

Ex. No. 3	IMPLEMENT TRANSPOSITION CIPHER
Date of Exercise	28/08/2024

Aim:

To implement the Transposition Cipher techniques - Rail Fence Cipher and Columnar Transposition cipher.

Description:**1. Rail Fence Cipher**

In the rail fence cipher, the plain text is written downwards and diagonally on successive "rails" of an imaginary fence, then moving up when we reach the bottom rail. When we reach the top rail, the message is written downwards again until the whole plaintext is written out. The message is then read off in rows.

Plaintext T H I S I S A S E C R E T M E S S A G E

Rail Fence
Encoding
key = 4

T						A						T					G	
	H				S		S				E		M				A	E
		I		I				E		R				E		S		
			S						C						S			

Ciphertext T A T G H S S E M A E I I E R E S S C S

2. Columnar Transposition Cipher

Columnar Transposition involves writing the plaintext out in rows, and then reading the ciphertext off in columns one by one.

F A N C Y	←	Key
3 1 4 2 5	←	order in alphabet
m e e t m	←	plaintext is written acrosswise
e a t n e		
x t m i d		
n i g h t		
	↑	ciphertext is read column-wise, this column first

Questions:**1. Perform encryption and decryption using rail fence cipher****Algorithm:**

- Read the problem statement
- Get the plaintext and number of rails from the user
- Create a 2D list (matrix) with k rows and columns equal to the length of the plaintext.
- Use a zigzag pattern to place each character of the plaintext into the rails
- Concatenate characters from each row to form the ciphertext and print the ciphertext
- Create a 2D list (matrix) with k rows and columns equal to the length of the ciphertext for decryption
- Use the zigzag pattern to mark positions in the matrix where characters of the ciphertext will be placed.
- Fill the marked positions in the matrix with characters from the ciphertext.
- Read characters in a zigzag pattern from the matrix to reconstruct the plaintext and print the plaintext

Program:

```
def rail_fence_encrypt(text, rails):
    rail = [" " for _ in range(len(text))] for _ in range(rails)
    direction = None
    row, col = 0, 0
    for char in text:
        if row == 0 or row == rails - 1:
            direction = not direction
        rail[row][col] = char
        col += 1
        row += 1 if direction else -1

    result = ".join(['.join(r) for r in rail])
    return result.replace('\n', ")

def rail_fence_decrypt(cipher, rails):
    n = len(cipher)
    rail = [['\n' for _ in range(n)] for _ in range(rails)]
    dir_down = None
    row, col = 0, 0

    for i in range(n):
        if row == 0 or row == rails - 1:
            dir_down = not dir_down

        rail[row][col] = '*'
        col += 1

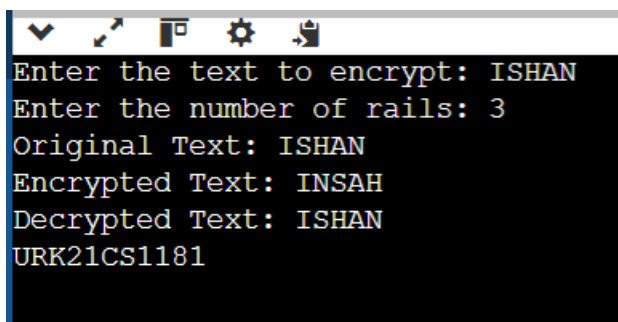
        if dir_down:
            row += 1
        else:
            row -= 1
    index = 0
    for i in range(rails):
        for j in range(n):
```

```
        if rail[i][j] == '*' and index < n:
            rail[i][j] = cipher[index]
            index += 1
    result = []
    row, col = 0, 0
    dir_down = None

    for i in range(n):
        if row == 0 or row == rails - 1:
            dir_down = not dir_down

        if rail[row][col] != '*':
            result.append(rail[row][col])
            col += 1
        if dir_down:
            row += 1
        else:
            row -= 1

    return ''.join(result)
text = input("Enter the text to encrypt: ")
rails = int(input("Enter the number of rails: "))
encrypted = rail_fence_encrypt(text, rails)
decrypted = rail_fence_decrypt(encrypted, rails)
print(f"Original Text: {text}")
print(f"Encrypted Text: {encrypted}")
print(f"Decrypted Text: {decrypted}")
print("URK21CS1187 Thaha")
```

Output:

```
Enter the text to encrypt: ISHAN
Enter the number of rails: 3
Original Text: ISHAN
Encrypted Text: INSAH
Decrypted Text: ISHAN
URK21CS1181
```

