CSE 231 Spring 2018

Programming Project 09

This assignment is worth 55 points (5.5% of the course grade) and must be **completed and turned in before 11:59 on Monday, April 9, 2018.**

Assignment Overview

This assignment will give you more experience on the use of:

1. lists
2. functions
3. dictionaries

The goal of this project is to comprehend and implement two type of attacks on cryptography:1. chosen plaintext attack and 2. quadgram statistics attack.

Assignment Background

Cryptanalysis is the art of code breaking. Secret messages are created using specific encryption algorithms. These codes can be decrypted using the correct algorithm and key. Some of these encryption schemes are more secure than others and cryptanalysis explores the possibility of deciphering pieces of encrypted text (known as ciphertext) without the knowledge of either the encryption algorithm, or the key, or both. The fundamental approach to code breaking lies in carefully observing and exploiting hidden patterns in the ciphertext.

During this assignment, we will be dealing with substitution ciphers and shift ciphers. A substitution cipher is when units of plaintext are replaced with ciphertext. For example, ‘a’ becomes ‘m’, ‘b’ becomes ‘o’ etc. Try to decrypt a simple substitution cipher on this page: https://www.trytodecrypt.com/decrypt.php?id=1#headline

A shift cipher is a kind of substitution cipher where every ‘character’ is shifted a specific number of spots to obtain the corresponding ciphertext. A very famous historical example of a shift cipher is a Caesar cipher where every character is shifted, e.g., three spots down, so ‘d’ becomes ‘a’, ‘e’ becomes ‘b’ etc. For example, CSE would be encrypted as ‘ZPB’.

In this programming exercise, your task will be to recreate two well-known cryptanalysis techniques. A **chosen plaintext attack** is when a cryptanalyst has the luxury to encrypt any plaintext and obtain the corresponding ciphertext. This ability makes it easier to deduce the encryption algorithm. For example, in the ‘trytodecrypt’ exercise above, enter ‘a’ and observe the corresponding ciphertext, then ‘b’ and so on. To make it easier, enter entire series of characters such as ‘abcdef…’ etc. and obtain the corresponding substitutions. Our program will ask for the plaintext and the corresponding ciphertext and then attempt decryption of an unknown ciphertext based on the observed substitutions.

The other cryptanalysis technique, **quadgram statistics**, is used to break a variety of ciphers such as Enigma, a shift cipher, and is a powerful method. The main idea is that certain letters are repeated more often than others in a language and we can exploit this knowledge to attempt mappings and notice if the

potential decryption makes sense. For example, in the English language letters 'E', 'T', 'A' have the highest probability of occurrence and so we can substitute the top three most occurring letters in the ciphertext with E, T, A respectively and so on. If the decryption is readable and makes sense, then we have successfully broken the code, otherwise we try other mappings. If interested, read more about frequency analysis here: https://en.wikipedia.org/wiki/Frequency_analysis

Quadgram statistics, which we will be implementing in this project, is based on a similar idea. Certain groups of four letters are more common than others in the English language. For example, if you read Leo Tolstoy's 'War and Peace', there are roughly 2.5 million quadgrams in the book. The most common quadgrams observed are: 'TION', 'NTHE', 'THER', 'THAT' (plaintext has spaces removed because they provide too many clues to assist decryption). We are using quadgram analysis as opposed to monogram, bigram or trigram analysis since quadgram statistics arguably perform slightly better than the others.

Project Description

We provide a file called `english_quadgrams.txt` that stores English quadgrams, one per line, with their frequency next to them. For example: '`TION 13168375`' is the first line in the file. We will read the file line by line, storing all quadgrams with their observed frequencies, then calculate the log probability of each quadgram. We display the top 10 quadgrams to the user with their frequency and log probability and then attempt at decrypting the shift cipher with every possible key (known as a **brute force** attack). While attempting decryption with every possible key, we use the total log probability in the resulting potential plaintext as a fitness function to determine if we have the right key. Basically, if the total log probability is the highest for a particular potential plaintext, then it looks more like English and is likely to be the correct decryption. We display the top three fits to the user along with the attempted decryption and the best fit will display the decrypted text if everything is implemented correctly.

