Учреждение образования

«Белорусский государственный технологический университет»

**Защита информации и надежность информационных систем**

**Лабораторная работа №9**

СЖАТИЕ/РАСПАКОВКА ДАННЫХ НА ОСНОВЕ СТАТИСТИЧЕСКИХ МЕТОДОВ

Выполнил:

Студент 3 курса 2 группы ФИТ

Максимова Вера Владимировна

Проверила:

Ржеутская Надежда Викентьевна

**2022 г.**

Цель: приобретение практических навыков использования статистических методов Шеннона − Фано и Хаффмана (ShannonFano and Huffman coding) для сжатия/распаковки данных.

Задачи:

1. Закрепить теоретические знания по алгебраическому описанию и использованию методов сжатия/распаковки (архивации/разархивации) данных на основе методов Шеннона − Фано и Хаффмана.

2. Разработать приложение для реализации методов Шеннона − Фано и Хаффмана.

3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

namespace lab9

{

public class ShannonFanoSymbol

{

public char symbol;

public int count;

public double viorite;

public string code;

public ShannonFanoSymbol(char sim, int count, double vior, string code)

{

this.viorite = vior;

this.symbol = sim;

this.count = count;

this.code = code;

}

public static List<ShannonFanoSymbol> AddSymbols(List<ShannonFanoSymbol> symbols, string line)

{

foreach (var character in line)

{

if (symbols.Find(x => x.symbol == character) == null)

{

symbols.Add(new ShannonFanoSymbol(character, 1, 0.0, ""));

}

else

{

symbols.Where(x => x.symbol == character).ToList().ForEach(x => x.count++);

}

}

return symbols;

}

public static void Show(List<ShannonFanoSymbol> symbols)

{

foreach (var symbol in symbols)

{

Console.Write("Symbol: {0} Amount:: {1} ", symbol.symbol, symbol.count);

if (symbol.viorite != 0) {

Console.Write("P: {0}", symbol.viorite);

}

if (symbol.code != "") {

Console.Write(" Code: {0}", symbol.code);

}

Console.WriteLine();

}

Console.WriteLine();

}

}

class Program

{

static void Main(string[] args)

{

List<ShannonFanoSymbol> symbols = new List<ShannonFanoSymbol>();

using (StreamReader stream = new StreamReader(@"..\..\text\latin.txt", Encoding.Default))

{

string messagef;

while ((messagef = stream.ReadLine()) != null)

{

symbols = ShannonFanoSymbol.AddSymbols(symbols, messagef);

}

}

Console.WriteLine();

Console.WriteLine("Analysis of statistic data:");

Console.WriteLine("Table of symbols: ");

Console.WriteLine();

ShannonFanoSymbol.Show(symbols);

double symbolssum = symbols.Sum(x => x.count);

Console.WriteLine("Sum of latin symbol in text: " + symbolssum);

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

{

symbols[i].viorite = symbols[i].count / symbolssum;

}

Console.WriteLine("Sum of table's symbols: " + (symbols.Sum(x => x.viorite)));

Console.WriteLine();

symbols = symbols.OrderByDescending(x => x.viorite).ToList();

ShannonFanoSymbol.Show(symbols);

Console.WriteLine();

Console.WriteLine("Table of code for each symbol: " );

Console.WriteLine();

symbols = AddCodes(symbols);

foreach (var symbol in symbols)

{

symbol.code = symbol.code.Remove(symbol.code.Length - 1, 1);

}

ShannonFanoSymbol.Show(symbols);

string blockofFIO = "Vera Maksimava";

string decodingOfFIO = "";

foreach (var charFIO in blockofFIO)

{

decodingOfFIO += (symbols.Where(x => x.symbol == charFIO).FirstOrDefault()).code;

}

Console.WriteLine("Inputed text: ");

Console.WriteLine(blockofFIO);

Console.WriteLine("Encoded: ");

Console.WriteLine(decodingOfFIO);

Console.WriteLine("Amount of symbols in ASCII: " + blockofFIO.Count() \* 8);

Console.WriteLine("Amount of symbols in encoded table: " + decodingOfFIO.Count());

Console.WriteLine("\nDecoded: ");

string Encoded = "";

string FIOdecoded = "";

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

{

Encoded += decodingOfFIO[i];

if (symbols.Find(x => x.code == Encoded) != null)

{

FIOdecoded += symbols.Find(x => x.code == Encoded).symbol;

Encoded = "";

}

}

Console.WriteLine(FIOdecoded);

Console.WriteLine();

Console.WriteLine("Dynamic, based on analysis of inputedd symbols: ");

Console.WriteLine();

symbols.Clear();

string message = "Maksimava";

symbols = ShannonFanoSymbol.AddSymbols(symbols, message);

ShannonFanoSymbol.Show(symbols);

symbolssum = symbols.Sum(x => x.count);

Console.WriteLine("Sum of symbols in latin: " + symbolssum);

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

{

symbols[i].viorite = symbols[i].count / symbolssum;

}

Console.WriteLine("Sum of P for each symbols: " + (symbols.Sum(x => x.viorite)));

Console.WriteLine();

symbols = symbols.OrderByDescending(x => x.viorite).ToList();

ShannonFanoSymbol.Show(symbols);

Console.WriteLine();

Console.WriteLine("Code for each symbol: ");

Console.WriteLine();

symbols = AddCodes(symbols);

foreach (var symbol in symbols)

{

symbol.code = symbol.code.Remove(symbol.code.Length - 1, 1);

}

ShannonFanoSymbol.Show(symbols);

blockofFIO = "Maksimava";

decodingOfFIO = "";

foreach (var charFIO in blockofFIO)

{

decodingOfFIO += (symbols.Where(x => x.symbol == charFIO).First()).code;

}

Console.WriteLine("Inputed text: ");

Console.WriteLine(blockofFIO);

Console.WriteLine("Encoded: ");

Console.WriteLine(decodingOfFIO);

Console.WriteLine("Amount of symbols in ASCII: " + blockofFIO.Count() \* 8);

Console.WriteLine("Amount of symbols in encoded table: " + decodingOfFIO.Count());

Console.WriteLine("\nDecoded:");

Encoded = "";

FIOdecoded = "";

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

{

Encoded += decodingOfFIO[i];

if (symbols.Find(x => x.code == Encoded) != null)

{

FIOdecoded += symbols.Find(x => x.code == Encoded).symbol;

Encoded = "";

}

}

Console.WriteLine(FIOdecoded);

}

public static List<ShannonFanoSymbol> AddCodes(List<ShannonFanoSymbol> symbols)

{

int counter = 0;

double probability = 0.0;

List<ShannonFanoSymbol> first = new List<ShannonFanoSymbol>();

List<ShannonFanoSymbol> second = new List<ShannonFanoSymbol>();

while (probability < (symbols.Sum(x => x.viorite) / 2))

{

probability += symbols[counter].viorite;

counter++;

}

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

{

symbols[i].code += "0";

first.Add(symbols[i]);

}

for (int i = counter; i < symbols.Count; i++)

{

symbols[i].code += "1";

second.Add(symbols[i]);

}

if (symbols.Count > 1)

{

first = AddCodes(first);

second = AddCodes(second);

first.AddRange(second);

symbols = first;

}

return symbols;

}

}

}