

突然就不香了的字符串

今日目标：

1：完成相关面试题

面试实战

对于字符串的相关题目，如果加上动态规划（DP），整个的难度就会上升。比如之前作过的一些题目：

[1143. 最长公共子序列](#)

[72. 编辑距离](#)

[5. 最长回文子串](#)

10. 正则表达式匹配

[字节，华为，阿里面试题，10. 正则表达式匹配](#)

解法1：递归

```
1  class Solution {
2      //递归解法
3      public boolean isMatch(String s, String p) {
4          char[] ss = s.toCharArray();
5          char[] pp = p.toCharArray();
6          return isMatch(ss,0,pp,0);
7      }
8      public boolean isMatch(char[] s,int i,char[] p,int j) {
9          //terminal
10         if (j==p.length) {
11             return i==s.length;
12         }
13         // current logic
14         //单一匹配模式:isMatch是当前这一位的匹配结果
15         boolean isMatch = i<s.length && (s[i]==p[j] || p[j]=='.');
16         //如果下一位是*, 则从当前位开始就是任意匹配模式
17         if (p.length -j >=2 && p[j+1] =='*') {
18             return isMatch(s,i,p,j+2) || (isMatch && isMatch(s,i+1,p,j));
19         }
20         return isMatch && isMatch(s,i+1,p,j+1); //&&isMatch的理由是要看所有位的匹
21         配结果
22     }
23 }
```

算法分析过程见资料：[字符串dp.xltx](#)

解法2：动态规划

```

1  class Solution {
2      //动态规划解法
3      public boolean isMatch(String s, String p) {
4          char[] ss = s.toCharArray();
5          char[] pp = p.toCharArray();
6          int m = ss.length;
7          int n = pp.length;
8          //定义dp数组
9          boolean [][] dp = new boolean[m+1][n+1];
10         //初始化
11         dp[0][0] = true;
12         for (int k = 0; k < pp.length; k++) {
13             if (pp[k] == '*' && dp[0][k - 1]) {
14                 dp[0][k + 1] = true; //此处k代表的是下标,而dp[i][j],代表的是第j个
15                                     //字符
16             }
17         }
18         //状态遍历
19         for (int i=1;i<=m;i++) {
20             for (int j=1;j<=n;j++) {
21                 if (ss[i-1] == pp[j-1] || pp[j-1] == '.') {
22                     dp[i][j] = dp[i-1][j-1];
23                 }else if (pp[j-1] == '*') {
24                     if ( pp[j-1-1] == ss[i-1] || pp[j-1-1] =='.') {
25                         dp[i][j] = dp[i-1][j] || dp[i][j-2];
26                     }else {
27                         dp[i][j] = dp[i][j-2];
28                     }
29                 }
30             }
31         }
32         return dp[m][n];
33     }
34 }

```

算法分析过程见资料: [字符串dp.xlsx](#)

进阶题目:

[44. 通配符匹配](#)

115. 不同的子序列

[头条, 百度, 字节, 优步面试题, 115. 不同的子序列](#)

解法1: 回溯

未AC版:

```

1  class Solution {

```

```

2     int total ;
3     //回溯
4     public int numDistinct(String s, String t) {
5         char[] ss = s.toCharArray();
6         char[] tt = t.toCharArray();
7         backtrack(ss,0,tt,0);
8         return total;
9     }
10
11    public void backtrack(char[] s,int i,char[] t,int j) {
12        //terminal
13        if (j == t.length) {
14            total++;
15            return;
16        }
17        if (i== s.length) {
18            return;
19        }
20        //current logic
21        //对于s中当前i位的字符,我们有两种选择,选和不选
22        /*
23            当前i和j位置的字符相等,我们可以选也可以不选
24            选:i,j均向后移动一位继续操作
25            不选:i向后移动一位,j不变
26            当前i和j位置的字符不等,我们只能不选
27            不选:i向后移动一位,j不变
28        */
29        if (s[i] == t[j]) {
30            backtrack(s,i+1,t,j+1);
31        }
32        backtrack(s,i+1,t,j);
33    }
34 }

