Assignment 11: Weighted Graphs
Part 1: Minimum Spanning Tree with Adjacency Matrix
CS3305/W01 Data Structures

Casey Hampson

November 20, 2024

Program Output

```
Weight: 20.000000
Root is: 0
Edges: (7, 1) (0, 2) (2, 3) (3, 4) (2, 5) (4, 6) (6, 7)
```

Source Code

```
// Name: Casey
// Class: CS 3305/W01
// Term: Fall 2024
// Instructor:  Sharon Perry
// Assignment:  11-Part-1-Prims

import java.util.List;
import java.util.ArrayList;


class WeightedGraphAdjacencyMatrix<V> extends WeightedGraph<V> {
        WeightedGraphAdjacencyMatrix(int[][] edges, int num_vertices) {
                super(edges, num_vertices);
        }

        // creates an adjacency matrix constructed from the
        // adjacency lists used in the previous algorithm definition
        public double[][] createAdjacencyMatrix() {
                int size = this.getSize();
                double[][] adjacency_matrix = new double[size][size];

                // initialize to -1 everywhere
                // this is our "null" equivalent
                // since all weights here are positive
                for (int i=0; i<size; i++) {
                        for (int j=0; j<size; j++) {
                                adjacency_matrix[i][j] = -1.0;
                        }
                }

            for (int i=0; i<size; i++) {
                        List<Edge> edges = this.neighbors.get(i);
                        for (Edge e : edges) {
                                // we need to cast to a weighted edge
                                // so that we have access to the weight
                                WeightedEdge we = (WeightedEdge)e;
                                adjacency_matrix[we.u][we.v] = we.weight;
                        }
                }


                return adjacency_matrix;
        }

        @Override
        public MST getMinimumSpanningTree(int starting_vertex) {
                // cost[v] stores the cost by adding v to the tree
                double[] cost = new double[getSize()];
                for (int i = 0; i < cost.length; i++) {
                        cost[i] = Double.POSITIVE_INFINITY; // Initial cost
                }
                cost[starting_vertex] = 0; // Cost of source is 0
```

```java
            int[] parent = new int[getSize()]; // Parent of a vertex
            parent[starting_vertex] = -1; // starting_vertex is the root
            double totalWeight = 0; // Total weight of the tree thus far

            // list of vertices in our MST
            List<Integer> T = new ArrayList<>();

            // grab the adjacency matrix for this graph
            // so we can use it in the algorithm
            double[][] adjacency_matrix = this.createAdjacencyMatrix();

            // Expand T
            while (T.size() < getSize()) {
                    // Find the vertex with the smallest cost
                    // in the set V - T
                    int u = -1; // Vertex to be determined
                    double currentMinCost = Double.POSITIVE_INFINITY;
                    for (int i = 0; i < getSize(); i++) {
                            if (!T.contains(i) && cost[i] < currentMinCost) {
                                    currentMinCost = cost[i];
                                    u = i;
                            }
                    }

                    T.add(u); // Add a new vertex to T
                    totalWeight += cost[u]; // Add cost[u] to the tree

                    // the following loop is basically all of what is changed.

                    // we now adjust the costs for all vertices adjacent to the new vertex
                    // for this we consider the row of the adjacency matrix given by
                    // adjacency_matrix[u][i] (we loop over i, the size/num of vertices)
                    // and replace all the weighted edge stuff with the value of the
                    // adjacency_matrix at position (u,i)
                    for (int i=0; i<this.getSize(); i++) {
                            if (
                                    (adjacency_matrix[u][i] >= 0.0) &&
                                    !T.contains(i) &&
                                    (cost[i] > adjacency_matrix[u][i])
                            ) {
                                    cost[i] = adjacency_matrix[u][i];
                                    parent[i] = u;
                            }
                    }
            } // End of while

            return new MST(starting_vertex, parent, T, totalWeight);
        }
}
```

```java
public class P1 {
	public static void main(String[] args) {
		// set up the graph given in the example
		final int num_vertices = 8;
		int[][] edges = {
			{0, 2, 4}, {0, 5, 7},
			{1, 4, 9}, {1, 7, 3},
			{2, 0, 4}, {2, 3, 3}, {2, 5, 2}, {2, 6, 9},
			{3, 2, 3}, {3, 4, 3}, {3, 6, 7},
			{4, 1, 9}, {4, 3, 3}, {4, 6, 2}, {4, 7, 7},
			{5, 0, 7}, {5, 2, 2}, {5, 6, 8},
			{6, 2, 9}, {6, 3, 7}, {6, 4, 2}, {6, 5, 8}, {6, 7, 3},
			{7, 1, 3}, {7, 4, 7}, {7, 6, 3}
		};

		WeightedGraphAdjacencyMatrix<Integer> weighted_graph =
			new WeightedGraphAdjacencyMatrix<>(edges, num_vertices);

		WeightedGraph.MST mst = weighted_graph.getMinimumSpanningTree(0);
		System.out.printf("Weight: %f\n", mst.getTotalWeight());
		mst.printTree();
	}
}
```