Editorial: 304. Range Sum Query 2D – Immutable

Naive solution: $O(n^3)$

This is the solution that comes to the mind to a beginner coder. Run over the rectangle $\{\{\text{row1}, \text{col1}\}, \{\text{row2}, \text{col2}\}$ and compute the sum s :

$s=\sum_{i=row1}^{row2}\sum_{j=col1}^{col2} M_{ij}$ where M is a $n\times m$ matrix.

Example (given in problem description):

$$M=\begin{bmatrix}3 & 0 & 1 & 4 & 2\\ 5 & 6 & 3 & 2 & 1\\ 1 & 2 & 0 & 1 & 5\\ 4 & 1 & 0 & 1 & 7\\ 1 & 0 & 3 & 0 & 5\end{bmatrix}$$ let's say that we wanna make the sum of the rectangle $\{\{2, 1\}, \{4, 3\}\}$

$$M=\begin{bmatrix}3 & 0 & 1 & 4 & 2\\ 5 & 6 & 3 & 2 & 1\\ 1 & \boxed{2 \;\; 0 \;\; 1} & & & 5\\ 4 & 1 & 0 & 1 & 7\\ 1 & 0 & 3 & 0 & 5\end{bmatrix}$$

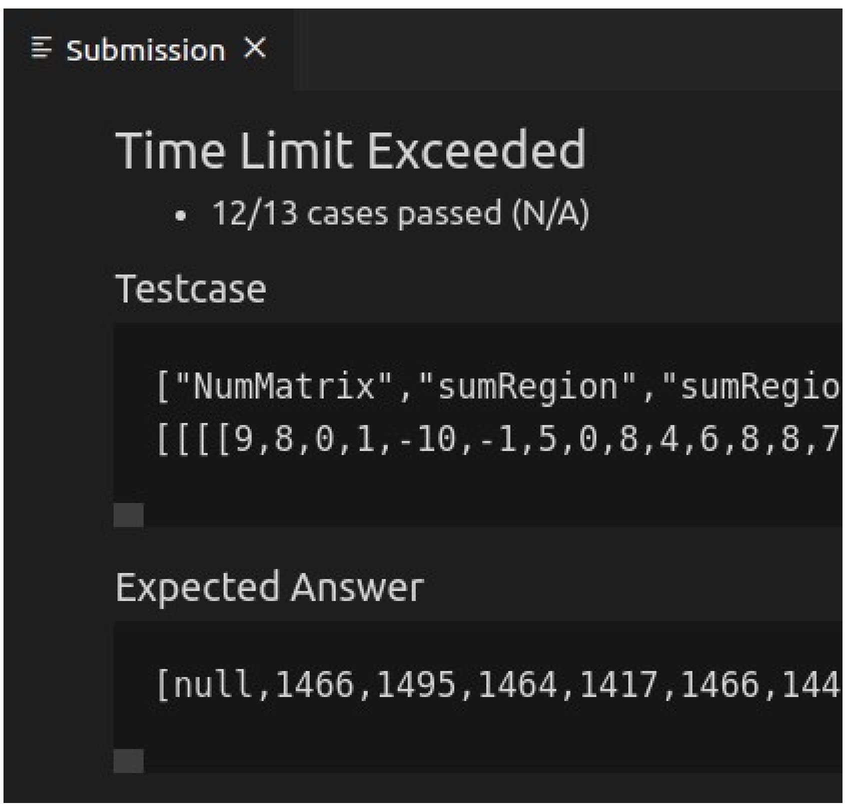
$$\begin{aligned}
s&=\sum_{i=row1}^{row2}\sum_{j=col1}^{col2} M_{ij}\\
&=\sum_{i=2}^{4}\sum_{j=1}^{3} M_{ij}\\
&=(M_{21}+M_{22}+M_{23})+(M_{31}+M_{32}+M_{33})+(M_{41}+M_{42}+M_{43})\\
&=(2+0+1)+(1+0+1)+(0+3+0)\\
&=8
\end{aligned}$$

