CS 170 Homework 7

Due **10/17/2023, at 10:00 pm (grace period until 11:59pm)**

4-Part Solutions

For all (and only) dynamic programming problems in this class, we would like you to follow a 4-part solution format:

1. **Algorithm Description:** since dynamic programming algorithms can be difficult to explain, you should follow the template below to optimize clarity.
   (a) *Define your subproblem.* In words, define a function f so that the evaluation of f on a certain input gives the answer to the stated problem.

   You should clearly state how many parameters f has, what those parameters represent, what f evaluated on those parameters represents, and what inputs you should feed into f to get the answer to the stated problem.

   (b) *Provide your recurrence relation.* More precisely, give a recurrence relation showing how to compute f recursively, and make sure to provide base cases. If you need to use certain data structures to make computation of f faster, you should say so.

   (c) *Subproblem Ordering:* describe the order in which you should solve the subproblems to obtain the final answer.
2. **Proof of Correctness:** provide some inductive proof that shows why your DP algorithm computes the correct result.
3. **Runtime Analysis:** analyze the runtime of your algorithm.
4. **Space Analysis:** analyze the space/memory complexity of your algorithm.

1 Study Group

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

2 Non-Prefix Code

As we have learned in lecture, the Huffman code satisfies the *Prefix Property*, which states that the bit string representing each symbol is not a prefix of the bit string representing any other symbol. One nice property of such codes is that, given a bit string, there is at most one way to decode it back to a sequence of symbols. However, this is not true anymore once we are working with codes that do not satisfy the Prefix Property. For example, consider the code that maps A to 1, B to 01 and C to 101. A bit string 101 can be interpreted in two ways: as C or as AB.

Your task is to, given a bit string s, determine whether it is possible to interpret s as a sequence of symbols. The mapping from symbols to bit strings of the code will be given to you as a dictionary d (e.g., in the example, $d = \{A : 1, B : 01, C : 101\}$); you may assume that you can access each symbol in the dictionary in constant time.

(a) Describe an algorithm that solves this problem in at most $O(nk\ell)$ where n is the length of the input bit string s, k is the number of symbols, and ℓ is an upper bound on the length of the bit strings representing symbols. **Please provide a 4-part solution**.

(b) **(Optional)** How can you modify the algorithm from part (a) to speed up the runtime to $O((n + k)\ell)$?

3 Ideal Targets

You are given a directed acyclic graph $G = (V, E)$ with unweighted edges. Every vertex $v \in V$ has an integer score $s[v]$. For a vertex v, we say that a vertex u is an ideal target for v if:

1. It is possible to go from v to u.
2. $s[u]$ is maximized.

In other words, out of all vertices that v can reach, u (i.e. v's ideal target) is the one with the maximum score.

Given the scores for all vertices, describe a linear-time algorithm to find the ideal targets for every vertex v. (Note that v can be its own ideal target.)

Just provide the algorithm description and runtime analysis. Proof of correctness and space complexity analysis are not required.

4 Egg Drop

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

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

(b) Consider dropping an egg at floor x 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 ℓ in terms of $f(\cdot, \cdot)$, n, m, and/or x.

(c) 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 ℓ regardless of its value, you should assume the worst-case event always happens.

(d) If we want to use dynamic programming to compute $f(n, m)$ given n and m, in what order do we solve the subproblems?

(e) Based on your responses to previous parts, analyze the runtime complexity of your DP algorithm.

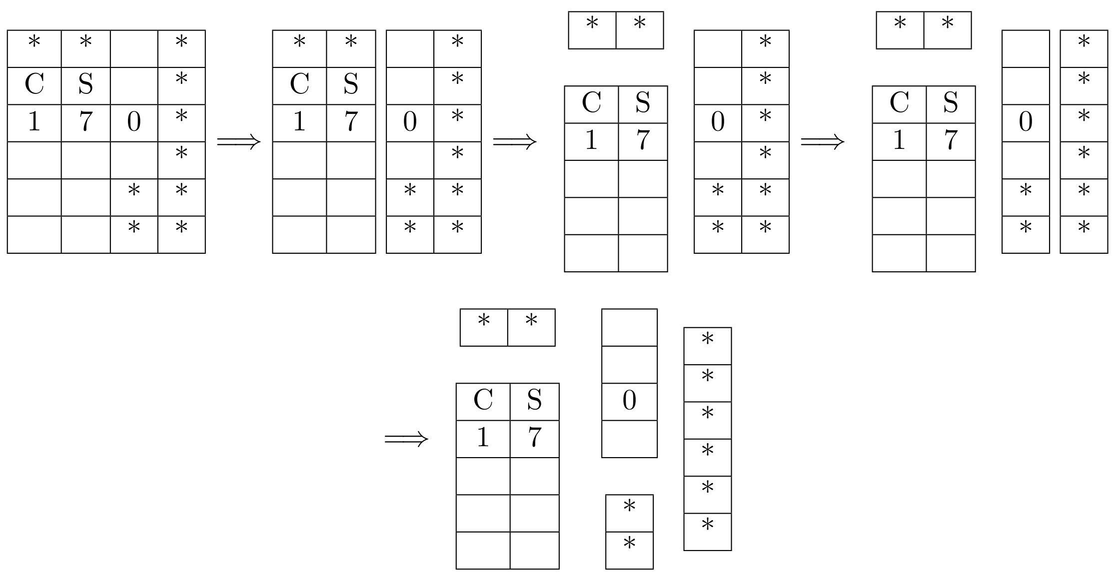
(f) Analyze the space complexity of your DP algorithm.

(g) (**Extra Credit**) 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

One morning, you wake up to realize that your dog ate some of your CS 170 homework paper, which is an $m \times n$ 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 as 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.)

Design a DP based algorithm to find the smallest number of cuts needed to separate all the bitten parts out. Formally, the problem is as follows:

> **Input:** Dimensions of the paper $m \times n$ 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.

(a) Define your subproblem.

Hint: try making any arbitrary cut. What two subproblems do you now have?

(b) Write down the recurrence relation for your subproblems. A fully correct recurrence relation will always have the base cases specified.

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

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

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

6 Coding Questions

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 Local Machine**: `git clone` (or if you already cloned it, `git pull`) from the coding homework repo,

   https://github.com/Berkeley-CS170/cs170-fa23-coding

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

2. **On Datahub**: Click here and navigate to the `hw07` folder if you prefer to complete this question on Berkeley DataHub.

Notes:

- *Submission Instructions:* Please download your completed submission `.zip` file and submit it to the Gradescope assignment titled "Homework 7 Coding Portion".
- *OH/HWP Instructions:* Designated coding course staff will provide conceptual and debugging help during office hours and homework parties.
- *Edstem Instructions:* Conceptual questions are always welcome on the public thread. If you need debugging help first try asking on the public 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 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.