CS300 Homework 5

İlker Gül – 26352

Question 1

Initially, all vertices are unknown. Starting node S is at zero distance from itself and other nodes are at distance ∞ from S.

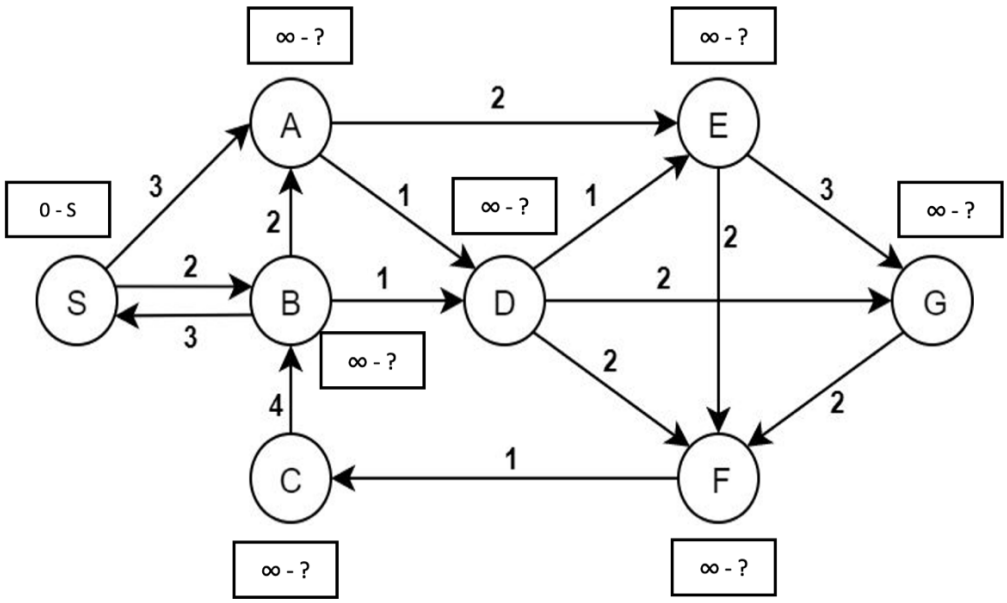


Figure 1: Initial status of the graph

X - Y ➔ Known vertex, X is distance, Y is previous node

X - Y ➔ Chosen vertex for the next step, X is distance, Y is previous node

X - Y ➔ Adjacent unknown vertex at current step, X is distance, Y is previous node

Then, I mark S as known and we are updating the unknown adjacent vertices' distance if the known vertex we are currently using gives a shorter distance to the adjacent unknown vertices.

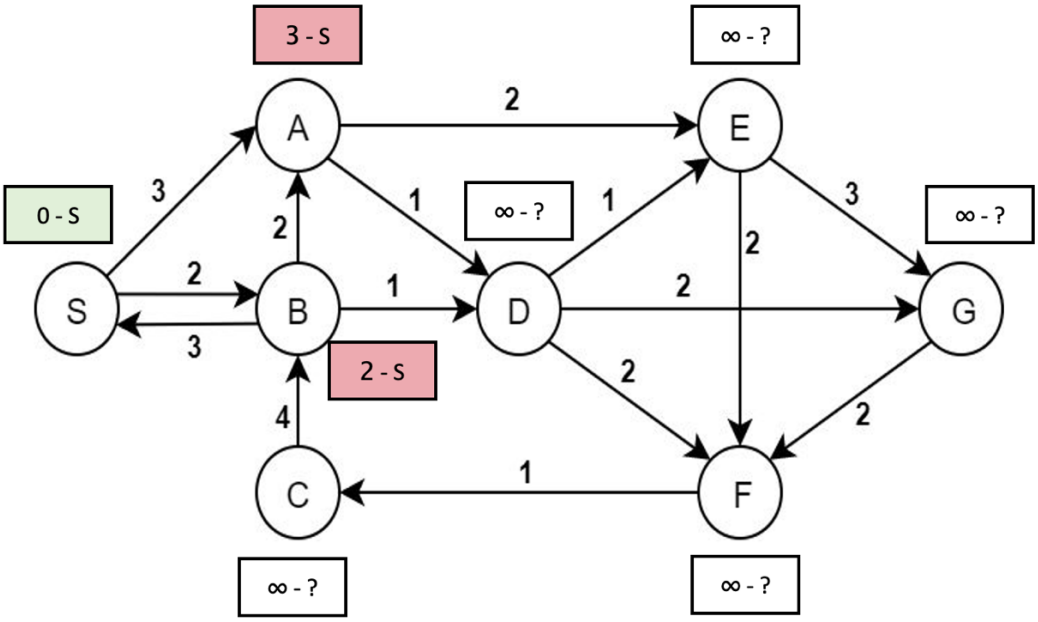


Figure 2: S is the current known vertex

Then, I choose an unknown vertex which has the shortest distance which is B in this case for the next step. I marked B as known vertex. Then, I change the unknown adjacent vertices' distance if A gives a shorter distance. Distance of vertex D has changed. However, since B does not give a shorter distance to A than A's current distance, we are not updating its distance.

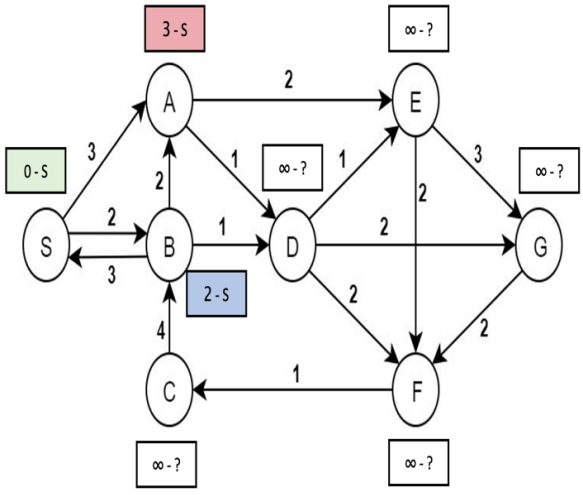


Figure 3: B is selected for the next step

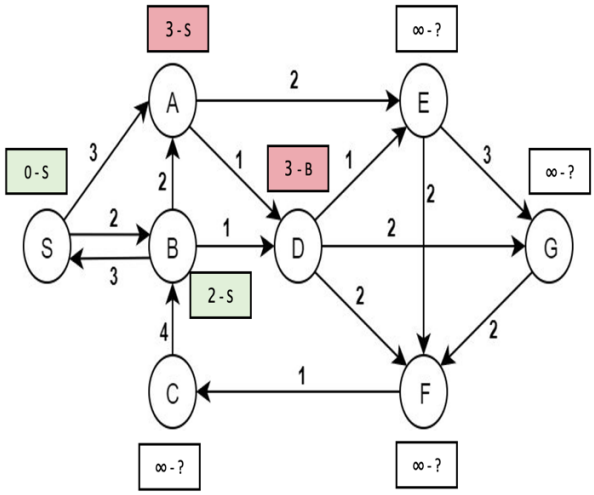


Figure 4: B is marked and checking unknown vertices

Then, we have two unknown vertices that has the same distance. I choose arbitrarily and select A for the next step. Then, A is marked as known. We are again checking the distance of the unknown adjacent vertices from A and do necessary changes if there is any.

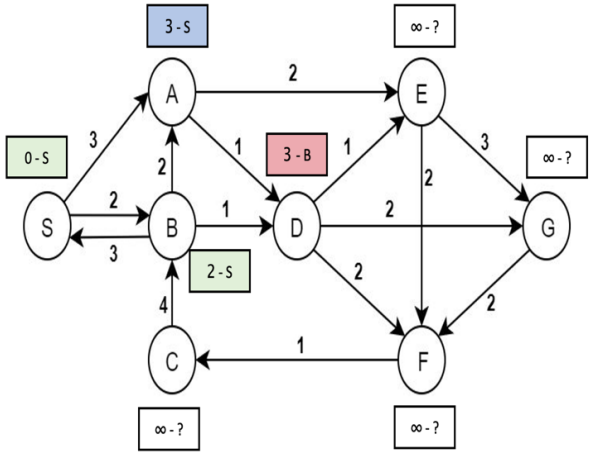


Figure 5: A is selected for the next step

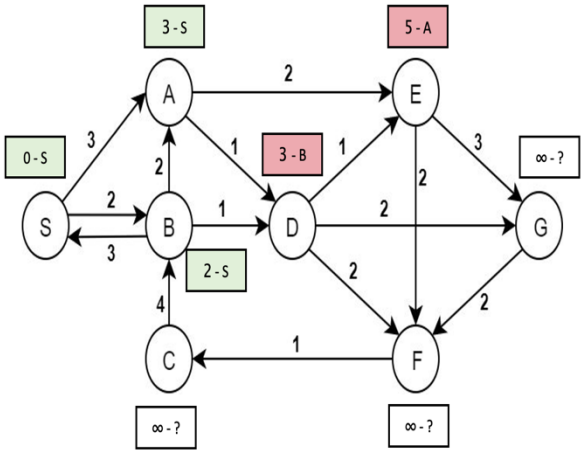


Figure 6: A is marked and checking unknown vertices

Then, we are choosing D for the next step since it hast the shortest distance among unknown vertices. Then, I marked D as known and change the distances of the adjacent unknown vertices if necessary.

