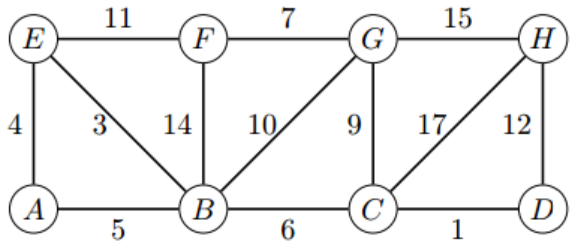
**CS 325 - Homework Assignment 3**

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.

AE, BE, BC, CD, CG, FG, DH

(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.

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

2. A Hamiltonian path in a graph G=(V,E) is a simple path 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:

Hamiltinian_path(G)

1. using DFS(G) to finishing time for each vertex v
2. insert vertex into the front of the list
3. iterate each through the list of vertices in the list
4. if there have any pair of vertices are not connected return False
5. if all pairs of vertices are pass return Ture

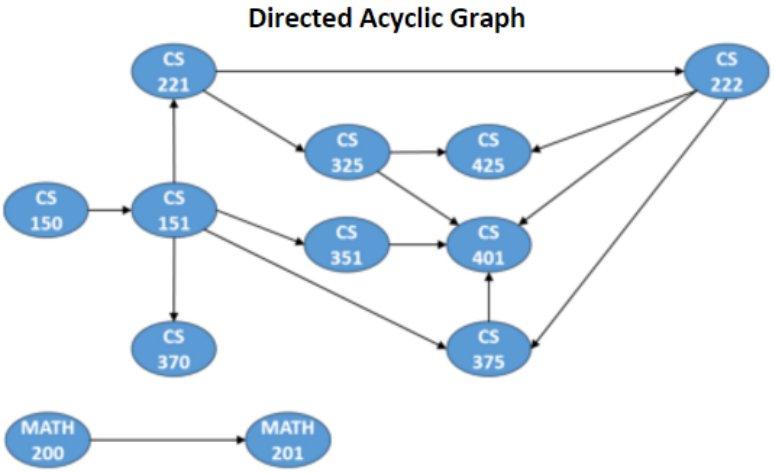
3. Below is a list of courses and prerequisites for a factious CS degree.

| Course | Prerequisite |
|---|---|

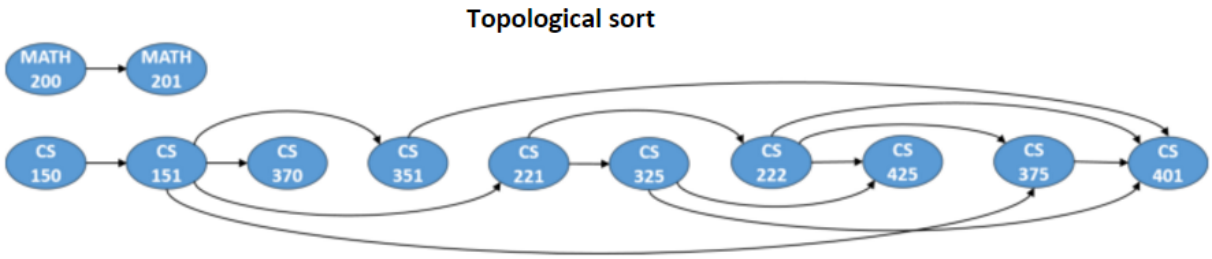
**CS 325 - Homework Assignment 3**

| | |
|---|---|
| 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.




(b) Give a topological sort of the graph.




(c) Find an order in which all the classes can be taken. You are allowed to take multiple courses at one time as long as there is no prerequisite conflict.

1st term: MATH 200, CS 150

2nd term: MATH 201, CS 151

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

4th term: CS 222, CS 325

5th term: CS 375, CS 425

6th term: CS 401

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

DAG(G)

1. using topologically to sort the graph
2. d(v) = 0 for all v in G
3. for each u in Adj[v]

   d(u) = max{d(u), d(v) + 1}

the longest path is CS150, CS151, CS 221, CS222, CS375, CS401.

This represent the minimum number of terms you should take to complete all the courses.

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.

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

For the problem, the basic idea is find an uncolored vertex and let this vertex to be color 1. And for other vertexes which connect to this vertex to be color 2. If there are two vertexes which are connect and have same color will be return False.

