Name - Khushi Nitinkumar Patel
PRN - 2020BTECS00037
Batch - B3

Assignment no 2: Implementation of Transposition Cipher

Introduction

a) **Columnar Transposition**
The Columnar Transposition Cipher is a form of transposition cipher just like Rail Fence Cipher. Columnar Transposition involves writing the plaintext out in rows, and then reading the ciphertext off in columns one by one.

Encryption

In a transposition cipher, the order of the alphabets is rearranged to obtain the cipher-text.

1. The message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order.
2. Width of the rows and the permutation of the columns are usually defined by a keyword.
3. For example, the word HACK is of length 4 (so the rows are of length 4), and the permutation is defined by the alphabetical order of the letters in the keyword. In this case, the order would be “3 1 2 4”.
4. Any spare spaces are filled with nulls or left blank or placed by a character (Example: _).
5. Finally, the message is read off in columns, in the order specified by the keyword.

Decryption

1. To decipher it, the recipient has to work out the column lengths by dividing the message length by the key length.
2. Then, write the message out in columns again, then re-order the columns by reforming the key word.

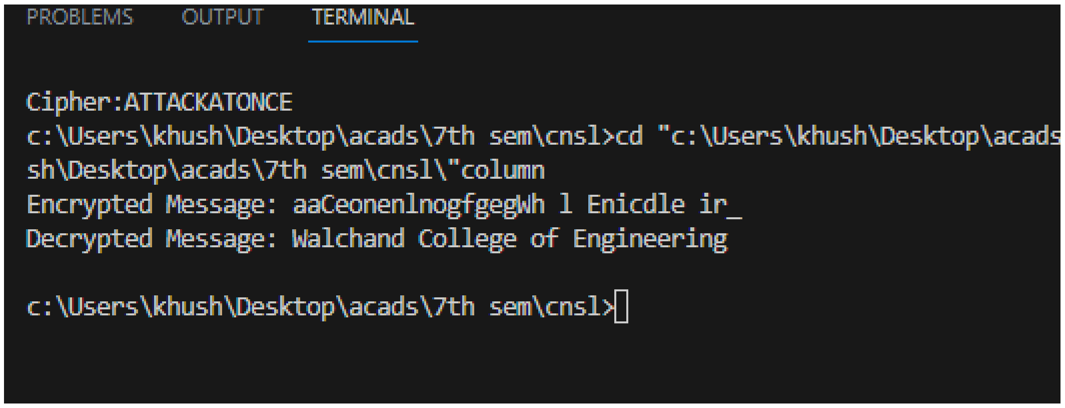
Encryption and Decryption Code:

```
// CPP program for illustrating
// Columnar Transposition Cipher
#include<bits/stdc++.h>
using namespace std;

// Key for Columnar Transposition
string const key = "HACK";
map<int,int> keyMap;

void setPermutationOrder()
{
    // Add the permutation order into map
    for(int i=0; i < key.length(); i++)
    {
        keyMap[key[i]] = i;
    }
}

// Encryption
string encryptMessage(string msg)
{
    int row,col,j;
    string cipher = "";

    /* calculate column of the matrix*/
    col = key.length();

    /* calculate Maximum row of the matrix*/
    row = msg.length()/col;

    if (msg.length() % col)
        row += 1;

    char matrix[row][col];

    for (int i=0,k=0; i < row; i++)
    {
        for (int j=0; j<col; )
        {
            if(msg[k] == '\0')
            {
                /* Adding the padding character '_' */
                matrix[i][j] = '_';
                j++;
            }

            if( isalpha(msg[k]) || msg[k]==' ')
            {
                /* Adding only space and alphabet into matrix*/
                matrix[i][j] = msg[k];
                j++;
            }
            k++;
        }
    }

    for (map<int,int>::iterator ii = keyMap.begin(); ii!=keyMap.end();
++ii)
    {
        j=ii->second;

        // getting cipher text from matrix column wise using permuted key
        for (int i=0; i<row; i++)
        {
            if( isalpha(matrix[i][j]) || matrix[i][j]==' ' ||
matrix[i][j]=='_')
                cipher += matrix[i][j];
        }
    }

    return cipher;
}

// Decryption
string decryptMessage(string cipher)
{
    /* calculate row and column for cipher Matrix */
    int col = key.length();
```

```cpp
    int row = cipher.length()/col;
    char cipherMat[row][col];

    /* add character into matrix column wise */
    for (int j=0,k=0; j<col; j++)
        for (int i=0; i<row; i++)
            cipherMat[i][j] = cipher[k++];

    /* update the order of key for decryption */
    int index = 0;
    for( map<int,int>::iterator ii=keyMap.begin(); ii!=keyMap.end(); ++ii)
        ii->second = index++;

    /* Arrange the matrix column wise according
    to permutation order by adding into new matrix */
    char decCipher[row][col];
    map<int,int>::iterator ii=keyMap.begin();
    int k = 0;
    for (int l=0,j; key[l]!='\0'; k++)
    {
        j = keyMap[key[l++]];
        for (int i=0; i<row; i++)
        {
            decCipher[i][k]=cipherMat[i][j];
        }
    }

    /* getting Message using matrix */
    string msg = "";
    for (int i=0; i<row; i++)
    {
        for(int j=0; j<col; j++)
        {
            if(decCipher[i][j] != '_')
                msg += decCipher[i][j];
        }
    }
    return msg;
}
```

```
// Driver Program
int main(void)
{
    /* message */
    string msg = "Walchand College of Engineering";

    setPermutationOrder();

    // Calling encryption function
    string cipher = encryptMessage(msg);
    cout << "Encrypted Message: " << cipher << endl;

    // Calling Decryption function
    cout << "Decrypted Message: " << decryptMessage(cipher) << endl;

    return 0;
}
```

