GÖKHAN DEMİR 26562 CS300 HW5 FALL 2020

QUESTION 1

Starting from S, I am tracing the operations of the Dijkstra's weighted shortest path algorithm on the graph given in Figure 1.

-Initializing
Visited nodes={}
Unvisited nodes={S,A,B,C,D,E,F,G}
Distance=[TO S=0, TO A,B,C,D,E,F,G= ∞]

- S is visited. Let's update visited and unvisited nodes.
Visited nodes={S} ||| Unvisited nodes={A,B,C,D,E,F,G}
From vertex S, A and B vertices can be reached. Therefore, we need to update distances.
Distance=[TO S=0, TO A= 3, TO B=2, TO C,D,E,F,G= ∞]

- It is time to choose the shortest edge to go. A is more far than B since 3>2. Therefore, our next stop is vertex B. Vertex B will be added to the visited ones. I need to update distances as well.
Visited nodes={S,B} ||||| Unvisited nodes={A,C,D,E,F,G}
From vertex B, (A,S and D) vertices can be reached. Only need to change distance to D since the other is greater than the existing one.
Distance=[TO S=0, TO A= 3, TO B=2, TO D=2+1=3, TO C,E,F,G= ∞]

- The shortest edge from vertex B is to vertex D. So adding vertex D to the visited nodes.
Visited nodes={S,B,D} ||||| Unvisited nodes={A,C,E,F,G}
From vertex D, (E,G and F) vertices can be reached. Need to update distances to E,F and G:
Distance=[TO S=0, TO A=3, TO B=2, TO D=3, TO E=4, TO G =5, TO F=5, TO C= ∞]

- The shortest edge from vertex D is to vertex E. Therefore, it is time to go to the vertex E and add it to the visited nodes.
Visited nodes={S,B,D,E} ||||| Unvisited nodes={A,C,F,G}
From vertex E, F and G vertices can be reached. Let's compare distances since distances to the both F and G already exist. New path to G is 4+3=7, however we already have 5 which is less than 7. New path to F from E is 4+2=6. But, again we already have 5 which is less than 6. Finally, no need to update distances to F or G.
Distance=[TO S=0, TO A=3, TO B=2, TO D=3, TO E=4, TO G =5, TO F=5, TO C= ∞]

- The shortest edge from vertex E is to vertex F. Now I am at vertex F and adding it to the visited nodes.
Visited nodes={S,B,D,E,F} ||||| Unvisited nodes={A,C,G}
From vertex F, vertex C can be reached. Therefore, I am updating the distance to C.
Distance=[TO S=0, TO A=3, TO B=2, TO D=3, TO E=4, TO G =5, TO F=5, TO C= 6]

- Went to vertex C from vertex F. Adding it to the visited ones.
Visited nodes={S,B,D,E,F,C} ||||| Unvisited nodes={A,G}

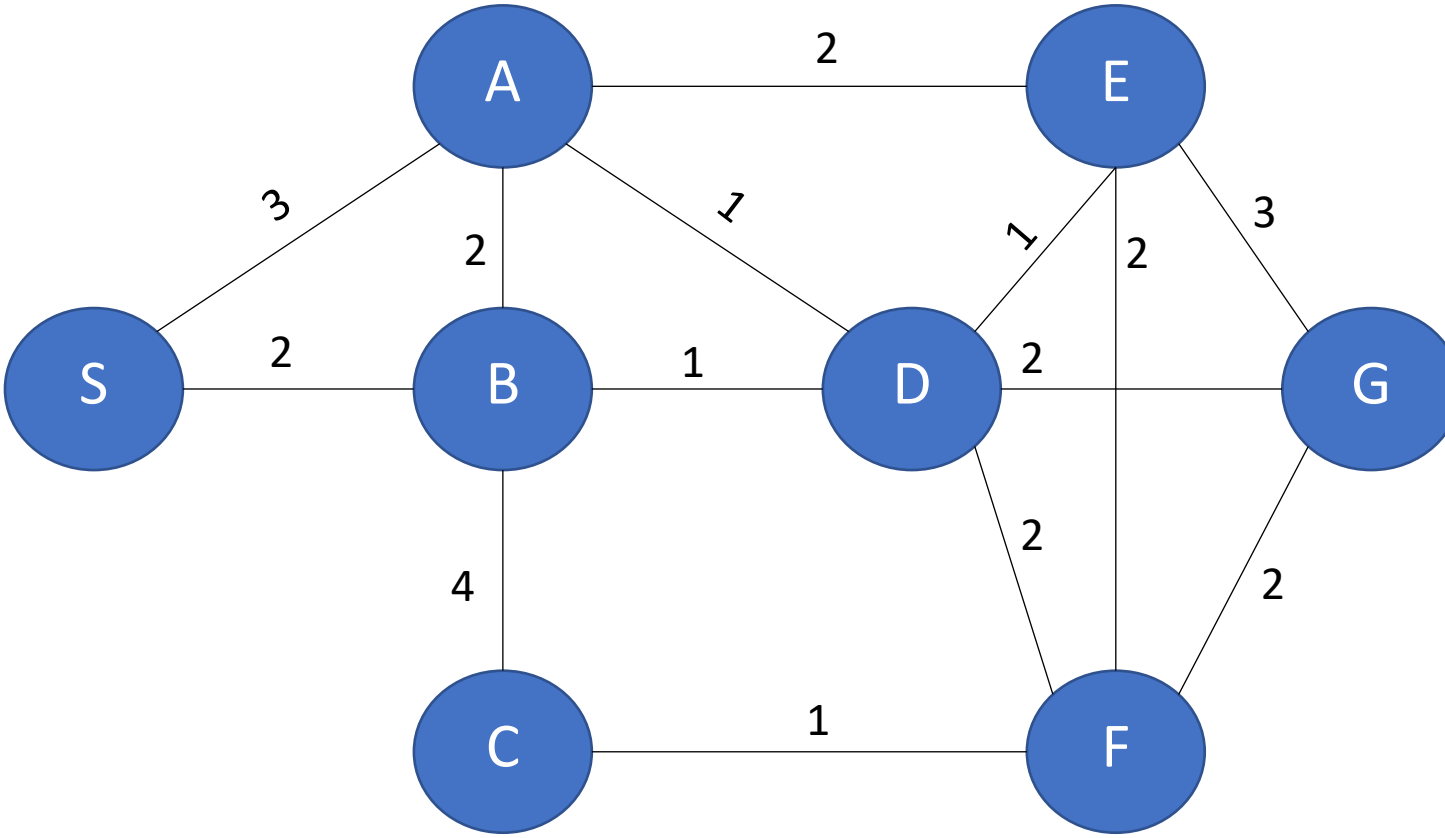
From vertex C, only vertex B can be reached directly and vertices A and G can be reached indirectly (with more than one edge) which are in unvisited nodes. A is closer than G since, 6<7. No need to update distances. At the end, my next stop will be A.
Distance=[TO S=0, TO A=3, TO B=2, TO D=3, TO E=4, TO G =5, TO F=5, TO C= 6]