C++ code: O(n²)

```
class NumMatrix {
    public:
        vector<vector<int>> m;
public:
    NumMatrix(vector<vector<int>>& matrix) {
        m = matrix;
    }

    int sumRegion(int row1, int col1, int row2, int col2) {
        int s = 0;
        for (int i = row1 ; i <= row2 ; ++i)
            for (int j = col1 ; j <= col2 ; ++j)
                s += m[i][j];
        return s;
    }
};
```

We have a matrix M of size $n \times m$. In the worst case the time complexity of the function $sumRegion()$ is $O(n \times m)$, but imagine that it's called k times in the program, the whole time complexity will be $O(k \times n \times m)$. In general is a $O(n^3)$ with this code, the judge return a TLE:



2D-Array with cumulative sum: O(1)

How this is came to my mind?

The range query sum of an 1D-array is very basic in competitive programming. This kind of problem is solved by using a *cumulative sum array*, when the <u>original array values don't change</u>. The cumulative sum array is useless when the <u>original array's values change</u>, a *Fenwick Tree* called too a *Binary Indexed Tree* or a *Segmentation Tree* is used is this case (May be I'll to a whole tutorial on these data structures).

But, here we have a 2D-array, the first intuition is to do the cumulative sum of the original matrix, like a 1D-Array:

$$M=\begin{bmatrix} 3 & 0 & 1 & 4 & 2 \\ 5 & 6 & 3 & 2 & 1 \\ 1 & 2 & 0 & 1 & 5 \\ 4 & 1 & 0 & 1 & 7 \\ 1 & 0 & 3 & 0 & 5 \end{bmatrix} \implies M=\begin{bmatrix} 3 & 3 & 4 & 8 & 10 \\ 15 & 21 & 24 & 26 & 27 \\ 28 & 30 & 30 & 31 & 36 \\ 40 & 41 & 41 & 42 & 49 \\ 50 & 50 & 53 & 53 & 58 \end{bmatrix}$$

you will quick notice that will not work. Because it gives the sum only by rows start always from $\{0,0\}$. For example, if we look for the sum of all numbers from $\{\{0, 0\}, \{4, 3\} = 53$:

$$M=\begin{bmatrix} 3 & 3 & 4 & 8 & 10 \\ 15 & 21 & 24 & 26 & 27 \\ 28 & 30 & 30 & 31 & 36 \\ 40 & 41 & 41 & 42 & 49 \\ 50 & 50 & 53 & 53 & 58 \end{bmatrix}$$

or, the sum of all numbers from $\{\{0, 0\}, \{1, 2\} = 24$:

$$M=\begin{bmatrix} 3 & 3 & 4 & 8 & 10 \\ 15 & 21 & 24 & 26 & 27 \\ 28 & 30 & 30 & 31 & 36 \\ 40 & 41 & 41 & 42 & 49 \\ 50 & 50 & 53 & 53 & 58 \end{bmatrix}$$

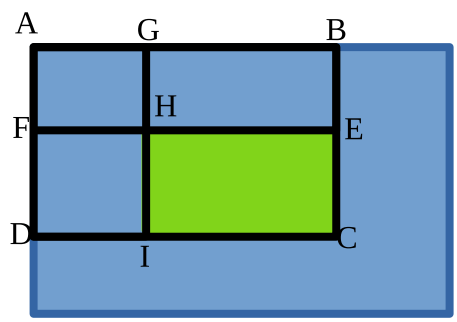
But, I don’t see a way, to compute $\{\{1, 1\}, \{3, 3\}\}$ for example:

$$M = \begin{bmatrix} 3 & 3 & 4 & 8 & 10 \\ 15 & 21 & 24 & 26 & 27 \\ 28 & 30 & 30 & 31 & 36 \\ 40 & 41 & 41 & 42 & 49 \\ 50 & 50 & 53 & 53 & 58 \end{bmatrix}$$

What to do?

