01418231 Data Structures

Tree

https://blog.trustedsite.com/2017/05/19/4-things-need-know-google-search-rankings-2017/

Agenda

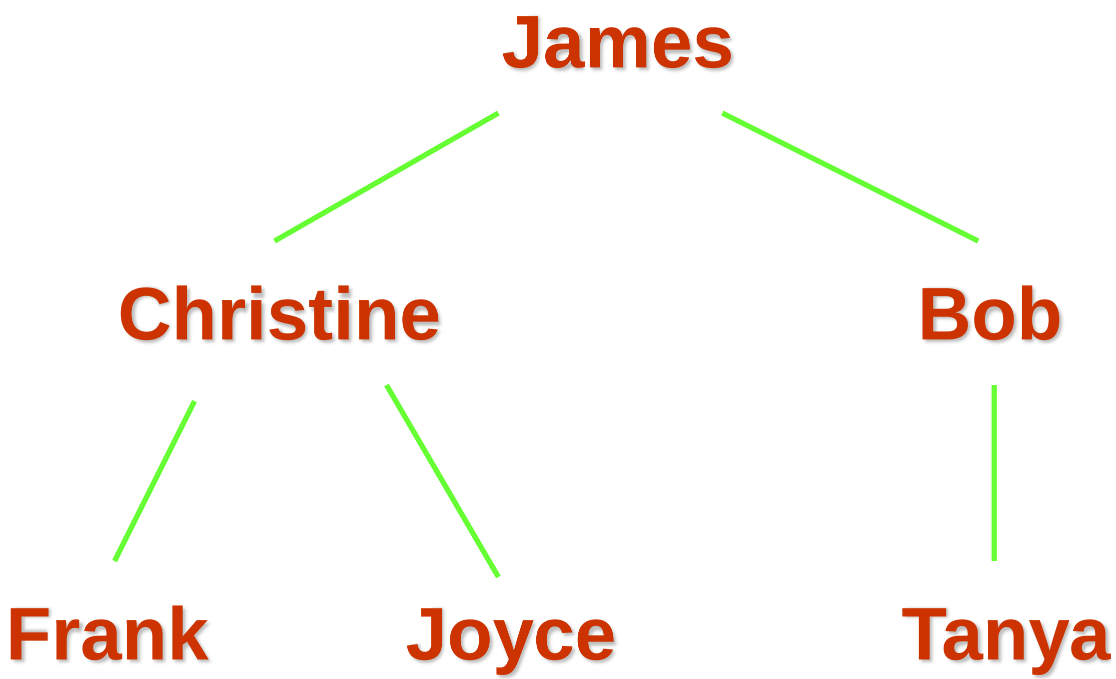
- Introduction to Tree
- What is Binary search tree (BST)
- Operations
  - Search, Insert, Delete
  - Findmin,Findmax
- Summary

Trees in our life

- Tree is an important data structure that represent a hierarchy
- Trees/hierarchies are very common in our life:
  - Family tree (parent-child)
  - Component tree (part-of)

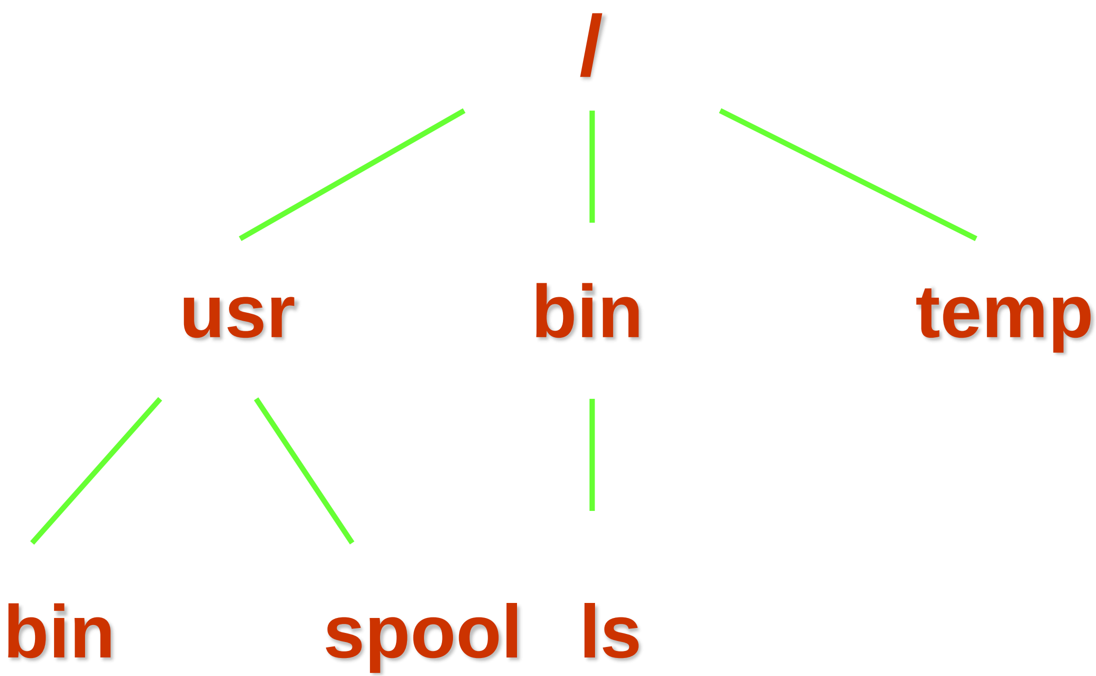
Trees

- Example : Family tree



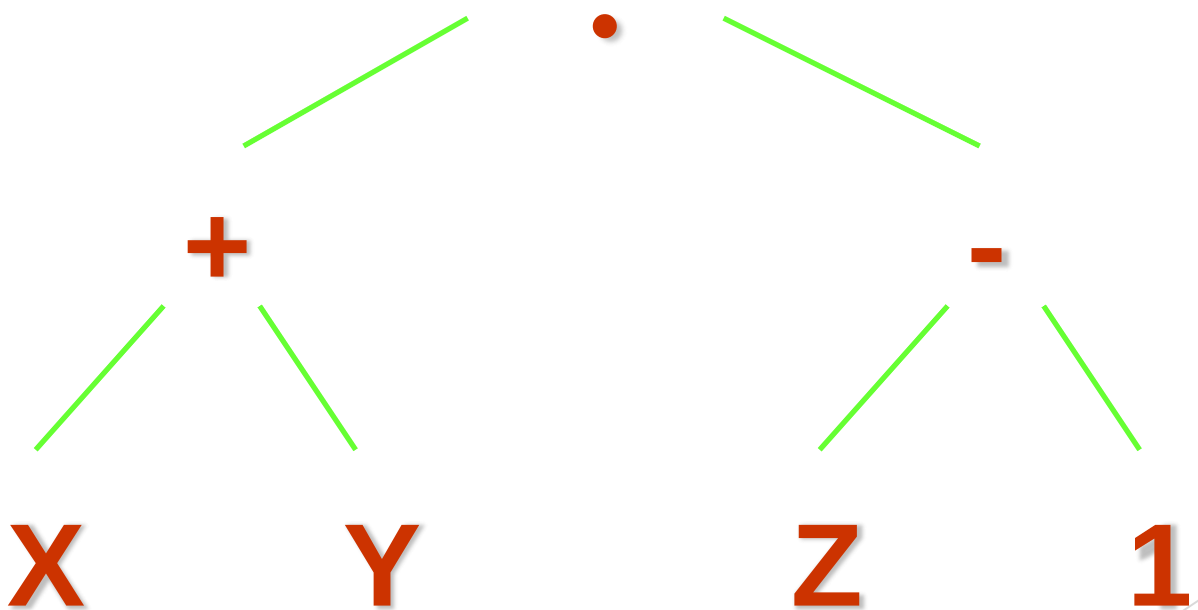
Trees

- Example : File system



Trees

- Example : Arithmetic expressions



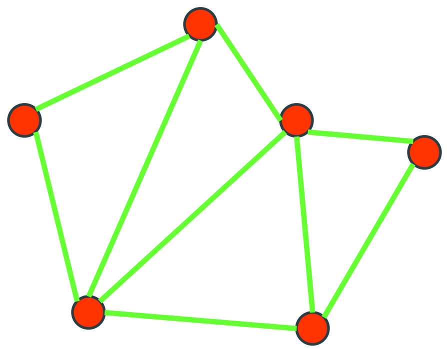
- **This tree represents the expression**
  - $(X + Y) \cdot (Z - 1)$

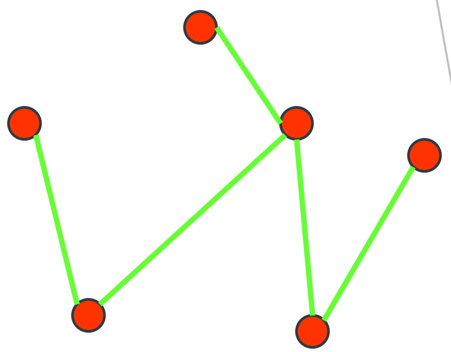
Trees

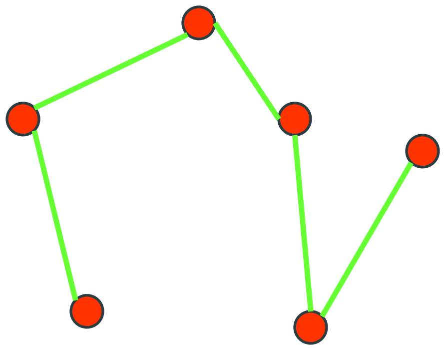
- Definition:
  - A tree is a connected undirected graph with no simple circuits
  - cannot have a simple circuit,
  - tree cannot contain multiple edges or loops
- Therefore, any tree must be a simple graph

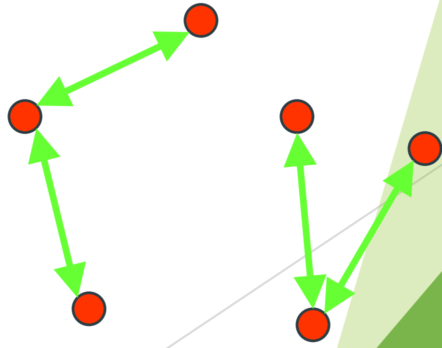
Trees

- Example: Are the following graphs trees?

