

# 【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

```
1 3,4
2 F,M,M,F
3 F,M,M,F
4 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。

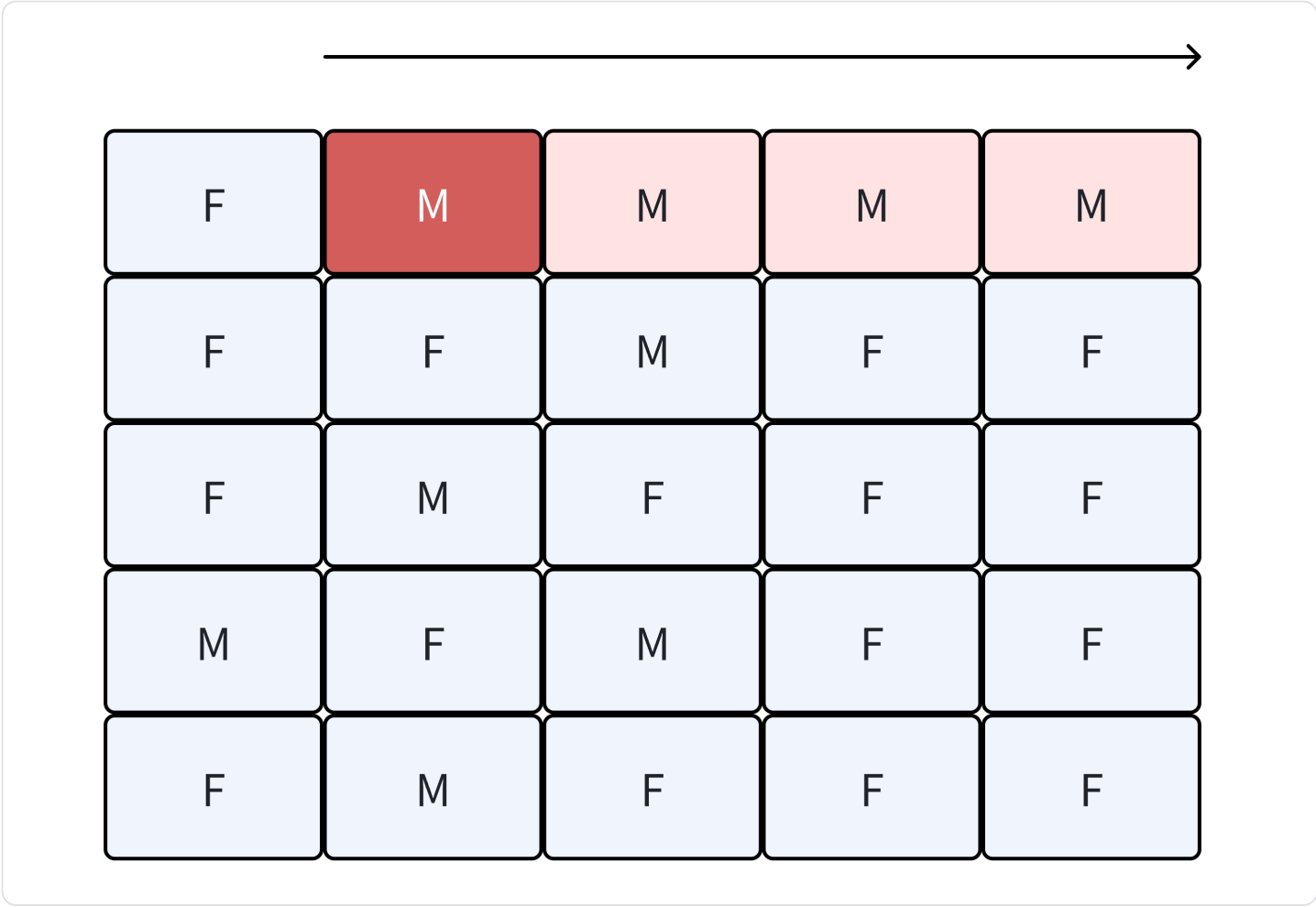
M 表示 male，F 表示 female，这一点在题目中没有明说，但是容易想到。

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

F	M	M	M	M
F	F	M	F	F
