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、呈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 1

输入：

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

输出：

```
3
```

```
import java.util.Scanner;

public class Main {
	static boolean[][] used1;
	static boolean[][] used2;
	public static void main(String[] args) {
		Scanner scanner = new Scanner(System.in);
		String[] line1 = scanner.nextLine().split(",");
		int m = Integer.parseInt(line1[0]);
		int n = Integer.parseInt(line1[1]);
		String[][] array = new String[m][n];
		for (int i = 0; i < m; i++) {
			array[i] = scanner.nextLine().split(",");
		}
		int max = 0;
		for (int i = 0; i < m; i++) {
			max = Math.max(getMaxLen1(i, n, array), max);
		}
		for (int i = 0; i < n; i++) {
			max = Math.max(getMaxLen2(i, m, array), max);
		}
		used1 = new boolean[m][n];
```

```
        used2 = new boolean[m][n];
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (!array[i][j].equals("M")) continue;
                if (!used1[i][j]) max = Math.max(getMaxLen3(i, j, m, n, array), max);
                if (!used2[i][j]) max = Math.max(getMaxLen4(i, j, m, n, array), max);
            }
        }
        System.out.println(max);
    }
    //行最大
    static int getMaxLen1(int row, int n, String[][] array) {
        int max = 0;
        int i = 0;
        while (i < n && !array[row][i].equals("M")) {
            i++;
        }
        int l = i;
        if (l >= n) return 0;
        int r = i + 1;
        while (r < n) {
            if (!array[row][r].equals("M")) {
                max = Math.max(max, r - l);
                l = r + 1;
                if (l >= n) {
                    l = r;
                    break;
                }
                while (l < n && !array[row][i].equals("M")) {
                    l++;
                }
                if (l >= n) {
                    l = r;
                    break;
                }
                r = l + 1;
            } else {
                r++;
            }
        }
        max = Math.max(max, r - l);
        return max;
    }
```

```
//列最大
static int getMaxLen2(int col, int m, String[][] array) {
    int max = 0;
    int i = 0;
    while (i < m && !array[i][col].equals("M")) {
        i++;
    }
    int l = i;
    if (l >= m) return 0;
    int r = i + 1;
    while (r < m) {
        if (!array[r][col].equals("M")) {
            max = Math.max(max, r - l);
            l = r + 1;
            if (l >= m) {
                l = r;
                break;
            }
            while (l < m && !array[r][col].equals("M")) {
                l++;
            }
            r = l + 1;
        } else {
            r++;
        }
    }
    max = Math.max(max, r - l);
    return max;
}

//对角线最大
static int getMaxLen3(int i, int j, int m, int n, String[][] array) {
    int[] r = new int[]{i + 1, j + 1};
    while (r[0] < m && r[1] < n) {
        if (!array[r[0]][r[1]].equals("M")) return r[0] - i;
        used1[r[0]][r[1]] = true;
        r[0]++;
        r[1]++;
    }
    return r[0] - i;
}

//反对角线最大
static int getMaxLen4(int i, int j, int m, int n, String[][] array) {
```

```
            int[] r = new int[]{i + 1, j - 1};
            while (r[0] < m && r[1] >= 0) {
                if (!array[r[0]][r[1]].equals("M")) return r[0] - i;
                used2[r[0]][r[1]] = true;
                r[0]++;
                r[1]--;
            }
            return r[0] - i;
        }

    }
```