|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
| Расчетно-графическое задание | | |
| по дисциплине «Операционные Системы» | | |
| РАЗРАБОТКА ПРОГРАММЫ СЖАТИЯ ДАННЫХ | | |
|  | | |
|  | Бригада 11 | Егупов иван |
|  | Группа ПМ-21 | Порсин данил |
|  | Вариант 19 |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  | Преподаватели | Кобылянский Валерий Григорьевич |
|  |  | Сивак Мария Алексеевна |
| Новосибирск, 2024 | | |

Оглавление

[Введение: 3](#_Toc166521665)

[Теоретическая часть: 3](#_Toc166521666)

[Описание программы: 4](#_Toc166521667)

[Описание процедур, используемых в программе: 4](#_Toc166521668)

[Описание функций, используемых в программе: 5](#_Toc166521669)

[**Интерфейс** 5](#_Toc166521670)

[Тестирование: 7](#_Toc166521671)

[Тестирование сжатия данных 7](#_Toc166521672)

[Тестирование расшифровки данных 7](#_Toc166521673)

[Заключение: 8](#_Toc166521674)

[Список использованных источников: 8](#_Toc166521675)

[**Приложение** 8](#_Toc166521677)

1)Задание:

Разработка программы сжатия данных, использующая алгоритм RLC

2)Введение:

*Компрессия данных* — это процесс сокращения объема данных без потерь или с минимальной потерей информации. Путем применения алгоритмов и методов компрессии, мы можем существенно уменьшить размер данных, сохраняя при этом их суть и основные характеристики. Это может быть полезно во множестве ситуаций, начиная от экономии места на хранилище и ускорения передачи данных до минимизации затрат на интернет-трафик и повышения производительности систем обработки и анализа информации.

Компрессия данных играет ключевую роль в многих областях, включая интернет, мультимедиа, хранение данных и мобильные коммуникации. Она продолжает развиваться с появлением новых технологий и увеличением объема данных.

Главная цель алгоритмов компрессии данных — уменьшить размер данных, сохраняя при этом их информационную значимость. Компрессия данных активно применяется во множестве областей, включая сжатие аудио и видеофайлов, сжатие изображений, архивирование файлов и многое другое.

Компрессия данных может быть ****без потерь**** или ****с потерями****. Компрессия без потерь позволяет восстановить исходные данные точно такими, какими они были до компрессии. Это обычно используется для текстовых и некоторых видов графических данных, где потеря информации недопустима.

В зависимости от обрабатываемого ресурса, разные алгоритмы сжатия и разжатия требуют разный объём вычислительных ресурсов. Например сжатие аудио и видео файлов требует больше ресурсов, чем их восстановление.

3)Теоретическая часть:

Кодирование длин серий(Run Length Coding - RLC) относится к компрессии без потерь, является одним из самых первых и простейших алгоритмов архивации. Он заменяет серии из двух или более одинаковых символов числом, обозначающим длину серии, за которым идёт сам символ.

Коэффициент сжатия - показатель, отражающий степень уменьшения объема данных после применения алгоритма сжатия. Он вычисляется как отношение исходного объема данных к объему данных после сжатия и измеряется в процентах или в коэффициентах.  
Формула для расчета коэффициента сжатия выглядит следующим образом:  
Коэффициент сжатия = Объем исходных данных / Объём данных после сжатия

 𝑘=𝑆𝑜𝑆𝑐

Таким образом, чем выше коэффициент сжатия, тем алгоритм эффективнее. Следует отметить: если *k* = 1, то алгоритм не производит сжатия, то есть выходное сообщение оказывается по объёму равным входному; если *k* < 1, то алгоритм порождает сообщение большего размера, нежели несжатое, то есть, совершает «вредную» работу.

Ситуация с *k* < 1 вполне возможна при сжатии. Принципиально невозможно получить алгоритм сжатия без потерь, который при любых данных образовывал бы на выходе данные меньшей или равной длины. Обоснование этого факта заключается в том, что поскольку число различных сообщений длиной *n* бит составляет ровно 2*n*, число различных сообщений с длиной меньшей или равной *n* (при наличии хотя бы одного сообщения меньшей длины) будет не больше 2*n*. Это значит, что невозможно однозначно сопоставить все исходные сообщения сжатым: либо некоторые исходные сообщения не будут иметь сжатого представления, либо нескольким исходным сообщениям будет соответствовать одно и то же сжатое, а значит их нельзя различить. Однако даже когда алгоритм сжатия увеличивает размер исходных данных, легко добиться того, чтобы их объём гарантировано не мог увеличиться более, чем на 1 бит. Тогда даже в самом худшем случае будет иметь место неравенство:

𝑘⩾𝑆𝑜𝑆𝑜+1  
Делается это следующим образом: если объём сжатых данных меньше объёма исходных, возвращаем сжатые данные, добавив к ним «1», иначе возвращаем исходные данные, добавив к ним «0»).

