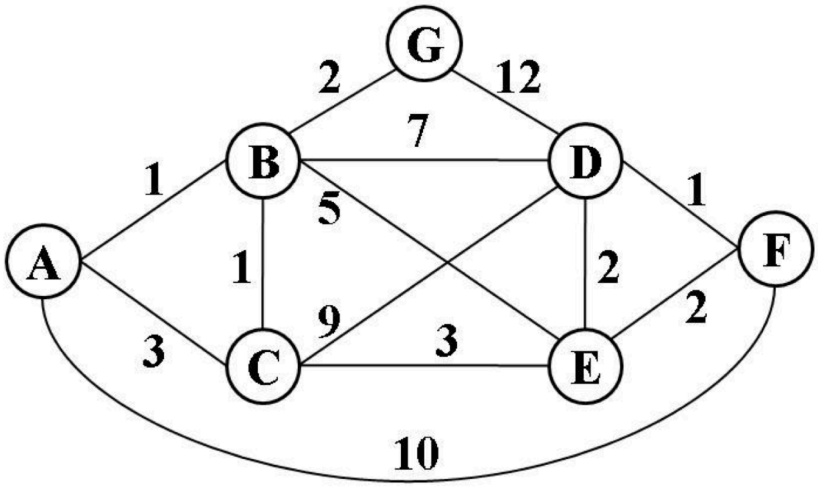
Name: Shikha Singh
Roll no:25

ASSIGNMENT 3 DSA_2

1. Consider the following undirected, weighted graph:



Step through Dijkstra's algorithm to calculate the single-source shortest paths from A to every other vertex. Show your steps in the table below. Cross out old values and write in new ones, from left to right within each cell, as the algorithm proceeds. Also list the vertices in the order in which you marked them known. Finally, indicate the lowest-cost path from node A to node F.

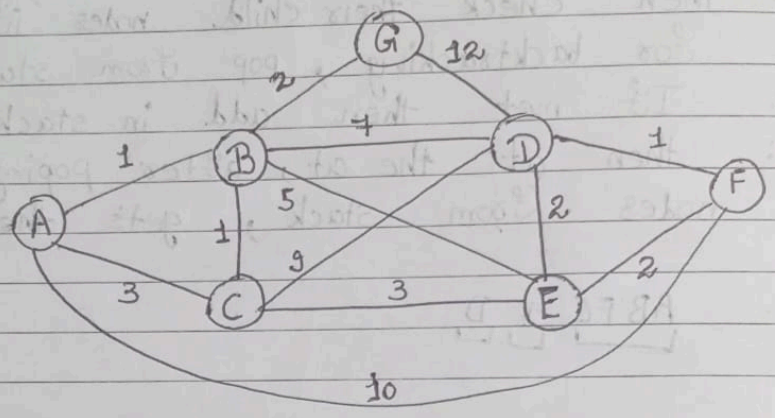
Ans:

Name :- shikha singh

Roll no :- 25

Assignment 3 dsa – 2

(1)



→

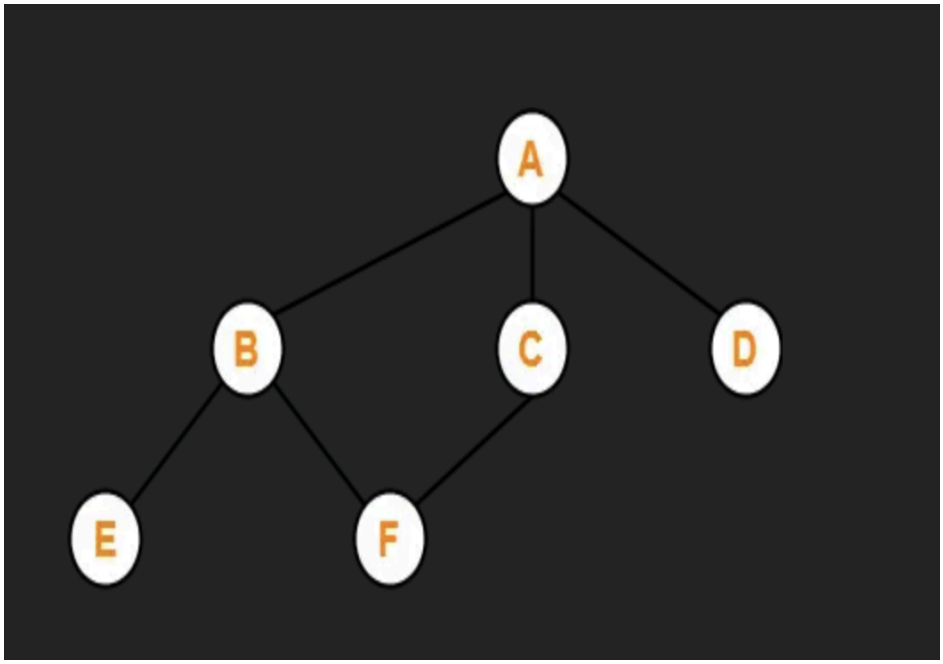
	A	B	C	D	E	F	G
	[0]	∞	∞	∞	∞	∞	∞
A	0	[1]	3	∞	∞	10	∞
B	0	1	[2]	8	6	10	3
C	0	1	2	8	5	10	[3]
G	0	1	2	8	[5]	10	3
E	0	1	2	7	5	[7]	3
F	0	1	2	[7]	5	7	3
D	0	1	2	7	5	[7]	3

