CS 3510 Section C: Design & Analysis of Algorithms October 3rd 2023

HW 5: Dynamic Programming

YOUR NAME HERE Due: October 11th 2023

- Please type your solutions using LaTeX or any other software. Handwritten solutions will not be accepted.
- If we ask for a specific running time, a correct solution achieving it will receive full credit even if a faster solution exists. If no target running time is provided, faster solutions will earn more credit.

1.) (20 points) A monkey is organizing a banana heist. Every day that the monkey chooses to go into the store, he steals every single banana that day. Unfortunately, there are three problems:

1. The number of bananas in stock changes from day to day.
2. He can't go two days in a row, they might catch on.
3. He can't go on both the first and the last day.[1]

Fortunately, the monkey has the inventory sheet, and knows the number of bananas that will be available in the following days. The array $B = [b_1, ..., b_n]$ of positive integers tells us that on day i, there will be b_i bananas. Design a dynamic programming algorithm to compute the max number of bananas he can eat in n days.

Example:
Input: $A = [5, 7, 8, 10, 9, 4, 2, 11, 6, 1]$
Output: 33
Explanation: the monkey can steal 5 bananas on the 1st day, 8 on the 3rd day, 9 on the 5th day and 11 bananas on the 8th day. He chose to go on day 1 and not go on day 10.

Example:
Input: $A = [3, 4, 3]$
Output 4
Explanation: the monkey can steal 4 bananas on the 2nd day. He can't steal bananas on the first day and last day.

(a) Define the entries of your table in words. E.g. $T[i]$ or $T[i][j]$ is ...

> **Solution:** $T[i]$ is the maximum total number of bananas that our monkey can steal from days 1 to i.

(b) State recurrence for entries of the table in terms of smaller subproblems. Briefly explain in words why it is correct.

> **Solution:** $T[i] = \max(T[i-1], T[i-2] + a_i)$
> Since the monkey is looking for maximizing the sum of b_i on alternate days, as he cannot steal bananas on two consecutive days, we take the maximum from our DP table from the $(i-1)^{th}$ day or $(i-2)^{th}$ day plus the cookies he can buy today.
>
> The base case for this problem is for $T[0] = B[0]$ and $T[1] = \max(B[0], B[1])$.

(c) Write **pseudocode** for your algorithm to solve this problem.

[1]He has to do community service on one of those days, for stealing bananas previously

Solution: Pseudo-code omitted, but it is required to make two calls to our problem to ensure we meet condition 3 (first and last day). We first call our function on $B[0 \ldots n-1]$, and then $B[1 \ldots n]$, and take the maximum result.

(d) Analyze the running time of your algorithm.

Solution: The runtime is $\mathcal{O}(n)$ as we are iterating over the size of the input array B.

2.) (20 points) You are given a integer n in base 10. Define a "step" as choosing a digit from n and subtracting it from n. You want to find the minimum number of steps required to reach 0.

Example: Given $n = 17$, the shortest sequence is $17 \to 10 \to 9 \to 0$ which takes 3 steps.

(a) Define the entries of your table in words. E.g. $T[i]$ or $T[i, j]$ is ...

> **Solution:** $T[i]$ represents the minimum number of steps to take i to 0.

(b) State recurrence for entries of the table in terms of smaller subproblems. Briefly explain in words why it is correct.

> **Solution:** $T[i] = min_{d \in i}(T[i - d])$ where we iterate through all digits d of i. We start by computing the minimum amount of steps for smaller numbers, and then work our way up to n.

(c) Write **pseudocode** for your algorithm to solve this problem.

> **Solution:** Pseudocode omitted.

(d) Analyze the running time of your algorithm.

> **Solution:** $\mathcal{O}(10n) = \mathcal{O}(n)$.

3.) (20 points) You are given two strings X and Y, of length n and m respectively. You want to find the length of the longest common *"substringquence"* of these two strings, i.e., you want to find the length of the longest string that appears as a substring of X and a subsequence of Y.

Example:
If X = "helloworld" and Y = "longwayhome", you should return 4, since "lowo" is the longest string that appears as a substring of X and as a subsequence of Y.

(a) Define the entries of your table in words. E.g. $T[i]$ or $T[i, j]$ is ...

> **Solution:** Define a 2-D table T where $T[i][j]$ is the longest common substringquence of $X = [x_0, x_1, ..., x_{i-1}]$ (ending at and including $X[i]$) and $Y = [y_0, y_1, ..., y_{j-1}]$.

(b) State recurrence for entries of the table in terms of smaller subproblems. Briefly explain in words why it is correct.

