【模拟】-最大相连男生数

题目描述与示例

题目描述

学校组织活动，将学生排成一个矩形方阵。

请在矩形方阵中找到最大的位置相连的男生数量。

这个**相连位置在一个直线上**，方向可以是水平的、垂直的、成对角线的或者反对角线的。

注：学生个数不会超过 `10000` 。

输入描述

输入的第一行为矩阵的行数和列数，接下来的 `n` 行为矩阵元素，元素间用 `,` 分隔。

输出描述

输出一个整数，表示矩阵中最长的位置相连的男生个数。

示例

输入

```
3,4
F,M,M,F
F,M,M,F
F,F,F,M
```

输出

```
1 3
```

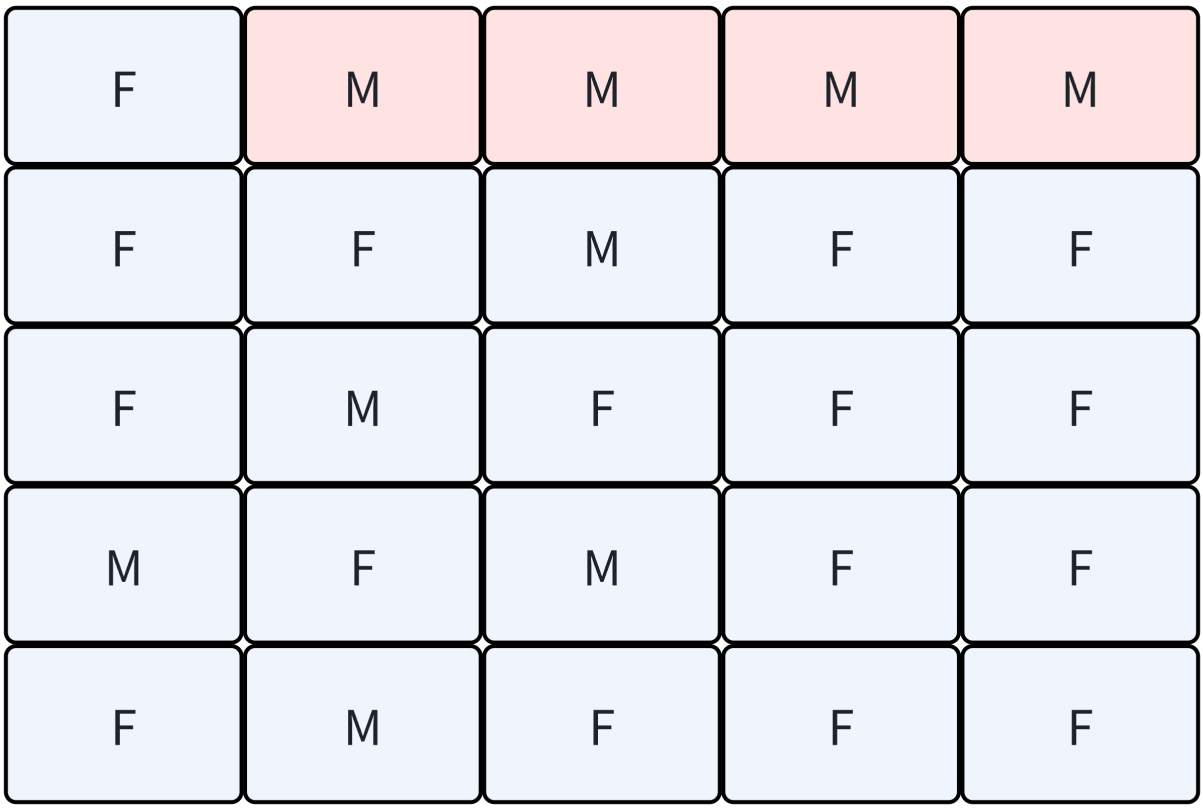
解题思路

本题题意并不难理解，其实就是考虑每一个 `M` 的横向、纵向、对角线、反对角线各自有多少连续的 `M` 。

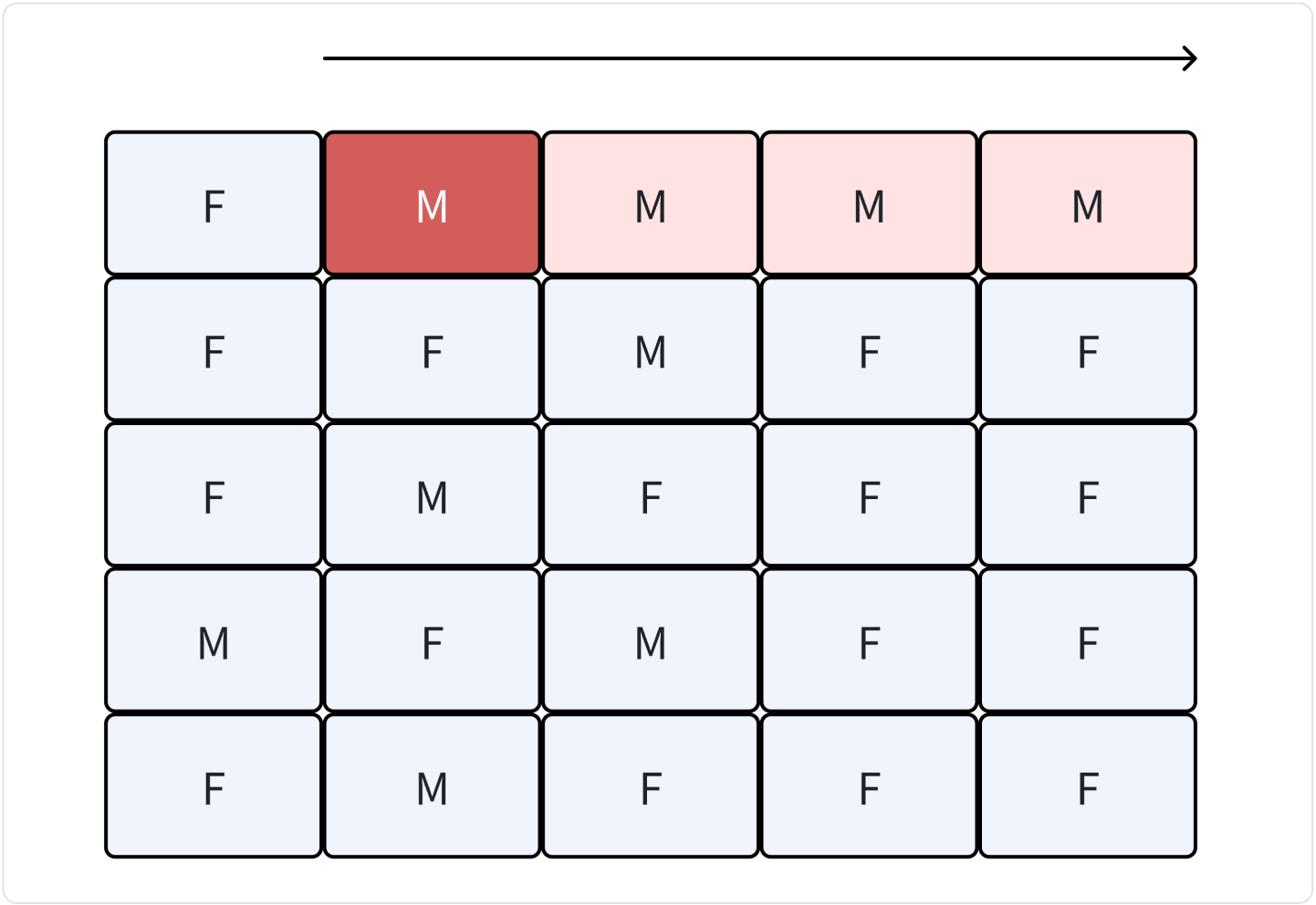
> `M` 表示 `male` ， `F` 表示 `female` ，这一点在题目中没有明说，但是容易想到。

