Decoding and Cryptography

[Project created for CS101 by TAs and inspired by Prof. Amitabha Sanyal's project and Simon Singh's book.]

Abstract

The assignment is about decoding a text document using various heuristics. The texts are encoded using a monoalphabetic substitution. The decoding proceeds in three phases: *etai* strategy, dictionary closure and secret word enumeration.

Introduction

The assignment is about cryptography. In particular, it is about an encoding technique called monoalphabetic substitution. However, the more interesting part is the decoding of an encoded message, which is what you have to implement in this assignment.

The idea behind monoalphabetic substitutions is easy. We are given a message in the English language (which we shall call plaintext from now) to be encoded. The basis of encryption is called a *key*, and the key is a secret that is shared between the sender of the message and the receiver. In the case of monoalphabetic substitution, the key is a fixed permutation of the alphabet. However, remembering a permutation of 26 letters without writing it down is difficult, and anything written may fall into the wrong hands. Therefore, we use an agreed upon word, called a *secret word*, to generate a permutation. As an example, the secret word could be the word ”WISDOM”. For the purposes of this assignment, the secret word will not have any repeated letter. Based on the secret word, the permutation is generated as follows: The first six letters of the alphabet (A-F) map to w, i, s, d, o and m respectively. The letter after F, namely G, maps to n, the letter after m. H maps to p, since o is already used up, and so on.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
w	i	s	d	o	m	n	p	q	r	t	u	v	x	y	z	a	b	c	e	f	g	h	j	k	l

Note that plaintext will be written in capital letters and the encoded message, also called ciphertext will be denoted by small letters. Given the key, the ciphertext for the plaintext in `example-out.txt`_shown in `example-in.txt`. Given a ciphertext, the problem in this assignment is to decode it using a set of strategies that we shall describe below. At the end of each strategy, we either discover the secret word from which we can recreate the complete

key. In this case our method is considered to be successful. Else we try the next strategy with the key returned by the previous strategy. If we exhaust all the strategies without producing the secret key, we report failure.

Strategies

We shall consider several strategies, and each strategy will be described by a function that takes as input: the partially completed key (an empty key for the first strategy) and produces as output either the complete key or a (possibly) modified key. To do this, each strategy first generates a list of substitutions. Each *substitution* is a partial map from plaintext alphabet to ciphertext alphabet. These are the mappings discovered by this strategy. An example of a substitution is `((E . e) (T . o) (A . q) (I . w)).`

To get a sense of how the substitutions are discovered, examine the inputs and outputs of the strategy called *etai* which tries to discover the mappings of letters E, T, A and I. The output contains several possible mappings for this group of letters. We generate several mappings since the reasoning is based on probabilities and we are not certain about the actual mapping. *example-desc.txt* shows the result of applying the strategy *etai*. The result is a list of substitutions.

We then pick each substitution in turn and perform an operation called DC*+SWE (dictionary closure and secret word enumeration or dictionary closure for short) that we describe below. There are three possible outcomes of performing DC*+SWE for a substitution.

1. It might result in the secret word being discovered, in which case the whole process ends with the complete key.
2. Or it may result in the modification of the key. It is important to note that when we modify the key, we ensure that if the modified key is used to partially decode the ciphertext, each word in the ciphertext has at least one possible completion. In this case, we do not try the rest of the substitutions, and move over to the next strategy.
3. However, if DC*+SWE on the current substitution does not result in a modification of the current key, we simply move over to the next substitution.

Simplified ETAI Strategy

Let us discuss the strategy for discovering the letters E, T, A and I. We call this strategy *etai* and describe how to implement it.

In the English language, the most frequently appearing letters are, in order, E, T, A, O, I, N, S, H, R, D, L, U, ... However, this is the frequency order on the average, and this order holds with high probability for texts of large size. For shorter texts consisting of a few paragraphs, we assume (note: this is an assumption) that E, T and A and I will be amongst the top five most frequently occurring letters in a text. This is true of the example that we are currently working with in which the five most frequent letters and their mapping are `(T, e), (E, o), (I, q), (O, y), (A, w)`. In the current example, they are e, o, q, y, w.

Summary: Find the top five frequently occurring letters in the ciphertext. Map these 5 letters to all possible combinations of *ETAI.* Check that there will be *120* partial mappings.

Output of ETAI strategy: A list of 120 ETAI substitutions with each substitution being a map M: [A-Z]->[a-z] with 4 substitutions.

