CS 300 ASSIGNMENT #4
ZEYNEP KURTULUS 29045

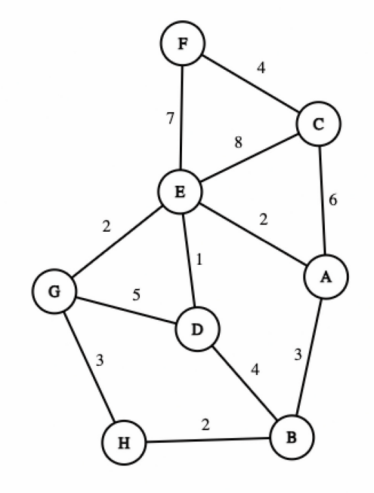



Figure 1: An undirected weighted graph.

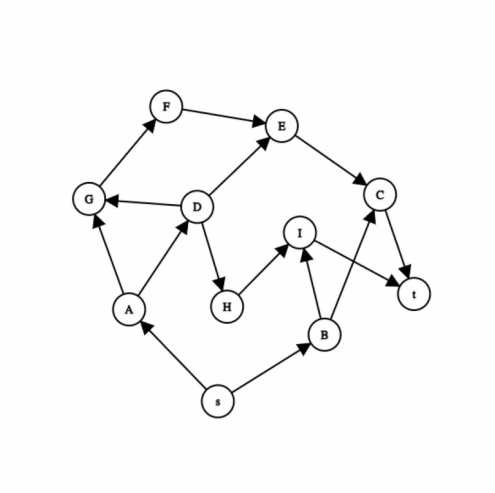
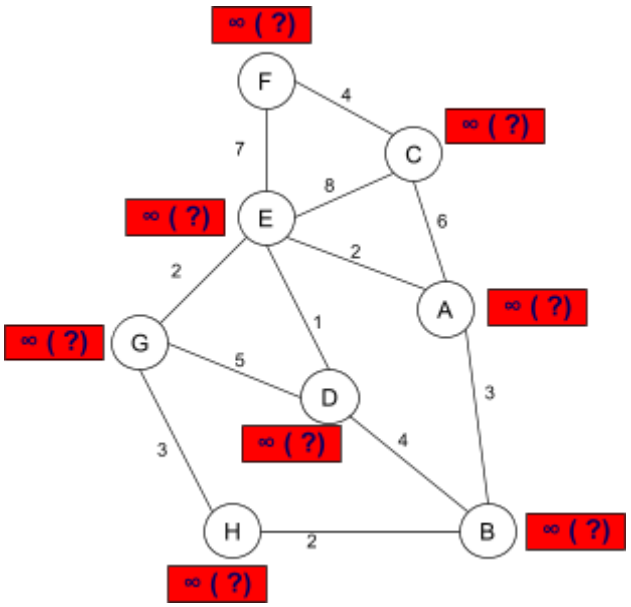



Figure 2: A directed acyclic graph.

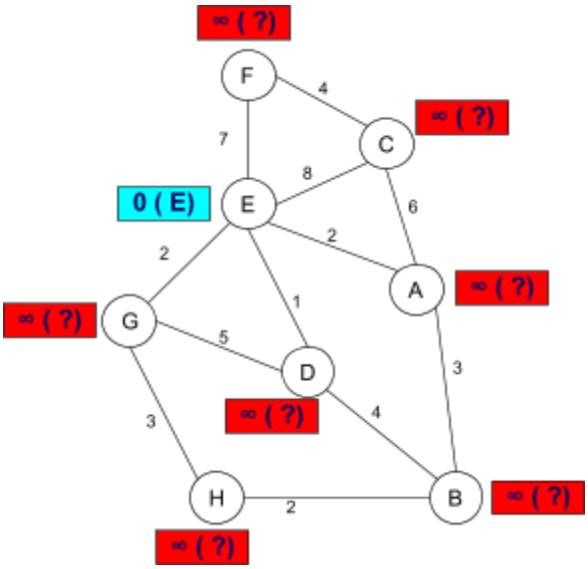
## Question 1

- Trace the Dijkstra's weighted shortest path algorithm on the graph given in Figure1 Use vertex E as your start vertex.**REPRESENTATION: BLUE COLOR FOR KNOWN NODES RED FOR UNKNOWN NODES, X = DISTANCE Y = PREVIOUS NODE** X (Y)

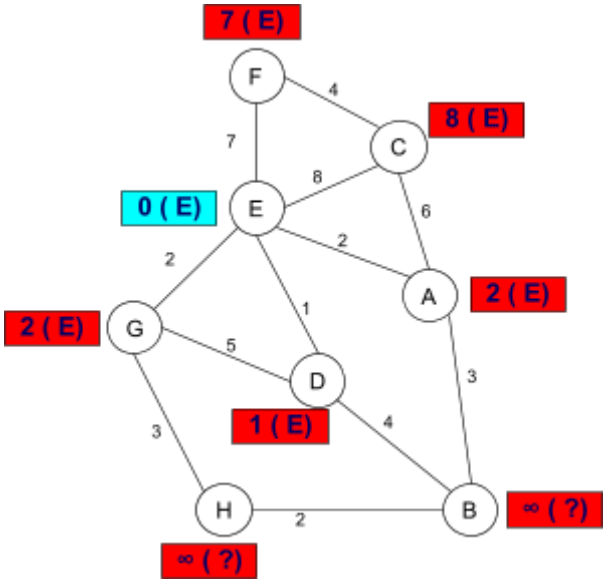
Initial State:



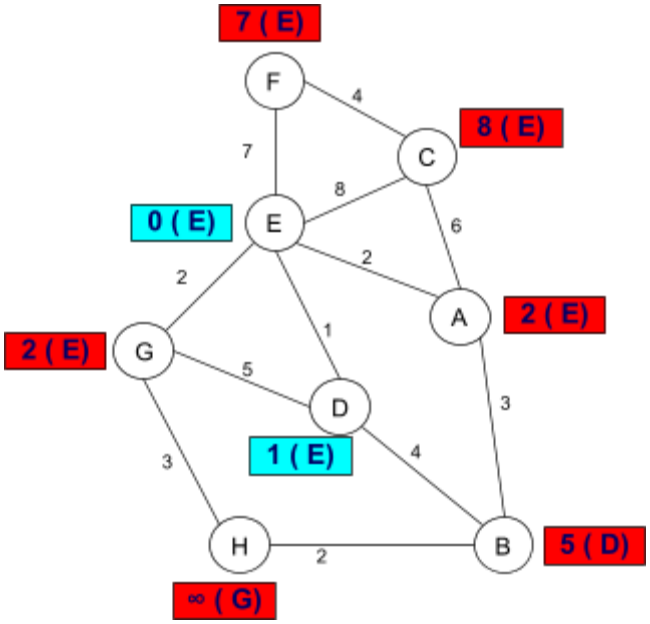
1st step: Select vertex E as the starting vertex and update E according to the starting vertex



