DATA STRUCTURES AND ALGORITHM

Module

8

- To understand application of a graph theory
- To apply the concept of Depth First Search and Breadth First Search

GRAPH

GRAPH is a way of representing relationships that exist between pairs of objects.

Graphs Application

- Computer networks
- Electronic circuits
- Engineering
- Geography
- Solving games and puzzles
- Transportation networks

GRAPH

A graph is a set of objects, called **vertices**, together with a collection of pairwise connections between them.

- Any vertex may be connected to any other, these connections are called **edges**.
- *Graph G* is simply a set *V* of ***vertices*** and a collection *E* of pairs of vertices from *V*, called ***edges***

  A graph G = (V,E) is composed of:

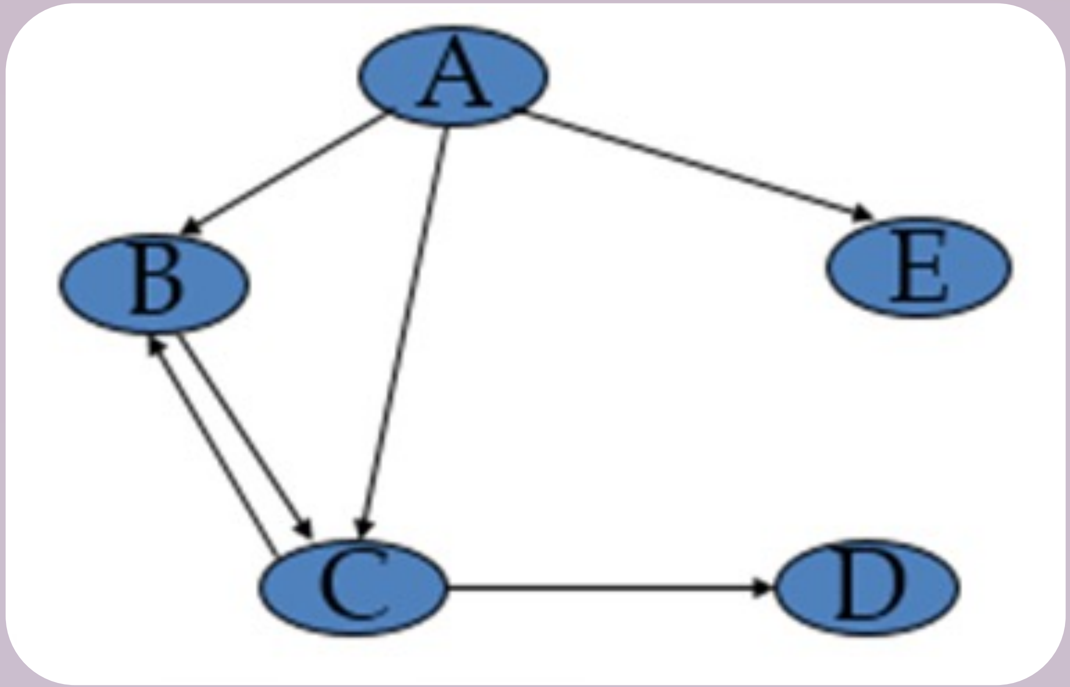
  V → set of vertices

  E → set of edges connecting the vertices in V
- An edge ***e = (u,v)*** is a pair of vertices

GRAPH

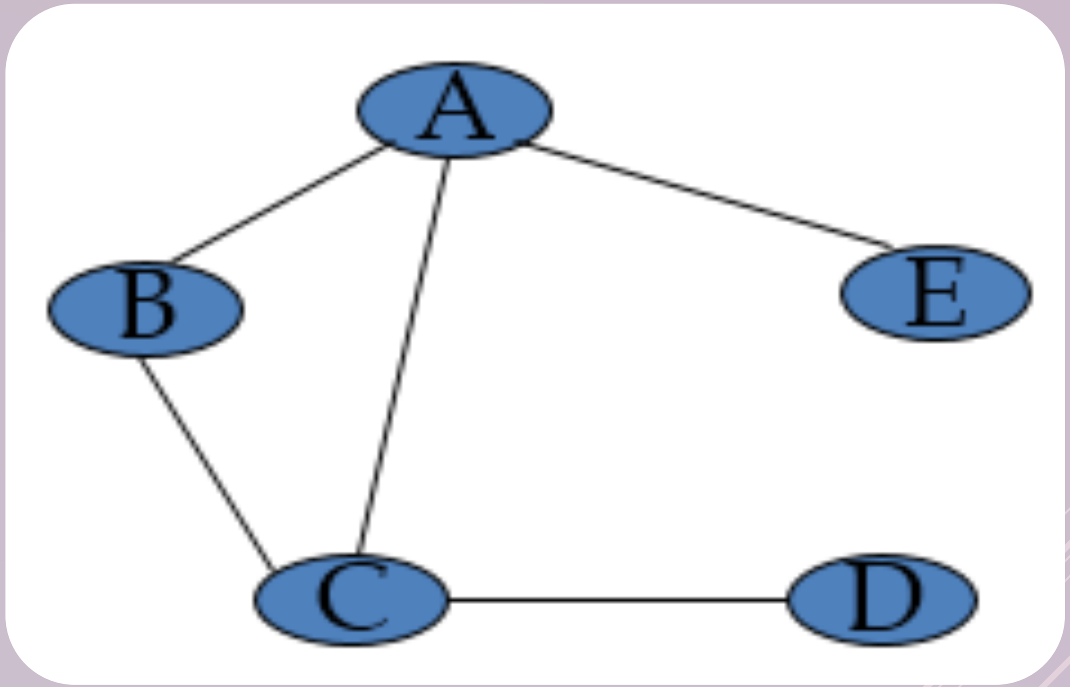
DIRECTED GRAPH	UNDIRECTED GRAPH
▪ An edge (*u*,v) is said to be ***directed*** from *u* to v if the pair (*u*,v) is ordered, with *u* preceding v. ▪ When the edges in a graph have a direction, the graph is called directed graph / **digraph**. ▪ This kind of graph contains ordered pair of vertices	▪ When the edges in a graph have no direction, the graph is called ***undirected*** ▪ An edge (u,v) is said to be undirected if the pair (u,v) is not ordered. ▪ Such edges are undirected because of a symmetric relation;

