Author: Carlos Fernando Castaneda

Class: CS 2302

Date Modified: December 1, 2019

Instructor: Diego Aguirre

Assignment: Lab 7 Edit Distance

TA: Gerardo Barraza

**Introduction:**

In programming, there is a specific type of problem-solving technique called Dynamic Programming that essentially splits the problem into smaller problems and stores these solutions in its memory so it may be used later to solve a bigger problem. Edit-distance is a type of dynamic programming that determines how dissimilar two strings are from each other by counting how many operations it will need for both of them to be the same. The job in this lab is to implement this algorithm using a separate file to contain my test cases.

**Proposed solution design and implementation:**

The implementation of this lab is pretty straightforward for me, if the length of the first string is zero, return the length of the second string, vice versa on the second string. This will help get the answer quicker in case one of the strings is empty. Then I will compare the first element of the each string and see if they are similar, if they aren’t, that’s when the operations will begin, either insert, remove, and replace. When everything is done, it will count how many operations it was necessary for the two strings to become the same.

Also, I want to add another section on the opening menu where the user can choose which two words they want to implement so that they can see how many operations it takes for those words to become the same.

**Experimental results**:

The implementation was exactly the same as above, there were some tweaks to be made to create the necessary count for both strings.

The test cases I decided to implement were made in mind to illustrate how the algorithm works. In test case 1, I choose to analyze “brand” and “random”; these two strings were present in class last week, and I choose this as my first case to test if the algorithm worked or not, obviously it worked as the count was returned as 3, which is the answer. In test case 2, I choose to analyze “any” and “any”; these two strings were the same to purposely return 0 and to prove that the algorithm will only return the necessary count needed to change the content of a string. In test case 3, I choose to analyze “magnificus” and “”; the second string is blank on purpose to show how many that my test cases mentioned above will work without any problem, even when dealing with empty strings.

As mentioned above, I added a section in the user interface that will allow them to implement two different words that the previous test cases that are already present in the algorithm.

My results from running option the program, saying yes, and choosing to compare apple and pie :

**A screenshot of a social media post

Description automatically generated**

**Run times (Custom Words):**

|  |  |
| --- | --- |
| **Attempt** | **Time** |
| **1 (pi, py)** | **0.0 seconds** |
| **2 (captain america, iron man)** | **2.0564 seconds** |
| **3 (coca cola, pepsi)** | **0.0129 seconds** |

**Graph representation:**

**Conclusions**:

With this lab, I was able to learn to code better using the Python language, including using multiple python files to replicate the edit distance dynamic programming as well as its necessary implementations. I was able to implement methods that act as the basic structure for the algorithm, as well as the necessary experiments to portray its capabilities.

**Appendix :**

**Main.py**

'''

Author: Carlos Fernando Castaneda

Class : CS 2302

Date Modified: November 26, 2019

Instructor: Diego Aguirre

Assingment: Lab 7 Edit Distance (1 0f 2)

TA: Gerardo Barraza

Purpose: To practice using dynamic porgramming

to implement te edit distance method.

'''

#Used to calculate the time for each option on the lab

import time

#Imports the edit\_distance program needed to implement the graphs

import edit\_distance as ed

def main():

print("Welcome to the Edit Distance Program!")

print("")

print("This program will perform three different tests. Let's get started!")

print("")

print("Test 1: ")

#Test case 1

string\_1 = "brand"

string\_2 = "random"

print("The two words being compared are '",string\_1,"' and '",string\_2,"'.")

start1 = time.time()

print(ed.edit\_distance(string\_1, string\_2, len(string\_1), len(string\_2)))

end1 = time.time()

print('Running time for test 1 was: ', end1 - start1, 'seconds.')

print("")

print("Test 2: ")

#Test case 2

string\_1 = "any"

string\_2 = "any"

print("The two words being compared are '",string\_1,"' and '",string\_2,"'.")

start2 = time.time()

print(ed.edit\_distance(string\_1, string\_2, len(string\_1), len(string\_2)))

end2 = time.time()

print('Running time for test 1 was: ', end2 - start2, 'seconds.')

print("")

print("Test 3: ")

#Test case 3

string\_1 = "magnificus"

string\_2 = ""

print("The two words being compared are '",string\_1,"' and '",string\_2,"'.")

start3 = time.time()

print(ed.edit\_distance(string\_1, string\_2, len(string\_1), len(string\_2)))

end3 = time.time()

print('Running time for test 1 was: ', end3 - start3, 'seconds.')

print("")

print("Tests completed!")

print("")

print("Would you like to try two words for yourself? Yes or No")

user\_selection = input()

if(user\_selection == 'yes' or user\_selection == 'Yes' or user\_selection == 'YES'):

print("Excellent! Now, which two words would you like to try?")

print("Word 1: ")

string\_1 = input()

print("Word 2: ")

string\_2 = input()

print("The two words being compared are '",string\_1,"' and '",string\_2,"'.")

start4 = time.time()

print(ed.edit\_distance(string\_1, string\_2, len(string\_1), len(string\_2)))

end4 = time.time()

print('Running time for test 1 was: ', end4 - start4, 'seconds.')

print("")

print("Program complete! See you again soon!")

elif(user\_selection == 'no' or user\_selection == 'No' or user\_selection == 'NO'):

print("Very well.")

print("Program complete! See you again soon!")

else:

print("ERROR! Input invalid!")

print("Goodbye!")

main()

**edit\_distance.py**

'''

Author: Carlos Fernando Castaneda

Class : CS 2302

Date Modified: November 26, 2019

Instructor: Diego Aguirre

Assingment: Lab 7 Edit Distance (2 0f 2)

TA: Gerardo Barraza

Purpose: To practice using dynamic porgramming

to implement te edit distance method.

'''

#Method that alters two given strings based on the rules of edit distance

def edit\_distance(string\_1, string\_2, len1 , len2):

# Will check if the length of the first string is none. If so, it will return the length of the second string

if len1 == 0:

return len2

# Will check if the length of the second string is none. If so, it will return the length of the first string

if len2 == 0:

return len1

#Will check if the last characters of both strings are the same; if so, it will return the count of the remaining characters and will ignore the duplicated ones

if string\_1[len1-1] == string\_2[len2-1]:

return edit\_distance(string\_1,string\_2,len1-1,len2-1)

#Returns one for the counter indicating that the last elements of the string are not the same. The program will then proceed to perform the three operations, insert, remove, and replace to find the best solution

return 1 + min(edit\_distance(string\_1, string\_2, len1, len2-1), # Insert

edit\_distance(string\_1, string\_2, len1-1, len2), # Remove

edit\_distance(string\_1, string\_2, len1-1, len2-1) # Replace

)

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provide inappropriate assistance to any student in the class.

