10

Graph Data Structure and Applications

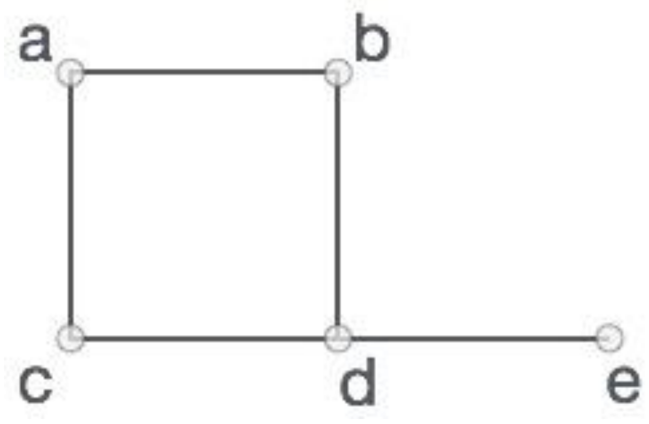
DATA STRUCTURE LAB

SESSION - 10

CSE135 | DAFFODIL INTERNATIONAL UNIVERSITY

Graph:

- → A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links.
- → Interconnected objects are called vertices (vertex)
- → Links are connect the vertices are called edges.

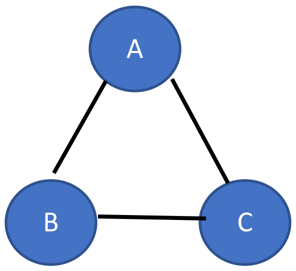


In the above graph,

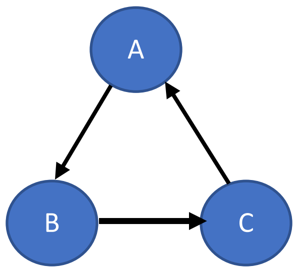
V = {a, b, c, d, e}

E = {ab, ac, bd, cd, de}

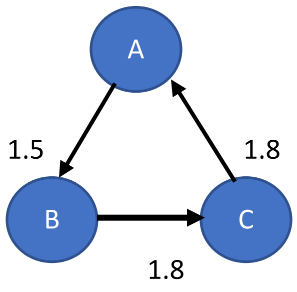
Different Types of Graph:



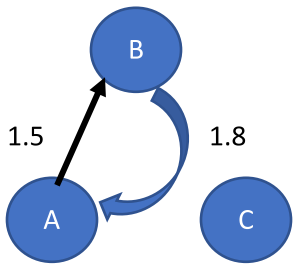
Unweighted connected graph



Unweighted directed cycle connected



Directed connected weighted



Directed weighted disconnected

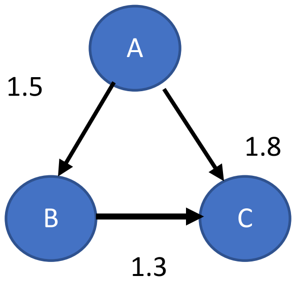
*****

Presenting a Graph in Memory:

- → Adjacency matrix
- → Adjacency list

Adjacency matrix:

- → Based on array

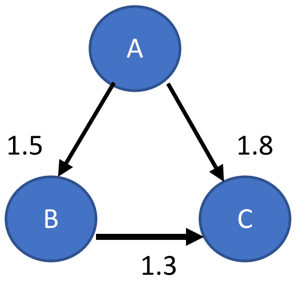


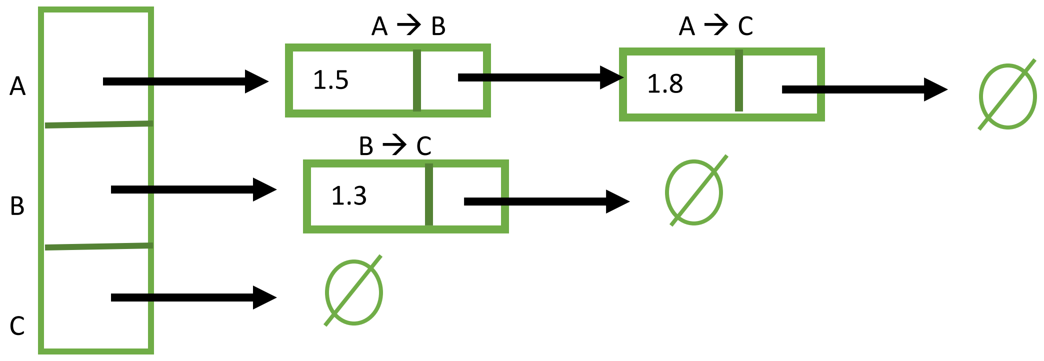
	A	B	C
A		1.5	1.8
B			1.3
C			

```
| 0     1.5     1.8 |
| 0      0       1.3|
| 0      0        0 |
```

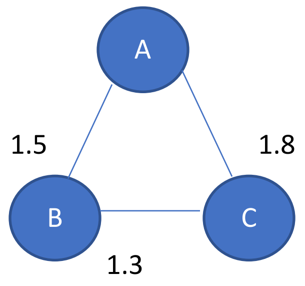
Adjacency list:

- → Using link list



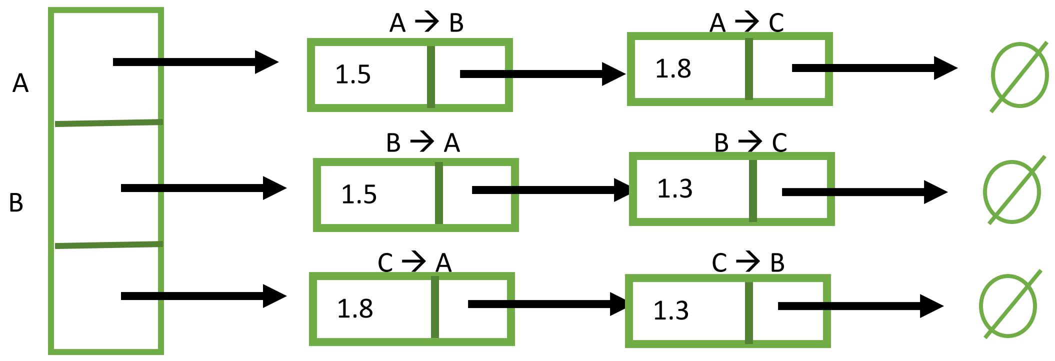


Adjacency List for undirected graph:



	A	B	C
A		1.5	1.8
B	1.5		1.3
C	1.8	1.3	

*Adjacency list for undirected graph is always symmetric.

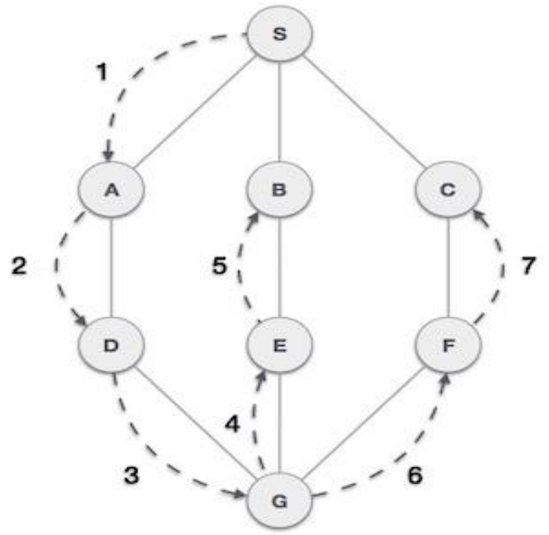


Graph Travarsal:

DFS(depth first search):

In DFS we start from a origin node then we go deeper and deeper until we find an end node. After an end node found we find another path to another end.

