Emma Zajonc

Professor Sipantzi

April 25, 2024

Minimum Cost Spanning Tree Approach

For this lab, my solution will consist of one header file Graph.h, main.cpp, and four text files named input.txt, output1.txt, output2.txt, and output3.txt respectively. The program will read input from the input.txt file and output the three trees to the three output files.

My class Graph will have five private elements – an integer array to hold the input values, an integer array to hold the output values, an integer variable to track the size of the graph, a bool array to keep track of which vertices have been visited, and an integer edgeCount to keep track of how many edges the original graph has.

Graph will contain a default constructor, a constructor with a parameter for size, a readIn() function to read in the graph from the input file, a readOut() function to read out the minimum cost spanning trees to the output files, a print() function to print the input graph, a printOutput() function to print the output graph, a createMST() function to create a MST from the inputted graph, and a helper function findNearest() to help the createMST() function to find the nearest vertex to the current output graph.

The first functionality I will create is the readIn() function. I will fill the input.txt file with a matrix in the form of six rows, with six numbers to a row and with spaces in between each number in a row. I will then loop through this text file and assign each value to its appropriate spot in a one-dimensional array. I am choosing to use a one-dimensional array to store the graph. I will access the correct inner elements of the graph by using the size of the graph as a modulus. The readIn() function takes one parameter, the string name of the input file. Each time the loop encounters a value in the matrix that is not equal to zero, it increases the edgeCount by one. At the conclusion of the loop, the edgeCount is divided by two to account for the symmetry of the matrix.

Next I will implement the createMST() function, which takes one integer parameter pos, the starting vertex. This function has the most complex logic. I will start by setting the Boolean array visited to false. I will then set the element at visited[pos] to true, setting the starting point of the MST. I will then initialize the output array to 0.

Next, I will start the main loop of the function. I will declare two variables – nearest and origin – that will keep track of the nearest vertex to the current graph and the member vertex the nearest vertex is closest to. I will loop five times, the number of vertices minus one. From this loop, I will call findNearest().

Nearest and origin will be passed by reference into findNearest(). The findNearest function also has a variable called minimum to keep track of the lowest weight found. I will loop through the matrix rows, using the Boolean array to check if each corresponding vertex has been visited and added to the MST. If it has, I will loop through the vertices represented by the columns in that row. For each vertex, I will check if it is not equal to zero, is less than the current minimum, and has not been visited before. If all of these conditions are met, I will set nearest to the counter in the inner loop, origin to the counter in the outer loop, and minimum to the weight of that position in the array.

Back in creatMST(), I will add the matrix value represented by origin and nearest to the output array and print the details of the added edge. I will also set visited[nearest] to true, adding the new vertex to the graph as processed by createMST().

The print(), printOutput(), and readOut() functions all work fairly similarly, looping through the values in the matrix and displaying them through either a file or the terminal.