Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Основы алгоритмизации и программирования (ОАиП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему:

«АНАЛИЗ АЛГОРИТМОВ СЖАТИЯ ФАЙЛОВ»

БГУИР КП 1-40 01 01  013  ПЗ

Студент: гр. 851002 Ковалевский М.Ю.

Руководитель: асс. Фадеева Е.Е.

Минск 2019

**СОДЕРЖАНИЕ**

Введение ........................................................................................................ 5

1 Анализ современных программ для сжатия данных............................... 7

* 1. Анализ актуальности сжатия.............................................................. 7

1.2 Программы для индивидуального пользования............................... 8

* 1. Описание популярных алгоритмов сжатия....................................... 11
  2. Формирование требований к проектируемому программному средству........................................................................................................ 14

2 Разработка алгоритма................................................................................. 13

2.1 Анализ требований к программному средству и разработка

функциональных требований...................................................................... 13

2.2 Разработка алгоритма программного средства.................................. 14

2.3 Разработка алгоритма выполнения операций над матрицами...........16

2.4 Разработка алгоритма открытия данных из файла и сохранения

результатов в файл вручную........................................................................17

2.5 Разработка алгоритма добавления и удаления результатов

работы программного калькулятора........................................................... 18

3 Разработка программного средства.............................................................19

3.1 Разработка используемых данных....................................................... 19

3.2 Разработка схемы работы системы...................................................... 21

4 Обоснование технических приемов программирования ..........................23

5 Тестирование ...................................................... .........................................25

5.1 Тестирование математической составляющей функционала

программы......................................................................................................25

5.2 Тестирование прочих функций программы.........................................31

6 Руководство пользователя............................................................................36

Заключение.........................................................................................................42

Список использованных источников .............................................................43

Приложение. Исходный код программы .......................................................44

**ВВЕДЕНИЕ**

Мы даже не задумываемся, какой объем данных генерируем в течение дня, и где все эти данные хранятся. Благодаря интернету, Google в день обрабатывает больше данных, чем было написано во всех литературных работах на всех языках до появления интернета, и это при том, что к паутине подключена далеко не вся планета.

Первый жесткий диск на 5 МБ появился 60 лет назад. Он [весил](http://gagadget.com/15739-evolyutsiya-kompyuternyih-nositelej-informatsii/) около тонны и по размеру был сравним с крупным современным холодильником. Внутри массивного корпуса находилось 50 дисков диаметром 60 сантиметров или примерно 23 дюйма. Сегодня на таком пространстве помещаются две серверные стойки, а маленькое устройство в кармане может хранить несколько сотен гигабайт информации.

Сегодня люди генерируют огромное количество данных. Если учесть, что эти данные затем копируются в облако, это только увеличивает объем памяти, необходимый для их хранения. Такие решения и технологии, как автономные автомобили, интеллектуальные фабрики, интернет вещей (IoT), блокчейн и домашняя автоматизация, будут генерировать дополнительные потоки данных, которые необходимо хранить.

Таким образом ценность сжатия данных ясна. Без уменьшения размера файлов, большим компаниям, как и совершенно обычным людям, потребуется значительно больше места на накопителе. А это привело бы к дополнительным тратам на оборудование и его более быстрому износу.

Сжатие данных – это алгоритмическое преобразование [данных](https://ru.wikipedia.org/wiki/%D0%94%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5), производимое с целью уменьшения занимаемого ими объёма. Применяется для более рационального использования устройств [хранения](https://ru.wikipedia.org/wiki/%D0%A5%D1%80%D0%B0%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%B8) и [передачи](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) данных.

Все методы сжатия данных делятся на два основных класса:

* [Сжатие без потерь](https://ru.wikipedia.org/wiki/%D0%A1%D0%B6%D0%B0%D1%82%D0%B8%D0%B5_%D0%B1%D0%B5%D0%B7_%D0%BF%D0%BE%D1%82%D0%B5%D1%80%D1%8C)
* [Сжатие с потерями](https://ru.wikipedia.org/wiki/%D0%A1%D0%B6%D0%B0%D1%82%D0%B8%D0%B5_%D1%81_%D0%BF%D0%BE%D1%82%D0%B5%D1%80%D1%8F%D0%BC%D0%B8)

При использовании сжатия без потерь возможно полное восстановление исходных данных, сжатие с потерями позволяет восстановить данные с искажениями.

Сжатие без потерь обычно используется для передачи и хранения текстовых данных, компьютерных программ, реже - для сокращения объёма [аудио](https://ru.wikipedia.org/wiki/%D0%90%D1%83%D0%B4%D0%B8%D0%BE)- и [видеоданных](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%B4%D0%B5%D0%BE), [цифровых фотографий](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D1%84%D0%BE%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F) и т. п., в случаях, когда искажения недопустимы или нежелательны. Сжатие с потерями, обладающее значительно большей, чем сжатие без потерь, эффективностью, обычно применяется для сокращения объёма аудио- и видеоданных и цифровых фотографий в тех случаях, когда такое сокращение является приоритетным, а полное соответствие исходных и восстановленных данных не требуется.

Данная записка содержит следующие разделы курсовой работы по разработке ПО для проведения операций над матрицами:

1. Анализ современных программ, дающих возможность сжимать данные;
2. Разработка алгоритма на основе сформированных в результате анализа требований для программы;
3. Разработка программного средства. Выбор структур данных для использования;
4. Обоснование технических приемов программирования. Их преимущества и выгодность использования;
5. Тестирование, анализ работы программы. Описание действий, необходимых для полного тестирования программы;
6. Руководство пользователю. Включает в себя описание действий, которые позволят успешно использовать приложение.

**1 АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ**

**1.1 Актуальности сжатия**

В наше время сжатие данных используется повсеместно. Такие программы, как архиваторы, установлены на каждом компьютере. С Windows 10 в комплекте идет алгоритм сжатия LZX, доступный через PowerShell.

Знакомство со сжатыми файлами начинается с обычных картинок. Большинство изображений хранится в таком формате, как JPEG. Файлы такого формата могут иметь расширения .jpg, .jfif, .jpe или .jpeg. Однако из них .jpg является самым популярным на всех платформах.

Каждый день ведутся разработки новых алгоритмов, позволяющих сжать файл с меньшими потерями и с меньшим итоговым размером.

Сжатие данных получило широкое распространение вместе с интернетом и после изобретения алгоритмов Лемпелем и Зивом (алгоритмы LZ).

