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 КП I-40 01 01 628 ПЗ

Студент: гр.351006 Шульга Е.С.

Руководитель: Марина И.М.

Минск 2014

СОДЕРЖАНИЕ

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.

1 ПОСТАНОВКА ЗАДАЧИ

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

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#, для разработки оконного приложения была выбрана технология WPF.

2 АНАЛИЗ ИСТОЧНИКОВ

2.1 Алгоритм Run Length Encoding

Алгоритм Run Length Encoding (кодирование длин серий[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:

WWWWWWWWWWWWBWWWWWWWWWWWWWWWBBBWWW
WWWWBWWW

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].

2.2 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.

[1)] Серия – это набор идущих подряд одинаковых элементов последовательности

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:

compres[s]i[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]:

compres[1,1,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[1)] [1].

2.3 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.

[1)] Данный факт будет показан при исследовании алгоритмов в разделе 6

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.

3 РАЗРАБОТКА ПРОГРАММНОГО ПРИЛОЖЕНИЯ

При разработке приложения необходимо было решить несколько проблем:

– разработать структуры данных;

– разработать модель данных, позволяющая оптимально сохранять блоки сжатой информации при использовании алгоритма LZ77;

– реализовать алгоритмы сжатия и распаковки.

3.1 Разработка 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.

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.

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

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 [8]. Поскольку при завершении обработки данным классом можно возвращать только один объект, 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 передавать 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

Продолжение таблицы 6

Элемент данных	Использованный тип	Назначение
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

Продолжение таблицы 8

Элемент данных	Использованный тип	Назначение
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

3.2 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:

$$a + b = 16. \quad (1)$$

Следовательно, размер словаря $L = 2^a - 1$, а размер буфера $n = 2^b - 1$.

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 $a = 10, b = 6$. Тогда размер словаря $L = 2^{10} = 1024$, а размер буфера $n = 2^6 = 64$.

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));
```

При распаковке ссылка на подпоследовательность считывается с помощью следующего кода:

