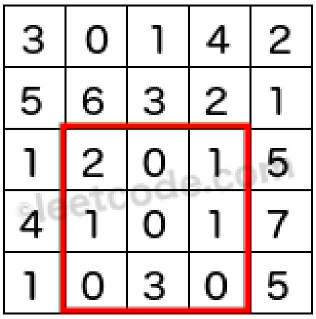
Range Sum Query 2D - Mutable

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).



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

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
update(3, 2, 2)
sumRegion(2, 1, 4, 3) -> 10
```

Note:

1. The matrix is only modifiable by the*update* function.
2. You may assume the number of calls to*update* and *sumRegion* function is distributed evenly.
3. You may assume that *row*1 ≤ *row*2 and *col*1 ≤ *col*2.

Solution 1

```
public class NumMatrix {

    int[][] tree;
    int[][] nums;
    int m;
    int n;

    public NumMatrix(int[][] matrix) {
        if (matrix.length == 0 || matrix[0].length == 0) return;
        m = matrix.length;
        n = matrix[0].length;
        tree = new int[m+1][n+1];
        nums = new int[m][n];
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                update(i, j, matrix[i][j]);
            }
        }
    }

    public void update(int row, int col, int val) {
        if (m == 0 || n == 0) return;
        int delta = val - nums[row][col];
        nums[row][col] = val;
        for (int i = row + 1; i <= m; i += i & (-i)) {
            for (int j = col + 1; j <= n; j += j & (-j)) {
                tree[i][j] += delta;
            }
        }
    }

    public int sumRegion(int row1, int col1, int row2, int col2) {
        if (m == 0 || n == 0) return 0;
        return sum(row2+1, col2+1) + sum(row1, col1) - sum(row1, col2+1) - sum(ro
w2+1, col1);
    }

    public int sum(int row, int col) {
        int sum = 0;
        for (int i = row; i > 0; i -= i & (-i)) {
            for (int j = col; j > 0; j -= j & (-j)) {
                sum += tree[i][j];
            }
        }
        return sum;
    }
}
// time should be O(log(m) * log(n))
```

Explanation of Binary Indexed Tree : https://www.topcoder.com/community/data-science/data-science-tutorials/binary-indexed-trees/

written by novice00 original link here

Solution 2

We use colSums[i][j] = the sum of (matrix[0][j], matrix[1][j], matrix[2][j],......,matrix[i - 1][j]).

```
private int[][] colSums;
private int[][] matrix;

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

     this.matrix = matrix;

     int m   = matrix.length;
     int n   = matrix[0].length;
     colSums = new int[m + 1][n];
     for(int i = 1; i <= m; i++){
         for(int j = 0; j < n; j++){
             colSums[i][j] = colSums[i - 1][j] + matrix[i - 1][j];
         }
     }
}
//time complexity for the worst case scenario: O(m)
public void update(int row, int col, int val) {
    for(int i = row + 1; i < colSums.length; i++){
        colSums[i][col] = colSums[i][col] - matrix[row][col] + val;
    }

    matrix[row][col] = val;
}
//time complexity for the worst case scenario: O(n)
public int sumRegion(int row1, int col1, int row2, int col2) {
    int ret = 0;

    for(int j = col1; j <= col2; j++){
        ret += colSums[row2 + 1][j] - colSums[row1][j];
    }

    return ret;
}
```

written by larrywang2014 original link here

Solution 3

I have written both the Quad tree based solution and the indexed tree based solution for c++.

Both are very straight-forward. I have made some mistake for my previous analysis of the quad-tree solution. The indexed tree solution is more efficient in general.

Method 1: Quad-tree based solution. Essentially, it is a divide and conquer algorithm that divide the whole matrix into 4 sub-matrices recursively. It can be shown that the algorithm is O(max(m, n)) per update/query.

```
class NumMatrix {
    struct TreeNode {
        int val = 0;
        TreeNode* neighbor[4] = {NULL, NULL, NULL, NULL};
        pair<int, int> leftTop = make_pair(0,0);
        pair<int, int> rightBottom = make_pair(0,0);
        TreeNode(int v):val(v){}
    };
public:
    NumMatrix(vector<vector<int>> &matrix) {
        nums = matrix;
        if (matrix.empty()) return;
        int row = matrix.size();
        if (row == 0) return;
        int col= matrix[0].size();
        root = createTree(matrix, make_pair(0,0), make_pair(row-1, col-1));
    }

    void update(int row, int col, int val) {
        int diff = val - nums[row][col];
        if (diff == 0) return;
        nums[row][col] = val;
        updateTree(row, col, diff, root);
    }

