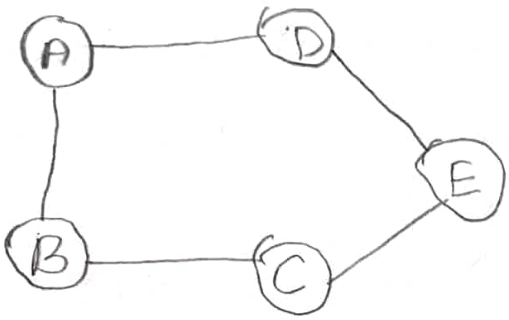
Unit 4: Graph

- A graph G can be defined as an ordered set G(V, E) where V(G) represent the set of vertices and E(G) represent the set of edges which are used to connect these vertices.

- 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, A)) is shown in the following fig.



Path: A path can be defined as the sequence of nodes that are followed in order to reach some terminal node V from the initial node U.

Weighted graph: In weighted graph, each edge is assigned with some value such as length or weight

Digraph: A digraph is a directed graph in which each edge of the graph is associated with some direction and the traversing can be done only in specified direction.

Loop:- an edge that is associated with the similar end points can be called as Loop.

Adjacent Nodes: If two nodes u and v are connected via an edge e, then the nodes u and v are called adjacent nodes.

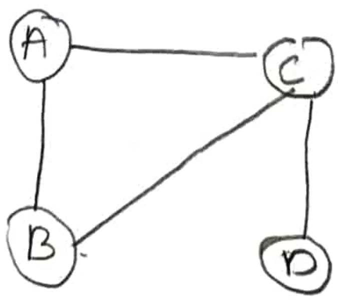
*Types of graph:

A graph can be
1. Undirected graph.
2. Directed graph.

1. Undirected graph:

- In undirected graph, edges are not associated with the direction with them.
- In undirected graph, The pair of vertices representing any edge is unordered That is pairs (V1, V2) and (V2, V1) represent same edge.

eg. consider a graph G=(V, E)
where V= {A, B, C, D}
G= {(A,B), (A,C), (B,C), (C,D)}



graph G.

- In This graph edges are not having directions, so this is known as undirected graph.
- Here we can Travel from A to B as well as from B to A also.

2. Directed graph:

- In a directed graph, edge ~~from at~~ form an ordered pair. edges represent a specific path from some vertex A to another vertex B. Node A is called initial node while node B is called terminal node. so that Here, pairs (A, B) and (B, A) are different edges.

- Directed graph is also called as Diagraph.

eg. consider a graph $G = (V, E)$
where $V = \{A, B, C, D, E\}$
$E = \{(A, B), (A, C), (C, D), (A, E), (E, D)\}$

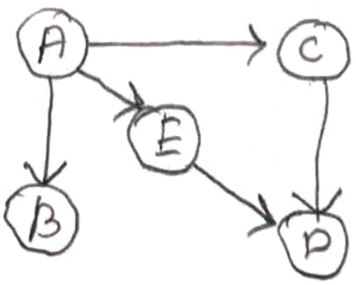


fig. graph G.

- In this graph edges are having direction. so it is known as directed graph.

- Here we can travel from A to B but we can not travel back from B to A.

* Graph Representation:

- Graph representation, means technique to store a graph in to computer's memory.

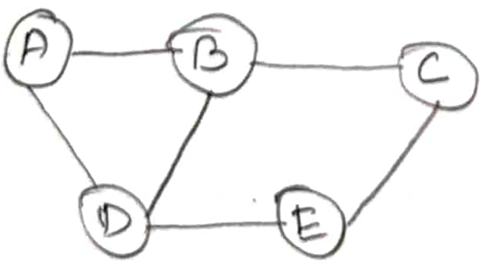
- There are two ways to store graph in to computer's memory.

1. Adjacency Matrix (sequential Representation)
2. Adjacency List (Linked Lis Representation)

1. Adjacency matrix:-

- In this method, as name indicate we have to use matrix that is two dimentional array to store graph into computer's memory.

eg. construct adjacncy matrix for following graph.

