CS 170 Homework 7

Due **3/11/2024, at 10:00 pm (grace period until 11:59pm)**

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".

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.

2 Not Too Much DP

(a) Given an array A with positive or negative integers (i.e. non-zero), we want to find the subarray (i.e. contiguous sequence of elements in the array) that creates the maximum product. We will use 1-dimensional DP to approach this problem where $dp[i]$ will return the maximum subarray product and minimum subarray product of $A[0...i]$ that include $A[i]$; or in other words, the maximum and minimum of any subarray that ends with $A[i]$. Notice here we need to keep track of the minimum product as well; in case the next element in the array is negative, the minimum product might become the maximum product after new multiplication.

Given the DP subproblems below, perform the following:

0	1	2	3	4	5
$(-2, -2)$	$(3, -6)$	$(24, -12)$	$(48, -24)$	$(24, -48)$	$(96, -192)$

i. Recover the original array.

ii. Identify the subarray that produces the maximum product.

(b) Given strings s_1, s_2, and s_3, find whether s_3 can be formed by an interleaving of s_1 and s_2. s_3 is defined to be an interleaving of s_1 and s_2 if s_3 contains all of the characters of s_1 and s_2 and only those characters. Additionally, the order of the characters of s_1 and s_2 are preserved in s_3.

Let l_1, l_2, l_3 be the lengths of s_1, s_2 and s_3 respectively. We use 2-dimensional DP to approach this problem where $dp[i][j] = True$ if the substring $s_3[: i+j]$ is an interleaving of substrings $s_1[: i]$ and $s_2[: j]$, and False otherwise.

(i) For this subpart, let s_1 = "cbadb", s_2 = "badda", s_3 = "cbbadadadb". Using those inputs, fill in the missing grids in the following table:

-	0	1	2	3	4	5
0	T	T	T	F	F	F
1	F	T	T	F	F	F
2	F	F	T	F	F	?
3	F	F	T	?	F	F
4	F	F	?	T	?	?
5	F	F	?	T	?	T

Note that for this table, the columns correspond to the characters of s_1 and the rows to s_2. Hence $dp[row][col]$ is true if $s_3[: row + col]$ is a valid interleaving of $s_1[: col]$ and $s_2[: row]$. Additionally, $dp[row][0]$ corresponds to checking whether $s_3[: row] == s_2[: row]$, and $dp[0][col]$ corresponds to checking whether $s_3[: col] == s_1[: col]$.

(ii) For this subpart, you are given s_3 = "sponpdaens" and part of the DP table. Using those information, recover s_1 and s_2. (Note there might be multiple s_1, s_2 combinations that will produce the same table. If multiple combinations are possible, list out all of them.)

-	0	1	2	3	4	5
0	T	-	-	-	-	-
1	-	T	-	-	-	-
2	-	-	T	F	F	-
3	-	-	-	-	T	-
4	-	-	-	-	T	F
5	-	-	-	-	-	T

(iii) For this subpart, determine whether the following subtables are possible (subtable is simply a small part of the entire table). Give a brief justification/reasoning to your answer.

1.

T	T
T	T

2.

T	F
F	T

3.

T	F
T	F

3 Egg Drop

You are given m identical eggs and an n story building. You need to figure out the highest floor $h \in \{0, 1, 2, \ldots n\}$ that you can drop an egg from without breaking it. Each egg will never break when dropped from floor h or lower, and always breaks if dropped from floor $h + 1$ or higher. ($h = 0$ 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 h (regardless of the value of h).

(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 h 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 h 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.

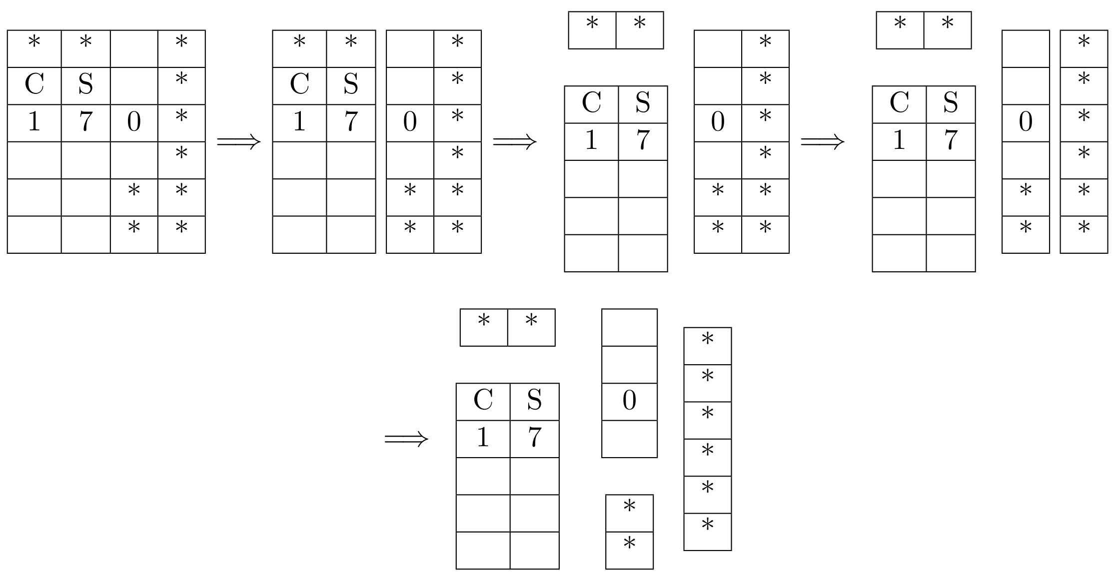
(f) Analyze the space complexity of your DP algorithm.

(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.

4 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 $P[i, j]$ such that $P[i, j] = 1$ if and only if the ij^{th} square has holes bitten into it.
>
> **Goal:** Find the minimum number of cuts needed so that the $P[i, j]$ 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.

5 Knightmare

Give a dynamic programming algorithm to find the number of ways you can place knights on an L by H ($L < H$) chessboard such that no two knights can attack each other (there can be any number of knights on the board, including zero knights). Knights can move in a 2×1 shape pattern in any direction.

Provide a 4-part solution. Your algorithm's runtime should be $O(2^{3L}LH)$, and return your answer mod 1773.

Hint: if a knight is on row i, what rows on the chessboard can it affect?

6 [Coding] Edit Distance

For this week's coding questions, we'll implement the Edit Distance algorithm you saw in lecture. 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-sp24-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 "Homework 7 Coding Portion".
- *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.