $2i$ if $2i <= n$. If $2i > n$, then i has no left child.
  - *rightChild*(i) is at $2i+1$ if $2i+1 <= n$. If $2i+1 > n$, then i has no right child.

Extended Binary Tree

- A binary tree T is said to be a *2-tree* or an *extended binary* tree if each node N has either 0 or 2 children.
- The nodes with 2 children are called internal nodes.
- The nodes with 0 children are called external nodes.

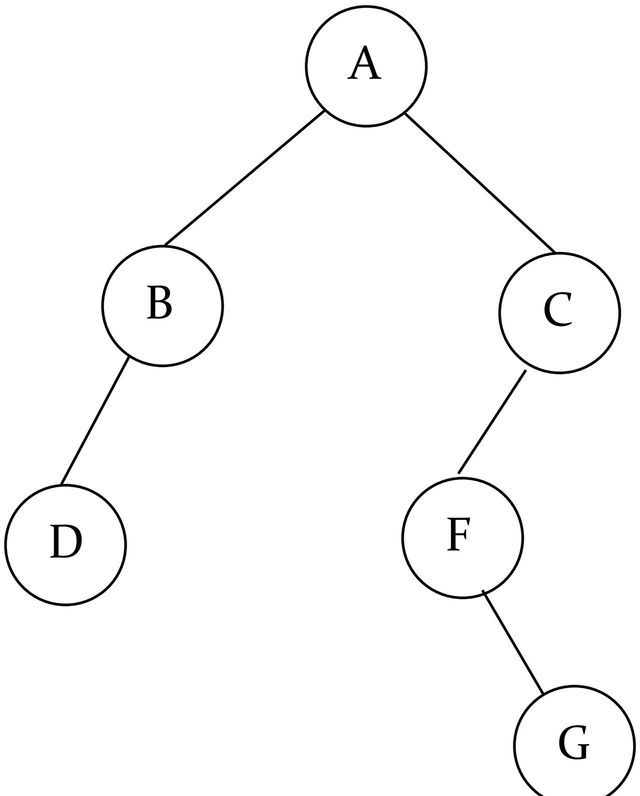


Fig: Binary Tree T

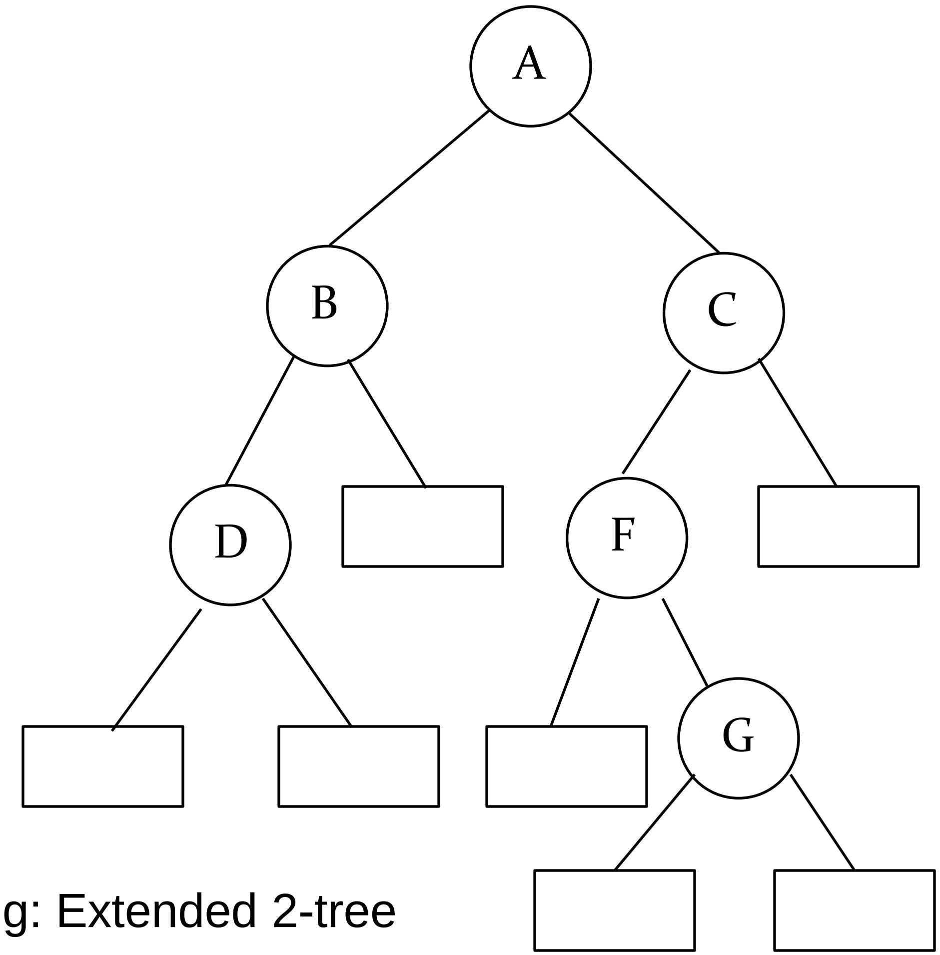
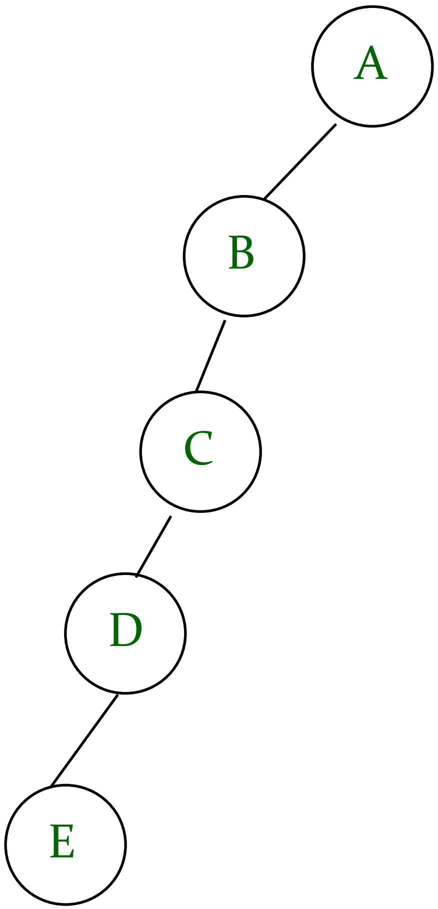