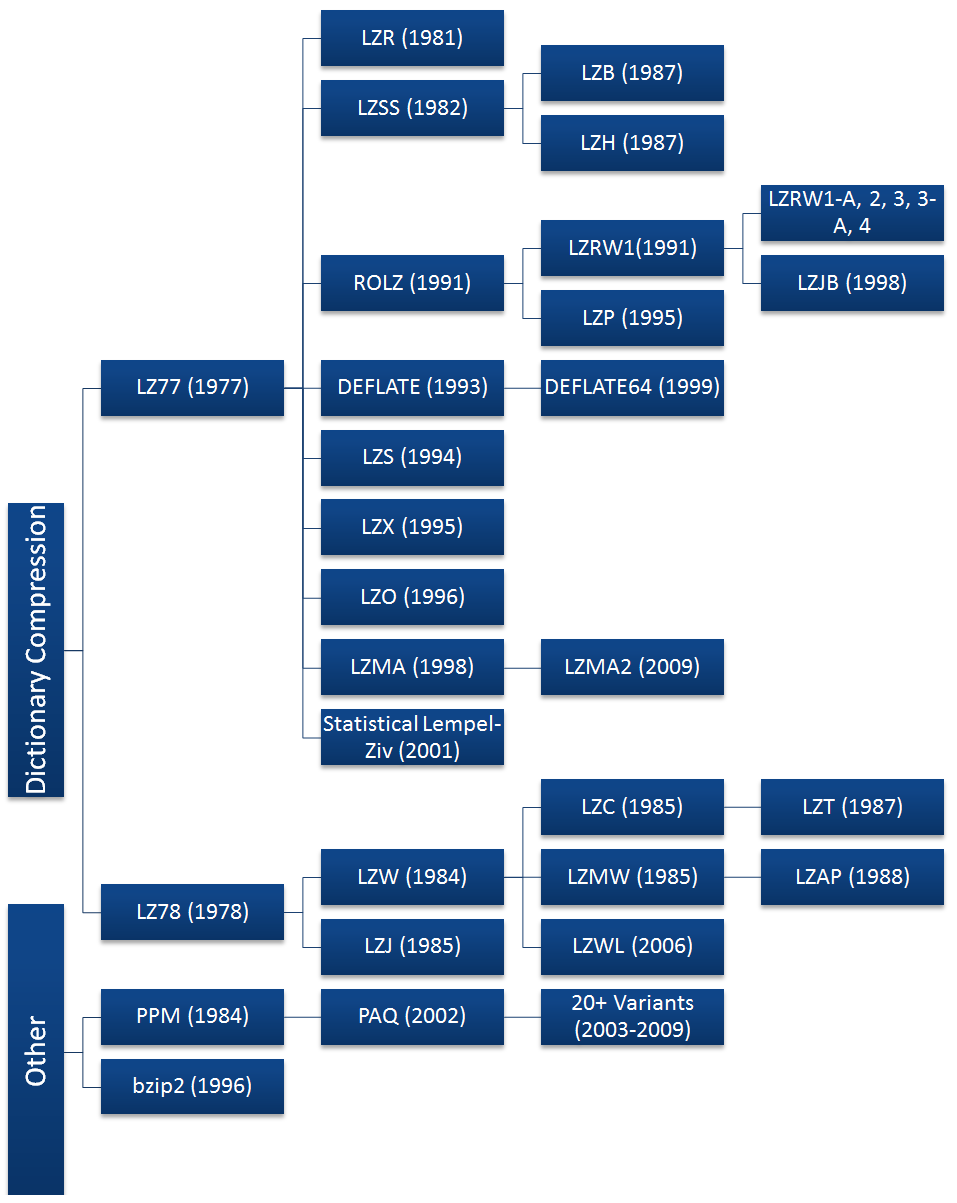


Рисунок 1.1 – Иерархия алгоритмов

**1.2 Программы для сжатия файлов**

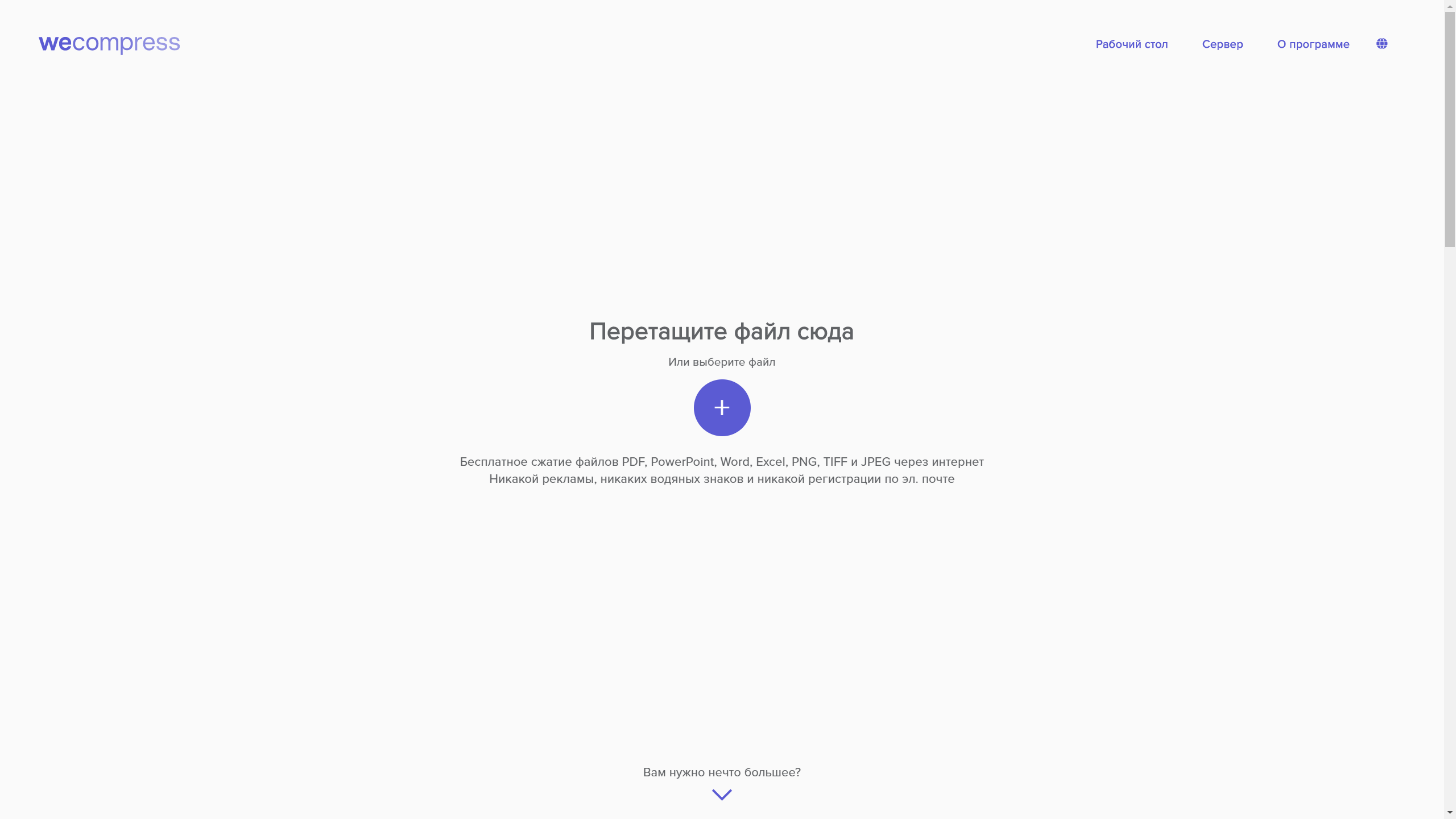
*Онлайн сервис «Wecompress»*.

Плюсы:

* Отличный сайт, позволяющий быстро сжать PDF, PNG, TIFF и JPEG. Приятный и интуитивно понятный пользовательский интерфейс. Содержит вкладку с инструкцией по эксплуатации для пользователя. Переведен на все основные языки мира.

Минусы:

* Онлайн программа, для доступа к ней необходимо подключение к сети, что не всегда является возможным. Сжатие происходит путем уменьшения качества файла без возможности восстановления.

Интерфейс изображен на рисунке 1.2 и 1.3.

