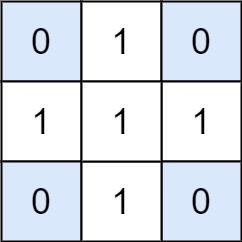
<https://leetcode.com/problems/number-of-submatrices-that-sum-to-target/description/>

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.

**Example 3**:

**Input**: matrix = [[904]], target = 0

**Output**: 0

**Constraints**:

1 <= matrix.length <= 100

1 <= matrix[0].length <= 100

-1000 <= matrix[i][j] <= 1000

-10^8 <= target <= 10^8

**Attempt 1: 2024-05-05**

**Solution 1: Pre Sum + Hash Table (30 min)**

class Solution {

    public int numSubmatrixSumTarget(int[][] matrix, int target) {

        // Create pre-sum matrix in 2D array row by row, for each row

        // the logic exactly same as 560. Subarray Sum Equals K

        int m = matrix.length;

        int n = matrix[0].length;

        for(int i = 0; i < m; i++) {

            for(int j = 1; j < n; j++) {

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

            }

        }

        int result = 0;

        Map<Integer, Integer> counter = new HashMap<>();

        // Outside two for loops with i, j to control columns range as sliding window

        // i range from 0 to n - 1, j range from i to n - 1, both for columns

        for(int i = 0; i < n; i++) {

            for(int j = i; j < n; j++) {

// Refresh counter for each pair of [i, j] columns range scenario

// The inner loop will calculate based on one pair of [i, j] columns

// range, between column i and j, use HashMap to scan from first to

// last row for presum added up strategy, which finds count of all

// potential rows combinations(range for each row limited between

// column i and j) sum up equals to target

                counter.clear();

// To handle cases where prefixSum itself equals target

                counter.put(0, 1);

                int cur = 0;

                // Inside for loop with k to go through all rows to sum up as 1D array,

                // exactly same idea as 560. Subarray Sum Equals K

                for(int k = 0; k < m; k++) {

                    // For first column element not able to substract previous column element

                    if(i == 0) {

                        cur += matrix[k][j];

                    } else {

                        cur += matrix[k][j] - matrix[k][i - 1];

                    }

                    result += counter.getOrDefault(cur - target, 0);

                    counter.put(cur, counter.getOrDefault(cur, 0) + 1);

                }

            }

        }

        return result;

    }

}

Time Complexity: O(M\*N\*N)

Space Complexity: O(M)

**Refer to**

<https://leetcode.com/problems/number-of-submatrices-that-sum-to-target/solutions/303750/java-c-python-find-the-subarray-with-target-sum/>

**Intuition**

Preaquis: [560. Subarray Sum Equals K](https://leetcode.com/problems/subarray-sum-equals-k/description/)

Find the Subarray with Target Sum in linear time.

**Explanation**

For each row, calculate the prefix sum.

For each pair of columns, calculate the accumulated sum of rows.

Now this problem is same to, "Find the Subarray with Target Sum".

**Complexity**

Time O(mnn)

Space O(m)

**Java**

public int numSubmatrixSumTarget(int[][] A, int target) {

int res = 0, m = A.length, n = A[0].length;

for (int i = 0; i < m; i++)

for (int j = 1; j < n; j++)

A[i][j] += A[i][j - 1];

Map<Integer, Integer> counter = new HashMap<>();

for (int i = 0; i < n; i++) {

for (int j = i; j < n; j++) {

counter.clear();

counter.put(0, 1);

int cur = 0;

for (int k = 0; k < m; k++) {

cur += A[k][j] - (i > 0 ? A[k][i - 1] : 0);

res += counter.getOrDefault(cur - target, 0);

counter.put(cur, counter.getOrDefault(cur, 0) + 1);

}

}

}

return res;

}

For those who do not understand the logic behind this code at first glance as I, here is my explanation to help you understand:

**1. calculate prefix sum for each row**

for (int i = 0; i < m; i++)

for (int j = 1; j < n; j++)

A[i][j] += A[i][j - 1];

For this double-for loop, we are calculating the prefix sum for each row in the matrix, which will be used later

**2. for every possible range between two columns, accumulate the prefix sum of submatrices that can be formed between these two columns by adding up the sum of values between these two columns for every row**

for (int i = 0; i < n; i++) {

for (int j = i; j < n; j++) {

Map<Integer, Integer> counter = new HashMap<>();

counter.put(0, 1);

int cur = 0;

for (int k = 0; k < m; k++) {

cur += A[k][j] - (i > 0 ? A[k][i - 1] : 0);

res += counter.getOrDefault(cur - target, 0);

counter.put(cur, counter.getOrDefault(cur, 0) + 1);

}

}

}

To understand what this triple-for loop does, let us try an example, assume i = 1 and j = 3, then for this part of code:

Map<Integer, Integer> counter = new HashMap<>();

counter.put(0, 1);

int cur = 0;

for (int k = 0; k < m; k++) {

cur += A[k][j] - (i > 0 ? A[k][i - 1] : 0);

res += counter.getOrDefault(cur - target, 0);

counter.put(cur, counter.getOrDefault(cur, 0) + 1);

}

I will break this piece of code into two major part:

Map<Integer, Integer> counter = new HashMap<>();

counter.put(0, 1);

key of this hashmap present the unique value of all possible prefix sum that we've seen so far

value of this hashmap represents the count (number of appearances) of each prefix sum value we've seen so far

an empty submatrix trivially has a sum of 0

for (int k = 0; k < m; k++) {

cur += A[k][j] - (i > 0 ? A[k][i - 1] : 0);

res += counter.getOrDefault(cur - target, 0);

counter.put(cur, counter.getOrDefault(cur, 0) + 1);

}

Here we are actually calculating the prefix sum of submatrices which has column 1, 2, and 3, by adding up the sum of matrix[0][1...3], matrix[1][1...3] ... matrix[m-1][1...3] row by row, starting from the first row (row 0). The way of getting the number of submatrices whose sum equals to K uses the same idea of 560. Subarray Sum Equals K so I won't repeat it again.

**Refer to**

[L560.Subarray Sum Equals K](note://5CE29DE019904808986311AFAC74C85C)