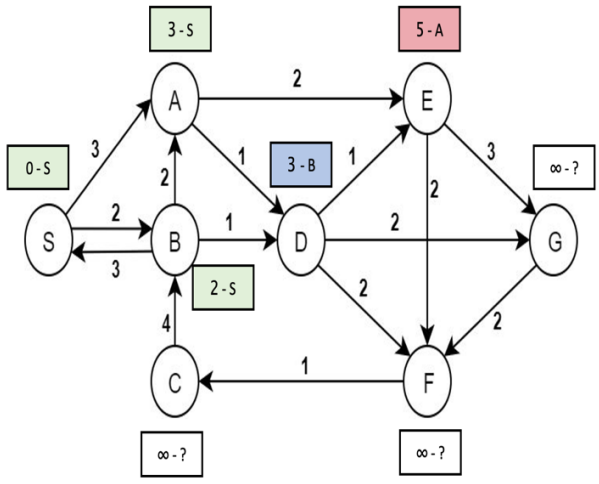


Figure 7: D is selected for the next step

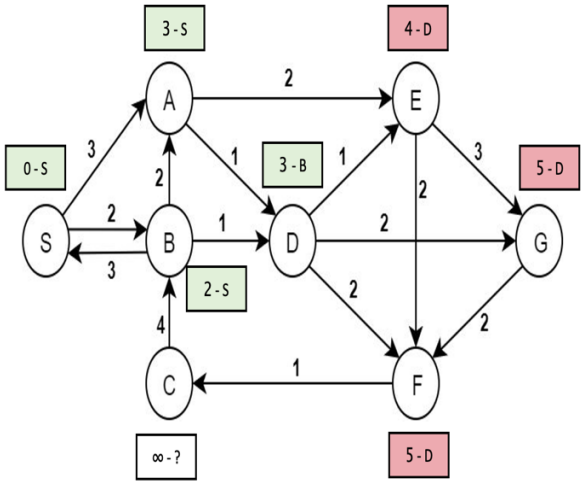


Figure 8: D is marked, unknown vertices are checked

D is chosen for the next step. Then, D is marked as known and adjacent unknown vertices are checked if there is an update for the distance for each is needed or not.

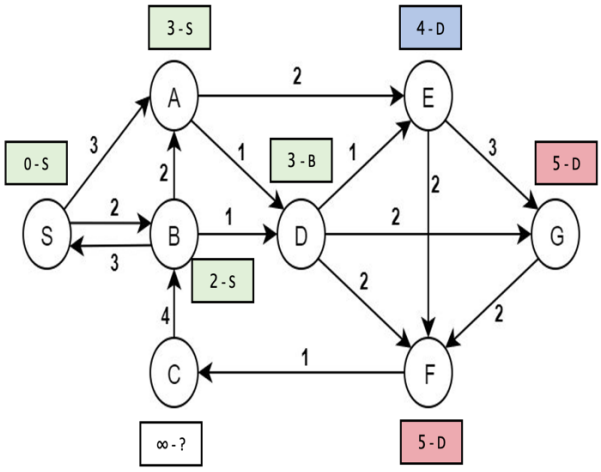


Figure 9: E is chosen for the next step

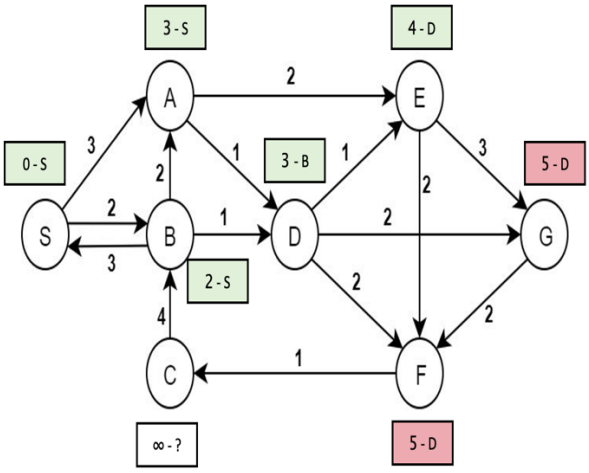


Figure 10: E is marked, unknown vertices are checked

Then, we have two unknown vertices having same distance. Thus, we choose arbitrarily and select G for the next step. Then, G is marked as known and check the adjacent unknown vertex F to update its distance or not.

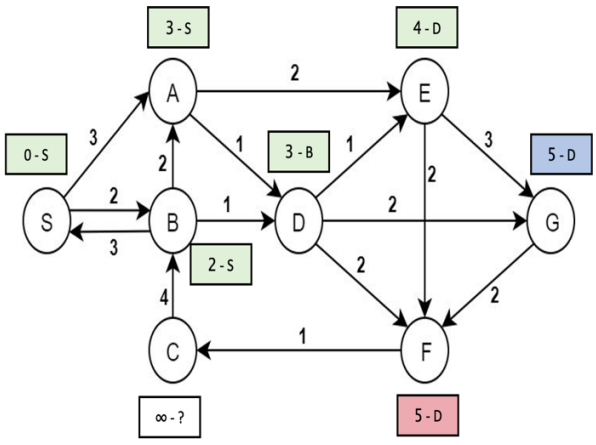


Figure 11: G is chosen for the next step

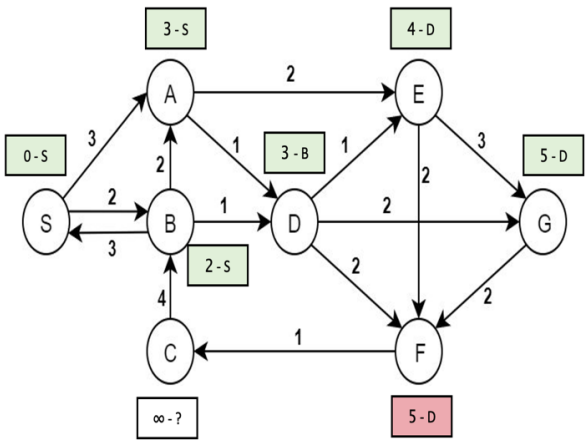


Figure 12: G is marked, unknown vertices are checked

Then, I choose F for the next step. I marked F as known and then check the unknown vertex C, since it is the only adjacent unknown vertex to F, whether to update is current distance or not. Then, I modified C's current distance since F is giving a shorter distance than its current one.

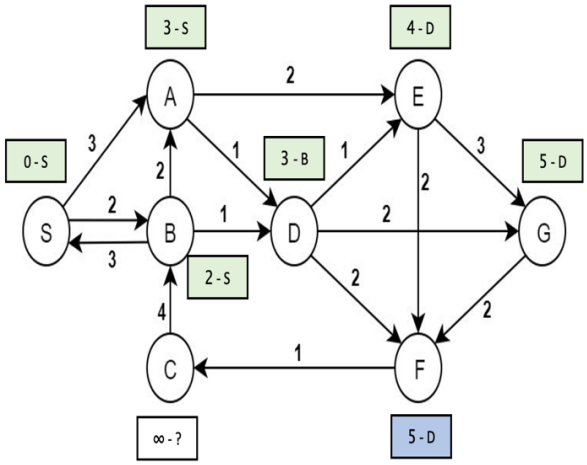


Figure 13: F is chosen for the next step

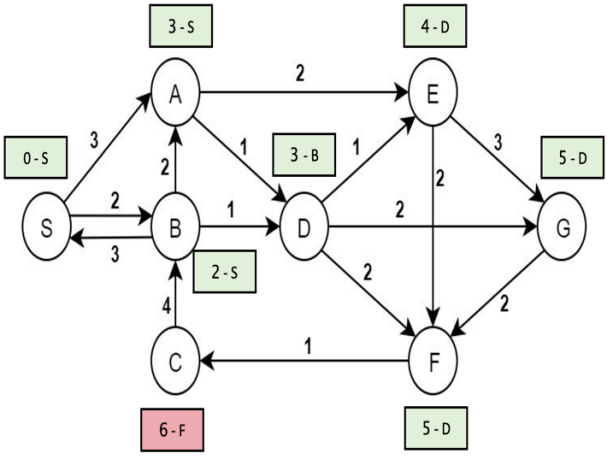


Figure 14: F is marked, unknown vertex C is checked

Then, C is chosen for the next step. C is marked as known. Then, it checks for adjacent unknown vertices, there is no adjacent unknown vertex. So, we checked for the next step.