```
dictionaryEqualStart = inputFile.ReadByte();
equalLength = inputFile.ReadByte();
dictionaryEqualStart = (dictionaryEqualStart << (substringIndexBits - 8)) +
                       (equalLength >> substringLengthBits);
equalLength = equalLength & (maxBufferSize - 1);
```

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

3.3 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 Б.1 – Б.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 Б.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.351006-01 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 Б.4 – Б.5.

3.3.5 Выбор алгоритма для распаковки сжатого файла можно осуществить на основании информации о том, каким алгоритмом был сжат файл. Будем сохранять данную информацию в сжимаемом файле в качестве первого байта файла: запишем 0, если при сжатии используется алгоритм RLE, и 1 – если используется алгоритм LZ77. При распаковке, считав первый байт файла, можно установить, каким алгоритмом он был сжат.

4 РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ ПРИЛОЖЕНИЯ

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

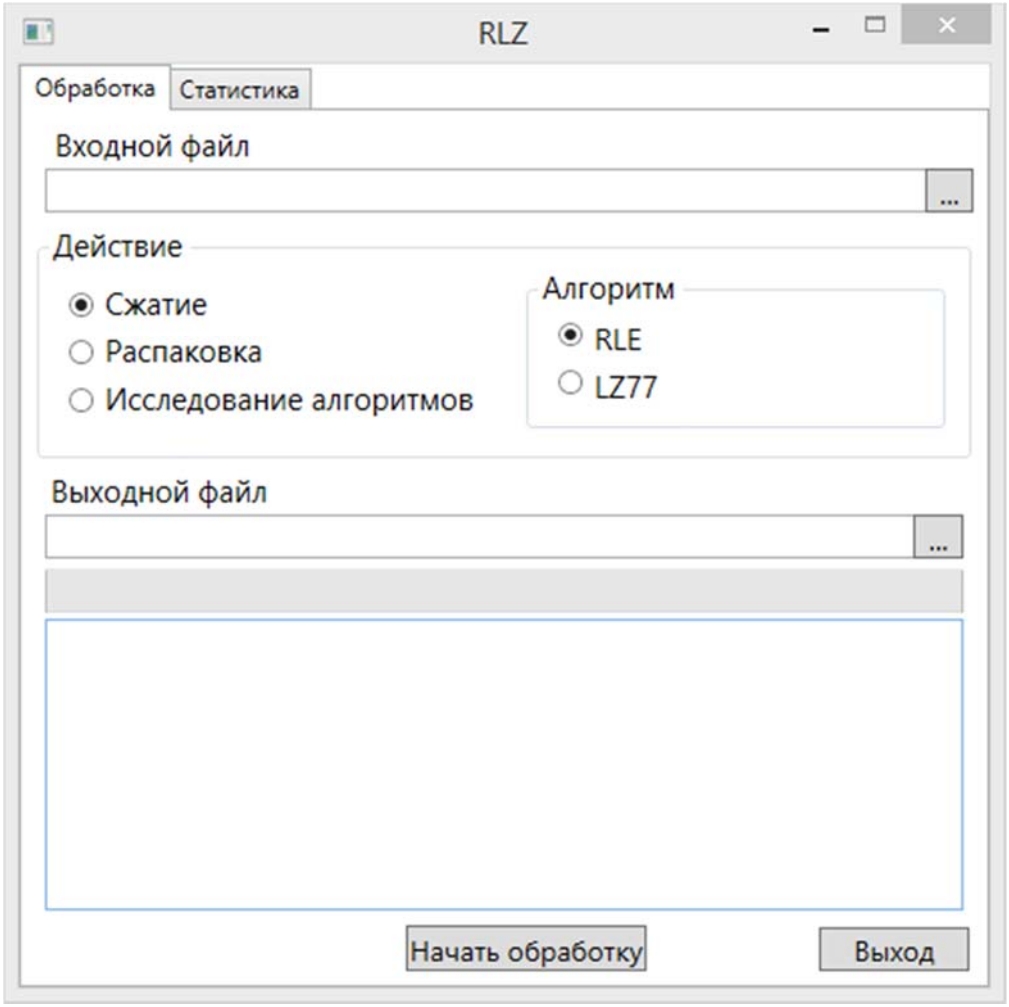

Рисунок 1 – Начальное окно приложения

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.

4.1 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.

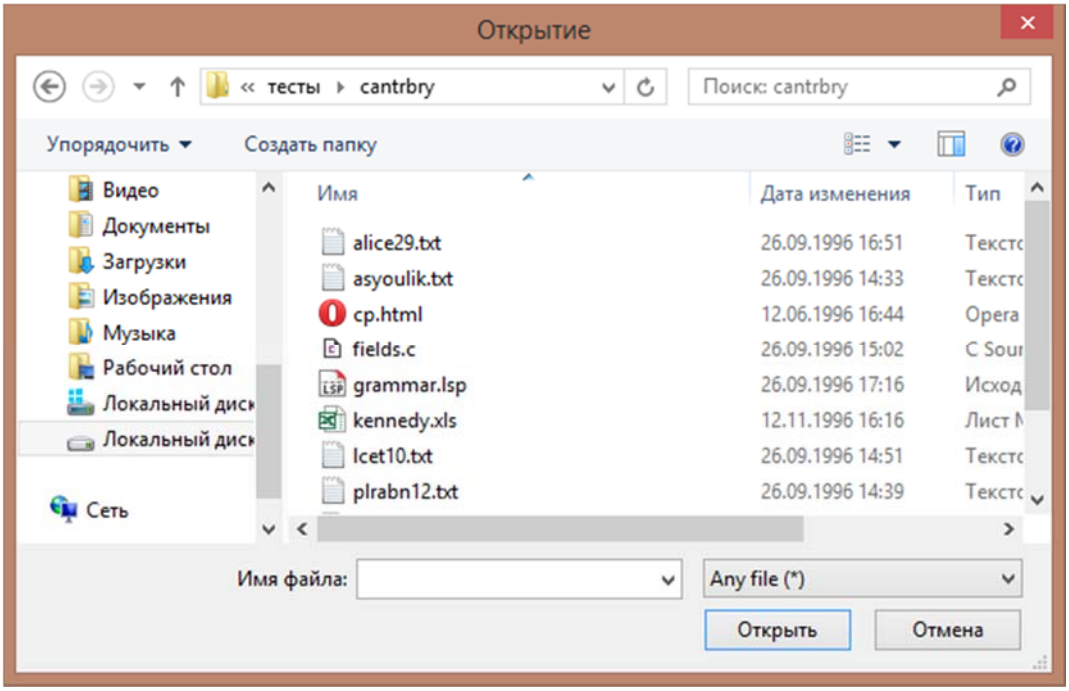

Рисунок 2 – Окно открытия файла при сжатии

При этом автоматически заполняются поля путей входного и выходного файла, причем 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, что приведено на рисунке 6.

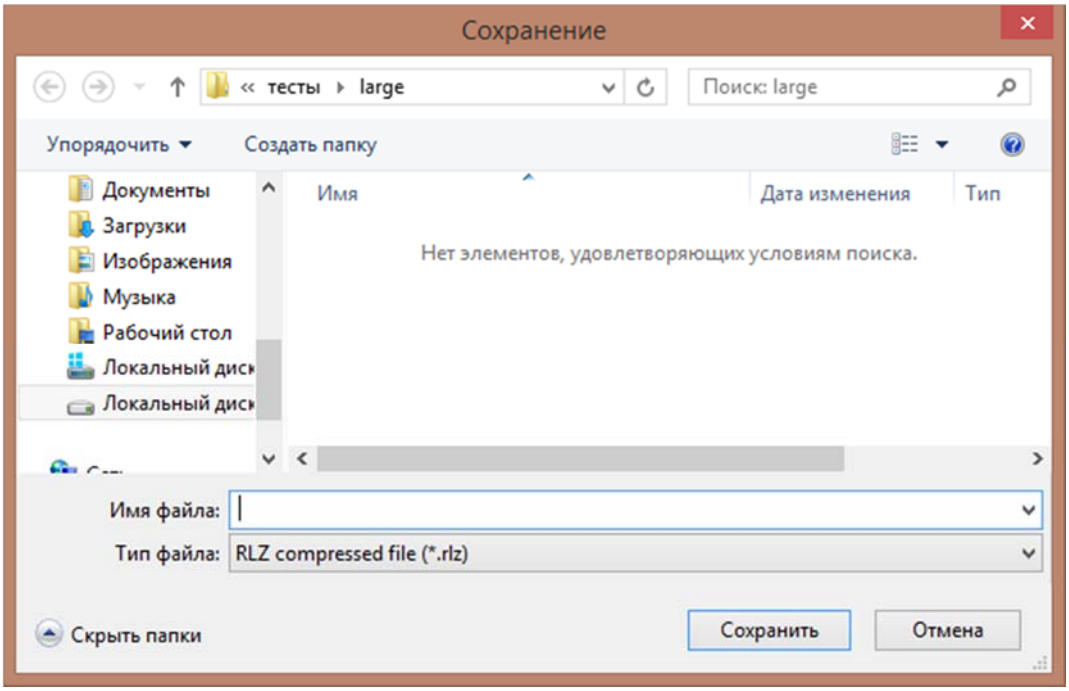

Рисунок 3 – Окно сохранения файла при сжатии

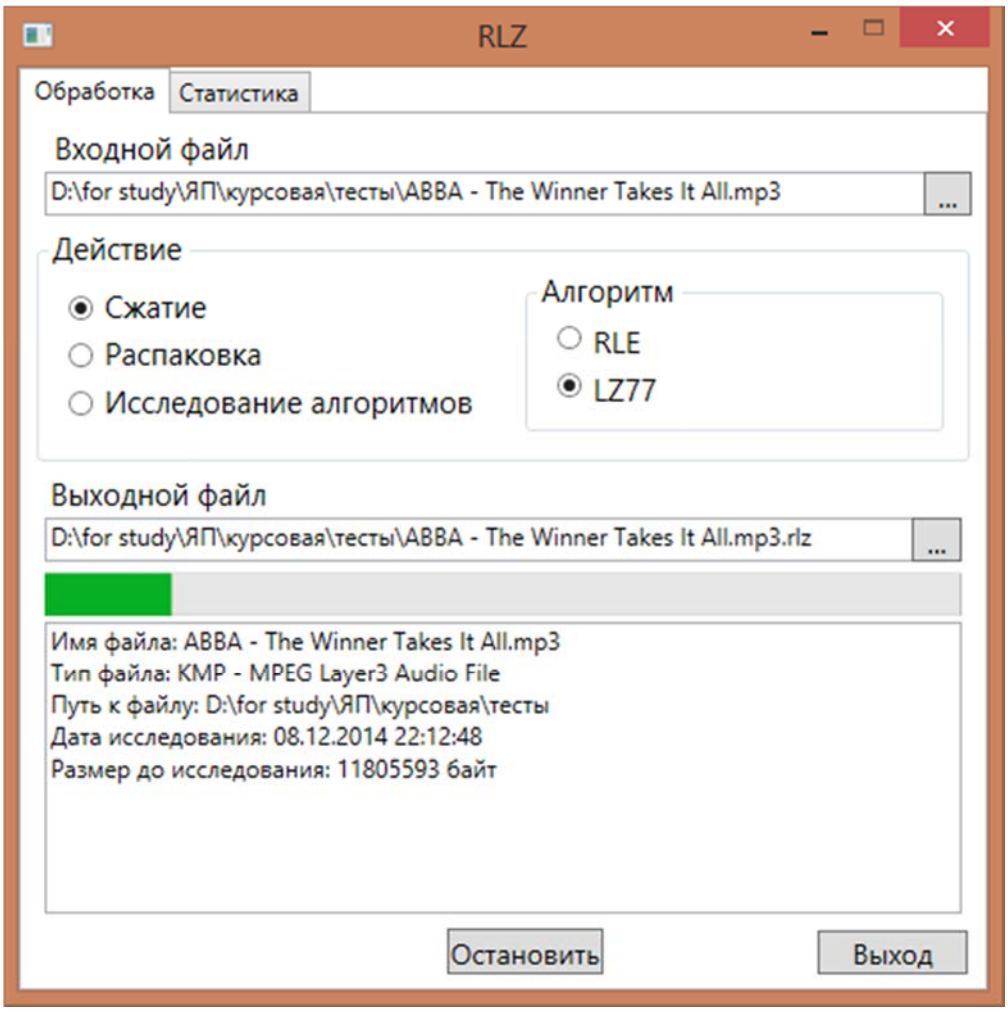

Рисунок 4 – Состояние окна после начала обработки

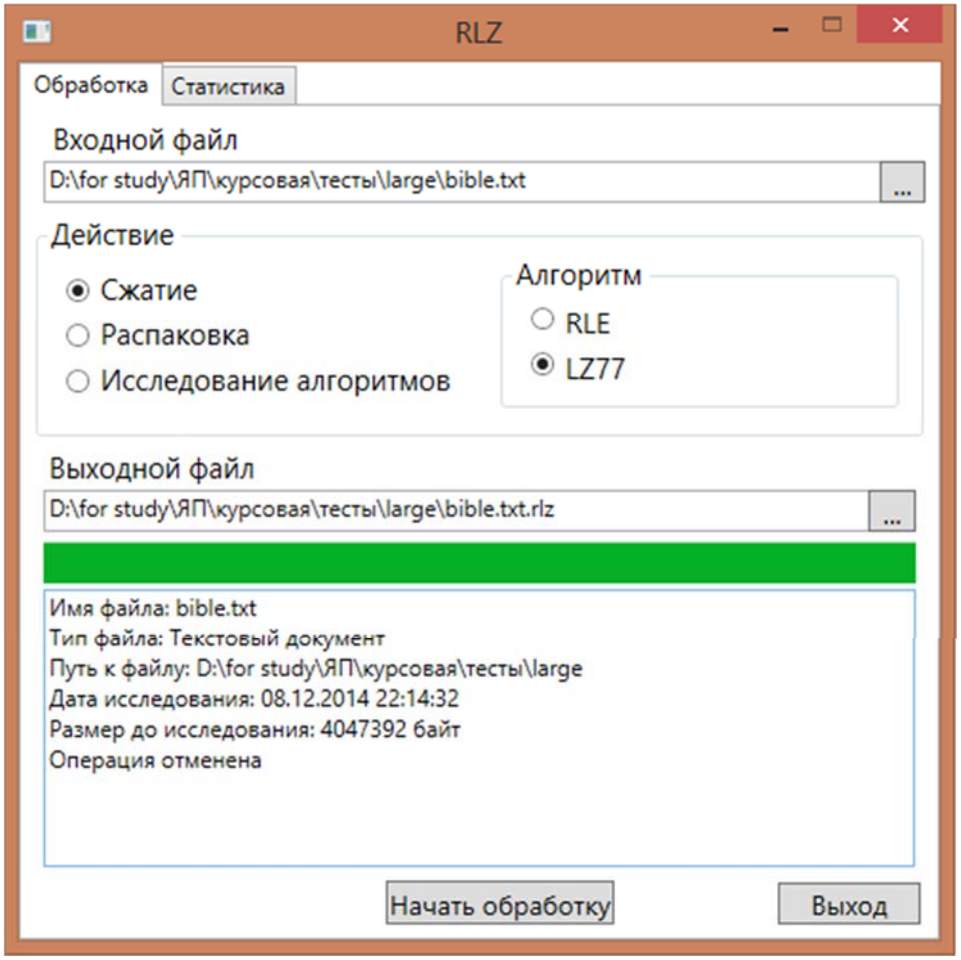

Рисунок 5 – Состояние окна при принудительной остановке обработки

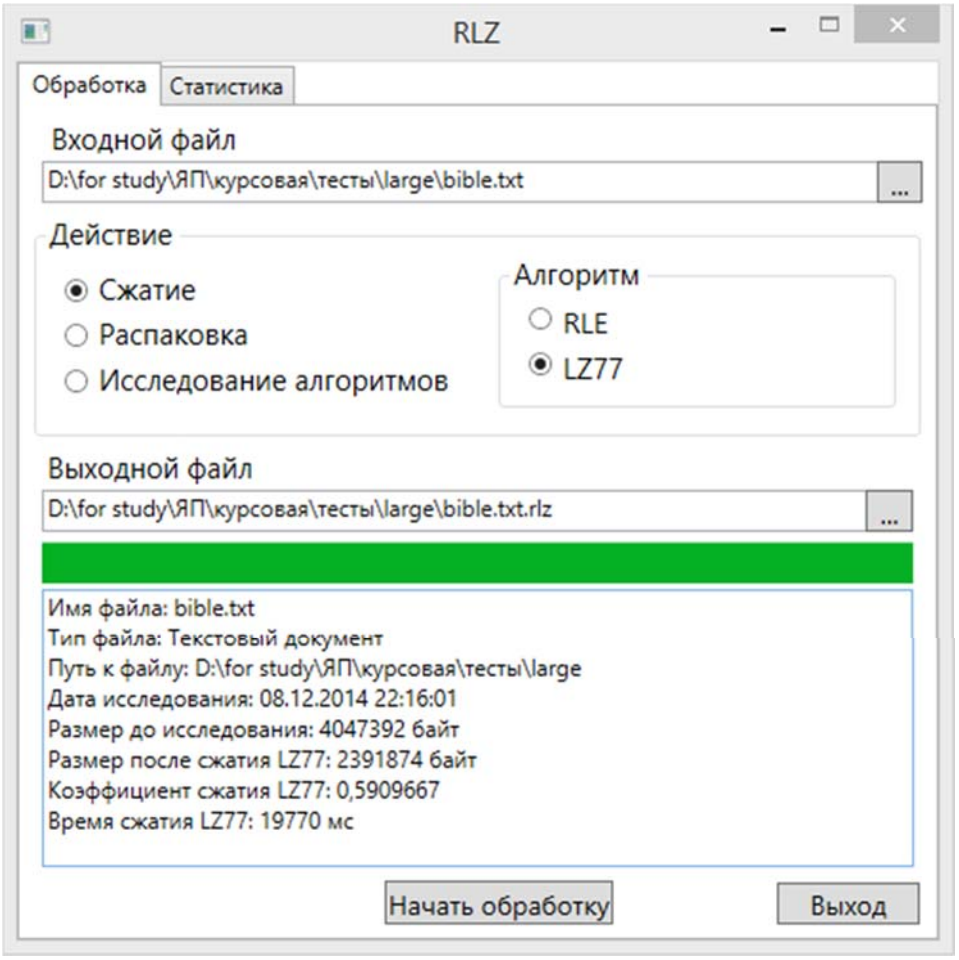

Рисунок 6 – Состояние окна при успешном завершении сжатия

4.2 Распаковка файлов

Для начала необходимо выбрать режим «Действие» - «Распаковка».

Так как выбор алгоритма при распаковке происходит автоматически, то выбор алгоритма становится невозможным.

Затем необходимо выбрать входной файл на диске. Для этого необходимо нажать кнопку «…», расположенную справа от поля пути входного файла. При этом открывается окно, представленное на рисунке 7.

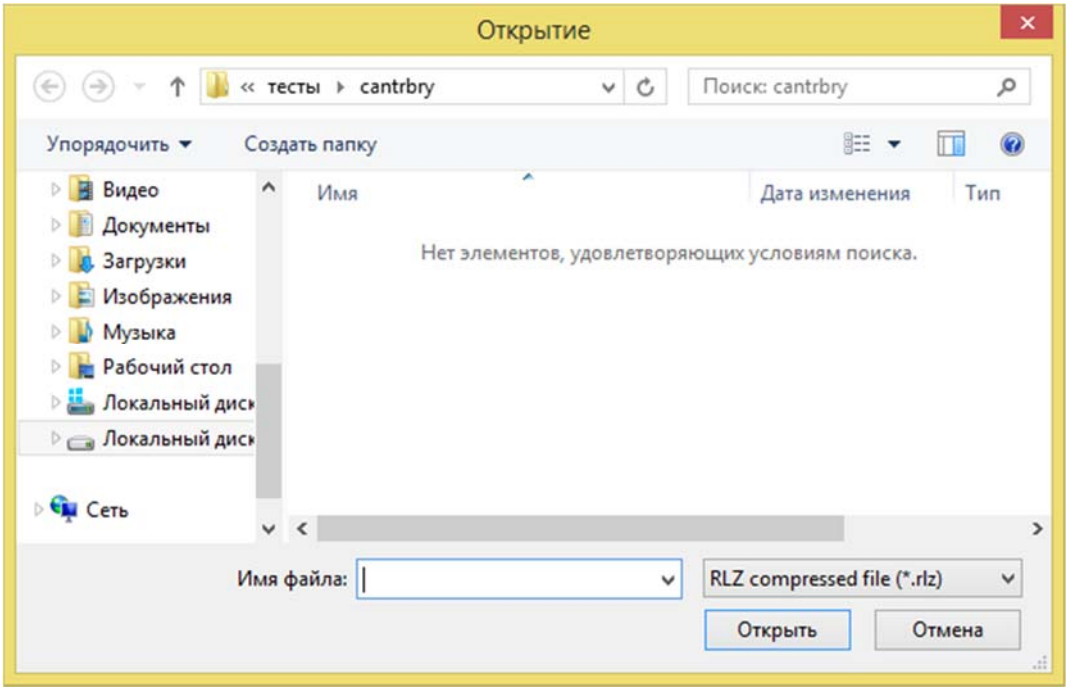

Рисунок 7 – Окно открытия файла при распаковке

После выбора пользователем входного файла, автоматически заполняются поля путей входного и выходного файлов, причем предлагается сохранить распакованный файл в той же директории, что и сжатый. Однако пользователь может сам выбрать директорию для сохранения. Для этого необходимо нажать кнопку «…», расположенную справа от поля пути выходного файла. При этом открывается окно, представленное на рисунке 8.

Затем пользователь, нажав клавишу «Enter» на клавиатуре или кнопку «Начать обработку» на форме, может запустить обработку входного файла; При этом пользователь увидит состояние окна, подобное тому, что представлено на рисунке 4.

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, что приведено на рисунке 9.

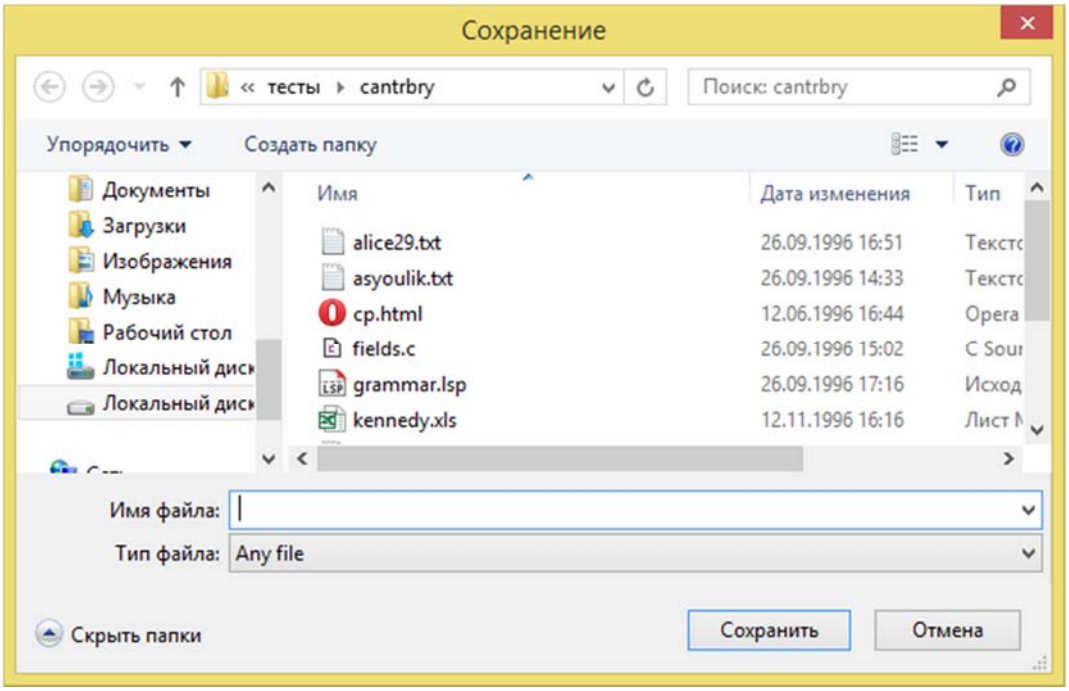

Рисунок 8 – Окно сохранение файла при распаковке

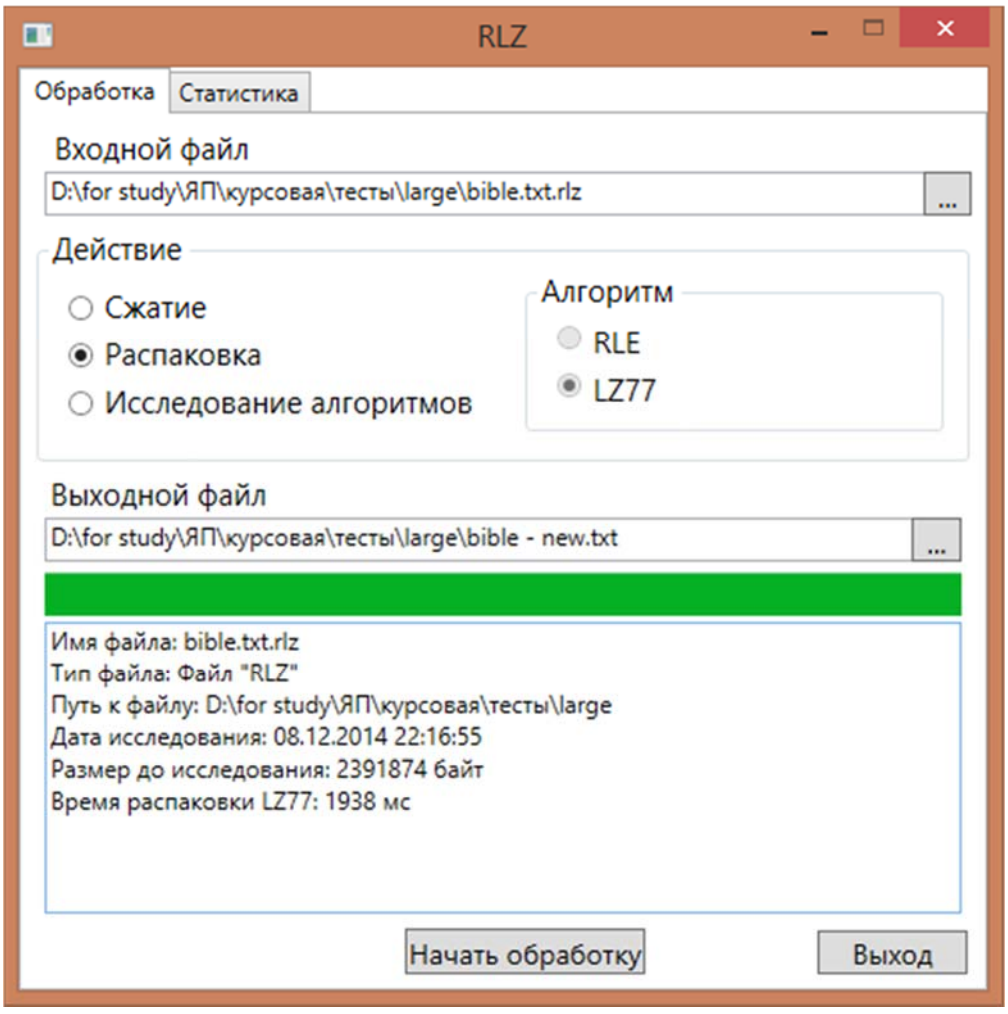

Рисунок 9 – Состояние окна после завершения распаковки

4.3 Исследование алгоритмов

Для начала необходимо выбрать режим «Действие» - «Исследование».

Так как при исследовании последовательно запускаются все алгоритмы, то выбор алгоритма становится невозможным.

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

При этом автоматически заполняются поля путей входного и выходного файла, причем предлагается сохранить временный файл в той же директории (но с добавленным расширением «.rlz»). Однако пользователь самостоятельно выбрать место для хранения временного файла.. 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. Так как последовательно запускаются все алгоритмы, то до завершения обработки индикатор заполнится четыре раза.

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

