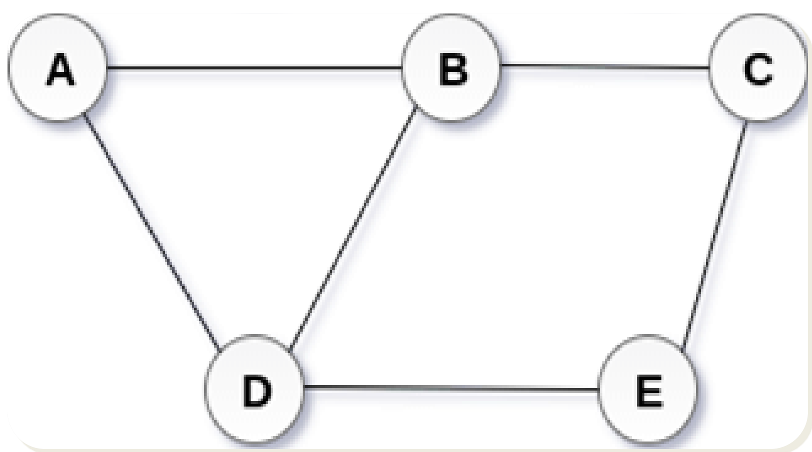
Chapter 6

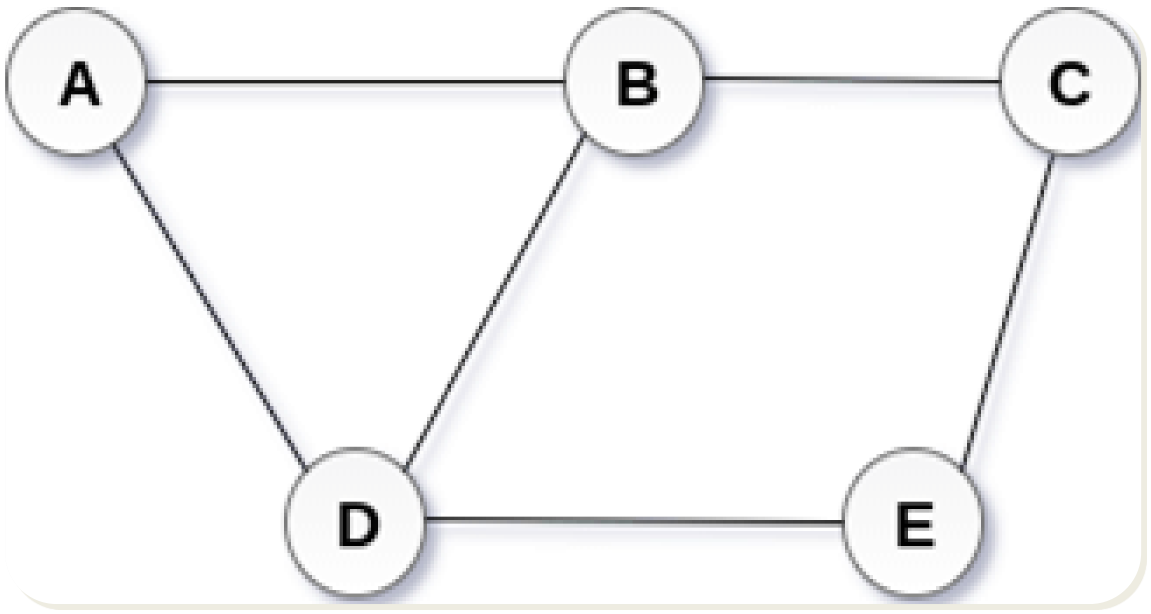
Graph and its Application

Introduction t o Graphs

- Graph is a non-linear data structure. It contains a set of points known as nodes (or vertices) and a set of links known as edges (or Arcs). Here edges are used to connect the vertices.
- Graph is a collection of vertices and arcs in which vertices are connected with arcs
- Generally, a graph G is represented as G = (V , E), where V is set of vertices and E is set of edges.



- A Graph G(V, E) with 5 vertices (A, B, C, D, E) and six edges ((A,B), (B,C), (C,E), (E,D), (D,B), (D,A)) is shown in the following figure.



Graph Terminology

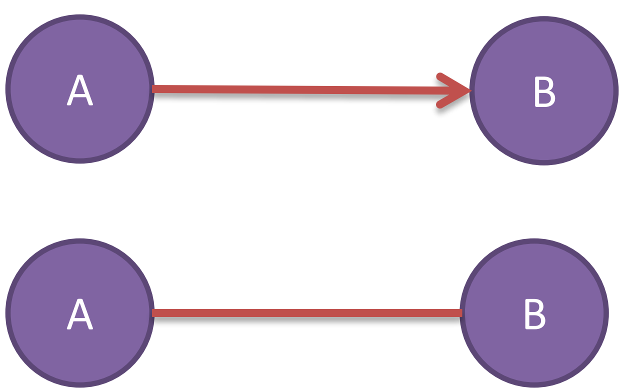
Vertex

- Individual data element of a graph is called as Vertex. Vertex is also known as node. In above example graph, A, B, C, D & E are known as vertices.

Edge

- An edge is a connecting link between two vertices. Edge is also known as Arc.
- An edge is represented as (starting Vertex, ending Vertex).

Example

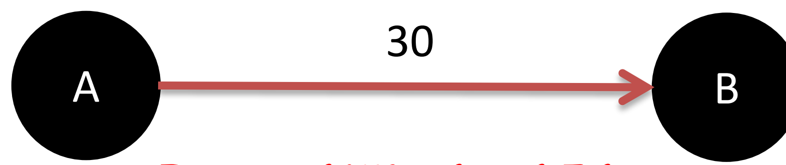


Graph Terminology (2)

Edges are three types

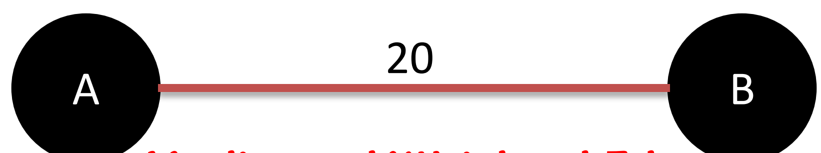
- **Undirected Edge** - An undirected egde is a bidirectional edge. If there is undirected edge between vertices A and B then edge (A, B) is equal to edge (B , A).
- **Directed Edge** - A directed egde is a unidirectional edge. If there is directed edge between vertices A and B then edge (A , B) is not equal to edge (B , A).
- **Weighted Edge** - A weighted egde is a edge with value (cost) on it.

Directed Edge



Directed Weighted Edge

Undirected Edge



Undirected Weighted Edge

Graph Terminology (3)

Origin

- If a edge is directed, its first endpoint is said to be the origin of it.

Destination

- If a edge is directed, its first endpoint is said to be the origin of it and the other endpoint is said to be the destination of that edge.

Graph Terminology (4)

Adjacent

- If there is an edge between vertices A and B then both A and B are said to be adjacent. In other words, vertices A and B are said to be adjacent if there is an edge between them.

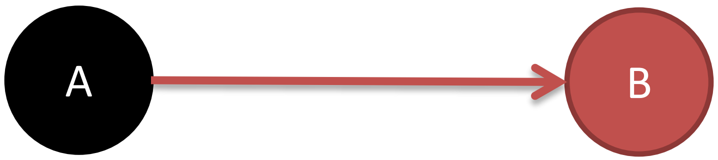
Incident

- an edge that is connected to a particular vertex within a graph.
- For instance, if you have a graph with vertices A, B, C, and an edge connecting vertices A and B, then we can say that this edge is incident to both vertices A and B.

Graph Terminology (5)

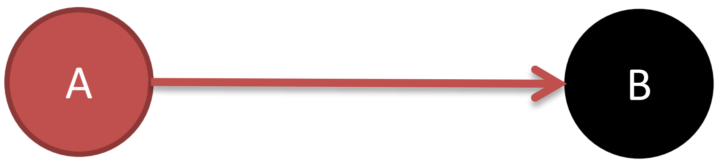
Incoming Edge

- A directed edge is said to be incoming edge on its destination vertex.



Outgoing Edge

- A directed edge is said to be outgoing edge on its origin vertex.



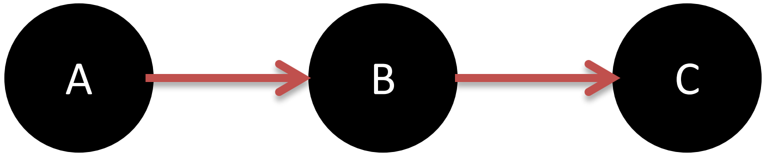
Degree

- Degree in Undirected Graphs: Total number of edges connected to it.
- Degree in Directed Graph: divided into indegree and outdegree

Graph Terminology (6)

Indegree

- Total number of incoming edges that are connected to a particular node within a directed graph



Outdegree

- Total number of outgoing edges connected to a specific node within a directed graph.

Parallel edges or Multiple edges

- If there are two undirected edges with same end vertices
- two directed edges with same origin and destination, such edges are called parallel edges or multiple edges.

Graph Terminology (7)

Path

- A path is a sequence of alternate vertices and edges that starts at a vertex and ends at other vertex.

Self-loop

- A self-loop in a graph occurs when an edge starts and ends on the same vertex (node).

Simple Graph

- A graph is said to be simple if there are no parallel and self-loop edges.

