自然语言处理NLP作业①

MED 实现

姓名：罗福杰

学号：3120305208

班级：S0078

目录

[1 作业要求：MED实现 3](#_Toc51500475)

[2 问题的数学描述 3](#_Toc51500476)

[2.1 最小编辑距离的概述 3](#_Toc51500477)

[2.2 问题描述 3](#_Toc51500478)

[2.3 数学语言描述 4](#_Toc51500479)

[3 编程实现 5](#_Toc51500480)

[3.1 编程思路 5](#_Toc51500481)

[3.2 核心代码（完整代码见附录） 6](#_Toc51500482)

[3.2.1 实现最小距离的代码 6](#_Toc51500483)

[3.2.2 实现回溯计算的代码 6](#_Toc51500484)

[3.2.3 实现界面开发的核心代码 8](#_Toc51500485)

[3.3 界面设计与实验结果 9](#_Toc51500486)

[参考资料 15](#_Toc51500487)

[附录 16](#_Toc51500488)

[免责与版权 23](#_Toc51500489)

# 1 作业要求：MED实现

作业具体要求：

* 实现英文（或中文）字符串的最小编辑距离，最小编辑路径的计算并显示结果。
* 可以输入不同字符串；可以通过修改操作代价来改变最小编辑距离。
* 需要提交报告、源代码和可执行程序。

# 2 问题的数学描述

## 2.1 最小编辑距离的概述

最小编辑距离（Edit Distance）是指两个字串之间，由一个转成另一个所需的最少编辑操作次数。允许对字符串中的字符进行的操作只有替换、插入、删除三种操作。

编辑距离是自然语言处理中的重要的文本比较算法之一。也是从多个相似的字符串组中提取字符串的有利的武器。编辑距离算法，也称为LD算法。LD算法就是自然语言处理(NLP)里的“编辑距离”算法。俄国科学家Levenshtein提出的，故又叫Levenshtein Distance （LD算法）。

最小编辑距离的用途：

* 计算衡量机器翻译和语音识别的好坏：将机器得到的字符串与专家写的字符串比较最小编辑距离，以一个单词为一个单位。
* 命名实体识别和链接：比如通过计算最小编辑距离，可以判定IBM.Inc和IBN非常相似，只有一个单词不同，所以认为这是指向同一个命名实体。

## 2.2 问题描述

给定两个字符串a和b，如果要将字符串a转换为b，转换过程中只有三种基本操作：

1. 插入（insert）字符；
2. 删除（delete）字符；
3. 替换（replace）字符。

例如从：单词intention通过删除i可以得到ntention，通过插入e可以得到eintention，通过将i换成e可以得到entention。以下是从intention到叶子节点的任意一个单词经过的操作数就是一条路径。

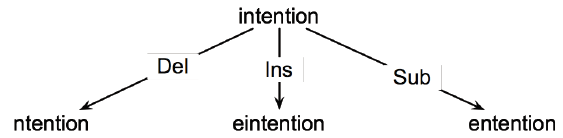


图2.1 字符串编辑示例

可以发现枚举出所有可转变成的单词的花费是十分巨大的，我们不可能用枚举遍历的方式来寻找一条最短路径，一种解决方法是：使用剪枝，每层中有很多路径被剪枝了，只在每一层中保留最短的那条路径

通常每一个操作的cost值为1，则最小编辑距离是字符串编辑距离的优化问题：如何组合这些操作从而使转换的代价cost最小。则可以转化为动态规划的最优值求解问题，下面给出其具体的数学语言描述过程。

## 2.3 数学语言描述

通过以上的分析，将两个字符串a,b的编辑距离Levenshtein Distance表示为可用以下的数学语言描述：

其中：

* 是a,b的长度；定义的是指a中前i个字符和b中的前j个字符之间的距离。这里的字符串的第一个字符index从1开始，因此在最后的编辑距离是i=|a|，j=|b|时的距离：。
* 当的时候，对应着字符串a中前i个字符和b中的前j个字符，此时的i,j有一个值是0，表示字符串a和b中有一个是空的字符串，那么从a转换到b只需要进行max(i,j)次单字符串的编辑操作，所以他们之间的编辑距离为，即i,j中的最大者。
* 当的时候，有三种情况的最小值：

这三种情况是：

1. 表示删除；
2. 表示插入；
3. 表示替换

* 是一个指示函数，表示当的时候取值为0，当的时候取值为1.