Если же исследование будет завершена успешно, то будут выведены характеристики процесса сжатия, и пользователь увидит состояние окна, подобное до того, что приведено на рисунке 10.

4.4 Просмотр статистики

Переключив вкладку можно просмотреть статистику, собранную во время предыдущих исследований. Окно, которое видит пользователь, представлено на рисунке 11.

Если таблица не пуста, то ее можно очистить с помощью кнопки «Удалить все записи». Если выбрана какая-либо запись, то ее можно удалить с помощью кнопки «Удалить запись».

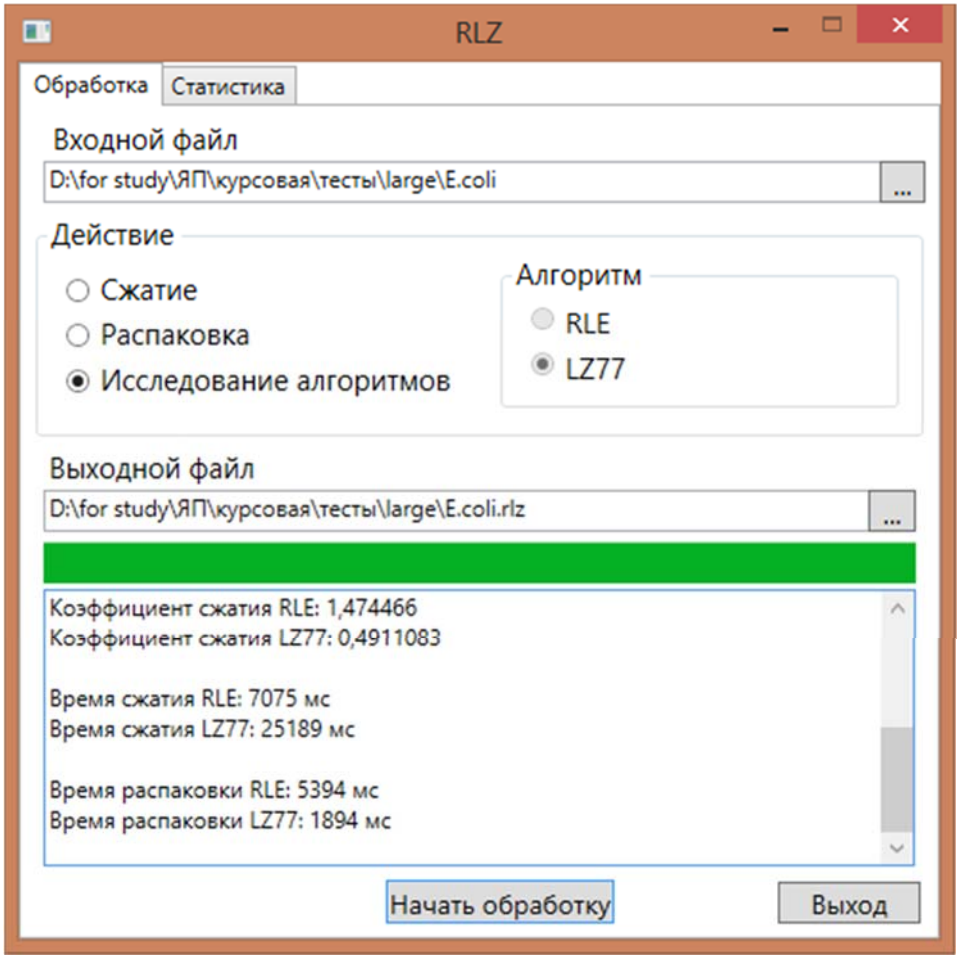

Рисунок 10 – Состояние окна после завершения исследования

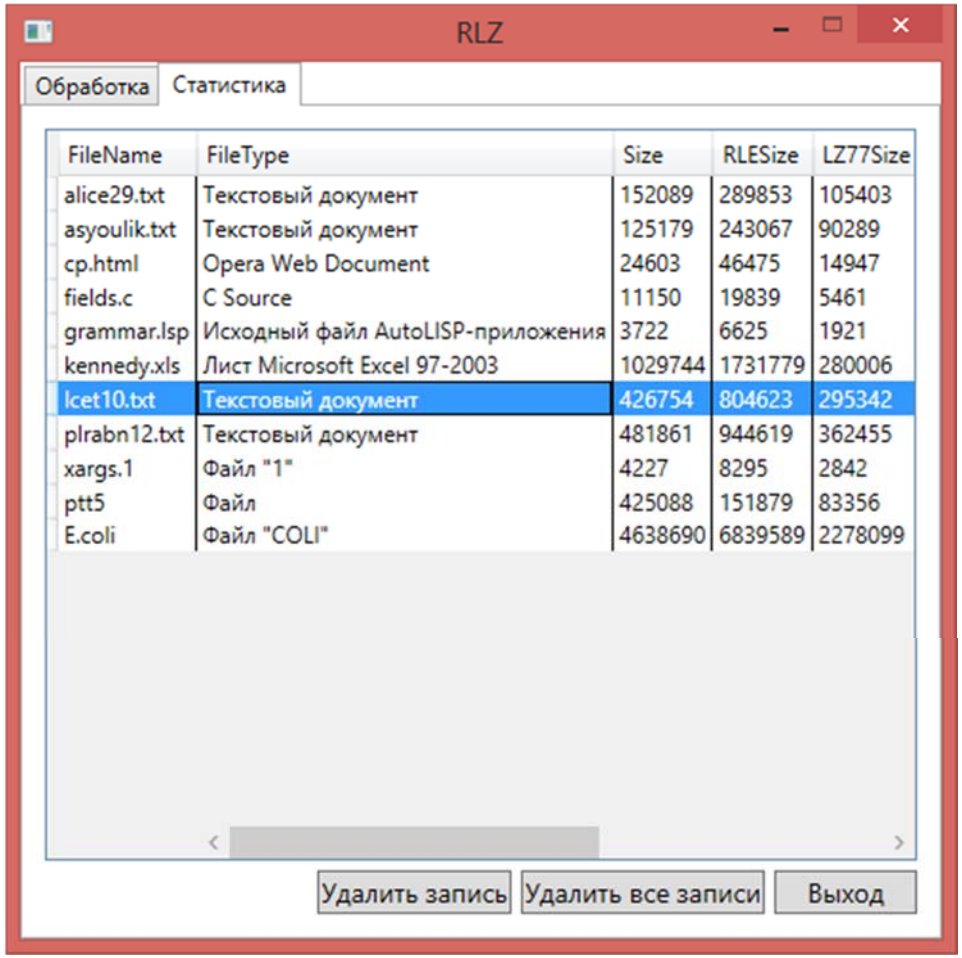

Рисунок 11 – Вкладка статистики

5 ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

Будем проводить тестирование по нескольким заранее разработанным ситуациям.

5.1 Тестирование ситуации сжатия

Шаги тестируемой ситуации сжатия, ожидаемый и фактический результаты приведены в таблице 10.

Таблица 10 – Тестирование ситуации сжатия

Шаг	Ожидаемый результат	Фактический результат
Запуск приложения	Приложения запускается без ошибок; выбрано действие «Сжатие», алгоритм – «RLE»	Приложение запускается без ошибок
Нажатие кнопки «…» выбора входного файла	Отображается окно выбора входного файла	Действие происходит без ошибок
Выбрать файл «тесты/large/world192.txt», нажать кнопку «Открыть»	Выбор файла происходит без ошибок; происходит заполнение полей путей входного и выходного файлов; пути входного и выходного файлов совпадают; выходной файл имеет расширение «.rlz»	Действия происходят без ошибок
Нажать кнопку «Начать обработку»	Начинается обработка файла; появляются начальные характеристики процесса сжатия	Действия происходят без ошибок
Не дожидаясь завершения, нажать кнопку «Остановить»	Кнопка видна и активна; происходит остановка обработки; появляется сообщение о том, что обработка прервана	Действия происходят без ошибок
Нажать кнопку «Начать обработку»	Начинается обработка файла; появляются начальные характеристики процесса сжатия	Действия происходят без ошибок
Дождаться завершения обработки	Обработка завершается успешно; появляются окончательные характеристики процесса сжатия	Действия происходят без ошибок

Продолжение таблицы 10

Шаг	Ожидаемый результат	Фактический результат
Переключить вкладку	В таблице статистики отсутствуют записи	Действие происходят без ошибок
Нажать кнопку на клавиатуре «Esc»	Приложение закрывается	Приложение закрывается без ошибок
Примечание – Файл, над которым проводится обработка в данной тестируемой ситуации, представляет собой текстовый файл размером 2 473 400 байт, по содержанию представляющий собой информационный бюллетень проекта «Гуттенберг» – проекта по созданию электронной универсальной библиотеки [9].		

5.2 Тестирование ситуации распаковки

Шаги тестируемой ситуации распаковки, ожидаемый и фактический результаты приведены в таблице 11.

Таблица 11 – Тестирование ситуации распаковки

Шаг	Ожидаемый результат	Фактический результат
Запуск приложения	Приложения запускается без ошибок; выбрано действие «Сжатие», алгоритм – «RLE»	Приложение запускается без ошибок
Нажать кнопку «…» выбора входного файла	Отображается окно выбора входного файла	Действие происходит без ошибок
Выбрать файл «тесты/large/world192.txt», нажать кнопку «Открыть»	Выбор файла происходит без ошибок; происходит заполнение полей путей входного и выходного файлов; пути входного и выходного файлов совпадают; выходной файл имеет расширение «.rlz»	Действия происходят без ошибок
Выбрать режим «Распаковка»	Происходит очищение полей путей входного и выходного файлов; выбор алгоритма становится недоступным	Действие происходит без ошибок
Нажать кнопку «…» выбора входного файла	Отображается окно выбора входного файла	Действие происходит без ошибок

Продолжение таблицы 10

Шаг	Ожидаемый результат	Фактический результат
Выбрать файл «тесты/large/ world192.txt.rlz», нажать кнопку «Открыть»	Выбор файла происходит без ошибок; происходит заполнение полей путей входного и выходного файлов; пути входного и выходного файлов совпадают; выходной файл имеет расширение «.txt»	Действия происходят без ошибок
Нажать кнопку «Начать обработку»	Начинается обработка файла; появляются начальные характеристики процесса распаковки	Действия происходят без ошибок