↳ A to F

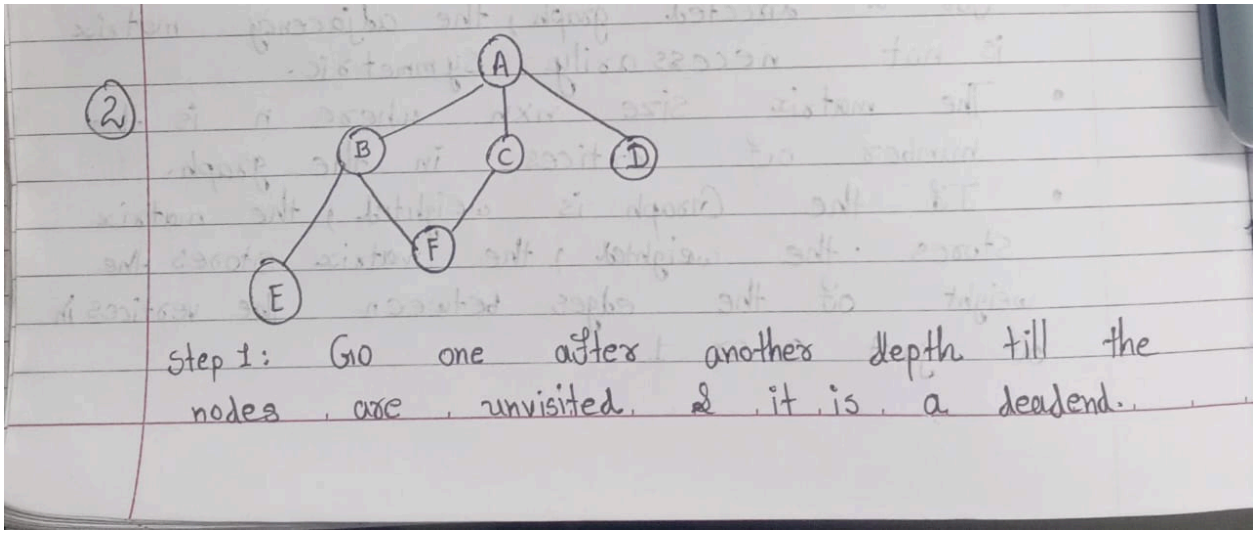
↓

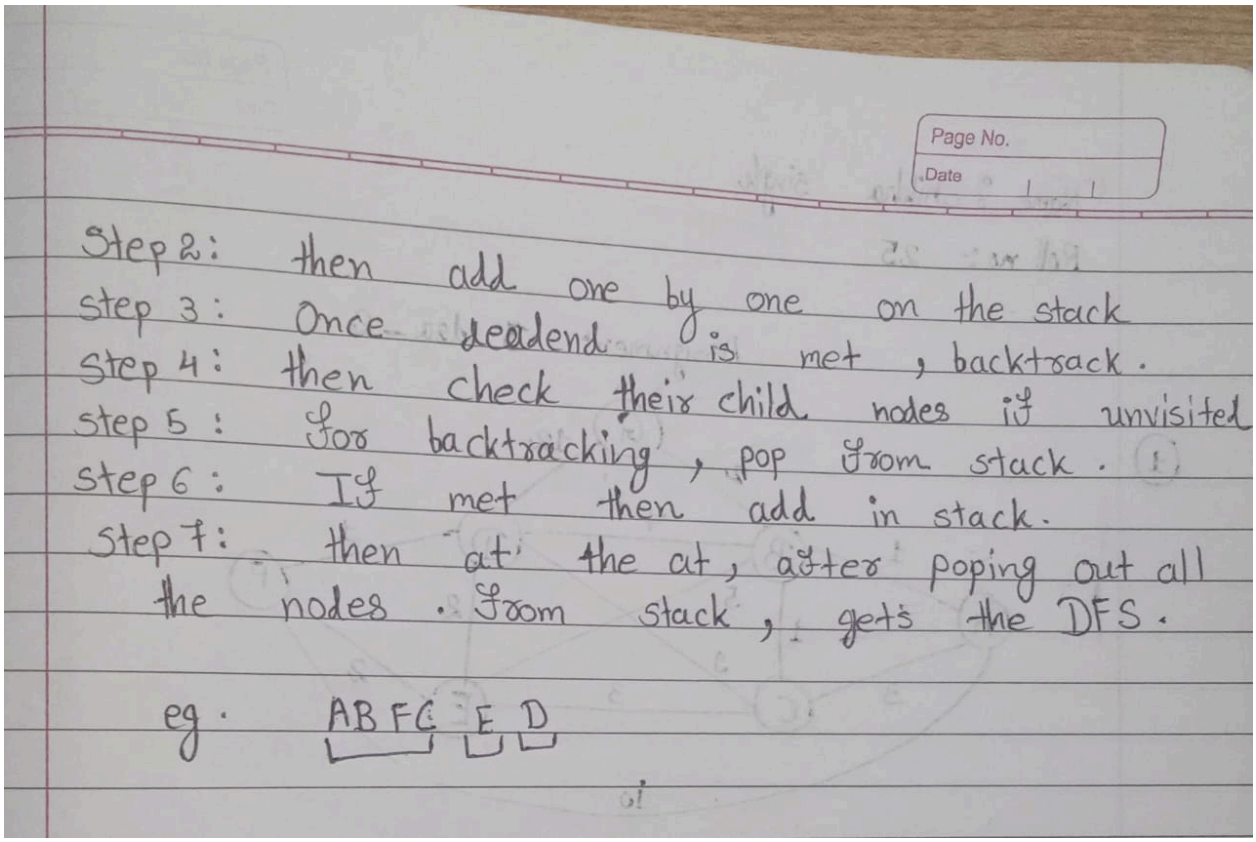
[7]

2. Consider the following graph



Compute DFS and write the sequence in steps and show how the backtracking will perform?





3. Explain adjacency matrix representation for a graph with the help of an example?
   Ans:

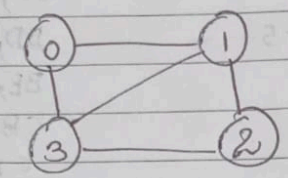
(3) An adjacency matrix is a way to represent a graph using a 2D array (C or matrix). In this representation, the matrix's raws & columns correspond to the graph's vertices. The matrix entries indicate whether pairs of vertices are adjacent. (ie., whether there is an edge between them)

key points :-

- For an undirected graph, the matrix will be symmetric
- For a directed graph, the adjacency matrix is not necessarily symmetric.
- The matrix size nxn where n is the number of vertices in the graph.
- If the Graph is weighted, the matrix stores the weight of the edges between the vertices in of just 0 or 1

Date

Example:

