Theory Assignment-3: ADA Winter-2024

Krishna Shukla (2022254) Kshitij (2022256)

1 Preprocessing

As such no preprocessing is required but we are assuming array indexing starting from 0 (standard) and the array size is n,

2 Subproblem Definition

The problem can be optimally solved using dynamic programming, which entails breaking it down into smaller, overlapping subproblems:

Subproblem:
Given a rectangular sub-slab of width i in centimeters and height j in centimeters (where $1 \leq i \leq m$ and $1 \leq j \leq n$), find the maximum profit obtainable (OC[i][j]) by cutting this sub-slab into smaller rectangular pieces (with integral dimensions) using horizontal or vertical cuts, considering the spot prices in P.

We can capture the subproblem essence with a recurrence relation:

```
OC[i][j] = min(
   max(OC[x][j] + P[x][j] + OC[i-x][j] for x in range(1, i)),  // Vertical cuts
   max(OC[i][y] + P[i][y] + OC[i][j-y] for y in range(1, j)),  // Horizontal cuts
         P(w,h)
)
```

minimizing the cuts and maximizing the profit (though original statement does not say anything about cuts while defining ideal recurrence we are assuming it)

Explanation:

- OC[i][j] represents the maximum profit obtainable from an i x j sub-slab.
- The inner max functions explore all possible vertical and horizontal cuts within the sub-slab, respectively.
- Each option considers the profit from the resulting smaller slabs (stored in OC) and adds the spot price P[x][j] or P[i][y] depending on the cut direction.
- The final max selects the cut yielding the highest overall profit for the sub-slab.

Visualizing the Subproblem Structure:
Imagine a grid representing the large marble slab, with each cell corresponding to a sub-slab of unit width and height. To calculate OC[i][j], we consider all possible ways to divide the i x j sub-slab into smaller rectangles using horizontal or vertical cuts. The optimal solution for OC[i][j] is the one that maximizes the sum ofOC[i][j], we consider all possible ways to divide the i x j sub-slab into smaller rectangles using horizontal or vertical cuts. The optimal solution for OC[i][j] is the one that maximizes the sum of:

- The profits from the resulting smaller sub-slabs (obtained from OC)
- The spot price of the cut rectangle

