CMSC-401 Algorithm Analysis with Advanced Data Structures

Homework #4

Lecturer: Hong-Sheng Zhou *Due: Thursday May 5th*

Problem 4-1 (Graphs and Paths) 10%

In this assignment, you will be implementing a graph data structure and calculating a shortest weighted path. We will use tools such as *an adjacency matrix or adjacency list and graph theory* during this exercise.

This assignment will likely take a *significant* amount of time to complete (not one night), please allocate your time appropriately. You should name your main source file `cmsc401.java` and place it, and any other source file in a zip file called `SURNAME_GIVENNAME.zip` and submit it to blackboard by May 5th, 2016.

Please read the remarks at the end to ensure you meet all of the criteria. Do not include any additional folders, directories, screen shots, mimes, class files, etc. in your submission. If you use an IDE, please make sure your source compiles and runs from command line as demonstrated in the example. Please do NOT use packages.

1 Problem Statement

After designing that new service to help hitch-hikers find their way home in assignment 3, it became apparent that this information is pretty useless without providing good directions on how to get there. On top if this, hitch-hiking is supposed to be cheap. Your task is to find the cheapest route between two cities, Richmond and Montgomery, Alabama– home, sweet home! Each city has a set of roads that connect it to other cities, you are tasked with finding the route with the lowest gas costs and travel through cities with the cheapest motels so you can save money. You did your research and found an awesome database on-line with all of the roads, speed limits, gas prices, and motel rates. Now you need to implement an algorithm that calculates the cheapest way to get there!

1.1 Tasks

For this assignment, you will need to implement the following components,

1. **Storage** You will need to read in a list of vertexes, edges, and weights (described below) and store them in either an adjacency list or matrix.
2. **Calculator.** Based on the set of data you read in, you need to calculate a route with minimize the total cost of traveling between city 1 and city 2.

1.2 Input/Output

Everything is through standard in and standard output. There are four types of inputs:

1. The first line is the number of cities (vertexes) N. This falls in the range 3 ≤N≤ 1000.
2. The second line of input are the number of roads (edges) M, in the range 2 ≤N≤ 10000.
3. The following N-2 lines are the motel prices at each city (excluding Richmond and Montgomery), each as a single line. These lines consists of two numbers: the city number (3..N) and the motel cost (1..200).
4. The Last M lines of input are the gas prices for traveling between two cities on a given road, each expressed on a single line. These line consists of three numbers: city number (1..N), city number (3..N), cost of gas between the two cities (1..200).

Correct input format:

```
5
7
3 78
4 87
5 98
1 4 98
5 4 45
1 5 140
4 3 87
2 5 150
3 5 109
3 2 73
```

Correct output:

```
388
```

1.3 Depiction

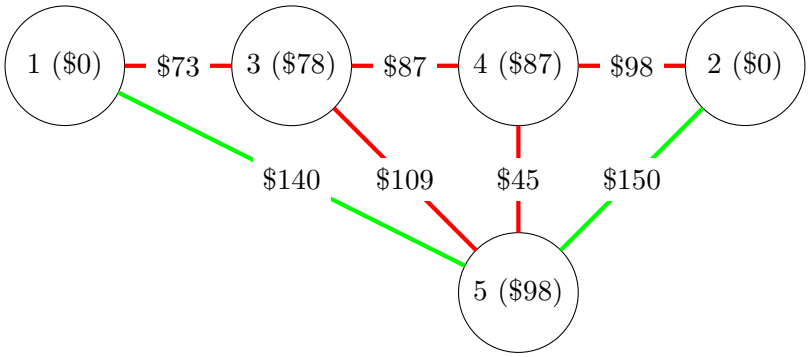

Figure 1: The green path shows the cheapest route from city 1 to city 2. The total cost is $338: $140 + $150 for gas and $98 motel price in city 5.

2 Grading

1. The *correct* implementation (60%)
2. The *successful* execution using our test file(s) (40%)

3 Remarks

1. Richmond is always city 1 and Montgomery is always city 2.
2. There will always be at least one path from city 1 to city 2.
3. If the cost for gas between two cities from A to B is in the input data, the cost from city B to A will be the same and **not** in the input data.
4. The input will **always** be in the correct format, with numbers in the specified ranges, there is no need to check it. We will not be trying to "break" your program with fuzzy input.
5. Do not add text comments, questions, prompts, human related things, or anything else than the answer to standard output.
6. The exact method your program will be tested, how commands are passed, and when and how to print output is demonstrated in the following example. In this example, `student@learning$` indicates a console/command line in the directory of your unzipped source file(s). The file `input.txt` is a file containing the input example in the format described above. The < redirects the file `input.txt` to standard input for your java program, this is equivalent to typing it in.

```
student@learning$ javac *.java
student@learning$ java cmsc401 < input.txt
   388
student@learning$
```

7. Failure to produce output in this exact way may result in point deduction. Please, follow directions.
8. If you have any questions about submissions, clarity, or need help, your TAs are available to help in person or via email at {`vealetm; trinpm`}`@vcu.edu`.