Graph Terminology (8)

Cycle

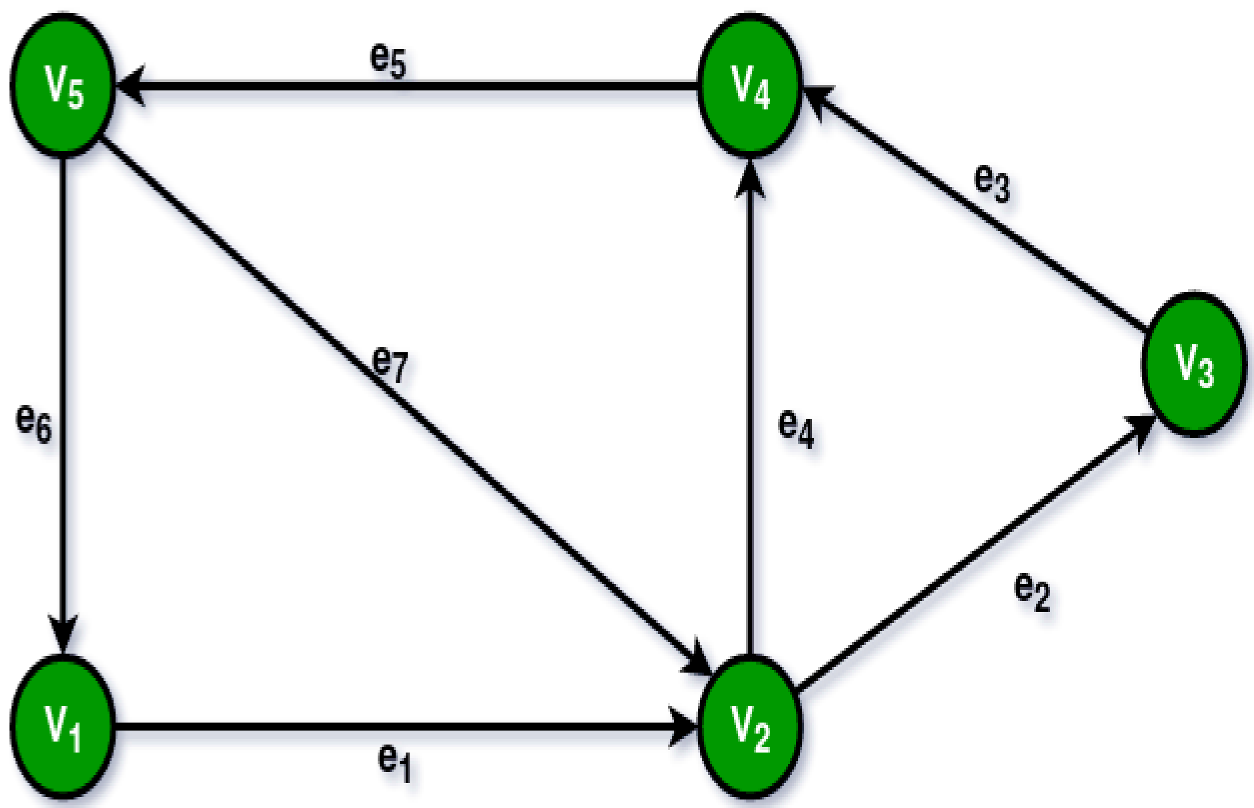
- A cycle in a graph is a sequence of vertices and edges that starts and ends at the same vertex, without passing through any other vertex more than once except for the starting and ending vertex

Connected Graph

- A connected graph is a type of graph in which there exists a path between every pair of vertices within the graph. In simpler terms, it means that every vertex in the graph is somehow reachable from every other vertex by following edges (directed or undirected).

Vertex
Edge
Origin
Destination
Adjacent
Outgoing Edge
Incoming Edge
Degree
Indegree
Outdegree
Path
Cycle
Connected Graph

Exercise



Graph representation

- Graph representation means the technique to be used to store some graph into the computer's memory.
- A graph is a data structure that consist a sets of vertices (called nodes) and edges. There are Three ways to store Graphs into the computer's memory:
  - ✓ Sequential representation (Adjacency matrix representation)
  - ✓ Linked list representation (or, Adjacency list representation)
  - ✓ Incidence Matrix

Sequential representation

- If an Undirected Graph G consists of n vertices, then the adjacency matrix for that graph is n x n, and the matrix A = [aij] can be defined as -

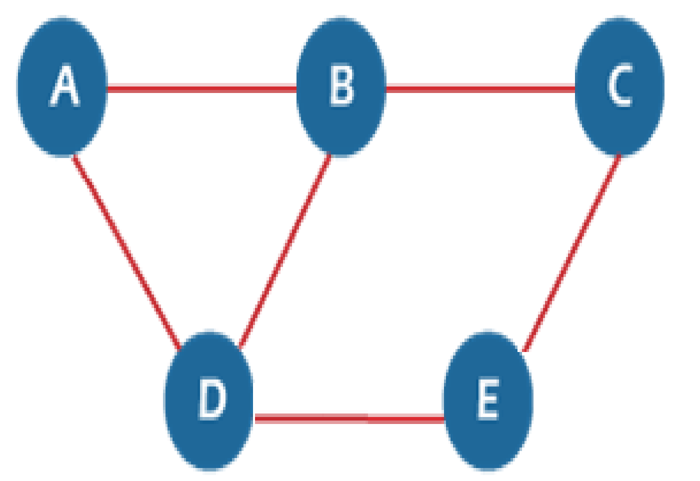
  aij = 1 {if there is a path exists from Vi to Vj}

  aij = 0 {Otherwise}

- An entry Aij in the adjacency matrix representation of an undirected graph G will be 1 if an edge exists between Vi and Vj.
- It means that, in an adjacency matrix, 0 represents that there is no association exists between the nodes.

Sequential representation

- We can use an adjacency matrix to represent the undirected graph, directed graph, weighted directed graph, and weighted undirected graph.



Undirected Graph

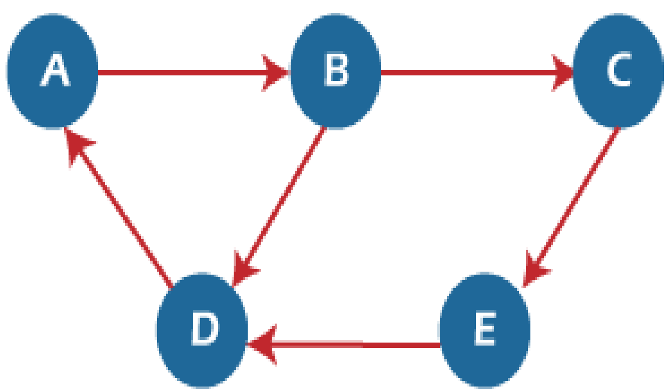
$$
\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array}
\begin{array}{c}
\begin{array}{ccccc} A & B & C & D & E \end{array} \\
\left[\begin{array}{ccccc}
0 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 & 0
\end{array}\right]
\end{array}
$$

Adjacency Matrix

Sequential representation

Adjacency matrix for a directed graph

- In a directed graph, an entry Aij will be 1 only when there is an edge directed from Vi to Vj.
- Entry A[i][j] is usually 1 if there is a directed edge from vertex i to vertex j, and 0 otherwise.



Directed Graph

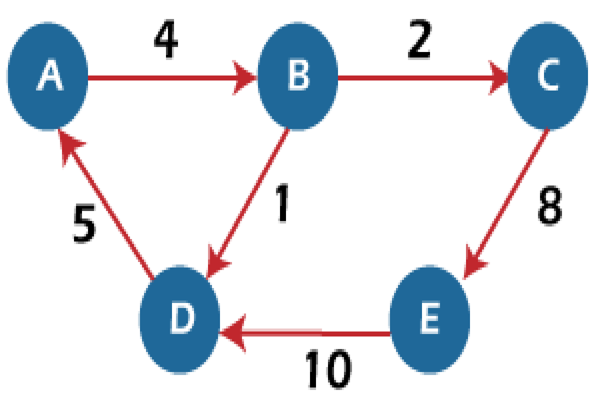
$$
\begin{array}{c|ccccc}
 & A & B & C & D & E \\
\hline
A & 0 & 1 & 0 & 0 & 0 \\
B & 0 & 0 & 1 & 1 & 0 \\
C & 0 & 0 & 0 & 0 & 1 \\
D & 1 & 0 & 0 & 0 & 0 \\
E & 0 & 0 & 0 & 1 & 0
\end{array}
$$

Adjacency Matrix

Sequential representation

Adjacency matrix for a weighted directed graph

- It is similar to an adjacency matrix representation of a directed graph except that instead of using the '1' for the existence of a path, here we have to use the weight associated with the edge. The weights on the graph edges will be represented as the entries of the adjacency matrix.

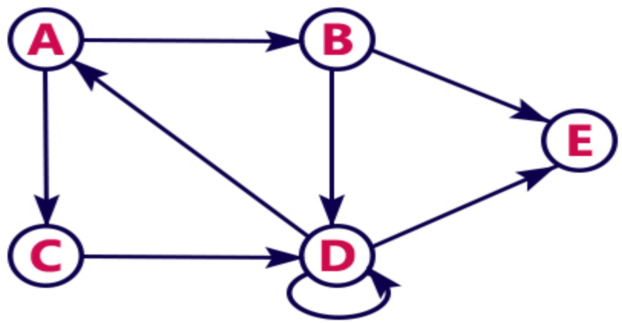


weighted Directed Graph

$$
\begin{array}{c}
\begin{array}{cccccc} & A & B & C & D & E \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \\ E \end{array}
\begin{pmatrix}
0 & 4 & 0 & 0 & 0 \\
0 & 0 & 2 & 1 & 0 \\
0 & 0 & 0 & 0 & 8 \\
5 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 10 & 0
\end{pmatrix}
\end{array}
$$

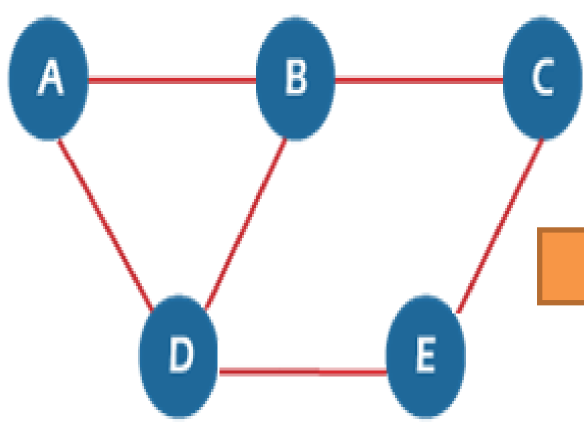
Adjacency Matrix

Directed graph representation



$$
\begin{array}{c|ccccc}
 & A & B & C & D & E \\
A & 0 & 1 & 1 & 0 & 0 \\
B & 0 & 0 & 0 & 1 & 1 \\
C & 0 & 0 & 0 & 1 & 0 \\
D & 1 & 0 & 0 & 1 & 1 \\
E & 0 & 0 & 0 & 0 & 0
\end{array}
$$

