1266.访问所有点的最小时间（简单）

1. 题目描述

   平面上有 `n` 个点，点的位置用整数坐标表示 `points[i] = [xi, yi]`。请你计算访问所有这些点需要的最小时间（以秒为单位）。

   你可以按照下面的规则在平面上移动：

   - 每一秒沿水平或者竖直方向移动一个单位长度，或者跨过对角线（可以看作在一秒内向水平和竖直方向各移动一个单位长度）。
   - 必须按照数组中出现的顺序来访问这些点。

   示例 1：

```
输入：points = [[1,1],[3,4],[-1,0]]
输出：7
解释：一条最佳的访问路径是： [1,1] -> [2,2] -> [3,3] -> [3,4] -> [2,3] -> [1,2] ->
[0,1] -> [-1,0]
从 [1,1] 到 [3,4] 需要 3 秒
从 [3,4] 到 [-1,0] 需要 4 秒
一共需要 7 秒
```

示例 2：

```
输入：points = [[3,2],[-2,2]]
输出：5
```

提示：

- `points.length == n`
- `1 <= n <= 100`
- `points[i].length == 2`
- `-1000 <= points[i][0], points[i][1] <= 1000`

2. 比赛实现

一步步模拟，横纵坐标都不一样就斜着走，否则水平竖直走

```cpp
class Solution {
public:
    int minTimeToVisitAllPoints(vector<vector<int>>& points) {
        int len = points.size();
        if(len <= 1) return 0;
        int ans = 0;
        vector<int> cur = points[0];
        for(int i = 1; i < len; i++){
            while(!(cur[0] == points[i][0] && cur[1] == points[i][1])){
                if(cur[0] != points[i][0]){
                    if(cur[0] > points[i][0])
                        cur[0]--;
                    else
                        cur[0]++;
                }
                if(cur[1] != points[i][1]){
                    if(cur[1] > points[i][1])
                        cur[1]--;
                    else
                        cur[1]++;
                }
                ans++;
            }
        }
        return ans;
    }
}
```

3. 自我改进

大跨步，不一步一步走了

```
class Solution {
public:
    int minTimeToVisitAllPoints(vector<vector<int>>& points) {
        int len = points.size();
        if(len <= 1) return 0;
        int ans = 0;
        vector<int> cur = points[0];
        for(int i = 1; i < len; i++){
            while(!(cur[0] == points[i][0] && cur[1] == points[i][1])){
                if(cur[0] != points[i][0] && cur[1] != points[i][1]){//横纵坐标都不一
样
                    //某一方向大跨步
                    int amount = min(abs(cur[0] - points[i][0]), abs(cur[1] -
points[i][1]));
                    if(cur[0] > points[i][0])
                        cur[0] -= amount;
                    else
                        cur[0] += amount;
                    if(cur[1] > points[i][1])
                        cur[1] -= amount;
                    else
                        cur[1] += amount;
                    ans += amount;
                }
                else if(cur[0] != points[i][0]){
                    ans += abs(cur[0] - points[i][0]);
                    cur[0] = points[i][0];
                    break;
                }
                else{ //(cur[1] != points[i][1])
                    ans += abs(cur[1] - points[i][1]);
                    cur[1] = points[i][1];
                    break;
                }
            }
        }
        return ans;
    }
};
```

4. 最优解法——切比雪夫距离

方法一：切比雪夫距离

对于平面上的两个点 `x = (x0, x1)` 和 `y = (y0, y1)`，设它们横坐标距离之差为 `dx = |x0 - y0|`，纵坐标距离之差为 `dy = |x1 - y1|`，对于以下三种情况，我们可以分别计算出从 `x` 移动到 `y` 的最少次数：

- `dx < dy`：沿对角线移动 `dx` 次，再竖直移动 `dy - dx` 次，总计 `dx + (dy - dx) = dy` 次；
- `dx == dy`：沿对角线移动 `dx` 次；
- `dx > dy`：沿对角线移动 `dy` 次，再水平移动 `dx - dy` 次，总计 `dy + (dx - dy) = dx` 次。

可以发现，对于任意一种情况，从 `x` 移动到 `y` 的最少次数为 `dx` 和 `dy` 中的较大值 `max(dx, dy)`，这也被称作 `x` 和 `y` 之间的 切比雪夫距离。

