30 正则表达式匹配

更新时间：2019-09-18 10:38:49

> 人的差异在于业余时间。
>
> ——爱因斯坦

刷题内容

> **难度: Hard**
>
> 原题连接：https://leetcode-cn.com/problems/regular-expression-matching/

内容描述

给定一个字符串 (s) 和一个字符模式 (p)。实现支持 '.' 和 '*' 的正则表达式匹配。

'.' 匹配任意单个字符。
'*' 匹配零个或多个前面的元素。
匹配应该覆盖整个字符串 (s) ，而不是部分字符串。

说明:

s 可能为空，且只包含从 a-z 的小写字母。
p 可能为空，且只包含从 a-z 的小写字母，以及字符 . 和 *。
示例 1:

输入:
s = "aa"
p = "a"
输出: false
解释: "a" 无法匹配 "aa" 整个字符串。
示例 2:

输入:
s = "aa"
p = "a*"
输出: true
解释: '*' 代表可匹配零个或多个前面的元素, 即可以匹配 'a' 。因此, 重复 'a' 一次, 字符串可变为 "aa"。
示例 3:

输入:
s = "ab"
p = ".*"
输出: true
解释: ".*" 表示可匹配零个或多个('*')任意字符('.')。
示例 4:

输入:
s = "aab"
p = "c*a*b"
输出: true
解释: 'c' 可以不被重复, 'a' 可以被重复一次。因此可以匹配字符串 "aab"。
示例 5:

输入:
s = "mississippi"
p = "mis*is*p*."
输出: false

题目详解

- 这道题很有意思，基本相当于实现一个小小的正则表达式了
- p就是我们的一个匹配模式，然后看看我们的s是否能够匹配上这个模式
- 至于解释匹配问题，看完上面所有的示例就全都可以明白了

解题方案

思路 1:时间复杂度: O(2^len§)

我们从左到右匹配，模式串p有四种情况：（以下用[a]表示单个字符）

- [a]：这种情况直接匹配s的下一个字符，如果不匹配，退出
- .：匹配下一个字符所有的情况
- .*：这种情况又有两种分情况（任一种情况能匹配就算匹配成功）：
  - 匹配下一个字符，s跳过一个字符，p仍然在.*这里

- 不匹配下个字符，s不变，p跳过.*

- [a]*：判断s的下一个字符是不是[a]

  - 如果是，同.*的处理方式
  - 如果不是，s不变，p跳过[a]*

由于第三种和第四种情况都有多个分支，需要回溯，因此我们采用递归的写法

Python beats 72.07%

```python
class Solution:
    def isMatch(self, s: str, p: str) -> bool:
        def helper(s, i, p, j):
            if j == -1: # p匹配完了
                return i == -1 # 如果此时s也匹配完了就说明可以匹配
            if i == -1:
                if p[j] != '*':
                    return False
                return helper(s, i, p, j-2)
            if p[j] == '*':
                if p[j-1] == '.' or p[j-1] == s[i]:
                    if helper(s, i-1, p, j):
                        return True
                return helper(s, i, p, j-2)
            if p[j] == '.' or p[j] == s[i]:
                return helper(s, i-1, p, j-1)
            return False

        return helper(s, len(s)-1, p, len(p)-1)
```

Java beats 65.39%

```java
class Solution {
    /**
     * @param s
     * @param p
     * @param i 表示 s 现在的位置
     * @param j 表示 p 现在的位置
     * @return 该函数返回 s[0..i] 能否和 p[0..i] 匹配
     */
    public boolean isMatch(String s, String p, int i, int j) {
        if (i == -1 && j == -1) {
            return true;
        }
        if (i == -1) {
            if (p.charAt(j) == '*') {
                return isMatch(s, p, i, j - 2);
            } else {
                return false;
            }
        }
        if (j == -1) {
            return false;
        }
        // 处理 .*
        if (j > 0 && p.charAt(j - 1) == '.' && p.charAt(j) == '*') {
            return isMatch(s, p, i - 1, j) || isMatch(s, p, i, j - 2);
        }
        // 处理 a*
        if (j > 0 && p.charAt(j) == '*') {
            if (s.charAt(i) == p.charAt(j - 1)) {
                return isMatch(s, p, i - 1, j) || isMatch(s, p, i, j - 2);
            } else {
                return isMatch(s, p, i, j - 2);
            }
        }
        // 处理.
        if (p.charAt(j) == '.') {
            return isMatch(s, p, i - 1, j - 1);
        }
        // 处理a
        if (p.charAt(j) == s.charAt(i)) {
            return isMatch(s, p, i - 1, j - 1);
        } else {
            return false;
        }
    }

    public boolean isMatch(String s, String p) {
        return isMatch(s, p, s.length() - 1, p.length() - 1);
    }
}
```

