21CSE1003 ASHISH SINGH

Q1. Write a program to encrypt and decrypt the message "Meet Me at the Bridge" using Play fair cipher where key is "Your Name".

Algorithm:

Input:

- **Key**: A string used to generate the Playfair matrix (e.g., "keyword").
- **Plaintext**: The message to be encrypted (e.g., "hello").
- **Ciphertext**: The encrypted message to be decrypted.

Output:

- **Ciphertext**: Encrypted message from the given plaintext.
- **Plaintext**: Decrypted message from the given ciphertext.

Steps:

1. Create Playfair Matrix:

Initialize an empty list l to store characters of the matrix.

Loop through each character in the key:

- If the character is not already in the list, and it's not 'i' or 'j', add it to the list.
- If the character is 'i' or 'j' and the combination 'i/j' is not already in the list, add 'i/j' to represent both 'i' and 'j'.

Complete the list l by adding remaining characters from the alphabet ('a' to 'z'), skipping characters already in the list.

Convert the list l into a 5x5 matrix.

2. Prepare Digraphs for Encryption (Preprocessing the Plaintext):

Loop through the plaintext in steps of two characters to create **digraphs** (two-letter blocks).

If both characters in a digraph are the same, insert an 'x' between them.

If the length of the plaintext is odd, append 'x' at the end to make it even.

Replace occurrences of 'i' with 'i/j' to handle both 'i' and 'j' together.

3. Encryption (Using the Matrix):

For each digraph:

- Find the positions of the two characters in the Playfair matrix.
- If the two characters are in the same row, replace each character with the one to its right (wrapping around if needed).

- If the two characters are in the same column, replace each character with the one below it (wrapping around if needed).
- If the two characters are neither in the same row nor in the same column, form a rectangle and swap the characters at the corners of the rectangle.

Continue this process for all digraphs to generate the **ciphertext**.

4. Decryption (Reverse Process):

Loop through the ciphertext in steps of two characters to form digraphs.

Use the same Playfair matrix to reverse the encryption process:

- If the two characters are in the same row, replace each with the character to its left (wrapping around if needed).
- If the two characters are in the same column, replace each with the character above it (wrapping around if needed).
- If the two characters are neither in the same row nor in the same column, swap the characters at the corners of the rectangle.

Replace 'i/j' occurrences back with 'i' to restore the original plaintext.

5. Return Results:

- After processing all digraphs, return the encrypted **ciphertext**.
- Use the reverse process to return the decrypted **plaintext** from the ciphertext.

LAB_6\play_fair.py

```python

# 21CSE1003
# Ashish Singh

# Q1. Write a program to encrypt and decrypt the message "Meet Me at the Bridge" using
# Play fair cipher where key is "Your Name".

import numpy as np

def index_2d(myList, v):
    for i, x in enumerate(myList):
        if v in x:
            return (i, x.index(v))


def play_fair_encryption(key, plaintext):
    # matrix
    l = []
    for char in key:
        if char not in l and (char != 'i' and char != 'j'):
            l.append(char)
        elif 'i/j' not in l and (char == 'i' or char == 'j'):
            l.append('i/j')
    for _ in ['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']:
        if _ not in l:
            l.append(_)
    try:
        matrix = np.array(l).reshape(5, 5)
    except:
        print("More than 25 characters!!!")

    print(f"\nMatrix:\n{matrix}")
    matrix = matrix.tolist()

    # digraph
    for s in range(0,len(plaintext)+1,2):
        if s < len(plaintext)-1:
            if plaintext[s] == plaintext[s+1]:
                plaintext = plaintext[:s+1] + 'x' + plaintext[s+1:]
    if len(plaintext) % 2 != 0:
        plaintext = plaintext[:] + 'x'

    print(f"\nNew plaintext: {plaintext}")

    plaintext = [i for i in plaintext]
    while 'i' in plaintext:
        index_of_i = plaintext.index('i')
        plaintext[index_of_i] = 'i/j'

    ciphertext = []

    for x in range(0, len(plaintext), 2):
        c1 = plaintext[x]
        c2 = plaintext[x+1]
        a, b = index_2d(matrix, c1)
        c, d = index_2d(matrix, c2)

        if a == c:
            ciphertext.append(matrix[a][(b + 1) % 5])
            ciphertext.append(matrix[c][(d + 1) % 5])
        elif b == d:
            ciphertext.append(matrix[(a + 1) % 5 ][b])
            ciphertext.append(matrix[(c + 1) % 5 ][d])

        elif a != c or b != d:
            ciphertext.append(matrix[a][d])
            ciphertext.append(matrix[c][b])

    ciphertext = [i for i in ciphertext]
    while 'i/j' in ciphertext:
        index_of_i = ciphertext.index('i/j')
        ciphertext[index_of_i] = 'i'

    return ''.join(ciphertext)

def play_fair_decryption(key, ciphertext):
    # matrix
    l = []
    for char in key:
        if char not in l and (char != 'i' and char != 'j'):
            l.append(char)
        elif 'i/j' not in l and (char == 'i' or char == 'j'):
            l.append('i/j')
    for _ in ['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']:
        if _ not in l:
            l.append(_)
    try:
        matrix = np.array(l).reshape(5, 5)
    except:
        print("More than 25 characters!!!")

    matrix = matrix.tolist()

    # digraph
    for s in range(0,len(ciphertext)+1,2):
        if s < len(ciphertext)-1:
            if ciphertext[s] == ciphertext[s+1]:
                ciphertext = ciphertext[:s+1] + 'x' + ciphertext[s+1:]
    if len(ciphertext) % 2 != 0:
        ciphertext = ciphertext[:] + 'x'

    print(f"\nNew ciphertext: {ciphertext}")

    ciphertext = [i for i in ciphertext]
    while 'i' in ciphertext:
        index_of_i = ciphertext.index('i')
        ciphertext[index_of_i] = 'i/j'

    plaintext = []

    for x in range(0, len(ciphertext), 2):
        c1 = ciphertext[x]
        c2 = ciphertext[x+1]
        a, b = index_2d(matrix, c1)
        c, d = index_2d(matrix, c2)

        if a == c:
            plaintext.append(matrix[a][(b - 1) % 5])
            plaintext.append(matrix[c][(d - 1) % 5])
        elif b == d:
            plaintext.append(matrix[(a - 1) % 5 ][b])
            plaintext.append(matrix[(c - 1) % 5 ][d])

        elif a != c or b != d:
            plaintext.append(matrix[a][d])
            plaintext.append(matrix[c][b])

    plaintext = [i for i in plaintext]
    while 'i/j' in plaintext:
        index_of_i = plaintext.index('i/j')
        plaintext[index_of_i] = 'i'

    return ''.join(plaintext)

k = [i for i in input("\nEnter key: ")]

p = input("Enter plaintext: ")

ciphertext = play_fair_encryption(k, p)
print(f"\nCiphertext: {ciphertext}\n")
print(f"\nPlaintext: {play_fair_decryption(k, ciphertext)}\n")

