Before beginning work on this assignment, carefully read the Assignment Submission Specifications posted on Canvas.

Introduction

As noted in chapter 8, the number of possible keys for the transposition cipher is equal to half the length of the message. This means that, unless the message is extremely long, it is still possible to use the brute force technique to decipher a message enciphered with the transposition cipher, even without the key. This exercise will guide you to implementing a stronger cipher, much more resistant to brute force attacks.

The transposition cipher enciphers a message by writing the message down in rows, and then reading of the columns, from left to right. For example, to encipher the message "CIPHERS ARE FUN" with the key 5, we create the following table. We then read the columns left to right to get the ciphertext "CREIS P FHAUERN".

```
C I P H E
R S _ A R
E _ F U N
```

Instead of always reading the columns left to right, suppose we read the columns of in the order 4, 5, 3, 1, 2. We would get the ciphertext "HAUERNP FCREIS" (note the trailing space character). On the other hand, if we choose the order 2, 4, 1, 5, 3, we get the ciphertext "IS HAUCREERNP F". In fact, the number of distinct orders in which we can read these five columns is 5! (120). Even if a hacker knows we used five columns in our table, they still wouldn't know which of the 120 possible orders will allow them to break the code.

Better still, the number of possible orders increases very quickly with the number of columns, since the factorial function grows extremely fast. If we used ten columns instead of five, the number of possible orders is over three million. If we used fifteen, the number of orders is over one trillion. The brute force technique will fail as long as the number of columns is not too small.

Note that the cipher's key is no longer a single number, but a particular permutation of the numbers from 1 up to a particular number. For example, "2, 4, 1, 5, 3" is a possible key to this cipher. This modification – changing the key from a number to a permutation, and reading the columns in that order – results in the *columnar transposition cipher*.

For your implementation, use transpositionEncrypt.py and transpositionDecrypt.py from the textbook source files as references.

You will produce a total of six files for this assignment:

- **a2p1.py, a2p2.py, a2p3.py, a2p4.py**: python solutions to problems 1, 2, 3, and 4 respectively
- **mystery.dec.txt**: file containing the decrypted message from "mystery.txt"
- a README file (**README.txt** or **README.pdf**)

Submit your files in a zip file named 331-as-2.zip.

Problem 1 (2 Marks)
While the columnar transposition cipher is our eventual goal, we will begin with a simple transposition cipher, the rail fence cipher. For this first problem, your task is to modify the provided skeleton file, "a2p1.py" to implement a rail fence cipher. You may use the provided "transpositionEncrypt.py" and "transpositionDecrypt.py" files as reference to implement two functions named "encryptMessage" and "decryptMessage" each taking two arguments:

- key: an integer between 2 and the length of the message to determine how many rails are to be used
- message: a string

Your encryptMessage function should return the message encrypted with the rail fence cipher using the number of rails specified by the key. Your decrypt function should undo the rail fence cipher and return the original plaintext. The following are examples of functions calls and their expected outputs:

```
>>> encryptMessage(2, "SECRET")
'SCEERT'
>>> decryptMessage(2, "SCEERT")
'SECRET'
>>> encryptMessage(3, "CIPHERS ARE FUN")
'CEAFIHR R UPSEN'
>>> decryptMessage(3, "CEAFIHR R UPSEN")
'CIPHERS ARE FUN'
>>> encryptMessage(4, "HELLO WORLD")
'HWE OLORDLL'
>>> decryptMessage(4, "HWE OLORDLL")
'HELLO WORLD'
```

Problem 2 (2 Marks)
For this problem, you will be implementing a columnar transposition cipher. Modify the provided skeleton file, "a2p2.py" to implement a function named "encryptMessage" that takes two arguments:

- key: a list of integers which determine the order in which the columns are to be rearranged
- message: a plaintext string to be encrypted

Every integer in the key is unique and is between 1 and the number of columns. For example, encryptMessage([2, 4, 1, 5, 3], "CIPHERS ARE FUN") should return "IS HAUCREERNP F".

If the length of the given message is not a multiple of the number of columns, your program should use the shaded box technique described in the textbook. For example, encryptMessage([1, 3, 2], "ABCDEFG") should return "ADGCFBE".

The following are examples of functions calls and their expected outputs:

```
>>> encryptMessage([2, 4, 1, 5, 3], "CIPHERS ARE FUN")
```

```
'IS HAUCREERNP F'
>>> encryptMessage([1, 3, 2], "ABCDEFG")
'ADGCFBE'
>>> encryptMessage([2, 1], "HELLO WORLD")
'EL OLHLOWRD'
```

Problem 3 (4 Marks)
Now that you can encrypt using the columnar transposition cipher, the next task is to be able to automatically decrypt messages. Modify the provided skeleton file, "a2p3.py" so that it implements a function named "decryptMessage" that takes two arguments:

- key: the list of integers that was used to encrypt the original plaintext
- message: a cipher string to be decrypted

To test your code, try decrypting the ciphertext
"XOV EK HLYR NUCO HEEEWADCRETL CEEOACT KD", which was encrypted using the key [2,4,6,8,10,1,3,5,7,9].

This program should be able to decrypt messages that were encrypted using the program from problem 2.

Problem 4 (2 Marks)
Use your decryption program to decrypt the file "mystery.txt", which contains some text extracted from Wikipedia, encrypted using the columnar transposition cipher with the key [8,1,6,2,10,4,5,3,7,9]. Modify the provided skeleton file, "a2p4.py" to implement a function named "decryptMystery". The function "decryptMystery" should do the following:

- read the ciphertext from mystery.txt using the **read()** python function (use the UTF-8 encoding)
- call the function "decryptMessage" imported from your "a2p3.py" using the provided key to decrypt "mystery.txt"
- write the deciphered message to "mystery.dec.txt"
- return the string containing the deciphered message

Finally, include the file named "mystery.dec.txt" that you created in your submission.

Note: Make sure your "decryptMessage" function does not add any characters or whitespace to the deciphered message