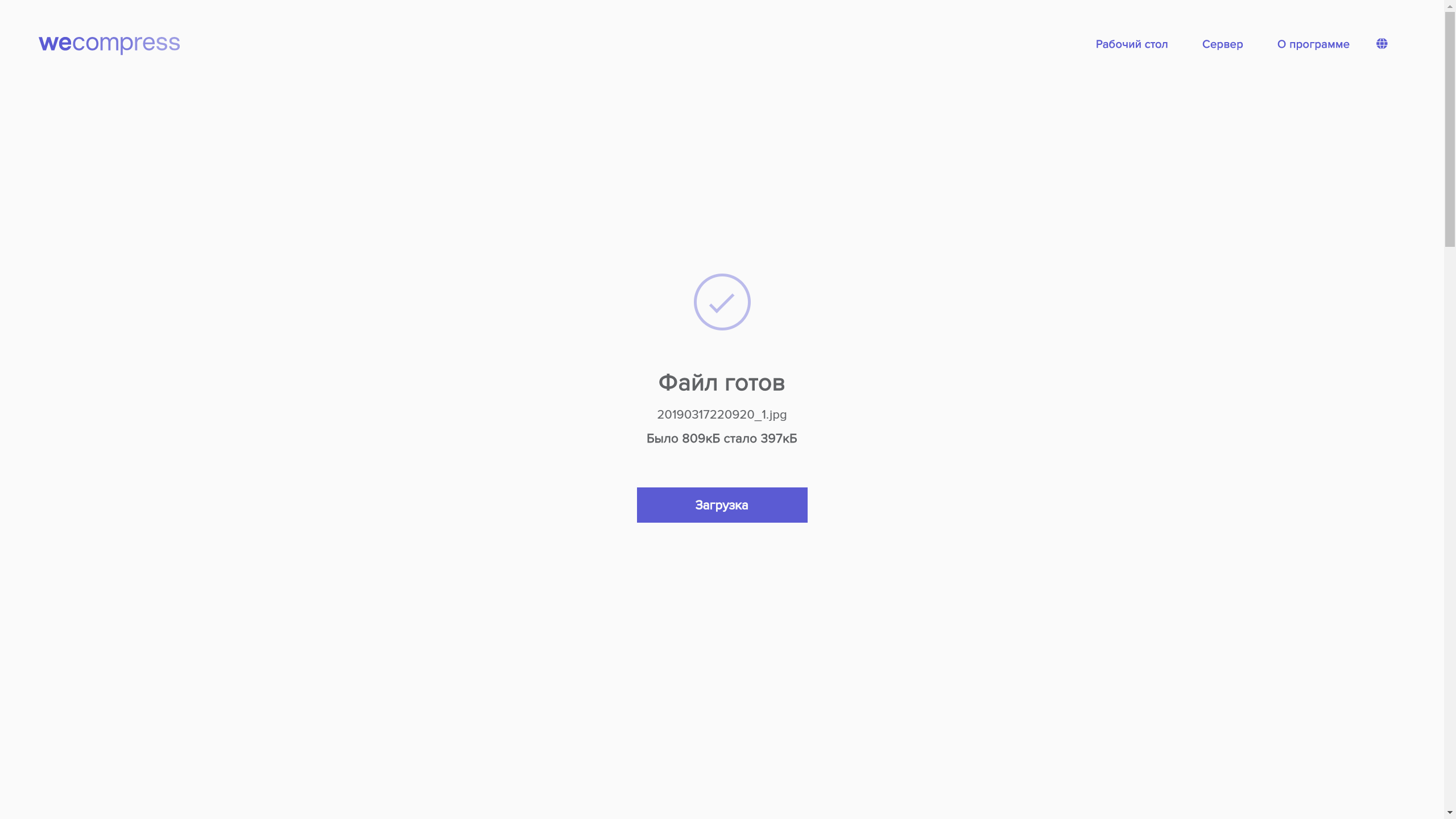
Рисунок 1.1 – Скриншот сайта “[www.wecompress.com](https://www.wecompress.com/ru/)” до сжатия файла

Рисунок 1.2 – Скриншот сайта “[www.wecompress.com](https://www.wecompress.com/ru/)” после сжатия файла

*Архиватор «WinRAR»*.

Плюсы:

* Разработчики поддерживают продукт и регулярно выпускают обновления;
* Создание архивов [RAR](https://ru.wikipedia.org/wiki/RAR) (с выбором формата — RAR4 или RAR5) и [ZIP](https://ru.wikipedia.org/wiki/ZIP) 2.0, их обновление и проверка целостности.
* Распаковка архивов [RAR](https://ru.wikipedia.org/wiki/RAR), а также [ARJ](https://ru.wikipedia.org/wiki/ARJ), [bz2](https://ru.wikipedia.org/wiki/Bzip2), [CAB](https://ru.wikipedia.org/wiki/CAB), [GZ](https://ru.wikipedia.org/wiki/GZ), [ISO](https://ru.wikipedia.org/wiki/ISO-%D0%BE%D0%B1%D1%80%D0%B0%D0%B7), [JAR](https://ru.wikipedia.org/wiki/JAR), [LZH](https://ru.wikipedia.org/wiki/LZH), [TAR](https://ru.wikipedia.org/wiki/TAR), [UUE](https://ru.wikipedia.org/wiki/UUE), [XZ](https://ru.wikipedia.org/wiki/XZ), [Z](https://ru.wikipedia.org/wiki/Z), [ZIP](https://ru.wikipedia.org/wiki/ZIP), [ZIPX](https://ru.wikipedia.org/wiki/WinZip), [7z](https://ru.wikipedia.org/wiki/7z), 001.
* Поддержка многоядерности на соответствующих [ЦП](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BD%D1%82%D1%80%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) при упаковке и распаковке.
* Полная поддержка имён файлов в [Юникоде](https://ru.wikipedia.org/wiki/%D0%AE%D0%BD%D0%B8%D0%BA%D0%BE%D0%B4).
* Создание [самораспаковывающихся](https://ru.wikipedia.org/wiki/%D0%A1%D0%B0%D0%BC%D0%BE%D1%80%D0%B0%D1%81%D0%BF%D0%B0%D0%BA%D0%BE%D0%B2%D1%8B%D0%B2%D0%B0%D1%8E%D1%89%D0%B8%D0%B9%D1%81%D1%8F_%D0%B0%D1%80%D1%85%D0%B8%D0%B2) (SFX) архивов;
* Возможность управления из [командной строки](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%BD%D0%B0%D1%8F_%D1%81%D1%82%D1%80%D0%BE%D0%BA%D0%B0);
* Позволяет выбрать метод и степень сжатия.

Минусы:

* Является платным по истечении пробного периода.