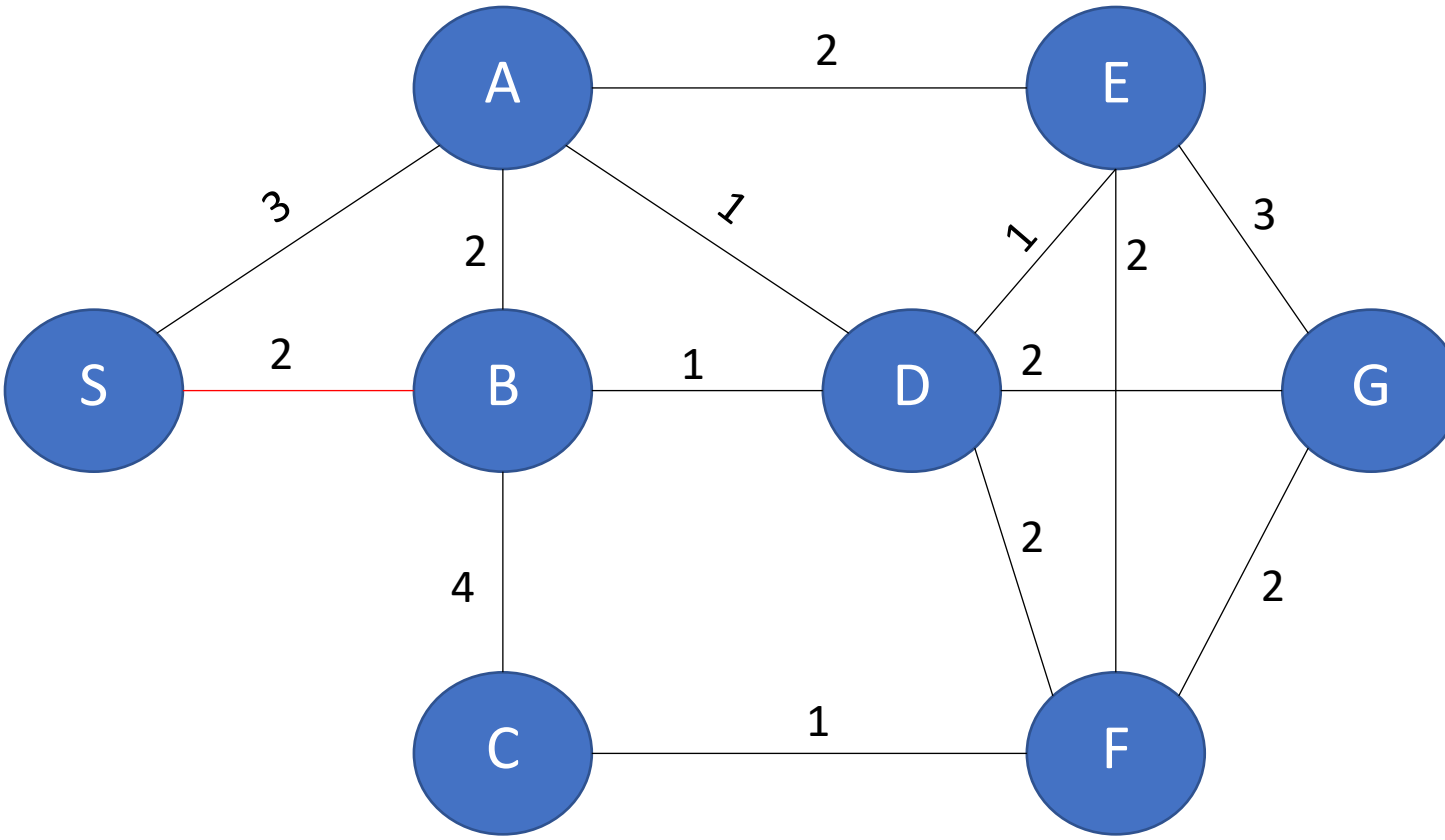
- My current stop is vertex A. Adding vertex A to the visited ones.
Visited nodes={S,B,D,E,F,A,C} ||||| Unvisited nodes={G}
From vertex A, vertices D and E can be reached directly. However, no need for updating distances. Now, I am moving to the vertex G which is in the unvisited nodes lonely.
Distance=[TO S=0, TO A=3, TO B=2, TO D=3, TO E=4, TO G =5, TO F=5, TO C= 6]

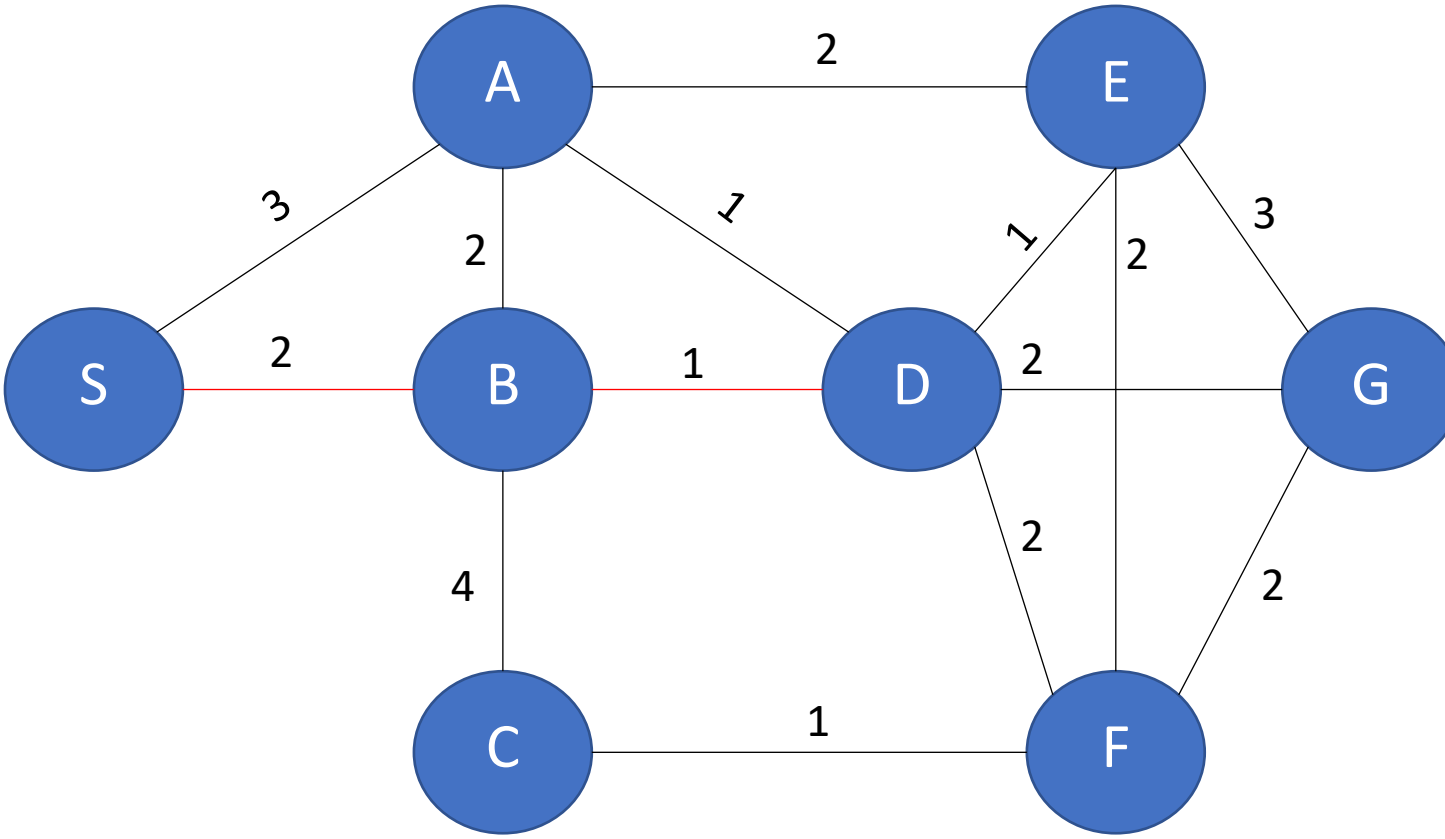
- My last stop is Vertex G. Adding it to the visited nodes and unvisited nodes are now empty.
Visited nodes={S,B,D,E,F,A,C,G} ||||| Unvisited nodes={}, no need for updating distances:
Distances=[TO S=0, TO A=3, TO B=2, TO D=3, TO E=4, TO G =5, TO F=5, TO C= 6]