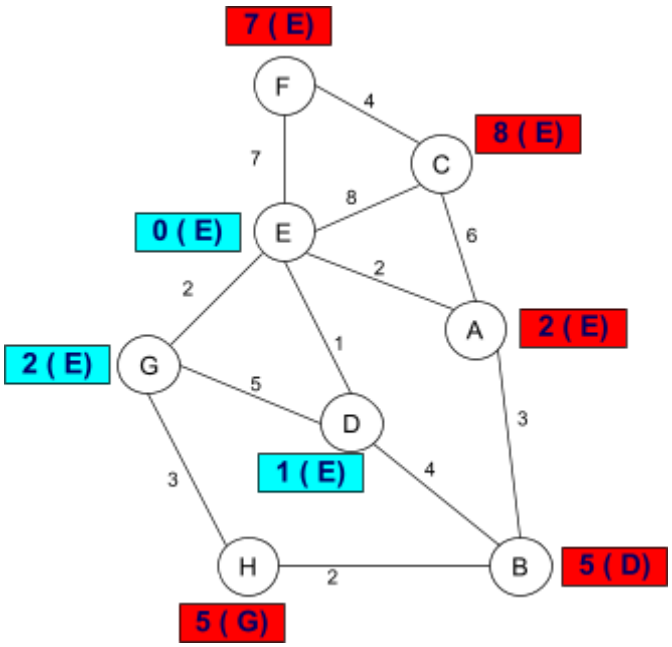

2nd step: Update the adjacent vertices of E (their path and distance values)



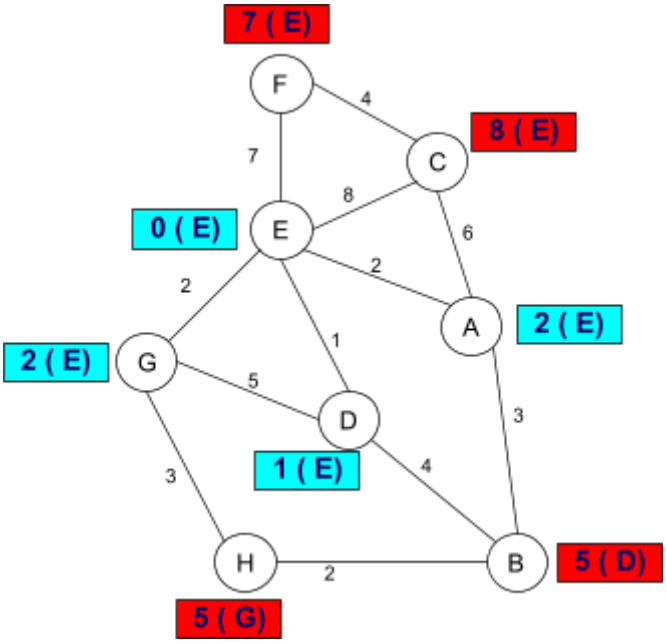
3rd step: Select the vertex with the smaller distance, which is vertex D in this case, make it known and update its adjacent vertices if needed (i.e. a shorted path is found, we updated B because 5 is smaller than infinity)



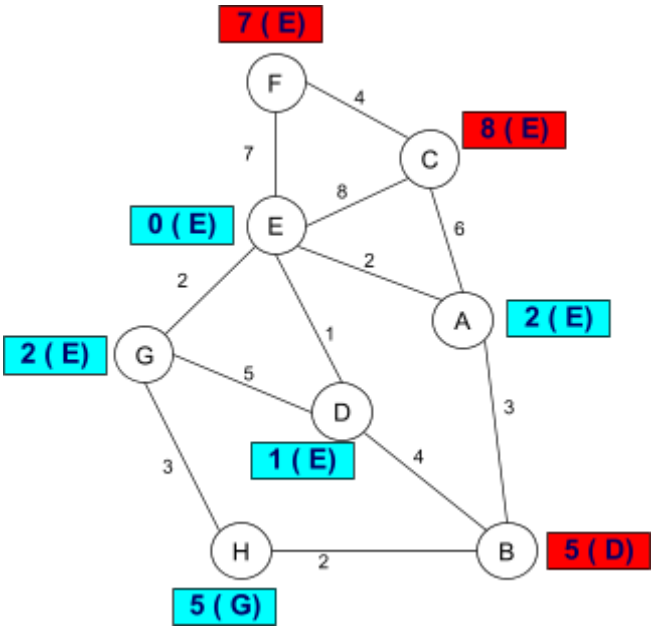

4th Step: Select the vertex with the smaller distance, which is vertex G or A in this case, I chose G; make G known and update its adjacent vertices if needed (i.e. a shorted path is found, however there is no shorted path found so no need to update.)



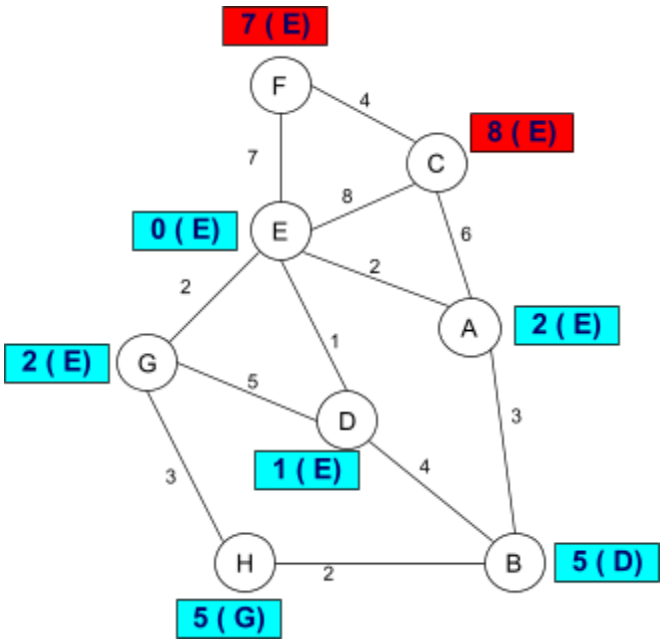
5th step: Select the vertex with the smaller distance, which is vertex A in this case; make A known and update its adjacent vertices if needed  (i.e. a shorted path is found, however there is no shorted path found so no need to update.)



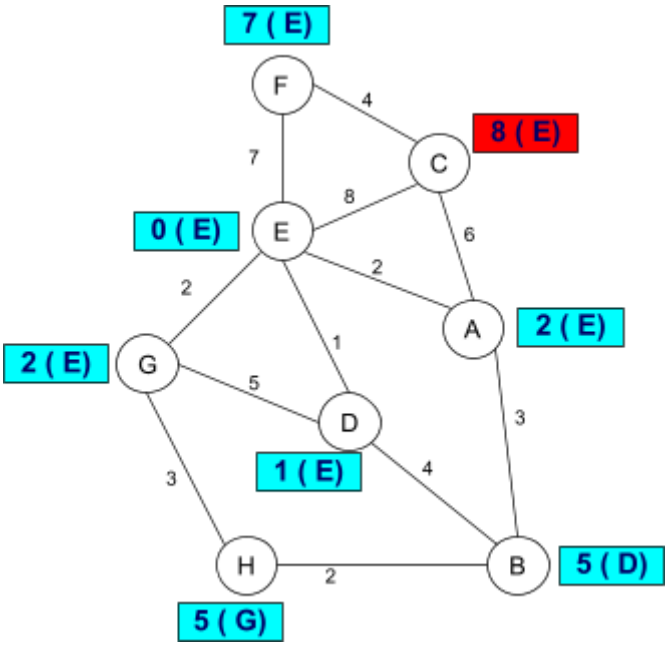

6th step: Select the vertex with the smaller distance, which is vertex H or B in this case, I chose G; make H known and update its adjacent vertices if needed  (i.e. a shorted path is found, however there is no shorted path found so no need to update.)



