CSE3080-3: Data Structures (Spring 2020)
Final Project 2: Shortest Paths

Handed out: June 10, Due: June 22, 11:59PM (KST)
No Late Submission for this project

1. Introduction

In this project, we will implement Dijkstra's algorithm, which calculates the shortest paths from a single vertex to all other vertices in a graph. We will write a program that reads a file which contains information on the graph and the source vertex, and outputs the shortest paths from the source vertex to the other vertices.

The time complexity of your implementation could be **O(n^2)** or **O(e log n)**, where n is the number of vertices and e is the number of edges in the graph.

2. Requirements (Read Carefully!)

(1) You should write a single C program, named **fp2**.

(2) The program takes one command-line argument, which is the input file name. For example, the user will run the program like this:

```
./fp2 input.txt
```

The first argument is the name of the input file.

(3) If the number of command-line arguments is not 1 (zero or more than 1), then you should print a usage string on the display and exit the program. The usage string looks like this:

```
usage: ./fp2 input_filename
```

(4) If the input file does not exist, your program should display an error message on the screen and terminate. The error message should be:

```
The input file does not exist.
```

(5) The input file format is like the following. Here we assume the graph is a **directed graph**.

- The first line contains the number of vertices in the graph. We assume that the vertex number **starts from 0**. Thus, if there are 6 vertices in the graph, the vertex numbers are 0 to 5.
- The second line contains the number of edges in the graph.
- From the third line, each line contains three numbers separated by spaces. The first number is the source vertex of the edge, the second number is the destination vertex of the edge, and

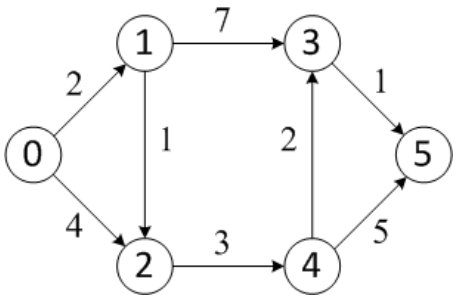
the third number is the weight of the edge.
- The last line contains the source vertex.

An example input file looks like this.

```
6
8
0 1 2
0 2 4
1 2 1
1 3 7
2 4 3
3 5 1
4 3 2
4 5 5
0
```

(You can assume that every weight is an integer larger than 0.)

The corresponding graph for this input will be:



(6) Your program should generate an output file named "fp2_result.txt". The format of the output file is as follows.

- From the first line, each line should print the length of the shortest path from the source vertex to a destination vertex, plus the path itself. For the graph shown above, the lines in the output file should be like this: (The lines should be in the ascending order of destination.)

```
SRC: 0, DST: 0, LENGTH: 0, PATH: 0
SRC: 0, DST: 1, LENGTH: 2, PATH: 0 1
SRC: 0, DST: 2, LENGTH: 3, PATH: 0 1 2
SRC: 0, DST: 3, LENGTH: 8, PATH: 0 1 2 4 3
SRC: 0, DST: 4, LENGTH: 6, PATH: 0 1 2 4
SRC: 0, DST: 5, LENGTH: 9, PATH: 0 1 2 4 3 5
```

- If there is a destination that cannot be reached from the source vertex, you should output the line for that destination as follows:

(For example, let us suppose our starting vertex is vertex 1.)

```
SRC: 1, DST: 0, LENGTH: -, PATH: -
SRC: 1, DST: 1, LENGTH: 0, PATH: 1
SRC: 1, DST: 2, LENGTH: 1, PATH: 1 2
SRC: 1, DST: 3, LENGTH: 6, PATH: 1 2 4 3
SRC: 1, DST: 4, LENGTH: 4, PATH: 1 2 4
SRC: 1, DST: 5, LENGTH: 7, PATH: 1 2 4 3 5
```

(7) At the end of the program, you should print the following lines on the screen.

```
output written to fp2_result.txt.
running time: 0.015364 seconds
```

The actual running time printed will be different on each run. In order to print out the running time, use the function `clock()`.

Once you implement the program, compare the execution time of your program with the given binary file (fp2_example) using the given input file "input_large.txt". For this particular input file, your program should take less than five times the amount of time taken for the example binary file.