Коэффициент сжатия может быть как постоянным, так и переменным. Во втором случае он может быть определён либо для каждого конкретного сообщения, либо оценён по некоторым критериям:

* средний (обычно по некоторому тестовому набору данных);
* максимальный (случай наилучшего сжатия);
* минимальный (случай наихудшего сжатия);

или каким-либо другим. Коэффициент сжатия с потерями при этом сильно зависит от допустимой погрешности сжатия или *качества*, которое обычно выступает как параметр алгоритма. В общем случае постоянный коэффициент сжатия способны обеспечить только методы сжатия данных с потерями.

Допустимость потерь

Основным критерием различия между алгоритмами сжатия является описанное выше наличие или отсутствие потерь. В общем случае алгоритмы сжатия без потерь универсальны в том смысле, что их применение безусловно возможно для данных любого типа, в то время как возможность применения сжатия с потерями должна быть обоснована. Для некоторых типов данных искажения не допустимы в принципе. В их числе

* символические данные, изменение которых неминуемо приводит к изменению их семантики: программы и их исходные тексты, двоичные массивы и т. п.;
* жизненно важные данные, изменения в которых могут привести к критическим ошибкам: например, получаемые с медицинской измерительной аппаратуры или контрольных приборов летательных, космических аппаратов и т. п.;
* многократно подвергаемые сжатию и восстановлению промежуточные данные при многоэтапной обработке графических, звуковых и видеоданных.

Несмотря на то, что кодер RLC, как правило, дает очень незначительное [сжатие](https://mf.grsu.by/UchProc/livak/en/po/comprsite/theory_classification_01.html#ratio), он может работать очень быстро. А скорость работы декодера RLC вообще близка к скорости простого копирования блока информации.

4) Описание программы:

Программа реализует алгоритм RLC компрессии для строковых данных. Визуализация сделана при помощи Windows Forms, язык программирования C#. Среда исполнения – ОС Windows. Текст программы находится в приложении.

Описание процедур, используемых в программе:

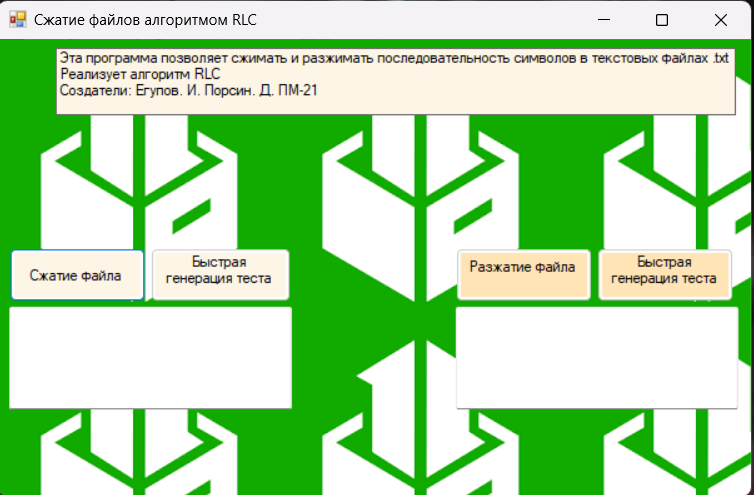
1. private void button1\_Click(object sender, EventArgs e) – Обработка нажатия пользователем кнопки “Сжатие файла”
2. private void BackgroundWorker1\_DoWork(object sender, DoWorkEventArgs e) – Вызов фонового процесса для сжатия данных
3. private void BackgroundWorker1\_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e) – Получение результата процесса сжатия данных с последующей обработкой
4. private void button2\_Click(object sender, EventArgs e) – Обработка нажатия пользователем кнопки “Разжатия файла”
5. private void BackgroundWorker2\_DoWork(object sender, DoWorkEventArgs e) – Аналогично 2), только для разжатия данных
6. private void BackgroundWorker2\_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e) – Аналогично 3) для разжатия
7. private void SaveFile(string content) – формирование и сохранение пользователем файла с результатом
8. private void button3\_Click(object sender, EventArgs e) – обработка пользователем нажатия кнопки “Быстрая генерация теста” для сжатия данных
9. private void button4\_Click(object sender, EventArgs e) – обработка пользователем нажатия кнопки “Быстрая генерация теста” для разжатия данных