`open_file() -> fp`
This function returns a file pointer. You likely have a copy from a previous project. It repeatedly prompts for a file until one is successfully opened. It should have a try-except statement.

`log_probability_dictionary(fp) -> dictionary`
This function takes a single parameter which is a file pointer to the quadgrams file. It reads the file line by line and builds a dictionary containing key-value pairs where each key is a quadgram and the corresponding value is a list. The first item in the list is frequency or count of the quadgram read from the file and the second item is the calculated log probability. This log probability is calculated as follows for quadgram 'TION':

$$log(p('TION')) = log_{10}(Count('TION')/TotalQuadgrams)$$

$$log(p('TION')) = log_{10}(13168375/4224127912) = -2.506205$$

After building the dictionary, this function prints the top ten most frequent quadgrams along with their log probabilities in a table as shown below. To get the top ten you must put all the values in a list, sort, and then display the top ten.

```
Quadgram        Count       Log Probability
----------------------------------------
TION         13168375             -2.506205
NTHE         11234972             -2.575165
THER         10218035             -2.616370
THAT          8980536             -2.672435
OFTH          8132597             -2.715508
FTHE          8100836             -2.717207
THES          7717675             -2.738251
WITH          7627991             -2.743327
INTH          7261789             -2.764693
ATIO          7104943             -2.774176
```

- Use the following format string to match Mimir output:
  `"{:<8s}{:>13s}{:>22s}".format('Quadgram','Count','Log Probability')`
- Display the float values to 6 decimal places

Finally, return the dictionary of quadgrams.

`fitness_calculator(potential_plaintext, quadgram_dictionary) -> float`
This function accepts potential plaintext and dictionary of quadgrams as parameters. It returns the sum of fitness values corresponding to each quadgram found in the potential plaintext. Quadgrams are sequences of four characters in the potential plaintext. For example in 'PYTHON' the quadgrams are: 'PYTH', 'YTHO', 'THON'. Extract all quadgrams from potential plaintext and look up the log probability of each in the quadgram dictionary (if a quadgram is not found in the dictionary, skip it). Find the sum of the probabilities (the overall log figness value) and return it.
(Hint: when developing this function start by printing the quadgrams in 'PYTHON' to get that part of the algorithm correct before continuing.)

`chosen_plaintext_attack(plaintext, ciphertext, bifurcation, texttodecrypt)`
This function takes four parameters: `plaintext`, `ciphertext` and `texttodecrypt` are strings, and `bifurcation` is an integer. The `plaintext` is a piece of text and `ciphertext` is the corresponding ciphertext. `Bifurcation` indicates the size of the mapping of ciphertext to plaintext. For example, if 'a' is encrypted as 'AA', the bifurcation size is 2. If 'a' is encrypted as 'AAAA', then the bifurcation size is 4. This function builds a dictionary of mappings based on the plaintext and the corresponding ciphertext. For example, if 'abc' is the plaintext and '0C0D0E' is the ciphertext and bifurcation size is 2, then the function will build a dictionary with keys '0C','0D','0E' and corresponding values 'a','b','c'. The function decrypts `texttodecrypt` by looking up all

substitutions in the dictionary just built and displays the decrypted text. For example, if the bifurcation is 2, every 2 characters in `texttodecrypt` will be looked up in the dictionary and the corresponding value will be added on to the decrypted text.

If a key is not found in the dictionary, it prints the message "Decryption interrupted. Key not found: {}" where {} represents the character that could not be mapped and doesn't print any decrypted charactes. i.e.

```
Text to decrypt: 131017171A48221A1D170F

Decryption interrupted. Key not found:  48
```

`bruteforce_shift_cipher(ciphertext, quadgram_dictionary)`

This function takes two arguments: `ciphertext` to decipher and the `quadgram_dictionary` returned by the function `log_probability_dictionary`. It attempts to brute force the key space by running through every possible key (0-25) and calculating overall log fitness of each attempted decryption and returning the top five fits and the top fit as the potential solution.