Undirected graph representation



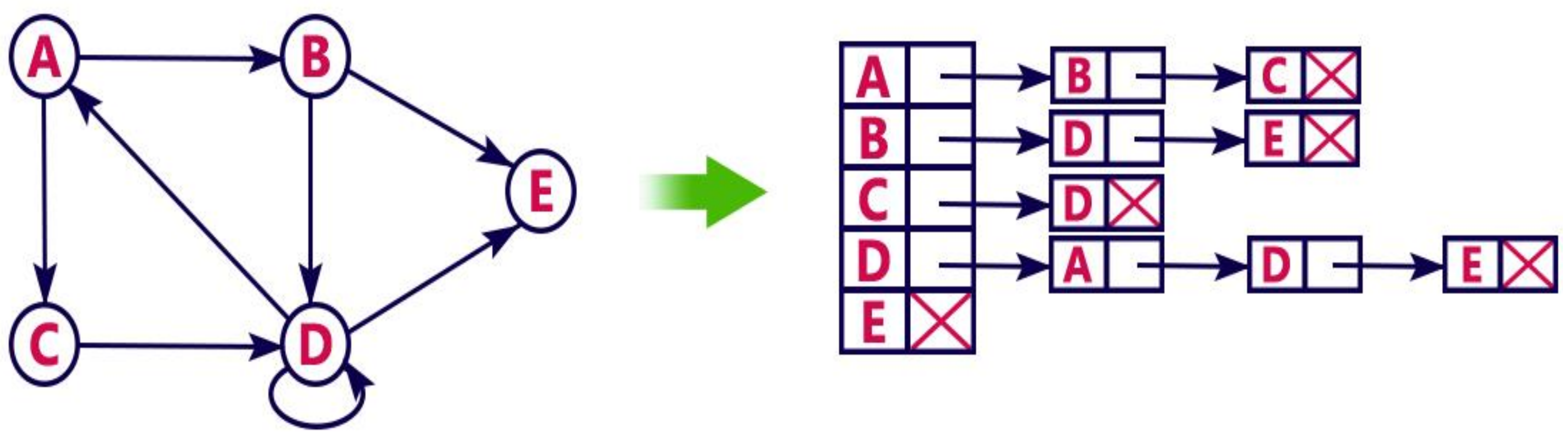
Undirected Graph

$$
\begin{array}{c|ccccc}
 & A & B & C & D & E \\
A & 0 & 1 & 0 & 1 & 0 \\
B & 1 & 0 & 1 & 1 & 0 \\
C & 0 & 1 & 0 & 0 & 1 \\
D & 1 & 1 & 0 & 0 & 1 \\
E & 0 & 0 & 1 & 1 & 0
\end{array}
$$

Adjacency Matrix

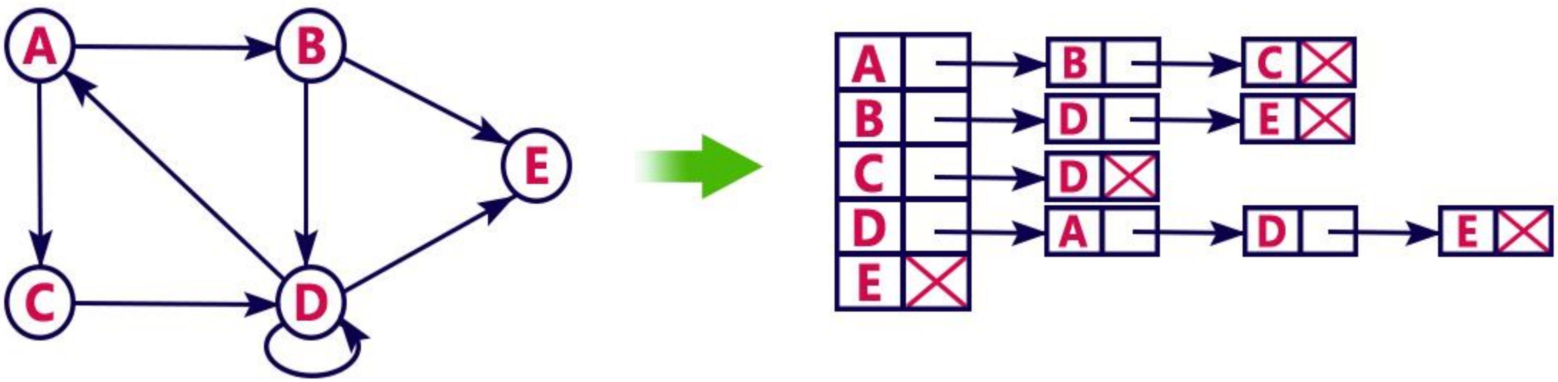
Linked list representation

- An adjacency list is maintained for each node present in the graph, which stores the node value and a pointer to the next adjacent node to the respective node. If all the adjacent nodes are traversed, then store the NULL in the pointer field of the last node of the list.

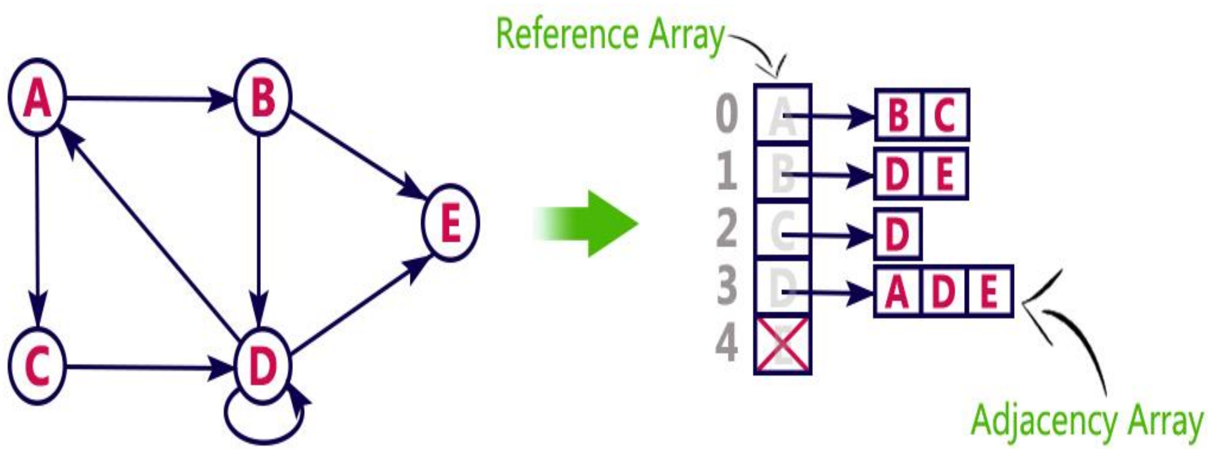


Linked list representation (2)

- In this representation, every vertex of a graph contains list of its adjacent vertices.
- For example, consider the following directed graph representation implemented using linked list...

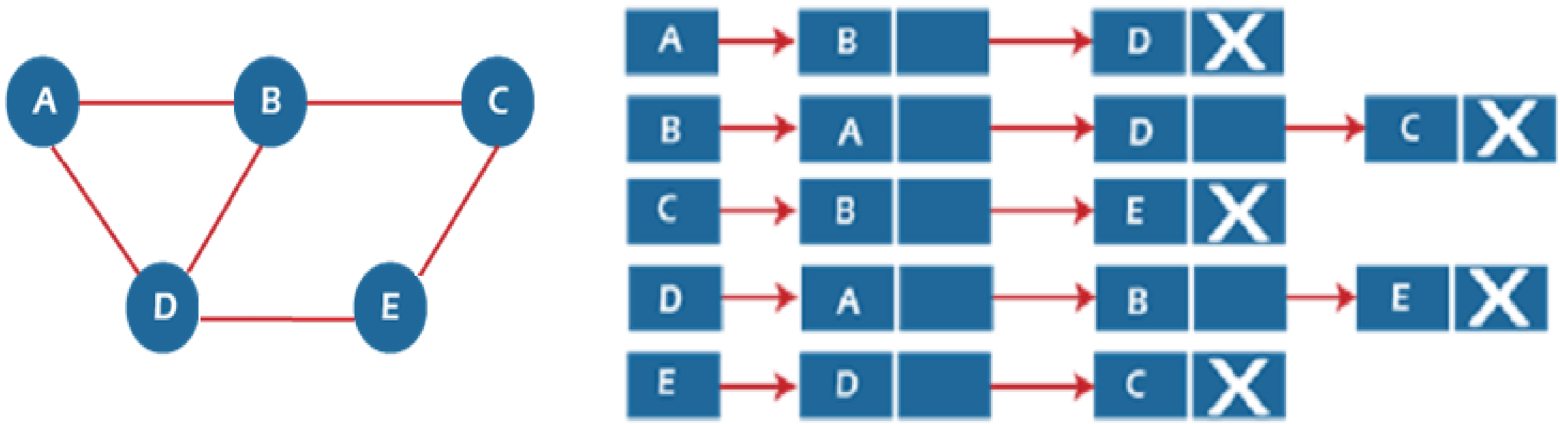


This representation can also be implemented using an array as follows..



Linked list representation (3)

Let's see the adjacency list representation of an undirected graph.

