Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Языки программирования (ЯП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

ИССЛЕДОВАНИЕ АЛГОРИТМОВ СЖАТИЯ И РАСПАКОВКИ RLE И LZ77

БГУИР КР I-40 01 01 131 ПЗ

Студент: гр.351006 Шульга Е.С.

Руководитель: Марина И.М.

Минск 2014

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc405907032)

[1 ПОСТАНОВКА ЗАДАЧИ 6](#_Toc405907033)

[2 АНАЛИЗ ИСТОЧНИКОВ 7](#_Toc405907034)

[2.1 Алгоритм Run Length Encoding 7](#_Toc405907035)

[2.2 Алгоритм Ziv-Lempel 1977 7](#_Toc405907036)

[2.3 Набор файлов Canterbury Corpus 9](#_Toc405907037)

[3 разработка программного приложения 11](#_Toc405907038)

[3.1 Выбор структур данных 11](#_Toc405907039)

[3.2 Модели данных алгоритмов 15](#_Toc405907040)

[3.3 Реализация алгоритмов 16](#_Toc405907041)

[3.4 Разработка экранных форм 18](#_Toc405907042)

# ВВЕДЕНИЕ

Сжатие данных – алгоритмическое преобразование данных, производимое с целью уменьшения занимаемого ими объема. Применяется для более рационального использования устройств хранения и передачи данных. Сжатие основано на устранении избыточности, содержащейся в исходных данных.

Все методы сжатия данных делятся на два основных типа: сжатие без потерь, сжатие с потерями. Метод сжатия с потерями – метод сжатия данных, при использовании которого распакованные данные отличаются от исходных, но степень отличия не является существенной с точки зрения их дальнейшего использования. Этот тип сжатия применяется в основном для сжатия аудио- и видеоданных.

Также широко применяется метод сжатия данных без потерь – такой метод, при использовании которого закодированные данные могут мыть восстановлены с точность до бита. Этот метод используется во всех файловых архиваторах. [1]

В данном курсовом проекте рассматриваются базовые алгоритмы сжатия-распаковки: RLE (Run-Length Encoding, кодирование длин серий) и LZ77 (назван по именам ученых, разработавших этот алгоритм: Якова Зива и Абрахама Лемпеля; был опубликован в 1977 г [2]), – которые относятся к методам сжатия без потерь. RLE – алгоритм, положивший начало многим другим алгоритмам, например, LZ77, который в свою очередь послужил основой для целого семейства современных эффективных алгоритмов.

# ПОСТАНОВКА ЗАДАЧИ

В рамках курсового проекта необходимо изучить методы сжатия и распаковки RLE и LZ77, разработать и реализовать в программном приложении соответствующие алгоритмы.

Спроектировать и реализовать графический интерфейс программного приложения, позволяющий осуществлять выбор любого файла с диска для обработки, выбор алгоритма обработки при сжатии, выбор каталога на диске для сохранения обработанного файла; при распаковке выбор применяемого алгоритма должен осуществляться автоматически.

Программное приложение также должно служить вспомогательным средством при проведении исследования алгоритмов, поэтому необходимо реализовать сбор информации, которая бы сохранялась в автоматически создаваемом файле.

Необходимо провести исследование алгоритмов сжатия распаковки по следующим параметрам:

* коэффициент сжатия;
* время, затраченное на сжатие;
* время, затраченное на распаковку.

Сделать вывод об эффективности применения исследуемых реализованных алгоритмах на различных типах файлов.

Для закрепления знаний, полученных на учебных занятиях в рамках курсов ЯП и ТИ, языком программирования был выбран язык C#.

# АНАЛИЗ ИСТОЧНИКОВ

## Алгоритм Run Length Encoding

Алгоритм Run Length Encoding (кодирование длин серий[[1]](#footnote-1)1)) является одним из самых старых и самых простых методов сжатия [1]. Суть этого метода заключается в следующем: если в сжимаемых данных последовательно друг за другом m раз встречается элемент d, то результирующий поток данных будет записана пара md. Серией элемента d будет называться m последовательных вхождений этого символа во входные данные [2].