GRAPH



V = {A, B, C, D, E}
E = {(A,B), (A,C), (A,E), (B,C), (C,B), (C,D)}

DIRECTED GRAPH



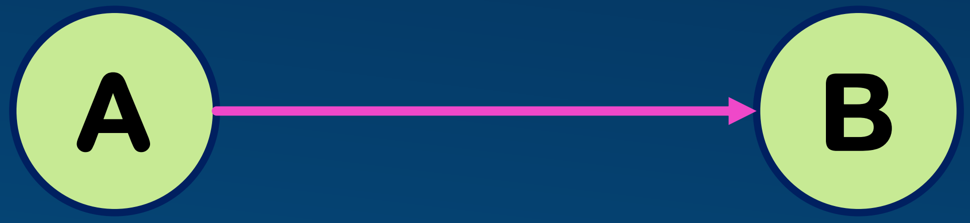
V = {A, B, C, D, E}
E = {(A,B), (A,C), (A,E), (B,A), (B,C), (C,A),(C,B), (C,D), (D,C), (E,A) }

UNDIRECTED GRAPH

GRAPH

BASIC TERMINOLOGIES

- If an edge is directed, its first endpoint is its ***origin*** and the other is the ***destination*** of the edge.



origin / initial vertex

destination/ Terminal vertex

GRAPH

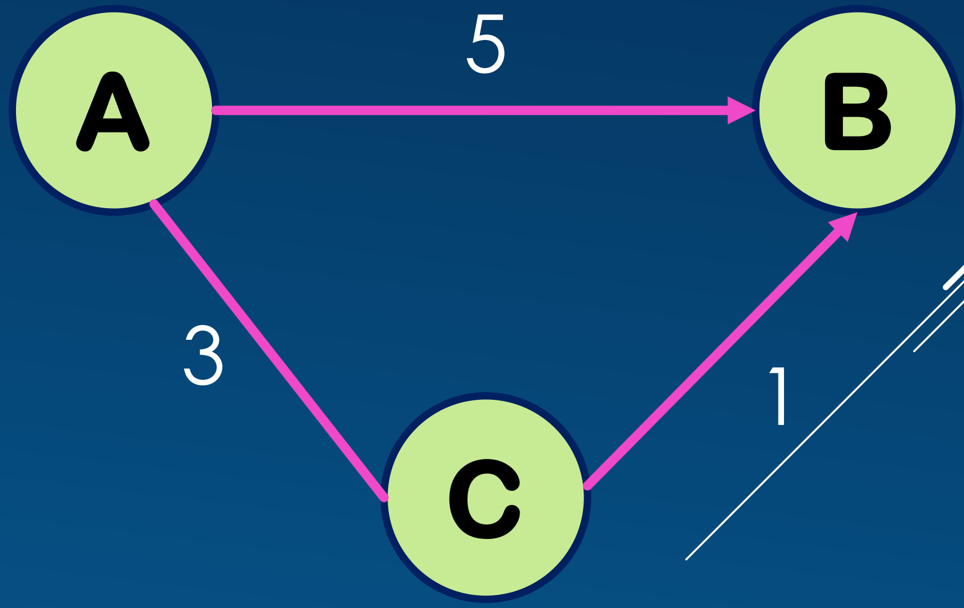
BASIC TERMINOLOGIES

- **Weighted Graph**

  Every edge is assigned some value which is greater than or equal to zero.

  Examples:

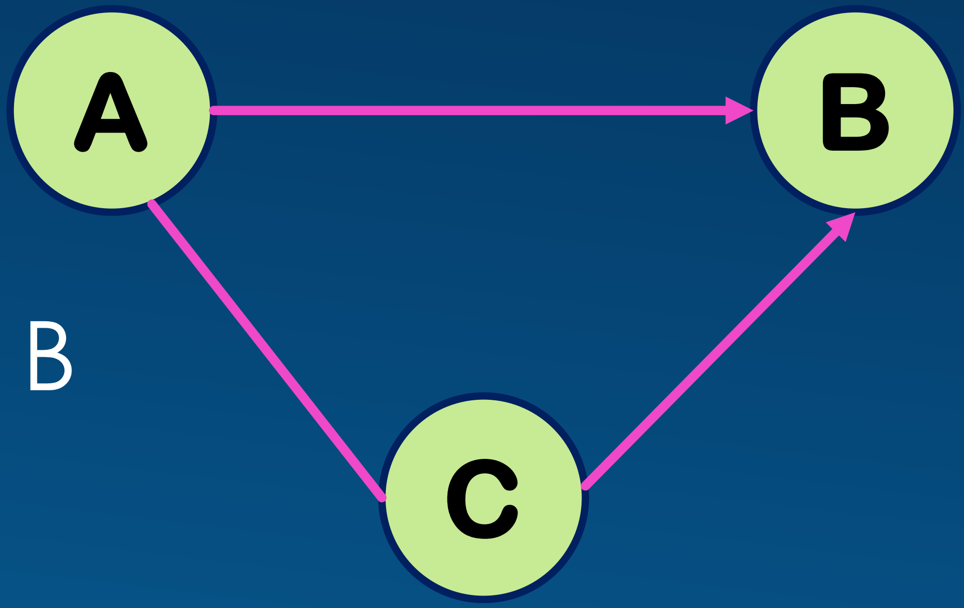
  - ✓ distance
  - ✓ time
  - ✓ cost
  - ✓ capacity



GRAPH

BASIC TERMINOLOGIES

- **Path** in a graph is a series of edges, none repeated, that can be traversed in order to travel from one vertex to another in a graph.



