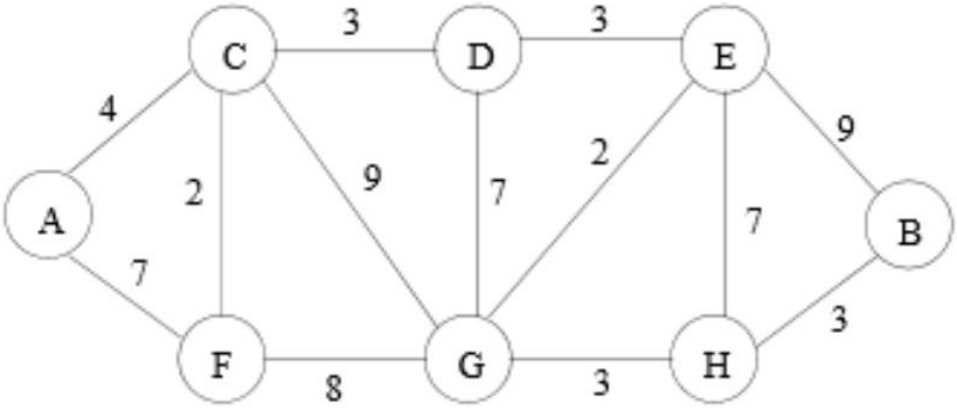
Problem 1.



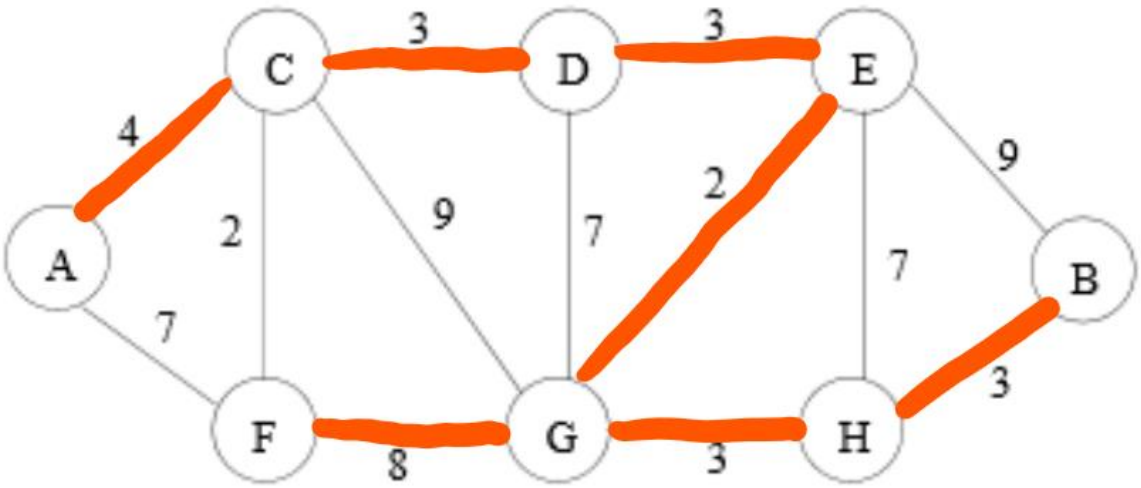
(a) Which algorithm would you recommend be used to find the fastest route from the distribution center to each of the towns? Demonstrate how it would work on the example above if the distribution center is placed at town G. Show the resulting routes.

We can use Dijkstra's algorithm to find fastest route from the distribution center to each of the towns.

Dijkstra's algorithm will work for distribution center G by selecting the fastest route for each vertex from G and update the routes for each iteration.

To be specific, the algorithm will set the distance from G and every vertex as infinite. Then the algorithm will select adjacent vertices from G, and put the edge of adjacent vertex to priority queue. Then the algorithm chooses minimum weight of the edge and then move to that vertex. The algorithm iterates these processes until all vertices in the graph are selected. Through these processes, Dijkstra's algorithm finds the fastest route from the distribution center G to each of the towns.

The resulting route is below. (The routes are marked in orange color.)



(b) Suppose one ”optimal” location (maybe instead of town G) must be selected for the distribution center such that it minimizes the time travelled to the farthest town. Devise an algorithm to solve this problem given an arbitrary road map. Analyze the time complexity of your algorithm when there are t possible towns (locations) for the distribution center and r possible roads.

