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 | КИ15-01 №031508683 | |  |  |  | М.С.Димаксян |
|  | *номер группы, зачетной книжки* | |  | *подпись, дата* |  | *инициалы, фамилия* |

Красноярск 2019

**Задание:**

Реализовать метод частотного анализа шифртекста на любом языке программирования.

**Теория:**

Частотный анализ, частотный криптоанализ — один из методов криптоанализа, основывающийся на предположении о существовании 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.

Упрощённо, частотный анализ предполагает, что частота появления заданной буквы алфавита в достаточно длинных текстах одна и та же для разных текстов одного языка. При этом в случае моноалфавитного шифрования если в шифротексте будет символ с аналогичной вероятностью появления, то можно предположить, что он и является указанной зашифрованной буквой.

Приблизительные частоты распределения букв уже давно составлены практически для всех языков мира (см. таблицы распределения букв).

Алгоритм для подсчета частоты появления символов алфавита в блоке исходного текста:

На первом шаге вводится символы исходного текста из файла в массив символов. Файл, содержащий исходный текст, должен иметь тек­стовый формат. Одновременно определяем принадлежность вводимого символа к множеству букв используемого алфавита или цифр. Если введенный символ не яв­ляется таковым, то исключаем его из рассмотрения.

В дальнейшем для всех букв/цифр просматриваем массив и подсчитываем число появлений каждой из них, а также определяем относительные частоты появления букв по формуле: (частота появления i-го символа/общее количество символов текста)\*100. После чего для криптоанализа и сравним их со среднестатистическими.

Достоверность получаемых в ходе эксперимента относительных час­тот повышается с увеличением размерности анализируемого блока текста.

**Исходный код:**

**Базовая программа - Информация об алфавитах:**

public string alf\_en = "abcdefghijklmnopqrstuvwxyz";

public string alf\_EN = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

public string alf\_ru = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя";

public string alf\_RU = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ";

//Тест 1 русский

public double[] freqs\_ru = { 0.062,0.014,0.038,0.013,0.025,0.072,0.0001,0.007,0.016,0.062,0.01,0.028,0.035,0.026,0.053,

0.09,0.023,0.04,0.045,0.053,0.021,0.002,0.009,0.004,0.012,0.006,0.003,0.0004,0.016,0.014,0.003,0.006,0.018};

//Тест 1 английский

public double[] freqs\_en = {0.081,0.016,0.032,0.036,0.123,0.023,0.016,0.051,0.071,0.001,0.005,0.04,0.022,0.072,

0.079,0.023,0.002,0.06,0.066,0.096,0.031,0.009,0.02,0.002,0.019,0.001};

//Тест 4 русский

char[,] forbidden\_pairs\_ru = {{'а','ъ'},{'а','ы'},{'а','ь'},{'б','й'},{'в','й'},{'в','э'},{'г','й'},{'г','ф'},{'г','х'},{'г','ъ'},{'г','ь'},{'г','э'},{'д','й'},

{'е','ъ'},{'е','ы'},{'е','ь'},{'е','э'},{'ё','а'},{'ё','е'},{'ё','ё'},{'ё','и'},{'ё','о'},{'ё','у'},{'ё','ф'},{'ё','ъ'},{'ё','ъ'},{'ё','ы'},{'ё','ь'},

{'ё','э'},{'ё','я'},{'ж','ё'},{'ж','ф'},{'ж','х'},{'ж','ш'},{'ж','щ'},{'ж','ы'},{'ж','э'},{'з','й'},{'з','щ'},{'и','ъ'},{'и','ы'},{'и','ь'},{'й','а'},

{'й','ё'},{'й','ж'},{'й','а'},{'й','1'},{'й','ъ'},{'й','ы'},{'й','ь'},{'й','э'},{'к','й'},{'к','щ'},{'к','ъ'},{'к','ь'},{'л','й'},{'л','ъ'},{'л','э'},

{'м','й'},{'м','ъ'},{'н','й'},{'о','ъ'},{'о','ы'},{'о','ь'},{'п','в'},{'п','г'},{'п','ж'},{'п','з'},{'п','й'},{'п','ъ'},{'р','ъ'},{'с','й'},{'т','й'},

{'у','ъ'},{'у','ы'},{'у','ь'},{'ф','б'},{'ф','ж'},{'ф','з'},{'ф','й'},{'ф','п'},{'ф','х'},{'ф','ц'},{'ф','ъ'},{'ф','э'},{'х','ё'},{'х','ж'},{'х','й'},

{'х','щ'},{'х','ы'},{'х','ь'},{'х','ю'},{'х','я'},{'ц','б'},{'ц','ё'},{'ц','ж'},{'ц','й'},{'ц','ф'},{'ц','х'},{'ц','ч'},{'ц','щ'},{'ц','ъ'},{'ц','ь'},

{'ц','э'},{'ц','ю'},{'ц','я'},{'ч','б'},{'ч','г'},{'ч','з'},{'ч','й'},{'ч','п'},{'ч','ф'},{'ч','щ'},{'ч','ъ'},{'ч','ы'},{'ч','э'},{'ч','ю'},{'ч','я'},

{'ш','д'},{'ш','ж'},{'ш','з'},{'ш','й'},{'ш','ш'},{'ш','щ'},{'ш','ъ'},{'ш','ы'},{'ш','э'},{'щ','б'},{'щ','г'},{'щ','д'},{'щ','ж'},{'щ','з'},{'щ','й'},

{'щ','л'},{'щ','п'},{'щ','т'},{'щ','ф'},{'щ','х'},{'щ','ц'},{'щ','ч'},{'щ','ш'},{'щ','щ'},{'щ','ъ'},{'щ','ы'},{'щ','э'},{'щ','ю'},{'щ','я'},{'ъ','а'},

{'ъ','б'},{'ъ','в'},{'ъ','г'},{'ъ','д'},{'ъ','ж'},{'ъ','з'},{'ъ','и'},{'ъ','ё'},{'ъ','к'},{'ъ','л'},{'ъ','м'},{'ъ','н'},{'ъ','о'},{'ъ','п'},{'ъ','р'},

{'ъ','с'},{'ъ','т'},{'ъ','у'},{'ъ','ф'},{'ъ','х'},{'ъ','ц'},{'ъ','ч'},{'ъ','ш'},{'ъ','щ'},{'ъ','ъ'},{'ъ','ы'},{'ъ','ь'},{'ъ','э'},{'ы','а'},{'ы','ё'},

{'ы','о'},{'ы','ф'},{'ы','ъ'},{'ы','ы'},{'ы','ь'},{'ы','э'},{'ь','а'},{'ь','й'},{'ь','л'},{'ь','у'},{'ь','ъ'},{'ь','ы'},{'ь','ь'},{'э','а'},{'э','е'},

{'э','ё'},{'э','ц'},{'э','ч'},{'э','ъ'},{'э','ы'},{'э','ь'},{'э','э'},{'э','ю'},{'ю','у'},{'ю','ъ'},{'ю','ы'},{'ю','ь'},{'я','а'},{'я','ё'},{'я','о'},

{'я','ъ'},{'я','ы'},{'я','ь'},{'я','э'}};

//Тест 4 английский

char[,] forbidden\_pairs\_en = {{'q','a'},{'q','b'},{'q','c'},{'q','d'},{'q','e'},{'q','f'},{'q','g'},{'q','h'},{'q','i'},{'q','j'},{'q','k'},{'q','l'},{'q','m'},

{'q','n'},{'q','o'},{'q','p'},{'q','q'},{'q','r'},{'q','s'},{'q','t'},{'q','v'},{'q','w'},{'q','x'},{'q','y'},{'q','z'},{'j','x'},{'x','z'},{'z','x'},{'j','z'},

{'j','x'},{'v','x'},{'b','x'},{'w','x'},{'k','x'},{'c','x'},{'j','q'},{'v','q'},{'x','q'},{'b','q'},{'z','q'},{'j','y'},{'p','x'},{'z','j'},{'h','x'},{'b','z'},

{'j','v'},{'p','z'},{'x','j'},{'k','z'}};

**Базовая программа - Функция заполнения таблицы и диаграммы:**

private void f14\_Table\_Chart\_func()

