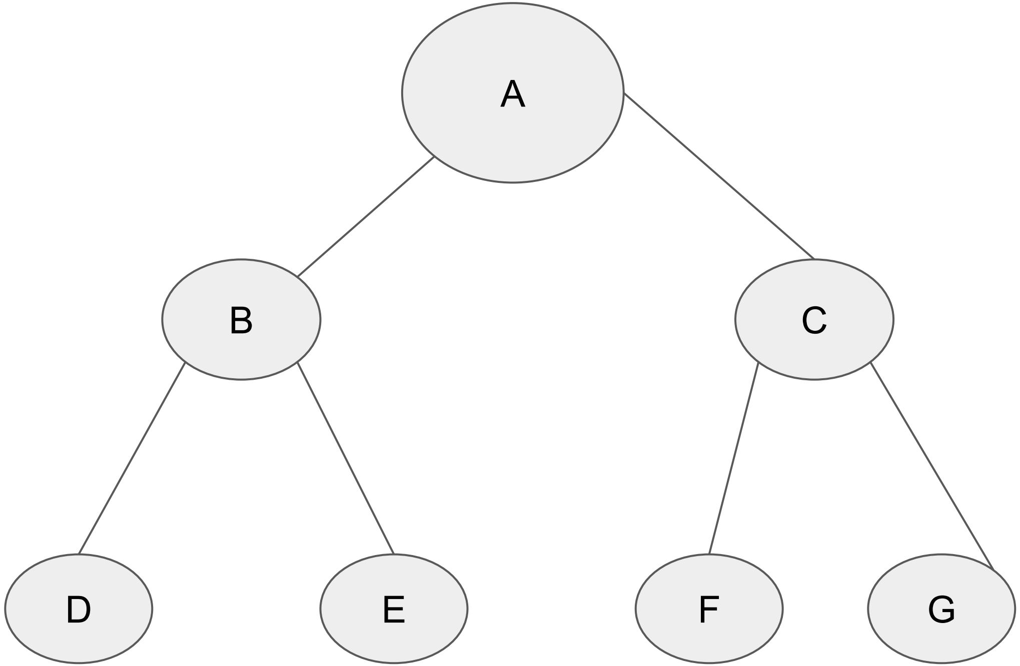
CWB-Module- 4

Tree and Graph Traversals

Traversals

Pre-order (Root, LST, RST)
Post-order (LST, RST, Root)
In-order (LST, Root, RST)

Example 1

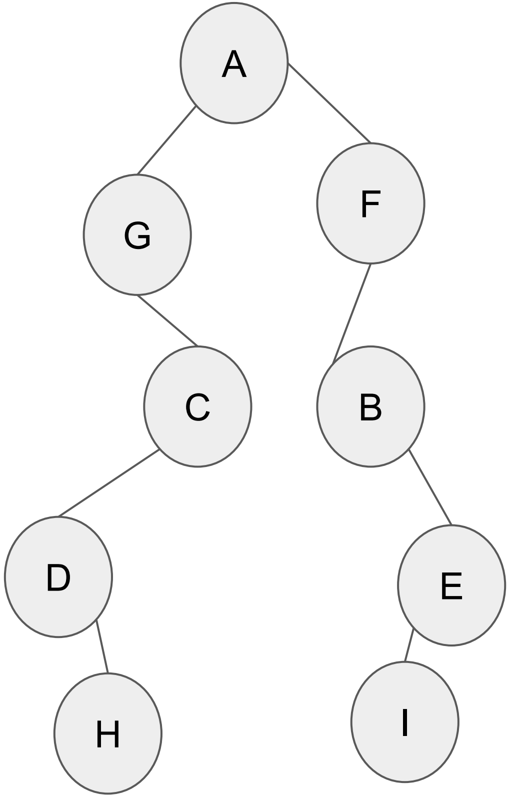


Pre-order - ABDECFG

Post-order - DEBFGCA

In-order - DBEAFCG

Example 2



Pre-order : AGCDHFBEI

Inorder : GDHCABIEF

Post-order : HDCGIEBFA

Binary Search Tree

A Binary Search Tree (BST) is a node-based binary tree data structure that maintains the following properties:

1. Left Subtree Property: All nodes in the left subtree of a node have values less than the node's value.
2. Right Subtree Property: All nodes in the right subtree of a node have values greater than the node's value.
3. No Duplicate Nodes: Typically, BSTs do not contain duplicate values (though some implementations may allow them in either the left or right subtree).
4. Recursive Structure: Both the left and right subtrees must also be binary search trees.

BST Example

```
     8
    / \
   3   10
  / \    \
 1   6    14
    / \   /
   4   7 13
```

- All left children are smaller than their parent.
- All right children are larger than their parent.

Important Notes

1. Inorder traversal of BST will always produce ascending order.

   Preorder(root)

   {

   print(root->data)

   Preorder(root->lc)

   Preorder(root->rc)

   }

Postorder

```
Postorder(root)
{
postorder(root->lc)
postorder(root->rc)
print(root->data)
}
```

Inorder

```
Inorder(root)
{
Inordert(root->lc)
print(root->data)
Inorder(root->rc)


}
```

Queue Data Structure

A queue is a linear data structure that follows the First-In-First-Out (FIFO) principle, meaning the first element added is the first one to be removed. It is widely used in:

- Breadth-First Search (BFS/BFT)
- Level-order traversal in trees
- CPU scheduling
- Print spooling

Basic Queue Operations

Operation	Description	Time Complexity
Enqueue	Adds an element to the rear (end) of the queue	O(1)
Dequeue	Removes and returns the front element	O(1)
Peek/Front	Returns the front element without removing it	O(1)
IsEmpty	Checks if the queue is empty	O(1)
Size	Returns the number of elements in the queue	O(1)

Graph Traversal - BFT

```
BFT(v)

{ visited(v)=1

add(v,Q)

while(Q is not empty)

{ x=delete(Q)

print(x)

for all w adjacent to x

{ if (w is not visited)

visited(w)=1

add(w,Q) } } }
```

BFT Details

1. To implement BFT we are using queue data structure.
2. Time Complexity = O(V+E)
3. It is also called Level Order Traversal.

Applications of BFT

1. We can verify given graph is connected or not.
2. Using BFT, we can find connected components in the given graph.
3. Using BFT, we can check given graph contain cycle or not.
4. Using BFT, we can find out shortest path from the given source to every vertex in the given unweighted graph.

DFT (Depth First Traversal)

```
DFT(V)
{
visited(v)=1
print(v)
for all w adjacent to v
{
if w is not visited
DFT(w)
}
}
```

DFT Details

1. To implement DFT, we are using stack.
2. Time Complexity = O(V+E)

Applications of DFT

1. We can verify given graph contain cycle or not.
2. We can verify given graph is connected or not.
3. We can find out the number of connected components.
4. DFT cannot work finding single shortest path in the given unweighted graph.