Theory Assignment-3: ADA Winter-2024

Ritika Thakur (2022408) Sidhartha Garg(202499)

1 Subproblem definition.

The subproblem definition for this code can be described as follows:

For a given slab of dimensions $m \times n$, the subproblem involves determining the maximum profit that can be obtained by cutting the slab into smaller pieces of various dimensions. The goal is to find the optimal way to cut the slab to maximize the total profit, where the profit is defined by the sum of profits of all the smaller pieces obtained after cutting.

The subproblem entails finding the maximum profit achievable for each possible combination of dimensions $i \times j$, where $1 \leq i \leq m$ and $1 \leq j \leq n$, considering all possible ways to cut the slab both horizontally and vertically.

The dynamic programming approach used in the code computes the maximum profit for each possible slab dimension incrementally, considering previously computed results for smaller subproblems. This approach ensures that each subproblem is solved only once and its solution is stored for future reference, leading to an efficient solution for the overall problem.

2 Recurrence of the subproblem.

The recurrence relation for the subproblem described in the code can be defined as follows:

Let $\text{dp}[i][j]$ represent the maximum profit that can be obtained from a slab of dimensions $i \times j$.

The base case for the recurrence is when $i = 1$ and $j = 1$, where $\text{dp}[i][j]$ equals the profit obtained from the single unit slab.

For any other slab dimensions $i \times j$, the recurrence relation is given by:

$$\text{dp}[i][j] = \max \begin{cases} \max(dp[i][j],\ \text{dp}[x][j] + \text{dp}[i-x][j]) & \text{for horizontal cuts} \\ \max(dp[i][j],\ \text{dp}[i][y] + \text{dp}[i][j-y]) & \text{for vertical cuts} \end{cases}$$

where $1 \leq x < i$ and $1 \leq y < j$.

This recurrence relation represents the decision to cut the slab either horizontally or vertically at a specific position x or y to obtain two smaller slabs, and the maximum profit is determined by choosing the option that yields the highest total profit.

3 The specific subproblem(s) that solves the actual problem.

Horizontal Cuts:

- We explore cutting the marble slab from top to bottom, considering different lengths.
- **Subproblem:** Find the best profit by cutting a piece of size (length, original width).

Vertical Cuts:

- We explore cutting the slab from left to right, considering different widths.
- **Subproblem:** Find the best profit by cutting a piece of size (original length, width).

Best Combo:

- For each position, we consider both horizontal and vertical cuts to maximize profit.
- Combine the information from these cuts to decide the best strategy.

Final Result:

- After exploring all possibilities, the maximum profit comes from the table (dp[m][n]).
- This guides us on the best way to cut the entire marble slab for maximum profit.

4 Algorithm description.

Algorithm: Maximum Profit from Cutting Slabs

Input:

- m: Length of the slab
- n: Width of the slab
- costArray: 2D array representing the cost/profit of each unit area of the slab

Output:

- Maximum profit achievable by cutting the slab optimally

Initialize Dynamic Programming Table: Create a 2D array dp[$m+1$][$n+1$] to store the maximum profit for slabs of different dimensions.

Compute Maximum Profit:

- Iterate over each possible length *length* from 1 to m:
- For each length, iterate over each possible width *width* from 1 to n:
  - Initialize dp[*length*][*width*] with the profit obtained from the corresponding unit area of the slab.
  - For each possible cut position i (from 1 to $length - 1$), compute the profit from cutting the slab horizontally:
    - Update dp[*length*][*width*] by taking the maximum between the current value and the sum of profits from two smaller slabs obtained by the cut.
  - For each possible cut position j (from 1 to $width - 1$), compute the profit from cutting the slab vertically:
    - Update dp[*length*][*width*] by taking the maximum between the current value and the sum of profits from two smaller slabs obtained by the cut.

Return Maximum Profit: The maximum profit achievable from cutting the slab optimally is stored in dp[m][n].

Note: The algorithm utilizes dynamic programming to efficiently compute the maximum profit for different dimensions of the slab by considering all possible ways of cutting the slab horizontally and vertically. The optimal solution is obtained by iteratively building upon previously computed subproblems.

5 Pseudocode

Algorithm 1 Maximize Profit

```
function maxProfit(m, n, costArray)
    Initialize dp[m + 1][n + 1] with all elements as 0
    for length ← 1 to m do
        for width ← 1 to n do
            dp[length][width] ← costArray[length][width]
            for i ← 1 to length do
                if dp[i][width] + dp[length − i][width] > dp[length][width] then
                    Output "cut a slab horizontally along i"
                end if
                dp[length][width] ← max(dp[length][width], dp[i][width] + dp[length − i][width])
                Output "slab is of size length × width and profit is dp[length][width]"
            end for
            for i ← 1 to width do
                if dp[length][i] + dp[length][width − i] > dp[length][width] then
                    Output "cut a slab vertically along i"
                end if
                dp[length][width] ← max(dp[length][width], dp[length][i] + dp[length][width − i])
                Output "slab is of size length × width and profit is dp[length][width]"
            end for
        end for
    end for
    return dp[m][n]
end function
```

6 Complexity Analysis

The time complexity of the given function is represented as:

$$\mathcal{O}(m \cdot n \cdot (m+n))$$

The outer for loop runs for m times and which in turn runs an inner loop for n times. The 2 loops further in can run upto at max the value of m and n, thus making the time complexity as above. We can represent this as a polynomial of m+n as :

$$\mathcal{O}((m+n)^3)$$

The space complexity of the algorithm is $\mathcal{O}(m \cdot n)$ because we are using a 2D array of size $(m+1) \times (n+1)$ to store the maximum profit for different dimensions of the slab.

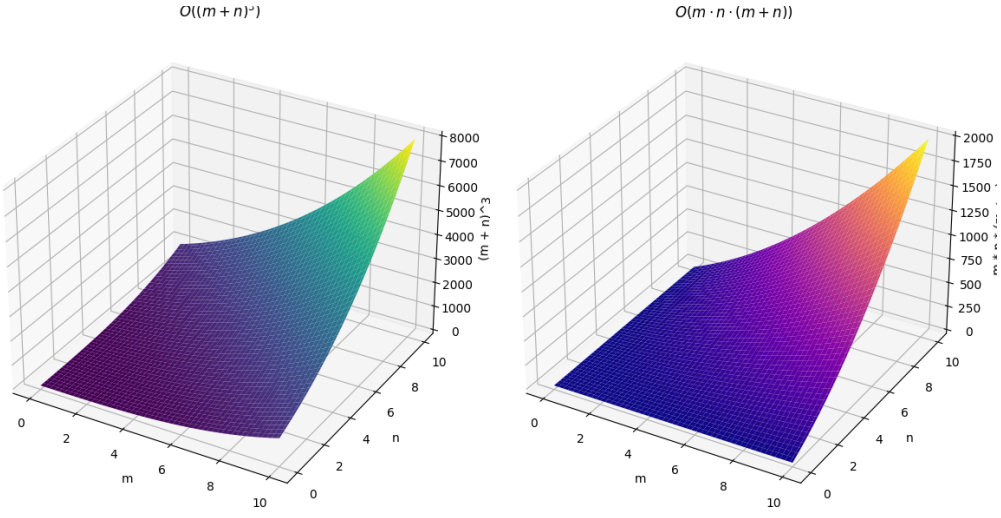

Figure 1: Growth Analysis