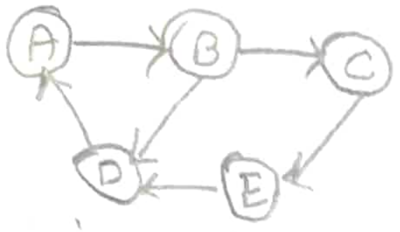


⇒ - This is undirected graph.

- Adjacency matrix for above graph is as shown →

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

- Adjacency matrix for directed graph.



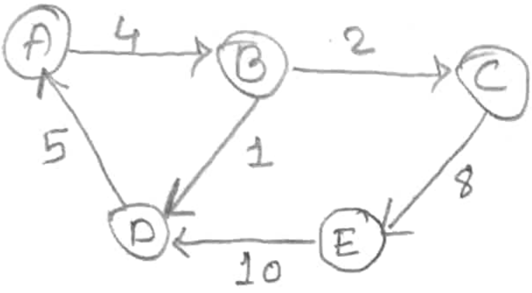
Directed graph

$\longrightarrow$

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

Adjacency matrix

- Adjacency matrix for weighted & directed graph.



Directed-weighted graph

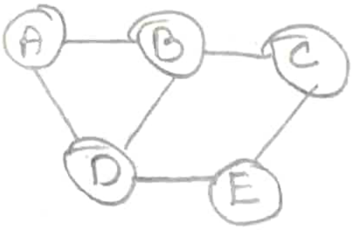
$$
\begin{array}{c|c}
 & \begin{matrix} A & B & C & D & E \end{matrix} \\
\begin{matrix} A \\ B \\ C \\ D \\ E \end{matrix} & \begin{bmatrix} 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 8 \\ 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 \end{bmatrix}
\end{array}
$$

Adjacency matrix

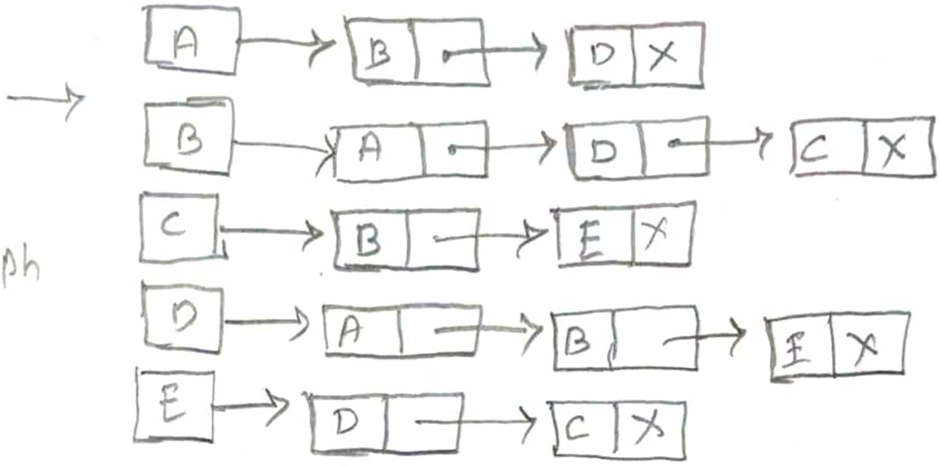
2. **Adjacency List:**

- Adjacency list is used in linked representation to store the graph in the computer's memory. it is efficient in terms of storage as we only have to store the value of edges.
- In this representation, for each vertex a linked list is maintained that stores it's adjacent vertices.

- Linked representation of an undirected graph.