Дождаться завершения обработки	Обработка завершается успешно; появляются окончательные характеристики процесса распаковки	Действия происходят без ошибок
Нажать кнопку «Выход»	Приложение закрывается без ошибок	Действие происходит без ошибок

5.3 Тестирование ситуации исследования

Шаги тестируемой ситуации исследования, ожидаемый и фактический результаты приведены в таблице 12.

Таблица 12 – Тестирование ситуации исследования

Шаг	Ожидаемый результат	Фактический результат
Запуск приложения	Приложения запускается без ошибок; выбрано действие «Сжатие», алгоритм – «RLE»	Приложение запускается без ошибок
Выбрать действие «Исследование алгоритмов»	Выбор алгоритма становится недоступным	Действие происходит без ошибок
Нажать кнопку «…» выбора входного файла	Отображается окно выбора входного файла	Действие происходит без ошибок
Выбрать файл «тесты/large/ world192.txt», нажать кнопку «Открыть»	Выбор файла происходит без ошибок; происходит заполнение полей путей входного и выходного файлов; выходной файл имеет расширение «.rlz»	Действия происходят без ошибок

Продолжение таблицы 12

Шаг	Ожидаемый результат	Фактический результат
Нажать клавишу на клавиатуре «Enter»	Начинается обработка файла; появляются начальные характеристики исследования	Действие происходит без ошибок
Не дожидаясь завершения обработки, переключить вкладку	Таблица статистики пуста;	Действие происходит без ошибок
Дождаться появление в таблице первой записи	В таблице запись появляется, это означает, что обработка завершена успешно	Действие происходит без ошибок
Переключить вкладку	Переключение возможно	Переключение происходит без ошибок
Нажать кнопку «Начать обработку» и дождаться завершения обработки еще дважды	Начинается обработка; индикатор выполнения полностью заполняется четыре раза за каждое исследование	Действия происходят без ошибок
Переключить вкладку	Переключение возможно; в таблице содержатся три записи с одинаковым именем файла	Действия происходят без ошибок
Выделить вторую запись; нажать кнопку «Удалить запись»	Происходит удаление выбранной записи	Действия происходят без ошибок
Нажать кнопку «Выход»	Происходит закрытие приложения	Приложение успешно закрывается
Запустить приложение	Приложения запускается без ошибок; выбрано действие «Сжатие», алгоритм – «RLE»	Приложение запускается без ошибок
Переключить вкладку	Переключение возможно; в таблице содержатся две записи с одинаковым именем файла; изменения таблицы сохраняются после закрытия приложения	Действия происходят без ошибок
Нажать кнопку «Выход»	Происходит закрытие приложения	Приложение успешно закрывается

6 ИССЛЕДОВАНИЕ АЛГОРИТМОВ

Будем проводить сравнение алгоритмов по следующим характеристикам: коэффициент сжатия, время сжатия, время распаковки.

Как было сказано в пункте 2.3, для исследования алгоритмов будем использовать набор Canterbury Corpus. В таблице 13 приведены результаты исследования файлов, входящих в этот набор.

Таблица 13 – Исследование алгоритмов RLE и LZ77 на наборе файлов Canterbury Corpus

Имя файла	Размер до обработки, байт	Коэффициент сжатия RLE	Коэффициент сжатия LZ77	Время сжатия RLE, мс	Время сжатия LZ77, мс	Время распаковки RLE, мс	Время распаковки LZ77, мс
alice29.txt	152089	1,91	0,69	238	432	234	76
asyoulik.txt	125179	1,94	0,72	199	354	194	69
cp.html	24603	1,88	0,61	40	61	38	12
fields.c	11150	1,77	0,49	22	26	18	4
grammar.lsp	3722	1,77	0,52	7	10	7	2
kennedy.xls	1029744	1,68	0,27	1597	2275	1402	268
lcet10.txt	426754	1,88	0,69	650	1182	643	211
plrabn12.txt	481861	1,96	0,75	728	1411	747	260
ptt5	513216	0,30	0,16	747	1155	127	92
sum	38242	1,53	0,57	61	104	46	16
xargs.1	4227	1,96	0,67	10	13	8	2

Проанализировав полученные данные, можно установить некоторые особенности алгоритмов RLE и LZ77:

– Неэффективность алгоритма RLE. Практически на всех файлах этот алгоритм показал наихудший результат по качеству сжатия (коэффициент сжатия приблизительно равен двум).

– Однако, высокая эффективность этого алгоритма проявляется на38 файлах с большим количеством серий. Таким файлом и является файл ptt5, который представляет собой необработанное факс-изображение.

