Daffodil International University

ASSIGNMEMT II

COURSE CODE: CSE 214

COURSE NAME: Algorithm

Submitted To

Subroto Nag Pinku

Department of CSE

Daffodil International University

Submitted By

Md. Foysal Ahmed

ID: 191-15-12486

Section: O-14

1. Full Tree Traversal

Ans:

Traversal is a process of visiting each node in the tree exactly once in some order. By visiting the nodes means printing the nodes in some order. For tree traversal there are two algorithms.

- Breadth-First Search
- Depth-First Search

BFS can be called as level order traversal algorithm. By level means that in each level first to visit all the nodes in the same level than shift to next level.

In DFS traversal there are three techniques to visit the nodes and those are,

- Pre-order Traversal
- In-order Traversal
- Post-order Traversal

Pre-order traversal

In pre-order traversal we visit the root first than the left subtree and then the right sub tree. We should always remember that every node may represent a subtree of itself.
The algorithm of Pre-order traversal can be written like this

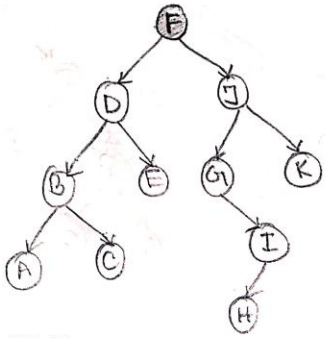
Until all nodes are traversed −
Step 1 – Visit root node
Step 2 – Recursively traverse left subtree
Step 3 − Recursively traverse right subtree.

Let's consider a binary tree and traverse in pre-order.

Md. Foysal Ahmed
ID: 191-15-12486
Section: O-14

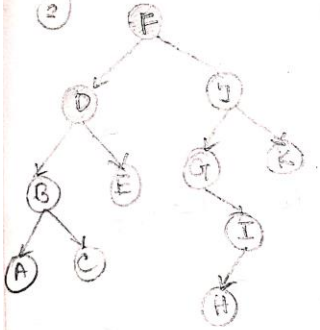
Preorder →

1



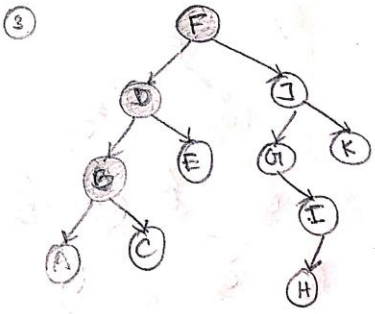
Result: F

2



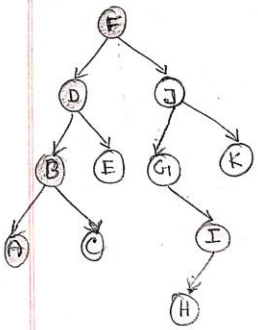
Result: F, D

3



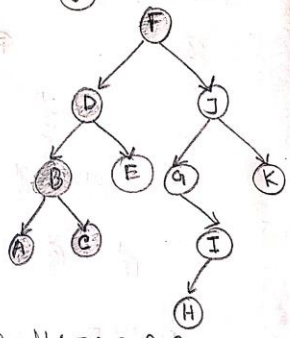
Result: F, D, B

4



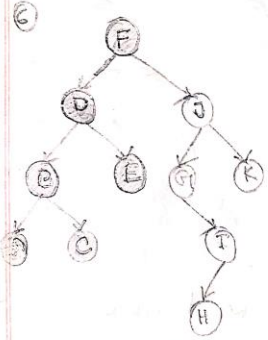
Result: F, D, B, A

5



Result: F, D, B, A, C

6

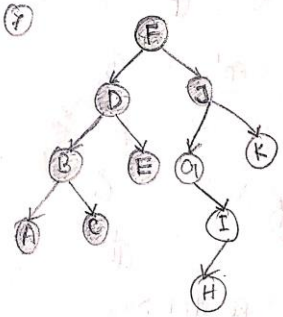


Result: F, D, B, A, C, E

Completion of left subtree.

