CSE101: Home Assignment 3

Extra resources: Shared along with this pdf is a text file called EnglishWords.txt and a module called GetData.py.

Background For Task 1

I'm shopping for colleges and use Google to search for "Corenell." The response is "Do you mean cornell?" I'm texting my spouse with the message "Honey I'm Home"1 but it goes out as "Money I'm Home." I'm checking out some sequences of nucleotides and want to know how close a match 'AGTCGTC' is to 'TAGTCGT'.
Each of these examples points to an underlying "nearness" problem that involves strings. The big question that we will explore in this assignment is this: When might we regard one string as being close to another? Or better, when might we regard one string as "near" to another?

Here are three possible definitions of string nearness based on how Professor Fat Fingers makes mistakes when texting:
• Two strings are near if they are the same except at some k positions. ("adcmno" and "abemno" for k=2)
• Two strings are near if they are different but we can obtain one by rearranging the characters in the other. ("abc" and "cba")
• Two strings are near if deleting some k characters from one string produces the other string. ("abc" and "abxcy" for k=2)

Task 1

The first order of business is to develop three Boolean-valued functions in the module **a3.py**. Each of these functions checks whether two strings are near or not, for some given value of k, wherever required. They correspond to the "definitions" given above.

Sub-task 1: Off-By-K

Two non-empty strings s1 and s2 are said to be off-by-k if they have the same length and differ in exactly k positions.

Here are some examples:

s1	s2	k	off-by-k?
"read"	"rexd"	1	True

“read”	“xexd”	4	False
“Black”	“Bleak”	2	True
“a”	“a”	1	False
“a”	“A”	1	True

Implement a function **offByK(s1,s2,k)** that takes two nonempty strings s1 and s2 and a non-negative integral value k, and returns True if they are off-by-k and False otherwise.

Sub-task 2: Off-By-Swaps

Two non-empty strings s1 and s2 are said to be off-by-swaps if s1 and s2 may be different but of same length, but s2 can be obtained by arranging/swapping characters in s1, any number of times.

Here are some examples:

s1	s2	off-by-swaps?
“read”	“raed”	True
“read”	“erad”	True
“read”	“reaxd”	False
“read”	“aedr”	True(2 swaps)
“aa”	“aa”	True(same strings)

Implement a function **offBySwaps(s1,s2)** that takes two nonempty strings s1 and s2 and returns True if they are off-by-swaps and False otherwise.

Sub-task 3: Off-By-K-Extra

Two non-empty strings s1 and s2 are off-by-k-extra if removing exactly k characters from s1 gives s2 or if removing k characters from s2 gives s1.

Here are some examples:

s1	s2	k	off-by-k-extra?
“read”	“raed”	1	False

“read”	“red”	1	True
“read”	“reaxder”	3	True
“read”	“read”	0	True
“aa”	“aa”	2	False(same strings)

Implement a function **offByKExtra(s1,s2,k)** that takes two nonempty strings s1 and s2 and a non-negative integral value k, and returns True if they are off-by-k-extra and False otherwise.

Sub-task 4: Testing

Thoroughly test your implementations of offByK, offBySwaps and offByKExtra before proceeding. Create a test module **testa3.py**. Write **atleast 5 extensive test cases** for each of these three functions.

Sub-task 5: Computing List of Near Words

In the next part of the assignment you write a function that return lists of strings.
We need to define what we mean when we say that one string is a near to another:

Two strings s1 and s2 are near if they are not the same string and for a given value of k they are either off-by-k, off-by-k-extra, or off-by-swaps.

Implement the given function in **a3.py** so that it performs as specified:

```
def ListOfNearStrings(s,L,k):
    """ Returns a list of all entries in L that are near to s for a given value of k.
     PreC: s is a nonempty string, k is a non-negative integral value and L is a list of
    nonempty strings. """
```

Sub-task 6: Testing List of Near Words Computation

To test ListOfNearStrings function,design test cases with different values of s, k and a list L. To compute the list L you may use a sub-list of words from EnglishWords.txt and find list of all words in this sub-list that are near to s. To obtain the list of words, you may use the GetData.py module to read data from the EnglishWords.txt file and hence, obtain a list. Choose a sublist to obtain L to pass as argument.

Using this idea write **atleast 5 extensive test cases** to test the ListOfNearStrings function. Write the test procedure in **testa3.py**

Task 2

Sub-task 1

Implement another function in **a3.py** so that it performs as specified

def compare_distr(L1, L2, bin):

""" L1 and L2 represent two lists, of same size, and containing integers. The function checks if the frequency distributions of the two lists are same or not, for a given bin width(class interval).
Return True if the distributions are same, False, otherwise"""

Example: compare_distr([1,2,4,1,3,1,4,6,9],[23,21,24,26,24,21,30,22,22], 2) returns True
Explanation: Frequency distribution for L1:

Class Interval	Freq.
1-3	4
3-5	3
5-7	1
7-9	0
9-11	1

Frequency distribution for L2:

Class Interval	Freq.
21-23	4
23-25	3
25-27	1
27-29	0
29-31	1

Sub-task 2: Testing compare_distr

Write **atleast 5 extensive test cases** to test the compare_distr function. Write the test procedure in **testa3.py**

Submission:

- You need to submit a3.py and testa3.py.
- Make sure each of the .py file contains name and roll number of all team mates.
- Zip the two .py files and upload on Backpack.
- Only one of the team mates needs to make a submission.