方法一：动态规划

思路与算法

题目中的匹配是一个「逐步匹配」的过程：我们每次从字符串 p 中取出一个字符或者「字符 + 星号」的组合，并在 s 中进行匹配。对于 p 中一个字符而言，它只能在 s 中匹配一个字符，匹配的方法具有唯一性；而对于 p 中字符 + 星号的组合而言，它可以在 s 中匹配任意自然数个字符，并不具有唯一性。因此我们可以考虑使用动态规划，对匹配的方案进行枚举。

我们用 f[i][j] 表示 s 的前 i 个字符与 p 中的前 j 个字符是否能够匹配。在进行状态转移时，我们考虑 p 的第 j 个字符的匹配情况：

如果 p 的第 j 个字符是一个小写字母，那么我们必须在 s 中匹配一个相同的小写字母，即



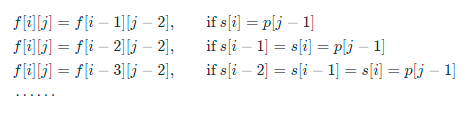
也就是说，如果 s 的第 i 个字符与 p 的第 j 个字符不相同，那么无法进行匹配；否则我们可以匹配两个字符串的最后一个字符，完整的匹配结果取决于两个字符串前面的部分。

如果 p 的第 j 个字符是 \*，那么就表示我们可以对 p 的第 j-1 个字符匹配任意自然数次。在匹配 0 次的情况下，我们有



也就是我们「浪费」了一个字符 + 星号的组合，没有匹配任何 s 中的字符。

在匹配 1,2,3,⋯ 次的情况下，类似地我们有

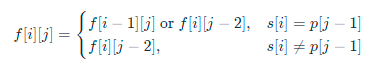


如果我们通过这种方法进行转移，那么我们就需要枚举这个组合到底匹配了 s 中的几个字符，会增导致时间复杂度增加，并且代码编写起来十分麻烦。我们不妨换个角度考虑这个问题：字母 + 星号的组合在匹配的过程中，本质上只会有两种情况：

1、匹配 s 末尾的一个字符，将该字符扔掉，而该组合还可以继续进行匹配；

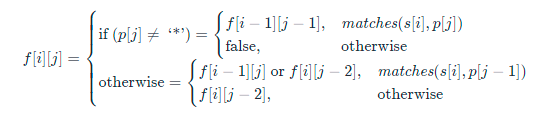
2、不匹配字符，将该组合扔掉，不再进行匹配。

如果按照这个角度进行思考，我们可以写出很精巧的状态转移方程：



在任意情况下，只要 p[j] 是 .，那么 p[j] 一定成功匹配 s 中的任意一个小写字母。

最终的状态转移方程如下：



其中 matches(x, y) 判断两个字符是否匹配的辅助函数。只有当 y 是 . 或者 x 和 y 本身相同时，这两个字符才会匹配。

细节

动态规划的边界条件为 f[0][0] = true，即两个空字符串是可以匹配的。最终的答案即为 f[m][n]，其中 m 和 n 分别是字符串 s 和 p 的长度。由于大部分语言中，字符串的字符下标是从 0 开始的，因此在实现上面的状态转移方程时，需要注意状态中每一维下标与实际字符下标的对应关系。

在上面的状态转移方程中，如果字符串 p 中包含一个「字符 + 星号」的组合（例如 a\*），那么在进行状态转移时，会先将 a 进行匹配（当 p[j] 为 a 时），再将 a\* 作为整体进行匹配（当 p[j] 为 \* 时）。然而，在题目描述中，我们必须将 a\* 看成一个整体，因此将 a 进行匹配是不符合题目要求的。看来我们进行了额外的状态转移，这样会对最终的答案产生影响吗？这个问题留给读者进行思考。