Министерство образования Республики Беларусь

Учреждение образования

Белорусский государственный университет информатики и

радиоэлектроники

Кафедра ЭВМ

Отчет

по лабораторной работе № 3

Выполнили: Проверил:

студенты группы 050503 доцент, к.т.н

Торчилов П.Р. Кобяк И. П.

Белько С.Д.

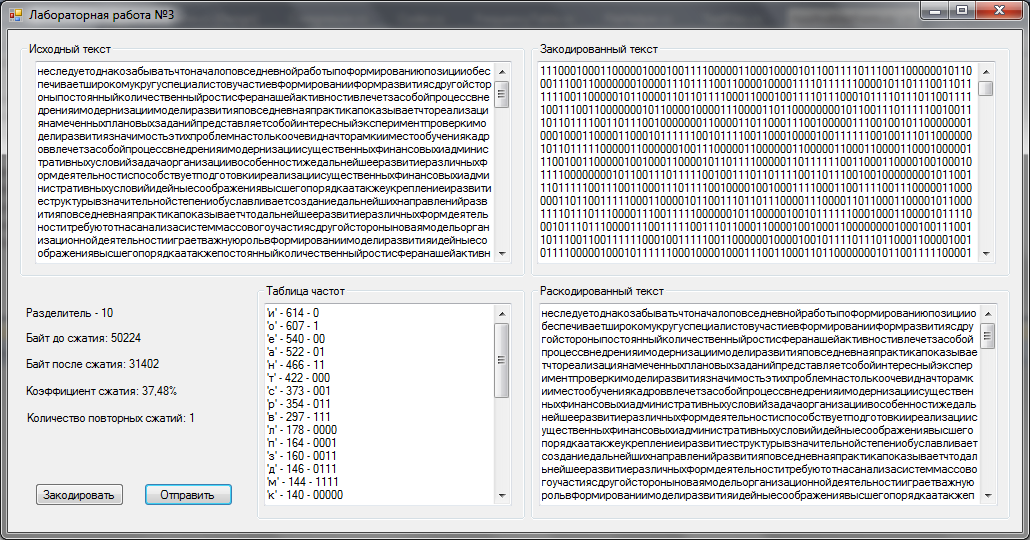
Минск 2014

1. **Исходные данные.**

Сущность метода заключается в следующем:

1. На первом этапе выбирается группа символов-разделителей букв.
2. На втором этапе выполняется частотный анализ текста, подлежащего шифрованию. По результатам анализа выполняется построение таблицы частот символов.
3. На третьем этапе генерируется последовательность символов, не содержащие кодов разделителей.
4. После формирования кодирующих наборов, символы, упорядоченные по возрастанию частоты, наделяются кодами по убыванию длины.
5. Полученный зашифрованный текст может быть разделен на группы и каждой группе может быть присвоена своя отвлеченная буква.
6. Вновь полученный текст повторно шифруется тем же алгоритмом.
7. Если вновь полученный текст получился большего размера, то система шифрования отбрасывает его перейдя к предыдущей.
8. **Задание.**

Выбрать разделитель, составить таблицу частот символов. Сгенерировать коды, не содержащие символов разделителей. Выполнить шифрование текста и передачу в канал связи. Приемнику необходимо расшифровать полученный текст.

1. **Результаты работы программы.**
2. **Листинг программы**

public class FileHelper

{

public String ReadFile(String path)

{

var stringBuilder = new StringBuilder();

using (var streamReader = new StreamReader(path, Encoding.Default))

{

var rawText = streamReader.ReadToEnd();

foreach (var symbol in rawText)

{

if ((symbol >= 'А' && symbol <= 'Я') ||

(symbol >= 'а' && symbol <= 'я'))

{

stringBuilder.Append(Char.ToLower(CorrectSymbol(symbol)));

}

}

}

return stringBuilder.ToString();

}

private char CorrectSymbol(char symbol)

{

switch (Char.ToLower(symbol))

{

case 'ё':

{

return 'е';

