

# 1277. Count Square Submatrices with All Ones

[Leetcode Link](#)

Given a  $m * n$  matrix of ones and zeros, return how many square submatrices have all ones.

**Example 1:**

**Input:**

```
1 matrix =
2 [
3   [0,1,1,1],
4   [1,1,1,1],
5   [0,1,1,1]
6 ]
```

**Output:** 15

**Explanation:**

There are 10 squares of side 1.

There are 4 squares of side 2.

There is 1 square of side 3.

Total number of squares =  $10 + 4 + 1 = 15$ .

**Example 2:**

**Input:**

```
1 matrix =
2 [
3   [1,0,1],
4   [1,1,0],
5   [1,1,0]
6 ]
```

**Output:** 7

**Explanation:**

There are 6 squares of side 1.

There is 1 square of side 2.

Total number of squares =  $6 + 1 = 7$ .

**Constraints:**

- $1 \leq arr.length \leq 300$
- $1 \leq arr[0].length \leq 300$
- $0 \leq arr[i][j] \leq 1$

## Solution

### Brute Force

First, let's say that a square is located at a cell  $(i,j)$  if its bottom right corner is located at that cell.

We can observe that if there is a length  $x$  square with all ones located at  $(i,j)$ , then there exists squares with all ones that have lengths  $x - 1, x - 2 \dots, 3, 2, 1$ . This is because if a square submatrix with length  $x$  located there exists, then square submatrices with lengths  $x - 1, x - 2 \dots 3, 2, 1$  must exist as well.

For each cell  $(i,j)$  we'll find the length of the largest square submatrix that has all ones with its bottom right corner located at  $(i,j)$ . Let's denote this value as  $x_{i,j}$ . We can observe that the number of square submatrices with all ones and its bottom right corner located at  $(i,j)$  is simply  $x_{i,j}$ . To find  $x_{i,j}$ , we can brute force through all possible lengths and check if a square submatrix with that respective length exists. Once we find  $x_{i,j}$  for each cell  $(i,j)$ , our final answer is the sum of all  $x_{i,j}$ .

### Full Solution

Our full solution will involve [dynamic programming](#).

Let  $dp[i][j]$  represent  $x_{i,j}$ .

Since [dynamic programming](#) uses the answers to sub-problems to calculate answers to a larger problem, let's try to see how we can use other values from  $dp$  to calculate  $dp[i][j]$  for some cell  $(i,j)$ .

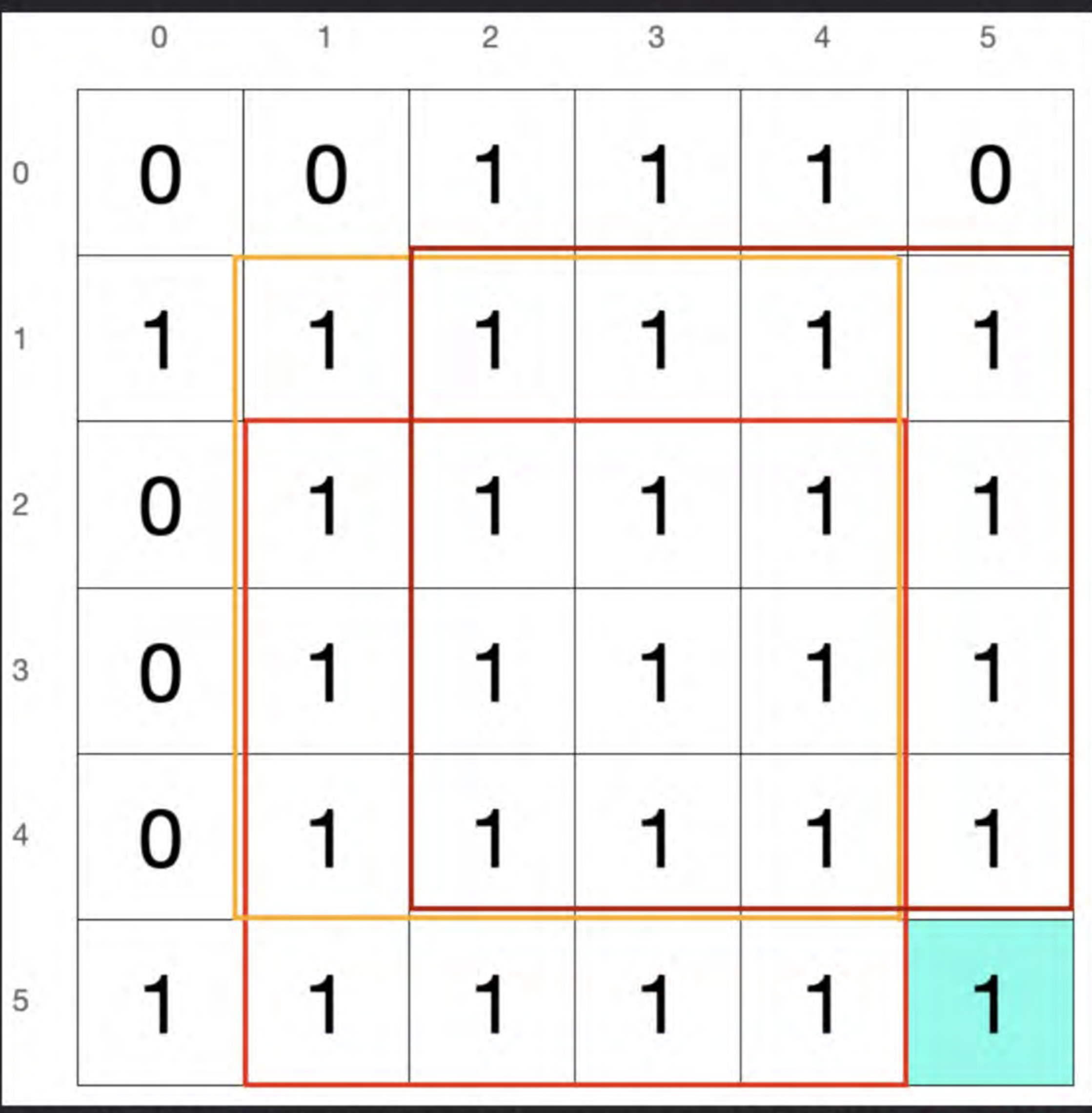
Let's say that a square submatrix is located at a cell  $(i,j)$  if its bottom right corner is located there.