{

int[] times\_ru = new int[alf\_ru.Length]; //Массив подсчета вхождений каждой буквы

int[] times\_en = new int[alf\_en.Length];

Array.Clear(times\_ru, 0, alf\_ru.Length);

Array.Clear(times\_en, 0, alf\_en.Length);

string analyzedText = f14\_fieldOriginal.Text.ToLower();

//Подсчитаем количество русских и английских букв, количество их вхождений

int amount\_ru = 0, amount\_en = 0;

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

{

if (alf\_ru.IndexOf(analyzedText[i]) >= 0)

{

amount\_ru++;

times\_ru[alf\_ru.IndexOf(analyzedText[i])]++;

}

if (alf\_en.IndexOf(analyzedText[i]) >= 0)

{

amount\_en++;

times\_en[alf\_en.IndexOf(analyzedText[i])]++;

}

}

f14\_dataGridView.ColumnCount = 4;

f14\_dataGridView.RowCount = 1;

int real\_index = 0;

for (int i = 0; i < times\_ru.Length; i++)

{

if (times\_ru[i] > 0)

{

f14\_dataGridView.RowCount++;

f14\_dataGridView.Rows[real\_index].Cells[0].Value = alf\_ru[i];

f14\_dataGridView.Rows[real\_index].Cells[1].Value = times\_ru[i];

f14\_dataGridView.Rows[real\_index].Cells[2].Value = Math.Round((double)times\_ru[i] / amount\_ru, 6); //округлим до 6 знаков после ,

f14\_dataGridView.Rows[real\_index].Cells[3].Value = freqs\_ru[i];

real\_index++;

}

}

for (int i = 0; i < times\_en.Length; i++)

{

if (times\_en[i] > 0)

{

f14\_dataGridView.RowCount++;

f14\_dataGridView.Rows[real\_index].Cells[0].Value = alf\_en[i];

f14\_dataGridView.Rows[real\_index].Cells[1].Value = times\_en[i];

f14\_dataGridView.Rows[real\_index].Cells[2].Value = Math.Round((double)times\_en[i] / amount\_en, 6); //округлим до 6 знаков после ,

f14\_dataGridView.Rows[real\_index].Cells[3].Value = freqs\_en[i];

real\_index++;

}

}

//Заполнение гистограммы

f14\_chart.Series[0].Points.Clear(); //Очистка старых точек

if (amount\_ru > 0)

{

for (int i = 0; i < times\_ru.Length; i++)

{

f14\_chart.Series[0].Points.Add(times\_ru[i]);

}

}

if (amount\_en > 0)

{

for (int i = 0; i < times\_en.Length; i++)

{

f14\_chart.Series[0].Points.Add(times\_en[i]);

}

}

}

**Базовая программа - Основная функция анализа и поиска исходного текста:**

private void f14\_buttonCountKey\_Click(object sender,EventArgs e)

{

int rot = Convert.ToInt32(f14\_numRotate.Value);

if (rot < 0 || rot > 33)

{

MessageBox.Show("Некорректное значение.");

}

else

{

double[] probability = new double[33];

for (int k = 0; k < 33; k++)

{

if (!(f14\_check1.Checked || f14\_check2.Checked || f14\_check3.Checked || f14\_check4.Checked || f14\_check5.Checked))

{

MessageBox.Show("Не выбрано ни одного параметра", "Ошибка");

goto metka\_noparams;

}

string Source = f14\_fieldCiphertext.Text;

char[] Code = Source.ToCharArray();

int rotRU = k;

int rotEN = (26 - k) % 26;

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

{

int tmp = Convert.ToInt16(Code[i]); // Код текущего символа

if ((tmp >= 65 && tmp <= 90) || (tmp >= 97 && tmp <= 122)) //Английские заглавные и строчные

{

tmp = tmp + rotEN;

if (tmp > 122 || (tmp > 90 && (tmp - rotEN < 91))) tmp = tmp - 26;

Code[i] = Convert.ToChar(tmp);

}

else if ((tmp >= 1040 && tmp <= 1071) || tmp == 1025) //Русские заглавные

{

char tmp2 = Convert.ToChar(tmp);

tmp = (alf\_RU.IndexOf(tmp2) - rotRU + 33) % 33;

tmp2 = alf\_RU[tmp];

Code[i] = tmp2;

}

else if ((tmp >= 1072 && tmp <= 1103) || tmp == 1105) //Русские строчные

{

char tmp2 = Convert.ToChar(tmp);

Code[i] = alf\_ru[(alf\_ru.IndexOf(tmp2) - rotRU + 33) % 33];

}

}

string possibleText = new string(Code);

possibleText = possibleText.ToLower(); //Нет необходимости хранить регистр

//Анализ полученной строки

double mistake = (double)f14\_numMistake.Value; //Погрешность частоты,при которой частота считается приемлемой

int[] times\_ru = new int[alf\_ru.Length]; //Массив подсчета вхождений каждой буквы

int[] times\_en = new int[alf\_en.Length];

Array.Clear(times\_ru, 0, alf\_ru.Length);

Array.Clear(times\_en, 0, alf\_en.Length);

//Подсчитаем количество русских и английских букв, количество их вхождений

int amount\_ru = 0, amount\_en = 0;

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

{

if (alf\_ru.IndexOf(possibleText[i]) >= 0)

{

amount\_ru++;

times\_ru[alf\_ru.IndexOf(possibleText[i])]++;

}

if (alf\_en.IndexOf(possibleText[i]) >= 0)

{

amount\_en++;

times\_en[alf\_en.IndexOf(possibleText[i])]++;

}

}

if (amount\_ru != 0) //Проверки,свойственные русскому

{

if (f14\_check1.Checked)

{

//За каждую букву с частотой,отличающейся от статистической не менее чем на погрешность,добавим частотность в квадрате к вероятности шифра.

for (int i = 0; i < alf\_ru.Length; i++)

{

double frq = (double)times\_ru[i] / amount\_ru;

if ((frq - mistake \* freqs\_ru[i] <= freqs\_ru[i]) && (frq + mistake \* freqs\_ru[i] >= freqs\_ru[i]))

probability[k] += (freqs\_ru[i]) \* (freqs\_ru[i]);

}

}

if (f14\_check2.Checked)

{

//Дополнительные баллы вероятности за популярные биграммы

if (possibleText.IndexOf("ов") >= 0) probability[k] += 0.00081;

if (possibleText.IndexOf("ст") >= 0) probability[k] += 0.00080;

if (possibleText.IndexOf("но") >= 0) probability[k] += 0.00063;

if (possibleText.IndexOf("ко") >= 0) probability[k] += 0.00062;

if (possibleText.IndexOf("ен") >= 0) probability[k] += 0.00061;

if (possibleText.IndexOf("во") >= 0) probability[k] += 0.00057;

if (possibleText.IndexOf("то") >= 0) probability[k] += 0.00056;

if (possibleText.IndexOf("ро") >= 0) probability[k] += 0.00054;

if (possibleText.IndexOf("ра") >= 0) probability[k] += 0.00053;

if (possibleText.IndexOf("на") >= 0) probability[k] += 0.00052;

if (possibleText.IndexOf("го") >= 0) probability[k] += 0.00050;

if (possibleText.IndexOf("ос") >= 0) probability[k] += 0.00048;

if (possibleText.IndexOf("нн") >= 0) probability[k] += 0.00023;

if (possibleText.IndexOf("ии") >= 0) probability[k] += 0.00018;

if (possibleText.IndexOf("сс") >= 0) probability[k] += 0.00011;

}

if (f14\_check3.Checked)

{

//Дополнительные баллы вероятности за популярные триграммы

if (possibleText.IndexOf("сто") >= 0) probability[k] += 0.00081;

if (possibleText.IndexOf("ено") >= 0) probability[k] += 0.00080;

if (possibleText.IndexOf("нов") >= 0) probability[k] += 0.00063;

if (possibleText.IndexOf("тов") >= 0) probability[k] += 0.00062;

if (possibleText.IndexOf("ово") >= 0) probability[k] += 0.00061;

if (possibleText.IndexOf("ова") >= 0) probability[k] += 0.00057;

if (possibleText.IndexOf("ков") >= 0) probability[k] += 0.00057;