F	M	F	F	F
M	F	M	F	F
F	M	F	F	F

那么从边上的 M 往另一边计算



和从中间的 M 往两边计算

← →

F	M	M	M	M
M	M	M	F	F
F	M	F	M	F
M	F	M	F	F
F	M	F	F	F

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。

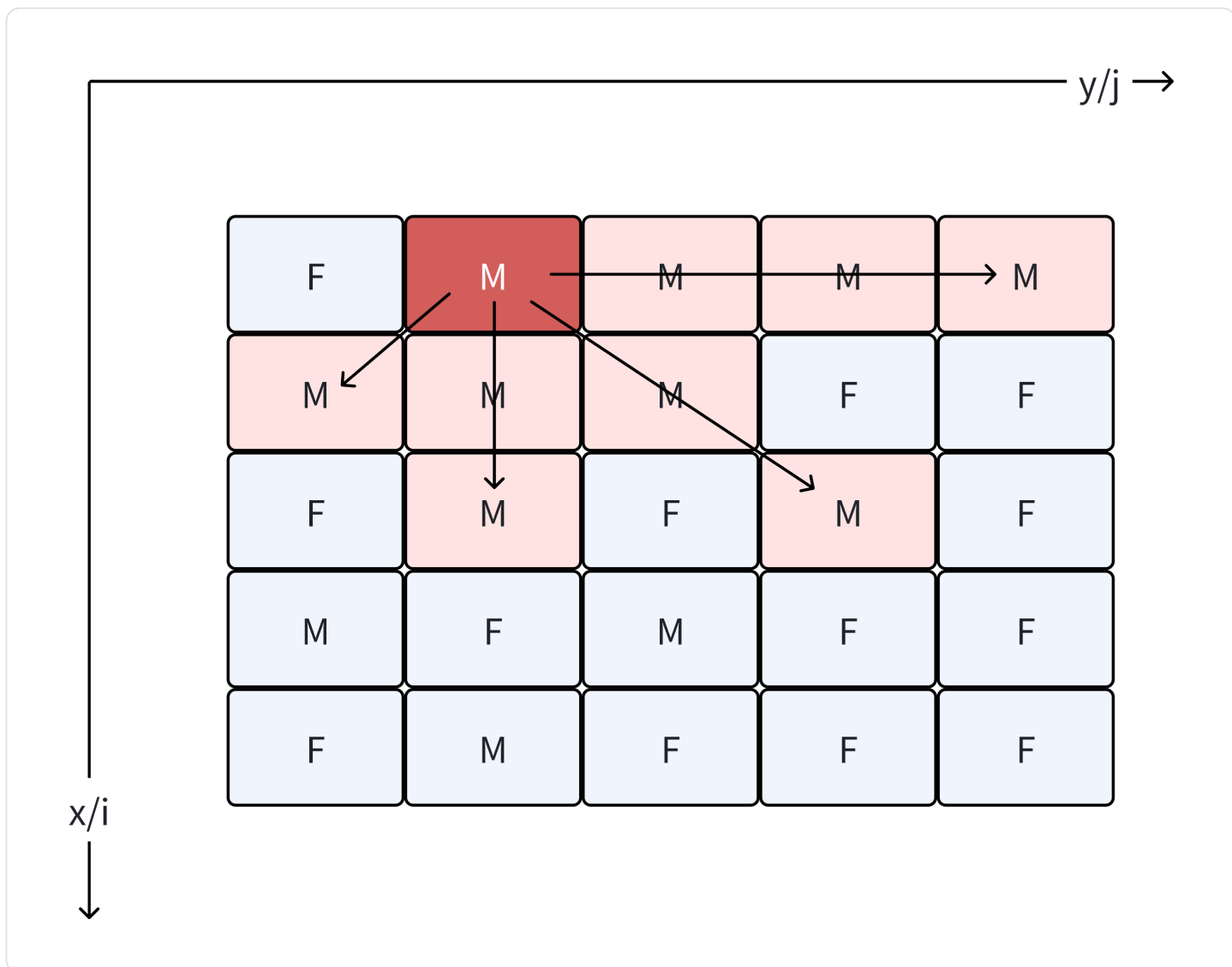
因此我们应该考虑，尽可能用前一种简单方式来计算连续的 M，且需要避免漏算。

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