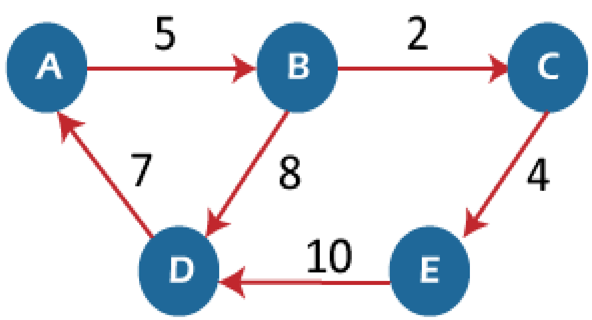


Undirected Graph Adjacency List

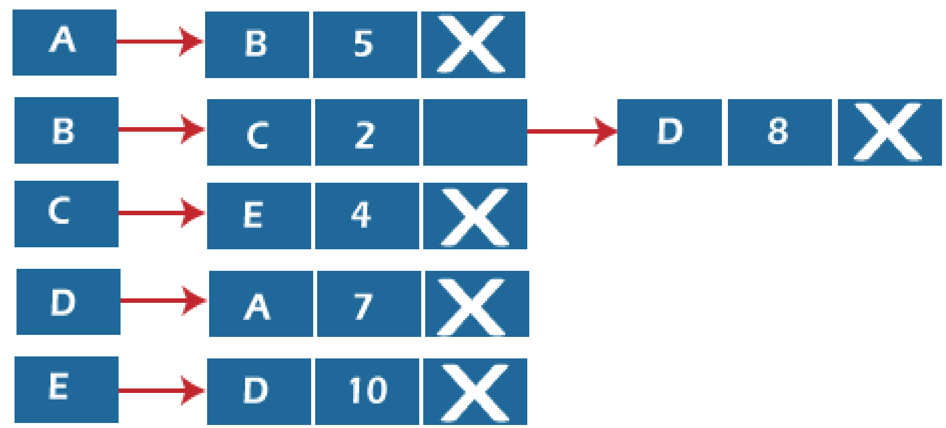
- In the above figure, we can see that there is a linked list or adjacency list for every node of the graph. From vertex A, there are paths to vertex B and vertex D. These nodes are linked to nodes A in the given adjacency list.

Linked list representation (4)

- For a directed graph, the sum of the lengths of adjacency lists is equal to the number of edges present in the graph.
- Now, consider the weighted directed graph, and let's see the adjacency list representation of that graph.



Weighted Directed Graph



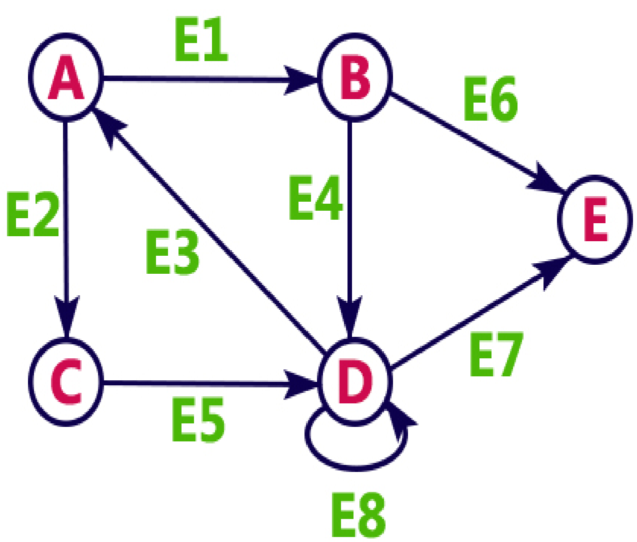
Adjacency List

Incidence Matrix

- In this representation, the graph is represented using a matrix of size total number of vertices by a total number of edges. That means graph with 4 vertices and 6 edges is represented using a matrix of size 4X6.
- In this matrix, rows represent vertices and columns represents edges.
- This matrix is filled with 0 or 1 or -1. Here, 0 represents that the row edge is not connected to column vertex, 1 represents that the row edge is connected as the outgoing edge to column vertex and -1 represents that the row edge is connected as the incoming edge to column vertex.

Incidence Matrix

For example, consider the following directed graph representation...



$$
\begin{array}{c}
 \\ A \\ B \\ C \\ D \\ E
\end{array}
\begin{array}{c}
\begin{array}{cccccccc} E1 & E2 & E3 & E4 & E5 & E6 & E7 & E8 \end{array} \\
\begin{pmatrix}
1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & -1 & -1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & -1 & -1 & 0
\end{pmatrix}
\end{array}
$$

Graph Traversal

- Graph traversal is a technique used for searching a vertex in a graph.
- The graph traversal is also used to decide the order of vertices is visited in the search process.
- graph traversal means visit all the vertices of the graph without getting into looping path.
- There are two graph traversal techniques and they are as follows...
- BFS (Breadth First Search)
- DFS (Depth First Search)

BFS (Breadth First Search)

BFS algorithm

- Breadth-first search is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighboring nodes. Then, it selects the nearest node and explores all the unexplored nodes. While using BFS for traversal, any node in the graph can be considered as the root node.
- BFS is the most commonly used approach. It is a recursive algorithm to search all the vertices of a graph data structure.
- BFS puts every vertex of the graph into two categories - visited and non-visited. It selects a single node in a graph and, after that, visits all the nodes adjacent to the selected node.
- BFS uses Queue data structure

BFS (Breadth First Search) Algorithm

BFS traversal of a graph produces a spanning tree as final result. Spanning Tree is a graph without loops.

Here's a high-level explanation of the BFS algorithm:

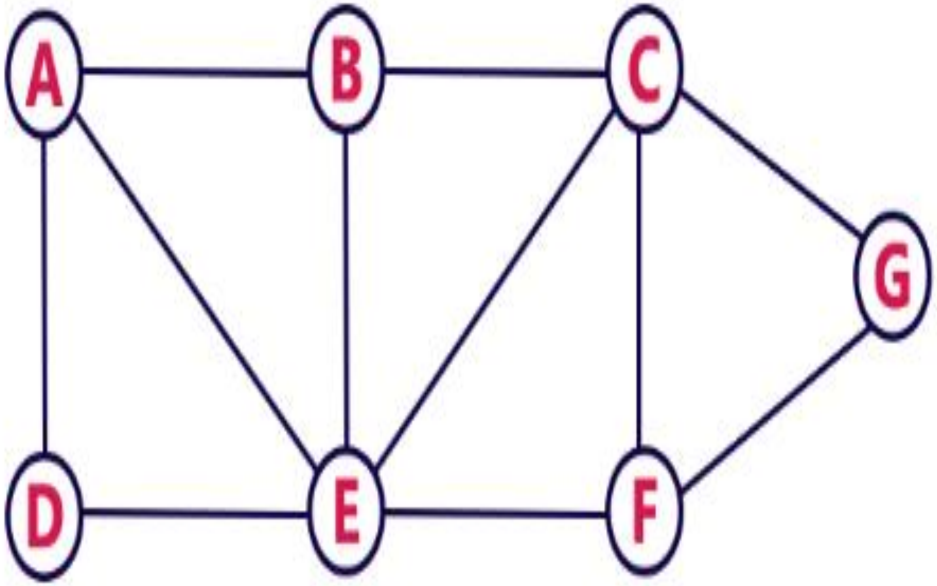
- Initialization: Choose a starting vertex.
- Explore: Visit the starting vertex and enqueue its neighbors.
- Iterate: While there are vertices in the queue, dequeue a vertex and explore its neighbors, enqueueing those that haven't been visited.
- Mark Visited: Keep track of visited vertices to avoid loops or revisiting vertices.

BFS can be implemented using a queue data structure. The steps typically involve:

- Enqueue the starting vertex into a queue and mark it as visited.
- While the queue is not empty:
  - Dequeue a vertex from the queue.
  - Visit and process the dequeued vertex.
  - Enqueue all its unvisited neighbors and mark them as visited.

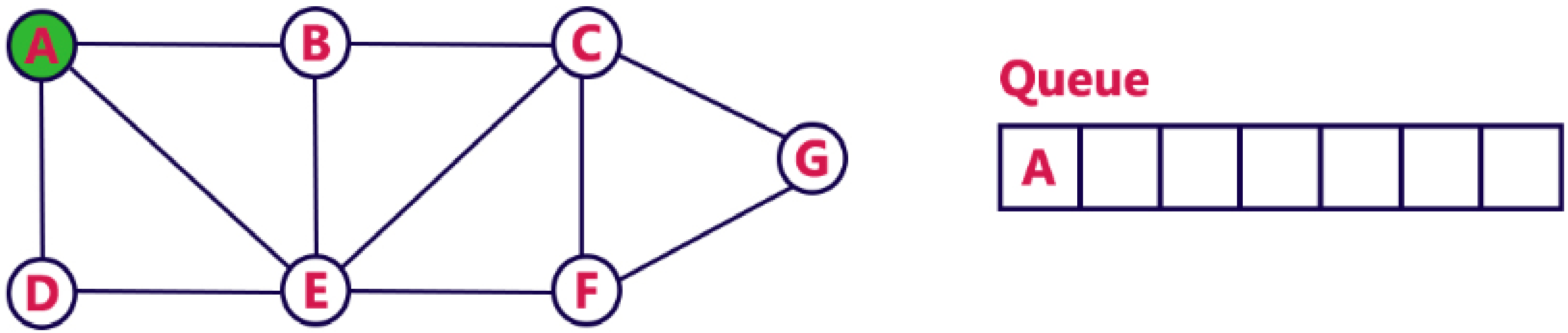
BFS Algorithm Example

Consider the following example graph to perform BFS traversal



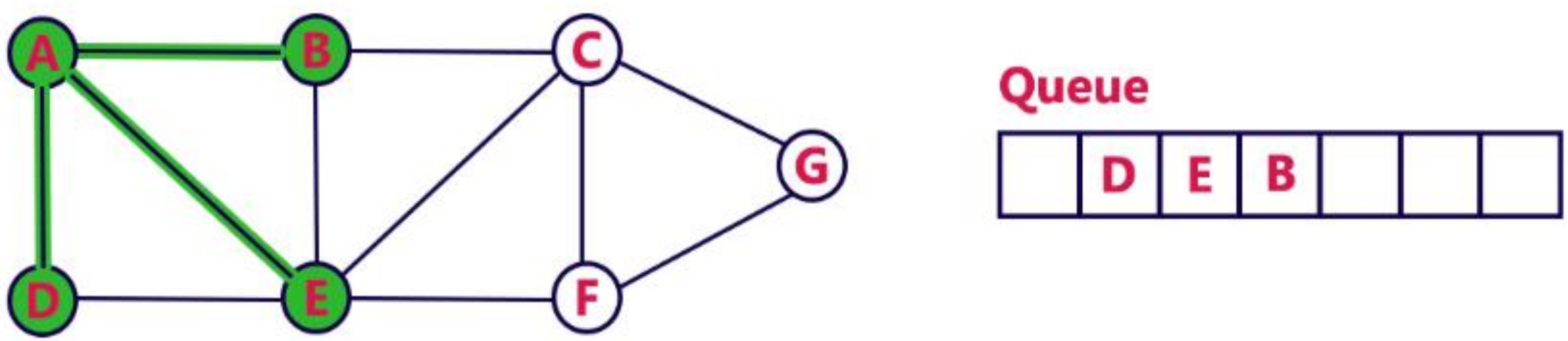
Step 1:

- Select the vertex **A** as starting point (visit **A**).
- Insert **A** into the Queue.



