CS473/573 Algorithms I
Programming Assignment #2

Dijkstra's algorithm is a greedy algorithm for solving single-source shortest-paths problems on a graph in which all edge weights are non-negative. Study the introductory section and Dijkstra's algorithm section in the Single-Source Shortest Paths chapter from your book to get a better understanding of the algorithm. In this assignment, you are going to implement the Dijkstra's algorithm using binary heap-based implementation of priority queues to solve a network reliability problem.

a. On your first day of work as a network reliability engineer, your manager wants you to solve the following problem:

You are given a directed graph $G = (V, E)$ on which each edge $(u, v) \in E$ has an associated value $w(u, v)$, which is a real number in the range $0 \leq w(u, v) \leq 1$ that represents the weakness of a communication channel from vertex u to vertex v. The weakness of a path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is defined as the weight sum of edges it contains, $w(v_0, v_1) + w(v_1, v_2) + \ldots + w(v_{k-1}, v_k)$. You are asked to write a **C** program to compute weakness values of the least weak paths from a given source vertex to all other vertices.

After some thinking, you realize it is a shortest-path problem and can be solved with the Dijkstra's algorithm using a min-priority queue. You need to implement min-priority queue with a binary min-heap and your priority queue implementation should support at least `insert`, `extract-min` and `decrease-key` operations.

b. After solving the problem and presenting your work, the manager tells you that there is a mistake in the previous problem definition. The weight values actually represent probabilities indicating the reliability of channels. The problem becomes as follows:

You are given a directed graph $G = (V, E)$. Each edge $(u, v) \in E$ has an associated value $w(u, v)$, which is a real number in the range $0 \leq w(u, v) \leq 1$ that represents the reliability of the communication channel from vertex u to vertex v. You may interpret $w(u, v)$ as the probability that communication over the channel from u to v will succeed, and you may assume that these probabilities are independent. You are asked to write a **C** program to compute success probabilities of most reliable paths between a given source vertex and all other vertices. The success probability of a path is the probability that the message will be delivered successfully over every channel in the path.

You think it is very similar to the previous problem. Here, the score of a path is the multiplication of weights instead of the summation of them. Modify your algorithm in part

(a) to solve the problem. You need to use a max-priority queue in this part and you need to tweak `RELAX` routine to address the change in the score calculation. You need to implement max-priority queue with a binary max-heap and your priority queue implementation should support at least `insert`, `extract-max` and `increase-key` operations.

c. After finding the correct solution to the correct problem, the manager says you didn't need to change the code, you could use your code in part (a) directly to find the most reliable paths described in part (b). The managers tells you to calculate new weight values $w'(u, v)$ such that $w'(u, v) = -\log_2 w(u, v)$. Run your code in part (a) with new weight values to find shortest path-weight $\delta'(s, v)$ from source vertex s to all other vertices. Then, compute the solution of the problem $\delta(s, v)$ by using $\delta(s, v) = 2^{-\delta'(s,v)}$.

Think about why this method should give the same solution. Apply the mentioned process and confirm that the solution you found here is the same as the solution you get in part (b). That is:

- Write a script to convert input graph file to a new graph file in which weights are calculated as $w'(u, v) = -\log_2 w(u, v)$.
- Run the executable you compiled in part (a) on this new graph file.
- Convert the resulting values as $\delta(s, v) = 2^{-\delta'(s,v)}$
- Confirm part (b) solution and part (c) solution are the same.

Input and output

- Use GNU Make tool to automate the compilation process. When we type make, two executable files should be generated: A for part (a) (**not a nor A.exe**) and B for part (b).
- Your programs should read the graph from an input file in the Matrix Market (.mtx) format. The name of the input file will be given as a command-line argument.

  The Matrix Market (.mtx) has the following format for storing a directed, weighted graph:

```
% some comment lines
50 50 230 % #Vertices #Vertices #Edges
1 3 0.5 % SRC DST WEIGHT
3 5 0.4
...
```

  You may assume that all vertex indices are between 1 and the number of total vertices.
- Your programs should produce output files (a.txt and b.txt) where line i shows the solution distance from vertex 1 to vertex i. If the distance is not valid (there is no path from 1 to i), output -1 instead.

```
0.0 % Distance from vertex 1 to vertex 1
0.8 % Distance from vertex 1 to vertex 2
0.6 % Distance from vertex 1 to vertex 3
-1 % It means no path from vertex 1 to vertex 4
0.7
...
```

- Therefore, the steps for compiling and running will be like:

```
> make                  # This should generate a binary executable files called A and B
> ./A input_graph.mtx       # This should produce an output file called a.txt
> ./B input_graph.mtx       # This should produce an output file called b.txt
```

- You are given three sample inputs (tiny.mtx, small.mtx and medium.mtx) to test and debug your code. We suggest you to use other inputs as well to confirm the correctness of your code. We may use additional test cases while grading.

Submission rules and guidelines

- You will submit your code and report in one zipped file, called STUDENT_ID.zip, which contains the following:
  - Your source code files (.c and .h files) for part (a) and part (b).
  - Makefile to be used to compile your program.
  - A report file (at most 1 page) in pdf format. In the report, describe your approach for part (a). Mention the changes you made in your code to solve part (b). Also, explain why the steps shown in part (c) should give the same solution.
  - [Optional] README.txt file that contains any (potential) technical issues.
  - Nothing else!, no graph input files nor IDE-related configuration files.

  by **December 28th, 2020 23:59** as an email to `selcuk.gulcan@bilkent.edu.tr`.
  The subject of the email should be: `CS473_F20_HW2_submission`

- Before submission, make sure that your code can be compiled on a Unix-like environment using `gcc`. If you use Windows OS, you can either use Cygwin or the newly added windows subsystem for Linux (WSL) feature of Windows 10.

- We'll use a black-box grading policy. That is, a script will be used to compile and run your code against a number of test cases. Your score will be determined mainly by whether your code produces correct results or not. Therefore, be extra careful about the format of input and output. You may get 0 if the format of your output file is wrong even if your solution is correct.

- You need to use min-heap in part (a) and max-heap in part (b). Not doing so may result in a penalty in your assignment grade.