}

if (f14\_check4.Checked)

{

//За каждое недопустимое сочетание обнулим вероятность шифра

for (int i = 0; i < forbidden\_pairs\_ru.Length / 2; i++)

{

string s = Convert.ToString(forbidden\_pairs\_ru[i, 0]) + Convert.ToString(forbidden\_pairs\_ru[i, 1]);

if (possibleText.IndexOf(s) >= 0)

probability[k] = -1;

}

}

if (f14\_check5.Checked)

{

//За каждоую недопустимую букву в начале или конце слова обнулим вероятность шифра

var words = (possibleText).Split(new char[] { ' ', ',', '.', ':', ';', '?', '!','&'}, StringSplitOptions.RemoveEmptyEntries);

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

{

if (words[i][0] == 'ъ') probability[k] = -1; //Первая буква - ъ

if (words[i][0] == 'ь') probability[k] = -1; //Первая буква - ь

if (words[i][words[i].Length - 1] == 'ъ') probability[k] = -1; //Последняя буква - ъ

//Проверим одиночные буквы (предлоги, союзы, частицы и местоимения)

string cantBeAlone = "бгдеёжзлмнпртфхцчшщъыьэю";

if (words[i].Length == 1)

{

if (cantBeAlone.IndexOf(words[i][0]) >= 0) probability[k] = -1; //Есть однобуквенное слово, которое не существует

}

}

}

}

if (amount\_en != 0) //Проверки,свойственные английскому

{

if (amount\_ru == 0) // Если русских букв нет, сдвиги от 26 до 32 не учитываем

for (int i = 26; i < 33; i++)

{

probability[i] = -1;

}

if (f14\_check1.Checked)

{

//За каждую букву с частотой,отличающейся от статистической не менее чем на погрешность,добавим частотность в квадрате к вероятности шифра.

for (int i = 0; i < alf\_en.Length; i++)

{

double frq = (double)times\_en[i] / amount\_en;

if ((frq - mistake \* freqs\_en[i] <= freqs\_en[i]) && (frq + mistake \* freqs\_en[i] >= freqs\_en[i]))

probability[k] += (freqs\_en[i]) \* (freqs\_en[i]);

}

}

if (f14\_check2.Checked)

{

//Дополнительные баллы вероятности за популярные биграммы

if (possibleText.IndexOf("th") >= 0) probability[k] += 0.00117;

if (possibleText.IndexOf("he") >= 0) probability[k] += 0.00101;

if (possibleText.IndexOf("in") >= 0) probability[k] += 0.00088;

if (possibleText.IndexOf("er") >= 0) probability[k] += 0.00077;

if (possibleText.IndexOf("an") >= 0) probability[k] += 0.00070;

if (possibleText.IndexOf("re") >= 0) probability[k] += 0.00061;

if (possibleText.IndexOf("es") >= 0) probability[k] += 0.00057;

if (possibleText.IndexOf("on") >= 0) probability[k] += 0.00057;

if (possibleText.IndexOf("st") >= 0) probability[k] += 0.00054;

if (possibleText.IndexOf("nt") >= 0) probability[k] += 0.00051;

if (possibleText.IndexOf("en") >= 0) probability[k] += 0.00049;

if (possibleText.IndexOf("at") >= 0) probability[k] += 0.00048;

if (possibleText.IndexOf("ed") >= 0) probability[k] += 0.00047;

if (possibleText.IndexOf("nd") >= 0) probability[k] += 0.00046;

if (possibleText.IndexOf("to") >= 0) probability[k] += 0.00046;

if (possibleText.IndexOf("or") >= 0) probability[k] += 0.00046;

if (possibleText.IndexOf("ea") >= 0) probability[k] += 0.00043;

if (possibleText.IndexOf("ti") >= 0) probability[k] += 0.00043;

if (possibleText.IndexOf("ar") >= 0) probability[k] += 0.00042;

if (possibleText.IndexOf("te") >= 0) probability[k] += 0.00042;

if (possibleText.IndexOf("ng") >= 0) probability[k] += 0.00039;

}

if (f14\_check3.Checked)

{

//Дополнительные баллы вероятности за популярные триграммы

if (possibleText.IndexOf("the") >= 0) probability[k] += 0.00117;

if (possibleText.IndexOf("and") >= 0) probability[k] += 0.00101;

if (possibleText.IndexOf("ing") >= 0) probability[k] += 0.00088;

if (possibleText.IndexOf("her") >= 0) probability[k] += 0.00077;

if (possibleText.IndexOf("hat") >= 0) probability[k] += 0.00070;

if (possibleText.IndexOf("his") >= 0) probability[k] += 0.00061;

if (possibleText.IndexOf("tha") >= 0) probability[k] += 0.00057;

if (possibleText.IndexOf("ere") >= 0) probability[k] += 0.00057;

if (possibleText.IndexOf("for") >= 0) probability[k] += 0.00054;

if (possibleText.IndexOf("ent") >= 0) probability[k] += 0.00051;

}

if (f14\_check4.Checked)

{

//За каждое недопустимое сочетание обнулим вероятность шифра

for (int i = 0; i < forbidden\_pairs\_en.Length / 2; i++)

{

string s = Convert.ToString(forbidden\_pairs\_en[i, 0]) + Convert.ToString(forbidden\_pairs\_en[i, 1]);

if (possibleText.IndexOf(s) >= 0)//

probability[k] = -1;

}

}

if (f14\_check5.Checked)

{

//За каждоую недопустимую букву в начале или конце слова обнулим вероятность шифра

var words = (possibleText).Split(new char[] { ' ', ',', '.', ':', ';', '?', '!', '&' }, StringSplitOptions.RemoveEmptyEntries);

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

{

if (words[i][words[i].Length - 1] == 'v') probability[k] = -1; //Последняя буква - v

if (words[i][words[i].Length - 1] == 'q') probability[k] = -1; //Последняя буква - q

//Проверим одиночные буквы (предлоги, союзы, частицы и местоимения)

string cantBeAlone = "bcdfhjklmnpqrstuvwxyz";

if (words[i].Length == 1)

{

if (cantBeAlone.IndexOf(words[i][0]) >= 0) probability[k] = -1; //Есть однобуквенное слово, которое не существует

}

}

}

}

}

//Дополнительный массив для сортировки и учета неподходящих ключей

int[] whatRot = new int[33];

for (int i = 0; i < 33; i++)

{

whatRot[i] = i;

if (probability[i] == -1) whatRot[i] = -1;

}

//Сортировка

for (int i = 0; i < 33; i++)

{

for (int j = i + 1; j < 33; j++)

{

if (probability[i] > probability[j])

{

double tmp = probability[i];

probability[i] = probability[j];

probability[j] = tmp;

int tmp1 = whatRot[i];

whatRot[i] = whatRot[j];

whatRot[j] = tmp1;

}

}

}

//Вывод результатов

if (whatRot[32] != -1) f14\_key1.Text = Convert.ToString(whatRot[32]);

else { MessageBox.Show("Нет корректных текстов."); f14\_key1.Text = null; }

if (whatRot[31] != -1) f14\_key2.Text = Convert.ToString(whatRot[31]); else f14\_key2.Text = null; ;

if (whatRot[30] != -1) f14\_key3.Text = Convert.ToString(whatRot[30]); else f14\_key3.Text = null; ;

metka\_noparams:;

}

}

**Базовая программа - Работа с файлами:**

private void f14\_ButtonReadCiphertext\_Click(object sender,EventArgs e)

{

string textFromFile = "";

Stream myStream = null;

OpenFileDialog myDialog = new OpenFileDialog();

myDialog.InitialDirectory = "D:\\Учеба\\Крипта\\Labs\\Labs\\bin\\Debug\\TextFiles";

myDialog.DefaultExt = "txt";

myDialog.FileName = "14\_in.txt";

myDialog.Filter = "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*";

myDialog.FilterIndex = 2;

myDialog.RestoreDirectory = true;

if (myDialog.ShowDialog() == DialogResult.OK)

{

try

{

if ((myStream = myDialog.OpenFile()) != null)

