Tyler Cope

Analysis of Algorithms Homework 3

Question 1

1. AE
2. EB
3. BC
4. CD
5. CG
6. GF
7. DH

b.

Step	A	B	C	D	E	F	G	H
0	**0**	∞	∞	∞	∞	∞	∞	∞
1	0	∞	∞	∞	**4**	∞	∞	∞
2	0	**5**	∞	∞	4	∞	∞	∞
3	0	5	**11**	∞	4	∞	∞	∞
4	0	5	11	**12**	4	∞	∞	∞
5	0	5	11	12	4	**15**	∞	∞
6	0	5	11	12	4	15	**15**	∞
7	0	5	11	12	4	15	15	**24**

So the order would be: A, E, B, C, D, F, G, H.

Question 2

Since the algorithm must run in O(V + E) time, that indicates that a topological sort could be used (as discussed in our group homework conversation). After the sort, we just need to look at every vertex and see if there is another vertex connected by an edge. Looking at each vertex will be O(V) where V is a vertex and O(V + E) is larger so that will dominate the overall running time.

```
def topSort(graph):
    myList = []
    color = u: gray for every vertex
    for v in graph:
        if color[v] == "gray":
            color[v] == “black”
            add color[u] to myList
    return myList
def checkHam(graph):
    graphSort = topSort(graph)
    for total number of elements (i) in graphSort:
            vertex = graphSort[i]
            vertex = graphsort[i + 1]
```

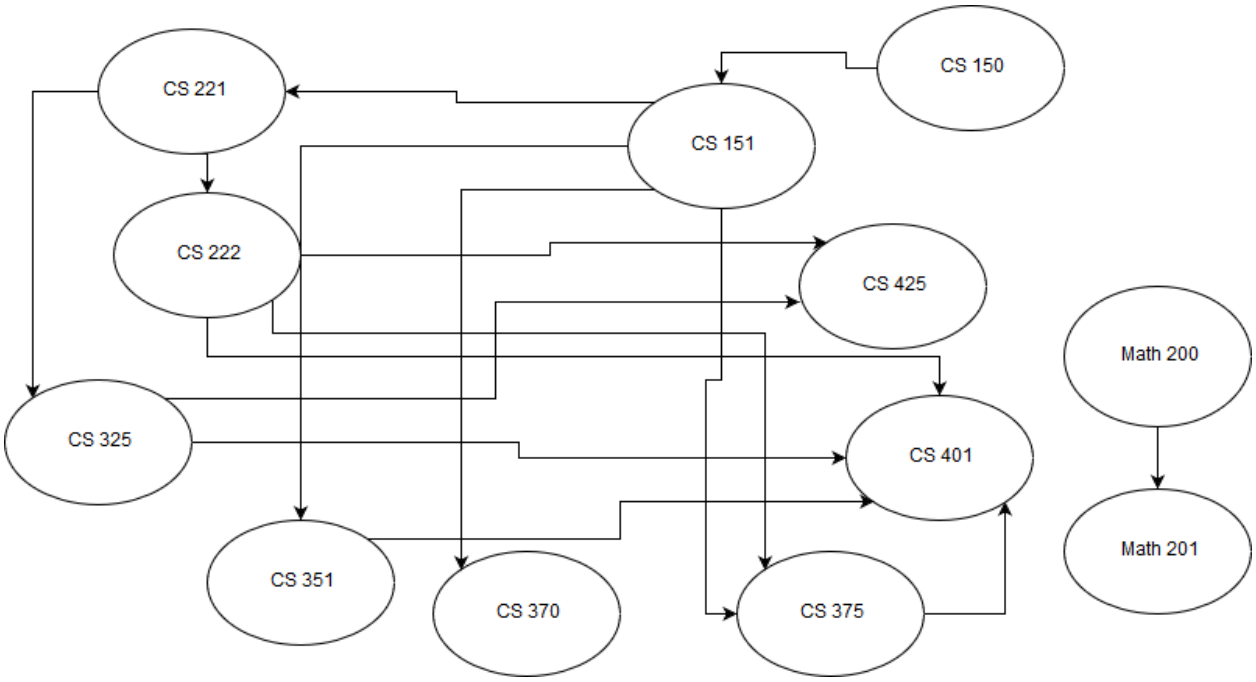
if nextNode not part of graph[node]:

return False (not found)

return True

Question 3

a.



b.