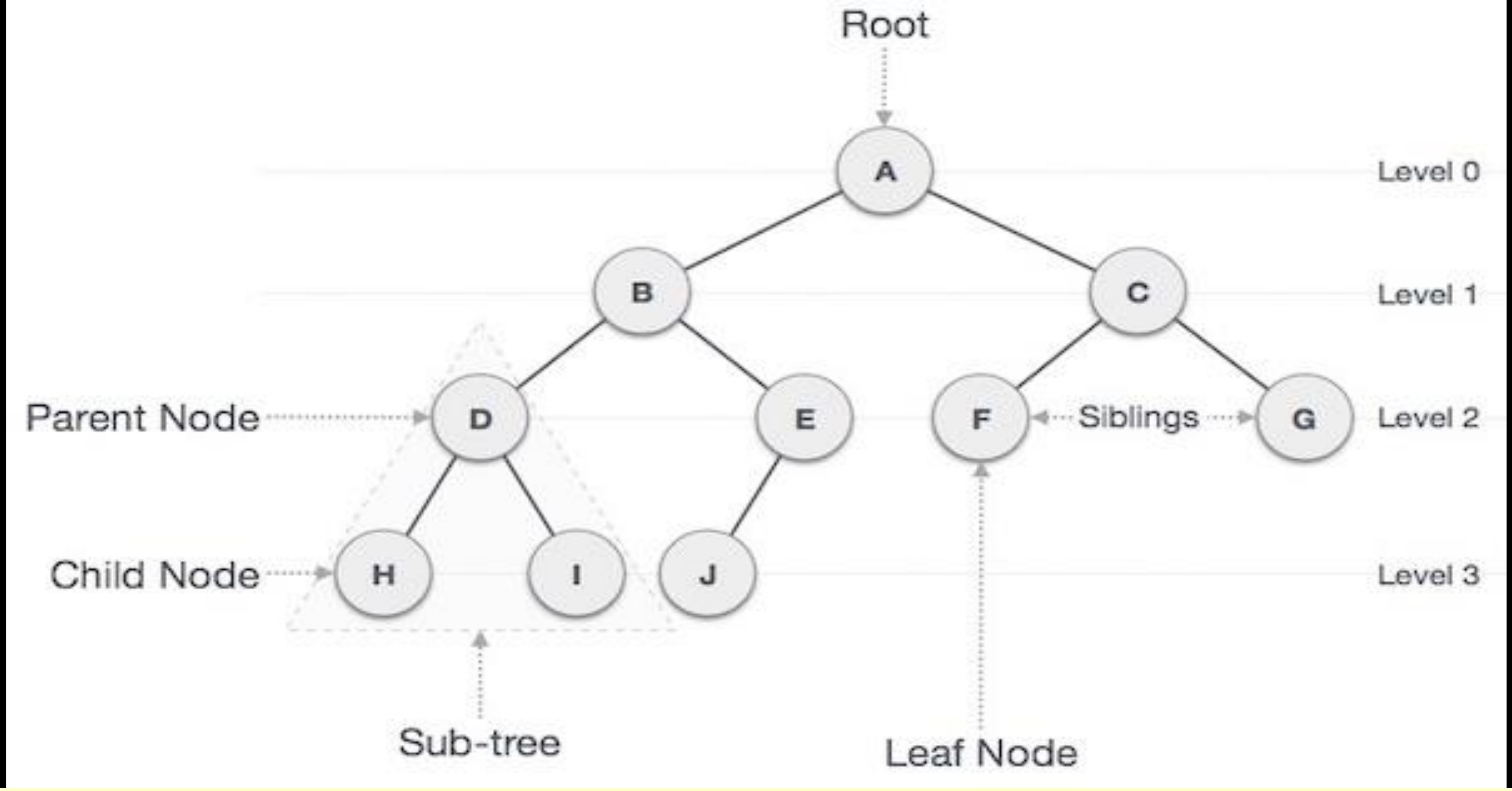




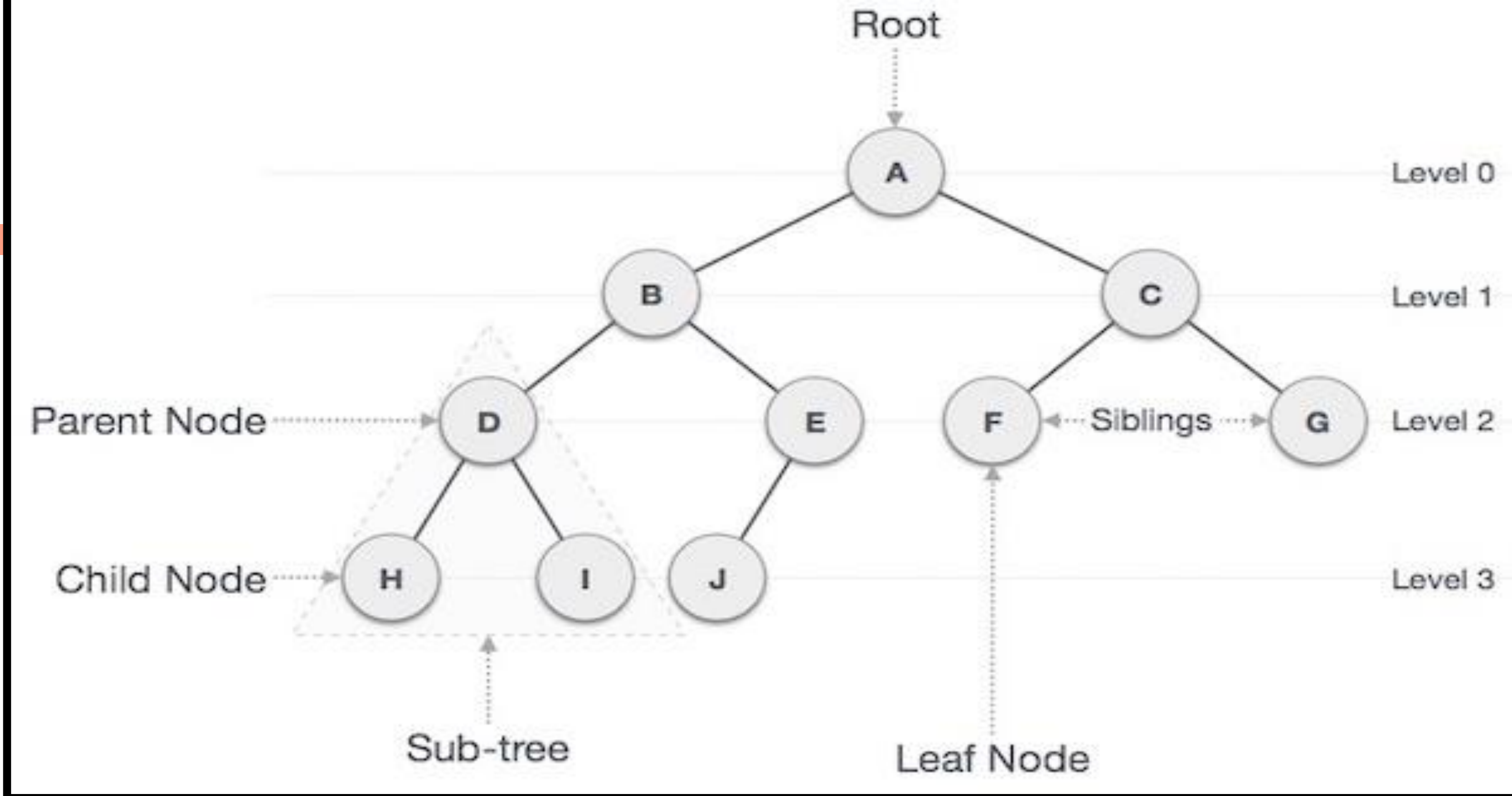




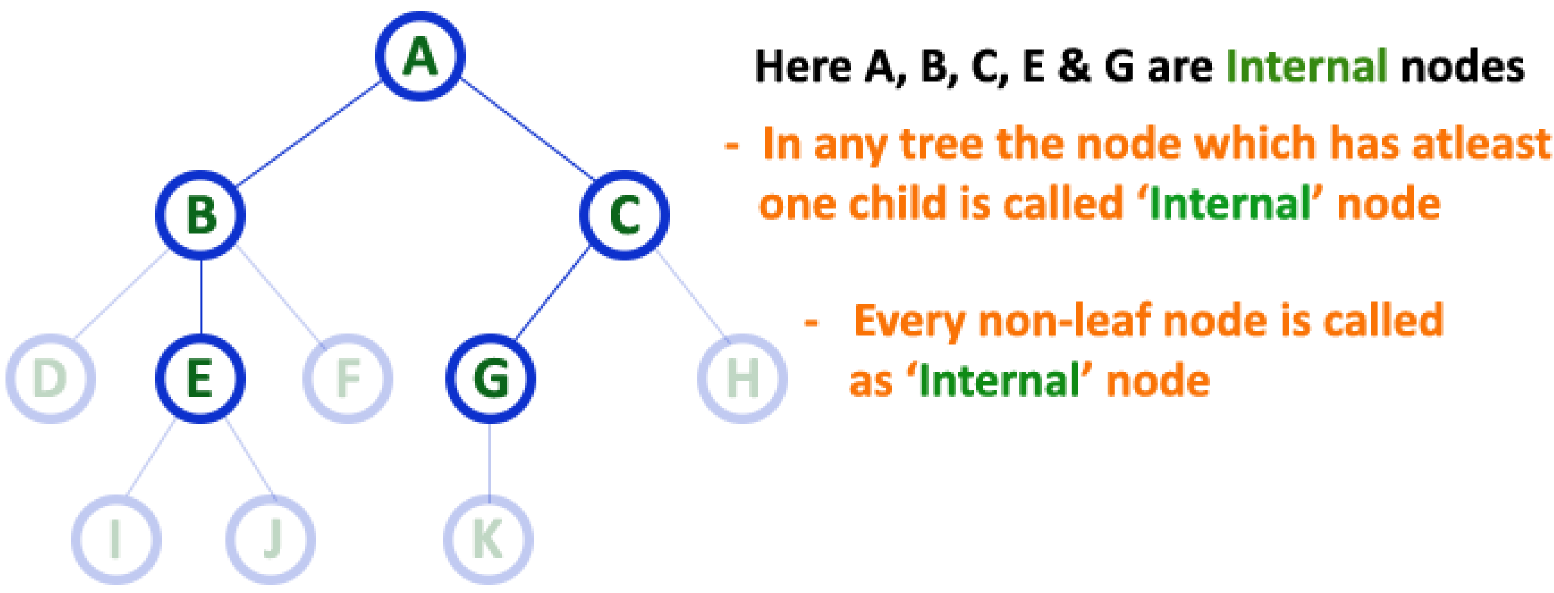
Terminology



- __________– The node at the top of the tree is called root. There is only one root per tree
- __________ – Any node except the root node has one edge upward to a node called parent.
- __________ – The node below a given node connected by its edge downward is called its child node.
- __________ – The node which does not have any child node is called the leaf node.
- __________ – The nodes which belong to same Parent are called as SIBLINGS.

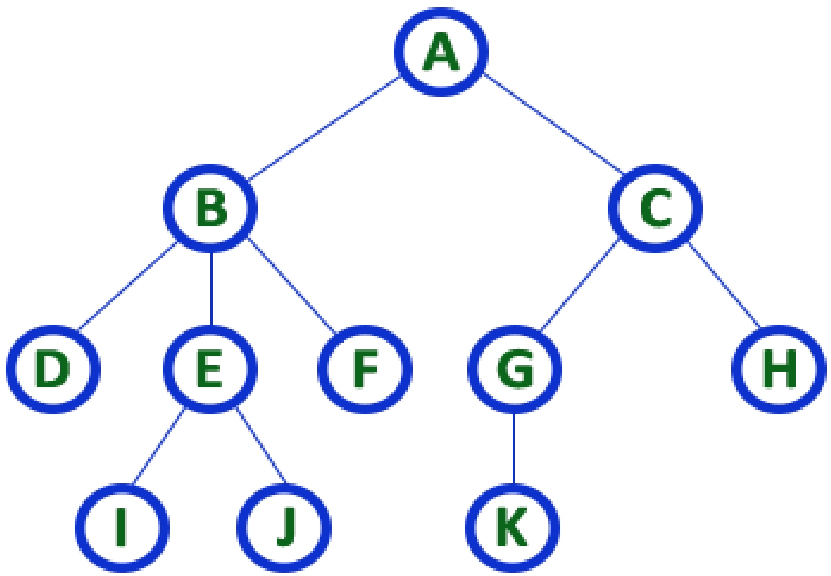


- ________ – Path refers to the sequence of nodes along the edges of a tree.
- ________ – Subtree represents the descendants of a node.
- ________ – Level of a node represents the generation of a node. If the root node is at level 0, then its next child node is at level 1, its grandchild is at level 2, and so on.



- ________ -The node which has at least one child , Nodes other than leaf nodes are called as Internal Nodes.
- The root node is also said to be Internal Node if the tree has more than one node.
- Internal nodes are also called as 'Non-Terminal' nodes.

http://btechsmartclass.com/DS/U3_T1.html



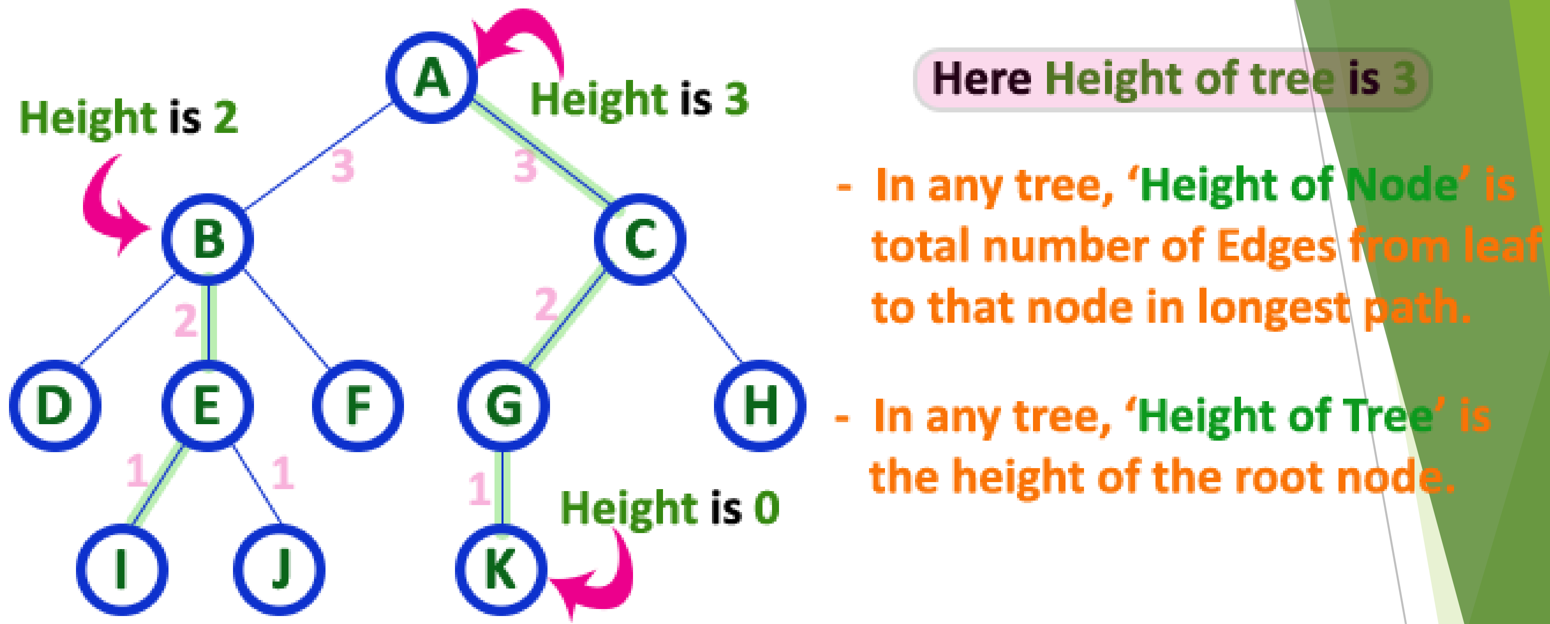
Here Degree of B is 3

Here Degree of A is 2

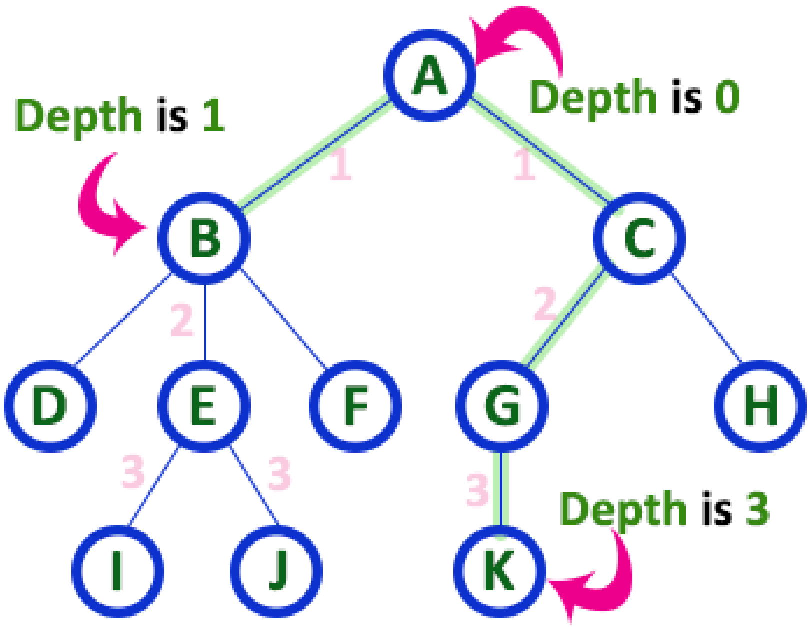
Here Degree of F is 0

- In any tree, 'Degree' a node is total number of children it has.

- ________ **-** the total number of children of a node is called as **DEGREE** of that Node.
- The highest degree of a node among all the nodes in a tree is called as '**Degree of Tree**'



- ________ the total number of egdes from leaf node to a particular node in the longest path is called as **HEIGHT** of that Node.
- The height of the root node is said to be height of the tree. And height of all leaf nodes is '0'.



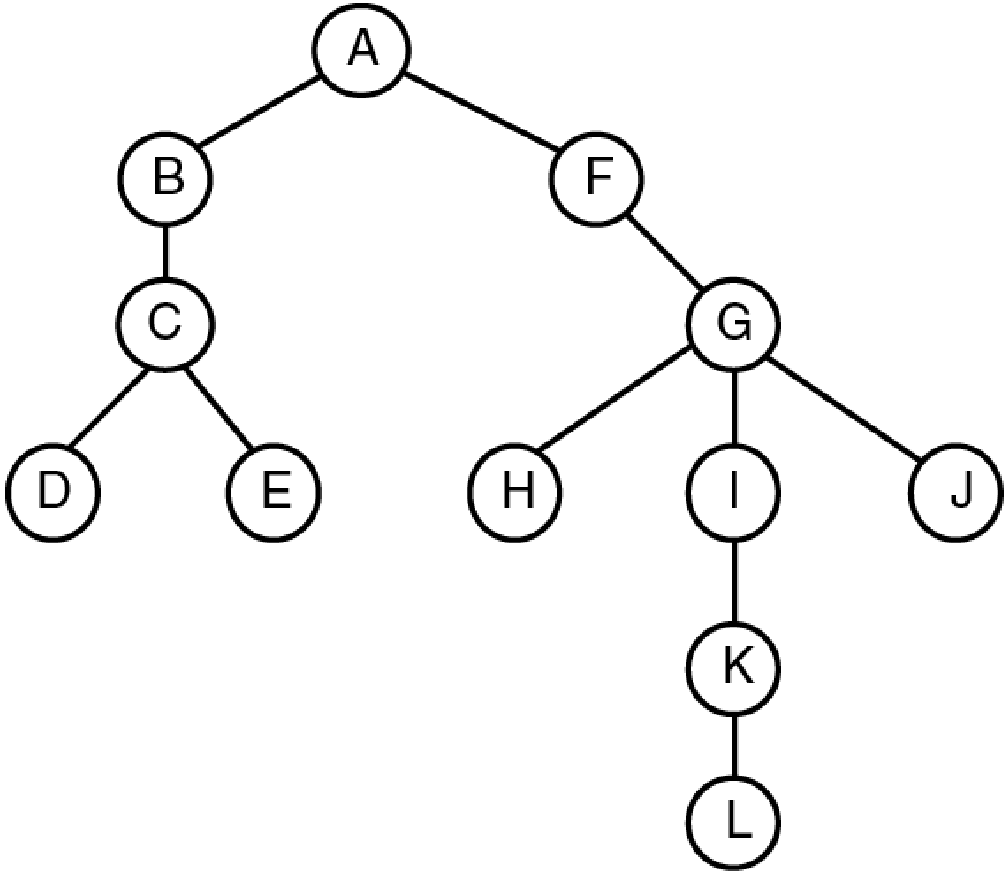
Here Depth of tree is 3

- In any tree, 'Depth of Node' is total number of Edges from root to that node.
- In any tree, 'Depth of Tree' is total number of edges from root to leaf in the longest path.

