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，成对角线的或者呈反对角线的。

注：学生个数不会超过10000

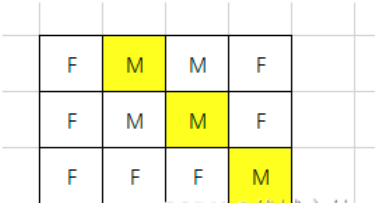
输入描述

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。

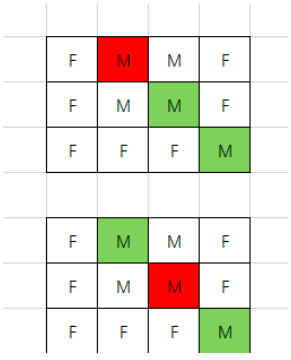
用例

输入	3,4 F,M,M,F F,M,M,F F,F,F,M
输出	3
说明	

题目解析

本题的解题思路其实不难，遍历查找矩阵中每一个M点，然后求该M点的水平、垂直、正对角线、反对角线，四个方向的M点个数，然后保留最大的个数，就是题解。

但是这种方法会存在很多重复的查找，比如

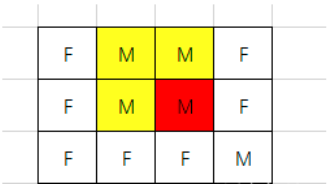


红色M是当前遍历到的M，绿色M是以红色M为原点查找到的M，如上图两个红色M点会重复查找同一条M链。

为了避免这种重复查找，我们可以增加判断：

如果当前M点的

- 左上角点是M，则反对角线不用查找了
- 右上角点是M，则正对角线不用查找了
- 上边点是M，则垂直线不用查找了
- 左边点是M，最水平线不用查找了



如上图红色M的左上、上、左点都是M，因此红色M的

- 反对角线已经被其左上点M查找过了，因此不用找了，
- 垂直线已经被其上边点M查找过了，因此不用找了
- 水平线已经被其左边点M查找过了，因此不用找了

Java算法源码

```java
import java.util.Scanner;

public class Main {
  static int n;
  static int m;
  static String[][] matrix;

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in).useDelimiter("[,\n]");

    n = sc.nextInt();
    m = sc.nextInt();

    matrix = new String[n][m];
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        matrix[i][j] = sc.next();
      }
    }

```

```
20
21      System.out.println(getResult());
22    }
23
24    public static int getResult() {
25      int ans = 0;
26
27      int[][] offsets = {{0, 1}, {1, 0}, {1, 1}, {1, -1}};
28
29      for (int i = 0; i < n; i++) {
30        for (int j = 0; j < m; j++) {
31          if ("M".equals(matrix[i][j])) {
32            for (int[] offset : offsets) {
33              int oldI = i - offset[0];
34              int oldJ = j - offset[1];
35
36              if (oldI >= 0 && oldI < n && oldJ >= 0 && oldJ < m && "M".equals(matrix[oldI][oldJ])) {
37                continue;
38              }
39
40              int len = 1;
41              int newI = i + offset[0];
42              int newJ = j + offset[1];
43
44              while (newI >= 0
45                  && newI < n
46                  && newJ >= 0
47                  && newJ < m
48                  && "M".equals(matrix[newI][newJ])) {
49                len++;
```

```
49                len++;
50                newI += offset[0];
51                newJ += offset[1];
52              }
53
54              ans = Math.max(ans, len);
55            }
56          }
57        }
58      }
59
60      return ans;
61    }
62  }
```

JS算法源码

```
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
let n, m;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    [n, m] = lines[0].split(",").map(Number);
  }

  if (n && lines.length === n + 1) {
    lines.shift();
    const matrix = lines.map((line) => line.split(","));

    console.log(getResult(matrix, n, m));

    lines.length = 0;
  }
});

function getResult(matrix, n, m) {
```

```
function getResult(matrix, n, m) {
  let ans = 0;

  const offsets = [
    [0, 1],
    [1, 0],
    [1, 1],
    [1, -1],
  ];

  for (let i = 0; i < n; i++) {
    for (let j = 0; j < m; j++) {
      if (matrix[i][j] == "M") {
        for (let offset of offsets) {
          const oldI = i - offset[0];
          const oldJ = j - offset[1];

          if (
            oldI >= 0 &&
            oldI < n &&
            oldJ >= 0 &&
            oldJ < m &&
            matrix[oldI][oldJ] == "M"
          ) {
            continue;
          }

          let len = 1;
          let newI = i + offset[0];
          let newJ = j + offset[1];
```

```
          let newJ = j + offset[1];

          while (
            newI >= 0 &&
            newI < n &&
            newJ >= 0 &&
            newJ < m &&
            matrix[newI][newJ] == "M"
          ) {
            len++;
            newI += offset[0];
            newJ += offset[1];
          }

          ans = Math.max(ans, len);
        }
      }
    }
  }

  return ans;
}
```

Python算法源码

```
# 输入获取
n, m = map(int, input().split(","))
matrix = [input().split(",") for _ in range(n)]


# 算法入口
def getResult():
    ans = 0

    offsets = ((0, 1), (1, 0), (1, 1), (1, -1))

    for i in range(n):
        for j in range(m):
            if matrix[i][j] == "M":
                for offset in offsets:
                    oldI = i - offset[0]
                    oldJ = j - offset[1]

                    if n > oldI >= 0 and m > oldJ >= 0 and matrix[oldI][oldJ] == "M":
                        continue

                    length = 1
                    newI = i + offset[0]
                    newJ = j + offset[1]

                    while n > newI >= 0 and m > newJ >= 0 and matrix[newI][newJ] == "M":
                        length += 1
                        newI += offset[0]
                        newJ += offset[1]

                    ans = max(ans, length)

    return ans


# 调用算法
print(getResult())
```