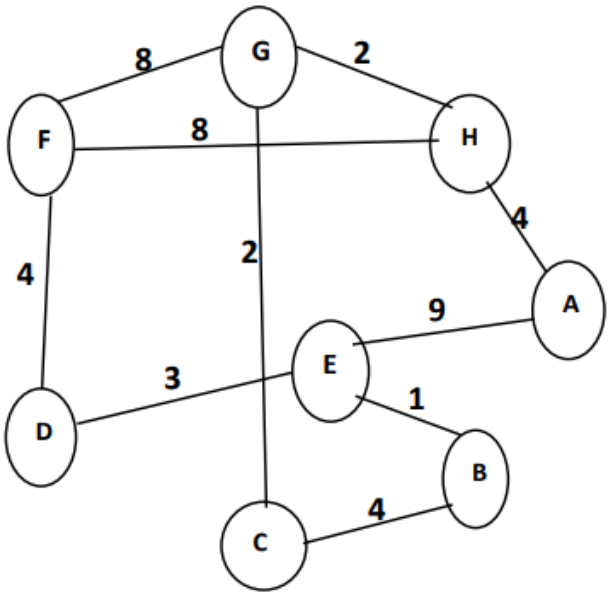
CS300 (Data Structures)

Homework 6 (Graphs)

Hagverdi Ibrahimli 30014

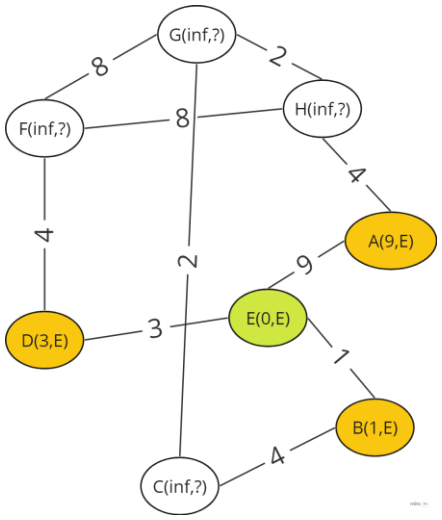
Question 1:

1) **(35 points)** Trace in detail the operation of Dijkstra's weighted shortest path algorithm for the undirected weighted graph below. Use **vertex E** as your start vertex.

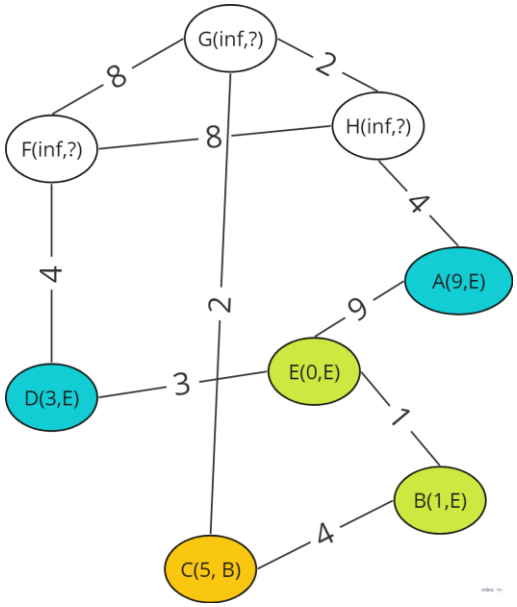


Solution:

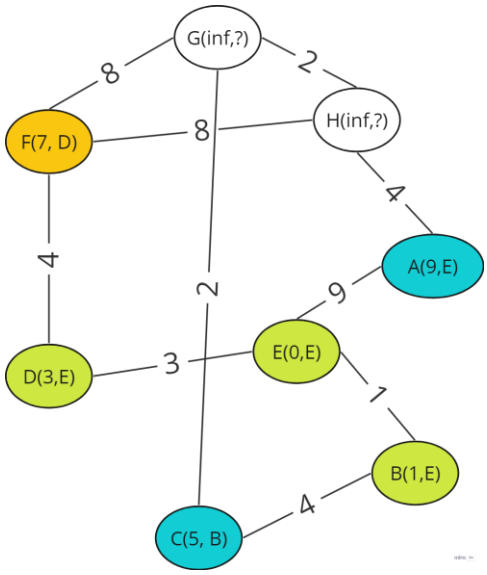
1. Selecting vertex E as the starting point and updating the distances of the adjacent vertices.



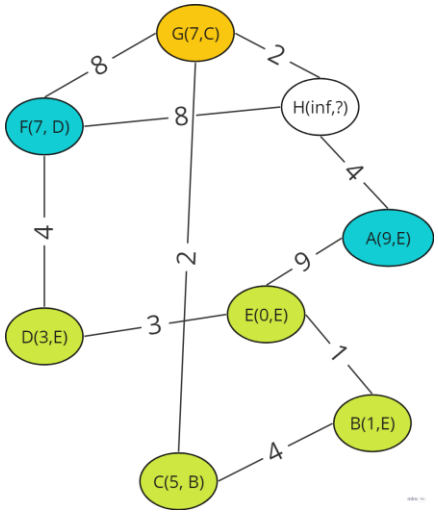
2. Selecting the vertex that is not visited with the smallest distance with respect to the vertex E. We chose vertex B and updated its adjacent vertices.