{

textFromFile = new StreamReader(myStream,Encoding.GetEncoding(1251)).ReadToEnd();

myStream.Close();

}

}

catch (Exception ex)

{

MessageBox.Show("Error: Could not read file from disk. Original error: " + ex.Message);

}

}

f14\_fieldCiphertext.Text = textFromFile;

}

private void f14\_ButtonSaveOriginal\_Click(object sender,EventArgs e)

{

StreamWriter myStream = null;

SaveFileDialog myDialog = new SaveFileDialog();

myDialog.InitialDirectory = "D:\\Учеба\\Крипта\\Labs\\Labs\\bin\\Debug\\TextFiles";

myDialog.DefaultExt = "txt";

myDialog.FileName = "14\_out.txt";

myDialog.Filter = "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*";

myDialog.FilterIndex = 2;

myDialog.RestoreDirectory = true;

if (myDialog.ShowDialog() == DialogResult.OK)

{

if ((myStream = new StreamWriter(myDialog.OpenFile(),Encoding.GetEncoding(1251))) != null)

{

myStream.WriteLine(Convert.ToString(f14\_fieldOriginal.Text));

myStream.Close();

}

}

}

**Расширенная программа - Информация об алфавитах:**

public string alf\_en = "abcdefghijklmnopqrstuvwxyz";

public string alf\_EN = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

public string alf\_ru = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя";

public string alf\_RU = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ";

public double[] freqs\_ru = { 0.062,0.014,0.038,0.013,0.025,0.072,0.0001,0.007,0.016,0.062,0.01,0.028,0.035,0.026,0.053,

0.09,0.023,0.04,0.045,0.053,0.021,0.002,0.009,0.004,0.012,0.006,0.003,0.0004,0.016,0.014,0.003,0.006,0.018};

public double[] freqs\_en = { 0.081,0.016,0.032,0.036,0.123,0.023,0.016,0.051,0.071,0.001,0.005,0.04,0.022,0.072,

0.079,0.023,0.002,0.06,0.066,0.096,0.031,0.009,0.02,0.002,0.019,0.001};

public string oftenAlf\_en = "etaonisrhldcupfmwygbvkzxqj";

public string oftenAlf\_ru = "оеиатнсрвлкмдпуяызьбгчйхжюшцэщфъё";

**Расширенная программа - Функция заполнения таблицы и диаграммы:**

private void f14\_Table\_Chart\_func()

{

int[] times\_ru = new int[alf\_ru.Length]; //Массив подсчета вхождений каждой буквы

int[] times\_en = new int[alf\_en.Length];

Array.Clear(times\_ru, 0, alf\_ru.Length);

Array.Clear(times\_en, 0, alf\_en.Length);

string analyzedText = f14\_2\_fieldOriginal.Text.ToLower();

//Подсчитаем количество русских и английских букв, количество их вхождений

int amount\_ru = 0, amount\_en = 0;

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

{

if (alf\_ru.IndexOf(analyzedText[i]) >= 0)

{

amount\_ru++;

times\_ru[alf\_ru.IndexOf(analyzedText[i])]++;

}

if (alf\_en.IndexOf(analyzedText[i]) >= 0)

{

amount\_en++;

times\_en[alf\_en.IndexOf(analyzedText[i])]++;

}

}

f14\_2\_dataGridView.ColumnCount = 4;

f14\_2\_dataGridView.RowCount = 1;

int real\_index = 0;

for (int i = 0; i < times\_ru.Length; i++)

{

if (times\_ru[i] > 0)

{

f14\_2\_dataGridView.RowCount++;

f14\_2\_dataGridView.Rows[real\_index].Cells[0].Value = alf\_ru[i];

f14\_2\_dataGridView.Rows[real\_index].Cells[1].Value = times\_ru[i];

f14\_2\_dataGridView.Rows[real\_index].Cells[2].Value = Math.Round((double)times\_ru[i] / amount\_ru, 6); //округлим до 6 знаков после ,

f14\_2\_dataGridView.Rows[real\_index].Cells[3].Value = freqs\_ru[i];

real\_index++;

}

}

for (int i = 0; i < times\_en.Length; i++)

{

if (times\_en[i] > 0)

{

f14\_2\_dataGridView.RowCount++;

f14\_2\_dataGridView.Rows[real\_index].Cells[0].Value = alf\_en[i];

f14\_2\_dataGridView.Rows[real\_index].Cells[1].Value = times\_en[i];

f14\_2\_dataGridView.Rows[real\_index].Cells[2].Value = Math.Round((double)times\_en[i] / amount\_en, 6); //округлим до 6 знаков после ,

f14\_2\_dataGridView.Rows[real\_index].Cells[3].Value = freqs\_en[i];

real\_index++;

}

}

//Заполнение гистограммы

f14\_2\_chart.Series[0].Points.Clear(); //Очистка старых точек

if (amount\_ru > 0)

{

for (int i = 0; i < times\_ru.Length; i++)

{

f14\_2\_chart.Series[0].Points.Add(times\_ru[i]);

}

}

if (amount\_en > 0)

{

for (int i = 0; i < times\_en.Length; i++)

{

f14\_2\_chart.Series[0].Points.Add(times\_en[i]);

}

}

}

**Расширенная программа - Основная функция анализа и поиска исходного текста:**

private void f14\_buttonCountKey\_Click(object sender, EventArgs e)

{

//Проверим шифртекст

bool isTextCorrect = true;

int languagesCode = 0; // +1 русский +2 английский +3 мешанина

for (int i = 0; i < f14\_2\_fieldCiphertext.TextLength; i++)

{

if (alf\_ru.IndexOf(f14\_2\_fieldCiphertext.Text[i]) >= 0 || alf\_RU.IndexOf(f14\_2\_fieldCiphertext.Text[i]) >= 0) //символ русский

{

if (languagesCode == 0 || languagesCode == 2) languagesCode++; //русский еще не был учтен - учтем

}

else //символ не русский

{

if (alf\_en.IndexOf(f14\_2\_fieldCiphertext.Text[i]) >= 0 || alf\_EN.IndexOf(f14\_2\_fieldCiphertext.Text[i]) >= 0) //символ русский

{

if (languagesCode == 0 || languagesCode == 1) languagesCode += 2; //английский еще не был учтен - учтем

}

}

}

//Используется 0 или 2 алфавита?

if (languagesCode == 0 || languagesCode == 3) isTextCorrect = false;

//Если используется один алфавит, сохраним его и отобразим для него поля замены

string alf = "", ALF = "", oftenAlf = "";

int popularLetterRot = 0;

if (languagesCode == 1)

{

f14\_2\_alf.Text = alf\_ru;

f14\_2\_alf2.Text = alf\_ru;

alf = alf\_ru;

ALF = alf\_RU;

oftenAlf = oftenAlf\_ru;

}

if (languagesCode == 2)

{

f14\_2\_alf.Text = alf\_en;

f14\_2\_alf2.Text = alf\_en;

alf = alf\_en;

ALF = alf\_EN;

oftenAlf = oftenAlf\_en;

}

//Проверим ключ

if (isTextCorrect)

{

//Проанализируем данный шифртекст

string cipherText = f14\_2\_fieldCiphertext.Text.ToLower();

//Массив подсчета вхождений каждой буквы

int[] times = new int[alf.Length];

Array.Clear(times, 0, times.Length);

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

{

if (alf.IndexOf(cipherText[i]) >= 0)

{

times[alf.IndexOf(cipherText[i])]++;

}

}

//ФОРМИРУЕМ КЛЮЧ №1

char[] key1 = alf.ToCharArray();

int [] isMoreOften = new int [key1.Length];

for (int i = 0; i < alf.Length; i++) //для каждого символа проверим, сколько символов встречаются чаще него

{

for (int j = 0; j < alf.Length; j++)

{

if (times[i] < times[j]) isMoreOften[i]++;

}

}

int mark = 0;

int[] isMarked = new int[key1.Length];

Array.Clear(isMarked, 0, key1.Length);

while (times.Max() != -1) //Пометим порядок убывания частоты символов

{

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

{

if (times[i] == times.Max() && times.Max() != -1)

{

times[i] = -1;

isMarked[i] = mark;

mark++;