Пример использования [3]:

Рассмотрим изображение, содержащее простой черный текст на сплошном белом фоне. На этом изображении будет много серий белых пикселей в пустых местах и много коротких серий черных пикселей в тексте. В качестве примера приведена некая произвольная строка изображения в черно-белом варианте. Здесь W представляет собой белый пиксель, а B – черный пиксель:

WWWWWWWWWWWWBWWWWWWWWWWWWWWBBBWWWWWWWBWWW

При применении алгоритма кодирования получается следующее:

11W0B13W2B6W0B2W

Можно увидеть, что после применения алгоритма длина строки изображения уменьшилась с 41 до 14 элементов.

Очевидно, что этот алгоритм будет эффективен только в тех случаях, если во входных данных содержится большое число серий. Примером таких данных может служить деловая графика – растровые изображения с большими областями повторяющегося цвета. Ситуация, когда выходной файл увеличивается (то есть коэффициент сжатия оказывается больше единицы) не так уж и редка: ее можно получить, например, применяя алгоритм к обработанным цветным фотографиям [1].

## Алгоритм Ziv-Lempel 1977

В процессе обработки данных часто могут встречаться последовательности, обладающие определенными повторяющимися свойствами. На основании этих свойств можно осуществить выбор метода кодирования, который будет наиболее эффективен в данном случае.

Но, когда априори неизвестны характеристики входных данных, а проведение статистических исследований по отношению к ним нерационально или невозможно, проблема сжатия значительно усложняется. Для решения этих трудностей Якобом Зивом и Абрахамом Лемпелем была предложена идея сопровождать процесс кодирования сбором информации о характеристиках входных данных. Такие методы будут одинаково производительны для различных типов входных данных [2].

Алгоритм LZ77 является родоначальником целого семейства словарных схем – так называемых алгоритмов со скользящим словарем, или скользящим окном. Действительно, в LZ77 в качестве словаря используется блок уже обработанной последовательности. По мере выполнения сжатия положение этого блока постоянно меняется, «скользит» по входному потоку данных.

Скользящее окно имеет длину N, то есть в него помещается N символов, и состоит из двух частей:

* буфер предварительного просмотра длины n;
* последовательность длины L=N-n уже обработанных символов.

Идея алгоритма заключается в поиске самого длинного совпадения между строкой, начинающейся с первого символа буфера, и фразами окна. Эти фразы могут начинаться с любого символа окна и выходить за пределы словаря, вторгаясь в область буфера, но должны начинаться в окне. Длина совпадения не должна превышать размер буфера [3].

Стоит отметить, что обычно n намного меньше L, так как вероятность нахождения длинного совпадения в буфере и словаре крайне мала.

Полученная в результате поиска фраза кодируется с помощью двух чисел: смещения от начала буфера и длины совпадения. Смещение и длина соответствия играют роль указателя (ссылки), однозначно определяющего фразу.

Дополнительно в выходной поток записывается символ s, непосредственно следующий за совпавшей строкой буфера. Таким образом, на каждом шаге алгоритм выдает описание трех объектов:

* смещение в словаре относительно начала буфера, o;
* длина подстроки, l;
* первый символ в буфере, следующий за подстрокой, s.

Затем окно смещается на l+1 символов вправо. Величина сдвига объясняется тем, что закодировано l символов с помощью ссылки на фразу в словаре и один символ с помощью обычного копирования. Передача одного символа в явном виде позволяет разрешить проблему обработки еще ни разу не виденных символов. Тем не менее это порождает проблему существенного увеличения размера сжатого блока.

Пример использования алгоритма:

Возьмем следующую строку:

compression and decompression and compression and decompression

Пусть размер словаря – 1024 символа, размер буфера – 64 символа, то есть вначале исходная строка полностью помещается в буфер, в конце обработки строка полностью окажется в словаре.

Для наглядности покажем те последовательности буфера, которые выделит алгоритм:

compre[s]si[o]n a[n]d[ ]d[e]c[ompression and ]c[ompression and decompressio]n

Далее каждое из этих совпадений будет заменено на комбинацию [смещение, длина, следующий символ]:

compre[1,1,s]i[8,1,n] a[3,1,d][4,1,d][12,1,c][18,15,c][34,25,n]

Можно увидеть, что количество элементов в выходной последовательности – 30, в отличии от количества элементов во входной последовательности – 63. Коэффициент сжатия составил 0.47.