# 3 编程实现

## 3.1 编程思路

在完成本次作业过程中，我主要是通过以下三个步骤来实现的：

1. 完成实现最小编辑距离的函数，minDistance()，参数是输入的两个字符串，输出是最小编辑距离和实现这一功能的二维矩阵；
2. 完成回溯的功能函数，backtrackingPath()，参数是两个字符串，输出是实现字符串编辑的具体步骤；
3. 界面开发，为了拥有更友好的用户体验，我开发了基于windows的“最小编辑距离计算器”软件，通过该软件可以实现最小编辑距离的计算和实现的具体步骤。

## 3.2 核心代码（完整代码见附录）

### 3.2.1 实现最小距离的代码

def minDistance(w1,w2):

    w1=w1.strip()

    w2=w2.strip()

    m,n = len(w1),len(w2)

    if (w1=="" or m==0):

        messagebox.showinfo('Warning!!!','请在Word1中输入字符！')

        return m

    if m>255:

        messagebox.showinfo('Warning!!!','字符长度不得超过255！')

        return m

    if (w2=="" or n==0):

        messagebox.showinfo('Warning!!!','请在Word2中输入字符！')

        return n

    if n>255:

        messagebox.showinfo('Warning!!!','字符长度不得超过255！')

        return n

    # 生成全零矩阵，形状是（m+1, n+1）

    #step = [[0]\*(n+1) for \_ in range(m+1)]

    step = np.zeros([m+1,n+1])

    for i in range(1,m+1):

        step[i][0] = i

    for j in range(1,n+1):

        step[0][j] = j

    for i in range(1,m+1):

        for j in range(1,n+1):

            if w1[i-1] == w2[j-1]:

                diff = 0

            else:

                diff = 1

            step[i][j] = min(step[i-1][j-1],min(step[i-1][j],step[i][j-1]))+diff

    return step,int(step[m][n])

### 3.2.2 实现回溯计算的代码

def backtrackingPath(word1,word2):

    dp,mindista = minDistance(word1,word2)

    m = len(dp)-1

    n = len(dp[0])-1

    operation = []

    spokenstr = []

    writtenstr = []

    operation\_process = []

    # 定义用来存放是否为最优值的矩阵

    back\_way = np.zeros([m+1,n+1])

    back\_way[m][n] = 1

    while n>=0 or m>=0:

        if n and dp[m][n-1]+1 == dp[m][n]:

            processer="Insert : \""+(word2[n-1])+'\".'

            operation\_process.append(processer)

            spokenstr.append("Insert")

            writtenstr.append(word2[n-1])

            operation.append("NULLREF:"+word2[n-1])

            n -= 1

            back\_way[m][n] = 1

            continue

        if m and dp[m-1][n]+1 == dp[m][n]:

            processer="Delete : \""+(word1[m-1])+'\".'

            operation\_process.append(processer)

            spokenstr.append(word1[m-1])

            writtenstr.append("Delete")

            operation.append(word1[m-1]+":NULLHYP")

            m -= 1

            back\_way[m][n] = 1

            continue

        if dp[m-1][n-1]+1 == dp[m][n]:

            processer="Replace : \""+(word1[m-1])+'\" To \"'+(word2[n-1])+'\".'

            operation\_process.append(processer)

            spokenstr.append(word1[m - 1])

            writtenstr.append(word2[n-1])

            operation.append(word1[m - 1] + ":"+word2[n-1])

            n -= 1

            m -= 1

            back\_way[m][n] = 1

            continue

        if dp[m-1][n-1] == dp[m][n]:

            spokenstr.append(' ')

            writtenstr.append(' ')

            operation.append(word1[m-1])

        n -= 1

        m -= 1

        back\_way[m][n] = 1

    spokenstr = spokenstr[::-1]

    writtenstr = writtenstr[::-1]

    operation = operation[::-1]