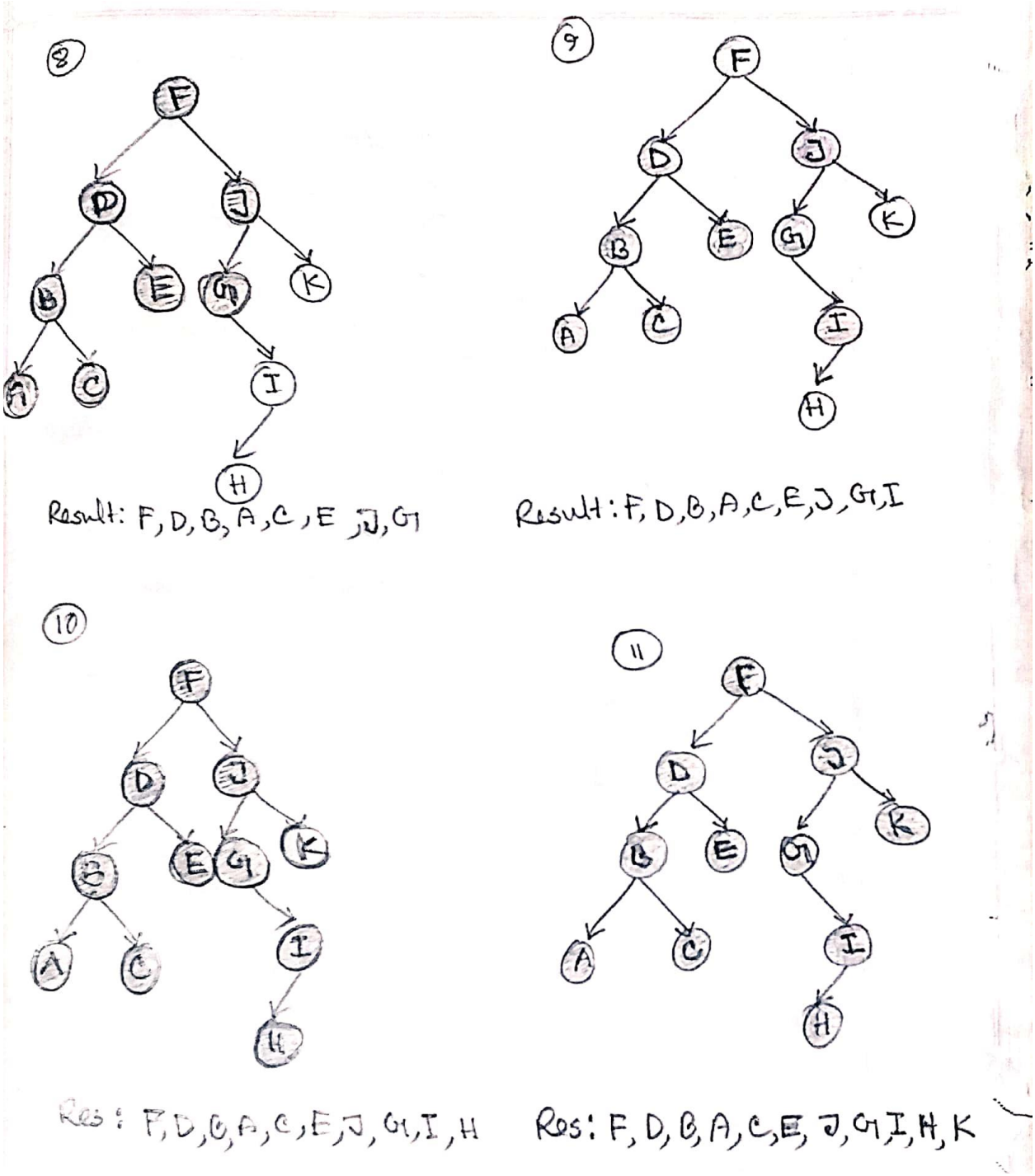
Now Right subtree:

7



Result: F, D, B, A, C, E, J

Md. Foysal Ahmed
ID: 191-15-12486
Section: O-14



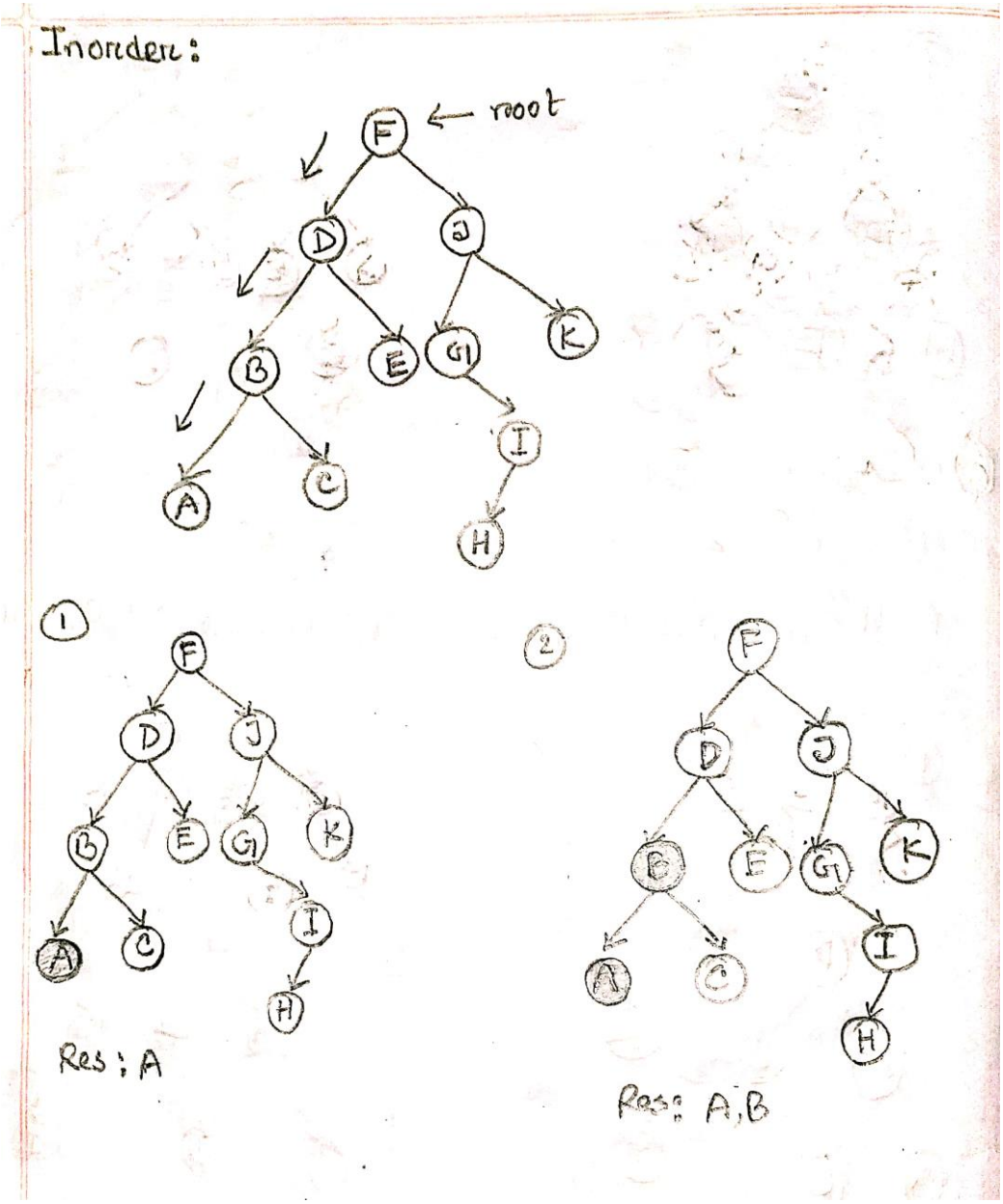
Therefore, the final result is **F, D, B, A, C, E, J, G, I, H, K**