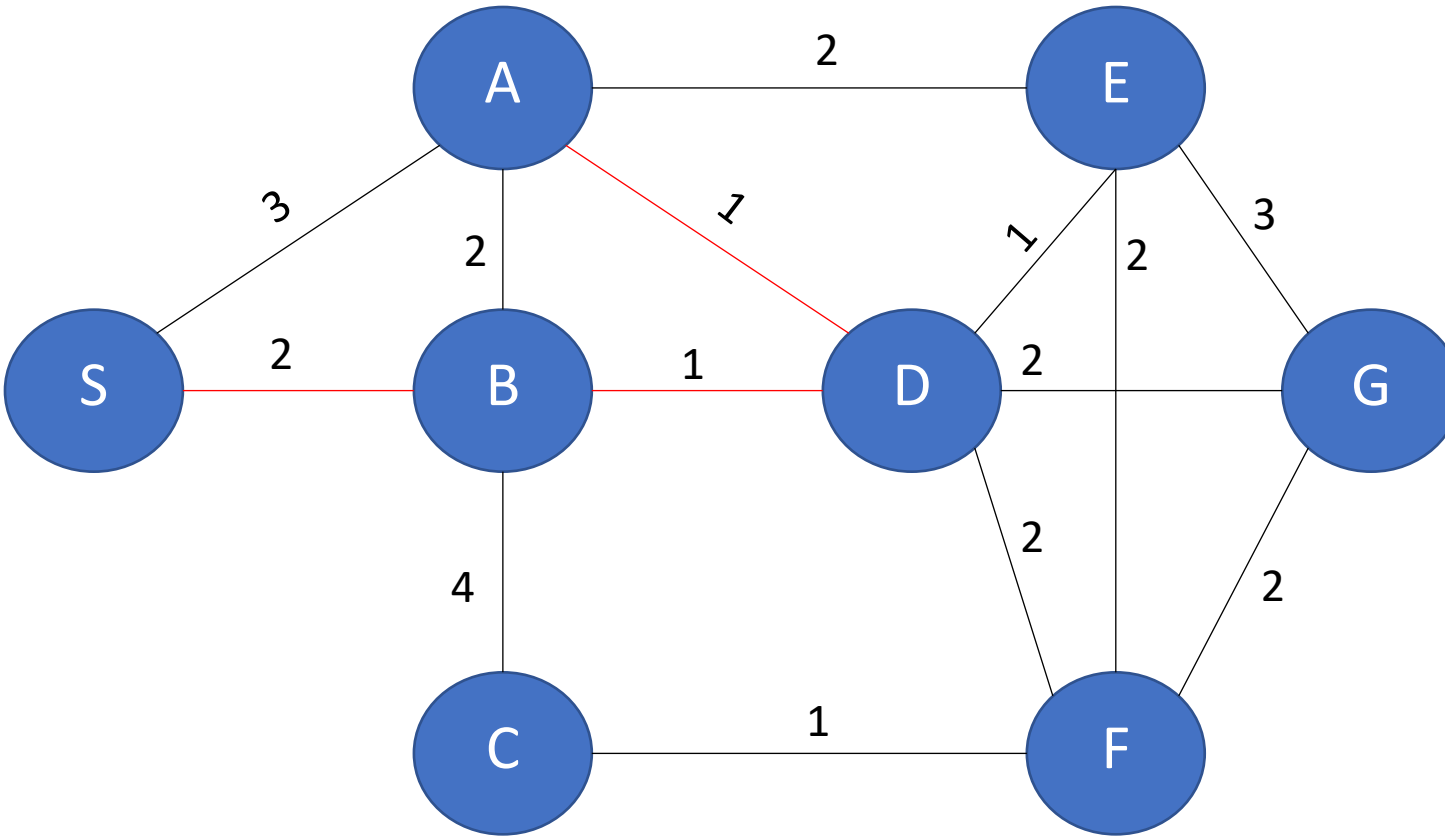
QUESTION 2

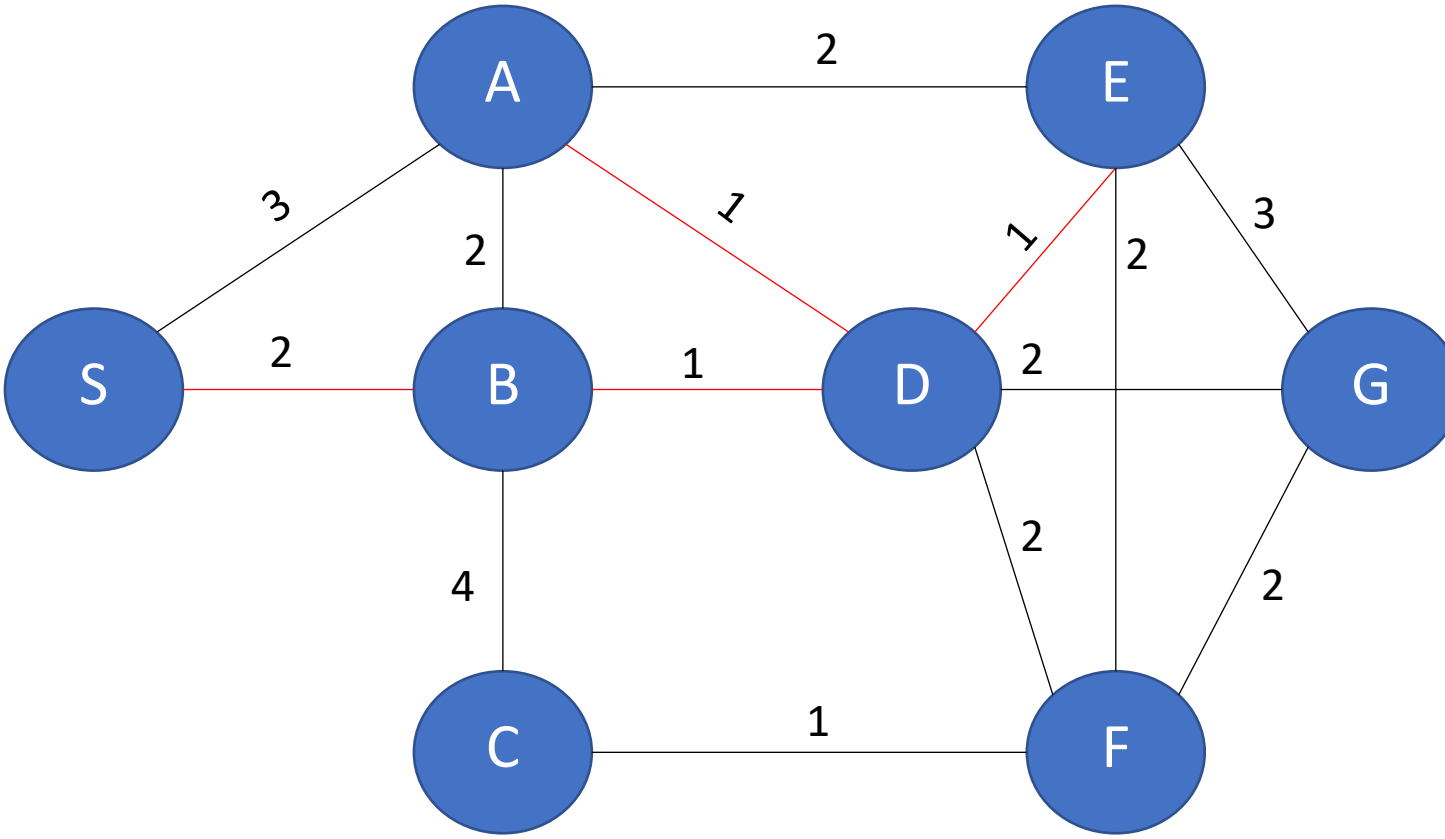
I am starting from S and tracing the operations of the Prim's minimum spanning tree algorithm on the graph. Selected edges are in red color.

