Assignment 4

General Instructions:

1. Deadline: Saturday, November 23rd at 11:59 pm.
2. This assignment is worth 10 pts of your course grade.
3. Download the assignment files, which consist of the following:
    - template_A4.py (Rename to solution_A4.py)
    - mod_template.py (Rename to mod.py)
    - matrix_template.py (Rename to matrix.py)
    - dictionary file: engmix.txt (do not change)
    - test_A4.py (do not change)
    - utilities_A4.py (do not change)
    - q1_sample.txt (do not change)
    - q4_sample.txt (do not change).
4. At the top of the three files: mod.py, matrix.py and solution_A4.py edit your name on top of file.
5. Your solutions should be entered into three files: mod.py, matrix.py and solution_A4.py.
6. In your solution, you can only import the following Python libraries: string, math. Any use of external libraries will cause rejection of your assignment.
7. In the utilities_A4.py file the implementation for load_dictionary analyze_text, text_to_words and is_plaintext have been changed for better performance. Make sure to observe how to use these functions.
8. At the end, submit one zip file called: “yourname_A4.zip”. The zip file contains three files: solution_A4.py, mod.py and matrix.py

Q1: Modular Arithmetic Library
(2 pts)

Your task in this question is to implement a library for modular arithmetic. This library will be your main source for any cipher that uses modular arithmetic. Therefore, ensure that your solution is accurate.

When you open the file: mod.py, you will notice that there are 18 functions to complete. Each function is a simple function that provides a specific operation in modular arithmetic.

The 18 functions are:

```
 1- residue_set(mod)
 2- residue(num,mod)
 3- is_congruent(a,b,m)
 4- add(a,b,m)
 5- sub(a,b,m)
 6- add_inv(a,m)
 7- add_table(m)
 8- sub_table(m)
 9- add_inv_table(m)
10-mul(a,b,m)
11-mul_table(m)
12-is_prime(n)
13-gcd(a,b)
14-is_relatively_prime(a,b)
15-has_mul_inv(a,m)
16-eea(a,b)
17-mul_inv(a,m)
18-mul_inv_table(m)
```

A description for each function is provided outlining the input parameters, return parameters, how the function is expected to work, and error handling mechanism. Some functions are also supported with examples.

Testing and error checking should be done locally at each function. The error message returns the function name and error type. Testing modules are provided for you to test your functions locally.

Running test_q1() will pipe your output into an output file, which is compared to q1_sample.txt which contains the expected output. A utility function is used to tell you where the mismatch happens in your file, in case it appears.

Q2: Decimation Cipher
(2 pts)

Your task in this question is to write the encryption, decryption and cryptanalysis functions for the decimation cipher. This requires writing three functions:

1- Encryption scheme: *e_decimation(plaintext,key)*

2- Decryption scheme: *d_decimation(ciphertext,key)*

3- Cryptanalysis: *cryptanalysis_decimation(ciphertext)*

The key is tuple of (*baseString,k*), where the *baseString* is a sub string of the output of the utility function: *get_baseString()* and *k* represents the value *k* in the following equation: $y = kx$. If an invalid key is passed to the encryption or decryption functions, it should print an error message and return an empty string.

The encryption and decryption functions preserve the case of the text, as much as possible (sometimes this is not possible). Characters not found in the *baseString* should be excluded from encryption/decryption.

The cryptanalysis function should return the key tuple as defined above. The function should also print the number of keys attempted. This refers to the number of valid keys that were tested. If cryptanalysis fails, the function should return a tuple of two empty strings.

The function definitions of the three functions are displayed below:

```
#-------------------------------------------------------------
# Parameters:   plaintext (str)
#               key (str,int)
# Return:       ciphertext (str)
# Description:  Encryption using Decimation Cipher
#               key is tuple (baseString,k)
#               Does not encrypt characters not in baseString
#               Case of letters should be preserved
# Errors:       if key has no multiplicative inverse -->
#                   print error msg and return empty string
#-------------------------------------------------------------
def e_decimation(plaintext,key):
```

```
#-------------------------------------------------------------
# Parameters:   ciphertext (str)
#               key (str,int)
# Return:       plaintext (str)
# Description:  Decryption using Decimation Cipher
#               key is tuple (baseString,k)
#               Does not decrypt characters not in baseString
#               Case of letters should be preserved
# Errors:       if key has no multiplicative inverse -->
#                   print error msg and return empty string
#-------------------------------------------------------------
def d_decimation(ciphertext,key)

#-------------------------------------------------------------
# Parameters:   ciphertext (str)
# Return:       plaintext,key
# Description:  Cryptanalysis of Decimation Cipher
#-------------------------------------------------------------
def cryptanalysis_decimation(ciphertext)
```

