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（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<单个字符串长度<100，0，1<字符串个数<100

第二行为字符规律，1<字符串个数<100

第二行为字符规律，1<=字符规律长度<=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，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

```java
import java.util.Scanner;
import java.util.StringJoiner;

public class Main {
  // 输入获取
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    String[] arr = sc.nextLine().split(" ");
    String reg = sc.nextLine();

    System.out.println(getResult(arr, reg));
  }

  // 算法入口
  public static String getResult(String[] arr, String reg) {
    StringJoiner sj = new StringJoiner(",");

    for (int i = 0; i < arr.length; i++) {
      if (isMatch(arr[i], reg)) sj.add(i + "");
    }

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

```java

  public static boolean isMatch(String s, String p) {
    s = " " + s;
    p = " " + p;

    int n = s.length();
    int m = p.length();

    boolean[][] dp = new boolean[n][m];
    dp[0][0] = true;

    for (int i = 0; i < n; i++) {
      // 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了fa
      for (int j = 1; j < m; j++) {
        if (p.charAt(j) == '*') {
          dp[i][j] =
              (j >= 2 && dp[i][j - 2])
                  || (eq(p.charAt(j - 1), s.charAt(i)) && i >= 1 && dp[i - 1][j]);
        } else {
          dp[i][j] = eq(p.charAt(j), s.charAt(i)) && i >= 1 && dp[i - 1][j - 1];
        }
      }
    }

    return dp[n - 1][m - 1];
  }

  public static boolean eq(char p, char s) {
    return p == s || p == '.';
  }
}
```

JavaScript算法源码

```
/* 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 = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    const arr = lines[0].split(" ");
    const reg = lines[1];

    console.log(getResutlt(arr, reg));
    lines.length = 0;
  }
});

function getResutlt(arr, reg) {
  const ans = [];
  for (let i = 0; i < arr.length; i++) {
    if (isMatch(arr[i], reg)) ans.push(i);
  }

  if (ans.length == 0) return -1;
```

```
    else return ans.join(",");
};

/**
 * @param {string} s
 * @param {string} p
 * @return {boolean}
 */
var isMatch = function (s, p) {
    s = " " + s;
    p = " " + p;

    const n = s.length;
    const m = p.length;

    const dp = new Array(n).fill(0).map(() => new Array(m).fill(false));
    dp[0][0] = true;

  for (let i = 0; i < n; i++) {
    // 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了fals
    for (let j = 1; j < m; j++) {
      if (p[j] == "*") {
        dp[i][j] =
          (j >= 2 && dp[i][j - 2]) ||
          (eq(p[j - 1], s[i]) && i >= 1 && dp[i - 1][j]);
      } else {
        dp[i][j] = eq(p[j], s[i]) && i >= 1 && dp[i - 1][j - 1];
      }
    }
  }

  return dp[n - 1][m - 1];
};

function eq(p, s) {
  return p == s || p == ".";
}
```

Python算法源码

```python
# 输入获取
arr = input().split()
reg = input()


def eq(p, s):
    return p == s or p == '.'


def isMatch(s, p):
    s = " " + s
    p = " " + p

    n = len(s)
    m = len(p)

    dp = [[False] * m for _ in range(n)]
    dp[0][0] = True

    for i in range(n):
        # 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了f
        for j in range(1, m):
            if p[j] == '*':
                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])
            else:
                dp[i][j] = eq(p[j], s[i]) and i >= 1 and dp[i - 1][j - 1]

    return dp[n - 1][m - 1]


# 算法入口
def getResult():
    ans = []
    for i in range(len(arr)):
        if isMatch(arr[i], reg):
            ans.append(i)

    if len(ans) == 0:
        return "-1"
    else:
        return ",".join(map(str, ans))


# 算法调用
print(getResult())
```

正则解法

JavaScript算法源码

```
/* 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 = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    const arr = lines[0].split(" ");
    const reg = lines[1];

    console.log(getResutlt(arr, reg));
    lines.length = 0;
  }
});

function getResutlt(arr, reg) {
  const regExp = new RegExp(`^${reg}$`);

  const ans = [];
  for (let i = 0; i < arr.length; i++) {
    if (regExp.test(arr[i])) ans.push(i);
  }

  if (ans.length == 0) return "-1";
  else return ans.join(",");
}
```

Java算法源码

```java
import java.util.Scanner;
import java.util.StringJoiner;
import java.util.regex.Pattern;

public class Main {
  // 输入获取
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    String[] arr = sc.nextLine().split(" ");
    String reg = sc.nextLine();

    System.out.println(getResult(arr, reg));
  }

  // 算法入口
  public static String getResult(String[] arr, String reg) {
    Pattern compile = Pattern.compile("^" + reg + "$");

    StringJoiner sj = new StringJoiner(",");
    for (int i = 0; i < arr.length; i++) {
      if (compile.matcher(arr[i]).matches()) {
        sj.add(i + "");
      }
    }

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

Python算法源码

```python
import re

# 输入获取
arr = input().split()
reg = input()


# 算法入口
def getResult():
    c = re.compile(f"^{reg}$")

    ans = []
    for i in range(len(arr)):
        if c.match(arr[i]):
            ans.append(i)

    if len(ans) == 0:
        return "-1"
    else:
        return ",".join(map(str, ans))


# 算法调用
print(getResult())
```