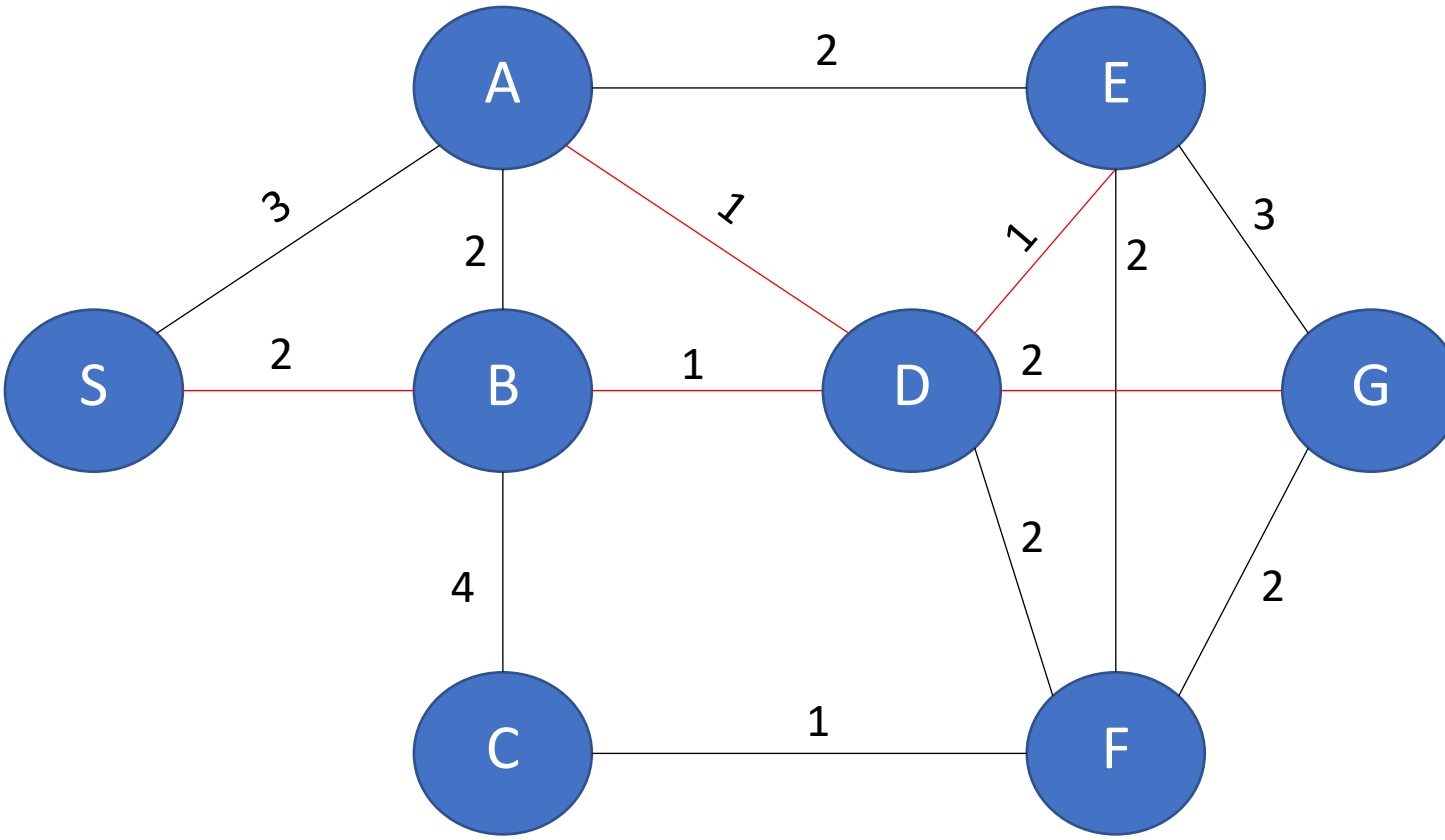


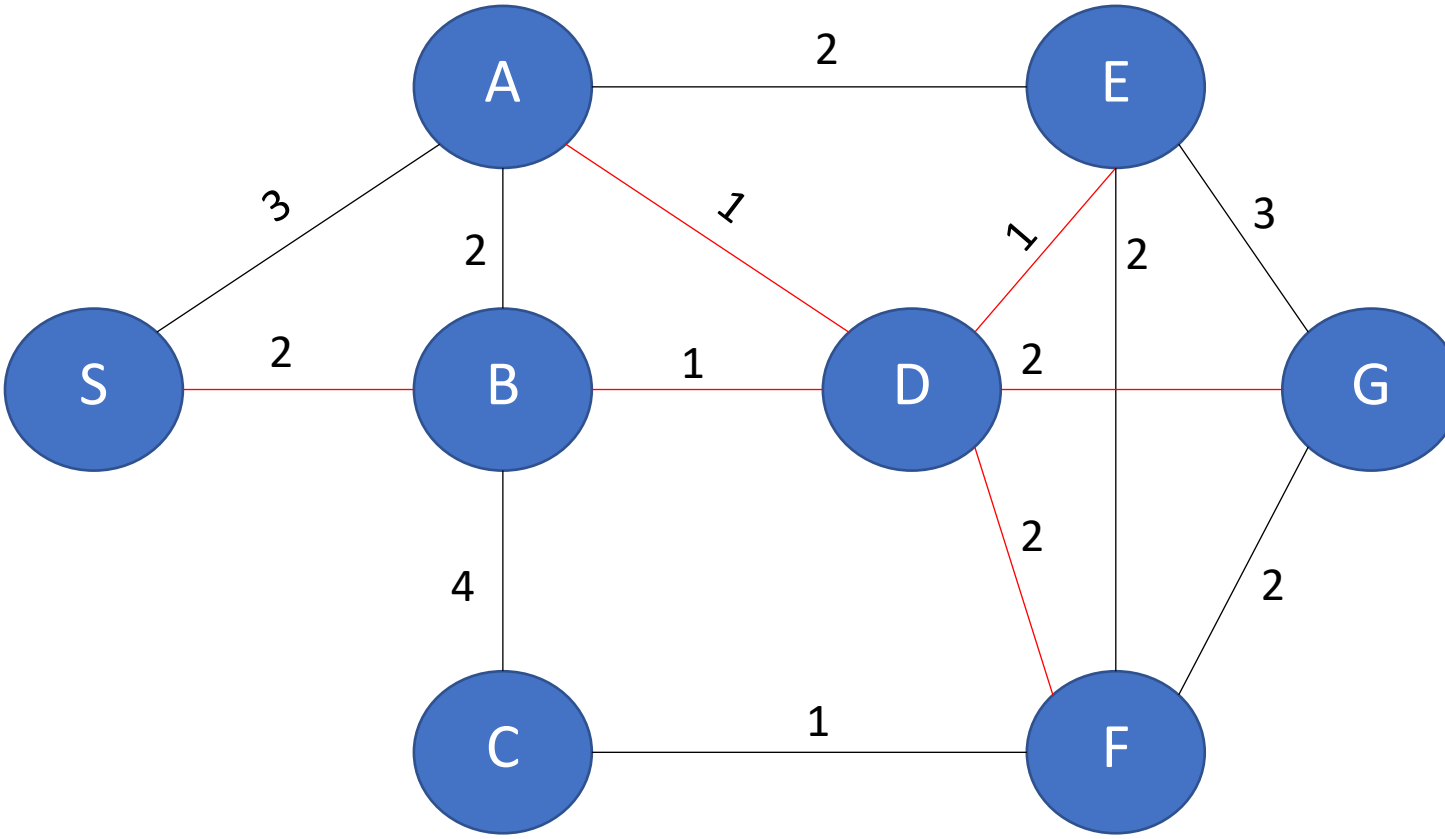

A
E
S
B
D
G
C
F
2
3
2
1
2
1
2
3
2
4
2
1

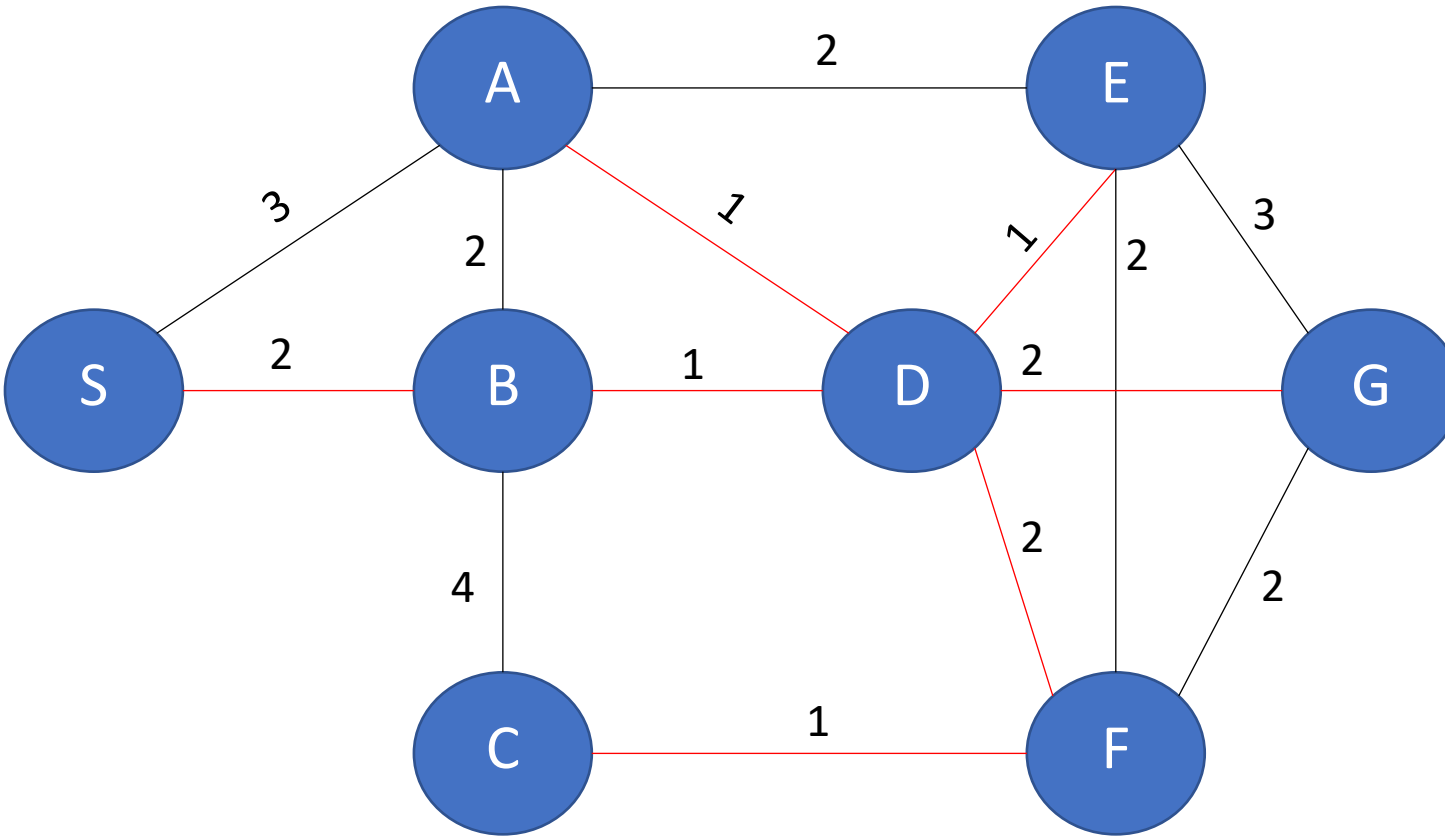

A
E
S
B
D
G
C
F
2
3
2
1
2
1
2
3
2
4
2
1


A
E
S
B
D
G
C
F
2
3
2
1
2
1
2
3
2
4
2
2
1


A
E
S
B
D
G
C
F
2
3
2
1
2
1
2
3
2
4
2
2
1


A
E
S
B
D
G
C
F
2
3
2
1
2
1
2
3
2
4
2
2
1


A
E
S
B
D
G
C
F
2
3
2
1
2
1
2
3
2
4
2
2
1



QUESTION 3