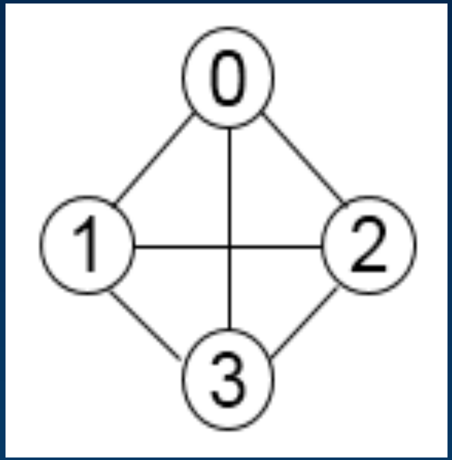
A → C → B
A → B

GRAPH

BASIC TERMINOLOGIES

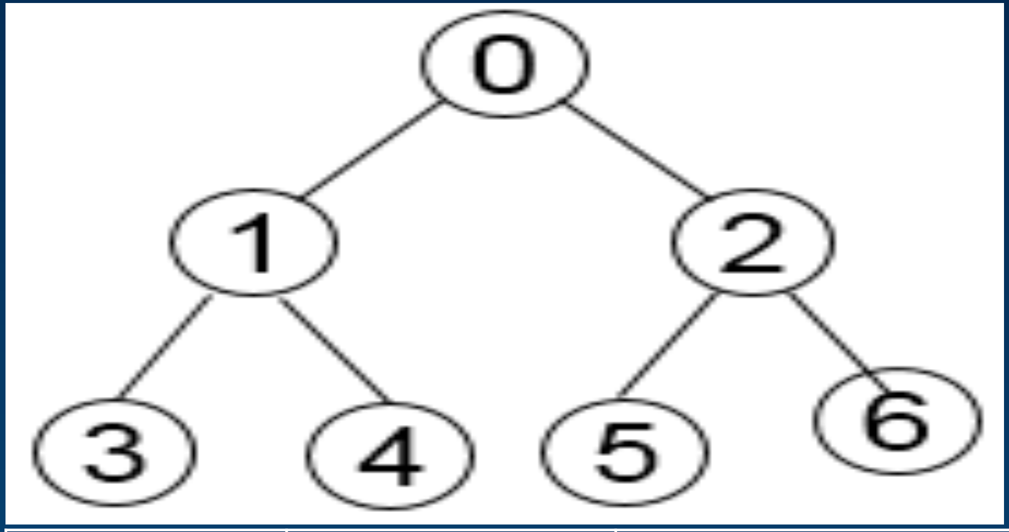
- **Degree** of a vertex v, denoted deg(v), is the number of incident edges of v.

The ***in-degree*** and ***out-degree*** of a vertex v are the number of the incoming and outgoing edges of v, and are denoted indeg(v) and outdeg(v), respectively

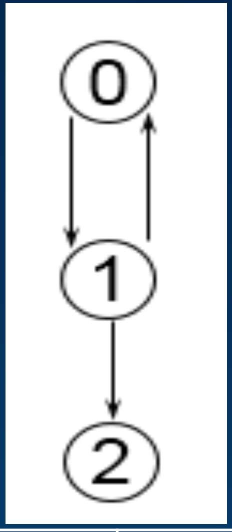


Vertices	In-Degree	Out-Degree
0	3	3
1	3	3
2	3	3
3	3	3

BINARY SEARCH TREE (BST) acyclic



Vertices	In-Degree	Out-Degree
0	2	2
1	3	3
2	3	3
3	1	1
4	1	1
5	1	1
6	1	1

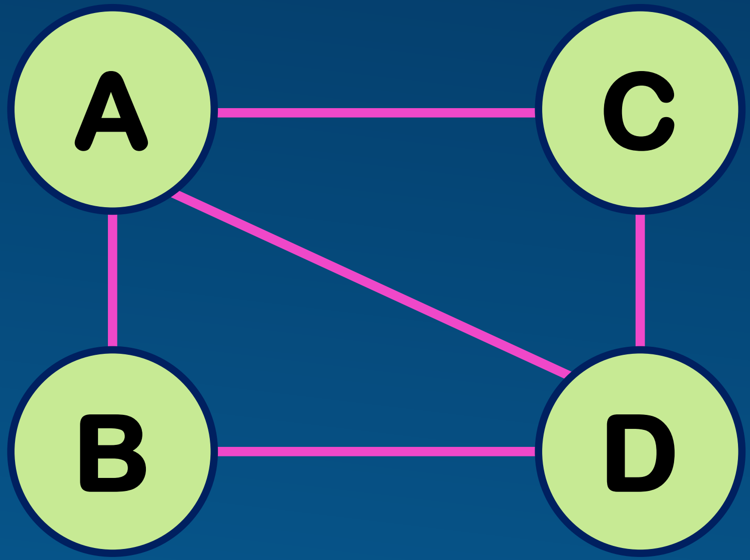


Vertices	In-Degree	Out-Degree
0	1	1
1	1	2
2	1	0

GRAPH

BASIC TERMINOLOGIES

- **Acyclic Graph** A graph without cycle
  Example: tree
- **Cycle** is a path with at least one edge that has the same start and end vertices.



