莱文斯坦距离实验报告

一、简介

莱文斯坦距离（Levenshtein Distance）又称编辑距离（Edit Distance），是自然语言处理中评价模型好坏的标准之一，经常会使用计算得到的莱文斯坦距离作为模型的评分（正确率或错误率）。

定义如下：两个字符之间，有一个转变成另一个所需的最少编辑操作次数。被允许的操作有以下几种：

a. Replace 替换，将一个字符替换成另一个字符

b. Insert 插入，插入一个字符

c. Delete 删除，删除一个字符

一般来说，编辑的距离越小，两个字符的相似度越大。不难分析出，两个字符串的编辑距离肯定不超过它们的最大长度（可以通过先把短串的每一位都修改成长串对应位置的字符，然后插入长串中的剩下字符）。

二、算法实现

通过编辑距离来判断两个字符串是否相等，属于动态规划算法的一种形式。其数学定义如下：

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

1、简单举例

通过一个例子整理思路，如计算 kitten 和 sitting 之间的编辑距离：

kitten → sitten （替换“k”-> “s”）

sitten → sittin （替换“e”-> “i”）

sittin → sitting （插入“g”）

上面的变化过程所需要的步数就是最小的步数，所以他们之间的编辑距离就是 3。

2．递归法实现

递归法计算莱文斯坦距离是直观但效率比较差，重复计算相同的子序列很多次，有效的改进方式是构造一个距离表存储每次计算的结果，最右下元素即为两个全字符串的莱文斯坦距离。

In [1]:
```
# 递归法实现
def Levenshtein_Distance_Normal(a,b):
        l1,l2 = len(a),len(b)
    if min(l1,l2) == 0:
        return max(l1,l2)
    elif a[0] == b[0]:
        return Levenshtein_Distance(a[1:],b[1:])
    else:
        return 1 + min(Levenshtein_Distance(a[1:],b),Levenshtein_Distance(a,b[1:]),Levenshtein_Distance(a[1:],b[1:]))
```

3. 全矩阵迭代法实现

使用了自底向上的动态规划实现方法，也即上面提到的构造距离表取代重复计算相同子序列，相对于递归实现除速度显著改进外，算法运行时只需 s_len * t_len 的内存（s_len 和 t_len 分别是字符串 s 和 t 的长度）

In [3]:
```
# 动态规划实现
def Levenshtein_Distance_Dynamic(str1, str2):
  len_str1 = len(str1) + 1
  len_str2 = len(str2) + 1
  # 创建矩阵
  matrix = [0 for n in range(len_str1 * len_str2)]
  #矩阵的第一行
  for i in range(len_str1):
    matrix[i] = i
  # 矩阵的第一列
  for j in range(0, len(matrix), len_str1):
    if j % len_str1 == 0:
      matrix[j] = j // len_str1
  # 根据状态转移方程逐步得到编辑距离
  for i in range(1, len_str1):
    for j in range(1, len_str2):
      if str1[i-1] == str2[j-1]:
        cost = 0
      else:
        cost = 1
      matrix[j*len_str1+i] = min(matrix[(j-1)*len_str1+i]+1,
                    matrix[j*len_str1+(i-1)]+1,
                    matrix[(j-1)*len_str1+(i-1)] + cost)
    # 返回矩阵的最后一个值，也就是我们需要的编辑距离
    return matrix[-1]
```

3. 调库实现

用相应的库函数快速实现。

In [4]:
```
# 调库实现（需要先pip install Levenshtein）

import Levenshtein
print(Levenshtein.distance('kitten', 'sitting'))
```

```
3
```