Md. Foysal Ahmed
ID: 191-15-12486
Section: O-14

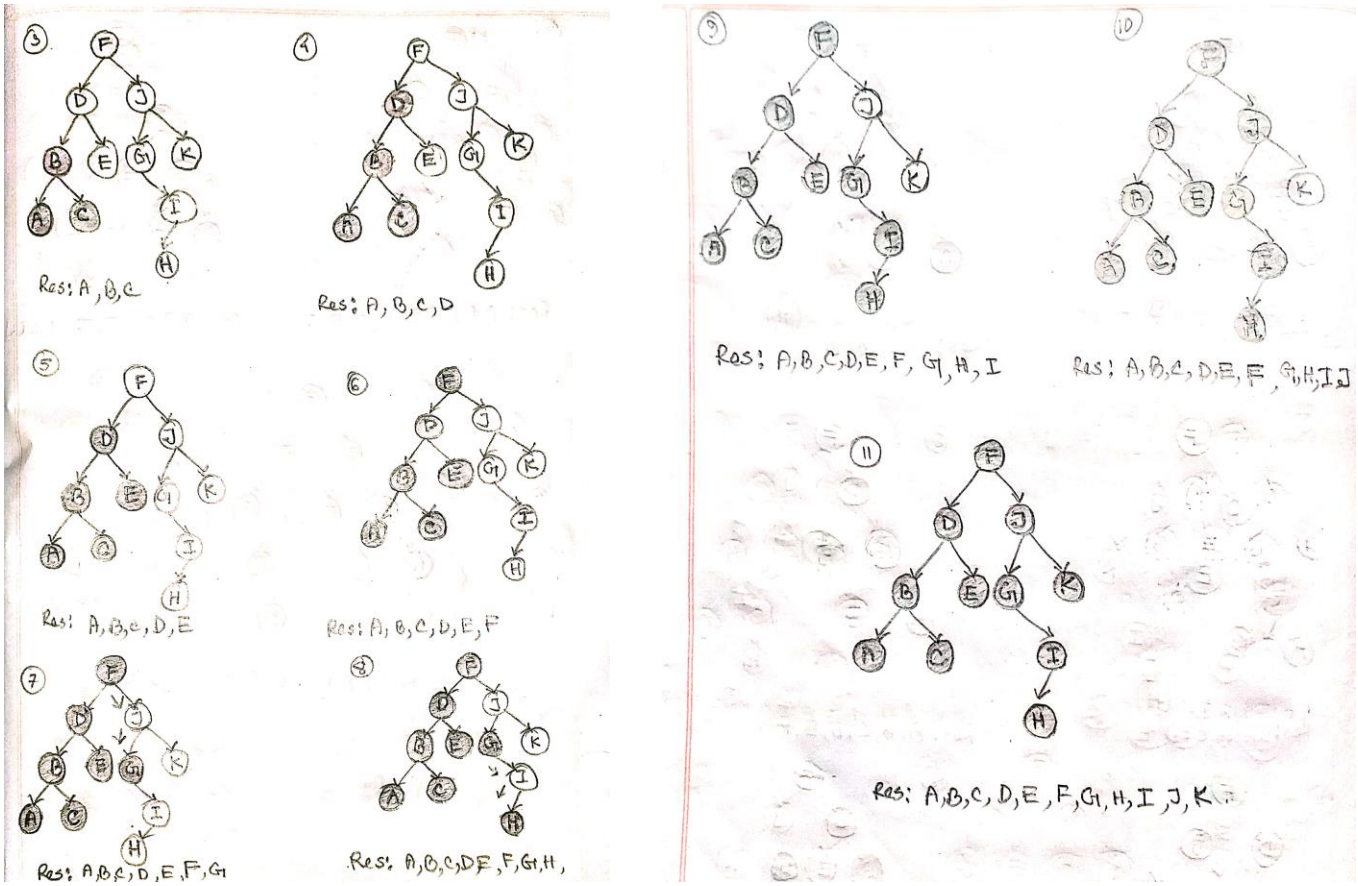
In-order Traversal

In in-order traversal we first the visit the left-subtree then the root and then the right-subtree. We should always remember that every node may represent a subtree of itself.
The algorithm of In-order traversal can be written like this
Step 1 − Recursively traverse left subtree.
Step 2 − Visit root node.
Step 3 − Recursively traverse right subtree.

Let's consider a binary tree and traverse in In-order.



Md. Foysal Ahmed
ID: 191-15-12486
Section: O-14



Therefore, the final result is **A, B, C, D, E, F, G, H, I, J, K**

Post-order Traversal

In post-order traversal we visit the left and right subtree first and then visit the root of the subtree. The algorithm of In-order traversal can be written like this:

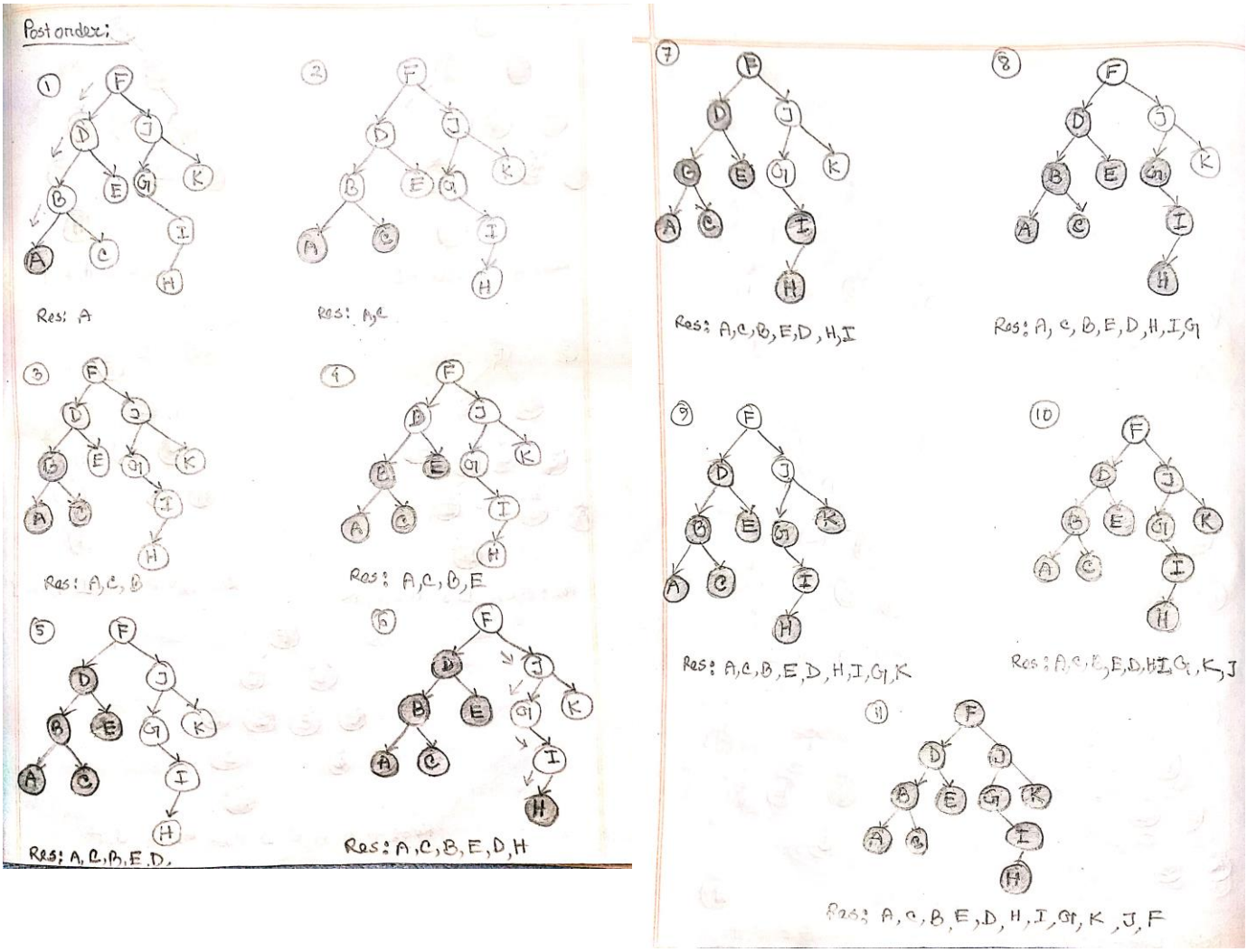
Step 1 − Recursively traverse left subtree.