```

1 # 从上到下，从左到右顺序枚举矩阵中的每一个元素
2 for i in range(n):
3     for j in range(m):
4         # 考虑横向
5         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**  $n \times m$  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

```

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

```
1 # 题目：【模拟】2024E-最大相连男生数
2 # 分值：200
3 # 作者：许老师-闭着眼睛学数理化
4 # 算法：模拟，矩阵
5 # 代码看不懂的地方，请直接在群上提问
6
7
8 # 输入矩阵的行数n，列数m
9 n, m = map(int, input().split(", "))
10
11 # 初始化二维矩阵
12 mat = list()
13 # 循环n行，输入矩阵
14 for _ in range(n):
15     mat.append(input().split(", "))
16
17
18 # 初始化答案变量，指的是最长连续男生数量
```

```

19 ans = 0
20
21 # 四个检查矩阵，分别表示横向、纵向、对角线、反对角线方向上
22 # 某一个M是否已经在某段连续的M中被考虑过
23 check1 = [[0] * m for _ in range(n)]
24 check2 = [[0] * m for _ in range(n)]
25 check3 = [[0] * m for _ in range(n)]
26 check4 = [[0] * m for _ in range(n)]
27
28
29 # 从上到下，从左到右顺序枚举矩阵中的每一个元素
30 for i in range(n):
31     for j in range(m):
32         # 如果某个M尚未被考虑包含在某段连续的【横向】的M中
33         # 则【向右】考虑这整段横向的M
34         if mat[i][j] == "M" and check1[i][j] == 0:
35             # (x,y)为横向移动时的坐标
36             # (dx,dy)为横向移动的坐标偏差
37             x, y, dx, dy = i, j, 0, 1
38             # 当x和y不越界，且mat[x][y]仍为M时，进行横向移动的考虑
39             # 修改check1[x][y]为已检查过，且(x,y)修改
40             while 0 <= x < n and 0 <= y < m and mat[x][y] == "M":
41                 check1[x][y] = 1
42                 x += dx
43                 y += dy
44             # 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
45             ans = max(ans, y-j)
46
47         # 如果某个M尚未被考虑包含在某段连续的【纵向】的M中
48         # 则【向下】考虑这整段纵向的M
49         if mat[i][j] == "M" and check2[i][j] == 0:
50             # (x,y)为纵向移动时的坐标
51             # (dx,dy)为纵向移动的坐标偏差
52             x, y, dx, dy = i, j, 1, 0
53             # 当x和y不越界，且mat[x][y]仍为M时，进行纵向移动的考虑
54             # 修改check2[x][y]为已检查过，且(x,y)修改
55             while 0 <= x < n and 0 <= y < m and mat[x][y] == "M":
56                 check2[x][y] = 1
57                 x += dx
58                 y += dy
59             # 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
60             ans = max(ans, x-i)
61
62         # 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
63         # 则【向右下】考虑这整段对角线方向的M
64         if mat[i][j] == "M" and check3[i][j] == 0:
65             # (x,y)为对角线方向移动时的坐标

```



```

66         # (dx,dy)为对角线方向移动的坐标偏差
67         x, y, dx, dy = i, j, 1, 1
68         # 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
69         # 修改check3[x][y]为已检查过, 且(x,y)修改
70         while 0 <= x < n and 0 <= y < m and mat[x][y] == "M":
71             check3[x][y] = 1
72             x += dx
73             y += dy
74         # 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
75         ans = max(ans, x-i)
76
77
78         # 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
79         # 则【向左下】考虑这整段反对角线方向的M
80         if mat[i][j] == "M" and check4[i][j] == 0:
81             # (x,y)为反对角线方向移动时的坐标
82             # (dx,dy)为反对角线方向移动的坐标偏差
83             x, y, dx, dy = i, j, 1, -1
84             # 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
85             # 修改check4[x][y]为已检查过, 且(x,y)修改
86             while 0 <= x < n and 0 <= y < m and mat[x][y] == "M":
87                 check4[x][y] = 1
88                 x += dx
89                 y += dy
90             # 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
91             ans = max(ans, x-i)
92
93     print(ans)

```

## Java

```

1  import java.util.Scanner;
2
3  public class Main {
4
5      public static void main(String[] args) {
6          Scanner scanner = new Scanner(System.in);
7
8          // 输入矩阵的行数n, 列数m
9          String[] input = scanner.nextLine().split(",");
10         int n = Integer.parseInt(input[0]);
11         int m = Integer.parseInt(input[1]);
12
13         // 初始化二维矩阵
14         String[][] mat = new String[n][m];

```