}

}

}

string result = "";

for (int i = 0; i < key1.Length; i++)

{

result += oftenAlf[isMarked[i]];

}

f14\_2\_key1.Text = Convert.ToString(result);

//ФОРМИРУЕМ КЛЮЧ №2

//Пересчитаем испорченный массив times

Array.Clear(times, 0, times.Length);

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

{

if (alf.IndexOf(cipherText[i]) >= 0)

{

times[alf.IndexOf(cipherText[i])]++;

}

}

char[] key2 = alf.ToCharArray();

int[] isMoreOften2 = new int[key2.Length];

for (int i = 0; i < alf.Length; i++) //для каждого символа проверим, сколько символов встречаются чаще него

{

for (int j = 0; j < alf.Length; j++)

{

if (times[i] < times[j]) isMoreOften2[i]++;

}

}

int mark2 = 0;

int[] isMarked2 = new int[key2.Length];

Array.Clear(isMarked2, 0, key2.Length);

while (times.Max() != -1) //Пометим порядок убывания частоты символов

{

for (int i = alf.Length - 1; i >= 0; i--)

{

if (times[i] == times.Max() && times.Max() != -1)

{

times[i] = -1;

isMarked2[i] = mark2;

mark2++;

}

}

}

result = "";

for (int i = 0; i < key2.Length; i++)

{

result += oftenAlf[isMarked2[i]];

}

f14\_2\_key2.Text = Convert.ToString(result);

//ФОРМИРУЕМ КЛЮЧ №3

//Пересчитаем испорченный массив times

Array.Clear(times, 0, times.Length);

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

{

if (alf.IndexOf(cipherText[i]) >= 0)

{

times[alf.IndexOf(cipherText[i])]++;

}

}

string key3 = alf;

for (int i = key3.Length - 1; i >= 0; i--)

{

if (times[i] == times.Max())

{

// i - номер места, на котором будет стоять буква англ "e" или русс "о"

while (key3[i] != 'e' && key3[i] != 'о')

{

var parts = key3.Split(key3[1]);

key3 = key3[1] + parts[1] + parts[0];

}

}

}

f14\_2\_key3.Text = Convert.ToString(key3);

}

}

**Расширенная программа - Работа с файлами:**

private void f14\_2\_ButtonReadCiphertext\_Click(object sender, EventArgs e)

{

string textFromFile = "";

Stream myStream = null;

OpenFileDialog myDialog = new OpenFileDialog();

myDialog.InitialDirectory = "D:\\Учеба\\Крипта\\Labs\\Labs\\bin\\Debug\\TextFiles";

myDialog.DefaultExt = "txt";

myDialog.FileName = "14\_in\_en.txt";

myDialog.Filter = "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*";

myDialog.FilterIndex = 2;

myDialog.RestoreDirectory = true;

if (myDialog.ShowDialog() == DialogResult.OK)

{

try

{

if ((myStream = myDialog.OpenFile()) != null)

{

textFromFile = new StreamReader(myStream, Encoding.GetEncoding(1251)).ReadToEnd();

myStream.Close();

}

}

catch (Exception ex)

{

MessageBox.Show("Error: Could not read file from disk. Original error: " + ex.Message);

}

}

f14\_2\_fieldCiphertext.Text = textFromFile;

f14\_2\_fieldOriginal.Text = null;

f14\_2\_fieldKey.Text = null;

f14\_2\_key1.Text = null;

f14\_2\_key2.Text = null;

f14\_2\_key3.Text = null;

f14\_2\_fieldKey.Text = "1";

f14\_2\_fieldKey.Text = null;

}

private void f14\_2\_ButtonSaveOriginal\_Click(object sender, EventArgs e)

{

StreamWriter myStream = null;

SaveFileDialog myDialog = new SaveFileDialog();

myDialog.InitialDirectory = "D:\\Учеба\\Крипта\\Labs\\Labs\\bin\\Debug\\TextFiles";

myDialog.DefaultExt = "txt";

myDialog.FileName = "14\_out.txt";

myDialog.Filter = "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*";

myDialog.FilterIndex = 2;

myDialog.RestoreDirectory = true;

if (myDialog.ShowDialog() == DialogResult.OK)

{

if ((myStream = new StreamWriter(myDialog.OpenFile(), Encoding.GetEncoding(1251))) != null)

{

myStream.WriteLine(Convert.ToString(f14\_2\_fieldOriginal.Text));

myStream.Close();

}

}

}

Расширенная **программа - Функция расчета исходного текста при каждом изменении ключа**

private void f14\_2\_fieldKey\_TextChanged(object sender, EventArgs e)

{

//Проверим шифртекст

bool isTextCorrect = true;

int languagesCode = 0; // +1 русский +2 английский +3 мешанина

for (int i = 0; i < f14\_2\_fieldCiphertext.TextLength; i++)

{

if (alf\_ru.IndexOf(f14\_2\_fieldCiphertext.Text[i]) >= 0 || alf\_RU.IndexOf(f14\_2\_fieldCiphertext.Text[i]) >= 0) //символ русский

{

if (languagesCode == 0 || languagesCode == 2) languagesCode++; //русский еще не был учтен - учтем

}

else //символ не русский

{

if (alf\_en.IndexOf(f14\_2\_fieldCiphertext.Text[i]) >= 0 || alf\_EN.IndexOf(f14\_2\_fieldCiphertext.Text[i]) >= 0) //символ русский

{

if (languagesCode == 0 || languagesCode == 1) languagesCode += 2; //английский еще не был учтен - учтем

}

}

}

//Используется 0 или 2 алфавита?

if (languagesCode == 0 || languagesCode == 3) isTextCorrect = false;

//Если используется один алфавит, отобразим для него поля замены

if (languagesCode == 1)

{

f14\_2\_alf.Text = alf\_ru;

f14\_2\_alf2.Text = alf\_ru;

}

if (languagesCode == 2)

{

f14\_2\_alf.Text = alf\_en;

f14\_2\_alf2.Text = alf\_en;

}

//Проверим ключ

f14\_2\_fieldKey.Text.ToLower();

bool isKeyCorrect = true;

for (int i = 0; i < f14\_2\_fieldKey.TextLength; i++)

{

//Тест русский, а ключ содержит английский символ

if (languagesCode == 1 && alf\_en.IndexOf(f14\_2\_fieldKey.Text[i]) >= 0) isKeyCorrect = false;

//Тест английский, а ключ содержит русский символ

if (languagesCode == 2 && alf\_ru.IndexOf(f14\_2\_fieldKey.Text[i]) >= 0) isKeyCorrect = false;

//Символ встречается более одного раза

if (f14\_2\_fieldKey.Text.IndexOf(f14\_2\_fieldKey.Text[i]) != f14\_2\_fieldKey.Text.LastIndexOf(f14\_2\_fieldKey.Text[i])) isKeyCorrect = false;

}

//Длина ключа не соответствует длине алфавита

if (languagesCode == 1 && f14\_2\_fieldKey.TextLength != alf\_ru.Length) isKeyCorrect = false;

if (languagesCode == 2 && f14\_2\_fieldKey.TextLength != alf\_en.Length) isKeyCorrect = false;

if (isTextCorrect && isKeyCorrect)

{

string result = "";

//Выберем алфавит

string alf = "";

string ALF = "";

if (languagesCode == 1)

{

alf = alf\_ru;

ALF = alf\_RU;

}

if (languagesCode == 2)

{

alf = alf\_en;

ALF = alf\_EN;

}

//Осуществляем замену

for (int i = 0; i < f14\_2\_fieldCiphertext.TextLength; i++)

{

if (alf.IndexOf(f14\_2\_fieldCiphertext.Text[i]) >= 0) //строчная буква

{

result += f14\_2\_fieldKey.Text[alf.IndexOf(f14\_2\_fieldCiphertext.Text[i])];

}

else

{

if (ALF.IndexOf(f14\_2\_fieldCiphertext.Text[i]) >= 0) //заглавная буква

{

result += f14\_2\_fieldKey.Text[ALF.IndexOf(f14\_2\_fieldCiphertext.Text[i])].ToString().ToUpper();

}

else //не буква

{

result += f14\_2\_fieldCiphertext.Text[i];