Step 2 − Recursively traverse right subtree.

Step 3 − Visit root node.

Let's consider a binary tree and traverse in post-order.

Md. Foysal Ahmed
ID: 191-15-12486
Section: O-14



Therefore, the final result is **A, C, B, E, D, H, I, G, K, J, F**

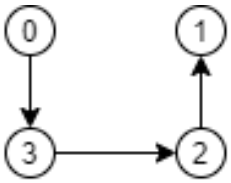
2. Cycle Finding

Ans:

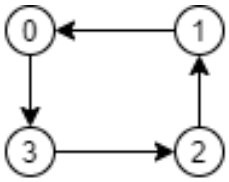
In graph theory, a cycle graph or circular graph is a graph that consists of a single cycle, or in other words, some number of vertices connected in a closed chain.

To determine the cycle in a graph we can implement DFS. rom every unvisited node. Depth First Traversal can be used to detect a cycle in a Graph. DFS for a connected graph produces a tree. There is a cycle in a graph only if there is a back edge present in the graph. A back edge is an edge that is joining a node to itself (self-loop) or one of its ancestor in the tree produced by DFS.

Md. Foysal Ahmed
ID: 191-15-12486
Section: O-14



Acyclic graph (no back edeges)



Cyclic graph (introducing back edeges)

To detect a cycle in a directed graph (i.e to find a back edge), we can use depth-first search (with some introduction of local state to tell we if a back edge occurs):
[We will maintain 3 buckets of vertices: white, grey, & black buckets. (We can also color vertices instead)]

- The white bucket will contain all of the unvisited vertices. At the start of our traversal, this means every vertex is in the white bucket.
- Before visiting a vertex, we will move it from the white bucket into the grey bucket.
- After fully visiting a vertex, it will get moved from the grey bucket into the black bucket.
- We can skip over vertices already in the black bucket, if we want to try and visit them again.
- When visiting the children/descendants of a vertex, if we come to a descendant vertex that is already in the grey bucket - that means we have found a back edge/cycle.
- This means the current vertex has a back edge to its ancestor - as we only arrived at the current vertex via it's ancestor. So we have just determined that there is more than one path between the two (a cycle).

To detect a cycle in an undirected graph, it is very similar to the approach for a directed graph. However, there are some key differences:

- We no longer color vertices/maintain buckets. Instead we use visited set to keep track the visited vertex.
- We have to make sure to account for the fact that edges are bi-directional - so an edge to an ancestor is allowed, if that ancestor is the parent vertex.
- We only keep track of visited vertices (similar to the grey bucket).
- When exploring/visiting all descendants of a vertex, if we come across a vertex that has already been visited then we have detected a cycle.

Time complexity is O (V + E) for an adjacency list. Space complexity is O(V). For an adjacency matrix, the time & space complexity would be O(V^2).

