字符串匹配与动态规划

原创 赵云龙 LOA算法学习笔记 2021-03-02 18:20

01 前言

本文分享两道背景为字符串匹配（通配符匹配和正则表达式匹配）的动态规划题目的解题思路。

通配符是一种模糊搜索字符串的技术，很多地方都支持通配符进行查找（如linux文件名的匹配查找）。正则表达式广泛应用于文本数据处理与分析，熟练掌握正则表达式能够方便地解决一些文本处理的任务。实际中正则表达式引擎的实现基于DFA、NFA等有限状态机技术。本文以简化形式的这两种字符串匹配技术的题目，介绍动态规划求解问题的思路。

02 通配符匹配

2.1 问题描述

给定一个字符串 (s) 和一个模式串 (p)，实现一个支持 **?** 和 ***** 的通配符匹配。

- **?** 可以匹配任何单个字符。
- ***** 可以匹配任意字符串（包括空字符串）。

两个字符串完全匹配才算匹配成功。

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 *****。

2.2 问题结构分析

根据卜老师上课介绍的思路，我们很容易就能够想到对该问题子问题形式的定义。设 $OPT(i,j)$ 以 i 结尾的字符和以 j 结尾的模板串能否正确匹配的子问题。下面考虑如何写出完整的最优子结构。

我们以模式串为视角考虑所有可能出现的匹配情况。当模式串在j处字符 p_j 为字母时，如果与字符串中字符相同，则在该处能够匹配，是否匹配取决于字符串与模板串前缀能否匹配；当模式串在j处字符 p_j 为 **?** 时，可以顺利匹配字符串中的任意字符，此时能否正确匹配也取决于字符串与模板串前缀能否匹配；当 p_j 为 ***** 时，能够匹配任意字符串，包括空字符串，此时有两种选择，一种是 ***** 匹配字符，则模板串不动，字符串向前一位。另一种 ***** 不匹配字符，则字符串不动，模板串向前一位。总结下来，最优子结构可以写作下式：

$$OPT[i][j] = \begin{cases} OPT[i-1][j-1], & s_i == p_j \text{ or } p_j == ? \\ OPT[i][j-1] \text{ or } OPT[i-1][j], & p_j == * \\ \text{False}, & otherwise \end{cases}$$

有了递归表达式，我们还需要知道最平凡的情况，作为递归的出口。当两个字符串均为空串时，匹配成功，$OPT[0][0]$ 初始化为True；当模板串为空串时，任意字符串均为匹配失败，初始化OPT表第一列为False；当字符串为空串时，模板串中只有 ***** 能够匹配成功，根据此转移状态初始化OPT表的第一行。

2.3 代码实现

```
def isMatch(self, s: str, p: str) -> bool:
        m = len(s)
        n = len(p)


        opt = [[False for i in range(n+1)] for j in range(m+1)]
        opt[0][0] = True;
        for i in range(1,n+1):
            if(p[i-1]=='*'):
                opt[0][i] = opt[0][i-1]
        for i in range(1,m+1):
            for j in range(1,n+1):
                if(p[j-1]==s[i-1] or p[j-1]=='?'):
                    opt[i][j] = opt[i-1][j-1]
                elif(p[j-1]=='*'):
                    opt[i][j] = opt[i][j-1] or opt[i-1][j]
        return opt[m][n]
    }
```

由于状态转移只依赖OPT表的前一行，可以用两个滚动数组替换OPT，对空间复杂度进行优化。

```
def isMatch(self, s: str, p: str) -> bool:
        m = len(s)
        n = len(p)
        opt = [False for i in range(n+1)]
        opt[0] = True


        for i in range(1,n+1):
            if(p[i-1]=='*'):
                opt[i] = opt[i-1]


        for i in range(1,m+1):
            old_opt = opt.copy()
            opt[0] = False
            for j in range(1,n+1):
                if(p[j-1]==s[i-1] or p[j-1]=='?'):
                    opt[j] = old_opt[j-1]
                elif(p[j-1]=='*'):
                    opt[j] = opt[j] or opt[j-1]
                else:
                    opt[j] = False
        return opt[n]
```

03 正则表达式匹配

3.1 问题描述

给定一个字符串 s 和一个模板 p，请你来实现一个支持 . 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那一个元素

两个字符串完全匹配才算匹配成功，题中保证模板串 * 前有字符。

3.2 问题结构分析

该问题与通配符匹配问题形式相似，不同的地方是该题中 * 的匹配内容需要取决于前一个字符。同样以模板串的视角考虑可能出现的情况，其他情况与上一题一致，只需要对 p_j 为 * 时修改。将 * 与其前一个字符看做一个整体，匹配字符时，字符串向前一位，模板串不动。不匹配字符时，字符串不动，模板串向前两位。当 * 前一个字符与字符串待匹配字符不同时，无法选择匹配，只能由不匹配字符的状态转移。总结下来，最优子结构可以写作下式：

$$OPT[i][j] = \begin{cases} OPT[i-1][j-1], & s_i == p_j \text{ or } p_j == . \\ \begin{cases} OPT[i][j-2] \text{ or } OPT[i-1][j], & p_{j-1} = s_i \text{ or } p_{j-1} == . \\ OPT[i][j-2], & otherwise \end{cases} & p_j == * \\ \text{False}, & otherwise \end{cases}$$

同样，我们也需要初始化最平凡的情况，作为递归的出口。其初始化与上一题仍然类似，只有当字符串为空时（OPT表的第一行），模板串进行匹配的状态转移需要修改。

3.3 代码实现

```
    def isMatch(self, s, p):
        m = len(s)
        n = len(p)



        opt = [[False for i in range(n+1)] for j in range(m+1)]
        opt[0][0] = True;
        for i in range(1,n+1):
            if(p[i-1]=='*'):
                opt[0][i] = opt[0][i-2]


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

```
                        opt[i][j] = opt[i][j-2]


        return opt[m][n]
```

同样可以通过滚动数组进行空间优化，形式类似，不再写出。

04 复杂度分析

这两道题目存在两重循环，时间复杂度均为 $O(mn)$，m、n为字符串和模板串的长度。未经过滚动数据优化的空间复杂度为 $O(mn)$ ，经过滚动数组优化空间复杂度为 $O(n)$ 。

05 总结

字符串匹配问题能够以状态转换的角度来求解，而动态规划也能够以状态转换的角度来看待。通过该题，我们可以分析动态规划问题的一般求解思路。首先定义子问题的形式，再寻求最优子结构的状态转移方程，并确定好状态转移方程最平凡情况下的出口，该动态规划问题就求解完成了。在分析状态转移方程时，我们需要把握好不同情况下的状态形式，以及其状态转移的来源。

以上是本人对字符串匹配和动态规划问题的的一些分析，水平有限，如有谬误，还请指正。