## Name: **Ketan Kunkalikar**

## Roll: **21CSE1016**

## Lab Subject: **Security Lab**

## Topic: **Transposition ciphers**

## Instructor: **Dr. Meenakshi Panda**

## Date: **09th October 2024**

// Q1. Write the encryption and decryption program for the following

// i. Column Transposition Cipher.

#include <bits/stdc++.h>

using namespace std;

string Encryption(int no\_rows, int len\_key, int len\_msg,

                  string msg, int col\_val[])

{

    cout << "Encryption function called with:\n";

    cout << "no\_rows: " << no\_rows << ", len\_key: " << len\_key << ", len\_msg: " << len\_msg << "\n";

    cout << "msg: " << msg << "\n";

    cout << "col\_val: ";

    for (int i = 0; i < len\_key; i++)

    {

        cout << col\_val[i] << " ";

    }

    cout << "\n";

    int x = 0;

    char enc\_mat[no\_rows + 1][len\_key];

    char fill\_chars[] = {'X', 'Y', 'Z', 'W', 'P', 'Q'};

    int fill\_index = 0;

    for (int i = 0; i < no\_rows + 1; i++)

    {

        for (int j = 0; j < len\_key; j++)

        {

            if (x >= len\_msg)

            {

                enc\_mat[i][j] = fill\_chars[fill\_index % 6];

                fill\_index++;

            }

            else

            {

                enc\_mat[i][j] = msg[x];

            }

            x++;

        }

    }

    cout << "Encryption matrix:\n";

    for (int i = 0; i < no\_rows + 1; i++)

    {

        for (int j = 0; j < len\_key; j++)

        {

            cout << enc\_mat[i][j] << " ";

        }

        cout << "\n";

    }

    int t = 1;

    string cipher = "";

    while (t <= len\_key)

    {

        for (int i = 0; i < len\_key; i++)

        {

            int k = col\_val[i];

            if (k == t)

            {

                for (int j = 0; j < no\_rows + 1; j++)

                {

                    cipher += enc\_mat[j][i];

                }

                t++;

            }

        }

    }

    cout << "Encrypted cipher: " << cipher << "\n";

    return cipher;

}

string Decryption(int no\_rows, int len\_key, string cipher,

                  int col\_val[])

{

    cout << "Decryption function called with:\n";

    cout << "no\_rows: " << no\_rows << ", len\_key: " << len\_key << "\n";

    cout << "cipher: " << cipher << "\n";

    cout << "col\_val: ";

    for (int i = 0; i < len\_key; i++)

    {

        cout << col\_val[i] << " ";

    }

    cout << "\n";

    char dec\_mat[no\_rows + 1][len\_key];

    int x = 0, t = 1;

    while (t <= len\_key)

    {

        for (int i = 0; i < len\_key; i++)

        {

            int k = col\_val[i];

            if (k == t)

            {

                for (int j = 0; j < no\_rows + 1; j++)

                {

                    dec\_mat[j][i] = cipher[x];

                    x++;

                }

                t++;

            }

        }

    }

    cout << "Decryption matrix:\n";

    for (int i = 0; i < no\_rows + 1; i++)

    {

        for (int j = 0; j < len\_key; j++)

        {

            cout << dec\_mat[i][j] << " ";

        }

        cout << "\n";

    }

    string message = "";

    for (int i = 0; i < no\_rows + 1; i++)

    {

        for (int j = 0; j < len\_key; j++)

        {

            message += dec\_mat[i][j];

        }

    }

    cout << "Decrypted message: " << message << "\n";

    return message;

}

int main()

{

    string msg, key;

    cout << "Enter the message in all caps: ";

    getline(cin, msg);

    cout << "Enter the key in all caps: ";

    getline(cin, key);

    msg.erase(remove(msg.begin(), msg.end(), ' '), msg.end());

    key.erase(remove(key.begin(), key.end(), ' '), key.end());

    cout << "Original message: " << msg << "\n";

    cout << "Key: " << key << "\n";

    int len\_key = key.length();

    int len\_msg = msg.length();

    int val = 1, count = 0, ind;

    int col\_val[len\_key];

    memset(col\_val, 0, sizeof(col\_val));

    while (count < len\_key)