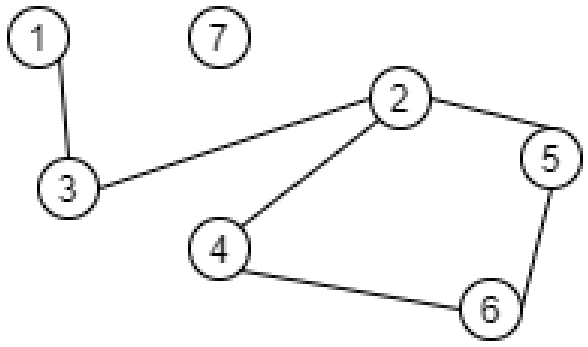
Md. Foysal Ahmed
ID: 191-15-12486
Section: O-14

3. Component Finding

Ans:

In a graph, connected components are set of vertices which are reachable. In other words, connected component is a subgraph in which any two vertices are connected by path.

Let's consider a graph shown below and find the component of the graph,



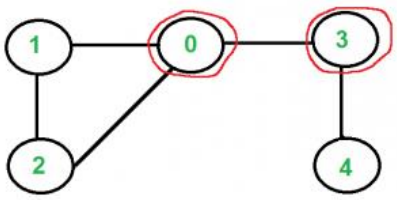
Component finding can be done by using both DFS and BFS. We will implement the DFS or BFS in a certain node and will see that if all the nodes are visited or not. If we found that there exist some nodes that are not visited at all then again we will run BFS or DFS on that particular node.
The above graph shows that If we start DFS using the vertex 1 then we can have visited all the nodes till 6. That means we found 1 connected components in the graph. But vertex 7 is not connected to any other vertex that means we have to run DFS again on this vertex and count the connected component to 1
Therefore, after summing up all the connected components in the graph we find that in the graph there are **2 connected component.**

4. Articulation Points in a Graph

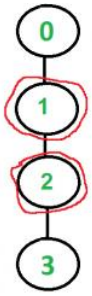
Ans:

A vertex in an undirected connected graph is an articulation point (or cut vertex) if removing it (and edges through it) disconnects the graph. Articulation points represent vulnerabilities in a connected network – single points whose failure would split the network into 2 or more components. They are useful for designing reliable networks.
For a disconnected undirected graph, an articulation point is a vertex removing which increases number of connected components.
Following are some example graphs with articulation points encircled with red color.

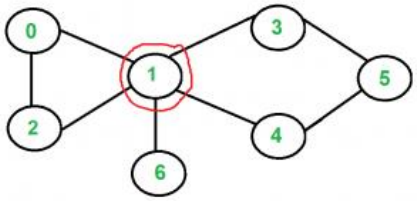
Md. Foysal Ahmed
ID: 191-15-12486
Section: O-14



Articulation points are 0 and 3



Artiulation
Points are 1 & 2



Articulation Point is 1

A simple approach to find articulation point is to all the vertices one by one and observe if removal of a vertex causes disconnected graph.
For every vertex v, do following

* **Remove v from graph**
* **See if the graph remains connected (We can either use BFS or DFS)**
* **Add v back to the graph**

Time complexity of above method is O(V*(V+E)) for a graph represented using adjacency list.

We also can use DFS to find the articulation point. In DFS, we follow vertices in tree form called DFS tree. In DFS tree, a vertex u is parent of another vertex v, if v is discovered by u (obviously v is an adjacent of u in graph). In DFS tree, a vertex u is articulation point if one of the following two conditions is true.
1) u is root of DFS tree and it has at least two children.
2) u is not root of DFS tree and it has a child v such that no vertex in subtree rooted with v has a back edge to one of the ancestors (in DFS tree) of u.
Time complexity using DFS is O(V+E).

We do DFS traversal of given graph with additional code to find out Articulation Points (APs). In DFS traversal, we maintain a parent [] array where parent[u] stores parent of vertex u. Among the above mentioned two cases, the first case is simple to detect. For every vertex, count children. If currently visited vertex u is root (parent[u] is NIL) and has more than two children, print it.
How to handle second case? The second case is trickier. We maintain an array disc [] to store discovery time of vertices. For every node u, we need to find out the earliest visited vertex (the vertex with minimum discovery time) that can be reached from subtree rooted with u. So we maintain an additional array low [] which is defined as follows.

low[u] = min(disc[u], disc[w])
where w is an ancestor of u and there is a back edge from
some descendant of u to w.