Output:



```
Cipher:ATTACKATONCE
c:\Users\khush\Desktop\acads\7th sem\cnsl>cd "c:\Users\khush\Desktop\acads
sh\Desktop\acads\7th sem\cnsl\"column
Encrypted Message: aaCeonenlnogfgegWh l Enicdle ir_
Decrypted Message: Walchand College of Engineering

c:\Users\khush\Desktop\acads\7th sem\cnsl>
```

b) Rail Fence cipher

The rail fence cipher (also called a zigzag cipher) is a form of transposition cipher. It derives its name from the way in which it is encoded**.**

Encryption

In a transposition cipher, the order of the alphabets is rearranged to obtain the cipher-text.

1. In the rail fence cipher, the plain-text is written downwards and diagonally on successive rails of an imaginary fence.
2. When we reach the bottom rail, we traverse upwards moving diagonally, after reaching the top rail, the direction is changed again. Thus the alphabets of the message are written in a zig-zag manner.
3. After each alphabet has been written, the individual rows are combined to obtain the cipher-text.

Decryption

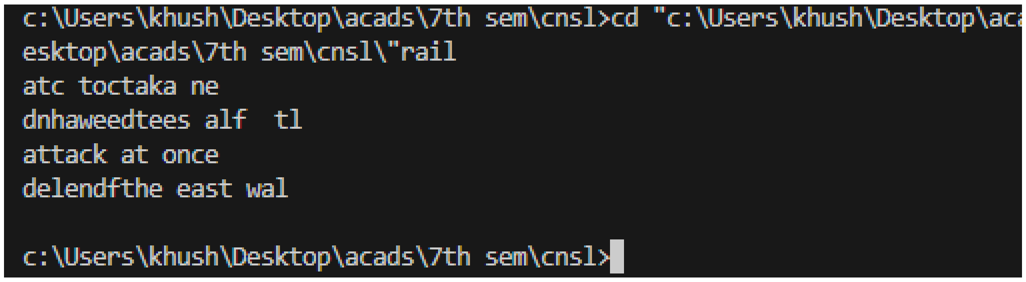
The number of columns in rail fence cipher remains equal to the length of plain-text message. And the key corresponds to the number of rails.