- ________ -the total number of egdes from root node to a particular node is called as **DEPTH** of that Node.
- The total number of edges from root node to a leaf node in the longest path is said to be Depth of the tree.
- The depth of the root node is **'0'.**

Quiz

Quiz



- Root
- Leaf
- Parent
- Children
- Sibling
- Internal node
- Level
- Degree

Types of Tree

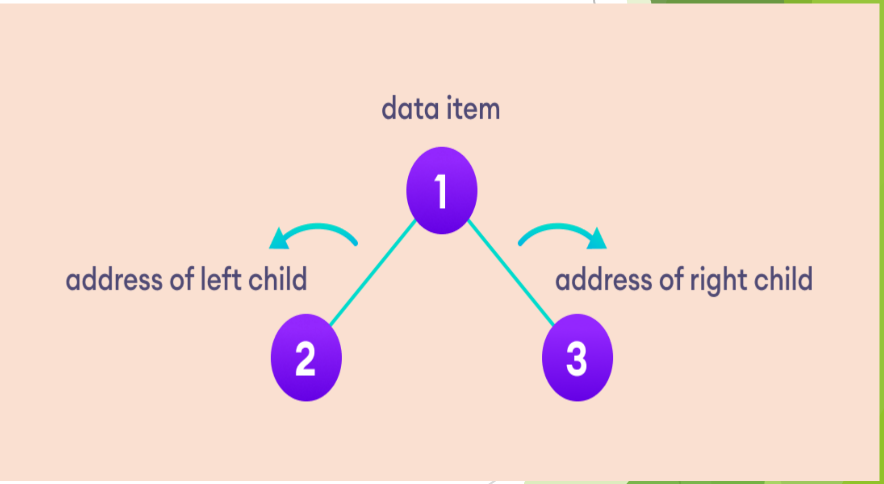
https://www.programiz.com/dsa/trees

Types of Tree

- Binary Tree
- Binary Search Tree
- AVL Tree
- B-Tree

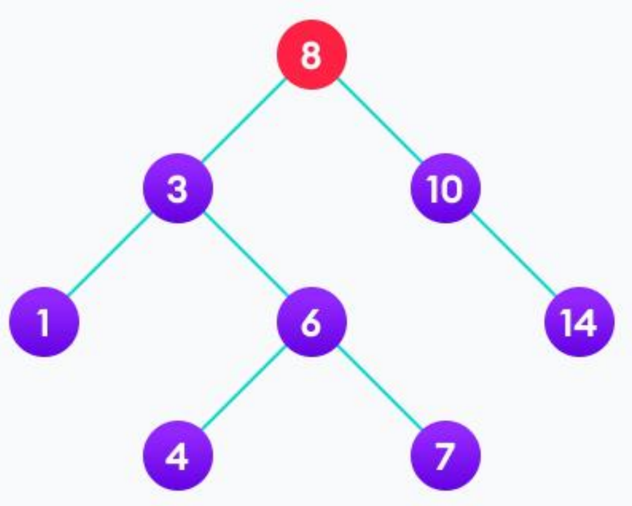
Tree

- A binary tree is a tree data structure in which each parent node can have at most two children.
- Each node of a binary tree consists of three items:



- data item
- address of left child
- address of right child

Tree

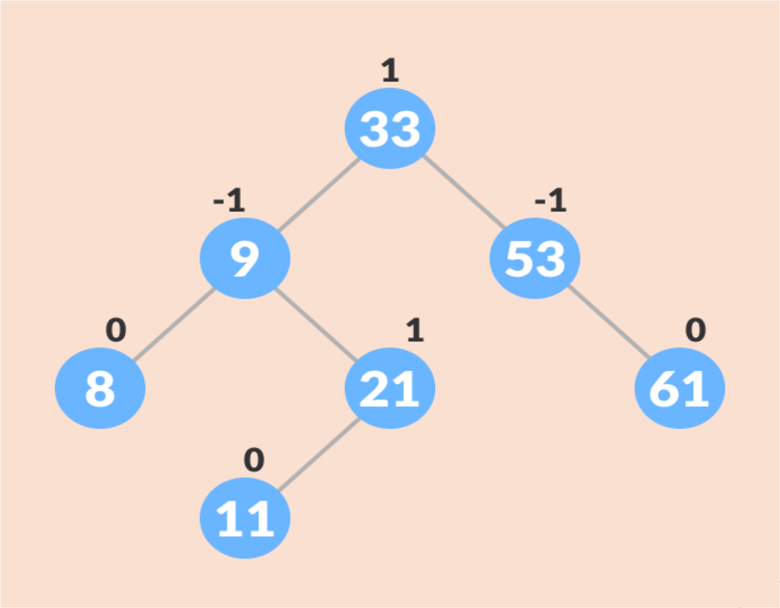


- Binary search tree is a data structure that quickly allows us to maintain a sorted list of numbers.

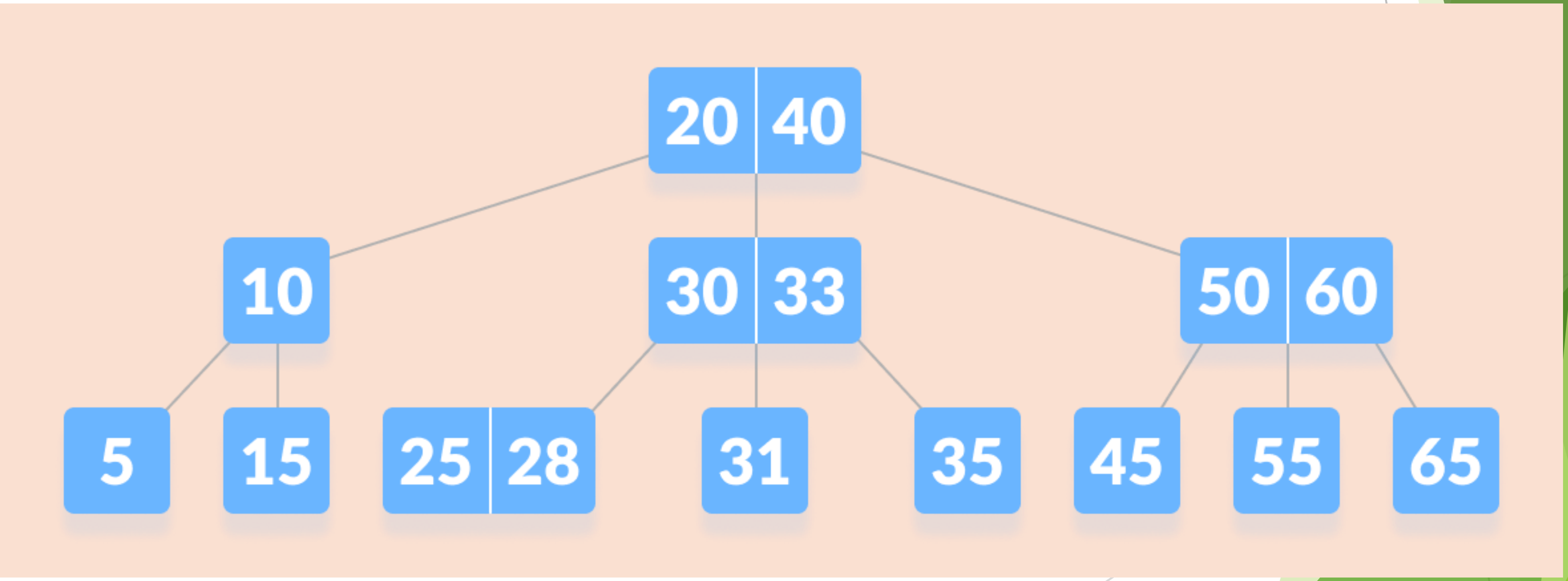
- The properties that separate a binary search tree from a regular binary tree is
  - All nodes of left subtree are less than the root node
  - All nodes of right subtree are more than the root node
  - Both subtrees of each node are also BSTs i.e. they have the above two properties

AVL Tree

- AVL tree is a self-balancing binary search tree in which each node maintains extra information called a balance factor whose value is either -1, 0 or +1.



- B-tree is a special type of self-balancing search tree in which each node can contain more than one key and can have more than two children.
- It is a generalized form of the binary search tree.



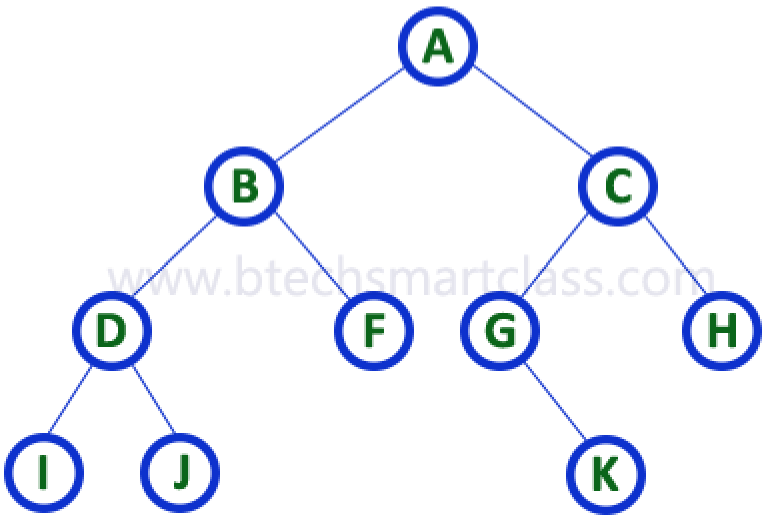
Representation of Trees

- 1. Array Representation
- 2. Linked List Representation

Representation of Trees

1. Array Representation

- In array representation of binary tree, we use a one dimensional array (1-D Array) to represent a binary tree.

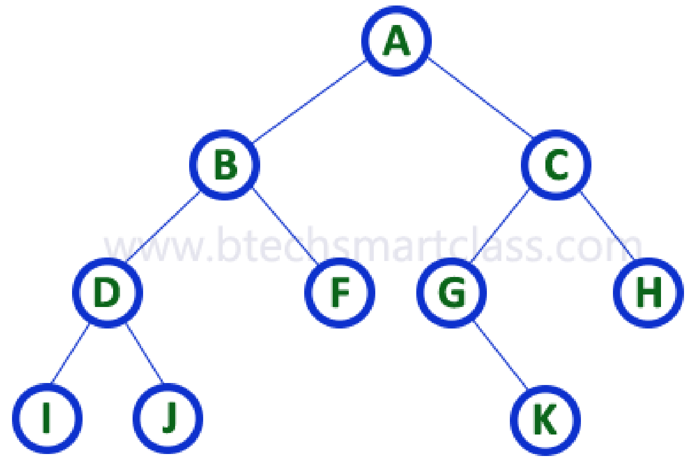


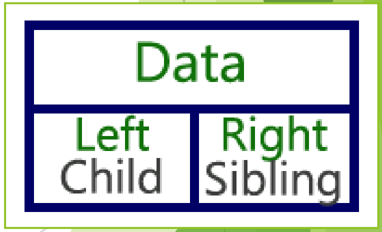
A	B	C	D	F	G	H	I	J	-	-	-	K	-	-	-	-	-	-	-	-

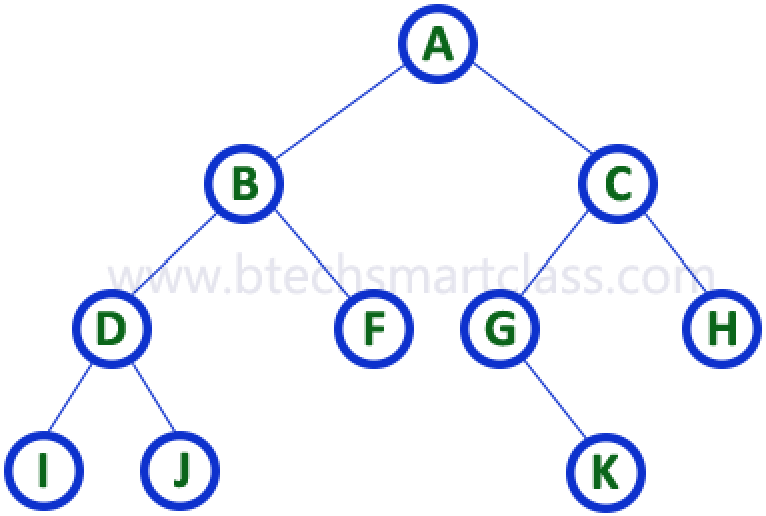
Representation of Trees

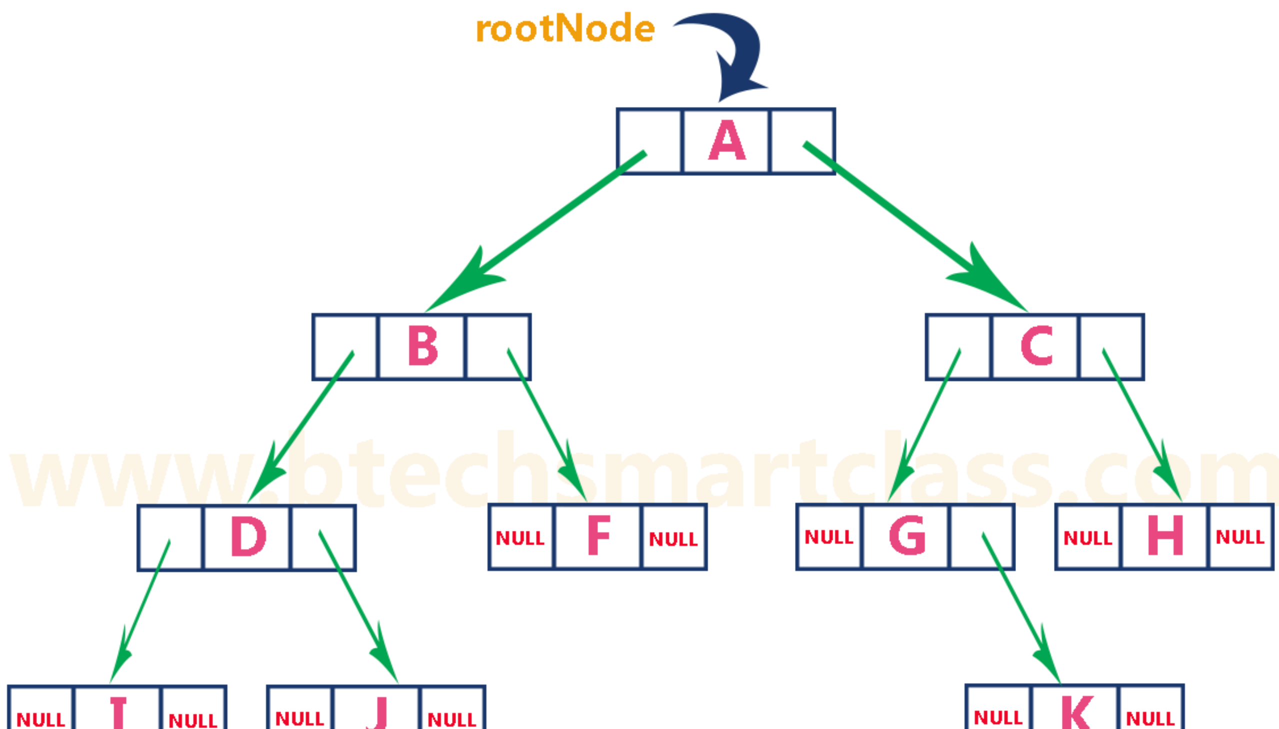
2. Linked List Representation

- Apply double linked list to represent a binary tree. In a double linked list, every node consists of three fields.
- First field for storing left child address
- second for storing actual data
- third for storing right child address.