由于题目要求，需要按照数组中出现的顺序来访问这些点。因此我们遍历整个数组，对于数组中的相邻两个点，计算出它们的切比雪夫距离，所有的距离之和即为答案。

C++ | Python

```
class Solution {
public:
    int minTimeToVisitAllPoints(vector<vector<int>>& points) {
        int x0 = points[0][0], x1 = points[0][1];
        int ans = 0;
        for (int i = 1; i < points.size(); ++i) {
            int y0 = points[i][0], y1 = points[i][1];
            ans += max(abs(x0 - y0), abs(x1 - y1));
            x0 = y0;
            x1 = y1;
        }
        return ans;
    }
};
```

1267.统计参与通信的服务器（中等）

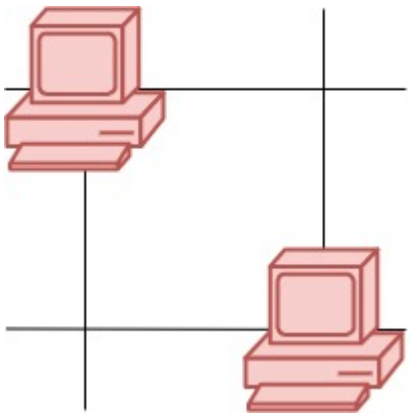
1. 题目描述

这里有一幅服务器分布图，服务器的位置标识在 `m * n` 的整数矩阵网格 `grid` 中，1 表示单元格上有服务器，0 表示没有。

如果两台服务器位于同一行或者同一列，我们就认为它们之间可以进行通信。

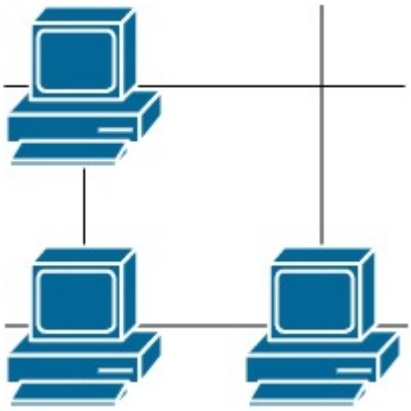
请你统计并返回能够与至少一台其他服务器进行通信的服务器的数量。

示例 1：



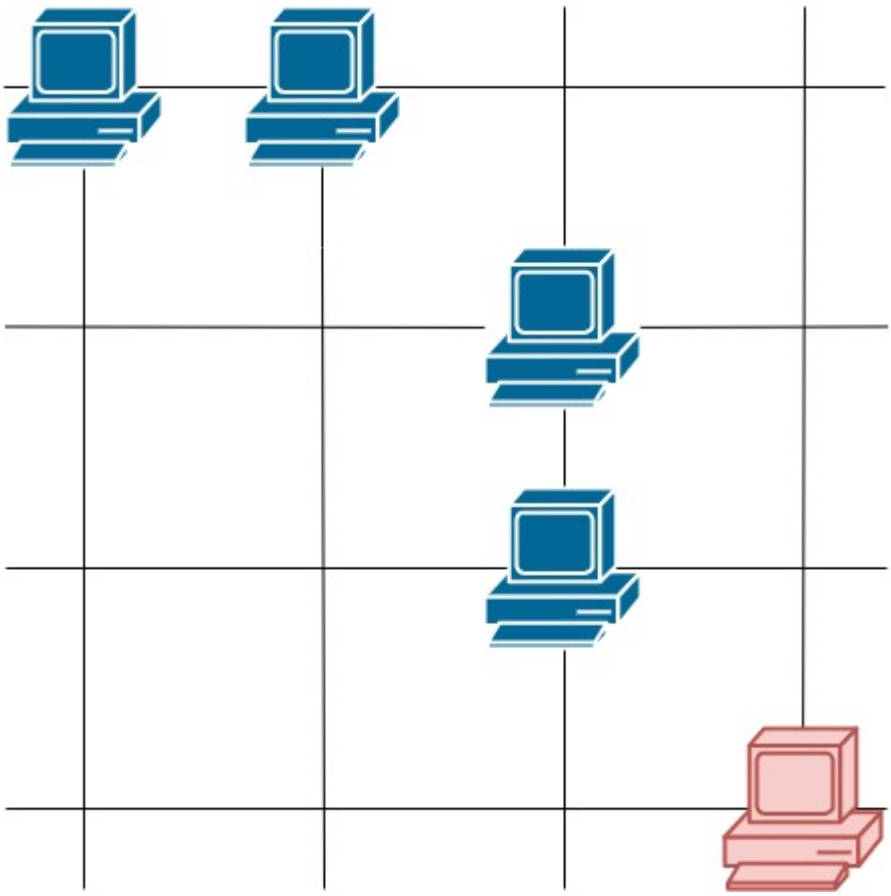
```
输入：grid = [[1,0],[0,1]]
输出：0
解释：没有一台服务器能与其他服务器进行通信。
```