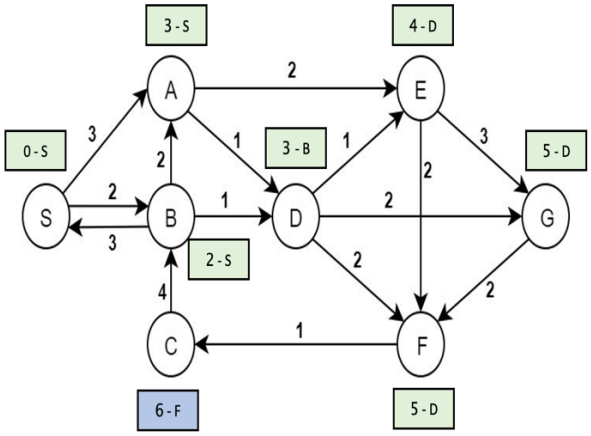


Figure 15: C is chosen for the next step

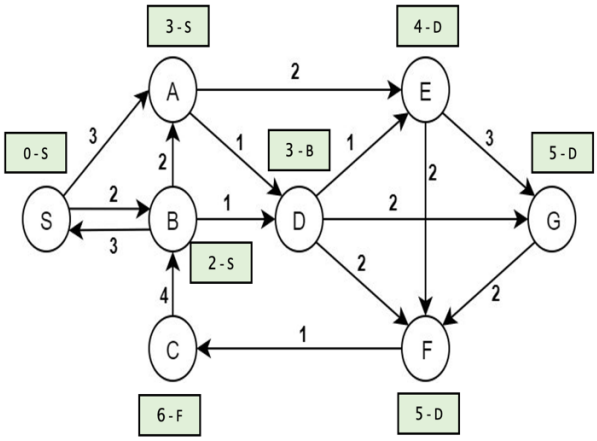


Figure 16: C is marked, check for unknown vertices

We checked for whether there is an unknown vertex left. All the nodes are marked as known we have no unknown vertex left. Hence, we are done. Now, all vertices have shortest distance to source vertex S by Dijkstra's shortest algorithm.

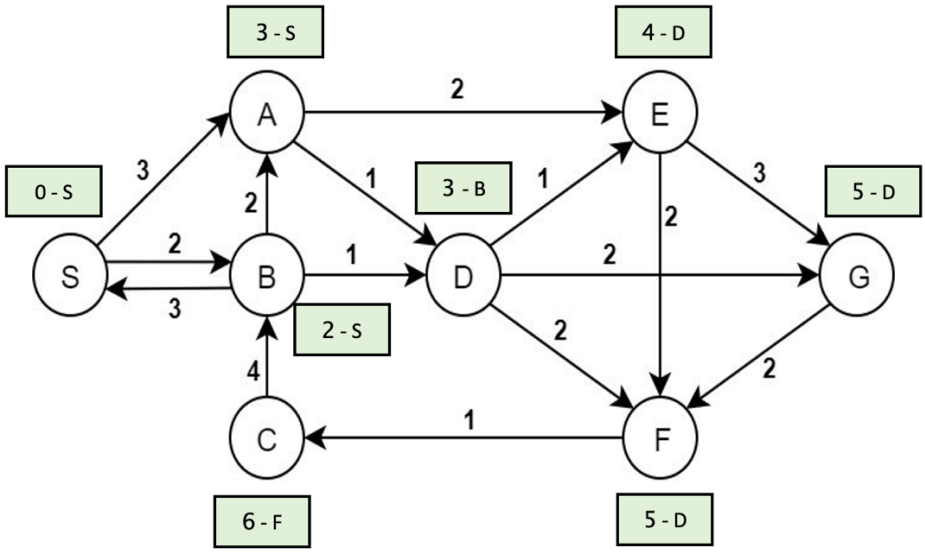


Figure 17: Final version by Dijkstra's shortest path algorithm

Question 2

We will start with S. We will grow the tree in a successive manner such that at each step we will select the shortest weighted edge which is from a vertex included in the tree to a vertex that is currently not in the tree. Then, we will mark the unknown vertex as visited to know that the vertex has been added to the tree and continue.

First, we will start from S, mark it as visited then we will check the adjacent unknown vertices

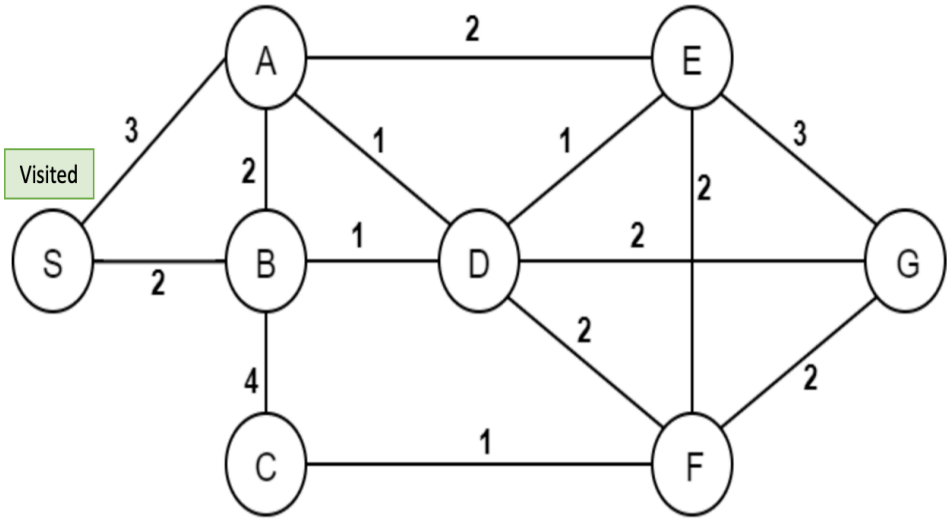


Figure 18: Initial status for the prim's algorithm with having S as visited

Then, we check between A and B. We see that B has shorter distance. Thus, we added the edge from S to B to our tree and then marked B as visited to know that the vertex B has been added to the tree.

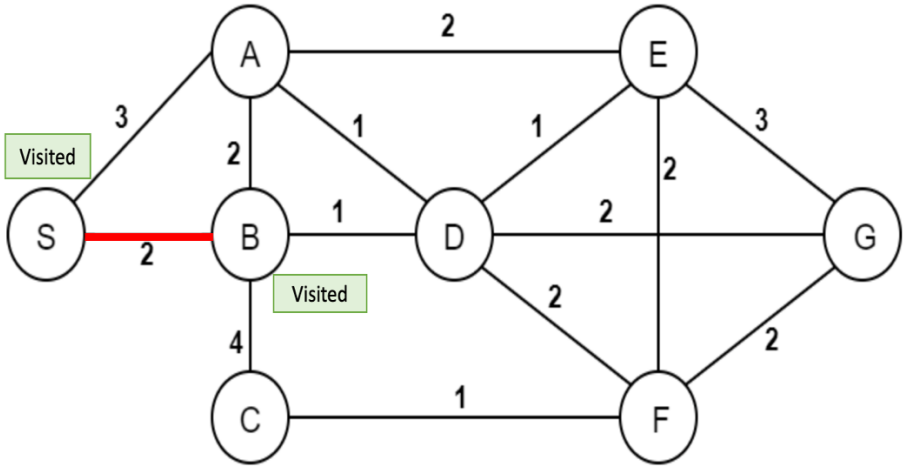


Figure 19: B is now added to the tree

Then, we check between A, B and D. We see that D has the shortest distance. Thus, we added the edge from B to D to our tree and then marked D as visited to know that the vertex D has been added to the tree.

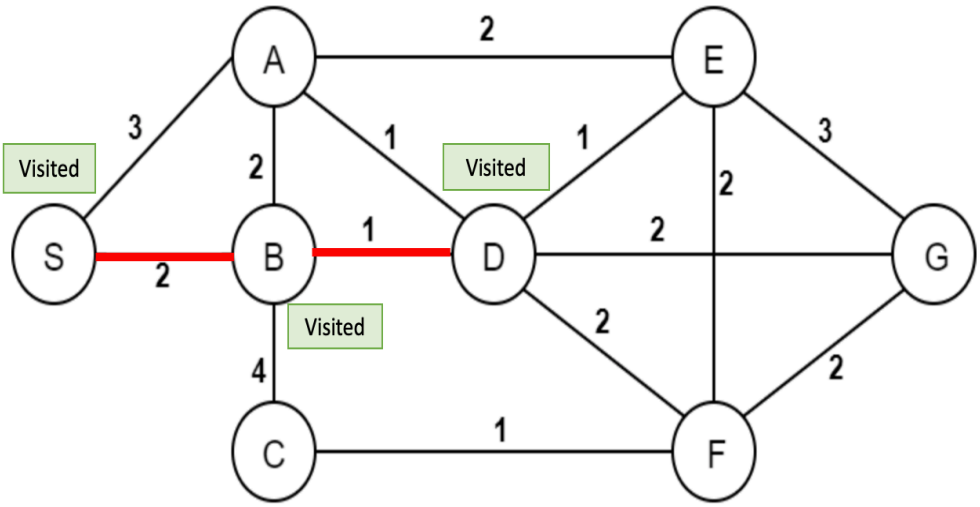


Figure 20: D is now added to the tree

Then, we check between A, C, E, G and F. We see that A and E has the shortest distance edge from D. We choose arbitrarily. Thus, we added the edge from D to A to our tree and then marked A as visited to know that the vertex A has been visited.