3 Recurrence Relation

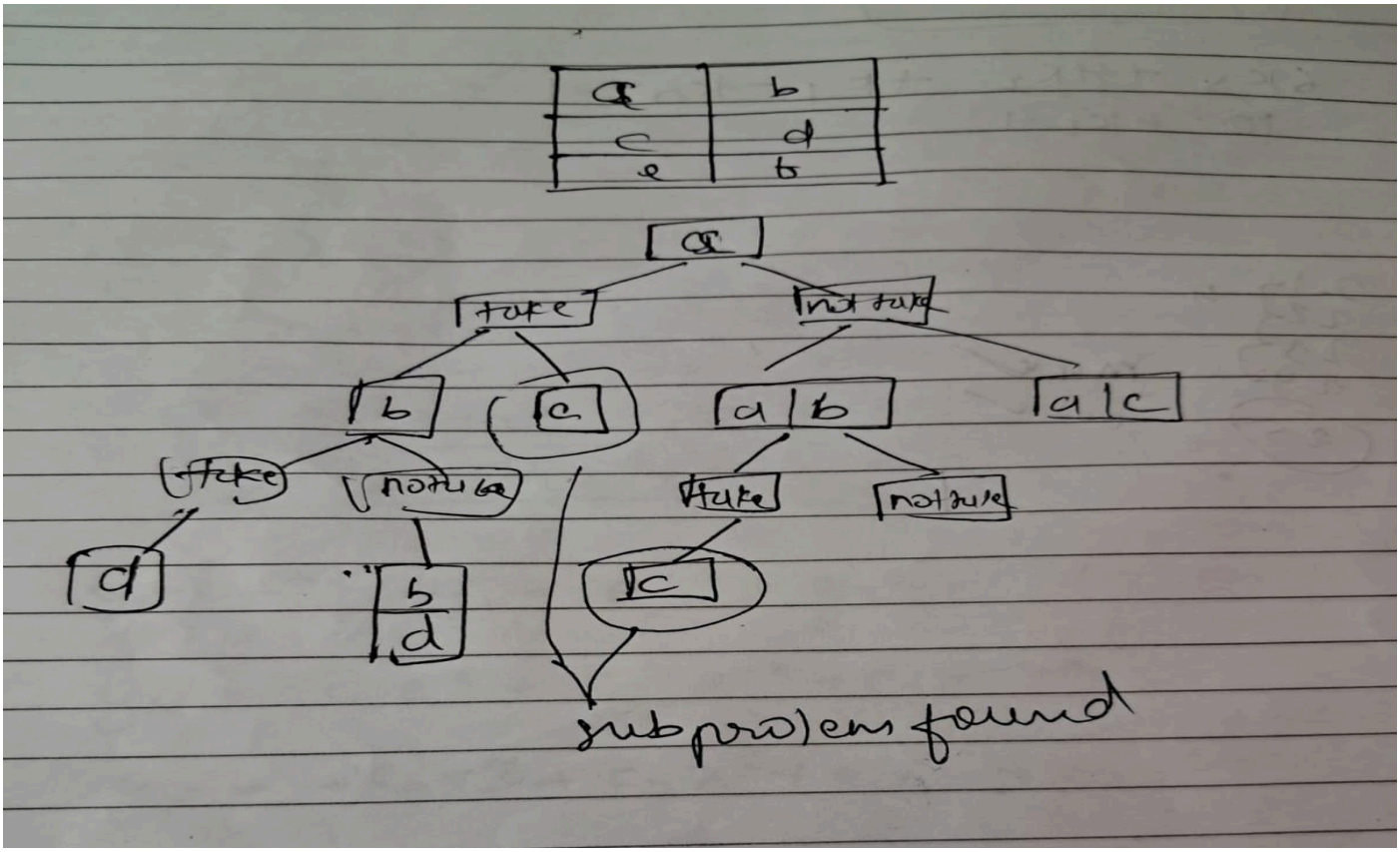
The recurrence relation can be formulated as follows:

```
OC[i][j] = min(
  max(OC[x][j] + P[x][j] + OC[i-x][j] for x in range(1, i)),  // Vertical cuts
  max(OC[i][y] + P[i][y] + OC[i][j-y] for y in range(1, j)),  // Horizontal cuts
      P(w,h))
```

Here, OC[i][j] represents the maximum profit obtainable by subdividing a marble slab of width i and height j.

We have used Iterative solution to code so we cant formally define the recurrence relation for the given case

4 Specific Subproblem(s)



Here a , b , c,d , e ,f denotes every single block each of length unit. We can view this algo as take not take ,we can decide whether to take that single piece of block or not. If we take that single piece of block we will add the cost of that block or if not take then we increase the length of the current block by 1 cm in both horizontal and vertical directions. Now recursively , we have to choose whether to take the block of size (2,1) or not . If we take then we add its cost of marble of size 2 x 1 and then move to next piece .If we not take either we increase the size of the marble by 3,1 or 2,2 because we have given the prices of only square or rectangle shapes not rectangle with triangle. Same with the 1,2 . And while doing this , we sometimes repeats the same problem which is our subproblem which is also defined in above diagram.

Memoizing data structure:There are h×w distinct subproblems, corresponding to different (smaller) sizes of rectangles. To facilitate our work we will store these results.

5 Algorithm Description

Algorithm Overview:

The marble cutting algorithm efficiently determines the maximum achievable profit by cutting a rectangular marble slab of width m and height n into smaller rectangular pieces using horizontal or vertical cuts. It accomplishes this by breaking down the problem into smaller, overlapping subproblems and employing dynamic programming for optimization.

Key Ideas:

- Optimal Decomposition: The maximum profit for the entire slab is achieved by combining the optimal solutions of its subproblems.
- Dynamic Programming: The algorithm avoids redundant calculations by storing intermediate results (maximum profits for sub-slabs) in a 2D array OC.
-

Dynamic Programming Table:

- Create a 2D array OC of size (m + 1) x (n + 1).
- OC[i][j] represents the maximum profit obtainable from a sub-slab of width i and height j.
- Initially, set all OC[i][j] elements to 0 (no cuts performed).