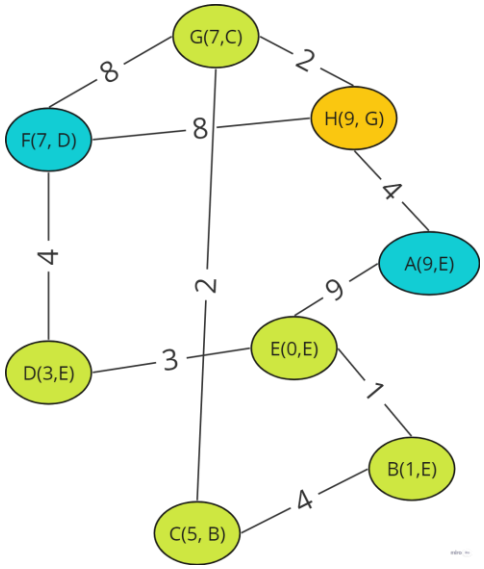
3. Selecting the vertex that is not visited with the smallest distance with respect to the vertex E. We chose vertex D and updated its adjacent vertices.



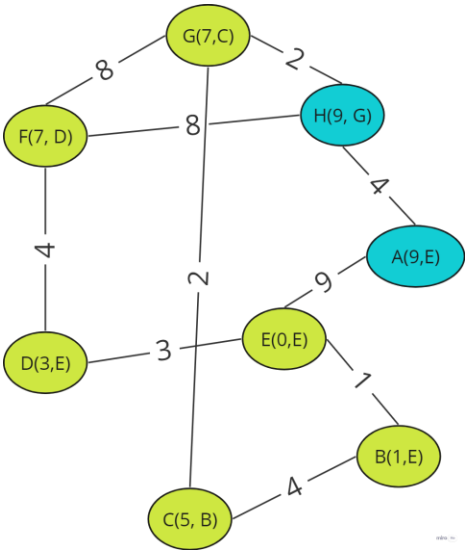
4. Selecting the vertex that is not visited with the smallest distance with respect to the vertex E. We chose vertex C and updated its adjacent vertices.



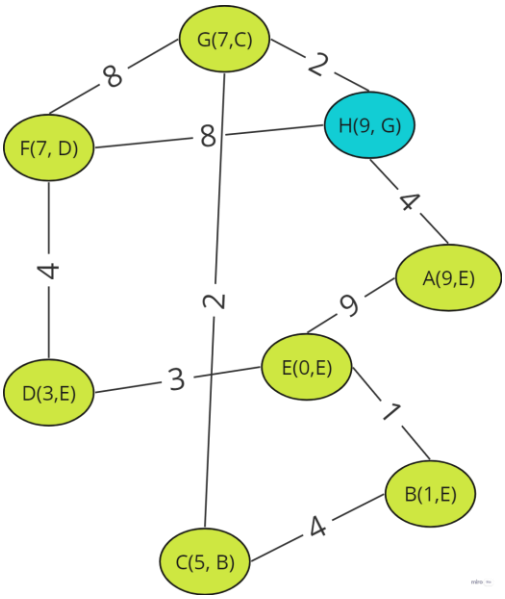
5. Selecting the vertex that is not visited with the smallest distance with respect to the vertex E. In this case, we can choose either F or G. We chose vertex G and updated its adjacent vertices.



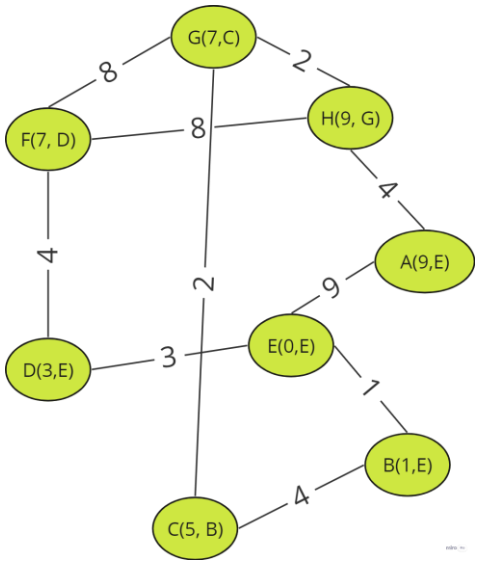
6. Selecting the vertex that is not visited with the smallest distance with respect to the vertex E. We chose vertex F and there is no change in its adjacent vertices.



7. Selecting the vertex that is not visited with the smallest distance with respect to the vertex E. We chose vertex A and there is no change in its adjacent vertices.