A → B → D → C → A

C → A → D → C

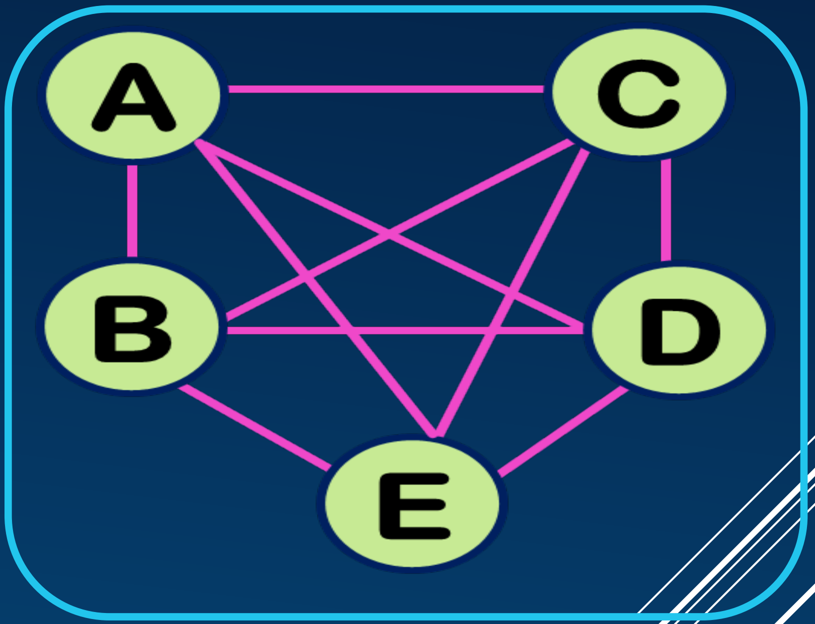
GRAPH

BASIC TERMINOLOGIES

- **Complete Graph**
  - A graph in which all pairs of vertices are adjacent
  - A graph in which every vertex is directly connected to every other vertex

    Let **n** = Number of vertices, and

    m = Number of edges
  - For a complete graph with *n* vertices, the number of edges is **n(n – 1)/2**.



n = 5

n = [n (n-1)] / 2

= [5 (5-1)] / 2

= [5 * 4] / 2

= 10

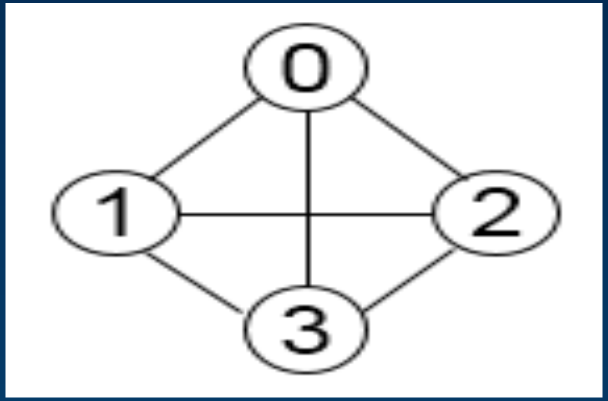
GRAPH

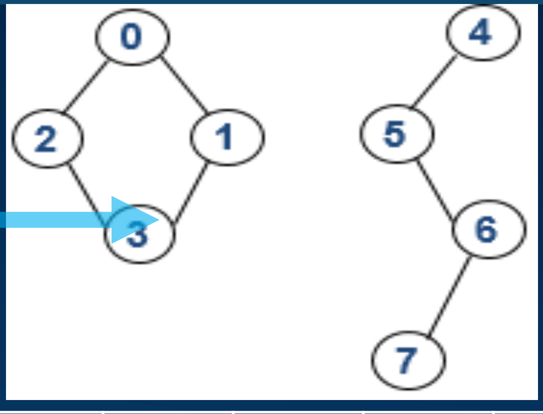
GRAPH REPRESENTATION

Adjacency Matrix (Array Implementation)

- Let G=(V,E) be a graph with n vertices.
- The adjacency matrix of G is a two-dimensional n by n array, adj_mat
  If the edge (vi, vj) is in E(G), adj_mat[i][j]=1
  Else adj_mat[i][j]=0
- The adjacency matrix for an undirected graph is symmetric; since adj_mat[i][j]=adj_mat[j][i]
- The adjacency matrix for a digraph may not be symmetric

Adjacency Matrix (Array Implementation)





0 (doesn’t have connection)

1 (have connection)

V	0	1	2	3
0				
1				
2				
3				

V	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4								
5								
6								
7								

