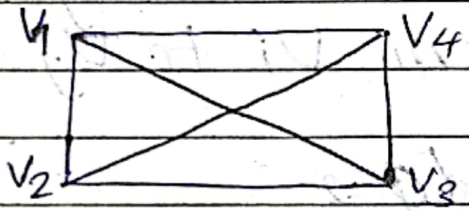
GRAPHS

→ Non linear data structure
→ Contains Vertices & Edges

Terminology:
G = (V, E)

→ Set V = set of vertices or nodes
→ Set E = Set of Edges

For e.g.



G1: [UnDirected graph]

$V = \{V_1, V_2, V_3, V_4\}$

$E = \{(V_1, V_2), (V_1, V_3), (V_1, V_4), (V_2, V_3), (V_2, V_4), (V_3, V_4)\}$

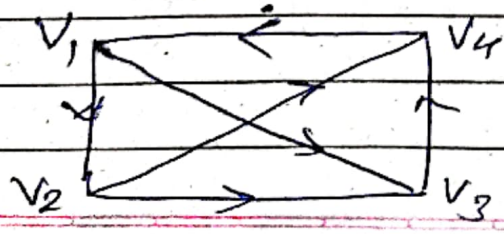
Digraph:

→ Directed graph: All edges are directed

OR

G G = < V, E >, where V is the set of all vertices & E is a set of ordered pairs

e.g.



→ In Digraph, (V_i, V_j) is in E then edge is directed from V_i to V_j

→ In undirected graph (V_i, V_j) is unordered

Weighted Graph:

All edges have weights assigned

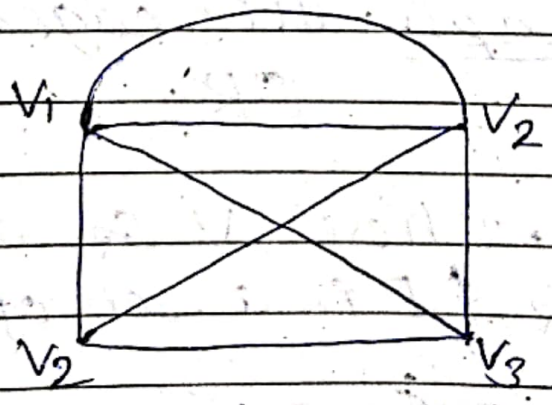
Adjacent Vertices:

V_i is adjacent to V_j if there is a edge from V_i to V_j

Parallel Edges

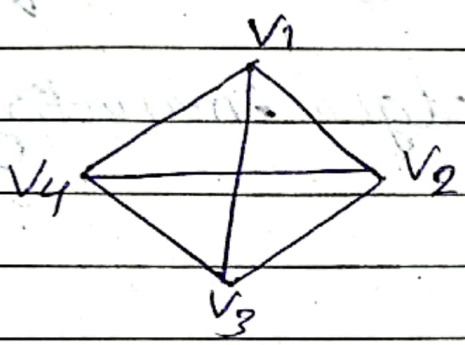
More than one edge between same pair of vertices

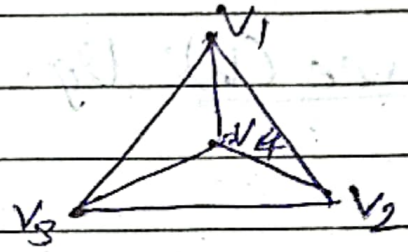
Multigraph → Either self-loop or parallel edges or ~~both~~ both - e.g.



Simple Graph: No self loop or parallel edges.

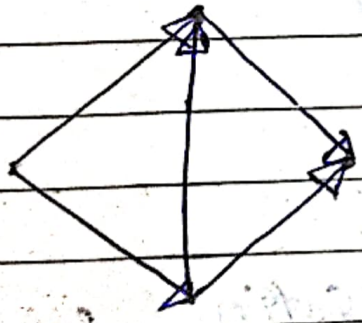
Complete graph: Each vertex V_i is adjacent to every other vertex V_j in G.
There are e edges from any vertex to all other vertices. e.g.

