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

```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 n, m;
let word;
rl.on("line", (line) => {
  lines.push(line);

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

  if (n && lines.length === n + 2) {
    lines.shift();
    lines.shift();

    console.log(canFind(lines, n, m, word));

    lines.length = 0;
  }
});

function canFind(matrix, n, m, word) {
  const len = word.length;
  const visited = new Array(n).fill(0).map(() => new Array(m).fill(false));

  const backTracking = (i, j, k) => {
    if (k === len) return true;

```

```
    if (
      i < 0 ||
      i >= n ||
      j < 0 ||
      j >= m ||
      visited[i][j] ||
      matrix[i][j] !== word[k]
    )
      return false;

    visited[i][j] = true;

    const newK = k + 1;
    const res =
      backTracking(i - 1, j, newK) ||
      backTracking(i + 1, j, newK) ||
      backTracking(i, j - 1, newK) ||
      backTracking(i, j + 1, newK);

    visited[i][j] = false;
    return res;
  }

  for (let i = 0; i < n; i++) {
    for (let j = 0; j < m; j++) {
      if (backTracking(i, j, 0)) {
        return `${i + 1} ${j + 1}`;
      }
    }
  }

  return "NO";
}
```

Java算法源码

```
import java.util.Scanner;

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

```

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

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

    word = sc.next();

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

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

  public static String getResult() {
    visited = new boolean[n][m];

    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        if (backTracking(i, j, 0)) {
          return (i + 1) + " " + (j + 1);
        }
      }
    }

    return "NO";
  }

```

```
  public static boolean backTracking(int i, int j, int k) {
    if (k == word.length()) return true;

    if (i < 0
        || i >= n
        || j < 0
        || j >= m
        || visited[i][j]
        || matrix[i].charAt(j) != word.charAt(k)) {
      return false;
    }

    visited[i][j] = true;

    int newK = k + 1;
    boolean res =
        backTracking(i - 1, j, newK)
            || backTracking(i + 1, j, newK)
            || backTracking(i, j - 1, newK)
            || backTracking(i, j + 1, newK);

    visited[i][j] = false;
    return res;
  }
}
```

Python算法源码

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


def backTracking(i, j, k):
    if k == len(word):
        return True

    if i < 0 or i >= n or j < 0 or j >= m or visited[i][j] or matrix[i][j] != word[k]:
        return False

    visited[i][j] = True

    newK = k + 1
    res = backTracking(i - 1, j, newK) or backTracking(i + 1, j, newK) or backTracking(i, j - 1, newK) or backTracking
        i, j + 1, newK)
    visited[i][j] = False
    return res


# 算法入口
def getResult():
    for i in range(n):
        for j in range(m):
            if backTracking(i, j, 0):
                return f"{i + 1} {j + 1}"

    return "NO"


# 算法调用
print(getResult())
```