Uses a stack to remember to get the next vertex:



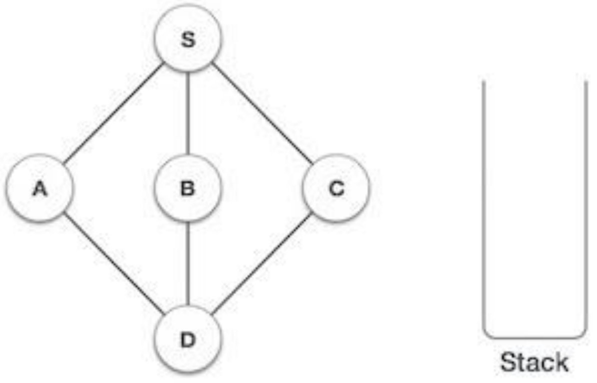
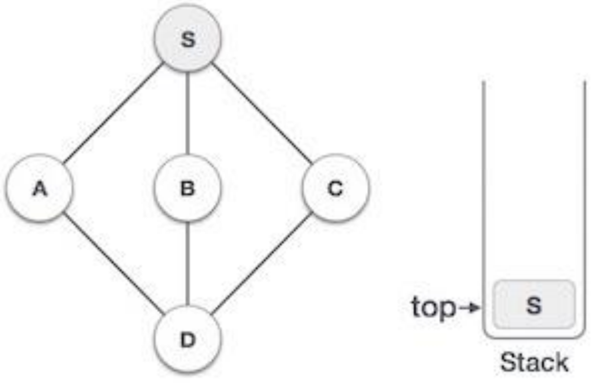
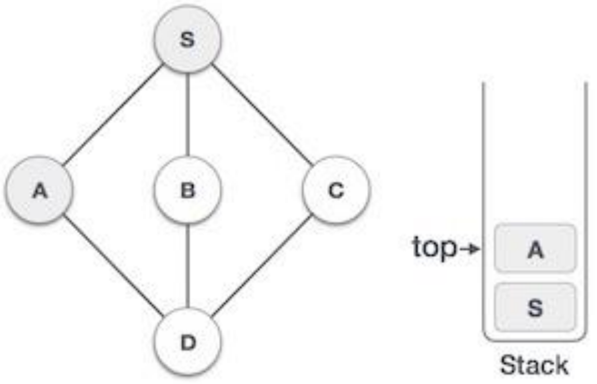
Here, we start from S. then visit A, then D, then G. now we got two way one is to E and other is to F. we visit E, then B. now there are no further unvisited vertex. So we backtrack to G and visit F, then C. now there are no unvisited vertex.

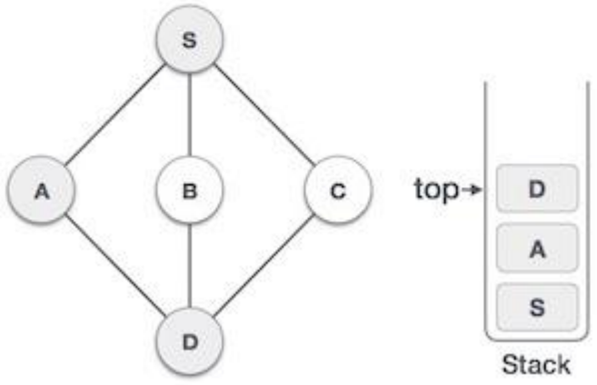
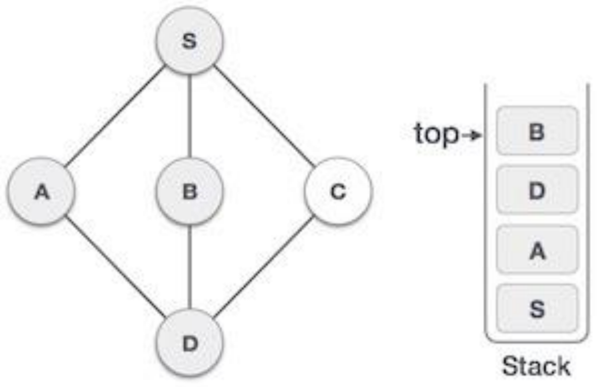
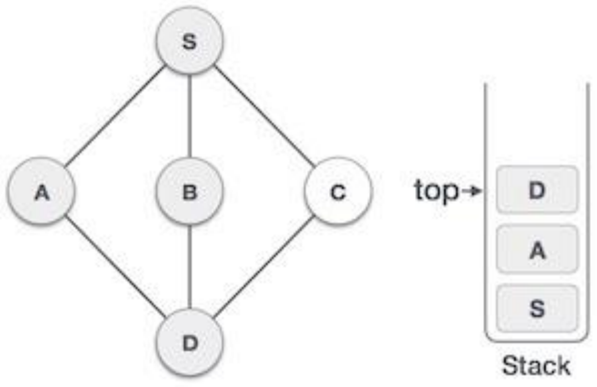
RULES OF TRAVERSAL IN DFS:

