Description | Solution | Discuss (279) | Submissions

308. Range Sum Query 2D - Mutable

Hard 603 71 Add to List Share

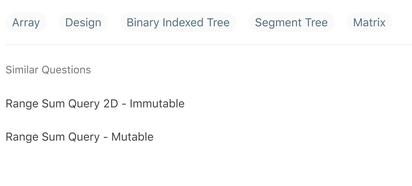
Given a 2D matrix `matrix`, handle multiple queries of the following types:

1. **Update** the value of a cell in `matrix`.
2. Calculate the **sum** of the elements of `matrix` inside the rectangle defined by its **upper left corner** `(row1, col1)` and **lower right corner** `(row2, col2)`.

Implement the NumMatrix class:

- `NumMatrix(int[][] matrix)` Initializes the object with the integer matrix `matrix`.
- `void update(int row, int col, int val)` **Updates** the value of `matrix[row][col]` to be `val`.
- `int sumRegion(int row1, int col1, int row2, int col2)` Returns the **sum** of the elements of `matrix` inside the rectangle defined by its **upper left corner** `(row1, col1)` and **lower right corner** `(row2, col2)`.

Example 1:




```
Input
["NumMatrix", "sumRegion", "update", "sumRegion"]
[[[[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]]], [2, 1, 4, 3], [3, 2, 2], [2, 1, 4, 3]]
Output
[null, 8, null, 10]

Explanation
NumMatrix numMatrix = new NumMatrix([[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]]);
numMatrix.sumRegion(2, 1, 4, 3); // return 8 (i.e. sum of the left red rectangle)
numMatrix.update(3, 2, 2);       // matrix changes from left image to right image
numMatrix.sumRegion(2, 1, 4, 3); // return 10 (i.e. sum of the right red rectangle)
```

Constraints:

- `m == matrix.length`
- `n == matrix[i].length`
- `1 <= m, n <= 200`
- `-10^5 <= matrix[i][j] <= 10^5`
- `0 <= row < m`
- `0 <= col < n`
- `-10^5 <= val <= 10^5`
- `0 <= row1 <= row2 < m`
- `0 <= col1 <= col2 < n`
- At most `10^4` calls will be made to `sumRegion` and `update`.

Accepted 61,791 | Submissions 154,298

Seen this question in a real interview before? Yes No

Companies

0 ~ 6 months | 6 months ~ 1 year | 1 year ~ 2 years

Google | 3

Related Topics

Array Design Binary Indexed Tree Segment Tree Matrix

Similar Questions

Range Sum Query 2D - Immutable Medium

Range Sum Query - Mutable Medium

Java | Autocomplete

```java
class NumMatrix {
    private int rows;
    private int cols;
    private int[][] bit; // The BIT matrix

    private int lsb(int n) {
        // the line below allows us to directly capture the right most non-zero bit of a number
        return n & (-n);
    }

    private void updateBIT(int r, int c, int val) {
        // keep adding lsb(i) to i, lsb(j) to j and add val to bit[i][j]
        // Using two nested for loops, one for the rows and one for the columns
        for (int i = r; i <= rows; i += lsb(i)) {
            for (int j = c; j <= cols; j += lsb(j)) {
                this.bit[i][j] += val;
            }
        }
    }

    private int queryBIT(int r, int c) {
        int sum = 0;
        // keep subtracting lsb(i) to i, lsb(j) to j and obtain the final sum as the sum of non-overlapping sub-rectangles
        // Using two nested for loops, one for the rows and one for the columns
        for (int i = r; i > 0; i -= lsb(i)) {
            for (int j = c; j > 0; j -= lsb(j)) {
                sum += this.bit[i][j];
            }
        }
        return sum;
    }

    private void buildBIT(int[][] matrix) {
        for (int i = 1; i <= rows; ++i) {
            for (int j = 1; j <= cols; ++j) {
                // call update function on each of the entries present in the matrix
                int val = matrix[i - 1][j - 1];
                updateBIT(i, j, val);
            }
        }
    }

    public NumMatrix(int[][] matrix) {
        rows = matrix.length;
        if (rows == 0) return;
        cols = matrix[0].length;
        bit = new int[rows + 1][];
        // Using 1 based indexing, hence resizing the bit array to (rows + 1, cols + 1)
        for (int i = 1; i <= rows; ++i)
            bit[i] = new int[cols + 1];
        buildBIT(matrix);
    }

    public void update(int row, int col, int val) {
        int old_val = sumRegion(row, col, row, col);
        // handling 1-based indexing
        row++; col++;
        int diff = val - old_val;
        updateBIT(row, col, diff);
    }

    public int sumRegion(int row1, int col1, int row2, int col2) {
        // handling 1-based indexing
        row1++; col1++; row2++; col2++;
        int a = queryBIT(row2, col2);
        int b = queryBIT(row1 - 1, col1 - 1);
        int c = queryBIT(row2, col1 - 1);
        int d = queryBIT(row1 - 1, col2);
        return (a + b) - (c + d);
    }
};
```