Algorithm:

I. Do Dijkstra’s algorithm for each vertex, and add up each weight.

II. Store the sum of the weight of edges and store it with source vertex in the list .

III. Find the minimum value in the list.

IV. Optimal Town is found.

Time Complexity:

Assume the Dijkstra’s algorithm is using array for priority queue.

i. Do Dijkstra’s algorithm for each vertex, and add up each weight → $O(t^3)$

ii. Store the sum of weight of edges and store it with source vertex in the list

→$O(1)$

iii. Find minimum breadth in the list → $O(t)$

Total Running Time:

$$O(t^3) + O(1) + O(t) = O(t^3)$$

$$\therefore O(t^3)$$

Therefore, running time of algorithm is $O(t^3)$

(c) In the above graph which is the “optimal” town to locate the distribution center?

The “optimal” town to locate the distribution center is town E (town A, town C, or town D).

(d) Now suppose you can build two distribution centers. Where would you place them to minimize the time travelled to the farthest town? Describe an algorithm to solve this problem given an arbitrary road map.

Algorithm:

I. Do Dijkstra's algorithm for each vertex.

II. Store the sum of the weight of edges and store it with source vertex in the list.

III. Find two minimum values in the list.

IV. Two optimal towns are found.

Time Complexity:

Assume Dijkstra's algorithm is using array for priority queue.

i. Do Dijkstra's algorithm for each vertex

→ $O(t^3)$

ii. Store the sum of the weight of edges and store it with source vertex in the list

→$O(1)$

iii. Find two minimum values in the list.

→ $O(t)$

Total Running Time:

$$O(t^3) + O(1) + O(t) = O(t^3)$$

$$\therefore O(t^3)$$

Therefore, running time of algorithm is $O(t^3)$.

(e) In the above graph what are the "optimal" towns to place the two distribution centers?

The "optimal" towns to place the two distribution centers are town D and town E.

(Or <town A, town C> , <town A, town D>, <town A, town E> , < town C, town D>, <town C, town E>)

Problem 2.

• A verbal description of your algorithm and data structures

I used prim's algorithm to find the weight of minimal spanning tree with using adjacency matrix because the graph is complete.

First, the program reads each coordinate from the text file, and calculate distance between each coordinate. After calculating distance, the program fills adjacency matrix with calculated distance. Then the program initializes key array with infinite value and checking array which checks if the vertex is visited with zero. Furthermore, the program initializes parent array in order to track the process.

The program searches entire adjacency matrix with extracting index which has minimum value in key array, and updating key array and checking array. In this process, the program adds up key value of visited vertex to the weight of MST initialized as zero.

When the entire vertices are visited, then the program returns the weight of Minimal Spanning Tree of the graph.

• The pseudo-code

I. Matrix[i][j] ← calDistance(start[i],start[j],end[i], end[j])

II. For each v, key[v] ← INF, check[v] ← 0

III. Parent[n] , MST Weight ← 0

IV. For i = 1~ n(the number of vertex){

'u' ← extractMin(key, check) // extract index which has minimum value

Check[u] ← 1

For j = 1 ~n{

If(key[j] > Matrix[u][j] and check[j] = 0){

Parent[j] ← u

Key[j] ← Matrix[u][j] }

}

MST Weight += Key[u]

V. Return MST Weight.

• Theoretical running time

Assume the number of vertices of the graph is n.

I. Filling Adjacency matrix → $O(n^2)$

II. Initialize and fill key array and checking array → $O(n)$

III. Initialize parent array and MST Weight→ $O(1)$

IV. Step four in pseudo-code → $O(n^2)$

➔ $\because n * (n + n + 1) = 2n^2 + n = O(n^2)$

➔ $ExtractMin \rightarrow O(n)$

➔ $key\ array, checking\ array, and\ parent\ array\ update \rightarrow O(n)$

➔ $Adding\ key[u]\ to\ MST\ Weight \rightarrow O(1)$

V. Return MST Weight → $O(1)$

Total Theoretical Running Time:

$$O(n^2) + O(n) + O(1) + O(n^2) + O(1) = O(n^2)$$

$$\therefore O(n^2)$$

Therefore, theoretical running time is $O(n^2)$