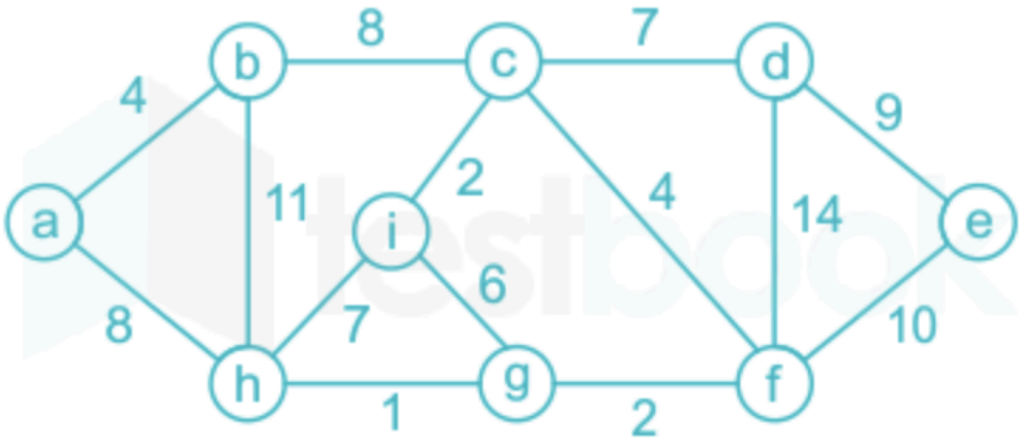


we create 4x4 matrix

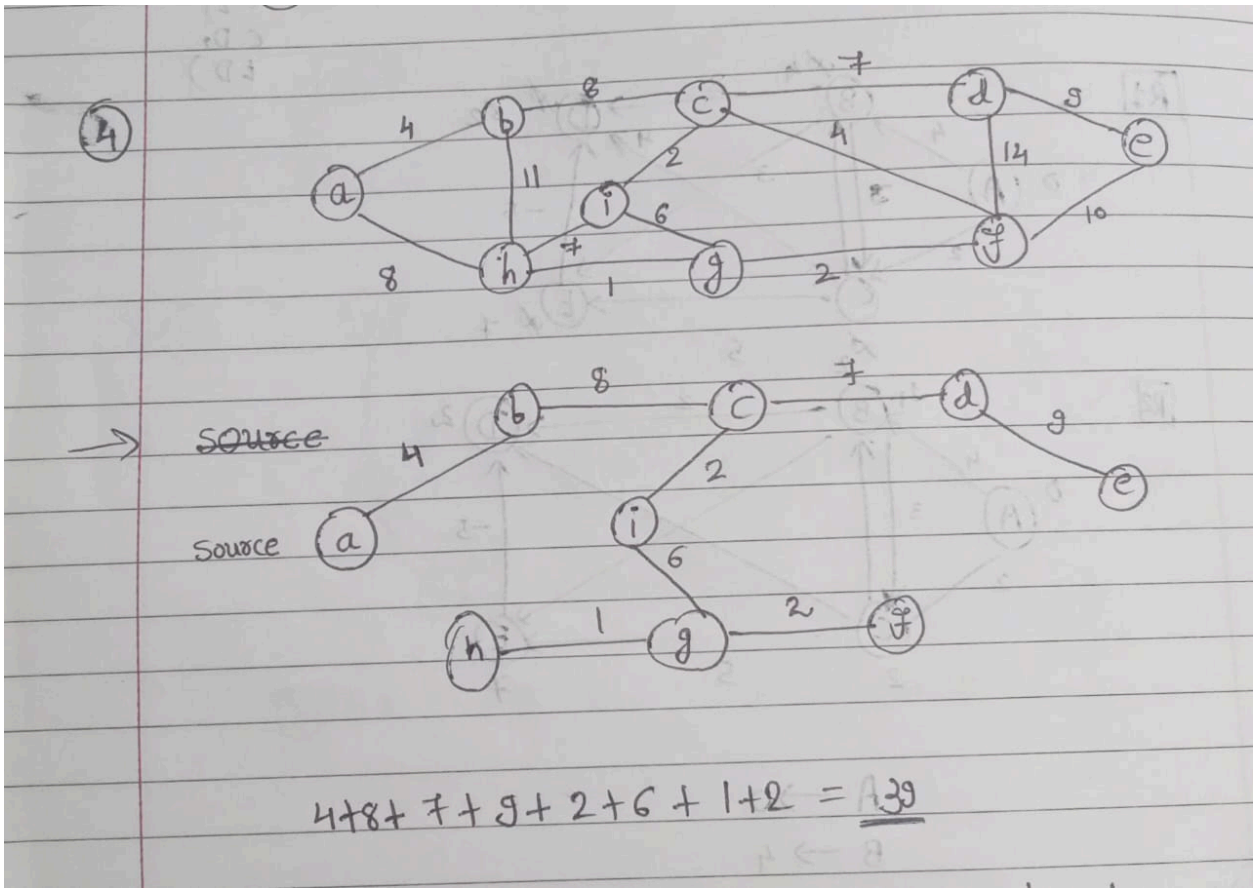
0	1	0	1
1	0	1	1
0	1	0	1
1	1	1	0

4.

Consider the undirected graph below:



Apply Prim's algorithm to find the minimum cost spanning tree?



5. Write down Prim's Algorithm step by step to find the minimum cost spanning tree?

Ans:

Prim’s Algorithm (Step by Step)

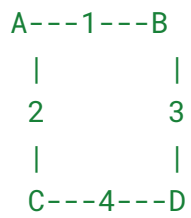
Prim's algorithm is a **greedy algorithm** used to find the **Minimum Cost Spanning Tree (MST)** of a connected, undirected graph. It starts with a single node and gradually adds edges to the MST until all nodes are included.

Steps for Prim’s Algorithm:

1. **Initialize**:
   - Start with any arbitrary node as the source (initial node).
   - Initialize the MST set as empty and mark the starting node as included in the MST.
   - Set the distance (cost) of all nodes to infinity (∞) except for the starting node, which has a distance of 0.
2. **Select the Minimum Edge**:
   - From the nodes included in the MST, find the edge with the smallest weight that connects to a node outside the MST.
3. **Add the Edge**:
   - Add the selected edge and the corresponding vertex to the MST set.
4. **Update Distances**:
   - Update the distances (costs) of the adjacent vertices to the newly added vertex. If a shorter path is found, update the distance.
5. **Repeat**:
   - Repeat steps 2 to 4 until all the vertices are included in the MST.

