Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Основы алгоритмами и программирования

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

на тему

**«Анализ алгоритмов архивации данных»**

БГУИР КР 1-40 01 01 226 ПЗ

Студент: 851002 Цыбулько К. Д.

Руководитель:

асс. Фадеева Е.Е.

Минск 2019

Задание

# **СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc9464936)

[1.АНАЛИЗ ПРОТОТИПОВ, ЛИЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМУ ПРОГРАММНОМУ СРЕДСТВУ 7](#_Toc9464937)

[1.1 Анализ программных средств архивации 7](#_Toc9464938)

[1.1.1 Архиватор WinRAR 7](#_Toc9464939)

[1.1.2 Архиватор 7-Zip 8](#_Toc9464940)

[1.1.3 Архиватор WinZIP 9](#_Toc9464942)

[1.1.4 Архиватор PowerArchive 10](#_Toc9464943)

[1.1.5 Выводы 11](#_Toc9464944)

[1.2 Формирование требований к проектируемому программному средству 11](#_Toc9464945)

[2. МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФЦНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ 12](#_Toc9464946)

[2.1 Описание алгоритмов сжатия информации, положенных в основу курсового проекта 12](#_Toc9464947)

[2.1.1 Алгоритм Хаффмана 12](#_Toc9464948)

[2.1.2 Алгоритм LZW 14](#_Toc9464949)

[2.1.3 Алгоритм RLE 17](#_Toc9464950)

[2.2 Описание функциональных требований к программному средству 19](#_Toc9464951)

[2.3 Спецификация функциональных требований 20](#_Toc9464952)

[3. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 21](#_Toc9464953)

[3.1 Обобщённый алгоритм работы программного средства 21](#_Toc9464954)

[3.2 Алгоритм архивации/разархивации по Хаффману 21](#_Toc9464955)

[3.2.1 Проектирование структур данных для алгоритма Хаффмана 21](#_Toc9464956)

[3.2.2. Алгоритм архивирования Хаффмана 22](#_Toc9464957)

[3.2.3. Алгоритм разархивирования Хаффмана 23](#_Toc9464958)

[3.3 Алгоритм архивации/разархивации LZW 25](#_Toc9464959)

[3.3.1 Проектирование структур данных для LZW 25](#_Toc9464960)

[3.3.2. Алгоритм архивирования LZW 25](#_Toc9464961)

[3.3.3. Алгоритм разархивирования LZW 27](#_Toc9464962)

[3.4 Алгоритм архивации/разархивации RLE 28](#_Toc9464963)

[3.4.1 Проектирование структур данных для RLE 28](#_Toc9464964)

[3.4.2. Алгоритм архивирования RLE 28](#_Toc9464965)

[3.4.3. Алгоритм разархивирования RLE 29](#_Toc9464966)

[4. СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА 31](#_Toc9464967)

[4.1 Разработка формы программного средства 31](#_Toc9464968)

[4.2 Описание процедур для архивации методом RLE 32](#_Toc9464969)

[4.3 Описание процедур для архивации методом LZW 33](#_Toc9464970)

[4.4 Описание процедур для архивации методом Хаффмана 34](#_Toc9464971)

[5. ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА 37](#_Toc9464972)

[6. РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ 40](#_Toc9464973)

[7. АНАЛИЗ РАБОТЫ АЛГОРИТМОВ АРХИВАЦИИ 42](#_Toc9464974)

[7.1 Исследование коэффициента сжатия по типу архивируемой информации алгоритмом Хаффмана 44](#_Toc9464975)

[7.2 Исследование коэффициента сжатия по типу архивируемой информации алгоритмом RLE 44](#_Toc9464976)

[7.3 Исследование коэффициента сжатия по типу архивируемой информации алгоритмом LZW 45](#_Toc9464977)

[7.4 Исследование коэффициента сжатия для файлов формата txt 46](#_Toc9464978)

[7.5 Исследование коэффициента сжатия для файлов формата jpg 46](#_Toc9464979)

[7.6 Исследование коэффициента сжатия для файлов формата pdf 47](#_Toc9464980)

[7.7 Исследование времени архивации 47](#_Toc9464981)

[ЗАКЛЮЧЕНИЕ 49](#_Toc9464982)

[Список использованной литературы туры 50](#_Toc9464983)

[ПРИЛОЖЕНИE 1 51](#_Toc9464984)

[ПРИЛОЖЕНИE 2 52](#_Toc9464985)

# **ВВЕДЕНИЕ**

Архиватор — [программа](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0), осуществляющая [сжатие](http://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) и/или упаковку одного и более [файлов](http://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB) в [архив](http://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D0%B2_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) или серию архивов для удобства переноса или хранения, а также распаковку архивов [1].