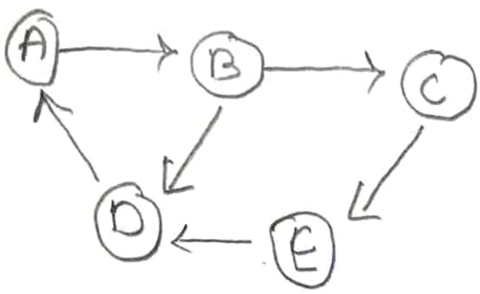


undirected graph

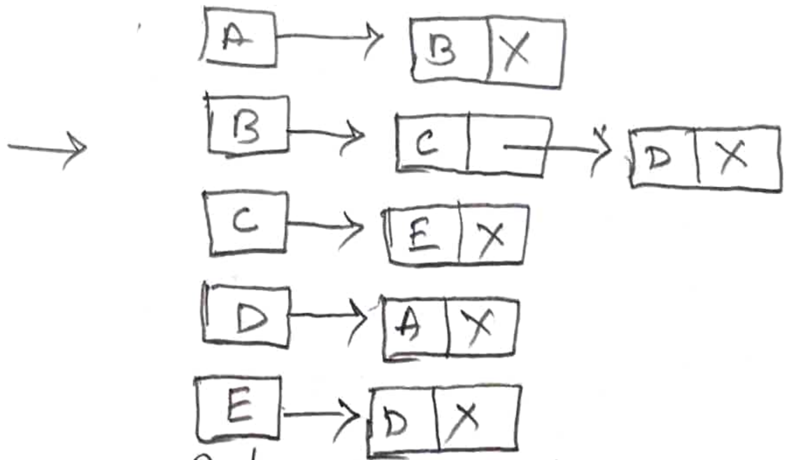


Adjacency List

- Linked Representation of Directed graph.

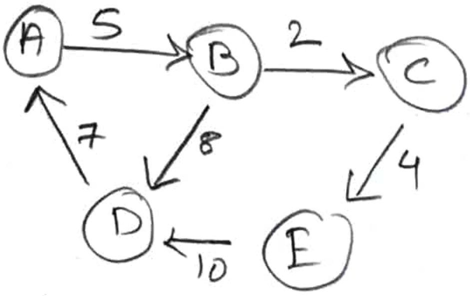


Directed graph

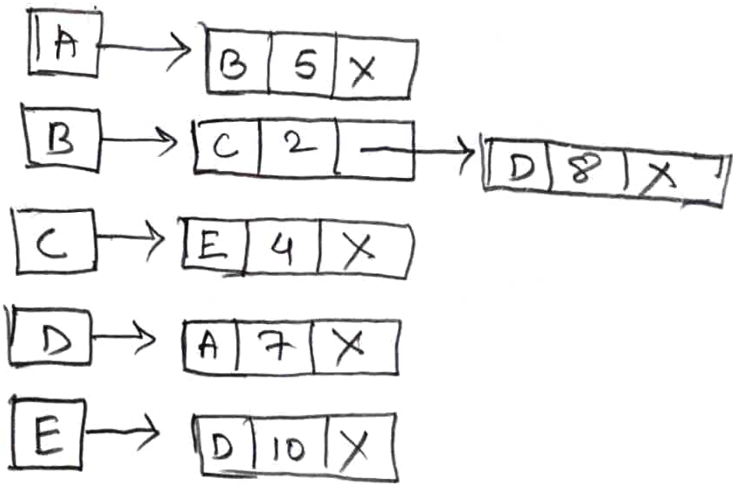


Adjacency List.

- Linked representation of Directed weighted graph.



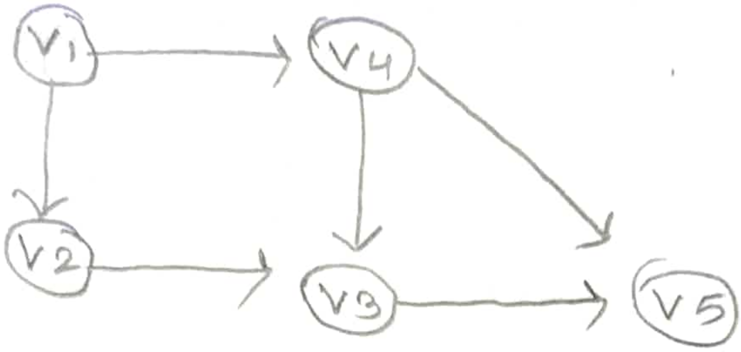
Directed-weighted graph



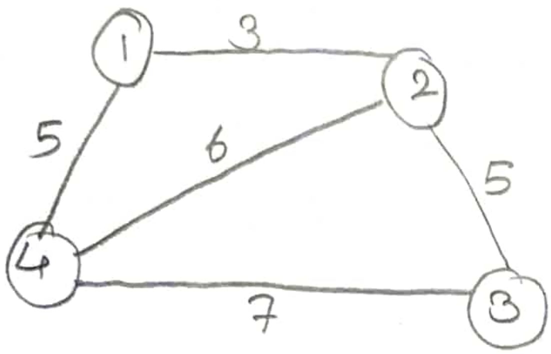
Adjacency List.

