Theory Assignment-3: ADA Winter-2024

Aditya Sharma (2022038) Ayan Kumar Singh (2022122)

1 Algorithm description

- The algorithm uses dynamic programming to solve the given problem. It employs top-down approach with memoization to avoid redundant computations and improve efficiency.
- The function takes the slab length, width, memoization matrix, and spot price matrix as input. Spot prices represent the price of selling a slab of that dimension directly.
- If length or width is 0, max profit is 0. This is stored in the memoization matrix. Also, the maximum profit is initialized to 0 to store the best solution found so far.
- It considers two types of cuts - horizontal and vertical. For vertical cuts, it tries all possible cut positions from 1 to width-1. For each cut position, it recursively calls the function on the two rectangles formed and takes the sum of their maximum profits. This sum is compared to the current maxProfit and updated if greater. It operates similarly for the horizontal cuts.
- Additionally, it compares the spot price of the entire slab against maxProfit. Selling directly can sometimes give more profit.
- Finally, the memoization table is updated with the maximum profit found after exploring all options. This stored result is returned.
- By considering all possible cut orientations and positions, the optimal solution is found and stored for future use. The memoization provides exponential time savings compared to a naive recursive solution.

2 Subproblem definition

For a slab of size $i \times j$ where $i < m$ and $j < n$, we have the problem $M[i][j]$. The problem can be reduced by exercising the following three choices:

1. Take the whole slab: $M[i][j]$.
2. Cut the slab horizontally: $M[i][j-k] + M[i][k]$. For this, we find all horizontal cuts and determine which maximizes the profit.
3. Cut the slab vertically: $M[i-k][j] + M[k][j]$. For this, we find all vertical cuts and determine which maximizes the profit.

3 The specific subproblem(s) that solves the final problem (or gives the final answer)

- Exploration of all three the choices at each subproblem $M[i][j]$, where $i < m$ and $j < n$ enables us to explore all possible combinations to maximize the profit.The recursive approach paired up with memoization reduces redundant calculations for maximumProfit possible for slab of size M[i][j].

4 Recurrence of the subproblem

Let $M[n, m]$ represent the maximum profit that can be obtained by subdividing an $n \times m$ marble slab. Then, using the provided spot price function $p(x, y)$, we observe that if $n = 0$ or $m = 0$, then $M[n, m] = 0$; otherwise, we have that $M[n, m]$ is:

$$\max \left\{ \begin{array}{ll} p(n,m), & \text{for } n,m > 0 \\ \max_{1 \le i \le \lfloor n/2 \rfloor} \{M[i,m] + M[n-i,m]\}, & \text{for } n > 0, m > 0 \\ \max_{1 \le j \le \lfloor m/2 \rfloor} \{M[n,j] + M[n,m-j]\}, & \text{for } n > 0, m > 0 \end{array} \right\}$$

5 Complexity Analysis

Our algorithm uses dynamic programming using the memoization approach, for implementing the memoization approach we have created a 2-D Array of dimensions $n + 1$, $m + 1$. Creating such an array would require $(n + 1) \times (m + 1)$ auxiliary space thus the space complexity of our algorithm is:

$$\mathbf{O(n * m)}$$

The number of sub-problems to our algorithm is determined by the size of our memoization table which is a 2D-Array of dimensions $n + 1$, $m + 1$. where each entry stores a unique value. This in turn implies that the number of unique sub-problems are

$$O((n + 1) \times (m + 1)) = O(n * m)$$

Also the time to solve each sub-problem depends upon the range of the for-loop they reside in. The loops have range m and n, and thus time taken to solve the sub-problem would be

$$O(n + m)$$

Hence, the overall time complexity of our algorithm really comes down to the size of the memoization table and the time taken to solve each sub-problem and since the size of the table was $O(n * m)$ and the calculation for sub-problem takes $O(n + m)$ time the complexity of our algorithm becomes

$$\mathbf{O((n * m)(n + m))}$$

.

If $m >> n$ or $n >> m$ then the time complexity can we estimated to be :

$$\mathbf{O((m * n)min(m, n))}$$

.

6 PseudoCode

Algorithm 1 ComputeMaximumProfit

procedure ComputeMaximumProfit($length$, $width$, $memoizationMatrix$, $prices$)
  if $memoizationMatrix[length][width] \neq -1$ **then**
    return $memoizationMatrix[length][width]$
  end if
  if $length = 0$ **or** $width = 0$ **then**
    return $memoizationMatrix[length][width] = 0$
  end if
  $maxProfit \leftarrow 0$
  for $verticalCut \leftarrow 1$ **to** $width - 1$ **do**
    $maxProfit \leftarrow \max(maxProfit, ComputeMaximumProfit(length, verticalCut, memoizationMatrix, prices)$
      $+ComputeMaximumProfit(length, width - verticalCut, memoizationMatrix, prices))$
  end for
  for $horizontalCut \leftarrow 1$ **to** $length - 1$ **do**
    $maxProfit \leftarrow \max(maxProfit, ComputeMaximumProfit(horizontalCut, width, memoizationMatrix, prices)$
      $+ComputeMaximumProfit(length - horizontalCut, width, memoizationMatrix, prices))$
  end for
  if $prices[length - 1][width - 1] > maxProfit$ **then**
    $maxProfit \leftarrow prices[length - 1][width - 1]$
  end if
  return $memoizationMatrix[length][width] = maxProfit$
end procedure

7 Main Function

1. This is the main function in C++ that we can use to drive the algorithm
2. The algorithm assumes 0-based indexing.
3. memoziationMatrix of size (n + 1) x (m + 1) is initialized to -1 before passing the alogirithm.

```
int main() {
    int n = 3, m = 4; // Dimensions of the marble slab
    vector<vector<int>> memoizationMatrix(n + 1, vector<int>(m + 1, -1))
        ;
    vector<vector<int>> spotPrices = {
        {1,2,3,4},
        {2,5,6,7},
        {3,6,9,8}
    };

    int maximumProfit = ComputeMaximumProfit(n, m, memoizationMatrix,
                                             spotPrices);
    cout << "Maximum profit: " << maximumProfit << endl;
    return 0;
}
```