Pseudocode:

Let G = (V, E) be the graph to which a 2 color applied

Let C be an array indexed as C[0] <- color 1 and C[1] <- color 2

2_color(G, C)

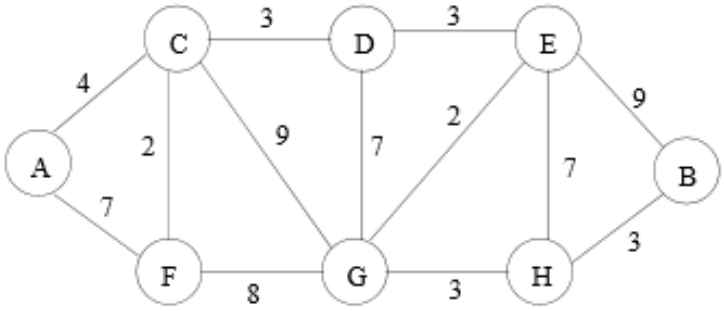
```
    for v belong to V
        v.visited <- False
        v.color <- None
    for v belong to V
        if v.visited == False
            two_color_visit(v, i, C)
2_color_visit(v, i, C)
    v.visit <- True
    V.color <- C[i]
    Let N be the set of vertices adjacent to v
    For n belong to N
        If v.visited == True
            If v.volor == n.color
                Return False
        Else
            2_color_visit(v, 1 – I, C)
    return true
```

(b) Analyze the running time.

For this algorithm, you should visit all the vertexes and edges. So for the vertex it take O(V) and for edge it take O(E), so the total is O(V) + O(E) = O(V + E)

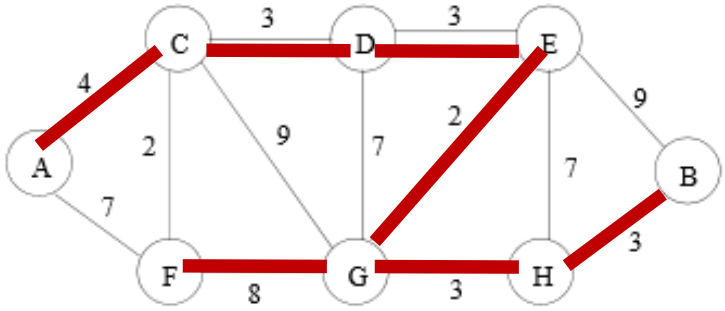
5. A region contains a number of towns connected by roads. Each road is labeled by the average number of minutes required for a fire engine to travel to it. Each intersection is labeled with a circle. Suppose that you work for a city that has decided to place a fire station at location G. (While this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

**CS 325 - Homework Assignment 3**




(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.

So if the fire station is locate at G, I will use Dijkstra's algorithm. We can find the shortest path form G to other vertexes.




G to A, the shortest path is G->E->D->C->A, and the distance is 12

G to B, the shortest path is G->H->B, and the distance is 6

G to C, the shortest path is G->E->D->C, and the distance is 8

G to D, the shortest path is G->E->D, and the distance is 5

G to E, the shortest path is G->E, and the distance is 2

G to F, the shortest path is G->F, and the distance is 8

G to G, the shortest path is G, and the distance is 0

G to H, the shortest path is G->G, and the distance is 3

(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.

Pseudocode:

```
Firestation(G)
    Minimum = inf
    Location = NULL
    For f belong to V(G)
        Distances = DijkstraSSSP(G, f)
        L = max(distances)
        If L < minimum
            Location = f
            minimum = L
    return location
```

For this algorithm, the total running time should be $O(f^3)$. In this algorithm, we use DijkstraSSSP. It will take $O(V^2)$ time. And for find the max distances, it will take O(V) time. So the total time is $O(V^2)*O(V) = O(V^3)$. So the running time for this algorithm is $O(f^3)$

(c) In the above graph what is the "optimal" location to place the fire station?

For what we want is to find a place that has shortest maxpath to each town.

| | A | B | C | D | E | F | G | H | Max distance |
|---|---|---|---|---|---|---|---|---|---|
| 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 |

So we can find that form E to other town, the max distance is 10. That means, if we put firestation at E, it will be close to each town.

**EXTRA CREDIT:** Now suppose you can build two fire stations. Where would you place them to minimize the farthest distance from an intersection to one of the fire stations? 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. Give the solution for the town in problem 5.

For this problem, the solution is similar as problem 5, just add one extra step. So in problem 5, we use DijkstraSSSP algorithm to find the shortest path from each vertex to others. In the part, we just need to combine all pairs of the vertexes. We can find the new shortest path from two vertexes to others. And compare each pair's max shortest path and find the minimum one that two vertexes are what we want.

So we can got a table like this:

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| A | X | 8 | 14 | 11 | 8 | 14 | 6 | 7 |
| B | 8 | X | 6 | 7 | 10 | 8 | 12 | 16 |
| C | 14 | 6 | X | 11 | 8 | 14 | 6 | ***5*** |
| D | 11 | 7 | 11 | X | 8 | 11 | 7 | 7 |
| E | 8 | 10 | 8 | 8 | X | 8 | 10 | 10 |
| F | 14 | 8 | 14 | 11 | 8 | X | 8 | 6 |
| G | 6 | 12 | 6 | 7 | 10 | 6 | X | 12 |
| H | 7 | 16 | ***5*** | 7 | 10 | 6 | 12 | X |

So we can see the best choose is C and H

So the running time should be $O(f^3)$, because the DijkstraSSSP take $O(f^2)$ and compare each pair of vertexes' max shortest path take O(f) time. So the total time is $O(f^2) * O(f) = O(f^3)$