对于某一段连续的 `M` ，很显然我们考虑最边上 `M` 往另一边延申，考虑中间的 `M` 往两边延申，最后计算得到的连续的 `M` 的个数是一样的。

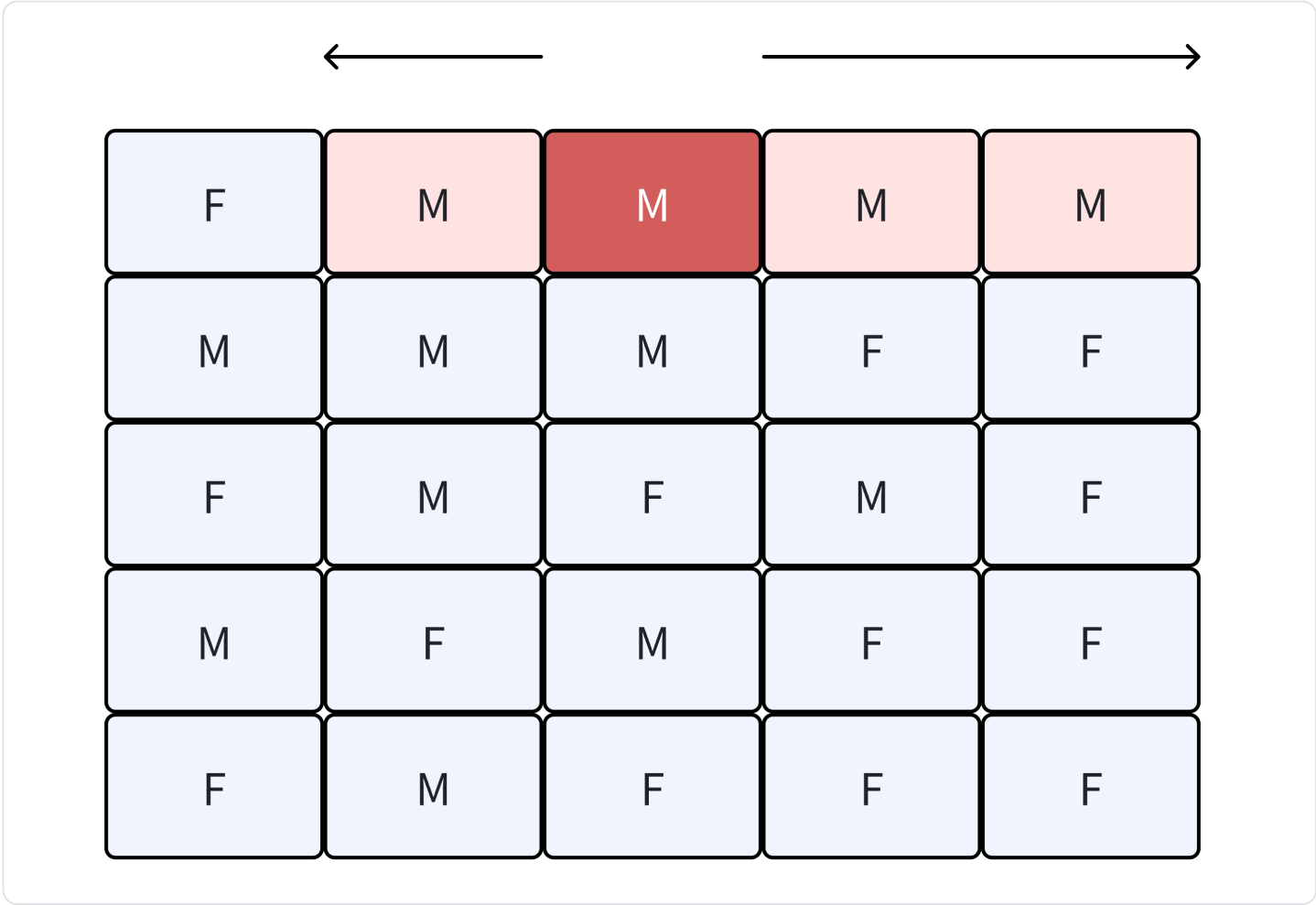
譬如



那么从边上的 `M` 往另一边计算



和从中间的 `M` 往两边计算



能够得到的连续的 `M` 的个数是一样的。

那么哪一种更加方便我们计算呢？

显然是前一种。

假设我们能够通过前一种方式，计算得到某段连续的 `M` 的个数，那么**后一种方式的计算是冗余的、无意义的计算**。

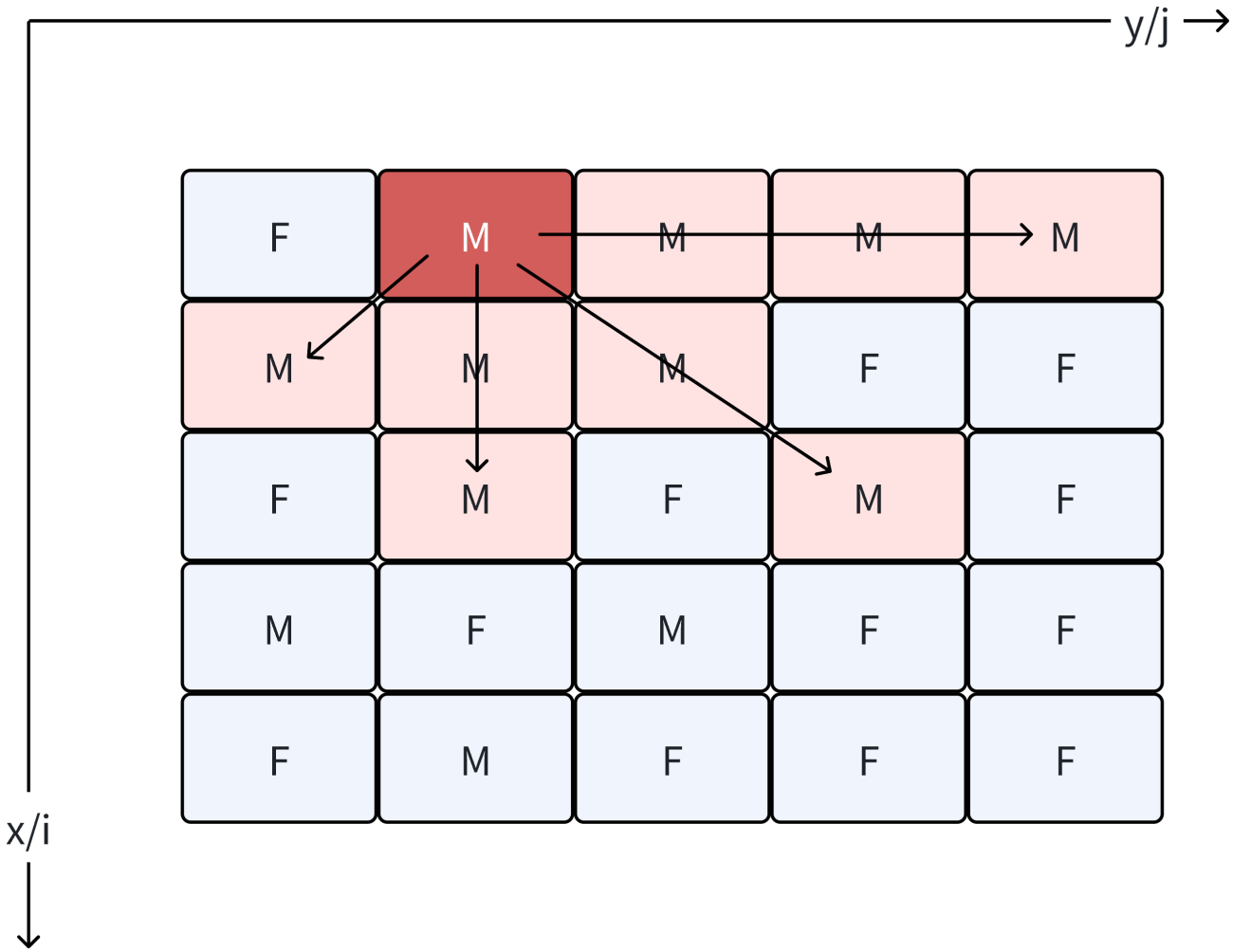
因此我们应该考虑，尽可能用**前一种简单方式**来计算连续的 `M` ，且需要**避免漏算**。

如何做到这一点呢？考虑一个例子。

如果我们**从上到下，从左到右**地考虑每一个矩阵中的字符串（即**顺序遍历**），容易发现，先遇到的 `M` ，一定是

- 横向的连续的 `M` 中最左边的那个
- 纵向的连续的 `M` 中最上边的那个

- 对角线的连续的 `M` 中最左上角的那个
- 反对角线的连续的 `M` 中最右上角的那个



显然，这个 `M` 对于四种方向而言，都是连续的 `M` 中，最边上的那个 `M` （深红色的 `M` ）。

所以，如果我们从上到下，从左到右地顺序遍历每一个元素的话，我们率先遇到那个 `M` 一定是满足前一种方式的计算思路的。

整体代码框架如下

```python
# 从上到下，从左到右顺序枚举矩阵中的每一个元素
for i in range(n):
    for j in range(m):
        # 考虑横向
        pass
```

```
6
7           # 考虑纵向
8           pass
9
10          # 考虑主对角线
11          pass
12
13          # 考虑反对角线
14          pass
```

那么，我们又应该如何避免对位于其他位置的 `M` （浅红色的 `M` ）进行重复计算，且不漏算呢？