Dynamic Programming Iteration:

- For each sub-slab (index i, j):
- Consider vertical cuts:
- Iterate over all possible vertical cut positions x from 1 to `i - 1.
- Calculate the profit profit_v for each cut:
- Add the profits from the two resulting sub-slabs (OC[x][j] and OC[i - x][j]).
- Add the spot price from P[x][j].

- Compare profit_v with the current maximum in OC[i][j]. If greater, update OC[i][j] with profit_v and store the cut direction ('V') and position (x).
- Consider horizontal cuts:
- Perform similar calculations iterating over y from 1 to j - 1.
- Update OC[i][j] and store the cut direction ('H') and position (y) if a horizontal cut yields a higher profit.

Finding the Maximum Profit:

- After iterating through all sub-slabs, the maximum achievable profit for the entire slab is located in OC[m][n].

6 Running Time Explanation

- The algorithm iterates over each cell in the OC table, which has dimensions (w + 1) x (h + 1).
- This results in a total of (w + 1) * (h + 1) iterations
- For each cell, the algorithm considers all possible cut positions either vertically or horizontally.
- For a given cell OC[i][j], it considers up to i-1 positions for vertical cuts and up to j-1 positions for horizontal cuts.
- The calculation of the maximum profit at each cell involves comparing the profits obtained from different cut positions and selecting the maximum.
- This comparison takes constant time as it involves simple arithmetic operations and comparisons.
- Considering all the factors above, the overall time complexity of the algorithm can be expressed as O(w * h * (w + h)).
- This is because for each cell, we consider up to O(w + h) possible cuts, resulting in a total of O(w * h * (w + h)) operations.

Space Complexity:

- The algorithm utilizes a 2D array OC to store the maximum profit and cut information for each possible width and height combination of the marble slab.
- The size of this table is (w + 1) x (h + 1).
- Hence, the space complexity contributed by the OC table is O(w * h).
- The algorithm uses a few additional variables such as m, s, x, y, L, R, and K for calculations and comparisons.
- These variables require constant space regardless of the input size.
- Considering the OC table and additional variables, the overall space complexity of the algorithm is O(w * h), dominated by the space required for the OC table.

7 Pseudocode

```
function solve(P, w, h):

    OC = 2D array of size (w + 1) x (h + 1)


    for i = 0 to w:
        for j = 0 to h:
            OC[i][j] = (0, None)


    for i = 1 to w:
        for j = 1 to h:
            m = P[i - 1][j - 1]
            s = None


            for x = 1 to i - 1:
                L, _ = OC[x][j]
                R, _ = OC[i - x][j]
                if L + R > m:
                    m = L + R
                    s = ('V', x)


            for y = 1 to j - 1:
                L, _ = OC[i][y]
                R, _ = OC[i][j - y]
                if L + R > m:
                    m = L + R
                    s = ('H', y)


            OC[i][j] = (m, s)


    return OC[w][h]
```

8 Proof of Correctness

Intuition based proof by Contradiction

1. Assume the opposite: There exists a cutting pattern different from the one determined by the algorithm that yields a higher profit.

2. This alternative cutting pattern can be broken down into a series of horizontal and vertical cuts.

3. Each individual cut in this alternative pattern must divide the slab into two smaller sub-slabs.

4. For each of these sub-slabs, the algorithm already considers all possible horizontal and vertical cuts during its iterations.

5. If any cut in the alternative pattern yielded a higher profit than the optimal cut calculated by the algorithm for the corresponding sub-slab, it would contradict the optimality of the algorithm's calculation for that sub-slab.

6. Since all sub-slabs in the alternative pattern are covered by the algorithm's calculations, and no cut in the alternative pattern can be demonstrably better than the algorithm's optimal cut for its corresponding sub-slab, the alternative pattern cannot generate a higher overall profit.

Therefore, if we assume the existence of a better cutting pattern, we arrive at a contradiction, proving that the original claim (the algorithm finds the maximum profit) must be true.