Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Исследование криптографических шифров на основе перестановки 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 перестановочных 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 500 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 использовалась маршрутная перестановка (маршрут – спираль), для второго задания использовалась множественная перестановка, ключевое слово – «РУБАШКА».

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

Для выполнения первого задания были разработаны функции SpiralRouteEncrypt и SpiralRouteDecrypt, которые осуществляют шифрование и дешифрование текста с использованием маршрутной перестановки. Они представлены в приложении 1 и 2 соответственно.

Входные данные и результаты выполнения данных функций представлены на рисунках 3.2 – 3.3.

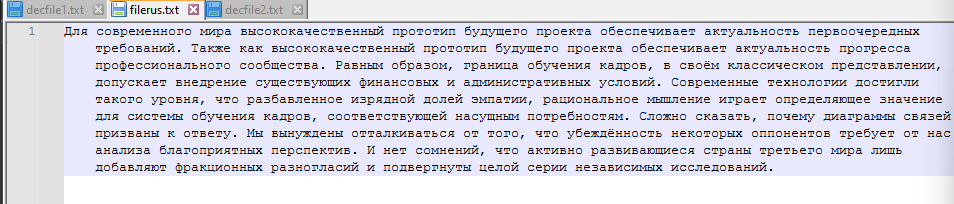


Рисунок 3.2 – Текст, зашифрованный с помощью функции SpiralRouteEncrypt

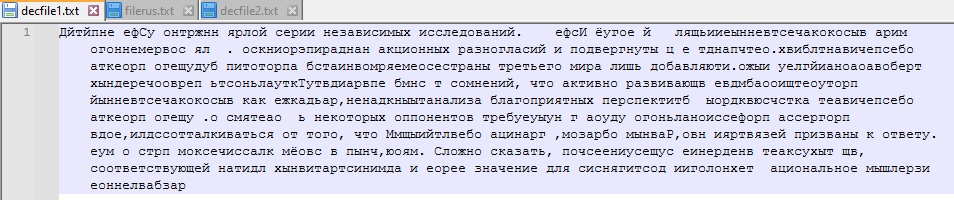


Рисунок 3.3 – Текст, расшифрованный с помощью функции SpiralRouteDecrypt

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.

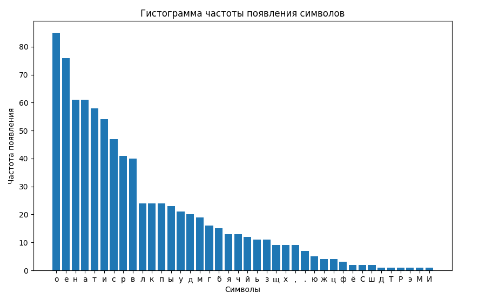


Рисунок 3.5 – Гистограмма частот появления символов для зашифрованного текста

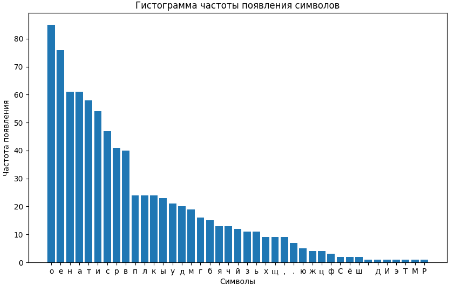


Рисунок 3.6 – Гистограмма частот появления символов для расшифрованного текста

Для выполнения второго задания были разработаны функции MultiplePermutationEncrypt и MultiplePermutationEncrypt, которые реализуют алгоритм шифрования текста с помощью алгоритма множественной перестановки (ключ – имя и фамилия). Код реализации этих функций представлен в приложениях 3 и 4.

В результате получаем следующие файлы и вывод, представленные на рисунках 3.7 – 3.8.