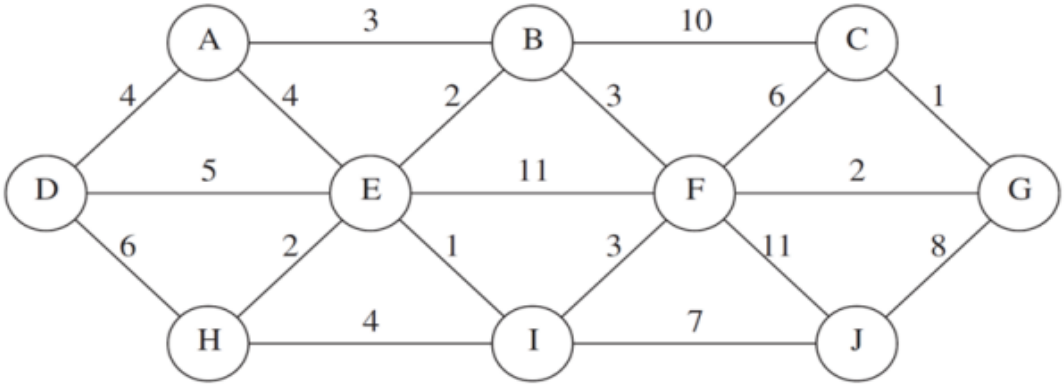


8. Selecting the vertex that is not visited with the smallest distance with respect to the vertex E. We chose vertex H and there is no change in its adjacent vertices. Tracing is completed.



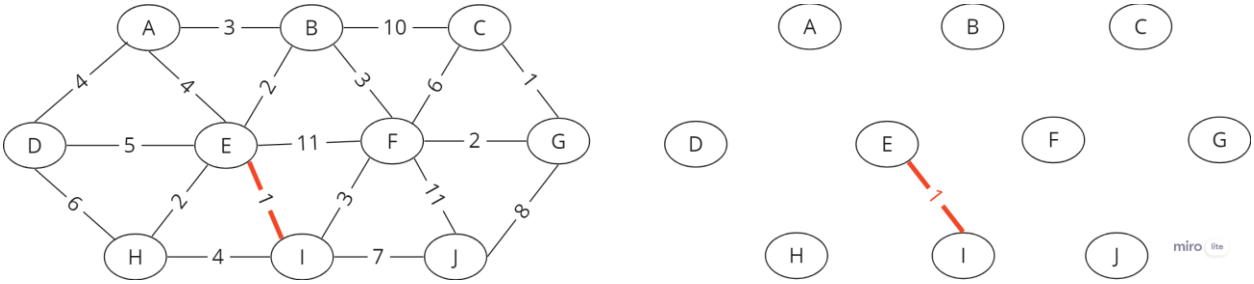
Question 2:

2) (20 points) Find a minimum spanning tree for the graph below using Kruskal's algorithm. Draw the resulting spanning tree as your solution where the edges are numbered with their selection order by Kruskal's algorithm.

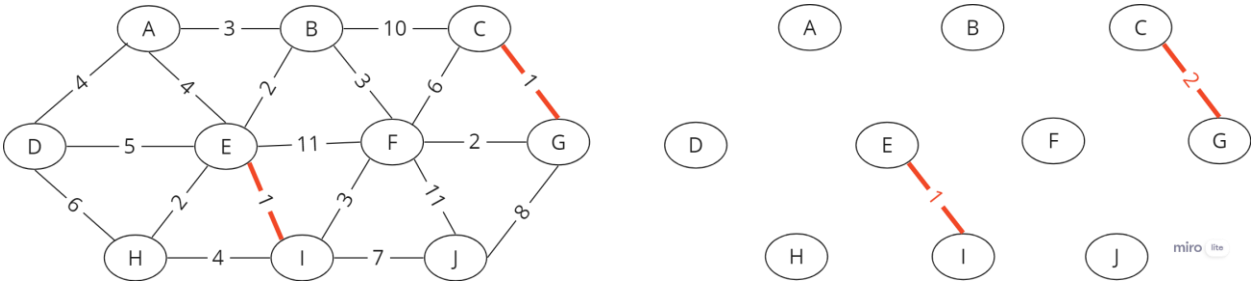


Solution:

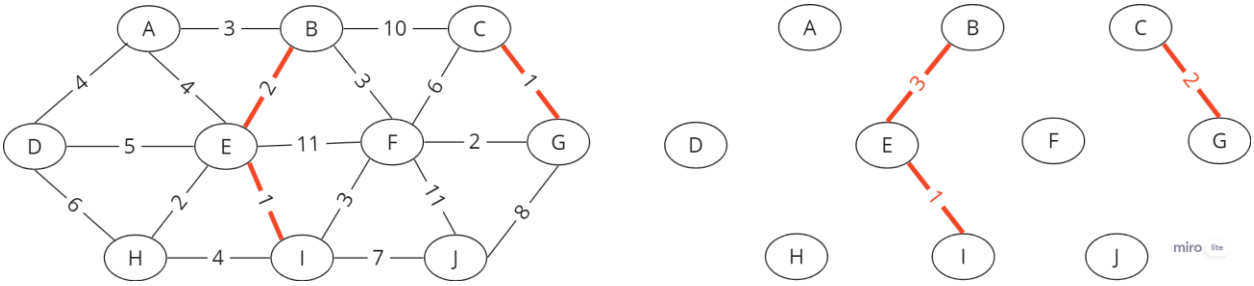
1. We get the edge with the minimum weight, which are E-I and C-G, but we chose E-I first and updated the minimum spanning tree accordingly.