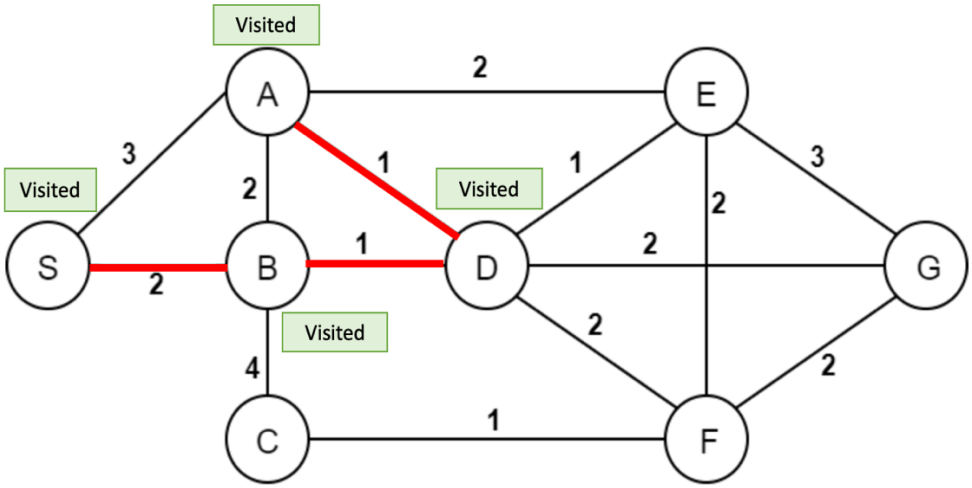


Figure 21: A is now added to the tree

Then, we check between C, E, G and F. We see that E has the shortest distance edge from D. Thus, we added the edge from D to E to our tree and then marked E as visited to know that the vertex E has been visited.

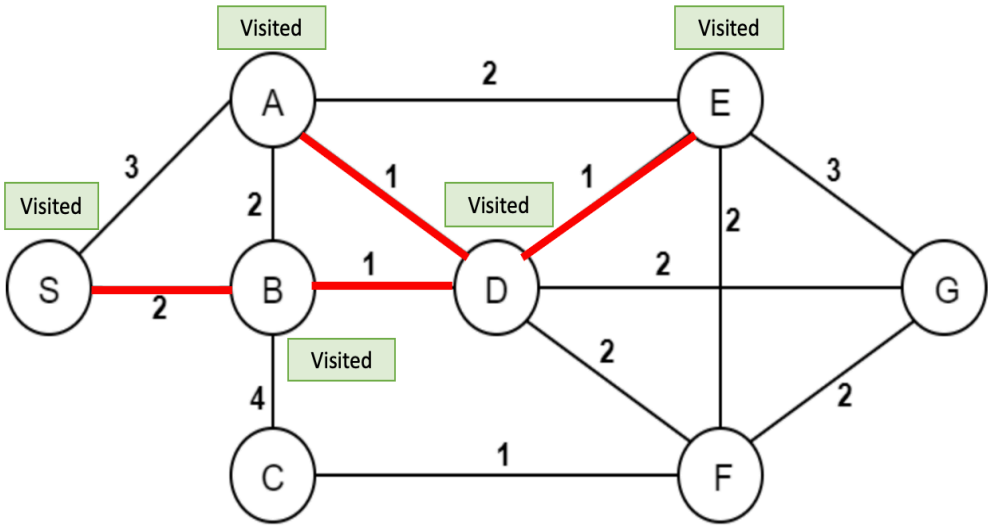


Figure 22: E is now added to the tree

Then, we check between C, G and F. We see that G has a shortest edge from D and F has shortest edges from D and E. Then, we choose arbitrarily. Thus, we added the edge from D to G to our tree and then marked G as visited to know that the vertex G has been visited.

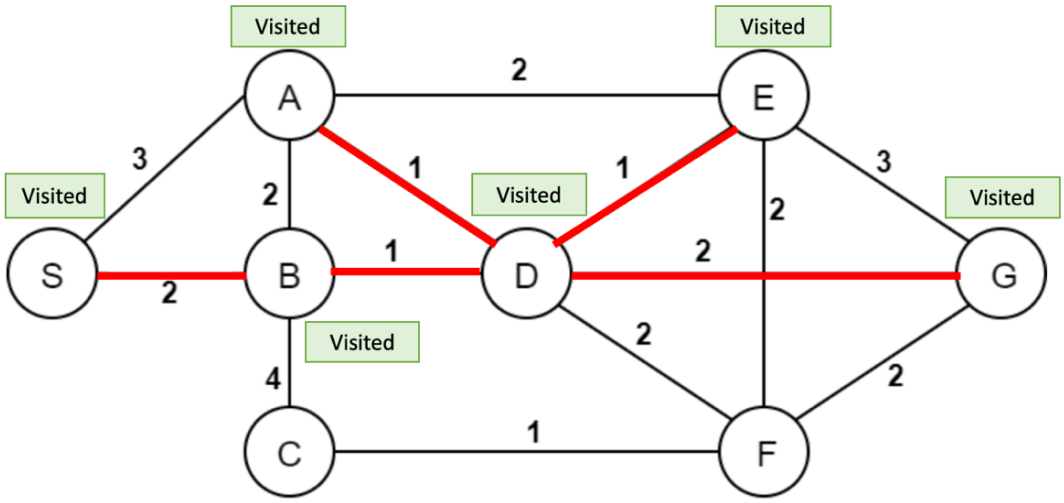


Figure 23: Vertex G is now added with an edge from D

Then, we check between C and F. We see that F has the shortest distance edges from D and G. Then, we choose arbitrarily. Thus, we added the edge from D to F to our tree and then marked F as visited to know that the vertex F has been added to the tree.

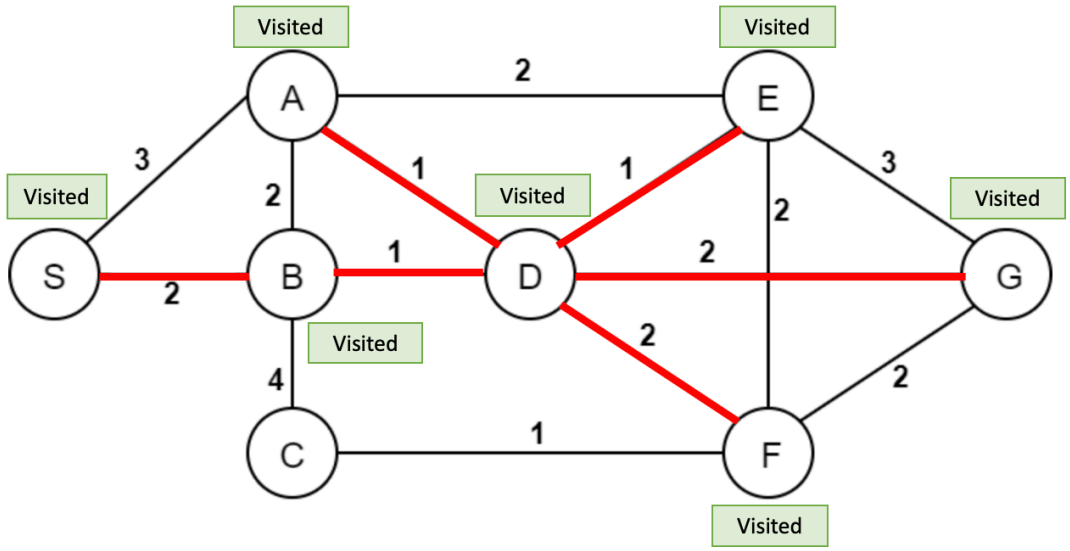


Figure 24: Vertex F is now added to the tree with an edge from D

Then, we check edges that goes to C. We see that C has the shortest distance edge from F. Thus, we added the edge from F to C to our tree and then marked C as visited.

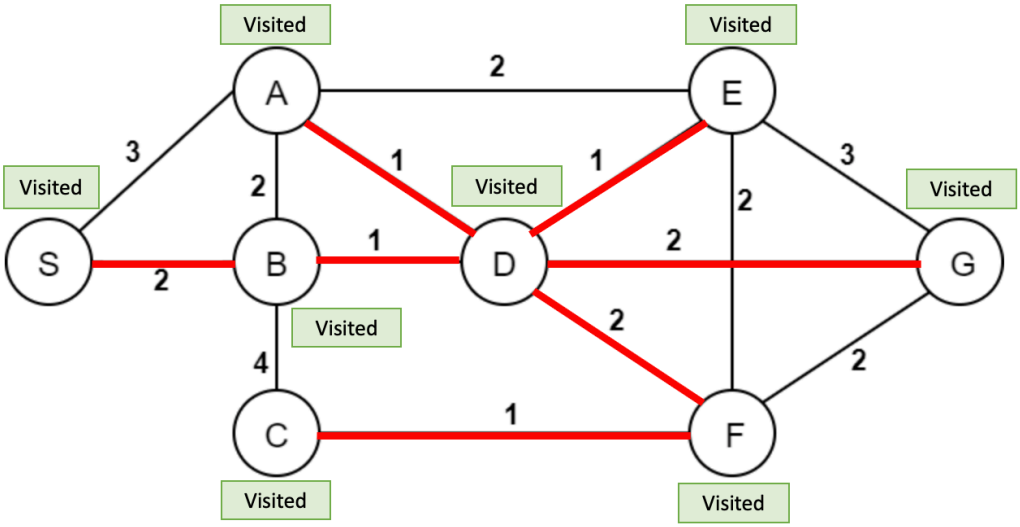


Figure 25: Vertex C is added

Then, there is no unvisited node left in our graph so Prim's algorithm for minimum spanning tree is done. In conclusion, we can check our spanning tree starting from S. Total cost is 10.

