Gretel Rajamoney

February 19, 2021

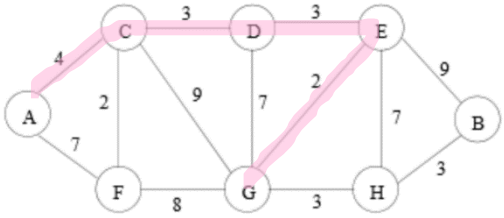
CS 325 Homework 5

Problem 1:

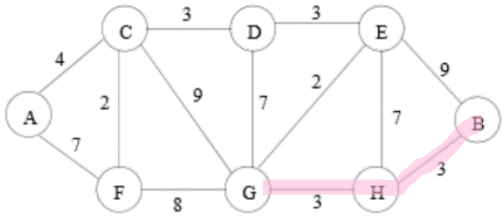
a.

I would use the dijkstra's shortest path algorithm in order to find the fastest route from the distribution center to each of the towns. If the distribution center is placed at Town G:

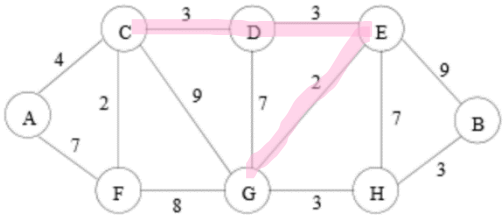
Town A to Distribution Center: 12



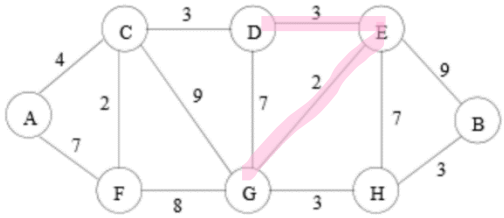
Town B to Distribution Center: 6



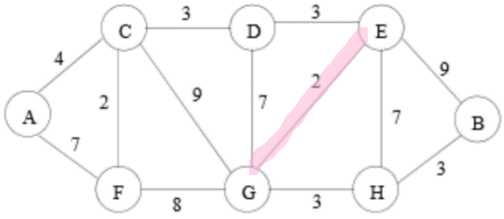
Town C to Distribution Center: 8



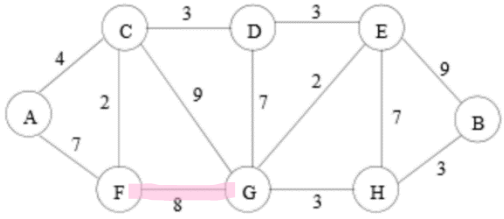
Town D to Distribution Center: 5



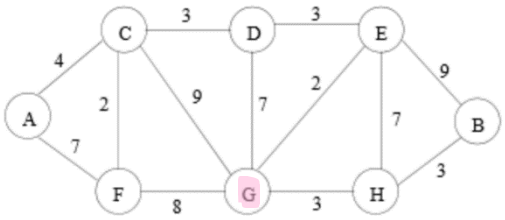
Town E to Distribution Center: 2



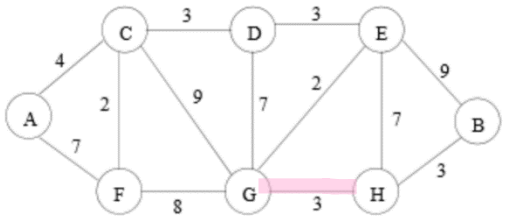
Town F to Distribution Center: 8



Town G to Distribution Center: 0



Town H to Distribution Center: 3



b.

