

28、字符匹配，考点 or 实现——动态规划

题目描述

给你一个 **字符串数组** ^Q（每个字符串均由小写字母组成）和一个字符规律（由小写字母和 `.` 和 `*` 组成），识别数组中哪些字符串可以匹配到字符规律上。

`'.'` 匹配任意单个字符，`'*'` 匹配零个或多个前面的那一个元素，所谓匹配，是要涵盖整个字符串的，而不是部分字符串。

输入描述

第一行为空格分割的多个字符串， $1 < \text{单个字符串长度} < 100$ ， $0 < 1 < \text{字符串个数} < 100$

第二行为字符规律， $1 < \text{字符串个数} < 100$

第二行为字符规律， $1 \leq \text{字符规律长度} \leq 50$

不需要考虑异常场景。

输出描述

匹配的字符串在数组中的下标（从0开始），多个匹配时下标升序并用分割，若均不匹配输出-1

用例

输入	ab aab .*
输出	0,1
说明	无

输入	ab abc bsd .*
输出	0,1,2
说明	无

输入	avd adb sss as adb
输出	1
说明	无

题目解析

本题最简单的解题策略是套皮正则表达式，但是根据考友实际机考反馈，套皮正则只能拿30%左右通过率，剩余用例超时。

本题的最佳题解策略是 **动态规划** ^Q，即基于动态规实现模拟正则匹配。

具体解析请看我写的这篇博客：[LeetCode - 10 正则表达式匹配_伏城之外](#)

动态规划解法

Java算法源码

```
1 import java.util.Scanner;
2 import java.util.StringJoiner;
3
4 public class Main {
5     // 输入获取
6     public static void main(String[] args) {
7         Scanner sc = new Scanner(System.in);
8
9         String[] arr = sc.nextLine().split(" ");
10        String reg = sc.nextLine();
11
12        System.out.println(getResult(arr, reg));
13    }
14
15    // 算法入口
16    public static String getResult(String[] arr, String reg) {
17        StringJoiner sj = new StringJoiner(",");
18
19        for (int i = 0; i < arr.length; i++) {
20            if (isMatch(arr[i], reg)) sj.add(i + "");
21        }
22
23        if (sj.length() == 0) return "-1";
24        else return sj.toString();
25    }
```

```
26
27 public static boolean isMatch(String s, String p) {
28     s = " " + s;
29     p = " " + p;
30
31     int n = s.length();
32     int m = p.length();
33
34     boolean[][] dp = new boolean[n][m];
35     dp[0][0] = true;
36
37     for (int i = 0; i < n; i++) {
38         // 内层循环遍历的是正则p的范围，如果j=0，那么代表正则为空，此时匹配结果必然为false，而dp数组初始化时所有元素都初始化为false
39         for (int j = 1; j < m; j++) {
40             if (p.charAt(j) == '*') {
41                 dp[i][j] =
42                     (j >= 2 && dp[i][j - 2])
43                     || (eq(p.charAt(j - 1), s.charAt(i)) && i >= 1 && dp[i - 1][j]);
44             } else {
45                 dp[i][j] = eq(p.charAt(j), s.charAt(i)) && i >= 1 && dp[i - 1][j - 1];
46             }
47         }
48     }
49
50     return dp[n - 1][m - 1];
51 }
52
53 public static boolean eq(char p, char s) {
54     return p == s || p == '.';
55 }
56 }
```

JavaScript算法源码

```
1  /* JavaScript Node ACM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 rl.on("line", (line) => {
11   lines.push(line);
12
13   if (lines.length === 2) {
14     const arr = lines[0].split(" ");
15     const reg = lines[1];
16
17     console.log(getResutlt(arr, reg));
18     lines.length = 0;
19   }
20 });
21
22 function getResutlt(arr, reg) {
23   const ans = [];
24   for (let i = 0; i < arr.length; i++) {
25     if (isMatch(arr[i], reg)) ans.push(i);
26   }
27
28   if (ans.length == 0) return -1;
```

```

29     else return ans.join(",");
30 }
31
32 /**
33  * @param {string} s
34  * @param {string} p
35  * @return {boolean}
36  */
37 var isMatch = function (s, p) {
38     s = " " + s;
39     p = " " + p;
40
41     const n = s.length;
42     const m = p.length;
43
44     const dp = new Array(n).fill(0).map(() => new Array(m).fill(false));
45     dp[0][0] = true;
46
47     for (let i = 0; i < n; i++) {
48         // 内层循环遍历的是正则p的范围，如果j=0.那么代表正则为空，此时匹配结果必然为false，而dp数组初始化时所有元素都初始化为false
49         for (let j = 1; j < m; j++) {
50             if (p[j] == "**") {
51                 dp[i][j] =
52                     (j >= 2 && dp[i][j - 2]) ||
53                     (eq(p[j - 1], s[i]) && i >= 1 && dp[i - 1][j]);
54             } else {
55                 dp[i][j] = eq(p[j], s[i]) && i >= 1 && dp[i - 1][j - 1];
56             }
57         }
58     }
59
60     return dp[n - 1][m - 1];
61 };
62
63 function eq(p, s) {
64     return p == s || p == ".";
65 }