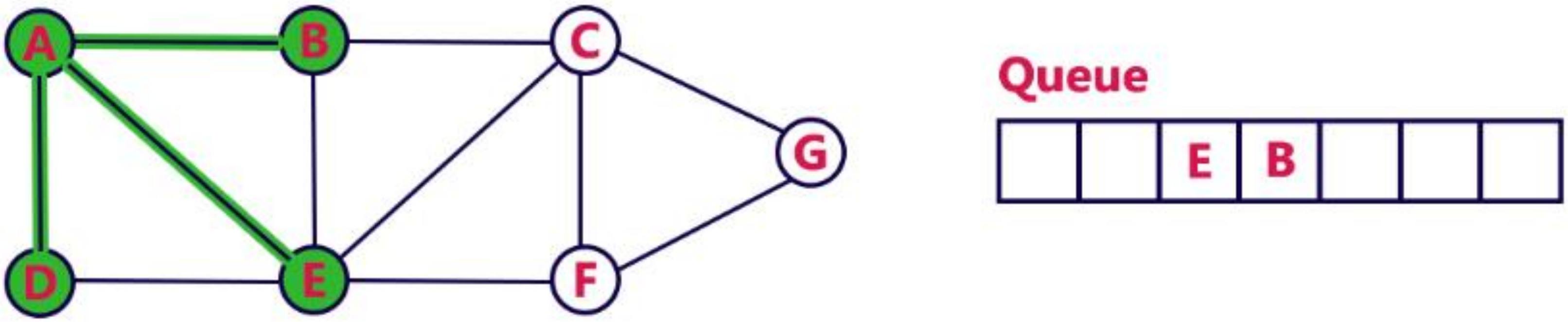
Step 2:

- Visit all adjacent vertices of **A** which are not visited (**D**, **E**, **B**).
- Insert newly visited vertices into the Queue and delete A from the Queue..



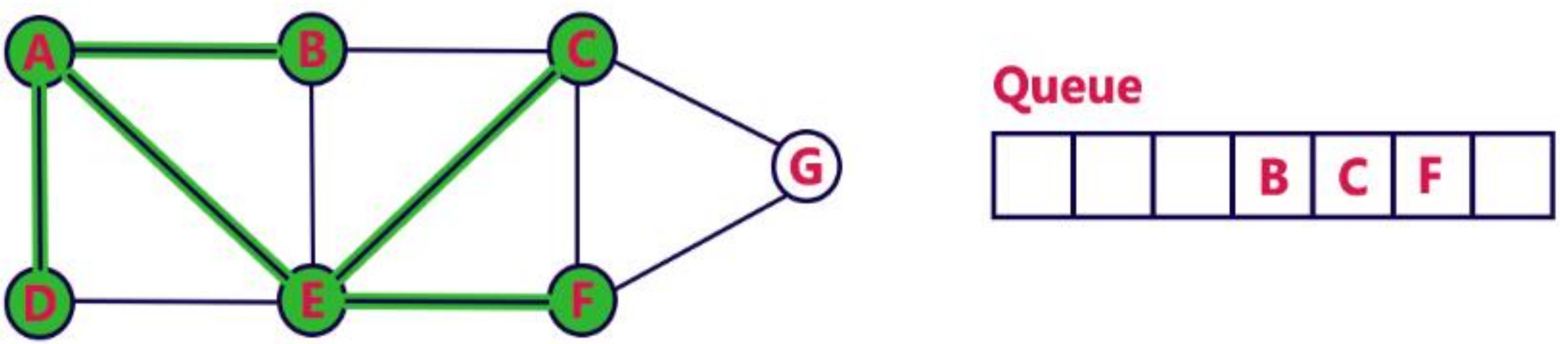
Step 3:

- Visit all adjacent vertices of **D** which are not visited (there is no vertex).
- Delete D from the Queue.



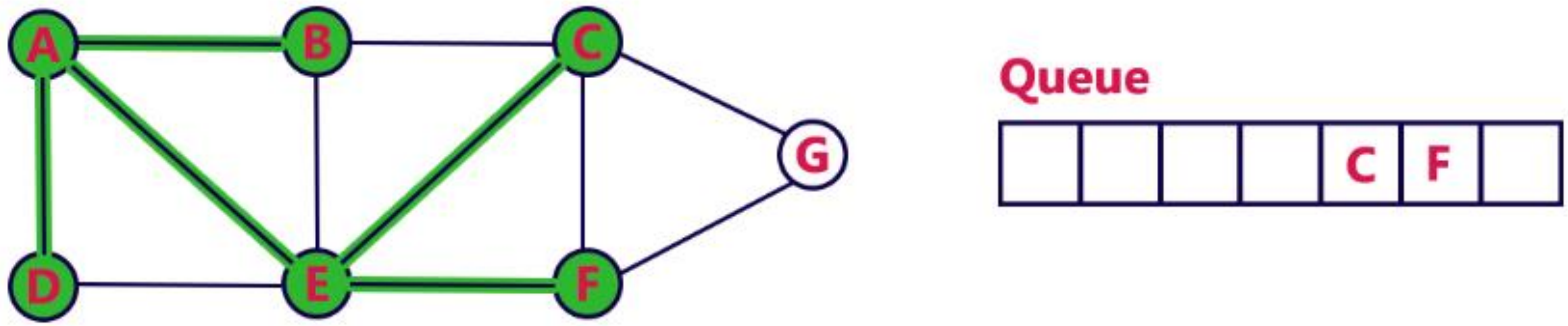
Step 4:

- Visit all adjacent vertices of **E** which are not visited (**C**, **F**).
- Insert newly visited vertices into the Queue and delete E from the Queue.



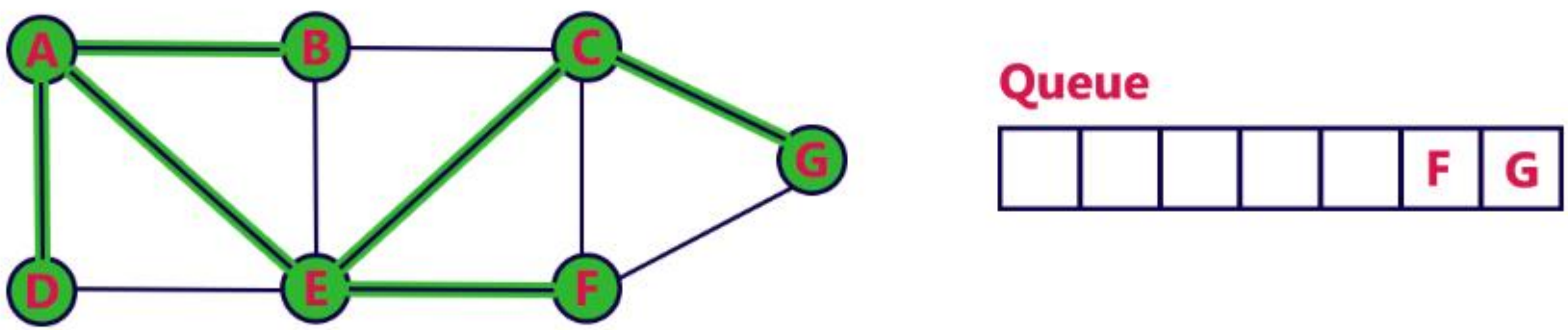
Step 5:

- Visit all adjacent vertices of **B** which are not visited (**there is no vertex**).
- Delete **B** from the Queue.



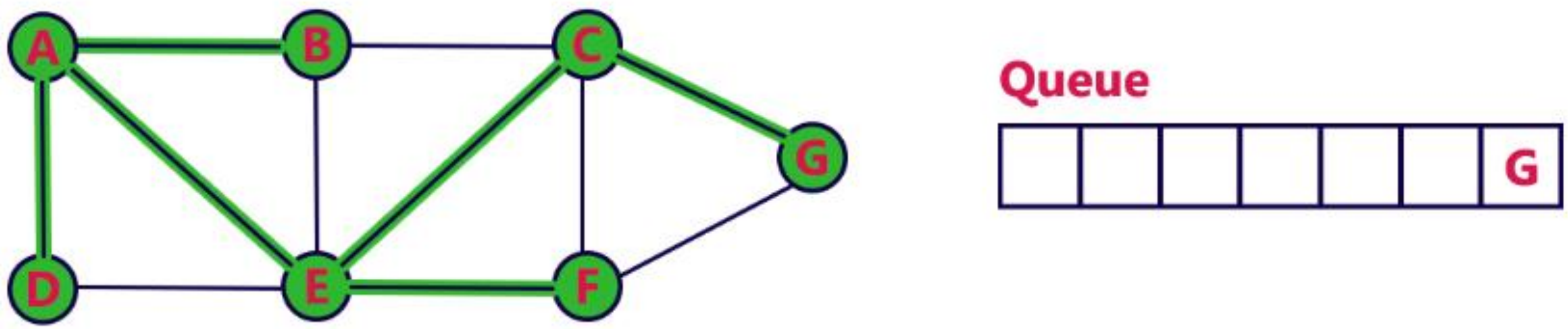
Step 6:

- Visit all adjacent vertices of **C** which are not visited (**G**).
- Insert newly visited vertex into the Queue and delete **C** from the Queue.



