Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Исследование криптографических шифров на основе**

**подстановки (замены) символов**

Студент: Рубашек А. А.

ФИТ 3 курс 5 группа

Преподаватель:

Савельева Маргарита Геннадьевна

1. **Цель работы**

Изучение и приобретение практических навыков разработки и использования приложений для реализации подстановочных шифров.

1. **Задание**

Разработать авторское приложение в соответствии с целью лабораторной работы. Приложение должно реализовывать следующие операции:

1. выполнять зашифрование/расшифрование текстовых документов (объемом не менее 5 тысяч знаков) созданных на основе алфавита языка в соответствии с нижеследующей таблицей вариантов задания; при этом следует использовать шифры подстановки из третьего столбца данной таблицы;
2. формировать гистограммы частот появления символов для исходного и зашифрованного сообщений;
3. оценивать время выполнения операций зашифрования/расшифрования.
4. **Ход работы**

В соответствии с вариантом, алфавит выбран русский, для первого задания использовался шифр Цезаря с ключевым словом, ключевое слово – безопасность, для второго задания использовалась таблица Трисемуса, ключевое слово – безопасность.

Для обоих заданий следовало оценить время выполнения операций шифрования/дешифрования сообщений. Для этого был использован специальный класс Stopwatch на языке C#, который предоставляет функциональность для измерения времени выполнения операций и который представлен в директиве System.Diagnostics. Пример использования объекта данного класса представлен на рисунке 3.1.



Рисунок 3.1 – Пример использования класса Stopwatch

Также, для этих заданий следовало сформировать гистограммы частот появления символов для исходного и зашифрованного сообщений. Для этой цели была разработана функция CountCharacterFrequency, представленная в приложении 1.

Для выполнения первого задания были разработаны функции EncryptCaesar и DecryptCaesar, которые осуществляют шифрование и дешифрование текста с использованием алгоритма Цезаря по ключу. Они представлены в приложении 2.

Входные данные и результаты выполнения данных функций представлены на рисунках 3.2 – 3.4.