Однако у алгоритма есть и недостаток: способ формирования кодов сравнительно неэффективен и позволяет сжимать только сравнительно длинные последовательности.

Также важной особенностью LZ77 является сильная несимметричность по времени – кодирование значительно медленнее декодирования, поскольку при компрессии значительное количество времени тратится на поиск совпадающих последовательностей[[2]](#footnote-2)1) [1].

## Набор файлов Canterbury Corpus

Решение задачи сравнения алгоритмов по достигаемой ими степени сжатия требует введения некоторого критерия, так как нельзя сравнивать производительность реализаций на каком-то абстрактном файле. Следует осторожно относиться к теоретическим оценкам, так как они вычисляются с точностью до констант. Величины этих констант на практике могут колебаться в очень больших пределах, особенно при сжатии небольших файлов.

В 1997 году группой исследователей был предложен набор файлов, специально отобранных, чтобы служить в качестве эталона при проведении исследований алгоритмов сжатия. Этот набор был назван Canterbury Corpus (информационный фонд Кентербери). Отбор файлов осуществлялся на основании того, что результаты их обработки подтверждали теоретические исследования алгоритмов. Это давало надежду, что результаты обработки этих файлов новыми алгоритмами, которые будут изобретены в будущем, будут также достоверными [6]. Описание файлов, входящие в состав Canterbury Corpus, представлено в таблице 1.

Таблица 1 – Файлы, входящие в состав Canterbury Corpus [6]

|  |  |  |  |
| --- | --- | --- | --- |
| Имя файла | Описание | Условное обозначение | Размер, байт |
| alice29.txt | Текст на английском языке («Алиса в стране чудес») | text | 152 089 |
| asyoulik.txt | Пьеса на английском языке («As you like it») | play | 125 179 |
| cp.html | Документ HTML | html | 24 603 |
| fields.c | Код программы на языке C | csrc | 11 150 |
| grammar.lsp | Код программы на языке LISP | list | 3 721 |
| kennedy.xls | Электронная таблица | excl | 1 029 744 |
| lcet10.txt | Технический документ | tech | 426 754 |
| plrabn12.txt | Стихотворение на английском языке («Paradise Lost») | poem | 481 861 |
| ptt5 | Факс-изображение | fax | 513 216 |
| sum | Исполнимый файл SPARC | sprc | 38 240 |
| xargs.1 | Руководство GNU | man | 4 227 |

При проведении исследования алгоритмов будем использовать именно этот набор файлов.

# разработка программного приложения

При разработке приложения необходимо было решить несколько проблем:

* выбрать структуры данных;
* разработать модель данных, позволяющая оптимально сохранять блоки сжатой информации при использовании алгоритма LZ77;
* реализовать алгоритмы сжатия и распаковки.

## Выбор структур данных

**3.1.1** Для измерения характеристик алгоритмов был создан класс

MetricsOfAlgorythm. Он является универсальным для всех исследуемых алгоритмов. Данный класс реализует подсчет времени, затраченное на выполнение алгоритма и представляет размер выходного для алгоритмов файла (при выполнении алгоритмов распаковки значение данного поля должно соответствовать размеру файла до сжатия и может использоваться для проверки правильности работы). Экземпляры данного класса

Структуры данных, использованные в данном классе, представлены в таблице 2.

Таблица 2 – Структуры данных класса MetricsOfAlgorytms

|  |  |  |
| --- | --- | --- |
| Элемент данных | Использованный тип | Назначение |
| SizeAfter | long | Размер выходного для алгоритма файла |
| ElapsedTime | long | Время, затраченное на выполнение алгоритма |
| timer | Stopwatch | Секундомер, использованный для измерения времени |

**3.1.2** Для вывода информации о проведенной операции сжатия или распаковки, а также для сохранения записей статистики при проведении исследования алгоритмов будем использовать класс Metrics. Структуры данных, использованные в данном классе, представлены в таблице 3.