}

default :

{

return symbol;

}

}

}

}

public class FrequencyTable

{

public SortedDictionary<char, int> SybmolsFrequency { get; private set; }

public List<char> SortedSymbols

{

get

{

return SybmolsFrequency.Keys.OrderByDescending(key =>

SybmolsFrequency[key]).ToList();

}

}

public FrequencyTable()

{

SybmolsFrequency = new SortedDictionary<char, int>();

}

public void CreateFrequencyTable(String source)

{

foreach (var sybmol in source)

{

if (SybmolsFrequency.Keys.Contains(sybmol))

{

SybmolsFrequency[sybmol]++;

}

else

{

SybmolsFrequency[sybmol] = 1;

}

}

}

}

public class Coder

{

private const String Separator = "10";

public Dictionary<char, String> CodesTable { get; private set; }

public Dictionary<String, char> SymbolsTable { get; private set; }

public Coder()

{

CodesTable = new Dictionary<char, String>();

SymbolsTable = new Dictionary<String, char>();

}

public String Encode(String text, FrequencyTable table)

{

var codes = GenerateCodes(table);

var symbolsFrequency = table.SortedSymbols;

FillTables(symbolsFrequency, codes);

var stringBuiler = new StringBuilder();

foreach (var symbol in text)

{

stringBuiler.Append(CodesTable[symbol]);

stringBuiler.Append(Separator);

}

stringBuiler.Remove(stringBuiler.Length - Separator.Length, Separator.Length);

return stringBuiler.ToString();

}

public String Decode(String encodedText)

{

var stringBuilder = new StringBuilder();

var symbols = encodedText.Split(new[] { Separator }, StringSplitOptions.None);

foreach (var symbol in symbols)

{

if (SymbolsTable.ContainsKey(symbol))

{

stringBuilder.Append(SymbolsTable[symbol]);

}

}

return stringBuilder.ToString();

}

private void FillTables(IList<char> symbolsFrequency, IList<String> codes)

{

for (var i = 0; i < codes.Count; i++)

{

CodesTable.Add(symbolsFrequency[i], codes[i]);

SymbolsTable.Add(codes[i], symbolsFrequency[i]);

}

}

private IList<String> GenerateCodes(FrequencyTable table)

{

var codes = new List<String>();

var number = 1;

while (true)

{

for (var i = 0; i < Math.Pow(2, number); i++)

{

var code = this.GetBinary(i, number);

if (IsValid(code))

{

codes.Add(code);

}

if (codes.Count >= table.SybmolsFrequency.Count)

{

break;

}

}

if (codes.Count >= table.SybmolsFrequency.Count)

{

break;

}

number++;

}

return codes;

}

private bool IsValid(String code)

{

if (!code.Contains(Separator))

{

return true;

}

return false;

}

private String GetBinary(int value, int length)

{

var binary = Convert.ToString(value, 2);

if (length > binary.Length)

{

return new String('0', length - binary.Length) + binary;

}

return binary;

}

}

public class Compressor

{

public Dictionary<char, String> SymbolsTable { get; private set; }

public Dictionary<String, char> CodesTable { get; private set; }

public int ExtraSymbols = 0;

public Compressor()

{

SymbolsTable = new Dictionary<char, String>();

CodesTable = new Dictionary<String, char>();

CreateCodesTabel();

}

public String GetNewText(String encodedText)

{

var stringBuilder = new StringBuilder();

encodedText = AlingText(encodedText);

for (var i = 0; i < encodedText.Length; i += 5)

{

var code = encodedText.Substring(i, 5);

var symbol = CodesTable[code];

stringBuilder.Append(symbol);

}

return stringBuilder.ToString();

}

public String DecompressText(String text)

{

var stringBuilder = new StringBuilder();

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

{

var code = SymbolsTable[text[i]];

stringBuilder.Append(code);

}

stringBuilder.Remove(stringBuilder.Length - ExtraSymbols, ExtraSymbols);

return stringBuilder.ToString();