– Эффективность алгоритма LZ77. Все файлы данного набора этот алгоритм обработал с коэффициентом сжатия. меньшим единицы. Фактически, алгоритм LZ77 является частным случаем алгоритмом RLE, поэтому его качество сжатия даже на файле ptt5 оказалось выше, чем у алгоритма RLE.

– Время сжатия и распаковки алгоритма RLE оказалось приблизительно равным, что говорит о примерно одинаковой алгоритмической сложности сжатия и распаковки.

– Алгоритм LZ77 отличается несимметричностью сжатия и распаковки по времени: распаковка во всех случаях в несколько раз быстрее чем сжатие. И в самом деле, при сжатии этим алгоритмом большое количество времени тратится на поиск совпадения, когда при распаковке этого производить не нужно.

– Также в полтора-два раза время сжатия алгоритмом LZ77 больше времени сжатия алгоритмом RLE. Так получилось вследствие того, что, опять же, большое время тратится на поиск совпадения, а также из-за использования неоптимальной структуры List.

Результаты исследования свидетельствуют о том, что простейший алгоритм RLE крайне неэффективен, и его приемлемо применять только на немногих типах файлов. Алгоритм LZ77 значительно сложнее в реализации, чем RLE. Но и эффективность сжатия также значительно выше.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта изучены базовые метода сжатия и распаковки RLE и LZ77. На основании этих методов составлены алгоритмы, которые затем были реализованы в программном приложении, позволяющее производить сжатие файлов, распаковку ранее сжатых файлов, а также проводить сравнение исследуемых алгоритмов с последующим наглядным представлением собранной в результате исследования информации.

Алгоритм LZ77, в основе которого лежит базовый алгоритм RLE, эффективен даже без применения каких-либо оптимизаций и улучшений. Тем не менее, различные модификации этого алгоритма образовывают целое семейство эффективных современных алгоритмов.

В результате выполнения курсового проекта были закреплены навыки программирования на языке C#, использования различных стандартных структур данного языка, создания оконных приложения с использованием технологии WPF.

При дальнейшей разработке будут реализованы следующие возможности:

– возможность приостановить обработку;

– сжатие и распаковка файлов другими алгоритмами семейства LZ, алгоритмами других семейств;

– исследование этих алгоритмов сжатия и распаковки;

– автоматический выбор применяемого алгоритма сжатия в зависимости от типа файла;

– реализация более оптимального алгоритма поиска совпадающей подпоследовательности для алгоритма LZ77.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

[1] Википедия, Сжатие данных [Электронный ресурс]. - Электронные данные. - Режим доступа: http://ru.wikipedia.org/wiki/Сжатие%20данных..

[2] A Universal Algorithm for Sequential Data Compression / J. Ziv, A. Lempel // IEEE Transactions of Information Theory. - 1977. - T.IT-23, № 3, pp. 337-343.

[3] Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео: учебно-справочное издание / Д. Ватолин [и др.] - М: ДИАЛОГ-МИФИ, 2003. - 384 с.

[4] Data Compression. The Complete Reference. Fourth Edition. / D. Salomon - London: Springer-Verlag, 2007.

[5] Википедия, Кодирование длин серий, [Электронный ресурс]. - Электронные данные. - Режим доступа: http://ru.wikipedia.org/wiki/RLE.

[6] A corpus for the evaluation of lossless compression algorithms / A. Ross, Bell T. - Christchurch: Department of Computer Science, University of Canterbury,

[7] Text Compression / T. C. Bell, J. G. Cleary, I. H. Witten. - Englewood Cliffs: Prentince Hall, 1990.

[8] MSDN, BackgroundWorker - класс [Электронный ресурс]. - Электронные данные. - Режим доступа: http://msdn.microsoft.com/ru-ru/library/system.componentmodel.backgroundworker(v=vs.110).aspx.

[9] Large Corpus [Электронный ресурс]. - Электронные данные. - Режим доступа: http://corpus.canterbury.ac.nz/resources/large.zip.

[10] ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. - Введ. 01.01.1992, М: Изд-во стандартов, 1991.

[11] Доманов, А.Т. Стандарт предприятия. Дипломные проекты (работы). Общие требования / А. Т. Доманов, Н. И. Сорока - Минск: БГУИР, 2010.

ПРИЛОЖЕНИЕ А
(обязательное)
Исходные коды функций