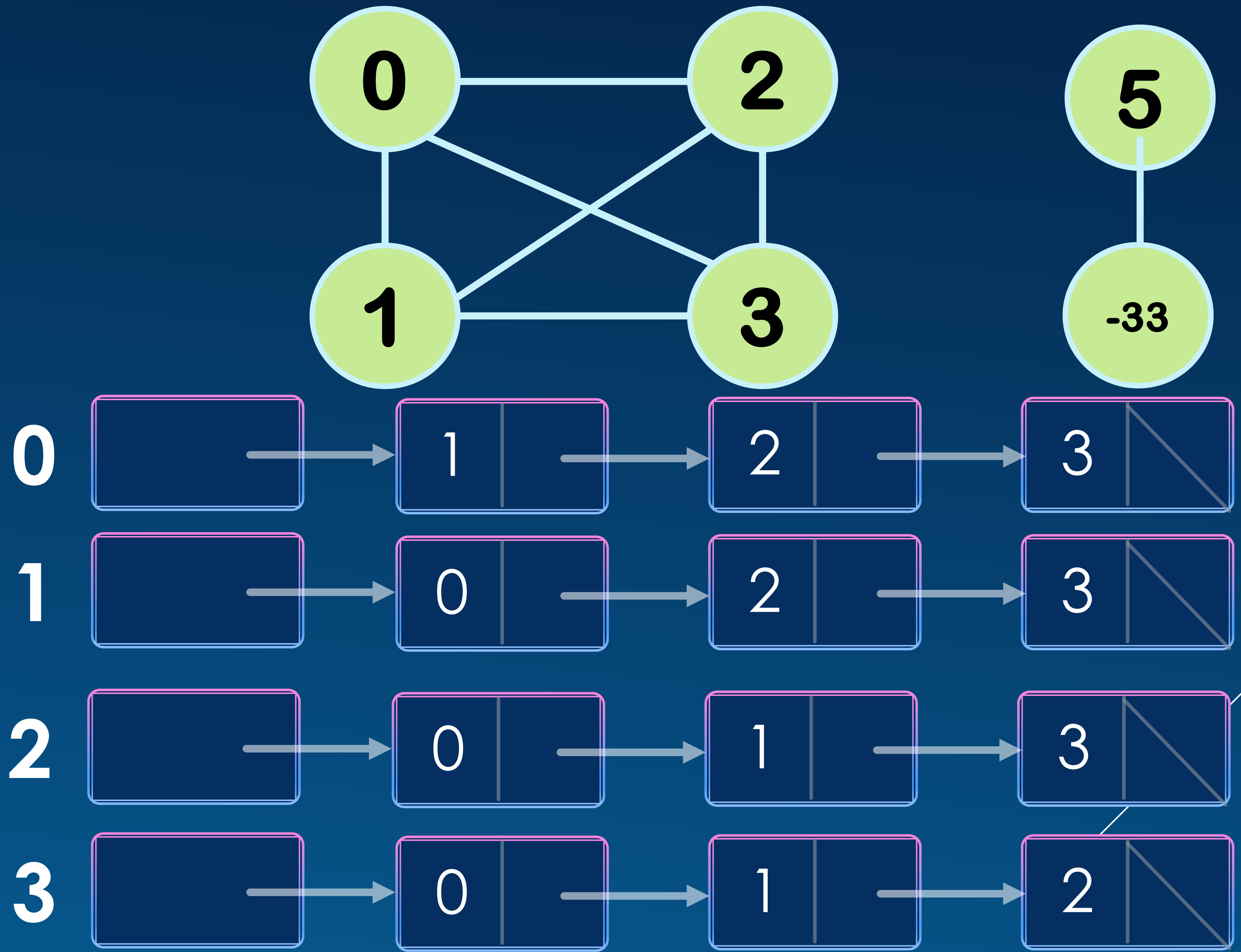
GRAPH

GRAPH REPRESENTATION

Adjacency List

- A single dimension array of structure is used to represent the vertices
- A Linked list is used for each vertex V which contains the vertices which are adjacent from V (adjacency list)

ADJACENCY LIST



BREADTH FIRST SEARCH (BFS)

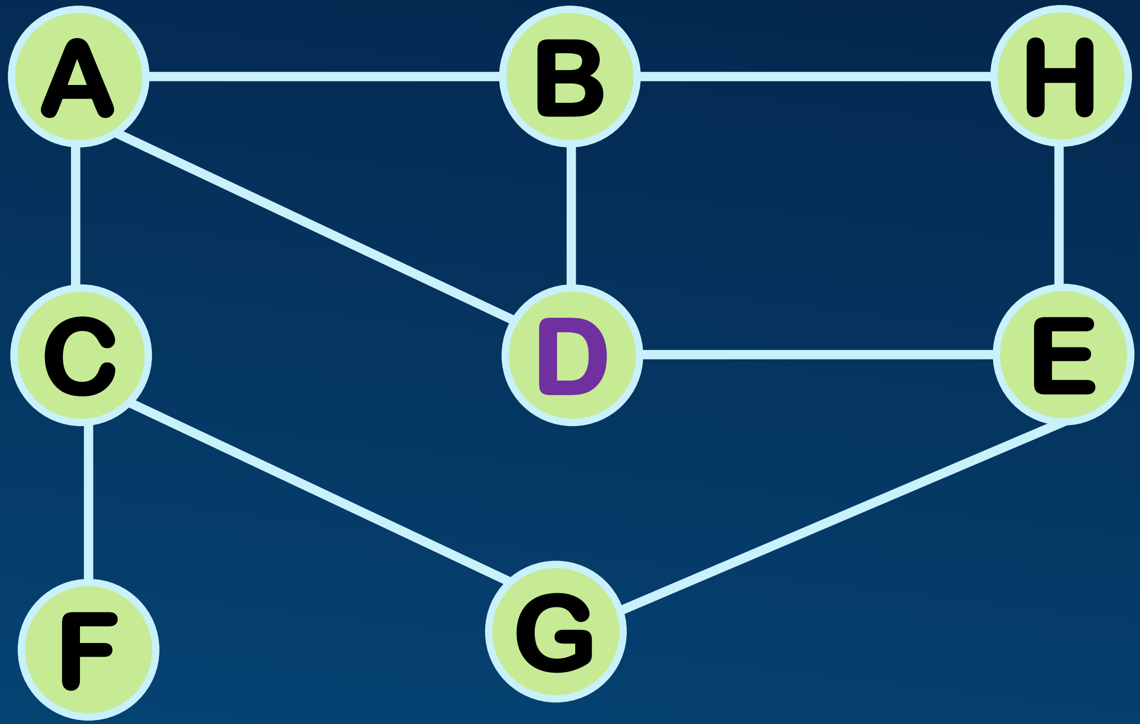
- This method visits all the vertices, beginning with a specified **start vertex**. It can be described roughly as "**neighbours-first**".
- No vertex is visited more than once, and vertices are visited only if they can be reached – that is, if there is a path from the start vertex.
- Use of a **queue data structure** (first-in first-out structure)
- Neighbours are not added to the queue if they are already in the queue, or have already been visited.

