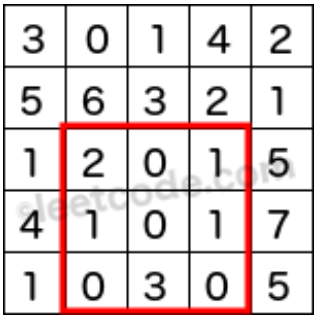
Range Sum Query 2D - Immutable

Given a 2D matrix *matrix*, find the sum of the elements inside the rectangle defined by its upper left corner (*row*1, *col*1) and lower right corner (*row*2, *col*2).



The above rectangle (with the red border) is defined by (row1, col1) = **(2, 1)** and (row2, col2) = **(4, 3)**, which contains sum = **8**.

Example:

```
Given matrix = [
  [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]
]

sumRegion(2, 1, 4, 3) -> 8
sumRegion(1, 1, 2, 2) -> 11
sumRegion(1, 2, 2, 4) -> 12
```

Note:

1. You may assume that the matrix does not change.
2. There are many calls to *sumRegion* function.
3. You may assume that *row*1 ≤ *row*2 and *col*1 ≤ *col*2.

Solution 1

```
private int[][] dp;

public NumMatrix(int[][] matrix) {
    if(   matrix           == null
       || matrix.length    == 0
       || matrix[0].length == 0   ){
        return;
    }

    int m = matrix.length;
    int n = matrix[0].length;

    dp = new int[m + 1][n + 1];
    for(int i = 1; i <= m; i++){
        for(int j = 1; j <= n; j++){
            dp[i][j] = dp[i - 1][j] + dp[i][j - 1] -dp[i - 1][j - 1] + matrix[i -
1][j - 1] ;
        }
    }
}

public int sumRegion(int row1, int col1, int row2, int col2) {
    int iMin = Math.min(row1, row2);
    int iMax = Math.max(row1, row2);

    int jMin = Math.min(col1, col2);
    int jMax = Math.max(col1, col2);

    return dp[iMax + 1][jMax + 1] - dp[iMax + 1][jMin] - dp[iMin][jMax + 1] + dp[
iMin][jMin];
}
```