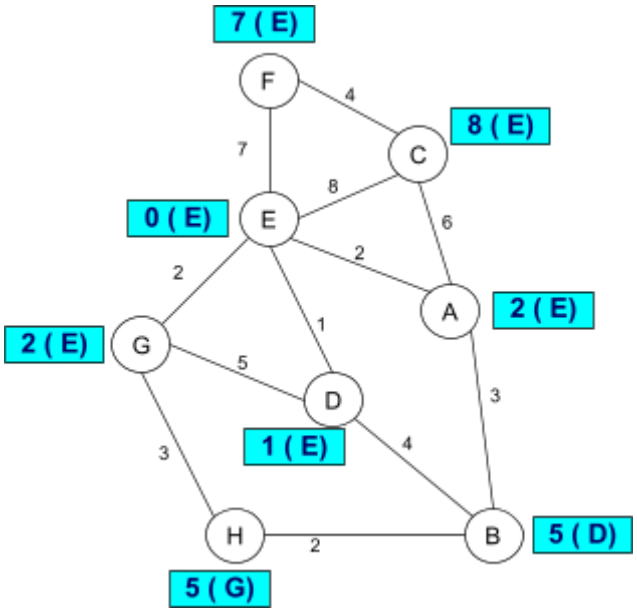
7th step: Select the vertex with the smaller distance, which is vertex B in this case; make B known and update its adjacent vertices if needed (i.e. a shorted path is found, however there is no shorted path found so no need to update.)



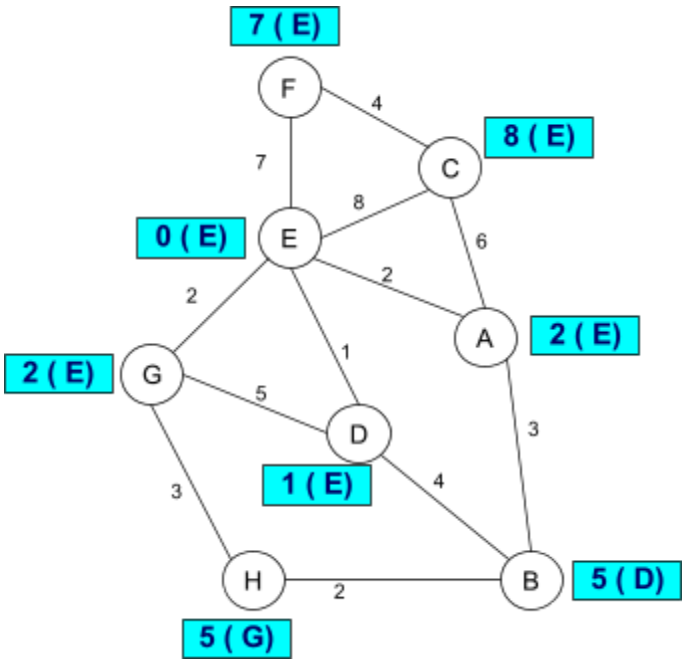

8th step: Select the vertex with the smaller distance, which is vertex F in this case; make F known and update its adjacent vertices if needed (i.e. a shorted path is found, however there is no shorted path found so no need to update)



9th step: Select the vertex with the smaller distance, which is vertex A in this case; make A known and update its adjacent vertices if needed  (i.e. a shorted path is found, however there is no shorted path found so no need to update.)




10th step: Select the vertex with the smaller distance; note that all of the vertices are known and we have completed the Dijekstra's Shortest Path Algorithm.

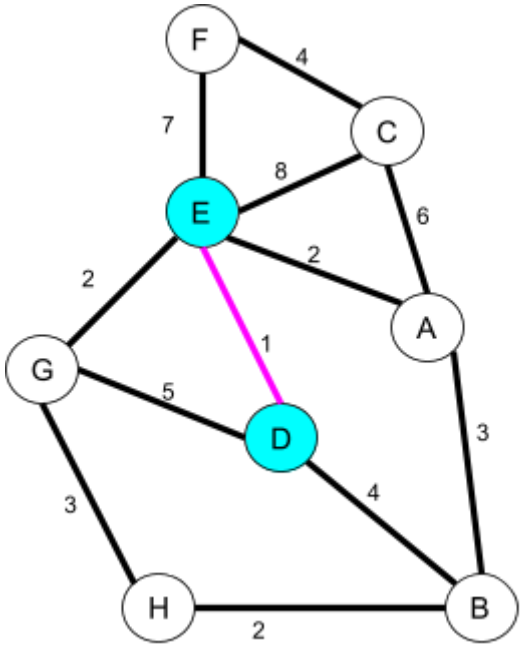



The path is: E, D, G, A, H, B, F, C

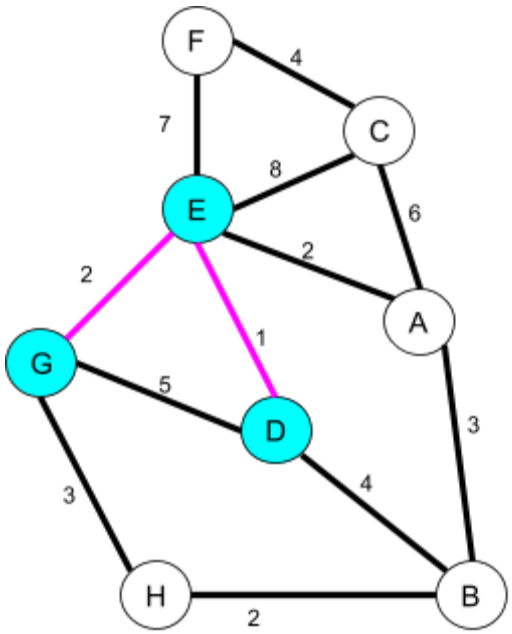
Question 2:

- Trace the Prim's minimum spanning tree algorithm on the graph in Figure 1. Use vertex E as your start vertex. **REPRESENTATION: PINK COLORED EDGES INCLUDED IN THE PRIMS SPANNING TREE ALGORITHM, BLUE COLORED NODES INDICATES THEY ARE KNOWN**

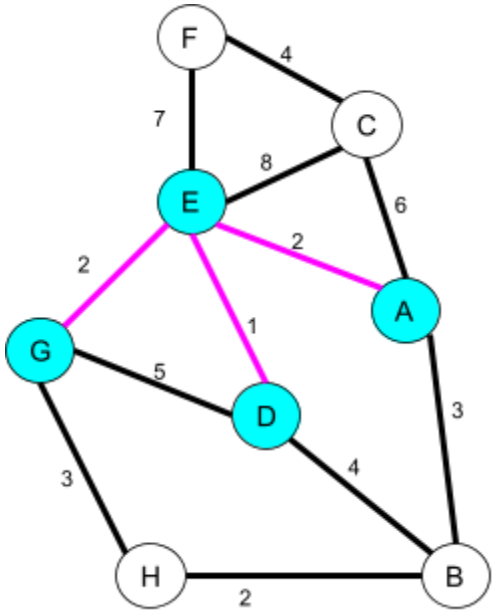
1st step: Start with vertex E and make it known, then pick the edge with the lowest weight that connects a known vertex to an unknown vertex. For this case pick the edge E-D, make D known.