```

Q2. Write a program to encrypt and decrypt the message "Pay more money" using trigraph Hill Cipher where key is "GYBNQKURP".

Algorithm:

Input:

- **Key:** A string of 9 characters ("GYBNQKURP").
- **Message:** A plaintext string ("Pay more money").

Output:

- **Encrypted Message:** A ciphertext generated from the input message using the Hill cipher.
- **Decrypted Message:** The original plaintext message obtained after decrypting the ciphertext.

Steps for Encryption:

1. **Convert Key to Matrix:**
    - Convert each character of the key string into its corresponding numerical value (A=0, B=1, ..., Z=25).
    - Reshape the resulting list of numbers into a 3×33 \times 33×3 matrix (the key matrix).
2. **Preprocess Message:**
    - Remove all spaces from the message and convert it to uppercase.
    - If the length of the message is not divisible by 3, append 'X' characters to make the length a multiple of 3.
3. **Divide Message into Blocks**: Split the message into groups of 3 letters (each group forms a block).
4. **Encrypt Each Block**: For each block of 3 letters:
    - Convert each letter into its corresponding numerical value (A=0, B=1, ..., Z=25).
    - Create a column vector from these numerical values.
    - Multiply the key matrix with this column vector (mod 26).
    - Convert the resulting vector back to letters (mod 26) to get the encrypted block.
5. **Combine Encrypted Blocks:** Combine all the encrypted blocks to form the final encrypted message.

Steps for Decryption:

6. **Calculate Inverse of Key Matrix:** Compute the inverse of the key matrix using modular arithmetic (mod 26).
7. **Decrypt Each Block:** For each block of the encrypted message:

- Convert each letter into its corresponding numerical value.
- Create a column vector from these values.
- Multiply the inverse key matrix with this column vector (mod 26).
- Convert the resulting vector back to letters (mod 26) to get the decrypted block.

8. **Combine Decrypted Blocks:** Combine all the decrypted blocks to form the final decrypted message.

LAB_6\hill_cipher.py

```python

# Q2. Write a program to encrypt and decrypt the message "Pay more money" using
# trigraph Hill Cipher where key is "GYBNQKURP".

import numpy as np

def letter_to_number(letter):
    return ord(letter.upper()) - ord('A')

def number_to_letter(number):
    return chr((number % 26) + ord('A'))

def create_key_matrix(key):
    key = key.upper()
    key_matrix = []
    for letter in key:
        key_matrix.append(letter_to_number(letter))
    return np.array(key_matrix).reshape(3, 3)

def encrypt(message, key_matrix):
    message = message.replace(" ", "").upper()
    while len(message) % 3 ≠ 0:
        message += 'X'

    encrypted_message = ''
    for i in range(0, len(message), 3):
        block = message[i:i+3]
        message_vector = np.array([letter_to_number(letter) for letter in block]).reshape(3, 1)
        encrypted_vector = np.dot(key_matrix, message_vector) % 26
        encrypted_block = ''.join([number_to_letter(num) for num in encrypted_vector.flatten()])
        encrypted_message += encrypted_block
    return encrypted_message

def mod_inverse(matrix, modulus):
    det = int(np.round(np.linalg.det(matrix)))
    det_inv = pow(det, -1, modulus)
    matrix_mod_inv = det_inv * np.round(det * np.linalg.inv(matrix)).astype(int) % modulus
    return matrix_mod_inv

def decrypt(encrypted_message, key_matrix):
    key_matrix_inv = mod_inverse(key_matrix, 26)

    decrypted_message = ''
    for i in range(0, len(encrypted_message), 3):
        block = encrypted_message[i:i+3]
        encrypted_vector = np.array([letter_to_number(letter) for letter in block]).reshape(3, 1)
        decrypted_vector = np.dot(key_matrix_inv, encrypted_vector) % 26
        decrypted_block = ''.join([number_to_letter(num) for num in decrypted_vector.flatten()])
        decrypted_message += decrypted_block
    return decrypted_message

key = "GYBNQKURP"
message = "PAY MORE MONEY"

key_matrix = create_key_matrix(key)

encrypted_message = encrypt(message, key_matrix)
print(f"Encrypted message: {encrypted_message}")

decrypted_message = decrypt(encrypted_message, key_matrix)
print(f"Decrypted message: {decrypted_message}")

```