#CPSC475 Dr.DePalma Fall 2016 asgn2

#Minimum Edit Distance

#Sebastian Vargas

#id: avargas

#

#To run on linux, go to your terminal then go to the directory in which

# this program is stored. Then type "python asgn2.py" in the command line

#This should execute the program

#This function calculatese the minimum edit distance as well as

#creates and initializes the arrays to form the matrix.

#Also, the backtracing and alignment are called through here

def minEditDistance():

#gets the users input for target and source

sourceString = raw\_input("Input the source string \n")

targetString = raw\_input("Input the target string \n")

#Lines of the matrix

sourceArray = ["#"]

targetArray = ["#"]

#the Distance matrix

nestedArray = []

#List of the tuples that form the back traced path

pathArray = []

backTracedSource = []

backTracedTarget = []

#the identifying rows and columns of the distance matrix are formed

n = len(sourceString)

m = len(targetString)

for i in range(0, n):

sourceArray.append(sourceString[i])

for i in range(0, m):

targetArray.append(targetString[i])

#Distance Matrix is made of lists of lists

for i in range(0,n+1):

nestedArray.append([])

for j in range(0,m+1):

nestedArray[i].append([])

#fill the first row and first column with the initial values

for i in range(0,m+1):

nestedArray[0][i] = i

for i in range(0,n+1):

nestedArray[i][0] = i

#Recurrence relation

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

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

nestedArray[i][j] = min(nestedArray[i-1][j] + delCost(sourceArray[i]),

nestedArray[i-1][j-1] + subCost(sourceArray[i],targetArray[j]),

nestedArray[i][j-1] + insCost(targetArray[j]))

#printArray(nestedArray, sourceArray)

print("Minimum Edit distance: " + str(nestedArray[n][m]))

computeAlignment(nestedArray,pathArray,sourceString,targetString)

backTracePrint(nestedArray,pathArray,sourceString,

targetString,sourceArray,targetArray)

return

#This function computes the letter alignment of the source and target

def computeAlignment(nestedArray,pathArray,sourceString,targetString):

n = len(sourceString)

m = len(targetString)

#loc is a variable that keeps track of the current tuple the path would be on

loc = [n,m] #starts at the location of the minimum edit distance

pathArray.append(loc)

i = n

j = m

#pathval simply represents the values in the matrix

pathVal = nestedArray[n][m]

#Finds the minimum value in relation to the neighbours of the current path

while loc != [0,0]:

pathVal = min(nestedArray[i-1][j],nestedArray[i-1][j-1],nestedArray[i][j-1])

if (pathVal == nestedArray[i-1][j]):

loc = [i-1,j]

i = i - 1

elif (pathVal == nestedArray[i-1][j-1]):

loc = [i-1,j-1]

i = i - 1

j = j - 1

elif (pathVal == nestedArray[i][j-1]):

loc = [i,j-1]

j = j - 1

pathArray.append(loc)

#print("loc: " + str(loc)) #Use to debug

return

#prints a distance matrix

def printArray(array, sourceArray):

print("Distance Matrix: ")

for i in range(0, len(sourceArray)):

print(array[i])

return

#delete cost for M.E.D (Minimum Edit Distance)

def delCost(letter):

if (letter != ' '):

return 1

return 0

#ins cost for M.E.D (Minimum Edit Distance)

def insCost(letter):

if (letter != ' '):

return 1

return 0

#substitute cost for M.E.D (Minimum Edit Distance)

def subCost(srcLetter,tarLetter):

if (srcLetter == tarLetter):

return 0

return 2

#This function traces the path back and finds the tuples the

#Minimum Edit Distance traverses to align the letters

def backTracePrint(nestedArray,pathArray,sourceString,

targetString, sourceArray, targetArray):

#Arrays to hold characters for the source and target

backTracedSource = []

backTracedTarget = []

operationsDone = []

n = len(sourceString)

m = len(targetString)

sourceCounter = n

targetCounter = m

#print("Back Traced Path: " + str(pathArray))

#This loop generates the arrays for

#source and the target to be aligned

# '\*' characters indicate deletion or insertion

for i in range (0,len(pathArray)-1):

#if substitution is done, add letters to both strings

if (pathArray[i][0] > pathArray[i + 1][0] and pathArray[i][1] > pathArray[i + 1][1]):

backTracedSource.append(sourceArray[sourceCounter] + " ")

backTracedTarget.append(targetArray[targetCounter] + " ")

sourceCounter = sourceCounter - 1

targetCounter = targetCounter - 1

operationsDone.append("ins")

#Insertion adds \* to the source

elif (pathArray[i][1] > pathArray[i + 1][1]):

operationsDone.append("sub")

backTracedSource.append("\* ")

backTracedTarget.append(targetArray[targetCounter] + " ")

targetCounter = targetCounter - 1

#Deletion adds \* to the target

elif (pathArray[i][0] > pathArray[i + 1][0]):

operationsDone.append("del")

backTracedTarget.append("\* ")

backTracedSource.append(sourceArray[sourceCounter] + " ")

sourceCounter = sourceCounter - 1

#reverse the arrays because they are done from the last letter to the first

backTracedSource.reverse()

backTracedTarget.reverse()

print("source : " + str(backTracedSource))

print("target : " + str(backTracedTarget))

print("operations: " + str(operationsDone))

return

def main():

minEditDistance()

return

main()