示例 2：



```
输入：grid = [[1,0],[1,1]]
输出：3
解释：所有这些服务器都至少可以与一台别的服务器进行通信。
```

示例 3：



```
输入：grid = [[1,1,0,0],[0,0,1,0],[0,0,1,0],[0,0,0,1]]
输出：4
解释：第一行的两台服务器互相通信，第三列的两台服务器互相通信，但右下角的服务器无法与其他服务器通信。
```

提示：

- `m == grid.length`
- `n == grid[i].length`
- `1 <= m <= 250`
- `1 <= n <= 250`
- `grid[i][j] == 0 or 1`

2. 比赛实现

遍历每一行/列，超过两个就都数上，用set防止重复计数

```
class Solution {
public:
    int countServers(vector<vector<int>>& grid) {
        int m = grid.size();
        int n = grid[0].size();
        unordered_set<string> s;//用string记录可通信服务器的坐标
        for(int i = 0; i < m; i++){//遍历行
            int col = -1;
            int j = 0;
            while(j < n){
                if(grid[i][j] == 1){//找到第一个服务器，记录列值
                    col = j;
                    j++;
                    break;
                }
                j++;
            }
            while(j < n){
                if(grid[i][j] == 1){//又找到一台，故全部放入set
                    if(col != -1){//第一台也放进去
                        s.insert(to_string(i) + ',' + to_string(col));
                        col = -1;
                    }
                    s.insert(to_string(i) + ',' + to_string(j));
                }
                j++;
            }
        }
        for(int j = 0; j < n; j++){//按列，同理
            int row = -1;
            int i = 0;
            while(i < m){
                if(grid[i][j] == 1){
                    row = i;
                    i++;
                    break;
                }
                i++;
            }
            while(i < m){
                if(grid[i][j] == 1){
                    if(row != -1){
```

```
                        s.insert(to_string(row) + ',' + to_string(j));
                        row = -1;
                    }
                    s.insert(to_string(i) + ',' + to_string(j));
                }
                i++;
            }
        }
        return s.size();
    }
};
```

1268.搜索推荐系统（中等）

1. 题目描述

给你一个产品数组 `products` 和一个字符串 `searchWord` ， `products` 数组中每个产品都是一个字符串。

请你设计一个推荐系统，在依次输入单词 `searchWord` 的每一个字母后，推荐 `products` 数组中前缀与 `searchWord` 相同的最多三个产品。如果前缀相同的可推荐产品超过三个，请按字典序返回最小的三个。

请你以二维列表的形式，返回在输入 `searchWord` 每个字母后相应的推荐产品的列表。

示例 1：

```
输入：products = ["mobile","mouse","moneypot","monitor","mousepad"], searchWord =
"mouse"
输出：[
["mobile","moneypot","monitor"],
["mobile","moneypot","monitor"],
["mouse","mousepad"],
["mouse","mousepad"],
["mouse","mousepad"]
]
解释：按字典序排序后的产品列表是 ["mobile","moneypot","monitor","mouse","mousepad"]
输入 m 和 mo，由于所有产品的前缀都相同，所以系统返回字典序最小的三个产品
["mobile","moneypot","monitor"]
输入 mou， mous 和 mouse 后系统都返回 ["mouse","mousepad"]
```

示例 2：

```
输入：products = ["havana"], searchWord = "havana"
输出：[["havana"],["havana"],["havana"],["havana"],["havana"],["havana"]]
```

示例 3：

```
输入：products = ["bags","baggage","banner","box","cloths"], searchWord = "bags"
输出：[["baggage","bags","banner"],["baggage","bags","banner"],["baggage","bags"],
["bags"]]
```

示例 4：

```
输入：products = ["havana"], searchWord = "tatiana"
输出：[[],[],[],[],[],[],[]]
```

提示：

- `1 <= products.length <= 1000`
- `1 <= Σ products[i].length <= 2 * 10^4`
- `products[i]` 中所有的字符都是小写英文字母。
- `1 <= searchWord.length <= 1000`
- `searchWord` 中所有字符都是小写英文字母。

