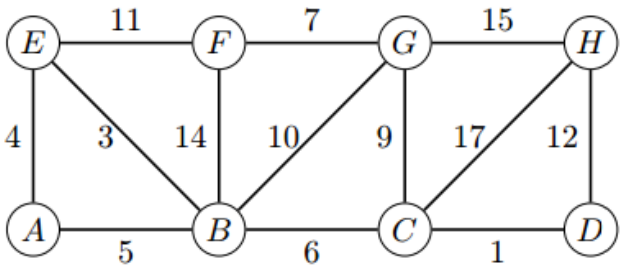
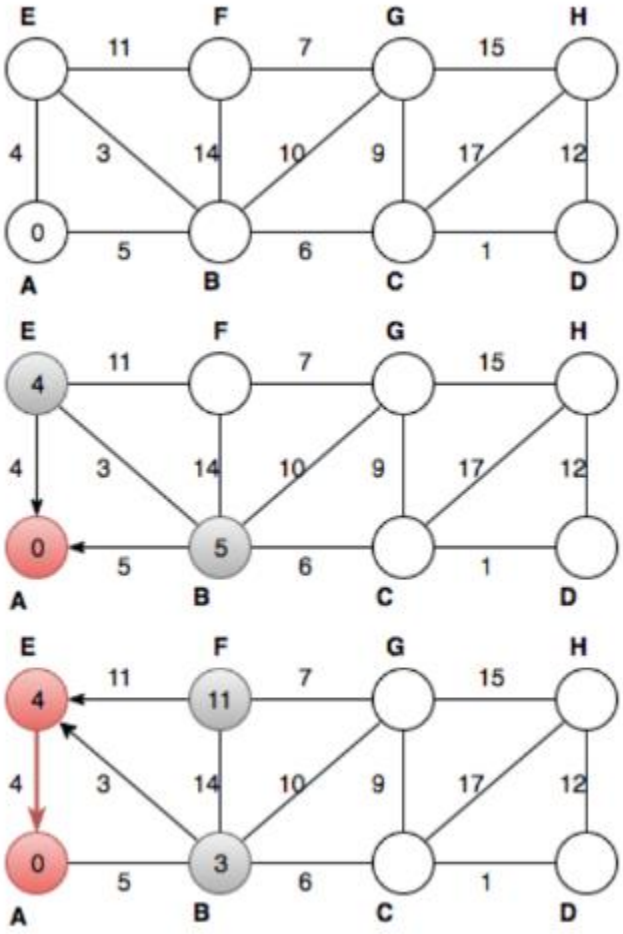
CS 325 - Homework Assignment 3 Solution

1. Consider the weighted graph below:

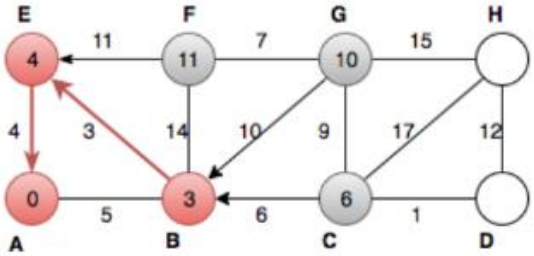


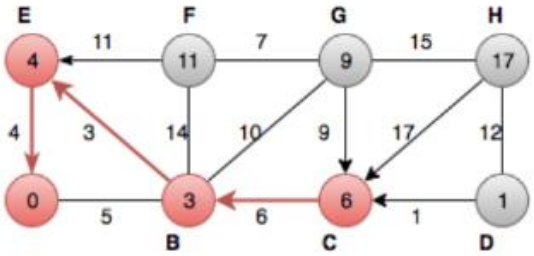
(a) Demonstrate Prim's algorithm starting from vertex A. Write the edges in the order they were added to the minimum spanning tree. **(3 points)**