Go beats 100%

```go
func helper(s string, i int, p string, j int) bool {
    if j == -1 { // p匹配完了
        return i == -1 // 如果此时s也匹配完了就说明可以匹配
    }
    if i == -1 {
        if p[j] != '*' {
            return false
        }
        return helper(s, i, p, j-2)
    }
    if p[j] == '*' {
        if p[j-1] == '.' || p[j-1] == s[i] {
            if helper(s, i-1, p, j) {
                return true
            }
        }
        return helper(s, i, p, j-2)
    }
    if p[j] == '.' || p[j] == s[i] {
        return helper(s, i-1, p, j-1)
    }
    return false
}

func isMatch(s string, p string) bool {
    return helper(s, len(s)-1, p, len(p)-1)
}
```

c++ beats 22.29%

```
class Solution {
public:
    //i表示s现在的位置
    //j表示p现在的位置
    //该函数返回s[0..i]是否能跟p[0..j]匹配
    bool isMatch(string s, string p, int i, int j) {
        if (i == -1 && j == -1) {
            return true;
        }
        if (i == -1) {
            if (p[j] == '*') {
                return isMatch(s, p, i, j - 2);
            } else {
                return false;
            }
        }
        if (j == -1) {
            return false;
        }
        //处理.*
        if (j > 0 && p[j - 1] == '.' && p[j] == '*') {
            return isMatch(s, p, i - 1, j) || isMatch(s, p, i, j - 2);
        }
        //处理a*
        if (j > 0 && p[j] == '*') {
            if (s[i] == p[j - 1]) {
                return isMatch(s, p, i - 1, j) || isMatch(s, p, i, j - 2);
            } else {
                return isMatch(s, p, i, j - 2);
            }
        }
        //处理.
        if (p[j] == '.') {
            return isMatch(s, p, i - 1, j - 1);
        }
        //处理a
        if (p[j] == s[i]) {
            return isMatch(s, p, i - 1, j - 1);
        } else {
            return false;
        }

    }
    bool isMatch(string s, string p) {
        return isMatch(s, p, s.size() - 1, p.size() - 1);
    }
};
```

这个解法相当于一种穷举了，时间消耗非常大，我们能否考虑将中间的一些重复计算省下来呢？答案是肯定的。

思路 2:时间复杂度: O(len(s) * len§) 空间复杂度: O(len(s) * len§)

思路一我们写的很暴力，就是穷举，现在我们可以把一些中间状态记录下来，这样的话就是dp的思路了

画一个表格来看一下状况

```
  c * a * b
 0 1 2 3 4 5
 0 1 0 1 0 1 0
a 1 0 0 0 1 1 0
a 2 0 0 0 0 1 0
b 3 0 0 0 0 0 1
```

这里有几个取巧/容易出问题的地方，这里画的表用的是1-based index（即不是从0开始写index）。一上来，做的事包括：

- 初始化，空字符匹配：dp[0][0] =1
- 第一行，c* 可以匹配空字符，c* a* 可以匹配空字符，p[j-1] != s[i]，匹配空字符
- 然后进入第二行再来看，实际上我们可以看到，如果没有碰到 * 匹配还是很朴素的，但是碰到 * ：
  - 1这个匹配可以从左侧传来，dp[i][j] = dp[i][j-1]，that is 匹配 1个
  - 1 也可以有上方传来，这种情况是p[j-1] = s[i]，匹配多个 dp[i][j] = dp[i-1][j]
  - 1 这个匹配也可以从间隔一个的左侧传来，that is也可以有个性的匹配0个,如同匹配空字符一样dp[i][j] = dp[i][j-2]，但是注意匹配0个实际上有两种状况，如果p[j-1]!=s[i],强制匹配0个，即使p[j-1] == s[i],我们也可以傲娇的用它来匹配0个。

再代码化一点：

- s[i] == p[j] 或者 p[j] == '.' : dp[i][j] = dp[i-1][j-1]
- p[j] == '*': 然后分几种情况
  - p[j-1] != s[i] : dp[i][j] = dp[i][j-2] 匹配0个的状况
  - p[j-1] == s[i] or p[i-1] == '.':
    - dp[i][j] = dp[i-1][j] 匹配多个s[i]
    - dp[i][j] = dp[i][j-2] 匹配0个

AC代码，注意一下，因为上表为了表达方便，用的是1-based string系统，实际写代码的时候我们心里还是清楚这个string还是从0开始的，不过也可以尝试往前面添东西来方便。

Python beats 79.65%

```
class Solution:
    def isMatch(self, s: str, p: str) -> bool:
        m, n = len(s), len(p)
        dp = [ [0 for i in range(n+1)] for j in range(m+1)]

        dp[0][0] = 1

        # 初始化第一行
        for i in range(2,n+1):
            if p[i-1] == '*':
                dp[0][i] = dp[0][i-2]

        for i in range(1,m+1):
            for j in range(1,n+1):
                if p[j-1] == '*':
                    if p[j-2] != s[i-1] and p[j-2] != '.':
                        dp[i][j] = dp[i][j-2]
                    elif p[j-2] == s[i-1] or p[j-2] == '.':
                        dp[i][j] = dp[i-1][j] or dp[i][j-2]

                elif s[i-1] == p[j-1] or p[j-1] == '.':
                    dp[i][j] = dp[i-1][j-1]

        return dp[-1][-1] == 1
```