以考虑横向方向为例，我们可以设计一**个大小和原矩阵完全一样**的 `n*m` 的检查矩阵 `check1` 。

`check1` 中的元素为 `0` 或 `1` 。初始化均为 `0` 。

- `check1[i][j] = 0` 表示某个位置 `(i, j)` 尚未被考虑包含在某段连续的横向的 `M` 中
- `check1[i][j] = 1` 表示某个位置 `(i, j)` 已经被考虑包含在某段连续的横向的 `M` 中了

对应的代码如下

```
1  check1 = [[0] * m for _ in range(n)]
2
3  # 从上到下，从左到右顺序枚举矩阵中的每一个元素
4  for i in range(n):
5      for j in range(m):
6          # 如果某个M尚未被考虑包含在某段连续的【横向】的M中
7          # 则【向右】考虑这整段横向的M
8          if mat[i][j] == "M" and check1[i][j] == 0:
9              # (x,y)为横向移动时的坐标
10             # (dx,dy)为横向移动的坐标偏差
11             x, y, dx, dy = i, j, 0, 1
12             # 当x和y不越界，且mat[x][y]仍为M时，进行横向移动的考虑
13             # 修改check[x][y]为已检查过，且(x,y)修改
14             while 0 <= x < n and 0 <= y < m and mat[x][y] == "M" and check1[x]
   [y] == 0:
15                 check1[x][y] = 1
16                 x += dx
17                 y += dy
18             # 退出while循环后，此段横向的连续的M的长度为y-j，更新答案
```

```
19                 ans = max(ans, y-j)
```