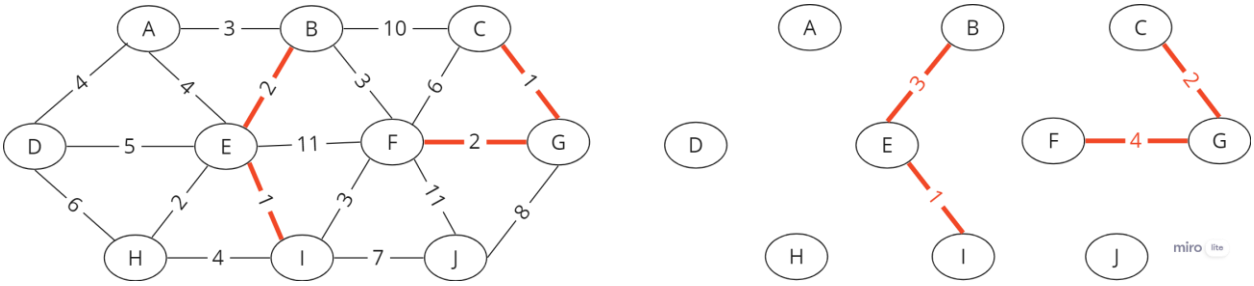
2. We get the next minimum edge which is C-G and update the minimum spanning tree accordingly.



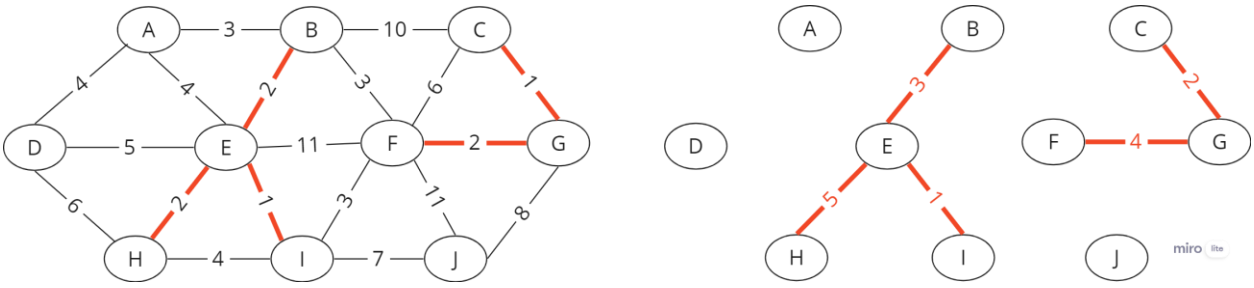
3. We get the next minimum edge which is B-E and update the minimum spanning tree accordingly.



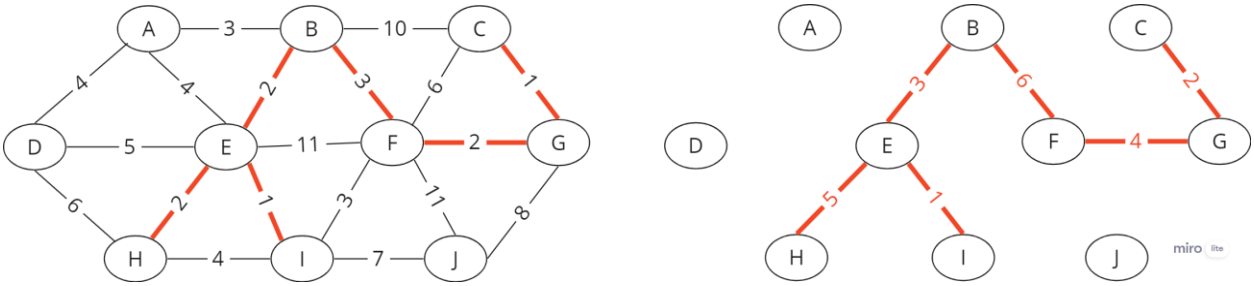
4. We get the next minimum edge which is F-G and update the minimum spanning tree accordingly.



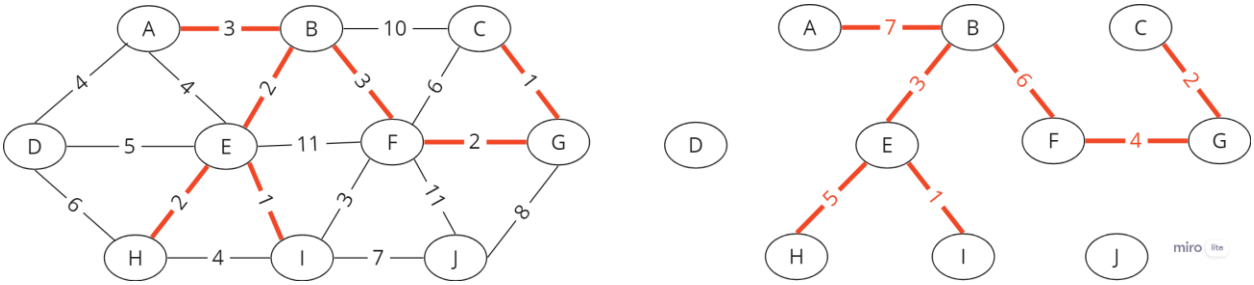
5. We get the next minimum edge which is E-H and update the minimum spanning tree accordingly.



