


## 54、找单词，考点 or 实现——深度优先搜索 DFS

### 题目描述

给一个字符串和一个 **二维字符数组** ，如果该字符串存在于该数组中，则按字符串的字符顺序输出字符串每个字符所在单元格的位置下标字符串，如果找不到返回字符串"N"。

- 1.需要按照字符串的字符组成顺序搜索，且搜索到的位置必须是相邻单元格，其中"相邻单元格"是指那些水平相邻或垂直相邻的单元格。
- 2.同一个单元格内的字母不允许被重复使用。
- 3.假定在数组中最多只存在一个可能的匹配。

### 输入描述

第1行为一个数字N指示二维数组在后续输入所占的行数。

第2行到第N+1行输入为一个二维大写字母数组，每行字符用半角,分割。

第N+2行为待查找的字符串，由大写字母组成。

二维数组的大小为N\*N， $0 < N \leq 100$ 。

**单词长度**  K， $0 < K < 1000$ 。

### 输出描述

输出一个位置下标字符串，拼接格式为：第1个字符行下标+","+第1个字符列下标+","+第2个字符行下标+","+第2个字符列下标... +","+第N个字符行下标+","+第N个字符列下标。

### 用例

输入	4 A,C,C,F C,D,E,D B,E,S,S F,E,C,A ACCESS
输出	0,0,0,1,0,2,1,2,2,2,3
说明	ACCESS分别对应二维数组的[0,0] [0,1] [0,2] [1,2] [2,2] [2,3]下标位置。

### 题目解析

本题和[华为机试 - 单词搜索\\_伏城之外的博客-CSDN博客](#)

很像。只是本题在上面这题的基础上，要求保存路径坐标。

题目假定在数组中最多只存在一个可能的匹配。因此我们只要找到即返回。

## JavaScript算法源码

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const path = require("path");
3  const readline = require("readline");
4
5  const rl = readline.createInterface({
6    input: process.stdin,
7    output: process.stdout,
8  });
9
10 const lines = [];
11 let n;
12 rl.on("line", (line) => {
13   lines.push(line);
14
15   if (lines.length === 1) {
16     n = parseInt(lines[0]);
17   }
18
19   if (n && lines.length === n + 2) {
20     lines.shift();
21     const str = lines.pop();
22
23     const grid = lines.map((line) => line.split(","));
24
25     console.log(findLetter(grid, n, str));
26
27     lines.length = 0;
28   }
29 });
30
31 function findLetter(grid, n, str) {
32   function dfs(i, j, k, path) {
33     if (i < 0 || i >= n || j < 0 || j >= n || grid[i][j] !== str[k])
34       return false;
35
36     path.push([i, j]);
37     if (path.length === str.length) return true;
38
39     const tmp = grid[i][j];
40     grid[i][j] = undefined;
41
42     const res =
43       dfs(i - 1, j, k + 1, path) ||
44       dfs(i + 1, j, k + 1, path) ||
45       dfs(i, j - 1, k + 1, path) ||
46       dfs(i, j + 1, k + 1, path);
47
48     if (!res) {
49       grid[i][j] = tmp;
50       path.pop();
51     }
52
53     return res;
54   }
```

```

55
56     for (let i = 0; i < n; i++) {
57         for (let j = 0; j < n; j++) {
58             const path = [];
59             if (dfs(i, j, 0, path)) {
60                 return path.toString();
61             }
62         }
63     }
64
65     return "N";
66 }

```

## Java算法源码

```

1  import java.util.LinkedList;
2  import java.util.Scanner;
3  import java.util.StringJoiner;
4
5  public class Main {
6      static int n;
7      static String[][] matrix;
8      static String tar;
9
10     public static void main(String[] args) {
11         // 将输入分隔符改为","和换行
12         Scanner sc = new Scanner(System.in).useDelimiter("[,\\n]");
13
14         n = sc.nextInt();
15
16         matrix = new String[n][n];
17         for (int i = 0; i < n; i++) {
18             for (int j = 0; j < n; j++) {
19                 matrix[i][j] = sc.next();
20             }
21         }
22
23         tar = sc.next();
24
25         System.out.println(getResult());
26     }

```

```

27
28 public static String getResult() {
29     for (int i = 0; i < n; i++) {
30         for (int j = 0; j < n; j++) {
31             LinkedList<Integer[]> path = new LinkedList<>();
32             if (dfs(i, j, 0, path)) {
33                 StringJoiner sj = new StringJoiner(",");
34                 for (Integer[] pos : path) sj.add(pos[0] + "," + pos[1]);
35                 return sj.toString();
36             }
37         }
38     }
39     return "N";
40 }
41
42 public static boolean dfs(int i, int j, int k, LinkedList<Integer[]> path) {
43     if (i < 0 || i >= n || j < 0 || j >= n || !tar.substring(k, k + 1).equals(matrix[i][j])) {
44         return false;
45     }
46
47     path.add(new Integer[] {i, j});
48     if (path.size() == tar.length()) return true;
49
50     String tmp = matrix[i][j];
51     matrix[i][j] = null;
52
53     boolean res =

```

```

54         dfs(i - 1, j, k + 1, path)
55         || dfs(i + 1, j, k + 1, path)
56         || dfs(i, j - 1, k + 1, path)
57         || dfs(i, j + 1, k + 1, path);
58
59     if (!res) {
60         matrix[i][j] = tmp;
61         path.removeLast();
62     }
63
64     return res;
65 }
66 }

```

## Python算法源码

```
1 # 输入数据
2 n = int(input())
3 matrix = [input().split(",") for _ in range(n)]
4 tar = input()
5
6
7 def dfs(i, j, k, path):
8     if i < 0 or i >= n or j < 0 or j >= n or matrix[i][j] != tar[k]:
9         return False
10
11     path.append([i, j])
12     if len(path) == len(tar):
13         return True
14
15     tmp = matrix[i][j]
16     matrix[i][j] = ""
17
18     res = dfs(i - 1, j, k + 1, path) or dfs(i + 1, j, k + 1, path) or dfs(i, j - 1, k + 1, path) or dfs(i, j + 1, k +
19
20     if not res:
21         matrix[i][j] = tmp
22         path.pop()
23
24     return res
25
26
27 # 算法入口
28 def getResult():
29     for i in range(n):
30         for j in range(n):
31             path = []
32             if dfs(i, j, 0, path):
33                 return ",".join(map(lambda x: ",".join(map(str, x)), path))
34
35     return "N"
36
37
38 # 算法调用
39 print(getResult())
```