* construct adjacency matrix and Adjacency graph list for following graph.

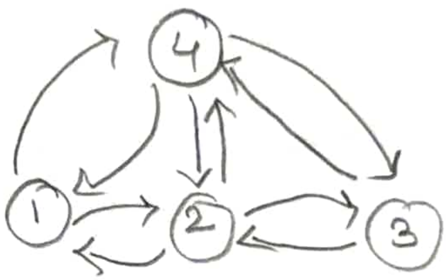
1)



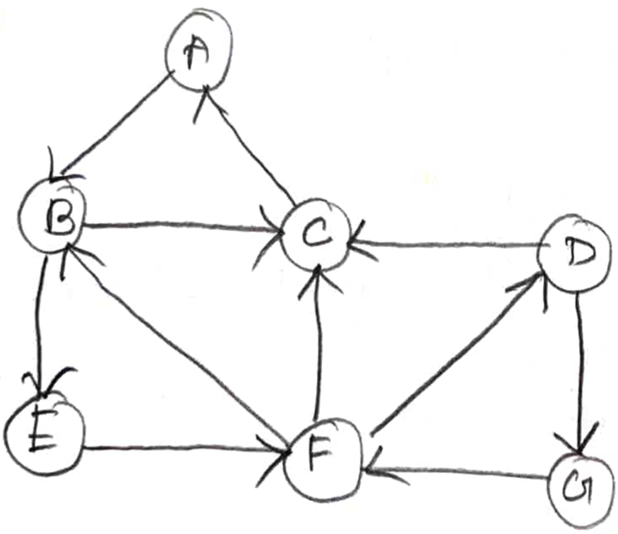
2)



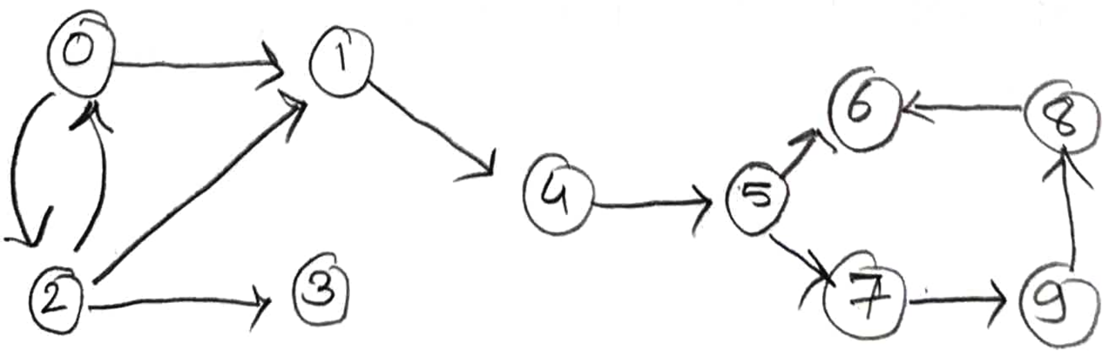
3)



4)



5



* Graph Traversal

- Graph Traversing technique is nothing but visiting every vertex at least once in systematic fashion.
- The graph traversal is used to decide the order of vertices to be visit in the search process.
- A graph traversal finds the edges to be used in the search process without creating loops that means using graph traversal we visit all the vertices of graph without getting in to looping path.

- Two techniques are used for traversal:
  1. Depth first search (DFS)
  2. Breadth first search (BFS)

1. Depth first search (DFS)

- DFS traversal of a graph produces a spanning tree as final result. spanning tree is a graph without any loop.
- we use stack data structure with maximum size of total no. of vertices in the graph to implement DFS traversal of graph.

* steps to implement DFS traversal.

