CSC373 Summer '22
Assignment 2

Due Date: June 29, 2022, 11:59pm ET

Instructions

1. Typed assignments are preferred (e.g., PDFs created using LaTeX or Word), especially if your handwriting is possibly illegible or if you do not have access to a good quality scanner. Either way, you need to submit a single PDF named "hwk2.pdf" on MarkUS at `https://markus.teach.cs.toronto.edu/2022-05`

2. You will receive 20% of the points for a (sub)question when you leave it blank (or cross off any written solution) and write "I do not know how to approach this problem." If you leave it blank but do not write this or a similar statement, you will receive 10%. This does not apply to any bonus (sub)questions.

3. You may receive partial credit for the work that is clearly on the right track. But if your answer is largely irrelevant, you will receive 0 points.

Q1 [20 Points] Gene Alignment

In bioinformatics, a common task involves determining *alignment* between two genes (represented as strings over the alphabet $\Sigma = \{A, C, G, T\}$). For example, a possible alignment of $x = ATGCC$ with $y = TACGCA$ is:

$$\begin{array}{cccccccc} - & A & T & - & G & C & C \\ T & A & - & C & G & C & A \end{array}$$

As you can see, this involves writing both strings in columns so that

- the characters of each string appear in order,
- each column contains a character from at least one string,
- subject to the previous constraint, columns can contain "gaps" (represented as "$-$").

The score of a possible alignment is specified through a "scoring matrix" δ that gives a value for each possible column in the alignment. In our example, the total score would be equal to:

$$\delta(-, T) + \delta(A, A) + \delta(T, -) + \delta(-, C) + \delta(G, G) + \delta(C, C) + \delta(C, A).$$

Your task is to come up with an efficient algorithm that takes as inputs two strings $x, y \in \{A, C, G, T\}^*$ with a $[5 \times 5]$ scoring matrix δ, and that returns the highest possible score for alignment between x and y. (Assume that $\delta(-, -) = -\infty$, representing the fact that no alignment can align two gaps together.)

(a) [5 Points] Define the array(s) or table(s) that your solution would compute. Clearly explain what each entry means, and how you would compute the final answer given all the entries in your array(s) or table(s).

(b) [7 Points] Write a Bellman equation and briefly justify its correctness.

(c) [4 Points] Describe a bottom-up implementation of your Bellman equation, and analyze its worst-case running time and space complexity.

(d) [4 Points] Now, modify your algorithm to give a way to print the highest-scoring *alignment* itself. For this *reconstruction of the solution*, you may want to define an additional array, although that is not strictly necessary. Re-analyze the running time and space complexity of the modified algorithm.
HINT: You may find it useful to read Section 15.4 of the textbook.

Q2 [20 Points] Board Cutting

A sawmill gets orders for different lengths of boards $\ell_1, \ell_2, \ldots, \ell_n$. All of the boards have to be cut from *planks*, long pieces of wood with a fixed length L, and for technical reasons, the boards have to be cut in the order they are given. No matter how the planks are cut into boards, there is usually some amount of waste left over from each plank.
For example, if the board lengths are $2, 4, 1, 5$ and $L = 7$, then we can cut the boards in many different ways — some of them are clearly silly:

- cut board 2 from one plank, 4 from another, 1 from another, and 5 from a fourth plank (leaving four waste pieces with lengths $5, 3, 6, 2$), or
- cut board 2 from one plank, 4 from another, and $1, 5$ from a third plank (leaving three waste pieces with lengths $5, 3, 1$), or
- cut board 2 from one plank, $4, 1$ from another, and 5 from a third plank (leaving three waste pieces with lengths $5, 2, 2$), or
- cut boards $2, 4$ from one plank, 1 from another, and 5 from a third plank (leaving three waste pieces with lengths $1, 6, 2$), or
- cut boards $2, 4$ from one plank and $1, 5$ from another (leaving two waste pieces of length 1 each), or
- cut boards $2, 4, 1$ from one plank and 5 from another (leaving only one waste piece of length 2).

Instructions like "cut boards $2, 5$ from one plank and $4, 1$ from another" are *not* valid solutions because they change the order of the boards.
From the point of view of the sawmill, what matters most is *not* how many planks are used, nor how much waste is left in total. What matters is that the waste pieces all be the same length, as much as possible, because this allows those pieces to be reused more easily for other purposes. So in the example above, the best choice is the second-last ($2, 4$ together and $1, 5$ together) — the last solution is not quite as good because the lengths of the waste pieces are 0 and 2 (the plank with no waste is considered to have waste of length 0).
Formally, consider the following "Board Cutting" problem:

Input: Board lengths $\ell_1, \ell_2, \ldots, \ell_n$ and plank length L, where each length is a positive integer, each $\ell_i \leq L$, and the board lengths are not necessarily all distinct.