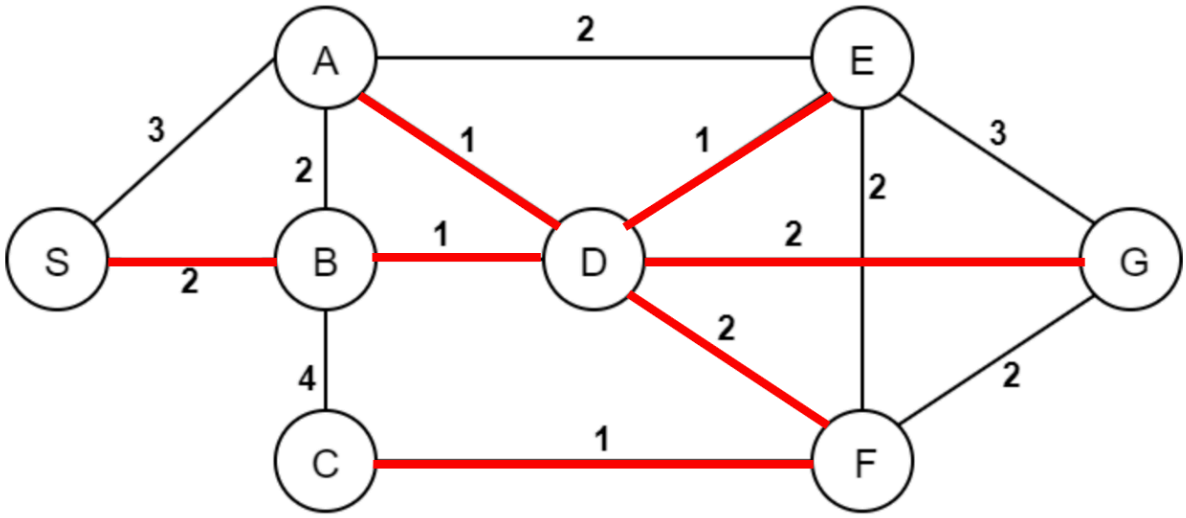


Figure 26: Final version of minimum spanning tree by Prim's algorithm with total cost 10.

Question 3

Initially, we have 8 singe vertex trees and accepted edges count as zero. Then, we will select the edges with minimum weight. If it does not cause a cycle when we try to merge the trees with that edge, then we will add this edge. If it causes a cycle, we will not add this edge.
We have 4 edges with having a weighted distance of 1. These are AD, BD, DE, CF edges. If they do not cause a cycle when we select them, we will use it.
First, we checked AD. It did not cause a cycle, so we added this edge. Then, we checked BD it did not cause a cycle, so we added this edge, too. Then, we checked DE, it did not a cause a cycle, hence we added. Then, we checked CF, it did not cause a cycle, thus, we also added this edge. After each successful step for adding an edge, we incremented the count of accepted edges Following figures will show the order of adding these aforementioned edges.

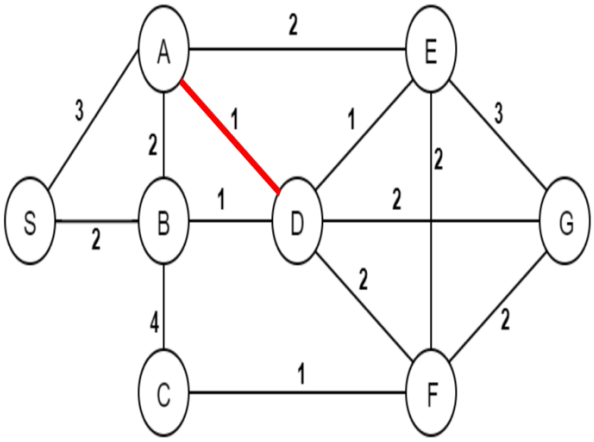


Figure 27: AD edge is added

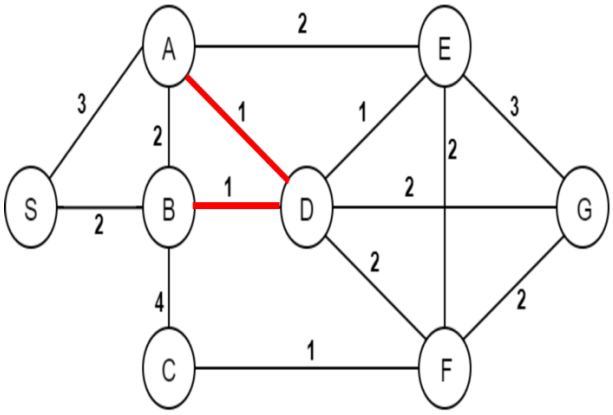


Figure 28: BD edge is added

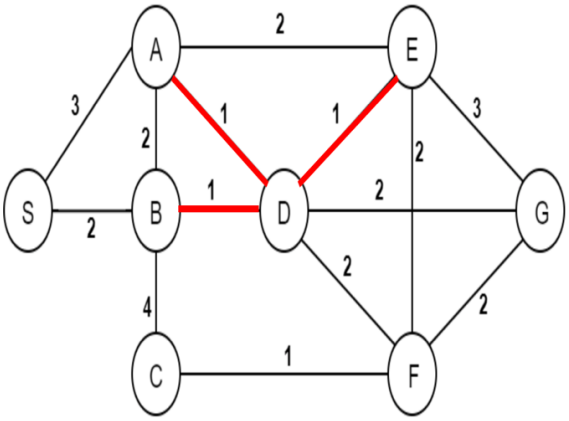


Figure 29: DE edge is added

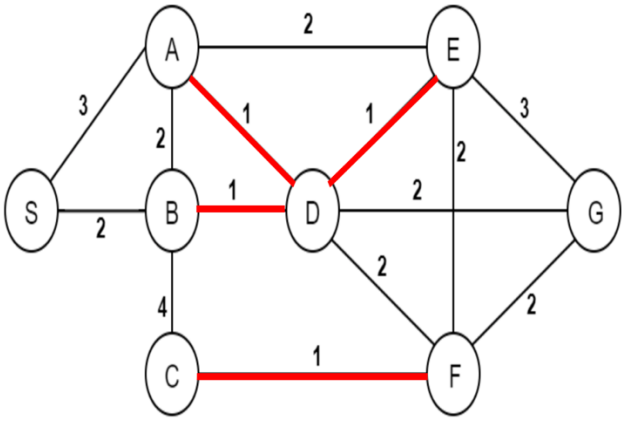


Figure 30: CF edge is added

Then, we have 7 edges with having a weighted distance of 2. These edges are SB, AB, AE, DG, GF, DF, EF edges. If they do not cause a cycle when we select them, we will use it. We check SB. It does not cause a cycle, so we added SB edge. Then, we checked AB if we add AB it causes a cycle. Hence, we do not add it. Then, we checked AE, it causes a cycle if add it. Thus, we do not add it. Then, we checked DG. It does not cause a cycle, so we added this edge. Then, we checked GF. It does not cause a cycle, so we added this edge. After this step we have reached V-1 = 8-1= 7 edges in our current tree. Hence, we now terminated the algorithm and reached our final tree by Kruskal's algorithm. Total cost is 10.