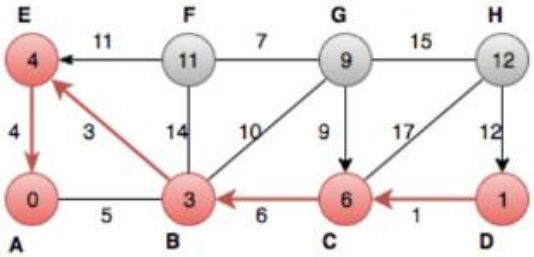
Solution $(AE, BE, BC, CD, CG, FG, DH)$

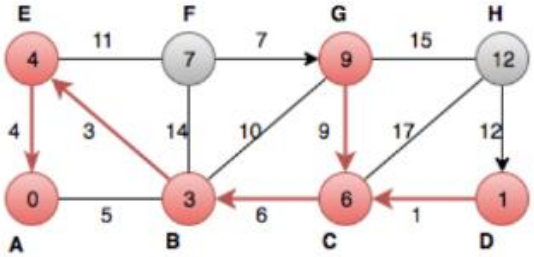


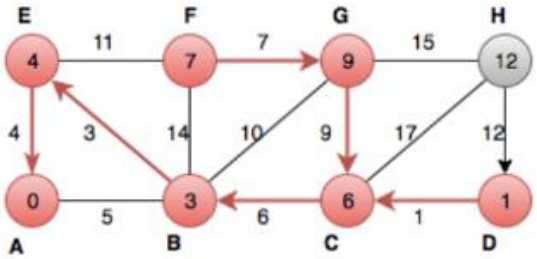
CS 325 - Homework Assignment 3 Solution

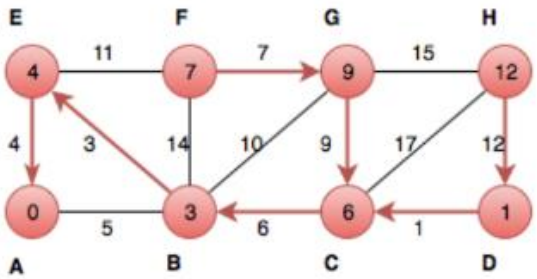












(b) Demonstrate Dijkstra's algorithm on the graph, using vertex A as the source. Write the vertices in the order which they are marked and compute all distances at each step. **(3 points)**

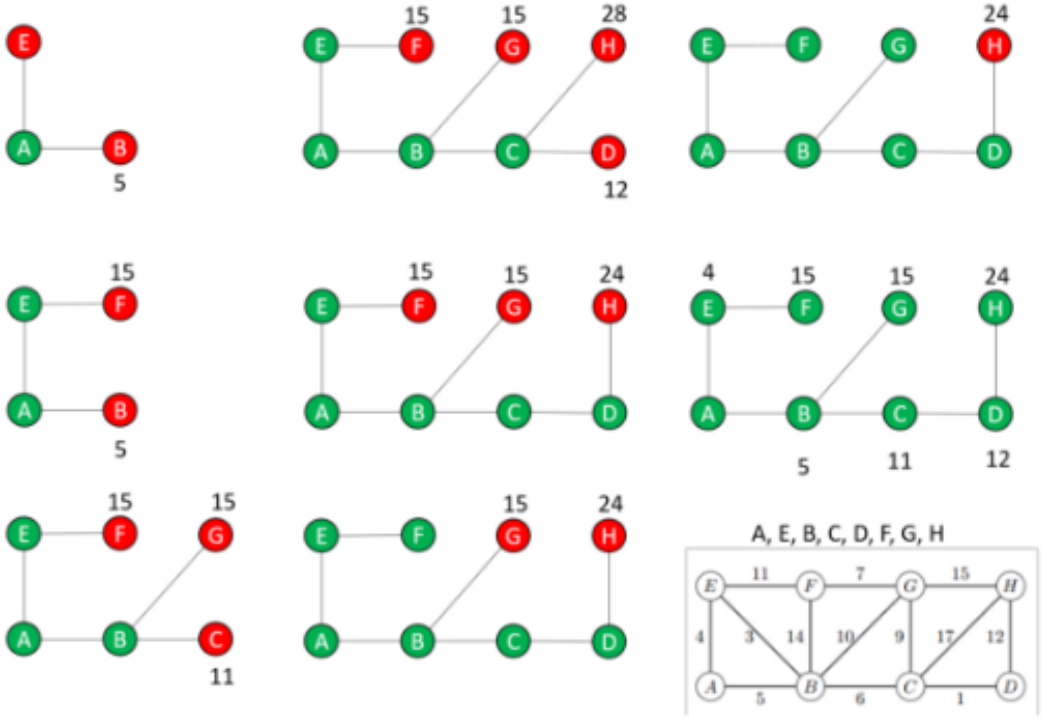
Solution Here is the vertices in the order which they are marked:

