Programming Assignment 2

CSCD320 Algorithms

Instructor: Dr. Bojian Xu
Eastern Washington University, Cheney, Washington

Due: 11:59pm, May 29, 2016 (Sunday)

Please follow these rules strictly:

1. Verbal discussions with classmates are encouraged, but each student must independently write his/her own work, without referring to anybody else's solution.
2. No one should give his/her code to anyone else.
3. The deadline is sharp. Late submissions will **NOT** be accepted (it is set on the Canvas system). Send in whatever you have by the deadline.
4. Every source code file must have the author's name on the top.
5. All source code must be written in Java and commented reasonably well.
6. Sharing any content of this assignment and its keys in any way with anyone who is not in this class of this quarter is NOT permitted.

Implementing Dijkstra's algorithm (100 points)

This programming assignment is to implement Dijkstra's algorithm for single-source shortest paths finding using graph's adjacency list representation and the min-heap data structure.

- You must implement your own min-heap and linked list data structures and CANNOT use Java APIs for those two data structures.
- You must write every line of code by yourself. That is, you CANNOT borrow any code from Internet or elsewhere; otherwise you get zero for this assignment.

The input of your program. The input of your program is a text file named that contains the adjacency list representation of a graph. Suppose below is the graph we want to work with.

$$\begin{tikzcd} 0 \arrow[r, "2"] \arrow[d, "5"'] \arrow[dr, "4"] & 1 \arrow[d, "1"] \\ 2 & 3 \arrow[l, "2"] \end{tikzcd}$$

Then, the content of the input text file representing the above example graph can be:

```
0:1,2;2,5;3,4
1:3,1
2:
3:2,2
```

The first column in the file represent all the vertices. Every line in the file has two parts. The first part before the colon represents one vertex; the second part after the colon is the collection of the next-hop neighbors of the vertex in the first part, along with the weight of the corresponding edges. **You are guaranteed that all the vertices in the graph are named as $0, 1, \ldots, n-1$, where n is the number vertices in the graph.** However, remind that a given graph can also be represented by other different text files. For example, the same example graph above can also be represented by the following two text files.

```
1:3,1                  1:3,1
0:3,4;1,2;2,5          0:2,5;1,2;3,4
2:                     3:2,2
3:2,2                  2:
```

In other words, we do not have any particular requirement on the order of the vertices that are listed before the colon; we also do not have any particular requirement on the order of the next-hop neighbors in a particular line of the file. Your program needs to be able to work with any one of these possible input files, all of which are representing the same graph.

Source code file's name and command line. The Java source file that contains the main function must be named as: `Test_Dijkstra.java`. Your program takes two parameters in the command line. The first parameter is the file name of the input text file that represents the graph. The second parameter is the source vertex that is needed in Dijkstra's algorithm. **Note that your program CANNOT hard-code these two parameters in your source code. Instead, they will be provided by the grader in the command line.** Suppose the file name of the input text file is "`graph.txt`" and we want to calculate the shortest paths and their corresponding shortest distances from the vertex `3` to all the other vertices in the graph, we will type the following command line:

```
$java Test_Dijkstra graph.txt 3
```

(Note that the `$` symbol represents the shell prompt and is NOT part of the command line that you will type.)

The output of your program. Your program will print the shortest paths and their corresponding shortest distances from the given source vertex to all the other vertices in the graph.

Example 1: Suppose the input file "`graph.txt`" represents the above example graph and the source vertex is `3`. After typing the command line:

```
$java Test_Dijkstra graph.txt 3
```

your program should print the following on the screen:

```
[0]unreachable
[1]unreachable
[2]shortest path:(3,2) shortest distance:2
```

meaning: there is no path from vertex `3` to either vertex `0` or vertex `1`; the shortest path from vertex `3` to vertex `2` is `3->2` and its corresponding shortest distance is 2. **Note that we DO require to list all the destination vertices in the ascending order of names in your screen print.**

Example 2: Suppose the input file "`graph.txt`" represents the above example graph and the source vertex is `0`. After typing the command line:

```
$java Test_Dijkstra graph.txt 0
```

your program should print the following on the screen:

```
[1]shortest path:(0,1) shortest distance:2
[2]shortest path:(0,2) shortest distance:5
[3]shortest path:(0,1,3) shortest distance:3
```

Your program may print the following, which is also correct, depending on how you write the code.

```
[1]shortest path:(0,1) shortest distance:2
[2]shortest path:(0,1,3,2) shortest distance:5
[3]shortest path:(0,1,3) shortest distance:3
```

(Note: Your program only needs to have one correct printing.)

A few suggestions. Make a good plan and design for your program and decompose it into a few loosely coupled components. Technically, you first need to design, plan, build, and well test: (1) an independent singly-linked list class; (2) an independent min-heap class. Then, you can start to implement Dijkstra's algorithm using the linked list and min-heap. Consider all possibilities in your implementation to make your program/product robust. Do not work on the next component before the previous one is finished and well tested. You will not succeed in debugging a large program, if it is built upon a collection of buggy components.

Submission

- All your work files must be saved in one folder, named: **firstname_lastname_EWUID_cscd320_prog2**
  (1) We use the underline '_' not the dash '-'.
  (2) All letters are in the lower case including your name's initial letters.
  (3) If you have middle name(s), you don't have to put them into the submission's filename.
  (4) If your name contains the dash symbol '-', you can keep them.
- You then compress the above whole folder into a .zip file.
- Submit .zip file onto the Canvas system by the deadline.