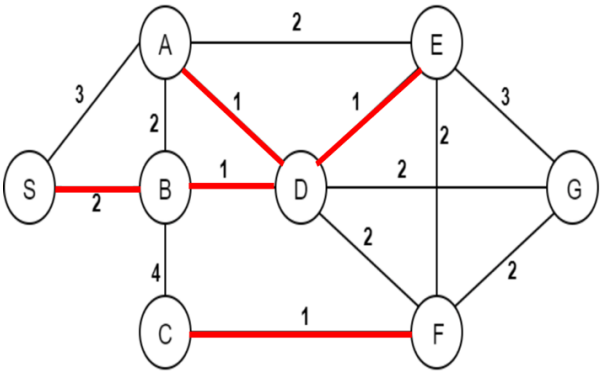


Figure 31: SB edge is added

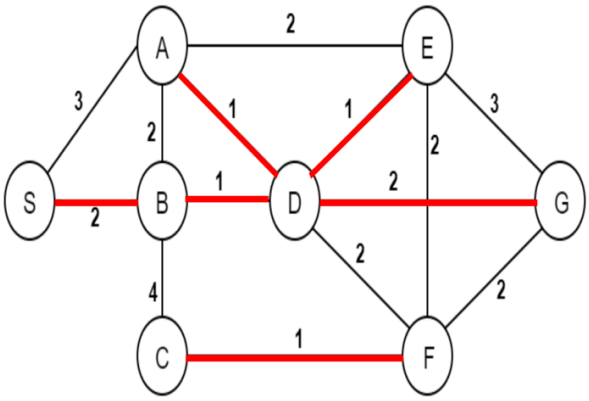


Figure 32: DG edge is added

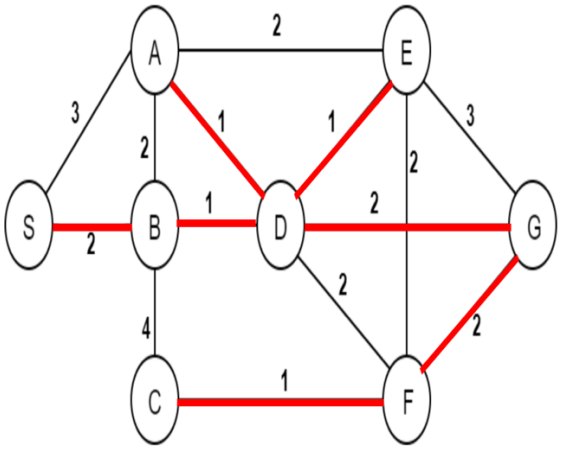


Figure 33: GF edge is added

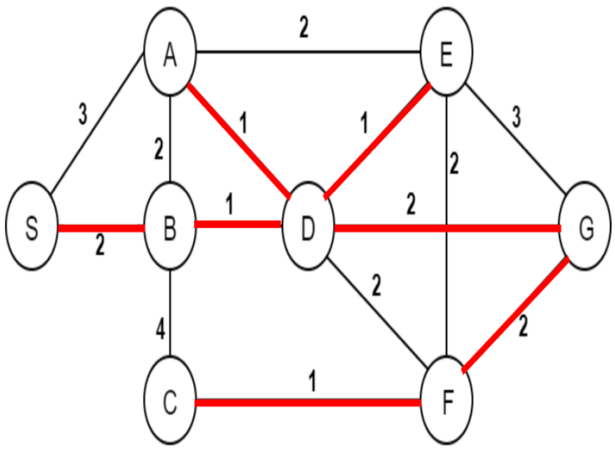


Figure 34: Minimum spanning tree by Kruskal

Question 4

Now, we will perform a breadth first traversal. I will use a queue to keep the order of the vertexes such that I will check their respected unvisited adjacent edges. Before adding nodes to the queue, I will mark them as visited. First, I will start with our starting node which is S. I will mark it as visited and add it to the queue. Then, I will remove S from the queue then check its adjacent unknown vertices. Thus, I will mark B and A as visited and added it to the queue respectively.

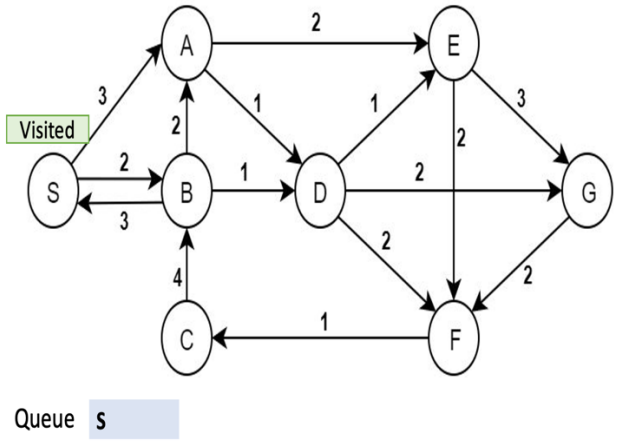


Figure 35: S is marked and added to the queue

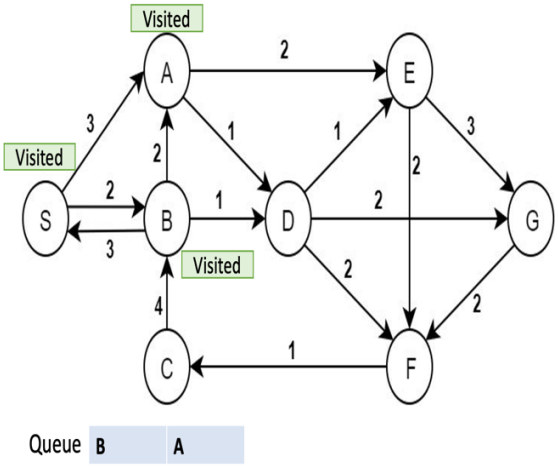


Figure 36: B and A are marked, added to the queue

Then, I will dequeue B and check its adjacent unvisited reachable vertices. Then, I will mark D as visited and added to the queue. Furthermore, I will dequeue A from the list and add E to the list since it is the adjacent unvisited vertex reachable from A

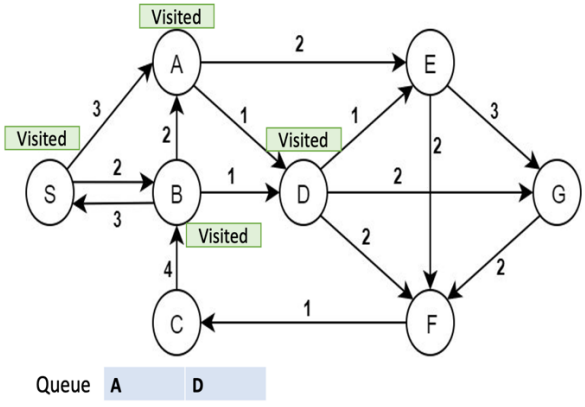


Figure 37: D is marked, added to the queue

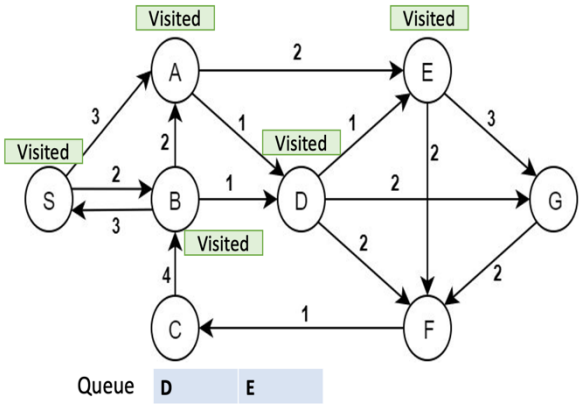


Figure 38: E is marked, added to the queue

Then, I will dequeue D and check adjacent unvisited vertices. Thus, I will mark F and G as visited and added it to the queue. Then, I will dequeue E from the queue. Since there is no adjacent unvisited vertices reachable from E, I will not add any vertex to the queue.

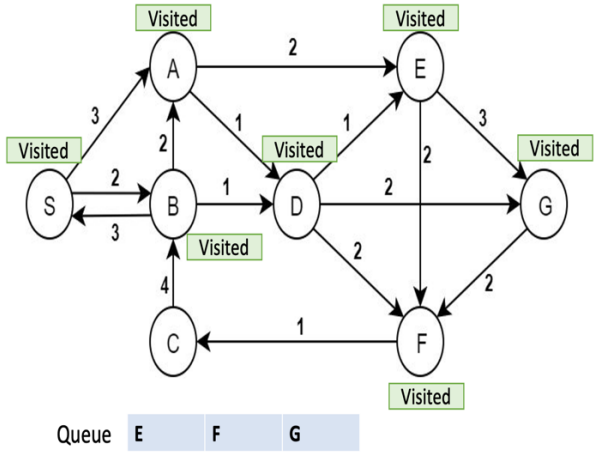


Figure 39: F and G are marked and added

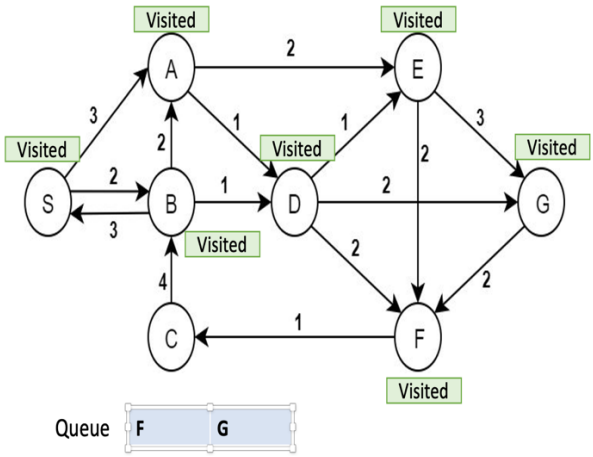


Figure 40: No vertices added to the queue in this step

Then, I will dequeue F and check for the unvisited adjacent vertices. Hence, I will mark C as visited add it to the queue. Then, I will dequeue G and check for the adjacent unvisited vertices. I will not add any vertex in this step since there is not any unvisited vertices left.

