Computer Science CSC373H
St. George Campus

February 1, 2021
University of Toronto

Homework Assignment #3

Due: February 17, 2021, by 11:00 am

- **You must submit your assignment through the Crowdmark system.** You will receive by email an invitation through which you can submit your work. If you haven't used Crowdmark before, give yourself plenty of time to figure it out!
- You must submit a **separate** PDF document with for **each** question of the assignment.
- To work with one or two partners, you and your partner(s) must form a **group** on Crowdmark (one submission only per group). We allow groups of up to three students, submissions by groups of more than three students will not be graded.
- The PDF file that you submit for each question must be typeset (**not** handwritten) and clearly legible. To this end, we encourage you to learn and use the LaTeX typesetting system, which is designed to produce high-quality documents that contain mathematical notation. You can use other typesetting systems if you prefer, but handwritten documents are not accepted.
- If this assignment is submitted by a group of two or three students, for each assignment question the PDF file that you submit should contain:
  1. The name(s) of the student(s) who *wrote* the solution to this question, and
  2. The name(s) of the student(s) who *read* this solution to verify its clarity and correctness.
- By virtue of submitting this assignment you (and your partners, if you have any) acknowledge that you are aware of the homework collaboration policy for this course, as stated in: `http://www.cs.toronto.edu/~sam/teaching/373/#HomeworkCollaboration`.
- For any question, you may use data structures and algorithms previously described in class, or in prerequisites of this course, without describing them. You may also use any result that we covered in class (in lectures or tutorials) by referring to it.
- Unless we explicitly state otherwise, you should justify your answers. Your paper will be graded based on the correctness and efficiency of your answers, and the clarity, precision, and conciseness of your presentation.
- The total length of your pdf submission should be no more than 4.5 pages long in a 11pt font.

Solutions of Dynamic Programming Problems. The following problems can all be solved by a dynamic programming approach. In writing up dynamic programming algorithms, all the guidelines from earlier in the course apply; but there are some additional things worth keeping in mind as well. If your solution consists of a dynamic programming algorithm, you must clearly specify the set of sub-problems you are using, and the recurrence you are using — describing what they mean in English as well as any notation you define. You must also explain why your recurrence leads to the correct solution of the sub-problems — this is the heart of a correctness proof for a dynamic programming algorithm. Finally, you should give the complete pseudo-code of the algorithm that makes use of the recurrence and sub-problems. A solution consisting of only pseudo-code without these explanations is not a valid answer.

Question 1. (12 marks) Suppose it's nearing the end of the semester and you are taking n courses, each with a final project that still has to be done. Each project will be graded on the following scale: it will be assigned an integer number on a scale of 1 to $g > 1$, higher numbers being better grades. Your goal, of course, is to maximize your average grade on the n projects.

Now, you have a total of $H > n$ hours in which to work on the n projects cumulatively, and you want to decide how to divide up this time. For simplicity, assume that H is a positive integer, and you will spend an integer number of hours on each project. So as to figure out how best to divide up your time, you have come up with a set of n functions $\{f_i : i = 1, 2, \ldots, n\}$ (rough estimates, of course :-) for your n courses: if you spend $h \leq H$ hours on the project for course i, you will get a grade of $f_i(h)$ on this project. You may assume that the functions f_i are non-decreasing: if $h < h'$ then $f_i(h) \leq f_i(h')$.

So the problem is: given the functions $\{f_i : i = 1, 2, \ldots, n\}$, find how many hours to spend on each project (in integer values only) so that your average grade, as computed according to the f_i's, is as high as possible. You must solve this problem with a dynamic programming algorithm whose running time is polynomial in n, g, and H (i.e., none of these quantities should appear as an exponent in your algorithm's running time).

a. (8 marks) Define the sub-problems that you use to solve the problem. Give the recursive formula to compute a sub-problem from smaller subproblems. Based on this formula, give the pseudo-code of a dynamic programming algorithm that finds the best achievable (average) grade. Give the algorithm's running time and justify it.

b. (4 marks) Augment your algorithm so that, in addition to the optimal grade, it also outputs the corresponding optimal division of time: for each course i, it should compute *hours(i)*, the number of hours that you should spend on course i to obtain that grade. Give the pseudocode.

Question 2. (12 marks) The *train car connecting problem* is defined as follows. There are n disconnected train cars sitting in order on a single track. The weights of the train cars are $w_1, w_2, \ldots, w_n$. The problem is to figure out the best order to connect all the train cars together into a single long train, under the following conditions.