BREADTH FIRST SEARCH (BFS)

BREADTH-FIRST SEARCHING ALGORITHM:

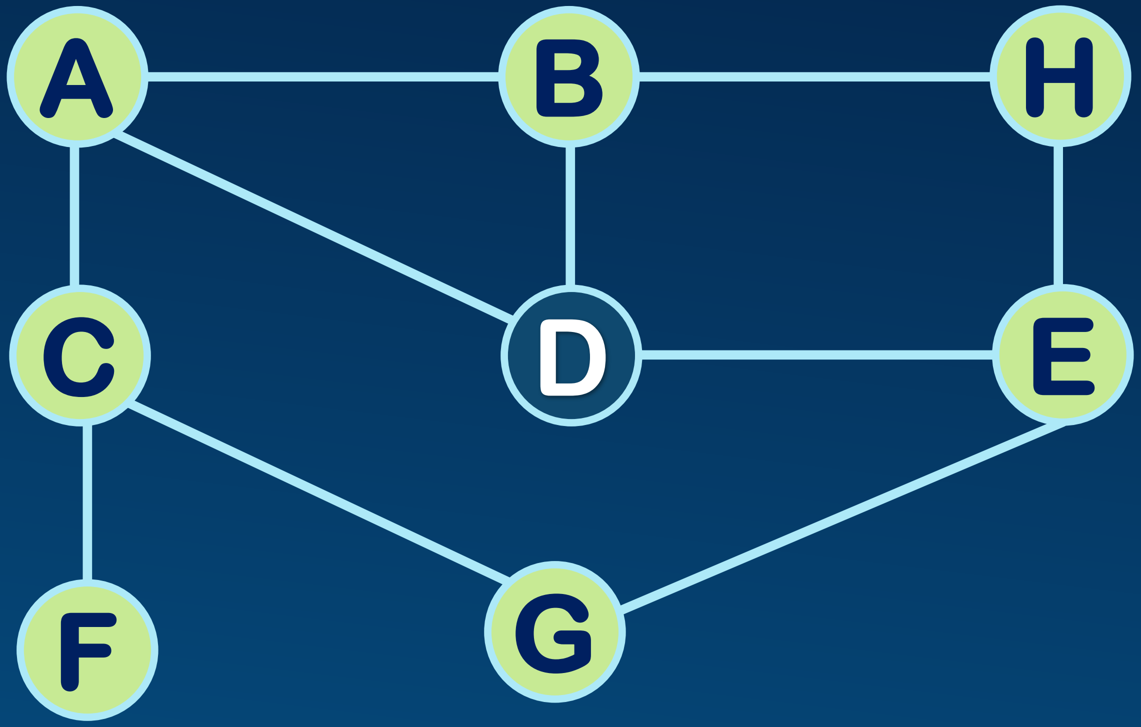
1. Push the starting vertex into a queue and then start a loop that will execute while the queue is not empty.
2. Once inside the loop, pop a vertex from the queue and make it the current vertex.
3. Place all unchecked vertices adjacent to the current vertex onto the queue and mark them as checked.
4. If there are no more vertices adjacent to the current vertex, check if the current vertex is the destination.
5. If the destination is found, the algorithm is done.
6. If the algorithm did not find the destination, repeat steps 2 through 5.

BREADTH FIRST SEARCH (BFS)



Start at Vertex D

BREADTH FIRST SEARCH (BFS) A-Z



D

A B C E F G H

Visit order:__________________________

DEPTH FIRST SEARCH (DFS)

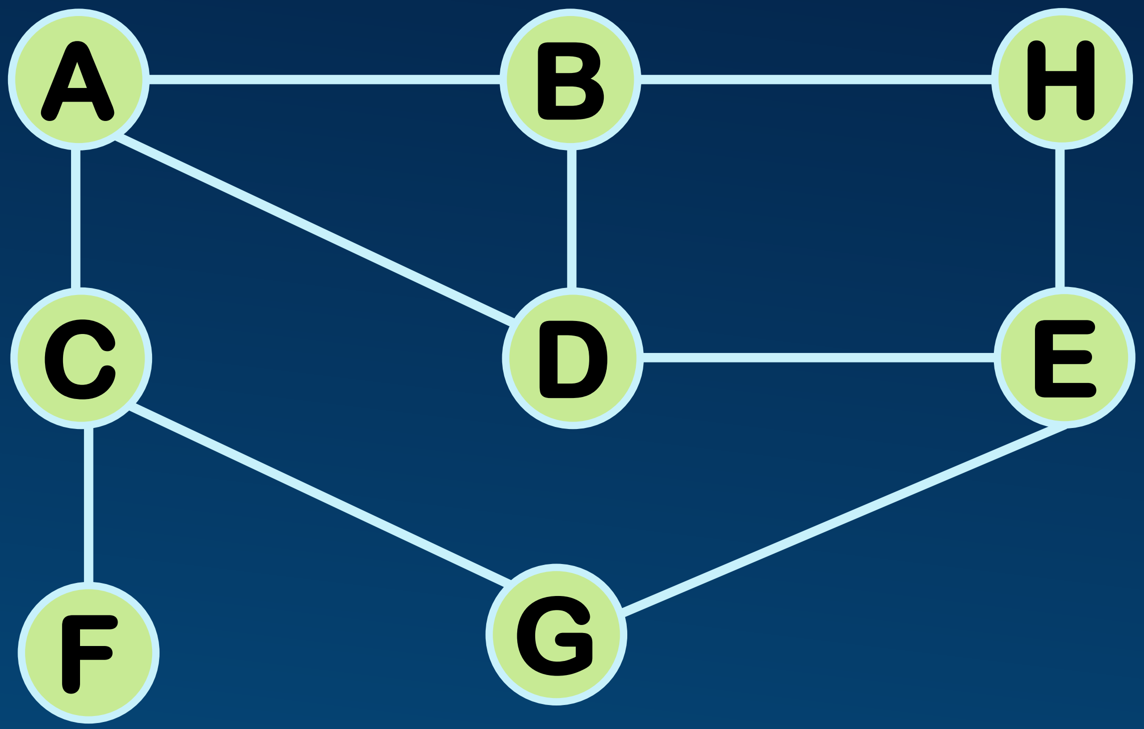
- This method visits all the vertices, beginning with a specified **start vertex**.
- This strategy proceeds along a path from vertex V as deeply into the graph as possible.
- This means that after visiting V, the algorithm tries to visit any unvisited vertex adjacent to V. When the traversal reaches a vertex which has no adjacent vertex, it back tracks and visits an unvisited adjacent vertex.
- Use of a Stack data structure.

