Josh Bell

CS – 325

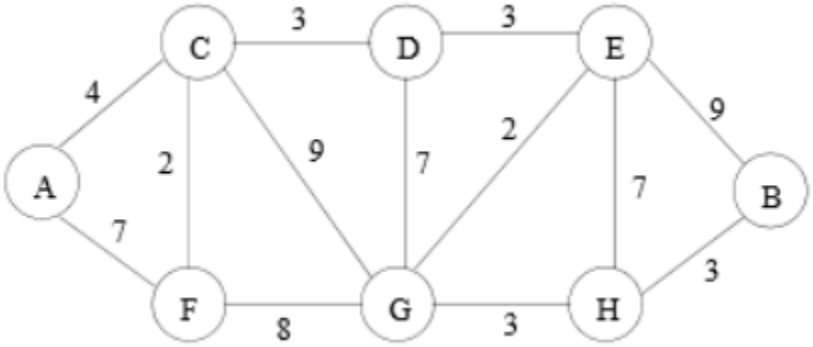
May 13th, 2021

# Homework 4

GitHub Repo: https://github.com/JoshBell302/CS325-AoA-Assignment-4

## Problem 1

**A region contains a number of towns connected by roads. In the graph below, each town is labeled with a circle and each road is labeled with the average number of hours required for a truck to travel between the cities. Suppose that you work for a large retailer that has decided to place a distribution center in town G. (While this problem is small, you want to devise a method to solve much larger problems).**




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.**

The algorithm I would recommend would be the Dykstra algorithm, it would start at G and would go through each vertex and find the fastest method to the individual vertex. Demonstrating how it would work, since we are starting at vertex G, let us say in this case we want to find the shortest path to D, we then look at all connected vertexes in this case they are: F, C, D, E, and H. List their weight and where they came from, so they all came from G, so we just add their weight …

*[(G➔E = 2), (G➔H = 3), (G➔D =7), (G➔F = 8), (G➔C = 9)]*

Then we check the next lowest weight which is E. So, then we look at all connecting vertices they are: D, G, B. We do the same things as last time, note their previous vertex they visited and their weight. Since D already has a weight, we check to see if it is

smaller which it is, so we note down its that it came from E instead of G and the weight which is 2 + 3 so 5. We ignore G in this case and fill out B, which gives us …

*(G➔E = 2)*

*[(G➔H = 3), (E➔D = 5), (G➔D =7), (G➔F = 8), (G➔C = 9) (E➔B = 11)]*

We then check H and fill out each vertex which gives us …

*(G➔E = 2), (G➔H = 3)*

*[(E➔D = 5), (H➔B = 6), (G➔D =7), (G➔F = 8), (G➔C = 9)]*

Now that we reached our destination D, we traverse backwards to find the shortest path and its weight which is G➔E➔D with weight 5. If we did not have a selected destination, we would continue this approach until we have completed the graph.

**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.**

This can be broken down into steps …

1. We start by having an array that has length equal to the number of vertices with the maximum weights.
2. Then loop equal to the number of vertices to set the max weight to be negative infinity (*i*).
3. Following that we start another loop equal to the number of vertices (*j*), then we set a value max equal to the returning value of the Dijkstra's algorithm (we give the algorithm the matrix of the graph and the starting vertex). Then we check to see if the maximum weight is less than the returned value max, if so then set the maximum weight to be the current max value.
4. Once the second loop is complete then we set the array's *ith* value to be the result of the maximum weight. We then go the first loop again.
5. After the loops complete, we then look into the array with the maximum weights and return the smallest value.

The theoretical running time for this algorithm is simple to compute, we loop our code twice, so our written portion is O (t^2). We also can add the time for the Dijkstra's algorithm is seen in O (|r| log(|t|)), we add these together to create the actual running time as O (|r| log(|t|) + t^2), but when we simplify it for theoretical purposes to O (t^2).

**c) In the above graph which is the "optimal" town to locate the distribution center?**

The "optimal" vertex is E.

**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.**

This can be broken down into steps …

1. We start by having a matrix of all towns (*matrix*) and creating a list that contains the key to the maximum weight and its value ( *list(max, val)* ).
2. Then we loop equal to the number of vertices (*i*).
3. Then we begin another loop equal to the number of vertices (*j*), then set the value of *list* the current *ith* and *jth* value to infinity.
4. Then begin to loop another time equal to the number of vertices (*k*), we then find the lowest value of either *matrix ith* and *kth* value or *matrix jth* and *kth* value and set the found value equal to min, then we check if the *list*'s current *ith* and *jth* value is less then the recently found min if so then we update the *list*'s value to that min.
5. Once those loops complete, we then return the lowest value in the *list*, which would return two values for the "optimal" vertices

6. **In the above graph what are the "optimal" towns to place the two distribution centers?**

The "optimal" vertices for this graph are vertex C and H.

# Problem 2: Euclidean MST Implementation

**Implement an algorithm in the language of your choice (C, C++, or Python) that determines the weight of an MST in a graph G = (V, E) 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. To avoid floating point precision problems in computing the square-root, we will always round the distance to the nearest integer. For all u, v ∈V, u = (x1, y1), v= (x2, y2) and**

$$w(u,v) = d(u,v) = nearestint\ \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

**For example, if u = (0,0), v = (1,3), d =** $\sqrt{(0-1)^2 + (0-3)^2} = 3$

**Assume that the graph G = (V, E) is complete so that there is an edge between ever two vertices. Your program should be named mst.py, mst.c or mst.cpp. Your program should read information about the graph from a file named graph.txt. The first line in graph.txt is the number of tests cases (k ≤ 10) followed by the number of vertices in the test case (n ≤ 100) and the coordinates of the points (vertices) in that test case.**

Description: This code reads the data from the text file and fills out the information in a 3D array. This information is then used to calculate the distance between all vertices, and that data is filled out into a 2D array. Once the data of the distances is found and filled, we then put that information into the Prim's algorithm and prints to the terminal what the weight of the minimum spanning tree is.

Pseudo-Code: The beginning of the code begins by looping to gather the information from the "graph.txt" and fills out a 3D array called *coordinates* with that information. This then can be broken into steps …

1. We create a 2D array that collects the distances between all vertices (*graph*).
2. Loop equal to the number of tests to do (*i*) and print to the terminal which test case we are solving for. Then set *graph* to all 0's.
3. Loop equal to the number of vertices of the test case (*j*).
4. Again, loop equal to the number of vertices of the test case (*k*). Then within this loop determine the distance between each vertex and store that information into *graph*. This information of the vertices will be found in *coordinates*.
5. Once out of loop *j* and *k* we then do the prims algorithm to determine the MST of the current graph, we then print to the terminal the weight of the MST.

Theoretical Running Time: The theoretical running time for this code, without counting the portion to read from the file can be found quite quickly. The Prim's Algorithm has a running time of O (E log V). The code runs equal to the number of vertices twice so this can be seen as O (V^2). If we add these values together, we can say that this code runs at around O (V^2).