Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3**

Преподаватель \_\_\_\_\_\_\_\_\_\_\_ А.Н. Шниперов

подпись, дата инициалы, фамилия

Студент КИ15–17Б, 031510065 \_\_\_\_\_\_\_\_\_\_\_ Т.В. Радионов

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2017

1. **Цели лабораторной работы**

* ознакомиться с основами ассиметричной криптографии;
* введение в криптосистемы полиалфавитной подстановки на примере шифра Виженера;
* изучение методов криптоанализа шифров полиалфавитной подстановки на примере шифра Виженера.

1. **Задание лабораторной работы**

Используя любой из способов криптоанализа шифров полиалфавитной подстановки расшифруйте текст сообщения, а также определите секретный ключ, которым оно было зашифровано.

Известные сведения о шифрограмме следующие:

* для зашифрования сообщения, которое представляет собой отрывок из

литературного произведения, использовался шифр Виженера, с

алфавитом, представленным в таблице;

* знаки препинания, а также знак пробела не входят в алфавит шифрования и, соответственно, не является частью шифротекста, т.е. шифротекст с расставленными знаками препинания и пробелами эквивалентен шифротексту без них;
* весь шифротекст представляет собой одну строку;
* в качестве ключа шифрования использовалось одно слово (из букв

алфавита), длина которого не более 10 символов.

Для осуществления статистического криптоанализа необходимо разработать программу для дешифрования со следующими возможностями:

* автоматическое приближённое нахождение периода шифрования с помощью теста Касиски;
* автоматическая проверка правильности выбора длины шифрования с помощью нахождения индексов соответствия (для всех подгрупп шифротекста);
* в случаях, когда тест Касиски не удаётся применить, найденное с его помощью, не подтверждается после определения индексов соответствия, период шифрования определяется подбором (такой случай возможен в случаях, когда длина ключа более 7 знаков и/или зашифрованный текст слишком короткий);
* автоматическое вычисление всех потенциально возможных вариантов ключа шифрования, и вывод их пользователю для принятия решения;
* расшифровка шифротекста, по выбранному пользователем ключу

шифрования.

Разработанная программа должна содержать графический интерфейс пользователя.

1. **Описание программы**

Данная программа предназначена для дешифрования шифротекста, зашифрованного шифром Виженера. Она вычисляет период шифрования методом Касиски и методом индексов совпадений для последующего определения комбинаций ключа и, сопоставляя такие комбинации со словарем, определяет потенциально возможные ключи, которые могут расшифровать шифротекст.

Краткое руководство пользования:

* Вставить/ввести шифротекст в соответствующее поле;
* Нажать на кнопку «Определить длину ключа» и ждать, пока выполнится операция и не выведется вычисленная длина ключа;
* Нажать кнопку «Поиск ключей» и ждать, пока не выведутся в соответствующее окно варианты ключей, в противном же случае повысить глубину поиска (необходимо учесть, что чем больше значение глубины поиска, тем дольше будет происходить процесс поиска ключей). Данный этап должен длиться не более, чем 5 минут, иначе операция станет довольно ресурсоемкой и довольно долго вычисляемой;
* Ввести поочередно все варианты ключей в соответствующее окно до тех пор, пока при нажатии на кнопку «Расшифрование» не появится осмысленный текст.

1. **Описание процесса дешифрации**

Процесс дешифрации условно делится на три этапа:

1. Считывание шифротекста и удаление оттуда ненужных знаков препинания, в том числе и пробелов.
2. Поиск повторяющихся фрагментов в шифротексте и нахождение расстояния между такими фрагментами, затем находится наибольший общий делитель этих расстояний для дальнейшего определения периода шифрования (метод Касиски). Далее методом индекса совпадений проверяем найденный период шифрования (длину ключа).
3. Путем сопоставления частоты встречи символа «О» русского алфавита (который является наиболее часто встречаемым в осмысленных текстах) с шифротекстом (а именно с блоками шифрования, количество которых равно длине ключа), определяем вероятные символы шифротекста, которые могут являться зашифрованным символом «О» и определяем для каждого блока смещения в зависимости от глубины поиска, которые, в дальнейшем, необходимы для расшифрования. Затем генерируем различные возможные варианты ключей и, сопоставляя со словарем русских слов, включающий существительные и прилагательные, находим осмысленные слова, которые являются потенциальными ключами;
4. Проверяем каждый потенциальный ключ путем попытки расшифрования им шифротекста.
5. **Код алгоритма**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Text.RegularExpressions;

using System.IO;

namespace WindowsFormsApp1

{

public partial class Form1 : Form

{

static string ciphertext = ""; // Шифротекст

static string[] l; // Шифроблоки (количество зависит от длины ключа)

static char[] alphabet = "АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ".ToCharArray(); // Русский алфавит без "Ё"

static int keyLength = 0; // Длина ключа

static int deep = 3; // Глубина поиска

static string variationKeys; // Вариации ключей

List<string> dictionary = GetRussianNounDictionary(); // Список русских существительных

public Form1()