Fig: Extended 2-tree

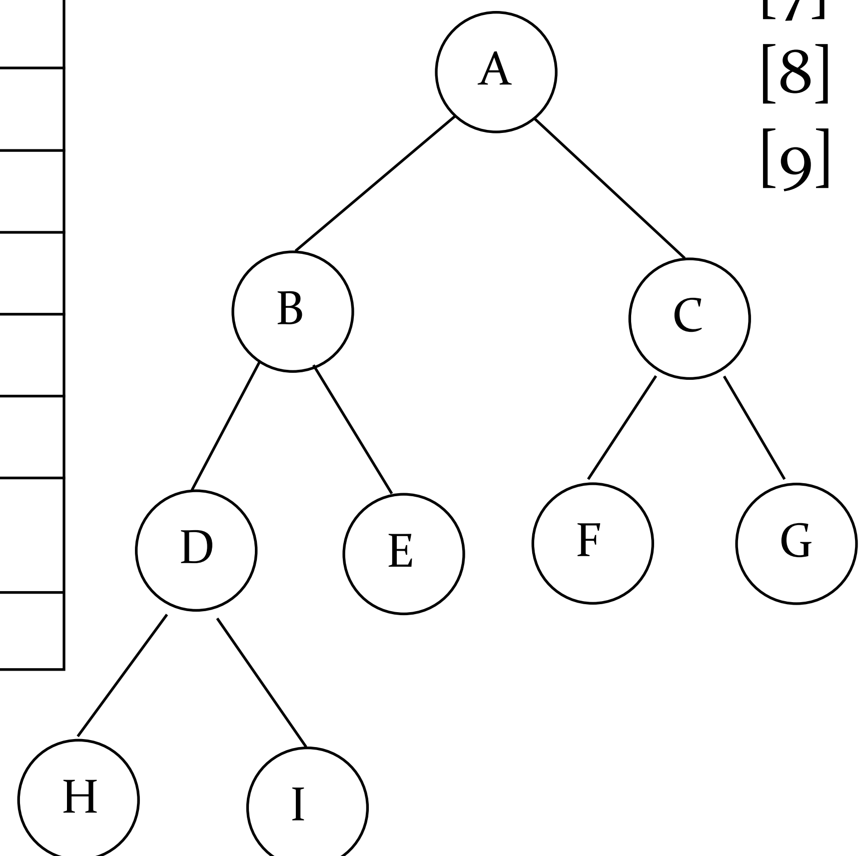
CODERINDEED

Sequential Representation



[1]	A
[2]	B
[3]	--
[4]	C
[5]	--
[6]	--
[7]	--
[8]	D
[9]	--
.	.
[16]	E

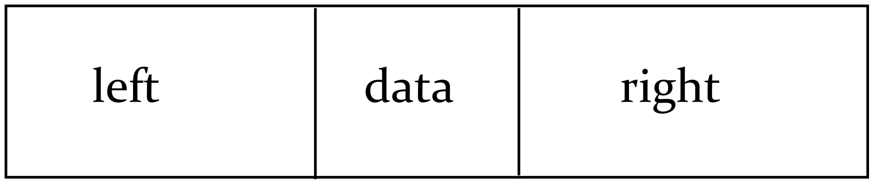
(1) waste space
(2) insertion/deletion problem