To achieve this goal, it converts ciphertext string to uppercase and maintains a `fitness_list` that is a list of tuples with each tuple containing 'log fitness, key, potential plaintext' corresponding to a particular potential plaintext. It runs through a cycle of every possible mapping, starting from 'A' becomes 'A' (key=0), then 'A' becomes 'B' (key=1), all the way to 'A' becomes 'Z' (key=25). To achieve this, maintain a rotating ASCII list which shifts by one character after the fitness of current mapping is calculated. Call the function `fitness_calculator()` (explained above) within each rotation to calculate the overall fitness of this mapping and append the returned tuple to the `fitness_list`. It would be helpful to keep a mapping dictionary within these rotations that keeps track of all character mappings for this rotation. Hence for key=1, the mapping dictionary contains: {'A':'B','B':'C'…}.

After all fitness values are stored in the `fitness_list` sort tuples within the list by fitness values and display the top five fits to the user in the following manner:

```
Key               Plaintext                    Fitness
-----------------------------------------------------
14    ITISPARADOXICALYETTRUETOSAYTHATTHEM   -1122.8891
0     UFUEBMDMPAJUOMXKQFFDGQFAEMKFTMFFTQY   -1564.9891
11    FQFPMXOXALUFZXIVBQQORBQLPXVQEXQQEBJ   -1608.6779
4     YJYIFQHQTENYSQBOUJJHKUJEIQOJXQJJXUC   -1614.5090
16    KVKURCTCFQZKECNAGVVTWGVQUCAVJCVVJGO   -1617.5991


press any key to continue...
```

Use the following format strings to match Mimir output:

- `"{:<5s}{:^35s}   {:>10s}".format("\nKey","Plaintext","Fitness")`

- Display only the first 35 characters of plaintext for the preview.
- Display the float values to 4 decimal places

Note that at this point, the program should wait for an input from the user. After receiving this input, print the decrpytion corresponding to the best fit as the solution as shown:

```
press any key to continue...

Decrypted ciphertext:
ITISPARADOXICALYETTRUETOSAYTHATTHEMOREWEKNOWTHEMOREIGNORANTWEBECOMEIN
THEABSOLUTESENSEFORITISONLYTHROUGHENLIGHTENMENTTHATWEBECOMECONSCIOUSO
FOURLIMITATIONSPRECISELYONEOFTHEMOSTGRATIFYINGRESULTSOFINTELLECTUALEV
OLUTIONISTHECONTINUOUSOPENINGUPOFNEWANDGREATERPROSPECTSNIKOLATESLA
```

`main()`

This function prints provided BANNER and MENU and then asks the user to make a choice between a chosen plaintext attack and Ngram frequency analysis attack. If the choice is 1, it asks for plaintext, ciphertext, bifurcation size, text to decrypt, and then calls the `chosen_plaintext_attack()` function with these parameters.

```
  ------------------------------------------------------------------------
    Welcome to the world of code breaking. This program is meant to help
    decipher encrypted ciphertext in absence of knowledge of algorithm/key.
    ------------------------------------------------------------------------

    1. Chosen plaintext attack
    2. Ngram frequency analysis


Choice: 1

Plaintext: 0123456789qwertyuiopasdfghjklzxcvbnm.,

Ciphertext:
02030405060708090A0B1C22101D1F2420141A1B0C1E0F1112131516172523 0E210D19184243

Bifurcation: 2
```