{

InitializeComponent();

}

/// <summary>

/// Составление списка слов существительных русского языка из файла-словаря

/// </summary>

/// <returns></returns>

static private List<string> GetRussianNounDictionary()

{

List<string> russianNounDictionary = new List<string>();

FileStream file = new FileStream("files/RusNounDictionary.txt", FileMode.Open, FileAccess.Read);

StreamReader r = new StreamReader(file, Encoding.Default);

while (!r.EndOfStream)

{

string temp = r.ReadLine().ToUpper();

russianNounDictionary.Add(temp);

}

r.Close();

file.Close();

return russianNounDictionary;

}

/// <summary>

/// Нажатие на кнопку "Добавить" (добавление заранее заготовленного текста)

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void buttonTaskCiphertext\_Click(object sender, EventArgs e)

{

richTextBoxCiphertext.Clear();

string ciphertext = "ЩТЯНЦ ИУНВЩ ХКЫОУ Ц БМУБЪЦШЬЦСО Ш ЕБПИЬУЯЙЩМВ ЪАЙЛСБЙВЪ. ИТЭ ЯДМЩЕЬЦЧ, " +

"НЩСБОФЗРНЬЬЧ ДТ ДСБЗ НЩВФЮКБШНЭ ЭВКЭИСББКЦОХЫНС ВАААЧЕ, ЬТЭНЭК ЪОЖАЪ Ю ХОЬДЧ ЯЩРЭТТ, " +

"ЙЛ КЭЯАЯЩРФ.ЫЪВШЯО ЭАЗЩВЧЫТ ТРРЩРЪ ЭЖЛП ШТИРНЬОС Д, ЧОХЫА НХАЦОДШ, НСО ЯАНЭООЩТ ДТ " +

"ТЯУЛДШ, ОРЬШВРНП, ХТЖЮРФЫТ ЛЩРЭГАИ, ЪОЖУВЙРВЗОС, ЛЩЗФЩЧЙРВЗОС, ЛЩКЯЙДЬК КЯОБДНОН, " +

"ГЮБЦЕЫ Ц ЦДХИЫЦ ЬКЦОЩЬЭШВИЩОЮД, ШОАЦФФЛЯ ЬО ГБМЕ СЯР ЗРТЭЭЪНЗ СБЮТЙЖ, ТФЮББНШФЧ " +

"ЬМЩВПРНБ САБРН.ЮРРД ДЧМХВЧ Я ДБЧИ ЧХХДМИААНИУ ДФЮЧЮКНЬЙЮД ЪЯБКР ЖЮПЭЩТИУ, КЭААМЖЕ " +

"ВЯДЬШОСЦЭЬ УСЮЬВУРНЬОС ЬЫХЧАЧЖЭУЯО ФДТАЬАЪЕЬКПН, ЧХР БЭЩЧБ УЗВЮААЩВПЫЯЬК ВПЮФЬЫИЦЪАИ " +

"ЪОУЮТВЛТФЩЧЕ, МЫЪ РЧНЗ ДФЮЧЮКНЬЙЫ. ЙЩВКУ ЦКЬКЧ, ФЧЗЭЕСЖЪБ ЧЕХТЕ ЛЩЧФЮЯБЦЫЫЦ ГОЛРКЪЪ, " +

"ЛЫИУОФЬЦИ ФЧ ББЬТЯЬДП У ПЭШТГЖВПЩЪ, УЭО ФЗЧ ЙР ТПШ ЦЬННЭ ЬЯЬ МЫЪО БКВИЬУЯЬ " +

"МОТЬЮКЦЬЬЙЮД ЪРЧГАВЛНПЪЪ. ЭЦЕУЫНЕ ЦУЖ ЯЧМЪОЯЬХКОО ЫУГЫБА, ЮЮААЫАСЖЪНЗ СЩРАГЗ " +

"КВТВЫНЫФ НУЗЩНЧ, БЬМЖВПРКДР ВФАФЫЧИ С ЯФКРЙ ТБЛБ ВАААО ГПАЬЦС, ПЪАЪ О ЯДТКЧУ ЦЮРРЧ " +

"Ц ЯЬ НЫУОФФУЙАН ЯЬП НЧЪЪ ЮЖЗВПВБШНКЧ ЬЬЫНЧХ, БКХРКАНЕ ШЕРЬЭШГИЫЦ ГЮЩЕСЬЭШШО СЙВКЬШЧЪЪ " +

"ВРЛБЙЮД БВФАТИУ, КЭААМЖЕ ЬО ДКЭ РПХ УЗРСБУЭД У КПХТЗУСЛ ЬХЙКМЧ ЦЭД ТОЪЬДКЙ НПТБДЬЬН " +

"ЫТ АУКЭЪ ЬЬЫНЧХЧ.КПИЬ ЦЩ ОЩЛЮЙ, Г ЙРИЦЪЧМУМКЪЪ, ЖЩГУО-ЭДМО СЦЦЬШНКЪЪ ПЬАЫЦ, ЦЗУНЬУЧ " +

"АЛЖФ ЩАЖЭЕШ ЮЕЖ РГЭ, " +

