题目描述

贪吃蛇[1]是一个经典游戏，蛇的身体由若干方格连接而成，身体随蛇头移动。蛇头触碰到食物时，蛇的长度会增加一格。蛇头和身体的任一方格或者游戏版图边界碰撞时，游戏结束。

下面让我们来完成贪吃蛇游戏的模拟。

给定一个N*M的数组arr，代表N*M个方格组成的版图，贪吃蛇每次移动一个方格。

若arr[i][j] == 'H'，表示该方格为贪吃蛇的起始位置；

若arr[i][j] == 'F'，表示该方格为食物；

若arr[i][j] == 'E'，表示该方格为空格。

贪吃蛇初始长度为1，初始移动方向为向左。

为给定一系列贪吃蛇的移动操作，返回操作后蛇的长度，如果在操作执行完之前已经游戏结束，返回游戏结束时蛇的长度。

贪吃蛇移动、吃食物和碰撞处理的细节见下面图示：

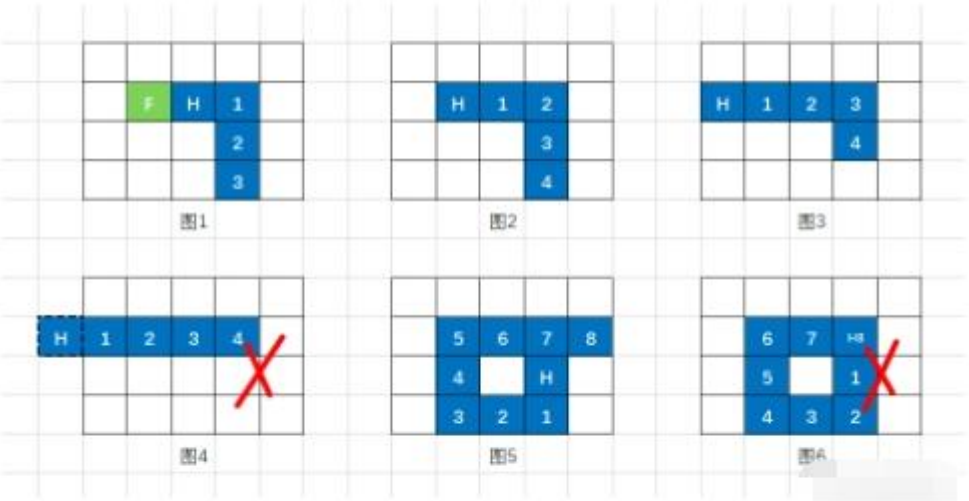

图1：截取了贪吃蛇移动的一个中间状态，H表示蛇头，F表示食物，数字为蛇身体各节的编号，蛇为向左移动，此时蛇头和食物已经相邻

图2：蛇头向左移动一格，蛇头和食物重叠，注意此时食物的格子成为了新的蛇头，第1节身体移动到蛇头位置，第2节身体移动到第1节身体位置，以此类推，最后添加第4节身体到原来第3节身体的位置。

图3：蛇头继续向左移动一格，身体的各节按上述规则移动，此时蛇头已经和边界相邻，但还未碰撞。

图4：蛇头继续向左移动一格，此时蛇头已经超过边界，发生碰撞，游戏结束。

图5和图6给出一个蛇头和身体碰撞的例子，蛇为向上移动。

图5时蛇头和第7节身体相邻，但还未碰撞；

图6蛇头向上移动一格，此时蛇头和第8节身体都移动到了原来第7节身体的位置，发生碰撞，游戏结束。

输入描述

输入第一行为空格分隔的字母，代表贪吃蛇的移动操作。

字母取值为U、D、L、R和G，

U、D、L、R分别表示贪吃蛇往上、下、左、右和转向，转向时贪吃蛇不移动，G表示贪吃蛇按当前的方向移动一格。

用例保证输入的操作正确。

第二行为空格分隔的两个数，指定N和M，为数组的行和列数。

余下N行每行是空格分隔的M个字母。字母取值为H、F和E，H表示贪吃蛇的起始位置，F表示食物，E表示该方格为空。

用例保证有且只有一个H，而F和E会有多个。

输出描述

输出一个数字，为蛇的长度。

用例

输入	D G G 3 3 F F F F F H E F E
输出	1
说明	地图表示为： • 蛇头 H(Head) • 食物 F(Food) • E表示该方格为空 四个方向分别表示为： • 向上 U(up) • 向下 D(down) • 向左 L(Left) • 向右 R(Right)