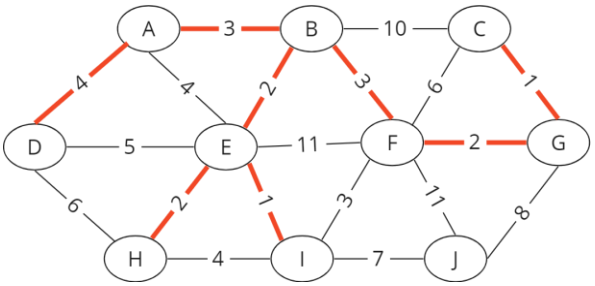
6. We get the next minimum edge which is B-F and update the minimum spanning tree accordingly.

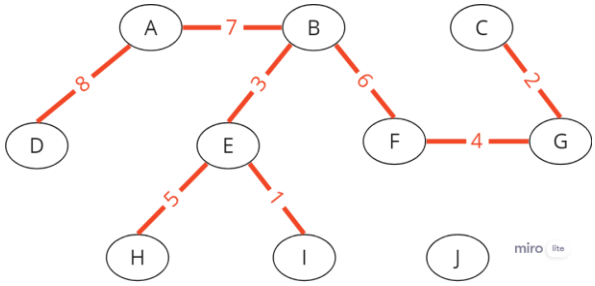


7. We get the next minimum edge which is A-B and update the minimum spanning tree accordingly.

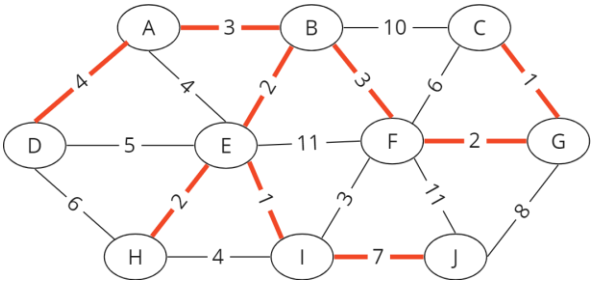


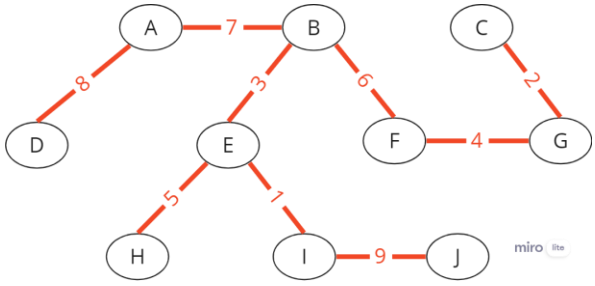
8. We get the next minimum edge which is A-D and update the minimum spanning tree accordingly.





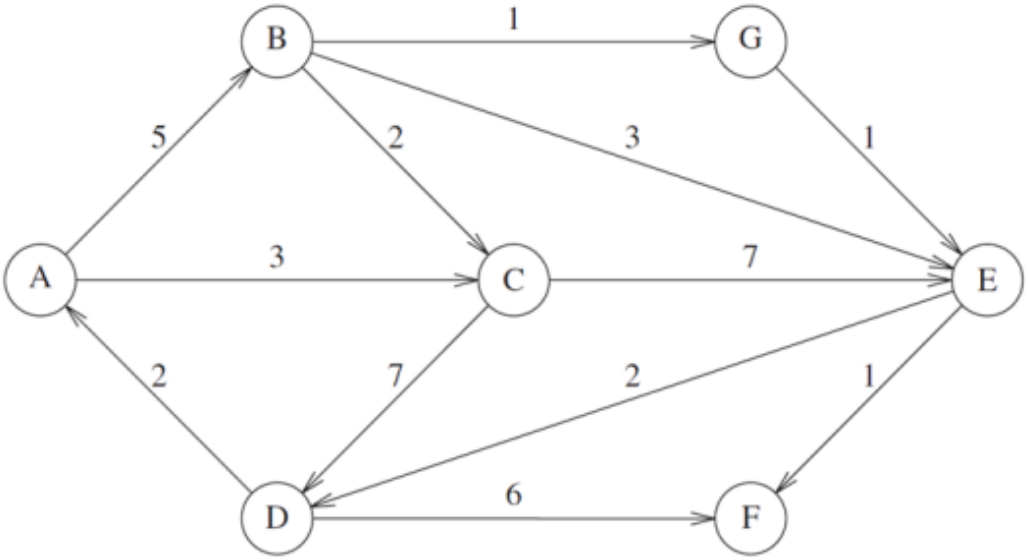
9. We get the next minimum edge which is I-J and update the minimum spanning tree accordingly. With this change, the minimum spanning tree is successfully achieved.





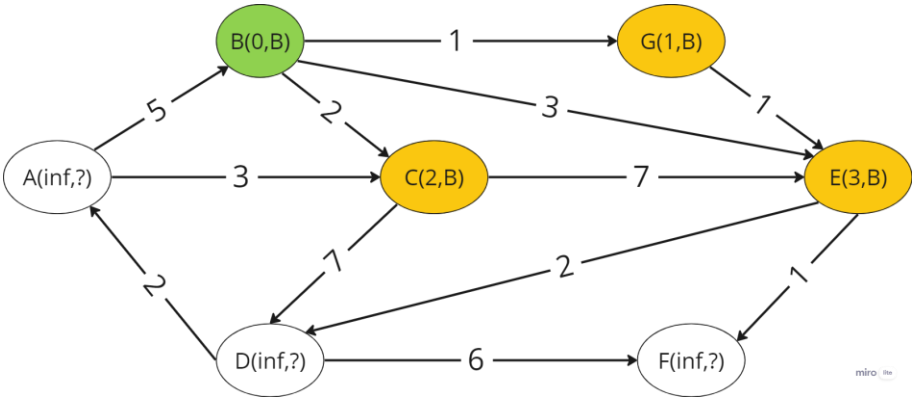
Question 3:

3) (25 points) Find the shortest path from *B* to ALL other vertices for the graph below. Show your steps while you construct the shortest path such as B-->E-->D.

