符号三角形

题目描述

符号三角形的第1行有n个由“+”和“-”组成的符号，以后每行符号比上行少1个，2个同号下面是“+”，2个异号下面是“-”。计算有多少个不同的符号三角形，使其所含“+”和“-”的个数相同。

n=7时的1个符号三角形如下：

```
+ + + - - + +
+ - - - + +
+ - - - - +
- + + + -
- + + -
- + -
- -
+
```

关于输入

每行1个正整数n<17，n=0退出。

关于输出

首先输出一个result:

然后一行输出'n 对应的解'

M16531:上机考试

总时间限制: 1000ms
内存限制: 65536KB

描述

一个班的同学在M行*N列的机房考试，给出学生的座位分布和每个人的做题情况，统计做题情况与周围（上下左右）任一同学相同的学生人数。
另外，由于考试的优秀率不能超过40%，请统计这个班的优秀人数（可能为0，相同分数的同学必须要么全是优秀，要么全不是优秀）。

输入

1. 第一行为两个整数，M和N。
2. 接下来M行每行N个整数，代表对应学生的编号，编号各不相同，范围由0到M*N-1。
3. 接下来M*N行，顺序给出编号由0到n-1的同学的做题情况，1代表做对，0代表做错。

输出

两个整数，以空格分隔。分别代表：

- “与周围（考虑上下左右四个方向）任一同学做题情况相同（不是“答题总数一样”，答题情况要一模一样）的学生人数”
- “班级优秀的人数”

样例输入1

```
2 5
0 1 2 3 4
5 6 7 8 9
1 1 1
1 0 0
0 0 1
0 0 1
0 0 1
0 1 0
1 0 1
1 1 1
1 1 1
1 1 1
```

样例输出1

```
6 0
```

样例1说明

编号为0 5 6 7 8 9的同学做题情况与周围同学相同，因此第一个整数是6。
全对的同学人数已经超过了40%，因此优秀的人数为0

样例输入2

```
1 3
0 1 2
1 0 0
0 0 0
0 0 0
```

样例输出2

```
0 1
```

样例2说明

并不存在与相邻同学做题情况相同的同学，并且做对一题的同学比例小于40%，因此有一人优秀

提示

- **tag**: matrix

1. M*N行做题情况可能为空。因为如果所有学生做题过程中，一直没有提交，则空。
2. 空行(即ASCII的回车符)。空行与全零不一样，不是“答题情况相同”。
3. 考试的题目数不定。

来源

cs10117 Final Exam

解题思路

1. **输入处理**：读取机房的行列数、座位编号矩阵，以及每个学生的做题情况。
2. **统计周围做题情况相同的学生**：遍历每个座位，检查其上下左右邻居的做题情况，若存在相同则计入集合（避免重复计数）。
3. **统计优秀人数**：
    - 按做题情况分组，计算每组的得分（做对题数）。
    - 找到得分最高的组，统计这些组的总人数。
    - 若总人数不超过班级人数的 40%，则该人数为优秀人数；否则优秀人数为 0。

Python 代码实现（带注释）

```
# 读取机房的行列数M和N
M, N = map(int, input().split())

# 读取座位矩阵：seat_matrix[i][j]表示第i行第j列的学生编号
seat_matrix = []
for _ in range(M):
    row = list(map(int, input().split()))
    seat_matrix.append(row)

# 读取每个学生的做题情况：solutions[id]存储编号id的做题情况（元组形式，方便比较）
total_students = M * N
solutions = {}
for student_id in range(total_students):
    # 处理空行情况（题目提示可能存在空提交）
    line = input().strip()
```

```
    # 将一行的0/1转为整数元组（空行则为空空元组）
    sol = tuple(map(int, line.split())) if line else ()
    solutions[student_id] = sol

# 第一步：统计“做题情况与周围（上下左右）任一同学相同”的学生人数
same_set = set()  # 用集合存符合条件的学生编号，避免重复
directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # 上下左右四个方向

for i in range(M):
    for j in range(N):
        current_id = seat_matrix[i][j]
        current_sol = solutions[current_id]
        # 检查四个方向的邻居
        for di, dj in directions:
            ni = i + di  # 邻居的行
            nj = j + dj  # 邻居的列
            # 判断邻居是否在合法范围内
            if 0 <= ni < M and 0 <= nj < N:
                neighbor_id = seat_matrix[ni][nj]
                neighbor_sol = solutions[neighbor_id]
                # 做题情况完全相同则计入集合
                if current_sol == neighbor_sol:
                    same_set.add(current_id)
                    break  # 找到一个邻居即可，无需继续检查

same_count = len(same_set)

# 第二步：统计优秀人数
# 1. 按做题情况分组，统计每组人数
group_dict = {}
for student_id in range(total_students):
    sol = solutions[student_id]
    if sol not in group_dict:
        group_dict[sol] = 0
    group_dict[sol] += 1

# 2. 计算每组的得分（做对题数），并找到最高得分
max_score = -1
group_scores = {}
for sol in group_dict:
    score = sum(sol)  # 做题情况中1的数量（做对题数）
    group_scores[sol] = score
    if score > max_score:
        max_score = score

# 3. 统计得分最高的所有组的总人数
sum_top_groups = 0
for sol in group_dict:
    if group_scores[sol] == max_score:
        sum_top_groups += group_dict[sol]

# 4. 计算优秀人数（不超过总人数的40%）
upper_limit = total_students * 0.4
```

```
excellent_count = sum_top_groups if sum_top_groups <= upper_limit else 0

# 输出结果
print(same_count, excellent_count)
```

代码详解

1. **输入处理**：
   - 读取`M`和`N`确定机房大小。
   - 读取`M`行座位编号，存储为二维列表`seat_matrix`。
   - 读取`M*N`行做题情况，用字典`solutions`存储（键为学生编号，值为做题情况的元组）。
2. **统计周围做题情况相同的学生**：
   - 遍历每个座位，检查其上下左右邻居。
   - 若邻居的做题情况与当前学生完全相同，将当前学生编号加入集合`same_set`（集合自动去重）。
   - 集合的长度即为第一个统计量。
3. **统计优秀人数**：
   - 按做题情况分组，统计每组人数（`group_dict`）。
   - 计算每组的得分（做对题数），找到最高得分`max_score`。
   - 统计所有最高得分组的总人数`sum_top_groups`。
   - 若`sum_top_groups`不超过班级人数的 40%，则为优秀人数；否则为 0。