题目解析

纯逻辑题。

本题难点在于当贪吃蛇移动后，更新贪吃蛇的位置，以及矩阵各坐标的信息的逻辑。

首先，我使用一个数组snake来维护贪吃蛇的位置，蛇头就是snake[0]。

当贪吃蛇移动时，如果蛇头去往的位置是空地，即matrix[i][j] = 'E'的话，则

```
snake.unshift([i,j])
let [aI, aJ] = snake.pop() // 由于去往的是空地，因此贪吃蛇不会生长，所以蛇尾的位置要pop出去
matrix[aI][aJ] = 'E' // 并且要更新pop出去的位置为空地
matrix[i][j] = 'H' // 而蛇头达到的新位置要更新为蛇头位置H
```

当贪吃蛇移动时，如果蛇头去往的位置是食物，即matrix[i][j] = 'F'的话，则

```
snake.unshift([i,j])
matrix[i][j] = 'H' // 更新蛇头位置
```

当贪吃蛇移动时，如果蛇头去往的位置是自己的身体，即matrix[i][j] = 'H'，

> 注意我这里并不需要根据'H'来判断移动中蛇头的位置，而是总是用snake[0]作为蛇头，因此matrix[i][j] = 'H'可以直接用于标记贪吃蛇身体，来区别F、E。

则，此时游戏结束，输出snake.length

另外，当贪吃蛇移动的位置越界了，游戏也结束，输出snake.length

自测用例

```
DGLGGUGGRGGDGLG
3 3
FFF
FFH
EFE
```

最终贪吃蛇的长度为7

```
DGLGUGRGUGLGDG
3 3
FFF
FFH
EFE
```

最终贪吃蛇的长度为5

JavaScript算法源码

```javascript
/* 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 operates;
let n, m;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    operates = lines[0].split(" ");
    [n, m] = lines[1].split(" ").map(Number);
  }

  if (n && lines.length === n + 2) {
    lines.shift();
    lines.shift();
    const matrix = lines.map((line) => line.split(" "));

    console.log(getSnakeLen(matrix, n, m, operates));

    lines.length = 0;
  }
});
```

```
30
31 function getSnakeLen(matrix, n, m, operates) {
32   const snake = [];
33
34   // 找到初始蛇头位置，并使用snake[0]来维护蛇头位置
35   for (let i = 0; i < n; i++) {
36     for (let j = 0; j < m; j++) {
37       if (matrix[i][j] === "H") {
38         snake.push([i, j]);
39       }
40     }
41   }
42
43   // 蛇头移动方向
44   let direction = "L"; // 初始默认向左
45   for (let i = 0; i < operates.length; i++) {
46     if (operates[i] === "G") {
47       // 如果为G，则表示验证direction方向移动一格
48       let [i, j] = snake[0];
49       let res = go(matrix, i, j, snake, direction, n, m); // 具体移动逻辑
50
51       if (res) {
52         return res;
53       }
54     } else {
55       direction = operates[i];
56     }
57   }
58
59   return snake.length;
60 }
61
62 function go(matrix, i, j, snake, direction, n, m) {
63   // [r,c]是当前蛇头位置，[i,j]是上一次蛇头位置
```

```
  let r = i,
    c = j;
  switch (direction) {
    case "U":
      r--;
      break;
    case "D":
      r++;
      break;
    case "L":
      c--;
      break;
    case "R":
      c++;
      break;
  }

  if (r < 0 || r >= n || c < 0 || c >= m) {
    // 越界，游戏结束，返回贪吃蛇长度
    return snake.length;
  } else {
    if (matrix[r][c] === "E") {
      // 如果蛇头去的位置是空地
      matrix[r][c] = "H";
      snake.unshift([r, c]);
      let [aI, aJ] = snake.pop();
      matrix[aI][aJ] = "E";
    } else if (matrix[r][c] === "F") {
      // 如果蛇头去的位置是食物
      snake.unshift([r, c]);
      matrix[r][c] = "H";
    } else {
      // 吃到自己身体，游戏结束，返回贪吃蛇长度
      return snake.length;
    }
  }
}
```

