CS 170 Homework 7

Due **Saturday 10/18/2025, at 10:00 pm (grace period until 11:59pm)**

1 Study Group (1 point)

List the names and SIDs of the members in your study group. If you have no collaborators, explicitly write "none".

2 Maximum subarray sum revisited (12 points)

Given an array $A = [A[0], A[1], \ldots, A[n-1]]$ of n integers, the maximum subarray sum is the largest sum of any contiguous subarray of A (including the empty subarray). In other words, the maximum subarray sum is:

$$\max_{0 \le i \le j \le n-1} \sum_{k=i}^{j} A[k]$$

For example, the maximum subarray sum of $[-2, 1, -3, 4, -1, 2, 1, -5, 4]$ is 6, the sum of the contiguous subarray $[4, -1, 2, 1]$.

In Discussion 2, we saw how to find the maximum subarray sum in $O(n \log n)$ time using divide and conquer. This problem can actually be solved in $O(n)$ time using dynamic programming.

(a) (7 points) Devise a dynamic programming algorithm solving this problem in $O(n)$ time.

(i) Define your subproblem $f(\cdot)$ in words. Make sure to include how many parameters there are and what they mean, and tell us what input(s) you feed into f to get the answer to your problem.

Hint: any nonempty subarray must end somewhere.

(ii) Write the recurrence relation for f, as well as the base cases.

(iii) Prove that the recurrence correctly solves the problem.

Hint: for almost all DP correctness proofs, we suggest induction.

(iv) Analyze the runtime and space complexity of your final DP algorithm. Note that the top-down and bottom-up approaches to DP have the same runtime complexity; however, bottom-up can potentially yield a better space complexity.

(b) (5 points) Describe an $O(n)$-time dynamic programming algorithm to find the maximum subarray sum among subarrays that have an odd number of odd numbers (i.e., $[-3, 4]$ and $[-1, 2, 1, -5]$, but not $[-2], [-2, 1, -3]$, or the empty array). You do not need to provide a proof of correctness or an analysis of the runtime or space complexity.

3 Longest common subsequence (10 points)

In lecture, we covered the longest increasing subsequence problem (LIS). Now, let us consider the longest common subsequence problem (LCS), which is a bit more involved. Given two arrays A and B of integers, you want to determine the length of their longest common subsequence. If they do not share any common elements, return 0.

For example, given $A = [1, 2, 3, 4, 5]$ and $B = [1, 3, 5, 7]$, their longest common subsequence is $[1, 3, 5]$ with length 3.

We will design an algorithm that solves this problem in $O(nm)$ time, where n is the length of A and m is the length of B.

(a) (2 points) Define your subproblem in words.

Hint: looking at the subproblem for Edit Distance may be helpful.

(b) (3 points) Write your recurrence relation and describe your base cases. (A fully correct recurrence relation will always have the base cases specified.)

(c) (1 point) In what order do we solve the subproblems?

(d) (2 points) What is the runtime of this dynamic programming algorithm?

(e) (2 points) Analyze the space complexity of your DP algorithm. Show how to reduce the space complexity of your algorithm to $O(\min\{m, n\})$ additional memory (i.e., not including the input).

Remark: for all space complexity analyses in this class, we only consider writeable auxiliary memory, which does not *include any input if kept read-only.*

4 Egg drop (14 points)

You are given m identical eggs and an n story building. You need to figure out the highest floor $b \in \{0, 1, 2, \ldots, n\}$ that you can drop an egg from without breaking it. Each egg will never break when dropped from floor b or lower, and always breaks if dropped from floor $b+1$ or higher. ($b = 0$ means the egg always breaks). Once an egg breaks, you cannot use it anymore. However, if an egg does not break, you can reuse it.

Let $f(n, m)$ be the minimum number of egg drops that are needed to find b (regardless of the value of b).

(a) (2 points) Find $f(1, m)$, $f(0, m)$, $f(n, 1)$, and $f(n, 0)$. Briefly explain your answers.

Hint: use ∞ to denote that it is impossible to find b.

(b) (3 points) Consider dropping an egg at floor h when there are n floors and m eggs left. Then, it either breaks or doesn't break. In either scenario, determine the minimum remaining number of egg drops that are needed to find b in terms of $f(\cdot, \cdot)$, n, m, and/or h.

(c) (3 points) Find a recurrence relation for $f(n, m)$.

Hint: whenever you drop an egg, call whichever of the egg breaking/not breaking leads to more drops the "worst-case event". Since we need to find b regardless of its value, you should assume the worst-case event always happens.

(d) (2 points) Write pseudocode to compute $f(n, m)$. In particular, make sure that your code processes the subproblems in the correct order.

(e) (1 point) Analyze the runtime complexity of your DP algorithm.