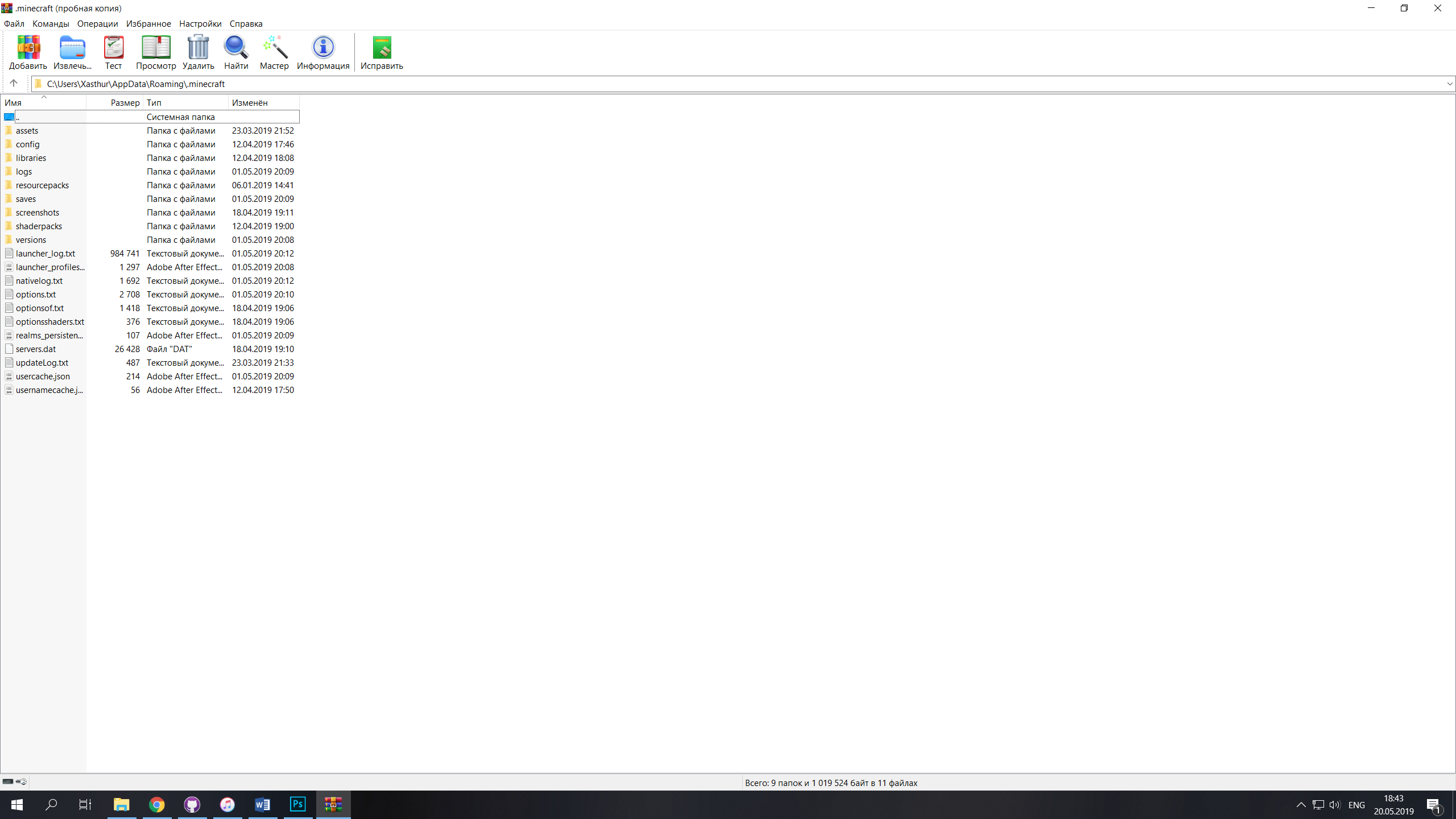
 Интерфейс программы показан на рисунках 1.3 и 1.4.

Рисунок 1.3 – Скриншот рабочего окна WinRAR

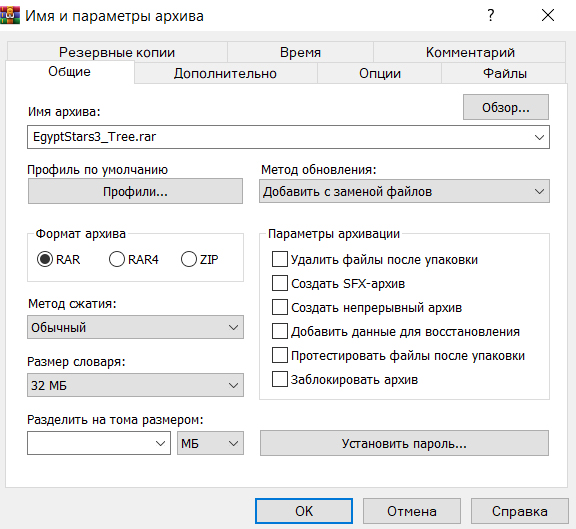


Рисунок 1.4 – Скриншот окна создания архива

**1.3 Описание популярных алгоритмов сжатия**

*Алгоритм сжатия* [*RLE*](http://en.wikipedia.org/wiki/Run-length_encoding) *(Кодирование длин серий)*.

RLE— [алгоритм](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) [сжатия данных](https://ru.wikipedia.org/wiki/%D0%A1%D0%B6%D0%B0%D1%82%D0%B8%D0%B5_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), заменяющий повторяющиеся символы (серии) на один символ и число его повторов. Серией называется последовательность, состоящая из нескольких одинаковых символов. При кодировании строка одинаковых символов, составляющих серию, заменяется строкой, содержащей сам повторяющийся символ и количество его повторов.

После применения алгоритма RLE строка ABCABCABCDDDFFFFFF превратится в 1A1B1C1A1B1C1A1B1C3D6F.

Допустим, реализация метода RLE для записи длин серий использует переменную символьного типа «Char». В такую переменную можно записать числа от 0 до 255 включительно. Но функция Chr(x) позволяет записать символ с кодом от 0 до 127 включительно. Как же быть, если длина серии равна 128 символам и более? В этом случае серию разделяют на части так, чтобы длина части не превышала 127 символов. Например, серия, состоящая из 256 символов «A», будет закодирована следующей строкой (256=127+127+2): 127A127A2A

Плюсы:

* Один из наиболее быстрых алгоритмов сжатия;
* Эффективен для сжатия текстовых файлов с длинными сериями повторяющихся символов;
* Эффективен для сжатия простых графических изображений, таких как [иконки](https://ru.wikipedia.org/wiki/%D0%97%D0%BD%D0%B0%D1%87%D0%BE%D0%BA_(%D1%8D%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B3%D0%BE_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81%D0%B0)) и графические рисунки;
* Звуковые данные, которые имеют длинные последовательные серии байт (такие как низкокачественные [звуковые семплы](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%BF%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) могут быть сжаты с помощью RLE после того, как к ним будет применено [Дельта-кодирование](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BB%D1%8C%D1%82%D0%B0-%D0%BA%D0%BE%D0%B4%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5).

Минусы:

* Плохо подходит для изображений с плавным переходом тонов, таких как фотографии.
* Не подходит для сжатия обычных текстов, где повторений незначительное количество.

*Алгоритм сжатия Хаффмана*.

Алгоритм Хаффмана — [жадный алгоритм](https://ru.wikipedia.org/wiki/%D0%96%D0%B0%D0%B4%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) оптимального [префиксного](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D1%84%D0%B8%D0%BA%D1%81%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) [кодирования](https://ru.wikipedia.org/wiki/%D0%AD%D0%BD%D1%82%D1%80%D0%BE%D0%BF%D0%B8%D0%B9%D0%BD%D0%BE%D0%B5_%D0%BA%D0%BE%D0%B4%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) алфавита с минимальной [избыточностью](https://ru.wikipedia.org/wiki/%D0%98%D0%B7%D0%B1%D1%8B%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C). В настоящее время используется во многих программах сжатия данных.

Этот метод кодирования состоит из двух основных этапов:

* Построение оптимального кодового дерева.
* Построение отображения код-символ на основе построенного дерева.

Идея алгоритма состоит в следующем: зная вероятности символов в сообщении, можно описать процедуру построения кодов переменной длины, состоящих из целого количества битов. Символам с большей вероятностью ставятся в соответствие более короткие коды. Коды Хаффмана обладают свойством [префиксности](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D1%84%D0%B8%D0%BA%D1%81%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4" \o "Префиксный код) (то есть ни одно кодовое слово не является префиксом другого), что позволяет однозначно их декодировать.

Плюсы:

* Очень эффективен при небольшом количестве уникальных символов;

Минусы:

* Для восстановления содержимого сжатого сообщения декодер должен знать таблицу частот, которой пользовался кодер. Следовательно, длина сжатого сообщения увеличивается на длину таблицы частот, которая должна посылаться впереди данных, что может свести на нет все усилия по сжатию;
* В процессе работы алгоритма сжатия вес узлов в дереве кодирования Хаффмана неуклонно растет, из-за чего при очень большом количество уникальных символов сжатие может быть неэффективно.

*Алгоритм сжатия LZ77*.

LZ77 – один из наиболее известных алгоритмов сжатия без потерь из семейства LZ\*. Использует словарный метод. В отличие от других методов уменьшения избыточности, таких как [RLE](https://ru.wikipedia.org/wiki/RLE) и [арифметическое сжатие](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D0%B8%D1%84%D0%BC%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D1%81%D0%B6%D0%B0%D1%82%D0%B8%D0%B5). LZ77 является алгоритмом со «скользящим окном», что эквивалентно неявному использованию словарного подхода, впервые предложенного в LZ78.