Таблица 3 – Структуры данных класса Metrics

|  |  |  |
| --- | --- | --- |
| Элемент данных | Используемый тип | Назначение |
| FileName | string | Имя файла |
| FileType | string | Тип файла, предоставляемый ОС |
| FilePath | string | Абсолютный путь к файлу |
| DateOfAnalysis | DateTime | Дата и время начала обработки |
| Size | long | Размер файла до обработки |
| RLESize | long | Размер файла после сжатия алгоритмом RLE |
| LZ77Size | long | Размер файла после сжатия алгоритмом LZ77 |
| RLERatio | float | Коэффициент сжатия алгоритма RLE |
| LZ77Ratio | float | Коэффициент сжатия алгоритма LZ77 |
| RLEC\_ms | long | Время, затраченное на сжатие алгоритмом RLE |
| LZ77C\_ms | long | Время, затраченное на сжатие алгоритмом LZ77 |
| RLED\_ms | long | Время, затраченное на распаковку алгоритмом RLE |
| LZ77D\_ms | long | Время, затраченное на распаковку алгоритмом LZ77 |

Все поля экземпляра данного класса будут использоваться только при исследовании алгоритмов, когда для выполнения вызываются все алгоритмы. При выборе режима сжатия или распаковки одним из алгоритмов будут использоваться только некоторые поля для последующего вывода информации на экран.

**3.1.3** При выполнении алгоритмов может создаться впечатление, что интерфейс пользователя не отвечает на запросы. Для решения этой проблемы можно использовать класс BackgroundWorker, который позволяет запускать операции, требующие большого количества времени в отдельном потоке. Поскольку при завершении обработки данным классом можно возвращать только один объект [8], то упакуем возвращемые данные в класс MetricsOfAlgorythm, описанный ранее в пункте 3.1.2. Затем эти данные будут передаваться экземпляру класса Metrics для дальнейшей обработки.

**3.1.4** Для удобства хранения путей входного и выходного файлов был создан класс IOFilePaths, позволяющий их в одном объекте. В программе объявлена одна переменная данного типа: filePaths.

Структуры данных данного алгоритма представлены в таблице 4.

Таблица 4 – Структуры данных класса IOFilePaths

|  |  |  |
| --- | --- | --- |
| Элемент данных | Использованный тип | Назначение |
| input, output | string | Пути к входному и выходному файлу |

**3.1.5** Как уже было сказано ранее, объектом для обработки для алгоритма LZ77 является скользящее окно. Будем далее называть части скользящего окна буфером и словарем.

Также при реализации алгоритма LZ77 будем использовать в качестве указателя на совпадающую последовательность в словаре пару значений индекс от начала словаря (считая от нуля) и длину совпадения. Далее будем называть эту пару значений ссылкой.

Структуры данных, использованные при реализации алгоритма сжатия LZ77 представлены в таблице 5.

Таблица 5 – Структуры данных алгоритма сжатия LZ77

|  |  |  |
| --- | --- | --- |
| Элемент данных | Использованный тип | Назначение |
| filePaths | IOFilePaths | Пути к входному и выходному файлам |
| LZ77EncodindMetrics | MetricsOfAlgorythm | Сведения об операции сжатия |
| inputFile, outputFile | BinaryReader | Входной и выходной файлы |
| slidingWindow | List<byte> | Скользящее окно |
| bufferSize, dictionarySize | int | Переменные для явного хранения размеров частей скользящего окна |
| bufferStart | int | Индекс в массиве скользящего окна, с которого начинается буфер |
| bufferCounter, dictionaryCounter | int | Счетчики по буферу и словарю |
| equalLength, dictionaryEqualStart | int | Индекс найденных совпадающих подпоследовательностей буфера и словаря |
| maxEqualLength, maxDictionaryEqualStart | int | Индекс найденных совпадающих подпоследовательностей буфера и словаря максимального размера |
| i | int | Счетчик |

**3.1.6** Структуры данных алгоритма распаковки LZ77 представлены в таблице 6.

Таблица 6 - Структуры данных алгоритма распаковки LZ77

|  |  |  |
| --- | --- | --- |
| Элемент данных | Использованный тип | Назначение |
| filePaths | IOFilePaths | Пути к входному и выходному файлам |
| LZ77DecodingMetrics | MetricsOfAlgorythm | Сведения об операции распаковки |
| inputFile, outputFile | BinaryReader | Входной и выходной файлы |
| slidingWindow | List<byte> | Скользящее окно |
| dictionarySize | int | Переменная для явного хранения размера словаря |
| equalLength, dictionaryEqualStart | int | Индекс найденных совпадающих подпоследовательностей буфера и словаря |
| nextSymbol | byte | Следующий за совпадающей последовательностью символ |
| i | int | Счетчик |