written by larrywang2014 original link here

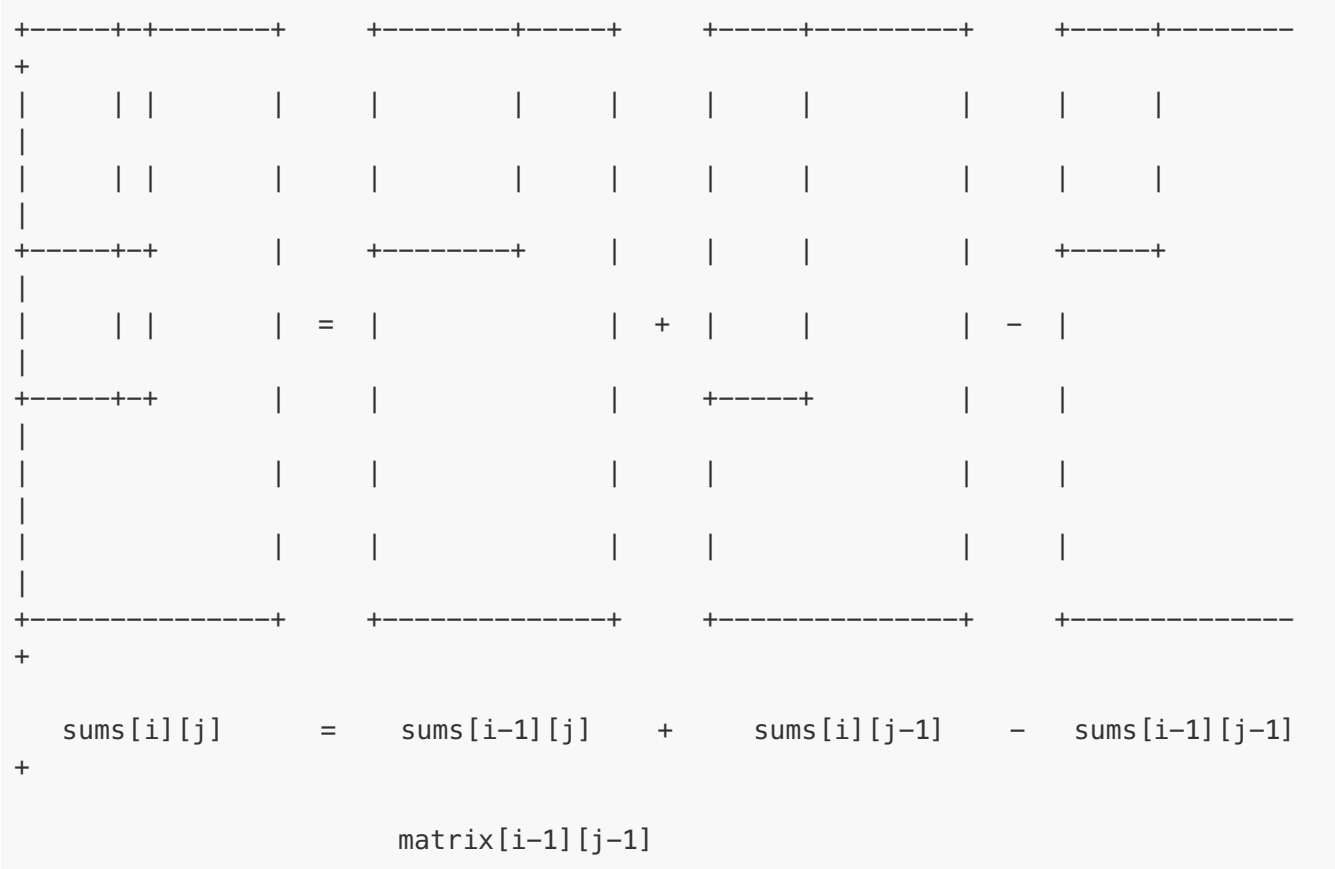
Solution 2

Construct a 2D array `sums[row+1][col+1]`

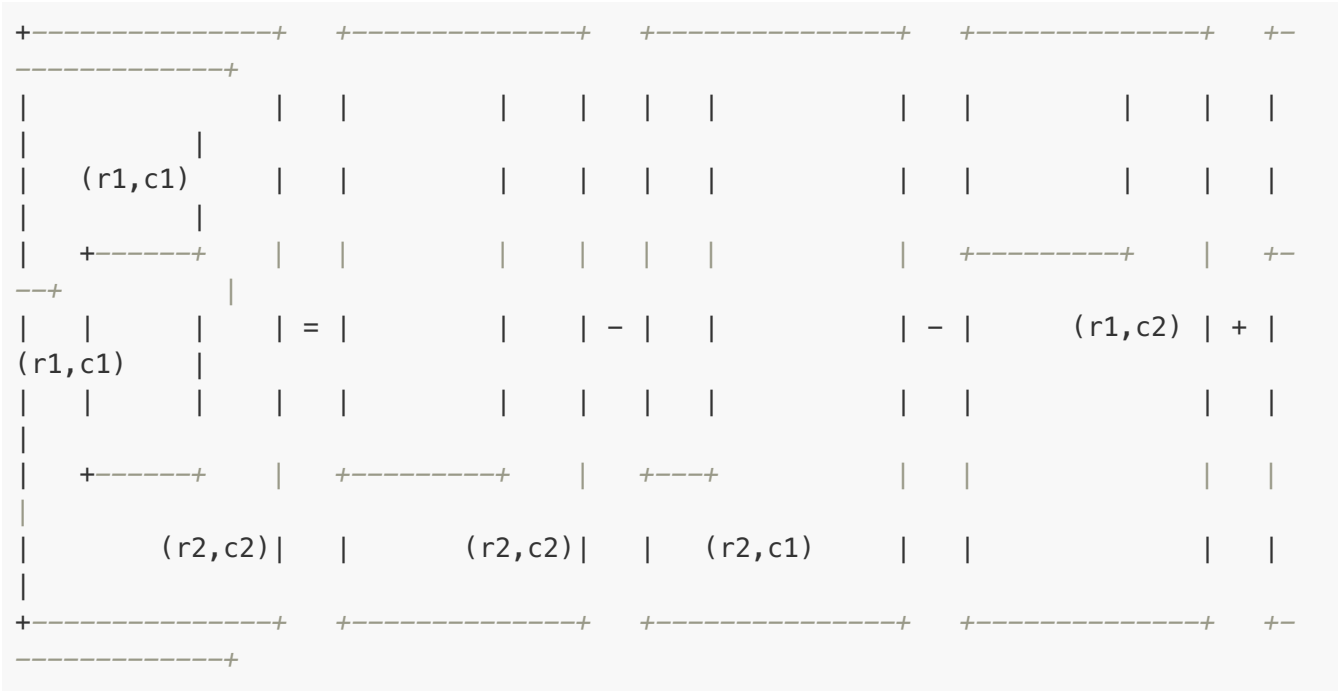
(**notice**: we add additional blank row `sums[0][col+1]={0}` and blank column `sums[row+1][0]={0}` to remove the edge case checking), so, we can have the following definition

`sums[i+1][j+1]` represents the sum of area from `matrix[0][0]` to `matrix[i][j]`

To calculate sums, the ideas as below



So, we use the same idea to find the specific area's sum.



And we can have the following code

```
class NumMatrix {
private:
    int row, col;
    vector<vector<int>> sums;
public:
    NumMatrix(vector<vector<int>> &matrix) {
        row = matrix.size();
        col = row>0 ? matrix[0].size() : 0;
        sums = vector<vector<int>>(row+1, vector<int>(col+1, 0));
        for(int i=1; i<=row; i++) {
            for(int j=1; j<=col; j++) {
                sums[i][j] = matrix[i-1][j-1] +
                             sums[i-1][j] + sums[i][j-1] - sums[i-1][j-1] ;
            }
        }
    }

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

written by haoel original link here

Solution 3

My `accu[i][j]` is the sum of `matrix[0..i][0..j]`, and `a(i, j)` helps with edge cases.

```
class NumMatrix {
public:
    NumMatrix(vector<vector<int>> &matrix) {
        accu = matrix;
        for (int i=0; i<matrix.size(); ++i)
            for (int j=0; j<matrix[0].size(); ++j)
                accu[i][j] += a(i-1, j) + a(i, j-1) - a(i-1, j-1);
    }

    int sumRegion(int row1, int col1, int row2, int col2) {
        return a(row2, col2) - a(row1-1, col2) - a(row2, col1-1) + a(row1-1, col1
-1);
    }

private:
    vector<vector<int>> accu;
    int a(int i, int j) {
        return i >= 0 && j >= 0 ? accu[i][j] : 0;
    }
};
```

Afterthought

Instead of

```
                accu[i][j] += a(i-1, j) + a(i, j-1) - a(i-1, j-1);
```

I could use

```
                accu[i][j] += a(i, j) - sumRegion(i, j, i, j);
```

which is shorter but I think less clear. I do like already using `sumRegion` in the precomputation, though.

written by StefanPochmann original link here