Theory Assignment-3: ADA Winter-2024

Vikranth Udandarao (2022570) Ansh Varshney (2022083)

1 Preprocessing and Assumptions

In this problem, we preprocess, assume or have been given the following:

1. We preprocess the input data to parse the dimensions of the marble slab and the spot prices of various marble rectangles.
2. We assume the input dimensions of the marble slab are positive integers.
3. We assume the spot prices of marble rectangles are provided in a 2D array, with each element representing the price of a specific rectangle size.
4. We assume all input values are valid and within the constraints of the problem
5. The marble slab is a rectangle measuring n centimeters in height and m centimeters in width.
6. At any time, you can query the spot price P[x, y] by an x cm by y cm marble rectangle in O(1)-time, for any positive integers x and y.

2 Algorithm Description

The algorithm aims to maximize profit by dividing the marble slab into smaller rectangles of integral pieces while considering the spot prices of different-sized rectangles. It utilizes dynamic programming to solve the problem efficiently. For each sub-rectangle (say of size axb), we consider three possibilities: divide the rectangle into two rectangles by horizontal cutting, divide the rectangle into two rectangles by vertical cutting, or not cutting the rectangle further. The maximum cost is taken out for all these cases (by looking at the prices grid given) and it becomes the optimal cost for the rectangle with size axb. In the algorithm, we have made a 2D dp array that has the dimensions (m+1)x(n+1). Each entry dp[a][b] stores the maximum cost that can be obtained from a rectangle of dimensions axb. We have initially filled the dp with all elements as 0, then for each row (i=1 to m) and each column (j=1 to j=n), we fill the table using the Table Filling (Tabulation) method, i.e. by visiting the previous values of the dp array and writing down the maximum. At the end, we return dp[m][n] which will be the maximum cost obtained for our original rectangle of dimensions mxn.

3 Subproblem Definition

The subproblem in this context involves determining the maximum profit achievable from a marble slab of dimensions $i \times j$. Let $dp[i][j]$ represent the maximum profit that can be obtained from such a marble slab.

4 Recurrence of the Subproblem

The recurrence relation for $dp[i][j]$ is defined as follows:

$$\begin{aligned} dp[i][j] = \max(&\text{spot price of } i \times j \text{ rectangle},\\ &\max_{k=1}^{i-1}(dp[k][j] + dp[i-k][j]),\\ &\max_{k=1}^{j-1}(dp[i][k] + dp[i][j-k])) \end{aligned}$$

This recurrence considers the maximum of profit achieved by not cutting the slab, the profit achievable by making horizontal cuts, and the profit achieved by vertical cuts at different positions.

5 Specific Subproblem to Solve the Actual Problem

The specific subproblem we aim to solve is finding $dp[m][n]$, which corresponds to the maximum profit achievable from the original marble slab of dimensions $m \times n$. By solving this specific subproblem, we effectively determine the optimal way to subdivide the entire marble slab to maximize our profit. This involves considering all possible ways to make horizontal and vertical cuts to partition the slab into smaller integral pieces, while also taking into account the spot prices of each resulting rectangle. The algorithm aims to systematically explore these possibilities and determine the configuration that yields the highest profit.

6 Pseudocode

Algorithm 1 MaximizeProfitMarbleSlab

```
function MaximizeProfit($m, n, P$)    ▷ P: Profit array, m: length, n: width of rectangle
    Initialize $dp[m+1][n+1]$ as a 2D array with all elements initialized to 0
    for $i \leftarrow 1$ to $m$ do    ▷ Iterate over rows
        for $j \leftarrow 1$ to $n$ do    ▷ Iterate over columns
            $hori \leftarrow 0$, $vert \leftarrow 0$, $nocut \leftarrow P[i-1][j-1]$    ▷ Initialize horizontal, vertical, and no-cut profit
            for $k \leftarrow 1$ to $i$ do    ▷ Iterate over possible horizontal cuts
                $hori \leftarrow \max(hori, dp[k][j] + dp[i-k][j])$    ▷ Update horizontal profit
            end for
            for $k \leftarrow 1$ to $j$ do    ▷ Iterate over possible vertical cuts
                $vert \leftarrow \max(vert, dp[i][k] + dp[i][j-k])$    ▷ Update vertical profit
            end for
            $dp[i][j] \leftarrow \max(nocut, \max(hori, vert))$    ▷ Choose maximum profit among no-cut, horizontal cut, and vertical cut
        end for
    end for
    return $dp[m][n]$    ▷ Return maximum profit for given dimensions
end function
```

7 Complexity Analysis

7.1 Time Complexity

The time complexity of the algorithm is $O((mn)^2)$ (polynomial in $O(m+n)$), where m and n are the dimensions of the marble slab. This is because there are two nested loops iterating over the dimensions of the slab, and within each iteration, there are two additional loops iterating over the possible cut positions.

7.2 Space Complexity

The space complexity of the algorithm is $O(mn)$, as we use a 2D array of size $(m+1) \times (n+1)$ to store the maximum profits.

8 Explanation of the Running Time Complexity

The running time complexity of the algorithm arises from the nested loops iterating over the dimensions of the marble slab, i.e. the first loop iterates from i=1 to m, nested with this loop, the second loop runs from j=1 to n. Also, there are further two loops nested (each individually nested) with the original loops. They are required to iterate over all possible cut positions. This results in a quadratic time complexity of $O((mn)^2)$.

9 Proof of Correctness

Proof of Correctness by Induction:

Base Step ($n = 1$): For the base case, we consider a marble slab of dimensions 1×1. In this case, there's only one small rectangle, and the maximum profit is simply the spot price of this rectangle. The algorithm correctly initializes the dp array and returns the spot price of the 1×1 rectangle, which is the correct result.

Inductive Hypothesis: Assume that the algorithm produces the correct result for all inputs up to size $n \times m$.

Inductive Step: We want to show that it also produces the correct result for input size $(n + 1) \times (m + 1)$.

The algorithm iterates through all possible subproblems, considering different ways to cut the marble slab. For each subproblem $dp[i][j]$, it computes the maximum profit achievable. By the inductive hypothesis, we assume that the algorithm correctly computes the maximum profit for all smaller subproblems.

Now, we need to show that considering all possible cuts and maximizing the profit indeed yields the correct result. We can do this by showing that the recurrence relation used in the algorithm correctly captures the optimal solution:

The recurrence relation considers three cases: 1. Not cutting the slab at all and taking the spot price of the whole slab. 2. Making horizontal cuts at different positions and considering the maximum profit achievable by adding the profits of the two resulting subproblems. 3. Making vertical cuts at different positions and considering the maximum profit achievable by adding the profits of the two resulting subproblems.

By considering all possible cuts and maximizing the profit in each case, the algorithm ensures that the maximum profit for the $(n + 1) \times (m + 1)$ slab is correctly computed.

Therefore, by the principle of mathematical induction, the algorithm correctly computes the maximum profit for any given marble slab dimensions.

SAMPLE TESTCASE:

$m = 2$, $n = 3$,

$$\begin{array}{ccc} 2 & 4 & 1 \\ 4 & 1 & 3 \end{array}$$

OUTPUT: 12

Thus, the proof of correctness by induction is established.