}

}

}

f14\_2\_fieldOriginal.Text = Convert.ToString(result);

}

if (f14\_2\_fieldKey.Text == "") f14\_2\_fieldOriginal.Text = null;

}

**Результат работы программы:**

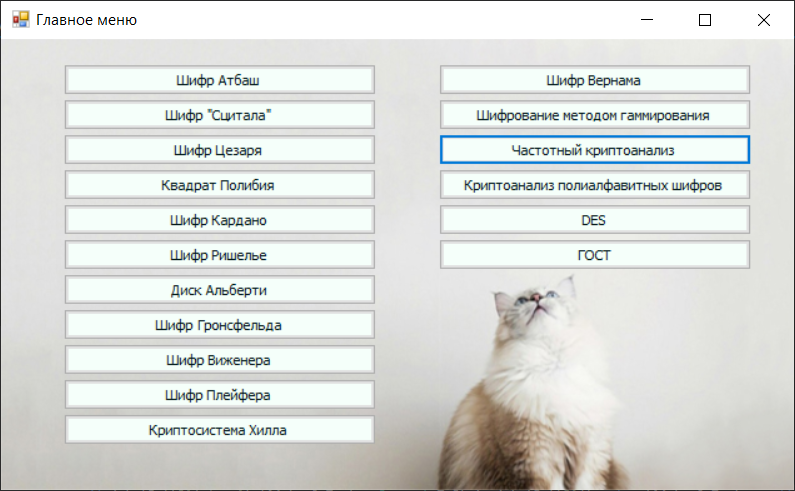


Рисунок 1. Главное меню программы.

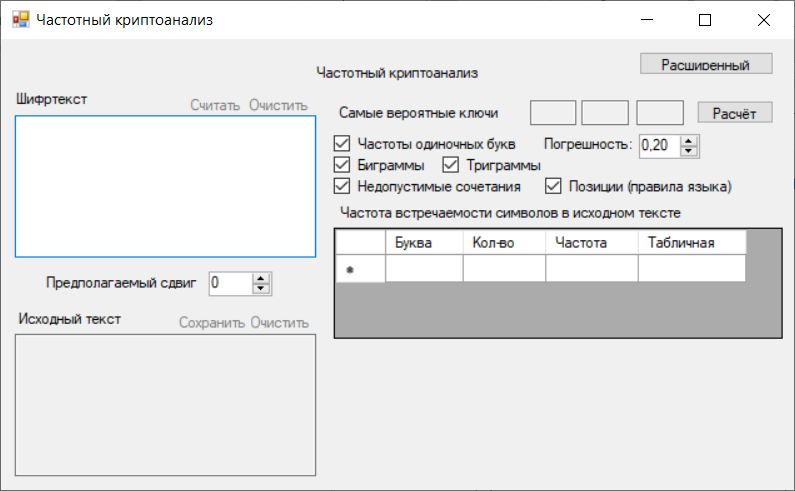


Рисунок 2. Окно частотного криптоанализа.

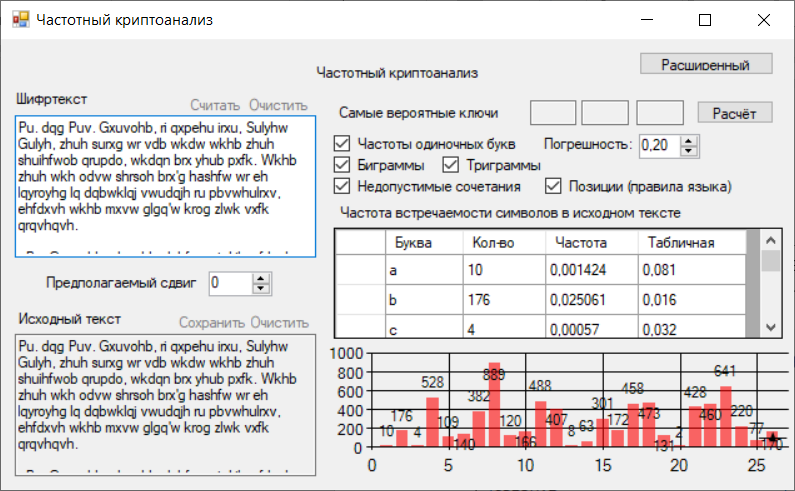


Рисунок 3. Загрузка шифртекста для анализа.

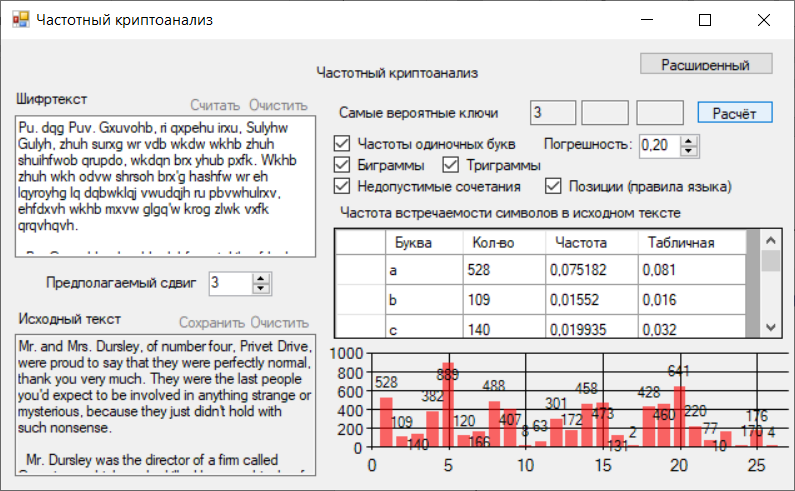


Рисунок 4. Расчет самого вероятного ключа с использованием всех инструментов.

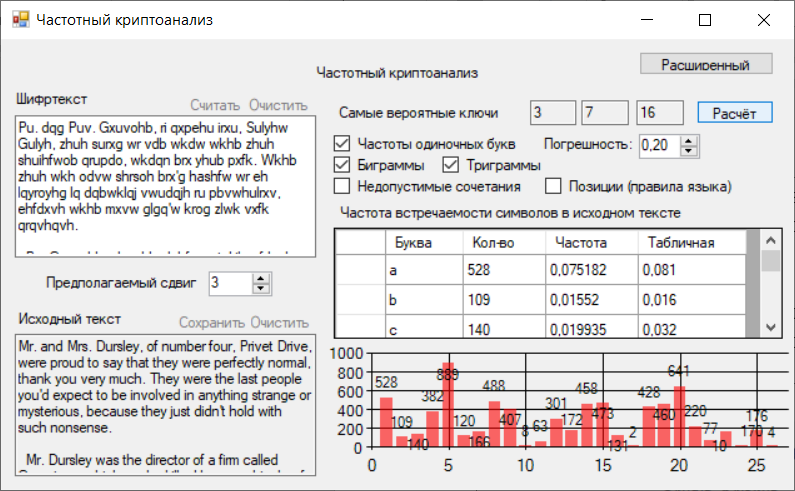


Рисунок 5. Расчет самого вероятного ключа с использованием вероятностных инструментов.

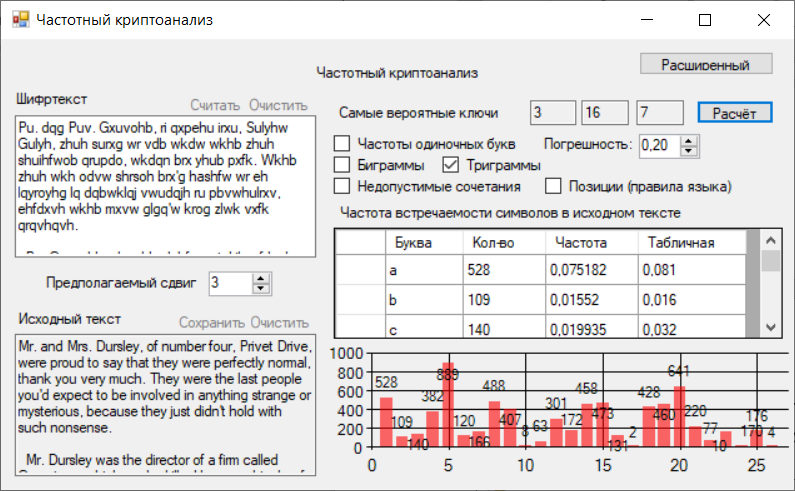


Рисунок 6. Расчет самого вероятного ключа с использованием исключительно триграмм.