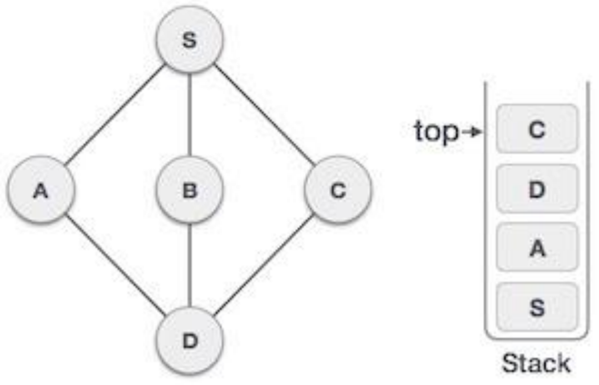
Rule 1 – Start from a vertex then Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.

Rule 2 − If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)

Rule 3 − Repeat Rule 1 and Rule 2 until the stack is empty.

Step	Traversal	Description
1.		Initialize the stack.
2.		Mark **S** as visited and put it onto the stack. Explore any unvisited adjacent node from **S**. We have three nodes and we can pick any of them. For this example, we shall take the node in an alphabetical order.
3.		Mark **A** as visited and put it onto the stack. Explore any unvisited adjacent node from A. Both **S** and **D** are adjacent to **A** but we are concerned for unvisited nodes only.

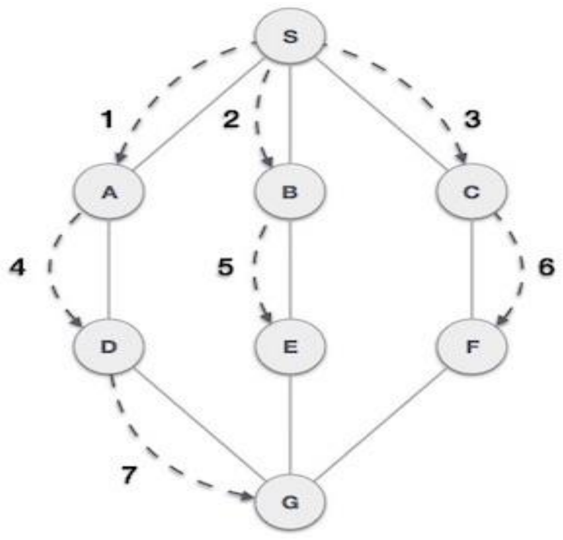
4.		Visit **D** and mark it as visited and put onto the stack. Here, we have **B** and **C** nodes, which are adjacent to **D** and both are unvisited. However, we shall again choose in an alphabetical order.
5.		We choose **B**, mark it as visited and put onto the stack. Here **B** does not have any unvisited adjacent node. So, we pop **B** from the stack.
6.		We check the stack top for return to the previous node and check if it has any unvisited nodes. Here, we find **D** to be on the top of the stack.

