Theory Assignment-3: ADA Winter-2024

Vivaswan Nawani (2021217) Animesh Pareek (2021131)

1 Subproblem Definition

- We need only one subproblem definition.
- Max_profit(i, j) is the maximum profit that we can earn from selling the marble (after some cuts along breadth and length) which is i centimeters long and j centimeters wide.
- We need to note that this marble, whose sides are i and j cm(s), can be cut vertically or horizontally in multiples of 1 cm.

2 Recurrence of the subproblem

ASSUMPTION:

- The shape of the Spot Price array (Array P) is (n x m) where n and m are at least 1.
- First index of this array refers to the height of the marble while the second index represents the breadth of the marble.
- We assume that P(i,j) is not necessarily same as P(j,i)
- Also please note that we will be using 1 based indexing.

Base Cases:

$$\texttt{Max\_profit(i, j) =} \begin{cases} \text{P(1,1)} & \text{if } i = 1 \text{ and } j = 1 \\ 0 & \text{if } i < 1 \text{ or } j < 1 \text{ or } i > n \text{ or } j > m \end{cases}$$

Recurrence case: For any index $1 < i < |P|$ and $1 < j < |P(0)|$ where $i, j \in N$, we have the following recurrence relation:

$$\texttt{Max\_profit(i, j) = max of } \begin{cases} \text{P}(i,j) \\ \texttt{Max\_profit}(i-k, j) + \texttt{Max\_profit}(k, j) & \text{for all k} \in [\ 1\ ,\ \lfloor i/2 \rfloor\ ]\ \text{(only integers)} \\ \texttt{Max\_profit}(i, j-k) + \texttt{Max\_profit}(i, k) & \text{for all k} \in [\ 1\ ,\ \lfloor j/2 \rfloor\ ]\ \text{(only integers)} \end{cases}$$

Also, please note here that, we are considering k only from 1 to floor(j/2) (or i/2 , as a matter of fact) because after floor(j/2) to j-1 we will start repeating the cases that we have already accounted. (sum's first term of this sequence can be related to the second term of the accounted sequence and vice versa)

3 The specific subproblem(s) that solve the actual problem

- The actual problem is Max_Profit(n, m)
- We need to solve $(n-1)+(m-1)$ subproblems to solve the actual problem.
- These subproblems are Max_Profit(n, i), $1 \le i \le m-1$, and Max_Profit(j, m), where $1 \le j \le n-1$.
- This is explained by the recurrence relation above where (i - 1) + (j - 1) calls are made in total for Max_Profit(i, j)

$$\texttt{Max\_profit(n, m) = max of} \begin{cases} \texttt{P}(n,m) \\ \texttt{Max\_profit}(n-k,m)+\texttt{Max\_profit}(k,m) & \text{for all k} \in [\ 1\ ,\ \lfloor n/2 \rfloor\ ]\text{ (only integers)} \\ \texttt{Max\_profit}(n,m-k)+\texttt{Max\_profit}(n,k) & \text{for all k} \in [\ 1\ ,\ \lfloor m/2 \rfloor\ ]\text{ (only integers)} \end{cases}$$

4 Algorithm description

- Input:
  - n, the height of the marble
  - m, the breadth of the marble
  - $P[i][j]$ $0 \leq i \leq n-1$,$0 \leq j \leq m-1$, which store the Spot Price of the marble of size (i x j).
- Let $DP[i][j]$, $0 \leq i \leq n-1$ and $0 \leq j \leq m-1$, store the max possible Profit that we can earn from marble of size (i x j).
- Let's then define 2 nested inner loops, which uses integers i (varying from 0 to n - 1) and j (varying from 0 to m - 1), respectively. This nest is needed for filling up the array DP.Thus we have:
- for(i <- 0 to n-1):
  - for(j <- 0 to m-1):
    * Initialize $maxy = P[i][j]$ where $maxy$ represents the maximum profit that can be earned from marble of size (i x j).
    * Now run a loop which uses integer k varying from 1 to $\lfloor i/2 \rfloor$.
    * for(k <- 1 to $\lfloor i/2 \rfloor$):
      · Initialize $sum = DP[i-k][j] + DP[k][j]$ where sum represents the maximum profit that can be earned by splitting the marble in the sizes (k x j) and ($(i-k)$ x j).
      · In Next step, $maxy = max(maxy, sum)$, we choose the maximum of previous profit and the new profit after splitting at k horizontally.
    * Now run a loop which uses integer k varying from 1 to $\lfloor j/2 \rfloor$.
    * for(k <- 1 to $\lfloor j/2 \rfloor$):
      · Initialize $sum = DP[i][j-k] + DP[i][k]$ where sum represents the maximum profit that can be earned by splitting the marble in the sizes (i x k) and (i x $(j-k)$).
      · In Next step, $maxy = max(maxy, sum)$, we choose the maximum of previous profit and the new profit after splitting at k vertically.
    * Now we finally do $DP[i][j] = maxy$. In this step we write this computed maximum profit ($maxy$) to its correct destination ($DP[i][j]$).
- Output $DP[n-1][m-1]$

5 Complexity Analysis

Let $T(n,m)$ denote the time complexity of the algorithm for the input array P of dimension (n x m).

- **Base Cases**: Setting up the base cases takes constant time as they involve simple conditional assignments, $O(1)$.
- **Recurrence Relation Evaluation**: Evaluating the recurrence relation involves computing each entry of the DP array. This is mainly done via our nested loop's execution which works for $n*m$ times. For each entry $DP[i][j]$, we make some comparisons:
  - **Initialization**: Initializing the $maxy$ variable requires DP array access which takes $O(1)$ time.
  - Each entry in the array involves computation in the comparing and selecting the maximum profit out of ($\lfloor i/2 \rfloor$ + $\lfloor j/2 \rfloor$) choices.

  - Each selection requires summing up of two values which can be done in constant ($O(1)$) time.
  - These values are DP array accesses which again can be done in constant ($O(1)$) time.
  - Also Each comparison can be done in constant ($O(1)$) time.
  - Thus the operation in the nested loop takes $O(n + m)$ time.

  Therefore, the time complexity of evaluating the recurrence relation (with Nested Loop in consideration) is $O((n * m) * (n + m)) = O((n^2)m + (m^2)n)$.

- **Find end Result**: At the end we return $DP[n - 1][m - 1]$ which takes $O(1)$ time.

Therefore, the overall time complexity $T(n, m)$ of the algorithm is:

$$T(n, m) = O(n * m) * O(n + m) = O(n * m * (n + m))$$

$$T(n, m) = O(n * m * (n + m))$$