TP1 Report

A85272	Jorge Mota
A83840	Maria Silva

This work has the objective to decipher 3 ciphertexts and do the respective cryptanalysis.

The decrypted results are in the files `plaintext1.txt` , `plaintext2.txt` and `plaintext3.txt` .

The ciphertexts are in uppercase characters and plaintexts in lowercase.

This ciphertexts were encrypted with *affine* , *monoalphabetic substitution* and *Vigenère* (not respectively). Alhtough we didn't know which method of encryption was aplied in order, we deduced that the first one wasn't the *Vigenère*, because there was a lot of texts and too few patterns for 3 letter words.

With this we could try to first find the *affine* ciphertext since its the easist one to get the key (only 2 numbers obtained from systems of equations with modular aritmethic).

But instead we decided to decypher the *affine* and the *monoalphabetic substitution* at the same time , since the *affine* is a type of *monoalphabetic substitution*.

Ciphertext #2 & #3

We deciphered this texts with de the `mono_decoder` function we developed in python

This function receives a ciphertext (in uppercase) and optionally a known_letters map and returns the plaintext if its possible to decode

```
mono_decoder(txt,kl={})
```

Important terms:

Confidence

Confidence is a value from `0` to `1` that represents the algorithm's confidence to deduce the word and make changes to the `known_letters` map. This value is just a fraction with the number of matches an encrypted word gives.

Match

For an encrypted word 'JGG', swapping uppercase characters for dots (`.`) and passing it to the `match` function developed, it returns a list of possible matched in the `words.txt` file, example:

```

> match('...') ['act', 'add', 'age', 'ago', 'aid', 'aim', 'air', 'all', 'and', 'any', 'are', ... ] ```
```

This can be done with a partially encrypted word too. For the word `Joo` passing `.oo` will reduce the number of matches in the result, example:

```

> match('.oo') ['too ] ```
```

Known Letters Map

It's a python dictionary used as a map to store the mapping of the cipher letterto plain letters, example:

```
 known_letters = { 'G':'t', 'U':'a', 'V':'s', 'I':'o', 'T':'r', 'R':'u', 'N':'b', 'O':'l', 'S':'w', 'H':'i', 'J':'p', 'C':'n', 'B':'m', 'M':'c', 'A':'y', 'K':'g', 'F':'v', 'Z':'f', 'L':'d', 'L':'d', 'Y':'q', 'P':'k', }
```

Algorithm

This implementations follows a simple algorithm:

1. Swap letters in `txt` with the `known_letters` map (this will increase the confidence in the results)

2. While text is not fully decrypted

2.1. Find confidence and matches for every word with any uppercase letter

2.2. Select the word with max confidence

2.3. Extract the word selected letters to the known_letters map

There's a catch to this algorithm, the results can go very wrong due to two main factors:

- The words file (`words.txt`) could contain insuficient significant words
- Confidence probability could be low in the

Alternative endings

In execution the algorithm can throw an Exception if it can't make any match, if that happens, then that decryption iteration is unviable.

Hints

If the user has any hint of known letters it should be inserted in the `known_letters` map, example:

```

> known*letters = { 'J':'t', # Assuming: JHA -> the, JGG -> too 'G':'o', # Assuming: JHA -> the, JGG -> too 'A':'e', # Assuming: JHA -> the, JGG -> too 'H':'h', # Assuming: JHA -> the, JGG -> too 'S':'i', # Hinted by the letter frequency swap } mono*decoder(ciphertext,kl=known_letters)

```

This will reduce execution time, and improve results due to increased confidence values.

Ciphertext #1

The Vigenere cipher was a little harder and lucky to decipher "manually".

The inicial take for the cipher was to assume that the inicial `WMP` was a `THE` , this was just a hint because it could be other words.

To make the `WMP` become a `THE` we subtractred the characters to obtain the key for that substitution that was `DFL`

Decrypting the text with vigénere with only the key `DFL` , at first sight we can capture the presence of the word `our` in the first sentence, this could be a coincidence, but we assumed not.

The next step we thought was to extend the key with `A` 's to see if any word could be deduced, with 2 `A` 's the result wasn't clear but we could defenitly have hints of what words could be in the text, such as `oribgn` for `origin` , `whei` for `when` , `of` ...

'n' + {value} = 'i'

i = 8

n = 13

value = 13-8 = -5

The positive of -5 is 26-5 = 21

21 is the letter `V`

With this information we tried the key `DFLVA`

At this point our suspects were right and the `origgn` word matched almost perfectly so we did one more subtitution of the `g` for `i` obtaining `Y`.

The key now was `DFLVY` and we could defenitly see the plain text.

The only detail left was the text wasn't fully decrypted so we removed 2 newlines from the ciphertext and all plain text was visible, declaring this as completed

Final Notes

We also wanted to note that we made a letter frequency attack and tried to swap letters with most used letters in english alphabet, but the results were inconclusive