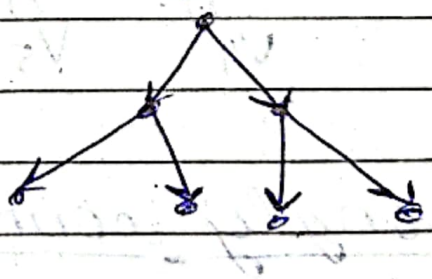




Acyclic graph:

If a digraph that doesn't have cycle (path starting & ending on same vertex)
e.g.

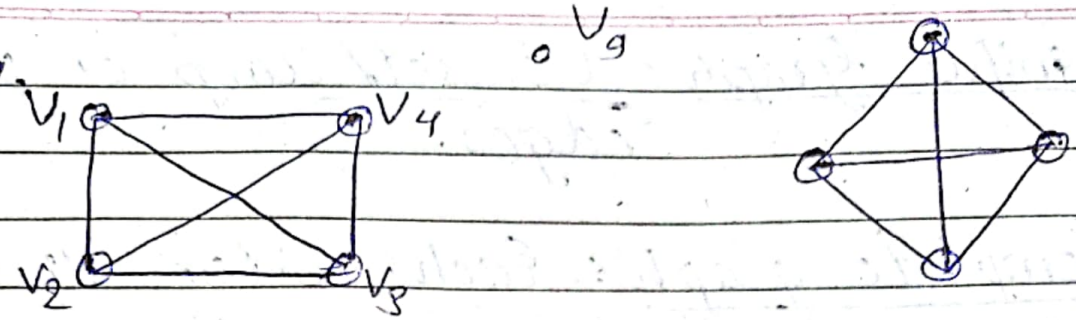




Isolated Graph:

A vertex is isolated if there is no edge from any other vertex to another.

E.g.



Degree of a vertex: No. of edges connected.

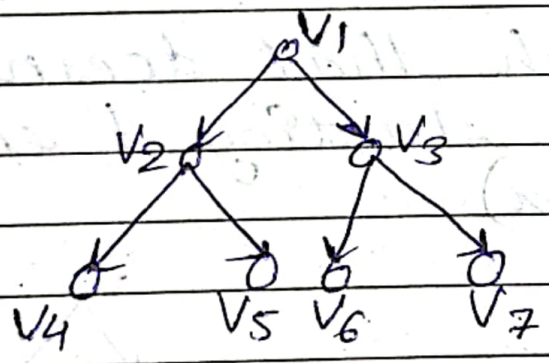
Indegree (V): No. of edges coming to V

Outdegree (V): No. of edges emanating from V

Pendant Vertex

Indegree = 1 & Outdegree = 0

E.g.



Pendant.
V_1, V_4, V_5, V_6

Strongly connected:

For every pair of distinct vertices V_i, V_j, there is a directed path from V_i to V_j & from V_j to V_i.

REPRESENTATION

→ Matrix or Sequential
→ Linked List representation

Matrix:
→ Adjacency → Incidence → Path

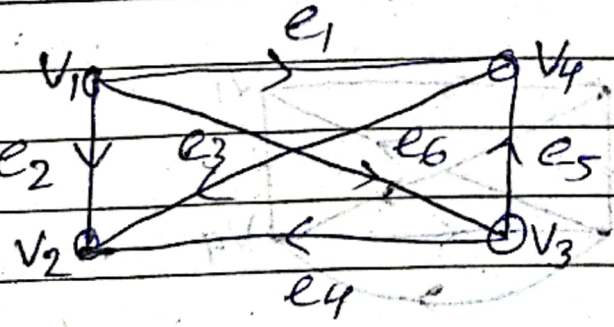
Adjacency

$$a_{ij} = \begin{cases} 1, & \text{Edge bet}^n\ v_i \& v_j \\ 0, & \text{No Edge} \end{cases}$$

No. of 1's is equal to no. of edges
Also called bit matrix or boolean matrix.

