Cryptography and Network Security
Assignment 2: Trransposition cipher
Name: Mrunal Khade.
PRN: 2020BTECS00057

Q.1 Implementation of Rail Fence Cipher.

Theory: The Rail Fence Cipher, also known as the Zigzag Cipher, is a transposition cipher that encrypts a message by writing it in a zigzag pattern on a set number of "rails" or rows, and then reading off the ciphertext row by row.

Algorithm:

Encryption Algorithm (Rail Fence Cipher)

1. Start with a plaintext message and determine the number of rails (rows) for the fence.
2. Create an empty matrix (2D array) with the number of rows equal to the number of rails and the number of columns equal to the length of the plaintext message.
3. Initialize a variable `rail` to 0 and a variable `direction` to 1. The `rail` variable keeps track of the current rail, and the `direction` variable determines whether you're moving up (1) or down (-1) the rails.
4. For each character in the plaintext message, do the following:
    - Place the character in the matrix at the current `rail` and the current column.
    - Update the `rail` by adding `direction`.
    - If the `rail` reaches 0 or the maximum number of rails minus 1, change the `direction` to its opposite value (if it was going up, make it go down, and vice versa).

5. After all characters are placed in the matrix, read off the ciphertext by concatenating the characters row by row.
6. The resulting ciphertext is the encrypted message.

Decryption Algorithm (Rail Fence Cipher)

1. Start with the ciphertext message and determine the number of rails (rows) used for encryption.
2. Create an empty matrix (2D array) with the same number of rows as in the encryption step and the number of columns equal to the length of the ciphertext message.
3. Initialize a variable `rail` to 0 and a variable `direction` to 1, just like in the encryption step.
4. For each position in the matrix, mark it as a valid position (e.g., by placing a symbol like '*') if it corresponds to a character in the ciphertext. You'll do this by following the same zigzag pattern and direction as in encryption.
5. Place each character from the ciphertext into the marked positions of the matrix in the same zigzag pattern as used in encryption. Start at the top-left corner of the matrix.
6. As you place characters, keep track of the current `rail` and `direction` and update them accordingly. If the `rail` reaches 0 or the maximum number of rails minus 1, change the `direction` as needed.
7. After placing all characters from the ciphertext, read off the plaintext by traversing the matrix in a zigzag pattern and concatenating the characters row by row.
8. The resulting plaintext is the decrypted message.

Input and Output:

2_1.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;

string encryptRailFence(string text, int key){
    char rail[key][(text.length())];

    for (int i=0; i < key; i++)
        for (int j = 0; j < text.length(); j++)
            rail[i][j] = '\n';

    bool dir_down = false;
    int row = 0, col = 0;

    for (int i=0; i < text.length(); i++){
        if (row == 0 || row == key-1)
            dir_down = !dir_down;

        rail[row][col++] = text[i];
        dir_down?row++ : row--;
    }

    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;
}

string decryptRailFence(string cipher, int key){
    char rail[key][cipher.length()];

    for (int i=0; i < key; i++)
        for (int j=0; j < cipher.length(); j++)
            rail[i][j] = '\n';

    bool dir_down;
    int row = 0, col = 0;
```

2_1.cpp

```
    bool dir_down;
    int row = 0, col = 0;


    for (int i=0; i < cipher.length(); i++){
        if (row == 0)
            dir_down = true;
        if (row == key-1)
            dir_down = false;

        rail[row][col++] = '*';
        dir_down?row++ : row--;
    }

    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++];

    string result;
    row = 0, col = 0;
    for (int i=0; i< cipher.length(); i++){
        if (row == 0)
            dir_down = true;

        if (row == key-1)
            dir_down = false;

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

        dir_down?row++: row--;
    }
    return result;
}

int main(){
    string s;
```

2_1.cpp

```cpp
int main(){
    string s;
    cout<<"Enter 1 for encryption and 2 for decryption: ";
    int x;
    int key;
    cin>>x;

    switch (x){
    case 1:
        cout<<"Plain text : ";
        cin>>s;

        cout<<"Key : ";
        cin>>key;
        cout << encryptRailFence(s, key) << endl;
        break;
    case 2:
        cout<<"Cipher text : ";
        cin>>s;

        cout<<"Key : ";
        cin>>key;
        cout << decryptRailFence(s, key) << endl;
        break;
    default:
        break;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

MaarnlhduKe
mrunal@mrunal:~/Desktop/CNS$ cd "/home/mrunal/Desktop/CNS/" && g++ 2_1.cpp -o 2_1 && "/home/mrunal/Desktop/CNS/"2_1
Enter 1 for encryption and 2 for decryption: 1
Plain text : MrunalKhade
Key : 3
MaarnlhduKe
mrunal@mrunal:~/Desktop/CNS$
```

Ln 103, Col 2 Tab Size: 4 UTF-8 LF C++

Output:

