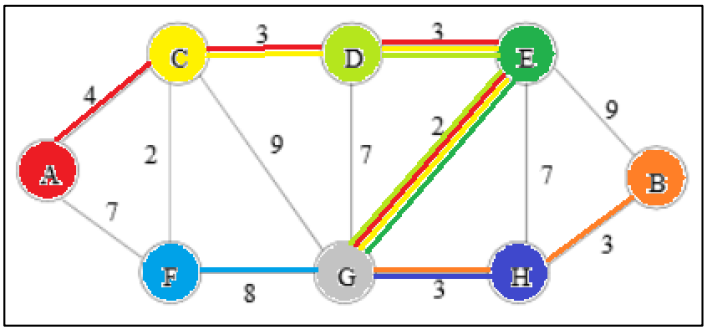
CS325H
Craig Weeks

HW4

1. Towns + Roads

   a. I would recommend using Dijkstra's Algorithm as it works well on directed and undirected graphs, and no road will ever take negative time to travel, so there is no concern about negative edges. Initially, all vertices would have weight infinity and the source vertex would have weight 0. As long as not all the vertices were looked at, the algorithm would start at G and work through the adjacent vertices, always selecting the minimum path weight. It would update ("relax") all of the adjacent vertices to the selected vertex to ensure a shortest path is found. The resulting routes are listed below.

Vertex	Distance
A	12
B	6
C	8
D	5
E	2
F	8
G	0
H	3

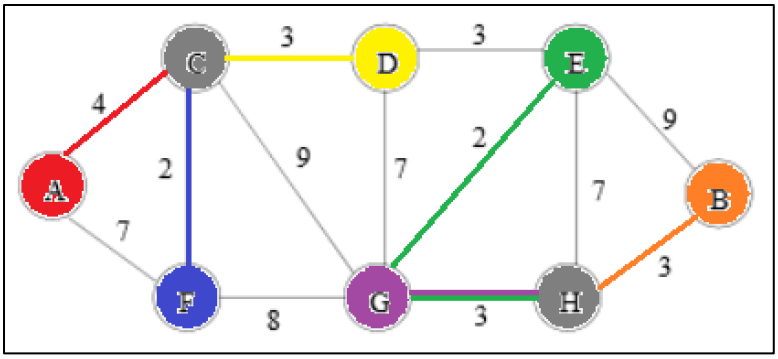


   b. To find the distribution center with the shortest "longest route", I would run the above algorithm starting at each of the vertices and select the vertex with the smallest "longest path." The time complexity of this algorithm would be O(T^3) where T is the number of towns since I'm using Dijkstra's with an array setup, (T^2) and Dijkstra's is run T times.

   c. After running the algorithm, the best place to locate the distribution center is at vertex E with a maximum distance of 10.

   d. To account for two distribution centers, I would run Dijkstra's algorithm on each pair of vertices and select the minimum distance between the two outputs. Then, whichever pair yields the shortest "longest path" would be returned. This has the advantage of being simple to implement, but it is a very inefficient algorithm, running in O(T^4) time.

   e. The two optimal towns for the distribution centers are C and H which yield an overall longest path of only length 5.



2. MST

Description

This approach uses a version of the Prim algorithm with an adjacency matrix. The code loops through, adding edges until the requisite V-1 edges have been selected for the MST. It loops through the vertices and checks if each one has been selected already. If it has, the code loops through all of the adjacent vertices in the matrix and updates the minimum key value and position. After the minimum adjacent vertex is found, it is selected and the appropriate weight is added. When the weights are summed in a sum variable, the output is the total MST weight.

Pseudocode

```
//Graph G, num vertices n, start vertex r
totalWeight = 0
sel = [0,0,0,0,0,0,0]    //length n included vertices list
edges = 0
sel[r] = 1  //start at r

while edges < n-1:   //go until n-1 for MST
    key = inf
    i = r           //set key to infinity and move to start vertex
    j = r

    for u in range(n):
        if u is selected:   //if vertex u is included
            for v in range(n):
                if v isn't selected and G[u][v] < key: //if vertex v isnt included and the weight is less than the current key
                    key = G[u][v] //set the key and move there in the graph
                    i = u
```

```
                j = v

        sel[j] = 1 //after looping through, select min vertex
        edges += 1 //increase number of edges
        totalWeight += key

    return totalWeight
```

Running Time

The theoretical running time for the algorithm is order V^2 since the data are stored in a 2D array and updating them takes constant time.