```
class LZ77
{
  const int stepToUpateProgressbar = 1000;

  const int substringIndexBits = 10;
  const int substringLengthBits = 6;

  const int maxDictionarySize = 1024;      // == 2 ^ 10
  const int maxBufferSize = 64;            // == 2 ^ 6

  static public MainWindow.MetricsOfAlgorythm
Encode(RLZ.MainWindow.IOFilePaths filePaths, ref
System.ComponentModel.BackgroundWorker backgroundWorker)
  {
    MainWindow.MetricsOfAlgorythm LZ77EncodindMetrics = new
MainWindow.MetricsOfAlgorythm();

    BinaryReader inputFile = new
BinaryReader(File.Open(filePaths.input, FileMode.Open));
    BinaryWriter outputFile = new
BinaryWriter(File.Open(filePaths.output, FileMode.Create));

    outputFile.Write((byte)RLZ.MainWindow.algorytm.lz77);

    List<byte> slidingWindow = new
List<byte>(inputFile.ReadBytes(maxBufferSize));
    int bufferSize = slidingWindow.Count;
    int bufferStart = 0;

    int dictionarySize = 0;
    while (bufferSize != 0)
    {
      int bufferCounter = bufferStart;
      int dictionaryCounter = 0;

      int maxEqualLength = 0;
      int maxDictionaryEqualStart = dictionaryCounter;

      if (dictionarySize != 0)
      {
        while (dictionaryCounter != bufferStart)
```

```
            {
              int equalLength = 0;
              int dictionaryEqualStart = dictionaryCounter;

              while ((bufferCounter < slidingWindow.Count - 1) &&
(slidingWindow[dictionaryCounter] == slidingWindow[bufferCounter]))
              {
                equalLength++;
                bufferCounter++;
                dictionaryCounter++;
              }

              if (equalLength > maxEqualLength)
              {
                maxEqualLength = equalLength;
                maxDictionaryEqualStart = dictionaryEqualStart;
              }

              dictionaryCounter = dictionaryEqualStart + 1;
              bufferCounter = bufferStart;
            }
          }

          outputFile.Write((byte)((maxDictionaryEqualStart) >>
(substringIndexBits - 8)));
          outputFile.Write((byte)(((maxDictionaryEqualStart) <<
substringLengthBits) + maxEqualLength));

          outputFile.Write(slidingWindow[bufferStart + maxEqualLength]);

          //восстановление плавающего окна
          for (int i = 0; i <= maxEqualLength; i++)
          {
            if (dictionarySize == maxDictionarySize)
            {
              slidingWindow.RemoveAt(0);
            }
            else
            {
              dictionarySize++;
              bufferStart++;
            }
            if (inputFile.BaseStream.Position != inputFile.BaseStream.Length)
            {
              slidingWindow.Add(inputFile.ReadByte());
```

```
            }
            else
            {
              bufferSize--;
            }
          }

          if (inputFile.BaseStream.Position % stepToUpateProgressbar == 0)
          {

backgroundWorker.ReportProgress((int)inputFile.BaseStream.Position);
            if (backgroundWorker.CancellationPending)
            {
              break;
            }
          }
        }


LZ77EncodindMetrics.FinalizeMetrics(outputFile.BaseStream.Length);

        inputFile.Close();
        outputFile.Close();
        if (backgroundWorker.CancellationPending)
        {
          File.Delete(filePaths.output);
        }

        return LZ77EncodindMetrics;
      }


      static public MainWindow.MetricsOfAlgorythm
Decode(RLZ.MainWindow.IOFilePaths filePaths, ref
System.ComponentModel.BackgroundWorker backgroundWorker)
      {
        MainWindow.MetricsOfAlgorythm LZ77DecodingMetrics = new
MainWindow.MetricsOfAlgorythm();

        BinaryReader inputFile = new
BinaryReader(File.Open(filePaths.input, FileMode.Open));
        BinaryWriter outputFile = new
BinaryWriter(File.Open(filePaths.output, FileMode.Create));

        inputFile.BaseStream.Seek(1, SeekOrigin.Begin);
```

```
        List<byte> slidingWindow = new List<byte>();
        int equalLength;
        int dictionaryEqualStart;
        int dictionarySize = 0;
        byte nextSymbol;

        while (inputFile.BaseStream.Position != inputFile.BaseStream.Length)
        {
          dictionaryEqualStart = inputFile.ReadByte();
          equalLength = inputFile.ReadByte();
          dictionaryEqualStart = (dictionaryEqualStart << (substringIndexBits
- 8)) + (equalLength >> substringLengthBits);
          equalLength = equalLength & (maxBufferSize - 1);

          nextSymbol = inputFile.ReadByte();

          for (int i = 1; i <= equalLength; i++)
          {
            slidingWindow.Add(slidingWindow[dictionaryEqualStart]);
            dictionaryEqualStart++;
            dictionarySize++;
          }

          slidingWindow.Add((byte)nextSymbol);
          dictionarySize++;

          while (dictionarySize > maxDictionarySize)
          {
            outputFile.Write(slidingWindow[0]);
            slidingWindow.RemoveAt(0);
            dictionarySize--;
          }

          if (inputFile.BaseStream.Position % stepToUpateProgressbar == 0)
          {

backgroundWorker.ReportProgress((int)inputFile.BaseStream.Position);
          }
          if (backgroundWorker.CancellationPending)
          {
            break;
          }
        }
```

```
        while (dictionarySize > 0)
        {
          outputFile.Write(slidingWindow[0]);
          slidingWindow.RemoveAt(0);
          dictionarySize--;
        }


LZ77DecodingMetrics.FinalizeMetrics(outputFile.BaseStream.Length);

        inputFile.Close();
        outputFile.Close();
        if (backgroundWorker.CancellationPending)
        {
          File.Delete(filePaths.output);
        }
        return LZ77DecodingMetrics;
      }
    }
```