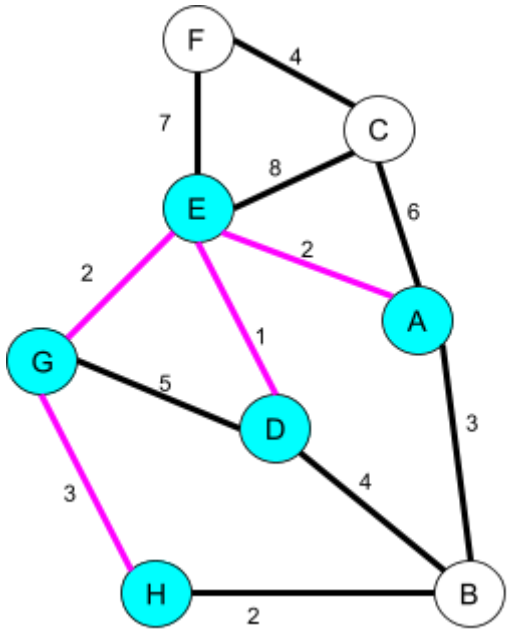
2nd step: Pick the edge with the lowest weight that connects a known vertex to an unknown vertex. For this case pick the edge E-G or E-A, I choose E-G; make G known.



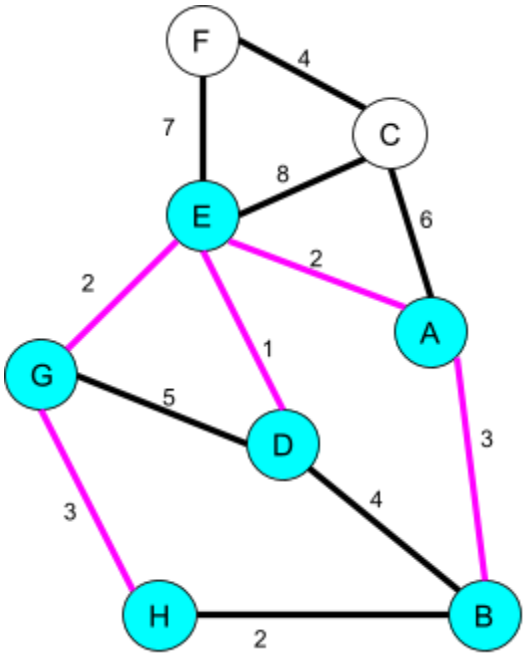

3rd step: Pick the edge with the lowest weight that connects a known vertex to an unknown vertex. For this case pick the edge E-A, I choose E-A; make A known.



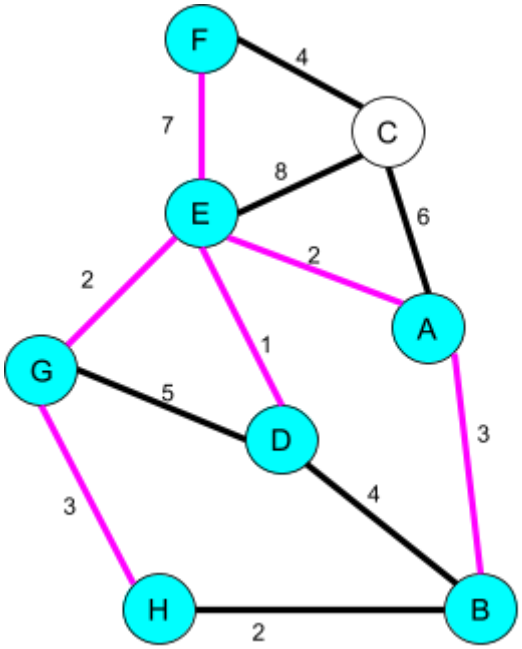
4th step: Pick the edge with the lowest weight that connects a known vertex to an unknown vertex. For this case pick the edge G-H, A-B or D-B, I choose G-H; make H known.



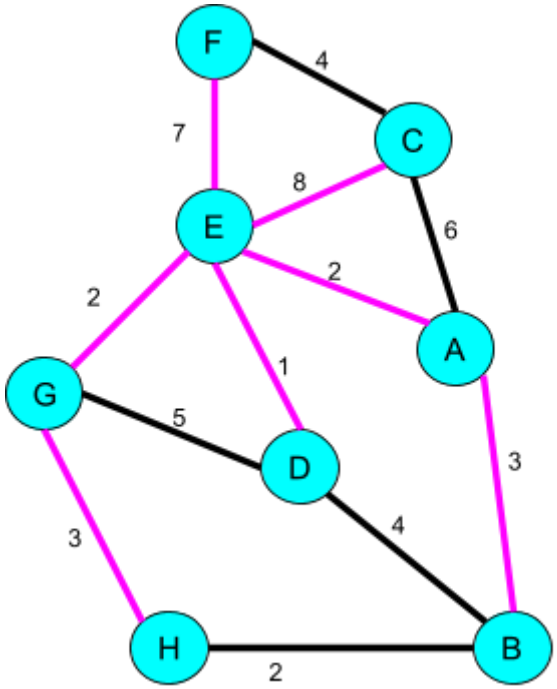

5th step: Pick the edge with the lowest weight that connects a known vertex to an unknown vertex. For this case pick the edge A-B or D-B, I choose A-B; make B known.



6th step: Pick the edge with the lowest weight that connects a known vertex to an unknown vertex. For this case pick the edge E-F: make F known.



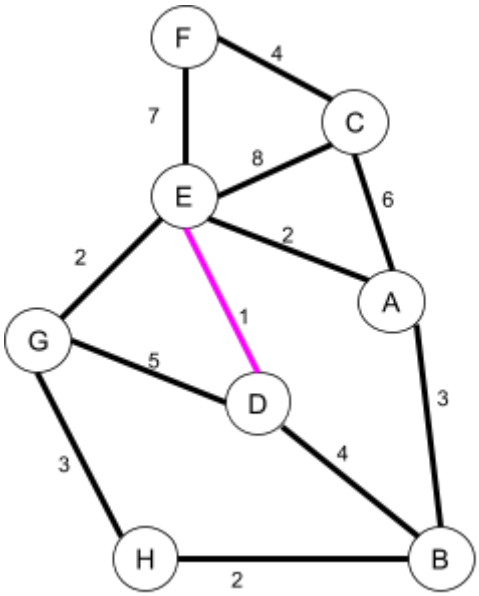

7th step: Pick the edge with the lowest weight that connects a known vertex to an unknown vertex. For this case pick the edge E-C or A-C, I choose E-C; make C known. All vertices are now included in the minimum spanning tree, the task is completed

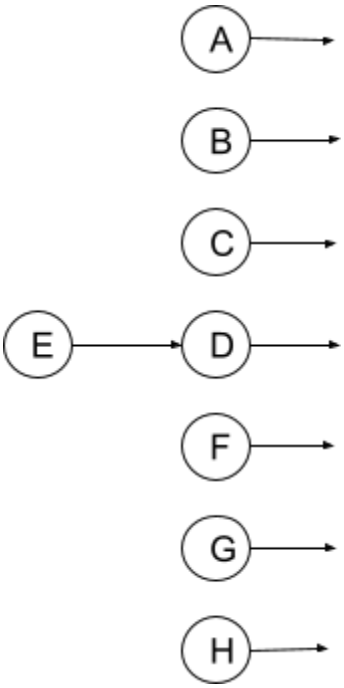


Question 3:

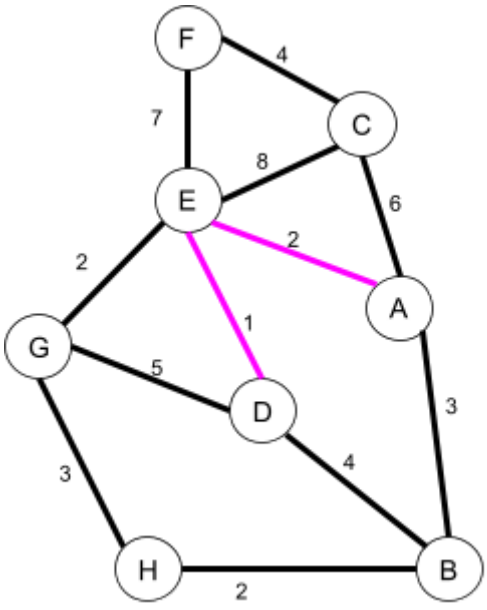
- Trace the Kruskal's minimum spanning tree algorithm on the graph in Figure 1.
  **REPRESENTATION: PINK COLORED EDGES INCLUDED IN THE PRIMS SPANNING TREE ALGORITHM**

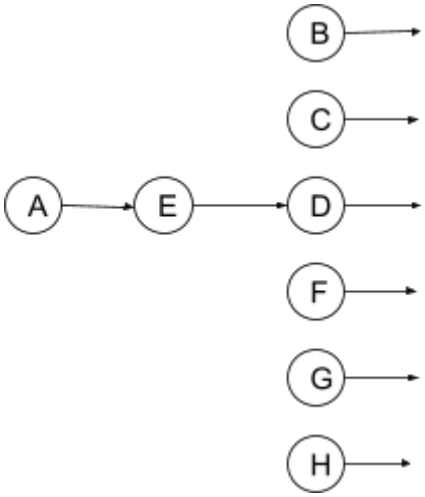
1st step: Pick the edge with the lowest weight. For this case pick the edge E-D. Union E and D then add the edge to the minimum spanning tree



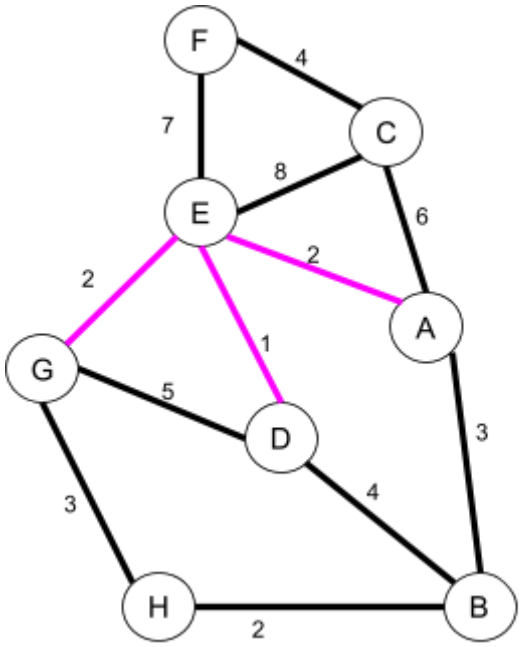



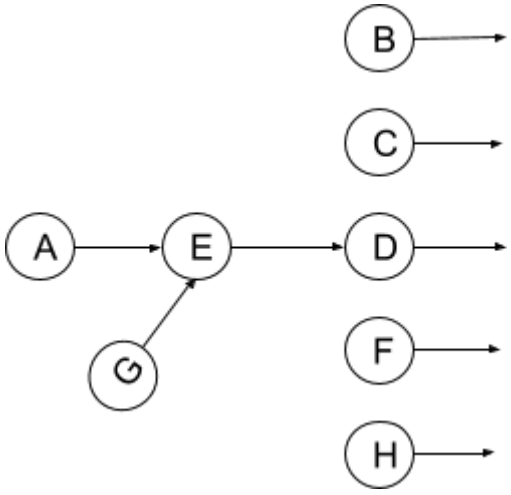

2nd step: Pick the edge with the lowest weight. For this case pick the edge E-A, E-G or H-B, I choose E-A. Union E and A then add the edge to the minimum spanning tree



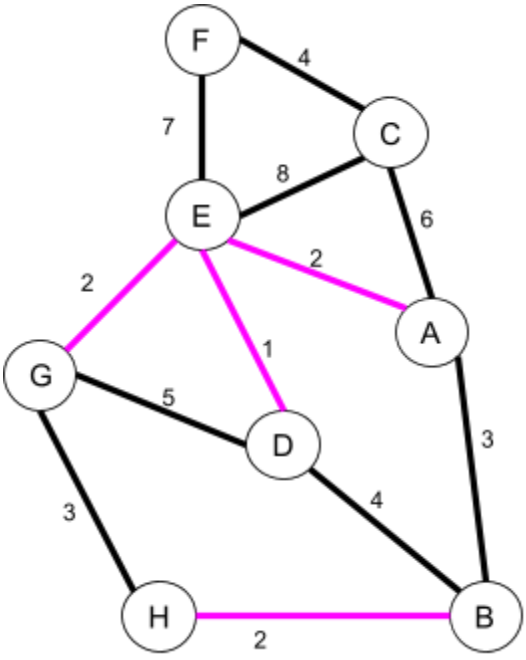



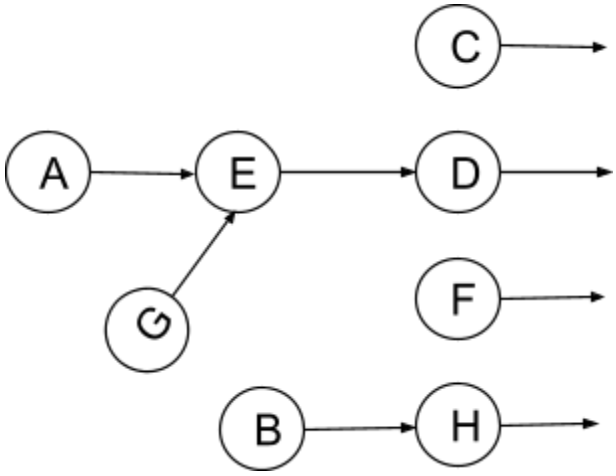
3rd step: Pick the edge with the lowest weight. For this case pick the edge E-G or H-B, I choose E-G. Union E and G then add the edge to the minimum spanning tree