7.		Only unvisited adjacent node is from **D** is **C** now. So we visit **C**, mark it as visited and put it onto the stack.

As **C** does not have any unvisited adjacent node so we keep popping the stack until we find a node that has an unvisited adjacent node. In this case, there's none and we keep popping until the stack is empty.

BFS(BREATH FIRST SEARCH):

In BFS firstly we visit each nodes adjacent nodes before visiting their grandchild or further away nodes.



Here we start from S then visit A then backtrack to S then visit B, then again backtrack to S and visit C.

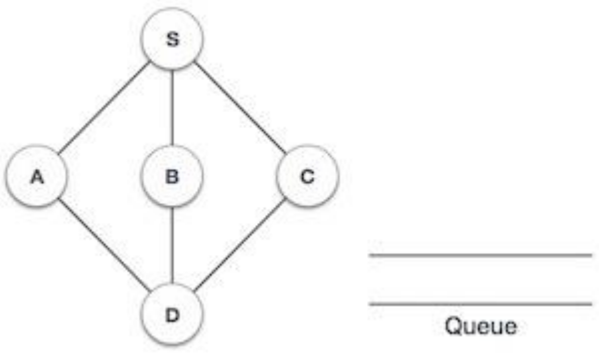
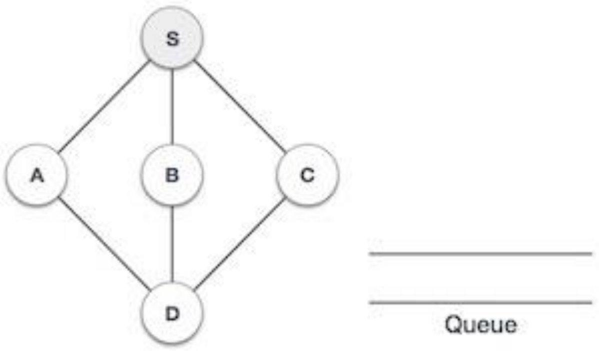
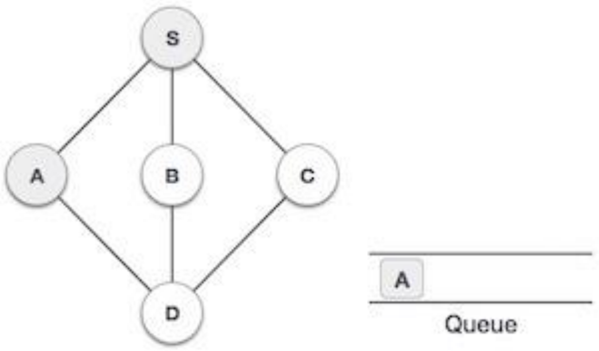
Then we backtrack to A and visit D, after that we backtrack to B and visit E, then we backtrack to C and visit F. after visiting them we again backtrack to D and visit G.

