CS 3510 Section C: Design & Analysis of Algorithms October 13th 2023

HW 6: Dynamic Programming II

YOUR NAME HERE Due: October 20th 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.) (33 points) Suppose you have n balloons, the ith of which has value A_i. In one move, you can pop a balloon at index i and gain $A_{i-1} \times A_i \times A_{i+1}$ points. After that, the ith balloon disappears, the array shrinks by 1, and balloon $i-1$ and $i+1$ are now next to each other and re-indexed appropriately. Design an algorithm to determine the maximum number of points we can earn by popping the balloons in optimal order.

Example: [3, 1, 5, 8] returns 167.

Explanation: $[3,1,5,8] \rightarrow [3,5,7] \rightarrow [3,8] \rightarrow [8] \rightarrow []$. The total points accumulated is $3 \times 1 \times 5 + 3 \times 5 \times 8 + 3 \times 8 + 1 \times 8 \times 1 = 167$.

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

Solution: Let $T[i][j]$ represent the maximum cost of popping all balloons between i and j, but not including i and j.

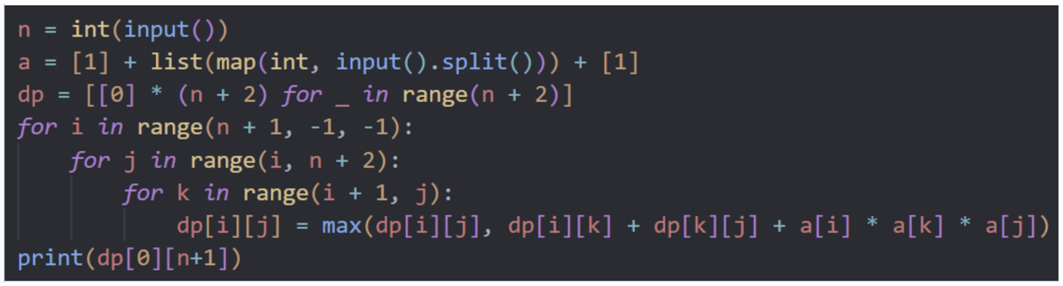
(b) State the base case(s) and recurrence for entries of the table in terms of smaller subproblems. Briefly explain in words why it is correct.

Solution: $T[i][j] = max_{i<k<j}\{T[i][k] + T[k][j] + A_i \times A_k \times A_j\}$

Similar to chain matrix. We will only fill in the top right corner

(c) Give pseudocode for your algorithm. **Note**: iterating through your table might not be straightforward, so feel free to describe the loop(s) in English.

Solution:



```
n = int(input())
a = [1] + list(map(int, input().split())) + [1]
dp = [[0] * (n + 2) for _ in range(n + 2)]
for i in range(n + 1, -1, -1):
    for j in range(i, n + 2):
        for k in range(i + 1, j):
            dp[i][j] = max(dp[i][j], dp[i][k] + dp[k][j] + a[i] * a[k] * a[j])
print(dp[0][n+1])
```

(c) Analyze the running time of your algorithm.

Solution: $\mathcal{O}(n^3)$.

2.) (33 points) Due to a shortage of jute, you have been provided only one piece of mini-rope each of length $s_1, ..., s_n$. You wish to create a long rope of integer length S by tying some of the ropes given. Design a dynamic programming algorithm that returns if it is possible to create a long rope of length exactly S using ropes of lengths $s_1, ..., s_n$. You are allowed to use each rope s_i only once and assume the knot is of length 0, i.e., the knot doesn't consume any of the lengths of either of the rope pieces.

Examples:
Input: $s_1 = 5, s_2 = 7, s_3 = 9, S = 17$
Output: False

Input: $s_1 = 5, s_2 = 7, s_3 = 9, S = 14$
Output: True
Explanation: Use one rope of length s_1, and one rope of length s_3 $5 + 9 = 14$.

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

Solution: $T(i, l)$ is boolean valued and denotes whether a rope of length l can be created from a subset of the mini-ropes of lengths $l_1, ..., l_i$ where each mini-rope is used at most once.

(b) State recurrence for entries of the table in terms of smaller subproblems. Briefly explain in words why it is correct. Remember to state your base case(s) as well.

Solution:

$$T[i][l] = \begin{cases} T[i-1][l-s_i] \cup T[i-1][l], & s_i \leq l \\ T[i-1][l], & s_i > l \end{cases} \tag{1}$$

Base case(s): $T[0][0] = $ True, $T[0][l] = $ False for $1 \leq l \leq S$.

To make a rope of length l using a subset of mini-ropes of lengths $s_1, ..., s_i$, we have two options: we either use the rope of length s_i or we don't. If we use it, then we need to check if we can make a rope of length $l - s_i$ using the ropes of lengths $s_1, ..., s_i - 1$. $T[i-1][l-s_i]$ gives us this answer. On the other hand, if don't use it, then we need to check if we can make a rope of length l using the ropes of lengths $s_1, ..., s_i 1$. $T[i-1][l]$ gives us this answer. Note that in the case that $s_i > l$, we do not have the option to use the rope of length s_i.

(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 fill a table of size $(n+1) * (S+1) = \mathcal{O}(nS)$ and our transitions take constant time, the time complexity is $O(n * S)$.

3.) (34 points) Assume you are taking CS 3510 Exam 3 and have exactly M minutes to answer N questions. You know the point value p_i for each question and have determined that each question will take you t_i time to answer. You also cannot answer more than K questions on your exam. Design a dynamic programming algorithm to determine if you can achieve at least 70% on the exam. Assume that you will answer each question you attempt correctly.

Big Hint: Start by ignoring the K constraint, then try to add it in as a separate dimension.

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

Solution: The first key observation is that we can get at least 70% if and only if our maximum possible score is at least 70%, so we can just consider the maximum score instead. The second observation is that the K questions constraint can just be encoded as a 3^{rd} dimension to our DP table.

$T(i, j, k)$ = The maximum points that can be achieved if we consider questions $1...i$, have spent maximum j time, and have answered maximum k questions.

(b) State recurrence for entries of the table in terms of smaller subproblems. Briefly explain in words why it is correct. Remember to state your base case(s) as well.

Solution:

$$T[i][j][k] = \max \begin{cases} T[i-1][j][k], \\ p_i + T[i-1][j-t_i][k-1] \end{cases} \tag{2}$$

The base case is setting all elements to 0.

There are two cases to consider. We either attempt the question or we don't. If we do not attempt it, then we transition from $T(i-1, j, k)$ as the current time spent and number of questions answered has not changed. If we do take the question, then we transitioned from $T(i-1, j-t_i, k-1)$. Since we want to maximize our score, we take the maximum of both.

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

Solution: Pseudo-code is ommited.

(d) Analyze the running time of your algorithm.

Solution: The running time is simply the dimension of the table, $\mathcal{O}(N * M * K)$.