}

private String AlingText(String encodedText)

{

var result = encodedText;

var residue = encodedText.Length % 5;

if (residue != 0)

{

for (var i = 0; i < 5 - residue; i++)

{

result += "0";

}

}

ExtraSymbols = 5 - residue;

return result;

}

private void CreateCodesTabel()

{

var unicode = 'а';

for (var i = 0; i < 32; i++)

{

var code = Convert.ToString(i, 2);

var codeLength = code.Length;

if (codeLength < 5)

{

for (var j = 0; j < 5 - codeLength; j++)

{

code = "0" + code;

}

}

SymbolsTable.Add((char) (unicode + i), code);

CodesTable.Add(code, (char) (unicode + i));

}

}

}

public partial class ApplicationForm : Form

{

private List<Coder> coders;

private List<FrequencyTable> frequencies;

private List<Compressor> compressors;

private int i = 1;

public ApplicationForm()

{

InitializeComponent();

coders = new List<Coder>();

frequencies = new List<FrequencyTable>();

compressors = new List<Compressor>();

}

private void ButtonEncodeClick(object sender, EventArgs e)

{

var filePath = TestFiles.TestFile2;

var fileString = ReadFile(filePath);

textBoxSource.Text = fileString;

var encodedText = fileString;

while (true)

{

var frecuencyTable = CreateFrequencyTable(encodedText);

frequencies.Add(frecuencyTable);

var coder = new Coder();

encodedText = Encode(coder, encodedText, frecuencyTable);

coders.Add(coder);

var newText = CompressText(encodedText);

var newFrequencyTable = CreateFrequencyTable(newText);

var newCoder = new Coder();

var newEncodedText = Encode(newCoder, newText, newFrequencyTable);

if (!IsTextCompressed(newEncodedText, encodedText))

{

if (compressors.Count > 1)

{

compressors.RemoveAt(compressors.Count - 1);

}

break;

}

i++;

encodedText = newText;

}

textBoxEncoded.Text = encodedText;

ShowFrequencyTable(frequencies[0]);

}

private bool IsTextCompressed(String newText, String previousText)

{

if (newText.Length < previousText.Length)

{

return true;

}

return false;

}

private String CompressText(String encodedText)

{

var compressor = new Compressor();

compressors.Add(compressor);

return compressor.GetNewText(encodedText);

}

private void ButtonSendClick(object sender, EventArgs e)

{

if (coders != null)

{

textBoxDecoded.Text = Decode(textBoxEncoded.Text);

}

label2.Text = @"Байт до сжатия: " + textBoxDecoded.Text.Length \* 8;

label3.Text = @"Байт после сжатия: " + textBoxEncoded.Text.Length;

label4.Text = @"Коэффициент сжатия: " + String.Format("{0:##0.00}",

100 - ((textBoxEncoded.Text.Length /

((double)textBoxDecoded.Text.Length \* 8))) \* 100) + @"%";

label5.Text = @"Количество повторных сжатий: " + i;

}

private void ShowFrequencyTable(FrequencyTable table)

{

foreach (var symbol in table.SortedSymbols)

{

textBoxFrequency.Text += String.Concat("'", symbol, "' - ",

table.SybmolsFrequency[symbol]);

textBoxFrequency.Text += String.Concat(" - ", coders[0].CodesTable[symbol],

Environment.NewLine);

}

}

private String Decode(String encodedText)

{

var result = encodedText;

for (var i = coders.Count - 1; i >= 0; i--)

{

result = coders[i].Decode(result);

if (i != 0)

{

result = compressors[i - 1].DecompressText(result);

}

}

return result;

}

private String Encode(Coder coder, String text, FrequencyTable table)

{

return coder.Encode(text, table);

}

private FrequencyTable CreateFrequencyTable(String fileString)

{

var table = new FrequencyTable();

table.CreateFrequencyTable(fileString);

return table;

}

private String ReadFile(String filePath)

{

var helper = new FileHelper();

return helper.ReadFile(filePath);

}

}