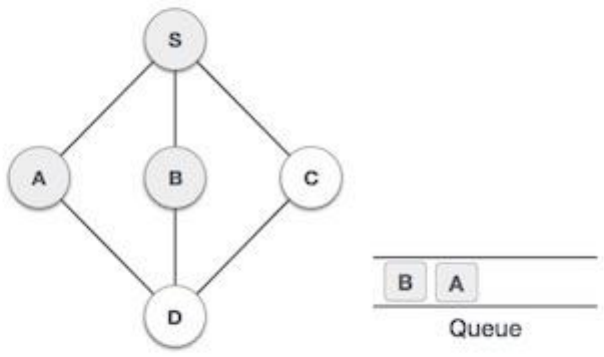
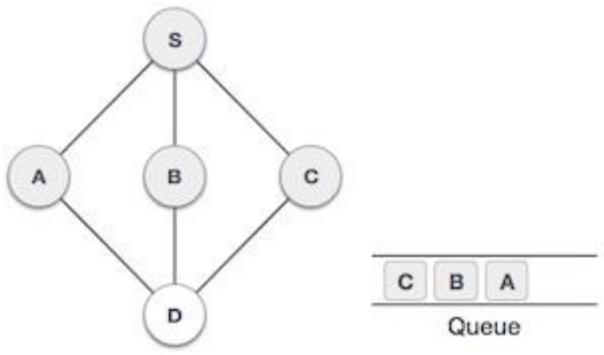
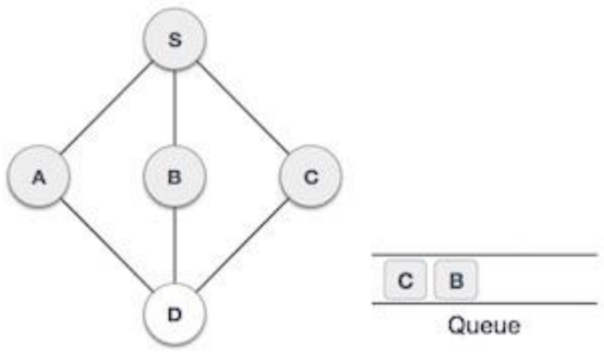
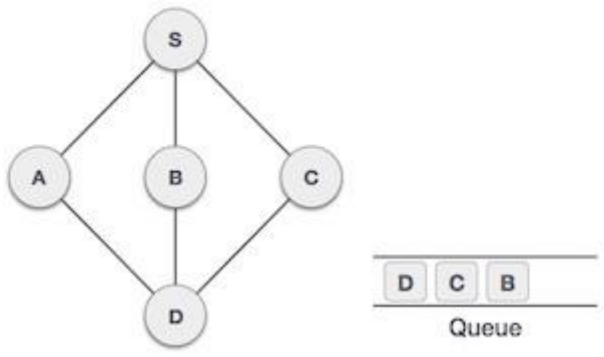
RULES:

Rule 1 – Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.

Rule 2 – If no adjacent vertex is found, remove the first vertex from the queue.

Rule 3 – Repeat Rule 1 and Rule 2 until the queue is empty.

Step	Traversal	Description
1.		Initialize the queue.
2.		We start from visiting **S** (starting node), and mark it as visited.
3.		We then see an unvisited adjacent node from **S**. In this example, we have three nodes but alphabetically we choose **A**, mark it as visited and enqueue it.

4.		Next, the unvisited adjacent node from **S** is **B**. We mark it as visited and enqueue it.
5.		Next, the unvisited adjacent node from **S** is **C**. We mark it as visited and enqueue it.
6.		Now, **S** is left with no unvisited adjacent nodes. So, we dequeue and find **A**.
7.		From **A** we have **D** as unvisited adjacent node. We mark it as visited and enqueue it.

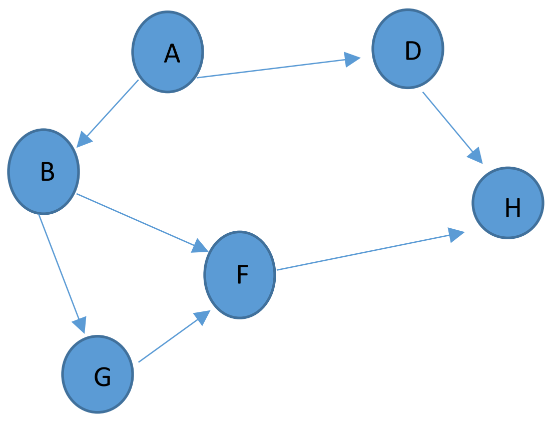
At this stage, there are left with no unmarked (unvisited) nodes. But as per the algorithm we keep on dequeuing in order to get all unvisited nodes. When the queue gets emptied, the program is over.