而对应的其他三个方向，也可以直接如法炮制了。

这样我们就可以使用 `4` 个不同的 `check` 矩阵，来判断某一个位置的 `M` 是否已经在某个方向上计算过，来避免发生无用的重复计算了。

这样就完成了本题分类写法（较为臃肿）。

> PS：四个方向的 `check` 矩阵，均仅由 `0` 或 `1` 构成。思考以前讲过的**状态压缩**技巧，你能否仅用一个其中元素范围为整数 `0-15` 的 `check` 矩阵来表示所有状态？题解的最后提供了状态压缩写法的代码。

代码

代码一：分类写法

Python

```python
# 题目：【模拟】2024E-最大相连男生数
# 分值：200
# 作者：许老师-闭着眼睛学数理化
# 算法：模拟，矩阵
# 代码看不懂的地方，请直接在群上提问


# 输入矩阵的行数n，列数m
n, m = map(int, input().split(","))

# 初始化二维矩阵
mat = list()
# 循环n行，输入矩阵
for _ in range(n):
    mat.append(input().split(","))


# 初始化答案变量，指的是最长连续男生数量
```

```python
ans = 0

# 四个检查矩阵，分别表示横向、纵向、对角线、反对角线方向上
# 某一个M是否已经在某段连续的M中被考虑过
check1 = [[0] * m for _ in range(n)]
check2 = [[0] * m for _ in range(n)]
check3 = [[0] * m for _ in range(n)]
check4 = [[0] * m for _ in range(n)]


# 从上到下，从左到右顺序枚举矩阵中的每一个元素
for i in range(n):
    for j in range(m):
        # 如果某个M尚未被考虑包含在某段连续的【横向】的M中
        # 则【向右】考虑这整段横向的M
        if mat[i][j] == "M" and check1[i][j] == 0:
            # (x,y)为横向移动时的坐标
            # (dx,dy)为横向移动的坐标偏差
            x, y, dx, dy = i, j, 0, 1
            # 当x和y不越界，且mat[x][y]仍为M时，进行横向移动的考虑
            # 修改check1[x][y]为已检查过，且(x,y)修改
            while 0 <= x < n and 0 <= y < m and mat[x][y] == "M":
                check1[x][y] = 1
                x += dx
                y += dy
            # 退出while循环后，此段横向的连续的M的长度为y-j，更新答案
            ans = max(ans, y-j)

        # 如果某个M尚未被考虑包含在某段连续的【纵向】的M中
        # 则【向下】考虑这整段纵向的M
        if mat[i][j] == "M" and check2[i][j] == 0:
            # (x,y)为纵向移动时的坐标
            # (dx,dy)为纵向移动的坐标偏差
            x, y, dx, dy = i, j, 1, 0
            # 当x和y不越界，且mat[x][y]仍为M时，进行纵向移动的考虑
            # 修改check2[x][y]为已检查过，且(x,y)修改
            while 0 <= x < n and 0 <= y < m and mat[x][y] == "M":
                check2[x][y] = 1
                x += dx
                y += dy
            # 退出while循环后，此段纵向的连续的M的长度为x-i，更新答案
            ans = max(ans, x-i)

        # 如果某个M尚未被考虑包含在某段连续的【对角线方向】的M中
        # 则【向右下】考虑这整段对角线方向的M
        if mat[i][j] == "M" and check3[i][j] == 0:
            # (x,y)为对角线方向移动时的坐标
```

```
            # (dx,dy)为对角线方向移动的坐标偏差
            x, y, dx, dy = i, j, 1, 1
            # 当x和y不越界，且mat[x][y]仍为M时，进行对角线方向移动的考虑
            # 修改check3[x][y]为已检查过，且(x,y)修改
            while 0 <= x < n and 0 <= y < m and mat[x][y] == "M":
                check3[x][y] = 1
                x += dx
                y += dy
            # 退出while循环后，此段对角线方向的连续的M的长度为x-i，更新答案
            ans = max(ans, x-i)


        # 如果某个M尚未被考虑包含在某段连续的【反对角线方向】的M中
        # 则【向左下】考虑这整段反对角线方向的M
        if mat[i][j] == "M" and check4[i][j] == 0:
            # (x,y)为反对角线方向移动时的坐标
            # (dx,dy)为反对角线方向移动的坐标偏差
            x, y, dx, dy = i, j, 1, -1
            # 当x和y不越界，且mat[x][y]仍为M时，进行反对角线方向移动的考虑
            # 修改check4[x][y]为已检查过，且(x,y)修改
            while 0 <= x < n and 0 <= y < m and mat[x][y] == "M":
                check4[x][y] = 1
                x += dx
                y += dy
            # 退出while循环后，此段反对角线方向的连续的M的长度为x-i，更新答案
            ans = max(ans, x-i)

print(ans)
```

Java

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // 输入矩阵的行数n，列数m
        String[] input = scanner.nextLine().split(",");
        int n = Integer.parseInt(input[0]);
        int m = Integer.parseInt(input[1]);

        // 初始化二维矩阵
        String[][] mat = new String[n][m];
```

```
        // 循环n行，输入矩阵
        for (int i = 0; i < n; i++) {
            mat[i] = scanner.nextLine().split(",");
        }

        // 初始化答案变量，指的是最长连续男生数量
        int ans = 0;

        // 四个检查矩阵，分别表示横向、纵向、对角线、反对角线方向上
        // 某一个M是否已经在某段连续的M中被考虑过
        boolean[][] check1 = new boolean[n][m];
        boolean[][] check2 = new boolean[n][m];
        boolean[][] check3 = new boolean[n][m];
        boolean[][] check4 = new boolean[n][m];

