FCS Lab 2 Report

Name: Aw Yong Jia Min Kellie
Student ID: 1005466

Part I: Substitution Cipher

```
1   import re
2   import string
3   import matplotlib.pyplot as plt
4   import argparse
5   import os
6
7   plt.xlabel('Characters')
8   plt.ylabel('Frequency')
9
10  def freq_analysis(text):
11      text_char_dict = {}
12      for char in text:
13          if char not in text_char_dict:
14              char_count = text.count(char)
15              text_char_dict[str(char)] = char_count
16
17      temp_dict = {}
18      for i in sorted(text_char_dict):
19          temp_dict[i] = text_char_dict[i]
20
21      text_char_dict = temp_dict
22      plt.bar(text_char_dict.keys(), text_char_dict.values(), width=0.3)
23      plt.show()
24      result = decrypt(text, text_char_dict)
25      return result
26
```

In line 10, the `freq_analysis(text)` function was implemented to obtain a graph that shows the frequency of each character in the string of text that was taken in as an argument for the `freq_analysis(text)` function.

From lines 11 to 15, a `text_char_dict` dictionary was created to store each of the characters in the string of text as a key, and the total number times that character appeared in the string of text, as its corresponding value. The method `.count(character)` was used to find the total number of times a particular character appeared in the string of text.

From lines 17 to 21, another `temp_dict` dictionary was created to store the key-value pairs in the `text_char_dict` dictionary, in alphabetical order based on the keys. The original content in `text_char_dict` dictionary is then overwritten with the content in `temp_dict` dictionary such that `text_char_dict` dictionary is now sorted in alphabetical order.

For lines 22 and 23, a bar graph that shows the frequency of each character that appeared in the string of text was plotted and displayed.
From the frequency bar graph, I then manually took note and ordered the characters in descending order based on each of their frequencies.

In line 24, a function, `decrypt(text, text_char_dict)` was called whereby it takes in the string of text and the dictionary that contains the alphabetically sorted keys with their corresponding value.

In line 25, the final decrypted string of text will be returned whereby it will then be written to a text file.

```
28  def decrypt(text, dict):
29      del dict[max(dict, key = dict.get)]
30      # first try to replace the common letters with the letters that appear most frequently
31      # ls_common_letters = ['e', 't', 'a', 'i', 'o', 'n', 's', 'h', 'r']
32      # second try
33      ls_common_letters = ['e', 't', 'i', 'a', 'o', 'n', 's', 'h', 'r']
34
35      # result_1 = text.replace('U', 'e')
36      # result_2 = result_1.replace('J', 't')
37      # result_3 = result_2.replace('Y', 'a')
38
39      for i in range(len(ls_common_letters)):
40          max_char = max(dict, key = dict.get)
41          result = text.replace(max_char, ls_common_letters[i])
42          text = result
43          del dict[max_char]
44
45      result = text.replace('M', 'w')
46      result1 = result.replace('V', 'f')
47      result2 = result1.replace('B', 'l')
48      result3 = result2.replace('W', 'g')
49      result4 = result3.replace('L','v')
50      result5 = result4.replace('T', 'd')
51      result6 = result5.replace('K', 'u')
52      result7 = result6.replace('O', 'y')
53      result8 = result7.replace('S', 'c')
54      result9 = result8.replace('R', 'b')
55      result10 = result9.replace('C', 'm')
56      result11 = result10.replace('F', 'p')
57      result12 = result11.replace('A', 'k')
58      result13 = result12.replace('N', 'x')
59      result14 = result13.replace('Z', 'j')
60      result15 = result14.replace('P', 'z')
61
62      return result15
```

In line 29, as the *space* character had the highest frequency, I used the `max()` function to find the highest value in the dictionary that contains the alphabetically sorted keys, `text_char_dict`. The value is then used to delete the key-value pair in the dictionary, `text_char_dict`.

From lines 30 to 33, I took reference from this website where I took the most commonly used letters of the English language and placed them in a list. I also created another permutation

of the list whereby I switched the order of one of the characters. This is to allow me to visually inspect and compare which partially decrypted texts made more sense.

From lines 39 to 43, I used a for-loop that ranges from the 0 to the last index of the list, `ls_common_letters` such that for each round, I get the highest frequency of each character and replace it with the character that has the index of that round in the list, `ls_common_letters`. I also made sure to remove the character with the highest frequency each round so that there would be no repetition of characters.

In line 45, I first did a visual inspection of the partially decrypted text by running the code first with only the for loop being inside the `decrypt()` function. From there I found the first word in the text being 'Mhat' where I inferred that logically, it could be either '**w**hat' or '**t**hat'. However, since 't' was already used in the `ls_common_letters` list, I replaced 'M' with 'w' instead.

From lines 46-60, I repeated the method of visual inspection where I searched for easier words and replaced illogical characters that were within them, manually with characters that made up a logical word.

In line 62, `result15` which contains the final decrypted string of text is returned and written to a text file in the main function.

Part II: Compromising Integrity

```
39  def hax():
40      # TODO: manipulate ciphertext to decrypt to:
41      # "Student ID 100XXXX gets 4 points"
42      # Remember your goal is to modify the encrypted message
43      # therefore, you do NOT decrypt the message here
44
45      new_plaintext = b"Student ID 1005466 gets 4 points\n"
46      diff_pt = XOR(original_plaintext, new_plaintext)
47      new_cipher = XOR(diff_pt, original_cipher)
48
49      return new_cipher
50
51
52  new_cipher = hax()
53
54
55  # When we finally decrypt the message, it should show the manipulated message
56  print(decrypt(new_cipher, OTP))
```

Let: A be the original plaintext, B be the new plaintext, A' be the original cipher and B' be the new cipher

$A' = A \oplus OTP$

$B' = B \oplus OTP$

Since we want to work from the original cipher, A', such that it will ultimately give us B, the new plaintext, using the idea earlier where $X \oplus X = 0$:

$A \oplus OTP \oplus A \oplus B = A' \oplus A \oplus B$

Thus, $A \oplus A = 0$ and we will only have $OTP \oplus B$

$OTP \oplus B = B'$

Hence, we have obtained the new cipher, B'

Using the explanation above, in lines 45 to 47, I first created a new plaintext that contains the message that should be shown instead of the original plaintext, when decrypted. I then put the original plaintext and new plaintext in the XOR function which corresponds to the $A \oplus B$ in the explanation above. The result of this XOR function is then placed into another XOR function with the original cipher which corresponds to the $A' \oplus A \oplus B$ in the explanation above.
Hence, when decrypting the new cipher obtained from the last XOR function in the function `hax()`, it gives us the new plaintext, "`b'Student ID 1005466 gets 4 points\n'`" instead of the original one.