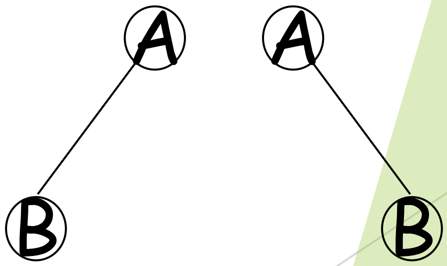

A
B
C
D
F
G
H
I
J
K
www.btechsmartclass.com


rootNode
A
B
C
D
NULL F NULL
NULL G
NULL H NULL
NULL I NULL
NULL J NULL
NULL K NULL
www.btechsmartclass.com

Binary trees

Binary Trees (1/9)

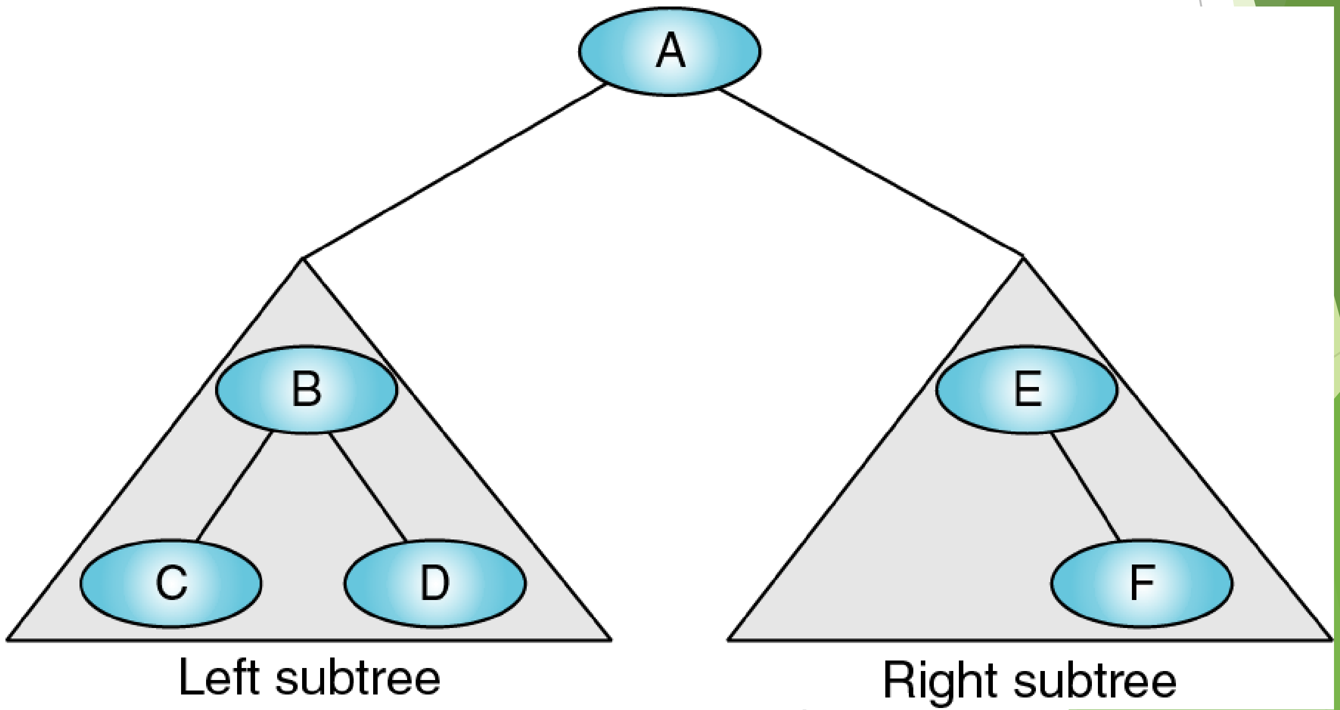
- Binary trees are characterized by the fact that any node can have at most two branches
- Definition (recursive):
  - A *binary tree* is a finite set of nodes that is either empty or consists of a root and two disjoint binary trees called the left subtree and the right subtree
- Thus the left subtree and the right subtree are distinguished



- Any tree can be transformed into binary tree
  - by left child-right sibling representation

Binary trees

- Empty or a root with <= 2 subtrees



Binary Trees (2/9)

- The abstract data type of binary tree

structure *Binary_Tree* (abbreviated *BinTree*) is

objects: a finite set of nodes either empty or consisting of a root node, left *Binary_Tree*, and right *Binary_Tree*.

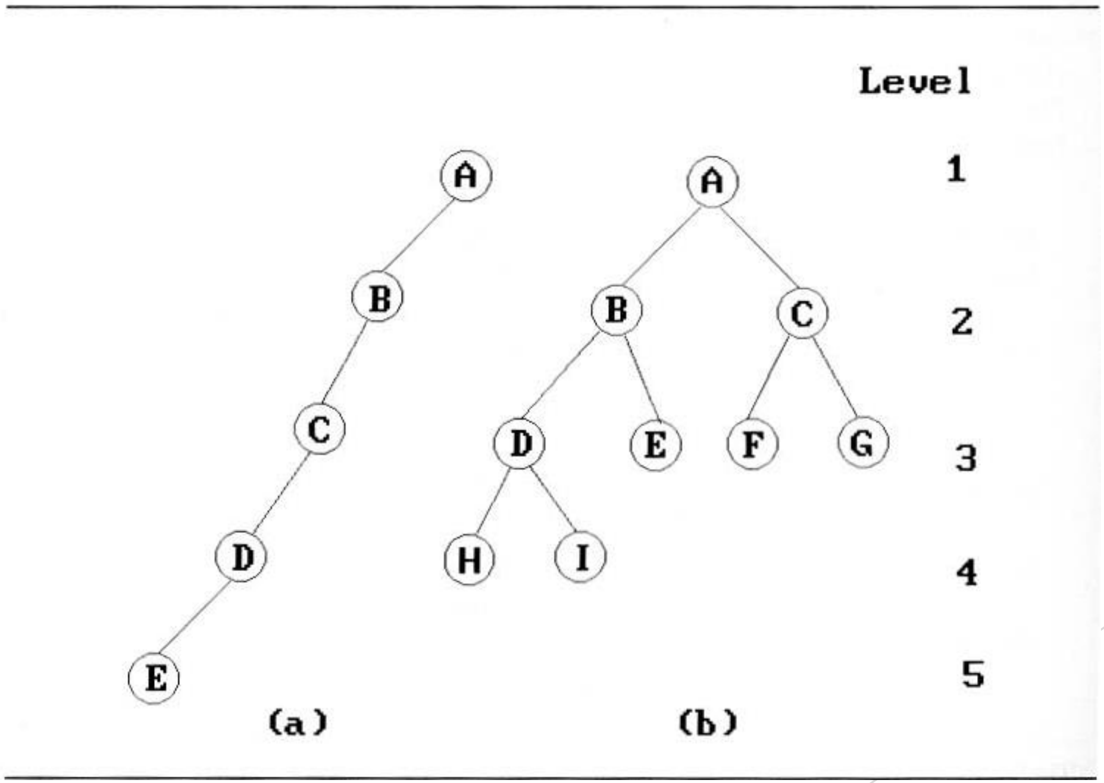
functions:

for all $bt, bt1, bt2 \in BinTree$, $item \in element$

BinTree Create()	::=	creates an empty binary tree
Boolean IsEmpty(*bt*)	::=	**if** (*bt* == empty binary tree) **return** *TRUE* **else return** *FALSE*
BinTree MakeBT(*bt*1, *item*, *bt*2)	::=	**return** a binary tree whose left subtree is *bt*1, whose right subtree is *bt*2, and whose root node contains the data *item*.
BinTree Lchild(*bt*)	::=	**if** (IsEmpty(*bt*)) **return** error **else return** the left subtree of *bt*.
element Data(*bt*)	::=	**if** (IsEmpty(*bt*)) **return** error **else return** the data in the root node of *bt*.
BinTree Rchild(*bt*)	::=	**if** (IsEmpty(*bt*)) **return** error **else return** the right subtree of *bt*.

Binary Trees (3/9)

- Two special kinds of binary trees:
  (a) *skewed tree*, (b) *complete binary tree*
  - The all leaf nodes of these trees are on two adjacent levels



Binary Trees (4/9)

- Properties of binary trees
  - Lemma 1 :[*Maximum number of nodes*]:

1. The maximum number of nodes on level i of a binary tree is 2^i , $i \geq 0$
2. The maximum number of nodes in a binary tree of depth k is 2^k , $k \geq 0$

  - Lemma 2 [*Relation between number of leaf nodes and degree-2 nodes*]:

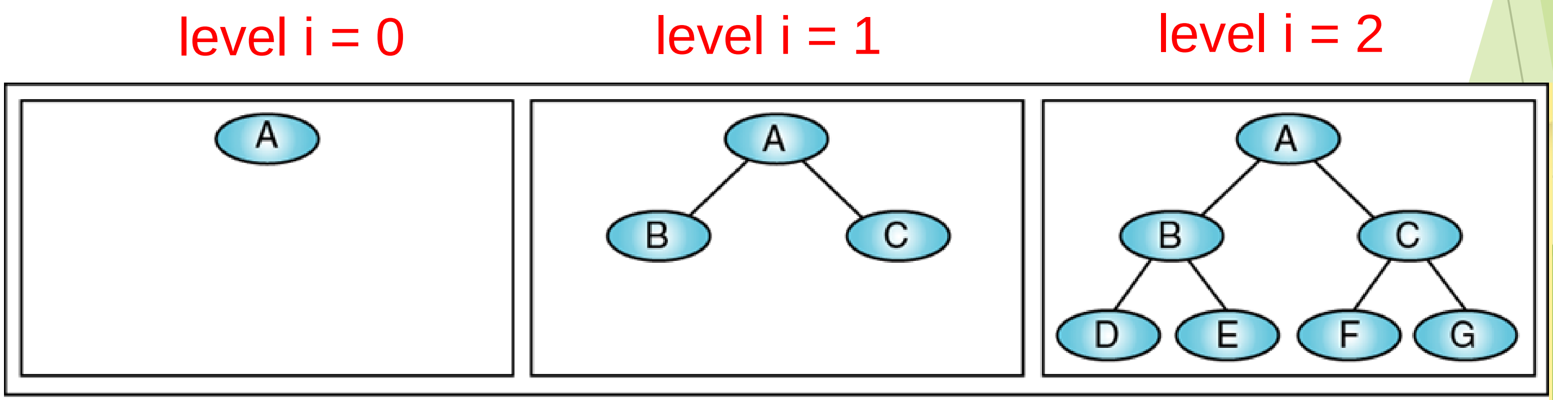
    For any nonempty binary tree, T, if n_0 is the number of leaf nodes and n_2 is the number of nodes of degree 2, then $n_0 = n_2 + 1$.

- These lemmas allow us to define full and complete binary trees

Binary Trees (4/9)

- Properties of binary trees
  - Lemma 1 :[*Maximum number of nodes*]:

1. The maximum number of nodes on level i of a binary tree is 2^i , $i \geq 0$
2. The maximum number of nodes in a binary tree of depth k is 2^k , $k \geq 0$



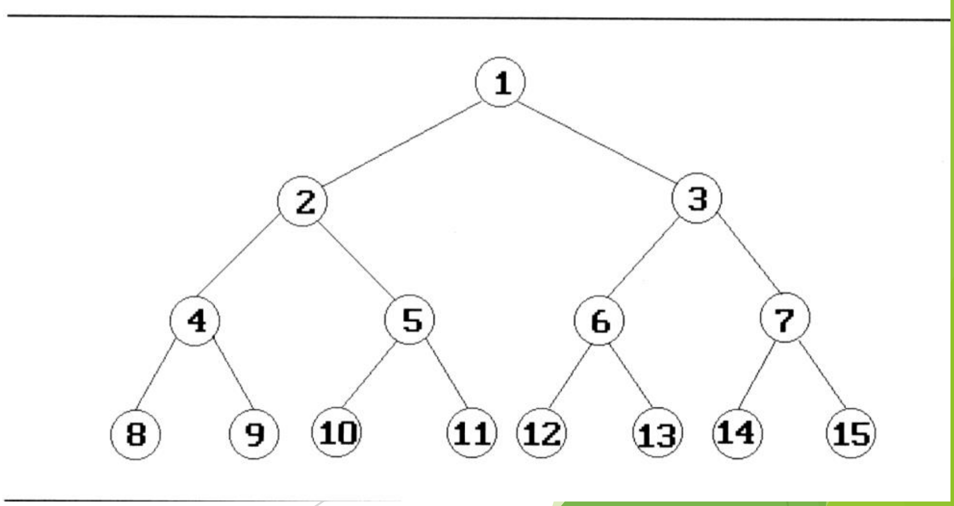
(a) Complete trees (at levels 0, 1, and 2)

Binary Trees (5/9)

- **Properties of binary trees**
  - **Lemma 1 :[*Maximum number of nodes*]:**

1. **The maximum number of nodes on level *i* of a binary tree is** 2^i , $i \geq 0$
2. **The maximum number of nodes in a binary tree of depth *k* is** 2^k , $k \geq 0$

depth $k = 3$



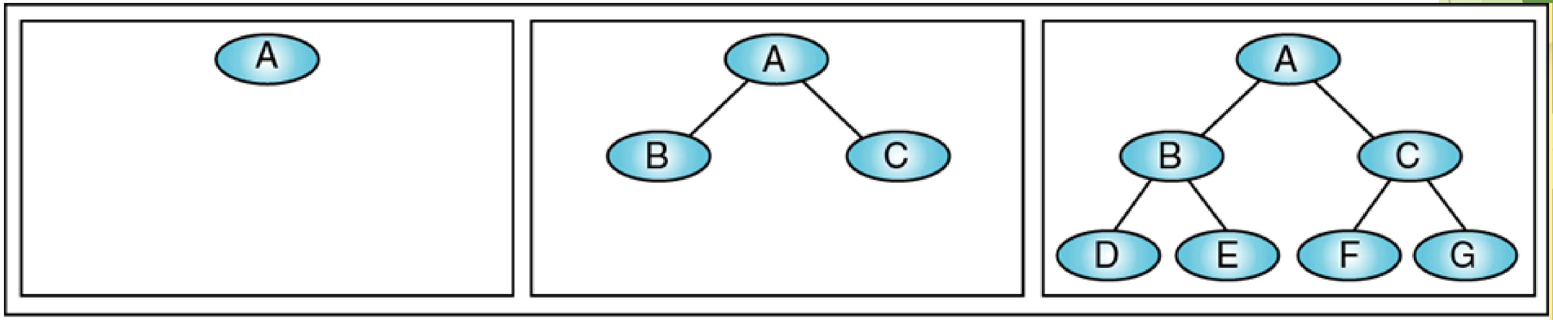
Binary Trees (4/9)

- Properties of binary trees
  - Lemma 2 [*Relation between number of leaf nodes and degree-2 nodes*]:

    For any nonempty binary tree, T, if n_0 is the number of leaf nodes and n_2 is the number of nodes of degree 2,

  then $n_0 = n_2 + 1$.