Простейшие архиваторы просто последовательно объединяют (упаковывают) содержимое файлов в архив. Архив должен также содержать информацию об именах и длине оригинальных файлов для их восстановления, поэтому большинство архиваторов также сохраняют [метаданные](http://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%B0%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5) файлов, предоставляемые [операционной системой](http://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), такие, как время создания и права доступа. Архивы создают в следующих случаях:

-  при передаче файлов по почте или с помощью терминальных программ критичным является каждый килобайт, да и, кроме того, при пересылке большого числа файлов (особенно со сложной структурой каталогов) проще в письмо вложить всего один файл - архив;

- пропускная способность локальных сетей ограничена. При пересылке больших массивов информации по сети рекомендуется использовать архиваторы не только для уменьшения трафика, но и с целью упрощения конечной верификации: после копирования проще проверить корректность архива, нежели проверять сотни файлов на целостность, просматривая их содержимое;

- многие из нас сталкиваются с проблемой, когда на диск необходимо перенести очень сложную структуру каталогов. Однако, файловая система компакт-диска не допускает более чем восьмикратный уровень вложенности папок. Это было бы очень серьезной проблемой, если бы не архиваторы, позволяющие упаковывать всю структуру в один файл;

- отдельные архиваторы являются очень эффективными кодировщиками, позволяющими скрыть конфиденциальную информацию от чужих глаз, запаковав свои файлы и установив на архив пароль.

В основе любого способа сжатия лежит модель источника данных, или, точнее, [модель избыточности](http://ru.wikipedia.org/w/index.php?title=%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C_%D0%B8%D0%B7%D0%B1%D1%8B%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D0%B8&action=edit&redlink=1). Иными словами, для сжатия данных используются некоторые априорные сведения о том, какого рода данные сжимаются. Не обладая такими сведениями об источнике, невозможно сделать никаких предположений о преобразовании, которое позволило бы уменьшить объём сообщения. Модель избыточности может быть статической, неизменной для всего сжимаемого сообщения, либо строиться или параметризоваться на этапе сжатия (и восстановления). Методы, позволяющие на основе входных данных изменять модель избыточности информации, называются адаптивными. Неадаптивными являются обычно узкоспециализированные алгоритмы, применяемые для работы с данными, обладающими хорошо определёнными и неизменными характеристиками. Подавляющая часть достаточно универсальных алгоритмов являются в той или иной мере адаптивными [1].

Все методы сжатия данных делятся на два основных класса:

- [сжатие без потерь](http://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](http://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, обычно несущественными с точки зрения дальнейшего использования восстановленных данных. Сжатие без потерь обычно используется для передачи и хранения текстовых данных, компьютерных программ, реже — для сокращения объёма [аудио](http://ru.wikipedia.org/wiki/%D0%90%D1%83%D0%B4%D0%B8%D0%BE)- и [видеоданных](http://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%B4%D0%B5%D0%BE), [цифровых фотографий](http://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.

Существует много разных практических методов сжатия без потери информации, которые, как правило, имеют разную эффективность для разных типов данных и разных объемов. Однако, в основе этих методов лежат три теоретических алгоритма [2-4]:

* алгоритм RLE (Run Length Encoding);
* алгоритмы группы KWE(KeyWord Encoding);
* алгоритм Хаффмана.

Целью курсовой работы является разработка программного средства, которое осуществляет сжатие/восстановление информации на основе алгоритмов: Хаффмана, RLE, LZW и выполняет сравнение алгоритмов по: времени выполнения операций сжатия/восстановления, проверки целостности архивов, степени сжатия, чувствительности к случайным ошибкам.

Данная пояснительная записка содержит следующие основные разделы. В первом разделе выполнен анализ прототипов, литературных источников. Во втором разделе сформированы функциональные требования к проектируемому программному средству. Третий раздел посвящен проектированию программного средства и содержит схемы работы модулей программного средства. Четвертый раздел описывает этапы создания программного средства. Пятый раздел содержит набор тестов и сценарии тестирования. В шестом разделе описано руководство по установке и использованию. Завершающий раздел содержит собранные статистические данные работы анализатора при различных условиях и выводы.