Example:

Given the graph below (weights between nodes shown):



6. Start with node `A`, initialize distances for other nodes (`B: 1`, `C: 2`, `D: ∞`).
7. Select the edge `(A-B)` with cost `1`.
8. Add edge `(A-B)` to the MST.
9. Update distances: New distance for `D` is `3` through `B`.
10. Select edge `(A-C)` with cost `2`.

11. Add edge `(A-C)` to the MST.
12. Repeat until the MST is complete.

6.Write down Kruskal Algorithm step by step to find the minimum cost spanning tree?
Ans:

Kruskal's Algorithm (Step by Step)

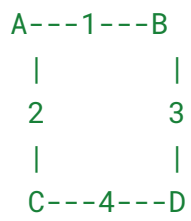
Kruskal's algorithm is another **greedy algorithm** used to find the **Minimum Cost Spanning Tree (MST)** of a graph. It works by selecting the smallest edges while ensuring no cycles are formed.

Steps for Kruskal's Algorithm:

1. **Sort Edges by Weight**:
   - Sort all the edges of the graph in non-decreasing order of their weights.
2. **Initialize Sets for Each Vertex**:
   - Each vertex starts in its own set (disjoint sets). These sets are used to detect cycles.
3. **Add the Smallest Edge**:
   - Pick the smallest edge from the sorted list of edges.
   - Check if adding this edge forms a cycle using the union-find algorithm (detect whether the vertices of the edge are in the same set).
   - If no cycle is formed, add the edge to the MST and merge the sets of the two vertices.
4. **Repeat**:
   - Continue selecting the smallest available edge that does not form a cycle, and add it to the MST.
5. **Finish**:
   - Repeat the process until the number of edges in the MST is equal to `V - 1` (where `V` is the number of vertices).

Example:

For the same graph:



- Sort edges: `(A-B: 1)`, `(A-C: 2)`, `(B-D: 3)`, `(C-D: 4)`.
- Start by adding edge `(A-B)`, no cycle, so add it.
- Add edge `(A-C)`, no cycle, so add it.
- Add edge `(B-D)`, no cycle, so add it.
- Edge `(C-D)` forms a cycle, so skip it.
- Final MST contains edges: `(A-B)`, `(A-C)`, `(B-D)`.

7.How does the Bellman Ford algorithm work?
Ans:

Bellman-Ford Algorithm (How it Works)

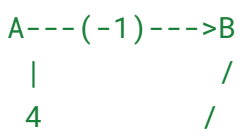
The **Bellman-Ford algorithm** is used to find the **shortest paths** from a source vertex to all other vertices in a weighted graph. It can handle graphs with **negative weights**, unlike Dijkstra's algorithm. It can also detect **negative weight cycles**.

Steps for Bellman-Ford Algorithm:

1. **Initialize**:
   - Set the distance to the source vertex as `0` and to all other vertices as ∞ (infinity).
2. **Relax All Edges**:
   - For each edge, if the distance to the destination vertex can be shortened by going through the source vertex of the edge, update the distance of the destination vertex.
   - This process is called "relaxation" and it is repeated for all edges.
   - Repeat the relaxation step for `V - 1` times, where `V` is the number of vertices in the graph.
3. **Check for Negative Weight Cycles**:
   - After `V - 1` relaxations, check all the edges again. If any distance can still be shortened, then there is a **negative weight cycle** in the graph.
4. **Result**:
   - If no negative weight cycles are detected, the algorithm returns the shortest paths from the source vertex to all other vertices.

Example:

Consider this graph with edge weights (negative weight included):