```
  static class RLE
  {
    const int stepToUpateProgressbar = 10000;

    static public MainWindow.MetricsOfAlgorythm
Encode(RLZ.MainWindow.IOFilePaths filePaths, ref
System.ComponentModel.BackgroundWorker backgroundWorker)
    {
      MainWindow.MetricsOfAlgorythm RLECompressionMetrics = new
MainWindow.MetricsOfAlgorythm();

      BinaryReader inputFile = new
BinaryReader(File.Open(filePaths.input, FileMode.Open));
      BinaryWriter outputFile = new
BinaryWriter(File.Open(filePaths.output, FileMode.Create));

      outputFile.Write((byte)RLZ.MainWindow.algorytm.rle);

      byte currentByte, nextByte;
      int currentLength = 0;
      currentByte = inputFile.ReadByte();
      while (inputFile.BaseStream.Position != inputFile.BaseStream.Length)
      {
        currentLength = 0;
        nextByte = inputFile.ReadByte();
        while ((nextByte == currentByte) && (currentLength < 256) &&
(inputFile.BaseStream.Position != inputFile.BaseStream.Length))
        {
          currentLength++;
          nextByte = inputFile.ReadByte();
        }

        outputFile.Write((byte)currentLength);
        outputFile.Write(currentByte);
        currentByte = nextByte;

        if (inputFile.BaseStream.Position % stepToUpateProgressbar == 0)
        {

backgroundWorker.ReportProgress((int)inputFile.BaseStream.Position);
        }
        if (backgroundWorker.CancellationPending)
        {
          break;
        }
```

```
        }
        outputFile.Write((byte)currentLength);
        outputFile.Write(currentByte);


RLECompressionMetrics.FinalizeMetrics(outputFile.BaseStream.Length);

        inputFile.Close();
        outputFile.Close();

        if (backgroundWorker.CancellationPending)
        {
          File.Delete(filePaths.output);
        }

        return RLECompressionMetrics;
      }

     static public MainWindow.MetricsOfAlgorythm
Decode(RLZ.MainWindow.IOFilePaths filePaths, ref
System.ComponentModel.BackgroundWorker backgroundWorker)
      {
        MainWindow.MetricsOfAlgorythm RLEDecompressionMetrics = new
MainWindow.MetricsOfAlgorythm();

        BinaryReader inputFile = new
BinaryReader(File.Open(filePaths.input, FileMode.Open));
        BinaryWriter outputFile = new
BinaryWriter(File.Open(filePaths.output, FileMode.Create));

        inputFile.BaseStream.Seek(1, SeekOrigin.Begin);

        int currentLength, i;
        byte currentByte;
        while (inputFile.BaseStream.Position != inputFile.BaseStream.Length)
        {
          currentLength = inputFile.ReadByte();
          currentByte = inputFile.ReadByte();
          for (i = 0; i <= currentLength; i++)
          {
            outputFile.Write(currentByte);
          }

          if ((inputFile.BaseStream.Position - 1) % stepToUpateProgressbar
== 0)
```

```
            {

backgroundWorker.ReportProgress((int)inputFile.BaseStream.Position);
            }
            if (backgroundWorker.CancellationPending)
            {
              break;
            }
          }


RLEDecompressionMetrics.FinalizeMetrics(outputFile.BaseStream.Length);

          inputFile.Close();
          outputFile.Close();
          if (backgroundWorker.CancellationPending)
          {
            File.Delete(filePaths.output);
          }