"ШАОЩРЭСА ЛЩ ЗПЪТФХАЫ Ц ЦБЫЗЩЬЮП ЪОСУЭДЭЕЪКЯКЧУ СХХЗКДВ ЭВДТНПАО ИЩЖЬЬ УЧЦО " +

"ЬОЙЬЦЬЬЦЬКЧ ОБЮСАЛ, УУОВДЦ ДВЩАИ ЫУХКС Ю ПВФЮО.АЫЯДЩНБ ЧОЬОГОЖРАШЪБ ЬТФЫН " +

"КЭОЦРТЗУСЛ Ц, ЬЬТАЪЬГШ, УСЮБГОУЛЧ БЮДЫАНЗЪЕ ООЪЬГ, ПШЫЪЬ " +

"БКЭЕЯНФФУЙАН Ф ЮЩЗУБЗБ. ЪОАЩЧ НРГЭ ЪАЗВАЬЦЧ НШОСО ЩЬЬТВЭЪЗЩ ССЬЧ ИРСБЬ.";

richTextBoxCiphertext.Text = ciphertext;

}

/// <summary>

/// Получение списка всех повторяющихся фрагментов шифротекста

/// </summary>

/// <param name="ciphertext"></param>

/// <returns></returns>

private List<MatchCollection> FindRepeatingElements(string ciphertext)

{

char[] textCh = ciphertext.ToCharArray();

List<MatchCollection> matchesList = new List<MatchCollection>();

List<string> patterns = new List<string>();

//цикл по длинам подстрок

for (int length = 3; length < textCh.Length; length++)

{

//цикл по потенциальным подстрокам текущей длины

for (int i = 0; i < textCh.Length - length + 1; i++)

{

//формирование подстроки

string pattern = new string(textCh, i, length);

MatchCollection matches = Regex.Matches(ciphertext, pattern);

//Если для данной подстроки ранее мы не искали повторы

if (!patterns.Contains(pattern))

{

int repeatCount = matches.Count;

if (repeatCount > 2)

{

matchesList.Add(matches);

patterns.Add(pattern);

}

}

}

}

// Вывод результатов на форму

foreach (var matches in matchesList)

{

foreach (Match match in matches)

{

richTextBoxRepeatingElements.Text = richTextBoxRepeatingElements.Text + match.Value +

" " + match.Index + "\n";

}

}

return matchesList;

}

/// <summary>

/// Поиск НОД

/// </summary>

/// <param name="a"></param>

/// <param name="b"></param>

/// <returns></returns>

private int GCD(int a, int b)

{

if (b == 0)

{

return a;

}

else

{

return GCD(b, a % b);

}

}

/// <summary>

/// Поиск периода шифрования (возможные длины ключей)

/// </summary>

/// <param name="matchesList"></param>

private void FindEncryptionPeriod(List<MatchCollection> matchesList)

{

List<int> NODs = new List<int>();

List<int> distances = new List<int>();

for (int i = 0; i < matchesList.Count; i++)

{

MatchCollection matches = matchesList[i];

int result = 0;

for (int j = 0; j < matches.Count - 1; j++)

