250709小码哥玩游戏（题解）

> 题目链接：
> https://www.matiji.net/exam/brushquestion/15/4446/16A92C42378232DEB56179D9C70DC45C?from=1

#以下题目可能有错别字等，完整题目请访问上述链接
小码哥和朋友在玩游戏，游戏内容为在给定的的m*nm*n列的数组中(1≤m，n≤1001≤m，n≤100)，该数组中元素由[A，Z]和0组成，其中0表示该位置无字符，然后由玩家指定一个字符（字符范围为[A，Z]），最后统计与该字符相邻的字符数量，最终数量最多的人胜出。现在请通过编程帮助小码哥计算出给定一个字符后最终相邻的字符数。

请注意：
• 最终的结果里，需要去重，比如，玩家指定的字符是A，A在[0,1]、[1,0]、[2,1]、[1,2]四个位置都有，而[1,1]是字符B，则B和这四个位置的字符A都是相邻的，但是B在最终的结果里只需要计数一次。
• 相邻只有上下左右相邻；
• 相邻的字符如果和指定字符一致，则不统计该相邻字符，即对于数组BAA，如果给定字符为A，则相邻字符数为1，其中B计数，而右侧的A不计数。

输入格式：
第一行输入三个数据，且以空格分隔，前两个数为数组的行m和列n（其中1≤m，n≤1001≤m，n≤100)，第三个字符为指定的字符;后面n行m列表示这个数组的字符分布情况，所有字符都在[A，Z]之间，且两个字符之间没有空格。
输出格式：
打印唯一的数字，表示相邻的字符数量。

输入：
```
2 3 H
00A
0HH
```
输出：
```
1
```

解法一：暴力遍历 + set去重

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    int m, n;
```

```
    char target;

    // 输入第一行：行数m，列数n，目标字符target
    cin >> m >> n >> target;

    // 创建一个vector<string>存储整个字符网格（m行n列）
    vector<string> grid(m);

    // 读取m行的字符矩阵，每行一个字符串
    for (int i = 0; i < m; ++i)
        cin >> grid[i];

    // 用set记录相邻的非目标字符（自动去重）
    set<char> neighbors;

    // 定义四个方向的偏移：上、下、左、右
    int dx[] = {-1, 1, 0, 0}; // 行方向：-1上，1下
    int dy[] = {0, 0, -1, 1}; // 列方向：-1左，1右

    // 遍历整个网格
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {

            // 如果当前位置是目标字符
            if (grid[i][j] == target) {

                // 检查四个方向的邻居
                for (int d = 0; d < 4; ++d) {
                    int ni = i + dx[d]; // 相邻格子的行
                    int nj = j + dy[d]; // 相邻格子的列

                    // 判断是否越界
                    if (ni >= 0 && ni < m && nj >= 0 && nj < n) {
                        char ch = grid[ni][nj];

                        // 如果相邻字符不是目标字符也不是空位（0），加入set中
                        if (ch != target && ch != '0')
                            neighbors.insert(ch);
                    }
                }
            }
        }
    }

    // 输出最终去重后的相邻字符个数
    cout << neighbors.size() << endl;
    return 0;
}
```

解法二：DFS连通块

> **DFS 连通块**：控制范围，避免重复访问，适用于区块式地图、连通区域

什么是 连通块?

在一个二维网格中，如果多个单元格值相同，且它们**通过上下左右方向相连**，我们称它们组成了一个“**连通块**”。

例子：

```
AAA
ABA
ACA
```

目标字符为 `A`，中间那一堆 `A` 连成了一个“连通块”。

为什么用 DFS 连通块?

在原问题中，我们是为了找出所有与目标字符相邻的**不同字符**，但是如果目标字符在地图上形成一个很大的连通区域，比如：

```
AAAA
AAAA
AAAA
```

我们不想对每个 `A` 反复访问重复的邻居。

用 DFS 来遍历目标字符形成的“连通块”可以：

- **只访问一次每个目标字符**（避免重复访问）
- **在 DFS 遍历时同时检查相邻格子**，直接收集邻居字符
- **结构清晰**，便于在大图中限定区域处理

DFS 连通块实现思路

1. 从任意一个目标字符格子出发
2. 遍历其上下左右方向
3. 如果是目标字符，就继续递归 DFS
4. 如果是非目标字符（且非 `'0'`），加入结果集合中

```cpp
#include <bits/stdc++.h>
using namespace std;
```

```cpp
int m, n;
char target;
vector<string> grid;
vector<vector<bool>> visited; // 标记每个位置是否已经访问过
set<char> neighbors; // 存储目标字符的邻接字符（去重）

// 方向数组：上、下、左、右
int dx[] = {-1, 1, 0, 0};
int dy[] = {0, 0, -1, 1};

// DFS函数：从(x, y)开始，遍历整个连通块
void dfs(int x, int y) {
    visited[x][y] = true;

    // 遍历四个方向
    for (int d = 0; d < 4; ++d) {
        int nx = x + dx[d];
        int ny = y + dy[d];

        // 边界检查
        if (nx >= 0 && nx < m && ny >= 0 && ny < n) {
            char ch = grid[nx][ny];

            if (ch == target && !visited[nx][ny]) {
                // 如果是未访问的目标字符，继续DFS
                dfs(nx, ny);
            } else if (ch != target && ch != '0') {
                // 是相邻的非目标字符且不是空格，加入结果集合
                neighbors.insert(ch);
            }
        }
    }
}

int main() {
    cin >> m >> n >> target;
    grid.resize(m);
    visited.resize(m, vector<bool>(n, false)); // 初始化访问标记

    for (int i = 0; i < m; ++i)
        cin >> grid[i];

    // 遍历全图，找出未访问的目标字符作为起点
    for (int i = 0; i < m; ++i)
        for (int j = 0; j < n; ++j)
            if (grid[i][j] == target && !visited[i][j])
                dfs(i, j); // 从该位置开始DFS

    cout << neighbors.size() << endl;
```

```
    return 0;
}
```

DFS 连通块适用场景总结

- 地图中目标字符 **成片分布**（例如岛屿、迷宫区域）
- 避免对同一个区域 **多次访问或重复统计**
- 扩展成“计算每个连通块的邻居数量”时更方便