    {

        int min = 999;

        for (int i = 0; i < len\_key; i++)

        {

            if ((min > int(key[i])) && (col\_val[i] == 0))

            {

                min = int(key[i]);

                ind = i;

            }

        }

        col\_val[ind] = val;

        count++;

        val++;

    }

    cout << "col\_val array: ";

    for (int i = 0; i < len\_key; i++)

    {

        cout << col\_val[i] << " ";

    }

    cout << "\n";

    int no\_rows = len\_msg / len\_key;

    string cipher\_text = " ";

    cipher\_text = Encryption(no\_rows, len\_key, len\_msg, msg,

                             col\_val);

    string original\_msg = " ";

    original\_msg = Decryption(no\_rows, len\_key, cipher\_text,

                              col\_val);

}

## **OUTPUT:**

## 

// ii. Route Cipher.

#include <bits/stdc++.h>

using namespace std;

enum class Route

{

    SPIRAL\_CLOCKWISE,

    SPIRAL\_COUNTERCLOCKWISE,

    DIAGONAL

};

class RouteCipher

{

private:

    int rows, cols;

    vector<vector<char>> grid;

    void fillGrid(const string &message)

    {

        int index = 0;

        for (int i = 0; i < rows; ++i)

        {

            for (int j = 0; j < cols; ++j)

            {

                if (index < message.length())

                {

                    grid[i][j] = message[index++];

                }

                else

                {

                    grid[i][j] = 'X';

                }

            }

        }

    }

    void printGrid(const string &title)

    {

        cout << "\n" << title << ":\n";

        for (int i = 0; i < rows; ++i)

        {

            for (int j = 0; j < cols; ++j)

            {

                cout << setw(2) << grid[i][j] << " ";

            }

            cout << "\n";

        }

        cout << "\n";

    }

    string readRoute(Route route)

    {

        string result;

        switch (route)

        {

        case Route::SPIRAL\_CLOCKWISE:

            result = readSpiralClockwise();

            break;

        case Route::SPIRAL\_COUNTERCLOCKWISE:

            result = readSpiralCounterclockwise();

            break;

        case Route::DIAGONAL:

            result = readDiagonal();

            break;

        }

        return result;

    }

    void writeRoute(const string &ciphertext, Route route)

    {

        switch (route)

        {

        case Route::SPIRAL\_CLOCKWISE:

            writeSpiralClockwise(ciphertext);

            break;

        case Route::SPIRAL\_COUNTERCLOCKWISE:

            writeSpiralCounterclockwise(ciphertext);

            break;

        case Route::DIAGONAL:

            writeDiagonal(ciphertext);

            break;

        }

    }

    string readSpiralClockwise()

    {

        string result;

        int top = 0, bottom = rows - 1, left = 0, right = cols - 1;

        while (top <= bottom && left <= right)

        {

            for (int i = left; i <= right; i++)

                result += grid[top][i];

            top++;

            for (int i = top; i <= bottom; i++)

                result += grid[i][right];

            right--;

            if (top <= bottom)

            {

                for (int i = right; i >= left; i--)

                    result += grid[bottom][i];

                bottom--;

            }

            if (left <= right)

            {

                for (int i = bottom; i >= top; i--)

                    result += grid[i][left];

                left++;

            }

        }

        return result;

    }

    void writeSpiralClockwise(const string &ciphertext)

    {

        int index = 0;

        int top = 0, bottom = rows - 1, left = 0, right = cols - 1;

        while (top <= bottom && left <= right && index < ciphertext.length())

        {

            for (int i = left; i <= right && index < ciphertext.length(); i++)

                grid[top][i] = ciphertext[index++];

            top++;

            for (int i = top; i <= bottom && index < ciphertext.length(); i++)

                grid[i][right] = ciphertext[index++];

            right--;

            if (top <= bottom)

