**INS task 1 – nnm22is104**

**Neil Mammen Prakash**

Objective: Perform an in-depth analysis of substitution and transposition techniques.

**Playfair Cipher:**

Setting up key: Constructing the 5x5 matrix from keyword (user input), this should take O(25) time which is constant time so O(n).

Setting up diagraphs: Since n length based on text, we can get a computational complexity of O(n).

Substituting pairs: n/2 total lookups since the diagraphs present are n/2. And then further appending to output should take another n/2 time [O(1)] per diagraph. This would lead to O(n/2 + n/2) which is essentially O(n) time complexity.

Since all the steps are O(n), we get O(3n) while doing all the steps which is essentially **O(n) time complexity.**

The **space complexity for the same would be only O(1)** since only the 5x5 matrix requires any extra space.

**Hill Cipher:**

Converting plaintext: n length would result in O(n \* 1).

Split plaintext: Split into blocks of size k, so this would lead to complexity of O(n/k). Since k might be small, we can take it as O(n).

Matrix multiplication: The matrix of size k^2 because of the blocks of size k. This would result in n/k \* k^2 multiplications which will essentially bring it down to O(n\*k).

Modulo operation: Would give O(n) time complexity since it’s dependant on word length n.

The final time complexity for the same would result in O(n\*m) + O(3n) which would essentially lead to **O(n\*m) time complexity**.

**Vigenere Cipher:**

Extend key: if key is shorter than plaintext, it will be repeated to match length of plaintext. This takes O(n).

Character conversion: For each character in plaintext, 4 steps followed. Each step takes O(1) for each character. So, this leads to O(4\*n) which is made down to O(n).

**The final time complexity will be boiled down to O(n)** since both these steps are in O(n).

**Space complexity remains O(n)** since the key, cipher and plaintexts are dependant on the length of input.

Hybrid Cipher (Rectangular Cipher implementing substitution and transposition technique)

/\*\*

\* A Rectangular Cipher is a cipher text that fits into a rectangle. Example are the Zodiac 408 and 340

\* This class wraps around a 2d array, and supports column transposition.

\*/

import org.apache.commons.math3.util.FastMath;

import org.jblas.util.Permutations;

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.util.\*;

import java.util.Random;

public class RectangularCipher {

public int[][] cipher;

int nrows;

int ncols;

long rng = 2018;

int[][] proposal;

FastTHMM hmmSolver;

TravelingSalesmanSolver tsSolver = new TravelingSalesmanSolver();

double[][] bigram;

double[][][] trigram;

public RectangularCipher(int[][] cipher, double[][] bigram, double[][][] trigram){

nrows = cipher.length;

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

assert(cipher[0].length == cipher[i].length);

}

ncols = cipher[0].length;

this.cipher = cipher.clone();

this.bigram = bigram;

this.trigram =trigram;

}

/\*

permute order of columns of the matrix based on new column indices

\*/

static int[][] transpose(int[][] matrix, int[] columnIndices){

int r = matrix.length;

int c = matrix[0].length;

assert(columnIndices.length == c);

int[][] newArray = new int[r][c];

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

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

newArray[i][j] = matrix[i][columnIndices[j]];

}

}

return newArray;

}

public int[] getCol(int c){

int[] res = new int[nrows];

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

res[i] = cipher[i][c];

}

return res;

}

public void setCol(int c, int[] val){

assert(val.length == nrows);

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

cipher[i][c] = val[i];

}

}

/\*

reshape 2d matrix into flat array

\*/

static int[] flatten(int[][] matrix){

int r = matrix.length;

int c = matrix[0].length;

int[] res = new int[r \* c];

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

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

res[i \* c + j] = matrix[i][j];

}

}

return res;

}

/\*

reshape the flat array into 2d array of same shape as the cipher matrix

\*/

static int[][] reshape(int[] arr, int r, int c){

int[][] res = new int[r][c];

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

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

res[i][j] = arr[i \* c + j];

}

}

return res;

}

/\*

Get distance of 2 columns based on bigrams between them

\*/

public double getCol2ColDistance(int[][] matrix, int i, int j){

double logP = 0;

for (int k = 0; k < nrows; k++) {

logP += FastMath.log(bigram[matrix[k][i]][matrix[k][j]]);

}

return logP;

}

/\*

return matrix of distances between all pairs of cols

\*/

public double[][] getCol2ColDistanceMatrix(int[][] matrix){

double[][] distance = new double[ncols][ncols];

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

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

distance[i][j] = getCol2ColDistance(matrix, i,j);

}

}

return distance;

}

public int[] deSubstitution(int[] cipher, int nHidden, int nObs){

hmmSolver = new FastTHMM(trigram, nHidden, nObs, cipher);

hmmSolver.train(200, false,rng, false);

return hmmSolver.viterbi();

}

public int[] deTransposition(int[][] matrix){

double[][] distance = getCol2ColDistanceMatrix(matrix);

tsSolver.solve(ncols, distance);

int[] tour = tsSolver.getTour();

boolean changed = false;

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

if (tour[i] != i){

changed = true;

break;

}

}

if (!changed) System.out.println("already at optimal value");

return tour;

}

/\*

MAIN PROCEDURE

\*/

public double[] iterativeDecipher(int iter){

LinkedList<Double> logScore = new LinkedList<>();

double currScore = Double.NEGATIVE\_INFINITY;

logScore.add(currScore);

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

System.out.println("Iteration "+i);

int[] seq = flatten(cipher);

System.out.print("Solving substitution...");

seq = deSubstitution(seq, 26, 26);

proposal = reshape(seq, nrows, ncols);

currScore = logProb();

