Project 4: Shortest Path

Submission deadline: 9pm on 12/19, 2022
(9% of the total grade)

Important: we will not accept any late submission for the final project. You will receive ZERO point for late submission. No penalty-free late days apply to this project.

> Why? The deadline of the final project is very close to the deadline for grade reporting by which all projects must be finished with grading. If we allow any late submission, we won't have time for grading to submit your grades to school in time. So, schedule your time wisely so that you can submit yours in time.

Implement the Bellman-Ford and Floyd-Warshall algorithms. Both programs should take input from stdin: the first line of the input consists of two numbers - #of vertices (n) and # of edges (m). Each line after the first line consists of three numbers – vertex id1 (1..n), vertex id2 (1..n), and an edge-weight. The edge-weight can be any integer number (positive, 0, or negative) and you can assume that the input is a simple directed graph that does not have any self-loop.

A. (Credit: 4% of total grade) Implement the Bellman-Ford algorithm, and name the program as 'bf'. bf can take one optional argument, source vertex id. When the source vertex id, s , is provided, then you need to compute the shortest path from s to all other vertices, and output the shortest path and path length per each destination vertex in each line. For example,

```
$ ./bf 3 < graph1
3 1 length: 4 path: 3 2 1
3 2 length: -1 path: 3 2
```

For each line, it should print out the output in the following format in the increasing order of the destination vertex.

source-vertex dest-vertex length: XX path: source-vertex … dest-vertex

```
$ ./bf 2 < graph1
2 1 length: 5 path: 2 1
2 3 length: inf path: none
```

In case the commandline argument is missing, bf prints out the output for every source vertex in the increasing order of the source vertex. If there's no path from a source to a dest vertices, the length should be "inf", and the path should be "none". If there's a negative-cost cycle, it should print out "Error: negative-cost cycle is found".

```
$ ./bf < graph1
1 2 length: inf path: none
```

1 3 length: inf path: none
2 1 length: 5 path: 2 1
2 3 length: inf path: none
3 1 length: 4 path: 3 2 1
3 2 length: -1 path: 3 2

B. (Credit: 5% of total grade) Implement the Floyd-Warshall algorithm and name the program as 'fw'. 'fw' does not take any argument and computes all pairs shortest paths (APSP) and print out the output. The output format should be the same as 'bf' when it's run without a commandline argument. Like bf, it should print out "Error: negative-cost cycle is found" if there's a negative-cost cycle.

$ ./fw < file1
1 2 length: inf path: none
1 3 length: inf path: none
2 1 length: 5 path: 2 1
2 3 length: inf path: none
3 1 length: 4 path: 3 2 1
3 2 length: -1 path: 3 2

Requirements & Information:

- Write the code in C – your code should compile with gcc installed on eelab5 or eelab6
- You can assume that the any path length does not go beyond the range of C's signed integer type.
- We provide a few graph data files for testing. Please see the attached files.
- You can use standard C runtime libraries but you cannot use any library that parses or processes the graph data.
- For both Bellman-ford and Floyd-Warshall algorithms, implement the space optimization.
- For performance comparison, you can measure the timing with 'time' and avoid writing the output to stdout by redirecting stdout to /dev/null. This would save the time for printing out the output.

  $ time bf < graph1 > /dev/null
  real 0m5.545s
  user 0m5.488s
  sys 0m0.016s

- Name the source code files as bf.c and fw.c. You can use more source code files than these two if necessary.
- Print a reasonable error message for a wrong input format or a wrong argument and stop the program.

Collaboration policy:

- You can discuss algorithms and implementation strategies with other students, but you should write the code on your own. That is, you should NOT look at anyone else' code.

Submission:

- You need to submit all source code files, "Makefile", and "readme" in a tarred gzipped file with the name as YourID.tar.gz. If your student ID is 20211234, then 20211234.tar.gz should be the file name.
- 'make' should build the binary, 'bf' and 'fw'.
- You will get **ZERO** point if the source code doesn't compile (due to compiling errors). Make sure that your code compiles well before submission.
- Suppress all warnings at compiling by turning on the "-W -Wall" options in the CFLAGS in Makefile. You will get **some penalty** if gcc produces *any* warnings.
- "readme" (a text file with "readme" as the file name) should describe how you implement the code. (1) Explain the data structure you chose to implement as well as the overall algorithm. (2) Measure the performance of 'bf' and 'fw' for all-pairs-shortest-path for large graphs. You can use the input graphs that are provided or can test with graphs of your own.

Grading:

- We will evaluate the correctness of your program.
- We will give penalty to the program that is way too slow – if you chose the right data structure/implementation/algorithm, it should work fine.
- We will evaluate your readme file.
- We will evaluate the clarity of your code – coding style, comments, etc.