Benjamin Tate
5/13/2017
CS 325 -- Section 400
Homework #3

1.

a. Edges in order added to MST:
(A,E), (E,B), (B,C), (C,D), (C,G), (G,F), (D,H)

b. Vertices in order: **A, E, B, C, D, F, G, H**
Steps:

i.

	S	**d_v**	**p_v**
A	T	0	--
B	F	5	A
C	F	∞	--
D	F	∞	--
E	F	4	A
F	F	∞	--
G	F	∞	--
H	F	∞	--

ii.

	S	**d_v**	**p_v**
A	T	0	--
B	F	5 ~~7~~	A ~~E~~
C	F	∞	--
D	F	∞	--
E	T	4	A
F	F	15	E
G	F	∞	--
H	F	∞	--

iii.

	S	**d_v**	**p_v**
A	T	0	--
B	T	5	A
C	F	11	B

D	F	∞	--
E	T	4	A
F	F	15 ~~19~~	E ~~B~~
G	F	15	B
H	F	∞	--

iv.

	S	**d_v**	**p_v**
A	T	0	--
B	T	5	A
C	T	11	B
D	F	12	C
E	T	4	A
F	F	15	E
G	F	15 ~~20~~	B ~~C~~
H	F	28	C

v.

	S	**d_v**	**p_v**
A	T	0	--
B	T	5	A
C	T	11	B
D	T	12	C
E	T	4	A
F	F	15	E
G	F	15	B
H	F	~~28~~ 24	~~C~~ D

vi.

	S	**d_v**	**p_v**
A	T	0	--
B	T	5	A
C	T	11	B
D	T	12	C

E	T	4	A
F	T	15	E
G	F	15 ~~22~~	B ~~F~~
H	F	24	D

vii.

	S	$\mathbf{d_v}$	$\mathbf{p_v}$
A	T	0	--
B	T	5	A
C	T	11	B
D	T	12	C
E	T	4	A
F	T	15	E
G	T	15	B
H	F	24 ~~30~~	D ~~G~~

viii.

	S	$\mathbf{d_v}$	$\mathbf{p_v}$
A	T	0	--
B	T	5	A
C	T	11	B
D	T	12	C
E	T	4	A
F	T	15	E
G	T	15	B
H	T	24	D

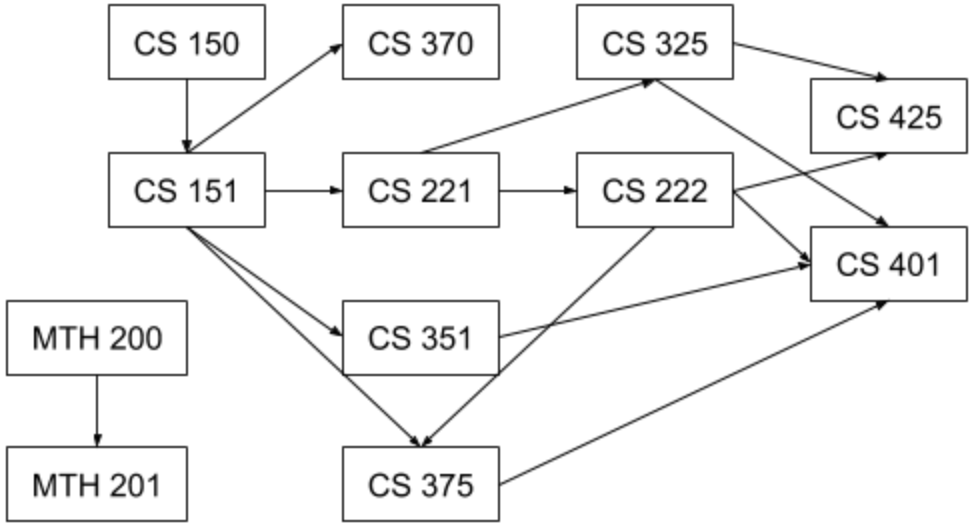
2. A simple solution for an algorithm that detects whether or not a Hamiltonian path exists in a DAG G would be to first topologically sort the graph, and then check that each pair of consecutive vertices in the sorted graph have an edge connecting them. This works because Hamiltonian paths cannot skip backward or forward, so a graph that has been topologically sorted would always have an edge connecting a vertex to the next vertex, if a Hamiltonian path exists. The topological sort is an important first step, because the graph must be made linear for this algorithm to work. This is okay though, since every DAG can be topologically sorted.