- Math 200
- Math 201
- CS 150
- CS 151
- CS 221
- CS 222

- CS 325
- CS 425
- CS 351
- CS 370
- CS 375
- CS 401

c.

Possible order of classes:

First semester

- Math 200
- CS 150

Second Semester

- Math 201
- CS 151

Third Semester

- CS 351
- CS 370
- CS 221

Fourth Semester

- CS 325
- CS 222

Fifth Semester

- CS 425
- CS 375

Sixth Semester

- CS 401

d.

The longest path in the DAG is 5, which I found by examining the graph and following the edges to determine the prerequisites. CS 150 to CS 151 to CS 221 to CS 222 to CS375 to CS 401. Since each course is a vertex and is representative of a class, and the edges show the prerequisites, the longest path represents the minimum number of semesters a student would need to complete the degree.

Question 4

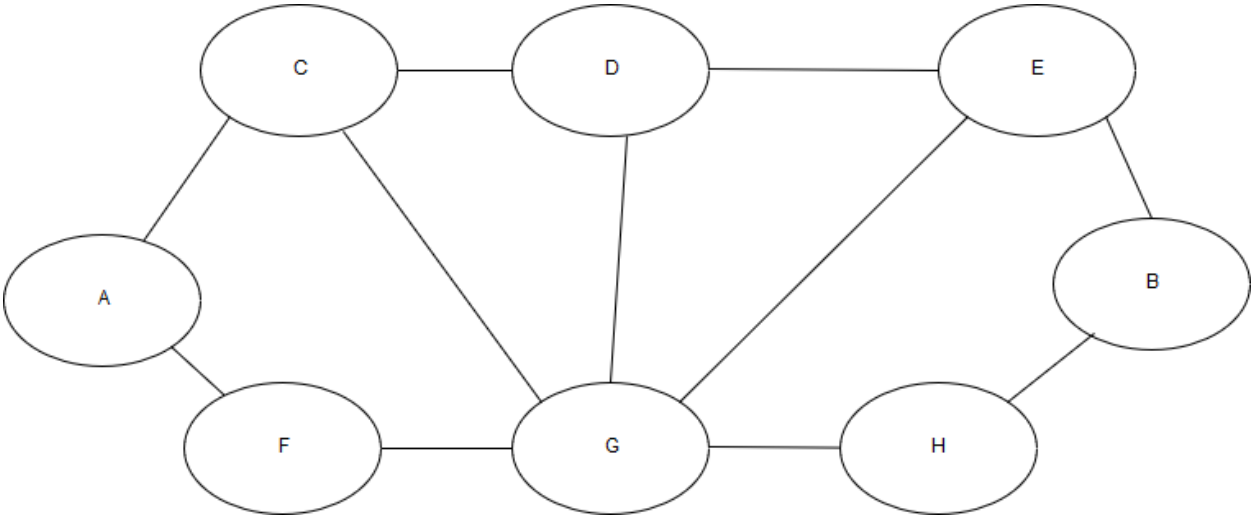
I'm going to assume that each vertex in the graph starts without a color. Under that assumption, the algorithm will go to each vertex and assign it a color that is opposite of the vertices adjacent to it. If a vertex is unable to be opposite colored, it will output the message that the graph is not colorable. Otherwise it will color the vertices, append them to a list, and return the list telling the user what color each vertex is.

