1)a) (AE, BE, BC, CD, CG, F G, DH)

b)

|  |  |
| --- | --- |
| Step | Vertex |
| 1 | A |
| 2 | E |
| 3 | B |
| 4 | C |
| 5 | D |
| 6 | F |
| 7 | G |
| 8 | H |

2) The algorithm is as follows: Sort the vertices per a topological ordering graph. Then go through this ordering and check if the graph has an edge between every consecutive pair of vertices.

Psesucode:

function hamilPath(G=(V,E))

nodes ordering(G)

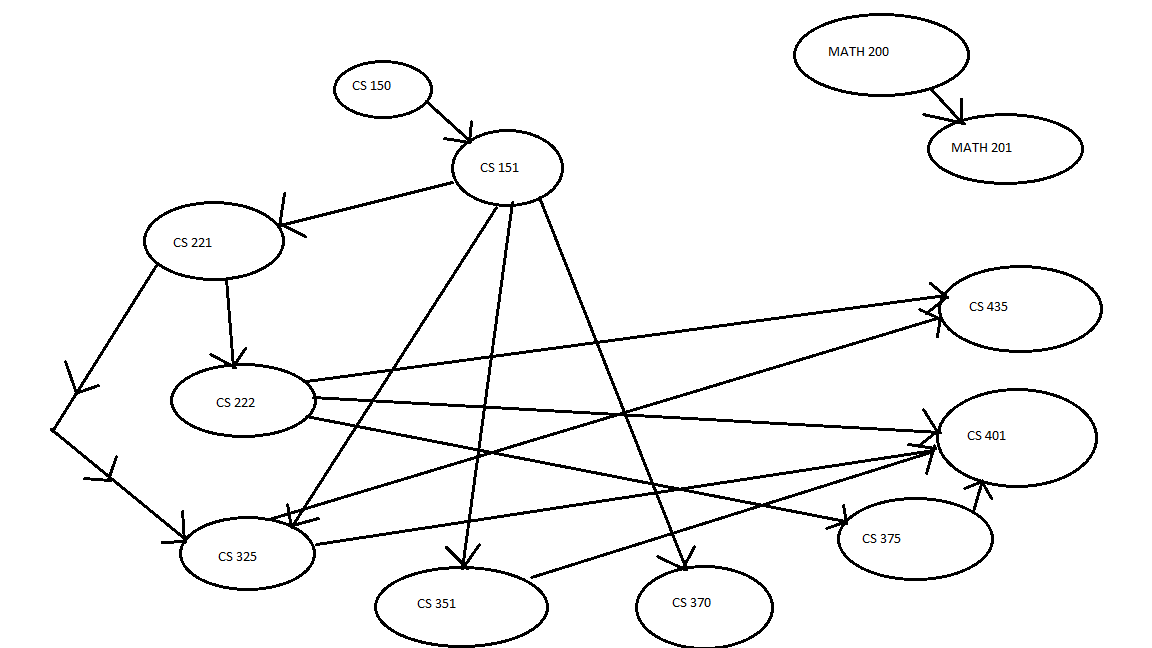
for i=1 …. Length(nodes)-1

if(nodes[i],nodes[i+1]) //not in E

return false

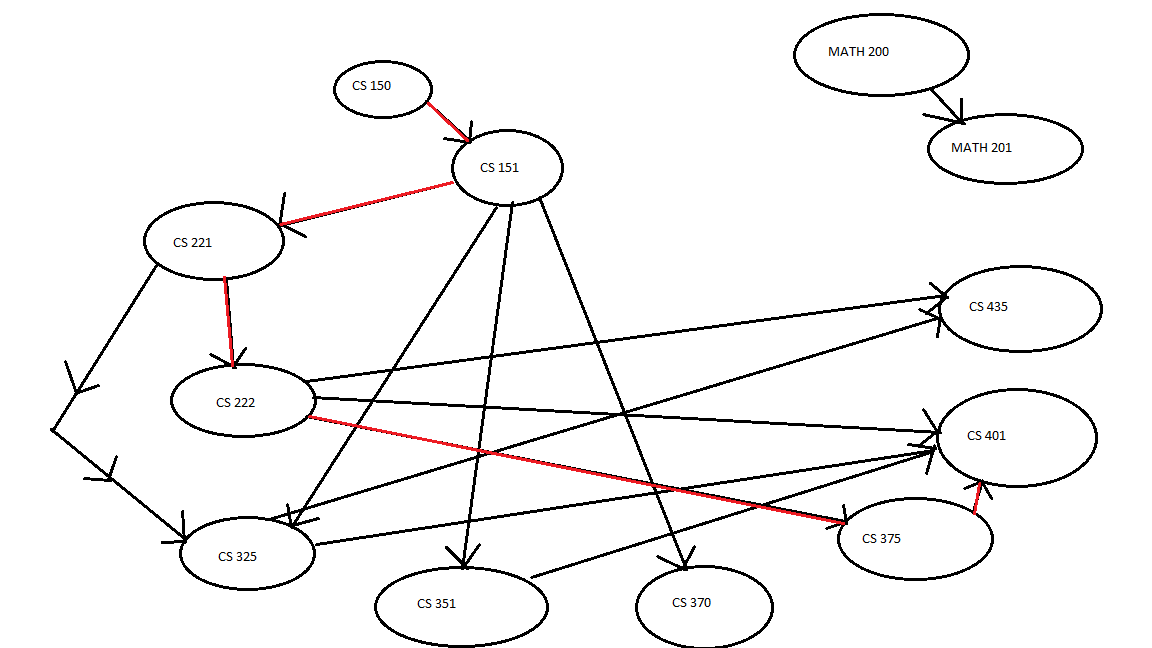
return true

The algorithm has a topological ordering (as seen above, which is linear, followed by a single linear loop. Therefore, the algorithm will work because it is linear. Run time for this is O(V+E). Since we run topological sort twice, runnning time is |V| + |E| + |V| + |E|. Drop the constants from |2V|+|2E| and its still O(|V| + |E|).

3)a) 

b) Topological sort gives us the ordered list in which all prerequisites of a given node should occur seen below: MATH200, MATH201, CS150, CS151, CS221, CS222, CS325, CS435, CS351, CS370, CS375, CS401

c) Again the topological sort is the answer. If we perform topological sort on the dependency graph we got a list which tells us a order which can be used to schedule the class.

d) The Longest path in this graph is marked below in red: Length = 5

4)

a)

2-colorable graph is the graph in which all vertices can be colored with two colors such that no two adjacent vertices have the same color. BFS algorithm can be used to check a graph that is -colorable or not. First color source vertex with first color. Then color neighbors of first vertex with the second color, tthen neighbors of these vertices with first color and so on. Then one can assign colors to all vertices of a graph such that no two adjacent vertices have the same color. If two accented vertices do have the same color then its not 2-colorable,  otherwise it is true.