    # print(spokenstr,writtenstr)

    # print(operation)

    return spokenstr,writtenstr,operation,operation\_process,back\_way

### 3.2.3 实现界面开发的核心代码

本软件是用Python中的tkinter框架进行开发的，核心代码如下：

import tkinter as tk

from tkinter import ttk

import numpy as np

from tkinter import messagebox

window = tk.Tk()

window.title("NLP作业①   最小编辑距离计算器 by 罗福杰")

#设置窗口的长和宽不可调节

window.resizable(0,0)

# 设置初始界面的长和高

window.geometry('700x600')

def compute():

    word1=str(tx\_word1.get(1.0, "end"))

    word2=str(tx\_word2.get(1.0, "end"))

    step,mindis=minDistance(word1,word2)

    result\_value.set(mindis)

    btn\_show.place(x=200,y=400)

btn\_compute = tk.Button(window,text ='Start Compute',width=15,height=5,command=compute)

btn\_compute.place(x=30,y=400)

btn\_show = tk.Button(window,text ='Show Operation',width=15,height=5,command=show\_operation)

#btn\_show.place(x=200,y=400)

btn\_show.place\_forget()

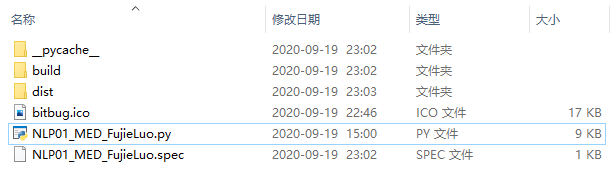
window.mainloop()

## 3.3 界面设计与实验结果

本实验首先由Python编写各个功能的程序，然后用pyinstaller进行打包，将文件打包成可执行文件（.exe文件）

下面是实验的结果：

* **打包后的文件示意图**

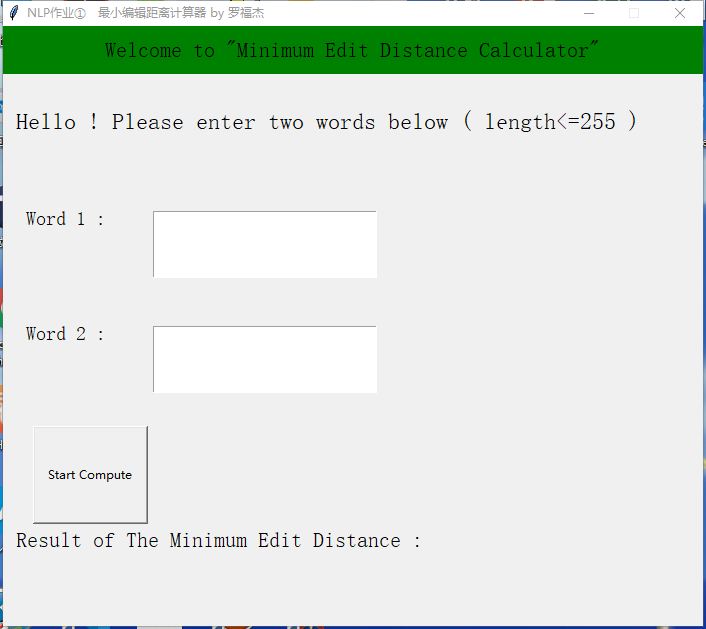


进入dist文件夹后：



* **双击软件即可运行：**

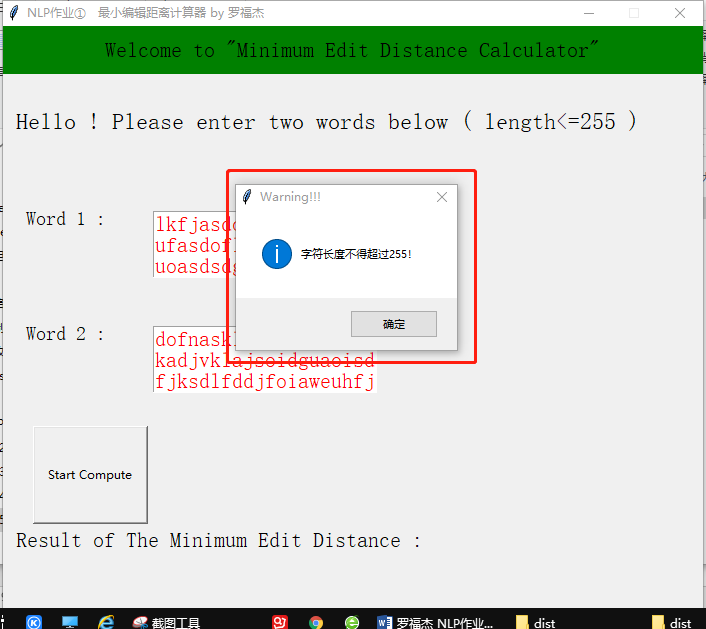
该软件有两个输入框，在里面输入有效的字符才能进行计算，一个字符串的长度不超过255。输入完成后，点击“Start Compute”开始计算。



