Drew Lickman
CSCI 4820-001
Project #1
Due: 9/9/24

Minimum Distance Edit Algorithm

Assignment Requirements:

Input

- words.txt is the input, which holds **lowercase** sets of words on each line.
  - The first word of each line is the target, and all the other words in the line are source words that will transform into the target
- Sample input files
  - costs.csv uses Levenshtein substitution costs
  - costs2.csv uses confusion matrix substitution costs

Processing

- Insertions and Deletions cost 1
- Substitution costs are read from the costs.csv files
- For each pair of source and target words, use the Minimum Edit Distance algorithm (using both cost methods)
  - Then output the backtrace of operations (K I S D)
  - Must be able to capture all possible sources for the minimum cost at each cell
  - Randomly select one of the possible cells that provide the minimum cost
  - Do NOT seed random number generator

Output

- 4 lines per method (costs and costs2)

1. Source word
2. Vertical bar for each operation per character
3. Target word
4. Operation for each character and sum of edit cost
   - k = keep
   - i = insert
   - s = substitute
   - d = delete

- 50 hyphens will separate a pair of words from the next pair

Python Code

In [157…

```
# This block of code processes words.txt and defines the targets and sources
targets = []
sources = []
pairTargSources = []

costsFileName = "costs.csv"
costs2FileName = "costs2.csv"
costMethods = [costsFileName, costs2FileName]

# Read & save the target and source words from words.txt
with open("words.txt") as wordList:
    lines = wordList.readlines()

    for line in lines:
        currentLine = line.split()      # Split the line into words
        targets.append(currentLine[0])  # Target word is first in each line
        sources.append(currentLine[1:]) # Source words are everything after the first word of the line
                                        # Also, the [1:] saves it as an array, even if it's just one source wor
```

```
for row in range(len(targets)):
    pairTargSources.append([targets[row], sources[row]]) # Explicitly pair the sources to their target

print("targets:", targets)
print("sources:", sources)
print("\nTargets with matching sources: ")
for pair in pairTargSources:
    print(pair)
```

```
targets: ['mischievous', 'execution', 'accommodate', 'definitely', 'separate', 'occurrence', 'receive', 'tagctat
cacgaccgcggtcgatttgcccgac']
sources: [['mischief', 'devious'], ['intention'], ['accomodate', 'acommodate', 'acomodate', 'accommadate', 'acom
madate'], ['definately', 'defiantly', 'definatly', 'defenitely', 'definetly', 'deffinitely'], ['seperate', 'sepa
rat', 'seperete', 'seperrate'], ['occurence', 'occurance', 'occurrance', 'ocurrence', 'occorence', 'occourence']
, ['recieve', 'receve', 'recyeve', 'receeve', 'recivve'], ['aggctatcacctgacctccaggccgatgccc']]

Targets with matching sources:
['mischievous', ['mischief', 'devious']]
['execution', ['intention']]
['accommodate', ['accomodate', 'acommodate', 'acomodate', 'accommadate', 'acommadate']]
['definitely', ['definately', 'defiantly', 'definatly', 'defenitely', 'definetly', 'deffinitely']]
['separate', ['seperate', 'separat', 'seperete', 'seperrate']]
['occurrence', ['occurence', 'occurance', 'occurrance', 'ocurrence', 'occorence', 'occourence']]
['receive', ['recieve', 'receve', 'recyeve', 'receeve', 'recivve']]
['tagctatcacgaccgcggtcgatttgcccgac', ['aggctatcacctgacctccaggccgatgccc']]
```

In [158…

```
# This block of code reads the CSV files and calculates the cost of substituting letters

alphabet = {letter: index for index, letter in enumerate("abcdefghijklmnopqrstuvwxyz")}
substitutionCost = []
substitutionCost2 = []
# Function able to work with costs.csv and costs2.csv
def readCostFromCSV(file, substitutionTable):
    with open(file) as costList:
        costLines = costList.readlines()

        for costLine in costLines:
            currentCostLine = costLine.split(",") # Split each value by commas
            substitutionTable.append(currentCostLine) # Save substitution cost to array, indexed by 2D alphabet
        substitutionTable.pop(0) # Remove the first line of cost.csv

        # Cleanup
        for letter in substitutionTable:
            letter.pop(0) # Remove the letter from each cost array

    # Accessible with alphabet dictionary
    return substitutionTable # Return the entire cost 2D array

costsCSV = readCostFromCSV(costsFileName, substitutionCost)
costs2CSV = readCostFromCSV(costs2FileName, substitutionCost2) #Manual, since it's only two methods
costCSVs = [costsCSV, costs2CSV]

# Returns either Levenshtein cost or Confusion Matrix cost
def getCostFromCSV(csv, letter1, letter2):
    intCost = int(csv[alphabet[letter1]][alphabet[letter2]]) # Calculate the specific substitution cost
    return intCost

print("costsCSV a -> i: ", getCostFromCSV(costsCSV, "a", "i")) # "a" substitution TO "i" returns 2
print("costs2CSV a -> i: ", getCostFromCSV(costs2CSV, "a", "i")) # "a" substitution TO "i" returns 118
```

```
costsCSV a -> i:  2
costs2CSV a -> i:  118
```