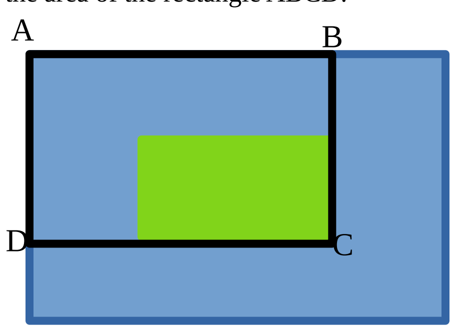
After thinking a while, it’s basic geometry:

$$M = \begin{bmatrix} .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. \end{bmatrix} \text{==>}$$

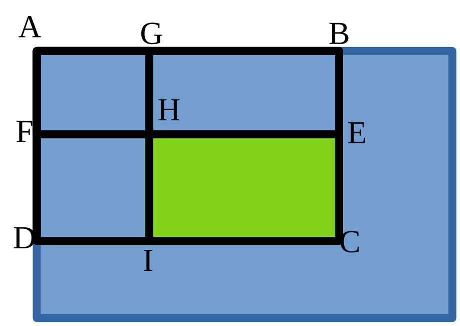


Imagine that we wanna calculate the area of the green rectangle(HECI) inside the blue one.

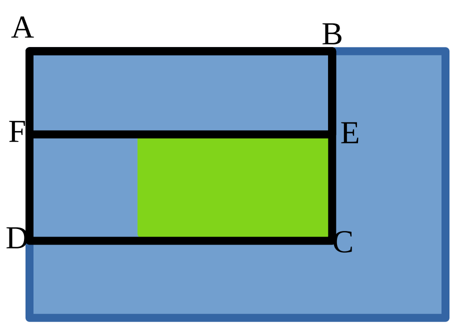
We must know the area of the rectangle ABCD:



FHID too



also, ABEF



So, the area of the green rectangle (HECI) is:

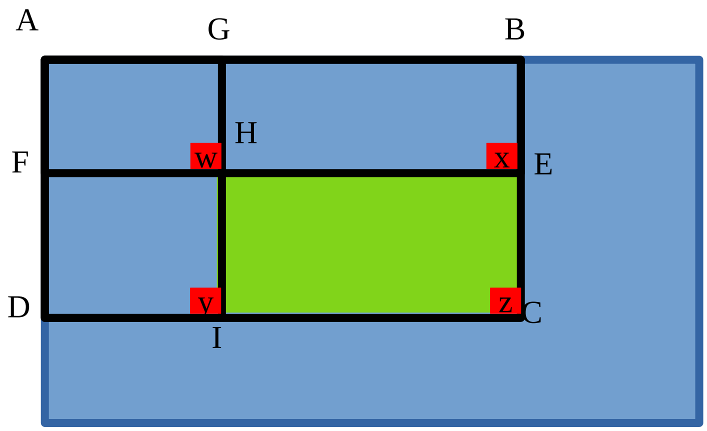
$$area(ABCD) - area(ABEF) - area(FHID)$$

$$area(FHID) = area(AGID) - area(AGHF)$$

so,

$$area(ABCD) - area(ABEF) - area(FHID) - (area(AGID) - area(AGHF))$$

In our cumulative matrix, the sum (area) of each rectangle will be found at the right bottom corner of the rectangle.



So, the area of the green rectangle (HECI) is:

$$z = area(ABCD)$$
$$x = area(ABEF)$$
$$y = area(AGID)$$
$$w = area(AGHF)$$

$$\text{area of green the rectangle} = z - x - (y - w)$$
$$= M'[row2, col2] - M'[row1-1, col2] - (M'[row2, col1-1] - M'[row1-1, col1-1])$$

where M' is the cumulative sum matrix.
In order to avoid coding much Ifs, because we have to check if row1 == 0 and col1 == 0 or not.
We gonna add a row and a column to M' with 0s.

$$M = \begin{bmatrix} .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. \end{bmatrix} \quad ==> \quad M' = \begin{bmatrix} 0 & 0 & 0 & 0 & .. & 0 \\ 0 & .. & .. & .. & .. & .. \\ 0 & .. & .. & .. & .. & .. \\ 0 & .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. & .. \\ 0 & .. & .. & .. & .. & .. \end{bmatrix}$$

the formula become:

$$M'[row2+1][col2+1] - M'[row1][col2+1] - (M'[row2+1][col1] - M'[row1][col1])$$

How to find the area of these rectangles?