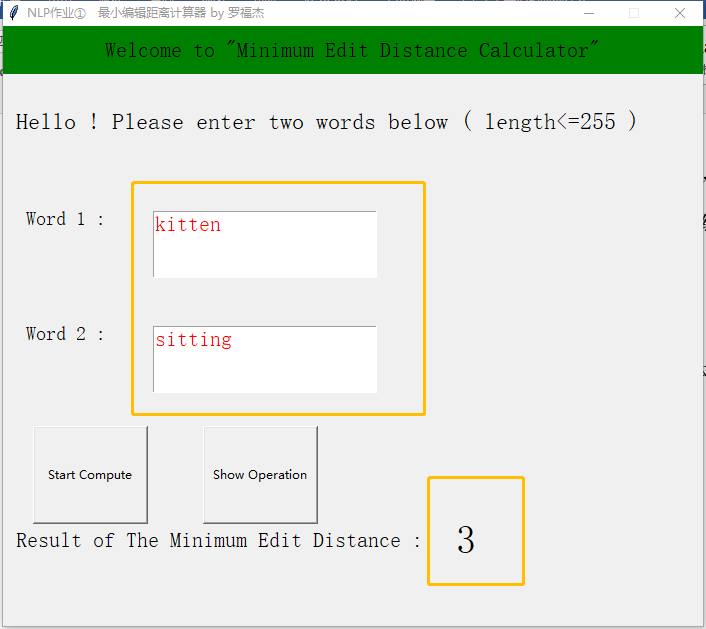
* 当没输入任何信息时，提示输入有效的字符



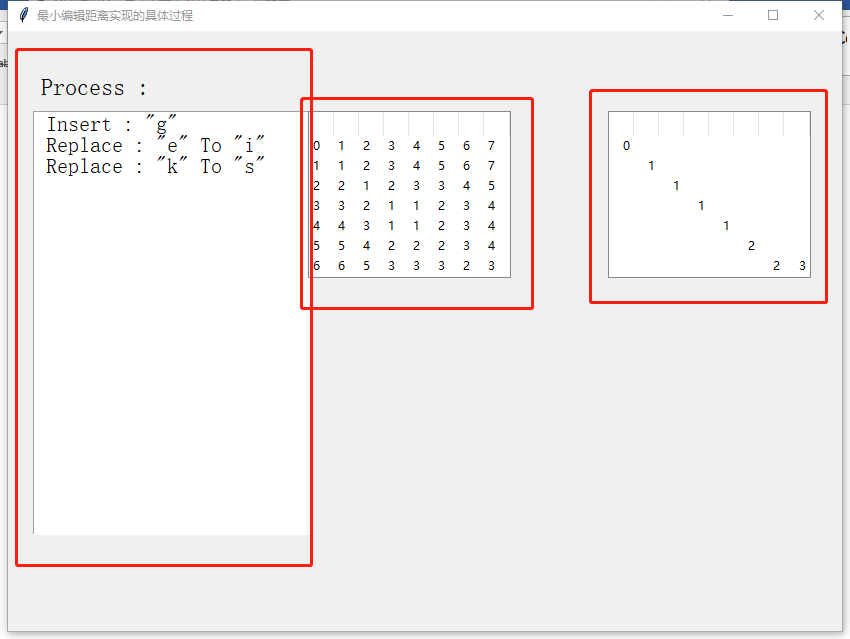
* 当输入的字符数大于255时，提示输入过长



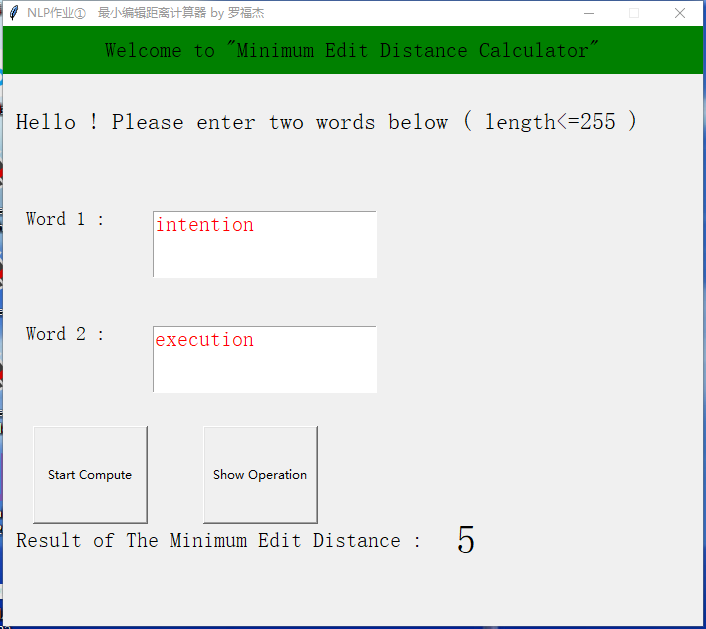
* 当输入有效的字符后，开始计算，并将最小编辑距离打印输出，示例如下：