В кодируемых строках часто содержатся совпадающие длинные подстроки. Идея, лежащая в основе LZ77, заключается в замене повторений на ссылки на позиции в тексте, где такие подстроки уже встречались.

Информацию о повторении можно закодировать парой чисел - смещением назад от текущей позиции (offset) и длиной совпадающей подстроки (length). В таком случае, например, строка pabcdeqabcdepabcdeqabcde может быть представлена как pabcdeq⟨6,5⟩. Выражение ⟨6,5⟩ означает «вернись на 6 символов назад и выведи 5 символов».

Алгоритм LZ77 кодирует ссылки блоками из трёх элементов ⟨offset,length,next⟩. В дополнение к двум уже описанным элементам, новый параметр (next) означает первый символ после найденного совпадающего фрагмента. Если алгоритму не удалось найти совпадение, то считается, что offset=length=0.

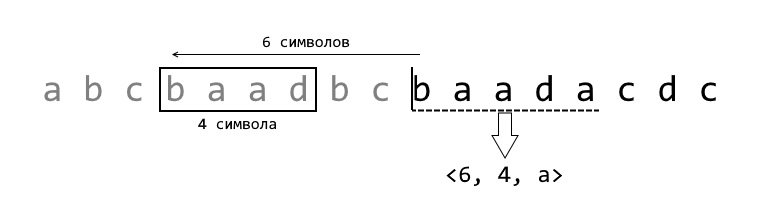
На рисунке 1.5 представлен принцип работы LZ77

Рисунок 1.5 – Принцип работы LZ77

Для декодирования LZ77 необходимо пройти по уже раскодированной строке назад, вывести необходимую последовательность, затем следующий символ.

Плюсы:

* Эффективен как для длинных серий символов, так и для строк, где повторения встречаются не в виде длинной последовательности символов.

Минусы:

* Невозможность кодирования подстрок, отстоящих друг от друга на расстоянии, большем длины словаря
* Длина подстроки, которую можно закодировать, ограничена размером буфера
* Малая эффективность при кодировании незначительного объёма данных

*Алгоритм сжатия Deflate*.

Deflate — это [алгоритм](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) [сжатия без потерь](https://ru.wikipedia.org/wiki/%D0%A1%D0%B6%D0%B0%D1%82%D0%B8%D0%B5_%D0%B1%D0%B5%D0%B7_%D0%BF%D0%BE%D1%82%D0%B5%D1%80%D1%8C), использующий комбинацию алгоритмов [LZ77](https://ru.wikipedia.org/wiki/LZ77) и [Хаффмана](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A5%D0%B0%D1%84%D1%84%D0%BC%D0%B0%D0%BD%D0%B0).

Компрессия выполняется в два этапа:

* замена повторяющихся строк указателями (алгоритм LZ77);
* замена символов новыми символами, основываясь на частоте их использования (алгоритм Хаффмана).

Плюсы:

Наиболее эффективен при сжатии строки с большим количеством повторений.

Минусы:

Неэффективен для строк с малым количеством повторений.

**1.4. Формирование требований к проектируемому программному средству**

Подробно изучив алгоритмы сжатия данных и программы, позволяющие это сделать, я решил что должна из себя представлять программа, разработанная мною на языке Delphi в среде программирования «Embarcadero Delphi 10.3»:

1. математический функционал:

* Программа должна уметь сжимать файлы, используя алгоритмы RLE, Хаффмана, LZ77 и Deflate;
* Программа должна уметь разжимать файлы, которые ею были сжаты;
* Программа должна уметь подсчитывать время выполнения алгоритма и считывать размер сжатого файла.

1. прочие функциональные требования к программному средству:

* Программа должна сама определять уже сжатый файл и предлагать его разжать соответствующим алгоритмом;
* Построение графиков сравнения времени выполнения и степени сжатия выбранных алгоритмов.

1. требования для удобства эксплуатации:

* программа доступна без интернета;
* простой интуитивно понятный интерфейс.

**1.5 Входные данные**

Входными данными для проектируемой программы может быть любой файл, нуждающийся в сжатии. Но важно, чтобы файл не был изначально сжат, потому что тогда нету никакого смысла в том, чтобы сжимать его еще раз. Для разжатия входной файл должен был ранее сжат данной программой, так как планируемая структура в сжатом файле не может быть построена в других программах или получена случайно.

**2. АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ**

**2.1** **Разработка используемых данных**

1) Соотношение алгоритма сжатия и расширения сжатого файла:

RLE - .xrle

Алгоритм Хаффмана - .xhfm

LZ77 - .xlz77

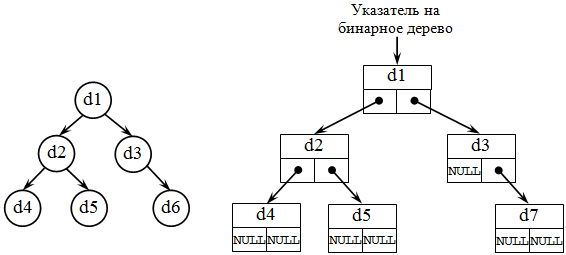
Deflate - .xdfl

2) Для хранения однотипной информации в большинстве случаев будет использоваться динамический массив.

3) Для реализации алгоритма Хаффмана потребуется использовать бинарное дерево.

Также в алгоритме Хаффмана будет использован динамический массив, состоящий из элементов типа THuffArrayElement.

Структура бинарного дерева и типа THuffArrayElement приведены на рисунках 2.1 и 2.2 соответственно.

Рисунок 2.1 – Структура бинарного дерева

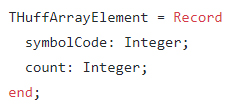


Рисунок 2.2 – Объявление типа THuffArrayElement

**2.2** **Анализ требований к программному средству и разработка фф функциональных требований**

В результате анализа требований к программному средству были составлены следующие функциональные требования:

* Корректность выполнения сжатия и декомпрессии с использованием следующих алгоритмов:

1. RLE;
2. Алгоритм Хаффмана;
3. LZ77;
4. Deflate;

* Поддержка открытия файлов с любым расширением
* Предпросмотр открытого файла
* Вывод результата декомпрессии в файл формата .txt
* Возможность не создавать не диске сжатые файлы
* Возможность выбрать любую комбинацию алгоритмов сжатия для сравнения
* Корректность построения графиков
* Цена деления на графике должна зависеть от результатов сжатия
* Возможность масштабировать графики

**2.3 Разработка алгоритма программного средства**

Изначально вниманию пользователя представляется начальная страница, где для продолжения работы программы ему необходимо нажать на кнопку «Open file». После открытия файла программа по его расширению определяет, был ли файл сжат данной программой в прошлом. Если программа определяет, что файл был сжат ею, то активируется функционал, позволяющий преобразовать данный файл в исходный. В противном случае активируется функционал, позволяющий сжать данный файл и проанализировать использованные алгоритмы.

Функционал сжатия:

При нажатии на кнопку Compress будет произведено сжатие выбранного файла выбранными алгоритмами.

Если стоит галочка на пункте “Analyse”, то после завершения сжатия всеми выбранными алгоритмами будут построены графики, с помощью которых можно сравнить эффективность каждого алгоритма на выбранном файле.

Если стоит галочка на пункте “Export comp.”, то после завершения сжатия всеми выбранными алгоритмами файлы, созданные в процессе сжатия, удалены не будут.

Функционал декомпрессии:

При нажатии на кнопку Decompress(\*алгоритм, использованный для сжатия\*) будет произведена декомпрессия файла с использованием соответствующего алгоритма.