```

15 // 循环n行，输入矩阵
16 for (int i = 0; i < n; i++) {
17     mat[i] = scanner.nextLine().split(",");
18 }
19
20 // 初始化答案变量，指的是最长连续男生数量
21 int ans = 0;
22
23 // 四个检查矩阵，分别表示横向、纵向、对角线、反对角线方向上
24 // 某一个M是否已经在某段连续的M中被考虑过
25 boolean[][] check1 = new boolean[n][m];
26 boolean[][] check2 = new boolean[n][m];
27 boolean[][] check3 = new boolean[n][m];
28 boolean[][] check4 = new boolean[n][m];
29
30 // 从上到下，从左到右顺序枚举矩阵中的每一个元素
31 for (int i = 0; i < n; i++) {
32     for (int j = 0; j < m; j++) {
33         // 如果某个M尚未被考虑包含在某段连续的【横向】的M中
34         // 则【向右】考虑这整段横向的M
35         if (mat[i][j].equals("M") && !check1[i][j]) {
36             // (x,y)为横向移动时的坐标
37             // (dx,dy)为横向移动的坐标偏差
38             int x = i, y = j, dx = 0, dy = 1;
39             // 当x和y不越界，且mat[x][y]仍为M时，进行横向移动的考虑
40             // 修改check1[x][y]为已检查过，且(x,y)修改
41             while (x >= 0 && x < n && y >= 0 && y < m && mat[x]
[y].equals("M")) {
42                 check1[x][y] = true;
43                 y += dy;
44             }
45             // 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
46             ans = Math.max(ans, y - j);
47         }
48
49         // 如果某个M尚未被考虑包含在某段连续的【纵向】的M中
50         // 则【向下】考虑这整段纵向的M
51         if (mat[i][j].equals("M") && !check2[i][j]) {
52             // (x,y)为纵向移动时的坐标
53             // (dx,dy)为纵向移动的坐标偏差
54             int x = i, y = j, dx = 1, dy = 0;
55             // 当x和y不越界，且mat[x][y]仍为M时，进行纵向移动的考虑
56             // 修改check2[x][y]为已检查过，且(x,y)修改
57             while (x >= 0 && x < n && y >= 0 && y < m && mat[x]
[y].equals("M")) {
58                 check2[x][y] = true;
59                 x += dx;

```

```

60         }
61         // 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
62         ans = Math.max(ans, x - i);
63     }
64
65     // 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
66     // 则【向右下】考虑这整段对角线方向的M
67     if (mat[i][j].equals("M") && !check3[i][j]) {
68         // (x,y)为对角线方向移动时的坐标
69         // (dx,dy)为对角线方向移动的坐标偏差
70         int x = i, y = j, dx = 1, dy = 1;
71         // 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
72         // 修改check3[x][y]为已检查过，且(x,y)修改
73         while (x >= 0 && x < n && y >= 0 && y < m && mat[x]
[y].equals("M")) {
74             check3[x][y] = true;
75             x += dx;
76             y += dy;
77         }
78         // 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
79         ans = Math.max(ans, x - i);
80     }
81
82     // 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
83     // 则【向左下】考虑这整段反对角线方向的M
84     if (mat[i][j].equals("M") && !check4[i][j]) {
85         // (x,y)为反对角线方向移动时的坐标
86         // (dx,dy)为反对角线方向移动的坐标偏差
87         int x = i, y = j, dx = 1, dy = -1;
88         // 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
89         // 修改check4[x][y]为已检查过，且(x,y)修改
90         while (x >= 0 && x < n && y >= 0 && y < m && mat[x]
[y].equals("M")) {
91             check4[x][y] = true;
92             x += dx;
93             y += dy;
94         }
95         // 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
96         ans = Math.max(ans, x - i);
97     }
98 }
99 }
100
101 // 输出答案
102 System.out.println(ans);
103
104 scanner.close();

```