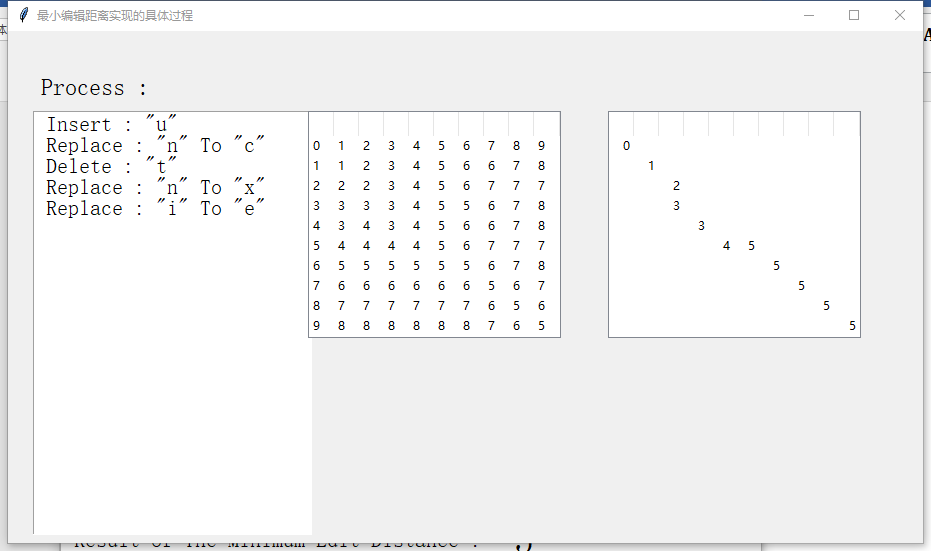


* 这个时候可以查看如何进行编辑操作的，当点击“Show Operation”即可弹出另一个窗口，显示了**如何进行字符编辑**的、**字符编辑表格**和**字符编辑的回溯路线**。如下图中红色框内部分所示，是由字符串“kitten”转换成“sitting”的操作结果。