**3.1.7** Структуры данных алгоритма сжатия RLE представлены в таблице 7.

Таблица 7 - Структуры данных алгоритма сжатия RLE

|  |  |  |
| --- | --- | --- |
| Элемент данных | Использованный тип | Назначение |
| filePaths | IOFilePaths | Пути к входному и выходному файлам |
| RLECompressionMetrics | MetricsOfAlgorythm | Сведения об операции сжатия |
| inputFile, outputFile | BinaryReader | Входной и выходной файлы |
| currentByte | byte | Текущий обрабатываемый байт входного файла |
| nextByte | byte | Следующий за обрабатываемым байтом входного файла |
| currentLength | int | Длина текущей серии |
| i | int | Счетчик |

**3.1.8** Структуры данных алгоритма сжатия RLE представлены в таблице 8.

Таблица 8 - Структуры данных алгоритма распаковки RLE

|  |  |  |
| --- | --- | --- |
| Элемент данных | Использованный тип | Назначение |
| filePaths | IOFilePaths | Пути к входному и выходному файлам |
| RLEDecompressionMetrics | MetricsOfAlgorythm | Сведения об операции сжатия |
| inputFile, outputFile | BinaryReader | Входной и выходной файлы |
| currentByte | byte | Текущий обрабатываемый байт входного файла |
| currentLength | int | Длина текущей серии |
| i | int | Счетчик |

## Модели данных алгоритмов

**3.2.1** По алгоритму RLE, серии, содержащиеся во входных данных, заменяются на пары: [длина серии, элемент серии]. Выберем элементарный размер блоков обрабатываемых файлов – байт. Исходя из этого появляется ограничение на длину серии в 255 байт – именно столько может сохранено с использованием выбранного типа. Серии большей длины будут автоматически разбиты на меньшие серии.

**3.2.2** В алгоритме LZ77 также выберем элементарный размер блоков сжимаемых файлов – байт.

По данному алгоритму в выходной файл записывается тройка значений: [индекс начала подпоследовательности, длина подпоследовательности, следующий символ]. Пусть следующий символ записывается в выходной файл с использованием размера один байт. А для сохранения в выходном файле ссылки выберем размер – два байта (16 бит). В пределах этих 16 бит можно по-разному выделить место для кодирования индекса начала и длины совпадающей подпоследовательности. Поскольку значение индекса начала подпоследовательности не может быть больше размера словаря, а размер этой подпоследовательности не может быть больше размера буфера, то появляется ограничение на размеры словаря и буфера.

Пусть a, b – количество бит, выделенное для хранения индекса начала и длины совпадающей подпоследовательности. Тогда должно выполняться равенство:

. (1)

Следовательно, размер словаря , а размер буфера .

Выбор различных пар чисел a и b, удовлетворяющих равенству (1) будет влиять на скорость сжатия (так как чем больше размер словаря, тем дольше будет осуществляться поиск совпадающей подстроки) и на качество сжатия (так как существует вероятность появления совпадающей подстроки, размер которой будет больше размера буфера; в этом случае она будет разделена на несколько частей и качество обработки снизится). Пусть тогда . Тогда размер словаря , а размер буфера

При реализации алгоритмов сжатия и распаковки LZ77 для целей оптимального сохранения данных значений были выбраны структуры данных, представленные в таблице 9.

Таблица 9 – Структуры данных, использованные при сохранении ссылки в   
 выходной файл

|  |  |  |
| --- | --- | --- |
| Элемент данных | Использованный тип | Назначение |
| substringIndexBits = 10 | int | Количество бит, отведенное для индекса начала подпоследовательности |
| substringLengthBits = 6 | int | Количество бит, отведенное для размера подпоследовательности |
| maxDictionarySize = 1024 | int | Максимальный размер словаря |
| maxBufferSize = 64 | int | Максимальный размер буфера |

Следующий код позволяет записать в выходной файл ссылку на подпоследовательность, представленную в переменных maxDictionaryEqualStart и maxEqualLength:

outputFile.Write((byte)((maxDictionaryEqualStart >> substringIndexBits) - 8);

outputFile.Write((byte)((maxDictionaryEqualStart << substringLengthBits) + maxEqualLength));

Такая модель данных была разработана, чтобы наиболее эффективно хранить ссылку на подпоследовательность.