In [158…

```
# Helper function

def operation(string, file, letter1, letter2):
    match(string):
        case "keep":
            return 0
        case "insert":
            return 1
        case "delete":
            return 1
        case "substitute":
            if (file is not None) and (letter1 is not None) and (letter2 is not None):
                cost = getCostFromCSV(file, letter1, letter2)
                return cost
            return -100
        case _:
            return -200

print("Substitution: ", operation("substitute", costsCSV, "a", "i")) #debug
```

```
Substitution:  2
```

In [158…

```
# Minimum Edit Distance Algorithm

#TestTable = []

def MED(source, target, costMethod):
    #save the source and target words for usage in backtrace
    global savedSource, savedTarget
    savedSource = source
    savedTarget = target

    sourceLen = len(source)
    targetLen = len(target)
    #print("Table dimensions: ", sourceLen, targetLen)
    DistTable = [[0 for i in range(targetLen+1)] for j in range(sourceLen+1)] # 2D array the size of source leng

    # Initialize table size and number axies
    for i in range(1, sourceLen+1):
        DistTable[i][0] = DistTable[i-1][0] + operation("delete", "", "", "") #cols
    for j in range(1, targetLen+1):
        DistTable[0][j] = DistTable[0][j-1] + operation("insert", "", "", "") #rows

    # Recurrence relation:
    for row in range(1, sourceLen+1):
        for col in range(1, targetLen+1):
            # How will I know which operation is performed?
            tryDelete = DistTable[row-1][col] + operation("delete", "", "", "")#delete
            trySubstitute = DistTable[row-1][col-1] + operation("substitute", costMethod, source[row-1], target
            tryInsert = DistTable[row][col-1] + operation("insert", "", "", "")#insert

            DistTable[row][col] = min(
                            tryDelete,
                            trySubstitute,
                            tryInsert)


    if False: #debug: displays visual table
        # Print table
        print("      ", end="")
        # Print the target on top
        for letter in target:
            print(letter, end="  ")
        print()
        # print the source on the left
        for row in range(len(DistTable)):
            if row != 0:
                print(source[row-1], end="")
            else:
                print(" ", end="")
            # print the minimum edit distance cost table
            for value in DistTable[row]:
                print(f'{value : >3}', end="") # Padding of 3, aligned right
            print()

    return DistTable # Returns the minimum edit distance value at the bottom right of the table

#file = costsCSV
#TestTable = MED("devious", "mischievous", file) #Debug testing
```