```

Python算法源码

```
1  # 输入获取
2  arr = input().split()
3  reg = input()
4
5
6  def eq(p, s):
7      return p == s or p == '.'
8
9
10 def isMatch(s, p):
11     s = " " + s
12     p = " " + p
13
14     n = len(s)
15     m = len(p)
16
17     dp = [[False] * m for _ in range(n)]
18     dp[0][0] = True
19
20     for i in range(n):
21         # 内层循环遍历的是正则p的范围, 如果j=0, 那么代表正则为空, 此时匹配结果必然为false, 而dp数组初始化时所有元素都初始化为false
22         for j in range(1, m):
23             if p[j] == '*':
24                 dp[i][j] = (j >= 2 and dp[i][j - 2]) or (eq(p[j - 1], s[i]) and i >= 1 and dp[i - 1][j])
25             else:
26                 dp[i][j] = eq(p[j], s[i]) and i >= 1 and dp[i - 1][j - 1]
27
28     return dp[n - 1][m - 1]
29
30
31 # 算法入口
32 def getResult():
33     ans = []
34     for i in range(len(arr)):
35         if isMatch(arr[i], reg):
36             ans.append(i)
37
38     if len(ans) == 0:
39         return "-1"
40     else:
41         return ",".join(map(str, ans))
42
43
44 # 算法调用
45 print(getResult())
```

正则解法

JavaScript算法源码

```
1  /* JavaScript Node ACM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 rl.on("line", (line) => {
11   lines.push(line);
12
13   if (lines.length === 2) {
14     const arr = lines[0].split(" ");
15     const reg = lines[1];
16
17     console.log(getResult(arr, reg));
18     lines.length = 0;
19   }
20 });
21
22 function getResult(arr, reg) {
23   const regExp = new RegExp(`^${reg}$`);
24
25   const ans = [];
26   for (let i = 0; i < arr.length; i++) {
27     if (regExp.test(arr[i])) ans.push(i);
28   }
29
30   if (ans.length == 0) return "-1";
31   else return ans.join(",");
32 }
```

Java算法源码

```
1 import java.util.Scanner;
2 import java.util.StringJoiner;
3 import java.util.regex.Pattern;
4
5 public class Main {
6     // 输入获取
7     public static void main(String[] args) {
8         Scanner sc = new Scanner(System.in);
9
10        String[] arr = sc.nextLine().split(" ");
11        String reg = sc.nextLine();
12
13        System.out.println(getResult(arr, reg));
14    }
15
16    // 算法入口
17    public static String getResult(String[] arr, String reg) {
18        Pattern compile = Pattern.compile("^" + reg + "$");
19
20        StringJoiner sj = new StringJoiner(",");
21        for (int i = 0; i < arr.length; i++) {
22            if (compile.matcher(arr[i]).matches()) {
23                sj.add(i + "");
24            }
25        }
26
27        if (sj.length() == 0) return "-1";
```

```
27         if (sj.length() == 0) return "-1";
28         else return sj.toString();
29     }
30 }
```


Python算法源码

```
1 import re
2
3 # 输入获取
4 arr = input().split()
5 reg = input()
6
7
8 # 算法入口
9 def getResult():
10     c = re.compile(f"^{reg}$")
11
12     ans = []
13     for i in range(len(arr)):
14         if c.match(arr[i]):
15             ans.append(i)
16
17     if len(ans) == 0:
18         return "-1"
19     else:
20         return ",".join(map(str, ans))
21
22
23 # 算法调用
24 print(getResult())
```