From the original compute the sum of each rectangle $\{\{0, 0\}, \{i, j\}\}$ where $0 \leq i \leq n$ and $0 \leq j \leq m$, and put the result in another matrix at $\{\{i+1, j+1\}\}$

let’s go throw an example:

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3				
0					
0					
0					
0					

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3			
0					
0					
0					
0					

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4		
0					
0					
0					
0					

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4	8	
0					
0					
0					
0					

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4	8	10
0					
0					
0					
0					

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4	8	10
0	8				
0					
0					
0					

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4	8	10
0	8				
0	9				
0					
0					

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4	8	10
0	8				
0	9				
0	13				
0					

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4	8	10
0	8				
0	9				
0	13				
0	14				

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4	8	10
0	8	14			
0	9				
0	13				
0	14				

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4	8	10
0	8	14	18		
0	9				
0	13				
0	14				

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4	8	10
0	8	14	18	24	
0	9				
0	13				
0	14				

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4	8	10
0	8	14	18	24	27
0	9				
0	13				
0	14				

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4	8	10
0	8	14	18	24	27
0	9	17			
0	13				
0	14				

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4	8	10
0	8	14	18	24	27
0	9	17	21		
0	13				
0	14				

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4	8	10
0	8	14	18	24	27
0	9	17	21	28	
0	13				
0	14				

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4	8	10
0	8	14	18	24	27
0	9	17	21	28	36
0	13				
0	14				

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4	8	10
0	8	14	18	24	27
0	9	17	21	28	36
0	13	22			
0	14				

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4	8	10
0	8	14	18	24	27
0	9	17	21	28	36
0	13	22	26		
0	14				

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4	8	10
0	8	14	18	24	27
0	9	17	21	28	36
0	13	22	26	34	
0	14				

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4	8	10
0	8	14	18	24	27
0	9	17	21	28	36
0	13	22	26	34	49
0	14				

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4	8	10
0	8	14	18	24	27
0	9	17	21	28	36
0	13	22	26	34	49
0	14	23			

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4	8	10
0	8	14	18	24	27
0	9	17	21	28	36
0	13	22	26	34	49
0	14	23	30		

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4	8	10
0	8	14	18	24	27
0	9	17	21	28	36
0	13	22	26	34	49
0	14	23	30	38	

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

0	0	0	0	0	0
0	3	3	4	8	10
0	8	14	18	24	27
0	9	17	21	28	36
0	13	22	26	34	49
0	14	23	30	38	58

C++ code: O(1)

```
class NumMatrix {
    public:
        vector<vector<int>> m;
public:
    NumMatrix(vector<vector<int>>& matrix) {
        int row_size = matrix.size();
        int col_size = matrix[0].size();

        m.resize(row_size + 1);
        for (int i = 0 ; i < row_size+1; ++i)
            m[i].resize(col_size + 1);

        for (int i = 0 ; i < row_size+1 ; ++i){
            for (int j = 0 ; j < col_size+1; ++j) {
                if (i == 0 || j == 0) m[i][j] = 0;
                else if (i == 0) m[i][j] = matrix[i][j-1] + m[i][j];
                else if (j == 0) m[i][j] = matrix[i-1][j] + m[i][j];
                else m[i][j] = matrix[i-1][j-1] + m[i-1][j] + m[i][j-1] - m[i-1][j-1];
            }
        }
    }

    int sumRegion(int row1, int col1, int row2, int col2) {
        return m[row2 + 1][col2 + 1] - m[row1][col2 + 1] - (m[row2 + 1][col1] - m[row1]
[col1]);
    }
};
```

imagine that it's called k times in the program, the whole time complexity will be $O(k \times 1)$. In general is a $O(n)$