Notes: → Does not depend on ordering of vertices
→ If graph is undirected, then adjacency matrix would be symmetric. i.e. $[a_{ij}] = [a_{ji}]$

E.g.



$$A = \begin{array}{c} \\ V_1 \\ V_2 \\ V_3 \\ V_4 \end{array} \begin{array}{c} \begin{array}{cccc} V_1 & V_2 & V_3 & V_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

Incidence:

If R^{th} edge is (V_i, V_j), R^{th} column has a value 1 in the i^{th} row, -1 in j^{th} row & 0 otherwise.

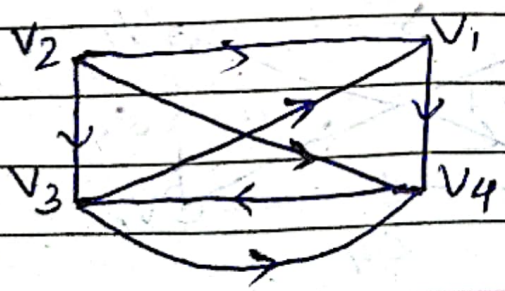
$$I = \begin{array}{c} \\ V_1 \\ V_2 \\ V_3 \\ V_4 \end{array} \begin{array}{c} \begin{array}{cccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 & -1 \\ 0 & 0 & -1 & 1 & 1 & 0 \\ -1 & 0 & 0 & 0 & -1 & 1 \end{bmatrix} \end{array}$$

Path Matrix:

$$P_{ij} = \begin{cases} 1 & \rightarrow \text{Path from } V_i \text{ to } V_j \\ 0 & \rightarrow \text{otherwise} \end{cases}$$

If A is adj. matrix & P be the path matrix of G. $P_{ij} = 1$ iff there is a non-zero entry in matrix.

$$B_n = A^1 + A^2 + A^3 + \cdots + A^n$$



No. of nodes, $n = 4$.

A^1, A^2, A^3, A^4

$$A^1 = \begin{array}{c|cccc} & V_1 & V_2 & V_3 & V_4 \\ V_1 & 0 & 0 & 0 & 1 \\ V_2 & 1 & 0 & 1 & 1 \\ V_3 & 1 & 0 & 0 & 1 \\ V_4 & 0 & 0 & 1 & 0 \end{array}$$

NOTE:

$a_k(i, j) = ij^{th}$ entry in A^k

$a_1(i, j) = a_{ij}$ no. of paths of length 1 from V_i to V_j

$a_k(i, j) =$ " —— " k

" —— "

$$A^2 = \begin{array}{c|cccc} & V_1 & V_2 & V_3 & V_4 \\ V_1 & 0 & 0 & 1 & 0 \\ V_2 & 1 & 0 & 1 & 2 \\ V_3 & 0 & 0 & 1 & 1 \\ V_4 & 1 & 0 & 0 & 1 \end{array}$$

$$A^3 = \begin{array}{c|cccc} & V_1 & V_2 & V_3 & V_4 \\ V_1 & 1 & 0 & 0 & 1 \\ V_2 & 1 & 0 & 2 & 2 \\ V_3 & 1 & 0 & 1 & 1 \\ V_4 & 0 & 0 & 1 & 1 \end{array}$$

$A^4 =$

	V_1	V_2	V_3	V_4
V_1	0	0	1	1
V_2	2	0	2	3
V_3	1	0	1	2
V_4	1	0	1	1

$B^4 \Rightarrow B_4 =$

	V_1	V_2	V_3	V_4
V_1	1	0	2	3
V_2	5	0	6	8
V_3	3	0	3	5
V_4	2	0	3	3

Replace 1's & 0's

$P =$

	V_1	V_2	V_3	V_4
V_1	1	0	1	1
V_2	1	0	1	1
V_3	1	0	1	1
V_4	1	0	1	1

A graph is strongly connected if it has no zeros.

Linked Representation:

It contains two lists:

→ Node list → Edge list

Node list: Each element corresponds to a node in G

~~node~~

node	next	adj

Node node : name or key value

next : pointer to next

adj : pointer to 1st element of adjacency list of node, maintained in list edge.