```
105     }
106 }
107
```

## C++

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4
5  using namespace std;
6
7  int main() {
8      // 输入矩阵的行数n, 列数m
9      string input;
10     getline(cin, input);
11     int n, m;
12     sscanf(input.c_str(), "%d,%d", &n, &m);
13
14     // 初始化二维矩阵
15     vector<vector<string>> mat(n, vector<string>(m));
16     // 循环n行, 输入矩阵
17     for (int i = 0; i < n; i++) {
18         string line;
19         getline(cin, line);
20         size_t pos = 0;
21         for (int j = 0; j < m; j++) {
22             size_t next_pos = line.find(',', pos);
23             mat[i][j] = line.substr(pos, next_pos - pos);
24             pos = next_pos + 1;
25         }
26     }
27
28     // 初始化答案变量, 指的是最长连续男生数量
29     int ans = 0;
30
31     // 四个检查矩阵, 分别表示横向、纵向、对角线、反对角线方向上
32     // 某一个M是否已经在某段连续的M中被考虑过
33     vector<vector<bool>> check1(n, vector<bool>(m, false));
34     vector<vector<bool>> check2(n, vector<bool>(m, false));
35     vector<vector<bool>> check3(n, vector<bool>(m, false));
36     vector<vector<bool>> check4(n, vector<bool>(m, false));
37
38     // 从上到下, 从左到右顺序枚举矩阵中的每一个元素
39     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             {
51                 check1[x][y] = true;
52                 y += dy;
53             }
54             // 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
55             ans = max(ans, y - j);
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             {
67                 check2[x][y] = true;
68                 x += dx;
69             }
70             // 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
71             ans = max(ans, x - i);
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             {
83                 check3[x][y] = true;
84                 x += dx;

```

```

84         y += dy;
85     }
86     // 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
87     ans = max(ans, x - i);
88 }
89
90 // 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
91 // 则【向左下】考虑这整段反对角线方向的M
92 if (mat[i][j] == "M" && !check4[i][j]) {
93     // (x,y)为反对角线方向移动时的坐标
94     // (dx,dy)为反对角线方向移动的坐标偏差
95     int x = i, y = j, dx = 1, dy = -1;
96     // 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
97     // 修改check4[x][y]为已检查过，且(x,y)修改
98     while (x >= 0 && x < n && y >= 0 && y < m && mat[x][y] == "M")
99     {
100         check4[x][y] = true;
101         x += dx;
102         y += dy;
103     }
104     // 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
105     ans = max(ans, x - i);
106 }
107 }
108
109 // 输出答案
110 cout << ans << endl;
111
112 return 0;
113 }
114

```

## 代码二：合并写法

### Python

```

1 # 题目：【模拟】2024E-最大相连男生数
2 # 分值：200
3 # 作者：许老师-闭着眼睛学数理化
4 # 算法：模拟，矩阵
5 # 代码看不懂的地方，请直接在群上提问
6

```

```

7
8 # 四个方向的偏移变量
9 # DIRECTIONS[0]、DIRECTIONS[1]、DIRECTIONS[2]、DIRECTIONS[3]分别表示
10 # 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
11 DIRECTIONS = [(0, 1), (1, 0), (1, 1), (1, -1)]
12
13 # 输入矩阵的行数n，列数m
14 n, m = map(int, input().split(","))
15
16 # 初始化二维矩阵
17 mat = list()
18 # 循环n行，输入矩阵
19 for _ in range(n):
20     mat.append(input().split(","))
21
22
23 # 初始化答案变量，指的是最长连续男生数量
24 ans = 0
25
26 # 四个检查矩阵，check[0]、check[1]、check[2]、check[3]
27 # 分别表示横向、纵向、对角线、反对角线方向上
28 # 某一个M是否已经在某段连续的M中被考虑过
29 check = [[[0] * m for _ in range(n)] for __ in range(4)]
30
31
32 # 从上到下，从左到右顺序枚举矩阵中的每一个元素
33 for i in range(n):
34     for j in range(m):
35         # 考虑横向、纵向、主对角线方向、反对角线方向4个方向
36         for k in range(4):
37             # 如果某个M尚未被考虑包含在某段连续的M中
38             # 则考虑这段M
39             if mat[i][j] == "M" and check[k][i][j] == 0:
40                 # (x,y)为移动时的坐标
41                 # (dx,dy)为移动的坐标偏差
42                 x, y = i, j
43                 dx, dy = DIRECTIONS[k]
44                 # 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
45                 # 修改check[k][x][y]为已检查过，且(x,y)修改
46                 while 0 <= x < n and 0 <= y < m and mat[x][y] == "M":
47                     check[k][x][y] = 1
48                     x += dx
49                     y += dy
50                 # 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
51                 ans = max(ans, max(x-i, y-j))
52