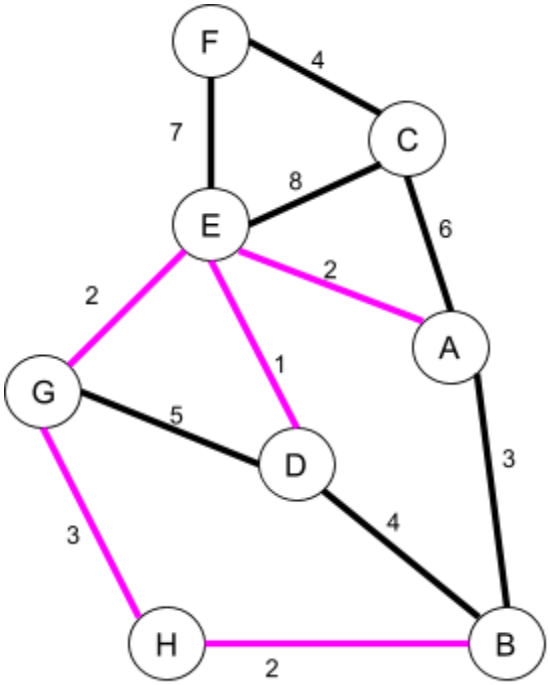



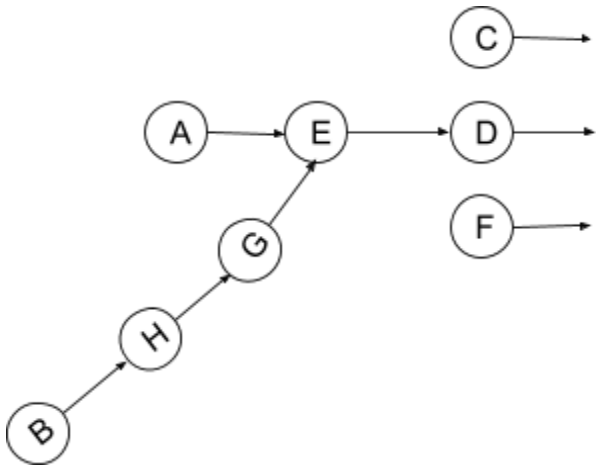

4th step: Pick the edge with the lowest weight. For this case pick the edge H-B. Union H and B then add the edge to the minimum spanning tree



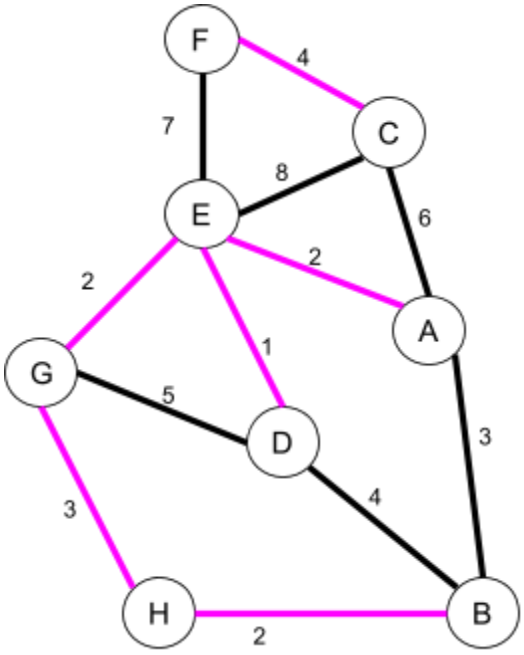



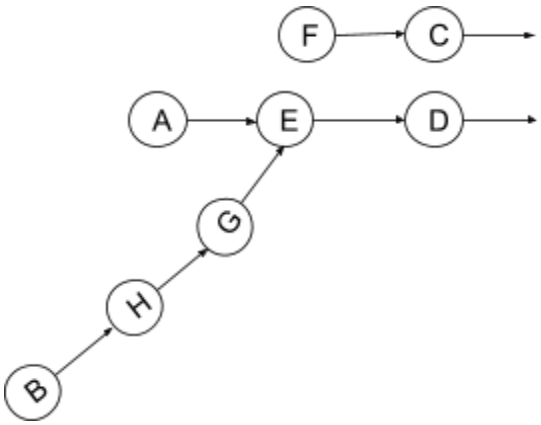
5th step: Pick the edge with the lowest weight. For this case pick the edge A-B or G-H I choose G-H. Union G and H then add the edge to the minimum spanning tree



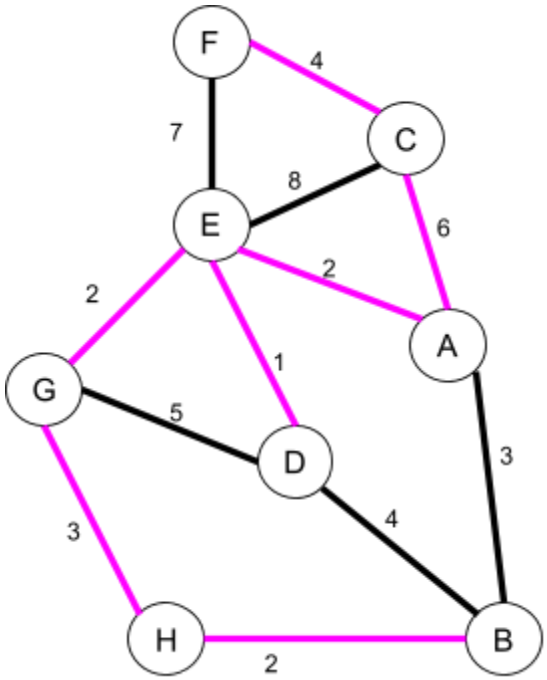



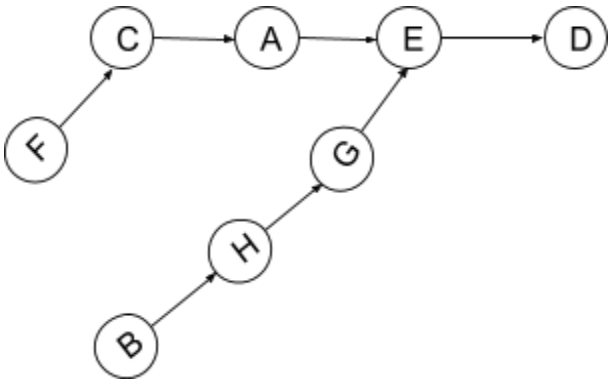

6th step:  Pick the edge with the lowest weight. For this case pick the edge F-C note that we cannot choose A-B because it will create a cycle. Union F and C then add the edge to the minimum spanning tree



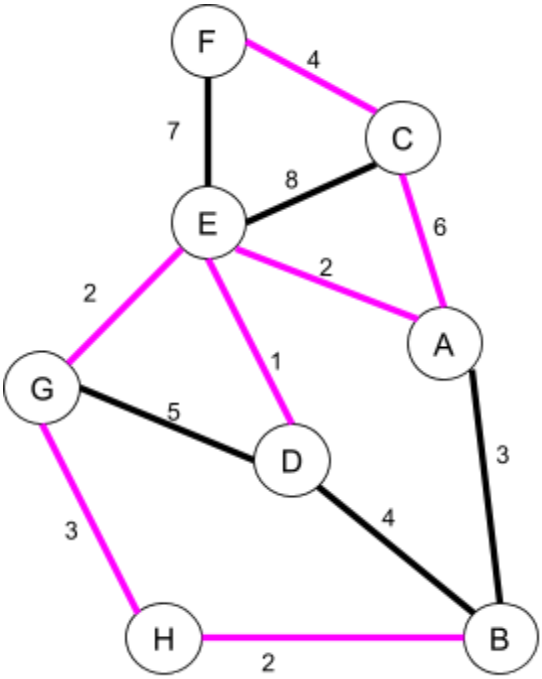



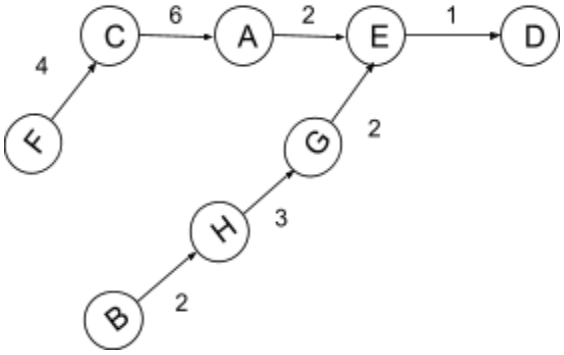
7th step:Pick the edge with the lowest weight. For this case pick the edge C-A note that we cannot choose D-B because it will create a cycle. Union C and A then add the edge to the minimum spanning tree