DEPTH FIRST SEARCH (DFS)

Depth-first traversal of a graph:

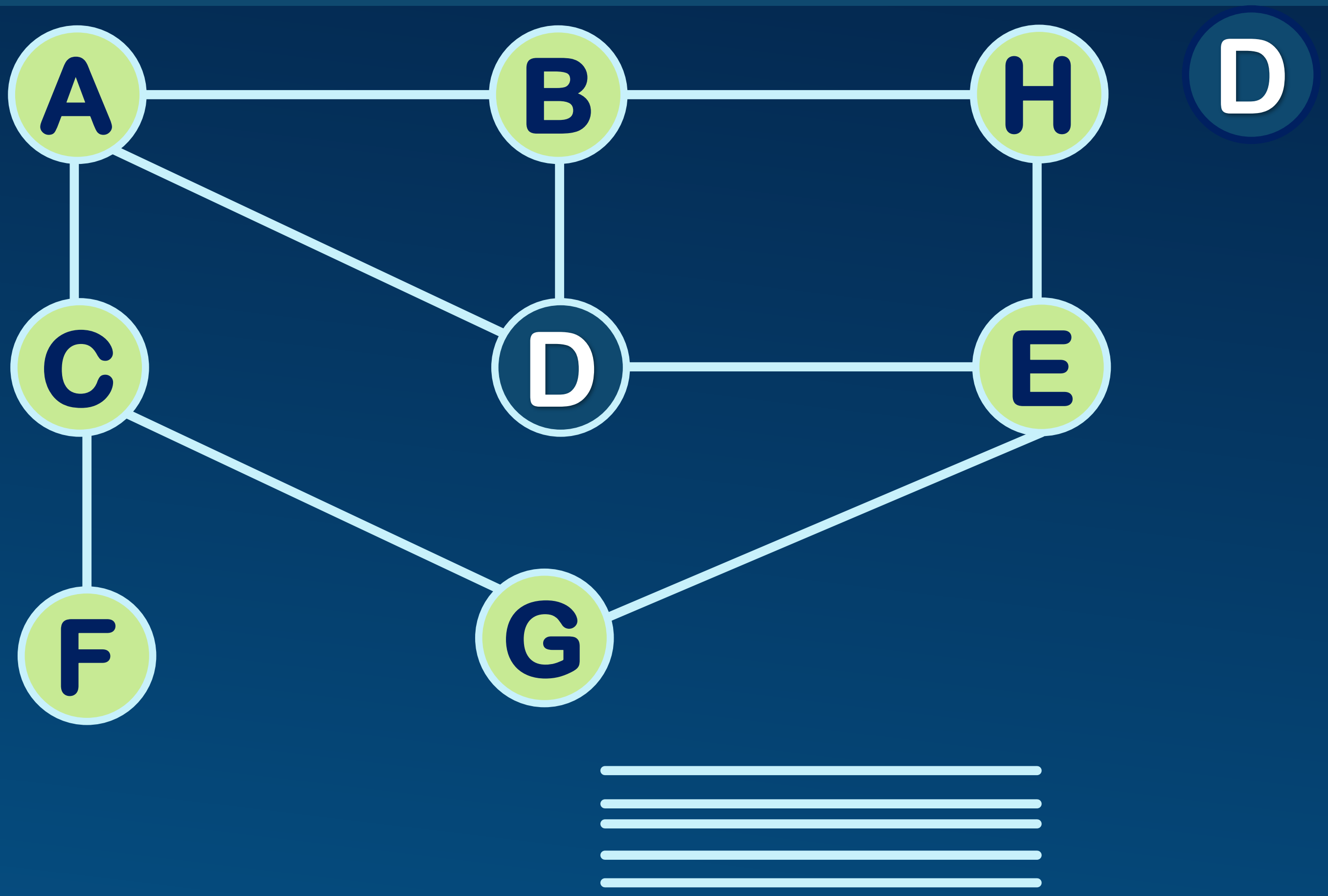
1. Start the traversal from an arbitrary vertex;
2. Apply depth-first search;
3. When the search terminates, backtrack to the previous vertex of the finishing point,
4. Repeat depth-first search on other adjacent vertices, then backtrack to one level up.
5. Continue the process until all the vertices that are reachable from the starting vertex are visited.
6. Repeat above processes until all vertices are visited.