If the choice is 2, invoke `open_file()` and pass the file pointer to `log_probability_dictionary()`. Ask the user for ciphertext to decrypt, *remove any punctuation or spaces from the ciphertext* and pass it to the `bruteforce_shift_cipher()` function along with the quadgrams dictionary returned by `log_probability_dictionary()`.

```
    ------------------------------------------------------------------------
    Welcome to the world of code breaking. This program is meant to help
```

```
    decipher encrypted ciphertext in absence of knowledge of algorithm/key.
    ------------------------------------------------------------------------

    1. Chosen plaintext attack
    2. Ngram frequency analysis


Choice: 2

Input a file name: english_quadgrams.txt

Quadgram        Count       Log Probability
----------------------------------------
TION         13168375             -2.506205
NTHE         11234972             -2.575165
THER         10218035             -2.616370
THAT          8980536             -2.672435
OFTH          8132597             -2.715508
FTHE          8100836             -2.717207
THES          7717675             -2.738251
WITH          7627991             -2.743327
INTH          7261789             -2.764693
ATIO          7104943             -2.774176

Ciphertext:
```

Ensure proper error and exception handling in case the user enters an invalid choice.

Assignment Deliverable

The deliverable for this assignment is the following file:

`proj09.py` – the source code for your Python program

Be sure to use the specified file name and to submit it for grading via Mimir before the project deadline.

Assignment Notes

- More about quadgram analysis as a fitness measure: http://practicalcryptography.com/cryptanalysis/text-characterisation/quadgrams/
- Use 'punctuation' from 'string' module to get rid of punctuation in the ciphertext.
- from `math` import `log10` to perform the log calculations.
- Solve more code breaking exercises at http://trytodecrypt.com/ -- this program will help you solve all of the easy exercises plus some medium level exercises in a matter of seconds.

- Items 1-9 of the Coding Standard will be enforced for this project.

Test Cases

Function Tests: for each there is an input and an "instructor value" that the instructor's version of the function returned.

Function Test log_probability_calculator

```
fp = open("small_quadgrams.txt", "r")
student_dict = log_probability_dictionary(fp)

instructor_dict = {'ATIO': [7104943, -1.3381806523725297],
 'DTHE': [6470280, -1.378818175408842],
 'ETHE': [6135216, -1.4019113931286349],
 'FTHE': [8100836, -1.2812114104406],
 'HERE': [5630500, -1.4391942876775983],
 'INGT': [6461147, -1.3794316285355102],
 'INTH': [7261789, -1.3286976246736686],
 'MENT': [4968019, -1.4935580023717034],
 'NTHE': [11234972, -1.1391692560862068],
 'OFTH': [8132597, -1.2795119985367518],
 'OTHE': [6900574, -1.3508560329859485],
 'SAND': [5996705, -1.4118285656676168],
 'STHE': [5748611, -1.4301783289108567],
 'THAT': [8980536, -1.2364389923382941],
 'THEC': [5466071, -1.4520659819462083],
 'THEM': [4911484, -1.49852851688969],
 'THER': [10218035, -1.1803738645414745],
 'THES': [7717675, -1.3022547644962412],
 'TION': [13168375, -1.0702090648998746],
 'TTHE': [6553056, -1.373297371193673],
 'WITH': [7627991, -1.307331078520768]}
```

Function Test fitness_calculator

```
student_value = fitness_calculator('PRANSHU', {'PRAN': [4911484, -
1.49852851688969], 'RANS': [10218035, -1.1803738645414745], 'NSHU':
[7717675, -1.3022547644962412], 'TION': [13168375, -
1.0702090648998746]})

instructor_value = -3.9811571459274058
```

Test 1: 'key not found' while decrypting text

```
---------------------------------------------------------------------
Welcome to the world of code breaking. This program is meant to help
decipher encrypted ciphertext in absence of knowledge of algorithm/key.
---------------------------------------------------------------------
```

```
    1. Chosen plaintext attack
    2. Ngram frequency analysis


Choice: 1

Plaintext: 0123456789qwertyuiopasdfghjklzxcvbnm.,     (There is a space after ",")

Ciphertext:
02030405060708090A0B1C22101D1F2420141A1B0C1E0F11121315161725230E210D19184243

Bifurcation: 2

Text to decrypt: 131017171A48221A1D170F

Decryption interrupted. Key not found:  48
```