Note that we have spanned all of the vertices and we end up having one tree in our forest, the task is complete

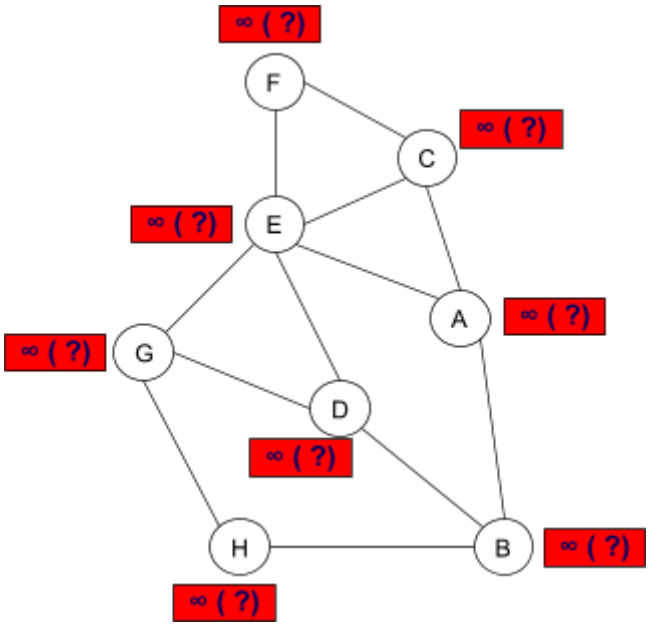


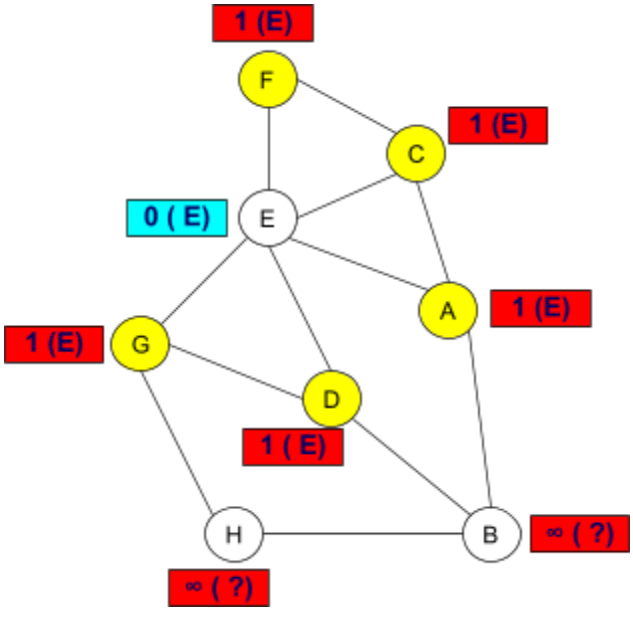
Question 4:

- Trace the breadth-first search traversal algorithm on the graph in Figure1 starting from vertex E. **REPRESENTATION: YELLOW NODES REPRESENTS THE VISITED VERTICES BLUE COLOR FOR KNOWN NODES RED FOR UNKNOWN NODES, X = DISTANCE Y = PREVIO** X (Y) **:**

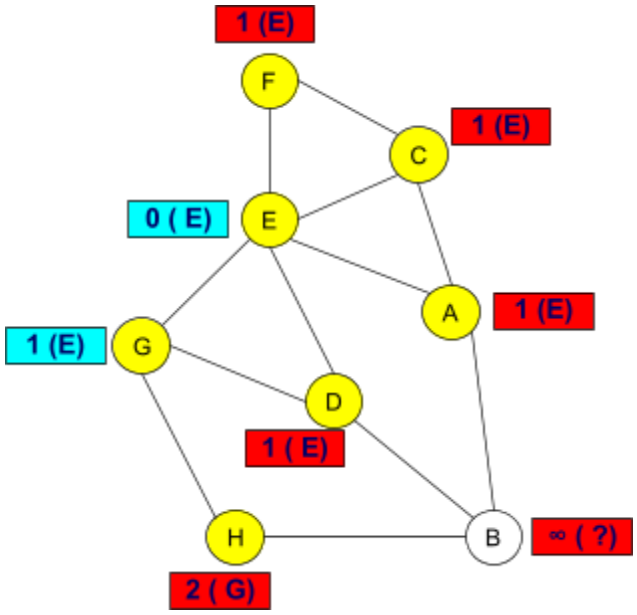
Initial State:



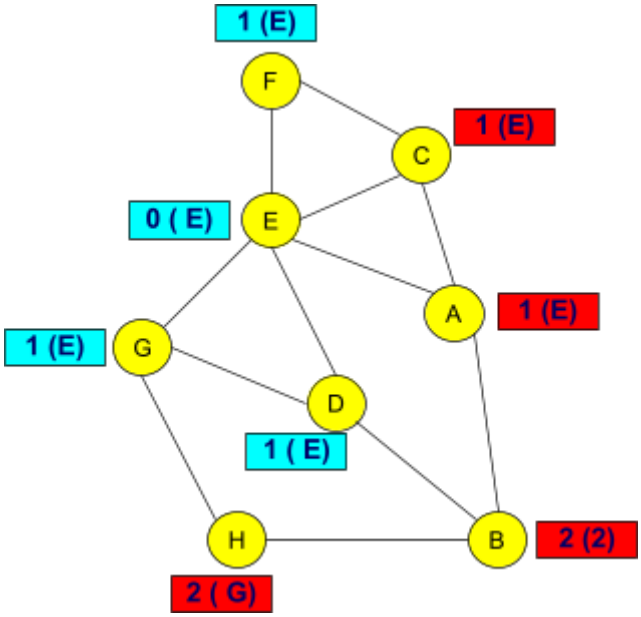

1st step: Select vertex E as the starting vertex, make it visited and update all unknown vertices adjacent to E. Note that in an unweighted graphs all weights are assumed to be 1, so we should update the distances accordingly