1. select any vertex as starting point for traversal. visit the vertex and push it on to the stack.

2. visit any one of the adjacent vertex of the vertex which is at top of the stack which is not visited and push it on to the stack.

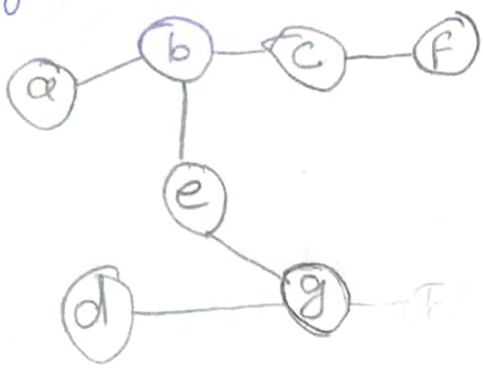
3. Repeat step 2 until there are no new vertex to be visited from the vertex on top of the stack.
4. When there is no new vertex to be visited then use backtracking and pop one vertex from the stack.
5. Repeat step 2, 3 & 4 until stack become empty.
6. When stack become empty, then stop.

* Algorithm for DFS

Step 1: set
status = 1 for each node in graph G
Step 2: Push the starting node A in the stack
set it's status = 2
Step 3: Repeat step 4 and 5 until stack is empty.
Step 4: Pop the top node N from the stack.
Process it and set status = 3
Step 5: push all the neighbours of N with status = 1 and set status = 2
Step 6: stop.

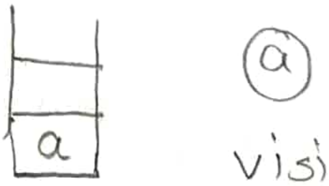
Example

*consider following graph and travers using DFS



→ step 1:

select vertex 'a as starting point. push 'a' on to the stack.

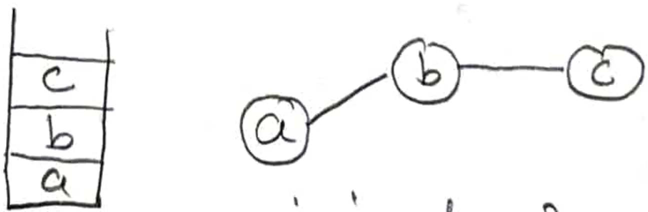


visited = {a}

step 2:
visit any adjacent vertex of 'a' which is not visited 'b'. push b on to the stack.

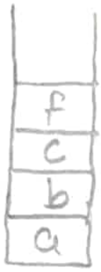
visited = {a,b}

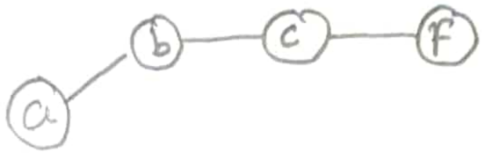
step 3: visit any adjacent and unvisited vertex of 'b'. and push 'c' on to the stack



visited = {a,b,c}

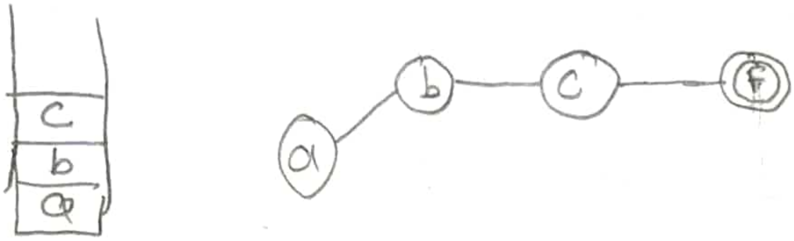
step 4: visit any adjacent vertex of 'c' which is not visited 'f'. push 'f' on to the stack.