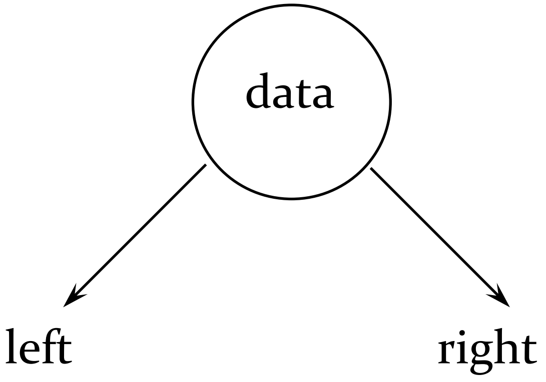


[1]	A
[2]	B
[3]	C
[4]	D
[5]	E
[6]	F
[7]	G
[8]	H
[9]	I

Linked Representation

```
struct btnode {
 int data;
 btnode *left, *right;
};
```

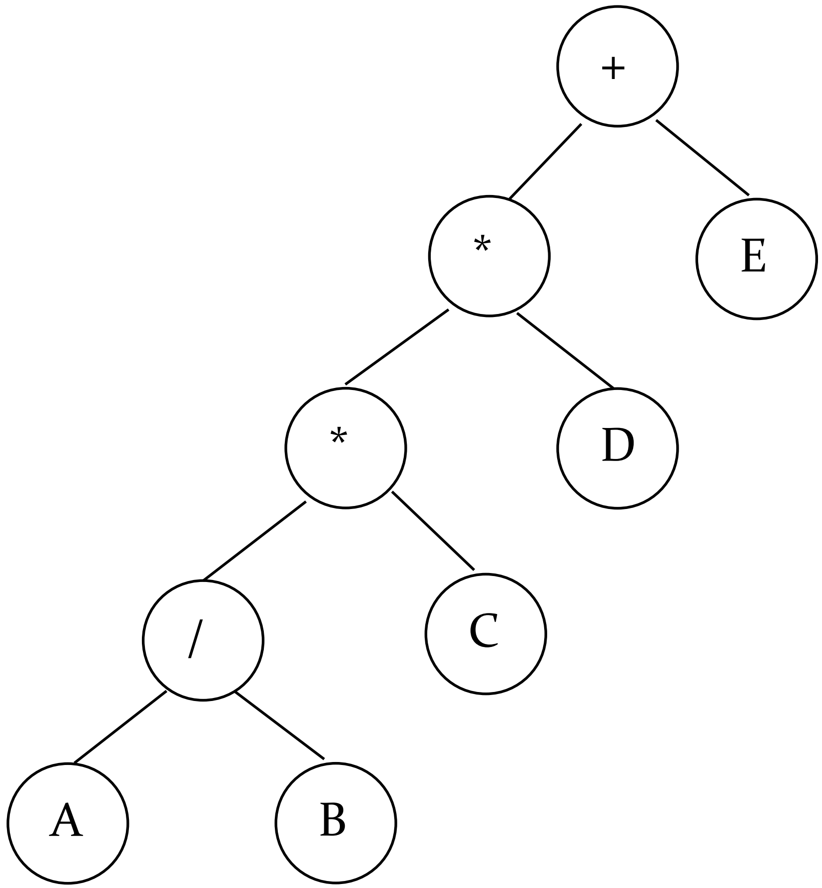




Binary Tree Traversals

- There are three standard ways of traversing a binary tree T with root R.
- These three algorithms, called:
    - preorder
    - inorder
    - postorder

Arithmetic Expression Using BT



preorder traversal

+ * * / A B C D E

prefix expression

inorder traversal

A / B * C * D + E

infix expression

postorder traversal

A B / C * D * E +

postfix expression

Preorder Traversal (recursive version)

```
Algorithm:
1. Process the root R.
2. Traverse the left subtree of R in preorder.
3. Traverse the right subtree of R in preorder.

Pseudo-Code:
void preorder(btnode ptr)
/* preorder tree traversal */
{
    if (ptr!=NULL) {
        cout<<ptr->data;
        preorder(ptr->left);
        predorder(ptr->right);
    }
}
```

+ * * / A B C D E

Inorder Traversal (recursive version)

Algorithm:

1. Traverse the left subtree of R in inorder.
2. Process the root R.
3. Traverse the right subtree of R in inorder.

A / B * C * D + E

Pseudo-Code:

```
void inorder(btnode ptr)
/* inorder tree traversal */
{
    if (ptr!=NULL) {
        inorder(ptr->left);
        cout<<ptr->data;
        indorder(ptr->right);
    }
}
```

Postorder Traversal (recursive version)

Algorithm:

1. Traverse the left subtree of R in postorder.
2. Traverse the right subtree of R in postorder.
3. Process the root R.

A B / C * D * E +

Pseudo-Code:

```
void postorder(btnode ptr)
/* postorder tree traversal */
{
     if (ptr!=NULL) {
        postorder(ptr->left);
        postorder(ptr->right);
        cout<<ptr->data;
     }
}
```

Thank You