$n_2 = 0$ $n_2 = 1$ $n_2 = 3$



(a) Complete trees (at levels 0, 1, and 2)

Binary Trees (6/9)

- Binary tree representations (using array)
  - Lemma 3: If a complete binary tree with n nodes is represented sequentially, then for any node with index i, $1 \le i \le n$, we have

1. *parent(i)* is at $\lfloor i/2 \rfloor$ if $i \neq 1$.
   If $i = 1$, i is at the root and has no parent.
2. *LeftChild(i)* is at $2i$ if $2i \le n$.
   If $2i > n$, then i has no left child.
3. *RightChild(i)* is at $2i+1$ if $2i+1 \le n$.
   If $2i+1 > n$, then i has no left child

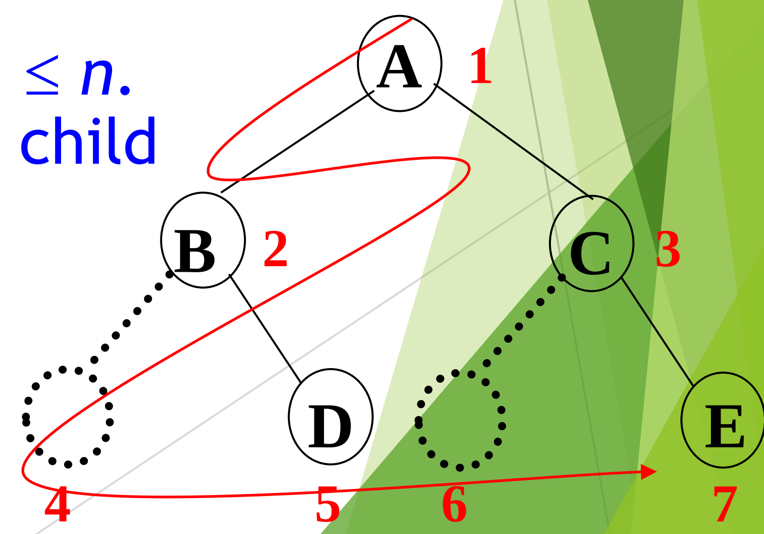
i = index of array
n = number of nodes
n = 5

[1]	[2]	[3]	[4]	[5]	[6]	[7]
A	B	C	—	D	—	E

Level 1

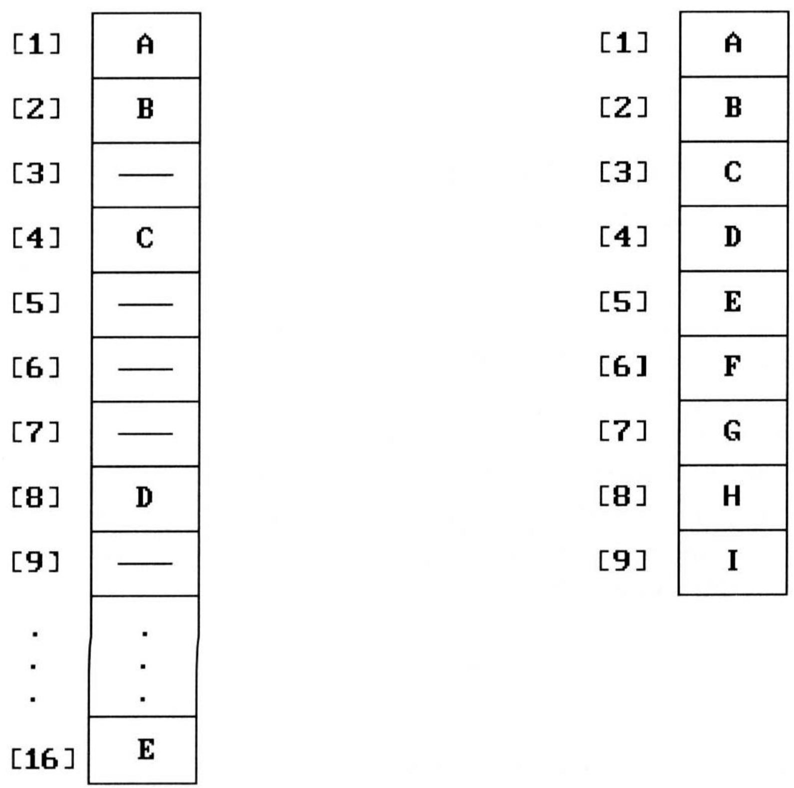
Level 2

Level 3



Binary Trees (7/9)

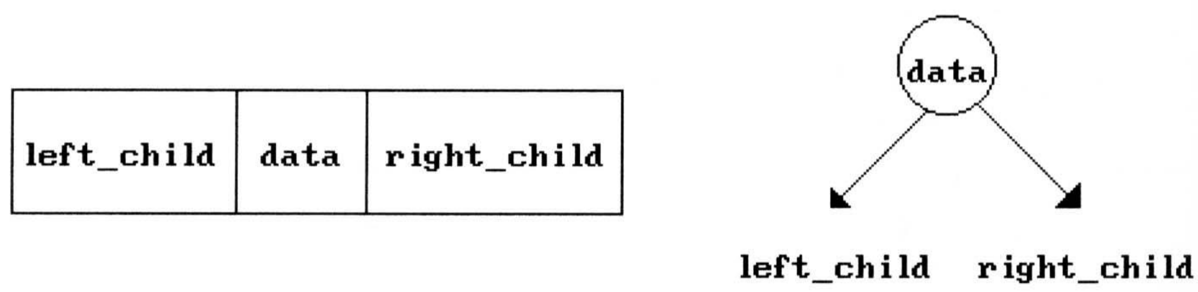
- Binary tree representations (using array)
  - Waste spaces: in the worst case, a skewed tree of depth k requires 2^k-1 spaces. Of these, only k spaces will be occupied
  - Insertion or deletion of nodes from the middle of a tree requires the movement of potentially many nodes to reflect the change in the level of these nodes



Binary Trees (8/9)

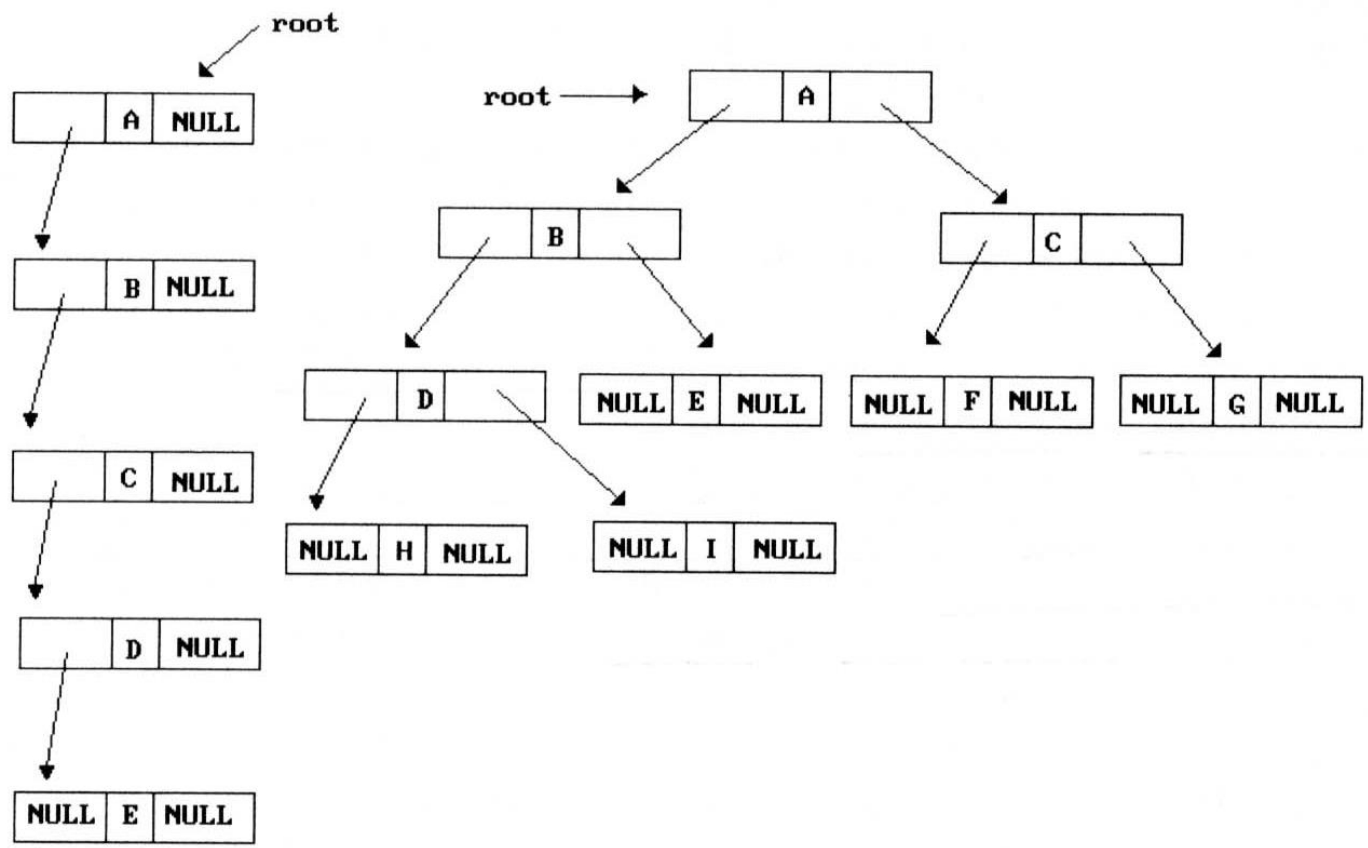
- Binary tree representations (using link)

```
typedef struct node *tree_pointer;
typedef struct node {
        int data;
        tree_pointer left_child, right_child;
        };
```

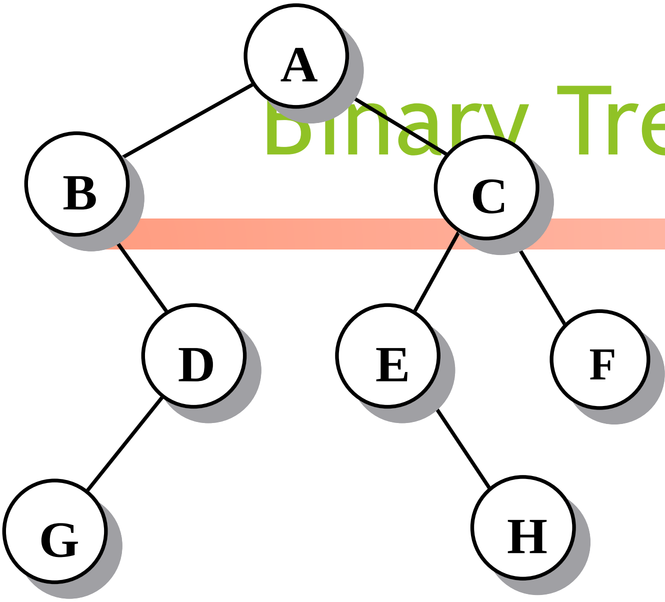


Binary Trees (9/9)

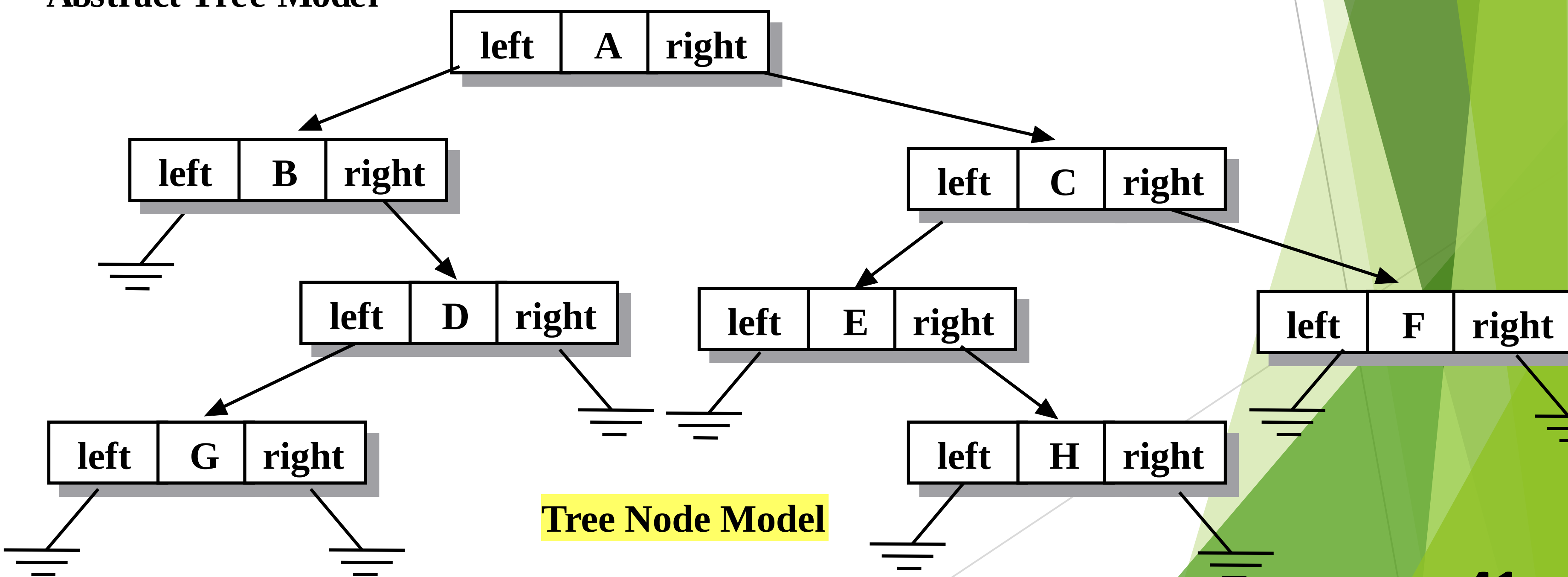
- Binary tree representations (using link)



