数据结构与算法题解（7）：最短编辑距离

现代搜索技术的发展很多以提供优质、高效的服务作为目标。比如说：baidu、google、sousou等知名全文搜索系统。当我们输入一个错误的query="Jave" 的时候，返回中有大量包含正确的拼写 "Java"的网页。是怎么做到的呢？这其中，字符串的相似度计算是做到这一点的方法之一。

一、字符串编辑距离

是一种字符串之间相似度计算的方法。给定两个字符串S、T，将S转换成T所需要的删除，插入，替换操作的数量就叫做S到T的编辑路径。而最短的编辑路径就叫做字符串S和T的编辑距离。

举个例子：S=“eeba” T="abac" 我们可以按照这样的步骤转变：(1) 将S中的第一个e变成a;(2) 删除S中的第二个e;(3)在S中最后添加一个c; 那么S到T的编辑路径就等于3。当然，这种变换并不是唯一的，但如果3是所有变换中最小值的话。那么我们就可以说S和T的编辑距离等于3了。

二、动态规划解决编辑距离

动态规划(dynamic programming)是一种解决复杂问题最优解的策略。它的基本思路就是：将一个复杂的最优解问题分解成一系列较为简单的最优解问题，再将较为简单的的最优解问题进一步分解，直到可以一眼看出最优解为止。

动态规划算法是解决复杂问题最优解的重要算法。其算法的难度并不在于算法本身的递归难以实现，而主要是编程者对问题本身的认识是否符合动态规划的思想。现在我们就来看看动态规划是如何解决编辑距离的。

假设$dp[i-1][j-1]$表示一个长为$i-1$的字符串$str1$变为长为$j-1$的字符串str2的最短距离，如果我们此时想要把$str1a$这个字符串变成$str2b$这个字符串，我们有如下几种选择：

- 替换： 在str1变成str2的步骤后，我们将str1a中的a替换为b，就得到str2b (如果a和b相等，就不用操作)
- 增加： 在str1a变成str2的步骤后，我们再在末尾添加一个b，就得到str2b (str1a先根据已知距离变成str2，再加个b)
- 删除： 在str1变成str2b的步骤后，对于str1a，我们将末尾的a删去，就得到str2b (str1a将a删去得到str1，而str1到str2b的编辑距离已知)

根据这三种操作，我们可以得到递推式

若a和b相等：

```
dp[i][j] = min(dp[i-1][j]+1, dp[i][j-1]+1, dp[i-1][j-1])
```

若a和b不相等：

```
dp[i][j] = min(dp[i-1][j]+1, dp[i][j-1]+1, dp[i-1][j-1]+1)
```

因为将一个非空字符串变成空字符串的最小操作数是字母个数（全删），反之亦然，所以：

```
dp[0][j]=j, dp[i][0]=i
```

最后我们只要返回dp[m][n]即可，其中m是word1的长度，n是word2的长度

三、代码

```
public class Demo {
    public static int minDistance(String word1, String word2) {
        int m = word1.length(), n = word2.length();
        int[][] dp = new int[m + 1][n + 1];
        // 初始化空字符串的情况
        for(int i = 1; i <= m; i++){
            dp[i][0] = i;
        }
        for(int i = 1; i <= n; i++){
            dp[0][i] = i;
        }
        for(int i = 1; i <= m; i++){
            for(int j = 1; j <= n; j++){
                // 增加操作：str1a变成str2后再加上b，得到str2b
                int insertion = dp[i][j-1] + 1;
                // 删除操作：str1a删除a后，再由str1变为str2b
                int deletion = dp[i-1][j] + 1;
                // 替换操作：先由str1变为str2，然后str1a的a替换为b，得到str2b
                int replace = dp[i-1][j-1] + (word1.charAt(i - 1) == word2.charAt(j
 - 1) ? 0 : 1);
                // 三者取最小
                dp[i][j] = Math.min(replace, Math.min(insertion, deletion));
            }
        }
        return dp[m][n];
    }
```

```
    public static void main(String[] args) {
        String source = "zhanghua";
        String target = "zhanghau";
        System.out.println("minDistance=" + minDistance(source, target));
    }
}
```

测试结果为：

```
minDistance=2
```