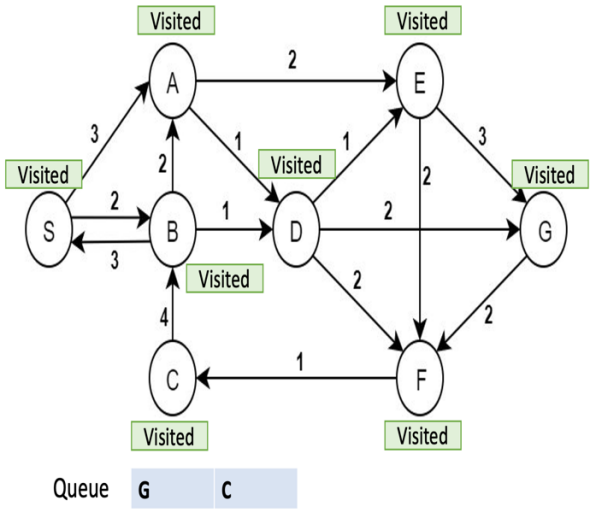


Figure 41: C is marked and added

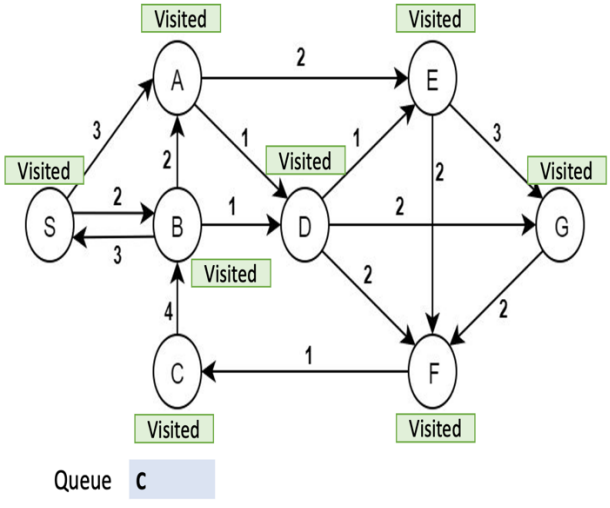


Figure 42: No vertices added in this step

Then, I will dequeue C from the list and check for adjacent unvisited edges. Since there is no adjacent unvisited edges, I will not add any vertex to the queue. At next step, the queue is empty. So, we have finished our breadth-tree traversal. After ordering the vertices with respect to their adding order to the queue, the output will be as follows.

S, B, A, D, E, F, G, C

Breadth-first tree of the graph will be shown with red edges in the graph figure, it will be as follows.

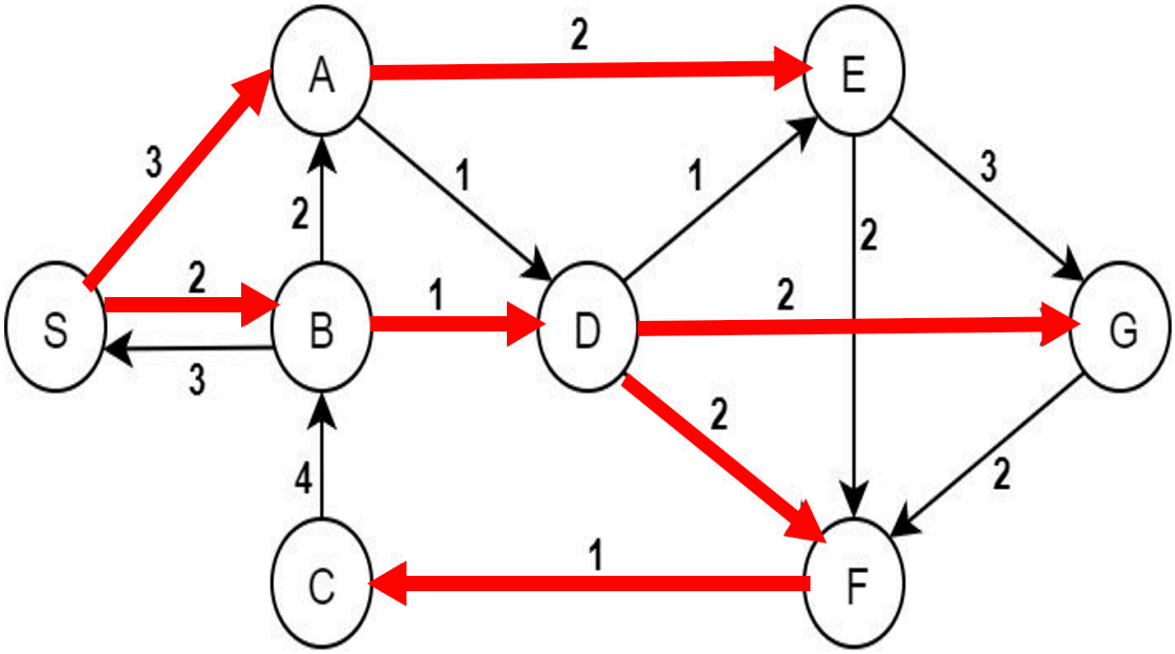


Figure 43: Breadth-first tree of the graph

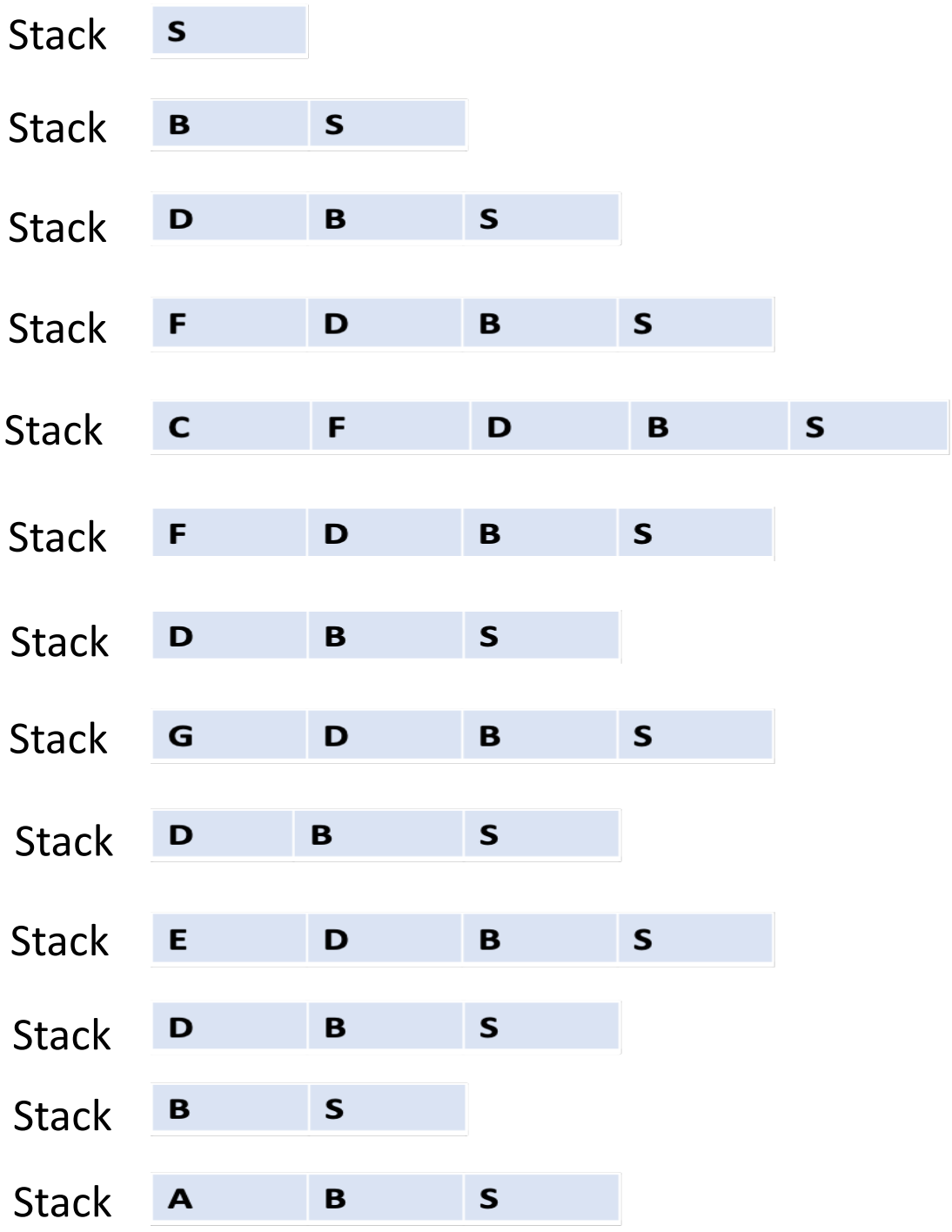
Question 5

Part a

I will perform depth-first traversal on the graph. We have a starting vertex which is S, I will mark S as visited and then I will check the adjacent unvisited vertices reachable from S. Then, I will make the same executions in the reachable edges. I will use a stack. When I am going to visit an unvisited adjacent vertex, I will mark it as visited and push it to the stack.

Hence, I will start with our starting vertex S I will mark it as visited and push to the stack then I will check the unvisited adjacent vertices reachable from S. Then, I will see I can select between B and A. I choose arbitrarily and select B mark it as visited push to the stack and I will differ from breadth-first traversal at this step I will go a one level deeper and check the unvisited adjacent matrixes reachable from B. Therefore, DFS can be easily implemented in a recursive manner.

