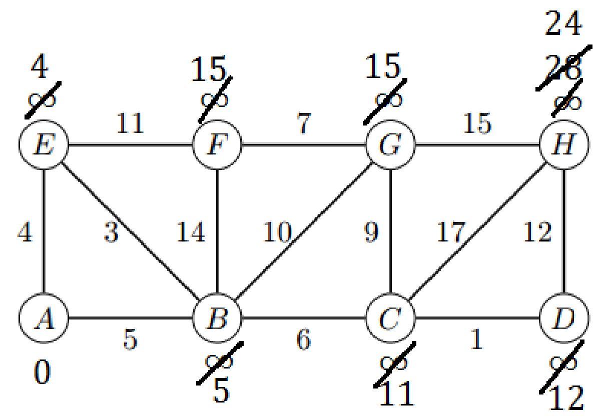
Jonathan Nicolosi

Homework 3

1.

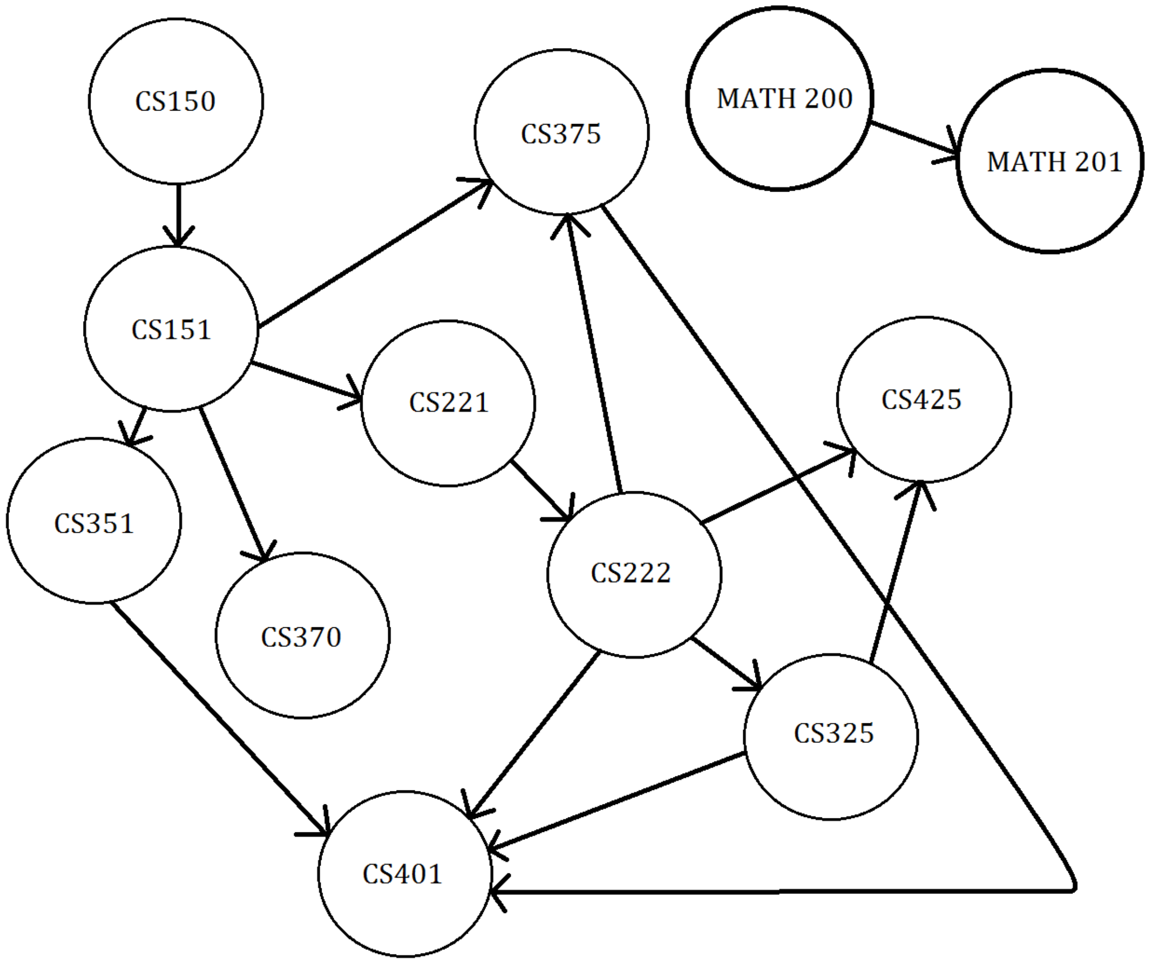
a. A, E, B, C, D, G, F, H



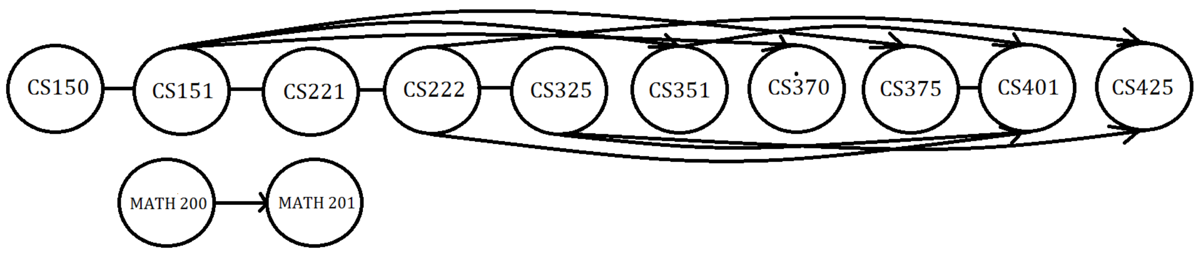
b.