          return RLEDecompressionMetrics;
        }
      }
```

ПРИЛОЖЕНИЕ Б
(обязательное)
Схемы алгоритмов

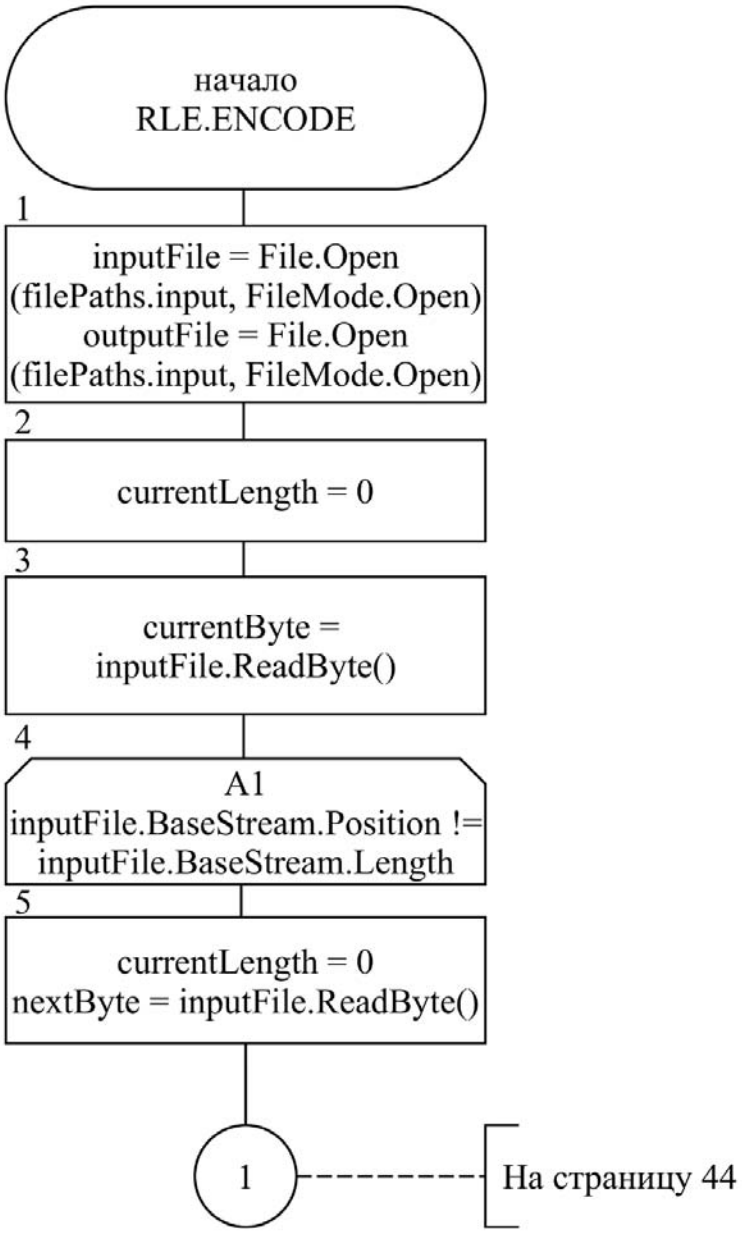

Рисунок Б.1 – Схема алгоритма сжатия RLE

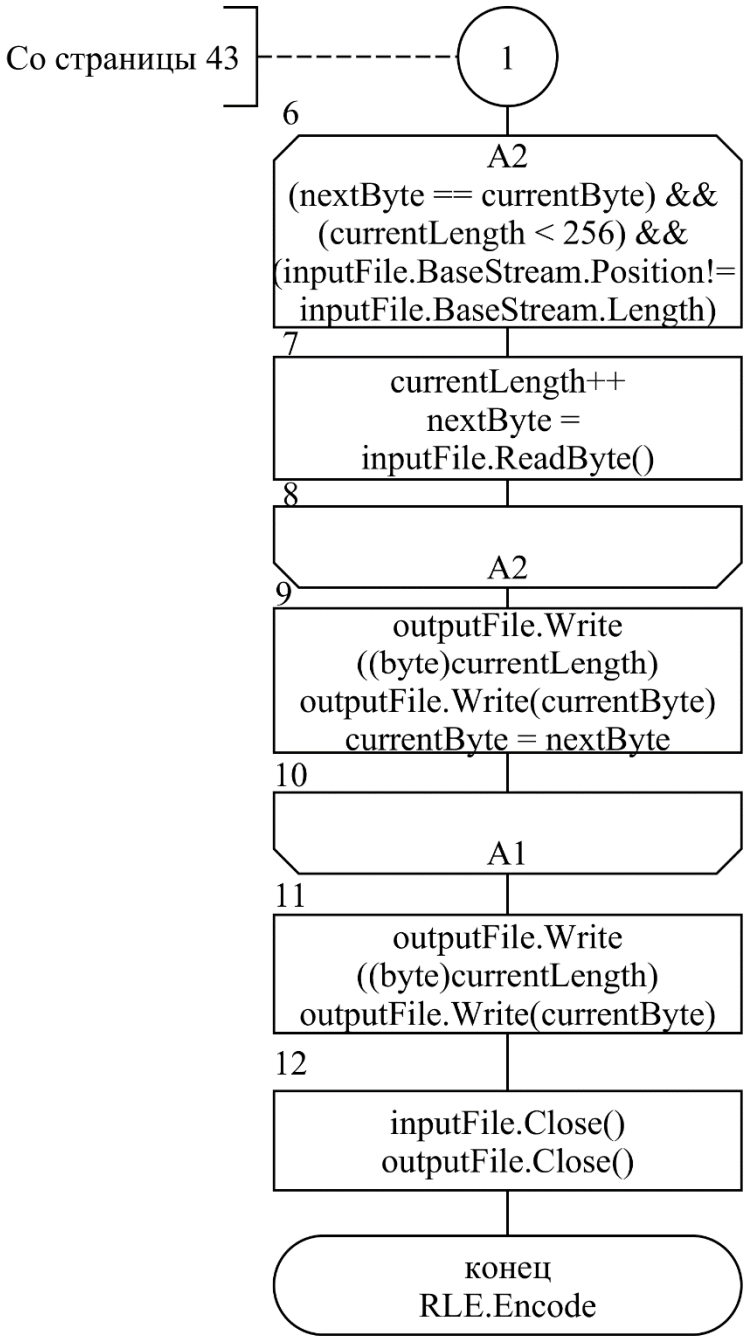

Рисунок Б.2 – Продолжение схемы алгоритма сжатия RLE

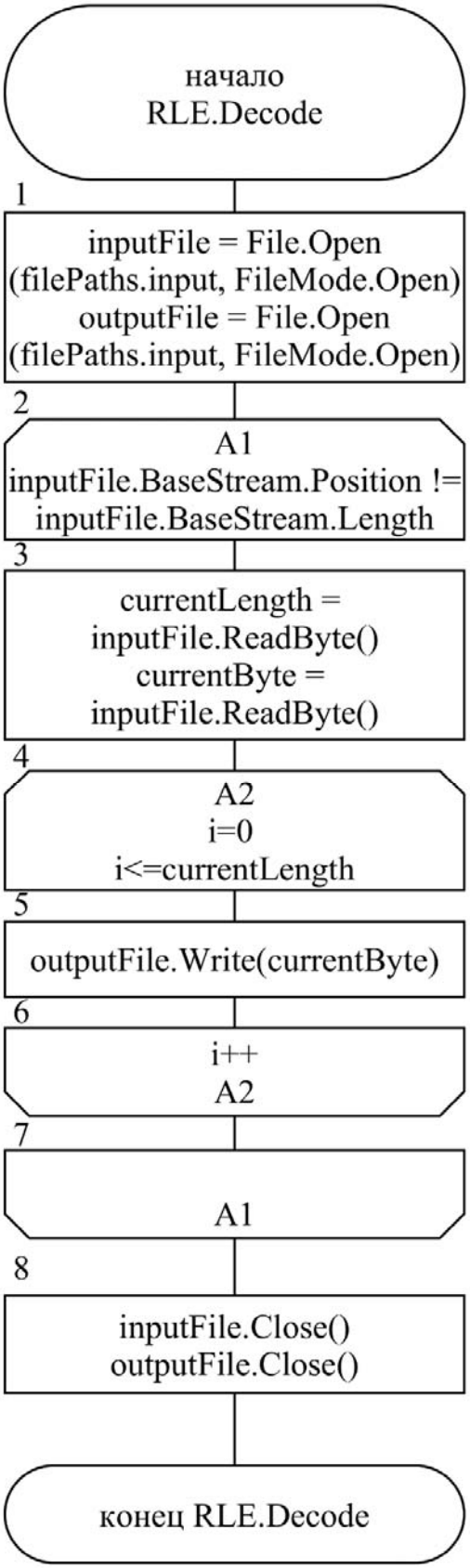

Рисунок Б.3 – Схема алгоритма распаковки RLE

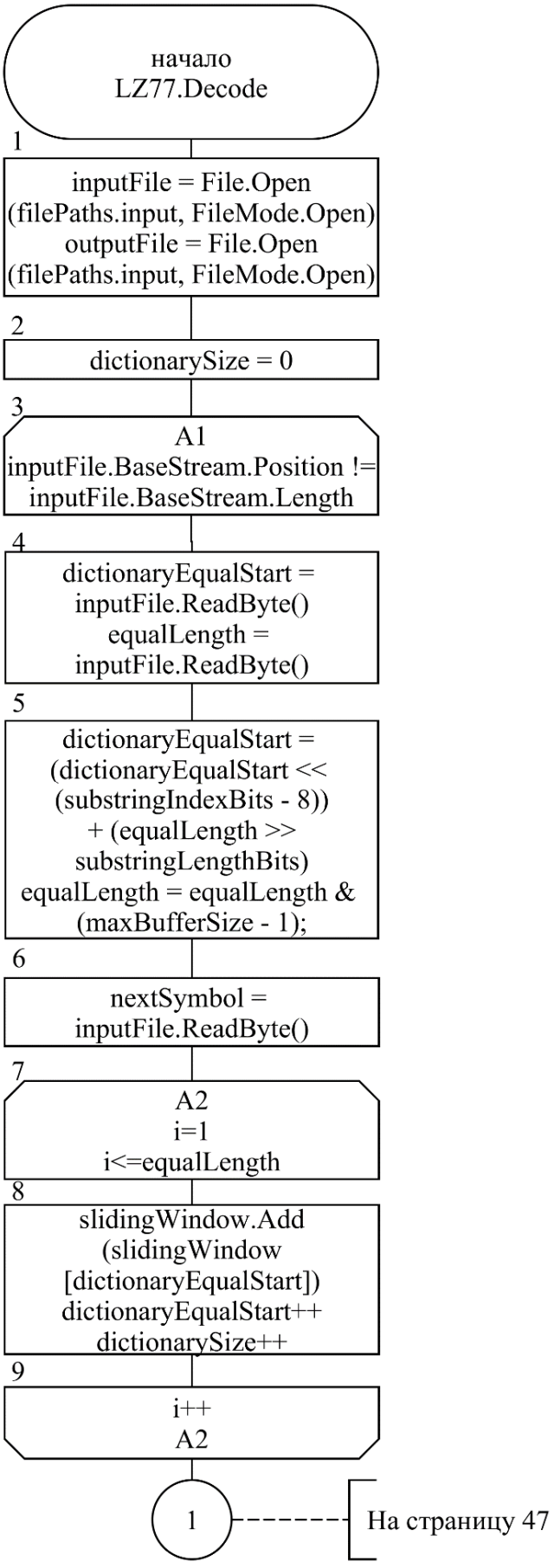

Рисунок Б.4 – Схема алгоритма распаковки LZ77

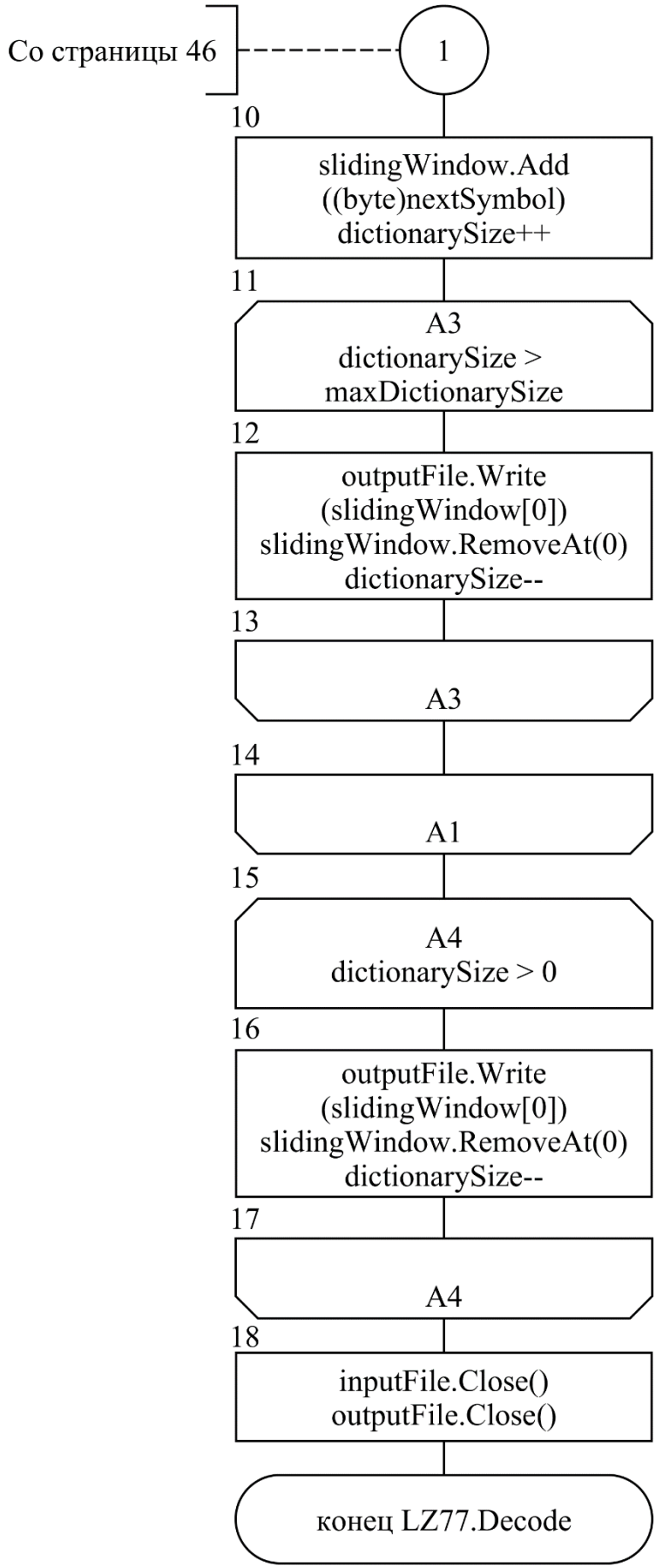

Рисунок Б.5 – Продолжение схемы алгоритма распаковки LZ77