Two\_Colorable(Adjacenttlist):

   for each vertex in adjacent list:

      for each neighbors vertices (edges from this vertex)

         color vertices with alternate color

         if color[u] == color [v]

         return false

      end

   end

   return true

b)

adjacent list contains:

V0: {edges from V0}

V1: [edges from V1}

                .

                .

                .

Vn: {edges from Vn}

5)

a) Dijkstra’s algorithm would be used to find the fastest route from the fire station to each intersection. With Dijkstra’s, it is shown that Path G,E,D,C,A is our fastest route with having the quickest access to all areas. Can see this below:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| visited | A | B | C | D | E | F | G | H |
| G | INF | INF | 9 | 7 | 2 | 8 | 0 | 3 |
| E | INF | 11 | 9 | 5 | 2 | 8 | 0 | 3 |
| H | INF | 6 | 9 | 5 | 2 | 8 | 0 | 3 |
| D | INF | 6 | 8 | 5 | 2 | 8 | 0 | 3 |
| B | INF | 6 | 8 | 5 | 2 | 8 | 0 | 3 |
| C | 12 | 6 | 8 | 5 | 2 | 8 | 0 | 3 |
| F | 12 | 6 | 8 | 5 | 2 | 8 | 0 | 3 |
| A | 12 | 6 | 8 | 5 | 2 | 8 | 0 | 3 |

b) Run Dijkstra’s algorithm with each vertex as the source. For each remember the maximum key extracted from extractMin and then select the one with the smallest maximum key. Since you just need to run Dijkstra’s algorithm f times, the time complexity is f · O(r + f log f) when using Fibonacci heaps. Thus the overall time complexity is O(rf + f 2 log f).

c) The above graph the optimal location is place D. With E as the parent and distance near it = 17 (shortest distances between nodes).