To find the optimal location that must be selected for the distribution center such that it minimizes the distance to the farthest intersection, we can use dijkstra's shortest path algorithm. This algorithm is very similar to prim's algorithm which has been used for problem 2 and implemented within the mst.cpp program. The algorithm will be implemented using an adjacency matrix which contains the all of the distances between each of the distribution centers. The algorithm begins by creating a function that returns the index of the minimum distance value, essentially where the path will be started when dijkstra's algorithm is implemented. Next when the algorithm function is called, it begins by initializing all of the distances to be set as infinite, as well as the boolean array to be set to false. Using the minimum distance function, the location of the minimum distance is marked with the value true in the boolean array. Each of the vertex distances are repeatedly compared and the shortest path is always selected, the total distances are also repeatedly counted as the loop iterates. The program finished by printing out the total distance that was totaled.

```
// function that makes the adjacency matrix
// takes in x and y coordinate arrays and total amount of vertices
void makeAdjMatrix(int x[], int y[], int v)
{
	adj[][]; // makes matrix

	for (all indexes of the matrix)
	{
		// calculate the distance between each coordinate
		formula = sqrt((x1 - x2)^2 + (y1 - y2)^2)
	}
	// calls the prims algorithm to find the mst
	dijkstraAlg(adj[][], v);
}

void dijkstraAlg(int adjMatrix[][], origin, destination)
{
	for (each vertex in the matrix)
	{
		// initializes distances to infinity and null besides first index
```

```
                dist[] = infinity
                prev[] = null
                dist[0] = 0;
        }
        for (each vertex in the matrix)
        {
                // finds the starting distribution center with smallest path
                u = findMin(adjMatrix[][])

                        // parses through array finding the smallest path
                        for (each vertex in the matrix)
                        {
                                if (dist[] > dist[] + distance between centers)
                                {
                                        dist[] = dist[] + distance between centers
                                        prev[] = loop counter value;
                                        findMin(adjMatrix[][], loop counter value)
                                }
                        }
        }

}

int findMin(int key[], bool mst[], int v)
{
        // repeatedly parses through array until key is found
        for (all indexes of array)
        {
                if (mst[] = false and key[] < min)
                {
                        set minindex = index;
                }
        }
        // return the minimum key
        return minindex
}
```

Since an adjacency matrix is implemented in order to store the distances between each and every vertex the time complexity of the above algorithm is:

Time Complexity = $O(V^2)$

c.

The optimal location to place the distribution center would be at Town E. This is because the distance of the farthest town with the distribution center located at Town E is totaled to be 10, which is lower than the farthest distance of placing the first center at all other towns. Below are calculated distance tables of placing the distribution center at all of the towns. Under each table is the calculated distance from the farthest town from the distribution center. Thus, proving that the most optimal town to place the distribution center is Town E.

TOWN A:

VERTEX	DISTANCE	ROUTE
A	0	A
B	18	B – H – G – E – D – C - A
C	4	C - A
D	7	D – C - A
E	10	E – D – C - A
F	6	F – C - A
G	14	G – F – C - A
H	15	H – G – E – D – C - A

FARTHEST TOWN DISTANCE = 18

TOWN B:

VERTEX	DISTANCE	ROUTE
A	18	B – H – G – E – D – C - A
B	0	B
C	14	C – D – E -G – H - B
D	11	D – E – G – H - B
E	8	B – H – G - E
F	14	B – H – G - F
G	6	G – H - B
H	3	H - B

FARTHEST TOWN DISTANCE = 18

TOWN C:

VERTEX	DISTANCE	ROUTE
A	4	C - A
B	14	C – D – E -G – H - B
C	0	C
D	3	C - D
E	6	C – D - E
F	2	C - F
G	9	C - G
H	11	H – G – E – D - C

FARTHEST TOWN DISTANCE = 14

TOWN D:

VERTEX	DISTANCE	ROUTE
A	7	D – C - A
B	11	D – E – G – H - B
C	3	D - C
D	0	D
E	3	D - E
F	5	D – C - F
G	5	D – E - G
H	8	H – G – E - D

FARTHEST TOWN DISTANCE = 11

TOWN E:

VERTEX	DISTANCE	ROUTE
A	10	A - C – D - E
B	8	B – H – G - E
C	6	C – D - E
D	3	D - E
E	0	E
F	8	F – C – D - E
G	2	G - E
H	5	H – G - E

FARTHEST TOWN DISTANCE = 10

TOWN F:

VERTEX	DISTANCE	ROUTE
A	6	A – C - F
B	14	B – H – G - F
C	2	C - F
D	5	D – C - F
E	8	E – D – C - F
F	0	F
G	8	G - F
H	11	H – G - F

FARTHEST TOWN DISTANCE = 14

TOWN G:

VERTEX	DISTANCE	ROUTE
A	14	G – F – C - A
B	6	G – H - B
C	9	C - G
D	5	D – E - G
E	2	G - E
F	8	G - F
G	0	G
H	3	G - H

FARTHEST TOWN DISTANCE = 14

TOWN H:

VERTEX	DISTANCE	ROUTE
A	15	H – G – E – D – C - A
B	3	H - B
C	11	H – G – E – D - C
D	8	H – G – E - D
E	5	H – G - E
F	11	H – G - F
G	3	H - G
H	0	H

FARTHEST TOWN DISTANCE = 15

d.