2.

a.


CS150
CS151
CS375
MATH 200
MATH 201
CS221
CS425
CS351
CS222
CS370
CS325
CS401

b.


CS150
CS151
CS221
CS222
CS325
CS351
CS370
CS375
CS401
CS425
MATH 200
MATH 201

c. CS150, CS151, CS221, CS222, CS325, CS351, CS370, CS375, CS401, CS425
MATH 200, MATH 201

d. The longest path in the DAG is 5. At most you must traverse 5 edges to get from CS151 to CS401. This means that CS 401 requires the most prerequisites to take.

3.

a. Pick some vertex v and initialize it to red. Then all its neighbors should be blue. All their neighbors should be red, their neighbors blue and so forth, sweeping across the graph until everything in v's connected component has a color. Then perform a check to see if there are any edges that violate our premise (both endpoints the same color). If so then the graph is not 2-colorable.

```
Int graphsize;

Int colorArray[graphsize];

Queue<int> q;

q.push(src);

while(!q.empty()){

int u = q.front();

q.pop();

        for(int v = 0; v < graphsize; v++){

                if(G[u][v] && colorArr[v] == -1){

                        colorArr = 1 – color[u];

                        q.push();

                }

                Else if(G[u][v] && colorArr[v] == colorArr[u])

                        Return false;

        }

}

Return true;
```

b. O(|V|+|E|)

4.

a. Dijkstra's Algorithm would be the best for this application.

Vertex	Distance from Source	Route
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	-
H	3	G, H

b. In order to find the optimal location, we should perform Dijkstra's algorithm using each vertex as a starting node and keep track of the shortest maximum length path for each starting node.

```
// Number of vertices in the graph
#define V 8

int maxArr[7];
int counter = 0;

// A utility function to find the vertex with minimum distance value, from
// the set of vertices not yet included in shortest path tree
int minDistance(int dist[], bool sptSet[])
{
  // Initialize min value
  int min = INT_MAX, min_index;

  for (int v = 0; v < V; v++)
    if (sptSet[v] == false && dist[v] <= min)
        min = dist[v], min_index = v;

  return min_index;
}
```

```
void dijkstra(int graph[V][V], int src)
{
    int dist[V];     // The output array.  dist[i] will hold the shortest
                    // distance from src to i

    bool sptSet[V]; // sptSet[i] will true if vertex i is included in shortest
                   // path tree or shortest distance from src to i is finalized

    // Initialize all distances as INFINITE and stpSet[] as false
    for (int i = 0; i < V; i++)
      dist[i] = INT_MAX, sptSet[i] = false;

    // Distance of source vertex from itself is always 0
    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V-1; count++)
    {
     // Pick the minimum distance vertex from the set of vertices not
     // yet processed. u is always equal to src in first iteration.
     int u = minDistance(dist, sptSet);

     // Mark the picked vertex as processed
     sptSet[u] = true;

     // Update dist value of the adjacent vertices of the picked vertex.
     for (int v = 0; v < V; v++)

       // Update dist[v] only if is not in sptSet, there is an edge from
       // u to v, and total weight of path from src to  v through u is
       // smaller than current value of dist[v]
       if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                              && dist[u]+graph[u][v] < dist[v])
          dist[v] = dist[u] + graph[u][v];
    }


    int maximum = dist[0];

    for(int b = 0; b<7; b++){

       if(dist[b]>maximum){
          maximum = dist[b];
       }

    }
    maxArr[counter] = maximum;
    counter++;

}

int main()
{
   /* Example graph */
   int graph[V][V] = {
      {0,0,4,0,0,7,0,0},
      {0,0,0,0,9,0,0,3},
      {4,0,0,3,0,2,9,0},
      {0,0,3,0,3,0,7,0},
      {0,9,0,3,0,0,2,7},
      {7,0,2,0,0,0,8,0},
      {0,0,9,7,2,8,0,3},
      {0,3,0,0,7,0,3,0}

                   };

   for(int c = 0; c<7; c++){
      dijkstra(graph, c);

   }

   int minimum = maxArr[0];

   for(int a = 0; a<7; a++){

      if(maxArr[a]<minimum){
          minimum = maxArr[a];
      }

   }
   printf("%i", minimum);
   return 0;
}
```

Time Complexity:
Implemented with Dijkstra’s algorithm using an adjacency matrix takes $O(V^2)$ time.
But we are computing the shortest path starting at every vertex so $O(V*V^2) = O(V^3)$.

c. The optimal location for the fire station would be location E as it has the shortest maximum distance to any other vertex (10).