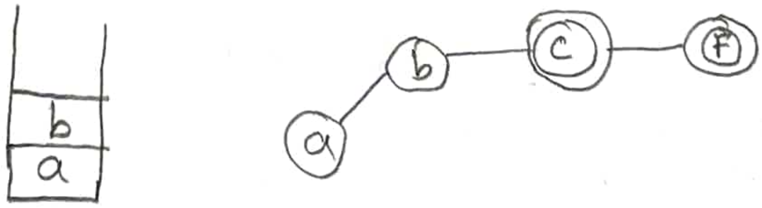


visited = {a, b, c, f}

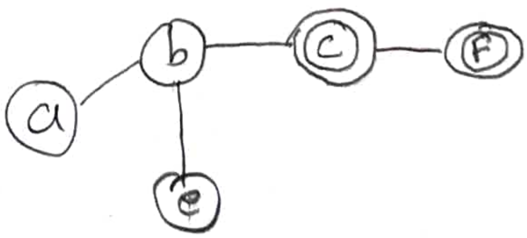
step 5: Now there is no new unvisited vertex from f, which we can visit. so use backtrack. Pop 'f' from stack.



step 6: There is no new vertex to be visited from 'c'. so we use backtrack pop 'c' from the stack.

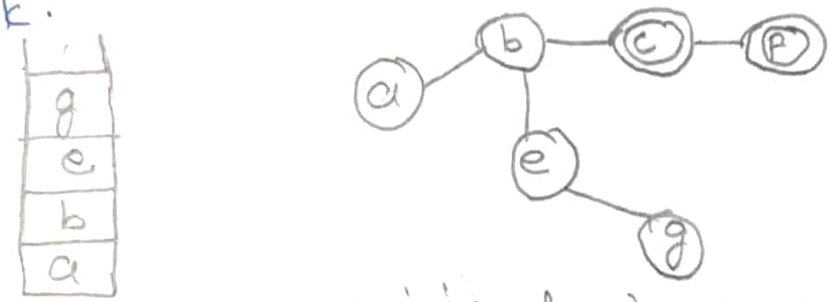


step 7: visit any adjacent vertex of b which is not visited 'e'. Push 'e' onto The stack.



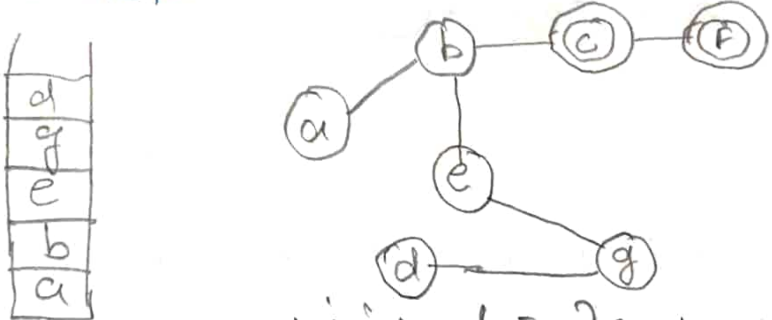
visited = {a, b, c, f, e}

step 8: visit any adjacent vertex of 'e' which is not visited 'g'. push 'g' on to the stack.



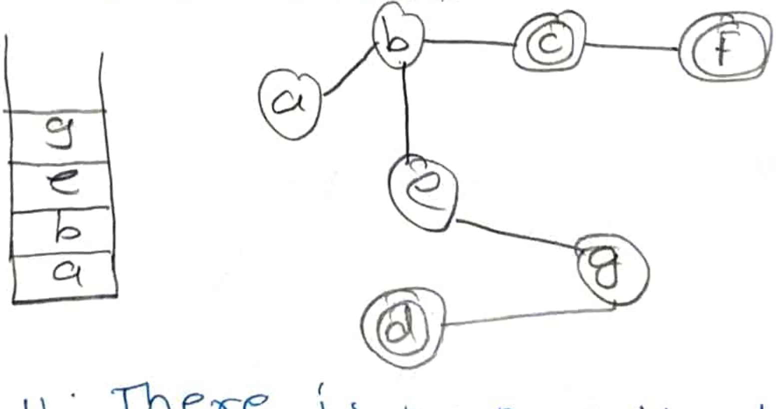
visited = {a, b, c, f, e, g}

Step 9: visit any adjacent vertex of g which is not visited 'd' push 'd' on to the stack.

