CSOR W4231.001 – Spring, 2023
Eleni Drinea

Homework 3 (160 points)

Out: Monday, February 27, 2023
Due: 11:59pm, Monday, March 27, 2023

Homework Instructions.

1. For all algorithms that you are asked to "give" or "design", you should

   - Describe your algorithm clearly in English.
   - Give pseudocode.
   - Argue correctness; you may provide a formal proof or a convincing argument.
   - Provide, with an explanation, the best (smallest) upper bound that you can for the running time. All bounds should be **worst-case** bounds, unless explicitly stated otherwise.

   You are also encouraged to analyze the space required by your algorithm. We will not remove marks if you don't, unless the problem explicitly asks you to analyze space complexity.

2. **If you give a Dynamic Programming algorithm, the above requirements are modified as follows:**

   (a) Clearly define the subproblems in English.
   (b) Explain the recurrence in English. (This counts as a proof of correctness; feel free to give an inductive proof of correctness too for practice but points will not be deducted if you don't.) Then give the recurrence in symbols.
   (c) State boundary conditions.
   (d) Analyze time.
   (e) Analyze space.
   (f) If you're filling in a matrix, explain the order to fill in subproblems in English.
   (g) Give pseudocode.

3. **You should not use any external resources for this homework.** Failure to follow this instruction may have a negative impact on your performance in exams and interviews. For the same reason, **you should avoid collaborating** with your classmates, at least until you have thought through the problems on your own for a reasonably long time. I encourage you to work on problems 1 and 2 immediately, then on recommended exercises 1, 2, 3, 4, then on problems 3, 4, 5, and finally on recommended exercises 5 and 6.

4. You should submit this assignment as a **pdf** file to Gradescope. Other file formats will not be graded, and will automatically receive a score of 0.

5. I recommend you type your solutions using LaTeX. For every assignment, you will earn 5 extra credit points if you type your solutions using LaTeX or other software that prints equations and algorithms neatly. If you do not type your solutions, make sure that your handwriting is very clear and that your scan is high quality.

6. You should write up your solutions **entirely on your own**. Collaboration is limited to discussion of ideas only. You should adhere to the department's academic honesty policy (see the course syllabus). Similarity between your solutions and solutions of your classmates or solutions posted online will result in receiving a 0 in this assignment, and possibly further disciplinary actions. There will be no exception to this policy and it may be applied retro-actively if we have reasons to re-evaluate this homework.

Homework Problems

1. (25 points) There is a network of roads $G = (V, E, \ell)$ connecting a set of cities V. Each road in E has an associated length (weight) $\ell(e)$. There is a proposal to add one new road on this network, and there is a list E' of pairs of cities between which the new road can be built. Each such potential road $e' \in E'$ has an associated length.

   As a designer for the public works department you are asked to determine the road $e' \in E'$ whose addition to the existing network G will result in the maximum decrease in the driving distance between two fixed cities s and t in the network.

   Give an efficient algorithm to solve this problem.

2. (25 points) You are consulting for a trucking company that does a large amount of business shipping packages (boxes) between New York and Boston. The volume is high enough that they have to send a number of trucks each day between the two locations. Trucks have a fixed integer upper bound $W > 0$ on the maximum weight they may carry.

   Boxes arrive at the New York station one by one. Each box i has an integer weight $w_i > 0$. The trucking station is quite small, so at most one truck can be at the station at any time. Boxes must be shipped in the order they arrive.

   Given a set of n boxes ordered by their arrival time and their weights $w_1, \ldots, w_n$, design an efficient algorithm that loads boxes in the trucks so that the number of trucks needed is minimized.

3. (40 points) There are n libraries $L_1, L_2, \ldots, L_n$. We want to store copies of a book in some of the libraries. Storing a copy at L_i incurs an integer purchase cost $c_i > 0$. A copy of the book is always stored at L_n. If a user requests the book from L_i and L_i does not have it, then $L_{i+1}, L_{i+2}, \ldots$ are searched sequentially until a copy of the book is found at L_j, for some $j > i$. This results in a *user delay* of $j - i$. (Note that, in this case, no library L_k with an index k smaller than i is searched; also, if the user finds the book at L_i, then the user delay is 0.)

   We define the **total cost** as the sum of all the purchase costs and all the user delays, assuming that there is one user in each of the n libraries. For example, if there are 4 libraries, and copies of the book are stored at L_1 and L_4, the total cost is $c_1 + c_4 + 2 + 1$.