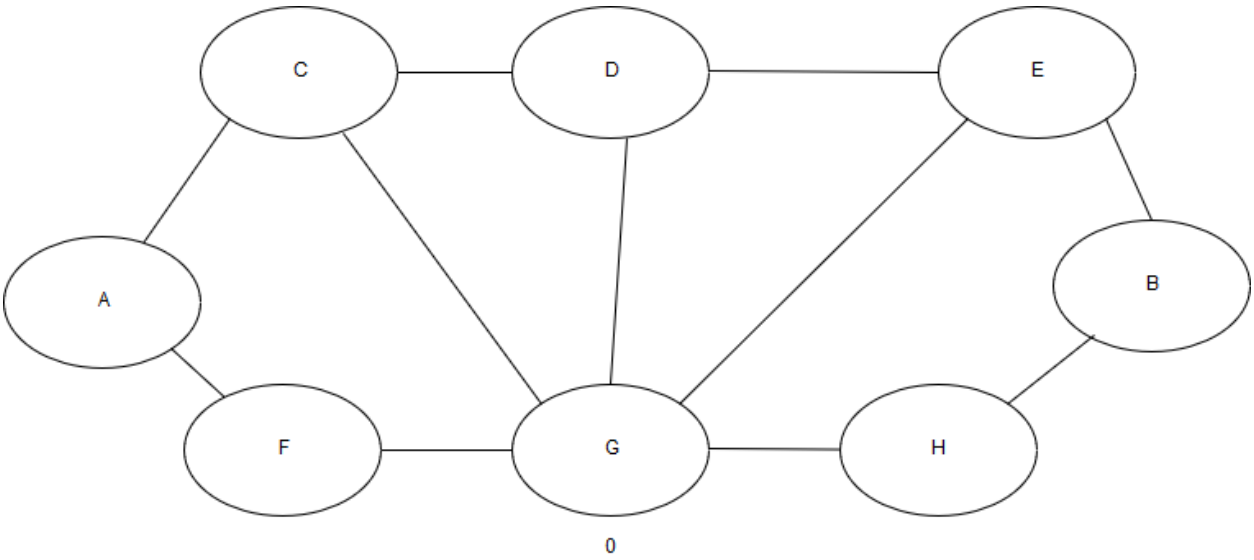
```
def vertexColor(graph g):

    holder = []

    while not every vertex has a color:

        for each vertex adjacent to vertex v:

            if vertex is color1 and other adjacent vertex is color2:

                return "The graph is not Two-Colorable"

            if vertex is color1:

                v is color2

            else if vertex is color2:

                v is color1

            else:

                v is color1

        holder.append(v);
```

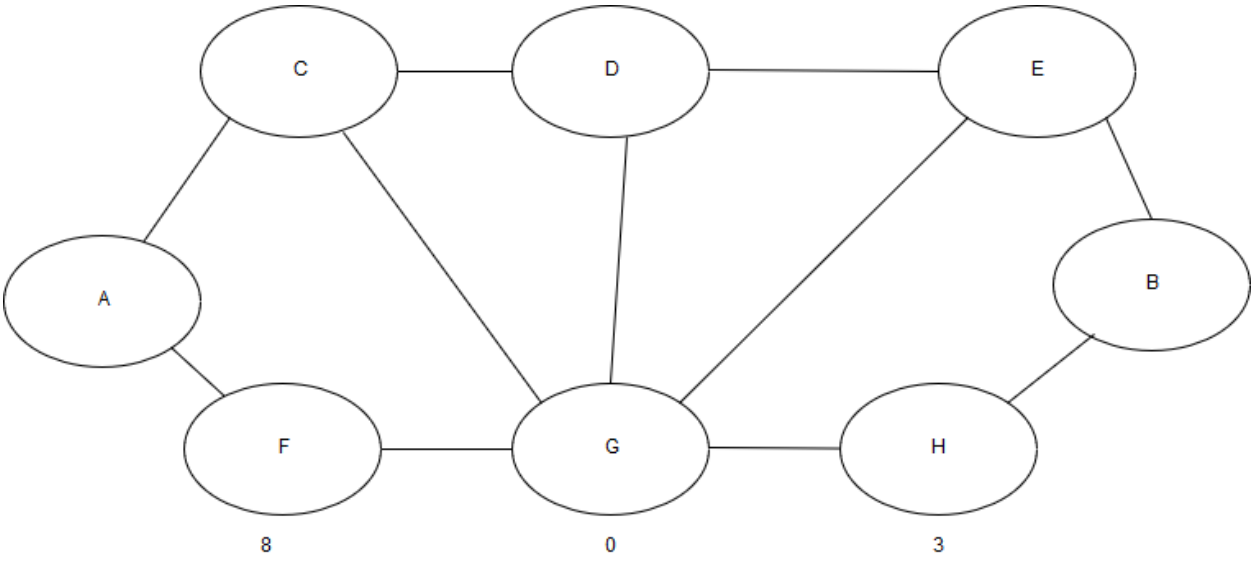
Coloring the vertices is a constant time operation and we only need to look at each edge and vertex a single time, so the algorithm runs in O(V + E) time.