            {

                for (int i = right; i >= left && index < ciphertext.length(); i--)

                    grid[bottom][i] = ciphertext[index++];

                bottom--;

            }

            if (left <= right)

            {

                for (int i = bottom; i >= top && index < ciphertext.length(); i--)

                    grid[i][left] = ciphertext[index++];

                left++;

            }

        }

    }

    string readSpiralCounterclockwise()

    {

        string result;

        int top = 0, bottom = rows - 1, left = 0, right = cols - 1;

        while (top <= bottom && left <= right)

        {

            for (int i = top; i <= bottom; i++)

                result += grid[i][left];

            left++;

            for (int i = left; i <= right; i++)

                result += grid[bottom][i];

            bottom--;

            if (left <= right)

            {

                for (int i = bottom; i >= top; i--)

                    result += grid[i][right];

                right--;

            }

            if (top <= bottom)

            {

                for (int i = right; i >= left; i--)

                    result += grid[top][i];

                top++;

            }

        }

        return result;

    }

    void writeSpiralCounterclockwise(const string &ciphertext)

    {

        int index = 0;

        int top = 0, bottom = rows - 1, left = 0, right = cols - 1;

        while (top <= bottom && left <= right && index < ciphertext.length())

        {

            for (int i = top; i <= bottom && index < ciphertext.length(); i++)

                grid[i][left] = ciphertext[index++];

            left++;

            for (int i = left; i <= right && index < ciphertext.length(); i++)

                grid[bottom][i] = ciphertext[index++];

            bottom--;

            if (left <= right)

            {

                for (int i = bottom; i >= top && index < ciphertext.length(); i--)

                    grid[i][right] = ciphertext[index++];

                right--;

            }

            if (top <= bottom)

            {

                for (int i = right; i >= left && index < ciphertext.length(); i--)

                    grid[top][i] = ciphertext[index++];

                top++;

            }

        }

    }

    string readDiagonal()

    {

        string result;

        for (int sum = 0; sum < rows + cols - 1; sum++)

        {

            for (int i = 0; i < rows; i++)

            {

                int j = sum - i;

                if (j >= 0 && j < cols)

                {

                    result += grid[i][j];

                }

            }

        }

        return result;

    }

    void writeDiagonal(const string &ciphertext)

    {

        int index = 0;

        for (int sum = 0; sum < rows + cols - 1 && index < ciphertext.length(); sum++)

        {

            for (int i = 0; i < rows && index < ciphertext.length(); i++)

            {

                int j = sum - i;

                if (j >= 0 && j < cols)

                {

                    grid[i][j] = ciphertext[index++];

                }

            }

        }

    }

public:

    RouteCipher(int r, int c) : rows(r), cols(c), grid(r, vector<char>(c)) {}

    string encrypt(const string &message, Route route)

    {

        fillGrid(message);

        printGrid("Initial Grid");

        string result = readRoute(route);

        cout << "Encryption Route: " << getRouteName(route) << "\n";

        return result;

    }

    string decrypt(const string &ciphertext, Route route)

    {

        writeRoute(ciphertext, route);

        printGrid("Decryption Grid");

        string result;

        for (int i = 0; i < rows; ++i)

        {

            for (int j = 0; j < cols; ++j)

            {

                if (grid[i][j] != 'X')

                {

                    result += grid[i][j];

                }

            }

        }

        cout << "Decryption Route: " << getRouteName(route) << "\n";

        return result;

    }

    string getRouteName(Route route)

    {

        switch (route)

        {

        case Route::SPIRAL\_CLOCKWISE:

            return "Spiral Clockwise";

        case Route::SPIRAL\_COUNTERCLOCKWISE:

            return "Spiral Counterclockwise";

        case Route::DIAGONAL:

            return "Diagonal";

        default:

            return "Unknown";

        }

    }

};

Route getRouteChoice()

{

    int choice;

    cout << "\nChoose a route:\n";

    cout << "1. Spiral Clockwise\n";

    cout << "2. Spiral Counterclockwise\n";

    cout << "3. Diagonal\n";

    cout << "Enter your choice (1-3): ";

    cin >> choice;

    cin.ignore();

    switch (choice)