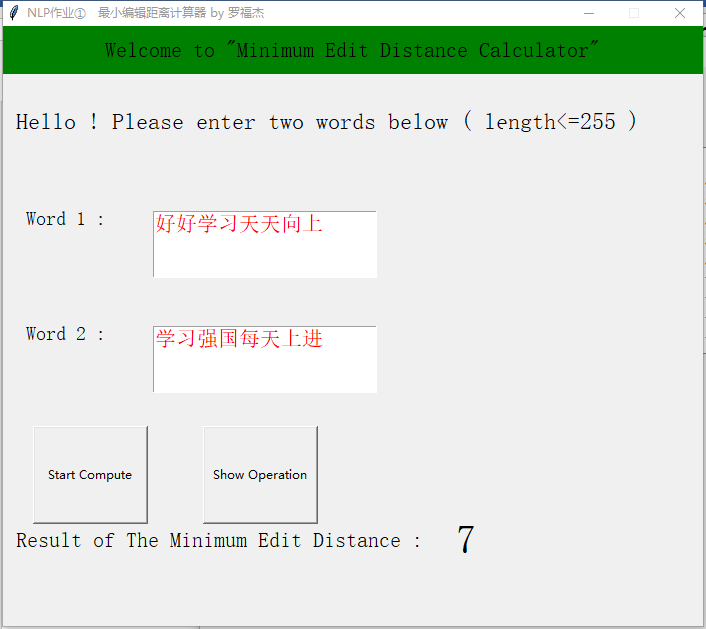


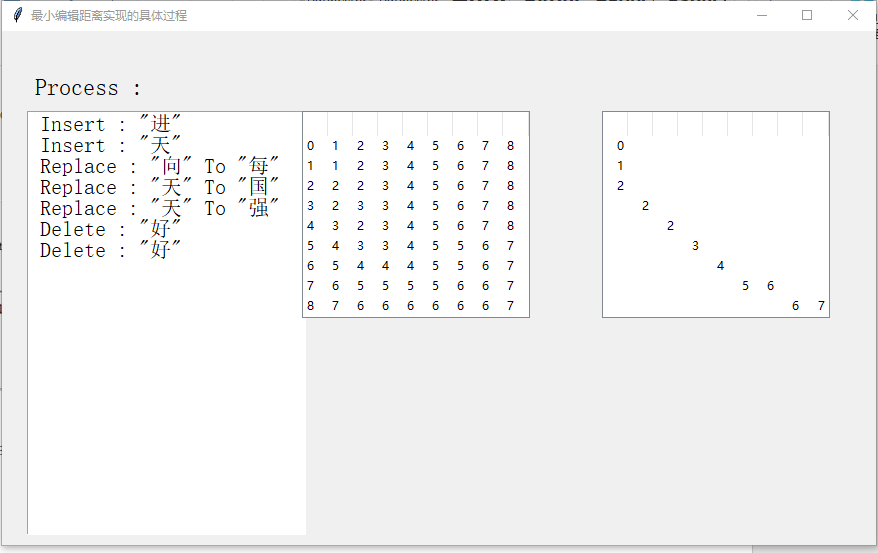
* 示例二的结果：从字符串“intention”转换为“execution”的结果。





* 中文字符示例一个中文汉字可以看作是一个字符，结果如下：





# 参考资料

[1] <https://blog.csdn.net/weixin_40446557/article/details/103456906>

[2] <https://blog.csdn.net/chl0000/article/details/7657887>

[3] <https://blog.csdn.net/weixin_45926547/article/details/108397563>

[4] <https://blog.csdn.net/qq_42382909/article/details/82883237>

[5] <https://blog.csdn.net/vivian_ll/article/details/93168926>

# 附录

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Thu Sep 17 22:29:38 2020

@author: fujie

"""

import tkinter as tk

from tkinter import ttk

import numpy as np

from tkinter import messagebox

window = tk.Tk()

window.title("NLP作业① 最小编辑距离计算器 by 罗福杰")

#设置窗口的长和宽不可调节

window.resizable(0,0)

# 设置初始界面的长和高

window.geometry('700x600')

#lb\_title = tk.Label(window,text='Welcome to "最 小 编 辑 距 离 计 算 器" hhaha',bg='green',font =('Arial',16),width=80,height=2)

lb\_title = tk.Label(window,text='Welcome to "Minimum Edit Distance Calculator"',bg='green',font =('Time New Roman',16),width=80,height=2)

#lb\_title = tk.Label(window,text='欢迎使用最小编辑距离计算器',font =('Arial',18),width=40,height=20)

lb\_title.pack(side='top')

#lb\_title.pack(side='bottom')

lb\_mode = tk.Label(window,text = 'Hello ! Please enter two words below ( length<=255 )',font =('Time New Roman',17))

#lb\_mode.pack(side='left')

#设置标签的位置

lb\_mode.place(x=10,y=80,anchor='nw')

#cmb = ttk.Combobox(window)