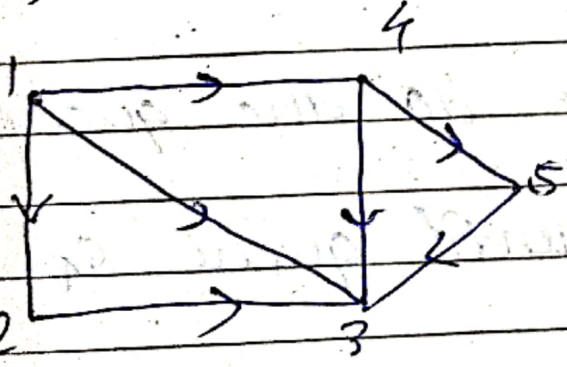
Edge list :

dest	link

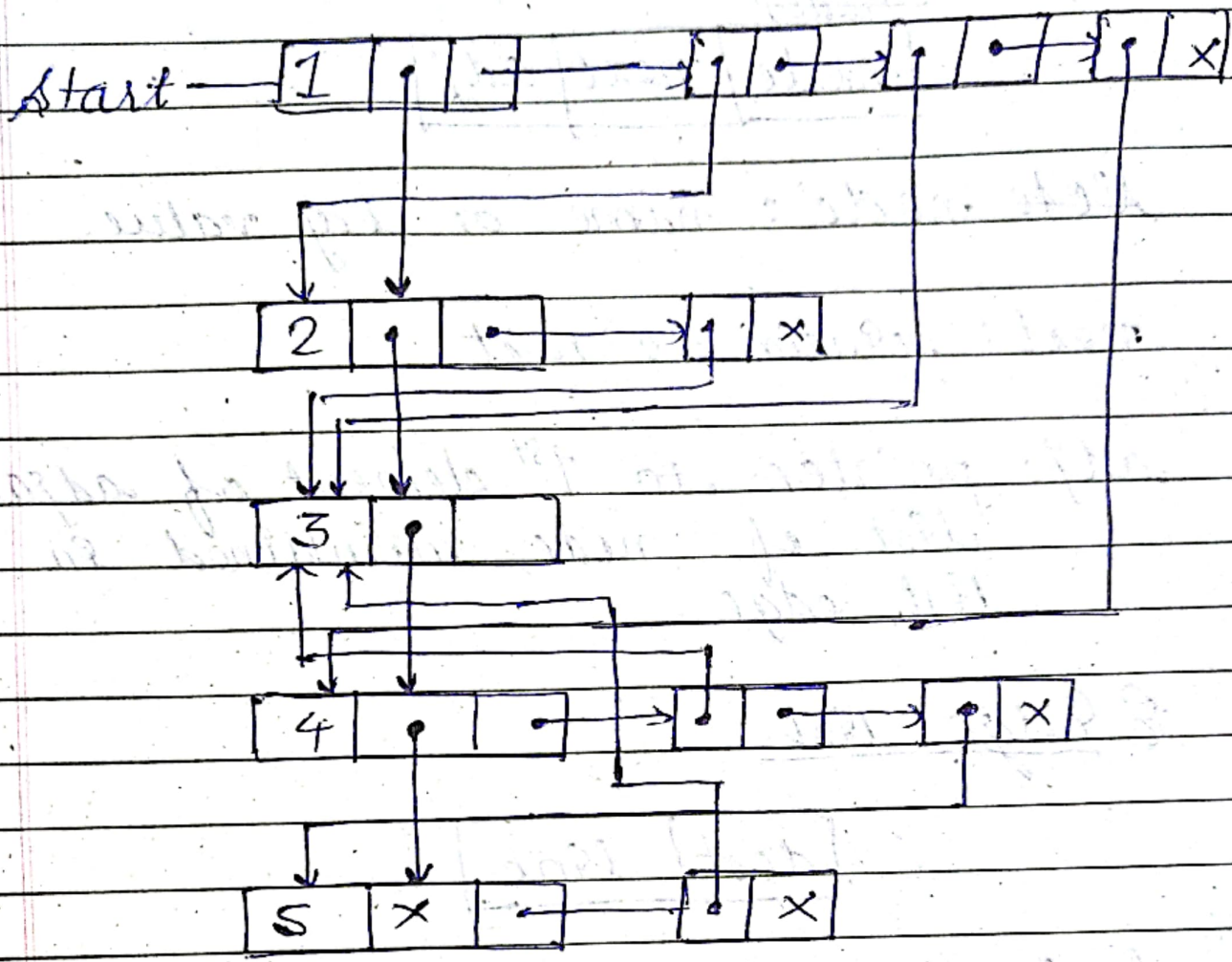
dest : loc" in the list node of the destination or terminal nodes in same adjacency list

link : links together edges with

E.g.



