Programming Assignment 5

CSCD320 Algorithms

Eastern Washington University, Spokane, Washington

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.

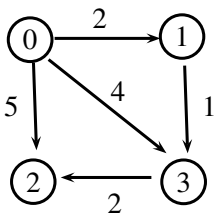
Implementing Dijkstra's algorithm for Single-Source Shortest Paths Finding

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.
- You are allowed to use Java API for linked lists.
- You must write every line of code by yourself.

Specification

- The Java class that has the `main` function must be named as "Dijkstra". That is, the name of the source file that contains the `main` function must be named as "Dijkstra.java".
- **The input of your program:** The command line that runs your program has two parameters as the input to your program. These two input parameters are provided by the grader and you CANNOT hardcode them.
  - The first input parameter is the name of the file that contains the list representation of the graph, for which you want to run Dijkastra's algorithm. For example, suppose below is the input graph:



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 column before the colon 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 of 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
0:3,4;1,2;2,5
2:
3:2,2
```

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

– The second parameter is the source node id, from which you want to find the single-source shortest paths to every one of the vertices in the graph. **You are guaranteed that the given source vertex id is a valid vertex in the given graph.**

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 every one of the vertices in the graph, we will type the following command line:

```
$java Dijkstra graph.txt 3
```

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

- **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 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
```

indicating: 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 their ids 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 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 the graph is represented in the input file and 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. 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.
- Finish a robust component that handles the reading of the input file.
- Finish a robust component that handles the loading/construction of the graph's list representation in the RAM.
- The array in the list representation should be an array of linked list class data type, rather than an array of linked list node class data type. The later choice is a bad design. For example, if you use Java API for linked lists, the adjacency list representation can be an array of array lists.
- Finish a robust min-heap data structure.

- Finish a robust implementation of the Dijkastra's algorithm.
- Design and decide where and how to save various tracking information within each vertex (`v.p` and `v.d`), in order to output the results in the end.
- Finish a robust module that handle the output printing work.
- Consider all possibilities in your implementation to make your program/product robust.

Submission

- All your work files must be saved in one folder, named: **firstname_lastname_EWUID_cscd320_prog5**
  (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.