DEPTH FIRST SEARCH (DFS)



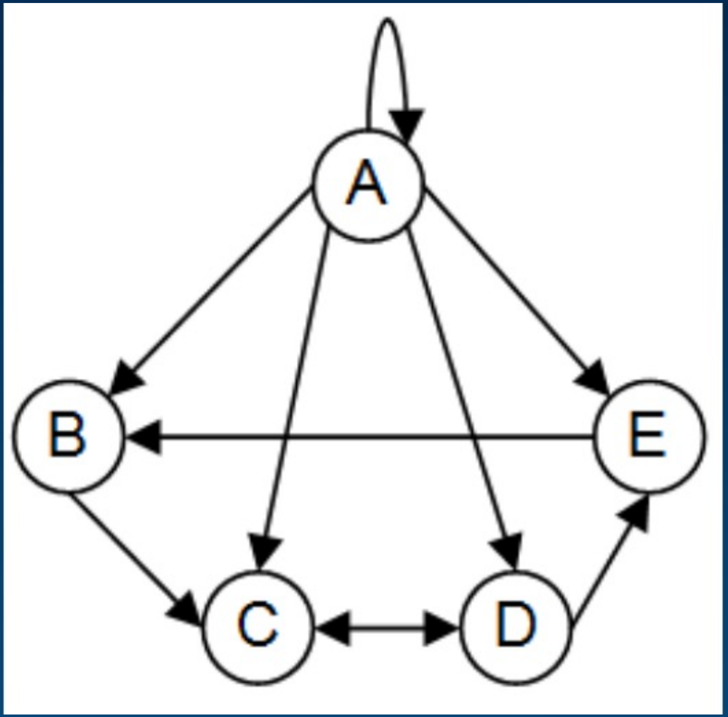
Start at Vertex D

DEPTH FIRST SEARCH (DFS) A-Z



Visit order:

EXAMPLE

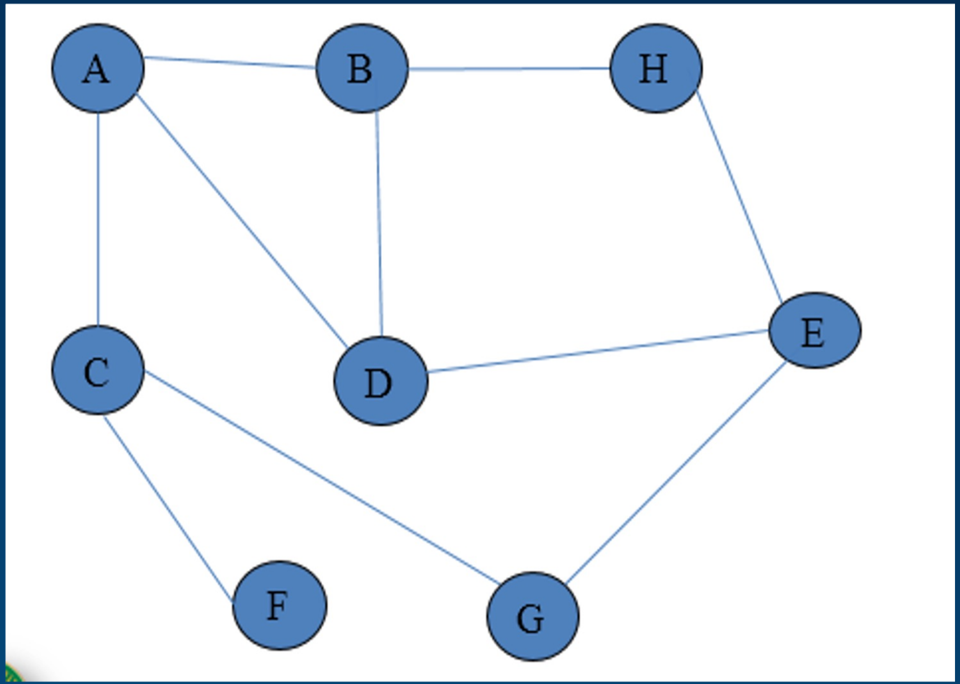


A B C D E

BFS order:________________________

DFS order:________________________

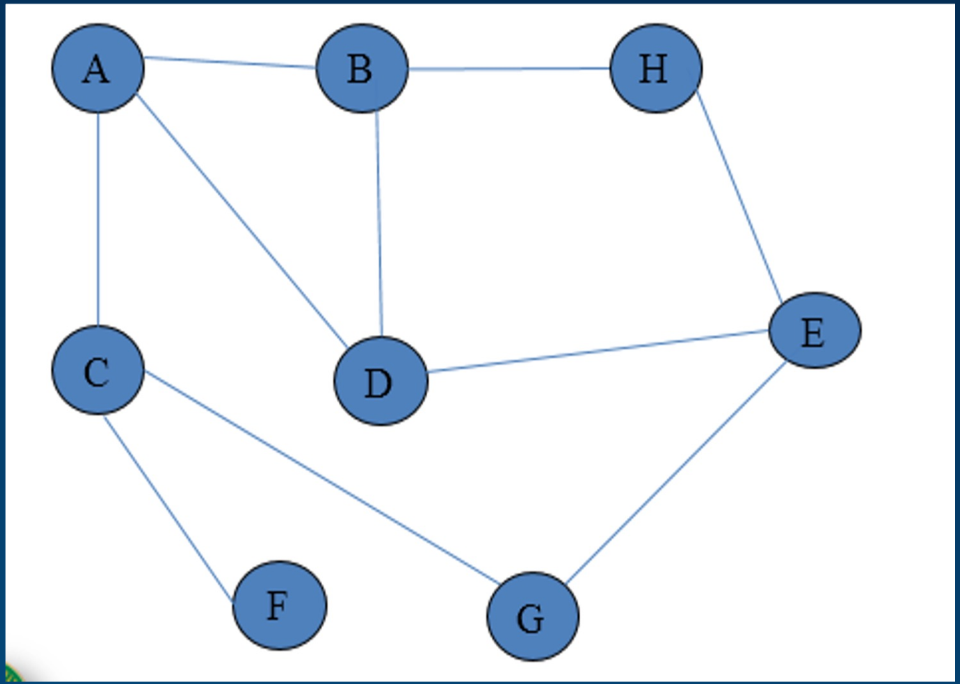
BFS and DFS



BFS

A B C D E F G H

BFS and DFS



DFS

A B C D E F G H