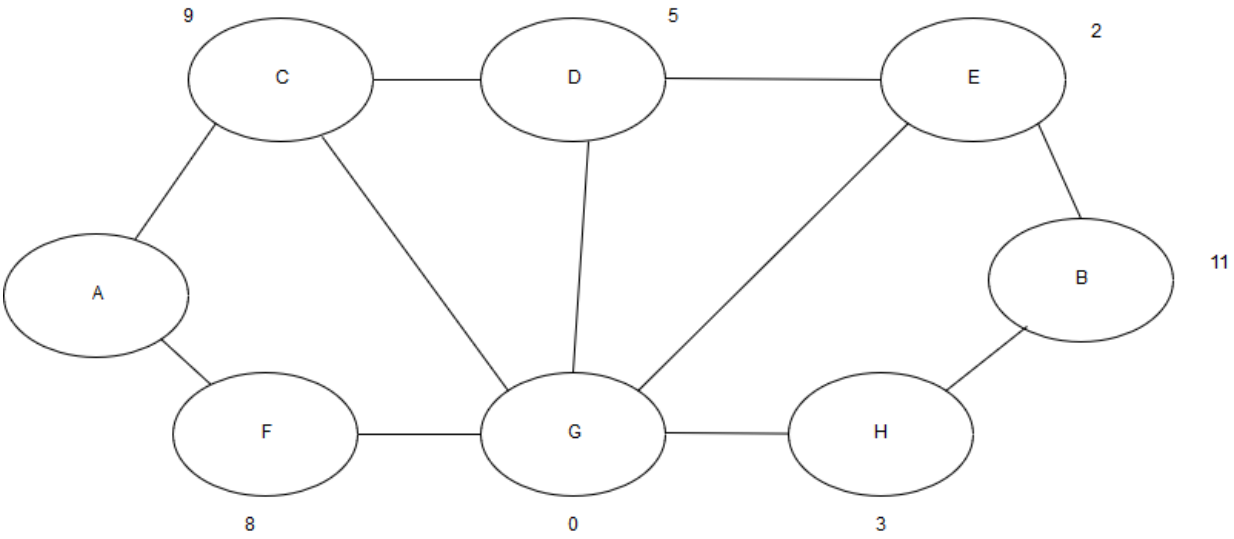
Question 5

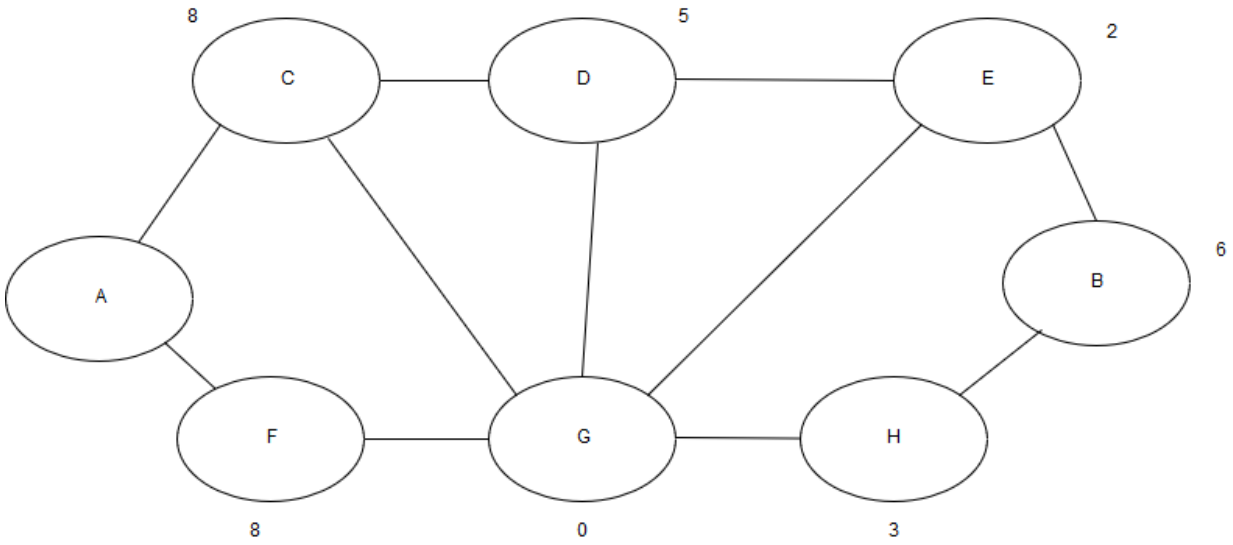
a. It would be best to use Dijkstra's algorithm to find the fastest routes to each intersection. I will show how the algorithm would work if it were placed at G.