    {

    case 1:

        return Route::SPIRAL\_CLOCKWISE;

    case 2:

        return Route::SPIRAL\_COUNTERCLOCKWISE;

    case 3:

        return Route::DIAGONAL;

    default:

        return Route::SPIRAL\_CLOCKWISE;

    }

}

int main()

{

    int rows, cols;

    string message;

    cout << "=== Route Cipher Encryption/Decryption ===\n\n";

    cout << "Enter the number of rows: ";

    cin >> rows;

    cout << "Enter the number of columns: ";

    cin >> cols;

    cin.ignore();

    cout << "Enter the message: ";

    getline(cin, message);

    Route route = getRouteChoice();

    RouteCipher cipher(rows, cols);

    cout << "\n--- Encryption Process ---\n";

    string encrypted = cipher.encrypt(message, route);

    cout << "Encrypted message: " << encrypted << "\n";

    cout << "\n--- Decryption Process ---\n";

    string decrypted = cipher.decrypt(encrypted, route);

    cout << "Decrypted message: " << decrypted << "\n";

    cout << "\n=== Process Complete ===\n";

    return 0;

}

## **OUTPUT:**

## 

// iii. Row-column transposition cipher.

#include <iostream>

#include <string>

#include <vector>

#include <cmath>

#include <algorithm>

using namespace std;

string removeSpaces(const string& str) {

    string result = str;

    result.erase(remove(result.begin(), result.end(), ' '), result.end());

    return result;

}

// New function to get filler character

char getFillerChar(int index) {

    const string fillers = "XYZWPQ";

    return fillers[index % fillers.length()];

}

string encrypt(const string& plaintext, int key) {

    int rows = ceil(static\_cast<double>(plaintext.length()) / key);

    vector<vector<char>> grid(rows, vector<char>(key));

    int index = 0;

    int fillerIndex = 0;

    for (int i = 0; i < rows; ++i) {

        for (int j = 0; j < key; ++j) {

            if (index < plaintext.length()) {

                grid[i][j] = plaintext[index++];

            } else {

                grid[i][j] = getFillerChar(fillerIndex++);

            }

        }

    }

    string ciphertext;

    for (int j = 0; j < key; ++j) {

        for (int i = 0; i < rows; ++i) {

            ciphertext += grid[i][j];

        }

    }

    cout << "Encryption grid:\n";

    for (const auto& row : grid) {

        for (char c : row) {

            cout << c << ' ';

        }

        cout << "\n";

    }

    return ciphertext;

}

string decrypt(const string& ciphertext, int key) {

    int rows = ceil(static\_cast<double>(ciphertext.length()) / key);

    vector<vector<char>> grid(rows, vector<char>(key));

    int index = 0;

    for (int j = 0; j < key; ++j) {

        for (int i = 0; i < rows; ++i) {

            grid[i][j] = ciphertext[index++];

        }

    }

    string plaintext;

    for (int i = 0; i < rows; ++i) {

        for (int j = 0; j < key; ++j) {

            plaintext += grid[i][j];

        }

    }

    cout << "Decryption grid:\n";

    for (const auto& row : grid) {

        for (char c : row) {

            cout << c << ' ';

        }

        cout << "\n";

    }

    return plaintext;

}

int main()

{

    string plaintext;

    int key;

    cout << "Enter the plaintext: ";

    getline(cin, plaintext);

    cout << "Enter the key (number of columns): ";

    cin >> key;

    plaintext = removeSpaces(plaintext);

    cout << "Plaintext (without spaces): " << plaintext << "\n";

    cout << "Key: " << key << "\n";

    string ciphertext = encrypt(plaintext, key);

    cout << "Ciphertext: " << ciphertext << "\n";

    string decrypted = decrypt(ciphertext, key);

    cout << "Decrypted: " << decrypted << "\n";

    return 0;

}

## **OUTPUT:**

## 

// iv. Rail-fence cipher.