1. Hence, rail matrix can be constructed accordingly. Once we've got the matrix we can figure-out the spots where texts should be placed (using the same way of moving diagonally up and down alternatively).
2. Then, we fill the cipher-text row wise. After filling it, we traverse the matrix in zig-zag manner to obtain the original text.

Encryption and Decryption Code:

```
// C++ program to illustrate Rail Fence Cipher
// Encryption and Decryption
#include <bits/stdc++.h>
using namespace std;

// function to encrypt a message
string encryptRailFence(string text, int key)
{
    // create the matrix to cipher plain text
    // key = rows , length(text) = columns
    char rail[key][(text.length())];

    // filling the rail matrix to distinguish filled
    // spaces from blank ones
    for (int i=0; i < key; i++)
        for (int j = 0; j < text.length(); j++)
            rail[i][j] = '\n';

    // to find the direction
    bool dir_down = false;
    int row = 0, col = 0;

    for (int i=0; i < text.length(); i++)
    {
        // check the direction of flow
        // reverse the direction if we've just
        // filled the top or bottom rail
        if (row == 0 || row == key-1)
            dir_down = !dir_down;

        // fill the corresponding alphabet
        rail[row][col++] = text[i];

        // find the next row using direction flag
        dir_down?row++ : row--;
    }

    //now we can construct the cipher using the rail matrix
    string result;
    for (int i=0; i < key; i++)
        for (int j=0; j < text.length(); j++)
            if (rail[i][j]!='\n')
                result.push_back(rail[i][j]);

    return result;
}

// This function receives cipher-text and key
// and returns the original text after decryption
string decryptRailFence(string cipher, int key)
{
    // create the matrix to cipher plain text
    // key = rows , length(text) = columns
    char rail[key][cipher.length()];

    // filling the rail matrix to distinguish filled
    // spaces from blank ones
    for (int i=0; i < key; i++)
        for (int j=0; j < cipher.length(); j++)
            rail[i][j] = '\n';

    // to find the direction
    bool dir_down;

    int row = 0, col = 0;

    // mark the places with '*'
    for (int i=0; i < cipher.length(); i++)
    {
        // check the direction of flow
        if (row == 0)
            dir_down = true;
        if (row == key-1)
            dir_down = false;

        // place the marker
        rail[row][col++] = '*';

        // find the next row using direction flag
        dir_down?row++ : row--;
    }

    // now we can construct the fill the rail matrix
    int index = 0;
    for (int i=0; i<key; i++)
        for (int j=0; j<cipher.length(); j++)
            if (rail[i][j] == '*' && index<cipher.length())
                rail[i][j] = cipher[index++];


    // now read the matrix in zig-zag manner to construct
    // the resultant text
    string result;

    row = 0, col = 0;
    for (int i=0; i< cipher.length(); i++)
    {
        // check the direction of flow
        if (row == 0)
            dir_down = true;
        if (row == key-1)
            dir_down = false;

        // place the marker
        if (rail[row][col] != '*')
            result.push_back(rail[row][col++]);

        // find the next row using direction flag
        dir_down?row++: row--;
    }
    return result;
}

//driver program to check the above functions
int main()
{
    cout << encryptRailFence("attack at once", 2) << endl;
    cout << encryptRailFence("defend the east wall", 3) << endl;
    //Now decryption of the same cipher-text

    cout << decryptRailFence("atc toctaka ne",2) << endl;
    cout << decryptRailFence("dnhaweedtees alf tl",3) << endl;

    return 0;
}
```

Output:



```
c:\Users\khush\Desktop\acads\7th sem\cnsl>cd "c:\Users\khush\Desktop\aca
esktop\acads\7th sem\cnsl\"rail
atc toctaka ne
dnhaweedtees alf  tl
attack at once
delendfthe east wal

c:\Users\khush\Desktop\acads\7th sem\cnsl>
```