Node	Adjacency list
1	2, 3, 4
2	3
3	-
4	3, 5
5	3



Breadth - First - Search:

Algorithm:

(1) Initialize all nodes to the ready state

(2) Put starting node to the queue

(3) Repeat (4) & (5) until queue is empty

(4) Remove the front node of queue (N)

(5) Add to rear all the neighbours of N (End of step 3 loop)

BFS (Queue) (Level Order)

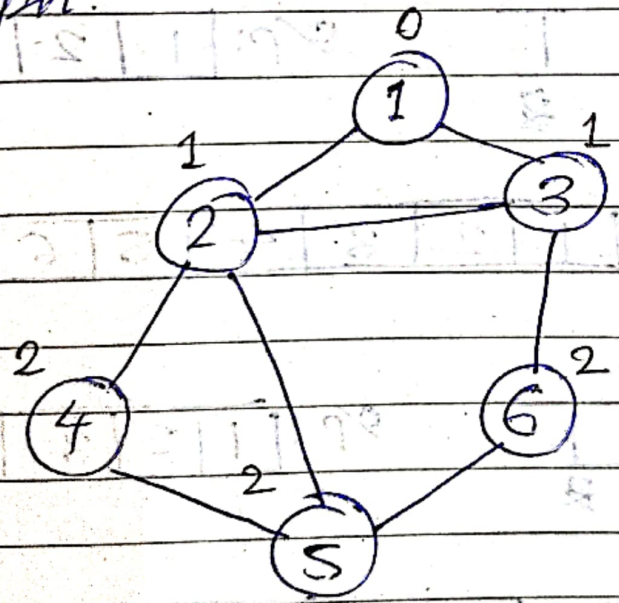
1 → Choose any vertex (V) as starting pt. & add it to Queue.

2 → Put ~~it~~ to the visited list & add it to output also Dequeue it
queue front

3 → Add all unvisited neighbours of V to queue ~~until it is empty~~

4 → Repeat 2 & 3 until queue is empty

Graph:



S-1:

1	2	3			

f ↑ R

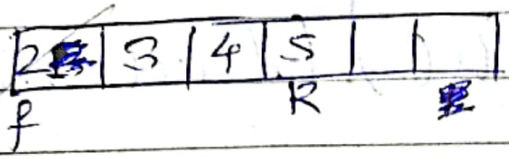
o/p :

1					

~~1~~	2	3			

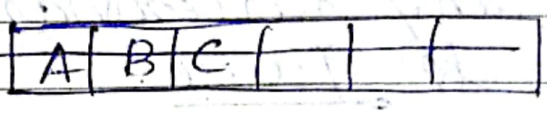
Q

S-2

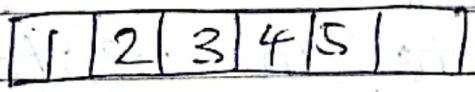


o/p:

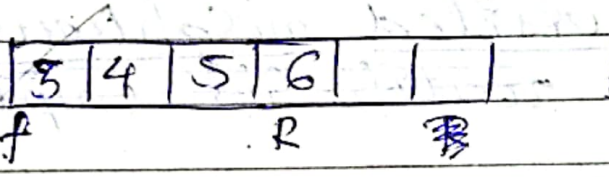
Visited



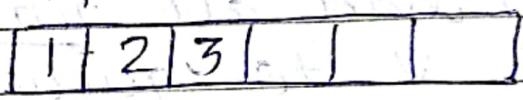
Visited



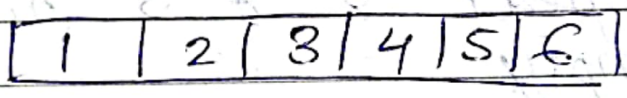
S-3



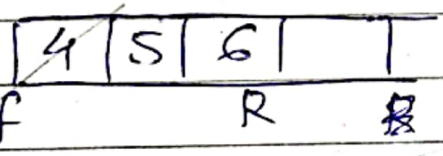
o/p



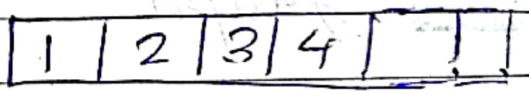
Visited



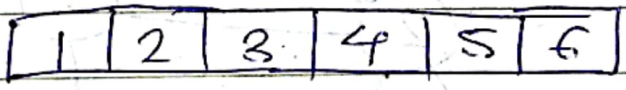
S-4



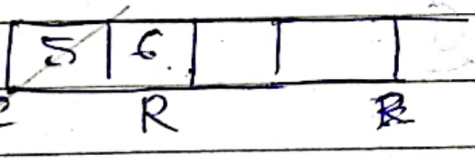
o/p:



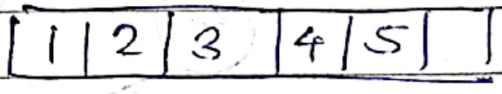
Visited



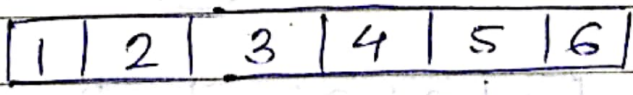
S-5



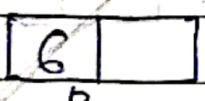
o/p:



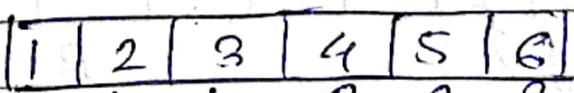
Visited



S-6

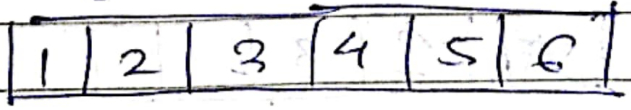


o/p:



Level: 0 1 1 2 2 2

Visited:



DFS (Stack)

1 → Choose any vertex V as starting pt. & add it to stack

2 → Put V to visited list & print V

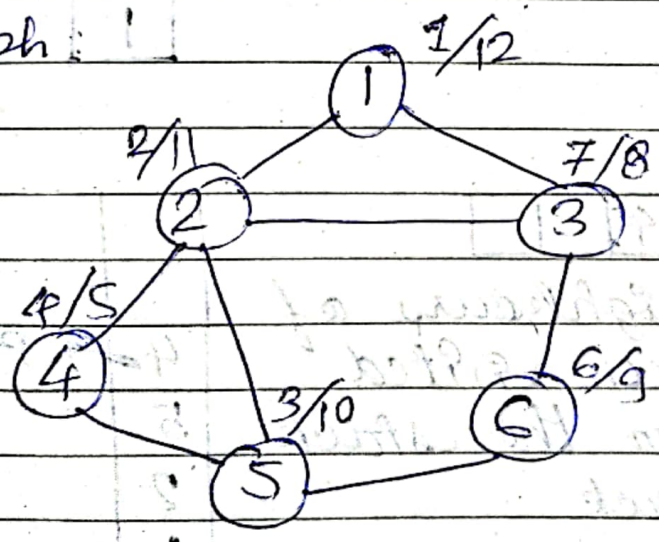
3 → Add any 1 unvisited neighbour of V to stack

4 → If current vertex has all its neighbours already visited, pop it from stack & backtrack

5 → Check remaining vertices on the stack ~~if backtrack~~ for any unvisited nodes

6 → Repeat till stack is empty

5-1 Graph:



S-1 o/p:

1					

Visited

Stack

1

S-2

o/p:

1	2				

Visited

1	2				

Stack

2
1

← 2

S-3: o/p:

1	2	5			

Visited

1	2	5			

Stack

5
2
1

S-4 o/p:

1	2	5	4		

As all neighbours of 4 have been visited pop it from the stack & backtrack

Visited

1	2	5	4		

Stack

4 → pop
5
2
1

S-5 o/p:

1	2	5	4	6	

Vis. | 1 | 2 | 5 | 4 | 6 |

6
5
2
1

Now check for unvisited nodes from 5 & add to stack

S-6 o/p:

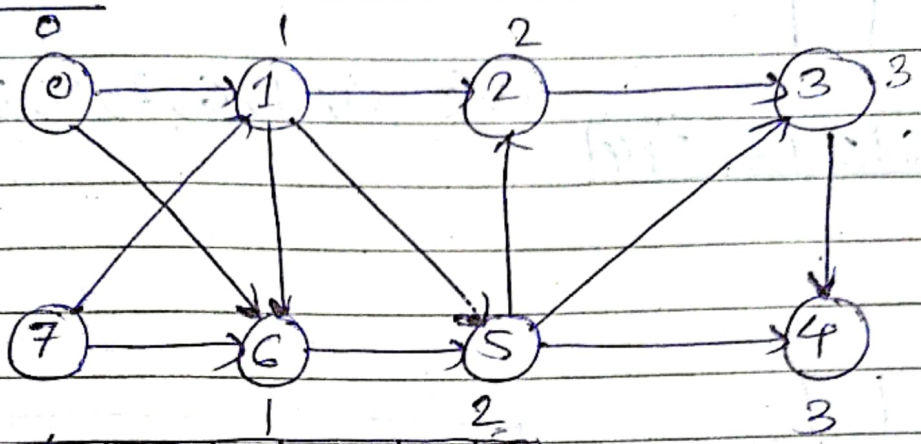
1	2	5	4	6	3

Vis. | 1 | 2 | 5 | 4 | 6 | 3 |

3
6
5
2
1

As all vertices have been visited pop & empty the stack.

BFS



S-1

Queue: 0 1 6 5

o/p: 0

Vis: 0 1 6

S-2

q: 1 6 2 5

o/p: 0, 1,

0 1 6 2 5

S-3:

q: 6 2 5

0 1 6 2 5

o/p: 0, 1, 6

S-4

2 5 3

o/p: 0, 1, 6, 2

0 1 6 2 5 3

S-5:

q: 5 3 4

o/p: 0, 1, 6, 2, 5

Vis: 0 1 6 2 5 3 4

S-6:

Q: | 3 | 4 | | |

o/p: 0, 1, 6, 2, 5, 3

Vis. | 0 | 1 | 6 | 2 | 5 | 3 | 4 |

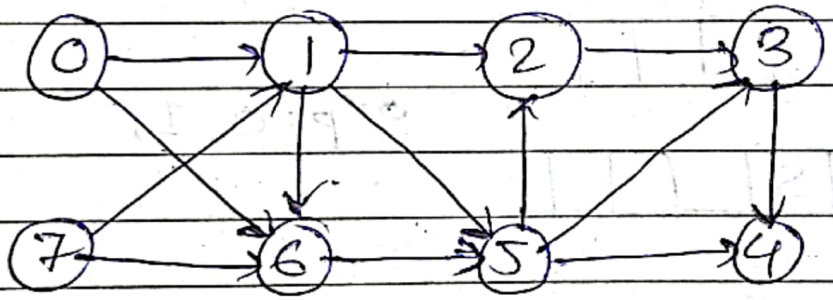
S-7:

| 4 | | | | |

o/p: 0, 1, 6, 2, 5, 3, 4

| 0 | 1 | 6 | 2 | 5 | 3 | 4 |

DFS:



S-1: o/p: 0, 1

Visited: 0, 1

1
0

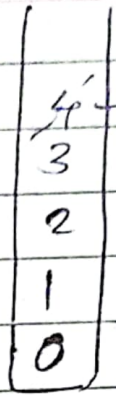
S-2: o/p: 0, 1, 2, 3

Vis.: 0, 1, 2, 3

3
2
1
0

S-3 O/p: 0, 1, 2, 3, 4

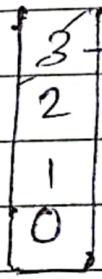
Vis. : 0, 1, 2, 3, 4



4 → All neigh. have been visited

S-4: O/p: 0, 1, 2, 3, 4

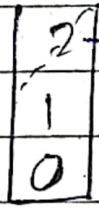
Vis. : 0, 1, 2, 3, 4



3 → All neigh. visited

S-5: O/p: 0, 1, 2, 3, 4

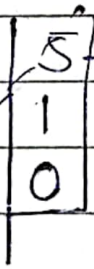
Vis. : 0, 1, 2, 3, 4



2 → All neigh. vis.

S-6: O/p: 0, 1, 2, 3, 4, 5

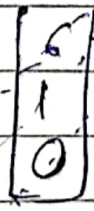
Vis: 0, 1, 2, 3, 4, 5



5 → All neigh. vis.

S-7: O/p: 0, 1, 2, 3, 4, 5, 6

Vis: 0, 1, 2, 3, 4, 5, 6



6 → Empty stack as all neigh. of everyone have been vis.

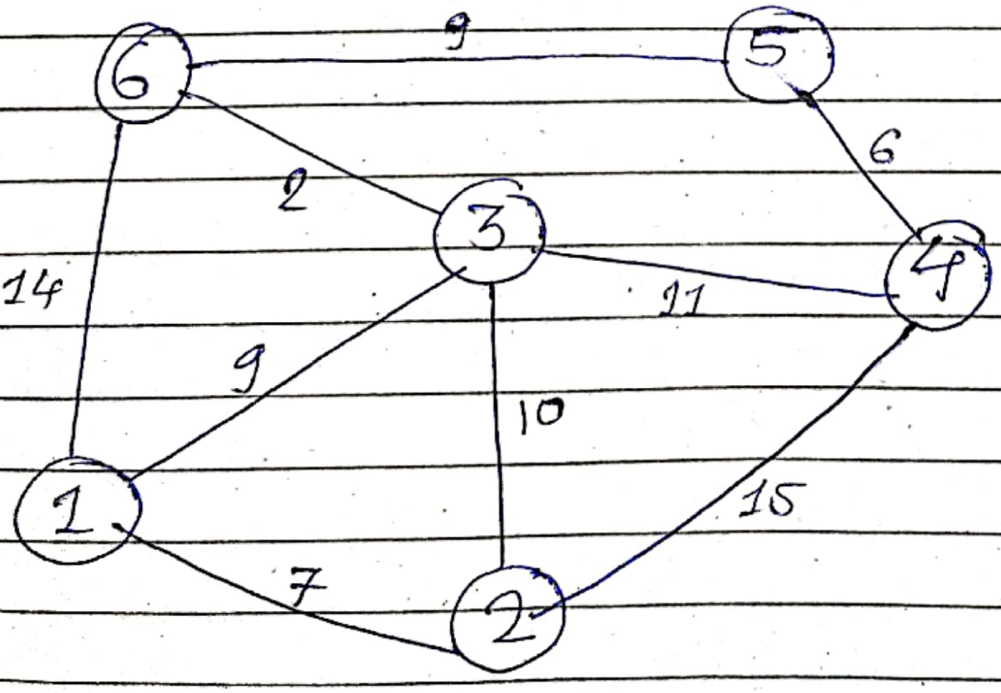
DJIKSTRA'S ALGORITHM

1) Assign a distance value to all vertices in the input graph. Initialize all distance values as ∞.
Assign dist. value as 0 for source vertex

2) While set is not empty
   A. Pick a vertex u from set that has min. distance value

   B. Update distance value of all adjacent vertices of u

E.g.



Source	Destination				
1	2	3	4	5	6
	∞	∞	∞	∞	∞
choose min.	7	9	∞	∞	14
now (1,2)	7	9	22	∞	14
1→2→3 is more so don't change (1,2,3)	7	9	20	∞	11
(1,2,3,6)	7	9	20	20	11
choose any one 1,2,3,6,4	7	9	20	20	11
~~1,2,3,6,4,5~~ 1,2,3,6,4,5	11		11		11