# **1.АНАЛИЗ ПРОТОТИПОВ, ЛИЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМУ ПРОГРАММНОМУ СРЕДСТВУ**

## 1.1 Анализ программных средств архивации

Существует множество разных форматов сжатых контейнеров. При упаковке некоторых видов файлов их конечный размер может быть уменьшен на 90-95%. Все современные программы для работы со сжатыми библиотеками могут открывать распространенные типы архивов, могут устанавливать пароль на конечный объект и позволяют создавать самораспаковывающиеся контейнеры в виде исполняемого файла

На данный момент не существует архиватора, где пользователь мог бы сам выбрать алгоритм сжатия данных (за исключением встроенных утилит), а также проанализировать их по ряду критериев [1]. Поэтому в качестве прототипов будут использованы известные архиваторы, которые используют три алгоритма сжатия, указанных в задании на курсовой проект.

## 1.1.1 Архиватор WinRAR

**WinRAR** - является быстрым и мощным архиватором для использования на любом персональном компьютере, на котором установлена операционная система семейства Windows [6].

В данном архиваторе для сжатия данных используется алгоритм Хаффмана, однако, кроме данного алгоритма, в зависимости от настроек архивации параллельно используются и другие методы, для увеличения эффективности архивации.

Приложение работает с большим количеством форматов, среди которых самые распространенные **ZIP** и **RAR**. Делая архивацию файлов при помощи WinRAR, легко уменьшается их размер, путем сжатия содержимого самого файла.

К достоинствам архиватора стоит отнести: быструю скорость сжатия и маленький размер выходного файла; встраиваемость в оболочку, то есть программу не нужно запускать отдельно; большое количество поддерживаемых форматов (RAR, ZIP, BZ2, TAR, SFX и другие); в настройках можно указывать путь, по которому будут распаковываться файлы; архиватор позволяет просматривать состав архива без его предварительной распаковки (рис.1.1).

Если говорить о недостатках, то следует отметить, что для мультимедийных файлов эффективность архивирования намного меньше (Данный недостаток касается не только WinRAR). Так же недостатком являются ограничение по размеру архива (8Гб) и стоимость программы.

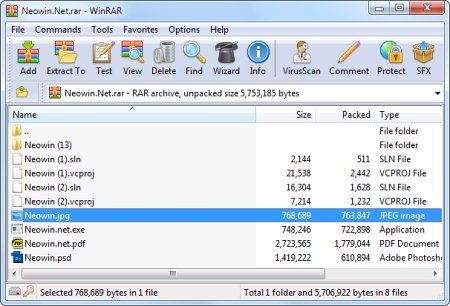


Рисунок 1.1 – Интерфейс WinRAR

## 1.1.2 Архиватор 7-Zip

7-Zip - свободный файловый архиватор с высокой степенью сжатия данных. Поддерживает несколько алгоритмов сжатия и множество форматов данных, включая собственный формат 7z c высокоэффективным алгоритмом сжатия LZMA [7].

Данный архиватор считается одним из лучших, благодаря таким возможностям, как создание самораспаковывающихся архивов 7z и формирование зашифрованных файлов. Приложение имеет мощный файловый менеджер и может встраиваться в контекстное меню проводника Windows. Данная программа поддерживает более семидесяти языков и полностью совместима с операционной системой Windows (рис. 1.2). Также она может работать в командной строке систем Linux/Unix [7].

К достоинствам архиватора стоит отнести:

- распространение на бесплатной основе;

- высокая степень сжатия (больше чем у WinRAR);

- достаточно высокая степень сжатия;

- способен разархивировать любой формат;

- создание самораспаковывающихся архивов 7z.

Следует отметить следующие недостатки:

- интерфейс требует привыкания;

- увеличенный размер самораспаковывающихся архивов SFX;

- не всегда хорошо работает с многотомными архивами, комментариями и информацией для восстановления архива.

## ÐÐ°ÑÑÐ¸Ð½ÐºÐ¸ Ð¿Ð¾ Ð·Ð°Ð¿ÑÐ¾ÑÑ ÐÑÑÐ¸Ð²Ð°ÑÐ¾Ñ 7-Zip

Рисунок 1.2 – Интерфейс 7-ZIP

## 1.1.3 Архиватор WinZIP

WinZIP – популярный многофункциональный архиватор с плотной интеграцией в оболочку Microsoft Windows [8].  Кроме стандартных опций для работы с архивами это приложение содержит функцию наложения «водяных знаков» на некоторые типы сжатых объектов, включает встроенный конвертер для преобразования офисных документов в PDF-файлы, может менять размер и другие параметры изображений (рис.1.3).

