0115. 不同的子序列

ITCharge 大约 3 分钟

- 标签：字符串、动态规划
- 难度：困难

题目链接

- 0115. 不同的子序列 - 力扣

题目大意

描述：给定两个字符串 `s` 和 `t` 。

要求：计算在 `s` 的子序列中 `t` 出现的个数。

说明：

- **字符串的子序列**：通过删除一些（也可以不删除）字符且不干扰剩余字符相对位置所组成的新字符串。（例如，`"ACE"` 是 `"ABCDE"` 的一个子序列，而 `"AEC"` 不是）。
- $0 \leq s.length, t.length \leq 1000$。
- `s` 和 `t` 由英文字母组成。

示例：

- 示例 1：

```py
输入：s = "rabbbit", t = "rabbit"
输出：3
解释：如下图所示，有 3 种可以从 s 中得到 "rabbit" 的方案。
```

$\underline{rabbbit}\underline{ra}\underline{bbbit}\underline{rabbbit}$

解题思路

思路 1：动态规划

1. 划分阶段

按照子序列的结尾位置进行阶段划分。

2. 定义状态

定义状态 `dp[i][j]` 表示为：以第 `i - 1` 个字符为结尾的 `s` 子序列中出现以第 `j - 1` 个字符为结尾的 `t` 的个数。

3. 状态转移方程

双重循环遍历字符串 `s` 和 `t`，则状态转移方程为：

- 如果 `s[i - 1] == t[j - 1]`，则：`dp[i][j] = dp[i - 1][j - 1] + dp[i - 1][j]`。即 `dp[i][j]` 来源于两部分：
  - 使用 `s[i - 1]` 匹配 `t[j - 1]`，则 `dp[i][j]` 取源于以 `i - 2` 为结尾的 `s` 子序列中出现以 `j - 2` 为结尾的 `t` 的个数，即 `dp[i - 1][j - 1]`。
  - 不使用 `s[i - 1]` 匹配 `t[j - 1]`，则 `dp[i][j]` 取源于以 `i - 2` 为结尾的 `s` 子序列中出现以 `j - 1` 为结尾的 `t` 的个数，即 `dp[i - 1][j]`。
- 如果 `s[i - 1] != t[j - 1]`，那么肯定不能用 `s[i - 1]` 匹配 `t[j - 1]`，则 `dp[i][j]` 取源于 `dp[i - 1][j]`。

4. 初始条件

- `dp[i][0]` 表示以 `i - 1` 为结尾的 `s` 子序列中出现空字符串的个数。把 `s` 中的元素全删除，出现空字符串的个数就是 `1`，则 `dp[i][0] = 1`。
- `dp[0][j]` 表示空字符串中出现以 `j - 1` 结尾的 `t` 的个数，空字符串无论怎么变都不会变成 `t`，则 `dp[0][j] = 0`
- `dp[0][0]` 表示空字符串中出现空字符串的个数，这个应该是 `1`，即 `dp[0][0] = 1`。

5. 最终结果

根据我们之前定义的状态，`dp[i][j]` 表示为：以第 `i - 1` 个字符为结尾的 `s` 子序列中出现以第 `j - 1` 个字符为结尾的 `t` 的个数。则最终结果为 `dp[size_s][size_t]`，将其返回即可。

思路 1：动态规划代码

```py
class Solution:
    def numDistinct(self, s: str, t: str) -> int:
        size_s = len(s)
        size_t = len(t)
        dp = [[0 for _ in range(size_t + 1)] for _ in range(size_s + 1)]
        for i in range(size_s):
            dp[i][0] = 1
        for i in range(1, size_s + 1):
            for j in range(1, size_t + 1):
                if s[i - 1] == t[j - 1]:
                    dp[i][j] = dp[i - 1][j - 1] + dp[i - 1][j]
                else:
                    dp[i][j] = dp[i - 1][j]
        return dp[size_s][size_t]
```

思路 1：复杂度分析

- **时间复杂度**：$O(n^2)$。两重循环遍历的时间复杂度是 $O(n^2)$，所以总的时间复杂度为 $O(n^2)$。
- **空间复杂度**：$O(n^2)$。用到了二维数组保存状态，所以总体空间复杂度为 $O(n^2)$。