CNS LAB

Name: Samrat Vishwas Jadhav
PRN: 2020BTECS00006

Assignment 2

Aim - Given the plain text, encrypt it using Columnar Encryption Algorithm

Columnar Cipher Encryption Algorithm

In a transposition cipher, the order of the alphabets is re-arranged 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 6.

Code:

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

#define ll long long
```

```cpp
int main()
{
    string plainText, key;

    cout << "\n Choose an option:\n";
    cout << "  1. Encryption\n";
    cout << "  2. Decryption\n";
    int choice;
    cin >> choice;
    cin.ignore();

    if (choice == 1)
    {
        // Encryption
        cout << "\n Enter plain text : ";
        getline(cin, plainText);

        cout << "\n Enter key : ";
        getline(cin, key);

        // Removing spaces and converting to lowercase from
plaintext
        string temp = "";
        for (int i = 0; i < plainText.size(); i++)
        {
            if (plainText[i] != ' ')
                temp += tolower(plainText[i]);
        }
        plainText = temp;

        // Removing spaces and converting to lowercase from key
        string temp2 = "";
        for (int i = 0; i < key.size(); i++)
        {
            if (key[i] != ' ')
                temp2 += tolower(key[i]);
```

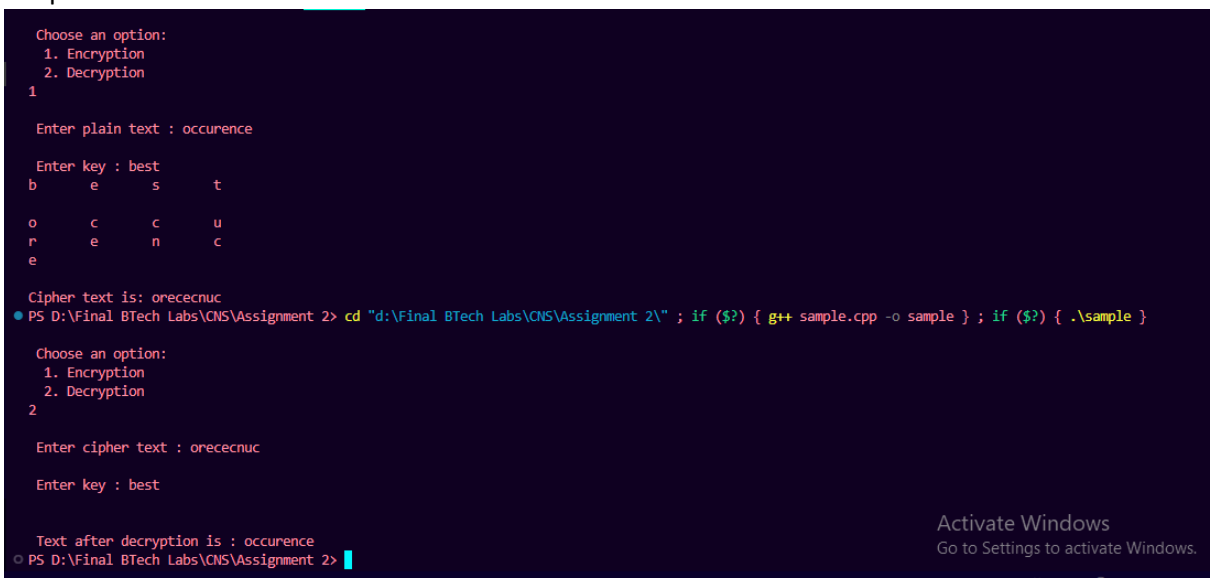
```
        }
        key = temp2;

        // Encryption
        map<char, vector<char>> mp;
        int keyCounter = 0;

        for (int i = 0; i < plainText.size(); i++)
        {
            mp[key[keyCounter++]].push_back(plainText[i]);

            if (keyCounter == key.size())
                keyCounter = 0;
        }

        // Print the key letters at the top
        for (const auto &it : mp)
        {
            cout << it.first << "\t";
        }
        cout << endl;
        cout<<endl;

        int maxColumnSize = 0;
        for (const auto &it : mp)
        {
            maxColumnSize = max(maxColumnSize,
static_cast<int>(it.second.size()));
        }

        for (int i = 0; i < maxColumnSize; i++)
        {
            for (const auto &it : mp)
            {
                if (i < it.second.size())
                {
                    cout << it.second[i] << "\t";
```

```
                }
                else
                {
                    cout << " \t"; // Fill with spaces if
there's no value in this column
                }
            }
            cout << endl;
        }
        cout << "\nCipher text is: ";
        string cipherText;
        for (const auto &it : mp)
        {
            for (char c : it.second)
            {
                cipherText += c;
            }
        }
        cout << cipherText << endl;
    }
    else if (choice == 2)
    {
        // Decryption
        cout << "\n Enter cipher text : ";
        getline(cin, plainText);

        cout << "\n Enter key : ";
        getline(cin, key);