Each time two train cars are joined, it is necessary to move one car or the other. Typically, one or both of these cars will already be at the end of a segment of several other cars, so that to move that car, *all* the cars of this segment must be moved. The cost of moving a segment of one or more cars is proportional to the square root of the total weight of the cars in that segment. In other words, suppose we want to join car r to car $r+1$. Suppose cars $q, q+1, \ldots, r-1, r$ are already connected, and cars $r+1, r+2, \ldots, s-1, s$ are also already connected. Then the cost of joining r to $r+1$ is:

$$min\big((w_q + w_{q+1} + \cdots + w_r)^{1/2}, (w_{r+1} + w_{r+2} + \cdots + w_s)^{1/2}\big)$$

For example, consider the six-car sequence whose weights are 4, 3, 5, 6, 2, 6. We might do the junctions in this order: car 1 – car 2; car 3 – car 4; car 5 – car 6; car 2 – car 3; car 4 – car 5. Then the cost of this way of connecting all the cars is: $3^{1/2} + 5^{1/2} + 2^{1/2} + (4+3)^{1/2} + (2+6)^{1/2}$.

So the problem is: given the weights of the train cars $w_1, w_2, \ldots, w_n$, find an order for carrying out the train car connections that minimizes the total cost of train car moves. You must solve this problem with a dynamic programming algorithm whose running time is polynomial in n.

a. (8 marks) Define the sub-problems that you use to solve the problem. Give the recursive formula to compute a sub-problem from smaller subproblems. Based on this formula, give the pseudo-code of a dynamic programming algorithm that finds the minimum total cost of train car moves. Give the algorithm's running time (under the simplifying assumption that each square root operation takes constant time) and justify it.

b. (4 marks) Explain how to augment your algorithm so that, in addition to the optimal cost, it also outputs an optimal order of train car connections. Give the pseudocode.

Question 3. (12 marks) You are consulting for people whose job is to monitor and analyze electronic transmissions coming from ships. They ask you to derive a fast algorithm to detect whether a message transmission they are hearing is the just the ***interleaving*** of two messages emitted by two different ships, with nothing extra added in. We now define the problem more precisely. Let x, y, and z, be strings over some alphabet Σ, where $|z| = |x| + |y|$ (here $|w|$ denotes the length of a string w). The string z (the message received on the shore) is a ***interleaving*** of x and y (possible messages from the two ships) if there are (***possibly empty***) strings $x_1, x_2, \ldots, x_k$ and $y_1, y_2, \ldots, y_k$ so that $x = x_1 x_2 \ldots x_k$, $y = y_1 y_2 \ldots y_k$, and $z = x_1 y_1 x_2 y_2 \ldots x_k y_k$. Intuitively, z can be constructed by concatenating pieces of x and y, in the order in which these pieces appear in x and y, respectively. For example, `alabama` is an interleaving of `aba` and `lama`, but `amalaba` is not.

We want an algorithm that, given as input three strings x, y, and z such that $|z| = |x| + |y|$, returns **true** if z is an interleaving of x and y, and returns **false** otherwise.

a. (3 marks) Consider the following greedy strategy. Find the longest nonempty prefix w of x or y that matches a prefix of z; remove the prefix w from z and from either x or y (whichever of the two it came from — only from one, if they both have the same longest prefix that matches a prefix of z). Repeat until all three of x, y, and z are empty, in which case return **true**; or there is no nonempty prefix w of either x of y that matches a prefix of z, in which case return **false**. Give a simple example that demonstrates that this greedy algorithm is not correct.

b. (9 marks) Give a polynomial-time dynamic programming algorithm that solves the problem. To do so, define the sub-problems, give the recurrence showing how to solve a sub-problem from smaller subproblems, and give the pseudo-code of the algorithm that is based on this recurrence. Analyze the algorithm's running time.

Do not turn in the question below. It shows how Floyd-Warshall's all-pairs shortest path algorithm can detect negative cycles. We mentioned this result in the lecture without proof; here you are asked to do prove it.

Question 4. (0 marks) Consider the Floyd-Warshall algorithm with input a directed graph $G = (V, E)$ and edge weights $\mathbf{wt} : E \to \mathbb{Z}$. Let $V = \{1, 2, \ldots, n\}$. Suppose somes edge may have negative weights. Prove that G has a negative-weight cycle ***if and only if*** when the Floyd-Warshall algorithm terminates, $C[i, i, n] < 0$ for some node i.[1]

Hint: Prove that $C[i, j, k]$ is ***at most*** the minimum weight of any $i \xrightarrow{k} j$ ***simple*** path. Recall that a path is ***simple*** if all nodes on it except (possibly) the first and last are distinct; a ***simple cycle*** is a simple path whose first and last nodes are the same. (Question to think about, but not to include in your answer: Give an example where $C[i, j, k]$ is ***less than*** the minimum weight of any $i \xrightarrow{k} j$ simple path.)

[1] In the lecture slides, $C[i, j, k]$ is denoted $OPT(i, j, k)$.