Test 2: exercise 3 from trytodecrypt.com

```
    ------------------------------------------------------------------------
    Welcome to the world of code breaking. This program is meant to help
    decipher encrypted ciphertext in absence of knowledge of algorithm/key.
    ------------------------------------------------------------------------

    1. Chosen plaintext attack
    2. Ngram frequency analysis


Choice: 1

Plaintext: 0123456789qwertyuiopasdfghjklzxcvbnm.,     (There is a space after ",")

Ciphertext:
F2F3F4F5F6F7F8F9FAFB0D13010E101511050B0CFC0F000203040607081614FE12FD0A09333439

Bifurcation: 2

Text to decrypt: 0A0B1339150B1139070A0B13390510

Decrypted text:         now you know it
```

Test 3

```
    ------------------------------------------------------------------------
    Welcome to the world of code breaking. This program is meant to help
    decipher encrypted ciphertext in absence of knowledge of algorithm/key.
    ------------------------------------------------------------------------

    1. Chosen plaintext attack
    2. Ngram frequency analysis


Choice: A
Invalid input.
```

```
Choice: 3
Invalid input.

Choice: 2

Input a file name: random.txt
Unable to open the file. Please try again.

Input a file name: english.txt
Unable to open the file. Please try again.

Input a file name: english_quadgrams.txt

Quadgram        Count       Log Probability
-----------------------------------------
TION         13168375             -2.506205
NTHE         11234972             -2.575165
THER         10218035             -2.616370
THAT          8980536             -2.672435
OFTH          8132597             -2.715508
FTHE          8100836             -2.717207
THES          7717675             -2.738251
WITH          7627991             -2.743327
INTH          7261789             -2.764693
ATIO          7104943             -2.774176

Ciphertext: Uf ue bmdmpajuomx, kqf fdgq, fa emk, ftmf ftq yadq iq wzai, ftq yadq
uszadmzf iq nqoayq uz ftq mneaxgfq eqzeq, rad uf ue azxk ftdagst qzxustfqzyqzf ftmf
iq nqoayq oazeouage ar agd xuyufmfuaze. Bdqoueqxk azq ar ftq yaef sdmfurkuzs
dqegxfe ar uzfqxxqofgmx qhaxgfuaz ue ftq oazfuzgage abqzuzs gb ar zqi mzp sdqmfqd
bdaebqofe - Zuwaxm Fqexm

Key              Plaintext                    Fitness
-----------------------------------------------------
14   ITISPARADOXICALYETTRUETOSAYTHATTHEM   -1122.8891
0    UFUEBMDMPAJUOMXKQFFDGQFAEMKFTMFFTQY   -1564.9891
11   FQFPMXOXALUFZXIVBQQORBQLPXVQEXQQEBJ   -1608.6779
4    YJYIFQHQTENYSQBOUJJHKUJEIQOJXQJJXUC   -1614.5090
16   KVKURCTCFQZKECNAGVVTWGVQUCAVJCVVJGO   -1617.5991


press any key to continue...

Decrypted ciphertext:
ITISPARADOXICALYETTRUETOSAYTHATTHEMOREWEKNOWTHEMOREIGNORANTWEBECOMEINTHEABSOLUTESEN
SEFORITISONLYTHROUGHENLIGHTENMENTTHATWEBECOMECONSCIOUSOFOURLIMITATIONSPRECISELYONEO
FTHEMOSTGRATIFYINGRESULTSOFINTELLECTUALEVOLUTIONISTHECONTINUOUSOPENINGUPOFNEWANDGRE
ATERPROSPECTSNIKOLATESLA
```

Grading Rubric

General Requirements:

5 pts Coding Standard 1-9
(descriptive comments, function headers, etc...)

Function Tests:
4 pts open_file (no Mimir test)
5 pts fitness_calculator
5 pts log_probability_dictionary

Program Tests
8 pts Test 1
8 pts Test 2
8 pts Test 3
12 pts Test 4 (Hidden test)