## Реализация алгоритмов

Исходные коды разработанных на основе алгоритмов подпрограмм приведены в приложении А.

**3.3.1** Алгоритм сжатия RLE отличается интуитивностью и простотой, однако большим недостатком является его неэффективность.

Вначале происходит считывание из входного файла первого символа. Затем циклически производятся следующие действия:

* Считывается следующий символ.
* Если этот символ равен ранее считанному, то циклически считываются следующие символы до тех пор, пока не будет найден отличающийся символ (то есть не прервется серия). При этом производится подсчет длины этой серии.
* В выходной файл записывается длина серии и символ, который ее составляет.

Схема алгоритма приведена на рисунках В.1 – В.2.

**3.3.2** Суть алгоритма распаковки RLE заключается в циклическом считывании очередных длины серии и символа и, на основании этих данных, восстановлении серии.

Схема алгоритма приведена на рисунке В.3.

**3.3.3** Как было сказано выше, при реализации скользящего окна в алгоритмах сжатия и распаковки LZ77 будем использовать структуру динамический массив List<byte>. В начале обработки происходит инициализация буфера (заполнение из входного файла). Затем циклически, пока не будет обработан весь входной файл, происходят следующие действия:

* Поиск самого большого по количеству элементов совпадения последовательности, содержащейся в начале буфера, и какой-либо последовательности, содержащейся в словаре.
* Запись в выходной файл индекса начала и длины найденной подпоследовательности. Если совпадение не было найдено, то записываются нулевые значения.
* Запись в файл следующего за совпадающей последовательностью символа.
* Модификация словаря и буфера: обработанная последовательность сдвигается из буфера в словарь; если достигнут максимальный размер окна, то удаляются символы из его начала; восстанавливается буфер.

Схема данного алгоритма приведена на графическом материале ГУИР.351001-01 СА.

**3.3.4** Алгоритм распаковки LZ77 проще и быстрее, чем алгоритм сжатия, так как не производится поиск совпадающей подстроки и не происходит заполнения буфера в начале обработки.

В процессе декомпрессии циклически, пока не будет достигнут конец файла, производятся следующие действия:

* Считывается и восстанавливается индекс начала и длина совпадающей последовательности.
* Из словаря по найденной ссылке копируется в буфер количество символов, равное найденной длине.
* При достижении словарем своего максимального размера лишние символы записываются в выходной файл.

В конце происходит запись из окна оставшихся там элементов.

Схема данного алгоритма приведена на рисунках В.4 – В.7.

# Руководство по использованию приложения

При запуске приложения отображается начальное окно, приведенное на рисунке 1.

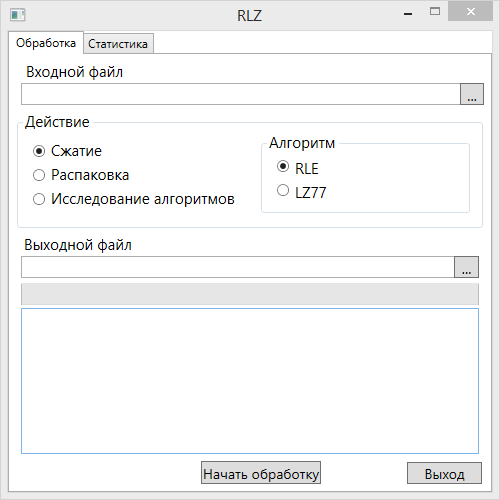


Рисунок 1 – Начальное окно приложения

С помощью данного программного средства можно выполнять следующие действия:

* выполнять сжатие любого файла одним из алгоритмов;
* выполнять распаковку ранее сжатого файла;
* проводить исследование алгоритмов на каком-либо файле;
* просматривать собранную статистику по исследованиям.

## Сжатие файлов

Для начала необходимо выбрать режим «Действие» – «Сжатие».

Затем необходимо выбрать, каким алгоритмом будет осуществляться сжатие входного файла: «RLE» или «LZ77».

Затем необходимо выбрать входной файл на диске. Для этого нужно нажать кнопку «…», расположенную справа от поля «Входной файл». При этом открывается окно, представленное на рисунке 2.