Running the testing function will give:

```
-----------------------------------------
Testing Q2: Decimation Cipher

key =  ('abcdefghijklmnopqrstuvwxyz', 5)
plaintext =   Strike in progress
ciphertext=   Mrhoyu on xhsehumm
plaintext2=   Strike in progress

key =  ('abcdefghijklmnopqrstuvwxyz ?!', 46)
plaintext =   Is mission accomplished?
ciphertext=   Uqhbuqqugshaffgbxnuqdkwy
plaintext2=   Is mission accomplished?

key =  ('abcdefghijklmnopqrstuvwxyz ?!.', 10)
plaintext =  Plan B ... initiated
ciphertext=  Error (e_decimation): Invalid key
plaintext2=  Error (d_decimation): Invalid key

Testing cryptanalysis:

key found after  183 attempts
Key found =  ('abcdefghijklmnopqrstuvwxyz 01234567', 33)
plaintext =  there are two kinds of cryptography in this world: cryptography
that will stop your kid sister from reading your files, and cryptography that
will stop major governments from reading your files.
-----------------------------------------
```

Q3: Affine Cipher
(2 pts)

Your task in this question is to write the encryption, decryption and cryptanalysis functions for the Affine Cipher. This requires writing three functions:

4- Encryption scheme: *e_affine(plaintext,key)*

5- Decryption scheme: *d_affine(ciphertext,key)*

6- Cryptanalysis: *cryptanalysis_affine(ciphertext)*

The key is the tuple (baseString, key). The baseString is similar to the definition provided in Q2. The key is the list [alpha,beta] as defined by the equation: $y = \alpha x + \beta$.

```
#-----------------------------------------------------------
# Parameters:   plaintext (str)
#               key (str,[int,int])
# Return:       ciphertext (str)
# Description:  Encryption using Affine Cipher
#               key is tuple (baseString,[alpha,beta])
#               Does not encrypt characters not in baseString
#               Case of letters should be preserved
# Errors:       if key can not be used for decryption
#                   print error msg and return empty string
#-----------------------------------------------------------
def e_affine(plaintext,key):

#-----------------------------------------------------------
# Parameters:   ciphertext (str)
#               key (str,[int,int])
# Return:       plaintext (str)
# Description:  Decryption using Affine Cipher
#               key is tuple (baseString,[alpha,beta])
#               Does not decrypt characters not in baseString
#               Case of letters should be preserved
# Errors:       if key can not be used for decryption
#                   print error msg and return empty string
#-----------------------------------------------------------
def d_affine(ciphertext,key):

#-----------------------------------------------------------
```

```
# Parameters:   ciphertext (str)
# Return:       plaintext,key
# Description:  Cryptanalysis of Affine Cipher
#-------------------------------------------------------------
def cryptanalysis_affine(ciphertext):
```

Running the testing module will give:

```
>>> test_q3()
-----------------------------------------
Testing Q3: Affine Cipher

key =  ('abcdefghijklmnopqrstuvwxyz', [5, 8])
plaintext =   AFFINE CIPHER
ciphertext=   IHHWVC SWFRCP
plaintext2=   AFFINE CIPHER

key =  ('abcdefghijklmnopqrstuvwxyz ?!', [45, 3])
plaintext =   Is mission accomplished?
ciphertext=   Pbnvpbbpyindggyvlfpb!jwa
plaintext2=   Is mission accomplished?

key =  ('abcdefghijklmnopqrstuvwxyz ?!.', [10, 19])
plaintext =  Plan B ... initiated
ciphertext=  Error (e_affine): Invalid key
plaintext2=  Error (e_affine): Invalid key
```

```
Testing cryptanalysis:
key found after  36 attempts
Key found =  ('abcdefghijklmnopqrstuvwxyz', [3, 9])
plaintext =  There are two kinds of cryptography in this world: cryptography
that will stop your kid sister from reading your files, and cryptography that
will stop major governments from reading your files.

key found after  621 attempts
Key found =  ('abcdefghijklmnopqrstuvwxyz ', [17, 11])
plaintext =  There are two kinds of cryptography in this world: cryptography
that will stop your kid sister from reading your files, and cryptography that
will stop major governments from reading your files.
-----------------------------------------
```

Q4: Matrix Library
(2 pts)

Your task in this question is to implement a library for matrix arithmetic. This library will be your main source for any cipher that uses matrices. Therefore, ensure that your solution is accurate.

When you open the file: matrix.py, you will notice that there are 18 functions to complete. Each function is a simple function that provides a specific operation over matrices.