*****

TOPOLOGICAL ORDERING:

This type of order is based on indegree zero. Here we start with a vertex having indegree 0.

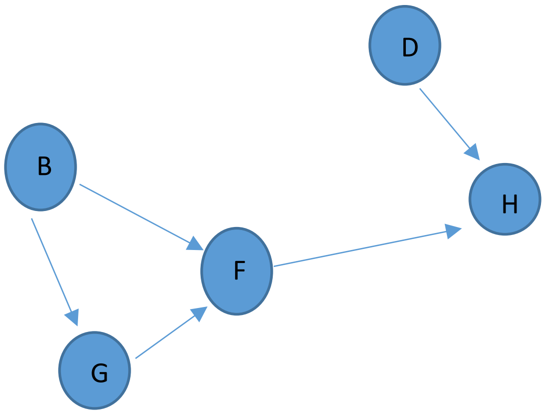
Step 1:



A" has indegree zero. So, we take A and remove it from the graph.

So, {A,

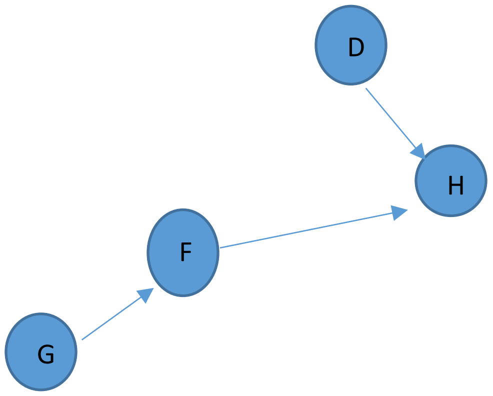
STEP 2:



Here both B&D have indegree zero. So we will take B as B has higher outdegree(2). Then we will remove B from graph.

So, {A,B..

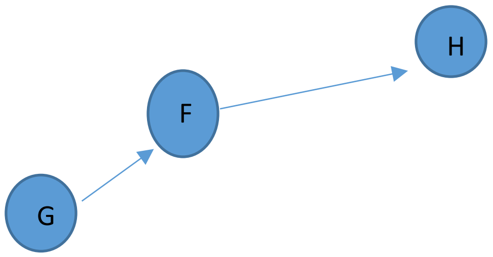
STEP 3:



Here both D&G have indegree zero. we will take D. Then we will remove D from graph.

So, {A,B,D..

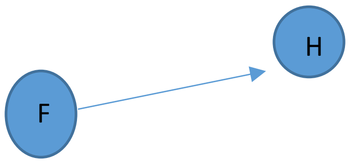
STEP 4:



we will take G. Then we will remove G from graph.

So, { A,B,D,G..

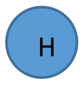
STEP 5:



we will take F. Then we will remove F from graph.

So, { A,B,D,G,F..

STEP 6:



we will take H. Then we will remove H from graph.

