自然语言处理及应用
实验报告

名称	最小编辑距离的实现
姓名	xxx
班级	xxx
学号	xxxxxxxxxxxxxxx
Email	xxiazixuan824@gmail.com
日期	2023/2/27

1. 实验题目

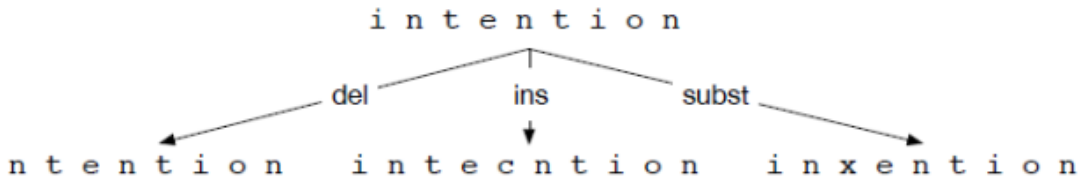
给出两个字符串str1和str2，对于字符串中，我们可以在str1上执行插入，删除和替换这3种操作来讲str1调整成为str2。我们的目标是通过设计算法来计算最小编辑距离

2. 实验环境

本次实验只需要对最小编辑距离进行实现，因此我们只需要在本地进行编程即可，这次用到的程序运行环境是VSCode和Python 3.11.

3. 算法设计

对于这个最小编辑距离的求解，可以看作一种最小路径的搜索，即从一个字符串到另一个字符串的过程。对于一个字符串，有三种形态：

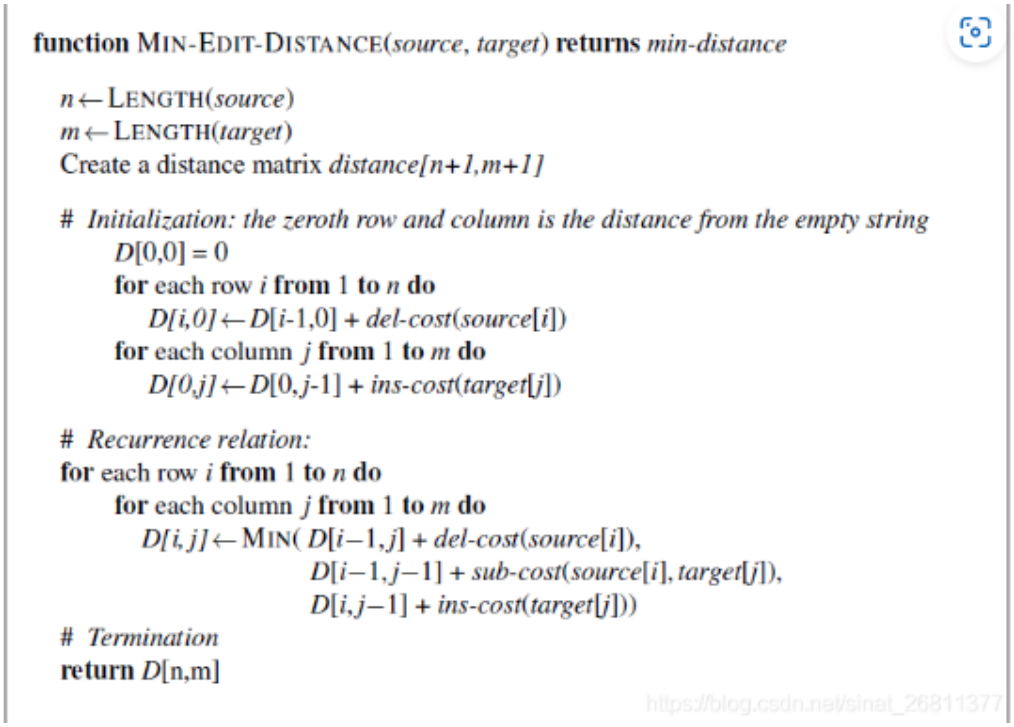


那么，给出str1和str2这两个字符串，显然我们可以通过DFS或者是BFS来进行搜索从而找到最优解，但是这样的时间复杂度是比较麻烦的，而不难发现字符串中的字符是有一定的递归关系的，而对于这种大规模的递归子问题求解，我们不难想到动态规划算法，算法设计如下：

1. dp数组的定义：由于我们要求的是两个字符串之间的最小操作次数，而对于每一个字符串，都有不同长度的字符，于是我们可以选择一个二维数组dp[i][j]进行dp函数的定义
2. dp数组的含义：上面我们提到了dp数组的定义是dp[i][j]，其中，i代表从str1的第一个字符到第i个字符构成的子字符串，j代表从str2的第一个字符到第j个字符构成的子字符串。
3. 递推关系的定义：对于dp[i][j]的值，我们可以分成以下几个情况：
   a. dp[i][j]与dp[i-1][j]有关
   b. dp[i][j]与dp[i][j-1]有关
   c. dp[i][j]与dp[i-1][j-1]有关

