# Sub Matrix Sum

The sub matrix sum is based on a two-dimension array. The idea is similar to one-dimension subarray prefix sum, you first calculate the prefix sum in each row, then you pick two columns and add the subarray sum for each row within these two columns, and get another level prefix sum, so actually you are calculating any sub matrix between any two rows within these two columns

## 363. Max Sum of Rectangle No Larger Than K

Hard

Given a non-empty 2D matrix *matrix* and an integer *k*, find the max sum of a rectangle in the *matrix* such that its sum is no larger than *k*.

**Example:**

**Input:** matrix = [[1,0,1],[0,-2,3]], k = 2

**Output:** 2

**Explanation:** Because the sum of rectangle [[0, 1], [-2, 3]] is 2,

  and 2 is the max number no larger than k (k = 2).

**Note:**

1. The rectangle inside the matrix must have an area > 0.
2. What if the number of rows is much larger than the number of columns?

### Analysis:

You calculate the prefix sum in each row, and the pick two columns c1 and c2 calculate the sum between any two rows with another prefix sum. In this way you get all submatrix sum, The time complexity is O(C\*C\*R). Get the maximum sum which is less than k.

/// <summary>

/// Leet code # 363. Max Sum of Rectangle No Larger Than K

///

/// Given a non-empty 2D matrix matrix and an integer k, find the max sum of a

/// rectangle in the matrix such that its sum is no larger than k.

///

/// Example:

/// Given matrix =

/// [

/// [1, 0, 1],

/// [0, -2, 3]

/// ]

/// k = 2

/// The answer is 2. Because the sum of rectangle [[0, 1], [-2, 3]] is 2 and 2 is the max number no larger than k (k = 2).

/// Note:

/// 1.The rectangle inside the matrix must have an area > 0.

/// 2.What if the number of rows is much larger than the number of columns?

/// </summary>

int LeetCodeArray::maxSumSubmatrix(vector<vector<int>>& matrix, int k)

{

int max\_value = INT\_MIN;

if (matrix.empty() || matrix[0].empty()) return 0;

vector<vector<int>>sum(matrix.size(), vector<int>(matrix[0].size()));

for (size\_t col1 = 0; col1 < matrix[0].size(); col1++)

{

vector<int> sum(matrix.size());

for (size\_t col2 = col1; col2 < matrix[0].size(); col2++)

{

set<int> accumulate\_set;

accumulate\_set.insert(0);

int accumulate = 0;

for (size\_t row = 0; row < matrix.size(); row++)

{

sum[row] += matrix[row][col2];

accumulate += sum[row];

set<int>::iterator itr = accumulate\_set.lower\_bound(accumulate - k);

if (itr != accumulate\_set.end())

{

max\_value = max(max\_value, accumulate - \*itr);

if (max\_value == k) return max\_value;

}

accumulate\_set.insert(accumulate);

}

}

}

return max\_value;

}

## 1074. Number of Submatrices That Sum to Target

Hard

Given a matrix, and a target, return the number of non-empty submatrices that sum to target.

A submatrix x1, y1, x2, y2 is the set of all cells matrix[x][y] with x1 <= x <= x2 and y1 <= y <= y2.

Two submatrices (x1, y1, x2, y2) and (x1', y1', x2', y2') are different if they have some coordinate that is different: for example, if x1 != x1'.

**Example 1:**

**Input:** matrix = [[0,1,0],[1,1,1],[0,1,0]], target = 0

**Output:** 4

**Explanation:** The four 1x1 submatrices that only contain 0.

**Example 2:**

**Input:** matrix = [[1,-1],[-1,1]], target = 0

**Output:** 5

**Explanation:** The two 1x2 submatrices, plus the two 2x1 submatrices, plus the 2x2 submatrix.

**Note:**

1. 1 <= matrix.length <= 300
2. 1 <= matrix[0].length <= 300
3. -1000 <= matrix[i] <= 1000
4. -10^8 <= target <= 10^8

### Analysis:

You calculate the prefix sum in each row, and the pick two columns c1 and c2 calculate the sum between any two rows with another prefix sum. The time complexity is O(C\*C\*R).

/// <summary>

/// Leet code #1074. Number of Submatrices That Sum to Target

///

/// Given a matrix, and a target, return the number of non-empty submatrices

/// that sum to target.

///

/// A submatrix x1, y1, x2, y2 is the set of all cells matrix[x][y] with

/// x1 <= x <= x2 and y1 <= y <= y2.

///

/// Two submatrices (x1, y1, x2, y2) and (x1', y1', x2', y2') are different

/// if they have some coordinate that is different: for example, if x1 != x1'.

///

/// Example 1:

///

/// Input: matrix = [[0,1,0],[1,1,1],[0,1,0]], target = 0

/// Output: 4

/// Explanation: The four 1x1 submatrices that only contain 0.

///

/// Example 2:

///

/// Input: matrix = [[1,-1],[-1,1]], target = 0

/// Output: 5

/// Explanation: The two 1x2 submatrices, plus the two 2x1 submatrices, plus

/// the 2x2 submatrix.