System.out.println("Score = "+currScore);

System.out.println(toString(proposal));

if ( logScore.getLast() >= currScore){

System.out.println("log prob not increasing");

break;

}

logScore.add(currScore);

System.out.print("Solving transposition...");

int[] tour = deTransposition(proposal);

cipher = transpose(cipher, tour);

proposal = transpose(proposal, tour);

currScore = logProb();

System.out.println("Score = "+currScore);

System.out.println(toString(proposal));

if ( logScore.getLast() >= currScore){

System.out.println("log prob not increasing");

}

logScore.add(currScore);

}

double[] arrayScores = logScore.stream().mapToDouble(i->i).toArray();

return arrayScores;

}

public static String toString(int[][] intMatrix) {

StringBuilder sb = new StringBuilder();

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

for (int j = 0; j < intMatrix[0].length; j++) {

sb.append((char) (intMatrix[i][j] + 65));

sb.append(' ');

}

sb.append('\n');

}

return sb.toString();

}

public static int[][] readFromFile(String cipherDir, int nrows, int ncols) throws FileNotFoundException {

Scanner reader = new Scanner(new BufferedReader(new FileReader(cipherDir)));

int[][] plain = new int[nrows][ncols];

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

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

plain[i][j] = reader.nextInt();

}

}

System.out.println("plain text\n"+toString(plain));

int[][] cipher = new int[nrows][ncols];

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

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

plain[i][j] = reader.nextInt();

}

}

System.out.println("cipher text\n"+toString(cipher));

return cipher;

}

public static int[] permuteKey(String method, long seed) {

int[] dict = new int[26];

if (method.equals("random")) {

Random rng = new Random(seed);

LinkedList<Integer> temp = new LinkedList<>();

for (int i = 0; i < 26; i++) {

temp.push(i);

}

for (int i = 25; i >= 0; i--) {

double r = rng.nextDouble();

dict[i] = temp.remove((int) (r \* (i + 1)));

}

} else if (method.equals("caesar")) {

for (int i = 0; i < 26; i++) {

dict[i] = (i + 3) % 26;

}

}

return dict;

}

/\*

compute log probability of current proposal based on trigram

\*/

public double logProb(){

int[] seq = flatten(proposal);

double lp = 0;

for (int i = 2; i < seq.length; i++) {

lp += FastMath.log(trigram[seq[i-2]][seq[i-1]][seq[i]]);

}

return lp;

}

public double logProbArray(int[] seq){

double lp = 0;

for (int i = 2; i < seq.length; i++) {

lp += FastMath.log(trigram[seq[i-2]][seq[i-1]][seq[i]]);

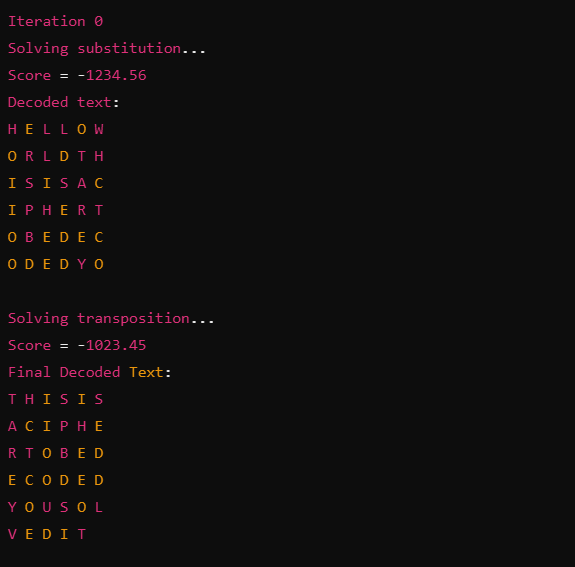
}

return lp;

}

}

Output



Using these techniques alone is relatively weak since it’s only has columns and if the matrix size is known, it is extremely vulnerable to frequency analysis and can be easily cracked. Hence, it’s blended to provide a better approach to security.  
Uses a rectangular structure for cipher, the text is reshaped into a 2D matrix. Column transposition applied, It’s then substituted using Caeser cipher. It’s security can be improved by adding complex key management systems to increase the randomness of key generation and combining with other algorithms for future.

It can be coupled with AES to provide the 128-bit standard encryption.

Mathematical explanation for breaking each cipher using **cryptanalysis techniques:**

**Playfair Cipher:**

Can be broken by using frequency analysis and known plaintext code.

Diagraph frequency should be used, and depends on the size of grid (likely 25) and the possible keys which is 25! (25 factorial). However, with known plaintext the search space will reduce.

**Hill Cipher:**

The Hill cipher is vulnerable if the attacker knows k pairs of plaintext and ciphertext, where k is the size of the key matrix. These pairs can be used to set up a system of linear equations:

C=P\*Kmod 26

where C = ciphertext matrix, P = plaintext matrix, and K = key matrix.

P is invertible, key can be computed as:

K=P^−1\*Cmod 26

The complexity of breaking the cipher is O(k^3) due to matrix inversion.

**Vignere Cipher:**

Can be broken by using kasiki examination and frequency analysis.

The key repeats periodically, so the key length can be determined. Once k is determined(by kasiki examination) ciphertext is divided into k separate caeser ciphers. Each can then be broken by using frequency analysis.

The complexity of breaking the cipher depends on the key length k and the length of the ciphertext m. The overall complexity is approximately O(k\*m).

Github repo: <https://github.com/neilplus21/INSciphers>