   递归函数定义如下：

$$dp[i][j] = min\begin{cases} dp[i-1][j]+1 \\ dp[i][j-1]+1 \\ dp[i-1][j-1]+\begin{cases}2, if\ str1(i) \neq str2(j) \\ 0. if\ str1(i) = str2(j)\end{cases}\end{cases}$$



```
function MIN-EDIT-DISTANCE(source, target) returns min-distance

  n ← LENGTH(source)
  m ← LENGTH(target)
  Create a distance matrix distance[n+1,m+1]

  # Initialization: the zeroth row and column is the distance from the empty string
      D[0,0] = 0
      for each row i from 1 to n do
          D[i,0] ← D[i-1,0] + del-cost(source[i])
      for each column j from 1 to m do
          D[0,j] ← D[0,j-1] + ins-cost(target[j])

  # Recurrence relation:
  for each row i from 1 to n do
      for each column j from 1 to m do
          D[i,j] ← MIN( D[i−1,j] + del-cost(source[i]),
                        D[i−1,j−1] + sub-cost(source[i], target[j]),
                        D[i,j−1] + ins-cost(target[j]))
  # Termination
  return D[n,m]
```


4. 代码实现与实现结果

根据算法设计，进行编写代码，如下所示：

```
1.  def Levenshtein_Distance(str1, str2):
2.      """
3.      计算字符串 str1 和 str2 的编辑距离
4.      :param str1
5.      :param str2
6.      :return:
7.      """
8.      dp = [[ i + j for j in range(len(str2) + 1)] for i in range(len(str1)
    + 1)]
9.
10.     for i in range(1, len(str1)+1):
11.         for j in range(1, len(str2)+1):
12.             if(str1[i-1] == str2[j-1]):
13.                 d = 0
14.             else:
15.                 d = 1
16.
17.             dp[i][j] = min(dp[i-1][j]+1, dp[i][j-1]+1, dp[i-1][j-1]+d)
18.
19.     return dp[len(str1)][len(str2)]
20.
21.
22. print(Levenshtein_Distance("abc", "bd"))
```

运行结果如下：

5. 算法的应用场景

当然，这堂课并不是单纯的算法课，因为这门课叫做“自然语言处理”，所以这个最小编辑距离一定是有一定的作用的，通过查阅资料我知道了，最小编辑距离可以应用在脱敏数据和明文数据匹配，拼写检查，抄袭检测等方面的应用。

在这些应用中，我最感兴趣的方向是文本检索提示，即随机输入一些不是很规律的字符，然后根据我们的数据库进行潜在搜索词的提示，这在我们现实生活中是非常常用的一个技术：

```python
from distance import Levenshtein_Distance

# 为简化起见，我在这里只举一个例子，来进行 wordnet 的模拟

words = ["xia", "xiazi", "xzx", "xiazx"]

def match_rate(word1, word2):
    return float(1 - Levenshtein_Distance(word1, word2) / max(len(word1), len(word2)))

if __name__=='__main__':
    t = 'izx'
    dic = {}
    for w in words:
        print("The candidate word is: " + w + ".The match rate is: ", match_rate(w, t))
        dic.update({w: match_rate(w, t)})

    most_likely = max(dic, key=lambda k: dic[k])
    print("Are you going to search for the keyword: "+ most_likely)
```

给出实验结果如下：

```
PS D:\CS\CSDIY\nlp_xjtu\assignment1> python keyword.py
2
The candidate word is: xia.The match rate is:  0.0
The candidate word is: xiazi.The match rate is:  0.4
The candidate word is: xzx.The match rate is:  0.6666666666666667
The candidate word is: xiazx.The match rate is:  0.6
Are you going to search for the keyword: xzx
PS D:\CS\CSDIY\nlp_xjtu\assignment1>
```

6. 收获与总结

在这次作业中，有如下收获：

1. 复习了动态规划相关算法分析与设计
2. 了解了最小编辑距离的应用场景，并对其中的部分应用进行了简化版的模拟
3. 在python中，想要获得字典中最大值所对应的键，需要借助lambda函数，否则，

python中的max函数默认会按照字典中的key进行比较而不是value。

4. 对于应用场景，我希望之后能有机会接触到真正意义上的wordnet，同时，当数据规模变大时，我也期待自己能优化自己的代码。