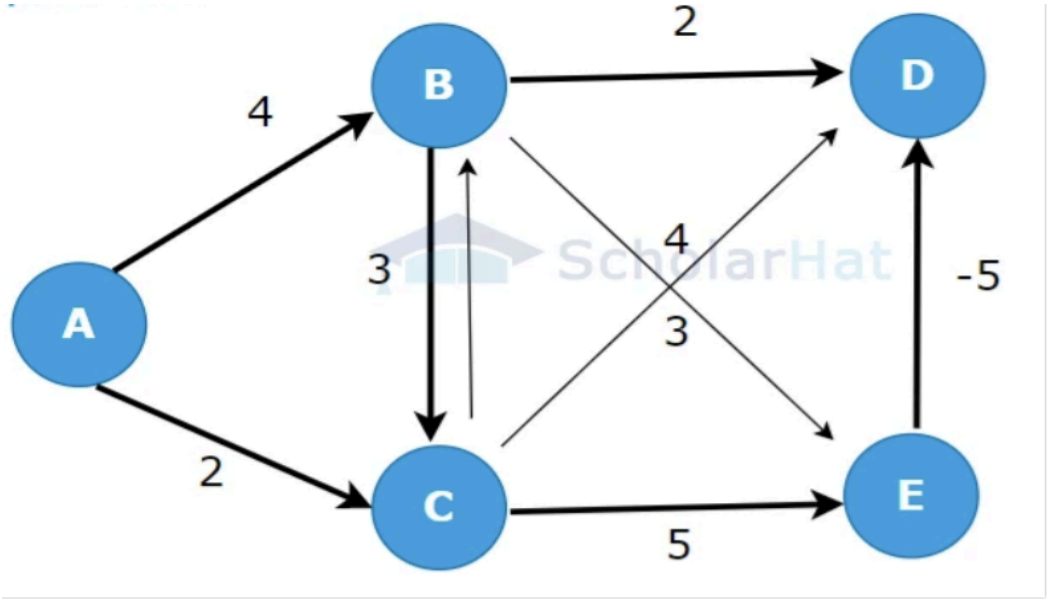


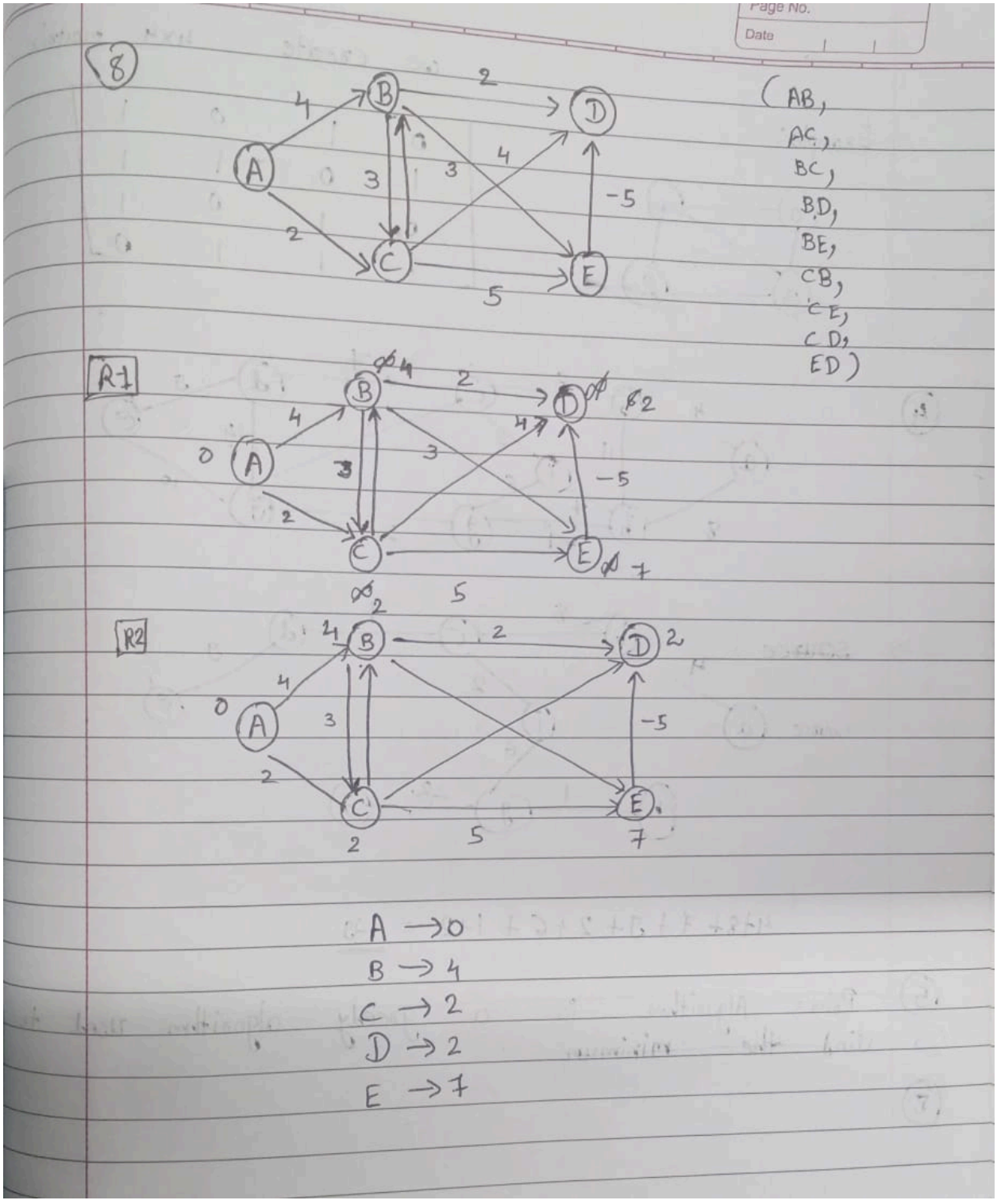
```
|        (-2)
C<----D
```

- Step 1: Initialize distances as `A = 0`, `B = ∞`, `C = ∞`, `D = ∞`.
- Step 2: Relax all edges:
  - `(A-B)` updates `B` to `-1`.
  - `(A-C)` updates `C` to `4`.
  - `(D-B)` updates `B` to `-3` (better path through `D`).
- Step 3: Repeat the process for `V - 1` iterations (3 times in this case).
- Step 4: Check for negative weight cycles by attempting another relaxation. If distances change again, a negative cycle exists.

