Readme File

Subject : Network and System Security
(Assignment 1)
Presented By: Suraj Kiran Mate
Entry No: 2021JCS2387

In this assignment we have to decrypt the cipher texts using substitution cipher.

For decryption of such type of cipher texts we are making some generalisations. We know in every language there are some words that are mostly used and same for the English language. We are taking advantage of these frequently used words for our frequency analysis.

During frequency analysis we observed that there are some words which are quad (4 characters), trigraphs (3 characters) and digraphs(2 characters) are frequently used also same for single characters.

We are first replacing frequent quad with some conditions involved before replacing and then trigraph and then digraphs and finally by characters.

We made a decision to replace only one quad, two trigraphs because more frequent quads and trigraphs are already rare to find and also replacement of one them will decide the fate for multiple characters hence also risky to do it more.
Most frequent quads are : **rank4** = {“THAT”, “THIS”}
Most frequent trigraphs are : **rank3** = {“THE”, “ING”, “AND”, “HER”, “ERE”, “ENT”}.
Since we are replacing only one quad and two trigraphs we also should do it precisely.
We are simply measuring the frequency of independent quads using dictionary.
During measuring the frequency of trigraphs we are creating custom data structure which stores the trigraph and its corresponding frequency and its positioning word. All these will be implemented using python dictionary. Here, additional storage of position tell us that whether the triple is suffix of some other word or it is independent. ie. “THE” will be mostly independent trigraph but “ING” will be mostly used in suffix of other words. Hence with additional position knowledge we can map the trigraph correctly with high probability.

so first we are replacing the quads.
While replacing the quads we have to keep in mind the property possessed by our most frequent quads. The first most frequent quad is “**THAT**” so if we are going to replace it with any cipher quad then the first and the last character of that cipher quad should be same. Also while replacing any quad by “**THIS**” all the characters should be different.

After replacing the quads we are calling the function which is made to replace the digraphs - “**mapDigraphs**”. In our code we have created the **rank2** array. This array stores all the possible digraphs in English around 144 entries are stored there. So after the mapping of quad successfully we will check all the possible and most frequent digraphs having these already mapped character and will figure out the second character in the digraph.

Now we will move on to trigraphs replacement.
During replacing trigraphs as well we have to consider the properties of out most frequent trigraphs. ie. all most frequent trigraphs have all unique characters. But the property to consider here somewhat different we have “**THE**, **AND**, **ING**”. It's obvious that **THE** and

AND will most probably come independent in sentence while **ING** come as a suffix of some words in sentence. Thus we have to replace them by keeping these two conditions in consideration.

After replacing the trigraphs we will again call the same function "**mapDigraphs**" and try to map as much digraphs as possible.

Also we have to mention here that after every call and execution of "**mapDigraphs"** function we are updating the **rank2** array. ie. we are deleting those entries from the array which are already mapped. Just to reduce our search complexity.

Most frequent digraphs in order of their decreasing frequencies are: **rank2** = {"TH", "HE", "IN", "AN", "RE", "ON", "ES", "ED", "ST", "EN", "AT", "TO", "NT", "HA","ND"}. We have only mentioned the most frequent digraphs here but there are actually 144 entries in our array.

Now for replacing these digraphs we are implementing some heuristics in our code since after replacing the quad and trigraphs we already got some of the mappings. We have to use these mappings intelligently so that we can figure out what mappings the digraphs should get.

Will first remove out all those digraphs which already got mapped from our **rank2** array. So after removing them we will see that there will be some digraphs which have at least one mapped character. So it will be more easier for us to find the other character from our **rank2** array.

Eg: Let say the cipher digraph which is more frequent is "q@" and after mapping the trigraphs and quad we figured out that the mapping for "@" is "N". Hence we have to figure out the mapping for "q". For this we will traverse the **rank2** array from front and see for those digraphs which has "N" as their second character. Thus we see that "IN" is the first digraph with "N" as the second character, hence we will map "q" to "I".

With this approach we have seen that we can figure out the mappings for most of the characters and with pretty good accuracy.

Now after completing this with digraphs we will look forward for single character frequency in our cipher text.

Most frequent character in English are: **rank1** = {"E", "T", "A", "O", "I", "N", "S", "H", "R", "D", "U", "L", "C", "M", "W", "F", "G", "Y", "P", "B", "V", "K"}

Now we will first remove all the characters from the array which are already mapped. After that from frequency analysis we will take those characters having highest frequency in our cipher text and map them to the most frequent alphabets in English. After every mapping we are again calling the "**mapDigraphs**" function so that we can map the remaining characters.

In this way we will get the complete mapping of all alphabetic characters. We can't guarantee that they all will be absolute, but this code can at least generate the plaintext which can be successfully deciphered with some English heuristics when done manually.

Decryption for Cipher Text 1:

For decryption of ciphertext-1 we are going with generalised approach. We first found all the frequencies of independent quads in the cipher text. Before replacing this most frequent quad we are doing small condition check :

We are checking whether the characters of top 2 most frequent trigraphs are in the quads.