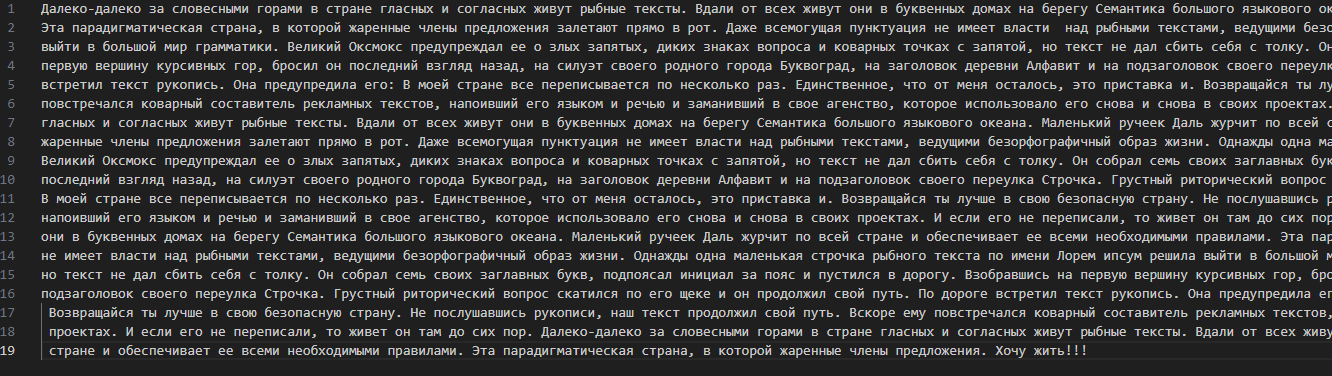


Рисунок 3.2 – Фрагмент текста расшифрованного с помощью функции EncryptCaesar

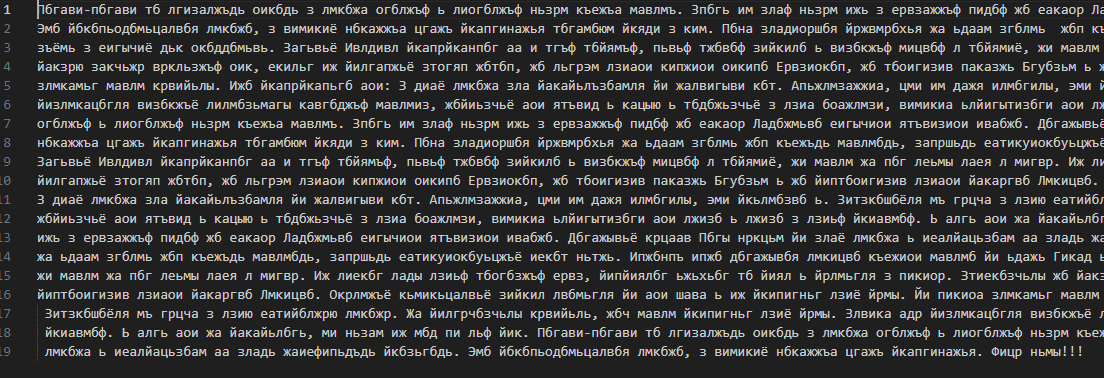


Рисунок 3.3 – Фрагмент текста зашифрованного с помощью функции DecryptCaesar



Рисунок 3.4 – Оценка скорости выполнения шифрования/дешифрования текста

Частота появления символов в этих текстовых файлах представлена на рисунках 3.5 и 3.6 соответственно.

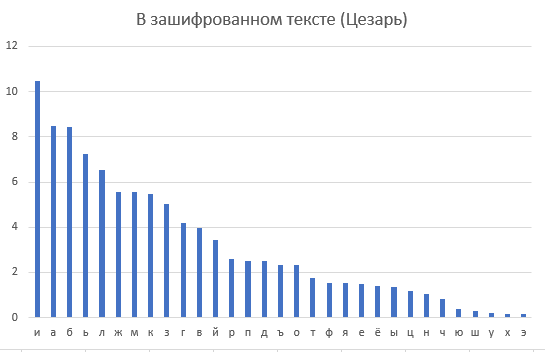


Рисунок 3.5– Гистограмма частот появления символов для зашифрованного текста



Рисунок 3.6 – Гистограмма частот появления символов для расшифрованного текста

Для выполнения второго задания были разработаны функции EncryptTrisemus, DecryptTrisemus и TrisemusTable, которые реализуют алгоритм шифрования текста с помощью таблицы Трисемуса. Код реализации этих функций представлен в приложении 3.

В результате получаем следующие файлы и вывод, представленные на рисунках 3.7 – 3.9.