step	vertex
1	A
2	E
3	B
4	C
5	D
6	F
7	G
8	H

All distances at each step:

step	dist[A]	dist[B]	dist[C]	dist[D]	dist[E]	dist[F]	dist[G]	dist[H]
1	0	5	∞	∞	4	∞	∞	∞
2	0	5	∞	∞	4	15	∞	∞
3	0	5	11	∞	4	15	15	∞
4	0	5	11	12	4	15	15	28
5	0	5	11	12	4	15	15	24
6	0	5	11	12	4	15	15	24
7	0	5	11	12	4	15	15	24
8	0	5	11	12	4	15	15	24

CS 325 - Homework Assignment 3 Solution



	dv	pv
A	0	
B	5	A
C	11	B
D	12	C
E	4	A
F	15	E
G	15	B
H	24	D

2. **(3 points)** A Hamiltonian path in a graph G=(V,E) is a simple that includes every vertex in V. Design an algorithm to determine if a directed acyclic graph (DAG) G has a Hamiltonian path. Your algorithm should run in O(V+E). Provide a written description of your algorithm including why it works, pseudocode and an explanation of the running time.

Compute a topological sort and check if there is an edge between each consecutive pair of vertices in the topological order. If each consecutive pair of vertices are connected, then every vertex in the DAG is connected, which indicates a Hamiltonian path exists. Running time for the topological sort is O(V+E), running time for the next step is O(V) so total running time is O(V+E).

PSEUDOCODE:

HAS_HAM_PATH(G)

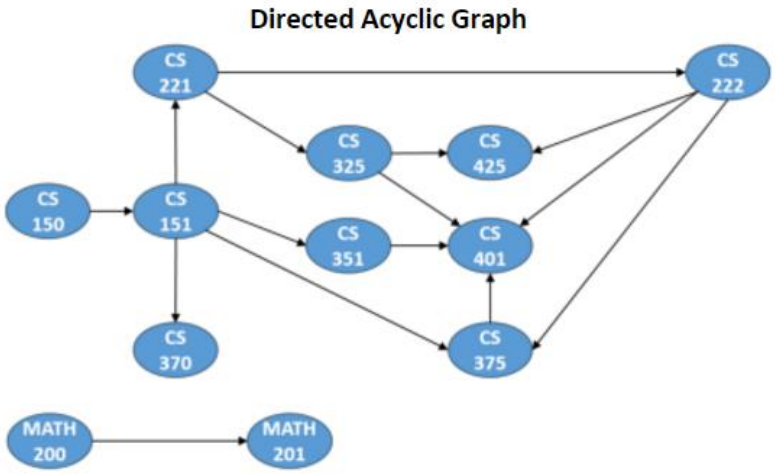
1. *Call DFS(G) to computer finishing time v.f for each vertex v.*
2. *As each vertex is finished, insert it into the front of the list*
3. *Iterate each through the lists of vertices in the list*
4. *If any pairs of consecutive vertices are not connected RETURN FALSE*
5. *After all pairs of vertices are examined RETURN TRUE*