Eg : In this cipher text most frequent quad is "**uy98**" and most frequent trigraphs are "**oq@**" and "**y98**". As "**y98**" is present in "**uy98**", since the probability of getting correct replacement for trigraph is more than quad, we will drop the plan of replacing the quad and we will replace the trigraphs.
Thus we will replace **oq@ —> THE** and **y98 —> AND** hence we got mappings for exact 6 letters. Now we will call "**mapDigraphs**" function to map all the possible digraphs having these 6 characters. And we will be able to find the possible mappings for other characters. Hence at last we will mention all the mappings for cipher text characters with actual English characters and also the source of their mapping.

After that we will go for single character and try to replace those characters which are not yet mapped and have high frequency in the cipher text. Hence we will replace them with most frequent English alphabets. After replacing every alphabet we will again call the **mapDigraphs** function. We are emphasising more on **mapDigraphs** function just because it will help us to get meaningful digraphs. i.e we know in English that "**CM**" is not possible but "**CA**" is possible in most words hence to get meaningful character to character link we are using **mapDigraphs** function.

Hence after complete execution we are getting the key as given below:

y5n8@03q1w4u79$#z2vospxt6r

These mapping are in alphabetic order. Some of the characters could not get mapping as they were not present in the given plaintext so we mapped them randomly.

Thus with this approach the complete plaintext obtained is as follows:

INDIA, OVVICIALLW THE REGUBLIC OV INDIA, IS A COUNTRW IN SOUTH ASIA. IT IS THE SEYENTH LARMEST COUNTRW BW AREA, THE SECOND FOST GOGULOUS COUNTRW, AND THE FOST GOGULOUS DEFOCRACW IN THE PORLD. BOUNDED BW THE INDIAN OCEAN ON THE SOUTH, THE ARABIAN SEA ON THE SOUTHPEST, AND THE BAW OV BENMAL ON THE SOUTHEAST, IT SHARES LAND BORDERS PITH GAJISTAN TO THE PEST; CHINA, NEGAL, AND BHUTAN TO THE NORTH; AND BANMLADESH AND FWANFAR TO THE EAST. IN THE INDIAN OCEAN, INDIA IS IN THE YICINITW OV SRI LANJA AND THE FALDIYES; ITS ANDAFAN AND NICOBAR ISLANDS SHARE A FARITIFE BORDER PITH THAILAND, FWANFAR AND INDONESIA. MOOD, NOP TURN VOR THE SECOND GART OV THE KUESTION, MOOD LUCJ!

Decryption for Cipher Text 2:

Going with the same approach here also we found that the most frequent quad is "**un8u**" also the most frequent trigraphs are "**ryp**" and "**58v**". Hence we can see here that it is not the case like ciphertext-1 where one of the trigraph was the part of quad. Hence we have no restrictions to replace this quad and we will replace it with "**THAT**". Thus we are now ready to call our **mapDigraphs** function which will map the digraphs to possible mappings. After that we will go to trigraphs. Now we haven't mentioned it earlier that we are using some specialised condition before replacing the trigraphs. We are checking whether the frequency of most frequent trigraphs is greater than **20%** of the total number

of trigraphs in the dictionary. If this condition satisfied then we proceed to replace them indicating that they are most frequent and they will probably map accurately. But in this case this condition is satisfied by only one trigraph hence we are going to replace only one trigraph. We fixed it to replace at max 2 trigraphs even though more than 2 trigraphs satisfy this condition.

As we have already mentioned that we are using custom data structure to store the trigraphs, their frequency and also their position in the word. We made this DS just because we can see the most frequent trigraphs in English are **THE**, **ING** and **AND**. Hence **THE** and **AND** are categorised under independent trigraphs means they appear independently in the sentence while on the other hand **ING** appears in suffix of most of the words.
Here the most frequent trigraph **ryp** is appearing in the suffix of words hence we mapped it to **ING.**

After that we called **mapDigraphs** function again and we mapped all the possible digraphs based on the known mappings. And at last we mapped all the remaining most frequent single characters.

The key obtained is:

8t5q4spnrxz2wyv173$u60o9#@

And the corresponding final plaintext is given as:

UEFEATEU ANU DEAVING HIO UINNER LNTSLBHEU, HE CENT TS WEU. THAT NIGHT HE UIU NST ODEEP CEDD, HAVING FEVERIOH UREAMO, HAVING NS REOT. HE CAO LNOLRE CHETHER HE CAO AODEEP SR UREAMING. BSNOBISLO, LNBSNOBISLO, ADD CAO A WDLR. HE REMEMWEREU BRYING, CIOHING, HSPING, WEGGING, EVEN DALGHING. HE FDSATEU THRSLGH THE LNIVEROE, OEEING OTARO, PDANETO, OEEING EARTH, ADD WLT HIMOEDF. CHEN HE DSSKEU USCN, TRYING TS OEE HIO WSUY, THERE CAO NSTHING. IT CAO JLOT THAT HE CAO THERE, WLT HE BSLDU NST FEED ANYTHING FSR JLOT HIO PREOENBE.

In this way we have mapped all the cipher text characters to their most probable alphabetic characters. We can't guarantee the perfect mapping here but if we look at the generated plaintext carefully we can at least try to correct them if we use some manual English heuristics.

Thank You