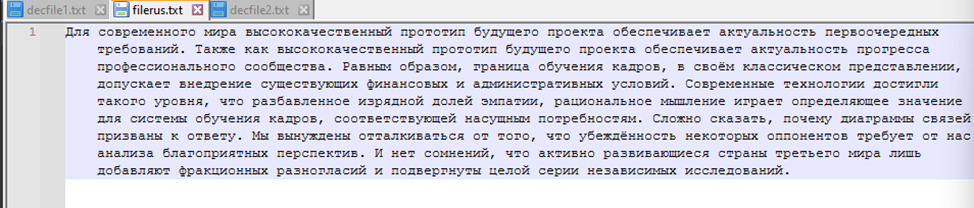


Рисунок 3.7 – Текст, зашифрованный с помощью функции MultiplePermutationEncrypt

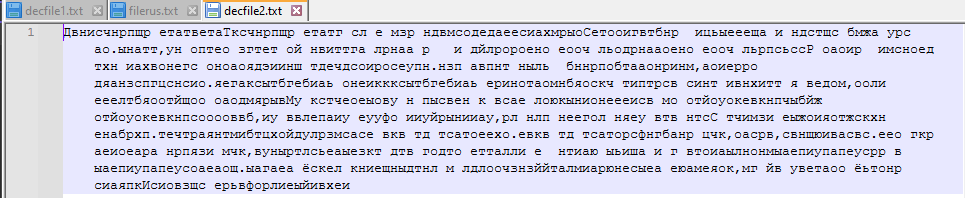


Рисунок 3.8 – Текст, расшифрованный с помощью функции MultiplePermutationDecrypt

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.

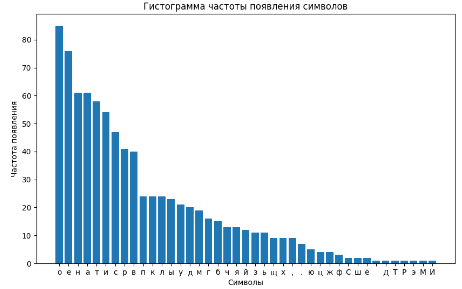


Рисунок 3.10 – Гистограмма частот появления символов для зашифрованного текста

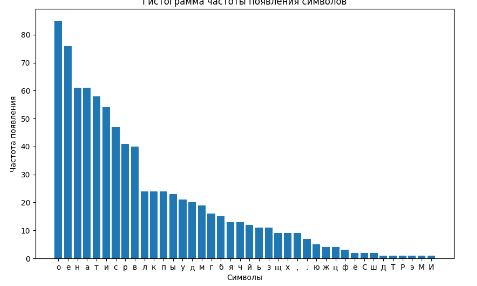


Рисунок 3.11 – Гистограмма частот появления символов для расшифрованного текста

**Вывод**

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

Кроме того, были сформированы гистограммы появления частот на основе зашифрованных/расшифрованных сообщений. Из графиков видно, что частота символов в зашифрованном тексте вовсе не отличается от частоты символов в исходном тексте. Это обусловлено тем, что перестановочные шифры не заменяют символы открытого текста на другие символы или последовательность символов, а просто меняют порядок их следования.

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

**Приложение 1**

static string SpiralRouteEncrypt(string input, int rows, int columns)

{

char[,] matrix = new char[rows, columns];

int index = 0;

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

{

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

{

if (index < input.Length)

{

matrix[i, j] = input[index];

index++;

}

else

{

matrix[i, j] = ' ';

}

}

}

StringBuilder encryptedText = new StringBuilder();

int topRow = 0, bottomRow = rows - 1, leftColumn = 0, rightColumn = columns - 1;

while (topRow <= bottomRow && leftColumn <= rightColumn)

{

for (int i = topRow; i <= bottomRow; i++) // движение вниз

{

encryptedText.Append(matrix[i, leftColumn]);

}

leftColumn++;

for (int i = leftColumn; i <= rightColumn; i++) // движение вправо

{

encryptedText.Append(matrix[bottomRow, i]);

}

bottomRow--;

if (leftColumn <= rightColumn) // условие для предотвращения двойного прохода

{

for (int i = bottomRow; i >= topRow; i--) // движение вверх

{

encryptedText.Append(matrix[i, rightColumn]);

}

rightColumn--;

}

if (topRow <= bottomRow) // условие для предотвращения двойного прохода

{

for (int i = rightColumn; i >= leftColumn; i--) // движение влево

{

encryptedText.Append(matrix[topRow, i]);

}

topRow++;

}

}

return encryptedText.ToString();

}

**Приложение 2**

static void SpiralRouteEncryptDecrypt(string text, out string encrypt)

{

Console.ForegroundColor = ConsoleColor.Blue;