Binary Tree Nodes



Abstract Tree Model

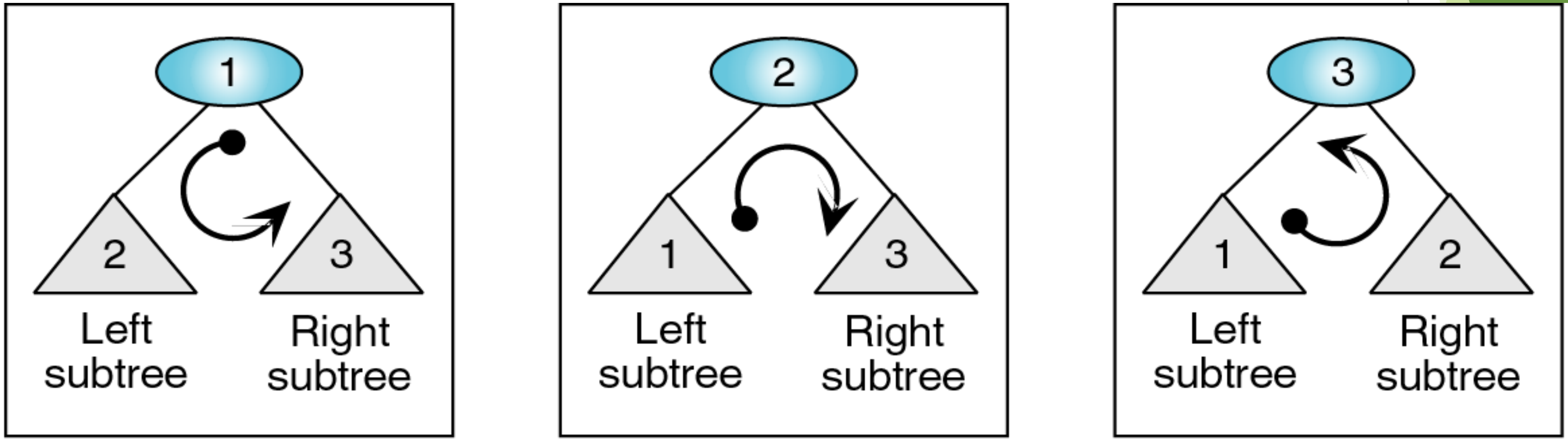


Tree Node Model

Traverse

Binary Trees: Traversals

- There are three classic ways to traverse a tree: NLR, LNR,LRN and Breadth-first



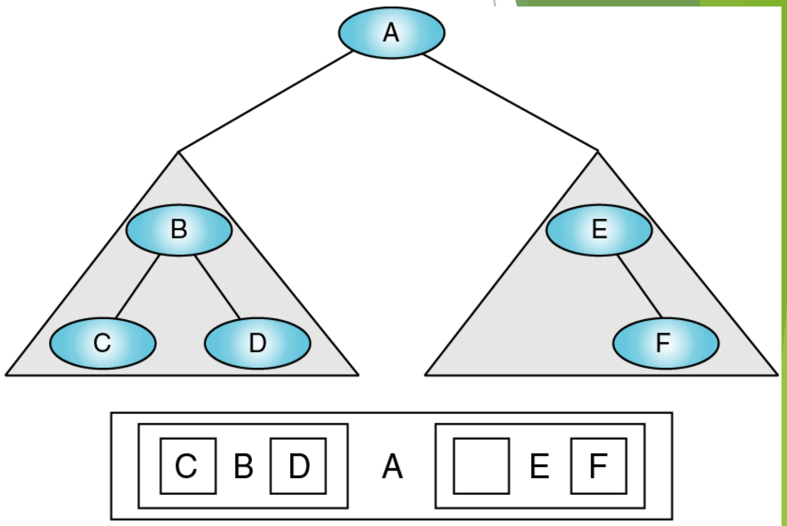
(a) Preorder traversal **(b) Inorder traversal** **(c) Postorder traversal**

Inorder (LNR) Traversal

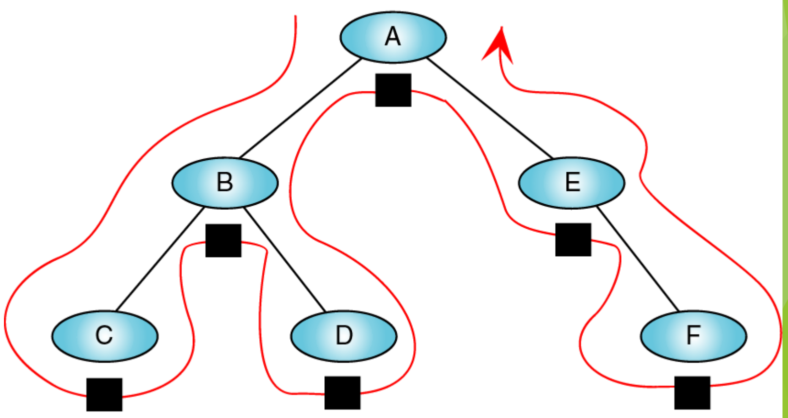
- Inorder traversal
- Visit the left subtree, then the node, then the right subtree.
- Algorithm:
  - If there is a left child visit it
  - Visit the node itself
  - If there is a right child visit it

Inorder (LNR) Traversal

```
inorder(node)
{
    if (node is not null)
     inorder(________________)
     print (________________)
     inorder (________________
}
```



(a) Processing order



(b) "Walking" order

Binary Tree Traversals

- Inorder traversal (*LVR*) (recursive version)

output A / B * C * D + E
:

```
void inorder(tree_pointer ptr)
/* inorder tree traversal */
{
    if (ptr) {
        inorder(ptr->left_child);
        printf("%d",ptr->data);
        inorder(ptr->right_child);
    }
}
```

L
V
R

Program 5.1: Inorder traversal of a binary tree

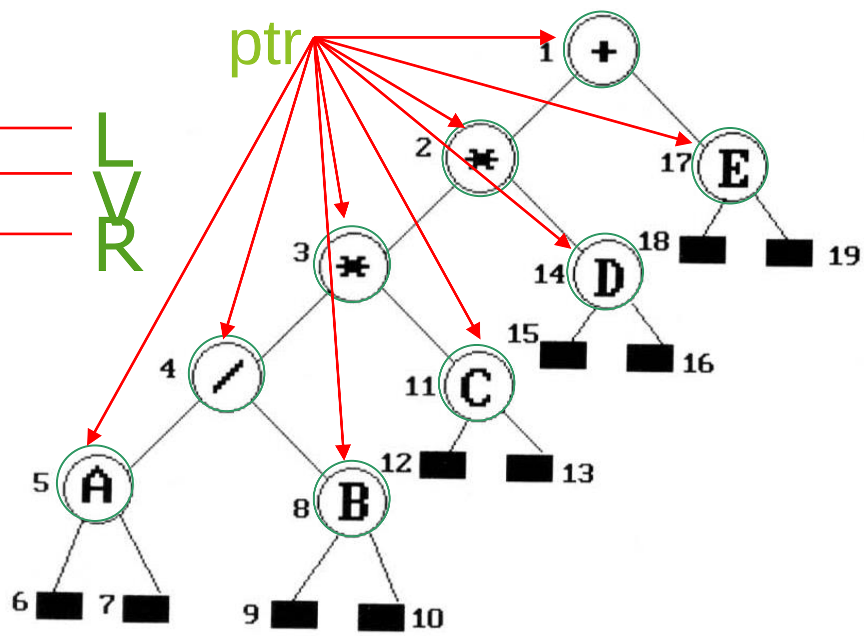


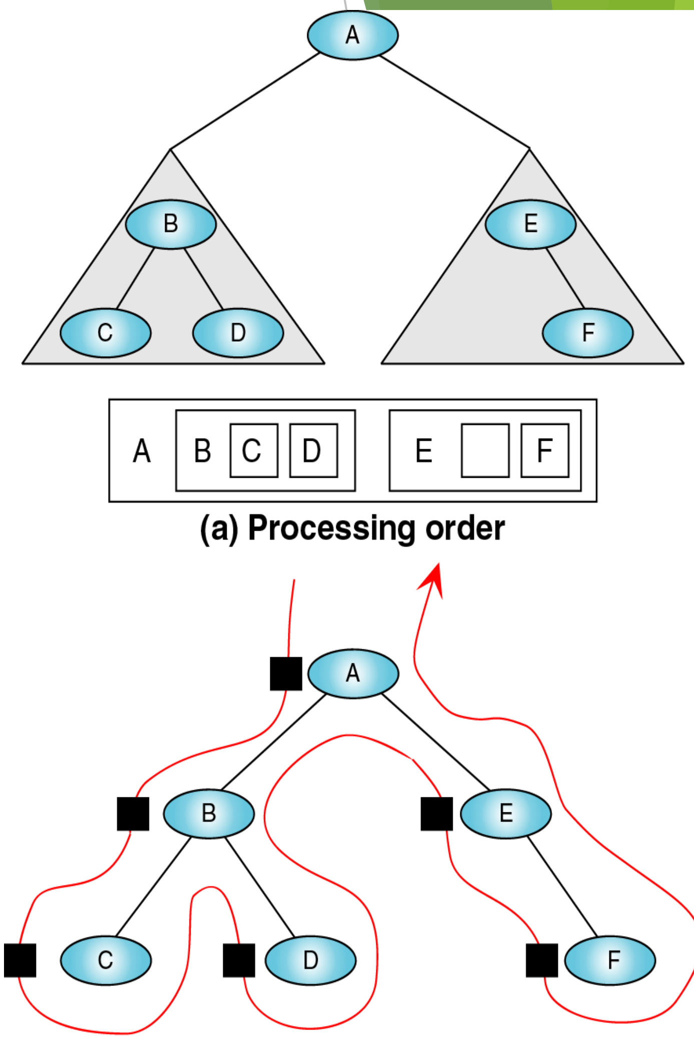
Figure 5.15: Binary tree with arithmetic expression

Preorder (NLR) Traversal

- Preorder traversal
- Visit each node then visit its children.
- Algorithm:
  - Visit the node itself
  - If there is a left child visit it
  - If there is a right child visit it

Preorder (NLR) Traversal

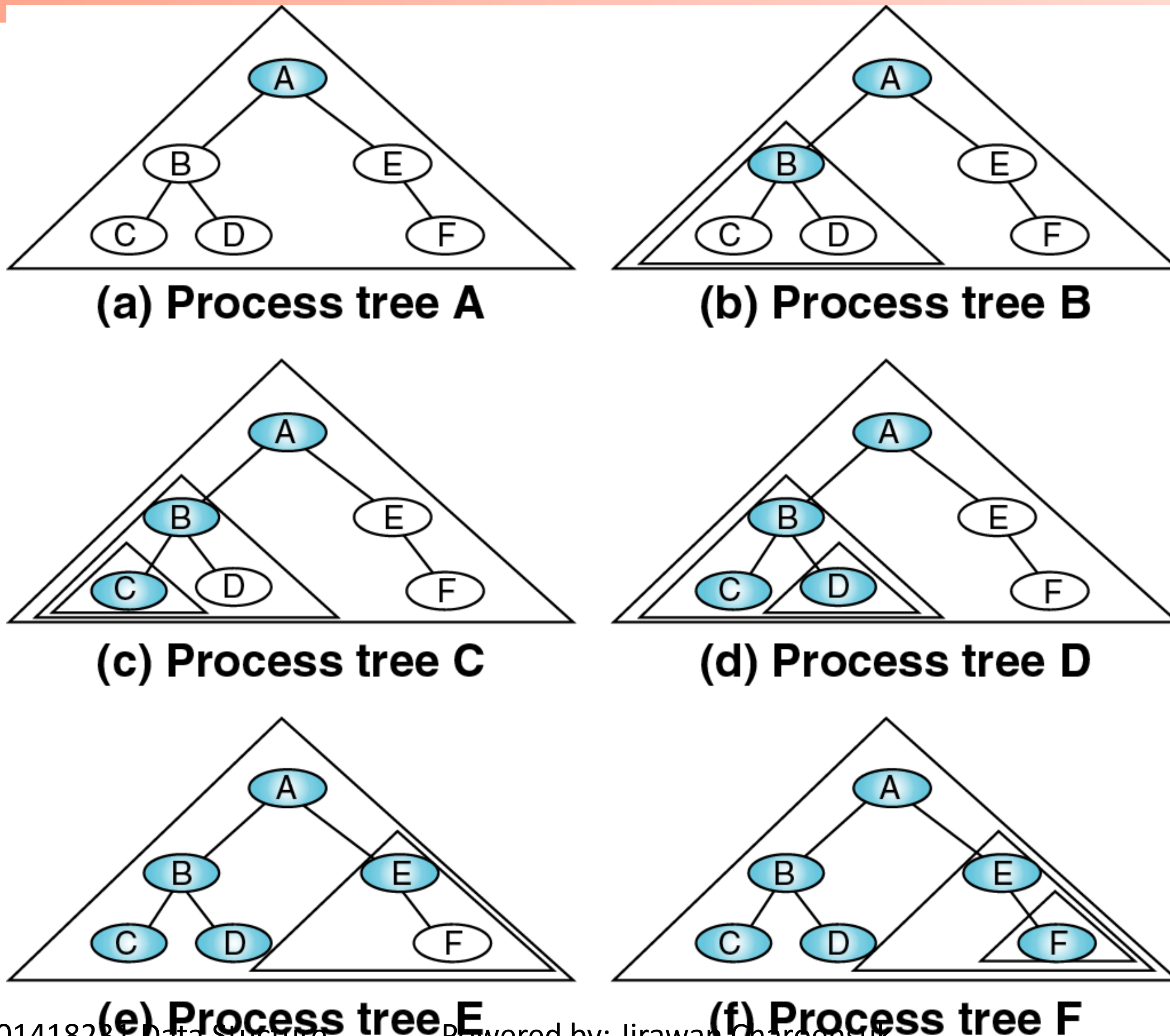
```
preorder(node)
{
    if (node is not null)
    print (_______________)
    preorder (_______________)
    preorder (_______________)


}
```



(a) Processing order

(b) "Walking" order

Preorder (NLR) Traversal



Binary Tree Traversals

- Preorder traversal (*VLR*) (recursive version)

output + * * / A B C D E
:

```
void preorder(tree_pointer ptr)
/* preorder tree traversal */
{
   if (ptr) {
      printf("%d",ptr->data);
      preorder(ptr->left_child);
      preorder(ptr->right_child);
   }
}
```

