CS 32

Machine Problem 2

Deadline: 04 May 2018, 11:59PM

General Instructions

- For the machine problem, you are to write your implementation using the **C programming language**. You must use **gcc (version 4 or later)** as a reference compiler.
- The name assigned to the source code file that will contain the main method (assuming multiple source code files will be implemented) is *apsp.c*.
- Within the source code, place a comment at the very beginning stating your student number and your full name.
- In the event that multiple source code files were created, place them all in a compressed folder named *apsp.zip*.
- If you have consulted references (books, journal articles, online materials, other people), cite them as references by adding a comment detailing the references after your name. Use the MLA format for listing references: http://www.sciencebuddies.org/science-fair-projects/project_mla_format_examples.shtml. (For lines requiring italics, just place them inside quotation marks.)
- Add a comment before each function/method in your code, save for the main method, detailing the following: 1) input (if any) 2) output (if any) and 3) what the method does.
- You are only to submit the source code of your implementation. Submission of the machine problem should be done via e-mail. Attach the source code file/s, and write as the subject header of the e-mail: [CS 32] $<$ *Last Name, First Name* $>$ − Machine Problem 2. For example, [CS 32] Kreuβ, Jonathan - Machine Problem 2. Send your answers to jcyap@dcs.upd.edu.ph. **Submissions made after 11:59pm of 04 May 2018, as per the timestamp of the received e-mail, will not be accepted**. Take note of this as this is directly tied to the "No MP, No Exam" rule.
- Together with your MP submission, **send a screenshot of your Command Prompt or Terminal after you have entered the following commands (and pressed Enter): gcc -v**. This will let me know the set-up of your GCC compiler in the event that I have technical problems with compiling and/or running your submission.
- **You should receive a confirmation e-mail from me stating receipt of your deliverable within 24 hours upon your submission of the MP**. If you have not received any, forward your previous submission using the same subject header once more.
- If your *program does not compile and/or does not run properly for at least one of the test inputs even after minor edits to the source code*, then your **submission is deemed invalid and will not be accepted**. You will be asked to submit a working deliverable. **This provision will also hold for subsequent (repeat) submissions**. **Date of submission is considered to be the date when an acceptable deliverable was submitted.**
- If you have any questions regarding an item (EXCEPT the answer and solution) in the machine problem, do not hesitate to e-mail me to ask them. However, **questions regarding this MP forwarded/received on or after 12:01am of 01 May 2018 will NOT be entertained**.
- Follow the instructions as mentioned above, especially those pertaining to actual submission of the deliverable. **Lack of or erroneous implementation of any part of the instructions may be considered a deficient submission, and thus may not be accepted**.

All-Pairs Shortest Path Algorithms: Iterated Dijkstra and Floyd Algorithms

The iterated Dijkstra algorithm and the Floyd algorithm are both techniques for solving the all-pairs-shortest-path (APSP) algorithms. The iterated Dijkstra algorithm is just an extension of the basic version of the Dijkstra algorithm for single-source-shortest path problems. This time, the algorithm is iterated several times to have it get all pairs of shortest paths instead of just one single source. The idea for the iterated Dijkstra algorithm is that you run the Dijkstra algorithm for each vertex in the graph. For the algorithm versions to follow, consult Chapter 10.2 of the book Data Structures by EP Quiwa as reference for the implementation of the Dijkstra and Floyd algorithms.

Implementation Specfications

For the implementation part, the input should be a text file with space and new line separated values representing the weighted adjacency graph. For example, we have a 2 x 2 matrix stored as:

0 2
1 0

There should be no restrictions as to the number of rows and columns (except of course, that the rows and columns be equal). The label of the vertices would be integers scaling from 1, 2... and so on. In case of disjoint vertices (e.g. vertex 1 does not connect to vertex 2), mark that with an x. For example:

0 x
1 0

The input above denotes that vertex 2 connects to vertex 1 but not vice versa. A .zip file containing text files representing adjacency matrices is posted in the web page for you to test out your implementation.

Your program must then compute all pairs of shortest paths via the two algorithms mentioned. A required specification of your program should be that *the Dijkstra and Floyd algorithms described in the aforementioned reference must strictly be followed, right down to the data structures involved in the implementation.* (NOTE: You might have to do some backreading to understand the implementation of the data structures used in the algorithms.) Your program should then output the path one must take per vertex as source that would result in the shortest path (plus the cost) per algorithm.

It is possible that a path *may not* exist between two vertices in a single graph. In such case, you must print a message stating that no path exists between the two vertices.

Sample Run

The following are the contents of a file named *graph0.txt*:

0 3 1
1 0 4
1 2 0

```
Program started.

Enter file name> graph0.txt

Iterated Dijkstra:
vertex 1 to vertex 2: 1 -> 2 (cost = 3)
vertex 1 to vertex 3: 1 -> 3 (cost = 1)
vertex 2 to vertex 1: 2 -> 1 (cost = 1)
vertex 2 to vertex 3: 2 -> 1 -> 3 (cost = 2)
vertex 3 to vertex 1: 3 -> 1  (cost = 1)
vertex 3 to vertex 2: 3 -> 2  (cost = 2)

Floyd:
vertex 1 to vertex 2: 1 -> 2 (cost = 3)
vertex 1 to vertex 3: 1 -> 3 (cost = 1)
vertex 2 to vertex 1: 2 -> 1 (cost = 1)
vertex 2 to vertex 3: 2 -> 1 -> 3 (cost = 2)
vertex 3 to vertex 1: 3 -> 1  (cost = 1)
vertex 3 to vertex 2: 3 -> 2  (cost = 2)
```

Criteria for Grading

The machine problem submission will be graded via the following criteria:

- Program effectivity (50%)
  The program should be able to properly process erroneous inputs and be able to perform the algorithm according to specification on a number of test cases.
- Adherence to specification (30%)
  The implementation should, to the fullest extent possible, adhere to the specifications set for the machine problem.
- Program documentation and readability (20%)
  The program should have proper internal documentation (i.e. comments detailing functionalities), and that the documentation details the mechanisms of each function and some code blocks clearly. The code should also be readable in the sense that function and variable names are as intuitive as possible, and that proper indentations/blocking of statements is observed (e.g. statements on the same level/block should be properly aligned, statements inside a *for* loop should be indented, etc.).