Java算法源码

```
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Scanner;

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

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

    String[] operates = sc.nextLine().split(" ");

```

```
    Integer[] tmp =
        Arrays.stream(sc.nextLine().split(" ")).map(Integer::parseInt).toArray(Integer[]::new);
    n = tmp[0];
    m = tmp[1];

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

    System.out.println(getResult(operates));
  }

  public static int getResult(String[] operates) {
    LinkedList<Integer[]> snake = new LinkedList<>();

    // 找到初始蛇头位置，并使用snake[0]来维护蛇头位置
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        if ("H".equals(matrix[i][j])) {
          snake.addLast(new Integer[] {i, j});
        }
      }
    }

    // 蛇头移动方向
    String direction = "L"; // 初始默认向左

    for (String operate : operates) {
      if ("G".equals(operate)) {
        // 如果为G，则表示继续direction方向移动一格
```

```
48          Integer[] pos = snake.get(0);
49          int res = go(pos[0], pos[1], snake, direction); // 具体移动逻辑
50          if (res > 0) {
51            return res;
52          }
53        } else {
54          direction = operate;
55        }
56      }
57
58      return snake.size();
59    }
60
61    public static int go(int i, int j, LinkedList<Integer[]> snake, String direction) {
62      // [r,c]是当前蛇头位置，[i,j]是上一次蛇头位置
63      int r = i, c = j;
64
65      switch (direction) {
66        case "U":
67          r--;
68          break;
69        case "D":
70          r++;
71          break;
72        case "L":
73          c--;
74          break;
75        case "R":
76          c++;
77          break;
78      }
```

```
79
80      if (r < 0 || r >= n || c < 0 || c >= m) {
81        // 越界，游戏结束，返回贪吃蛇长度
82        return snake.size();
83      } else {
84        if ("E".equals(matrix[r][c])) {
85          // 如果蛇头去的位置是空地
86          matrix[r][c] = "H";
87          snake.addFirst(new Integer[] {r, c});
88          Integer[] tmp = snake.removeLast();
89          matrix[tmp[0]][tmp[1]] = "E";
90        } else if ("F".equals(matrix[r][c])) {
91          // 如果蛇头去的位置是食物
92          snake.addFirst(new Integer[] {r, c});
93          matrix[r][c] = "H";
94        } else {
95          // 吃到自己身体，游戏结束，返回贪吃蛇长度
96          return snake.size();
97        }
98      }
99
100     // 返回0表示继续下一步移动
101     return 0;
102   }
103 }
```

Python算法源码

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


def go(i, j, snake, direction):
    # [r,c]是当前蛇头位置，[i,j]是上一次蛇头位置
    r, c = i, j

    if "U" == direction:
        r -= 1
    elif "D" == direction:
        r += 1
    elif "L" == direction:
        c -= 1
    elif "R" == direction:
        c += 1

    if r < 0 or r >= n or c < 0 or c >= m:
        # 越界，游戏结束，返回食吃蛇长度
        return len(snake)
    else:
        if "E" == matrix[r][c]:
            # 如果蛇头去的位置是空地
            matrix[r][c] = "H"
            snake.insert(0, [r, c])
            x, y = snake.pop()
            matrix[x][y] = "E"
        elif "F" == matrix[r][c]:
            # 如果蛇头去的位置是食物
```

```
            snake.insert(0, [r, c])
            matrix[r][c] = "H"
        else:
            # 吃到自己身体，游戏结束，返回食吃蛇长度
            return len(snake)

    # 返回0表示继续下一步移动
    return 0


# 算法入口
def getResult():
    snake = []

    # 找到初始蛇头位置，并使用snake[0]来维护蛇头位置
    for i in range(n):
        for j in range(m):
            if "H" == matrix[i][j]:
                snake.append([i, j])

    # 蛇头移动方向,初始默认向左
    direction = "L"

    for operate in operates:
        # 如果为G，则表示继续direction方向移动一格
        if "G" == operate:
            x, y = snake[0]
            res = go(x, y, snake, direction)  # 具体移动逻辑
            if res > 0:
                return res
        else:
            direction = operate

    return len(snake)


# 算法调用
print(getResult())
```