Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

|  |
| --- |
| Институт космических и информационных технологий |
| *институт* |
| Кафедра Прикладной математики и компьютерной безопасности |
| *кафедра* |

**ЛАБОРАТОРНАЯ РАБОТА № 14**

|  |
| --- |
| **Частотный криптоанализ** |
| *тема* |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Руководитель | | |  |  |  | В.И. Вайнштейн |
|  | |  |  | *подпись, дата* |  | *инициалы, фамилия* |
| Студент | КИ16-02/2Б №031632156 | |  |  |  | Н.С.Тимошенко |
|  | *номер группы, зачетной книжки* | |  | *подпись, дата* |  | *инициалы, фамилия* |

Красноярск 2019

**Задание:**

Необходимо реализовать частотный криптоанализ с возможностью работы на двух языках, и который будет предлагать наиболее вероятный ключ и расшифровывать сообщение.

**Теория:**

Частотный анализ, частотный криптоанализ — один из методов криптоанализа, основывающийся на предположении о существовании нетривиального статистического распределения отдельных символов и их последовательностей как в открытом тексте, так и в шифротексте, которое, с точностью до замены символов, будет сохраняться в процессе шифрования и дешифрования. Утверждается, что вероятность появления отдельных букв, а также их порядок в словах и фразах естественного языка подчиняются статистическим закономерностям: например, пара стоящих рядом букв «ся» в русском языке более вероятна, чем «цы», а «оь» в русском языке не встречается вовсе (зато часто встречается, например, в чеченском). Анализируя достаточно длинный текст, зашифрованный методом замены, можно по частотам появления символов произвести обратную замену и восстановить исходный текст. Как упоминалось выше, важными характеристиками текста являются повторяемость букв (количество различных букв в каждом языке ограничено), пар букв, то есть m (m-грамм), сочетаемость букв друг с другом, чередование гласных и согласных и некоторые другие особенности. Примечательно, что эти характеристики являются достаточно устойчивыми.

Идея программы состоит в подсчёте чисел вхождений каждой символа возможных достаточно длинных открытых текстах.

Если L — число появлений этого символа в тексте T, а N — общее число подсчитанных символов в тексте.

Частоту букв в процентном выражении можно определить следующим образом: подсчитывается, сколько раз она встречается в шифротексте, затем полученное число делится на общее число символов шифротекста;Probability\_symbol=Li/N.

Частотность существенно зависит, однако, не только от длины текста, но и от его характера. Например, в техническом тексте обычно редкая буква Ф может появляться гораздо чаще. Поэтому для надёжного определения средней частоты букв желательно иметь набор различных текстов.

**Исходный код:**

Алфавит:

double[] StaticFruquencyENG = new double[26]

{ 0.081, 0.016, 0.032, 0.036,

0.123, 0.023, 0.016, 0.051,

0.071, 0.001, 0.005, 0.040,

0.022, 0.072, 0.079, 0.023,

0.002, 0.060, 0.066, 0.096,

0.031, 0.009, 0.020, 0.002,

0.019,0.001};

double[] StaticFruquencyRUS = new double[33]

{ 0.0801, 0.0159, 0.0454, 0.0170,//абвг

0.0298, 0.0845, 0.0004, 0.0094,//деёж

0.0165, 0.0735, 0.0121, 0.0349,//зийк

0.0440,//л

0.0321, 0.0670, 0.1097, 0.0281,//мноп

0.0473, 0.0547, 0.0626, 0.0262, //рсту

0.00260, 0.0097, 0.0048, 0.0144, 0.0073,//фхцч

0.0036, 0.0004, 0.0190, 0.0174,//шщъы

0.0032, 0.0640, 0.0201};//

string[] AlphabetENG = { "ABCDEFGHIJKLMNOPQRSTUVWXYZ", "abcdefghijklmnopqrstuvwxyz" };

string[] AlphabetRUS = { "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ", "абвгдеёжзийклмнопрстуфхцчшщъыьэюя" };

Функция нахождения наиболее вероятного соотношения заданного алфавита и табличного:

private string ProccesAsociationAlphabet(string encript\_text,string[] Alphabet,double[] StaticFruquency)

{

int register=0;

SortedDictionary<char, double> AlphabetMap = new SortedDictionary<char, double>();

double[] Frequency = new double[Alphabet[register].Length];

int lenght\_clear\_text = 0;

foreach (var Symbols in Alphabet)

{

for (int i = 0; i < Symbols.Length; i++)