Если стоит галочка на пункте “Export as .txt”, то ,после завершения декомпрессии соответствующим алгоритмом, файл, созданный в процессе декомпрессии, удален не будет.

**3. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

**3.1 Общая схема программы**

Схема алгоритма программы представлен в *приложении 1*. Также он выполнен на листе, формата А1.

Словесный алгоритм:

1. Программа начинается с создания пользовательского интерфейса
2. У пользователя есть возможность открыть файл
3. Программа определяет, был ли данный файл сжат ею ранее
4. Если программа определяет файл как сжатый ею ранее, то перейти к пункту 11
5. Программа определила файл как неизвестный, открывается возможность его сжать
6. Происходит выбор алгоритмов сжатия
7. Происходит сжатие файлов соответствующим алгоритмом и создание сжатых файлов
8. Если на пункте “Analyse” стояла галочка, то происходит вывод графиков, позволяющих сравнить выбранный алгоритмы по степени сжатия и времени
9. Если на пункте “Export comp.” не стояла галочка, то файлы, созданные в процессе сжатия, удаляются.
10. Перейти к пункту 2
11. Программа определила файл как сжатый ею ранее, открывается возможность его разжать
12. Происходит декомпрессия выбранного файла
13. Если на пункте “Export as .txt” не стояла галочка, то файлы, созданные в процессе декомпрессии, удаляются.
14. Перейти к пункту 2

**3.2 Алгоритм сжатия RLE**

(Блок-схема алгоритма в *приложении 2*)

Словесный алгоритм:

1. Считывание текущего символа, если не достигли конца строки
2. Подсчет длины последовательности одинаковых символов до достижения длины последовательности равной 127 символов
3. Запись в файл длину последовательности и сам символ
4. Перейти к пункту 1

**3.3 Алгоритм сжатия Хаффмана**

(Блок-схема алгоритма в *приложении 3*)

При сжатии алгоритмом Хаффмана в начало файла должна быть помещена информация для восстановления дерева при декомпрессии(символы, записанные в порядке убывания их встречаемости). Данная таблица отделяется от сжатой информацией 2мя символами ESC(код 27). Также обрабатывается ситуация, когда в конце таблицы находится 3 символа ESC. Данная ситуация возможно в том случае, если в самом тексте встречается символ ESC и он является самым редким.

Также количество “бит”, записанных в строку, не всегда кратно 7. Последний “байт” дополняется нужным количеством 0, а в конец всей строки пишется количество 0, которое потребовалось добавить.

Словесный алгоритм:

1. Создание массива с данными о количестве каждого символа на основе полученной строки
2. Сортировка полученного массива по встречаемости
3. Построение дерева на основе полученного массива
4. Преобразование строки на основе построенного дерева
5. Запись в файл информации для восстановления дерева
6. Запись в файл преобразованной строки
7. Очистка дерева

**3.4 Алгоритм сжатия LZ77**

(Блок-схема алгоритма в *приложении 4*)

Данная реализация алгоритма LZ77 имеет буфер в 127 символов. Используется именно 127 символов, т.к функция Chr(x) возвращает символ с номером от 0 до 127.

Словесный алгоритм:

1. Считывание очередного символа, если не достигнут конец строки
2. Сравнение текущей последовательности с буфером. Если не найдено общей части, перейти к пункту 5
3. Добавление символа в последовательность, если не достигнут конец строки
4. Сохранение информации о смещении и длине повторяющейся части
5. Перейти к пункту 2
6. Запись в файл информации о смещении, длине повторяющейся части и символе, следующем за повторяющейся частью.
7. Очистка строки, которая содержит текущую последовательность.
8. Перейти к пункту 1

**3.5 Алгоритм сжатия Deflate**

(Блок-схема алгоритма в *приложении 5*)

Сжатие алгоритмом Deflate использует комбинацию LZ77 и алгоритма Хаффмага, которые были уже реализованы в программе.

Словесный алгоритм:

1. Преобразование строки с использованием алгоритма LZ77
2. Преобразование строки с использованием алгоритма Хаффмана

**3.6 Алгоритм декомпрессии файла, сжатого RLE**

(Блок-схема алгоритма в *приложении 6*)

Декомпрессия происходит путем последовательного считывания 2х символов. Код первого означает длину последовательности, состоящей из второго символа.

Словесный алгоритм:

1. Считывается последовательность из 2х символов, если не достигнут конце строки
2. Запись в строку 2го символа из последовательности количество раз, равное коду 1го символа.
3. Перейти к пункту 1

**3.7 Алгоритм декомпрессии файла, сжатого алгоритмом Хаффмана**

(Блок-схема алгоритма в *приложении 7*)

Создание массива происходит путем добавления в массив всех символов, находящихся в строке до последовательности символов, состоящей из 2х символов ESC(код 27). Если встречается последовательность состоящая из 3х символов ESC, то данный символ также заносится в массив. Далее на основе данного массива строится дерево, которое полностью повторяет дерево, полученное при сжатии. После этого происходит преобразование строки таким образом, что каждый символ записывается как его двоичная запись. С конца строки удаляются лишние 0, которые потребовались при сжатии. И происходит преобразование сжатого текста в исходный на основе построенного дерева.

Словесный алгоритм:

1. Создание массива с данными о количестве каждого символа на основе таблицы в начале файла
2. Построение дерева на основе полученного массива
3. Преобразование строки на основе построенного дерева
4. Запись в файл преобразованной строки
5. Очистка дерева

**3.8 Алгоритм декомпрессии файла, сжатого LZ77**

(Блок-схема алгоритма в *приложении 8*)

Данная реализация алгоритма LZ77 имеет буфер в 127 символов. Используется именно 127 символов, т.к функция Chr(x) возвращает символ с номером от 0 до 127.

Словесный алгоритм:

1. Происходит считывание 3х символов из строки, если не достигнут её конец
2. Если 1 и 2 символы имеют код не равный 0, то происходит смещение в буфере и выводится требуемая последовательность
3. Выводится 3 символ
4. Перейти к пункту 1

**3.9 Алгоритм декомпрессии файла, сжатого Deflate**

(Блок-схема алгоритма в *приложении 9*)

Декомпрессия файла, сжатого алгоритмом Deflate, происходит путем преобразования строки из файла сначала с использованием алгоритма Хаффмана, а потом LZ77.

Словесный алгоритм:

1. Применение к строке алгоритма, позволяющего разжать строку, сжатую с использованием алгоритма Хаффмана.
2. Применение к строке алгоритма, позволяющего разжать строку, сжатую с использованием алгоритма LZ77.

**3.10 Алгоритм построения графиков**

(Блок-схема алгоритма в *приложении 10*)

Графики строятся на основе данных, которые были переданы в динамическом массиве после сжатия. Для добавления алгоритма к сравнению требуется увеличить длину массива и добавить соответствующую информацию. После добавления в графике появится новое поле, характеризующее добавленный алгоритм.