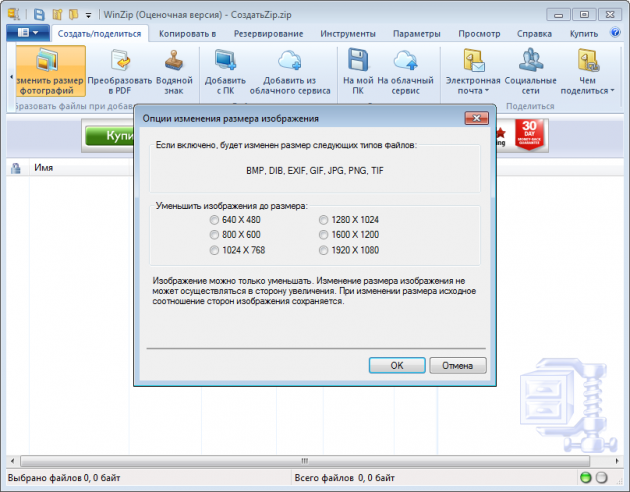


Рисунок 1.3 – Интерфейс WinZIP

В данном архиваторе при процессе архивации используется алгоритм LZMA для сжатия информации, который является аналогом LZW. Нужно понимать, что кроме данного алгоритма, в зависимости от настроек архивации параллельно используются и другие методы, для увеличения эффективности архивации, например, комбинация алгоритмов LZ77 и алгоритма Хаффмана.

К достоинствам следует отнести:

* многофункциональность;
* высокая скорость создания и качество сжатия;
* наложение «водяных знаков»;
* создание самораспаковывающихся архивов.

Недостатки:

* жесткая лицензионная политика (пробная версия WinZip доступна в течение двух недель);
* недружелюбный пользовательский интерфейс;
* три формата конечных контейнеров - ZIP, ZIPX и LHA.

## 1.1.4 Архиватор PowerArchive

PowerArchive - универсальный архиватор (архиваторная оболочка), распознающий распространенные форматы библиотек, включая образы оптических дисков [9]. С помощью PowerArchiver можно не только извлекать файлы из упакованных контейнеров, но и «на лету» преобразовывать формат архивов (рис.1.4).

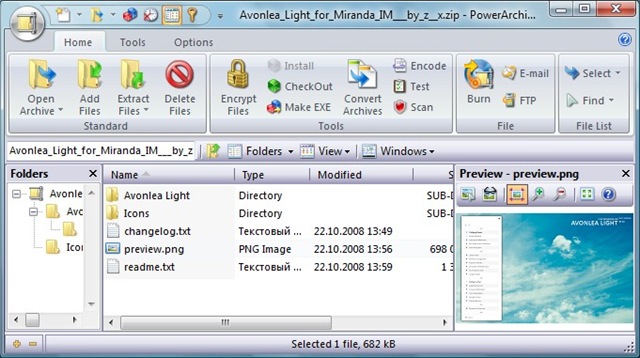


Рисунок 1.4 – Интерфейс PowerArchive

К достоинствам можно отнести следующее:

* максимальная версия приложения может создавать инкрементальные бэкапы данных в автоматическом режиме;
* поддерживает контекстное меню;
* встроенный CD/DVD-рекордер

Следует отметить следующие недостатки:

* небольшое количество поддерживаемых форматов для распаковки
* жесткая лицензионная политика (пробная версия доступна в течении месяца);

## 1.1.5 Выводы

Однозначного ответа на вопрос какой из существующих архиваторов самый лучший, имеет наибольший коэффициент сжатия и быстрее всего работает нет, поскольку скорость работы зависит от формата исходных объектов, применяемого метода (или комбинации) компрессии и типа конечной библиотеки. Установлено [1], что при использовании программ на мощных компьютерах разница в быстродействии практически незаметна.

Целью данного курсового проекта является разработка программного средства, которое осуществляет сжатие/восстановление информации на основе алгоритмов: Хаффмана, RLE, LZW и выполняет сравнение алгоритмов по ряду параметров. Данная работа не является коммерческим проектом и не может конкурировать (в силу ограниченности человеческих и временных ресурсов) с архиваторами, выпускаемыми компаниями на протяжении десятилетий.