So, my stack will be changing in the following order for each step



Stack	B	S

Stack	S

At next step our stack will be empty so depth-first traversal will be finished our vertex order with respect to depth-first traversal will be

S, B, D, F, C, G, E, A

Depth-first tree of the graph will be shown with red edges on the graph in the following figure

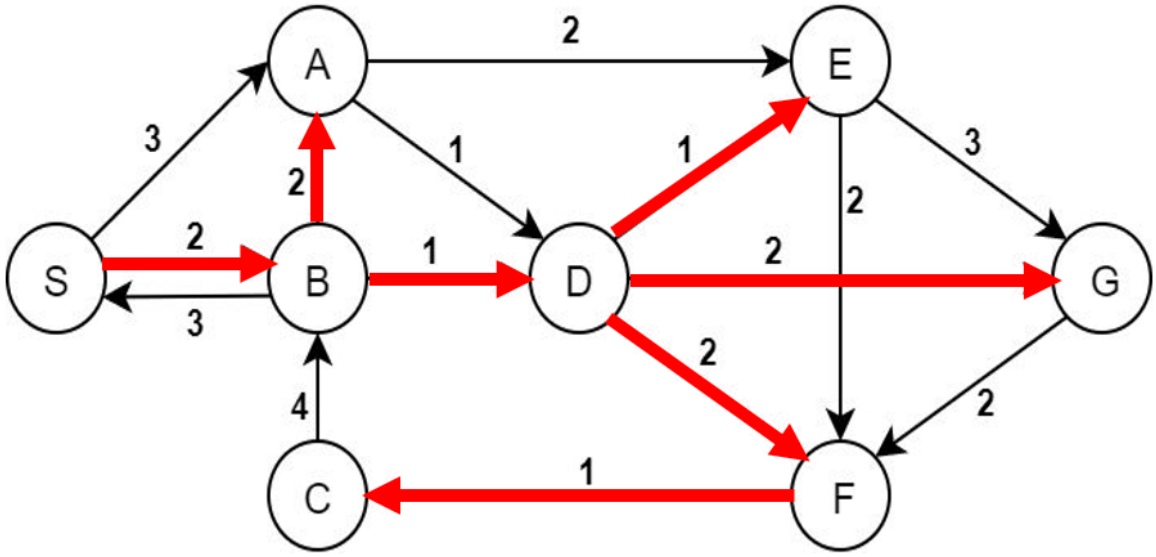


Figure 44: DFS of the graph

Part B

Now we will check the post-order numbers of the nodes in the graph. Post-order numbers of the nodes will be as follows

S	A	B	C	D	E	F	G
8	6	7	1	5	4	2	3

Table 1: Post-order number of the nodes of the graph

Part C

Now we will check the pre-order numbers of the vertices in the graph. They will be as follows

S	A	B	C	D	E	F	G
1	8	2	5	3	7	4	6

Table 2: Pre-order number of the nodes of the graph

Part D

Now, I will check the arc characterization in our graph. Let us remember the types of arcs which are as follows: Tree arcs, Cross arcs, Backward arcs and Forward arcs.

Tree Arcs (represented with blue)

- SB, BD, DF, DG, DE, BA, FC

Cross Arcs (represented with orange)

- AD, AE, EG, EF, GF

Forward Arcs (represented with green)

- SA

Backward Arcs (represented with purple)

- BS, CB

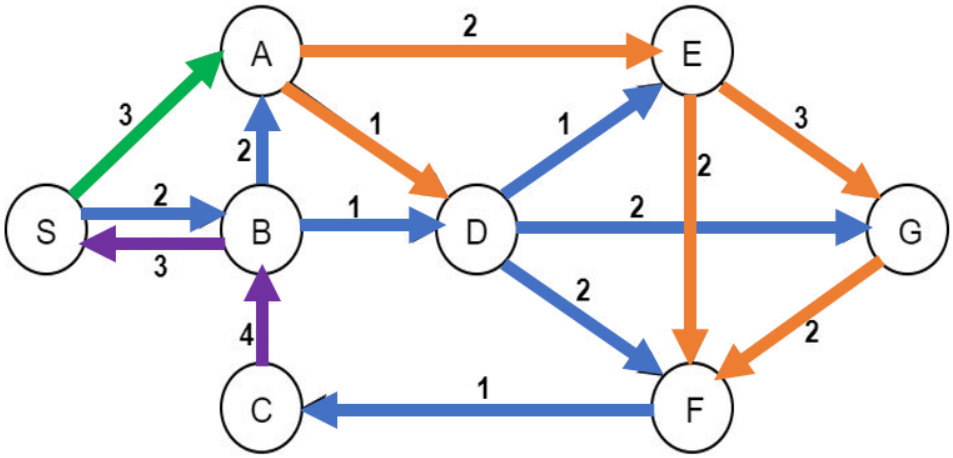


Figure 45: Representation of arcs with respect to their characteristics

Question 6

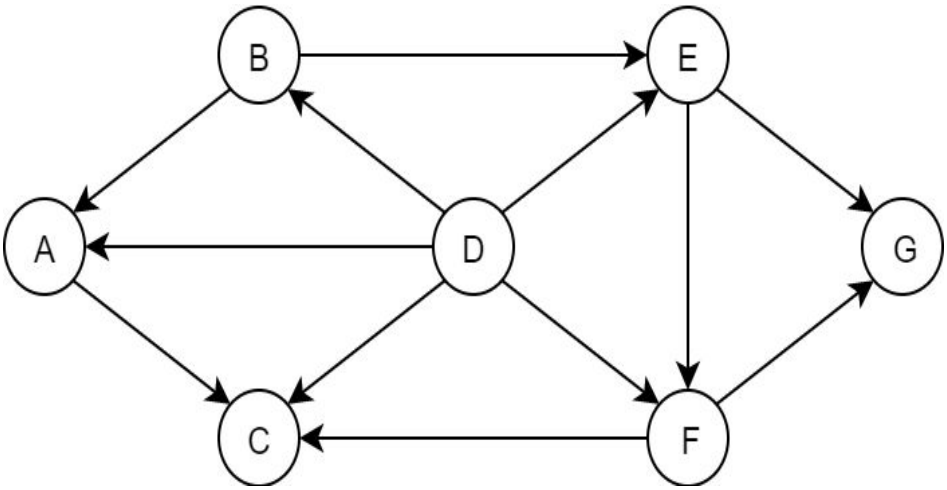


Figure 46: Graph for the topological order

We have a directed acyclic graph, so we can perform topological ordering. First let's check vertices with in-degree 0 and out-degree 0 at initial point.

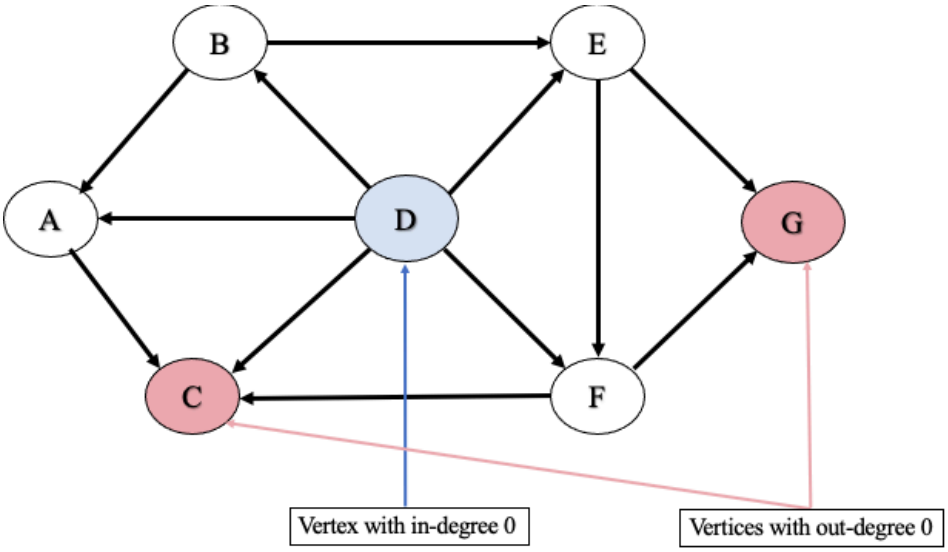


Figure 47: Initial state of the graph for in-degree and out-degree values

I will add D to the queue initially since it is the only one with in-degree 0. Then, I start with D since it is the only edge with in-degree 0. So, I dequeue vertex D from the queue and check the adjacent vertices and decrement the in-degree values of adjacent vertices by 1. If any vertex has in-degree value as zero, I will add the vertex to the queue. I will add B to the queue in this step. At the end of this step, what I will have in the topological order will be;

D

At next step, I will dequeue B and check the adjacent vertices and decrement the in-degree values of adjacent vertices by 1. In this step, I will add A and E to the queue. At the end of this step, I will have in the topological order;

D, B

Then, I will dequeue A. after checking to decrement in-degree values of adjacent vertices to A, I will not add any vertex at this step to the queue. So, I will have at the end of this step;

D, B, A

Then, I will dequeue vertex E in this step. Then, I will decrement the in-degree values of adjacent matrixes to vertex E in this step. I will add F to the queue in this step. At the end of this step, I have in topological order;

D, B, A, E

Then, I will dequeue F and decrement the adjacent vertices of F' in-degree value by one. Then, I will add C and G to the queue. At the end of this step, I have;

D, B, A, E, F

At next step, I will dequeue C and decrement the adjacent vertices of C's in degree values by 1. I will not add any other vertex to the queue at this step. At the end of this step, we have;

D, B, A, E, F, C

Then, I will dequeue G perform the same execution mentioned before for its adjacent vertices. Then, I am not adding any other vertex to the queue. At the end of this step, I have;

D, B, A, E, F, C, G

At next step, I see that queue is empty. Thus, I will skip the queue executions, so I will check whether we have a cycle or not. We do not have cycle, since we a directed acyclic graph. Thus, our topological ordering for this graph is done and the order will be as follows;

D, B, A, E, F, C, G