Note: It is mandatory to implement just the above explained *etai* strategies to obtain the results on most of the test cases. But optionally/additionally you can do (much) more! **The document will be updated soon with other additional strategies.**

Dictionary Closure

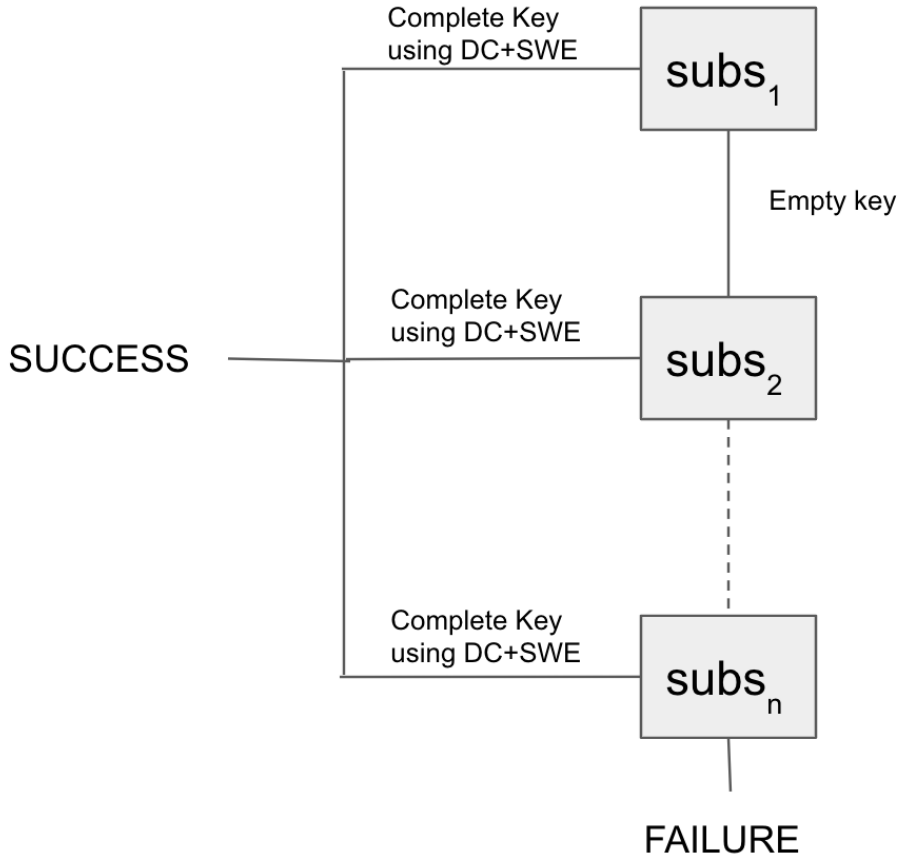
Dictionary closure is an amazingly powerful operation. The idea behind this is as follows. Suppose we currently have a key the current strategy *strat* has resulted in a set of substitutions *subs* of which we are examining the substitution *subk*. Using *subk,* we modify the current-key to obtain a new key. The new key can be applied on the ciphertext to obtain a partially decoded ciphertext. Now we can examine the first word of the partially decoded ciphertext, and use an English dictionary to attempt to complete it. There are three possibilities.

1. The word can be completed in a unique way: The completion results in a further substitution. We change the ciphertext on the basis of this substitution and continue from the next word.
2. The word can be completed, but completion is not unique: We simply step over to the next word with the current key.
3. The word cannot be completed: The substitution *subk* was wrong in the first place. Exit from this substitution with the original key that we started with.

If we have finished the complete list of words without coming out in step 3, then we examine whether the secret word can be uniquely completed (the SWE step). Unique completion

signifies success. If the secret word can be completed in more than one way, we accept the modifications made in this substitution and return the modified key. Finally, if the secret word cannot be completed at all, we undo the modifications of this substitution and return the original key that we came with.

Continuing with the run of the example described in `example-in.txt`, The first substitution to be tried is `((E . e) (T . o) (A . q) (I . w))`. After incorporating this substitution in the empty key we get the modified key. With this modified key we start the dictionary closure. The dictionary closure fails after a few multiple matches, and we try the next substitution. The first success comes with the substitution `((E . o) (T . e) (A . w) (I . q))` and you can see that after incorporating this into the empty key we get some letters of the secret word right. This in turn completes the word DISGUSTS uniquely and more substitutions follow. The key now fills rapidly, which enables more unique completions. When DC* is over, the key is near complete: `(w i s d o m n p q _ t u v x y z a b c e f g h _ k _)`. SWE completes the secret word uniquely and consistently with the rest of the key, thereby completing the key!



Pre-Req (what you need to read before you start)

1. Maps and Vectors data structures [Optional]
2. File Handling in C++

What do you have to do

Summary:

- Generate a list of partial substitutions using ***etai*** strategy.
- Iterate over each substitution:
    - **Dictionary Closure:**
        - Try to extend this substitution if you find a unique completion in the dictionary for a partially decoded word.
        - If a word doesn't have a completion in the dictionary, move to the next substitution.
    - **Secret Word Enumeration:**
        - Now, we have (possibly partially decoded) a key.
        - Find a list of words in the dictionary consistent with the partially decoded *secret word*. Say the word is `a m _ _ e _`. The list of words can be [`amused`, `amazed`].
        - For each word in the list:
            - Construct the entire key using the construction.
            - Decode the text using this mapping.
            - If the text is consistent with the dictionary, consider the key as valid.