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

using namespace std;

void printGrid(const vector<string>& rails) {

    for (const auto& rail : rails) {

        cout << rail << "\n";

    }

    cout << "\n";

}

string removeSpaces(const string& str) {

    string result = str;

    result.erase(remove(result.begin(), result.end(), ' '), result.end());

    return result;

}

string encrypt(const string& plaintext, int key) {

    vector<string> rails(key, string(plaintext.length(), '.'));

    int rail = 0;

    bool down = true;

    for (size\_t i = 0; i < plaintext.length(); ++i) {

        rails[rail][i] = plaintext[i];

        if (rail == 0) {

            down = true;

        } else if (rail == key - 1) {

            down = false;

        }

        rail += down ? 1 : -1;

    }

    cout << "Final encryption grid:\n";

    printGrid(rails);

    string ciphertext;

    for (const auto& rail : rails) {

        for (char c : rail) {

            if (c != '.') ciphertext += c;

        }

    }

    return ciphertext;

}

string decrypt(const string& ciphertext, int key) {

    vector<string> rails(key, string(ciphertext.length(), '.'));

    int rail = 0;

    bool down = true;

    for (size\_t i = 0; i < ciphertext.length(); ++i) {

        rails[rail][i] = '\*';

        if (rail == 0) {

            down = true;

        } else if (rail == key - 1) {

            down = false;

        }

        rail += down ? 1 : -1;

    }

    int index = 0;

    for (auto& rail : rails) {

        for (char& c : rail) {

            if (c == '\*') c = ciphertext[index++];

        }

    }

    cout << "Final decryption grid:\n";

    printGrid(rails);

    string plaintext(ciphertext.length(), ' ');

    rail = 0;

    down = true;

    for (size\_t i = 0; i < ciphertext.length(); ++i) {

        plaintext[i] = rails[rail][i];

        if (rail == 0) {

            down = true;

        } else if (rail == key - 1) {

            down = false;

        }

        rail += down ? 1 : -1;

    }

    return plaintext;

}

int main() {

    string plaintext;

    int key;

    cout << "Enter the plaintext: ";

    getline(cin, plaintext);

    plaintext = removeSpaces(plaintext);

    cout << "Enter the key (number of rails): ";

    cin >> key;

    cout << "\nEncrypting...\n";

    string ciphertext = encrypt(plaintext, key);

    cout << "Encrypted text: " << ciphertext << "\n";

    cout << "\nDecrypting...\n";

    string decrypted = decrypt(ciphertext, key);

    cout << "Decrypted text: " << decrypted << "\n";

    return 0;

}

## 

// 2. Write a program to encrypt and decrypt text using affine cipher.

#include <bits/stdc++.h>

using namespace std;

class AffineCipher

{

private:

    int a, b;

    const int m = 26;

    int modInverse(int a, int m)

    {

        for (int x = 1; x < m; x++)

            if (((a % m) \* (x % m)) % m == 1)

                return x;

        return -1;

    }

    int charToInt(char c)

    {

        return c - 'A';

    }

    char intToChar(int i)

    {

        return static\_cast<char>(i + 'A');

    }

public:

    AffineCipher(int a, int b) : a(a), b(b)

    {

        cout << "Affine Cipher initialized with a=" << a << ", b=" << b << endl;

    }

    string encrypt(const string &plaintext)

    {

        cout << "Encrypting: " << plaintext << endl;

        string ciphertext;

        for (char c : plaintext)

        {

            if (isalpha(c))

            {

                int x = charToInt(toupper(c));

                int encrypted = (a \* x + b) % m;

                char encryptedChar = intToChar(encrypted);

                ciphertext += encryptedChar;

                cout << "  " << c << " -> " << encryptedChar << " (x=" << x << ", E(x)=" << encrypted << ")" << endl;

            }

        }

        return ciphertext;

    }

    string decrypt(const string &ciphertext)

    {

        cout << "Decrypting: " << ciphertext << endl;

        string plaintext;

        int a\_inv = modInverse(a, m);

        cout << "  Modular inverse of a: " << a\_inv << endl;

        for (char c : ciphertext)