        // 从上到下，从左到右顺序枚举矩阵中的每一个元素
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                // 如果某个M尚未被考虑包含在某段连续的【横向】的M中
                // 则【向右】考虑这整段横向的M
                if (mat[i][j].equals("M") && !check1[i][j]) {
                    // (x,y)为横向移动时的坐标
                    // (dx,dy)为横向移动的坐标偏差
                    int x = i, y = j, dx = 0, dy = 1;
                    // 当x和y不越界，且mat[x][y]仍为M时，进行横向移动的考虑
                    // 修改check1[x][y]为已检查过，且(x,y)修改
                    while (x >= 0 && x < n && y >= 0 && y < m && mat[x]
[y].equals("M")) {
                        check1[x][y] = true;
                        y += dy;
                    }
                    // 退出while循环后，此段横向的连续的M的长度为y-j，更新答案
                    ans = Math.max(ans, y - j);
                }

                // 如果某个M尚未被考虑包含在某段连续的【纵向】的M中
                // 则【向下】考虑这整段纵向的M
                if (mat[i][j].equals("M") && !check2[i][j]) {
                    // (x,y)为纵向移动时的坐标
                    // (dx,dy)为纵向移动的坐标偏差
                    int x = i, y = j, dx = 1, dy = 0;
                    // 当x和y不越界，且mat[x][y]仍为M时，进行纵向移动的考虑
                    // 修改check2[x][y]为已检查过，且(x,y)修改
                    while (x >= 0 && x < n && y >= 0 && y < m && mat[x]
[y].equals("M")) {
                        check2[x][y] = true;
                        x += dx;
```

```
                    }
                    // 退出while循环后，此段纵向的连续的M的长度为x-i，更新答案
                    ans = Math.max(ans, x - i);
                }

                // 如果某个M尚未被考虑包含在某段连续的【对角线方向】的M中
                // 则【向右下】考虑这整段对角线方向的M
                if (mat[i][j].equals("M") && !check3[i][j]) {
                    // (x,y)为对角线方向移动时的坐标
                    // (dx,dy)为对角线方向移动的坐标偏差
                    int x = i, y = j, dx = 1, dy = 1;
                    // 当x和y不越界，且mat[x][y]仍为M时，进行对角线方向移动的考虑
                    // 修改check3[x][y]为已检查过，且(x,y)修改
                    while (x >= 0 && x < n && y >= 0 && y < m && mat[x][y].equals("M")) {
                        check3[x][y] = true;
                        x += dx;
                        y += dy;
                    }
                    // 退出while循环后，此段对角线方向的连续的M的长度为x-i，更新答案
                    ans = Math.max(ans, x - i);
                }

                // 如果某个M尚未被考虑包含在某段连续的【反对角线方向】的M中
                // 则【向左下】考虑这整段反对角线方向的M
                if (mat[i][j].equals("M") && !check4[i][j]) {
                    // (x,y)为反对角线方向移动时的坐标
                    // (dx,dy)为反对角线方向移动的坐标偏差
                    int x = i, y = j, dx = 1, dy = -1;
                    // 当x和y不越界，且mat[x][y]仍为M时，进行反对角线方向移动的考虑
                    // 修改check4[x][y]为已检查过，且(x,y)修改
                    while (x >= 0 && x < n && y >= 0 && y < m && mat[x][y].equals("M")) {
                        check4[x][y] = true;
                        x += dx;
                        y += dy;
                    }
                    // 退出while循环后，此段反对角线方向的连续的M的长度为x-i，更新答案
                    ans = Math.max(ans, x - i);
                }
            }
        }

        // 输出答案
        System.out.println(ans);

        scanner.close();
```

```
    }
}

```

C++

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main() {
    // 输入矩阵的行数n，列数m
    string input;
    getline(cin, input);
    int n, m;
    sscanf(input.c_str(), "%d,%d", &n, &m);

    // 初始化二维矩阵
    vector<vector<string>> mat(n, vector<string>(m));
    // 循环n行，输入矩阵
    for (int i = 0; i < n; i++) {
        string line;
        getline(cin, line);
        size_t pos = 0;
        for (int j = 0; j < m; j++) {
            size_t next_pos = line.find(',', pos);
            mat[i][j] = line.substr(pos, next_pos - pos);
            pos = next_pos + 1;
        }
    }

    // 初始化答案变量，指的是最长连续男生数量
    int ans = 0;

    // 四个检查矩阵，分别表示横向、纵向、对角线、反对角线方向上
    // 某一个M是否已经在某段连续的M中被考虑过
    vector<vector<bool>> check1(n, vector<bool>(m, false));
    vector<vector<bool>> check2(n, vector<bool>(m, false));
    vector<vector<bool>> check3(n, vector<bool>(m, false));
    vector<vector<bool>> check4(n, vector<bool>(m, false));

    // 从上到下，从左到右顺序枚举矩阵中的每一个元素
    for (int i = 0; i < n; i++) {
```

```
40          for (int j = 0; j < m; j++) {
41              // 如果某个M尚未被考虑包含在某段连续的【横向】的M中
42              // 则【向右】考虑这整段横向的M
43              if (mat[i][j] == "M" && !check1[i][j]) {
44                  // (x,y)为横向移动时的坐标
45                  // (dx,dy)为横向移动的坐标偏差
46                  int x = i, y = j, dx = 0, dy = 1;
47                  // 当x和y不越界，且mat[x][y]仍为M时，进行横向移动的考虑
48                  // 修改check1[x][y]为已检查过，且(x,y)修改
49                  while (x >= 0 && x < n && y >= 0 && y < m && mat[x][y] == "M")
    {
50                      check1[x][y] = true;
51                      y += dy;
52                  }
53                  // 退出while循环后，此段横向的连续的M的长度为y-j，更新答案
54                  ans = max(ans, y - j);
55              }
56
57              // 如果某个M尚未被考虑包含在某段连续的【纵向】的M中
58              // 则【向下】考虑这整段纵向的M
59              if (mat[i][j] == "M" && !check2[i][j]) {
60                  // (x,y)为纵向移动时的坐标
61                  // (dx,dy)为纵向移动的坐标偏差
62                  int x = i, y = j, dx = 1, dy = 0;
63                  // 当x和y不越界，且mat[x][y]仍为M时，进行纵向移动的考虑
64                  // 修改check2[x][y]为已检查过，且(x,y)修改
65                  while (x >= 0 && x < n && y >= 0 && y < m && mat[x][y] == "M")
    {
66                      check2[x][y] = true;
67                      x += dx;
68                  }
69                  // 退出while循环后，此段纵向的连续的M的长度为x-i，更新答案
70                  ans = max(ans, x - i);
71              }
72
73              // 如果某个M尚未被考虑包含在某段连续的【对角线方向】的M中
74              // 则【向右下】考虑这整段对角线方向的M
75              if (mat[i][j] == "M" && !check3[i][j]) {
76                  // (x,y)为对角线方向移动时的坐标
77                  // (dx,dy)为对角线方向移动的坐标偏差
78                  int x = i, y = j, dx = 1, dy = 1;
79                  // 当x和y不越界，且mat[x][y]仍为M时，进行对角线方向移动的考虑
80                  // 修改check3[x][y]为已检查过，且(x,y)修改
81                  while (x >= 0 && x < n && y >= 0 && y < m && mat[x][y] == "M")
    {
82                      check3[x][y] = true;
83                      x += dx;
```

```
                    y += dy;
                }
                // 退出while循环后，此段对角线方向的连续的M的长度为x-i，更新答案
                ans = max(ans, x - i);
            }

            // 如果某个M尚未被考虑包含在某段连续的【反对角线方向】的M中
            // 则【向左下】考虑这整段反对角线方向的M
            if (mat[i][j] == "M" && !check4[i][j]) {
                // (x,y)为反对角线方向移动时的坐标
                // (dx,dy)为反对角线方向移动的坐标偏差
                int x = i, y = j, dx = 1, dy = -1;
                // 当x和y不越界，且mat[x][y]仍为M时，进行反对角线方向移动的考虑
                // 修改check4[x][y]为已检查过，且(x,y)修改
                while (x >= 0 && x < n && y >= 0 && y < m && mat[x][y] == "M")
{
                    check4[x][y] = true;
                    x += dx;
                    y += dy;
                }
                // 退出while循环后，此段反对角线方向的连续的M的长度为x-i，更新答案
                ans = max(ans, x - i);
            }
        }
    }

    // 输出答案
    cout << ans << endl;

    return 0;
}