Give an efficient algorithm that returns

(a) the minimum **total cost**; and

(b) a set of libraries where copies of the books must be placed to achieve the minimum total cost.

4. (35 points) A *time-series* is a sequence of integers $a_1, a_2, \ldots, a_n$, where a_i represents some signal at time i.

   Often we need to be able to compare two time-series and declare whether they are similar or not. E.g., one time-series might represent the heart beats of a patient and the other a "normal" heart beat.

   One way to compare two such time series, say $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_m$ is as follows. Define a *matching* to be a non-decreasing map $f : [n] \to [m]$, thought of as a map from indices of a to indices of b, such that $f(i) \leq f(j)$ whenever $i < j$. For a given matching f, the cost of the matching is $cost(f) = \sum_{i=1}^{n} |a_i - b_{f(i)}|$.

   We define the *distance* between $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_m$ to be the minimum $cost(f)$ over all non-decreasing matchings f.

   Design an efficient algorithm that computes the *distance* between two given sequences $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_m$. For example, if the input sequences are $4, 3, 8, 8, 1$ and $3, 4, 8, 2$, then the *distance* is $0 + 1 + 0 + 0 + 1 = 2$, corresponding to the matching $f(1) = 2, f(2) = 2, f(3) = 3, f(4) = 3, f(5) = 4$ (other matchings also achieve this distance).

5. (35 points) You want to break a string of length n into $m + 1$ pieces. To this end, you are using a string-processing language that offers a primitive operation which splits the string into two pieces. This operation involves copying the original string, so it takes n units of time for a string of length n, regardless of the position of the cut.

   Note that the order in which you make the cuts matters: for example, if you want to cut a 20-character string at positions 3 and 10, making the first cut at position 3 incurs a total cost of $20 + 17 = 37$, while cutting at position 10 first has a better cost of $20 + 10 = 30$.

   Given n, and the positions of m cuts in a string of length n, design an efficient algorithm that computes the minimum cost of breaking the string into $m + 1$ pieces.

RECOMMENDED exercises: *Do NOT return, they will not be graded.*

1. A server has n customers waiting to be served. The service time for customer i is t_i minutes. So if the customers are served in order of increasing i, the i-th customer spends $\sum_{j=1}^{i} t_j$ minutes waiting to be served.

   Given n, $\{t_1, t_2, \ldots, t_n\}$, design an efficient algorithm to compute the optimal order in which to process the customers so that the total waiting time below is minimized:

$$T = \sum_{i=1}^{n} \text{(time spent waiting by customer } i\text{)}$$

2. `https://leetcode.com/problems/climbing-stairs/`.

3. You are managing a team of engineers and need to assign tasks to them for n weeks. Every week i for $1 \leq i \leq n$, you can assign to the team one of two kinds of tasks: an *easy* task with revenue $\ell_i > 0$ or a *difficult* task with revenue $h_i > 0$. There is one constraint: if you pick a difficult task in week i, then you must assign no task to your team in week $i-1$. That is, your team must be idle during week $i-1$, thus earn revenue 0 in week $i-1$.

   Given sets of revenues $\ell_1, \ell_2, \ldots, \ell_n$ and $h_1, h_2, \ldots, h_n$, you want to assign tasks to your team during the n weeks so that the total revenue V of the tasks is **maximized**, subject to the constraint above. Give an efficient algorithm that computes the optimal revenue V. (It is ok for your team to start with a difficult task in week 1.)

4. Give an $O(n^2)$ algorithm to compute the length of the **longest** increasing subsequence of a sequence of numbers $a_1, \ldots, a_n$. A subsequence is any subset of these numbers taken in order, of the form $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ where $1 \leq i_1 < i_2 < \ldots < i_k \leq n$ and an increasing subsequence is one in which the numbers are getting strictly larger.

   For example, the longest increasing subsequence of $5, 2, 8, 6, 3, 6, 7$ is $2, 3, 6, 7$ and its length is 4.

5. Consider an array A with n numbers, some of which may be negative. We wish to find indices i and j such that $\sum_{k=i}^{j} A[k]$ is maximized.

   Design and analyze an algorithm for this task that runs in $O(n)$ time.

6. Given two strings $x = x_1 x_2 \cdots x_m$ and $y = y_1 y_2 \cdots y_n$, we wish to find the length of their longest common substring, that is, the largest k for which there are indices i and j such that

$$x_i x_{i+1} \cdots x_{i+k-1} = y_j y_{j+1} \cdots y_{j+k-1}.$$

   Give an efficient algorithm for this problem.