750. Number of Corner Rectangles

Dec. 16, 2017 | 17.3K views
Average Rating: 4.59 (17 votes)

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],
```

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
```

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 `1`s in the grid will be at most `6000`.

Approach #1: Count Corners [Accepted]

Intuition

We ask the question: for each additional row, how many more rectangles are added?

For each pair of 1s in the new row (say at `new_row[i]` and `new_row[j]`), we could create more rectangles where that pair forms the base. The number of new rectangles is the number of times some previous row had `row[i] = row[j] = 1`.

Algorithm

Let's maintain a count `count[i, j]`, the number of times we saw `row[i] = row[j] = 1`. When we process a new row, for every pair `new_row[i] = new_row[j] = 1`, we add `count[i, j]` to the answer, then we increment `count[i, j]`.

Java | Python — Copy

```
class Solution(object):
    def countCornerRectangles(self, grid):
        count = collections.Counter()
        ans = 0
        for row in grid:
            for c1, v1 in enumerate(row):
                if v1:
                    for c2 in xrange(c1+1, len(row)):
                        if row[c2]:
                            ans += count[c1, c2]
                            count[c1, c2] += 1
        return ans
```

Complexity Analysis

- Time Complexity: $O(R * C^2)$ where R, C is the number of rows and columns.
- Space Complexity: $O(C^2)$ in additional space.

Approach #2: Heavy and Light Rows [Accepted]

Intuition and Algorithm

Can we improve on the ideas in *Approach #1*? When a row is filled with X 1s, we do $O(X^2)$ work to enumerate every pair of 1s. This is okay when X is small, but expensive when X is big.

Say the entire top row is filled with 1s. When looking at the next row with say, `f` 1s that match the top row, the number of rectangles created is just the number of pairs of 1s, which is `f * (f-1) / 2`. We could find each `f` quickly using a Set and a simple linear scan of each row.

Let's call a row to be *heavy* if it has more than $\sqrt{N}$ points. The above algorithm changes the complexity of counting a heavy row from $O(C^2)$ to $O(N)$, and there are at most $\sqrt{N}$ heavy rows.

Java | Python — Copy

```
class Solution(object):
    def countCornerRectangles(self, grid):
        rows = [{c for c, val in enumerate(row) if val]
                for row in grid]
        N = sum(len(row) for row in grid)
        SQRTN = int(N**.5)

        ans = 0
        count = collections.Counter()
        for r, row in enumerate(rows):
            if len(row) >= SQRTN:
                target = set(row)
                for r2, row2 in enumerate(rows):
                    if r2 >= r and len(row2) >= SQRTN:
                        continue
                    found = sum(1 for c2 in row2 if c2 in target)
                    ans += found * (found - 1) / 2
            else:
                for pair in itertools.combinations(row, 2):
                    ans += count[pair]
                    count[pair] += 1

        return ans
```

Complexity Analysis

- Time Complexity: $O(N\sqrt{N} + R * C)$ where N is the number of ones in the grid.
- Space Complexity: $O(N + R + C^2)$ in additional space, for `rows`, `target`, and `count`.

Analysis written by: @awice.

Rate this article:

Previous Next

Comments: 15

Sort By

Type comment here... (Markdown is supported)

Preview Post

buynowlitang ★ 52 February 3, 2019 4:52 AM
May I ask why this problem is marked as DP, it's more like observation and then do math...
20 Share Reply
SHOW 4 REPLIES

Han_V ★ 169 March 26, 2018 3:37 PM
It seems to me that the second approach degrades in case of a matrix with a lot of 1s. Is that the case?
6 Share Reply
SHOW 1 REPLY

actionlee0405 ★ 93 March 16, 2018 3:45 AM
For solution 2, when a heavy row arrives, it uses O(N) time to count as it goes over all the 1's in all rows(i.e. count the rectangles between this heavy row with all the other rows). Consider a light row arrives, it uses O(N) time to enumerate all pairs of 1's in its row. Let the number of heavy row be h, then the time for heavy row with all the other rows is O(hN) for light row, it is O(N/R - h), where R is the number of rows. Therefore the total counting time is O(hN). Plus the preprocessing step, it should be
Read More
4 Share Reply

kstan ★ 9 June 18, 2019 9:11 AM
What's the point of the 200 in:
`int pos = c1 * 200 + c2;`
I can't think of any reason why you would want to multiply 200 to pos. I get 200 is the upper limit but what does that have anything to do with multiplying it to c1?
Read More
0 Share Reply
SHOW 3 REPLIES

Patchouli ★ 10 June 18, 2019 1:02 AM
There is no need for the HashSet in the second solution. It's not like you have deleted the original grid input or anything... Just check if grid[r][c2] == 1;
0 Share Reply

VectorX ★ 0 April 5, 2019 2:52 AM
question
N = sum(len(row) for row in grid)
or
N = sum(len(row) for row in rows) ?
0 Share Reply

mwacc ★ 2 February 3, 2019 7:04 AM
magic number 200 in java solution, 1st approach
0 Share Reply
SHOW 2 REPLIES

edaengineer ★ 255 June 3, 2018 5:59 AM
@awice
> Say the entire top row is filled with 1s. When looking at the next row with say,
so the worst case complexity for 2nd approach should still be same as the 1st one. Correct? because
Read More
0 Share Reply

leetcode_deleted_user ★ 246 February 19, 2018 1:18 PM
For solution 2, there can be at most R "weak rows" and each row take O(N). So I think the time complexity would be O(N(R+sqrt(N) + RC). Please correct me if I am wrong..
0 Share Reply

protocol1919 ★ 12 January 23, 2018 1:45 PM
The Time Complexity of Approach #2 is wrong. It should be at least `O(R*C)`
Because of the follow code:

```
for (int r = 0; r < grid.length; ++r) {
```

Read More
0 Share Reply
SHOW 1 REPLY