#cmb.place(x = 210,y = 85,anchor='nw')

#cmb['value']=('English','中文')

#设置默认值

#cmb.current(0)

lb\_char1=tk.Label(window,text='Word 1 :',font=('Time New Roman',14))

lb\_char1.place(x=20,y=180)

lb\_char2=tk.Label(window,text='Word 2 :',font=('Time New Roman',14))

lb\_char2.place(x=20,y=295)

lb\_result = tk.Label(window,text='Result of The Minimum Edit Distance : ',font=('Time New Roman',16))

lb\_result.place(x=10,y=500)

result\_value = tk.StringVar()

lb\_result\_value = tk.Label(window,textvariable=result\_value,font=('Time New Roman',30))

lb\_result\_value.place(x=450,y=490)

result\_lb = tk.StringVar()

lb\_result\_lb =tk.Label(window,textvariable=result\_lb,font=('Time New Roman',20))

lb\_result\_lb.place(x=500,y=100)

result\_process\_lb=tk.StringVar()

lb\_result\_process=tk.Label(window,textvariable=result\_process\_lb,font=('Time New Roman',20))

lb\_result\_process.place(x=350,y=150)

#entry\_word1=tk.Entry(window,bd=3,xscrollcommand=True)

#entry\_word1.place(x=150,y=185)

#entry\_word2=tk.Entry(window,bd=3)

#entry\_word2.place(x=150,y=245)

f=tk.Frame(window)

s1 = tk.Scrollbar(f,orient=tk.VERTICAL)

tx\_word1=tk.Text(window,autoseparators=2,font=('Time New Roman',16),height=3,width=20 ,fg='red')

tx\_word1.place(x=150,y=185)

tx\_word2=tk.Text(window,autoseparators=2,font=('Time New Roman',16),height=3,width=20, ,fg='red')

tx\_word2.place(x=150,y=300)

def minDistance(w1,w2):

w1=w1.strip()

w2=w2.strip()

m,n = len(w1),len(w2)

if (w1=="" or m==0):

messagebox.showinfo('Warning!!!','请在Word1中输入字符！')

return m

if m>255:

messagebox.showinfo('Warning!!!','字符长度不得超过255！')

return m

if (w2=="" or n==0):

messagebox.showinfo('Warning!!!','请在Word2中输入字符！')

return n

if n>255:

messagebox.showinfo('Warning!!!','字符长度不得超过255！')

return n

# 生成全零矩阵，形状是（m+1, n+1）

#step = [[0]\*(n+1) for \_ in range(m+1)]

step = np.zeros([m+1,n+1])

for i in range(1,m+1):

step[i][0] = i

for j in range(1,n+1):

step[0][j] = j

for i in range(1,m+1):

for j in range(1,n+1):

if w1[i-1] == w2[j-1]:

diff = 0

else:

diff = 1

step[i][j] = min(step[i-1][j-1],min(step[i-1][j],step[i][j-1]))+diff

#return step[m][n]

return step,int(step[m][n])

def backtrackingPath(word1,word2):

dp,mindista = minDistance(word1,word2)

m = len(dp)-1

n = len(dp[0])-1

operation = []

spokenstr = []

writtenstr = []

operation\_process = []

back\_way = np.zeros([m+1,n+1])

back\_way[m][n] = 1

while n>=0 or m>=0:

if n and dp[m][n-1]+1 == dp[m][n]:

processer="Insert : \""+(word2[n-1])+'\".'

operation\_process.append(processer)

spokenstr.append("Insert")

writtenstr.append(word2[n-1])

operation.append("NULLREF:"+word2[n-1])

n -= 1

back\_way[m][n] = 1

continue

if m and dp[m-1][n]+1 == dp[m][n]:

processer="Delete : \""+(word1[m-1])+'\".'

operation\_process.append(processer)

spokenstr.append(word1[m-1])

writtenstr.append("Delete")

operation.append(word1[m-1]+":NULLHYP")

m -= 1

back\_way[m][n] = 1

continue

if dp[m-1][n-1]+1 == dp[m][n]:

processer="Replace : \""+(word1[m-1])+'\" To \"'+(word2[n-1])+'\".'

