# LAB - 6

ROll NO. : CE146
Name : Shingala Shubham P.
Id NO. : 19CEUOS159

* AIM: Write a program to implement Encryption and Decryption using Hill cipher for 2X2 and 3x3 matrices

* Source code :

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

int mod(int a)
{
    int modN = a % 26;
    if (modN < 0)
        modN += 26;
    return modN;
}

int gcd (int a, int b)
{
    if (b == 0)
        return a;
    return gcd(b, a % b);
}
```

```
string removeSpaces (string str)
{
    str.erase(remove(str.begin(), str.end(),
                ' '), str.end());
    transform(str.begin(), str.end(),
              str.begin(), ::tolower);
    return str;
}

int multiplicativeInverse(int a, int b)
{
    int q, r, t, t1 = 0, t2 = 1, r1 = b, r2 = a;
    while (r2 > 0)
    {
        q = r1 / r2;
        r = r1 - q * r2;
        r1 = r2;
        r2 = r;
        t = t1 - q * t2;
        t1 = t2;
        t2 = t;
    }
    if (r1 == 1)
    {
        if (t1 < 0)
            t1 += b;
        return t1;
    }
    return -1;
}
```

(2)

```
int ** getCofactor(int** mat ; int p, int q, int n)
{
    int** ans = new int* [n-1];
    int i=0, j=0;
    ans[0] = new int [n-1];
    for (int row=0; row<n; row++)
    {
        for(int col=0; col<n; col++)
        {
            if (row!=p && col!=q)
            {
                ans[i][j] = mat[row][col];
                j++;
                if(j == n-1) {
                    j=0;
                    i++;
                    ans[i] = new int[n-1];
                }
            }
        }
    }
    return ans;
}
```

```
int determinantOfMatrix(int** mat, int n)
{
    int det = 0;
    if (n == 1) // base case
        return mat[0][0];
    int sign = 1;
    for (int i=0; i<n; i++)
    {
        det += sign * mat[0][i] *
               determinantOfMatrix(
               getCofactor(mat, 0, i, n), n-1);
        sign = -sign;
    }
    return det;
}

int** transposeOfMatrix(int** mat, int n, int m)
{
    int** T = new int*[n];
    for (int i=0; i<n; i++)
    {
        T[i] = new int[m];
        for (int j=0; j<m; j++)
            T[i][j] = mat[j][i];
    }
    return T;
}
```

③

```
int **  adjointOfMatrix (int ** mat, int n)
{
    int ** adj = new int * [n];
    for (int i=0 ; i<n ; i++)
    {
        adj [i] = new int [n];
        for (int j=0; j<n ; j++)
        {
            int d = determinantOfMatrix (
             getCofactor (mat, i, j, n) , n-1 );
            adj [i] [j] = pow(-1, i+j+2) * d;
        }
    }
    adj = transposeOfMatrix (adj, n, n) ;
    return adj ;
}

Int ** inverseMatrix (int ** mat, int n)
{
    int ** inverse = adjointOfMatrix(mat, n) ;
    int det = mod (determinantOfMatrix (mat, n));
    det = multiplicativeInverse (det, 26);
    for (int i=0 ; i<n ; i++)
        for (int j=0; j<n ; j++)
            inverse [i] [j] = mod (inverse [i] [j] * det);
    return inverse ;
}
```

```
string multiply_Text_X_key(string text, int** key, int n)
{
    string resultText;
    int i=0;
    while(i< text.size())
    {
        int t[n];
        for(int j=0; j<n; j++, i++)
            t[j] = text[i] - 'a';
        for(int j=0; j<n; j++)
        {
            int ans = 0;
            for(int k=0; k<n; k++)
                ans += t[k] * key[k][j];
            resultText += mod(ans) + 'a';
        }
    }
    return resultText;
}

string encrypt(string plainText, int** key, int n)
{
    while(plainText.size() % n != 0)
        plainText += "z";
    string cipherText = multiply_Text_X_key
                        (plainText, key, n);
    return cipherText;
}
```

④

```
                                                    int n)
string decrypt ( string cipherText, int** key,
{
    int** key_inverse = inverseMatrix (key, n);
    string plainText = multiply_Text_X_key
                (cipherText, key_inverse_n);
    return plainText;
}

int main()
{
    int n;
    cout << " Enter key matrix size : ";
    cin >> n;
    int** key = new int *[n];
    cout << " Enter key matrix such that
        the gcd (determinantOfkey %26,26)
        should be 1 : \n" ;
    for (int i=0; i<n; i++)
    {
        key[i] = new int [n] ;
        for (int j=0; j<n; j++)
            cin >> key [i] [j];
    }
    int detOfKey = mod (determinantOfMatrix
                (key, n)) ;
    if (gcd (detOfkey, 26) != 1) {
        cout << " key matrix determinant
            gcd (determinant, 26) should be 1";
        return 1;
    }
```

```
string plainText;
cout << " Enter plainText : ";
cin >> ws ; // remove buffer.
getline (cin, plainText);
plainText = removeSpaces( plainText);
string encryptText = encrypt (
        plainText, key, n);
string decryptText = decrypt (
        encryptText, key, n);
transform (encryptText.begin(),
                encryptText.end(),
                encryptText.begin();
                :: toupper );
cout << " Encrypt Text :" << encryptText;
cout << " Decrypt Text :" << decryptText;
return 0;
}
```

→ Test case - 1 :

Input: key size n = 3

key matrix key = $\begin{bmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{bmatrix}$

Plain Text = Hello This is Hill cipher Encryption And Decryption.

⑤

Output :

Encrypt Text : RFQZJILNZICFZSMYAVDRAH
VTRXQZBJNNASJYXYRVPCPCM

Decrypt Text : hellothisishillcipherencryptionand
decryptionz

→ Test case - 2 :

Input-1 : key size = 5

key matrix = $\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 10 & 20 & 30 & 40 & 50 \\ 31 & 32 & 33 & 34 & 35 \\ 6 & 7 & 8 & 9 & 10 \\ 90 & 91 & 92 & 93 & 94 \end{bmatrix}$

Input-2 : key size = 2

key matrix = $\begin{bmatrix} 31 & 42 \\ 24 & 56 \end{bmatrix}$

Output : For input-1 and input-2 same output :

key matrix determinant gcd (determinant, 26) should be 1
because we can't find MI of det. and 26.

→ Test case - 3 :

key size = 4

key matrix =

| | | | |
|---|---|---|---|
| 9 | 7 | 11 | 13 |
| 4 | 7 | 5 | 6 |
| 2 | 21 | 14 | 9 |
| 3 | 23 | 21 | 8 |

Plain Text : Hello shubham Shingala your Hill cipher code is Ready.

Encrypt Text : EPOQMZIT VUMV LSUK YNMVZLJK URIO XADF URJY JHXTWIKJ ZNQD

Decrypt Text : he llo shubham Shingala your hill cipher code is ready zzz