Output: A division of $\ell_1, \ell_2, \ldots, \ell_n$ into groups $(\ell_{i_0+1}, \ldots, \ell_{i_1}); (\ell_{i_1+1}, \ldots, \ell_{i_2}); \ldots; (\ell_{i_{k-1}+1}, \ldots, \ell_{i_k})$, where $i_0 = 0$, $i_k = n$, $\ell_{i_j+1} + \ell_{i_j+2} + \cdots + \ell_{i_{j+1}} \leq L$ for $0 \leq j \leq k-1$ (intuitively: the total length of each group is no more than the length of one plank), and $\sum_{j=0}^{k-1} \left(L - \ell_{i_j+1} - \ell_{i_j+2} - \cdots - \ell_{i_{j+1}}\right)^2$ is minimized (intuitively: the lengths of the leftover pieces of plank are all as close to each other as possible).

(a) [4 Points] Define the array(s) or table(s) that your solution would compute. Clearly explain what each entry means, and how you would compute the final answer given all the entries in your array(s) or table(s).

(b) [6 Points] Write a Bellman equation and briefly justify its correctness.

(c) [5 Points] Describe a bottom-up implementation of your Bellman equation, and analyze its worst-case running time and space complexity.

(d) [5 Points] Now, modify your algorithm to output the optimal *division* itself. You may define an additional array to do so, in which case, specify its definition clearly. Re-analyze the running time and space complexity of the modified algorithm.

Q3 [20 Points] Reducing Edge Capacities

(a) [4 Points] Prove or disprove: if $N = (V, E)$ is a network, f^* is a maximum flow in N, $e_0 \in E$ is an edge with $f^*(e_0) = c(e_0)$, and N' is the same network as N except that $c'(e_0) = c(e_0) - 1$, then the maximum flow f' in N' satisfies $|f'| < |f^*|$.

(b) [12 Points] Write an algorithm that takes a network $N = (V, E)$, maximum flow f^* in N, and an edge $e_0 \in E$ with $f^*(e_0) = c(e_0)$, and that outputs a maximum flow for network N', where N' is the same as N except that $c'(e_0) = c(e_0) - 1$.
Provide a brief argument that your algorithm is correct and analyze its time complexity. For full marks, your algorithm should be as efficient as possible.

(c) [4 Points] Write an algorithm that takes a network $N = (V, E)$ and that outputs a list of **all** edges $e_1, \ldots, e_k$ with the property that if the capacity of any edge in that list is reduced **by one unit**, the value of the maximum flow in N is also reduced.

Q4 [20 Points] Matrix Puzzle

You want to fill a matrix of n rows and m columns with non-negative integers so that the sum of each row or each column corresponds to a predetermined number. Each matrix entry also has an upper bound constraint. Specifically, given **non-negative** integers $r_1, \ldots, r_n$, $c_1, \ldots, c_m$, and $b_{1,1}, \ldots, b_{n,m}$, find integers $a_{1,1}, \ldots, a_{n,m}$ such that:

1. $0 \leq a_{i,j} \leq b_{i,j}$ for all $1 \leq i \leq n$, $1 \leq j \leq m$;
2. $r_i = \sum_{k=1}^{m} a_{i,k}$ for all $1 \leq i \leq n$;
3. $c_j = \sum_{k=1}^{n} a_{k,j}$ for all $1 \leq j \leq m$.

(a) [10 Points] Design an algorithm to find such integers. Your algorithm should return NIL if such integers do not exist.

(b) [8 Points] Prove the correctness of your algorithm.

(c) [2 Points] Analyze the running time of your algorithm.

BONUS QUESTION

Q5 [10 Points] Knight Coexistence

You want to place knights on an $n \times n$ chessboard. The chessboard has n^2 positions from $(0, 0)$ to $(n-1, n-1)$. A knight at position (x, y) is able to attack up to eight different positions: $(x+2, y+1)$, $(x+2, y-1)$, $(x+1, y+2)$, $(x+1, y-2)$, $(x-1, y+2)$, $(x-1, y-2)$, $(x-2, y+1)$, and $(x-2, y-1)$ – for each position that lies on the board, of course. Moreover, some positions are blocked so that you cannot place knights on them. Given n and a list of all blocked positions, find a way to place as many knights as possible on the chessboard so that the placed knights cannot attack each other.