        {

            if (isalpha(c))

            {

                int y = charToInt(toupper(c));

                int decrypted = (a\_inv \* (y - b + m)) % m;

                char decryptedChar = intToChar(decrypted);

                plaintext += decryptedChar;

                cout << "  " << c << " -> " << decryptedChar << " (y=" << y << ", D(y)=" << decrypted << ")" << endl;

            }

        }

        return plaintext;

    }

};

string removeSpaces(const string &str)

{

    string result = str;

    result.erase(remove\_if(result.begin(), result.end(), ::isspace), result.end());

    cout << "Removed spaces: " << result << endl;

    return result;

}

int main()

{

    int a, b;

    string input;

    cout << "Enter the value of 'a' (must be coprime with 26): ";

    cin >> a;

    cout << "Enter the value of 'b': ";

    cin >> b;

    AffineCipher cipher(a, b);

    cin.ignore();

    cout << "Enter the plaintext: ";

    getline(cin, input);

    cout << "Original input: " << input << endl;

    input = removeSpaces(input);

    transform(input.begin(), input.end(), input.begin(), ::toupper);

    cout << "Processed input: " << input << endl;

    string encrypted = cipher.encrypt(input);

    cout << "Encrypted result: " << encrypted << endl;

    string decrypted = cipher.decrypt(encrypted);

    cout << "Decrypted result: " << decrypted << endl;

    return 0;

}

## **Output:**

## 

// Q3. Write a program to encrypt and decrypt the text using vigenere and vernam cipher.

#include <iostream>

#include <string>

#include <vector>

using namespace std;

string vigenereEncrypt(const string& plaintext, const string& key) {

    string ciphertext = plaintext;

    int keyLength = key.length();

    for (int i = 0; i < plaintext.length(); i++) {

        if (isalpha(plaintext[i])) {

            char base = isupper(plaintext[i]) ? 'A' : 'a';

            ciphertext[i] = (plaintext[i] - base + key[i % keyLength] - 'A') % 26 + base;

        }

    }

    cout << "Vigenere Encryption: " << ciphertext << "\n";

    return ciphertext;

}

string vigenereDecrypt(const string& ciphertext, const string& key) {

    string plaintext = ciphertext;

    int keyLength = key.length();

    for (int i = 0; i < ciphertext.length(); i++) {

        if (isalpha(ciphertext[i])) {

            char base = isupper(ciphertext[i]) ? 'A' : 'a';

            plaintext[i] = (ciphertext[i] - base - (key[i % keyLength] - 'A') + 26) % 26 + base;

        }

    }

    cout << "Vigenere Decryption: " << plaintext << "\n";

    return plaintext;

}

string vernamEncrypt(const string& plaintext, const string& key) {

    string ciphertext = plaintext;

    for (int i = 0; i < plaintext.length(); i++) {

        ciphertext[i] = plaintext[i] ^ key[i % key.length()];

    }

    cout << "Vernam Encryption: ";

    for (char c : ciphertext) {

        cout << hex << (int)c;

    }

    cout << "\n";

    return ciphertext;

}

string vernamDecrypt(const string& ciphertext, const string& key) {

    string plaintext = ciphertext;

    for (int i = 0; i < ciphertext.length(); i++) {

        plaintext[i] = ciphertext[i] ^ key[i % key.length()];

    }

    cout << "Vernam Decryption: " << plaintext << "\n";

    return plaintext;

}

int main() {

    string plaintext, key;

    cout << "Enter plaintext: ";

    getline(cin, plaintext);

    cout << "Enter key: ";

    getline(cin, key);

    cout << "\nVigenere Cipher:\n";

    string vigenereEncrypted = vigenereEncrypt(plaintext, key);

    vigenereDecrypt(vigenereEncrypted, key);

    cout << "\nVernam Cipher:\n";

    string vernamEncrypted = vernamEncrypt(plaintext, key);

    vernamDecrypt(vernamEncrypted, key);

    return 0;

}

## **Output:**

## 