V
L
R

Program 5.2: Preorder traversal of a binary tree

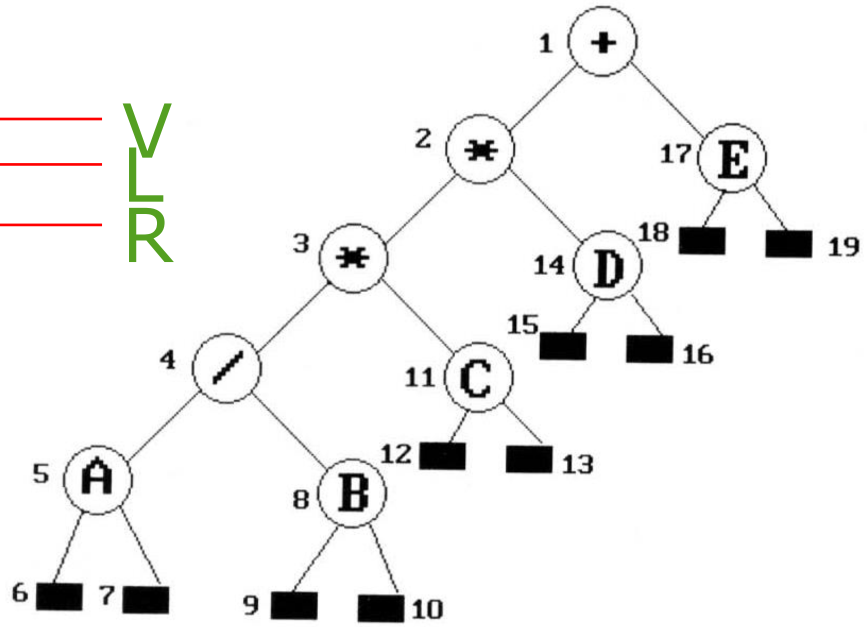


Figure 5.15: Binary tree with arithmetic expression

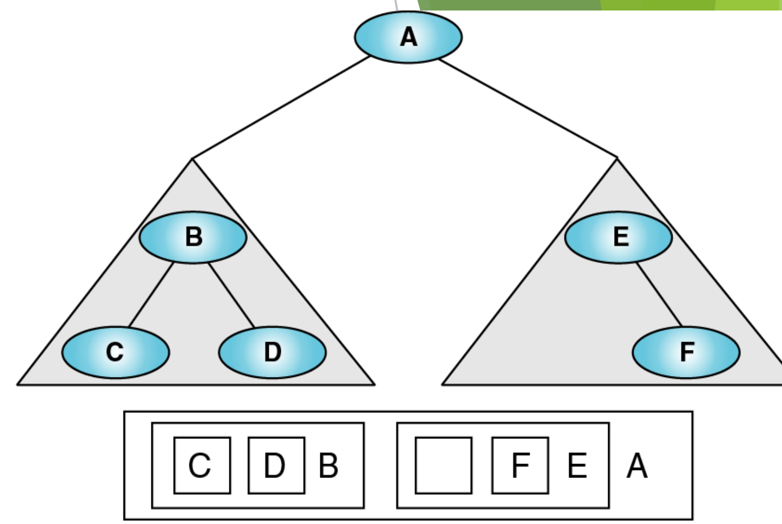
Postorder (LRN) Traversal

- Postorder traversal
- Visit each node after visiting its children.

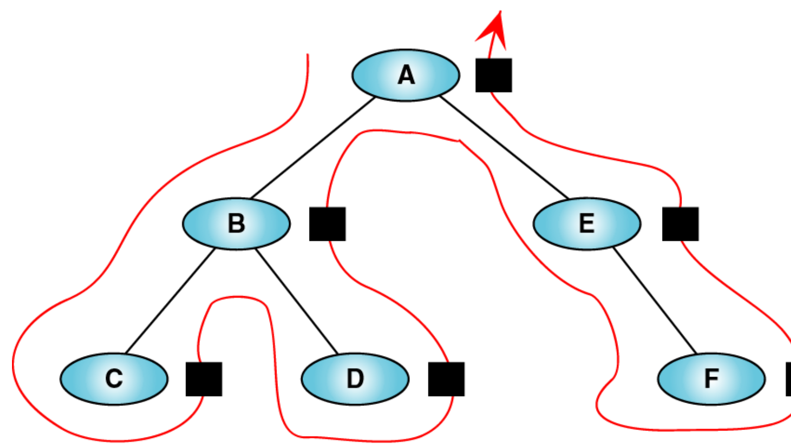
- Algorithm:
  - If there is a left child visit it
  - If there is a right child visit it
  - Visit the node itself

Postorder (LRN) Traversal

```
postorder(node)
{
    if (node is not null)
     postorder (_______________)
     postorder (_______________)
     print (_______________)
    }
```



(a) Processing order



(b) "Walking" order

Binary Tree Traversals

- Postorder traversal (*LRV*) (recursive version)

output A B / C * D * E +
:

```
void postorder(tree_pointer ptr)
/* postorder tree traversal */
{
   if (ptr) {
      postorder(ptr->left_child);
      postorder(ptr->right_child);
      printf("%d",ptr->data);
   }
}
```

L
R
V

Program 5.3: Postorder traversal of a binary tree

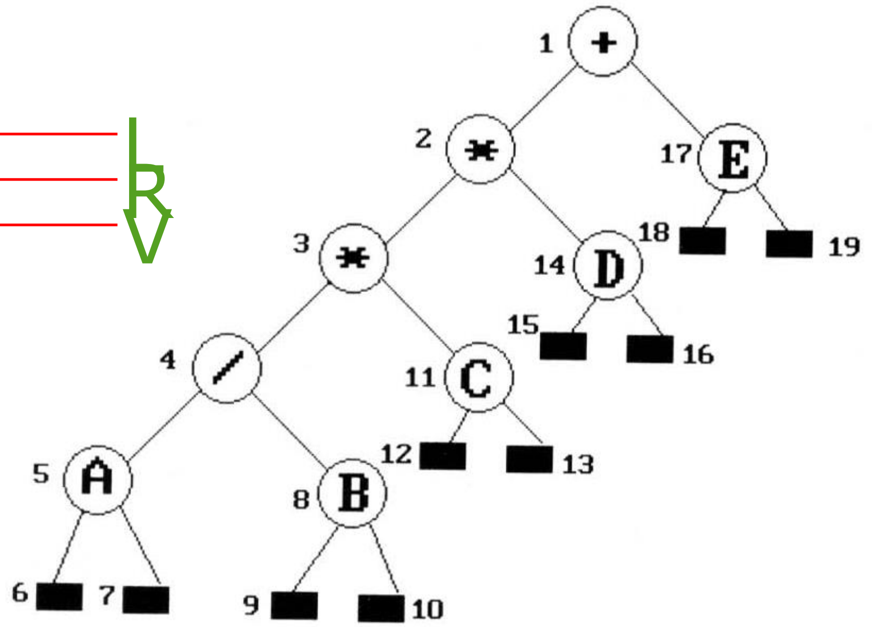
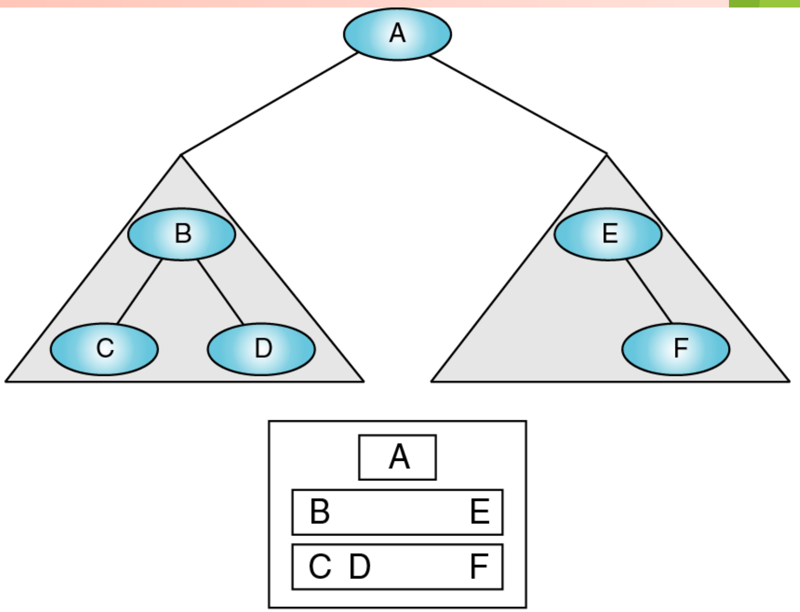


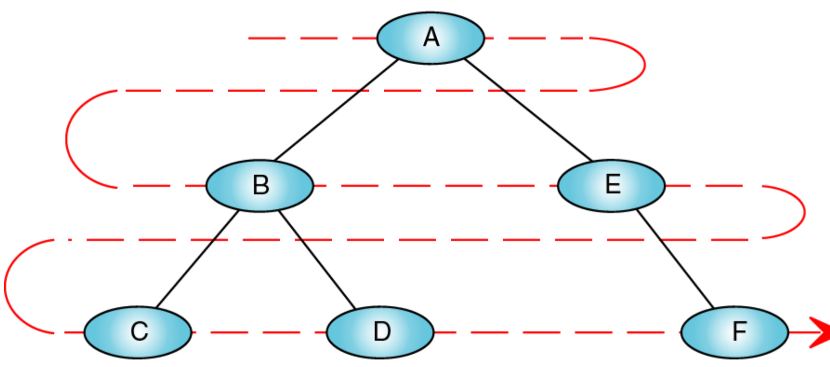
Figure 5.15: Binary tree with arithmetic expression

Breadth-first Traversal

```
Breadth-first(node)
{
    print (________________)
    if (________________ is not null)
     enqueue(ptr->left_child )
    if (________________is not null)
     enqueue(ptr->right_child )
    if (________________)
     dequeue(node)
}
```