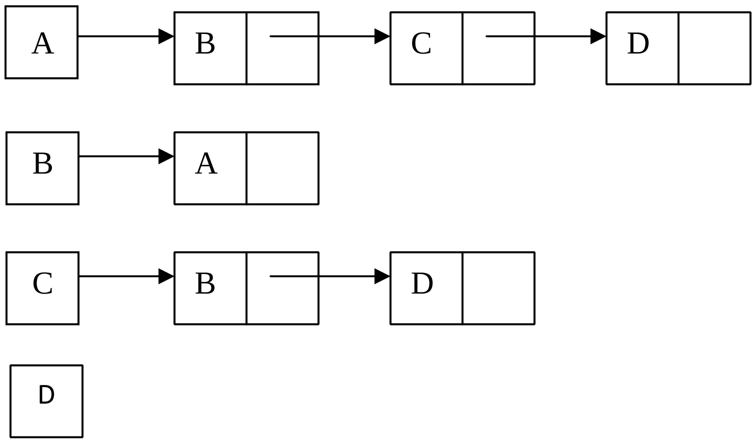
So, { A,B,D,G,F,H}

HERE THE ORDER BECOMES { A, B, D, G, F, H }

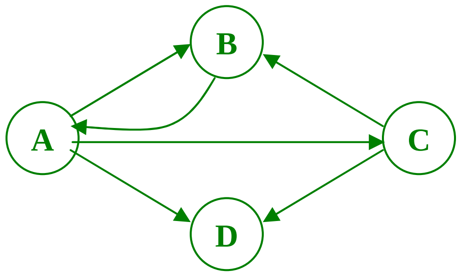
##EXERCISE:

Graph:

1. Here is an adjacency list representation of a *directed* graph where there are no weights assigned to the edges. Draw a picture of the directed graph that has the following adjacency list representation.



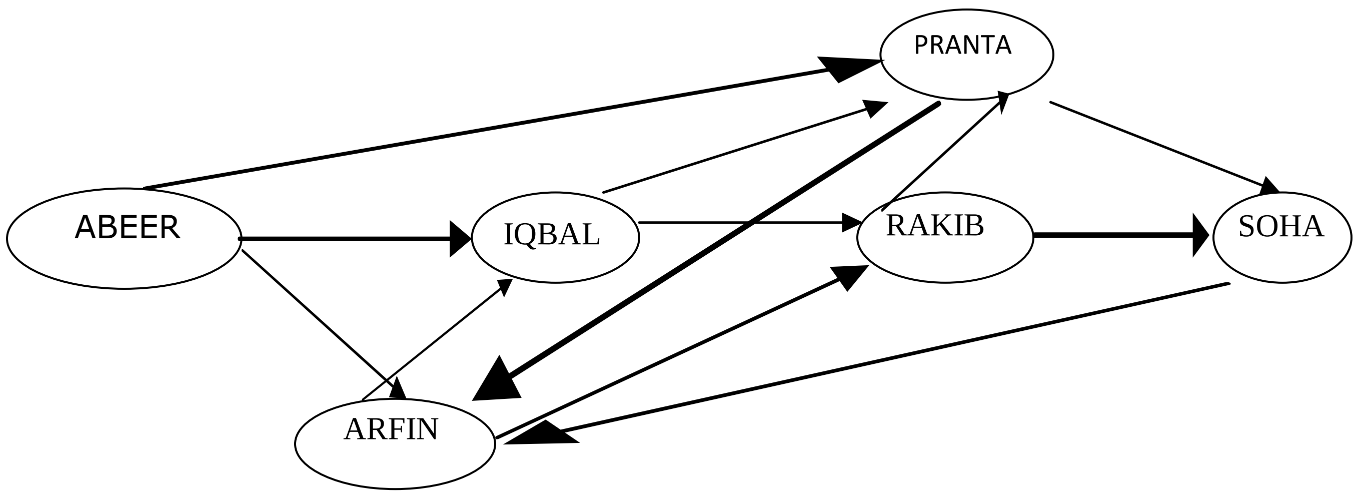
Ans:



2. Another way to represent a graph is an *adjacency matrix*. Draw the adjacency matrix for this graph.

	A	**B**	**C**	**D**
A	**0**	**1**	**1**	**1**
B	**1**	**0**	**0**	**0**
C	**0**	**1**	**0**	**1**
D	**0**	**0**	**0**	**0**

3. Consider the following directed graph. Ans the interesting questions.



a. Calculate indegree & outdegree of each edge of this graph.

Q1. Who is most talkative among them?

Q2. Who is most introvert among them?

Q3. Who is most confusing person among them??

Q4. Which person is mostly loved by others here?

4. Traverse the following graph using BFS & DFS and show them. Then show the topological order if the edges.