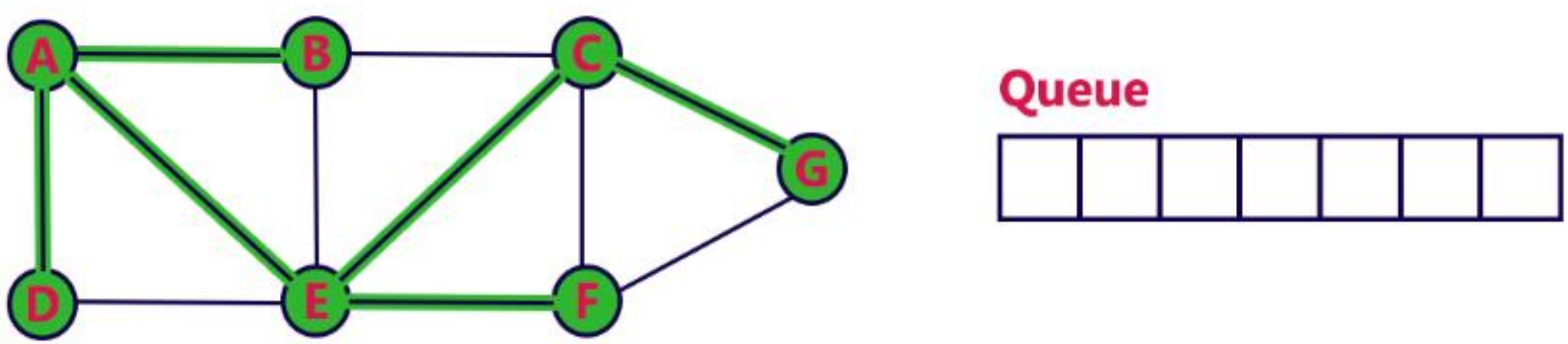
Step 7:

- Visit all adjacent vertices of **F** which are not visited (**there is no vertex**).
- Delete **F** from the Queue.



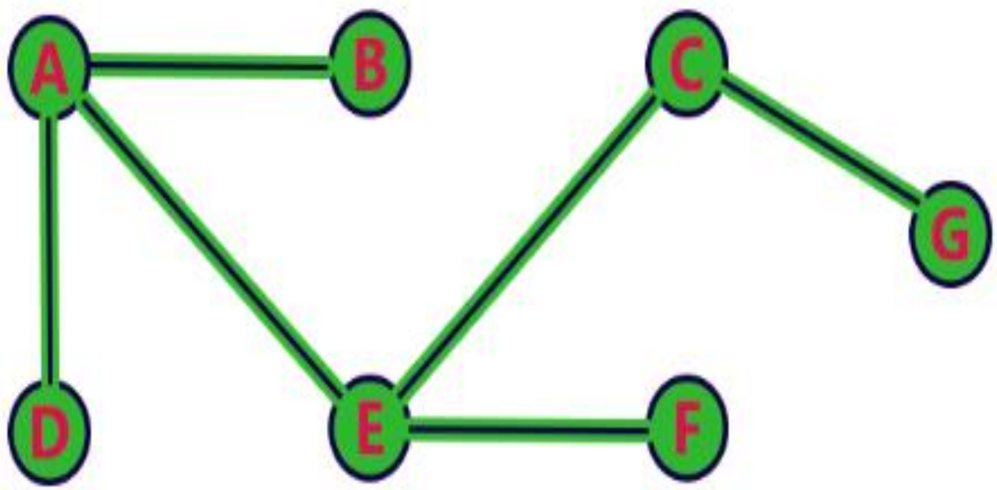
Step 8:

- Visit all adjacent vertices of **G** which are not visited (**there is no vertex**).
- Delete **G** from the Queue.



BFS (Breadth First Search)
Final Output

- Queue became Empty. So, stop the BFS process.
- Final result of BFS is a Spanning Tree as shown below...



DFS (Depth First Search)

- The depth-first search (DFS) algorithm starts with the initial node of graph G and goes deeper until we find the node with no children.
- DFS uses Stack data structure
- Depth-First Search (DFS) is another graph traversal algorithm used to systematically explore all the vertices in a graph. Unlike BFS, which explores neighbors level by level, DFS explores as far as possible along each branch before backtracking.

DFS (Depth First Search)

- DFS traversal of a graph produces a spanning tree as final result. Spanning Tree is a graph without loops.

The basic steps of the DFS algorithm are:

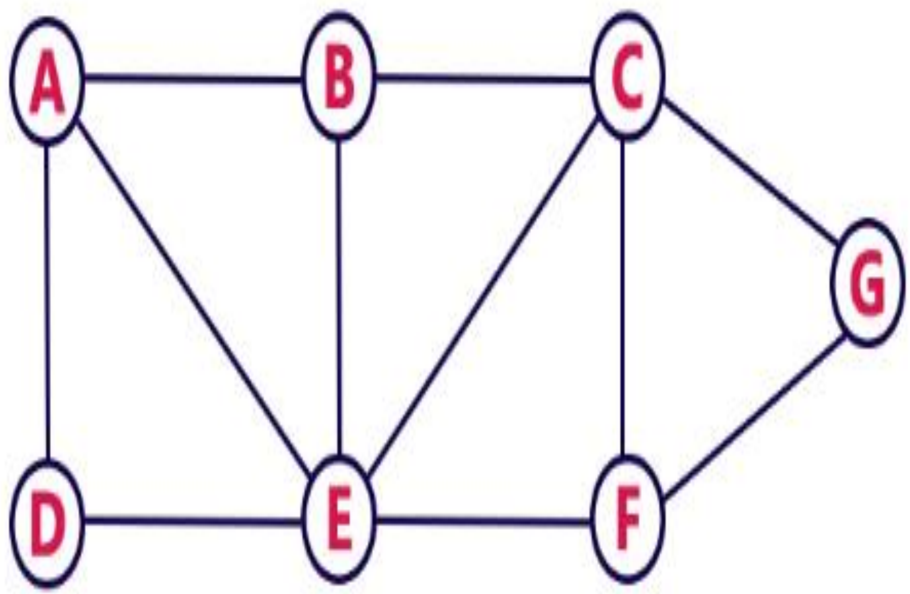
- Initialization: Choose a starting vertex.
- Explore: Visit the starting vertex and recursively explore its unvisited neighbors.
- Mark Visited: Keep track of visited vertices to avoid loops or revisiting vertices.

Here's a simple explanation of the DFS algorithm:

- Start at a specific vertex and mark it as visited.
- Explore its adjacent unvisited vertices.
- If an unvisited vertex is found, recursively apply DFS to it.
- Repeat this process until all vertices are visited. Back tracking is coming back to the vertex from which we reached the current vertex.

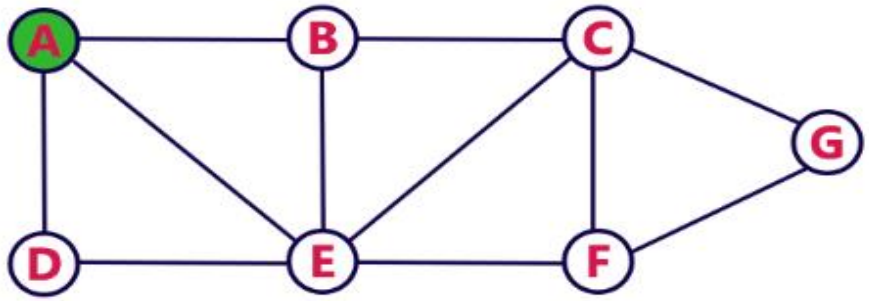
DFS (Depth First Search)

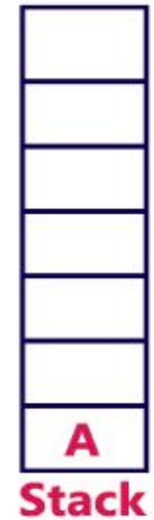
Consider the following example graph to perform DFS traversal



Step 1:

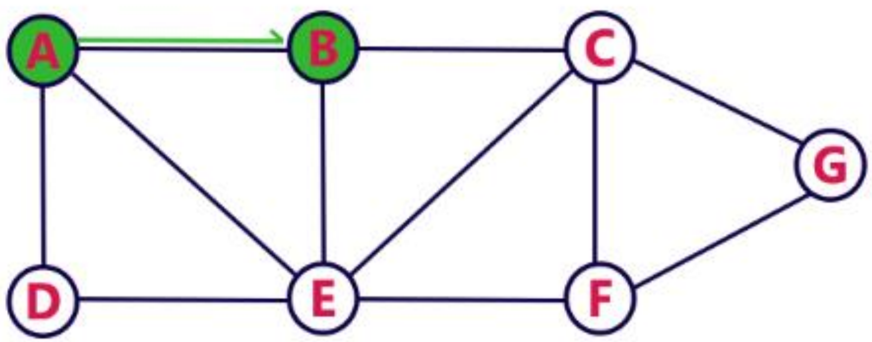
- Select the vertex **A** as starting point (visit **A**).
- Push **A** on to the Stack.

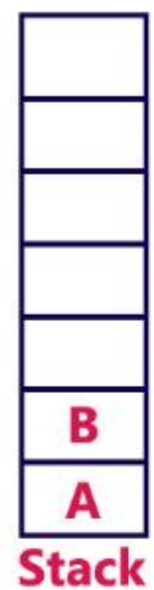




Step 2:

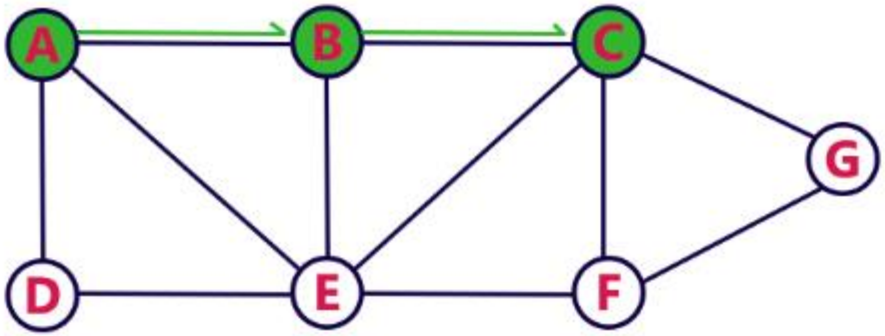
- Visit any adjacent vertex of **A** which is not visited (**B**).
- Push newly visited vertex B on to the Stack.