///

///

/// Note:

///

/// 1. 1 <= matrix.length <= 300

/// 2. 1 <= matrix[0].length <= 300

/// 3. -1000 <= matrix[i] <= 1000

/// 4. -10^8 <= target <= 10^8

/// </summary>

int LeetCodeArray::numSubmatrixSumTarget(vector<vector<int>>& matrix, int target)

{

int result = 0;

vector<vector<int>> sum(matrix.size(), vector<int>(matrix[0].size()));

for (size\_t i = 0; i < matrix.size(); i++)

{

for (size\_t j = 0; j < matrix[0].size(); j++)

{

if (j == 0) sum[i][j] = matrix[i][j];

else sum[i][j] = sum[i][j - 1] + matrix[i][j];

}

}

for (size\_t c1 = 0; c1 < matrix[0].size(); c1++)

{

for (size\_t c2 = c1; c2 < matrix[0].size(); c2++)

{

unordered\_map<int, int> num\_map;

int sum\_num = 0;

num\_map[sum\_num]++;

for (size\_t r = 0; r < matrix.size(); r++)

{

int num = 0;

if (c1 == 0) num = sum[r][c2];

else num = sum[r][c2] - sum[r][c1 - 1];

sum\_num += num;

if (num\_map.count(sum\_num - target) > 0)

{

result += num\_map[sum\_num - target];

}

num\_map[sum\_num]++;

}

}

}

return result;

}

## 750. Number Of Corner Rectangles

Medium

Given a grid where each entry is only 0 or 1, find the number of corner rectangles.

A *corner rectangle* is 4 distinct 1s on the grid that form an axis-aligned rectangle. Note that only the corners need to have the value 1. Also, all four 1s used must be distinct.

**Example 1:**

**Input:** grid =

[[1, 0, 0, 1, 0],

[0, 0, 1, 0, 1],

[0, 0, 0, 1, 0],

[1, 0, 1, 0, 1]]

**Output:** 1

**Explanation:** There is only one corner rectangle, with corners grid[1][2], grid[1][4], grid[3][2], grid[3][4].

**Example 2:**

**Input:** grid =

[[1, 1, 1],

[1, 1, 1],

[1, 1, 1]]

**Output:** 9

**Explanation:** There are four 2x2 rectangles, four 2x3 and 3x2 rectangles, and one 3x3 rectangle.

**Example 3:**

**Input:** grid =

[[1, 1, 1, 1]]

**Output:** 0

**Explanation:** Rectangles must have four distinct corners.

**Note:**

1. The number of rows and columns of grid will each be in the range [1, 200].
2. Each grid[i][j] will be either 0 or 1.
3. The number of 1s in the grid will be at most 6000.

### Analysis:

It is easy to make it as O(R \* R \* C \* C), by select two rows and iterate 2 columns. The better way is to make it either O(R\*R\*C) or O(R\*C\*C). We can iterate two columns, and check every row to see how many rows contain both ‘1’ in these two columns, another way is select two rows and see how many common ‘1’s in these two rows.

/// <summary>

/// Leet code #750. Number Of Corner Rectangles

///

/// Given a grid where each entry is only 0 or 1, find the number of corner

/// rectangles.

/// A corner rectangle is 4 distinct 1s on the grid that form an

/// axis-aligned rectangle. Note that only the corners need to have the

/// value 1. Also, all four 1s used must be distinct.

///

/// Example 1:

/// Input: grid =

/// [[1, 0, 0, 1, 0],

/// [0, 0, 1, 0, 1],

/// [0, 0, 0, 1, 0],

/// [1, 0, 1, 0, 1]]

/// Output: 1

/// Explanation: There is only one corner rectangle, with corners

/// grid[1][2], grid[1][4], grid[3][2], grid[3][4].

/// Example 2:

/// Input: grid =

/// [[1, 1, 1],

/// [1, 1, 1],

/// [1, 1, 1]]

/// Output: 9

/// Explanation: There are four 2x2 rectangles, four 2x3 and 3x2

/// rectangles, and one 3x3 rectangle.

/// Example 3:

/// Input: grid =

/// [[1, 1, 1, 1]]

/// Output: 0

/// Explanation: Rectangles must have four distinct corners.

/// Note:

/// 1. The number of rows and columns of grid will each be in the range

/// [1, 200].

/// 2. Each grid[i][j] will be either 0 or 1.

/// 3. The number of 1s in the grid will be at most 6000.

/// </summary>

int LeetCodeArray::countCornerRectangles(vector<vector<int>>& grid)

{

int result = 0;

for (size\_t i = 0; i < grid[0].size(); i++)

{

for (size\_t j = i + 1; j < grid[0].size(); j++)

{

int count = 0;

for (size\_t k = 0; k < grid.size(); k++)

{

if (grid[k][i] == 1 && grid[k][j] == 1)

{

count++;

}

}

result += count \* (count - 1) / 2;

}

}

return result;

}