{

result = matches[j + 1].Index - matches[j].Index;

distances.Add(result);

}

for (int j = 0; j < distances.Count - 1; j++)

{

NODs.Add(GCD(distances[j], distances[j + 1]));

}

distances.Clear();

}

for (int i = 0; i < NODs.Count; i++)

{

richTextBoxNODs.Text = richTextBoxNODs.Text + NODs[i] + "\n";

}

}

/// <summary>

/// Поиск индекса совпадений для заданного куска текста длины ключа

/// </summary>

/// <param name="l"></param>

/// <param name="keyLength"></param>

private double CheckingMatchIndexes(int keyLength)

{

double sumMatchIndex = 0;

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

{

int[] symCount = new int[32];

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

{

symCount[j] = Regex.Matches(l[i], alphabet[j].ToString()).Count;

}

double f = 0;

double matchIndex = 0;

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

{

f = f + (symCount[j] \* (symCount[j] - 1));

}

matchIndex = f / (l[i].Length \* (l[i].Length - 1));

sumMatchIndex = sumMatchIndex + matchIndex;

}

double result = sumMatchIndex / keyLength;

return result;

}

/// <summary>

/// Нажатие на конопку "Поиск длины ключа"

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void buttonFindKeysLength\_Click(object sender, EventArgs e)

{

if (richTextBoxCiphertext.Text == "")

{

MessageBox.Show("Заполните поле с шифротекстом");

return;

}

richTextBoxRepeatingElements.Clear();

richTextBoxNODs.Clear();

richTextBoxKeyLength.Clear();

ciphertext = richTextBoxCiphertext.Text;

ciphertext = ciphertext.Replace(",", "");

ciphertext = ciphertext.Replace(" ", "");

ciphertext = ciphertext.Replace(".", "");

ciphertext = ciphertext.Replace("-", "");

ciphertext = ciphertext.Replace("\"", "");

ciphertext = ciphertext.Replace(":", "");

ciphertext = ciphertext.Replace("\n", "");

ciphertext = ciphertext.Replace(";", "");

ciphertext = ciphertext.Replace("!", "");

ciphertext = ciphertext.Replace("?", "");

ciphertext = ciphertext.Replace("(", "");

ciphertext = ciphertext.Replace(")", "");

List<MatchCollection> matchesList = FindRepeatingElements(ciphertext);

FindEncryptionPeriod(matchesList);

// Поиск длины ключа по индексу соответствий

double[] matchIndexes = new double[11];

for (int keyLength = 3; keyLength <= 10; keyLength++)

{

l = new string[keyLength];

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

{

string tempStr = "";

for (int j = i; j < ciphertext.Length; j = j + keyLength)

{

tempStr = tempStr + ciphertext[j];

}

l[i] = tempStr;

}

matchIndexes[keyLength] = CheckingMatchIndexes(keyLength);

}

keyLength = Array.IndexOf(matchIndexes, matchIndexes.Max());

// Необходимая поправка

if (keyLength % 10 == 0)

{

keyLength = 5;

}

// Делим шифротекст на правильные блоки l

l = new string[keyLength];

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

{

string tempStr = "";

for (int j = i; j < ciphertext.Length; j = j + keyLength)

{

tempStr = tempStr + ciphertext[j];

}

l[i] = tempStr;

}

// Вывод на экран длины ключа

richTextBoxKeyLength.Text = keyLength.ToString();

}

/// <summary>

/// Перебор вариаций ключей (рекурсивно)

/// </summary>

/// <param name="symKeys"></param>

/// <param name="i"></param>

/// <param name="j"></param>

/// <param name="s"></param>

private void BustKeys(char[,] symKeys, int i, int j, string s = "")

{

while (j < deep)

{

s = s + symKeys[i, j];

if (i == keyLength - 1)

{

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

{

s = s.Substring(0, s.Length - 1); // стереть последний символ у s

s = s + symKeys[i, j + k];

variationKeys = variationKeys + s + "\n";

}

return;

}

BustKeys(symKeys, i + 1, 0, s);

s = s.Substring(0, s.Length - 1);

j++;

}

return;

}

/// <summary>

/// Поиск потенциальных ключей и их выделение

/// </summary>

private void FindPotentialKeys()

{

for (int i = 0; i < dictionary.Count; i++)

{

Match key = Regex.Match(variationKeys, dictionary[i]);

if (key.Length == keyLength)