CS 325 - Homework Assignment 3 Solution

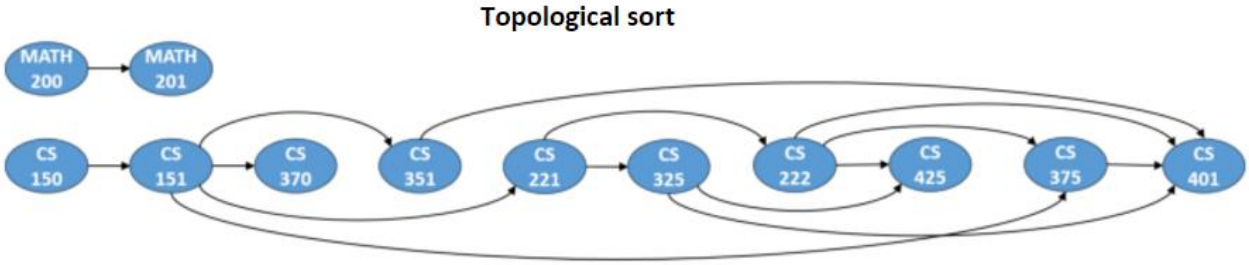
3. Below is a list of courses and prerequisites for a factious CS degree.

Course	Prerequisite
CS 150	None
CS 151	CS 150
CS 221	CS 151
CS 222	CS 221
CS 325	CS 221
CS 351	CS 151
CS 370	CS 151
CS 375	CS 151, CS 222
CS 401	CS 375, CS 351, CS 325, CS 222
CS 425	CS 325, CS 222
MATH 200	None
MATH 201	MATH 200

(a) Draw a directed acyclic graph (DAG) that represents the precedence among the courses. **(1 points)**



(b) Give a topological sort of the graph. **(2 points)**



Several correct variations

CS150 CS151 CS370 CS 351 CS221 CS325 CS222 CS425 CS375 CS 401 MATH 200 MATH 201 or

CS 325 - Homework Assignment 3 Solution

MATH 200 MATH 201 CS150 CS151 CS370 CS 351 CS221 CS325 CS222 CS425 CS375 CS 401

(c) Find an order in which all the classes can be taken. **(1 point)**

Again several correct variations.

1st Term: CS 150, MATH 200

2nd Term: CS 151, CS 201

3rd Term: CS 221, CS 351, CS 370

4th Term: CS 325, CS 222

5th Term: CS 425, CS 375

6th Term: CS 401

(d) Determine the length of the longest path in the DAG. How did you find it? What does this represent?

(2 points total)

Algorithm to find the longest path in a DAG O(E+V)

1) Topologically sort the graph

2) Initialize the distances associated with vertices in the graph to 0. That is d(v)=0 for all v in G.

3) Start with the first vertex v in the topological sort.

- For each u in Adj[v] set d(u) = max { d(u), d(v)+1 }

4) Repeat step 4 with the next vertex in the topological sorted order until all vertices have been examined.

The length of the longest path is 5 which corresponds to the courses (1 point)

CS150, CS151, CS 221, CS222, CS375, CS401.

The length of the path+1 = 5+1 = 6 represented the minimum number of terms required to complete all courses with the given prerequisites. (Note: this is often called the Critical Path)

4. Suppose you have an undirected graph G=(V,E) and you want to determine if you can assign two colors (blue and red) to the vertices such that adjacent vertices are different colors. This is the graph Two-Color problem. If the assignment of two colors is possible, then a 2-coloring is a function C: V -> {blue, red} such that $C(u) \neq C(v)$ for every edge $(u,v) \in E$. Note: a graph can have more than one 2-coloring. Give an O(V + E) algorithm to determine the 2-coloring of a graph if one exists or terminate with the message that the graph is not Two-Colorable. Assume that the input graph G=(V,E) is represented using adjacency lists.

(4 points total)

(a) Give a verbal description of the algorithm and provide detailed pseudocode. **(3 points)**

There are several variations. The basic idea is to select an uncolored vertex v and set the color=color1. Then examine all of the "children" u of v in Adj[v] and color them color2. If any of the children are the same color as there parent then a two coloring is impossible. You next examine the adjacency lists of the children and color them color1. This continues in a DFS or BFS fashion until all the vertices have been colored and all the edges have been examined for conflicts. This is also equivalent to determining if a graph is bipartite.

Let $G = (V, E)$ be the graph to which a 2-coloring is applied.
Let C be an array indexed as $C[0] \leftarrow color1$ and $C[1] \leftarrow color2$.