```

代码二：合并写法

Python

```
# 题目：【模拟】2024E-最大相连男生数
# 分值：200
# 作者：许老师-闭着眼睛学数理化
# 算法：模拟，矩阵
# 代码看不懂的地方，请直接在群上提问

```

```python

# 四个方向的偏移变量
# DIRECTIONS[0]、DIRECTIONS[1]、DIRECTIONS[2]、DIRECTIONS[3]分别表示
# 横向向右、纵向向下、主对角线方向向右下、反对角线反向向左下的4个方向
DIRECTIONS = [(0, 1), (1, 0), (1, 1), (1, -1)]

# 输入矩阵的行数n，列数m
n, m = map(int, input().split(","))

# 初始化二维矩阵
mat = list()
# 循环n行，输入矩阵
for _ in range(n):
    mat.append(input().split(","))


# 初始化答案变量，指的是最长连续男生数量
ans = 0

# 四个检查矩阵，check[0]、check[1]、check[2]、check[3]
# 分别表示横向、纵向、对角线、反对角线方向上
# 某一个M是否已经在某段连续的M中被考虑过
check = [[[0] * m for _ in range(n)] for __ in range(4)]


# 从上到下，从左到右顺序枚举矩阵中的每一个元素
for i in range(n):
    for j in range(m):
        # 考虑横向、纵向、主对角线方向、反对角线方向4个方向
        for k in range(4):
            # 如果某个M尚未被考虑包含在某段连续的的M中
            # 则考虑这段M
            if mat[i][j] == "M" and check[k][i][j] == 0:
                # (x,y)为移动时的坐标
                # (dx,dy)为移动的坐标偏差
                x, y = i, j
                dx, dy = DIRECTIONS[k]
                # 当x和y不越界，且mat[x][y]仍为M时，进行进一步移动的考虑
                # 修改check[k][x][y]为已检查过，且(x,y)修改
                while 0 <= x < n and 0 <= y < m and mat[x][y] == "M":
                    check[k][x][y] = 1
                    x += dx
                    y += dy
                # 退出while循环后，此段横向的连续的M的长度为x-i或y-j（取其中较大值），更新答案
                ans = max(ans, max(x-i, y-j))

```

```
print(ans)
```

Java

```
import java.util.Scanner;

public class Main {

    // 四个方向的偏移变量
    // DIRECTIONS[0]、DIRECTIONS[1]、DIRECTIONS[2]、DIRECTIONS[3]分别表示
    // 横向向右、纵向向下、主对角线方向向右下、反对角线方向向左下的4个方向
    private static final int[][] DIRECTIONS = {
            {0, 1},   // 横向向右
            {1, 0},   // 纵向向下
            {1, 1},   // 主对角线方向向右下
            {1, -1}   // 反对角线方向向左下
    };

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // 输入矩阵的行数n，列数m
        String[] input = scanner.nextLine().split(",");
        int n = Integer.parseInt(input[0]);
        int m = Integer.parseInt(input[1]);

        // 初始化二维矩阵
        String[][] mat = new String[n][m];
        // 循环n行，输入矩阵
        for (int i = 0; i < n; i++) {
            mat[i] = scanner.nextLine().split(",");
        }

        // 初始化答案变量，指的是最长连续男生数量
        int ans = 0;

        // 四个检查矩阵，check[0]、check[1]、check[2]、check[3]
        // 分别表示横向、纵向、对角线、反对角线方向上
        // 某一个M是否已经在某段连续的M中被考虑过
        boolean[][][] check = new boolean[4][n][m];

        // 从上到下，从左到右顺序枚举矩阵中的每一个元素
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                // 考虑横向、纵向、主对角线方向、反对角线方向4个方向
```

```
                for (int k = 0; k < 4; k++) {
                    // 如果某个M尚未被考虑包含在某段连续的的M中
                    // 则考虑这段M
                    if (mat[i][j].equals("M") && !check[k][i][j]) {
                        // (x,y)为移动时的坐标
                        // (dx,dy)为移动的坐标偏差
                        int x = i, y = j;
                        int dx = DIRECTIONS[k][0], dy = DIRECTIONS[k][1];
                        // 当x和y不越界，且mat[x][y]仍为M时，进行进一步移动的考虑
                        // 修改check[k][x][y]为已检查过，且(x,y)修改
                        while (x >= 0 && x < n && y >= 0 && y < m && mat[x][y].equals("M")) {
                            check[k][x][y] = true;
                            x += dx;
                            y += dy;
                        }
                        // 退出while循环后，此段横向的连续的M的长度为x-i或y-j（取其中较大值），更新答案
                        ans = Math.max(ans, Math.max(x - i, y - j));
                    }
                }
            }
        }

        // 输出答案
        System.out.println(ans);

        scanner.close();
    }
}
```

