Cipher project report

We divided the project into three different modules. Per default, all the modules are opened. In the main loop, we added the necessary code to run each of the modules with comments on top of every part. To run a specific module, simply uncomment the part that wants to be run.

Caesar cipher

Logical solution

To decrypt the given text for the first part of this project, we used brute force. We have another text file with a list of frequent words in English that we use to compare with the original text. To proceed, we take the original text, swap all the letters n steps forward, and check for occurrences of the most frequent words. We save the new text in a list as a tuple of type int*string list. That way we can save the potential decipher and the amount of matches with the list of frequent words. At the end of the program, we show the three potential deciphers that have the most matches and prompt the user to choose the one that he thinks is correct. That text is then printed again in the console.

The program works because it computes and creates every possible combination, then checks the amount of matches with real words there is in the text, and saves the ones that have over the average amount of matches. There is no need to check every word with all existing words in the English vocabulary, it is instead enough to check with those words that are most frequent in English. After the computations, the first option shown to the user should be the correct message, but if the program is wrong for some reason, the second or third choice needs to be the correct message. In case that the user isn't satisfied with the three possible decryptions shown, the user can browse the next answers by using the commands "NEXT" and "PREV".

Coding-wise solution

The code for this part is all located in the module CaesarCipher. The main function in our CaesarCipher module is the runCCipher, which executes the logic that breaks the cipher. We have a function called cleanString that splits the string into an array of strings where elements in the array represent a word. This array is converted into a list of strings and sent to the decipher function.

This function goes through the entire list and converts every letter in each word into lowercase. We chose to do this because the list with frequent words is in lowercase letters. After that, it calls a function that compares each word with the list of frequent words. It counts the actual matches and returns it to the decipher function. In order to

know how many matches each decipher attempt has, we created a tuple that stores the actual decipher attempt and the amount of matches that each attempt got. At the end of the decipher function we concatenate the string list into a simple string, sort the attempts by amount of matches and print the three attempts with the most matches.

Frequency analysis

Logical solution

For this cipher, we used a similar method as for the first part. The difference is that here, we used a list with the most frequent characters in English. That was a great advantage, because we found out that the most frequent characters were the space followed by the letter ‘e’. We didn’t stop there and replaced every letter according to the frequency list we found on the internet. In order to help the user a little bit more, we created a help functionality where we count the most frequent words and print them on the screen, right before we print the text in the actual state. After the frequent words in our text and the actual text are printed, we prompt the user to examine the text and choose a character to change and a character to change to. Everything is done via our program, which makes the whole process easier and faster. Because the user gets a list of the most frequent words in our text and because the space and the letter ‘e’ are probably right, we can easily identify a word that most likely is ‘the’ and change the first and second letters in that word to get the correct word. From there, the user needs to carefully examine the text in order to identify possible words and try to change letters to get a readable text.

Coding-wise solution

The program executes through the runFA function, and the first thing it runs is the runWC function, that counts the frequency of each word, and prints all of them on the screen in descending order. Then the text is printed on the screen, according to the replaceWith list. This means that the text never changes, and it’s translated every time it’s printed onto the screen. Every time the user types a character to change, the replaceWith list is changed so that the print function correctly prints the text.

The entire solution is found in the module FrequencyAnalysis, while some key parts are kept in the main function of the entire project.

Beating simple frequency analysis

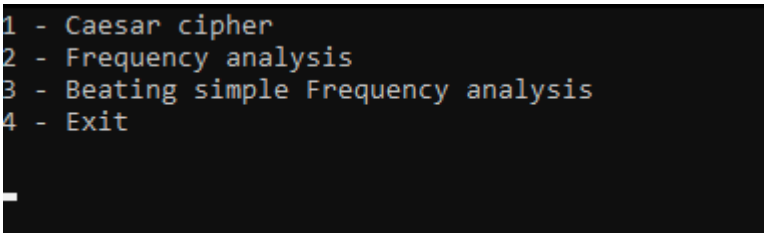
Logical solution

The logic behind this encryption and decryption program lies behind the key. The key is responsible for keeping track of which character needs to be changed and which character is supposed to change to. Then we perform the right instruction according to the key. In order to decrypt, the key is inverted, so that each character is changed for the character it was before, so instead of moving that character x positions to the right, we move it x positions to the left. This works because the encrypt and decrypt use the same algorithm, but inverted.

Coding-wise solution

As usual, the program starts with a main function, called runBC in this case. The program converts the text into a char list in the background. To be able to correctly encrypt the text, we create an offset key and a key length, which are found in the main function. This key decides how many places each character is moved to the right. The encrypt function simply goes through the entire text, and changes each character according to the key offset and length. The decrypt function is also very simple. This function does exactly the same as encrypt, but it does with the inverse of the key, so if the key has the number 1, 2, 3, the decrypt function uses -1, -2, -3 in the same algorithm. This causes the characters to move x positions back to the left.

