CSOR W4231.001 – Spring, 2022
Eleni Drinea

Homework 2 (135 points)

Out: Monday, February 21, 2022
Due: 11:59pm, Monday, March 7, 2022

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. **Full credit will be given to the fastest correct solution.** For example, an algorithm that solves the problem but runs in time $O(n^2)$ will not receive full marks if there is another algorithm that solves the problem and runs in $O(n)$ (possibly at the expense of some additional space).

4. **You should not use any external resources for this homework.** Failure to follow this instruction will have a negative impact on your performance in the exams and in interviews. For the same reason, **you should avoid collaborating** with your classmates, at least when working on problems 1-2 and 4-5. I encourage you to work on problems 1 and 2 immediately, then on recommended exercise 1, then on problem 3. Once we introduce Dynamic Programming in class, work on recommended exercises 2-4, then on problems 4-5. Finally, work on recommended exercises 5-6.

5. 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.

6. 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.

7. You should write up the 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. (20 points) Design an efficient algorithm for the following task.

   On input an undirected weighted graph $G = (V, E, w)$ with positive edge weights, and a specific vertex $s \in V$, return a Boolean array `unique[]` of size $n = |V|$ such that `unique`$[i] = 1$ if and only if there is a unique shortest $s - i$ path.

2. (20 points) Suppose we introduce a further optimization criterion in our single-origin shortest-paths problem: We are now looking for the shortest $s - i$ path with the *fewest* edges. We define

   $$minedges[i] = \text{minimum number of edges in a shortest path from } s \text{ to } i$$

   Give an efficient algorithm that, on input a directed weighted graph $G = (V, E, w)$ with positive edge weights, and an origin node $s \in V$, computes $minedges[i]$ for all $i \in V$.

3. (30 points) You're helping to organize a mini-triathlon event: each of n contestants must first swim 20 laps of a pool, then bike 10 miles, then run 3 miles. The contestants must use the pool **one at a time**—that is, as soon as the first person is out of the pool, a second contestant can begin swimming the 20 laps—but many contestants may bike and run simultaneously.

   For each contestant i, you know the expected swimming time s_i, expected biking time b_i and expected running time r_i. Assume that all contestants need **exactly** their expected times to finish the three parts of the triathlon. You are tasked with sequencing the starts of the contestants so that the triathlon event ends as soon as possible.

   More formally, a *schedule* for the triathlon is an order in which to sequence the starts of the contestants. The *completion* time of a schedule is the earliest time at which **all** contestants will be finished. Given n triples (s_i, b_i, r_i), design an efficient algorithm that produces the schedule with the minimum completion time.

4. (35 points) You are given a string of n characters $s[1, \ldots, n]$, which you believe to be a corrupted text document in which all punctuation has vanished (e.g., "itwasthebestoftimes"). You wish to reconstruct the document using a dictionary available in the form of a Boolean function `dict(·)`: for any string w,

   $$\texttt{dict}(w) = \begin{cases} 1, & \text{if } w \text{ is a valid word} \\ 0, & \text{otherwise} \end{cases}$$

   You may assume that calls to `dict` take constant time.

Give an $O(n^2)$ dynamic programming algorithm that determines whether or not the string $s[\cdot]$ can be reconstituted as a sequence of valid words. If the answer is yes, you should also output the corresponding sequence of words.

5. (30 points) Alice and Bob are playing a match to see who is the first to win n games, for some fixed $n > 0$. Suppose Alice and Bob are equally competent, that is, each of them wins a game with probability 1/2. Further, suppose that they have already played $i + j$ games, of which Alice won i and Bob won j.

   Give an efficient algorithm to compute the probability that Alice will go on to win the match. For example, if $i = n - 1$ and $j = n - 3$, then the probability that Alice will win the match is 7/8, since she must win any of the next three games.

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. 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.

4. 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.)

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.