

题目描述

找到它是一个小游戏，你需要在一个矩阵中找到给定的单词。

假设给定单词 **HELLOWORD**，在矩阵中只要能找到 H->E->L->L->O->W->O->R->L->D连成的单词，就算通过。

注意区分英文字母大小写，并且您只能上下左右行走，不能走回头路。

输入描述

输入第 1 行包含两个整数 n、m (0 < n,m < 21) 分别表示 n 行 m 列的矩阵，

第 2 行是长度不超过100的单词 W (在整个矩阵中给定单词 W 只会出现一次)，

从第 3 行到第 n+2 行是指包含大小写英文字母的长度为 m 的字符串矩阵。

输出描述

如果能在矩阵中连成给定的单词，则输出给定单词首字母在矩阵中的位置(第几行 第几列)，

否则输出“NO”。

用例

输入	5 5 HELLOWORLD CPUCY EKLQH CHELL LROWO DGRBC
输出	3 2
说明	无

输入	5 5 HELLOWORLD CPUCY EKLQH CHELL LROWO AGRBC
输出	NO
说明	无

题目解析

本题就是 

的变种题，具体解析请看上面的博客。

JavaScript算法源码

```
1  /* JavaScript Node ACN模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 let n, m;
11 let word;
12 rl.on("line", (line) => {
13   lines.push(line);
14
15   if (lines.length === 2) {
16     [n, m] = lines[0].split(" ").map(Number);
17     word = lines[1];
18   }
19
20   if (n && lines.length === n + 2) {
21     lines.shift();
22     lines.shift();
23
24     console.log(canFind(lines, n, m, word));
25
26     lines.length = 0;
27   }
28 });
29
30 function canFind(matrix, n, m, word) {
31   const len = word.length;
32   const visited = new Array(n).fill(0).map(() => new Array(m).fill(false));
33
34   const backTracking = (i, j, k) => {
35     if (k === len) return true;
36
```

```

37     if (
38         i < 0 ||
39         i >= n ||
40         j < 0 ||
41         j >= m ||
42         visited[i][j] ||
43         matrix[i][j] !== word[k]
44     )
45         return false;
46
47     visited[i][j] = true;
48
49     const newK = k + 1;
50     const res =
51         backTracking(i - 1, j, newK) ||
52         backTracking(i + 1, j, newK) ||
53         backTracking(i, j - 1, newK) ||
54         backTracking(i, j + 1, newK);
55
56     visited[i][j] = false;
57     return res;
58 }
59
60 for (let i = 0; i < n; i++) {
61     for (let j = 0; j < m; j++) {
62         if (backTracking(i, j, 0)) {
63             return `${i + 1} ${j + 1}`;
64         }
65     }
66 }
67
68 return "NO";
69 }

```

Java算法源码

```

1 import java.util.Scanner;
2
3 public class Main {
4     static String[] matrix;
5     static String word;
6     static int n;
7     static int m;
8     static boolean[][] visited;
9

```

```

10 public static void main(String[] args) {
11     Scanner sc = new Scanner(System.in);
12
13     n = sc.nextInt();
14     m = sc.nextInt();
15
16     word = sc.next();
17
18     matrix = new String[n];
19     for (int i = 0; i < n; i++) {
20         matrix[i] = sc.next();
21     }
22
23     System.out.println(getResult());
24 }
25
26 public static String getResult() {
27     visited = new boolean[n][m];
28
29     for (int i = 0; i < n; i++) {
30         for (int j = 0; j < m; j++) {
31             if (backTracking(i, j, 0)) {
32                 return (i + 1) + " " + (j + 1);
33             }
34         }
35     }
36
37     return "No";
38 }
39

```

```

40 public static boolean backTracking(int i, int j, int k) {
41     if (k == word.length()) return true;
42
43     if (i < 0
44         || i >= n
45         || j < 0
46         || j >= m
47         || visited[i][j]
48         || matrix[i].charAt(j) != word.charAt(k)) {
49         return false;
50     }
51
52     visited[i][j] = true;
53
54     int newK = k + 1;
55     boolean res =
56         backTracking(i - 1, j, newK)
57         || backTracking(i + 1, j, newK)
58         || backTracking(i, j - 1, newK)
59         || backTracking(i, j + 1, newK);
60
61     visited[i][j] = false;
62     return res;
63 }
64 }

```

Python算法源码

```
1 # 输入数据
2 n, m = map(int, input().split())
3 word = input()
4 matrix = [input() for _ in range(n)]
5 visited = [[False for _ in range(m)] for _ in range(n)]
6
7
8 def backTracking(i, j, k):
9     if k == len(word):
10         return True
11
12     if i < 0 or i >= n or j < 0 or j >= m or visited[i][j] or matrix[i][j] != word[k]:
13         return False
14
15     visited[i][j] = True
16
17     newK = k + 1
18     res = backTracking(i - 1, j, newK) or backTracking(i + 1, j, newK) or backTracking(i, j - 1, newK) or backTracking(i, j + 1, newK)
19     visited[i][j] = False
20     return res
21
22
23
24 # 算法入口
25 def getResult():
26     for i in range(n):
27         for j in range(m):
28             if backTracking(i, j, 0):
29                 return f"{i + 1} {j + 1}"
30
31     return "NO"
32
33
34 # 算法调用
35 print(getResult())
```