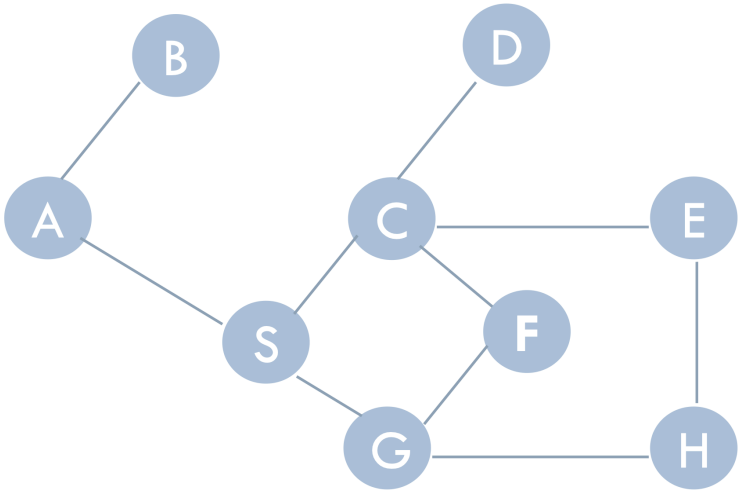
This algorithm will either return the shortest path for all vertices or indicate the presence of negative cycles.

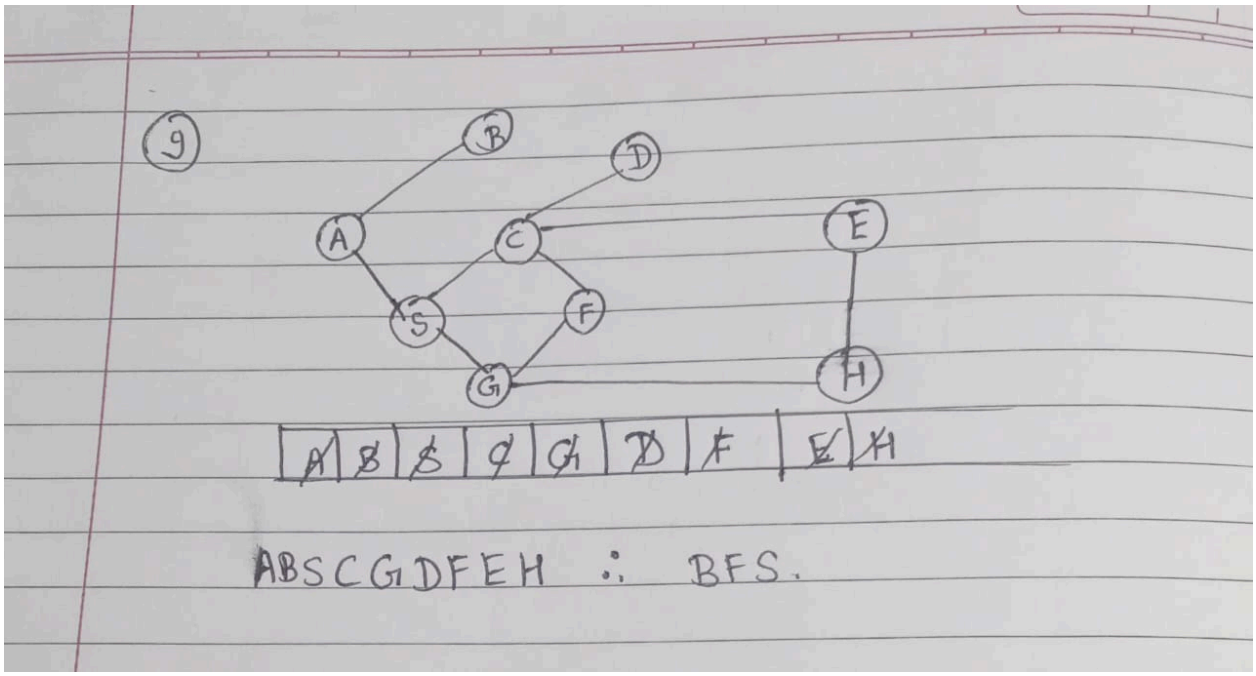
8.Apply Bellman Ford algorithm to the below graph



