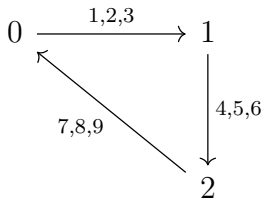
ECE 36800
PA 3: Shortest Paths in Dynamic Graphs

Due Wednesday, July 31th at 11:59 PM

Goal.

Consider the following scenario: you want to find an optimal route for your commute to work while considering factors like traffic, which may change over time. We can model this as a graph where each edge has a *list of weights*, such that each element of the list reflects the weight of that edge after a certain number of steps. For instance, consider the path given by $0 \to 1 \to 2 \to 0$ on the graph below:



The weight of the $0 \to 1$ edge is initially 1. After taking the first step, the weight of the $1 \to 2$ edge is 5, and after the second step, the weight of the $2 \to 0$ edge is 9, so the total weight of the path is $1 + 5 + 9 = 15$.

For simplicity, we only consider graphs where the lists of weights for each edge all have the same length, called the *period*. In general, if an edge $u \to v$ shows up at position i in the path and has a weight list of $(x_1, \ldots, x_n)$, it contributes $x_{i \bmod n}$ to the weight of that path. The goal of this assignment is to write a program which, given a graph like the one above, and a pair of start/end vertices, outputs a path from start to end with *minimum weight*.

Input/Output

As before, your program should take as a command-line argument the name of a text file describing a graph, and then respond to queries on standard input. For each query, you should print to standard output the shortest path as a space-separated list of vertices.

The format of the text file is as follows:

- The first line will look like V N, where V is the number of vertices in the graph, and N is the period of the edge weights.
- Following that, there will be one line per edge, in the format

  $$v_s \ v_t \ w_1 \ \ldots \ w_N$$

  where v_s is an integer representing the source vertex of the edge, v_t is an integer representing the target vertex of the edge, and each of w_1 through w_N are integers representing the list of weights for that edge.

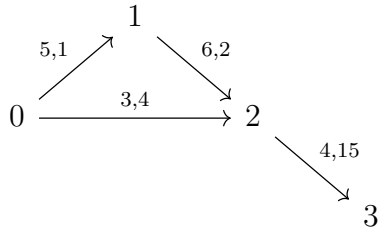
Each query will be given by a space-separated pair of integers, representing the indices of the start and end vertices, respectively.

Example

Consider the following `graph.txt` file:

```
4 2
0 1 5 1
1 2 6 2
0 2 3 4
2 3 4 15
```

This represents the graph:



A sample run of the program is shown below:

```
$ ./pa3 graph.txt
> 0 3
0 1 2 3
```

where `0 3` is the query input on stdin and `0 1 2 3` is the shortest path. Explanation: the shortest path from 0 to 3 is $0 \to 1 \to 2 \to 3$ with a total weight of $5 + 2 + 4 = 11$, since the other path $0 \to 2 \to 3$ has a total weight of $3 + 15 = 18$.

There will be multiple queries for each graph, but you may assume that all queries with the same start vertex will be given consecutively.

Grading

Similar to the previous programming assignment, your submission must be both correct and efficient (and not have any memory leaks/errors, as usual) to receive full points. The graphs will range in size from 10 vertices to roughly 1k vertices, and each graph will have approximately 10 queries. For each test case, your program must produce an output within the time limit to receive credit.

Submission Instructions.

Submit any source/header files with your implementation, as well as a Makefile that builds a target called `pa3`, to Gradescope. Note that to receive points, your submission must work on `eceprog`. (See the syllabus for more details.)
If you choose to do the optional writeup, submit it to the assignment on Brightspace. The writeup can be formatted however you want, but here's a guideline of some questions you may want to think about when writing it:

- What data structures did you choose to use, and why? Are there different choices that would have made a more efficient solution?

- What other factors did you consider? How did they influence your final implementation?
- What, if anything, did you find most challenging about this project? Was there anything in particular you enjoyed/disliked?
- If you took a particularly creative approach to solving this problem, or ended up implementing something we didn't ask for, tell me about it!