ADA-2024: Homework 3

Rachit Arora, 2022384
Mahi Mann, 2022272

February 19, 2024

Note: Everything except the actual code implementation uses 1-indexed arrays. Also, it is assumed that P is the spot price function if not explicitly stated otherwise.

1 Subproblem Definition

Define $\textsc{Mxp}(y,\, x)$ as the maximum price possible to tile a y-centimeter in height by x-centimeters in width marble slab.

2 Recurrence of the subproblem

Trivially, we have the base-cases

$$\textsc{Mxp}(y,\, 0) = 0$$
$$\textsc{Mxp}(0,\, x) = 0$$

Otherwise,

$$\textsc{Mxp}(y,\, x) = \max\left(\mathrm{P}[y,\, x], \max_{1\le i\le y-1}\left(\textsc{Mxp}(i,\, x) + \textsc{Mxp}(y-i,\, x)\right), \max_{1\le j\le x-1}\left(\textsc{Mxp}(y,\, j) + \textsc{Mxp}(y,\, x-j)\right)\right)$$

See footnote [1] for possible edge-cases.

[1] Here, we are assuming that a max function on an empty interval returns an arbitrarily low integer (possibly negative). This is done so that we can express the recurrence as a single equation, saving us the painful process of specifying every case.

3 Subproblem that solves the actual problem

$$\text{MXP}(n,\ m)$$

For an n by m sized slab.

4 Algorithm Description

4.1 Idea

Suppose there is no splitting in the y by x slab. Then, the maximum possible price is simply $\text{P}[y,\ x]$, which is the spot price.

Otherwise, there is either a horizontal splitting or a vertical splitting, giving us two subproblems for each position of splitting lines. We can take the sum of these two subproblems and take the maximum of them over all splitting points, giving us the recurrences as above.

4.2 Pseudocode

```
function MaxPrice(P, $n$, $m$)
    Mxp $\leftarrow$ 2D-array of size $n \times m$
    $i \leftarrow 1$
    while $i \leq n$ do
        $j \leftarrow 1$
        while $j \leq m$ do
            Mxp[$i$][$j$] $\leftarrow$ P[$i$][$j$]
            $j \leftarrow j + 1$
            $y \leftarrow 1$
            while $y \leq j - 1$ do
                Mxp[$i$][$j$] $\leftarrow$ max(Mxp[$i$][$j$],Mxp[$i$][$y$] + Mxp[$i$][$j - y$])
                $y \leftarrow y + 1$
            end while
            $x \leftarrow 1$
            while $x \leq i - 1$ do
                Mxp[$i$][$j$] $\leftarrow$ max(Mxp[$i$][$j$], Mxp[$x$][$j$] + Mxp[$i - x$][$j$])
                $x \leftarrow x + 1$
            end while
        end while
        $i \leftarrow i + 1$
    end while
    return Mxp[$n$][$m$]
end function
```

MaxPrice(P, n, m) returns the answer for a given spot price function P and an n by m sized slab.

5 Running time

For a fixed i and j in the pseudocode, a single iteration has $O(y) = O(m)$ iterations for vertical splits, and $O(x) = O(n)$ iterations for horizontal splits. Therefore, for all iterations combined, there are

$$O(mn)(O(m) + O(n))O(1) = O(mn)O(m + n) = O(mn(m + n))$$

constant time operations in the algorithm. **Thus, the algorithm runs in $O(mn(m + n))$ time.**

6 Addendum 1: C++ implementation

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){

    int n,m;
    cin >> n >> m;

    // spot prices
    int p[n+1][m+1];
    // max price
    int mxp[n+1][m+1];

    // base cases: spot prices for empty stuff is 0
    for(int i = 0; i < n+1; ++i){
        p[i][0] = 0;
        mxp[i][0] = 0;
    }
    for(int i = 0; i < m+1; ++i){
        p[0][i] = 0;
        mxp[0][i] = 0;
    }

    //
    for(int i = 1; i <= n; ++i){
        for(int j = 1; j <= m; ++j){
            cin >> p[i][j];
            mxp[i][j] = p[i][j];
        }
    }

    for(int i = 1; i < n+1; ++i){
        for(int j = 1; j < m+1; ++j){
            // vertical splitting
            for(int y = 1; y < j; ++y){
                mxp[i][j] = max(mxp[i][j], mxp[i][y] + mxp[i][j-y]);
            }
            // horizontal splitting
            for(int x = 1; x < i; ++x){
                mxp[i][j] = max(mxp[i][j], mxp[x][j] + mxp[i-x][j]);
            }
        }
    }

    cout << mxp[n][m] << '\n';

}
```

7 Addendum 2: Tests

-
  ```
  n = 2
  m = 3
  p = {{2,4,1},{4,1,3}}

  output: 12
  comments: 6 1x1 tiles
  ```

-
  ```
  n = 1
  m = 8
  p = {{10,10,10,1000,1000,10,10,10}}

  output: 2000
  comments: two 1x4s next to each other
  ```

8 People discussed with

Swapnil Panigrahi and Rohak Kansal