In order to find the two optimal towns to place two distribution centers, we must modify dijkstra's shortest path algorithm to take account for two starting points as oppose to one. As performed in the original dijkstra's shortest path algorithm above, we must first start by finding the distances to each town from each town. We do this by creating an adjacency matrix that stores the distances from each town. Next to incorporate the algorithm, we wil, run the algorithm for each pair of coordinates, essentially dijkstraAlg(x) and dijkstraAlg(y). Next, we compare the result of both of these functions repeatedly keep track of the smallest path. Once the function iterate for all of the towns on the map, we will know the two most optimal locations to place the distribution centers such that the distance of the farthest town from the distribution center is minimalized.

e.

The two most optimal towns to place the distributions centers in order to minimalize the distance from the farthest town from a distribution center are at Towns C and H.

Problem 2:

Verbal Description: To determine the weight of a MST where the vertices are points in the plane and the weight of the edge between each pair of points is the Euclidean distance between those points, I have made the decision to implement Prim's Algorithm. The program begins by reading in all the vertex coordinates from graph.txt file, next it calls the function in order to locate the minimum key vertex which the algorithm will start with. After locating the minimum key, the program will next call the prim's algorithm function in order to calculate the MST of all the vertices. The algorithm begins by creating an array that initializes all of the values to start with the value infinite. Next through calling the minimum key function, the index of the key value is replaced with the value 0 so that the program knows to start at that location. Next the algorithm creates the path for the MST by comparing every path's distance and selecting the shortest distance. As the path is being chosen, each distance is repeatedly added and accumulated so that the MST can be outputted after the test case finishes running.

Pseudocode:

```
// function that makes the adjacency matrix
// takes in x and y coordinate arrays and total amount of vertices
void makeAdjMatrix(int x[], int y[], int v)
{
	adj[][]; // makes matrix

	for (all indexes of the matrix)
	{
		// calculate the distance between each coordinate
		formula = sqrt((x1 - x2)^2 + (y1 - y2)^2)
	}
	// calls the prims algorithm to find the mst
	findMST(adj[][], v);
}

void findMST(int adj[][], int v)
{
	// creates storage arrays
	og[];
	key[];
	mst[];

	for (all indexes of array)
	{
```

```
                // stores infinity and false at all indexes
                key[] = infinity;
                mst[] = false;
        }
        // stores a 0 and -1 at the first index
        key[0] = 0
        og[0] = -1

        for (all indexes of array)
        {
                // calls function to find the minimum key index
                findMin()
                // sets the index to true to mark it
                mst[minimum key index] = true;
        }

        for (all indexes of array)
        {
                if (matrix is smaller than the key value)
                {
                        // update the key
                        og[] = value;
                        key[] = adj[][];
                }
        }

        for (all indexes of array)
        {
                // repeatedly adds distance
                measure = measure + distance
        }

        cout << measure
}

int findMin(int key[], bool mst[], int v)
{
        // repeatedly parses through array until key is found
        for (all indexes of array)
        {
                if (mst[] = false and key[] < min)
                {
                        set minindex = index;
                }
        }
        // return the minimum key
        return minindex
}

int main()
{
        read in graph.txt file using ifstream
        // store total test cases
        graph >> testCases
        // store total vertices
        graph >> v

                for (int x = 0 to x = total vertices v)
```

```
            {
                // store x coordinate
                graph >> x[]
                // store y coordinate
                graph >> y[]
            }
        // call function to create adjacency matrix
        makeAdjMatrix(x, y, v)
}
```

Theoretical Running Time: Since an adjacency matrix is implemented in order to store the distances between each and every vertex the time complexity of the above algorithm is:

Time Complexity = $O(V^2)$