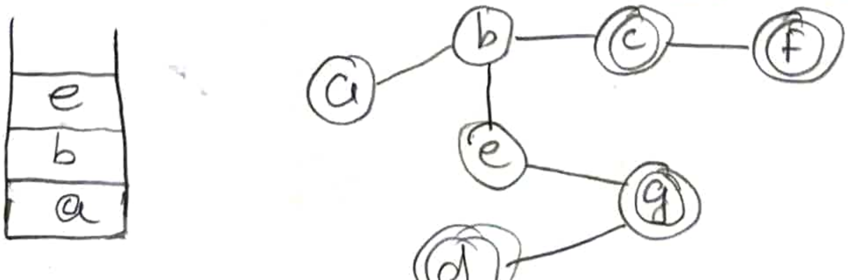


visited = {a, b, c, f, e, g, d}

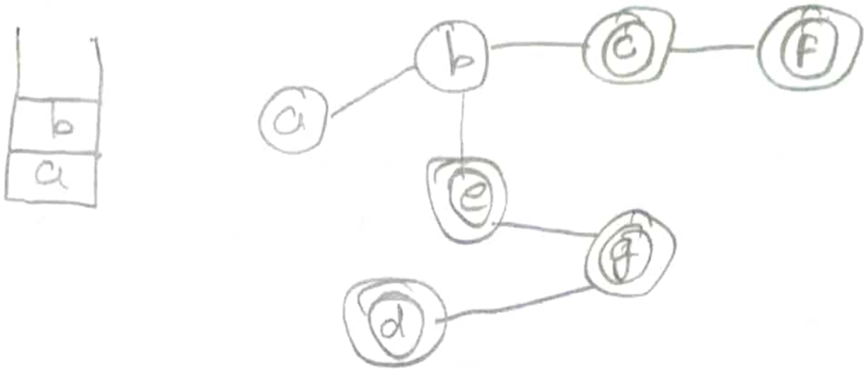
step 10: There is no new vertex to be visited from 'd' so use backtrack pop d. From stack.



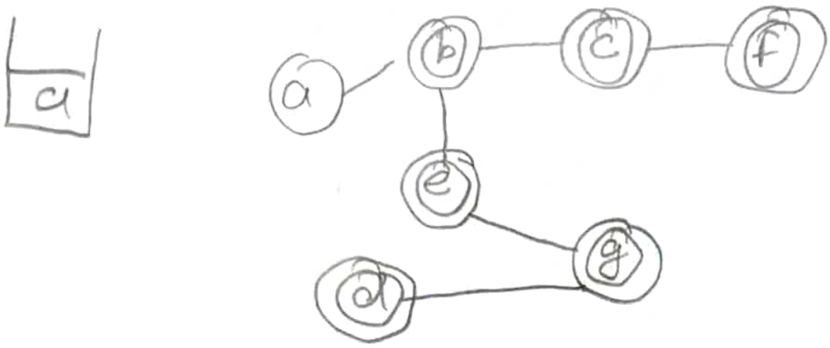
step 11: There is no new vertex to be visited from g so use backtrack. pop g.



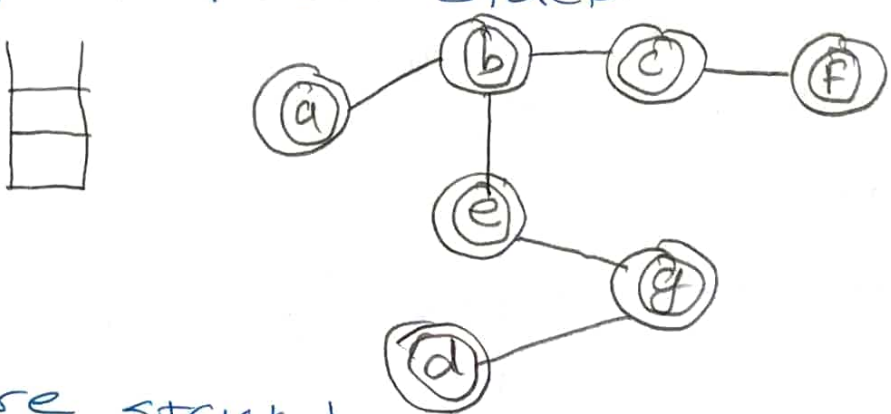
step 12: There is no new vertex to be visited from e. so use backtrack Pop e from stack.



step 13: There is no new vertex to be visited from b. so use backtrack. Pop 'b' from stack.