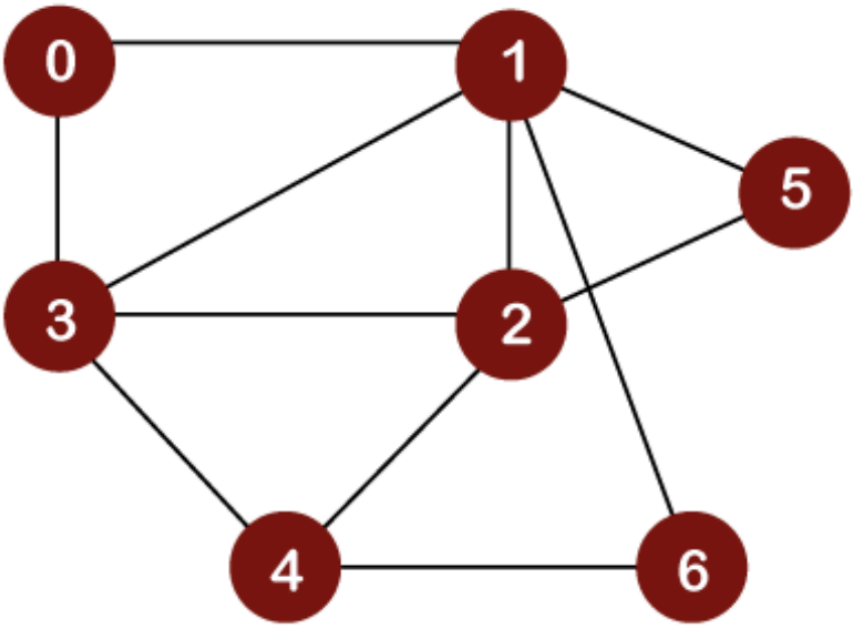


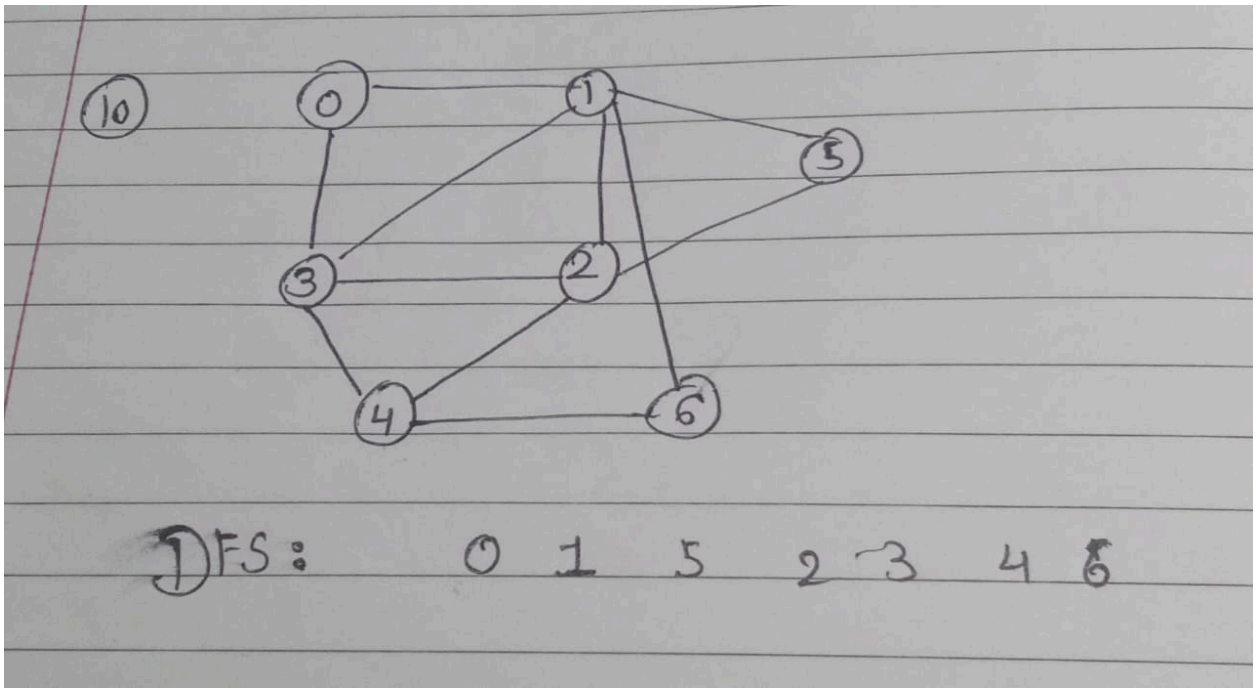
9.Compute BFS for the following graph





10. Compute DFS for the following graph


0
1
5
3
2
4
6


10
0
1
5
3
2
4
6
DFS: 0 1 5 2 3 4 6