CS 180 Programming Assignment

Due Date: May 13, 2014 at 10:00AM.
Out: April 29, 2014

Description: Consider the following problem faced by your instructor when preparing lectures. I have n topics that I wish to cover in this class, and these must be presented in order (that is, topic 1 must be before topic 2, and so on). Topic i is projected to take t_i minutes to present in lecture (including questions and other important considerations!), and these projections may be assumed to be accurate. Each lecture has M minutes available, and cannot run over, and each topic must be presented in its entirety in a lecture: one topic cannot be split across lectures. Each lecture topic t_i is at most M minutes long. I can schedule as many or as few lectures as I wish (for purposes of this problem, I am not obligated to fill an entire quarter's worth of classes, or I can run the class across multiple quarters if necessary).

Furthermore, students generally want their instructor to use as much of the lecture as possible: after all, if they've taken the time and energy to come to class, they expect to learn as much as possible, although they'll forgive if the "excess" time is 5 minutes or less. In particular, the students' dissatisfaction can be quantified for any particular lecture: 0 dissatisfaction if at least $M - 5$ minutes are used, and $(M - 5 - x)^4$ dissatisfaction if $x < M - 5$ minutes are used for the lecture. The exception to this is the last lecture: students don't care how much time is left at the end of the final lecture[1].

Input: Your program will take a single command-line parameter, which will contain the name of a file that will be in the same directory as your executable.

The file will be in the following format:

- The first line contains a single integer, the value M, the maximum length of a single lecture.
- The second line contains a single integer, the value n, the number of topics to be covered.
- The next n lines contain the values for t_i; in particular, line $i + 2$ contains the integer t_i.

You may assume that the file is in the appropriate format ; that is, there are a total of $n + 2$ lines, as described, and that each value is a positive integer. You may also (correctly) assume that each t_i value is at most M. We assume that you know how to read input by this point in your academic career, and we're interested in testing your ability to design and implement an algorithm. None of our test cases will be improperly formed files, and you may do whatever you would like to do if one happens.

[1]More formally, the lecture that covers topic n contributes zero dissatisfaction by the students, regardless of time used

Output: After reading the input, you will need to determine the optimal partitioning of lecture topics in such a way as to minimize the total dissatisfaction. If the optimal partitioning of the topics is for l lectures, you will output a total of $l + 2$ lines, as follows:

- Your first line of output is a single integer l, the number of lectures.
- The next l lines will be of the form `x-y`, where x and y are integers in the range $[1, n]$ with $x \leq y$. Line $i + 1$ of your total output will contain the set of topics to cover in lecture i. If that lecture covers only one topic z, your output will be `z-z`.
- The final line of output will be the total student dissatisfaction.

Report: You will also submit a short report. In this report, give a brief description of the algorithm you used to solve this problem, similar to what you would do if the section "description" were a homework problem instead. Briefly explain why it is correct and what its asymptotic running time is (you do not need to explicitly describe the iterative algorithm). Describe also any problems you encountered along the way in the implementation.

Submission Requirements and Grading Criteria

- Any code you submit must be an original piece of code. All ideas must come from the textbook, assignment sheet, or course staff. As students taking an upper-division course, we expect you to understand and adhere to UCLA's standards of academic honesty. Course staff may compare all submitted code this quarter to code submitted by other students this quarter and to code submitted in previous quarters by using an automated plagiarism detection tool. Any instances of academic dishonesty will be reported according to university and department policies. High level discussion of concepts is acceptable, but sharing of code is not.
- Your program **must not** produce output except where specified. Excess output will be deemed an incorrect run to a test case, and partial credit will *not* be given in such cases.
- Your program **must** read the input file from a command-line parameter; you will not read anything from standard in.
- Your algorithm must be reasonably efficient; while it does not have to have optimal runtime, we will not allow a program to run for more than three minutes total; this will be far more than enough time for a reasonably efficient algorithm for the test cases we use.
- Your algorithm must use dynamic programming.
- Submit your code file via CourseWeb. Be sure to give yourself enough time to be sure you can turn something in, because we will not accept excuses like "My network connection at home was down, and I didn't have a way to copy my files and bring them to a SEASnet machine."
- Submit your report either at the start of lecture or in the homework dropbox prior to 10:00 AM on the due date. Your report must not be more than a single piece of paper, front and back.

- Your program will be scored on visible results, not effort. A program's correctness score will be closely correlated with its correctness, not with the amount of time you spent working on it. If you're working on your program on your own computer, make sure to backup to some external drive or file server so that you can survive misfortune like a stolen computer or a hard drive failure (which seems to happen to someone every quarter).
- Your program will be graded on `lnxsrv.seas.ucla.edu` using a shell script. Whether you do your project in C++, Java, or Python, it **must** compile and run correctly on this server.
- Your program must not rely on undefined behavior. We will *not* entertain any regrade requests of the form "it works on my computer," nor will we use your computer to grade the program.
- If you choose to do your project in C++:
  - Your submission must consist of a single file and must be named `lectures.cpp`.
  - You may use any built-in C++ STL data structure, so long as it does not make this project trivial. For example, a class that represents a priority queue is acceptable, while a utility library that solves the problem itself would not be.
  - Your program will be compiled on `lnxsrv.seas.ucla.edu` using a shell script and g++ (gcc version 4.4.7) with the following command: `g++ lectures.cpp -o lectures`. If the compilation fails without producing an executable, your program will receive a zero.
- If you choose to do your project in Java:
  - Your submission must consist of a single file and must be named `lectures.java`.
  - You may use any built-in Java data structure, so long as it does not make this project trivial. For example, a class that represents a priority queue is acceptable, while a utility library that solve the problem itself would not be.
  - Your program will be compiled on `lnxsrv.seas.ucla.edu` using a shell script and the Java compiler (version 1.6.0_45) with the following command: `javac lectures.java`.[2] If the compilation fails without producing a class file, your program will receive a zero.
- If you choose to do your project in Python:
  - Your submission must consist of a single file and must be named `lectures.py`.
  - You may use any built-in standard data structure, so long as it does not make this project trivial. For example, a class that represents a priority queue is acceptable, while a utility library that solve the problem itself would not be.
  - Your program will be run on `lnxsrv.seas.ucla.edu` using a shell script and the Python interpreter (version 2.6.6) with the following command: `python lectures.py <file>`. If the interpreter chokes for any reason, your program will receive a zero.

[2]You may need to set `JAVA_TOOL_OPTIONS=-Xmx2048m` in your environment.

Sample Input and Output

Sample Input

```
100
4
90
10
80
25
```

Sample Output

```
3
1-1
2-3
4-4
1250
```