operation\_process.append(processer)

spokenstr.append(word1[m - 1])

writtenstr.append(word2[n-1])

operation.append(word1[m - 1] + ":"+word2[n-1])

n -= 1

m -= 1

back\_way[m][n] = 1

continue

if dp[m-1][n-1] == dp[m][n]:

spokenstr.append(' ')

writtenstr.append(' ')

operation.append(word1[m-1])

n -= 1

m -= 1

back\_way[m][n] = 1

spokenstr = spokenstr[::-1]

writtenstr = writtenstr[::-1]

operation = operation[::-1]

# print(spokenstr,writtenstr)

# print(operation)

return spokenstr,writtenstr,operation,operation\_process,back\_way

def compute():

#word1=entry\_word1.get()

#word2=entry\_word2.get()

word1=str(tx\_word1.get(1.0, "end"))

word2=str(tx\_word2.get(1.0, "end"))

step,mindis=minDistance(word1,word2)

result\_value.set(mindis)

btn\_show.place(x=200,y=400)

def show\_operation():

compute()

word1=tx\_word1.get(1.0, "end")

word2=tx\_word2.get(1.0, "end")

window\_show=tk.Toplevel(window)

window\_show.geometry('800x600')

window\_show.title('最小编辑距离实现的具体过程')

lb\_show\_Process=tk.Label(window\_show,text='Process : ',font =('Time New Roman',18))

lb\_show\_Process.place(x=30,y=40)

tx\_show=tk.Text(window\_show,autoseparators=2,font =('Time New Roman',16),height=20,width=25)

tx\_show.place(x=25,y=80)

spokenstr,writtenstr,operation,operation\_process,back\_way=backtrackingPath(word1,word2)

#result\_process\_lb.set(operation\_process)

tx\_show.insert(1.0," ")

process=str(operation\_process)

process=process.replace('.\',', '\n')

process=process.replace('.\']', "")

process=process.replace('\'', "")

process=process.replace('[', "")

tx\_show.insert(1.1,process)

stepff,\_=minDistance(word1, word2)

lines,column\_num=stepff.shape[0],stepff.shape[1]

columnskk=[str(int(stepff[0][i])) for i in range(0,column\_num)]

tree = ttk.Treeview(window\_show, show = "headings", columns = columnskk,height=lines, selectmode = tk.BROWSE)

tree["columns"]=columnskk

#设置每一列的属性

for i in range(0,column\_num):

tree.column(columnskk[i], width=25)

# 开始填充数据

for i in range(0,lines):

step\_line\_val=[str(int(stepff[i][j])) for j in range(0,column\_num)]

tree.insert('','end',values=step\_line\_val)

tree.place(x=300,y=80)

back\_way\_only\_1 = np.zeros([lines,column\_num])

for i in range(0,lines):

for j in range(0,column\_num):

back\_way\_only\_1[i][j]=stepff[i][j]\*back\_way[i][j]

back\_way\_only\_1\_str = [['0']\*column\_num for \_ in range(lines)]

for i in range(0,lines):

for j in range(0,column\_num):

if back\_way[i][j]==1:

back\_way\_only\_1\_str[i][j]=str(int(stepff[i][j]))

else:

back\_way\_only\_1\_str[i][j]=' '

back\_way\_only\_1\_str[0][0]='0'

tree2 = ttk.Treeview(window\_show, show = "headings", columns = columnskk,height=lines, selectmode = tk.BROWSE)

tree2["columns"]=columnskk

#设置每一列的属性

for i in range(0,column\_num):

tree2.column(columnskk[i], width=25,anchor='e')

for i in range(0,lines):

tree2.insert('','end',values=back\_way\_only\_1\_str[i])

tree2.place(x=600,y=80)

btn\_compute = tk.Button(window,text ='Start Compute',width=15,height=5,command=compute)

btn\_compute.place(x=30,y=400)

btn\_show = tk.Button(window,text ='Show Operation',width=15,height=5,command=show\_operation)

#btn\_show.place(x=200,y=400)

btn\_show.place\_forget()

window.mainloop()

# 免责与版权

1. 该软件由罗福杰同学（邮箱：1626027173@qq.com）开发与维护，版权归其所有；
2. 该软件仅供学习交流使用，禁止商业用途，如有因学习以外的用途产生的问题，软件开发人不承担法律责任。