C++

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

// 四个方向的偏移变量
// DIRECTIONS[0]、DIRECTIONS[1]、DIRECTIONS[2]、DIRECTIONS[3]分别表示
// 横向向右、纵向向下、主对角线方向向右下、反对角线方向向左下的4个方向
const int DIRECTIONS[4][2] = {
    {0, 1},   // 横向向右
```

```cpp
    {1, 0},   // 纵向向下
    {1, 1},   // 主对角线方向向右下
    {1, -1}   // 反对角线方向向左下
};

int main() {
    // 输入矩阵的行数n，列数m
    int n, m;
    char comma;
    cin >> n >> comma >> m;
    cin.ignore(); // 忽略换行符

    // 初始化二维矩阵
    vector<vector<string>> mat(n, vector<string>(m));
    // 循环n行，输入矩阵
    for (int i = 0; i < n; i++) {
        string line;
        getline(cin, line);
        size_t pos = 0;
        for (int j = 0; j < m; j++) {
            size_t next_pos = line.find(',', pos);
            mat[i][j] = line.substr(pos, next_pos - pos);
            pos = next_pos + 1;
        }
    }

    // 初始化答案变量，指的是最长连续男生数量
    int ans = 0;

    // 四个检查矩阵，check[0]、check[1]、check[2]、check[3]
    // 分别表示横向、纵向、对角线、反对角线方向上
    // 某一个M是否已经在某段连续的M中被考虑过
    vector<vector<vector<bool>>> check(4, vector<vector<bool>>(n, vector<bool>
(m, false)));

    // 从上到下，从左到右顺序枚举矩阵中的每一个元素
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            // 考虑横向、纵向、主对角线方向、反对角线方向4个方向
            for (int k = 0; k < 4; k++) {
                // 如果某个M尚未被考虑包含在某段连续的的M中
                // 则考虑这段M
                if (mat[i][j] == "M" && !check[k][i][j]) {
                    // (x,y)为移动时的坐标
                    // (dx,dy)为移动的坐标偏差
                    int x = i, y = j;
                    int dx = DIRECTIONS[k][0], dy = DIRECTIONS[k][1];
```

```
                    // 当x和y不越界，且mat[x][y]仍为M时，进行进一步移动的考虑
                    // 修改check[k][x][y]为已检查过，且(x,y)修改
                    while (x >= 0 && x < n && y >= 0 && y < m && mat[x][y] ==
"M") {
                        check[k][x][y] = true;
                        x += dx;
                        y += dy;
                    }
                    // 退出while循环后，此段横向的连续的M的长度为x-i或y-j（取其中较大
值），更新答案
                    ans = max(ans, max(x - i, y - j));
                }
            }
        }
    }

    // 输出答案
    cout << ans << endl;

    return 0;
}

```

*代码三：状态压缩写法

Python

```
# 题目：【模拟】2024E-最大相连男生数
# 分值：200
# 作者：许老师-闭着眼睛学数理化
# 算法：模拟，矩阵，位运算，状态压缩
# 代码看不懂的地方，请直接在群上提问


# 四个方向的偏移变量
# DIRECTIONS[0]、DIRECTIONS[1]、DIRECTIONS[2]、DIRECTIONS[3]分别表示
# 横向向右、纵向向下、主对角线方向向右下、反对角线反向向左下的4个方向
DIRECTIONS = [(0, 1), (1, 0), (1, 1), (1, -1)]

# 输入矩阵的行数n，列数m
n, m = map(int, input().split(","))

# 初始化二维矩阵
```

```
mat = list()
# 循环n行，输入矩阵
for _ in range(n):
    mat.append(input().split(","))


# 初始化答案变量，指的是最长连续男生数量
ans = 0

# 检查矩阵check，其中的数字的范围为0-15
# 每一个数字的二进制可以由4个数位来表示
# 从低位数起，第0-3位数位
# 分别表示横向、纵向、对角线方向、反对角线方向上
# 某一个M是否已经在某段连续的M中被考虑过
# 譬如3 = bin(0011)，表示对角线和反对角线尚未考虑，横向和纵向已经考虑过了
# 初始化均为0，表示都没有考虑过
check = [[0] * m for _ in range(n)]


# 从上到下，从左到右顺序枚举矩阵中的每一个元素
for i in range(n):
    for j in range(m):
        # 考虑横向、纵向、主对角线方向、反对角线方向4个方向
        for k in range(4):
            # 如果某个M尚未被考虑包含在某段连续的的M中
            # 则考虑这段M
            # (1 >> k)可以得到1左移k位后的结果，
            # 譬如考虑纵向时左移k = 1位得到 2 = bin(0010)
            # 如果check[i][j]和(1 << k)进行与运算后得到0，
            # 说明check[i][j]的二进制表示，从低位数起的第k位为0
            # 第k位对应的方向没有被考虑
            if mat[i][j] == "M" and (check[i][j] & (1 << k)) == 0:
                # (x,y)为移动时的坐标
                # (dx,dy)为横向移动的坐标偏差
                x, y = i, j
                dx, dy = DIRECTIONS[k]
                # 当x和y不越界，且mat[x][y]仍为M时，进行进一步移动的考虑
                # 修改check[k][x][y]为已检查过，且(x,y)修改
                while 0 <= x < n and 0 <= y < m and mat[x][y] == "M":
                    # 将check[x][y]的二进制表示的第k位，通过或运算从0修改为1
                    check[x][y] |= (1 << k)
                    x += dx
                    y += dy
                # 退出while循环后，此段横向的连续的M的长度为x-i或y-j（取其中较大值），更新答案
                ans = max(ans, max(x-i, y-j))

```