Md. Foysal Ahmed
ID: 191-15-12486
Section: O-14

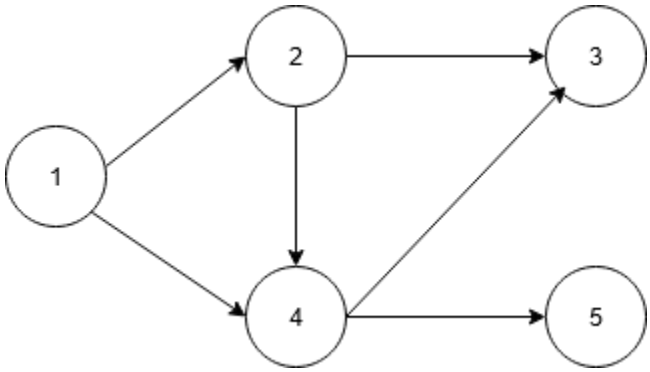
5. Topological Sort

Ans:

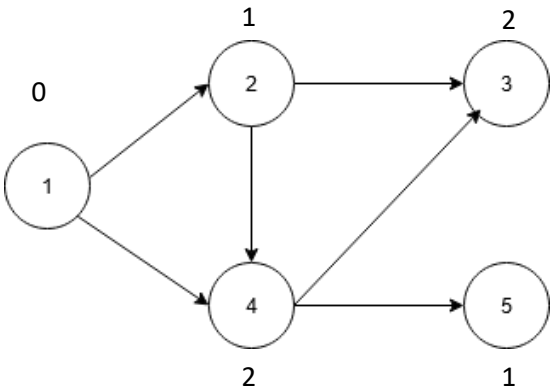
It is a linear ordering of its vertices such that for directed edge uv for vertex u to v, u comes before vertex v. The condition of topological sort is the graph must be DAF (Directed Acyclic Graph). In every DAG there should have at least one topological ordering.
Steps of topological sort given below:

- Find out the in degree of given graph.
- Start with the vertex having 0 in degree.
- Store the vertex and remove the vertex from the graph along with its edges.
- After removing the vertex update the in degree for the remaining graph and repeat the above steps till the graph is not becomes empty.

Let's consider a graph and find out the topological ordering from the graph:



First we find out the in-degree of the graph. The graph becomes,

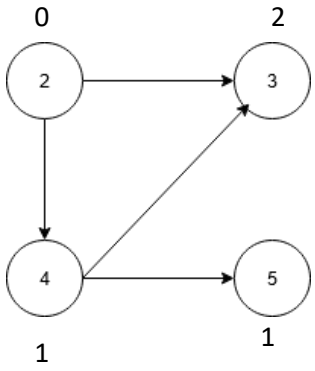


Md. Foysal Ahmed
ID: 191-15-12486
Section: O-14

Storing the vertex having in-degree 0.

Store [] = 1

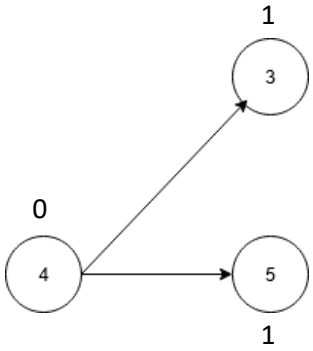
Now removing the adjacent vertex of 1 and update the in-degree



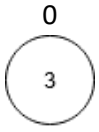
Storing the vertex having in-degree 0.

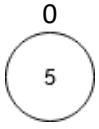
Store [] = 1,2

Again repeating above steps;



Store [] = 1,2,4





Md. Foysal Ahmed
ID: 191-15-12486
Section: O-14

Finally Store [] = 1,2,4,3,5

It is the topological sort of the following graph.

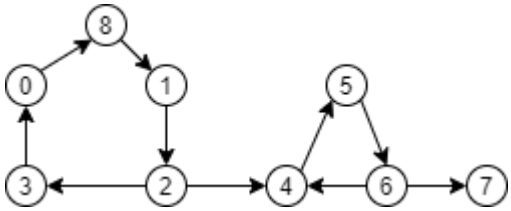
6. Strongly Connected Components

Ans:

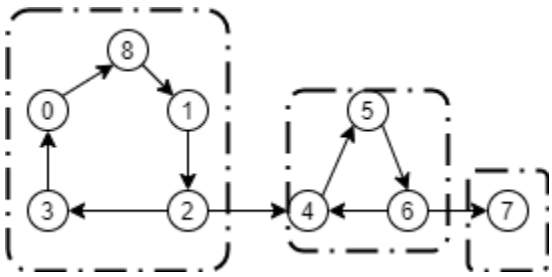
In a directed graph when we can go from node u to v and also v to u then both are called strongly connected node (SSC). Also A strongly connected component is the portion of a directed graph in which there is a path from each vertex to another vertex. It is applicable only on a directed graph.

For example:

Let us take the graph below.



The strongly connected components of the above graph are:



We can observe that in the first strongly connected component, every vertex can reach the other vertex through the directed path.

These components can be found using Kosaraju's Algorithm.

Kosaraju's Algorithm:
1)Kosaraju's Algorithm is based on the depth-first search algorithm implemented twice.