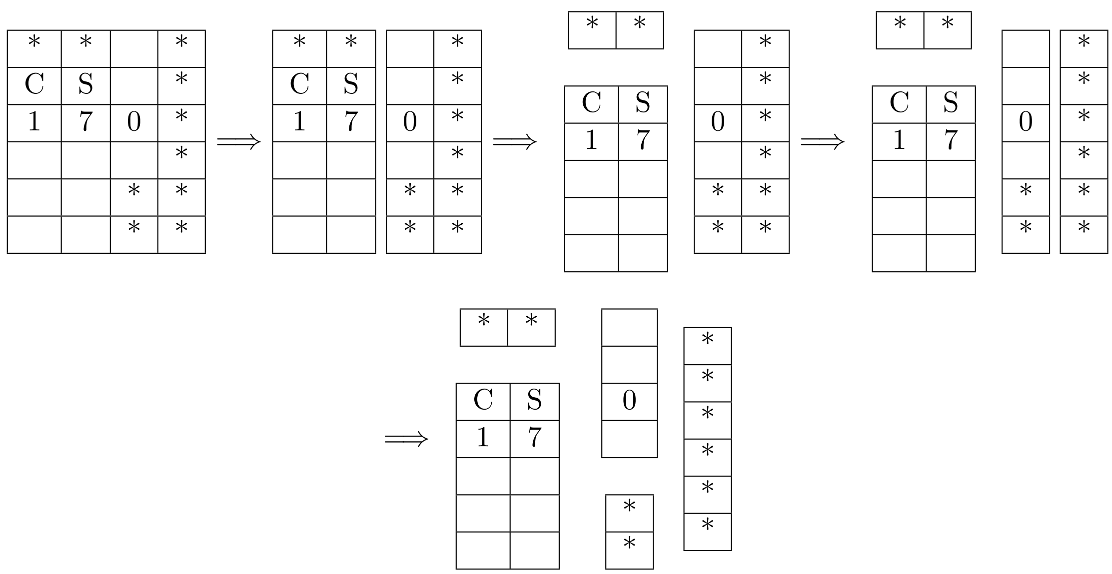
(f) (1 point) Analyze the space complexity of your DP algorithm.

(g) (2 points) Is it possible to modify your algorithm above to use less space? If so, describe your modification and re-analyze the space complexity. If not, briefly justify.

5 My dog ate my homework (12 points)

One morning, you wake up to realize that your dog ate some of your CS 170 homework paper, which is an $\ell \times w$ rectangular grid of squares. Some of the squares have holes chewed through them, and you cannot use paper that has a hole in it. You would like to cut the paper into pieces so as to separate all the tattered squares from all the clean, un-bitten squares. You want to do this so that you can save as much of your work as possible.

For example, shown below is a 6×4 piece of paper where the bitten squares are marked with *. As shown in the picture, one can separate the bitten parts out in exactly four cuts.



(Each *cut* is either horizontal or vertical, and of one piece of paper at a time.)

Formally, the problem is as follows:

> **Input:** Dimensions of the paper $\ell \times w$ and an array $A[i, j]$ such that $A[i, j] = 1$ if and only if the ij^{th} square has holes bitten into it.
>
> **Goal:** Find the minimum number of cuts needed so that the $A[i, j]$ values of each piece are either all 0 or all 1.

Design a DP-based algorithm to find the smallest number of cuts needed to separate all the bitten parts out in $O(\ell^3 w^3)$ time. For **extra credit**, try to optimize your algorithm to achieve a runtime of $O((\ell + w)\ell^2 w^2)$.

(a) (3 points) Define your subproblem.

Hint: try making any arbitrary cut. What two subproblems do you now have? What parameters do you need to properly handle recursing on these two resulting subproblems?

(b) (5 points) Write down the recurrence relation for your subproblems as well as your base cases. Provide a brief justification of your recurrence relation.

(c) (1 point) Describe the order in which we should solve the subproblems in your DP algorithm.

(d) (2 points) What is the runtime complexity of your DP algorithm? Provide a justification.

(e) (1 point) What is the space complexity of your algorithm? Provide a justification.

6 [Coding] Dynamic programming (8 points)

For this week's coding questions, we'll implement dynamic programming algorithms to solve two classic problems: **Longest Increasing Subsequence** and **Edit Distance**. There are two ways that you can access the notebook and complete the problems:

1. **On Datahub**: click here and navigate to the `hw07` folder.
2. **On Local Machine**: `git clone` (or if you already cloned it, `git pull`) from the coding homework repo,

   https://github.com/Berkeley-CS170/cs170-fa25-coding

   and navigate to the `hw07` folder. Refer to the `README.md` for local setup instructions.

Notes:

- *Submission Instructions:* Please download your completed submission `.zip` file and submit it to the Gradescope assignment titled "HW07 (Coding)."
- *Getting Help:* Conceptual questions are always welcome on Edstem and office hours; *note that support for debugging help during OH will be limited.* If you need debugging help first try asking on the public Edstem threads. To ensure others can help you, make sure to:

  1. Describe the steps you've taken to debug the issue prior to posting on Ed.
  2. Describe the specific error you're running into.
  3. Include a few small but nontrivial test cases, alongside both the output you expected to receive and your function's actual output.

  If staff tells you to make a private Ed post, make sure to include *all of the above items* plus your full function implementation. If you don't provide them, we will ask you to provide them.
- *Academic Honesty Guideline:* We realize that code for some of the algorithms we ask you to implement may be readily available online, but we strongly encourage you to not directly copy code from these sources. Instead, try to refer to the resources mentioned in the notebook and come up with code yourself. That being said, we **do acknowledge** that there may not be many different ways to code up particular algorithms and that your solution may be similar to other solutions available online.

