Project 3: Process Scheduling v2

CMPE 250, Data Structures and Algorithms, Fall 2016

Instructor: H. L. Akın, A. T. Cemgil
TA: Çağatay Yıldız
SA: Rıza Özçelik

Due: 20.11.2016, 23.55

Introduction

'What? Processors? Again?'. Well, I can hear you saying these kind of stuff already when you will have reached the end of the description - nothing more offensive I hope:). But don't worry, this will be a much shorter description.

Description

In this project **you are going to calculate the minimum time for a computer with infinitely many processors to complete a set of processes.** This time no task scheduling is needed, so processes can be executed simultaneously. Plus, processes depend on each other. For example, you can't process P1 before completing P2, if P1 depends of P2.

The goal of this project is you to learn implementing graphs in C++ and code a couple of graph algorithms. So you are expected to represent the system as a graph and work on it to make your life easier in following projects.

Input & Output

Input:

As usual you are going to be provided an input file as input which will be in the following format:

- In the first line, two integers V and E, which are the number of processes and dependencies, respectively. Note that process id's are in $\{0,1,\ldots V\text{-}1\}$
- In the following V lines one double per line which are time needed to complete each process, ordered by process id.
- In the following E lines two integers per line that represent the dependencies. Note that **the second one depends on the first one**.

Output:

As output you are expected to print just one double which is the minimum time required for server to complete all the processes.

An example is as follows:

Input File	Output File
8 9	29.5
5.1	
2.2	
6.3	
8.4	
3.5	
7.6	
12.7	
4.8	
0 1	
1 2	
2 3	
2 4	
3 7	
4 5	
4 7	
5 7	
6 5	

Table 1: Sample Input and Output files

Warnings

Consider a simple case in which there are two processes and both of them depend on the other (This corresponds to a graph with two vertices and two edges, and neither of the edges is a self-loop). In this case, and in cases where the graph contains a cycle, there is no legitimate order of processes and your output must be -1.

Before implementing, think about the solution and its complexity. This is vital as some of the input files are really huge and inefficient algorithms will be very likely to timeout in testing.

Test cases are here. Download the test cases but please do not add those to your git repo as file sizes are too big. Send an e-mail to cagatay.yildiz1@gmail.com if test cases are missing.

Submission Details

You are supposed to use the Git system provided to you for all projects. No other type of submission will be accepted. Also pay attention to the following points:

- Your code must be compiled by the simple `cmake CMakeLists.txt` and `make` commands and your code must read the arguments from the command line. Your code will be tested with the command:

  ```
  ./project3 inputFile outputFile
  ```

  Command line arguments, namely `inputFile` and `outputFile`, must be absolute paths (such as **/home/student/Downloads/testcases/testcase1.txt**) rather than just file names

- All source codes are checked automatically for similarity with other submissions and exercises from previous years. Make sure you write and submit your own code.

- Your program will be graded based the correctness of your output and the clarity of the source code. Correctness of your output will be tested automatically so make sure you stick with the format described above.

- There are several issues that makes a code piece 'quality'. In our case, you are expected to use C++ as powerful, steady and flexible as possible. Use mechanisms that affects these issues positively.

- Make sure you document your code with necessary inline comments, and use meaningful variable names. Do not over-comment, or make your variable names unnecessarily long.

- Try to write as efficient (both in terms of space and time) as possible.