First, let's assume  $dp[i][j] = k$  for some positive integer  $k$ . This means that the largest square submatrix with all ones located at  $(i,j)$  has length  $k$ . We can observe that this means there exists square submatrices with all ones at  $(i,j-1)$ ,  $(i-1,j)$ , and  $(i-1,j-1)$  that have length  $k - 1$ . In addition, the cell  $(i,j)$  is also 1.

If we look at all the cells that are covered by the square submatrices at  $(i,j-1)$ ,  $(i-1,j)$ , and  $(i-1,j-1)$  with length  $k - 1$  and the cell at  $(i,j)$ , we obtain a square with length  $k$  located at  $(i,j)$ .

### Example

Here is a diagram with  $k = 5$  and the cell at  $(5,5)$  to help visualize this observation.



From the observation mentioned above, we can deduce that  $dp[i-1][j], dp[i][j-1], dp[i-1][j-1] \geq k - 1$ .

For  $dp[i][j]$  to be greater or equal to  $k$ , then  $(i,j)$  must be 1 and  $dp[i-1][j], dp[i][j-1], dp[i-1][j-1]$  all have to be greater or equal to  $k - 1$ .

This translates into  $dp[i][j] = \min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1]) + 1$  if the cell  $(i,j)$  is a 1. For most cases, this is how we can calculate  $dp[i][j]$ .

If the cell is in the first row or column however (i.e.  $i = 0$  or  $j = 0$ ), then  $dp[i][j]$  is the same value as the cell  $(i,j)$ .

Obviously, if the cell  $(i,j)$  is 0, then  $dp[i][j] = 0$ .

Our final answer is just the sum of all values stored in  $dp$ .

### Time Complexity

We can calculate the value of a cell in  $dp$  in  $\mathcal{O}(1)$  and since there are  $\mathcal{O}(MN)$  cells, our time complexity is  $\mathcal{O}(MN)$ .

**Time Complexity:**  $\mathcal{O}(MN)$

### Space Complexity

We store  $\mathcal{O}(MN)$  cells in  $dp$  so our space complexity is  $\mathcal{O}(MN)$ .

**Space Complexity:**  $\mathcal{O}(MN)$

**Bonus:** We can use the space optimization mentioned in [this article](#) to optimize memory to  $\mathcal{O}(N)$ . Since we only use rows  $i - 1$  and  $i$  in  $dp$  to calculate  $dp$  values for row  $i$ , we only need to maintain two rows of memory for  $dp$ , which is  $\mathcal{O}(N)$ .

## C++ Solution

```
1 class Solution {
2 public:
3     int countSquares(vector<vector<int>>& matrix) {
4         int m = matrix.size();
5         int n = matrix[0].size(); // dimensions for matrix
6         int ans = 0;
7         vector<vector<int>> dp(m, vector<int>(n));
8         for (int i = 0; i < m; i++) {
9             for (int j = 0; j < n; j++) {
10                 if (i == 0 || j == 0) { // cell is in first row or column
11                     dp[i][j] = matrix[i][j];
12                 } else if (matrix[i][j] == 1) {
13                     dp[i][j] = min({dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1]}) + 1;
14                 }
15                 ans += dp[i][j];
16             }
17         }
18         return ans;
19     }
20 };
```

## Java Solution

```
1 class Solution {
2     public int countSquares(int[][] matrix) {
3         int m = matrix.length;
4         int n = matrix[0].length; // dimensions for matrix
5         int[][] dp = new int[m][n];
6         int ans = 0;
7         for (int i = 0; i < m; i++) {
8             for (int j = 0; j < n; j++) {
9                 if (i == 0 || j == 0) { // cell is in first row or column
10                     dp[i][j] = matrix[i][j];
11                 } else if (matrix[i][j] == 1) {
12                     dp[i][j] = Math.min(dp[i - 1][j], Math.min(dp[i][j - 1], dp[i - 1][j - 1])) + 1;
13                 }
14                 ans += dp[i][j];
15             }
16         }
17         return ans;
18     }
19 }
```

## Python Solution

```
1 class Solution:
2     def countSquares(self, matrix: List[List[int]]) -> int:
3         m = len(matrix)
4         n = len(matrix[0]) # dimensions for matrix
5         dp = [[0] * n for a in range(m)]
6         ans = 0
7         for i in range(m):
8             for j in range(n):
9                 if i == 0 or j == 0: # cell is in first row or column
10                     dp[i][j] = matrix[i][j]
11                 elif matrix[i][j] == 1:
12                     dp[i][j] = min(dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1]) + 1
13                 ans += dp[i][j]
14         return ans
```

Got a question? [Ask the Teaching Assistant](#) anything you don't understand.