2. 比赛实现

先把products按字典序排好，依次匹配searchWord的第i位与products中各字符串的第i位，不一致则删去，每轮保留剩余的前三个字符串加入ans中

```
class Solution {
public:
    vector<vector<string>> suggestedProducts(vector<string>& products, string
searchWord) {
        vector<vector<string>> ans(searchWord.size());
        sort(products.begin(), products.end());
        for(int i = 0; i < searchWord.size(); i++){
            for(int j = 0; j < products.size(); j++){
                if(products[j][i] != searchWord[i]){
                    products.erase(products.begin()+j);
                    j--;
                }
            }
            vector<string> temp;
            for(int k = 0; k < 3 && k < products.size(); k++)
                temp.push_back(products[k]);
            ans[i] = temp;
        }
        return ans;
    }
};
```

3. 其他解法

官方题解有排序+二分的方法

1269.停在原地的方案数（困难）

1. 题目描述

有一个长度为 `arrLen` 的数组，开始有一个指针在索引 `0` 处。每一步操作中，你可以将指针向左或向右移动 1 步，或者停在原地（指针不能被移动到数组范围外）。

给你两个整数 `steps` 和 `arrLen` ，请你计算并返回：在恰好执行 `steps` 次操作以后，指针仍然指向索引 `0` 处的方案数。

由于答案可能会很大，请返回方案数 **模** `10^9 + 7` 后的结果。

示例 1：

```
输入：steps = 3, arrLen = 2
输出：4
解释：3 步后，总共有 4 种不同的方法可以停在索引 0 处。
向右，向左，不动
不动，向右，向左
向右，不动，向左
不动，不动，不动
```

示例 2：

```
输入：steps = 2, arrLen = 4
输出：2
解释：2 步后，总共有 2 种不同的方法可以停在索引 0 处。
向右，向左
不动，不动
```

示例 3：

```
输入：steps = 4, arrLen = 2
输出：8
```

提示：

- `1 <= steps <= 500`
- `1 <= arrLen <= 10^6`

2. 比赛超时

尝试了回溯法和动态规划，都超时，其中动态规划代码如下，`dp[i][j]`表示当前在索引i，还有j步可以走，此时回到原点0的方案数有多少，则状态转移方程为`dp[i][j] = dp[i-1][j-1] + dp[i][j-1] + dp[i+1][j-1]`

```
class Solution {
public:
    int numWays(int steps, int arrLen) {
        int ans = 0;
        int cur = 0;
        vector<vector<int>> dp = vector<vector<int>>(arrLen, vector<int>(steps+1,
0));
        dp[1][1] = dp[0][1] = 1;
        for(int j = 2; j <= steps; j++){
            for(int i = 0; i < arrLen; i++){
                dp[i][j] = dp[i][j-1];
                if(i > 0)
                    dp[i][j] += dp[i-1][j-1];
                dp[i][j] %= int(1e9+7);
                if(i < arrLen-1)
                    dp[i][j] += dp[i+1][j-1];
                dp[i][j] %= int(1e9+7);
            }
```

```
        }
        return dp[0][steps];
    }
};
```

即使后来发现第二维仅两个维度即可（只与j-1有关），但改完还是会超时

3. 官方题解——动态规划

WTF，因为能到达的地方不会超过steps，所以加一行代码就可以极大优化时间，从而通过！！！

`arrLen = min(arrLen, steps);`，再加上前面尝试过的空间优化，代码如下

```
class Solution {
public:
    int numWays(int steps, int arrLen) {
        int ans = 0;
        int cur = 0;
        arrLen = min(arrLen, steps);
        vector<vector<int>> dp = vector<vector<int>>(arrLen, vector<int>(2, 0));
        bool idx = 1;//指示奇数行或偶数行
        dp[1][idx] = dp[0][idx] = 1;
        for(int j = 2; j <= steps; j++){
            idx = !idx;
            for(int i = 0; i < arrLen; i++){
                dp[i][idx] = dp[i][!idx];
                if(i > 0)
                    dp[i][idx] += dp[i-1][!idx];
                dp[i][idx] %= int(1e9+7);
                if(i < arrLen-1)
                    dp[i][idx] += dp[i+1][!idx];
                dp[i][idx] %= int(1e9+7);
            }
        }
        return dp[0][steps%2];
    }
};
```