{

AlphabetMap.Add(Symbols[i], StaticFruquency[i]);

}

}

char[] DeCriptAlphabet = new char[Alphabet[register].Length];

for (int i = 0; i < Alphabet[register].Length; i++)

{

Frequency[i] = 0;

}

foreach (char a in encript\_text)//считаем кол-во символов без знаков пунктуации и пробела плюс частота

{

foreach (var Symbols in Alphabet)

{

for (int i = 0; i < Symbols.Length; i++)

{

if (a == Symbols[i])

{

lenght\_clear\_text++;

Frequency[i]++;

}

}

}

}

for (int i = 0; i < Alphabet[register].Length; i++)

{

Frequency[i] = Frequency[i] / lenght\_clear\_text;

}

for (int k = 0; k < Alphabet[register].Length; k++)

{

double count = 1;

int index = 0;

for (int i = 0; i < Alphabet[register].Length; i++)

{

double difference;

if (Frequency[k] > StaticFruquency[i])

{

difference = Frequency[k] - StaticFruquency[i];

}

else

{

difference = StaticFruquency[i] - Frequency[k];

}

if (count > difference)

{

count = difference;

index = i;

}

}

DeCriptAlphabet[k] = Alphabet[register][index];

}

string DeCriptAlphabetStr = new string(DeCriptAlphabet);

string outstr = null;

for (int i = 0; i < Alphabet[register].Length; i++)

{

outstr += "[ ";

outstr += Alphabet[register][i];

outstr = outstr + "][" + Frequency[i] + "]";

outstr += "[ ";

outstr += DeCriptAlphabet[i];

outstr = outstr + "][" + AlphabetMap[DeCriptAlphabet[i]] + "]";

outstr += "\r\n";

}

ShowSwapBox.Text = outstr;

string open\_text = null;

foreach (var a in encript\_text)

{

bool flag = false;

foreach (var Symbols in Alphabet)

{

for (int i = 0; i < Symbols.Length; i++)

{

if (a == Symbols[i])

{

flag = true;

open\_text = open\_text + DeCriptAlphabet[i];

}

}

}

if (!flag)

{

open\_text = open\_text + a;

}

}

DecryptTextBox.Text = open\_text;

GistogramChart.Series[0].Points.Clear();

GistogramChart.Series[1].Points.Clear();

for (int i = 0; i < Alphabet[register].Length; i++)

{

double y = 0.0;

int index=0;

y = Frequency[i];

for (int k = 0; k < Alphabet[register].Length; k++)

{

if (Alphabet[register][k] == DeCriptAlphabet[i])

{

index = k;

}

}

string x = Convert.ToString(Alphabet[register][i]);

GistogramChart.Series[0].Points.AddXY(x, y);

GistogramChart.Series[1].Points.AddXY(x, StaticFruquency[index]);

}

return DeCriptAlphabetStr;

}

Функция нахождения наиболее вероятного ключа:

private void ProcessShifts(string[] Alphabet, string DeCriptAlphabet)

{

int register = 0;

int[,] probability\_of\_shifts = new int[Alphabet[register].Length, 2];

int[] Shifts = new int[Alphabet[register].Length];

for (int i = 0; i < Alphabet[register].Length; i++)

{

int shift;

int index = 0;

foreach (var Symbols in Alphabet)

{

for (int k = 0; k < Symbols.Length; k++)

{

if (DeCriptAlphabet[i] == Symbols[k])

{

index = k;

}

}

}

shift = index - i;

if ((shift) < 0)

{

shift = (shift + Alphabet[register].Length) % Alphabet[register].Length;

}

Shifts[i] = shift;

}

int count\_number = 0;

for (int i = 0; i < Alphabet[register].Length; i++)

{

int index = 0;

bool flag = false;

for (int k = 0; k < count\_number; k++)

{

if (probability\_of\_shifts[k, 0] == Shifts[i])

{

flag = true;

index = k;

}

}

if (flag == false)

{

probability\_of\_shifts[count\_number, 0] = Shifts[i];

probability\_of\_shifts[count\_number, 1]++;

count\_number++;

}

else

{

probability\_of\_shifts[index, 1]++;

}

}

string outstr = null;

for (int i = 0; i < count\_number; i++)

{

outstr += "[ ";

outstr += probability\_of\_shifts[i, 0] + "] - [" + probability\_of\_shifts[i, 1] + "]";

outstr += "\r\n";