```

太长的测试用例耗时太长, 如何优化, 从上而下记忆化递归

```

1    class Solution {
2        int total ;
3        //回溯
4        public int numDistinct(String s, String t) {
5            char[] ss = s.toCharArray();
6            char[] tt = t.toCharArray();
7            //创建缓存:缓存的key是我们作过选择的下标,value是从对应下标位置处开始做选择最终匹
            配成功的个数
8            Map<String,Integer> cache = new HashMap();
9            backtrack(ss,0,tt,0,cache);
10           return total;
11       }
12       //回溯+记忆化
13       public void backtrack(char[] s,int i,char[] t,int j,Map<String,Integer>
            cache) {
14           //terminal
15           if (j == t.length) {
16               total++;
17               return;

```

```

18     }
19     if (i == s.length) {
20         return;
21     }
22
23     String key = i + "_" + j;
24     if (cache.containsKey(key)) {
25         total += cache.get(key);
26         return;
27     }
28     //后面需要将当前的, i, j 位置做选择最后能匹配成功的个数记录到cache中
29     int currentTotal = total;
30
31     //current logic
32     //对于s中当前i位的字符, 我们两种选择, 选和不选
33     /*
34         当前i和j位置的字符相等, 我们可以选也可以不选
35         选: i, j 均向后移动一位继续操作
36         不选: i 向后移动一位, j 不变
37         当前i和j位置的字符不等, 我们只能不选
38         不选: i 向后移动一位, j 不变
39     */
40     if (s[i] == t[j]) {
41         backtrack(s, i+1, t, j+1, cache);
42     }
43     backtrack(s, i+1, t, j, cache);
44
45     //将total的增量存起来即为从当前的i, j 位置开始做选择最后能匹配成功的个数
46     cache.put(key, total - currentTotal);
47 }
48 }

```

解法2: 动态规划

```

1 class Solution {
2     //dp解法
3     public int numDistinct(String s, String t) {
4         char[] ss = s.toCharArray();
5         char[] tt = t.toCharArray();
6         int m = ss.length;
7         int n = tt.length;
8         //定义dp数组
9         int[][] dp = new int[m+1][n+1];
10        //初始化
11        for (int i=0; i<=m; i++) {
12            dp[i][0] = 1;
13        }
14
15        for (int i=1; i<=m; i++) {
16            for (int j=1; j<=n; j++) {
17                if (ss[i-1] == tt[j-1]) {
18                    dp[i][j] = dp[i-1][j-1] + dp[i-1][j];
19                } else {
20                    dp[i][j] = dp[i-1][j];
21                }
22            }
23        }
24    }
25 }

```

```

22     }
23 }
24 return dp[m][n];
25 }
26 }

```

状态压缩:

```

1  class Solution {
2      //dp解法
3      public int numDistinct(String s, String t) {
4          char[] ss = s.toCharArray();
5          char[] tt = t.toCharArray();
6          int m = ss.length;
7          int n = tt.length;
8          //定义dp数组
9          //int[][] dp = new int[m+1][n+1];
10         //状态压缩
11         int[] dp = new int[n+1];
12         //初始化
13         // for (int i=0;i<=m;i++) {
14         //     dp[i][0] = 1;
15         // }
16         dp[0] = 1;
17
18         for (int i=1;i<=m;i++) {
19             //状态压缩后需要从后向前遍历----特别注意
20             for (int j=n;j>=1;j--) {
21                 if (ss[i-1] == tt[j-1]) {
22                     dp[j] = dp[j-1] + dp[j];
23                 } else {
24                     dp[j] = dp[j];
25                 }
26             }
27         }
28         return dp[n];
29     }
30 }

```

字符串匹配/查找算法介绍

[28.实现 strStr\(\)](#)

参考精选题解

字符串匹配/查找算法有：BF，RK，KMP等等！