```

## Java

```

1  import java.util.Scanner;
2
3  public class Main {
4
5      // 四个方向的偏移变量
6      // DIRECTIONS[0]、DIRECTIONS[1]、DIRECTIONS[2]、DIRECTIONS[3]分别表示
7      // 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
8      private static final int[][] DIRECTIONS = {
9          {0, 1},    // 横向向右
10         {1, 0},    // 纵向向下
11         {1, 1},    // 主对角线方向向右下
12         {1, -1}    // 反对角线方向向左下
13     };
14
15     public static void main(String[] args) {
16         Scanner scanner = new Scanner(System.in);
17
18         // 输入矩阵的行数n, 列数m
19         String[] input = scanner.nextLine().split(",");
20         int n = Integer.parseInt(input[0]);
21         int m = Integer.parseInt(input[1]);
22
23         // 初始化二维矩阵
24         String[][] mat = new String[n][m];
25         // 循环n行, 输入矩阵
26         for (int i = 0; i < n; i++) {
27             mat[i] = scanner.nextLine().split(",");
28         }
29
30         // 初始化答案变量, 指的是最长连续男生数量
31         int ans = 0;
32
33         // 四个检查矩阵, check[0]、check[1]、check[2]、check[3]
34         // 分别表示横向、纵向、对角线、反对角线方向上
35         // 某一个M是否已经在某段连续的M中被考虑过
36         boolean[][][] check = new boolean[4][n][m];
37
38         // 从上到下, 从左到右顺序枚举矩阵中的每一个元素
39         for (int i = 0; i < n; i++) {
40             for (int j = 0; j < m; j++) {
41                 // 考虑横向、纵向、主对角线方向、反对角线方向4个方向

```



```

42         for (int k = 0; k < 4; k++) {
43             // 如果某个M尚未被考虑包含在某段连续的M中
44             // 则考虑这段M
45             if (mat[i][j].equals("M") && !check[k][i][j]) {
46                 // (x,y)为移动时的坐标
47                 // (dx,dy)为移动的坐标偏差
48                 int x = i, y = j;
49                 int dx = DIRECTIONS[k][0], dy = DIRECTIONS[k][1];
50                 // 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
51                 // 修改check[k][x][y]为已检查过, 且(x,y)修改
52                 while (x >= 0 && x < n && y >= 0 && y < m && mat[x]
[y].equals("M")) {
53                     check[k][x][y] = true;
54                     x += dx;
55                     y += dy;
56                 }
57                 // 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
58                 ans = Math.max(ans, Math.max(x - i, y - j));
59             }
60         }
61     }
62 }
63
64 // 输出答案
65 System.out.println(ans);
66
67 scanner.close();
68 }
69 }
70

```

## C++

```

1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4
5  using namespace std;
6
7  // 四个方向的偏移变量
8  // DIRECTIONS[0]、DIRECTIONS[1]、DIRECTIONS[2]、DIRECTIONS[3]分别表示
9  // 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
10 const int DIRECTIONS[4][2] = {
11     {0, 1},    // 横向向右

```

```

12     {1, 0},    // 纵向向下
13     {1, 1},    // 主对角线方向向右下
14     {1, -1}    // 反对角线方向向左下
15 };
16
17 int main() {
18     // 输入矩阵的行数n, 列数m
19     int n, m;
20     char comma;
21     cin >> n >> comma >> m;
22     cin.ignore(); // 忽略换行符
23
24     // 初始化二维矩阵
25     vector<vector<string>> mat(n, vector<string>(m));
26     // 循环n行, 输入矩阵
27     for (int i = 0; i < n; i++) {
28         string line;
29         getline(cin, line);
30         size_t pos = 0;
31         for (int j = 0; j < m; j++) {
32             size_t next_pos = line.find(',', pos);
33             mat[i][j] = line.substr(pos, next_pos - pos);
34             pos = next_pos + 1;
35         }
36     }
37
38     // 初始化答案变量, 指的是最长连续男生数量
39     int ans = 0;
40
41     // 四个检查矩阵, check[0]、check[1]、check[2]、check[3]
42     // 分别表示横向、纵向、对角线、反对角线方向上
43     // 某一个M是否已经在某段连续的M中被考虑过
44     vector<vector<vector<bool>>> check(4, vector<vector<bool>>(n, vector<bool>
(m, false)));
45
46     // 从上到下, 从左到右顺序枚举矩阵中的每一个元素
47     for (int i = 0; i < n; i++) {
48         for (int j = 0; j < m; j++) {
49             // 考虑横向、纵向、主对角线方向、反对角线方向4个方向
50             for (int k = 0; k < 4; k++) {
51                 // 如果某个M尚未被考虑包含在某段连续的M中
52                 // 则考虑这段M
53                 if (mat[i][j] == "M" && !check[k][i][j]) {
54                     // (x,y)为移动时的坐标
55                     // (dx,dy)为移动的坐标偏差
56                     int x = i, y = j;
57                     int dx = DIRECTIONS[k][0], dy = DIRECTIONS[k][1];