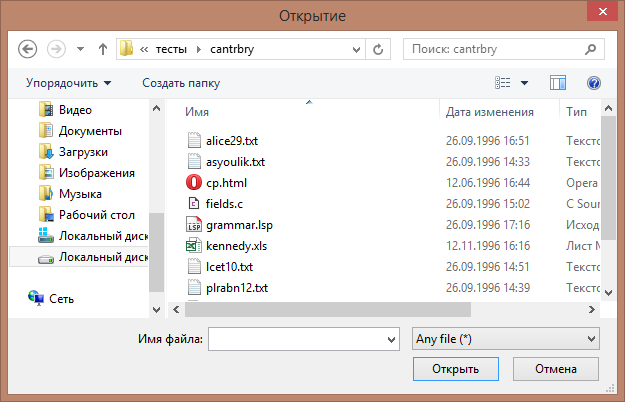


Рисунок 2 – Окно открытия входного файла

При этом автоматически заполняется поле пути выходного файла: предлагается сохранить сжатый файл в той же директории (но с добавленным расширением «.rlz»). Однако пользователь может сохранить выходной файл в любое другое место на диске. Для этого необходимо нажать кнопку «…», расположенную справа от поля пути выходного файла. При этом открывается окно, представленное на рисунке 3.

Затем пользователь, нажав клавишу «Enter» на клавиатуре или кнопку «Начать обработку» на форме, может запустить обработку входного файла; если входной или выходной файл не были выбраны, то появится соответствующее сообщения и обработка запущена не будет. При этом пользователь увидит состояние окна, подобное тому, что представлено на рисунке 4. Процесс обработки визуализируется с помощью индикатора выполнения, постепенно заполняющегося по мере обработки.

Пользователь может остановить обработку. Для этого необходимо нажать кнопку «Остановить». Состояние окна при принудительной остановке обработки приведено на рисунке 5.

Если же сжатие будет завершена успешно, то будут выведены характеристики процесса сжатия, и пользователь увидит состояние окна, подобное до того, которое приведено на рисунке 6.

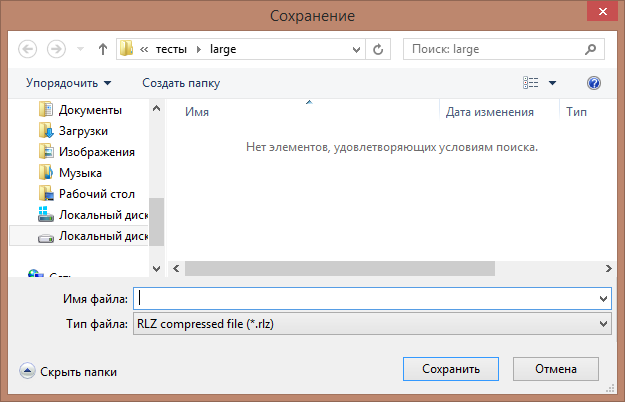


Рисунок 3 – Окно сохранения файла

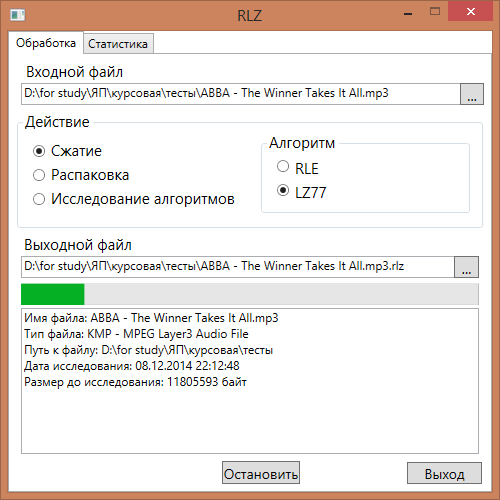


Рисунок 4 – Состояние окна после начала обработки

|  |  |
| --- | --- |
| Рисунок 5 – Состояние окна при принудительной  остановке обработки | Рисунок 6 – Состояние окна при успешном завершении сжатия |

1. 1) Серия – это набор идущих подряд одинаковых элементов последовательности [↑](#footnote-ref-1)
2. 1) Данный факт будет показан при исследовании алгоритмов в разделе N [↑](#footnote-ref-2)