Описание функций, используемых в программе:

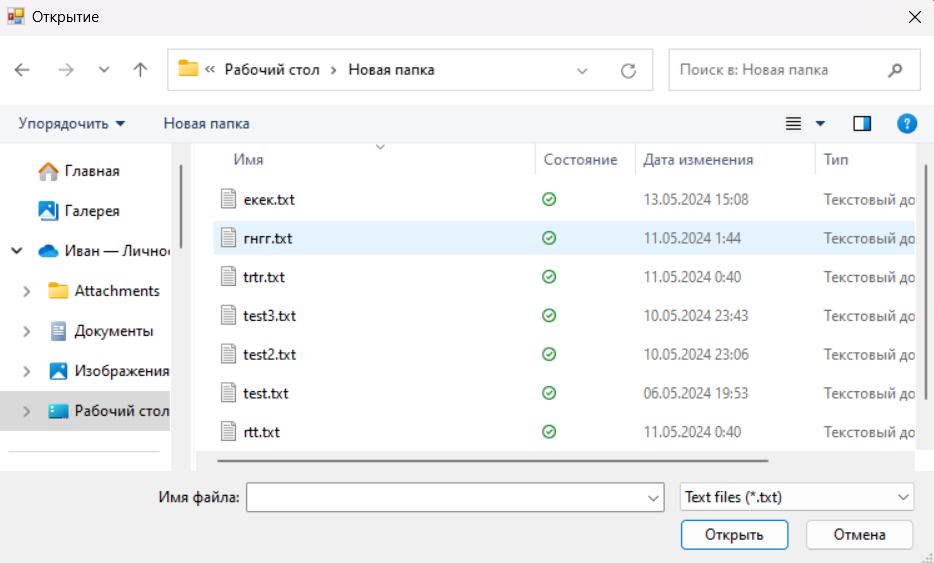
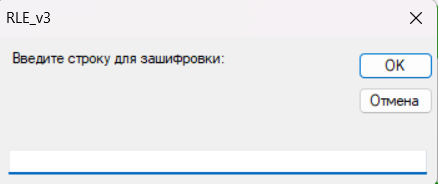
1. static string EncodeRLC(string input) – Компрессия входных данных в виде строки, выходные данные тоже строка.
2. static string DecodeRLC(string encoded) – Декомпрессия входных данных. Вход/выход строка

Эти функции вызываются в соответствующих фоновых процесса BackgroundWorker1\_DoWork и BackgroundWorker2\_DoWork. Результат передаётся в процедуры BackgroundWorker1/2\_RunWorkerCompleted, которые его обрабатывают и демонстрируют пользователю.

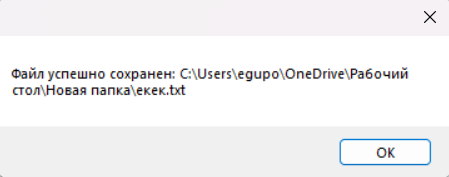
## **Интерфейс**

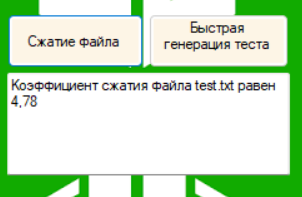


Программа работает в двух режимах: Выбор файла из имеющихся на компьютере пользователя или быстрой генерации теста, где в диалоговом окне пользователь может ввести строку, подлезжащую компрессии/декомпрессии.