```

```

58 // 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
59 // 修改check[k][x][y]为已检查过, 且(x,y)修改
60 while (x >= 0 && x < n && y >= 0 && y < m && mat[x][y] ==
    "M") {
61     check[k][x][y] = true;
62     x += dx;
63     y += dy;
64 }
65 // 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
66 ans = max(ans, max(x - i, y - j));
67     }
68     }
69     }
70 }
71
72 // 输出答案
73 cout << ans << endl;
74
75 return 0;
76 }
77

```

## \*代码三：状态压缩写法

### Python

```

1 # 题目：【模拟】2024E-最大相连男生数
2 # 分值：200
3 # 作者：许老师-闭着眼睛学数理化
4 # 算法：模拟, 矩阵, 位运算, 状态压缩
5 # 代码看不懂的地方, 请直接在群上提问
6
7
8 # 四个方向的偏移变量
9 # DIRECTIONS[0]、DIRECTIONS[1]、DIRECTIONS[2]、DIRECTIONS[3]分别表示
10 # 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
11 DIRECTIONS = [(0, 1), (1, 0), (1, 1), (1, -1)]
12
13 # 输入矩阵的行数n, 列数m
14 n, m = map(int, input().split(", "))
15
16 # 初始化二维矩阵

```

```

17 mat = list()
18 # 循环n行，输入矩阵
19 for _ in range(n):
20     mat.append(input().split(","))
21
22
23 # 初始化答案变量，指的是最长连续男生数量
24 ans = 0
25
26 # 检查矩阵check，其中的数字的范围为0-15
27 # 每一个数字的二进制可以由4个数位来表示
28 # 从低位数起，第0-3位数位
29 # 分别表示横向、纵向、对角线方向、反对角线方向上
30 # 某一个M是否已经在某段连续的M中被考虑过
31 # 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
32 # 初始化均为0，表示都没有考虑过
33 check = [[0] * m for _ in range(n)]
34
35
36 # 从上到下，从左到右顺序枚举矩阵中的每一个元素
37 for i in range(n):
38     for j in range(m):
39         # 考虑横向、纵向、主对角线方向、反对角线方向4个方向
40         for k in range(4):
41             # 如果某个M尚未被考虑包含在某段连续的M中
42             # 则考虑这段M
43             # (1 >> k)可以得到1左移k位后的结果，
44             # 譬如考虑纵向时左移k = 1位得到 2 = bin(0010)
45             # 如果check[i][j]和(1 << k)进行与运算后得到0，
46             # 说明check[i][j]的二进制表示，从低位数起的第k位为0
47             # 第k位对应的方向没有被考虑
48             if mat[i][j] == "M" and (check[i][j] & (1 << k)) == 0:
49                 # (x,y)为移动时的坐标
50                 # (dx,dy)为横向移动的坐标偏差
51                 x, y = i, j
52                 dx, dy = DIRECTIONS[k]
53                 # 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
54                 # 修改check[k][x][y]为已检查过，且(x,y)修改
55                 while 0 <= x < n and 0 <= y < m and mat[x][y] == "M":
56                     # 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
57                     check[x][y] |= (1 << k)
58                     x += dx
59                     y += dy
60                 # 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
61         ans = max(ans, max(x-i, y-j))
62