Step 14: There is no new vertex to be visited from a. so use backtrack POP 'a' from stack.



Here stack become empty now. so stop DFS traversal. and final result of traversal = {a, b, c, f, e, g, d}

(draw above graph Here again)

*** Breadth first search:**

Breadth first search algorithm travers a graph in a breadthward and uses a queue to remember to get the next vertex to start a search. when dead end occurs is any iteration.

*** Algorithm for BFS:**

step 1: select any one vertex to start traversing.

step 2: Insert that vertex at the front of the queue

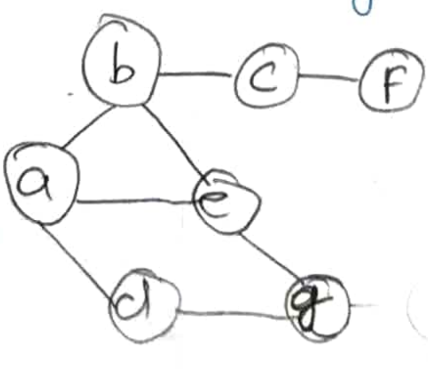
step 3: make a list of the nodes as visited that are close to that vertex

step 4: dequeue the nodes once they are visited.

step 5: Repeat the action untill queue is empty.

step 6: stop.

example: consider following graph and travers using BFS.



→ lets take 'a' as a starting vertex

Step 1: ~~ad~~ insert 'a' to the queue

Mark 'a' as visited.

$V = \{a\}$

a	

Step 2: Now queue is not empty.

- so delete 'a' from queue
- Add all ~~a~~ adjacent vertices of 'a' to the queue

$V = \{a, b, d, e\}$

b	d	e	
↑ Front		↑ rear	

Step 3: as queue is not empty.

- delete 'b' from queue.
- Add all adjacent & unvisited vertices of 'b' in queue.

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

d	e	c	
↑ Front		↑ rear	

Step 4: as queue is not empty

- delete d from queue.
- add all adjacent & unvisited vertices of 'd' in queue.

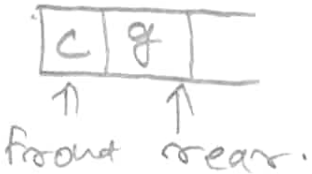
$V = \{a, b, d, e, c, g\}$

e	c	g
Front		rear

Step 5 : delete 'e' from queue.

- There are no unvisited vertices from 'e'.
- so queue is as:

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



Step 6 : delete 'c' from queue.

- Insert all adjacent and unvisited vertices of 'c' to the queue.

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

Step 7 : delete 'g' from queue.

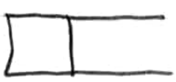
- There is no unvisited vertex is remaining from g
- so queue is as.

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

Step 8 : delete f from queue.

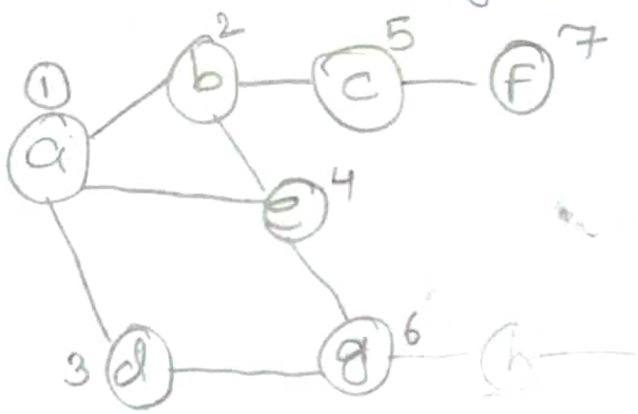
- There is no unvisited vertex is pending from f.
- so queue is as

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



- Here queue is empty now
- so, stop
- The sequence in which vertices are visited by BFS as follow.

a, b, d, e, c, g, f, h, i



Q) Differentiate between BFS & DFS