Для удобства Тестирования, пользователю, помимо результирующего файла, в соответсующие текстовые блоки записывается результат выбранной опции, с посчитанным коэффициентом сжатия:





5)Тестирование:

Тестирование сжатия данных

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № теста | Входные данные | Размер входных данных (Байт) | Выходные данные | Размер выходных данных (Байт) | Коэффициент сжатия |
| 1 | VVVVVVVVVVVVVVOOOOOOOOOOOOPPPPPPPPPP | 36 | 14V12O10P | 9 | 4.0 |
| 2 | abcderf | 7 | 1a1b1c1d1e1r1f | 14 | 0.5 |
| 3 | 100 подряд идущих букв V | 100 | 100V | 4 | 25 |
| 4 | 10 подряд идущих 8 и 10 подряд идущих 9 | 20 | 108109 | 6 | 3.33 |
| 5 | Uuuu Pppp\*\*\*\*\*\* | 18 | 1U3u4 1P3p6\* | 12 | 1.5 |
| 6 | Последовательность из Большого числа повторяющихся символов разного вида | 5500 | 268v268P14-  10 раз | 110 | 50 |
| 7 | Отрывок из Романа Х. Мураками “В поисках овцы” | 34267 | Зашифрованный роман | 65538 | 0.53 |

Тестирование расшифровки данных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № теста | Входные данные | Размерность Входных данных (Байт | Выходные данные | Размерность Выходных данных |
| 1 | 8B6X4$ | 6 | BBBBBBBBXXXXXX$$$$ | 18 |
| 2 | 800B700A | 8 | 800 символов B 700 символо A | 1500 |
| 3 | 1a1b1c1d1e1f1g | 14 | abcdefg | 7 |
| 4 | Зашифрованный роман | 65538 | Восстановить исходный текст не удалось | 220млн |
| 5 | Отрывок из романа | 1633 | Восстановленный отрывок | 597 |

6) Заключение:

В ходе выполнения расчётно-графической работы были получены теоретические знания по алгоритмам компрессии данных. Получены практические навыки создания простых приложений под ОС Windows.

Алгоритм RLC эффективен при шифровании последовательностей одинаковых символов. Кодирование/декодирование произвольных текстовых данных оказалось малоэффективным. В связи с этим его использование уместно для кодирования изображений, где часто встречаются одинаковые байты, передающие цветовую палитру (красный синий и зелёный).

7) Список использованных источников:

*1)Д. Ватолин, А. Ратушняк, М. Смирнов, В. Юкин.* Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео

# 2)Алгоритмы компрессии данных: принципы и эффективность - <https://habr.com/ru/companies/otus/articles/745628/>

Представитель OTUS [MaxRokatansky](https://habr.com/ru/users/MaxRokatansky/)

3)<https://metanit.com/sharp/windowsforms/2.5.php> - Разработка приложений на Windows Forms

# **8) Приложение**

**Код файла Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace RLE\_v3

{

internal static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

**Исходный код Form1.cs**

using System;

using System.IO;

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 static System.Windows.Forms.VisualStyles.VisualStyleElement;

using Microsoft.VisualBasic;

namespace RLE\_v3

{

public partial class Form1 : Form

{

private BackgroundWorker backgroundWorker;

private string selectedFileName;

public Form1()

{

InitializeComponent();

backgroundWorker1.DoWork += BackgroundWorker1\_DoWork;

backgroundWorker1.RunWorkerCompleted += BackgroundWorker1\_RunWorkerCompleted;

backgroundWorker2.DoWork += BackgroundWorker2\_DoWork;

backgroundWorker2.RunWorkerCompleted += BackgroundWorker2\_RunWorkerCompleted;

button1.Click += button1\_Click;

button2.Click += button2\_Click;

saveFileDialog1.Filter = "Text files(\*.txt)|\*.txt|All files(\*.\*)|\*.\*";

}

private string currentFilePath;

private void button1\_Click(object sender, EventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "Text files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*";

if (openFileDialog.ShowDialog() == DialogResult.OK)

{

try

{

// Сохраняем путь к файлу в член класса

currentFilePath = openFileDialog.FileName;

// Запускаем фоновую операцию после выбора файла

backgroundWorker1.RunWorkerAsync();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при открытии файла: " + ex.Message);

}

}

}

private void BackgroundWorker1\_DoWork(object sender, DoWorkEventArgs e)

{

try

{

string fileContent = File.ReadAllText(currentFilePath);

string encoded = EncodeRLC(fileContent);

e.Result = new Tuple<string, string>(encoded, currentFilePath);

}

catch (Exception ex)

{

e.Result = ex;

}

}

private void BackgroundWorker1\_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)

{

if (e.Error != null)

{

MessageBox.Show("Ошибка: " + e.Error.Message);

}

else if (e.Result is Exception)