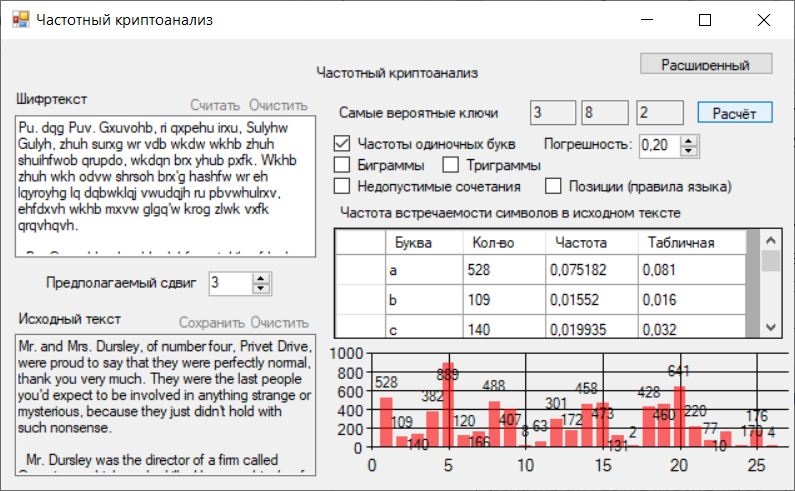


Рисунок 7. Расчет самого вероятного ключа с использованием исключительно частот букв при допустимом отклонении частоты от статистической в 20%.

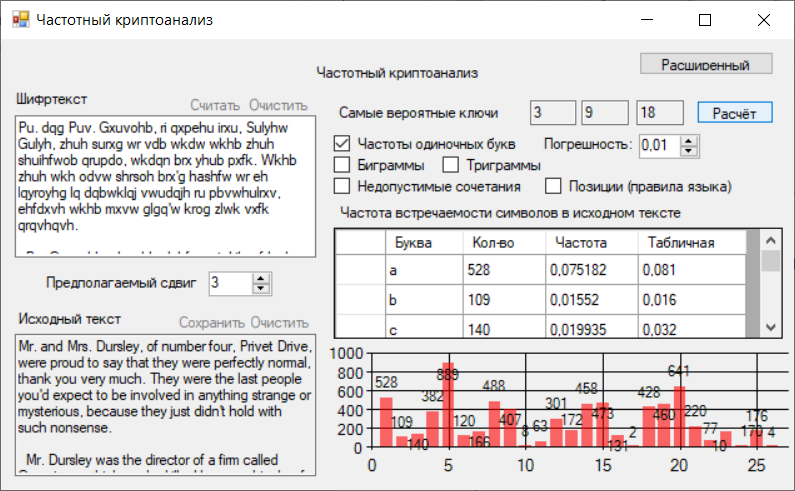


Рисунок 8. Расчет самого вероятного ключа с использованием исключительно частот букв при допустимом отклонении частоты от статистической в 1%.

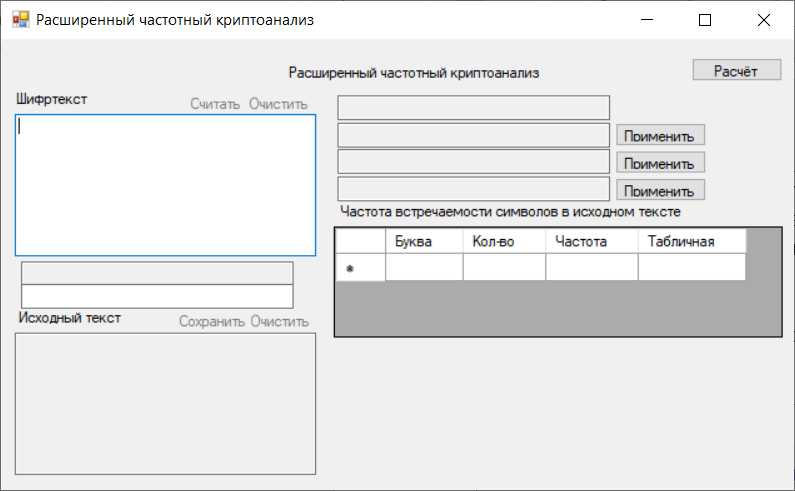


Рисунок 9. Окно расширенного частотного криптоанализа.

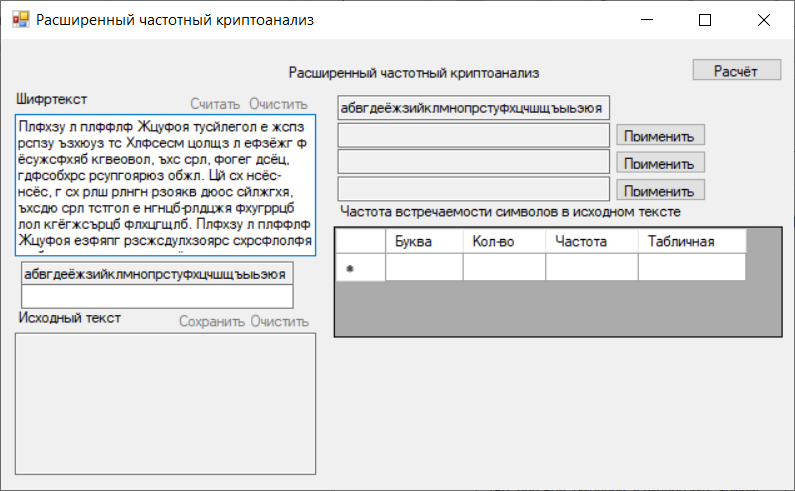


Рисунок 10. Загрузка шифртекста для анализа.

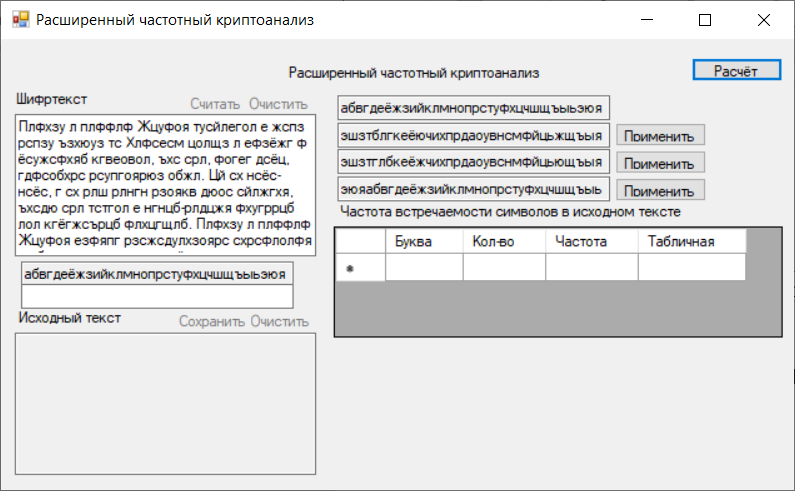


Рисунок 11. Расчет возможных ключей.

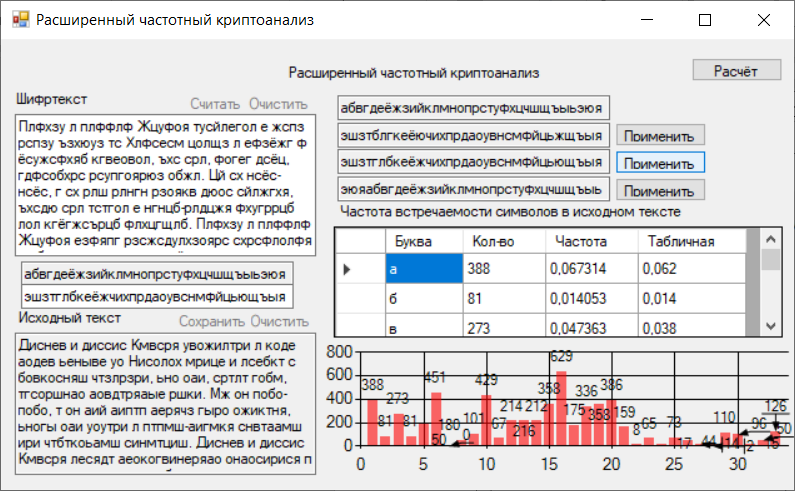


Рисунок 12. Применим второй ключ.