   Pseudocode:

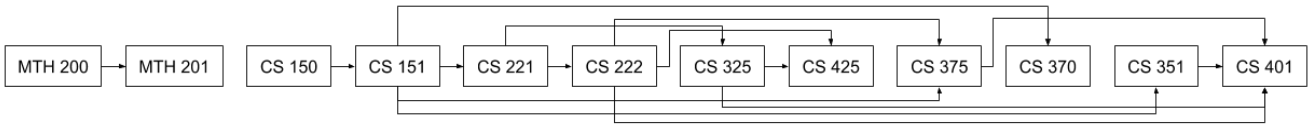
```
Function Hamiltonian(G=(V,E)): //For directed acyclic graph, G
        Bool hamPath = true
        V' = TopologicalSort(G) //To get linear order of vertices
        For vertex in V':
                If no edge from vertex to vertex.nextConsecutive:
                        hamPath = False
        Return hamPath
```

3.

a.



b.



c.

Term #	Classes
1	MTH 200, CS 150
2	MTH 201, CS 151
3	CS 221, 351, 370
4	CS 222, 325
5	CS 375, 425
6	CS 401

d. The length of the longest path in the DAG, which I found by observing the DAG (but which could also be found using a reversed version of an algorithm such as Dijkstra's Algorithm), is 5 edges. This represents the minimum number of terms

that it would take to earn this CS degree, minus 1, because this path is the longest route of prerequisites to the final course in the topologically sorted graph.

4.

a. An algorithm that determines whether an undirected graph is two-colorable, would need to start at a particular vertex, pick a color for it, then color its neighbors with the other color, and their neighbors with the first color, and so on until every vertex has been visited once and colored. Then, it would loop through all of the edges in the graph and determine if any edge has the same color on both ends. If so, then the graph is not two-colorable.

```
Psuedocode:
Function Color(N=(V',E'), color):
        For vertex in V':
                If vertex.color != red AND vertex.color != blue:
                        If color == red:
                                Vertex.color = blue
                        Else:
                                Vertex.color = red
                Color(vertex's unused neighbors, vertex.color)
                //unused determined by whether neighbor is already colored

Function TwoColor(G=(V,E)): //For an undirected, connected graph G
        twoColorable = True
        S = any vertex
        S.color = red
        Color(S's neighbors, red)
        For edge in E:
                If edge[1].color == edge[2].color:
                        twoColorable = False
        Return twoColorable
```

b. Since the above algorithm loops through each vertex once and each edge once and only once, the runtime is O(V+E)

5.

a. Dijkstra's Algorithm would be a good choice for finding the fastest route from the fire station to each of the intersections, since it is an efficient algorithm for finding shortest paths on graphs.

Demonstration (completed vertices in blue, vertices being updated in **bold**):
A(∞),B(∞),C(∞),D(∞),E(∞),F(∞),G(0min),H(∞) →
A(∞),B(∞),**C(9),D(7),E(2),F(8),**G(0),**H(3)** → Path = G
A(∞),**B(11),**C(9),**D(5),**E(2),F(8),G(0),**H(3)** → Path = G,E
A(∞),**B(6),**C(9),D(5),E(2),F(8),G(0),H(3) → Path = G,H
A(∞),B(6),**C(8)**,D(5),E(2),F(8),G(0),H(3) → Path = G,E,D
A(∞),B(6),C(8),D(5),E(2),F(8),G(0),H(3) → Path = G,H,B
A(15),B(6),C(8),D(5),E(2),F(8),G(0),H(3) → Path = G,F
A(12),B(6),C(8),D(5),E(2),F(8),G(0),H(3) → Path = G,E,D,C
A(12),B(6),C(8),D(5),E(2),F(8),G(0),H(3) → Path = G,E,D,C,A

b. The simplest (though definitely not the most efficient) solution would be to make an algorithm that simply runs Dijkstra's Algorithm on each vertex, keeping a running max distance for each, and then return the starting vertex with the lowest max distance associated with it. So Dijkstra's would be run V times, and then findMax would be called once at the end. Since Dijkstra's is $O(V^2)$ time, and findMax is O(V) time, the final complexity of the optimal start-point algorithm when there are f possible locations is $O(f^3)$.
c. The optimal location for the fire station is at intersection E, because the maximum travel time from E to any other intersection is 10 minutes, which is the lowest max travel time of any of the 8 intersections. I found this using the algorithm described in part b.