Edges with the corresponding distances:

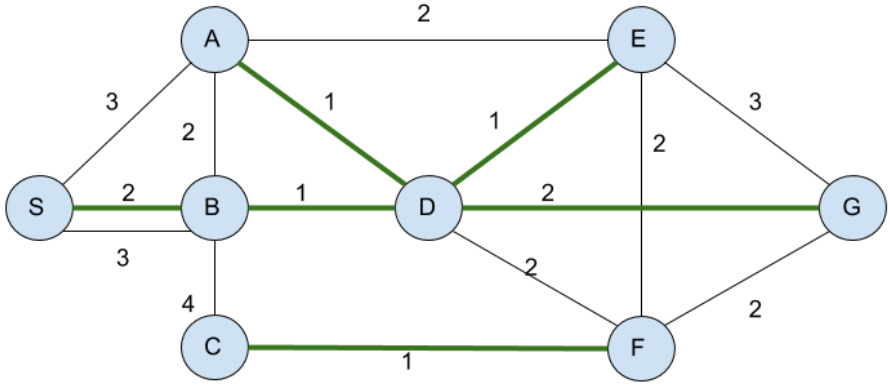
- A-D 1 INCLUDED
- B-D 1 INCLUDED
- D-E 1 INCLUDED
- F-C 1 INCLUDED
- S-B 2 INCLUDED
- B-A 2 NOT INCLUDED
- A-E 2 NOT INCLUDED
- E-F 2 NOT INCLUDED
- D-F 2 NOT INCLUDED
- D-G 2 INCLUDED
- G-F 2 NOT INCLUDED
- E-G 3 NOT INCLUDED
- S-A 3 NOT INCLUDED
- B-S 3 NOT INCLUDED
- E-G 3 NOT INCLUDED
- C-B 4 NOT INCLUDED