In [158…

```
# Backtracing

import random

backtracePathPositions = []  # (x,y)
backtracePathOperations = [] # ("d", "i", "s", "k")

def shortestPath(MEDTable):
    return MEDTable[-1][-1] # Return the bottom right cell cost, which is the shortest distance

def backtrace(MEDTable):
    # Make sure backtrace arrays are emptied
    backtracePathPositions = []
    backtracePathOperations = []

    if False: #Debug viewer
        for row in MEDTable:
            for cell in row:
                print(f'{cell : >2}', end=" ")
            print()
        print(savedSource, savedTarget)

    x = len(MEDTable[0])-1
    y = len(MEDTable)-1
```

```
    while (x > -1 or y > -1): #while current position is not at the origin (start from bottom right of table)
        # x,y will end at -1,-1 becuase the strings end at 0,0
        # using or because table edges need to be processed
        currentTargetLetter = x-1
        currentSourceLetter = y-1

        randomQueue = []

        # Cell positions [x,y]
        currentCell = MEDTable[y][x]

        diagonalCell = MEDTable[y-1][x-1]
        horizontalCell = MEDTable[y][x-1]
        verticalCell = MEDTable[y-1][x]


        if False: # Debug to view every step
            print("XY: ", x, y)

            print("Curr letter pos: ", currentSourceLetter, currentTargetLetter)
            print("Curr letter s t: ", savedSource[currentSourceLetter], savedTarget[currentTargetLetter])

            print(diagonalCell, verticalCell)
            print(horizontalCell, currentCell)

            print("same?", savedSource[currentSourceLetter] == savedTarget[currentTargetLetter],\
                ", diag-2?", currentCell-2 == diagonalCell,\
                ", horz-1?", currentCell-1 == horizontalCell,\
                ", vert-1?", currentCell-1 == verticalCell)

        sameLetter = savedSource[currentSourceLetter] == savedTarget[currentTargetLetter] #only evaluate once
        if sameLetter and x > 0 and y > 0: #if diag == same letter, append diag to backtracePath and move on
            randomQueue.append("k")
        # If we're traversing the border
        elif x == 0 and y > 1:
            randomQueue.append("d")
        elif x > 1 and y == 0:
            randomQueue.append("i")
        elif not sameLetter: #and not on edge
            if currentCell-2 == diagonalCell or currentCell-1 == diagonalCell or currentCell == diagonalCell:
                randomQueue.append("s")
            if currentCell-1 == horizontalCell:
                randomQueue.append("i")
            if currentCell-1 == verticalCell:
                randomQueue.append("d")

        # End loop early so I don't do random.choice(null) and get error
        #print("End at 0,0:", currentSourceLetter, currentTargetLetter)
        #if currentSourceLetter == 0 and currentTargetLetter == 0:
        if x == 0 and y == 0: # not sure why but this makes the program run correctly
            trueBacktracePathOperations = backtracePathOperations[::-1] # Reverse array
            #print(backtracePathPositions)
            #print(backtracePathOperations)
            return trueBacktracePathOperations

        # Choose random path
        backtracePathPositions.append((x,y))
        randomChoice = random.choice(randomQueue) #choose random path
        #print("RandomQueue: ", randomQueue, "Choice: ", randomChoice)
        backtracePathOperations.append(randomChoice)

        # Traverse backtrace
        if randomChoice == "k" or randomChoice == "s":
            #print("keep/substitute", savedTarget[currentTargetLetter])
            x-=1
            y-=1
        elif randomChoice == "d":
            #print("insert", savedTarget[currentTargetLetter])
            y-=1
        elif randomChoice == "i":
            #print("delete", savedSource[currentSourceLetter])
            x-=1

    print()

    # Always print debug info IF the loop fails
    print("XY: ", x, y)

    print("Curr letter pos: ", currentSourceLetter, currentTargetLetter)
    print("Curr letter s t: ", savedSource[currentSourceLetter], savedTarget[currentTargetLetter])

    print(diagonalCell, verticalCell)
    print(horizontalCell, currentCell)
```

```
    print("same?", savedSource[currentSourceLetter] == savedTarget[currentTargetLetter])
    print("diag-2?", currentCell-2 == diagonalCell)
    print("horz-1?", currentCell-1 == horizontalCell)
    print("vert-1?", currentCell-1 == verticalCell)

    print(savedSource, savedTarget)
    print(backtracePathPositions)
    print(backtracePathOperations)
    return -1 #prevents output loop from continuing

#backtrace(TestTable) #Debug testing
#shortestPath()
```