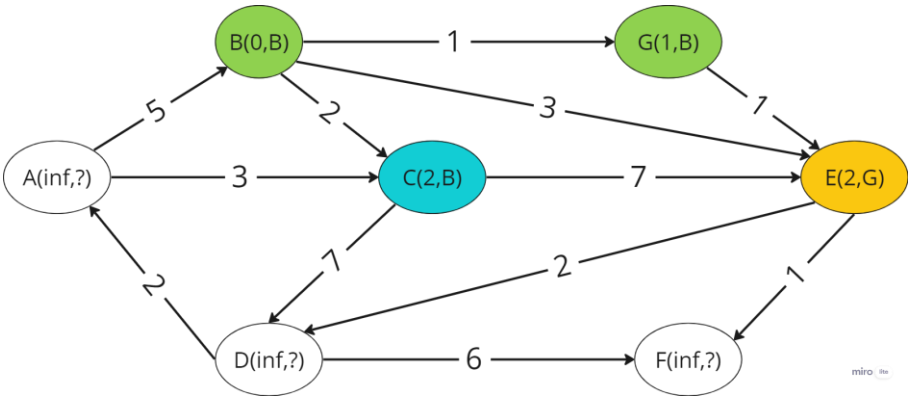


Solution:

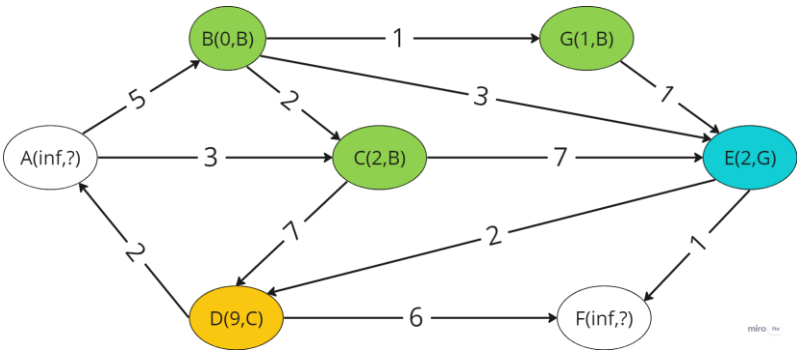
1. To find the shortest path from vertex B to all other vertices in the graph above, we will use Dijkstra's weighted shortest path algorithm for the directed graphs. First, we will select vertex B as the starting point and adjust the distances of the adjacent vertices.



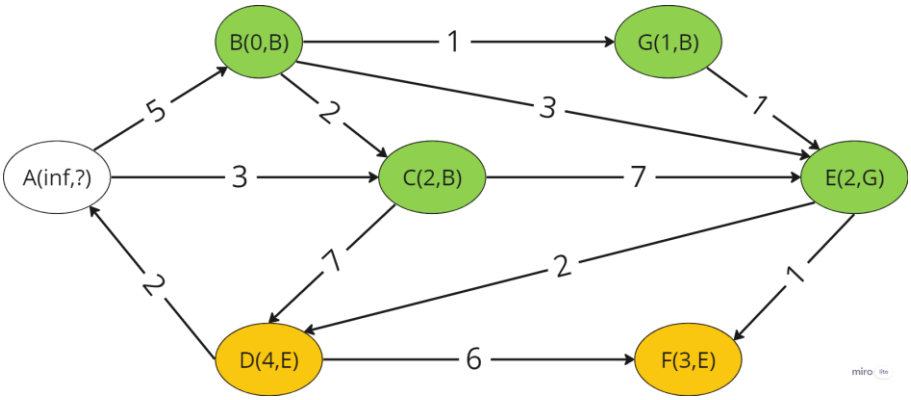
2. Selecting the vertex that is not visited with the smallest distance with respect to the vertex B. We chose vertex G and updated its adjacent vertices.



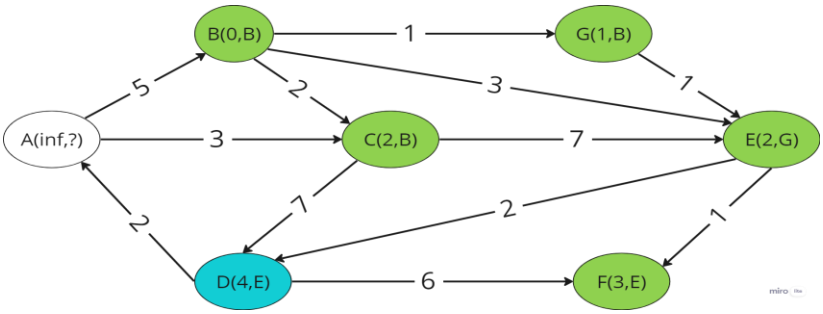
3. Selecting the vertex that is not visited with the smallest distance with respect to the vertex B. In this case, we have two options, C and E. We chose vertex C and updated its adjacent vertices.