Console.WriteLine("Маршрутная перестановка:");

Console.ResetColor();

int a = 25;

int b = 20;

// Замер времени шифрования

Stopwatch stopwatch = Stopwatch.StartNew();

string encryptedText = SpiralRouteEncrypt(text, a, b);

stopwatch.Stop();

Console.WriteLine($"Зашифрованный текст:\n {encryptedText}");

Console.ForegroundColor = ConsoleColor.Blue;

Console.WriteLine($"Время шифрования: {stopwatch.ElapsedMilliseconds} мс");

Console.ResetColor();

encrypt = encryptedText;

// Замер времени расшифрования

stopwatch.Restart();

string decryptedText = SpiralRouteDecrypt(encryptedText, a, b);

stopwatch.Stop();

Console.WriteLine($"Расшифрованный текст:\n {decryptedText}");

Console.ForegroundColor = ConsoleColor.Blue;

Console.WriteLine($"Время расшифрования: {stopwatch.ElapsedMilliseconds} мс");

Console.ResetColor();

}

**Приложение 3**

static string MultiplePermutationEncrypt(string plaintext, string keyword)

{

// Преобразуем ключевое слово в массив символов и удаляем повторяющиеся символы

char[] keywordChars = keyword.ToLower().Where(char.IsLetter).Distinct().OrderBy(c => c).ToArray();

// Создаем словарь для хранения индексов символов в ключевом слове

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

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

{

keywordIndex[keywordChars[i]] = i;

}

// Создаем массив индексов для сортировки символов ключевого слова

int[] sortedIndexes = keywordChars.Select((c, index) => index).ToArray();

// Сортируем индексы по алфавиту

Array.Sort(keywordChars, sortedIndexes);

// Создаем список строк для хранения зашифрованных столбцов

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

// Заполняем столбцы зашифрованными символами

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

{

string column = "";

for (int j = i; j < plaintext.Length; j += keywordChars.Length)

{

column += plaintext[j];

}

columns.Add(column);

}

// Формируем зашифрованный текст, объединяя символы из столбцов в соответствии с отсортированными индексами

string ciphertext = "";

foreach (int index in sortedIndexes)

{

ciphertext += columns[index];

}

return ciphertext;

}

**Приложение 4**

static string MultiplePermutationDecrypt(string ciphertext, string keyword)

{

// Преобразуем ключевое слово в массив символов и удаляем повторяющиеся символы

char[] keywordChars = keyword.ToLower().Where(char.IsLetter).Distinct().OrderBy(c => c).ToArray();

// Создаем словарь для хранения индексов символов в ключевом слове

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

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

{

keywordIndex[keywordChars[i]] = i;

}

// Создаем массив индексов для сортировки символов ключевого слова

int[] sortedIndexes = keywordChars.Select((c, index) => index).ToArray();

// Сортируем индексы по алфавиту

Array.Sort(keywordChars, sortedIndexes);

// Вычисляем количество символов в каждом столбце

int columnLength = ciphertext.Length / keywordChars.Length;

int remainder = ciphertext.Length % keywordChars.Length;

// Создаем словарь для хранения длин столбцов

Dictionary<int, int> columnLengths = new Dictionary<int, int>();

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

{

columnLengths[sortedIndexes[i]] = i < remainder ? columnLength + 1 : columnLength;

}

// Создаем список строк для хранения столбцов шифротекста

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

// Добавляем символы в столбцы

int currentIndex = 0;

foreach (var pair in columnLengths.OrderBy(x => x.Key))

{

columns.Add(ciphertext.Substring(currentIndex, pair.Value));

currentIndex += pair.Value;

}

// Создаем массив строк для хранения расшифрованных столбцов

string[] sortedColumns = new string[keywordChars.Length];

// Распределяем столбцы в порядке, соответствующем отсортированным индексам ключевого слова

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

{

sortedColumns[i] = columns[keywordIndex[keywordChars[i]]];

}

// Формируем расшифрованный текст, объединяя символы из столбцов

string plaintext = "";

for (int i = 0; i < columnLength + (remainder > 0 ? 1 : 0); i++)

{

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

{

if (i < sortedColumns[j].Length)

{

plaintext += sortedColumns[j][i];

}}}

return plaintext;

}