{

MessageBox.Show("Ошибка при обработке файла: " + ((Exception)e.Result).Message);

}

else

{

var resultTuple = (Tuple<string, string>)e.Result;

string encoded = resultTuple.Item1;

string filePath = resultTuple.Item2;

// Получение имени файла из пути

string filename = Path.GetFileName(filePath);

// Расчет коэффициента сжатия

double originalLength = File.ReadAllText(filePath).Length;

double encodedLength = encoded.Length;

double compressionRatio = originalLength / encodedLength;

string message = $"Коэффициент сжатия файла {filename} равен {compressionRatio:F2}";

SaveFile(encoded);

textBox1.Text = message;

}

}

private void button2\_Click(object sender, EventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "Text files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*";

if (openFileDialog.ShowDialog() == DialogResult.OK)

{

try

{

string fileName = openFileDialog.FileName;

backgroundWorker2.RunWorkerAsync(fileName);

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при открытии файла: " + ex.Message);

}

}

}

private void BackgroundWorker2\_DoWork(object sender, DoWorkEventArgs e)

{

string fileName = e.Argument as string;

try

{

string encodedContent = File.ReadAllText(fileName);

string decoded = DecodeRLC(encodedContent);

e.Result = decoded;

}

catch (Exception ex)

{

e.Result = ex;

}

}

private void BackgroundWorker2\_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)

{

if (e.Error != null)

{

MessageBox.Show("Ошибка: " + e.Error.Message);

}

else if (e.Result is Exception)

{

MessageBox.Show("Ошибка при обработке файла: " + ((Exception)e.Result).Message);

}

else

{

string decodedContent = e.Result as string;

SaveFile(decodedContent);

textBox2.Text = $"Содержимое зашифрованного файла: {(decodedContent.Length > 200 ? decodedContent.Substring(0, 200) + "..." : decodedContent)}";

}

}

private void SaveFile(string content)

{

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.Filter = "Text files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*";

saveFileDialog.DefaultExt = "txt";

if (saveFileDialog.ShowDialog() == DialogResult.OK)

{

File.WriteAllText(saveFileDialog.FileName, content);

MessageBox.Show("Файл успешно сохранен: " + saveFileDialog.FileName);

}

}

static string EncodeRLC(string input)

{

try

{

if (string.IsNullOrEmpty(input))

{

return "Error: Empty input string.";

}

StringBuilder encoded = new StringBuilder();

char currentChar = input[0];

int count = 1;

for (int i = 1; i < input.Length; i++)

{

if (input[i] == currentChar)

{

count++;

}

else

{

encoded.Append(count).Append(currentChar);

currentChar = input[i];

count = 1;

}

}

encoded.Append(count).Append(currentChar);

return encoded.ToString();

}

catch (Exception ex)

{

return "Error: " + ex.Message;

}

}

static string DecodeRLC(string encoded)

{

try

{

if (string.IsNullOrEmpty(encoded))

{

return "Error: Empty input string.";

}

StringBuilder decoded = new StringBuilder();

int i = 0;

while (i < encoded.Length)

{

int count = 0;

while (char.IsDigit(encoded[i]))

{

count = count \* 10 + (encoded[i] - '0');

i++;

}

char character = encoded[i++];

decoded.Append(character, count);

}

return decoded.ToString();

}

catch (Exception ex)

{

return "Error: " + ex.Message;

}

}

private void button3\_Click(object sender, EventArgs e)

{

string Content = Interaction.InputBox("Введите строку для зашифровки: ");

if (Content != " ")

{

double originalLength = Content.Length;

string encoded = EncodeRLC(Content);

double encodedLength = encoded.Length;

double compressionRatio = originalLength / encodedLength;

string res = $"Коэффициент сжатия строки равен {compressionRatio:F2}\n Содержимое: {encoded}";

textBox1.Text = res;

}

else

{

MessageBox.Show("Пустая строка: ");

}

}

private void button4\_Click(object sender, EventArgs e)

{

string Content = Interaction.InputBox("Введите строку для расшифровки в формате количество символов/символ. Пример 89B: ");

if (Content != " ")

{

string decoded = DecodeRLC(Content);

textBox2.Text = $"Содержимое зашифрованной строки: {(decoded.Length > 200 ? decoded.Substring(0, 200) + "..." : decoded)}";

}

else

{

MessageBox.Show("Пустая строка: ");

}

}

}

}