In [158…

```
# This block of code outputs the results of the targets and sources

for pair in pairTargSources: # mischevious, execution
    #print("TARGET: ", pair[0])
    for sourceList in pair[1:]: # [mischief, devious], [intention]
        #print("SOURCELIST: ", sourceList, end="\n")
        for singleSource in sourceList: # mischief, devious
            #print("SOURCE WORD: ", singleSource)
            for csv in costCSVs: # costsFileName, costs2FileName
                #print("Levenshtein" if costMethod == costsFile else "Confusion Matrix")
                MEDTable = MED(singleSource, pair[0], csv)
                backtraceOperations = backtrace(MEDTable) # Each cost method has to do a backtrace
                if backtraceOperations == -1:
                    print("Backtrace error")
                    break
                print()

                # print source word
                sourceIndex = 0
                for oper in backtraceOperations:
                    if oper == "i":
                        print("* ", end="")
                    elif sourceIndex <= len(singleSource):
                        print(singleSource[sourceIndex], "", end="")
                        sourceIndex += 1

                print()

                # print vertical bars
                for oper in range(len(backtraceOperations)):
                    print("| ", end="")

                print()

                # print target word
                targetIndex = 0
                for oper in backtraceOperations:
                    if oper == "d":
                        print("* ", end="")
                    elif targetIndex <= len(pair[0]):
                        print(pair[0][targetIndex], "", end="")
                        targetIndex += 1

                print()

                # print backtrace operations and shortest path
                for oper in range(len(backtraceOperations)):
                    print(backtraceOperations[oper], end=" ")
                print("(",shortestPath(MEDTable),")")
                print()

            # Print 50 hyphens to separate source words
            print("-" * 50)
```

```
m * i * * s c h i e f
| | | | | | | | | |
m i s c h i e v o u s
k i s i i s s s s s s ( 5 )


m i s c h i * * e * f
| | | | | | | | | |
m i s c h i e v o u s
k k k k k k i i s i s ( 3 )

--------------------------------------------------

* * * * * d e v i o u s
| | | | | | | | | | |
```

```
m i s c h i e * v o u s
i i i i i s k d s k k k ( 8 )


* * * d e * v i o u s
| | | | | | | | | | |
m i s c h i e v o u s
i i i s s i s s k k k ( 4 )

-----------------------------------------------

i n * t e n t i o n
| | | | | | | | | |
* e x e c u t i o n
d s i s s s k k k ( 8 )


* i n t e n t i o n
| | | | | | | | | |
e x * e c u t i o n
i s d s s s k k k ( 4 )

-----------------------------------------------

a * c c o m o d a t e
| | | | | | | | | | |
a c c o m m o d a t e
k i k s s k k k k k ( 1 )


a c c o * m o d a t e
| | | | | | | | | | |
a c c o m m o d a t e
k k k k i k k k k k ( 1 )

-----------------------------------------------

a * c o m m o d a t e
| | | | | | | | | | |
a c c o m m o d a t e
k i k k k k k k k k ( 1 )


a * c o m m o d a t e
| | | | | | | | | | |
a c c o m m o d a t e
k i k k k k k k k k ( 1 )

-----------------------------------------------

* a c * o m o d a t e
| | | | | | | | | | |
a c c o m m o d a t e
i s k i s k k k k k ( 2 )


a * c * o m o d a t e
| | | | | | | | | | |
a c c o m m o d a t e
k i k i s k k k k k ( 2 )

-----------------------------------------------

a c c o m m a d a t e
| | | | | | | | | | |
a c c o m m o d a t e
k k k k k k s k k k ( 2 )


a c c o * m m a d a t e
| | | | | | | | | | | |
a c c o m m o * d a t e
k k k k i k s d k k k ( 2 )

-----------------------------------------------

* a c * o m m a d a t e
| | | | | | | | | | |
a c c o m m o * d a t e
i s k i s k s d k k k ( 3 )


a c o m m a * d a t e
```

```
| | | | | | | | | | |
a c c o m m o d a t e
k k s s k s i k k k k ( 2 )

------------------------------------------------

d e f i n a * t e l y
| | | | | | | | | | |
d * e f i n i t e l y
k d s s s s i k k k k ( 2 )


d e f i n a * t e l y
| | | | | | | | | | |
d e * f i n i t e l y
k k d s s s i k k k k ( 2 )

------------------------------------------------

d e f i a n * t * l y
| | | | | | | | | | |
d e f i * n i t e l y
k k k k d k i k i k k ( 3 )


d e f i a * n * t l y