Словесный алгоритм:

1. Получение информации об использовавшихся в сжатии алгоритмах
2. Построение осей
3. Построение штрихов
4. Подсчет и вывод цены деления около штрихов
5. Построения столбцов на графике с их названиями
6. **СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА**

**4.1 Программирование отдельных модулей**

4.1.1 Функция получения размера файла:

(Блок-схема алгоритма в *приложении 1*)

function GetFileSize(FileName: String): Int64;

var FS: TFileStream;

begin

FS := TFileStream.Create(FileName, fmOpenRead);

Result := FS.Size;

FS.Free;

end;

4.1.2 Функция преобразования строки, состоящей из 0 и 1 в число:

(Блок-схема алгоритма в *приложении 1*)

function StrBinToInt(s: String; pos,l:integer):Integer;

var

i: integer;

value: integer;

begin

value:=0;

l:=l+(pos-1);

for i:=pos to l do

begin

if s[i]='1' then

begin

value:=value+round(power(2,abs(i-l)));

end;

end;

StrBinToInt:=value;

end;

4.1.3 Процедура вызова сжатия:

(Блок-схема алгоритма в *приложении 1*)

procedure CompressionStart(s:string;z:integer;var newPath,newName:String);

var

p: Integer;

begin

newPath:=ExtractFilePath(Form1.OpenDialog1.FileName);

newName:=ExtractFileName(Form1.OpenDialog1.FileName);

p:=pos(ExtractFileExt(Form1.OpenDialog1.FileName),newName);

if p<>0 then

begin

Delete(newName,p,length(ExtractFileExt(Form1.OpenDialog1.FileName)));

end;

case z of

1:

begin

newName:=newName+'.xrle';

newPath:=newPath+newName;

RLECompress(s,newPath);

write('RLE ');

end;

2:

begin

newName:=newName+'.xhfm';

newPath:=newPath+newName;

HUFFCompress(s,newPath);

write('Huffman ');

end;

3:

begin

newName:=newName+'.xlz77';

newPath:=newPath+newName;

LZ77Compress(s,newPath);

write('LZ77 ');

end;

4:

begin

newName:=newName+'.xdfl';

newPath:=newPath+newName;

DFLCompress(s,newPath);

write('Deflate ');

end;

end;

s:=System.IOUtils.TFile.ReadAllText(newPath);

writeln('Compression Result:');

if outputFiles then

begin

writeln(s);

end;

writeln;

printFileInfo(newPath);

writeln;

writeln;

end;

4.1.4 Функция преобразования числа в строку, содержащую двоичную запись данного числа:

(Блок-схема алгоритма в *приложении 1*)

function IntToBin7(d: Integer): string;

var

x: Integer;

bin: string;

begin

bin := '';

for x := 1 to 7 do

begin

if Odd(d) then

begin

bin := '1' + bin

end else

begin

bin := '0' + bin;

end;

d := d shr 1;

end;

Delete(bin, 1, 7 \* ((Pos('1', bin) - 1) div 7));

Result := bin;

end;

4.1.5 Процедура начала декомпрессии:

(Блок-схема алгоритма в *приложении 1*)

procedure StartDecompression(path:String;z:integer);

var

p: Integer;

s:string;

newPath,newName: String;

begin

AllocConsole;

Form1.Memo1.Clear;

newPath:=ExtractFilePath(path);

newName:=ExtractFileName(path);

p:=pos(ExtractFileExt(path),newName);

if p<>0 then

begin

Delete(newName,p,length(ExtractFileExt(path)));

end;

case z of

1:

begin

newName:='RLE\_' + newName;

newName:=newName+'.txt';

newPath:=newPath+newName;

s:=System.IOUtils.TFile.ReadAllText(path);

decompressRLE(s,newPath);

write('RLE ');

end;

2:

begin

newName:='HFM\_' + newName;

newName:=newName+'.txt';

newPath:=newPath+newName;

s:=System.IOUtils.TFile.ReadAllText(path);

decompressHFM(s,newPath);

write('Huffman ');

end;

3:

begin

newName:='LZ77\_' + newName;

newName:=newName+'.txt';

newPath:=newPath+newName;

s:=System.IOUtils.TFile.ReadAllText(path);

decompressLZ77(s,newPath);

write('LZ77 ');

end;

4:

begin

newName:='DFL\_' + newName;

newName:=newName+'.txt';

newPath:=newPath+newName;

s:=System.IOUtils.TFile.ReadAllText(path);

decompressDFL(s,newPath);

write('Deflate ');

end;

end;

s:=System.IOUtils.TFile.ReadAllText(newPath);

writeln('Decompression Result:');

if outputFiles then

begin

writeln(s);

end;

Form1.Memo1.Text:=s;

writeln;

printFileInfo(newPath);

writeln;

writeln;

if Form1.CheckBoxExportTXT.Checked=false then

begin

DeleteFile(newPath);

end;

end;

**4.2 Список все процедур, их параметры и назначение:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| Unit MainForm.pas | | | | |
| GetFileSize | Процедура получения размера файла | function GetFileSize(FileName: String): Int64; | FileName | Имя файла |
| RLECompressString | Сжатие строки алгоритмом RLE | function RLECompressString(str: String):String; | str | Сжимаемая строка |
| RLECompress | Начало сжатия RLE | procedure RLECompress(s:string; newPath:String); | s | Рабочая строка |
| newPath | Путь сохранения |
| DisposeHuffTree | Очистка дерева Хаффмана | procedure DisposeHuffTree(var head: HuffTreePointer); | head | Указатель на голову дерева |
| HUFFGetCount | Заполнение массива символов | function HUFFGetCount(var arr:THuffArray; str: String):integer; | str | Строка для подсчета |
| HUFFSort | Сортировка массива | procedure HUFFSort(var arr:THuffArray); | arr | Массив сортируемый |
| HUFFCreateTree | Создание дерева Хаффмана | procedure HUFFCreateTree(var head: HuffTreePointer; arr:THuffArray); | head | Указатель на голову дерева |
| arr | Массив с вероятностями |
| HUFFGetBSymbol | Преобразование символа в строку используя дерево Хаффмана | function HUFFGetBSymbol(head:HuffTreePointer; c:Char):String; | head | Указатель на голову дерева |
| c | Символ |
| StrBinToInt | Преобразование последовательности 0 и 1 в число десятичной сс | function StrBinToInt(s: String; pos,l:integer):Integer; | s | Строка сжатия |
| pos | Начальная позиция |
| l | Длина последовтельности |
| HUFFStringBinaryToChar | Запуск преобразования строки из 0 и 1 в символы | function HUFFStringBinaryToChar(s:String):String; | s | Строка сжатия |
| HUFFCompressString | Сжатие строки алгоритмом Хаффмана | function HUFFCompressString(head:HuffTreePointer; s: String):String; | head | Указатель на голову дерева |
| s | Строка сжатия |
| HUFFCompress | Начало сжатия Хаффмана | procedure HUFFCompress(s:String; newPath:String); | s | Строка |
| newPath | Путь сохранения |
| LZ77CompressString | Сжатие строки методом LZ77 | function LZ77CompressString(s:String):String; | s | Строка сжатия |
| LZ77Compress | Начало сжатия LZ77 | procedure LZ77Compress(s:String; newPath:String); | s | Рабочая строка |
| newPath | Путь сохранения |
| DFLCompress | Сжатие методом Deflate | procedure DFLCompress(s:String; newPath:String); | s | Рабочая строка |
| newPath | Путь сохранения |
| CompressionStart | Запуск сжатия в зависимости от выбранного типа | procedure CompressionStart(s:string;z:integer;var newPath,newName:String); | s | Строка для сжатия |
| z | Тип сжатия |
| newPath | Путь сохранения |
| newName | Новое имя |
| decompressRLEString | Декомпрессия RLE строки | function decompressRLEString(str:String):String; | str | Строка разжатия |
| decompressRLE | Декомпрессия RLD | procedure decompressRLE(s,newPath:String); | s | Рабочая строка |
| newPath | Путь сохранения |
| IntToBin7 | Преобразование числа в строку, содержащую двоичную запись данного числа: | function IntToBin7(d: Integer): string; | d | Преобразуемое число |
| HUFFCharToStringBinary | Преобразование символов в строку 0 и 1 | function HUFFCharToStringBinary(s:string):String; | s | Строка разжатия |
| getSymbolFromHuffTree | Поиск символа в дереве Хаффмана | function getSymbolFromHuffTree(head:HuffTreePointer; buffSequence:string):Integer; | head | Указатель на голову дерева |
| buffSequence | Последовательность |
| decodeHuffString | Расшифровка строки Хаффмана | function decodeHuffString(head:HuffTreePointer; s:string):String; | head | Указатель на голову дерева |
| s | Строка разжатия |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| decompressHFM | Декомпрессия строки сжатой алгоритмом Хаффмана | procedure decompressHFM(s:string;newPath:String); | s | Рабочая строка |
| newPath | Путь сохранения |
| decompressLZ77String | Декомпрессия строки сжатой алгоритмом LZ77 | function decompressLZ77String(s:string):String; | s | Строка разжатия |
| decompressLZ77 | Начало декомпрессии LZ77 | procedure decompressLZ77(s:string;newPath:String); | s | Рабочая строка |
| newPath | Путь сохранения |
| decompressDFL | Декомпрессия строки сжатой методом Deflate | procedure decompressDFL(s:string;newPath:String); | s | Рабочая строка |
| newPath | Путь сохранения |
| StartDecompression | Начало декомпрессии в зависимости от выбранного метода | procedure StartDecompression(path:String;z:integer); | path | Путь к файлу |
| z | Тип разжатия |
| Unit ChartForm.pas | | | | |
| LoadGraphData | Процедура загрузки данных в переменные | procedure LoadGraphData(TimeArr:TGraphDataArray;SizeArr:TGraphDataArray); | TimeArr | Массив времени |
| SizeArr | Массив размеров |
| drawDashes | Рисование штрихов на осях | procedure drawDashes(I:TImage;offsetX,offsetY:Integer); | I | Изображение TImage |
| offsetX | Смещение по X |
| offsetY | Смещение по Y |
| calcY | Подсчет Y | function calcY(I:TImage;offsetY,min,max:Integer;Element:TAlgRec):Integer; | I | Изображение TImage |
| offsetX | Смещение по X |
| offsetY | Смещение по Y |
| Min | Минимум на оси |
| Мax | Максимум на оси |
| Element | Информация о текущем столбце |
| drawMinMax | Вывод значений делений | procedure drawMinMax(I:TImage; offsetX,offsetY,min,max,T:Integer); | I | Изображение TImage |
| offsetX | Смещение по X |
| offsetY | Смещение по Y |
| Min | Минимум на оси |
| Мax | Максимум на оси |
| T | Тип графика |
| drawPillars | Построение столбцов | procedure drawPillars(I:TImage; offsetX,offsetY:Integer; DataArr:TGraphDataArray;T:Integer); | I | Изображение TImage |
| offsetX | Смещение по X |
| offsetY | Смещение по Y |
| DataArr | Массив с данными |
| T | Тип графика |
| drawAxises | Построение осей | procedure drawAxises(I:TImage;DA:TGraphDataArray;T:Integer); | I | Изображение TImage |
| DA | Массив с данными |
| T | Тип графика |
| drawSizeGraph | Построение графика размера | procedure drawSizeGraph(I:TImage;SA:TGraphDataArray); | I | Изображение TImage |
| SA | Массив размеров |
| drawTimeGraph | Построение графика времени | procedure drawTimeGraph(I:TImage;TA:TGraphDataArray); | I | Изображение TImage |
| TA | Массив времени |