```
mrunal@mrunal:~/Desktop/CNS$ cd "/home/mrunal/Desktop/CNS/" && g++ 2_1.cpp -o 2_1 && "/home/mrunal/Desktop/CNS/"2_1
Enter 1 for encryption and 2 for decryption: 1
Plain text : MrunalKhade
Key : 3
MaarnlhduKe
mrunal@mrunal:~/Desktop/CNS$ cd "/home/mrunal/Desktop/CNS/" && g++ 2_1.cpp -o 2_1 && "/home/mrunal/Desktop/CNS/"2_1
Enter 1 for encryption and 2 for decryption: 2
Cipher text : MaarnlhduKe
Key : 3
MrunalKhade
mrunal@mrunal:~/Desktop/CNS$
```

Ln 103, Col 2 Tab Size: 4 UTF-8 LF C++

Columnar Transposition:

Algorithm for Columnar Transposition Cipher:

- **Encryption:**

1. Start with the plaintext message and a keyword.
2. Create a table/grid with the number of rows equal to the length of the keyword and the number of columns equal to the length of the plaintext divided by the length of the keyword (rounded up).
3. Fill in the table by writing the plaintext message column by column, following the order of the keyword. If a column is not completely filled, pad it with a character (e.g., 'X').
4. Read off the ciphertext by reading the table row by row.

- **Decryption:**

1. Start with the ciphertext message and the keyword used for encryption.
2. Create a table/grid with the same number of rows as the length of the keyword and the number of columns equal to the length of the ciphertext divided by the length of the keyword (rounded up).
3. Calculate the number of characters in the last column of the table.
4. Fill in the table by writing the ciphertext message row by row, following the order of the keyword.
5. Read off the plaintext by reading the table column by column.

Input:

2-2.cpp

```cpp
//2020BTECS00057


#include<bits/stdc++.h>
using namespace std;

string encryptMessage(string msg,string const key,map<int,int> keyMap){
    int row,col,j;
    string cipher = "";

    col = key.length();
    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'){
                matrix[i][j] = '_';
                j++;
            }

            if( isalpha(msg[k]) || msg[k]==' '){
                matrix[i][j] = msg[k];
                j++;
            }
            k++;
        }
    }cout<<endl;

    cout<<"Matrix : "<<endl;
    for(int i=0; i < row; i++){
        for(int j=0; j<col; j++){
            cout<<matrix[i][j]<<" ";
        }cout<<endl;
    }cout<<endl;

```

2-2.cpp

```
        for(int j=0; j<col; j++){
            cout<<matrix[i][j]<<" ";
        }cout<<endl;
    }cout<<endl;

    for (auto& it:keyMap){
        j=it.second;
        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,string const key,map<int,int> keyMap){
    int col = key.length();
    int row = cipher.length()/col;
    char cipherMat[row][col];

    for (int j=0,k=0; j<col; j++)
        for (int i=0; i<row; i++)
            cipherMat[i][j] = cipher[k++];

    int index = 0;
    for( auto& it: keyMap)
        it.second = index++;

    char decCipher[row][col];
    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];
        }
    }

```

2-2.cpp

```cpp
    }

    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;
}

int main(){
    string msg;
    cout<<"Enter string: ";
    getline(cin, msg);

    string key;
    cout<<"Enter key: ";
    getline(cin, key);
    map<int,int> keyMap;

    for(int i=0; i < key.length(); i++){
        keyMap[key[i]] = i;
    }

    // for(auto it:keyMap){
    //     cout<<it.first<<" "<<it.second<<endl;
    // }

    string cipher = encryptMessage(msg,key,keyMap);
    cout << "Encrypted Message: " << cipher << endl;
    cout<<endl;
    cout << "Decrypted Message: " << decryptMessage(cipher,key,keyMap) << endl;

    return 0;
}
```

OutPut:

```
mrunal@mrunal:~/Desktop/CNS$ cd "/home/mrunal/Desktop/CNS/" && g++ 2-2.cpp -o 2-2 && "/home/mrunal/Desktop/CNS/"2-2
Enter string: MrunalKhade
Enter key: 2314

Matrix :
M r u n
a l K h
a d e _

Encrypted Message: uKeMaarldnh_

Decrypted Message: MrunalKhade
```

Conclusion:

Performed the experiment successfully. Encrypted the data with the provided key. Output of this encryption is decrypted to match the

plaintext that was inputted by the user as shown in the above diagram.