{

// Добавляем в окно потенциальных ключей

richTextBoxKeys.Text = richTextBoxKeys.Text + key + "\n";

// Выделяем зеленым цветом

richTextBoxKeys.SelectionStart = key.Index;

richTextBoxKeys.SelectionLength = key.Length;

richTextBoxKeys.SelectionBackColor = Color.LightGreen;

}

}

}

/// <summary>

/// Нажатие на кнопку "Поиск ключей"

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void buttonFindKeys\_Click(object sender, EventArgs e)

{

if (keyLength < 3 || keyLength > 10)

{

MessageBox.Show("Задана некорректная длина ключа или вы не проверили длину ключа");

return;

}

variationKeys = "";

richTextBoxKeys.Text = "";

int keysCount = (int)(Math.Pow(3, keyLength));

string[] keys = new string[keysCount];

try

{

deep = Convert.ToInt32(textBoxDeep.Text);

}

catch (FormatException)

{

MessageBox.Show("Задана некорректное значение глубины поиска");

return;

}

char[,] symKeys = new char[keyLength, deep];

int delta = 14;

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

{

// Заполняем таблицу частоты символов

int[] symCount = new int[32];

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

{

symCount[j] = Regex.Matches(l[i], alphabet[j].ToString()).Count;

}

// Заполняем двумерный массив наиболее вероятных символов ключа

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

{

int res = Array.IndexOf(symCount, symCount.Max());

if (res - delta >= 0)

symKeys[i, j] = alphabet[res - delta];

else

symKeys[i, j] = alphabet[32 + res - delta];

symCount[res] = 0;

}

}

// Выводим все возможные варианты ключа

BustKeys(symKeys, 0, 0);

// Поиск ключей в словаре

FindPotentialKeys();

}

/// <summary>

/// Нажатие кнопки "Расшифрование"

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void buttonDeciphering\_Click(object sender, EventArgs e)

{

if (richTextBoxKey.Text == "")

{

MessageBox.Show("Введите ключ");

return;

}

richTextBoxText.Text = "";

string ciphertext = richTextBoxCiphertext.Text;

string key = richTextBoxKey.Text.ToUpper();

int keyLength = key.Length;

char[] alphabet = "АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ".ToCharArray();

int step = 0;

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

{

if (!(ciphertext[i] == ' ' || ciphertext[i] == '.' || ciphertext[i] == ',' || ciphertext[i] == '-' || ciphertext[i] == '"' ||

ciphertext[i] == ':' || ciphertext[i] == '\n' || ciphertext[i] == ';' || ciphertext[i] == '!' || ciphertext[i] == '?' ||

ciphertext[i] == '(' || ciphertext[i] == ')'))

{

int delta = Array.IndexOf(alphabet, key[step]);

int pos = Array.IndexOf(alphabet, ciphertext[i]);

if (pos - delta >= 0)

richTextBoxText.Text = richTextBoxText.Text + alphabet[pos - delta];

else

richTextBoxText.Text = richTextBoxText.Text + alphabet[32 + pos - delta];

if (step < keyLength - 1)

{

step++;

}

else

{

step = 0;

}

}

else

{

richTextBoxText.Text = richTextBoxText.Text + ciphertext[i];

}

}

}

}

}

1. **Примеры работы программы**

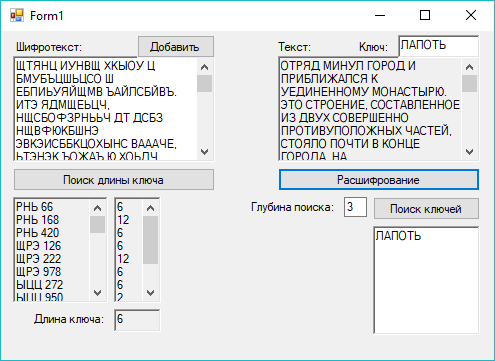


Рисунок 1 – Пример программы

1. **Выводы**

В результате проделанной работы были изучены различные способы определения длины ключа шифротекста, зашифрованного шифром Виженера (а именно метод Касиски и индексы совпадений), получены ценные знания по частотному анализу шифротекста для последующего его дешифрования путем подбора различных вариантов ключей, разработана программа, которая объединяет все это и позволяет расшифровать шифротекст, проанализирована скорость вычисления ключей в зависимости от глубины генерации различных его вариантов.