Theory Assignment-3: ADA Winter-2024

Adarsh Jha (2022024) Akshat Kothari (2022053)

18 February 2024

1 Introduction

You have mined a large slab of marble from a quarry, and you want to cut it into smaller rectangles to maximize profit. The marble slab has dimensions n centimeters in height and m centimeters in width. Spot prices $P[x, y]$ can be queried for any x cm by y cm marble rectangle. The goal is to describe an algorithm that subdivides the marble slab into integral pieces to maximize profit.

2 Algorithm: COMPUTE-M

Algorithm 1 COMPUTE-M Algorithm

```
1:  function COMPUTE-M(n, m, p, M)
2:      if M[n][m] ≠ NIL then
3:          return                                   ▷ Avoid recomputing
4:      else if n = 0 or m = 0 then
5:          M[n][m] ← 0                              ▷ Base Case
6:      else
7:          max ← 0
8:          for i ← 1 to ⌊n/2⌋ do                    ▷ Check all horizontal cuts
9:              COMPUTE-M(i, m, p, M)
10:             COMPUTE-M(n − i, m, p, M)
11:             if M[i][m] + M[n − i][m] > max then
12:                 max ← M[i][m] + M[n − i][m]
13:             end if
14:         end for
15:         for j ← 1 to ⌊m/2⌋ do                    ▷ Check all vertical cuts
16:             COMPUTE-M(n, j, p, M)
17:             COMPUTE-M(n, m − j, p, M)
18:             if M[n][j] + M[n][m − j] > max then
19:                 max ← M[n][j] + M[n][m − j]
20:             end if
21:         end for
22:         if p(n, m) > max then                    ▷ Sell the entire slab?
23:             max ← P(n, m)
24:         end if
25:         M[n][m] ← max
26:     end if
27:     return M[n][m]
28: end function
```

3 Preprocessing

The preprocessing step involves querying spot prices for all possible rectangle dimensions and storing the results in the array p. This step ensures constant-time access to spot prices during the algorithm's execution.

4 Recurrence Relation

The recurrence relation for the algorithm is given by:

$$T\left(\frac{n}{2}, m\right) + T\left(n, \frac{m}{2}\right) + O\left(\frac{n}{2}\right) + O\left(\frac{m}{2}\right) + O(1)$$

Where:

- $T(n, m)$ is the time complexity of the algorithm for a problem of size $n \times m$.
- The recursive calls $T\left(\frac{n}{2}, m\right)$ and $T\left(n, \frac{m}{2}\right)$ represent the checks for horizontal and vertical cuts, respectively.
- $O\left(\frac{n}{2}\right)$ and $O\left(\frac{m}{2}\right)$ represent the work done in checking cuts.
- $O(1)$ represents the constant-time spot price query.

5 Complexity Analysis

5.1 Time Complexity

- The recursive calls contribute $\log n + \log m$ levels, assuming each dimension is halved in each recursive call.
- The work done in checking cuts contributes $O(n) + O(m)$.
- The constant-time spot price query contributes $O(1)$.

Overall Time Complexity: $O((\log n + \log m) \cdot (n + m))$

5.2 Space Complexity

- The space complexity is influenced by the recursive call stack, contributing $O(\log n + \log m)$.
- The memoization table M contributes $O(n \cdot m)$.

Overall Space Complexity: $O(n \cdot m)$

6 Assumptions

In the context of the marble subdivision problem, the following assumptions are made:

1. **Rectangular Slab Dimensions:** The algorithm assumes that the dimensions of the marble slab are positive integers, with the height denoted by n centimeters and the width by m centimeters. Negative or non-integer dimensions are not considered.
2. **Spot Prices:** The algorithm assumes that spot prices for marble rectangles can be queried in $O(1)$ time for any given dimensions. This assumes efficient data structures or preprocessed information for spot prices.
3. **Profit Maximization Objective:** The algorithm is designed with the objective of maximizing profit through the subdivision of the marble slab. Profit is defined based on the sum of spot prices for individual rectangular pieces.
4. **Spot Price Dependence:** The algorithm does not make assumptions about the behavior of spot prices. It is agnostic to the specific characteristics of spot prices, including any dependencies on the dimensions of the rectangles.

