Computer Science 1, Fall 2017

Homework 4

For Canvas submission, please put all .py files in a folder, compressed as *HW4_Lastname_Firstname*

Please include the below in the top of your .py program
#HW4 – Part I; otherwise Part II
#Name: Your Last, first
Collaborators: Last Name, First Name I, Last Name, First Name 2; etc.
#Any other brief program comments on what your program does

You may collaborate with anyone you want (or post questions on Piazza). However, when you create your programs, it must be your own code. If you need additional assistance, please reach out.

**Note: in this problem set, you cannot utilize the IN operator (for verifying if a substring appears in a string) or call any string methods provided by Python. You MUST utilize LOOPS and INDEXING.

Part I: Create your new pattern.py file

In this part, we will do some basic pattern matching (similar to what is done in DNA testing, but on a smaller scale.) String-matching is an important subject in the wider domain of text processing. String-matching algorithms are basic components used in implementations of practical software packages existing under most operating systems. Moreover, they emphasize programming methods that serve as paradigms in other fields of Computer science. Sorting-matching consists in finding at least one of the occurrences of a string (more generally called a pattern) in a text (the bigger string). To do this, you will need to read two strings into your program from the user: a pattern string and a text string. The text is n characters long, while the pattern (the smaller string) has a length of m characters, where m<=n. Both strings consist of a finite set of characters, limited to the standard letters in the English alphabet.

The brute force algorithm (which you should implement) consists in checking, at all positions in the text between 0 and n-m (why stop at n-m), whether an occurrence of the pattern starts there or not. Then, after each checking attempt, it shifts the pattern exactly one position to the right.

Let's look at an example:

Brute Force example:

TEXT

G C A T C G C A G A G A G T A T A C A G T A C G

PATTERN

G C A G A G A G

First attempt to find a match

G C A T C G C A G A G A G T A T A C A G T A C G

1 2 3 4

G C A G A G A G

Second attempt

G C A T C G C A G A G A G T A T A C A G T A C G

1

G C A G A G A G

Third attempt

G C A T C G C A G A G A G T A T A C A G T A C G

1

G C A G A G A G

Fourth attempt

G C A T C G C A G A G A G T A T A C A G T A C G

1

G C A G A G A G

Fifth attempt

G C A T C G C A G A G A G T A T A C A G T A C G

1

G C A G A G A G

Sixth attempt

G C A T C G C A G A G A G T A T A C A G T A C G

1

G C A G A G A G

A Match Has Been Found!!

At this point, you will code some functions as well as your main program. Your main program should read in two strings from the user (one string at a time, so you will have two separate strings). The first line (string) that is read is called the text string, and the second line (string) is called the pattern string. Then you will implement a function, called match that implements the simple brute force pattern matching described above**. This function takes two strings (the text string and the pattern string), and returns True if the pattern string matches some portion of the text string: otherwise, the function returns False.** Your main program must keep doing the following task: **ask the user for a text string and a pattern string and print a nice message that states whether the pattern was found in the text or not (something like "yippee a match is found" or "Too bad no match"; please be creative). The program stops when the user enters qqq as the text string (your program must stop regardless of the case of the letter; so any of the inputs qqq, qQq, qQQ, Qqq, etc. must stop the program).**

PartII: Create a new file encoding.py.

We will define two functions in this file (one function corresponds to Part A below and the second function will be the solution to Part B below).

In Part A below, we create a message encoder, and in Part B, we write the decoder.

A. In encoding.py, write a function named **msg_encoder** that takes a string and encodes it based on the following encoding rule: All the occurrences of **a** in the string are replaced with **b**. All the occurrences of **b** in the string are replaced with **a**. All **e** occurrences in the string must be replaced with **&**. After making all these character substitutions, the encoder changes the order of the characters in the string based on the length of the string:
   - If there are an **even number of characters in the strin**g, the encoders break the sting into two halves with equal lengths and puts the second half before the first half to create the final encoded message. For example, assume that after making all the substitutions, the string is c1,c2,c3,c4,c5,c6,c7,c8 (there are 8 characters in this string). The encoder puts the last 4 characters before the first 4 hence, c5 c6 c7 c8 c1 c2 c3 c4 is the final encoded message.
   - If there are an **odd number of characters in the string**, say **2n + 1 characters**, the encoder keeps the **middle letter (character)** and swaps the first and last n characters to create the final encoded message. For example, assume that after making all the substitutions the string is c1 c2 c3 c4 c5 c6 c7 c8 c9 (there are 9 characters in this string). The encoder keeps the mid character c5 and swaps the first and last 4 characters; therefore c6 7 c8 c9 c5 c1 c2 c3 c4 is the final encoded message.

The **msg_encoder** function **returns** the final encoded string (message). **We assume that the original string does not contain the character ^ (look at your keyboard, this is the SHIFT+6) or the character &(SHIFT+7**). However, the string may contain any other characters. Also we don't have to worry about the case of the letters in the message because we assume that all letters in the input string are lowercase.

Hint: you may find it helpful to write a separate function to replace all the occurrences of a given character in a string and call that function in your **msg_encoder** function a number of times.

Some examples:

Original string: 'banana'

Encoded string: 'bnbabn'

Original string: 'boston college'

Encoded string: 'coll&g&aoston'

Original string: 'jessica'

Encoded string: 'icbsj&s'

Original string: 'same'

Encoded string: 'm&sb'

Original string: 'how are you today?'

Encoded string: 'ou todby?how br& y'

b. In **encoding.py** write another function named **msg_decoder** that accepts an encoded string (that has been encoded based on the rules described in Part II. A below) and decodes the encoded string to obtain the original string. The **msg_decoder** function must **return** the decoded string (message).

Note: please note that **msg_decoder (msg_encoder(s)) must result in s for any string s that contains only lowercase letters and doesn't contain ^ or &

***Please test both of your functions with several different strings to verify they are working correctly. However, please do not print anything. We will test your functions using our own testing.*