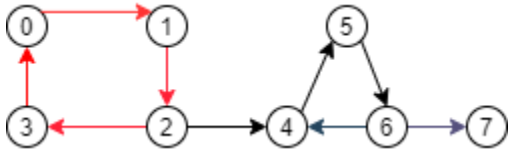
Three steps are involved.

Md. Foysal Ahmed
ID: 191-15-12486
Section: O-14

Perform a depth first search on the whole graph.

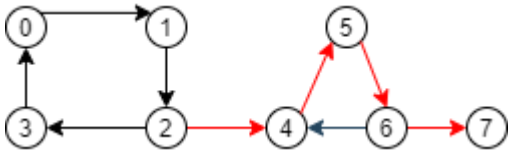
Let us start from vertex-0, visit all of its child vertices, and mark the visited vertices as done. If a vertex leads to an already visited vertex, then push this vertex to the stack.

For example: Starting from vertex-0, go to vertex-1, vertex-2, and then to vertex-3. Vertex-3 leads to already visited vertex-0, so push the source vertex (ie. vertex-3) into the stack.



Visited	0	1	2	3				
Stack	3							

Go to the previous vertex (vertex-2) and visit its child vertices i.e. vertex-4, vertex-5, vertex-6 and vertex-7 sequentially. Since there is nowhere to go from vertex-7, push it into the stack.



Visited	0	1	2	3	4	5	6	7
Stack	3	7						

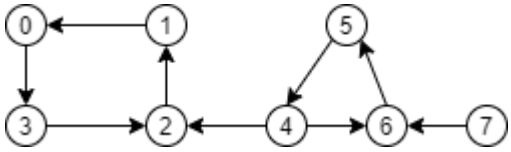
Go to the previous vertex (vertex-6) and visit its child vertices. But, all of its child vertices are visited, so push it into the stack.

Visited	0	1	2	3	4	5	6	7
Stack	3	7	6					

Similarly, a final stack is created.

Visited	0	1	2	3	4	5	6	7
Stack	3	7	6	5	4	2	1	0

2) Reverse the original graph.

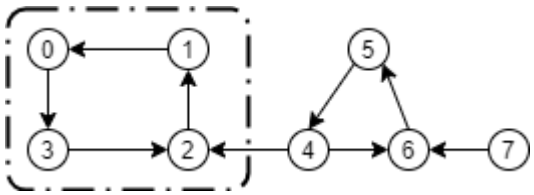


Md. Foysal Ahmed
ID: 191-15-12486
Section: O-14

3) Perform depth-first search on the reversed graph.

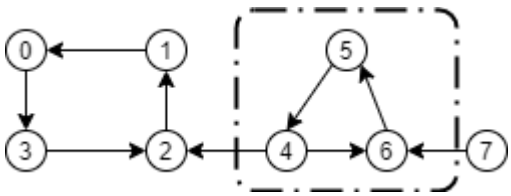
Start from the top vertex of the stack. Traverse through all of its child vertices. Once the already visited vertex is reached, one strongly connected component is formed.

For example: Pop vertex-0 from the stack. Starting from vertex-0, traverse through its child vertices (vertex-0, vertex-1, vertex-2, vertex-3 in sequence) and mark them as visited. The child of vertex-3 is already visited, so these visited vertices form one strongly connected component.

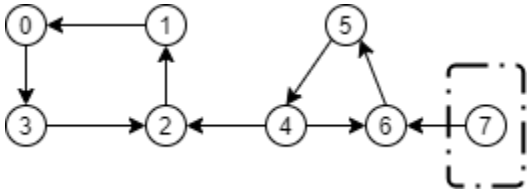


Visited	0	1	2	3				
Stack	3	7	6	5	4	2	1	
SCC	0	1	2	3				

Go to the stack and pop the top vertex if already visited. Otherwise, choose the top vertex from the stack and traverse through its child vertices as presented above.



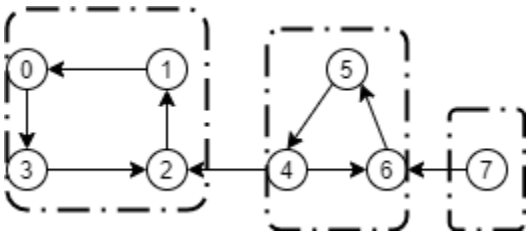
Visited	0	1	2	3	4	5	6	
Stack	3	7	6	5				
SCC	4	5	6					



Md. Foysal Ahmed
ID: 191-15-12486
Section: O-14

visited	0	1	2	3	4	5	6	
Stack								
SCC	7							

4) Thus, the strongly connected components are:



****

[THE END]

Md. Foysal Ahmed
ID: 191-15-12486
Section: O-14