5. **Spot Price Precision:** The algorithm assumes that spot prices are provided with sufficient precision to differentiate between different configurations. This ensures accurate comparison and selection of profitable cuts.

6. **Space and Time Complexity Model:** The complexity analysis assumes a standard computational model where basic arithmetic operations, array access, and spot price queries take constant time. Deviations from this model may impact the provided complexities.

7. **Rectangular Cuts:** The algorithm assumes that cuts can only be made along the boundaries of the marble slab, resulting in rectangular pieces. Diagonal cuts or irregular shapes are not considered.

8. **No Constraints on Cut Dimensions:** The algorithm does not impose constraints on the dimensions of the cut rectangles, except that they should be positive integers. It explores all possible combinations of cuts.

9. **Initial Memoization Table Values:** The algorithm initializes a memoization table to store computed values. It assumes that the initial values in the table are set to a special marker (e.g., `NIL`) to identify uncomputed cases.

10. **Threshold for Selling Entire Slab:** The algorithm considers selling the entire marble slab as a potential option. It determines whether selling the entire slab is more profitable than the subdivided pieces based on spot prices.

These assumptions collectively provide a framework for the algorithm to address the marble subdivision problem while considering various dimensions and spot prices.

7 C++ Code

```
#include <iostream>
#include <vector>
#include <cmath>
#include <climits>
#include <chrono>
using namespace std;
using namespace chrono;


// Function to take input for spot prices
void inputSpotPrices(vector<vector<long long>>& prices, int n, int m) {
    cout << "Enter spot prices for the marble slab:" << endl;
    for (int i = 0; i < n; ++i) {
        cout << "Row " << i + 1 << ": ";
        for (int j = 0; j < m; ++j) {
            cin >> prices[i][j];
        }
    }
}

// Recursive function to compute maximum profit
long long computeM(int n, int m, vector<vector<long long>>& M, vector<vector<long long>>& prices) {
    if (M[n][m] != -1) {
        return M[n][m]; // Avoid recomputing
    } else if (n == 0 || m == 0) {
        M[n][m] = 0; // Base case
    } else {
        long long maxProfit = 0;
```

```
        // Check all horizontal cuts
        for (int i = 1; i <= floor(n / 2); ++i) {
            long long profit = computeM(i, m, M, prices) + computeM(n - i, m, M, prices);
            if (profit > maxProfit) {
                maxProfit = profit;
            }
        }

        // Check all vertical cuts
        for (int j = 1; j <= floor(m / 2); ++j) {
            long long profit = computeM(n, j, M, prices) + computeM(n, m - j, M, prices);
            if (profit > maxProfit) {
                maxProfit = profit;
            }
        }

        // Should we sell the entire slab?
        if (prices[n-1][m-1] > maxProfit) {
            maxProfit = prices[n-1][m-1];
        }

        M[n][m] = maxProfit;
    }

    return M[n][m];
}

int main() {
    int n, m; // Dimensions of the marble slab
    cout << "Enter the dimensions of the marble slab (n m): ";
    cin >> n >> m;

    // Initialize memoization table with NIL values
    vector<vector<long long>> memoTable(n + 1, vector<long long>(m + 1, -1));

    // Initialize spot prices vector
    vector<vector<long long>> spotPrices(n, vector<long long>(m, 0));

    // Call function to input spot prices
    inputSpotPrices(spotPrices, n, m);

    // Start measuring time
    auto start = high_resolution_clock::now();

    long long maxProfit = computeM(n, m, memoTable, spotPrices);

    // Stop measuring time
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<milliseconds>(stop - start);

    cout << "Maximum profit: " << maxProfit << endl;
    cout << "Time elapsed: " << duration.count() << " milliseconds" << endl;

    return 0;
}
```

8 Conclusion

The algorithm efficiently subdivides the marble slab to maximize profit by exploring all possible cuts and considering spot prices. The time and space complexities provide insights into the algorithm's scalability and resource requirements.