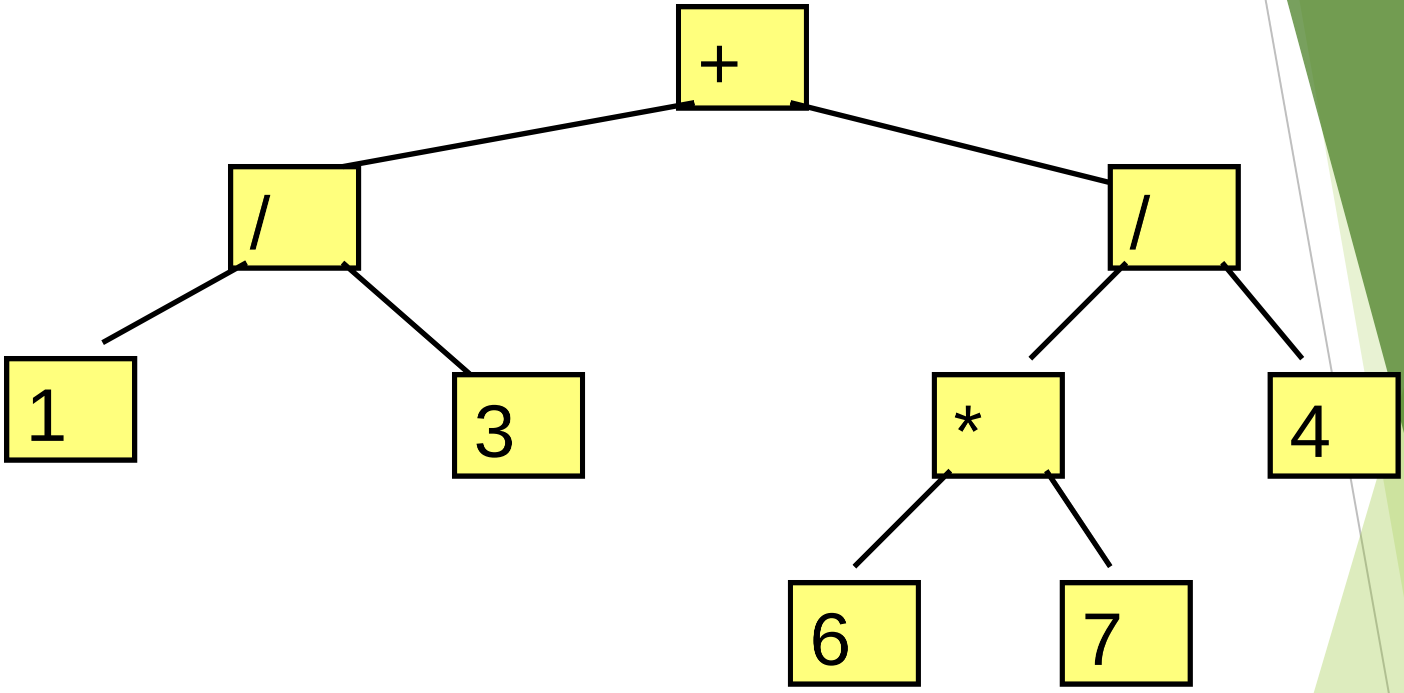
(a) Processing order



(b) “Walking” order

Expression Tree

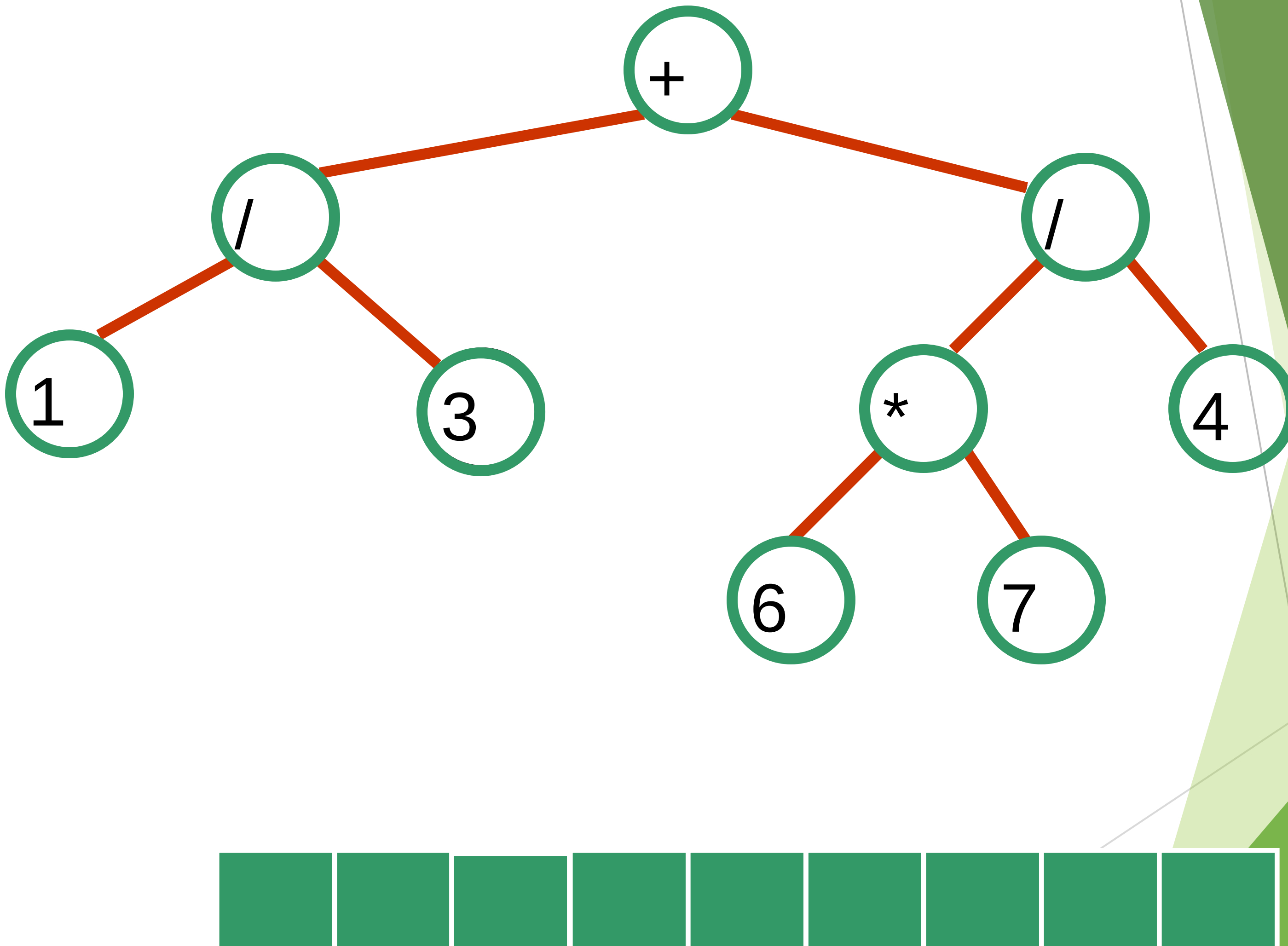
Example: Expression Tree



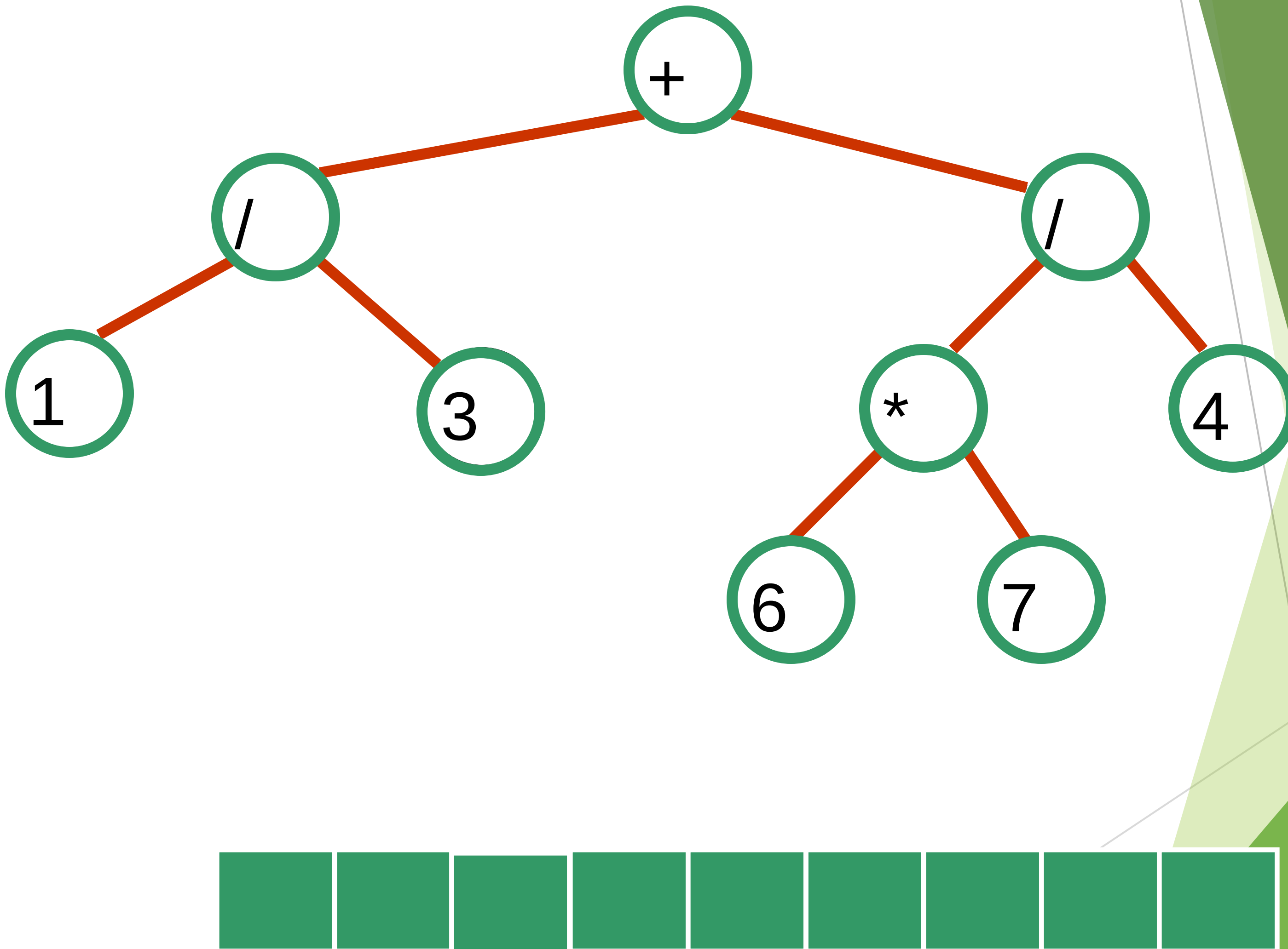
Notation

- Inorder
  - Infix Notation
- Preorder
  - Prefix Notation
- Postorder
  - Postfix Notation

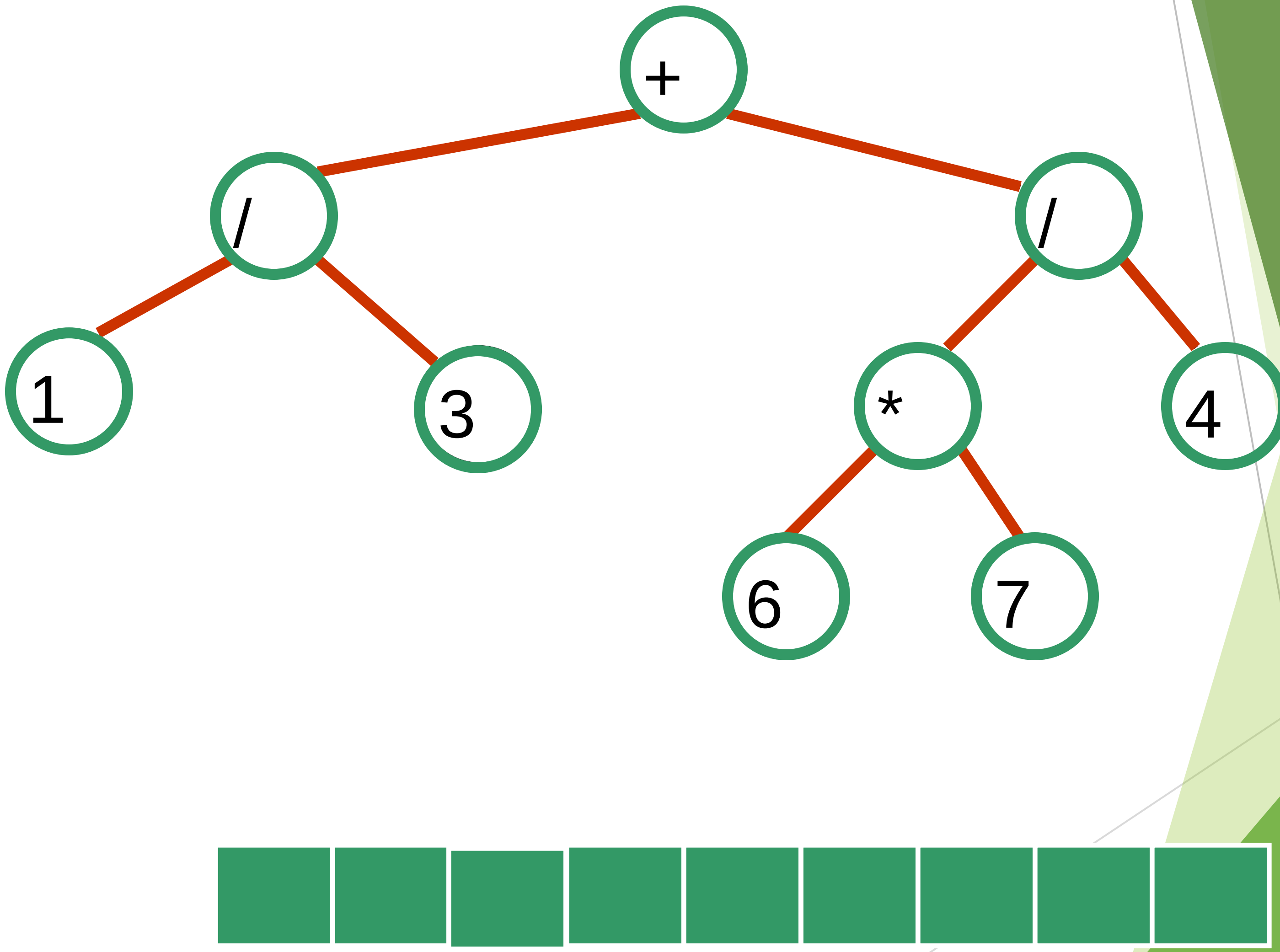
Example: Infix



Example: Prefix



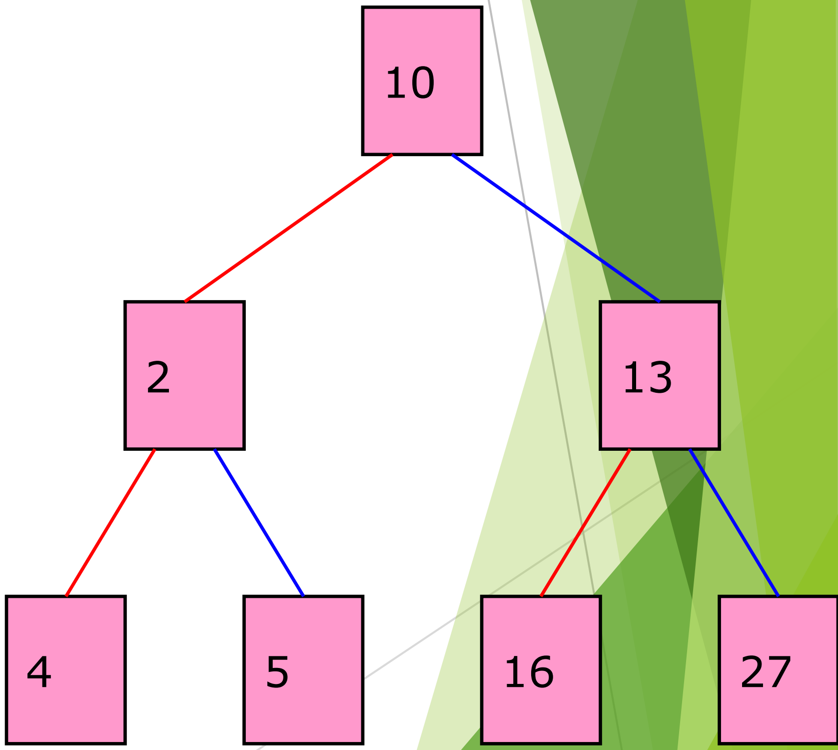
Example: Postfix



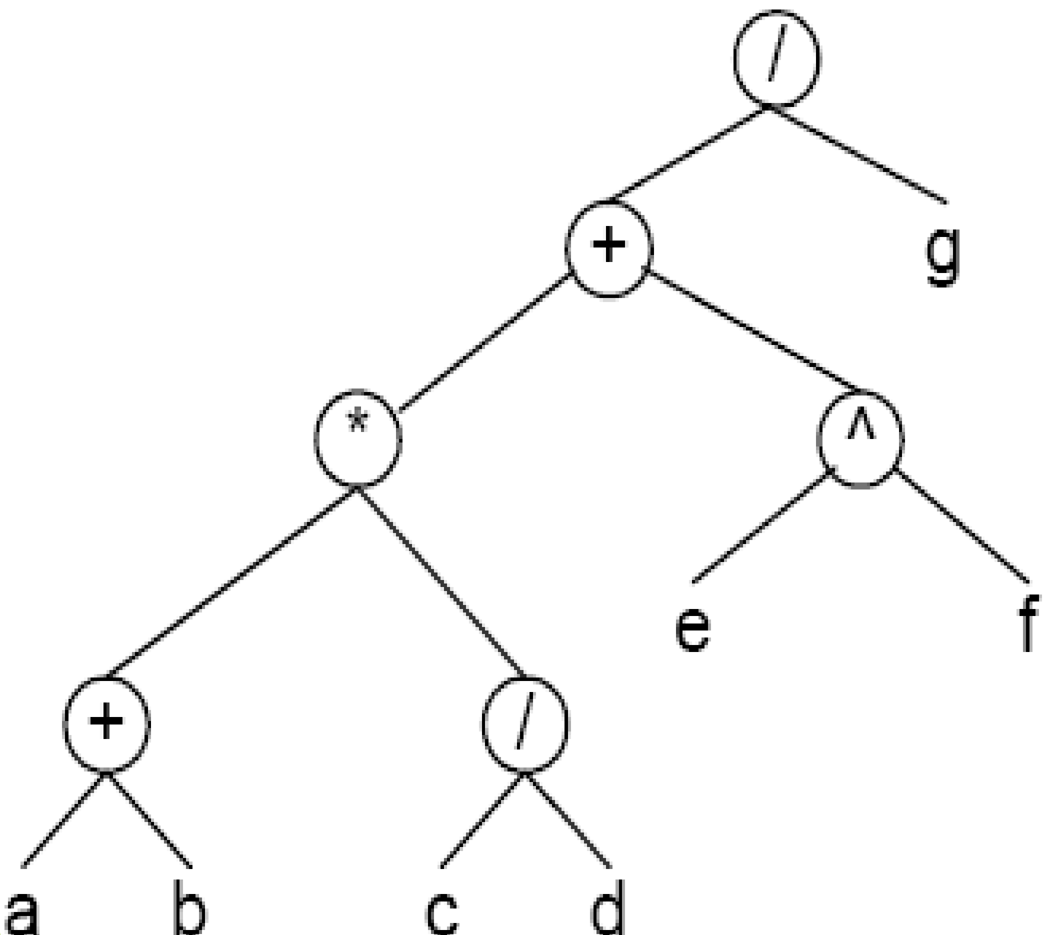
Quiz

Traversals

- Three traversals of a binary search tree
  - Preorder Traversal
    -
  - Inorder Traversal
    -
  - Postorder Traversal
    -



QUIZ: Expression Tree



Summary

- Tree Terminology
- Binary tree
  - Structure and properties
  - Implement
    - Array, Linked list
  - Traversals
    - Inorder, Preorder, Postorder, Bread first search
  - Expression Tree

Question