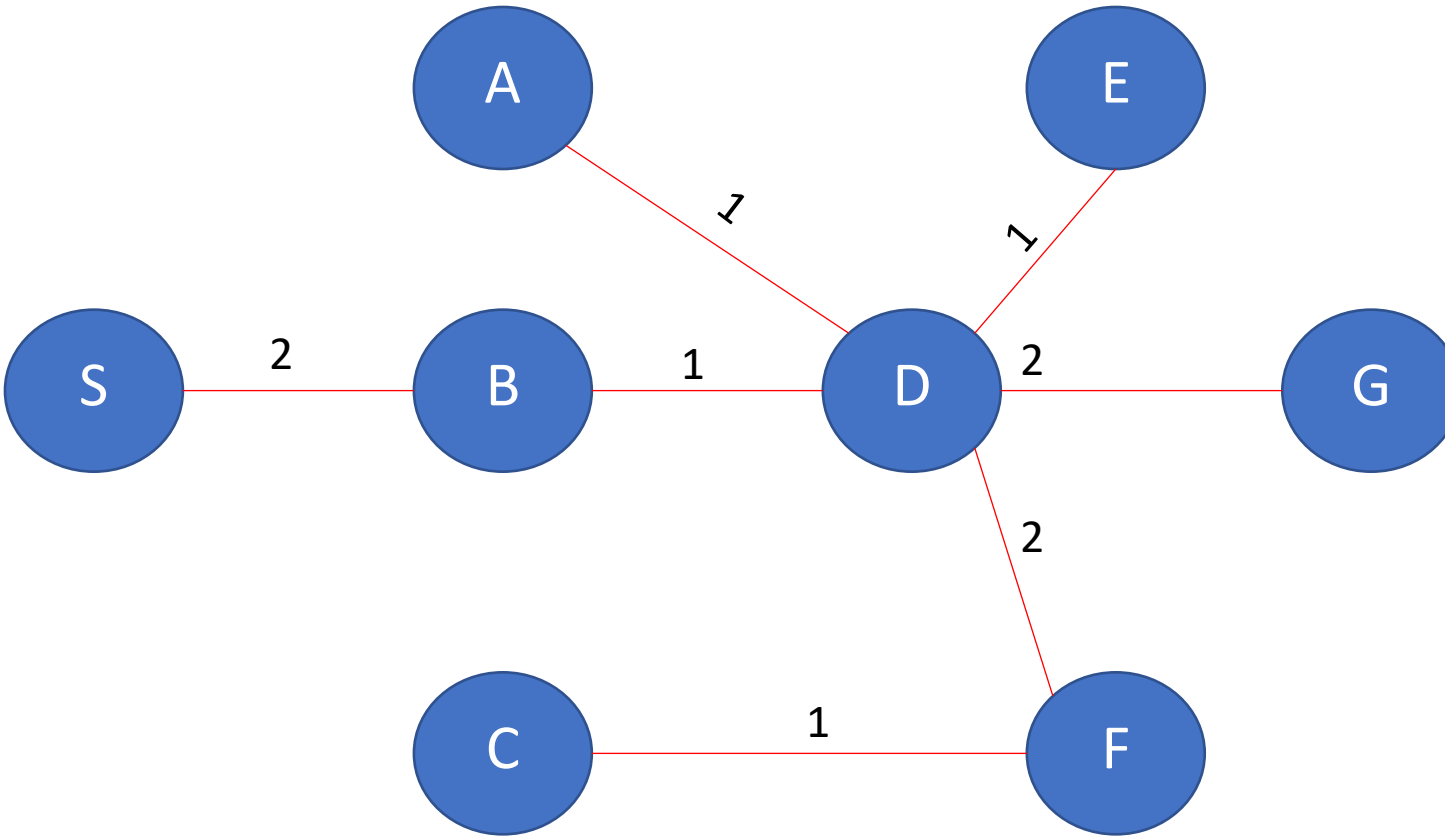
I have started from A and chose the edge which is between A and D. Then B-D has taken, and B is visited. After that D-E has taken and E is visited. Then, F-C has taken and both vertexes are visited. Then, S-B has taken, and S is visited. Edges B-A, A-E, E-F, D-F aren't taken since all of the vertexes in these edges are visited. Lastly D-G has taken, and G is visited.

Numbers: Steps | V: Visited

A	B	C	D	E	F	G	S
V 1	V 2	V 4	V 1	V 3	V 4	V 6	V 5

Minimum spanning trees: (BOTH ARE THE SAME BELOW)