Глядя на тексте, попробуем преобразовать примененный ключ. Из предположения, что третье слово – «миссис», попробуем поменять буквы местами.

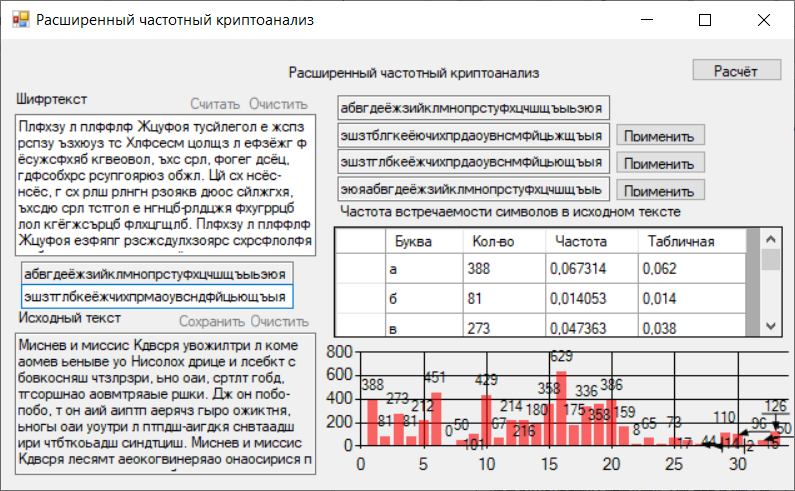


Рисунок 13. Результат преобразования ключа.

Из предположения, что первое слово – «мистер», преобразуем ключ снова.

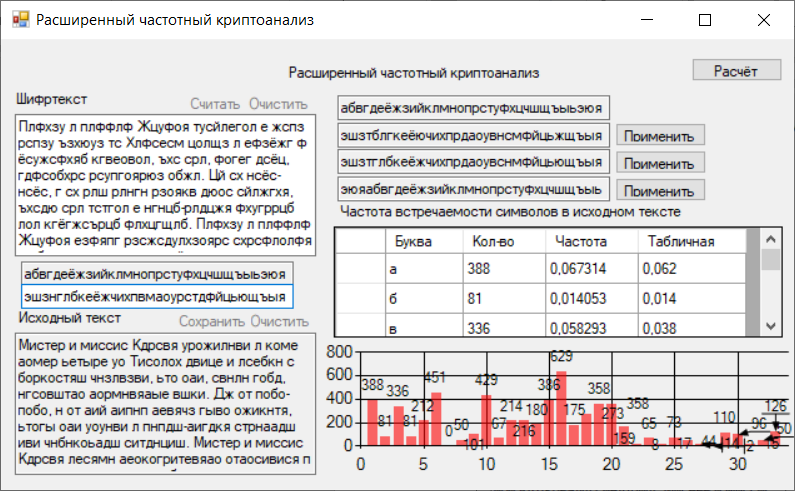


Рисунок 14. Результат преобразования ключа.

Заметим ряд слов: «аомер» - «номер», «ьетыре» - «четыре», «ьто» - «что». Преобразуем ключ и просмотрим другую часть текста на предмет возможных подсказок.

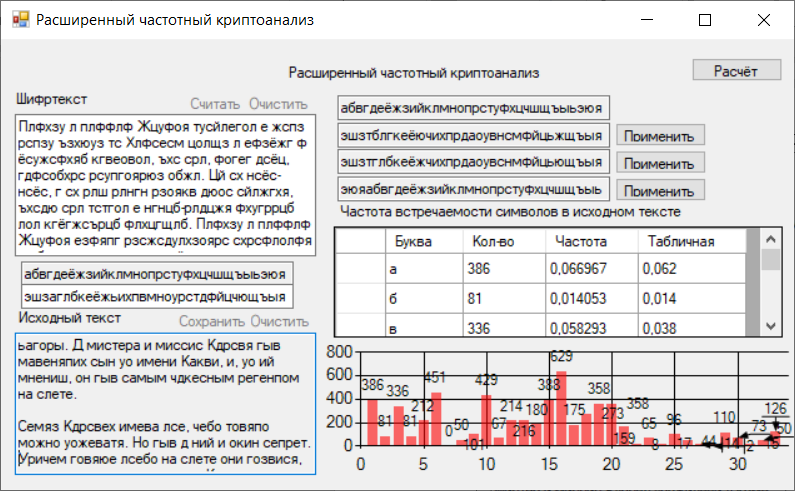


Рисунок 15. Информативный для аналитика отрывок текста.

Преобразуем все явные слова.

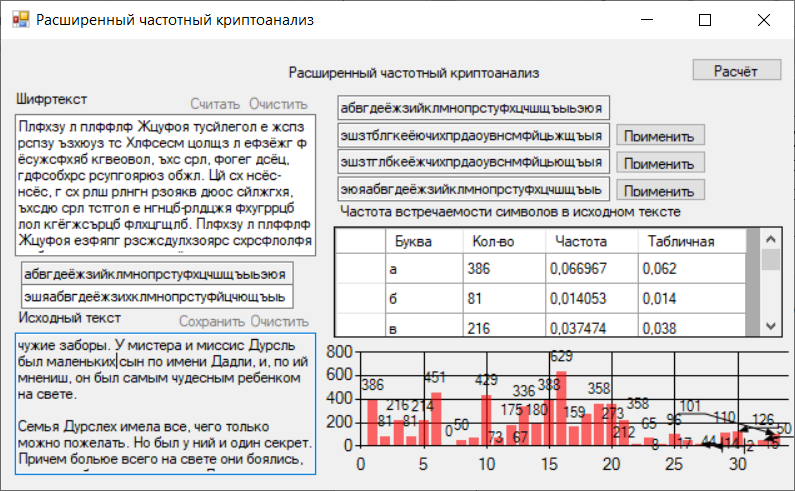


Рисунок 16. Промежуточный результат.

Заменим оставшиеся буквы.

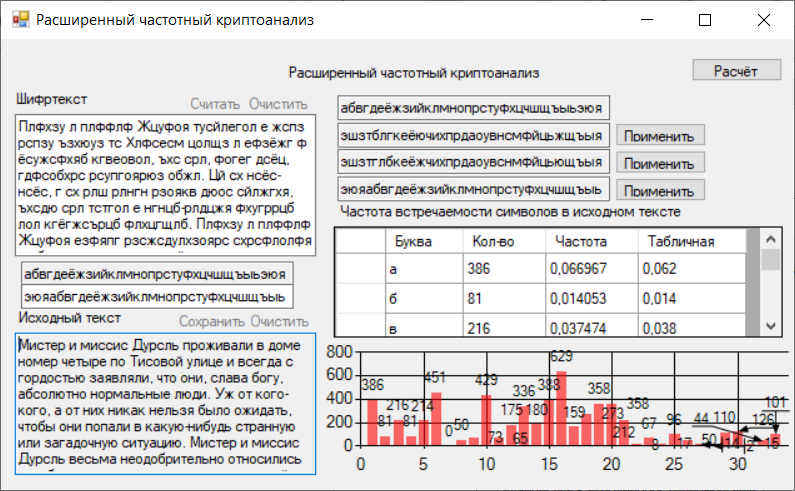


Рисунок 17. Результат криптоанализа.

Получившийся ключ говорит о том, что использовался шифр Цезаря. Кроме того, верный ключ был сразу программно получен при расчете (ключ №3).

**Вывод:**

В ходе выполнения лабораторной работы я ознакомилась с методами частотного криптоанализа, реализовала на практике программу по дешифрованию данными методами на языке C#. В реализованной программе есть возможность комбинировать инструменты и анализировать сложные случаи вручную.