1. **ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ**

Для того, чтобы соответствовать требованиям к проектируемому программному средству, необходимо, чтобы оно прошло некоторое тестирование, способное выявить его недостатки.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  Теста | Тестируемая функциональность | Последовательность  действий | Ожидаемый результат | Полученный результат |
| 1 | Сжатие алгоритмом Хаффмана с выводом файла | Запустить программы. Открыть сжимаемый файла. Поставить галочку на “Export comp.” и “Huffman” Нажать на кнопку Compress | Будет создан сжатый файл с расширением .xhfm. После чего разжат программой. Содержание начального файла должно полностью совпадать с тем, что получилось после декомпрессии. | Тест пройден |
| 2 | Сжатие алгоритмом RLE с выводом файла | Запустить программы. Открыть сжимаемый файла. Поставить галочку на “Export comp.” и “RLE” Нажать на кнопку Compress | Будет создан сжатый файл с расширением .xrle. После чего разжат программой. Содержание начального файла должно полностью совпадать с тем, что получилось после декомпрессии. | Тест пройден |
| 3 | Сжатие алгоритмом LZ77 с выводом файла | Запустить программы. Открыть сжимаемый файла. Поставить галочку на “Export comp.” и “LZ77” Нажать на кнопку Compress | Будет создан сжатый файл с расширением .xlz77. После чего разжат программой. Содержание начального файла должно полностью совпадать с тем, что получилось после декомпрессии. | Тест пройден |
| 4 | Сжатие алгоритмом Deflate с выводом файла | Запустить программы. Открыть сжимаемый файла. Поставить галочку на “Export comp.” и “Deflate” Нажать на кнопку Compress | Будет создан сжатый файл с расширением .xdfl. После чего разжат программой. Содержание начального файла должно полностью совпадать с тем, что получилось после декомпрессии. | Тест пройден |
| 5 | Разжатие алгоритмом Хаффмана с выводом файла | Запустить программы. Открыть разжимаемый файла с расширением .xhfm. Поставить галочку на “Export as .txt” Нажать на кнопку Decompress | Будет создан файл с расширением .txt. Содержание начального файла должно полностью совпадать с тем, что получилось после декомпрессии | Тест пройден |
| 6 | Разжатие алгоритмом RLE с выводом файла | Запустить программы. Открыть разжимаемый файла с расширением .xrle. Поставить галочку на “Export as .txt” Нажать на кнопку Decompress | Будет создан файл с расширением .txt. Содержание начального файла должно полностью совпадать с тем, что получилось после декомпрессии | Тест пройден |
| 7 | Разжатие алгоритмом LZ77 с выводом файла | Запустить программы. Открыть разжимаемый файла с расширением .xlz77. Поставить галочку на “Export as .txt” Нажать на кнопку Decompress | Будет создан файл с расширением .txt. Содержание начального файла должно полностью совпадать с тем, что получилось после декомпрессии | Тест пройден |
| 8 | Разжатие алгоритмом Deflate с выводом файла | Запустить программы. Открыть разжимаемый файла с расширением .xdfl. Поставить галочку на “Export as .txt” Нажать на кнопку Decompress | Будет создан файл с расширением .txt. Содержание начального файла должно полностью совпадать с тем, что получилось после декомпрессии | Тест пройден |
| 9 | Попытка сжатия без выбранных алгоритмов | Запустить программы. Открыть сжимаемый файла. Нажать на кнопку Compress | Ничего не произойдет | Тест пройден |
| 10 | Попытка сжатия 4мя алгоритмами с последующим выводом графиков | Запустить программы. Открыть сжимаемый файла. Поставить галочку на “Analyse” Нажать на кнопку Compress | После процесса сжатия открылось окно с построенными графиками | Тест пройден |
| 11 | Попытка сжатия без выбора алгоритмов с последующим выводом графиков | Запустить программы. Открыть сжимаемый файла. Поставить галочку на “Analyse” Нажать на кнопку Compress | Откроются графики, содержащие информацию только о несжатом файле | Тест пройден |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | Дополнительные сведения | | | |
|  | | | | Текстовые документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| БГУИР КП 1–40 01 01 013ПЗ | | | | Пояснительная записка | | | | 86 с. | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | | Графические документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| ГУИР 851002  013 ПД | | | | Схема программы | | | | Формат А1 | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  |  |  |  |  | БГУИР КП 1-40 01 01 013 Д1 | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | ПС «Матричный калькулятор»  Ведомость курсового  проекта |  | | | | Лист | Листов |
| Разраб. | | Плиска В.С. |  | 25.05.17 | Т |  | |  | 86 | 86 |
| Пров. | | Болтак С.В. |  | 06.06.17 | Кафедра ПОИТ  Гр. 651005 | | | | | |
|  | |  |  |  |