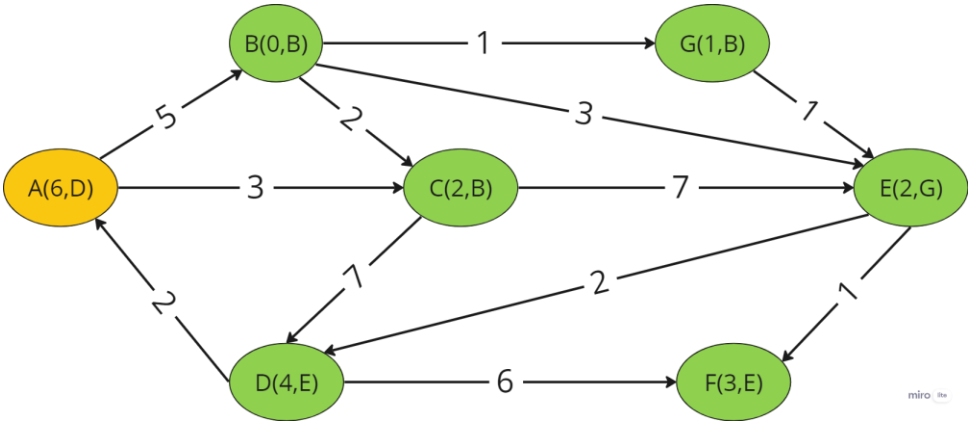
4. Selecting the vertex that is not visited with the smallest distance with respect to the vertex B. We chose vertex E and updated its adjacent vertices.



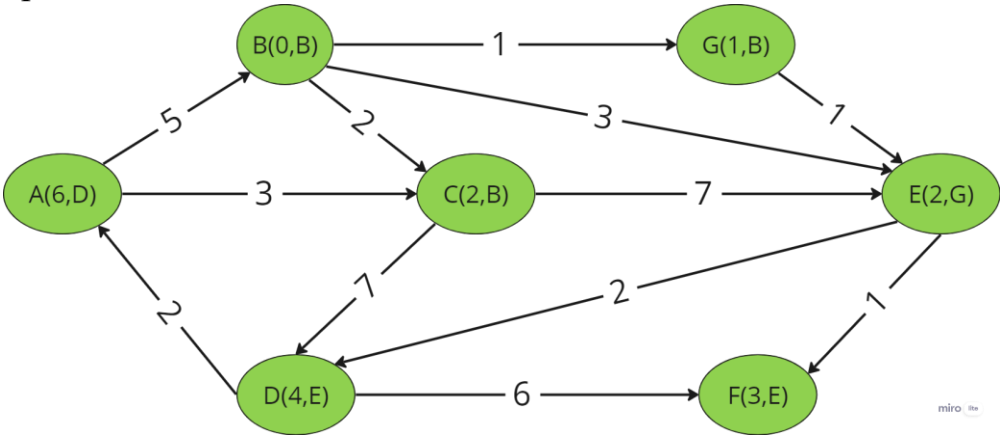
5. Selecting the vertex that is not visited with the smallest distance with respect to the vertex B. We chose vertex F and there is nothing to update since F has no outgoing edges.



6. Selecting the vertex that is not visited with the smallest distance with respect to the vertex B. We chose vertex D and updated its adjacent vertices.



7. Selecting the vertex that is not visited with the smallest distance with respect to the vertex B. We chose the last unvisited vertex A and there is nothing to update.



8. Now that the graph is ready, we can see that the values of each vertices show the actual minimum distance from the original vertex B, and they also show the latest vertex that had changed their distance value. So, by tracking back the vertices that lead to the shortest path to the vertex X, we can reverse the order and find out the actual minimum path from B to X.

   - Shortest path from B to G:
     G is 1 distance away from B and is preceded by B itself:
     B => G

   - Shortest path from B to E:
     E is 2 distances away from B and is preceded by G which is preceded by B:
     B => G => E

   - Shortest path from B to F:
     F is 3 distances away from B and is preceded by E which is preceded by G which is preceded by B:
     B => G => E => F

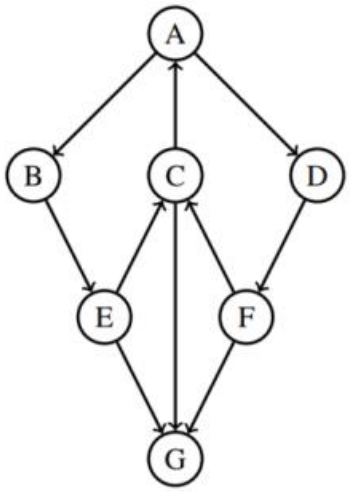
   - Shortest path form B to D:
     D is 4 distances away from B and is preceded by E which is preceded by G which is preceded by B:
     B => G => E => D

   - Shortest path from B to A:
     A is 6 distances away from B and is preceded by D, which is preceded by E, which is preceded by G which is preceded by B:
     B => G => E => D => A

   - Shortest path from B to C:
   - C is 2 distances away from B and is preceded by B itself:
     B => C

