找单词

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：

1.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。

2.第 2 行到第 N+1 行输入为一个二维大写字符数组，每行字符用半角,分割。

3.第 N+2 行为待查找的字符串，由大写字符组成。

4.二维数组的大小为 N*N，0<N<=100。

5.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

补充说明：

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：

1.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。

2.第 2 行到第 N+1 行输入为一个二维大写字符数组，每行字符用半角,分割。

3.第 N+2 行为待查找的字符串，由大写字符组成。

4.二维数组的大小为 N*N，0<N<=100。

5.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

补充说明：

```
import java.util.*;

public class Main {
    // 本题为考试单行多行输入输出规范示例，无需提交，不计分。
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int N  = in.nextInt();
        in.nextLine();
        char[][] mat = new char[N][N];
        for (int i = 0; i < N; i++) {
            String[] arr = in.nextLine().split(",");
            for (int j = 0; j < arr.length; j++) {
                mat[i][j] = arr[j].charAt(0);
            }
        }
        String tarStr = in.nextLine();
        char[] tarArr = tarStr.toCharArray();
        boolean[][] visitedFlag = new boolean[N][tarArr.length]; // 标识
        int[][] positions = new int[tarArr.length][2]; // 存储
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                if(mat[i][j] == tarStr.charAt(0)){
                    boolean find = backtrack(mat, tarArr, visitedFlag, positions, i,
j, 0);
                    if(find){
                        putoutRes(positions);
                        return;
                    }
                }
            }

        }
        System.out.println("N");
    }
    public static boolean backtrack(char[][] mat, char[] chars, boolean[][] visited,
int[][] positions, int i, int j, int k){
        if(k == chars.length){
            return true;
        }
```

```
        if(i < 0 || i >= mat.length || j < 0 || j >= mat[0].length || visited[i][k] ||
mat[i][j] != chars[k]){
            return false;
        }
        visited[i][k] = true;
        positions[k][0] = i;
        positions[k][1] = j;
        boolean findFlag = false;
        // 上
        findFlag = backtrack(mat, chars, visited, positions, i, j+1, k+1);
        if(findFlag)    return true;
        // 下
        findFlag = backtrack(mat, chars, visited, positions, i, j-1, k+1);
        if(findFlag)    return true;
        // 左
        findFlag = backtrack(mat, chars, visited, positions, i-1, j, k+1);
        if(findFlag)    return true;
        // 右
        findFlag = backtrack(mat, chars, visited, positions, i+1, j, k+1);
        if(findFlag)    return true;
        visited[i][k] = false;
        return false;
    }

    public static void putoutRes(int[][] positions){
        StringBuilder sb = new StringBuilder();
        for (int k = 0; k < positions.length; k++) {

sb.append(positions[k][0]).append(",").append(positions[k][1]).append(",");
        }
        sb.deleteCharAt(sb.length() - 1);
        System.out.println(sb.toString());
    }
    static class Position{
        public int x;
        public int y;
        public Position(int i, int i1) {
            x = i;
            y = i1;
        }
    }

}
```