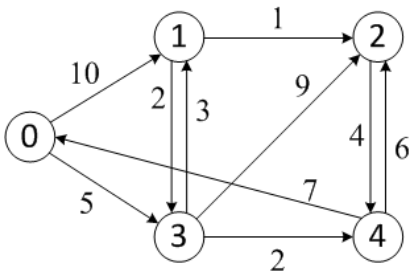
(8) Similar to other homework, you should write a Makefile. The TA will build your code by running "make". It should create a binary file, "fp2".

※ Test-run the given binary code and write your program so that it produces the same result as the given binary code.

※ Your implementation should support up to **10,000 vertices and 100,000,000 edges**.

3. Report

Aside from writing code, this project includes a written report. In the report, you should find the shortest paths from a source vertex to all other vertices in a graph, using the Dijkstra's algorithm. Your graph is the one shown below, and your starting vertex is **vertex 0**.



Your task is, for each step, to draw the i) shortest path tree, and the ii) table which indicates the shortest path length and the previous vertex in the path from the source vertex.

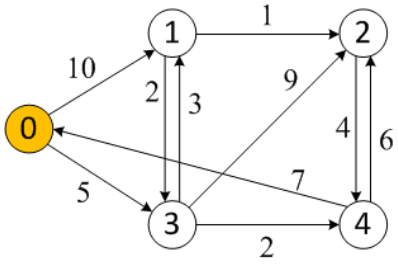
i) For the shortest path tree, you can color the vertices that were chosen in the SPT. Refer to the slides of lecture #23.

ii) In the lecture slides, we only maintained the shortest path length in the table. However, here we also record the "previous vertex" in the path from the source vertex. In terms of shortest path tree rooted at the source vertex, the previous vertex will be the "parent" node.

Here is the result of the 1st step. You can draw the result of the following steps, using the format shown here.

1st step

SPT



Table

vertex	length	parent
0	0	-
1	10	0
2	inf	-
3	5	0
4	inf	-

In the 1st step, the source vertex is included in the SPT. Then, the shortest path length for all other vertices are calculated by relaxation. After the 1st step, we can only reach vertex 1 and 3 because SPT only includes vertex 0. Vertex 1 can be reached via edge <0, 1>, and vertex 3 can be reached via edge <0, 3>. Thus, their lengths are updated in the table. Also, their parent nodes are updated to vertex 0, because it is the previous node in the shortest path.

In your report, copy the 1st step from here, and draw the result of the following steps, until the Dijkstra's algorithm terminates.

- You must write the report by hand. Please remember to write your name on the document. For submission, you should scan your report, make it into a PDF file and submit.

※ If you use pencils or colored pens, it may be hard to recognize the drawings once you scan the document. Make sure your writing is clearly shown.

4. Submission

You should submit your source codes and the Makefile. (You do not need to submit compiled binary files.) For this project, you should also submit the report file (in PDF format.) Combine your files into a zip file named cse3080_fp2_**20190001**.zip. The red numbers should be changed to **your student ID**. Submit your files on the cyber campus.

5. Evaluation Criteria

(1) Your program should compile without problem and produce the binary file.

(2) If the user does not give correct number of arguments, your program should print the usage message on the display and terminate.

(3) If the input file does not exist, your program should correctly print an error message on the display and terminate.

(4) When the user correctly executes the program with a command-line argument, your program should produce the result file and output the message on the screen.

(5) The output must correctly show the shortest path length for each destination vertex. The case where source vertex and destination vertex are the same should be included.

(6) The output must correctly show the shortest path for each destination vertex.

(7) The running time of your program should be less than five times the running time of the given binary file.

6. Grading

The programming part will be counted for 15% of the final grade. (Half of what is assigned to the final exam.)

The report will be counted for 5% of the final grade. (Half of what is assigned to the participation.)

7. Notes

- You must write your own code. You may discuss ideas with other students but must not copy their work. Similarly, you can find ideas on the Internet, but must not copy the code from the sources obtained from the Internet.

- Our department uses a software that detects duplicate codes. Since the software uses an assembly code level detection, just changing the variable names will not bypass the software. Make sure you write your own code from the scratch. Then, you will have no problem.

- Explained earlier, but your program should compile and run on the **cspro** machine. Make sure you check before you submit your work. The TA will download your code, run "make" and execute your binary files to check if they produce correct outputs.

8. Late Policy

Due to the grade submission deadline, no late turn-ins will be accepted for this project.