# 题目

给定一个只包含0和1的网格，找出其中角矩形的数量。

一个「角矩形」是由四个不同的在网格上的1形成的轴对称的矩形。注意只有角的位置才需要为1。并且，4个1需要是不同的。

示例 1：

输入：grid =

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

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

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

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

输出：1

解释：只有一个角矩形，角的位置为 grid[1][2], grid[1][4], grid[3][2], grid[3][4]。

示例 2：

输入：grid =

[[1, 1, 1],

[1, 1, 1],

[1, 1, 1]]

输出：9

解释：这里有 4 个 2x2 的矩形，4 个 2x3 和 3x2 的矩形和 1 个 3x3 的矩形。

示例 3：

输入：grid =

[[1, 1, 1, 1]]

输出：0

解释：矩形必须有 4 个不同的角。

提示：

网格 grid 中行和列的数目范围为 [1, 200]。

每个网格 grid[i][j] 中的值不是 0 就是 1 。

网格中 1 的个数不会超过 6000。

# 分析

## 方法一：

我们可以转换一下思想：每增加一行，角矩形的数量增加了多少。

算法：

我们用count[i, j] 来记录row[i] = row[j] = 1的次数。当我们处理新的一行时，对于每一对new\_row[i] = new\_row[j] = 1，我们添加count[i, j]到答案中，然后count[i, j]++。

class Solution {

public int countCornerRectangles(int[][] grid) {

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

int ans = 0;

for (int[] row: grid) {

for (int c1 = 0; c1 < row.length; ++c1) if (row[c1] == 1) {

for (int c2 = c1+1; c2 < row.length; ++c2) if (row[c2] == 1) {

int pos = c1 \* 200 + c2;

int c = count.getOrDefault(pos, 0);

ans += c;

count.put(pos, c+1);

}

}

}

return ans;

}

}

复杂度分析

时间复杂度：O(R\*C^2)。其中R, C指的是行和列。

空间复杂度：使用了O(C^2)的额外空间。

## 方法二：

算法：

我们能改进方法1中的方法吗？当一行有X个1时，我们需要O(X2)的时间来枚举每对1。当X很小时，这是可以接受的；但当X很大时，这是较为耗时的操作。

假设第一行的元素都是1时，f 指的是下一行和第一行所匹配 1 的数量。所能够构造角矩形的数量就是所匹配1的数量的对数，即f \* (f-1) / 2。我们可以使用一个集合和对每行进行简单线性扫描快速找到每个f。

**代码：**

class Solution {

public int countCornerRectangles(int[][] grid) {

List<List<Integer>> rows = new ArrayList();

int N = 0;

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

rows.add(new ArrayList());

for (int c = 0; c < grid[r].length; ++c)

if (grid[r][c] == 1) {

rows.get(r).add(c);

N++;

}

}

int sqrtN = (int) Math.sqrt(N);

int ans = 0;

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

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

if (rows.get(r).size() >= sqrtN) {

Set<Integer> target = new HashSet(rows.get(r));

for (int r2 = 0; r2 < grid.length; ++r2) {

if (r2 <= r && rows.get(r2).size() >= sqrtN)

continue;

int found = 0;

for (int c2: rows.get(r2))

if (target.contains(c2))

found++;

ans += found \* (found - 1) / 2;

}

} else {

for (int i1 = 0; i1 < rows.get(r).size(); ++i1) {

int c1 = rows.get(r).get(i1);

for (int i2 = i1 + 1; i2 < rows.get(r).size(); ++i2) {

int c2 = rows.get(r).get(i2);

int ct = count.getOrDefault(200\*c1 + c2, 0);

ans += ct;

count.put(200\*c1 + c2, ct + 1);

}

}

}

}

return ans;

}

}

复杂度分析：

时间复杂度：



其中N是网格中的个数。

空间复杂度：O(N + R + C^2)，rows, target和count所使用的空间。

## 方法三：动态规划

dp[j][k] 表示矩阵从第1行开始到当前正在处理的行，j点和k点两个点都为1的行的个数，比如下面的矩阵；

当i=0, dp[1][3]=0;

当i=1, dp[1][3]=1;

当i=2, dp[1][3]=2;

当i=3, dp[1][3]=3;

当i=4, dp[1][3]=3;

简单说dp[j][k]就是表示的截止到当前行 以j、k为顶点 的边数；记为边j-k

0 0 0 0 0

0 1 0 1 0

0 1 0 1 0

0 1 0 1 0

0 0 0 0 0

注意代码中的grid[i][j]和grid[i][k]；表示处理到第i行，发现这一行的j和k都为1；那我们以j-k为底边，往上找有几条边可以和这条边组成矩形；注意此时dp[j][k] 没有 更新，它正好表示的就是当前行之上的j-k边数

最后别忘了更新当前行的dp[j][k]++，后面的行里可能还要用

代码：

int countCornerRectangles(vector<vector<int>>& grid) {

int m = grid.size(), n = grid[0].size();

vector<vector<int>> dp(n, vector<int>(n, 0));

int ans = 0;

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

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

if (grid[i][j]) {

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

if (grid[i][k]) {

ans += dp[j][k];

dp[j][k]++;

}

}

}

}

}

return ans;

}