COMP3010 Assignment 3 Report

Ameer Karas

44948956

Introduction

The third problem for Assignment 3 can be used to solve the first and second, so this is the problem that the document will focus on. This task is to determine the cheapest cost for sending a maximal number of packages from a warehouse to a disaster location. This is accomplished by sending the packages along supply routes which are paths that lead to the intended destination. The supply routes have a toll, however, with some routes being more/less expensive than others. Additionally, these supply routes can only transport a specific capacity of packages. For the purpose of this assignment, the warehouse, disaster location, route cost and route capacity will be referred to as start/source, end/destination, cost and capacity, respectively.

Variables And Their Purpose

v: the number of vertices for a given input.

e: the number of edges/connections for a given input.

capMatrix: a 2-dimensional int array to store the capacity of a given input.

costMatrix: a 2-dimensional int array to store the cost of a given input.

Visited: a boolean array used to determine whether an edge has been visited.

n: an int data type that stores the number of nodes.

Cap: a 2-dimensional int array that stores the capacity of each edge.

Flow: a 2-dimensional int array that stores the flow between vertices.

Cost: a 2-dimensional int array that stores the cost of flow.

nN: an int array that stores the negative edges to be removed from the flow.

Dist: an int array that stores the distance from a node.

cE: an int array that stores the chosen edges.

Methods

readData

The readData method takes a string input file as an input. Two 2-dimensional arrays are created to store values for two adjacency matrices, that are populated by the given data. From this data, the readData method outputs 2 adjacency matrices for the given graph: one adjacency matrix for capacity, one adjacency matrix for cost. These are capMatrix and costMatrix, respectively.

findPath

The helper method for the main method is findPath. This is a boolean method that takes the source vertex and destination vertex as input parameters and returns the best possible path from the source vertex to the destination vertex. This method is used in the main method bestCostMaxCap as the condition in the while loop.

bestCostMaxCap

The main method of the algorithm is the bestCostMaxCap method. This method takes 4 parameters:

1. A two-dimensional int array that stores the capacity.
2. A two-dimensional int array that stores the cost.
3. The source vertex.
4. The destination vertex.

This method returns a one-dimensional int array with the values for the maximum capacity and minimum cost. The bestCostMaxCap method uses the helper method findPath as the condition for a while loop. The loop iterates until there is a path from the source vertex to the destination vertex. During the iterations, a variable *amount* is instantiated as infinity, representing the distance to the unvisited nodes. Its value is then updated as the smaller of 2 options:

1. The flow from the destination vertex to the current vertex, or
2. The capacity from the current vertex to the destination vertex subtracted by the flow of the current vertex to the destination vertex.

A Bellman-Ford algorithm is utilised here (focused on flow), as this *amount* value is the value for the shortest path. In this case, the flow of each edge determines the shortest path. Therefore, the Bellman-Ford finds the minimal flow for a path from the source vertex to the destination vertex.

Once this calculation has been completed, the value for *amount* is used to calculate the minimum cost and maximum capacity. The minimum cost is calculated as the previous iteration's minimum cost subtracted by the *amount* value multiplied by the cost for each edge. The maximum capacity is the sum of the *amount* values.

Algorithm Outline

This algorithm finds the cheapest cost for sending a maximal capacity when given a graph. This graph is provided by the readData method and represented as an adjacency matrix.

The main method's while loop uses the helper method findPath. The findPath method checks each vertex's adjacent neighbours from the source to the destination to verify that there exists a path. That is to say, this method works from the source vertex to the destination, selecting the best available path along the way. The best available path is defined as the path with the shortest distance. This is performed at each iteration of the while loop, which will iterate until the best current path reaches the source vertex from the destination vertex.

The Bellman-Ford algorithm is used in the main method's while loop to calculate the shortest path from the source vertex to the destination vertex. However, like with reduction in NP-complete, the result of the Bellman-Ford algorithm has been augmented to return the lowest cost for the largest capacity, rather than the shortest path.

The primary method of the algorithm is the bestCostMaxCap method. This method takes the source node, destination node, and two adjacency matrices, costMatrix and capMatrix, as inputs. The data from these inputs is provided from the given files and are read by the readData method. The helper method findPath is used as the condition for a while loop, that iterates if there is a path from the source vertex to the destination vertex. Within the while loop, a for loop iterates through the graph until a variable *x* reaches the source, where *x* is initialised as the destination vertex. At each iteration, a variable *amount* is calculated; *amount* is instantiated as infinity and is later updated to be the shortest path in terms of flow. The solution to minimum cost can then be found as the value for *amount* is multiplied by each edge's cost. This value is then subtracted from the previous iteration's minimum cost until the for loop completes. To calculate the maximum capacity the *amount* value is summed for each iteration, as this value is the maximum possible flow for each iteration. The output of this method is an int array that returns the maximum flow and minimal cost as the first and second elements of the array, respectively.