| | | | | | | | | | |
d e f i * n i t e l y
k k k k d i s i s k k ( 3 )

------------------------------------------------

d e f i n a * * t l y
| | | | | | | | | | |
d e f * i n i t e l y
k k k d s s i i s k k ( 3 )


d e f i * n a t * l y
| | | | | | | | | | |
d e f i n i * t e l y
k k k k i s d k i k k ( 3 )

------------------------------------------------

d e f e n i t e l y
| | | | | | | | | |
d e f i n i t e l y
k k k s k k k k k k ( 2 )


d e f e n i t e l y
| | | | | | | | | |
d e f i n i t e l y
k k k s k k k k k k ( 2 )

------------------------------------------------

d e f i n * * e t l y
| | | | | | | | | | |
d e f i n i t * e l y
k k k k k i i d s k k ( 3 )


d e f i n * e * t l y
| | | | | | | | | | |
d e f i n i * t e l y
k k k k k i d i s k k ( 3 )

------------------------------------------------

d e f f i n i t e l y
| | | | | | | | | | |
* d e f i n i t e l y
d s s k k k k k k k k ( 1 )


d e f f i n i t e l y
| | | | | | | | | | |
* d e f i n i t e l y
d s s k k k k k k k k ( 1 )

------------------------------------------------
```

```
s e p e r a t e
| | | | | | | |
s e p a r a t e
k k k s k k k k ( 2 )


s e p * e r a t e
| | | | | | | | |
s e p a * r a t e
k k k i d k k k k ( 2 )
```

```
s e p a r * a t
| | | | | | | |
s e p a r a t e
k k k k k i s s ( 1 )


s e p a r a t *
| | | | | | | |
s e p a r a t e
k k k k k k k i ( 1 )
```

```
s e p * e r e t e
| | | | | | | | |
s e p a * r a t e
k k k i d k s k k ( 4 )


s * e p e r e * t e
| | | | | | | | | |
s e p a * r * a t e
k i s s d k d i k k ( 4 )
```

```
s e p e * r r a t e
| | | | | | | | | |
s e * p a * r a t e
k k d s i d k k k k ( 3 )


s e p e r r a t e
| | | | | | | | |
s e p * a r a t e
k k k d s k k k k ( 1 )
```

```
o * c c u r e n c e
| | | | | | | | | |
o c c u r r e n c e
k i k s s k k k k k ( 1 )


o c c u * r e n c e
| | | | | | | | | |
o c c u r r e n c e
k k k k i k k k k k ( 1 )
```

```
* o c c u r a * n c e
| | | | | | | | | | |
o c c u r r * e n c e
i s k s s k d i k k k ( 3 )


o c c u * r a n c e
| | | | | | | | | |
o c c u r r e n c e
k k k k i k s k k k ( 2 )
```

```
o c c u r r a * n c e
| | | | | | | | | | |
o c c u * r r e n c e
k k k k d k s i k k k ( 2 )
```

```
o c c u r r a n c e
| | | | | | | | | |
o c c u r r e n c e
k k k k k k s k k k ( 2 )
```

```
* o c u r r e n c e
| | | | | | | | | |
o c c u r r e n c e
i s k k k k k k k k ( 1 )
```

```
o * c u r r e n c e
| | | | | | | | | |
o c c u r r e n c e
k i k k k k k k k k ( 1 )
```

```
o c c * o r e n c e
| | | | | | | | | |
o c c u r r e n c e
k k k i s k k k k k ( 3 )
```

```
o c c o * r e n c e
| | | | | | | | | |
o c c u r r e n c e
k k k s i k k k k k ( 3 )
```

```
o c c o u r e n c e
| | | | | | | | | |
o c c u r r e n c e
k k k s s k k k k k ( 2 )
```

```
* o c c o u r e n c e
| | | | | | | | | | |
o c c u * r r e n c e
i s k s d s k k k k k ( 2 )
```

```
r e c i e * v e
| | | | | | | | |
r e * c e i v e
k k d s k i k k ( 2 )
```

```
* r e c i e v e
| | | | | | | | |
r e c e i * v e
i s s s k d k k ( 2 )
```

```
r e c * e v e
| | | | | | | |
r e c e i v e
k k k i s k k ( 1 )
```

```
r e c * e v e
| | | | | | | |
r e c e i v e
k k k i s k k ( 1 )
```

```
* r e c y e v e
| | | | | | | | |
r e c e * i v e
i s s s d s k k ( 2 )
```

```
r e c y e * v e
| | | | | | | | |
r e c * e i v e
```

```
k k k d k i k k ( 2 )

------------------------------------------------

r e c e * e v e
| | | | | | | |
r e c e i * v e
k k k k i d k k ( 2 )


r e c e * e v e
| | | | | | | |
r e c e i * v e
k k k k i d k k ( 2 )

------------------------------------------------

r e c i * v v e
| | | | | | | |
r e c * e i v e
k k k d i s k k ( 2 )


r e c i v * v e
| | | | | | | |
* r e c e i v e
d s s s s i k k ( 2 )

------------------------------------------------

* a g g c t a t c a c c t g a c c t c c a g * g c * c g * * * a t g c c * * c
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
t a * g c t a t * * c a c g * * * a c c * g c g g t c g a t t t g c c c g a c
i k d k k k k k d d k s s k d d d s k k d k i k s i k k i i i s s s k k i i k ( 15 )


a * g g c t a t c a c c t g a c c t c * c a g * g c c g a * * t * g * c * c c
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
* t a g c t a t c a * c * g a * c * c g c g g t * * c g a t t t g c c c g a c
d i s k k k k k k k d k d k k d k d k i k s k i d d k k k i i k i s i k i s k ( 15 )

------------------------------------------------
```