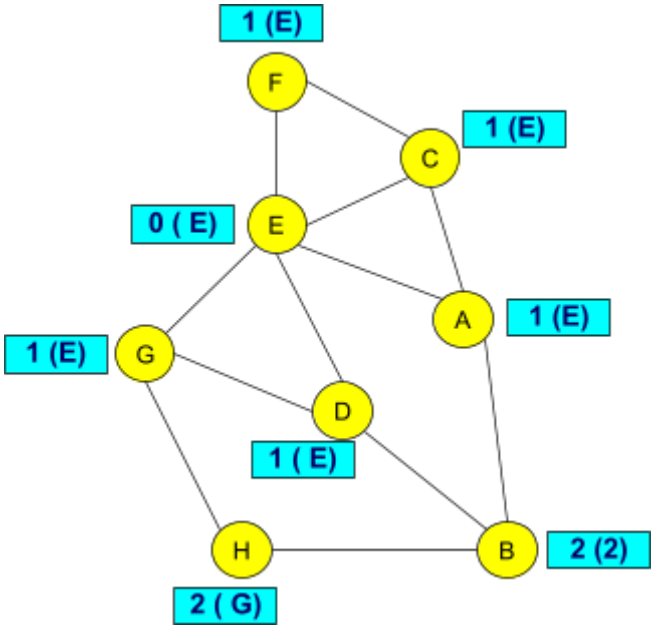
2nd step: Among the unknown adjacent vertices(F,G,C,A and D), select one and update the distances and known infos of the vertices that are adjacent to the selected one. I choose vertex G




3rd step: Among the unknown adjacent vertices (F,C,A and D), select one and update the distances and known infos of the vertices that are adjacent to the selected one. I choose vertex D



3rd step: As you can see all of the vertices are visited and each vertices' shortest distance values are known thus the task is complete.

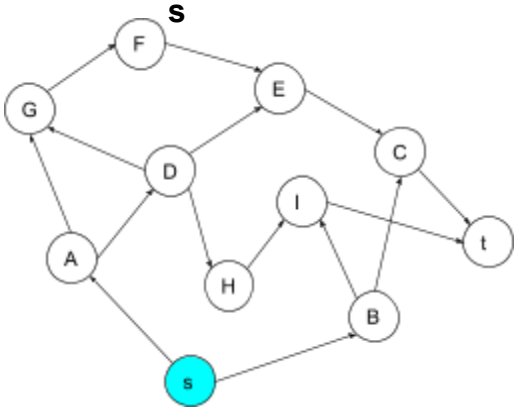


Question 5:

- Find a topological ordering of the graph in Figure 2.
  At each iteration we need to choose a vertex with in-degree 0, print it and remove it from the list. **NOTE THAT PICKED VERTICES WILL BE GIVEN BLUE COLOR**

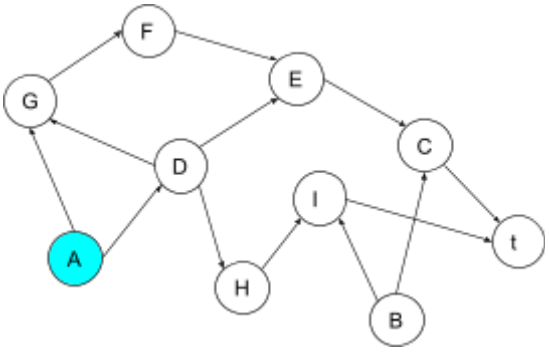
1st step:
get s, print s, remove s

**s**



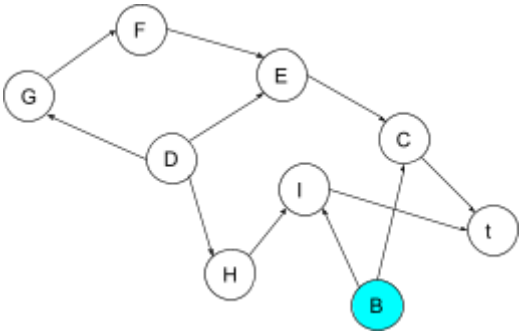

2nd step:
get A, print A, remove A

**s,A**



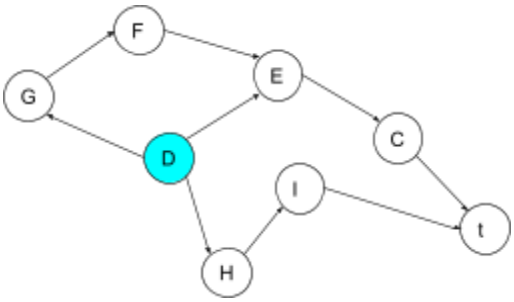

3rd step:
get B, print B, remove B

**s,A,B**



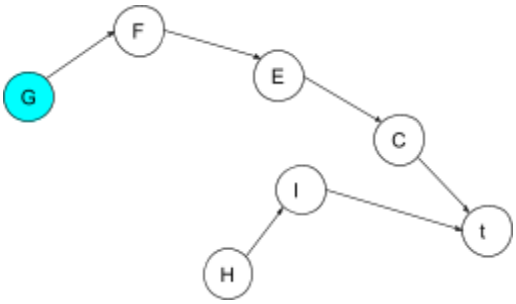

4th step:
get D, print D, remove D
remove F

**s,A,B,D**



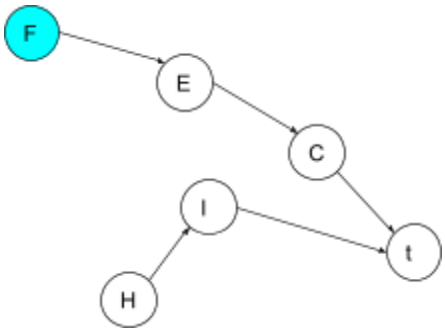

5th step:
get G, print G, remove G

**s,A,B,D,G**



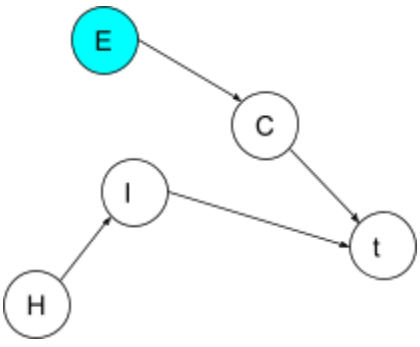

6th step:
get F, print F,

**s,A,B,D,G,F**



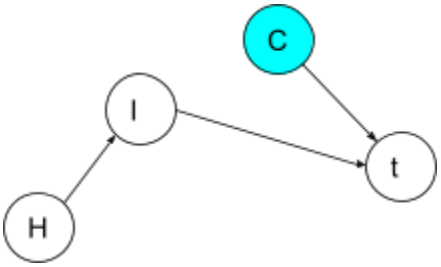

7th step:
get E, print E, remove E
remove H

**s,A,B,D,G,F**



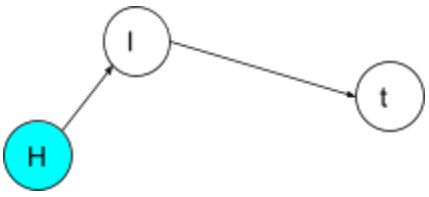

8th step:
get C, print C, remove C

**s,A,B,D,G,F,C**




9th step:
get H, print H,

**s,A,B,D,G,F,C,H**



10th step:
get I, print I, remove I
**s,A,B,D,G,F,C,H,I**


11th step:
get t, print t, remove t
**s,A,B,D,G,F,C,H,I,t**


**order would be: s, A, B, D, G, F, E, C, H, I, t**