The 18 functions are:

```
1-is_vector(A)
 2- is_matrix(A)
 3- parint_matrix(A)
 4- get_rowCount(A)
 5- get_columnCount(A)
 6- get_size(A)
 7- is_square(A)
 8- get_row(A,i)
 9- get_column(A,j)
10-get_element(A,i,j)
11-new_matrix(r,c,pad)
12-get_I(size)
13-is_identity(A)
14-scalar_mul(c,A)
15-mul(A,B)
16-matrix_mod(A,m)
17-det(A)
18-inverse(A,m)
```

A description for each function is provided outlining the input parameters, return parameters, how the function is expected to work, and error handling mechanism.

Testing and error checking should be done locally at each function. The error message returns the function name and error type.

Running test_q4() will pipe your output into an output file (q4_solution.txt), which is compared to q4_sample.txt which contains the expected output. A utility function is used to tell you where the mismatch happens in your file, in case it appears.

Q5: Hill Cipher
(2 pts)

Your task in this question is to write the encryption and decryption functions for the Hill cipher, for matrices of size 2x2. This requires writing two functions:

1- Encryption scheme: *e_hill(plaintext,key)*

2- Decryption scheme: *d_hill(ciphertext,key)*

The key is a string which will be converted into a 2x2 matrix. For instance, the key 'abcd' will correspond to the matrix: `[[0,1],[2,3]]`. If the key is less than four characters, then the concept of a running key will be used. . For instance, the key 'abc' will correspond to the key: 'abca', and the key 'ab' will correspond to the key 'abab'.

One careful design consideration for the Hill cipher is that the key should be invertible mod 26, i.e. the key matrix should have an inverse in mod 26. If a key is passed, to the encryption or decryption function, and it does not have an inverse, then the function should print an error message and return an empty string.

The encryption function will pick a pair of characters from plaintext and continue until all characters are processed. If necessary, the plaintext could be appended with the character 'Q'. All nonalphabetical symbols should be ignored, i.e. no encryption is performed over them. The encryption function ignores the case of the plaintext characters and produces a ciphertext that is all upper case.

The decryption function also process the ciphertext in pairs. All nonalphabetical symbols will be ignored as described above. If any padding exist, it should be stripped before returning the recovered plaintext.

If you need to create utility function create them inside the solution_A4.py file not at the utilities_A4.py file.

The function definitions of the two functions are displayed below:

```
#-------------------------------------------------------------
# Parameters:   plaintext (str)
#               key (str)
# Return:       ciphertext (str)
# Description:  Encryption using Hill Cipher, 2x2 (mod 26)
#               key is a string consisting of 4 characters
#                   if key is too short, make it a running key
#                   if key is too long, use first 4 characters
#               Encrypts only alphabet
#               Case of characters can be ignored --> cipher is upper case
#               If necessary pad with 'Q'
# Errors:       if key is not inveritble or if plaintext is empty
#                   print error msg and return empty string
#-------------------------------------------------------------
def e_hill(plaintext,key):
     #your code here
     return ciphertext

#-------------------------------------------------------------
# Parameters:   ciphertext (str)
#               key (str)
# Return:       plaintext (str)
# Description:  Decryption using Hill Cipher, 2x2 (mod 26)
#               key is a string consisting of 4 characters
#                   if key is too short, make it a running key
#                   if key is too long, use first 4 characters
#               Decrypts only alphabet
#               Case of characters can be ignored --> plain is lower case
#               Remove padding of q's
# Errors:       if key is not inveritble or if ciphertext is empty
#                   print error msg and return empty string
#-------------------------------------------------------------
def d_hill(ciphertext,key):
    #your code here
    return plaintext
```

Running the testing function will give:

```
>>> test_q5()
-----------------------------------------
Testing Q5: Hill Cipher

key =  pear
```

```
plaintext =   Lunch
ciphertext=   LCVINM
plaintext2=   lunch

key =  apple
plaintext =   Brunch
ciphertext=   VUNBBD
plaintext2=   brunch

key =  fig
plaintext =   Breakfast
ciphertext=   LNUYMHOMPM
plaintext2=   breakfast

key =  mellon
plaintext =   dinner
Error(e_hill): key is not invertible
ciphertext=
Error(d_hill): invalid ciphertext
plaintext2=

key =  apricot
plaintext =   Lunch Bag
ciphertext=   OJEDP XMW
plaintext2=   lunch bag

key =  cherry
plaintext =   empty water bottle?
Error(e_hill): key is not invertible
ciphertext=
Error(d_hill): invalid ciphertext
plaintext2=

key =  prune
plaintext =   Full plates..of salad
ciphertext=   ZWOZ WBLNCC..JH KWJMFI
plaintext2=   full plates..of salad

-----------------------------------------
```