> **Solution:**
>
> $$T[i][j] = \begin{cases} T[i-1][j-1] + 1, & \text{if } X[i] == Y[j] \\ T[i][j-1], & \text{else} \end{cases} \tag{1}$$
>
> for $0 \leq i \leq n-1$ and $0 \leq j \leq m-1$ where n is the length of X and m is the length of Y.
>
> The base cases are:
>
> $$T[i][0] = \begin{cases} 1, & \text{if } X[i] == Y[0] \text{ for } 0 \leq i \leq n-1 \\ 0, & \text{else} \end{cases} \tag{2}$$
>
> $$T[0][j] = \begin{cases} 1, & \text{if } X[0] == Y[j] \text{ for } 1 \leq j \leq m-1 \\ T[0][j-1], & \text{else} \end{cases} \tag{3}$$

(c) Write **pseudocode** for your algorithm to solve this problem.

> **Solution:** Pseudo-code omitted.

(d) Analyze the running time of your algorithm.

> **Solution:** Since we have two nested loops of size $\mathcal{O}(n)$ and $\mathcal{O}(m)$, and filling each entry of the table is $\mathcal{O}(1)$, we get $\mathcal{O}(n) * \mathcal{O}(1) + \mathcal{O}(m) * \mathcal{O}(1) + \mathcal{O}(m) * \mathcal{O}(n) = \mathcal{O}(mn)$

4.) (20 points) Suppose we have a list L containing k base strings. Given a string w of length n, determine if you can split up w into a sequence of base strings. Note that base strings may be re-used.

Example: Consider L = ["georgia", "tech"], and w = "georgiatechgeorgia". Your algorithm would return True. If w was "ttechgeorgia", the algorithm would return False.

(a) Define the entries of your table in words. E.g. $T[i]$ or $T[i, j]$ is ...

Solution: $T[i]$ represents a boolean value that determines whether it is possible to split up $s[1 \ldots i]$ using words in L.

(b) State recurrence for entries of the table in terms of smaller subproblems. Briefly explain in words why it is correct.

Solution: $T[i] = \vee_{a \in L}(s[i - len(a) + 1 \ldots i] == a \wedge T[i - len(a)])$. Each time we calculate $T[i]$ we iterate through words in L to check if it is the rightmost characters in s. If so, we need to check if the previous characters before the word can be split up, which is given by $T[i - len(a)]$.

(c) Write **pseudocode** for your algorithm to solve this problem.

Solution: Pseudocode omitted.

(d) Analyze the running time of your algorithm.

Solution: $\mathcal{O}(nk)$.

5.) (20 points) We call a sequence of integers $a_1, ..., a_n$ *noisy* when the signs of the differences between two consecutive terms in the sequence strictly alternate between + and − (the difference is never zero). So the sequence either follows $a_1 < a_2 > a_3 < a_4 > ...$ or it follows $a_1 > a_2 < a_3 > a_4 < ...$ You are given an array of integers $A = [a_1, ..., a_n]$.

Design a dynamic programming algorithm to find the length of the longest *noisy subsequence* in A.

Example:
Input: $A = [2, 4, -1, -5, -9, 7, 9, 0, 5, 5, -2]$
Output: 7
Explanation: An example of a *noisy subsequence* from the above array A is $2, 4, -1, 9, 0, 5, -2$. There are more examples, but the length is the same for all. On the other hand, $2, 4, 7, 9, 0, 5, 5$ is not a noisy subsequence because the differences between the three consecutive elements 2, 4, and 7 do not alternate. Two 5's also show up at the end of the sequence causing the consecutive difference to be zero.

Hint: a $1 \times n$ table will not be sufficient, but it doesn't have to be an $n \times n$ table!

(a) Define the entries of your table in words. E.g. $T[i]$ or $T[i, j]$ is ...

> **Solution:** Assume the sequence $a_1, a_2, ..., a_n$. Define a 2D table of size $T[n + 1][2]$.
>
> $T[0][j]$ = the longest noisy subsequence ending at index i and last element is greater than its previous element.
>
> $T[1][j]$ = contains the longest noisy subsequence ending at index i and last element is smaller than its previous element.

(b) State recurrence for entries of the table in terms of smaller subproblems. Briefly explain in words why it is correct.

> **Solution:**
>
> $T[0][j] = \max(T[0][j], T[1][k] + 1)$; for all $k < j$ and $a_k < a_j$. Note that $T[1][k]$ is the longest noisy subsequence that ends at index k and a_k is smaller than the previous element we took. Therefore, we need the next element, a_j to be larger than a_k. Out of all indices that satisfy this constraint, we will pick the best (max) one and add 1 as we are appending a_k.
>
> $T[1][j] = \max(T[1][j], T[0][m] + 1)$; for all $m < j$ and $a_m > a_j$. A similar reasoning can be used as above.
>
> The base case is that all table entries should be initialized with value 1 as taking one element is always a valid noisy subsequence.

(c) Write **pseudocode** for your algorithm to solve this problem.

Solution: Pseudo-code omitted.

(d) Analyze the running time of your algorithm.

Solution: Since we must iterate $\mathcal{O}(n)$ times to fill up the table, and filling one entry of the table takes $\mathcal{O}(n)$ time, we get $\mathcal{O}(n) * \mathcal{O}(n) = O(n^2)$.