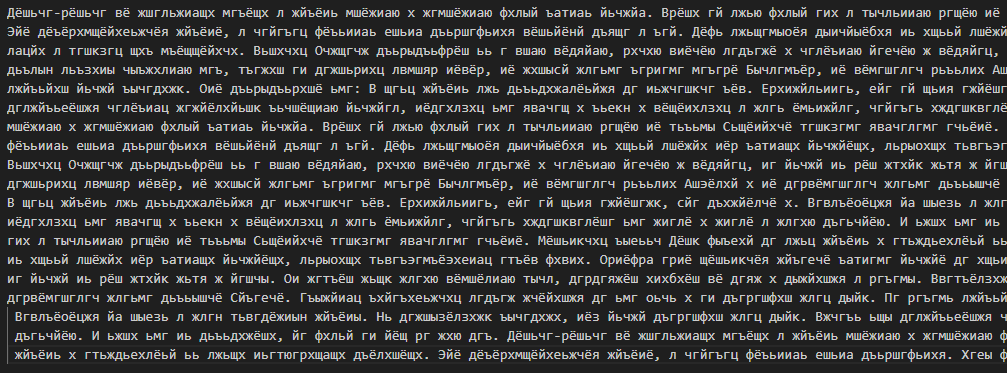


Рисунок 3.7 – Текст, зашифрованный с помощью функции EncryptTrisemus

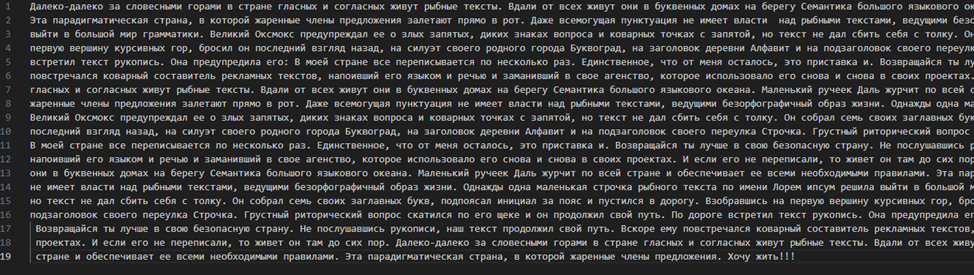


Рисунок 3.8 – Текст, расшифрованный с помощью функции DecryptTrisemus



Рисунок 3.9 – Оценка скорости выполнения шифрования/дешифрования текста

Частота появления символов в этих текстовых файлах представлена на рисунках 3.10 и 3.11 соответственно.



Рисунок 3.10– Гистограмма частот появления символов для зашифрованного текста

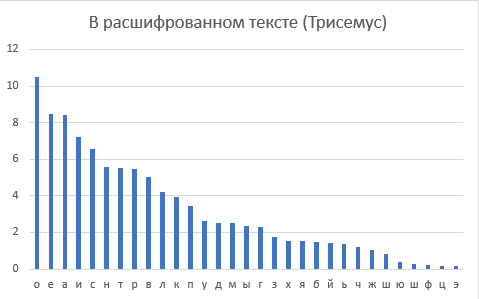


Рисунок 3.11– Гистограмма частот появления символов для расшифрованного текста

**Вывод**

В ходе лабораторной работы были изучены и приобретены практические навыки разработки и использования приложений для реализации подстановочных шифров.

Кроме того, были сформированы гистограммы появления частот на основе зашифрованных/расшифрованных сообщений. Из графиков видно, что частота символов в зашифрованном тексте существенно отличается от частоты символов в исходном тексте. Это обусловлено тем, что подстановочные шифры заменяют каждый символ открытого текста на другой символ или последовательность символов в зашифрованном тексте. Так как подстановка была случайной и не учитывала частоту появления символов в открытом тексте, то частотный анализ зашифрованного текста сильно отличается от анализа открытого текста.

Также было разработано авторское приложение в соответствии с целью лабораторной работы.

**ПРИЛОЖЕНИЕ 1**

Функция для определения частот появления символов

static void CountCharacterFrequency(string text)

{

// Создаем словарь, который будет содержать частоту появления каждого символа английского алфавита в тексте

Dictionary<char, int> characterFrequency = new Dictionary<char, int>();

// Проходим по каждому символу в тексте и увеличиваем его частоту на 1 в словаре, если это символ английского алфавита

foreach (char c in text)

{

text = text.ToLower();

if (char.IsLetter(c) && char.IsLower(c))

{

if (characterFrequency.ContainsKey(c))

{

characterFrequency[c]++;

}

else

{

characterFrequency.Add(c, 1);

}

}

}

// Вычисляем общее количество символов английского алфавита в тексте

int totalCharacters = characterFrequency.Sum(x => x.Value);

// Сортируем словарь по убыванию частоты появления символов и выводим пары "символ - частота появления в процентах"

foreach (KeyValuePair<char, int> pair in characterFrequency.OrderByDescending(key => key.Value))

{

double frequencyPercentage = (double)pair.Value / totalCharacters \* 100;

Console.WriteLine("{0} - {1:F2}%", pair.Key, frequencyPercentage);

}

}

**ПРИЛОЖЕНИЕ 2**

Функции шифрования и дешифрования текста с помощью шифра Цезаря

static void EncryptCaesar(string inputFile, string encryptedFile, string key)

{

Console.ForegroundColor = ConsoleColor.Green;

string alpha = GenerateAlphabet(key);

string inputText = File.ReadAllText(inputFile);

string encryptedText = "";

foreach (char c in inputText)

{

// Переводим символ в нижний регистр

char lowercaseChar = char.ToLower(c);

int index = Alphabet.IndexOf(lowercaseChar);

if (char.IsLetter(c))