    int sumRegion(int row1, int col1, int row2, int col2) {
        int res = 0;
        if (root != NULL)
            sumRegion(row1, col1, row2, col2, root, res);
        return res;
    }

private:
    TreeNode* root = NULL;
    vector<vector<int>> nums;
    TreeNode* createTree(vector<vector<int>> &matrix, pair<int, int> start, pair<
int, int> end) {
        if (start.first > end.first || start.second > end.second)
            return NULL;
        TreeNode* cur = new TreeNode(0);
        cur->leftTop = start;
        cur->rightBottom = end;
        if (start == end) {
            cur->val = matrix[start.first][start.second];
            return cur;
        }

        int midx = ( start.first + end.first ) / 2;
        int midy = (start.second + end.second) / 2;
        cur->neighbor[0] = createTree(matrix, start, make_pair(midx, midy));
        cur->neighbor[1] = createTree(matrix, make_pair(start.first, midy+1), make_pair(midx, end.second));
        cur->neighbor[2] = createTree(matrix, make_pair(midx+1, start.second), make_pair(end.first, midy));
        cur->neighbor[3] = createTree(matrix, make_pair(midx+1, midy+1), end);
        for (int i = 0; i < 4; i++) {
            if (cur->neighbor[i])
                cur->val += cur->neighbor[i]->val;
        }
        return cur;
    }

    void sumRegion(int row1, int col1, int row2, int col2, TreeNode* ptr, int &res) {
        pair<int, int> start = ptr->leftTop;
        pair<int, int> end = ptr->rightBottom;
        // determine whether there is overlapping
        int top = max(start.first, row1);
        int bottom = min(end.first, row2);
        if (bottom < top) return;
        int left = max(start.second, col1);
        int right = min(end.second, col2);
        if (left > right) return;


        if (row1 <= start.first && col1 <= start.second && row2 >= end.first && col2 >= end.second) {
            res += ptr->val;
            return;
        }

        for (int i = 0; i < 4; i ++)
            if (ptr->neighbor[i])
                sumRegion(row1, col1, row2, col2, ptr->neighbor[i], res);

    }


    void updateTree(int row, int col, int diff, TreeNode* ptr){
        if (row >= (ptr->leftTop).first && row <= (ptr->rightBottom).first &&
            col >= (ptr->leftTop).second && col <= (ptr->rightBottom).second)
        {
            ptr->val += diff;
            for (int i = 0; i < 4; i++)
                if (ptr->neighbor[i])
                    updateTree(row, col, diff, ptr->neighbor[i]);

        }
    }
};
```

Method 2: the 2D indexed-tree solution. It is a simple generalization of the 1D indexed tree solution. The complexity should be O(log(m)log(n)).

```
class NumMatrix {
public:
    NumMatrix(vector<vector<int>> &matrix) {
        if (matrix.size() == 0 || matrix[0].size() == 0) return;
        nrow = matrix.size();
        ncol = matrix[0].size();
        nums = matrix;
        BIT = vector<vector<int>> (nrow+1, vector<int>(ncol+1, 0));
        for (int i = 0; i < nrow; i++)
            for (int j = 0; j < ncol; j++)
                add(i, j, matrix[i][j]);

    }

    void update(int row, int col, int val) {
        int diff = val - nums[row][col];
        add(row, col,diff);
        nums[row][col] = val;
    }

    int sumRegion(int row1, int col1, int row2, int col2) {
        int regionL = 0, regionS = 0;
        int regionLeft = 0, regionTop = 0;

        regionL = region(row2, col2);

        if (row1 > 0 && col1 > 0) regionS = region(row1-1, col1-1);

        if (row1 > 0) regionTop  = region(row1-1, col2);

        if (col1 > 0) regionLeft = region(row2, col1-1);

        return regionL - regionTop - regionLeft + regionS;
    }
private:
    vector<vector<int>> nums;
    vector<vector<int>> BIT;
    int nrow = 0;
    int ncol = 0;
    void add(int row, int col, int val) {
        row++;
        col++;
        while(row <= nrow) {
            int colIdx = col;
            while(colIdx <= ncol) {
                BIT[row][colIdx] += val;
                colIdx += (colIdx & (-colIdx));
            }
            row +=  (row & (-row));
        }
    }
    int region(int row, int col) {
        row++;
        col++;
        int res = 0;
        while(row > 0) {
            int colIdx = col;
            while(colIdx > 0) {
                res += BIT[row][colIdx];
                colIdx -= (colIdx & (-colIdx));
            }
            row -= (row & (-row));
        }
        return res;
    }
};
```

written by whnzinc original link here

From Leetcoder.