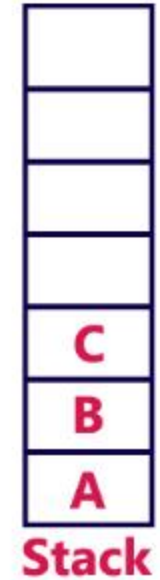




Step 3:

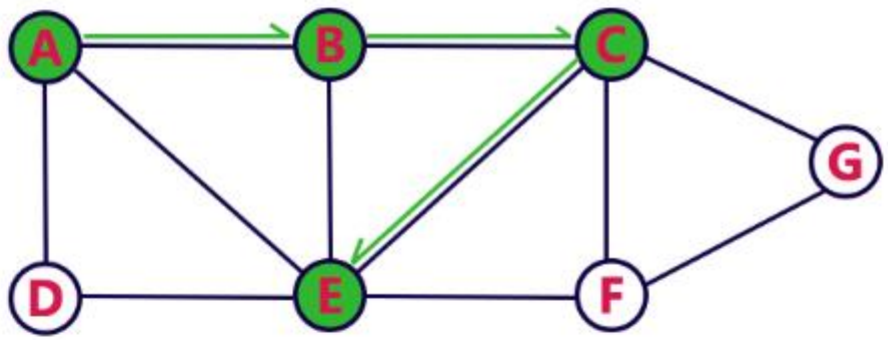
- Visit any adjacent vertext of **B** which is not visited (**C**).
- Push C on to the Stack.

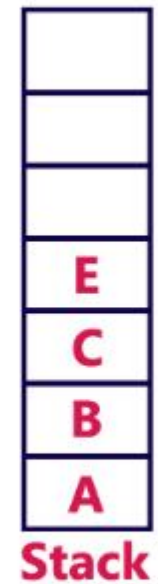




Step 4:

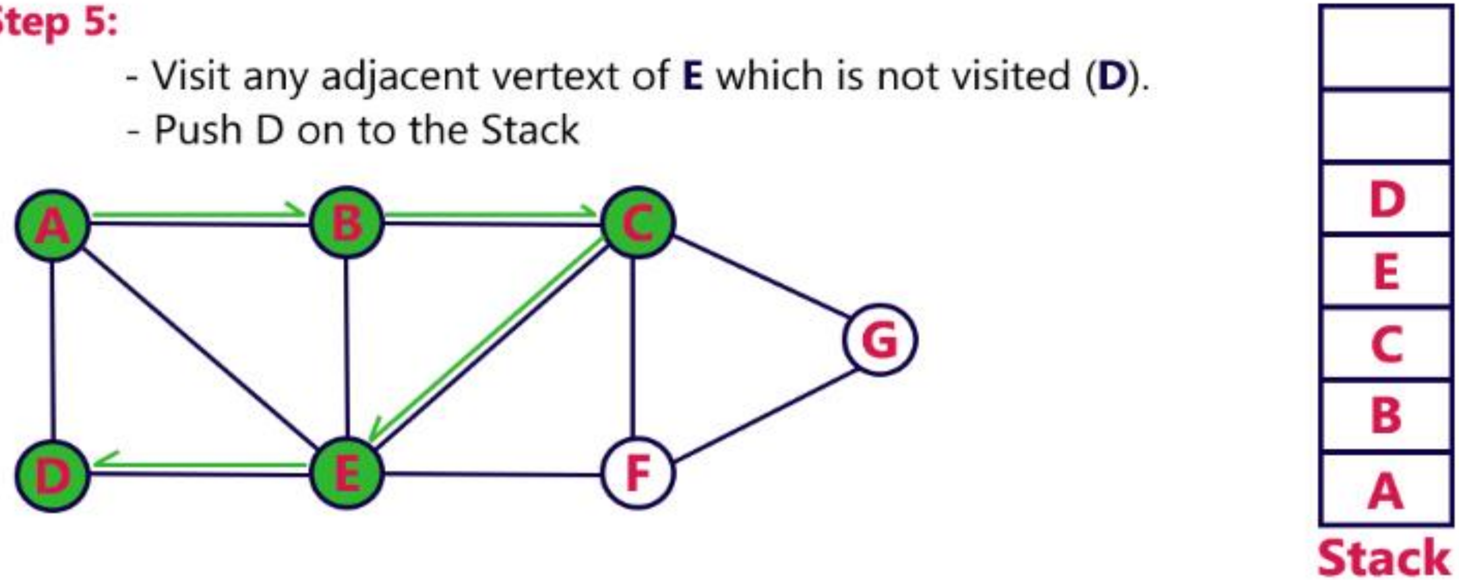
- Visit any adjacent vertext of **C** which is not visited (**E**).
- Push E on to the Stack





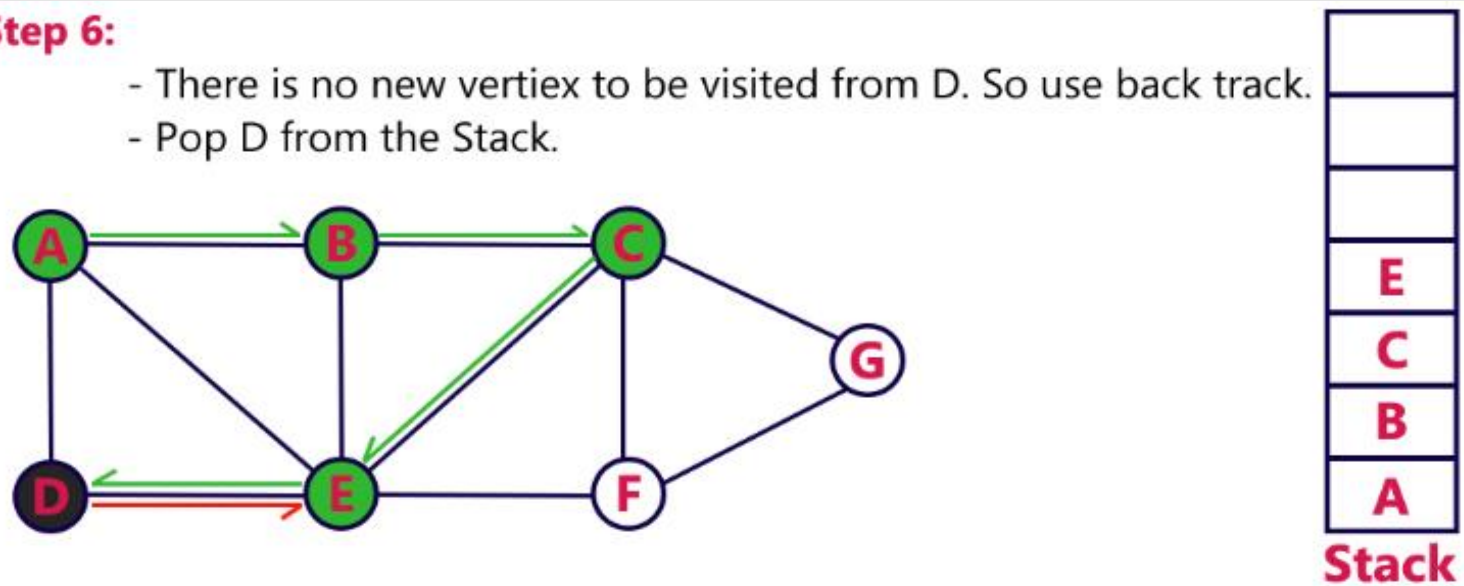
Step 5:

- Visit any adjacent vertext of **E** which is not visited (**D**).
- Push D on to the Stack



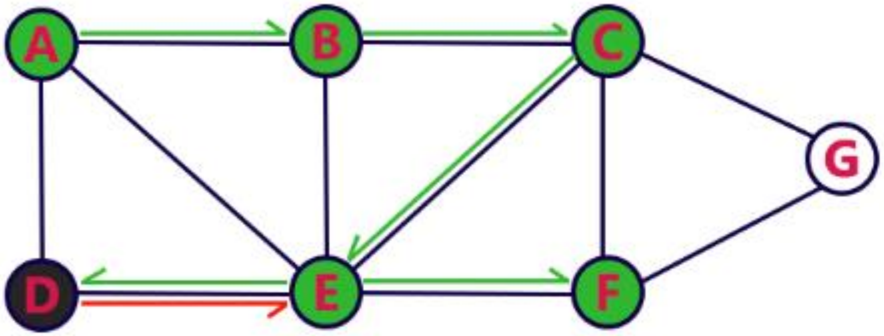
Step 6:

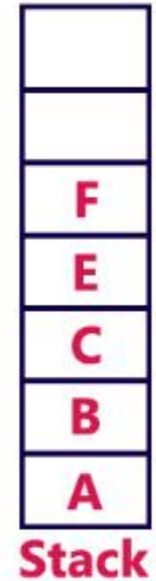
- There is no new vertiex to be visited from D. So use back track.
- Pop D from the Stack.



Step 7:

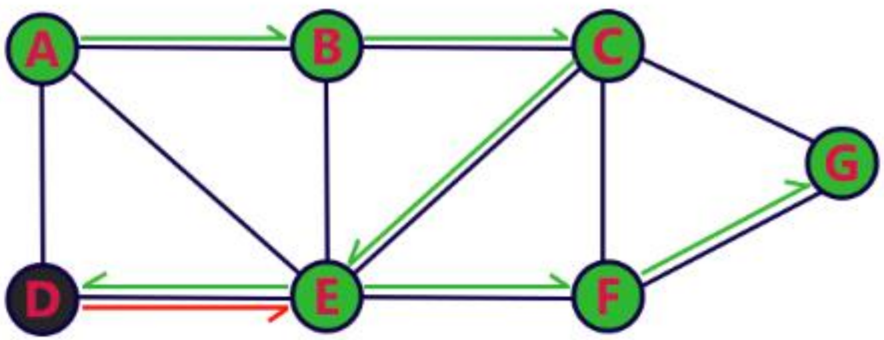
- Visit any adjacent vertex of **E** which is not visited (**F**).
- Push **F** on to the Stack.