- **Input Format and Running:**
    - An input file (say `input.txt`) and dictionary (say `dict.txt`).
    - Compile the code using `g++ main.cpp`
    - Run the code as `./a.out dict.txt input.txt`
- **Output Format:**
    - If there is a unique valid key, print the six letter *secret word.*
    - If there are more than one valid key, print ***all*** six letter *secret words*.
    - If there are no valid keys, print *INVALID*.

References

In our experience, the `etai` strategy followed by DC*+SWE can decode most of the

ciphertexts. Therefore in this assignment you are required only to implement `etai` followed by DC*+SWE. However, those of you who would like to go beyond the assignment can attempt the following more difficult problem. What makes the current encoding scheme to crack is that the key is based on a secret word. If we remove this restriction and let the key be any permutation of the plaintext alphabet, then the problem becomes harder to solve and you have to use the other strategies (some will be updated in this document soon).

Apart from this writeup, we provide the following support materials.

1. The book by Simon Singh from which the original idea was taken.
2. The web page `The Black Chamber` contains interesting information about codes.
3. An executable of the model.
4. The file `example-desc.txt` which contains a trace of what happens when some the example `example-in.txt` is run on the model implementation.

MODE OF SUBMISSION : to be announced

Doubt clearance and help session on MSTeams once a week (precise times to be announced later).

Additional Strategies [Optional]

Note: Do not implement the below strategies before the entire pipeline. Implement all sections except this first.

We saw an important strategy *Simplified-ETAI* that helped us decode the texts. We suggest some improvements to *ETAI* to improve the speed of execution.

ETAI

We were able to generate 120 partial mappings using the *Simplified-ETAI* strategy. We try to eliminate/order these strategies for faster execution.

- *Identifying A or I:* The single letters in the text can map only to 'A' or 'I'. Separate the single lettered words in the text. These are w and q. We can assume that either A maps to w and I to q or vice versa.
- *Identifying T:* A way of separating the consonant T from the vowels E and A. Note that each vowel can have a large number of alphabets as their neighbours. The table in *example-desc.txt* shows what are the letters that each of the cipher-letters e, o, q, y, w has as its neighbours. From the table, we can conclude that the (relatively) unfriendly alphabet e is the consonant T. However, we do try all mappings for *T*.

Other Strategies

Using the ETAI strategy, we were able to decode a large number of English texts. However, ETAI used only monogram-frequency to generate partial mappings. Here, we list out a large number of statistics that can be used to generate partial mappings.
Note: In ETAI we specified that we take the top 5 occurring letters to generate mappings. However, here you will have to design the strategy yourself. One way of doing it would be to download a few English texts and observe different statistics.

1. common-words-double: This is a step that identifies two lettered words based on their frequency of occurrence. The common two letter words in order of frequency are: OF, TO, IN, IT, IS, BE, AS, AT, SO, WE, HE, BY, OR, ON, DO, IF, ME, MY, UP, AN, GO, NO, US, AM . . . .
2. common-words-triple: Similar. The list of three lettered words in frequency order are: THE, AND, FOR, ARE, BUT, NOT, YOU, ALL, ANY, CAN, HAD, HER, WAS, ONE,

OUR, OUT, DAY, GET, HAS, HIM, HIS, HOW, MAN, NEW, NOW, OLD, SEE, TWO, WAY, WHO, BOY, DID, ITS, LET, PUT, SAY, SHE, TOO, USE . . .

3. bigrams: An n-gram is a contiguous sequence of n characters which can occur somewhere within a word. We shall provide tables of the most frequently occurring n-grams for n= 1,2,3 and 4. Thus etai uses a 1-gram also called a monogram. We can also identify letters through the use of 2-grams or bigrams. The most frequently occurring bigram in the ciphertext can be taken to be the encoding of the most frequent bigram in the bigram list. The bigram list starts with: TH, ER, ON, AN, RE, HE, IN, ED, ND, HA, AT, EN, ES, OF, OR, NT, EA, TI, TO, IT, ST, IO, LE, IS, OU, AR, AS, DE, RT, VE . . .
4. trigrams: Similar. The trigram list is: THE, AND, THA, ENT, ION, TIO, FOR, NDE, HAS, NCE, EDT, TIS, OFT, STH, MEN . . .
5. common-initial-letters: Not all letters are equally likely to be the first letter of a word. This is the order of frequency with which a word is likely to begin a word: T, O, A, W, B, C, D, S, M, R, H, I, Y, E, G, L, N, P, U, J, K. . .
6. common-final-letters: The list is: E, S, T, D, N, R, Y, F, L, O, G, H, A, K, M, P, U, W . . .
7. common-double-letters: Finally, only some letters can appear together in words. The frequency order for such letters are: SS, EE, TT, FF, LL, MM, OO. . .