{

// Если символ найден в алфавите, получаем соответствующий зашифрованный символ

char encryptedChar = alpha[index];

// Проверяем оригинальный регистр символа и переводим зашифрованный символ в тот же регистр

if (char.IsUpper(c))

{

encryptedChar = char.ToUpper(encryptedChar);

}

encryptedText += encryptedChar;

}

else

{

// Если символ не найден в алфавите, добавляем его как есть

encryptedText += c;

}

}

Console.WriteLine("Зашифрованный текст:\n---------------\n" + encryptedText);

File.WriteAllText(encryptedFile, encryptedText);

Console.ResetColor();

}

static void DecryptCaesar(string encryptedFile, string decryptedFile, string key)

{

Console.ForegroundColor = ConsoleColor.Green;

string alpha = GenerateAlphabet(key);

string encryptedText = File.ReadAllText(encryptedFile);

string decryptedText = "";

foreach (char c in encryptedText)

{

// Переводим символ в нижний регистр

char lowercaseChar = char.ToLower(c);

int index = alpha.IndexOf(lowercaseChar);

if (char.IsLetter(c))

{

// Если символ найден в алфавите, получаем соответствующий расшифрованный символ

char decryptedChar = Alphabet[index];

// Проверяем оригинальный регистр символа и переводим расшифрованный символ в тот же регистр

if (char.IsUpper(c))

{

decryptedChar = char.ToUpper(decryptedChar);

}

decryptedText += decryptedChar;

}

else

{

// Если символ не найден в алфавите, добавляем его как есть

decryptedText += c;

}

}

Console.WriteLine("Расшифрованный текст:\n---------------\n" + decryptedText);

File.WriteAllText(decryptedFile, decryptedText);

Console.ResetColor();

**ПРИЛОЖЕНИЕ 3**

Функции шифровки и дешифровки Трисемуса

public static char[,] TrisemusTable(string keyword)

{

var table = new char[rows, cols];

var index = 0;

foreach (var c in keyword.Distinct())

{

table[index / cols, index % cols] = c;

index++;

}

foreach (var c in Alphabet)

{

if (index >= rows \* cols)

break;

if (!keyword.Contains(c))

{

table[index / cols, index % cols] = c;

index++;

}

}

return table;

}

static void EncryptTrisemus(string inputFile, string encryptedFile, char[,] table)

{

Console.ForegroundColor = ConsoleColor.Green;

try

{

string text = File.ReadAllText(inputFile);

var encryptedText = new StringBuilder(text.Length);

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

{

char currentChar = text[i];

bool isFound = false;

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

{

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

{

if (table[row, col] == currentChar)

{

int newRow = (row + 1) % rows;

encryptedText.Append(table[newRow, col]);

isFound = true;

break;

}

}

if (isFound)

{

break;

}

}

if (!isFound)

{

encryptedText.Append(currentChar);

}

}

Console.WriteLine("Зашифрованный текст:\n---------------\n" + encryptedText);

File.WriteAllText(encryptedFile, encryptedText.ToString());

}

catch (Exception e)

{

Console.WriteLine("Error: " + e.Message);

}

Console.ResetColor();

}

static void DecryptTrisemus(string encryptedFile, string decryptedFile, char[,] table)

{

Console.ForegroundColor = ConsoleColor.Green;

try

{

var text = File.ReadAllText(encryptedFile);

var decryptedText = new StringBuilder(text.Length);

for (var i = 0; i < text.Length; ++i)

{

bool isReplaced = false;

for (var row = 0; row < rows && !isReplaced; ++row)

{

for (var column = 0; column < cols; ++column)

{

if (text[i] == table[row, column])

{

var newRow = (row == 0) ? rows - 1 : row - 1;

decryptedText.Append(table[newRow, column]);

isReplaced = true;

break;

}

}

}

if (!isReplaced)

{

decryptedText.Append(text[i]);

}

}

Console.WriteLine("Расшифрованный текст:\n---------------\n" + decryptedText);

File.WriteAllText(decryptedFile, decryptedText.ToString());

}

catch (Exception e)

{

Console.WriteLine("Error: " + e.Message);

}

Console.ResetColor();

}