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

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

> 算法分析过程见资料：`字符串dp.xlsx`

解法2：动态规划

```java
class Solution {
    //动态规划解法
    public boolean isMatch(String s, String p) {
        char[] ss = s.toCharArray();
        char[] pp = p.toCharArray();
        int m = ss.length;
        int n= pp.length;
        //定义dp数组
        boolean [][] dp = new boolean[m+1][n+1];
        //初始化
        dp[0][0] = true;
        for (int k = 0; k < pp.length; k++) {
            if (pp[k] == '*' && dp[0][k - 1]) {
                dp[0][k + 1] = true; //此处k代表的是下标,而dp[i][j],代表的是第j个字符
            }
        }

        //状态遍历
        for (int i=1;i<=m;i++) {
            for (int j=1;j<=n;j++) {
                if ( ss[i-1] == pp[j-1] || pp[j-1] == '.') {
                    dp[i][j] = dp[i-1][j-1];
                }else if (pp[j-1] == '*') {
                    if ( pp[j-1-1] == ss[i-1] || pp[j-1-1] =='.') {
                        dp[i][j] = dp[i-1][j] || dp[i][j-2];
                    }else {
                        dp[i][j] = dp[i][j-2];
                    }
                }
            }
        }
        return dp[m][n];
    }
}
```

> 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：

```java
class Solution {
```

```
    int total ;
    //回溯
    public int numDistinct(String s, String t) {
        char[] ss = s.toCharArray();
        char[] tt = t.toCharArray();
        backtrack(ss,0,tt,0);
        return total;
    }

    public void backtrack(char[] s,int i,char[] t,int j) {
        //terminal
        if (j == t.length) {
            total++;
            return;
        }
        if (i== s.length) {
            return;
        }
        //current logic
        //对于s中当前i位的字符,我们有两种选择,选和不选
        /*
            当前i和j位置的字符相等,我们可以选也可以不选
                选:i,j均向后移动一位继续操作
                不选:i向后移动一位,j不变
            当前i和j位置的字符不等,我们只能不选
                不选:i向后移动一位,j不变
        */
        if (s[i] == t[j]) {
            backtrack(s,i+1,t,j+1);
        }
        backtrack(s,i+1,t,j);
    }
}
```

太长的测试用例耗时太长，如何优化，从上而下记忆化递归

```
class Solution {
    int total ;
    //回溯
    public int numDistinct(String s, String t) {
        char[] ss = s.toCharArray();
        char[] tt = t.toCharArray();
        //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
        Map<String,Integer> cache = new HashMap();
        backtrack(ss,0,tt,0,cache);
        return total;
    }
    //回溯+记忆化
    public void backtrack(char[] s,int i,char[] t,int j,Map<String,Integer> cache) {
        //terminal
        if (j == t.length) {
            total++;
            return;
```

```
        }
        if (i== s.length) {
            return;
        }

        String key = i+"_" + j;
        if (cache.containsKey(key)) {
            total += cache.get(key);
            return;
        }
        //后面需要将从当前的，i，j位置做选择最后能匹配成功的个数记录到cache中
        int currentTotal = total;

        //current logic
        //对于s中当前i位的字符,我们有两种选择,选和不选
        /*
            当前i和j位置的字符相等,我们可以选也可以不选
                选:i,j均向后移动一位继续操作
                不选:i向后移动一位,j不变
            当前i和j位置的字符不等,我们只能不选
                不选:i向后移动一位,j不变
        */
        if (s[i] == t[j]) {
            backtrack(s,i+1,t,j+1,cache);
        }
        backtrack(s,i+1,t,j,cache);

        //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
        cache.put(key,total - currentTotal);
    }
}
```

解法2：动态规划

```
class Solution {
    //dp解法
    public int numDistinct(String s, String t) {
        char[] ss = s.toCharArray();
        char[] tt = t.toCharArray();
        int m = ss.length;
        int n = tt.length;
        //定义dp数组
        int[][] dp = new int[m+1][n+1];
        //初始化
        for (int i=0;i<=m;i++) {
            dp[i][0] = 1;
        }

        for (int i=1;i<=m;i++) {
            for (int j=1;j<=n;j++) {
                if (ss[i-1] == tt[j-1]) {
                    dp[i][j] = dp[i-1][j-1] + dp[i-1][j];
                }else {
                    dp[i][j] = dp[i-1][j];
                }
```

```
            }
        }
        return dp[m][n];
    }
}
```

状态压缩：

```
class Solution {
    //dp解法
    public int numDistinct(String s, String t) {
        char[] ss = s.toCharArray();
        char[] tt = t.toCharArray();
        int m = ss.length;
        int n = tt.length;
        //定义dp数组
        //int[][] dp = new int[m+1][n+1];
        //状态压缩
        int[] dp = new int[n+1];
        //初始化
        // for (int i=0;i<=m;i++) {
        //     dp[i][0] = 1;
        // }
        dp[0] = 1;

        for (int i=1;i<=m;i++) {
            //状态压缩后需要从后向前遍历----特别注意
            for (int j=n;j>=1;j--) {
                if (ss[i-1] == tt[j-1]) {
                    dp[j] = dp[j-1] + dp[j];
                }else {
                    dp[j] = dp[j];
                }
            }
        }
        return dp[n];
    }
}
```

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！