Java beats 65.39%

```
class Solution {
    public boolean isMatch(String s, String p) {
        boolean[][] dp = new boolean[s.length() + 1][p.length() + 1];
        for (int i = 0; i <= s.length(); i++) {
            for (int j = 0; j <= p.length(); j++) {
                if (i == 0 && j == 0) {
                    dp[i][j] = true;
                }
                if (i == 0) {
                    if (j >= 2 && p.charAt(j - 1) == '*') {
                        dp[i][j]= dp[i][j - 2];
                    }
                    continue;
                }
                if (j == 0) {
                    continue;
                }
                // 处理.*
                if (j >= 2 && p.charAt(j - 2) == '.' && p.charAt(j - 1) == '*') {
                    dp[i][j] = dp[i - 1][j] || dp[i][j - 2];
                    continue;
                }
                // 处理 a*
                if (j >= 2 && p.charAt(j - 1) == '*') {
                    if (s.charAt(i - 1) == p.charAt(j - 2)) {
                        dp[i][j] = dp[i - 1][j] || dp[i][j - 2];
                    } else {
                        dp[i][j] = dp[i][j - 2];
                    }
                    continue;
                }
                // 处理.
                if (p.charAt(j - 1) == '.') {
                    dp[i][j] = dp[i - 1][j - 1];
                }
                // 处理a
                if (p.charAt(j - 1) == s.charAt(i - 1)) {
                    dp[i][j] = dp[i - 1][j - 1];
                }
            }

        }
        return dp[s.length()][p.length()];
    }
}
```

Go beats 100%

```
func isMatch(s string, p string) bool {
    m, n := len(s), len(p)
    dp := make([][]int, m+1)
    for i := 0; i < m + 1; i += 1 {
        dp[i] = make([]int, n+1)
    }
    dp[0][0] = 1
    // 初始化第一行
    for i := 2; i < n+1; i += 1 {
        if p[i-1] == '*' {
            dp[0][i] = dp[0][i-2]
        }
    }

    for i := 1; i < m+1; i += 1 {
        for j := 1; j < n+1; j += 1 {
            if p[j-1] == '*' {
                if p[j-2] != s[i-1] && p[j-2] != '.' {
                    dp[i][j] = dp[i][j-2]
                } else if p[j-2] == s[i-1] || p[j-2] == '.' {
                    if dp[i-1][j] == 1 || dp[i][j-2] == 1 {
                        dp[i][j] = 1
                    } else {
                        dp[i][j] = 0
                    }
                }
            } else if s[i-1] == p[j-1] || p[j-1] == '.' {
                dp[i][j] = dp[i-1][j-1]
            }
        }
    }
    return dp[m][n] == 1
}
```

c++ beats: 89.19%

```
class Solution {
public:
    bool isMatch(string s, string p) {
        vector<vector<bool>> dp(s.size() + 1, vector<bool>(p.size() + 1, false));
        for (int i = 0;i <= s.size();i++) {
            for (int j = 0;j <= p.size();j++) {
                if (i == 0 && j == 0) {
                    dp[i][j] = true;
                    continue;
                }
                if (i == 0) {
                    if (j >= 2 && p[j - 1] == '*') {
                        dp[i][j] = dp[i][j - 2];
                    }
                    continue;
                }
                if (j == 0) {
                    continue;
                }
                //处理.*
                if (j >= 2 && p[j - 2] == '.' && p[j - 1] == '*') {
                    dp[i][j] = dp[i - 1][j] || dp[i][j - 2];
                    continue;
                }
                //处理a*
                if (j >= 2 && p[j - 1] == '*') {
                    if (s[i - 1] == p[j - 2]) {
                        dp[i][j] = dp[i - 1][j] || dp[i][j - 2];
                    } else {
                        dp[i][j] = dp[i][j - 2];
                    }
                    continue;
                }
                //处理.
                if (p[j - 1] == '.') {
                    dp[i][j] = dp[i - 1][j - 1];
                }
                //处理a
                if (p[j - 1] == s[i - 1]) {
                    dp[i][j] = dp[i - 1][j - 1];
                }
            }
        }
        return dp[s.size()][p.size()];
    }
};
```

思路二通过将我们计算过程中的状态记录下来，使得我们的时间消耗大大减小

总结

当我们看到一个问题，发现这个问题用穷举暴力等解法可以解决，但是中间会有很多重复计算的话，我们就可以通过将中间状态保存下来的方法来避免重复计算，如果这些状态之间有关系，可以互相转化，我们就可以想到用dp的思路了

}

29 寻找两个有序数组的中位数

31 合并K个排序链表