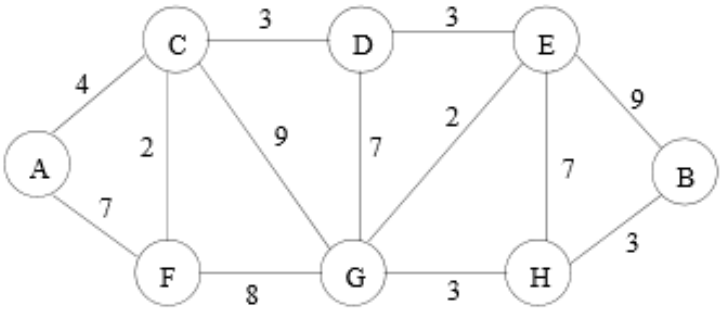
```
2-COLOR(G, C)
    for v ∈ V
        v.visited ← false
        v.color ← none
    for v ∈ V
        if v.visited == false
            TWO-COLOR-VISIT(v, 0, C)

2-COLOR-VISIT(v, i, C)
    v.visited ← true
    v.color ← C[i]
    Let N be the set of vertices adjacent to v.
    for n ∈ N
        if v.visited == true
            if v.color == n.color
                return false
        else
            2-COLOR-VISIT(v, 1 - i, C)
    return true
```

A return value of $true$ indicates that a 2-coloring was assigned successfully. Like DFS, the above algorithm runs in $O(V + E)$ time.

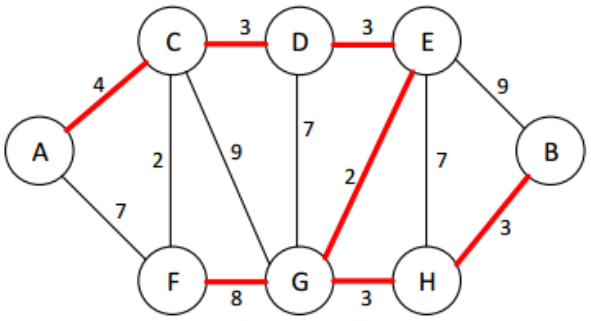
(b) Analyze the running time. O(E+V) **(1 point)**

5. A region contains a number of towns connected by roads. Each road is labeled by the average number of minutes required for a fire engine to travel to it. Each intersection is labeled with a circle. Suppose that you work for a city that has decided to place a fire station at location G. (While this problem is small, you want to devise a method to solve much larger problems).



(a) What algorithm would you recommend be used to find the fastest route from the fire station to each of the intersections? Demonstrate how it would work on the example above if the fire station is placed at G. Show the resulting routes. **(2 points)**

If the fire station is placed at G, using Dijkstra's algorithm, we find the following shortest paths to each intersection (the red thick edges):



Shortest Paths and Distances from G

Vertex	Distance	Path
A	12	G-E-D-C-A
B	6	G-H-B
C	8	G-E-D-C
D	5	G-E-D
E	2	G-E
F	8	G-F
G	0	G
H	3	G-H

(b) Suppose one "optimal" location (maybe instead of G) must be selected for the fire station such that it minimizes the distance to the farthest intersection. Devise an algorithm to solve this problem given an arbitrary road map. Analyze the time complexity of your algorithm when there are f possible locations for the fire station (which must be at one of the intersections) and r possible roads. (Again several possible answers). **(2 points)**

Algorithm Description:

To determine the optimal location for the fire station such that it minimizes the distance to the farthest intersection, one possible method is, assign EACH of the vertices/intersections of the graph as the source and calculate the shortest paths to every other intersection using Dijkstra's algorithm. Let L(f) be the farthest intersection from any source vertex/intersection, i.e. the longest of the shortest paths from that intersection. To determine the optimal location for the fire station, iterate through each of the candidate fire station location intersections and determine the overall minimum L(f). This is the optimal location for the fire station.