}

ProbabilityShiftsBox.Text = outstr;

}

Функция определения алфавита:

private string ChooseAlphabet()

{

int count\_eng\_symbols=0;

int count\_rus\_symbols=0;

string encript\_text = EncryptTextBox.Text;

foreach (var a in encript\_text)

{

for (int k = 0; k < AlphabetENG.Length; k++)

{

if (a == AlphabetENG[0][k])

{

count\_eng\_symbols++;

}

if (a == AlphabetENG[1][k])

{

count\_eng\_symbols++;

}

}

for (int k = 0; k < AlphabetRUS.Length; k++)

{

if (a == AlphabetRUS[0][k])

{

count\_rus\_symbols++;

}

if (a == AlphabetRUS[1][k])

{

count\_rus\_symbols++;

}

}

}

if (count\_eng\_symbols > count\_rus\_symbols)

{

return "ENG";

}

else

{

return "RUS";

}

}

**Результат работы программы:**

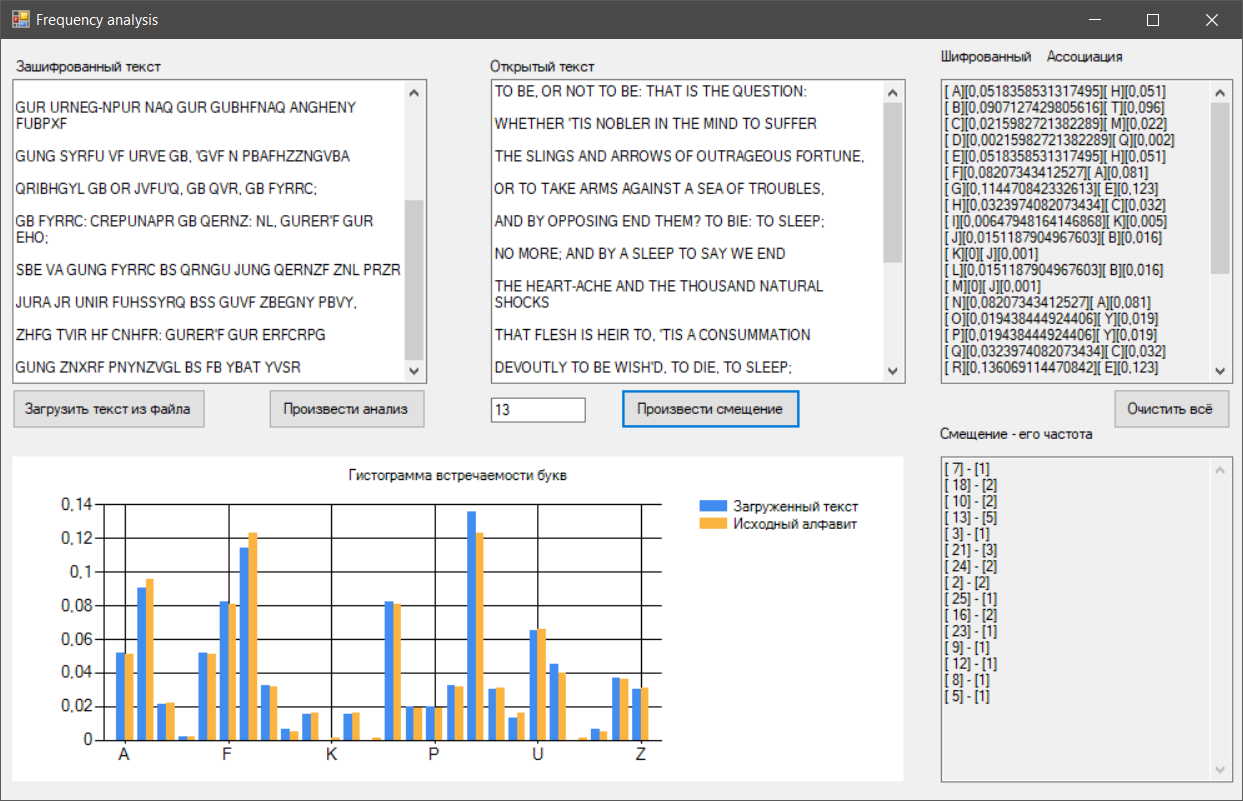


Рисунок 1 – Частотный анализ

**Вывод:**

В ходе лабораторной работы я познакомился с частотным анализом, получил практические навыки в реализации программы по дешифрованию алгоритма Цезаря.