CNS LAB 6

SAHIL BONDRE: U18CO021

Implement N-Gram Hill Cipher and perform the following functions.
(1) Read encryption key and plain text and give cipher text in the output.
(2) Read decryption key and cipher text and give plain text in the output.

```
import math

import numpy as np


from itertools import zip_longest




def grouper(iterable, n, fillvalue=None):

    args = [iter(iterable)] * n

    return zip_longest(*args, fillvalue=fillvalue)




def recursive_read(allowed_input, message=""):

    # Recursively reads user input until input is not in allowd_input

    while True:

        user_input = input(message)

        if user_input in allowed_input:

            return user_input




def recursive_read_int(message=""):
```

```python
    # Recursively reads user input until input is not in allowd_input

    while True:

        user_input = input(message)

        try:

            value = int(user_input)

            return value

        except:

            pass



allowed_a = [1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25]

m = 26



def multiplicative_inverse(a):

    for i in range(1, m):

        remainder = ((i * m) + 1) % a

        if remainder == 0:

            return ((i * m) + 1) / a


    return 0



def perform_encryption():

    message = input("Enter message: ").upper()

    key = input("Enter key: ").upper()
```

```
    key_length = int(math.sqrt(len(key)))
    result = ""
    if key_length ** 2 != len(key):
        print("Error: Key cannot be formed into a matrix")
        exit()

    matrix = np.array([ord(x) - ord('A')
                  for x in key]).reshape((key_length, key_length))

    if np.linalg.det(matrix) == 0:
        print("Error: determinant is not invertible")
        exit()

    print(f"Matrix:\n {matrix}")
    message_chunks = grouper(message, key_length, "X")

    for v in message_chunks:
        vec = np.array([ord(x) - ord("A") for x in v]).reshape((key_length,
1))
        for c in np.matmul(matrix, vec):
            result += chr((c[0]) % 26 + ord("A"))
    print(f"Final string: {result}")



def mod_inverse(a, m):
    for x in range(1, m):
        if (((a % m) * (x % m)) % m == 1):
```

```python
            return x
    return -1


def adjoint(m):
    return (np.linalg.det(m) * np.linalg.inv(m))


def mod_inverse_matrix(mat, mod):
    adj = adjoint(mat)
    det = np.linalg.det(mat)
    det_inv = mod_inverse(det, mod)
    return ((det_inv * adj) % mod)


def perform_decryption():
    message = input("Enter message: ").upper()
    key = input("Enter key: ").upper()
    key_length = int(math.sqrt(len(key)))
    result = ""
    if key_length ** 2 != len(key):
        print("Error: Key cannot be formed into a matrix")
        exit()

    matrix = np.array([ord(x) - ord('A')
                      for x in key]).reshape((key_length, key_length))
```

```
    if np.linalg.det(matrix) == 0:
        print("Error: determinant is not invertible")
        exit()

    matrix = mod_inverse_matrix(matrix, 26)
    print(f"Matrix:\n {matrix}")
    message_chunks = grouper(message, key_length, "X")

    for v in message_chunks:
        vec = np.array([ord(x) - ord("A") for x in v]).reshape((key_length,
1))
        for c in np.matmul(matrix, vec):
            result += chr(round(c[0]) % 26 + ord("A"))
    print(f"Final string: {result}")


is_encrypt = recursive_read(
    ["e", "d"], "Enter 'e' for encryption or 'd' for decryption: ") == "e"


if is_encrypt:
    perform_encryption()
else:
    perform_decryption()
```

```
PS F:\code\github.com\godcrampy\college-notes\cns\lab-06> python .\hill.py
Enter 'e' for encryption or 'd' for decryption: e
Enter message: HELLOWORLD
Enter key: GYBNQKURP
Matrix:
 [[ 6 24  1]
 [13 16 10]
 [20 17 15]]
Final string: TFJIPIJSGVNQ
PS F:\code\github.com\godcrampy\college-notes\cns\lab-06> python .\hill.py
Enter 'e' for encryption or 'd' for decryption: d
Enter message: TFJIPIJSGVNQ
Enter key: GYBNQKURP
Matrix:
 [[ 8.  5. 10.]
 [21.  8. 21.]
 [21. 12.  8.]]
Final string: HELLOWORLDXX
PS F:\code\github.com\godcrampy\college-notes\cns\lab-06>
```