Pseudocode:

```
Firestation(G)
  minimum = ∞ // minimum of the farthest intersections
  location = NULL // optimal location of the fire station
  For f ∈ V(G)                                              Θ(V)
    distances = DijkstraSSSP(G, f) // distances to each other vertex from source f   Θ(V²)
    L = MAX(distances) // maximum of the shortest paths, furthest intersection distance   Θ(V)
    if L < minimum // if current farthest inters'n closer than prior min. farthest inters'n...
      location = f // update optimal location                                        Θ(1)
      minimum = L // update minimum farthest intersection
  return location                                                                   Θ(V³)
```

Running Time:

This algorithm is based around repeated use of Dijkstra's single-source shortest path algorithm. Dijkstra's algorithm has an asymptotic running time of $\Theta(V^2)$.

```
Dijkstra(G, w, s)
  InitializeSingleSource(G, s)                              Θ(V)
  S ← ∅
  Q ← V[G]
  while Q ≠ 0 do                                            Θ(V)
    u ← ExtractMin(Q)
    S ← S U {u}                                             Θ(V²)
    for v ∈ Adj[u] do                                       Θ(V)
      Relax(u,v,w)
```

The algorithm defined above executes Dijkstra's algorithm $\Theta(V)$ times in a loop, for each vertex, to find the shortest paths to all other vertices from that vertex. As such, the overall asymptotic running time of the algorithm to find the optimum location of the fire station is $\Theta(V^3)$. **In the nomenclature defined by the question, if there are *f* possible locations for the firestation and *r* roads, this algorithm runs in $\Theta(f^3)$ time.**

(c) **2 point** Location E is optimal for the fire station since it's longest shortest path is 10 from E to A. All other locations have at least one longer shortest path.

In the table below we see the lengths of the shortest distances from each intersection to another intersection. **E has the minimum max of 10**. Table not required.

	A	B	C	D	E	F	G	H	**max**
A	0	18	4	7	10	6	12	15	**18**
B	18	0	14	11	8	14	6	3	**18**
C	4	14	0	3	6	2	8	11	**14**
D	7	11	3	0	3	5	5	8	**11**
E	10	8	6	3	0	8	2	5	**10**
F	6	14	2	5	8	0	8	11	**14**
G	12	6	8	5	2	8	0	3	**12**
H	15	3	11	8	5	11	3	0	**15**

CS 325 - Homework Assignment 3 Solution

EXTRA CREDIT: 2 points for explanation , 1 point run time, 1 point solve problem above best locations are C and H. (4 points total possible)

(+ 3 points for $O(f^3)$ algorithm)

1) Compute as in part 6 b) keeping track of the minimum distance from each potential fire station location to all other intersections. This takes time $O(f^3)$, $O(f^2 \log f + fr)$ as discussed above. Results can be stored in an fxf matrix.

2) For each possible pair of placements of fire stations compare the distances to each other intersection, keeping track of the maximum length of the minimum path. There are $O(f^2)$ possible pairs and it takes O(f) for the comparisons to find the max min for that combination. The result is $O(f^3)$.

3) Combining the results from 1) and 2) we have $O(f^3) + O(f^3) = O(f^3)$

Another algorithm may give $O(f^4)$ running time. (+ 2points)

The optimal location for the two fire stations can be determined by using Dijkstras algorithm to calculate the shortest path lengths from each possible pair of locations to an intersection. For each pair you can run Dijkstra algorithm on each vertex and keep track of the shortest distances. Compare the distances of to the two vertices and keep the smallest in the shortest path. You need to record the maximum path among the shortest paths for that pair. Repeat with all pairs of vertices. Select the pair of locations with the smallest maximum distance to an intersection. Runtime depends on implementation probably $O(f^4)$.

Shortest Path Length to Farthest Intersection

	A	B	C	D	E	F	G	H
A	N/A	8	14	11	8	14	6	7
B	8	N/A	6	7	10	8	12	16
C	14	6	N/A	11	8	14	6	5
D	11	7	11	N/A	8	11	7	7
E	8	10	8	8	N/A	8	10	10
F	14	8	14	11	8	N/A	6	6
G	6	12	6	7	10	6	N/A	12
H	7	16	5	7	10	6	12	N/A

Best locations are H and C