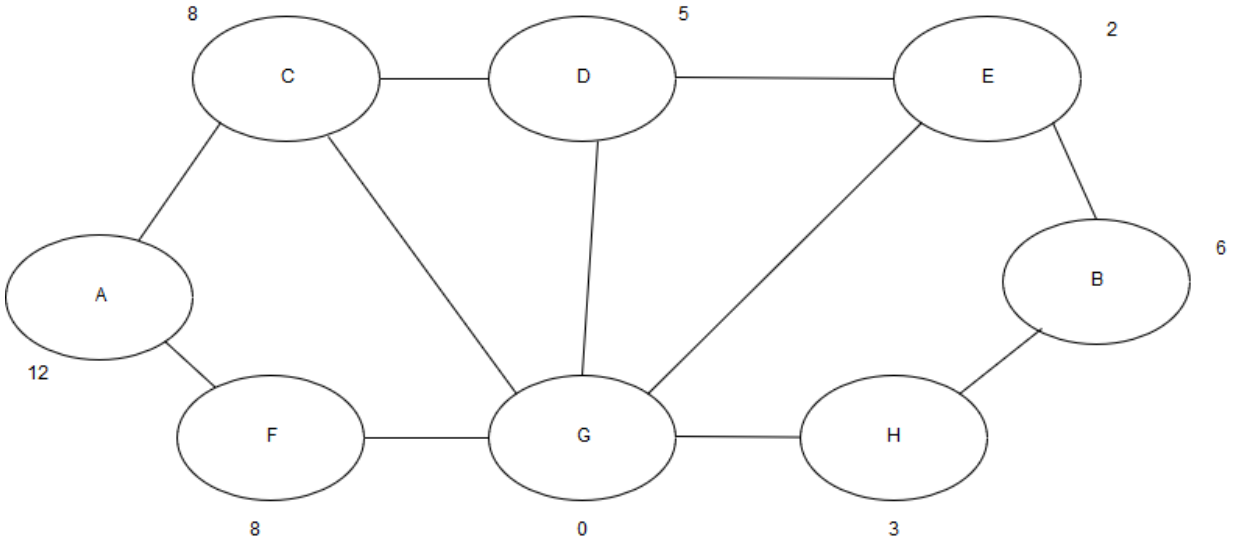





C
D
E
A
B
F
G
H
8
0
3


9
5
2
C
D
E
A
B
11
F
G
H
8
0
3





b.

One possible algorithm would be to use Dijkstra's Algorithm at every vertex and then return the vertex that has the smallest max distance. The runtime of Dijkstra's Algorithm is O(E* logV) where E is the number of edges and V is the number of vertices. However, since it will be run on every vertex, n, the running time would be O(n * E * logV).

c.

The optimal location to place the fire station is at vertex E. The max distance is from E to A, which is 10. By placing the fire station here, it ensures the quickest routes to each vertex from the fire station.

Extra Credit

I think the optimal locations for two stations would be at vertex H and vertex C. From vertex H you can easily get to B, G, and H (of course). At C, there are very short routes to A, E, D, F, and of course, C. The max distance would be 6 from C to E. For a general algorithm to solve with any arbitrary road map, place each firehouse spot in a list along with the potential places that would need to be reached by the fire service. Then, loop over each spot from the first location, compared with the second location using r roads. This would be $O(f^3)$ because you need to three loops: one to get the distance to every location from the first spot, one to get the distance to every location from the second spot, and a third loop to return whatever the minimum distance is for each location.