CNS LAB 3

SAHIL BONDRE: U18CO021

Implement a menu driven program for 5 X 5 Playfair Cipher with following functions.

(a) Encrypt given plain text.

(b) Decrypt given cipher text.

Note: In encryption, if both characters occur neither in the same line nor in the same column of key matrix -> Then the first character is replaced by the character that occurs in the same line and in the column in which the second character occurs.

```
import math
from types import new_class
from tabulate import tabulate


dummy = "x"



def recursive_read_func(predicate, message=""):
    # Recursively reads user input until input is allowed by predicate
    while True:
        user_input = input(message)
        if predicate(user_input):
            return user_input



def recursive_read(allowed_input, message=""):
    # Recursively reads user input until input is not in allowed_input
    while True:
```

```
        user_input = input(message)
        if user_input in allowed_input:
            return user_input



def recursive_read_int(message=""):
    # 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



def file_to_str(filename):
    try:
        with open(filename, 'r') as file:
            return file.read()
    except:
        print("Error: File not found!")
        exit(1)



def make_alpha(message):
```

```
    # remove numbers, whitespace, special charecters
    message = message.lower()
    return ''.join([i for i in message if i.isalpha()]).replace('j', 'i')



def generate_matrix(key):
    m = []
    itr = 0
    next_alpha = 0
    used_alphabets = set()
    [used_alphabets.add(x) for x in key]
    used_alphabets.add("j")

    for i in range(5):
        row = []
        for j in range(5):
            if itr == len(key):
                # chose from remaining alphabet
                while chr(next_alpha + ord('a')) in used_alphabets:
                    next_alpha += 1

                row.append(chr(next_alpha + ord('a')))
                used_alphabets.add(chr(next_alpha + ord('a')))
            else:
                row.append(key[itr])
                itr += 1
```

```
        m.append(row)

    return m



def message_to_pair(message):
    i = 0
    res = []
    current_str = ""
    while True:
        if i == len(message):
            if len(current_str) == 1:
                current_str += dummy
            res.append(current_str)
            break
        else:
            if len(current_str) == 2:
                res.append(current_str)
                current_str = message[i]
                i += 1
            else:
                if message[i] == current_str:
                    res.append(current_str + dummy)
                    current_str = ""
                else:
```

```
                    current_str += message[i]
                    i += 1
    return res


def encrypt_pair(matrix, lmap, pair):
    lr, lc = lmap[pair[0]]
    rr, rc = lmap[pair[1]]
    if lr == rr:
        return matrix[lr][(lc + 1) % 5] + matrix[rr][(rc + 1) % 5]

    if lc == rc:
        return matrix[(lr + 1) % 5][lc] + matrix[(rr + 1) % 5][rc]

    return matrix[lr][rc] + matrix[rr][lc]


def decrypt_pair(matrix, lmap, pair):
    lr, lc = lmap[pair[0]]
    rr, rc = lmap[pair[1]]
    if lr == rr:
        return matrix[lr][(lc - 1) % 5] + matrix[rr][(rc - 1) % 5]

    if lc == rc:
        return matrix[(lr - 1) % 5][lc] + matrix[(rr - 1) % 5][rc]
```

```
    return matrix[lr][rc] + matrix[rr][lc]


def generate_lmap(matrix):
    d = {}
    for i in range(5):
        for j in range(5):
            d[matrix[i][j]] = (i, j)

    return d



def perform_encryption(message, key):
    m = generate_matrix(key)
    pairs = message_to_pair(message)
    lmap = generate_lmap(m)
    print(f"Initial Pairs: {pairs}")
    print(tabulate(m, tablefmt="grid"))

    new_pairs = []

    for pair in pairs:
        new_pairs.append(encrypt_pair(m, lmap, pair))

    print(f"Final Pairs: {new_pairs}")
    print(''.join(new_pairs).upper())
```

```
def perform_decryption(message, key):
    m = generate_matrix(key)
    pairs = message_to_pair(message)
    lmap = generate_lmap(m)
    print(f"Initial Pairs: {pairs}")
    print(tabulate(m, tablefmt="grid"))

    new_pairs = []

    for pair in pairs:
        new_pairs.append(decrypt_pair(m, lmap, pair))

    print(f"Final Pairs: {new_pairs}")
    print(f"{''.join(new_pairs).upper()} =>
{''.join(new_pairs).upper().replace(dummy.upper(), '')}")



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

key = recursive_read_func(lambda x: len(set(x)) == len(
    x) and x.isalpha(), "Enter key for algorithm: ")
key = key.lower()

message = make_alpha(input("Enter message: "))
```

```python
if is_encrypt:

    perform_encryption(message, key)

else:

    perform_decryption(message, key)
```

```
PS F:\code\github.com\godcrampy\college-notes\cns\lab-03> python .\playfair.py
Enter 'e' for encryption or 'd' for decryption: e
Enter key for algorithm: svnitxyz
Enter message: Hello World!
Initial Pairs: ['he', 'lx', 'lo', 'wo', 'rl', 'dx']
+---+---+---+---+---+
| s | v | n | i | t |
+---+---+---+---+---+
| x | y | z | a | b |
+---+---+---+---+---+
| c | d | e | f | g |
+---+---+---+---+---+
| h | k | l | m | o |
+---+---+---+---+---+
| p | q | r | u | w |
+---+---+---+---+---+
Final Pairs: ['lc', 'hz', 'mh', 'tw', 'nr', 'cy']
LCHZMHTWNRCY
PS F:\code\github.com\godcrampy\college-notes\cns\lab-03> python .\playfair.py
Enter 'e' for encryption or 'd' for decryption: d
Enter key for algorithm: svnitxyz
Enter message: LCHZMHTWNRCY
Initial Pairs: ['lc', 'hz', 'mh', 'tw', 'nr', 'cy']
+---+---+---+---+---+
| s | v | n | i | t |
+---+---+---+---+---+
| x | y | z | a | b |
+---+---+---+---+---+
| c | d | e | f | g |
+---+---+---+---+---+
| h | k | l | m | o |
+---+---+---+---+---+
| p | q | r | u | w |
+---+---+---+---+---+
Final Pairs: ['he', 'lx', 'lo', 'wo', 'rl', 'dx']
HELXLOWORLDX => HELLOWORLD
PS F:\code\github.com\godcrampy\college-notes\cns\lab-03>
```