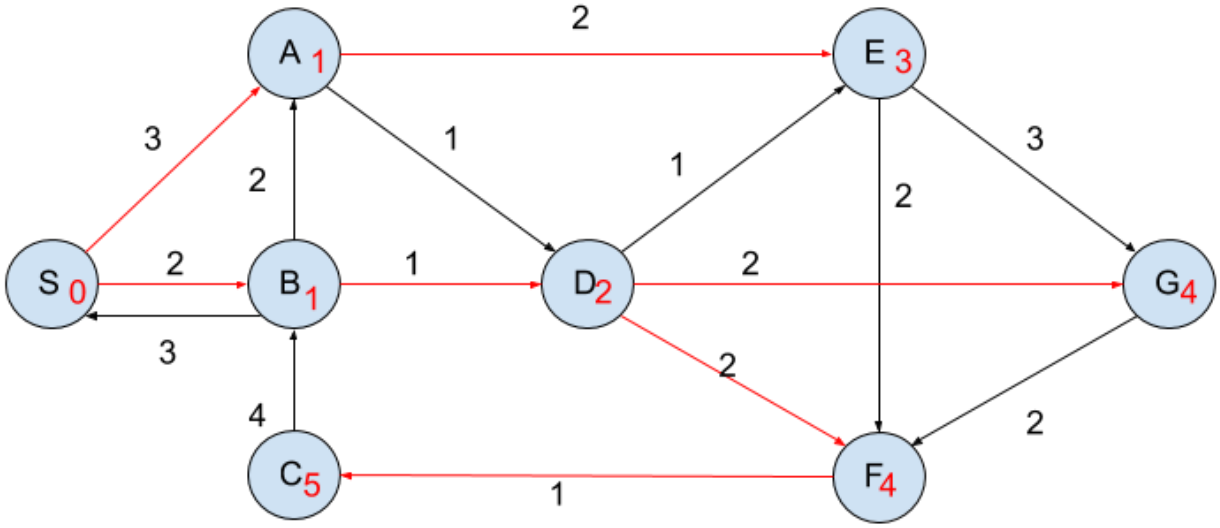




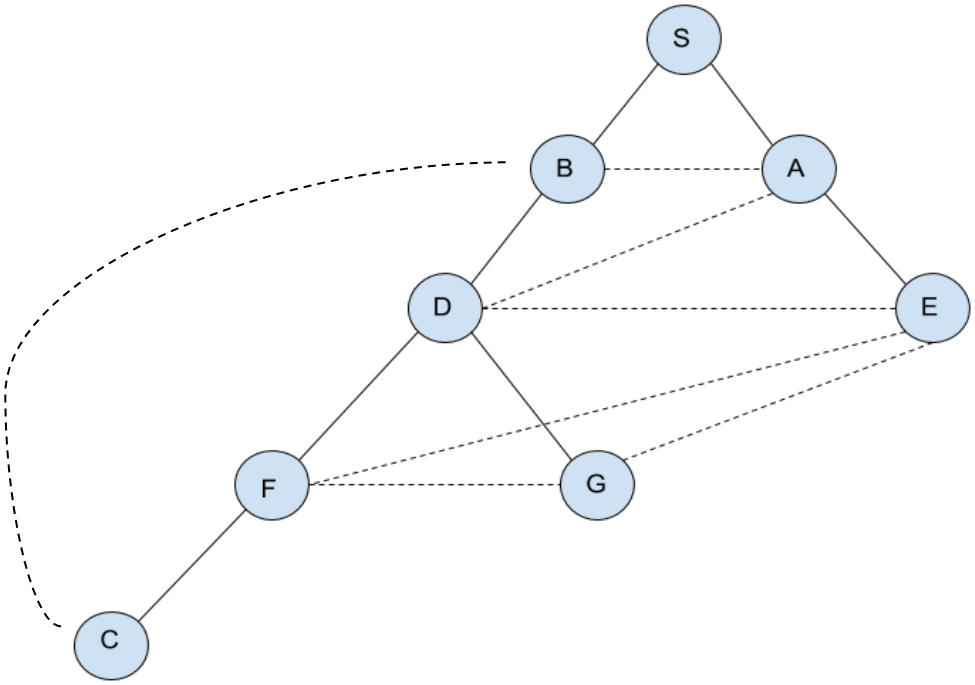
QUESTION 4

Starting from *S*, I am tracing the operations of breadth-first traversal on the graph given in Figure 1. For BFS, queue data structure is used. Queue data structure has a property which is called first in first out (FIFO) like in the real life, unlike the stack. The idea here is that starting with the first node, enqueue it. Then, go to its connections and add visited nodes to the queue as well. By doing these, we will be able to handle BFS. 2 ways to go from S: A and B. I added B first, then added A. From now on, the order is important. I can go to vertex D from vertex B. Therefore, I enqueued D. I can go to vertex E from vertex A. Therefore, I enqueued E. This is getting done until the last vertex is added to the queue. At the end, the queue will be like below and relationships between nodes are got from the order and by dequeuing. The queue is below.

S Front	B	A	D	E	F	G	C Rear



IT SHOULD LOOK LIKE THIS:

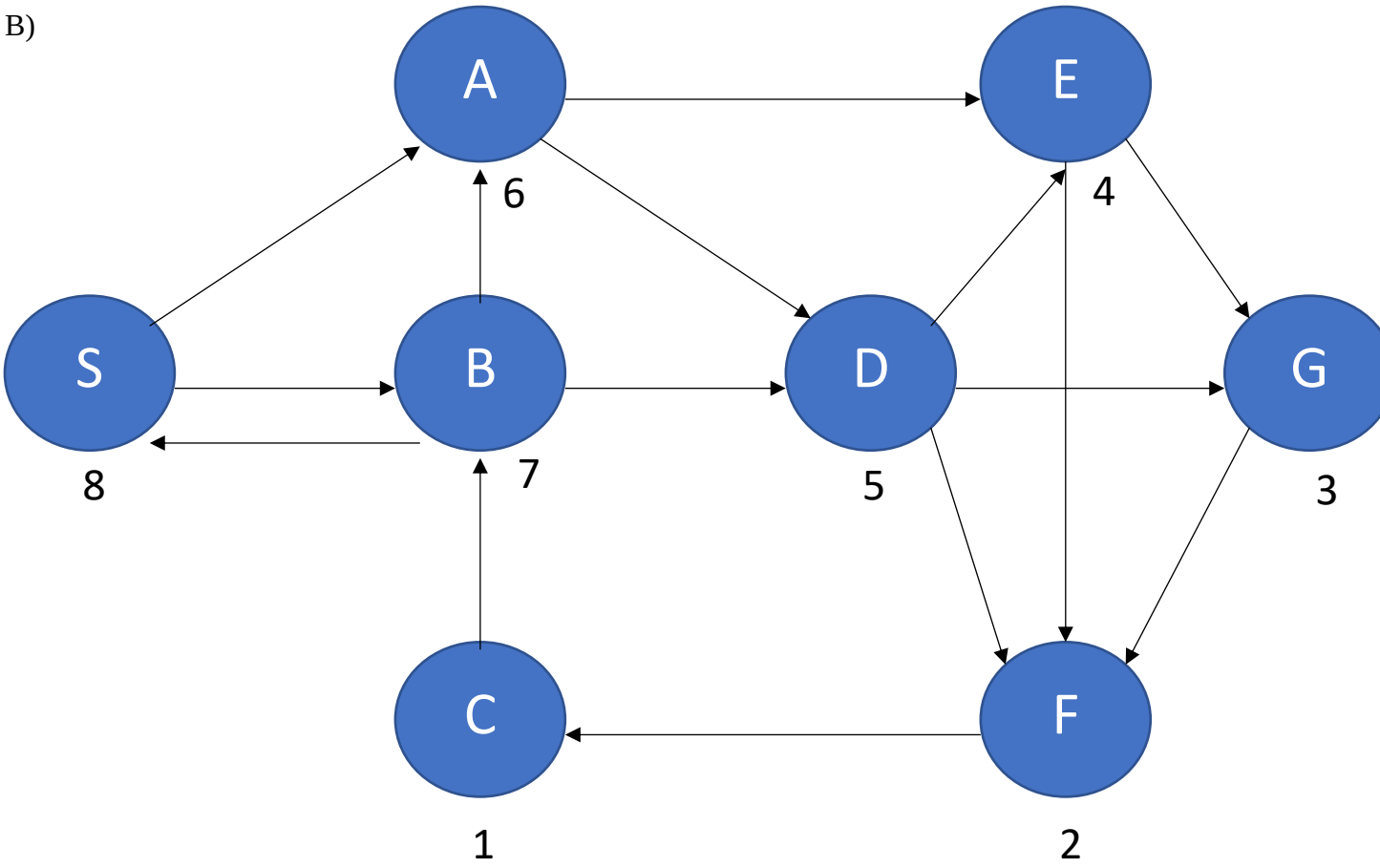


QUESTION 5

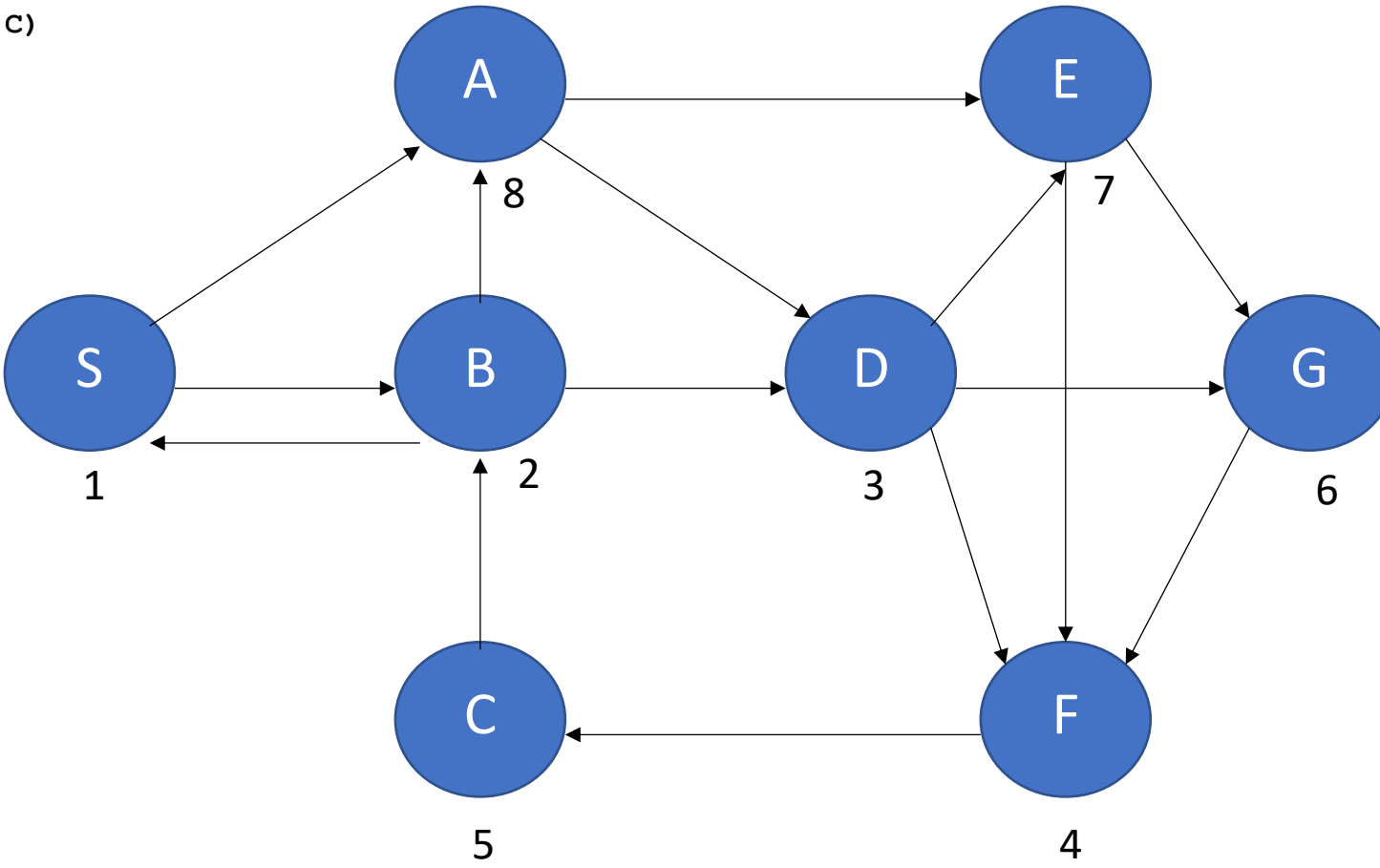
a) For DFS, stack data structure is used. I am starting from the vertex S. I am traversing S->A->E->G->F->C->B->D and I am enqueuing the vertices as long as I traverse. After traversing the deepest point in the graph, the stack will look like below and this is DFS. Depths of every element inside the stack are different. The depth of the one on the top is the greatest, in the bottom is the least. The depth is decreasing by 1 downwards for each element in the stack.

D TOP
B
C
F
G
E
A
S Bottom

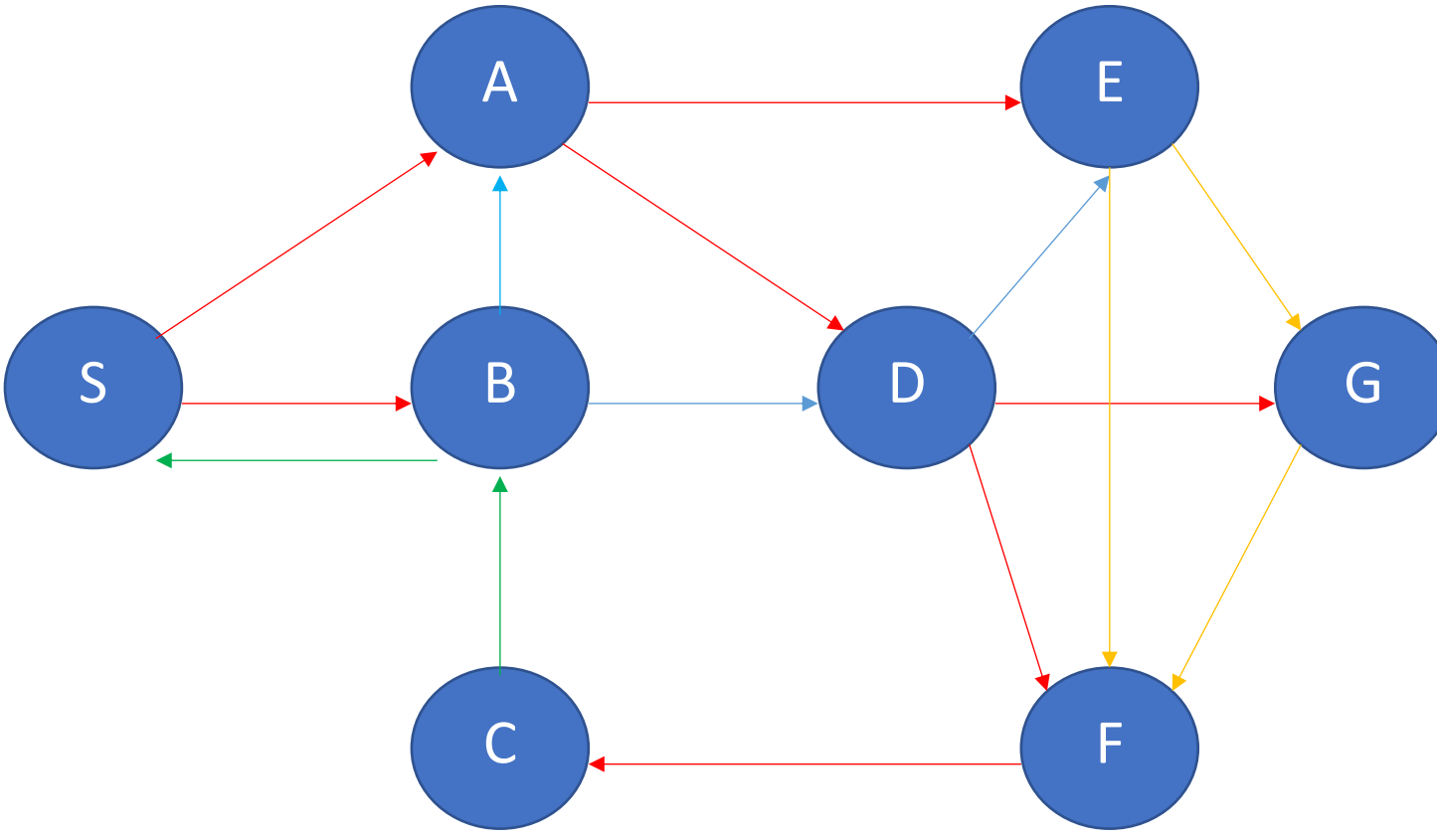
B)



c)



d) Red: Tree arcs, Green: Backward arcs, Blue: Forward arcs, Yellow: Cross arcs



QUESTION 6

-Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge u v, vertex u comes before v in the ordering. Topological Sorting for a graph is not possible if the graph is not a DAG.

To begin with, I have selected the vertex with inner degree 0. That is the vertex D and I have added it to the stack.

Then, I have added the vertex B to the stack which has inner degree 0.

Then, I have added the vertex E to the stack which has inner degree 0.

Then, I have added the vertex F to the stack which has inner degree 0.

Then, I have added the vertex G to the stack which has inner degree 0.

Then, I have added the vertex A to the stack which has inner degree 0.

Lastly, I have added the vertex C to the stack.

In conclusion,

ANSWER: Stack: (BOTTOM) CAGFEBD (TOP) → Order: DBEFGAC