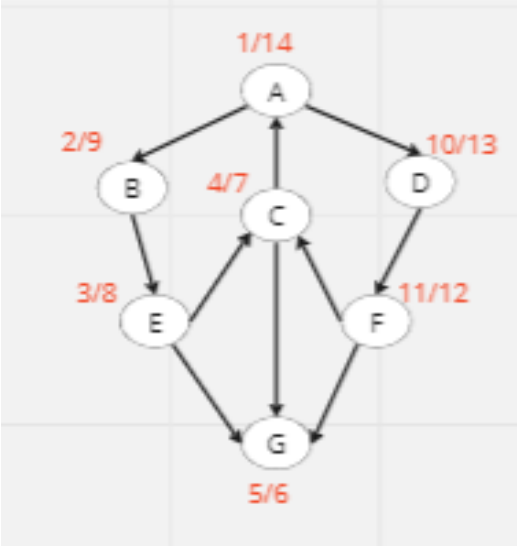
Question 4:

4) **(20 points)** Perform a depth-first search on the following graph starting at A. Label every edge in the graph with T if it's a tree edge, B if it's a backward edge, F if it's a forward edge, and C if it's a cross arc (edge). Assume that whenever faced with a decision of which node to pick from a set of nodes, pick the node whose label occurs earliest in the alphabet.

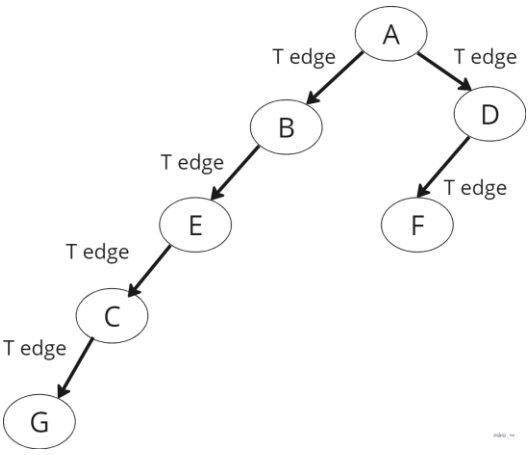


Solution:

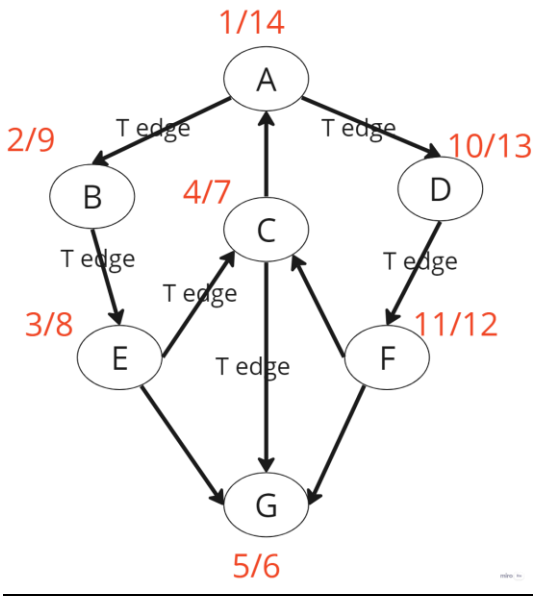
1. First, we should perform a depth-first search (DFS) on the graph starting from the vertex A. Below is the result of the DFS operation. Note the syntax *x/y* which we use for timing of the operations (*x* denotes the starting time and *y* denotes the ending time of the recursion).



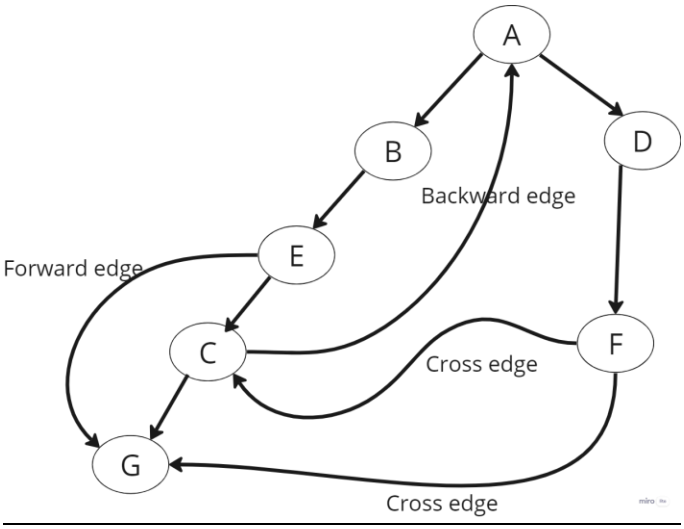
2. Second, we draw the tree representation of the nodes we visited by running the depth-first search on the provided graph. Note that the edges which take us to previously unvisited nodes are called the tree edges which are simply the edges that are drawn in our DFS tree below.



3. We name the same edges on our graph as ‘tree edges’ that we found from step 2.



4. Notice that there are still some edges not named in our graph such as the E -> G, C -> A, F -> C, F -> G. We can also name them by drawing the corresponding edges on our DFS tree representation of the graph.



5. Lastly, we insert the missing names of the edges into our graph. Note the shorthand notations where T edge denotes the tree edge, B edge denotes the

Backward edge, C edge denotes the Cross edge, and F edge denotes the forward edge.