```
print(ans)
```

Java

```
import java.util.Scanner;

public class Main {

    // 四个方向的偏移变量
    // DIRECTIONS[0]、DIRECTIONS[1]、DIRECTIONS[2]、DIRECTIONS[3]分别表示
    // 横向向右、纵向向下、主对角线方向向右下、反对角线方向向左下的4个方向
    private static final int[][] DIRECTIONS = {
            {0, 1},   // 横向向右
            {1, 0},   // 纵向向下
            {1, 1},   // 主对角线方向向右下
            {1, -1}   // 反对角线方向向左下
    };

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // 输入矩阵的行数n，列数m
        String[] input = scanner.nextLine().split(",");
        int n = Integer.parseInt(input[0]);
        int m = Integer.parseInt(input[1]);

        // 初始化二维矩阵
        String[][] mat = new String[n][m];
        // 循环n行，输入矩阵
        for (int i = 0; i < n; i++) {
            mat[i] = scanner.nextLine().split(",");
        }

        // 初始化答案变量，指的是最长连续男生数量
        int ans = 0;

        // 检查矩阵check，其中的数字的范围为0-15
        // 每一个数字的二进制可以由4个数位来表示
        // 从低位数起，第0-3位数位
        // 分别表示横向、纵向、对角线方向、反对角线方向上
        // 某一个M是否已经在某段连续的M中被考虑过
        // 初始化均为0，表示都没有考虑过
        int[][] check = new int[n][m];

        // 从上到下，从左到右顺序枚举矩阵中的每一个元素
```

```
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                // 考虑横向、纵向、主对角线方向、反对角线方向4个方向
                for (int k = 0; k < 4; k++) {
                    // 如果某个M尚未被考虑包含在某段连续的的M中
                    // 则考虑这段M
                    // (1 >> k)可以得到1左移k位后的结果，
                    // 譬如考虑纵向时左移k = 1位得到 2 = bin(0010)
                    // 如果check[i][j]和(1 << k)进行与运算后得到0，
                    // 说明check[i][j]的二进制表示，从低位数起的第k位为0
                    // 第k位对应的方向没有被考虑
                    if (mat[i][j].equals("M") && (check[i][j] & (1 << k)) ==
0) {
                        // (x,y)为移动时的坐标
                        // (dx,dy)为移动的坐标偏差
                        int x = i, y = j;
                        int dx = DIRECTIONS[k][0], dy = DIRECTIONS[k][1];
                        // 当x和y不越界，且mat[x][y]仍为M时，进行进一步移动的考虑
                        // 修改check[x][y]为已检查过，且(x,y)修改
                        while (x >= 0 && x < n && y >= 0 && y < m && mat[x]
[y].equals("M")) {
                            // 将check[x][y]的二进制表示的第k位，通过或运算从0修改为
1
                            check[x][y] |= (1 << k);
                            x += dx;
                            y += dy;
                        }
                        // 退出while循环后，此段方向的连续的M的长度为x-i或y-j（取其中
较大值），更新答案
                        ans = Math.max(ans, Math.max(x - i, y - j));
                    }
                }
            }
        }

        // 输出答案
        System.out.println(ans);

        scanner.close();
    }
}

```

C++

```
```

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

// 四个方向的偏移变量
// DIRECTIONS[0]、DIRECTIONS[1]、DIRECTIONS[2]、DIRECTIONS[3]分别表示
// 横向向右、纵向向下、主对角线方向向右下、反对角线方向向左下的4个方向
const int DIRECTIONS[4][2] = {
    {0, 1},   // 横向向右
    {1, 0},   // 纵向向下
    {1, 1},   // 主对角线方向向右下
    {1, -1}   // 反对角线方向向左下
};

int main() {
    // 输入矩阵的行数n，列数m
    int n, m;
    char comma;
    cin >> n >> comma >> m;
    cin.ignore(); // 忽略换行符

    // 初始化二维矩阵
    vector<vector<string>> mat(n, vector<string>(m));
    // 循环n行，输入矩阵
    for (int i = 0; i < n; i++) {
        string line;
        getline(cin, line);
        size_t pos = 0;
        for (int j = 0; j < m; j++) {
            size_t next_pos = line.find(',', pos);
            mat[i][j] = line.substr(pos, next_pos - pos);
            pos = next_pos + 1;
        }
    }

    // 初始化答案变量，指的是最长连续男生数量
    int ans = 0;

    // 检查矩阵check，其中的数字的范围为0-15
    // 每一个数字的二进制可以由4个数位来表示
    // 从低位数起，第0-3位数位
    // 分别表示横向、纵向、对角线方向、反对角线方向上
    // 某一个M是否已经在某段连续的M中被考虑过
    // 初始化均为0，表示都没有考虑过
    vector<vector<int>> check(n, vector<int>(m, 0));
```

```
48
49      // 从上到下，从左到右顺序枚举矩阵中的每一个元素
50      for (int i = 0; i < n; i++) {
51          for (int j = 0; j < m; j++) {
52              // 考虑横向、纵向、主对角线方向、反对角线方向4个方向
53              for (int k = 0; k < 4; k++) {
54                  // 如果某个M尚未被考虑包含在某段连续的的M中
55                  // 则考虑这段M
56                  if (mat[i][j] == "M" && (check[i][j] & (1 << k)) == 0) {
57                      // (x,y)为移动时的坐标
58                      // (dx,dy)为移动的坐标偏差
59                      int x = i, y = j;
60                      int dx = DIRECTIONS[k][0], dy = DIRECTIONS[k][1];
61                      // 当x和y不越界，且mat[x][y]仍为M时，进行进一步移动的考虑
62                      // 修改check[x][y]为已检查过，且(x,y)修改
63                      while (x >= 0 && x < n && y >= 0 && y < m && mat[x][y] ==
    "M") {
64                          // 将check[x][y]的二进制表示的第k位，通过或运算从0修改为1
65                          check[x][y] |= (1 << k);
66                          x += dx;
67                          y += dy;
68                      }
69                      // 退出while循环后，此段方向的连续的M的长度为x-i或y-j（取其中较大
    值），更新答案
70                      ans = max(ans, max(x - i, y - j));
71                  }
72              }
73          }
74      }
75
76      // 输出答案
77      cout << ans << endl;
78
79      return 0;
80  }
81
```

时空复杂度

时间复杂度：`O(4NM)`。

空间复杂度：`O(4NM)`。检查数组所占空间。如果使用状态压缩，则降低到`O(NM)`。