```

## Java

```

1  import java.util.Scanner;
2
3  public class Main {
4
5      // 四个方向的偏移变量
6      // DIRECTIONS[0]、DIRECTIONS[1]、DIRECTIONS[2]、DIRECTIONS[3]分别表示
7      // 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
8      private static final int[][] DIRECTIONS = {
9          {0, 1},    // 横向向右
10         {1, 0},    // 纵向向下
11         {1, 1},    // 主对角线方向向右下
12         {1, -1}    // 反对角线方向向左下
13     };
14
15     public static void main(String[] args) {
16         Scanner scanner = new Scanner(System.in);
17
18         // 输入矩阵的行数n，列数m
19         String[] input = scanner.nextLine().split(",");
20         int n = Integer.parseInt(input[0]);
21         int m = Integer.parseInt(input[1]);
22
23         // 初始化二维矩阵
24         String[][] mat = new String[n][m];
25         // 循环n行，输入矩阵
26         for (int i = 0; i < n; i++) {
27             mat[i] = scanner.nextLine().split(",");
28         }
29
30         // 初始化答案变量，指的是最长连续男生数量
31         int ans = 0;
32
33         // 检查矩阵check，其中的数字的范围为0-15
34         // 每一个数字的二进制可以由4个数位来表示
35         // 从低位数起，第0-3位数位
36         // 分别表示横向、纵向、对角线方向、反对角线方向上
37         // 某一个M是否已经在某段连续的M中被考虑过
38         // 初始化均为0，表示都没有考虑过
39         int[][] check = new int[n][m];
40
41         // 从上到下，从左到右顺序枚举矩阵中的每一个元素

```

```

42     for (int i = 0; i < n; i++) {
43         for (int j = 0; j < m; j++) {
44             // 考虑横向、纵向、主对角线方向、反对角线方向4个方向
45             for (int k = 0; k < 4; k++) {
46                 // 如果某个M尚未被考虑包含在某段连续的M中
47                 // 则考虑这段M
48                 // (1 >> k)可以得到1左移k位后的结果,
49                 // 譬如考虑纵向时左移k = 1位得到 2 = bin(0010)
50                 // 如果check[i][j]和(1 << k)进行与运算后得到0,
51                 // 说明check[i][j]的二进制表示, 从低位数起的第k位为0
52                 // 第k位对应的方向没有被考虑
53                 if (mat[i][j].equals("M") && (check[i][j] & (1 << k)) ==
0) {
54                     // (x,y)为移动时的坐标
55                     // (dx,dy)为移动的坐标偏差
56                     int x = i, y = j;
57                     int dx = DIRECTIONS[k][0], dy = DIRECTIONS[k][1];
58                     // 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
59                     // 修改check[x][y]为已检查过, 且(x,y)修改
60                     while (x >= 0 && x < n && y >= 0 && y < m && mat[x]
[y].equals("M")) {
61                         // 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
62                         check[x][y] |= (1 << k);
63                         x += dx;
64                         y += dy;
65                     }
66                     // 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
67                     ans = Math.max(ans, Math.max(x - i, y - j));
68                 }
69             }
70         }
71     }
72
73     // 输出答案
74     System.out.println(ans);
75
76     scanner.close();
77 }
78 }
79

```

```

1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4
5  using namespace std;
6
7  // 四个方向的偏移变量
8  // DIRECTIONS[0]、DIRECTIONS[1]、DIRECTIONS[2]、DIRECTIONS[3]分别表示
9  // 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
10 const int DIRECTIONS[4][2] = {
11     {0, 1},    // 横向向右
12     {1, 0},    // 纵向向下
13     {1, 1},    // 主对角线方向向右下
14     {1, -1}    // 反对角线方向向左下
15 };
16
17 int main() {
18     // 输入矩阵的行数n，列数m
19     int n, m;
20     char comma;
21     cin >> n >> comma >> m;
22     cin.ignore(); // 忽略换行符
23
24     // 初始化二维矩阵
25     vector<vector<string>> mat(n, vector<string>(m));
26     // 循环n行，输入矩阵
27     for (int i = 0; i < n; i++) {
28         string line;
29         getline(cin, line);
30         size_t pos = 0;
31         for (int j = 0; j < m; j++) {
32             size_t next_pos = line.find(',', pos);
33             mat[i][j] = line.substr(pos, next_pos - pos);
34             pos = next_pos + 1;
35         }
36     }
37
38     // 初始化答案变量，指的是最长连续男生数量
39     int ans = 0;
40
41     // 检查矩阵check，其中的数字的范围为0-15
42     // 每一个数字的二进制可以由4个数位来表示
43     // 从低位数起，第0-3位数位
44     // 分别表示横向、纵向、对角线方向、反对角线方向上
45     // 某一个M是否已经在某段连续的M中被考虑过
46     // 初始化均为0，表示都没有考虑过
47     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  $O(NM)$ 。