Alternative**Program:**

```
def rail_fence_encrypt(text, rails):
    if rails < 2:
        return text
    fence = [[' ' for _ in range(len(text))] for _ in range(rails)]
    rail = 0
    for i, char in enumerate(text):
        fence[rail][i] = char
        rail += 1
        if rail == rails: # Restart from the top when reaching the last rail
            rail = 0
    cipher_text = ''.join(char for rail in fence for char in rail if char != ' ')
    return cipher_text

def rail_fence_decrypt(cipher_text, rails):
    if rails < 2:
        return cipher_text
    fence = [[' ' for _ in range(len(cipher_text))] for _ in range(rails)]
    rail = 0
    # Mark the positions that will contain characters
    for i in range(len(cipher_text)):
        fence[rail][i] = '*'
        rail += 1
        if rail == rails: # Restart from the top
            rail = 0

    # Fill the marked positions with the cipher text
    index = 0
    for i in range(rails):
        for j in range(len(cipher_text)):
            if fence[i][j] == '*' and index < len(cipher_text):
                fence[i][j] = cipher_text[index]
                index += 1
```

```
# Now, read the fence to get the original text
plain_text = ""
rail = 0
for i in range(len(cipher_text)):
    plain_text += fence[rail][i]
    rail += 1
    if rail == rails: # Restart from the top
        rail = 0
return plain_text

def visualize_encryption(text, rails):
    fence = [[' ' for _ in range(len(text))] for _ in range(rails)]
    rail = 0
    for i, char in enumerate(text):
        fence[rail][i] = char
        rail += 1
        if rail == rails:
            rail = 0
    for rail in fence:
        print("".join(rail).rstrip())

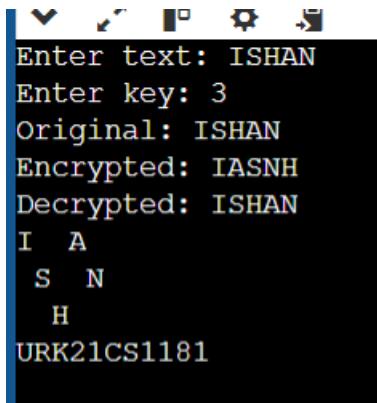
print("URK21CS1187 Thaha")
text = input("Enter text: ")
rail_fence_key = int(input("Enter key: "))

rail_fence_encrypted = rail_fence_encrypt(text, rail_fence_key)
rail_fence_decrypted = rail_fence_decrypt(rail_fence_encrypted, rail_fence_key)

print("Original:", text)
print("Encrypted:", rail_fence_encrypted)
print("Decrypted:", rail_fence_decrypted)

visualize_encryption(text, rail_fence_key)
```

Output:



```
Enter text: ISHAN
Enter key: 3
Original: ISHAN
Encrypted: IASNH
Decrypted: ISHAN
I  A
S  N
H
URK21CS1181
```

2. Perform encryption using columnar transposition technique

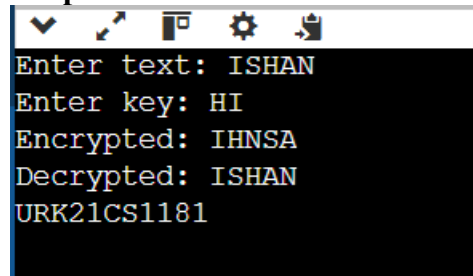
Algorithm:

- Read the problem statement
- Get the plaintext and key from the user
- Assign a numerical position to each character in the key based on alphabetical order.
- Sort the key characters by these positions and assign new sequential numbers.
- Create a matrix with rows equal to the length of the plaintext divided by the length of the key and columns equal to the length of the key.
- Fill the matrix row by row with characters from the plaintext.
- Rearrange columns of the matrix according to the numerical positions assigned to the key characters.
- Read the rearranged matrix column by column to form the ciphertext and print the ciphertext.

Program:

```
import math
def columnar_transposition_encrypt(text, key):
    num_cols = len(key)
    num_rows = math.ceil(len(text) / num_cols)
    grid = [" " for _ in range(num_cols)] for _ in range(num_rows)]
    for i in range(len(text)):
        row = i // num_cols
```

```
        col = i % num_cols
        grid[row][col] = text[i]
    key_order = sorted(range(len(key)), key=lambda k: key[k])
    encrypted_text = ".join(
        grid[row][key_order[col]] for col in range(num_cols) for row in range(num_rows)
        if grid[row][key_order[col]] != "
    )
    return encrypted_text
def columnar_transposition_decrypt(cipher, key):
    num_cols = len(key)
    num_rows = math.ceil(len(cipher) / num_cols)
    grid = [["_ " for _ in range(num_cols)] for _ in range(num_rows)]
    key_order = sorted(range(len(key)), key=lambda k: key[k])
    index = 0
    for col in key_order:
        for row in range(num_rows):
            if index < len(cipher):
                grid[row][col] = cipher[index]
                index += 1
    decrypted_text = ".join(grid[row][col] for row in range(num_rows) for col in
range(num_cols) if grid[row][col] != "
    return decrypted_text
text = input("Enter text:")
key = input("Enter key:")
cipher = columnar_transposition_encrypt(text, key)
print(f"Encrypted: {cipher}")
print(f"Decrypted: {columnar_transposition_decrypt(cipher, key)}")
print("URK21CS1187 Thaha")
```

Output:

```
Enter text: ISHAN
Enter key: HI
Encrypted: IHNSA
Decrypted: ISHAN
URK21CS1181
```


Result:

Therefore, we have successfully implemented the transposition ciphers - Rail Fence Cipher and Columnar Transposition Cipher.