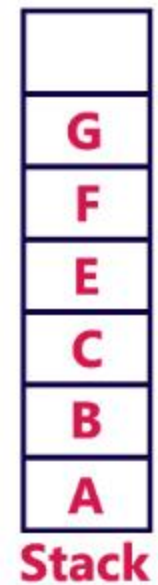




Step 8:

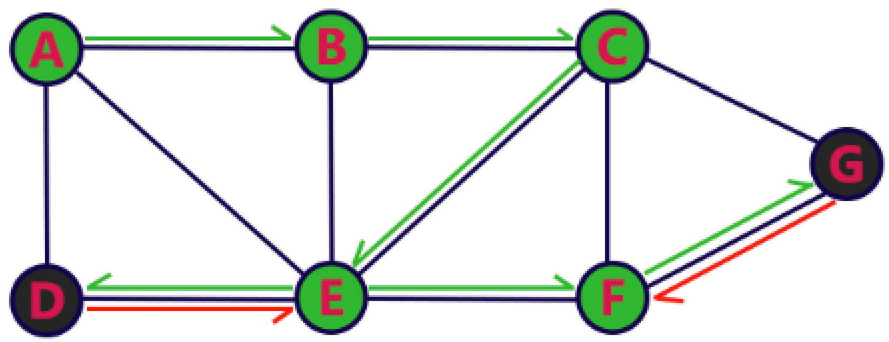
- Visit any adjacent vertex of **F** which is not visited (**G**).
- Push **G** on to the Stack.

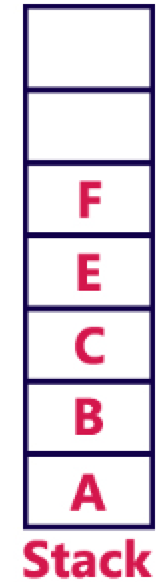




Step 9:

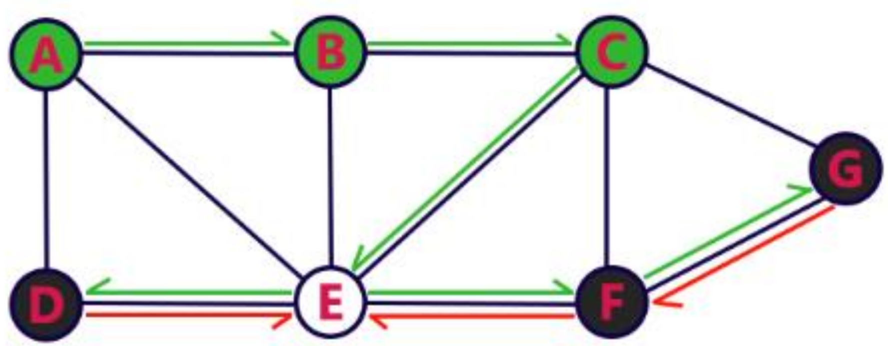
- There is no new vertiex to be visited from G. So use back track.
- Pop G from the Stack.





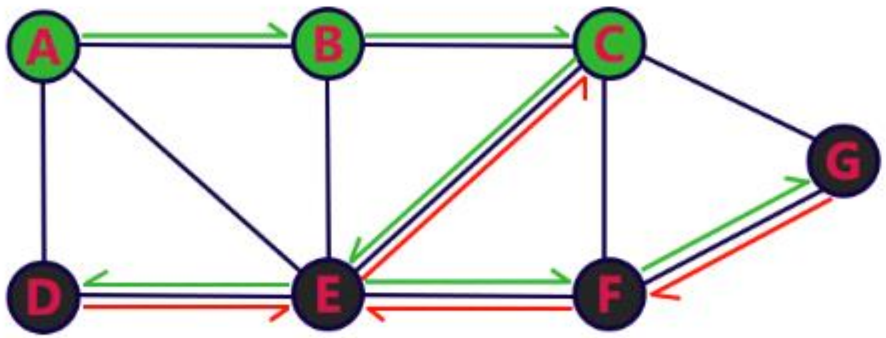
Step 10:

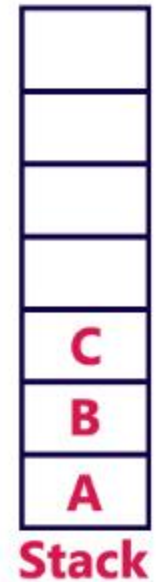
- There is no new vertiex to be visited from F. So use back track.
- Pop F from the Stack.



Step 11:

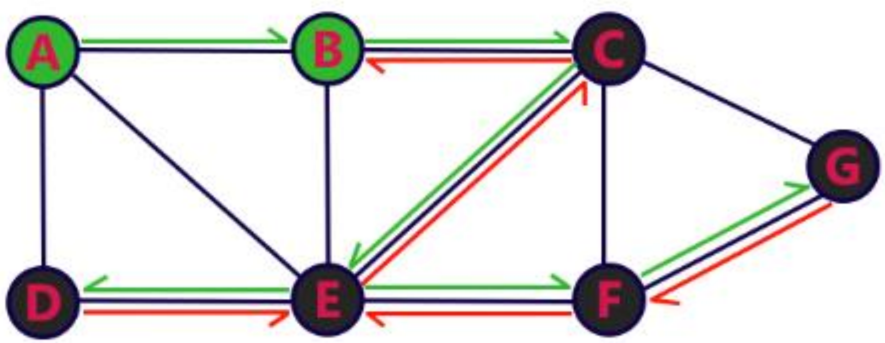
- There is no new vertiex to be visited from E. So use back track.
- Pop E from the Stack.

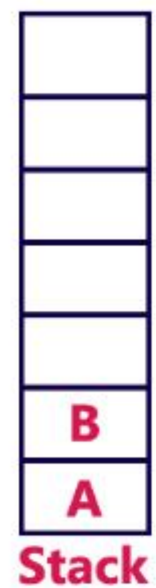




Step 12:

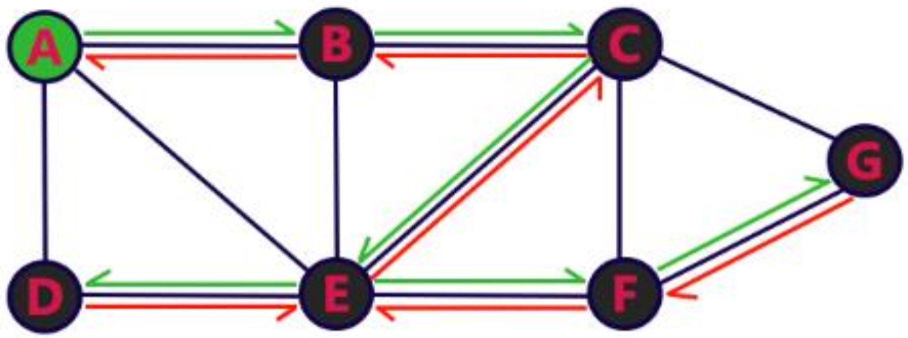
- There is no new vertiex to be visited from C. So use back track.
- Pop C from the Stack.

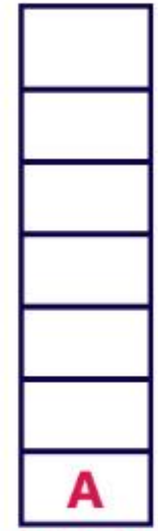




Step 13:

- There is no new vertiex to be visited from B. So use back track.
- Pop B from the Stack.

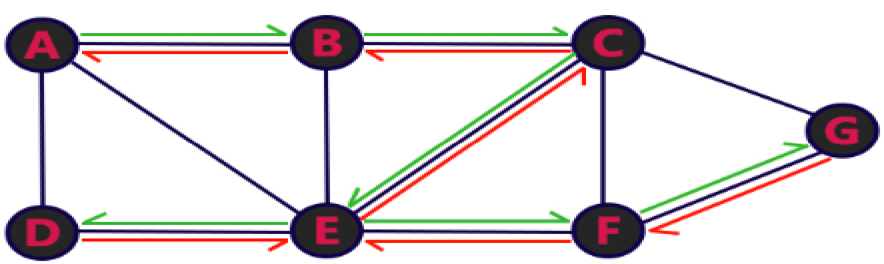


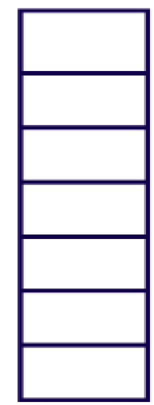


Stack

Step 14:

- There is no new vertiex to be visited from A. So use back track.
- Pop A from the Stack.



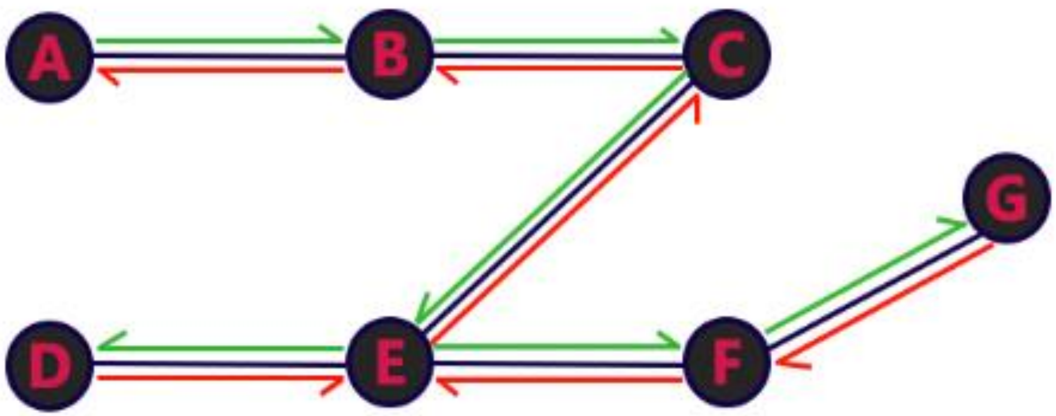


Stack

- Stack became Empty. So stop DFS Treversal.
- Final result of DFS traversal is following spanning tree.

DFS (Depth First Search)
Final Output

- Stack became Empty. So stop DFS Treversal.
- Final result of DFS traversal is following spanning tree.



Question