        // Removing spaces and converting to lowercase from key
        string temp2 = "";
        for (int i = 0; i < key.size(); i++)
        {
            if (key[i] != ' ')
                temp2 += tolower(key[i]);
        }
        key = temp2;
```

```
        // Decryption
        map<int, int> dmp;

        int common = plainText.size() / key.size();
        int extra = plainText.size() % key.size();

        for (int i = 0; i < key.size(); i++)
        {
            if (i < extra)
                dmp[i] = common + 1;
            else
                dmp[i] = common;
        }

        map<int, vector<char>> dmp2;

        int start = 0;

        string sortedKey = key;
        sort(sortedKey.begin(), sortedKey.end());

        for (int i = 0; i < sortedKey.size(); i++)
        {
            for (int j = 0; j < key.size(); j++)
            {
                if (sortedKey[i] == key[j])
                {
                    for (int k = 0; k < dmp[j]; k++)
                    {
                        dmp2[key[j]].push_back(plainText[start+
+]);
                    }
                }
            }
        }
```

```
        string afterDecryption;

        vector<int> counters(key.size(), 0);

        int i = 0;

        while (afterDecryption.size() < plainText.size())
        {
            for (int i = 0; i < key.size(); i++)
            {
                if (counters[i] < dmp[i])
                    afterDecryption +=
dmp2[key[i]][counters[i]++];
            }
        }

        cout << "\n\n Text after decryption is : " <<
afterDecryption << endl;
    }
    else
    {
        cout << "\n Invalid choice" << endl;
    }

    return 0;
}
```

Output:



Rail fence Cipher Encryption Algorithm

In the rail fence cipher, the plain-text is written downwards and diagonallyon successive rails of an imaginary fence.

When we reach the bottom rail, we traverse upwards moving diagonally, after reachingthe top rail, the direction is changed again. Thus the alphabets of the message are written in a zig-zag manner.

After each alphabet has been written, the individual rows are combined to obtain thecipher-text.

Code:

```
def encryptRailFence(text, key):


    rail = [['*' for i in range(len(text))]
                  for j in range(key)]
```

```
    dir_down = False
    row, col = 0, 0

    for i in range(len(text)):


        if (row == 0) or (row == key - 1):
            dir_down = not dir_down

        rail[row][col] = text[i]
        col += 1

        if dir_down:
            row += 1
        else:
            row -= 1

    result = []
    for i in range(key):
        for j in range(len(text)):
            print(rail[i][j],end=" ")
            if rail[i][j] != '*':
                result.append(rail[i][j])
        print()
    return("" . join(result))



def decryptRailFence(cipher, key):

    rail = [['\n' for i in range(len(cipher))]
                for j in range(key)]


    dir_down = None
    row, col = 0, 0
```

```
for i in range(len(cipher)):
    if row == 0:
        dir_down = True
    if row == key - 1:
        dir_down = False

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

    if dir_down:
        row += 1
    else:
        row -= 1

index = 0
for i in range(key):
    for j in range(len(cipher)):
        if ((rail[i][j] == '*') and
        (index < len(cipher))):
            rail[i][j] = cipher[index]
            index += 1

result = []
row, col = 0, 0
for i in range(len(cipher)):

    if row == 0:
        dir_down = True
    if row == key-1:
        dir_down = False

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

```
        if dir_down:
            row += 1
        else:
            row -= 1
    return("".join(result))

# Driver code
if __name__ == "__main__":
    c = int(input("What do you want to perform?\n1.
Encryption\n2. Decryption\n"))
    if c==1:
        txt = str(input("Enter the text to be encrypted: "))
        shift = int(input("Enter the Key: "))
        print(f"Ciphertext is:\n{encryptRailFence(txt,
shift)}")

    elif c==2:
        # Decryption
        txt = str(input("Enter the text to be decrypted: "))
        shift = int(input("Enter the Key: "))
        print(f"Ciphertext is:\n{decryptRailFence(txt,
shift)}")
```

Output:

```
PS D:\Final BTech Labs\CNS> python -u "d:\Final BTech Labs\CNS\Assignment 2\rail_fence.py"
What do you want to perform?
1. Encryption
2. Decryption
1
Enter the text to be encrypted: occurence
Enter the Key: 3
o * * * r * * * e
* c * u * e * c *
* * c * * * n * *
Ciphertext is:
orecueccn
PS D:\Final BTech Labs\CNS>
```

Activate Win
Go to Settings to