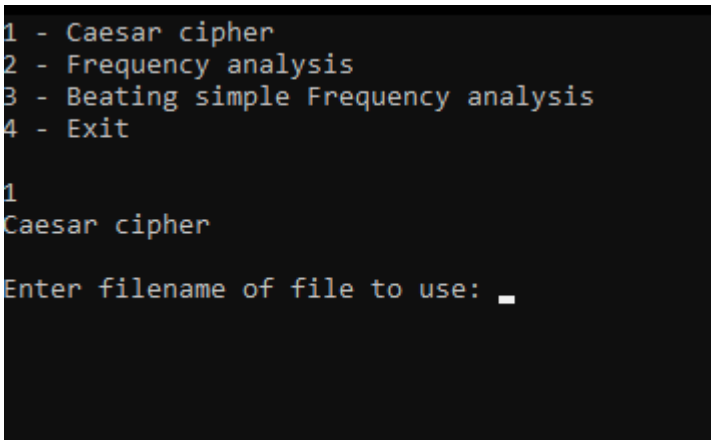
User manual



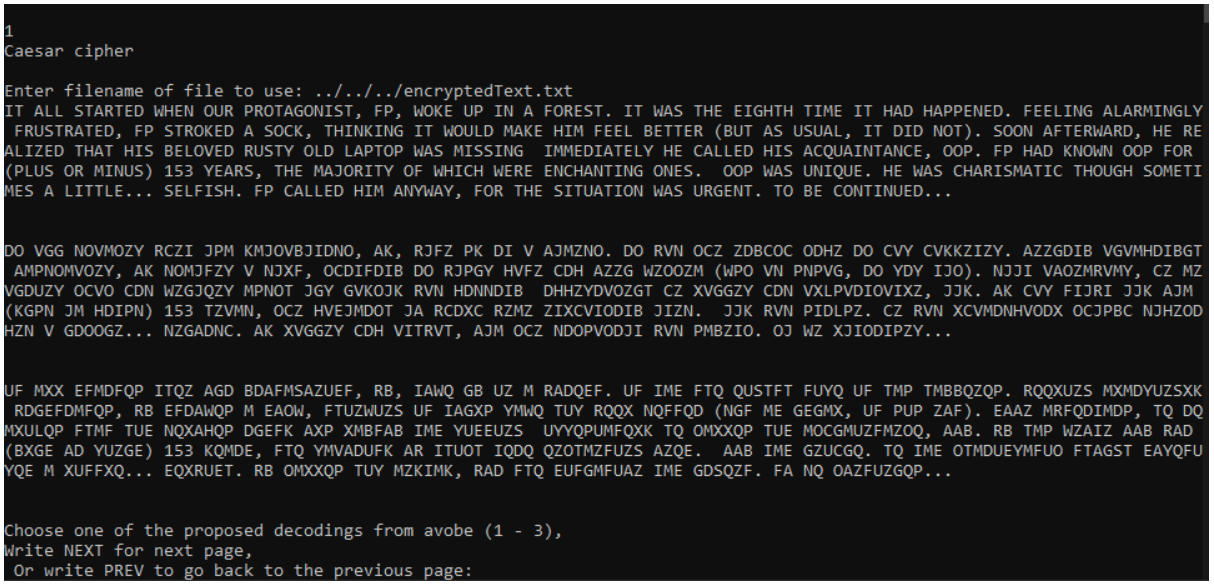
When starting the program the first you will see is the menu. To use any of the functions enter the relevant number and press enter.

When asked for a filename, the path to the file is required.

Caesar cipher



To run the Caesar cipher, simply run the program, press 1, choose the path where the encrypted text is located and you will be presented with the three most likely deciphers. The top one had the most matches with frequent words which are in a text file along the encrypted text. When you choose a decipher it will be printed again and the program will exit. In the case that the user isn't satisfied with the three possible decryptions shown, the user can browse the next answers by using the commands "NEXT" and "PREV".



Frequency analysis

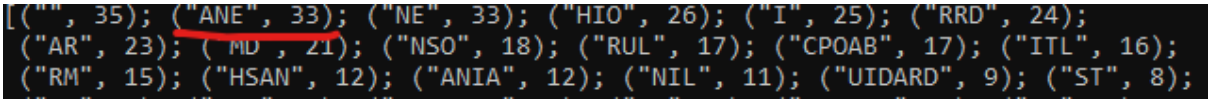
Run the program and press 2. You will be prompted to choose a file that you want to decrypt and a text file with the initial character permutation. You should see:

- A list of tuples with strings and an int
- The current alphabet permutation used to translate
- The current decryption with that permutation
- And lastly, the program asks if you are done.

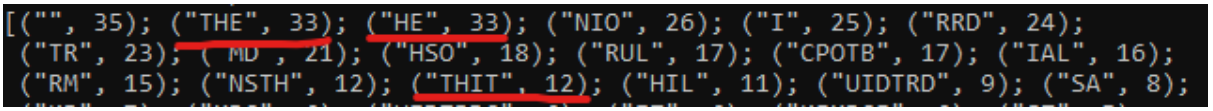
To exit, type EXIT, otherwise, type anything else. Now you should see a prompt for swapping letters.

Tuples on the top

The tuples on the top are words and their respective frequency in the text (the amount of that specific word). They are there to help the decryptor see which words are most frequent in this text and draw conclusions using the knowledge of which words usually are most frequent in english. The list helps identify the word “the” for example (it is usually the most frequent 3 letter word in English).
Example:



This is most likely the word “the”. Then we can swap A with T and N with H.



We got another word “he” and most likely found a second one “that”!

Current translation and swapping letters

Below the tuples are the current alphabet permutation and the translation using that permutation. The order in which the letters are entered doesn’t matter, swapping A and T is the same as swapping T and A. When the letters have been inputted, the program will print the new tuple list, current alphabet and decryption.
Note that you can only use letters that are present in the alphabet permutation.

Beating simple frequency analysis

Run the program and press 3. Now you will be prompted to choose the file that you want to encrypt and decrypt. First the encrypted text will be printed then the encrypted text is decrypted and printed to the screen.