## 1.2 Формирование требований к проектируемому программному средству

Согласно заданию на курсовой проект необходимо разработать программное средство анализа алгоритма архивации. Основные функциональные возможности:

- выполнять сжатие и восстановление исходных файлов тремя алгоритмами: Хаффмана, RLE, LZW по выбору пользователя;

- поддерживать три формата архивации .rle, .huff,.lzw ;

- записывать в начало архива сигнатуру, содержащую изначальное расширение файла и контрольную сумму;

- отображать информацию о ходе сжатия и восстановления;

- входными форматами файлов для архивации могут быть любые форматы;

- обрабатывать ошибки при работе с программным средством;

- содержать (записывать) информацию для анализа процесса архивирования с параметрами:

* + 1. времени выполнения операций сжатия/восстановления,
    2. проверки целостности архивов,
    3. степени сжатия,
    4. чувствительности к случайным ошибкам;

- иметь справку;

- иметь удобный графический пользовательский интерфейс.

В качестве языка программирования было предложено использовать язык Delphi.

# **2. МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФЦНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ**

## 2.1 Описание алгоритмов сжатия информации, положенных в основу курсового проекта

## 2.1.1 Алгоритм Хаффмана

Алгоритм был предложен Хаффманом в 1952 году. Этот алгоритм стал базой для большого количества программ сжатия информации [1-6]. Кодирование Хаффмана широко применяется при сжатии данных, в том числе при сжатии фото- и видеоизображений (JPEG, MPEG), в популярных архиваторах (PKZIP, LZH и др.), в протоколах передачи данных HTTP (Deflate), MNP5 и MNP7 и др.

Сжатие по Хаффману состоит в представлении наиболее вероятных (часто встречающихся) букв двоичными кодами наименьшей длины, а менее вероятных - кодами большей длины (если все кодовые слова меньшей длины уже исчерпаны). Основная задача – чтобы средняя длина кода на букву исходного сообщения была минимальной [3].

Алгоритм имеет следующую последовательность действий.

Шаг 1. Создается статистика встречаемости (вероятности) каждого символа в строке.

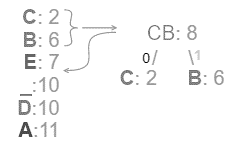
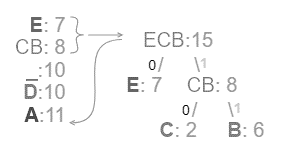
Например, задана исходная строка

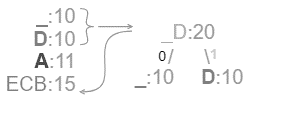
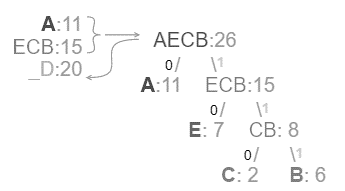


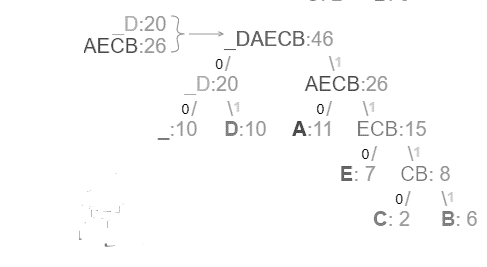
Частота встречаемости:



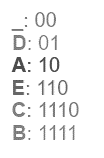
Шаг 2. Проектируется бинарное дерево, в котором корень – это сумма вероятностей предыдущих двух его потомков, в свою очередь составленных из сумм их потомков и так далее.



Шаг 3. Осуществляется проход по дереву от корня до каждого символа с записью «поворотов», например, при переходе в левую ветку записывается ноль, в правую – единицу. Когда достигается конечный символ, записанная последовательность из нулей и единиц будет кодом данного символа.



Заархивированное сообщение:



Для разархивирования сообщения, необходимо сохранять таблицу вероятностей (для построения дерева), что прибавляет еще 20-30 % к весу выходного файла.

Процесс разархивирования - это процесс преобразования потока префиксных кодов в отдельные байтовые значения, обычно путем обхода узла дерева Хаффмана за узлом, когда каждый бит читается из входного потока (достижение конечного узла обязательно прекращает поиск для этого конкретного значения байта).

Сложность алгоритма, при использовании сортировки элементов после каждого суммирования или использовании приоритетной очереди, будет *O(NlogN)*. Асимптотику можно улучшить до *O(N)*, используя обычные массивы.

Достоинства алгоритма: достижение высокой степени сжатия, минимальная избыточность, высокое качество сжатия информации.

Недостатки: для восстановления содержимого сжатого сообщения декодер должен знать таблицу частот, следовательно, длина сжатого сообщения увеличивается на длину таблицы частот; зависимость степени сжатия от близости вероятностей символов значениям отрицательной степени 2, что связано с необходимостью округления длины каждой кодовой комбинации до большего целого.

## 2.1.2 Алгоритм LZW

LZW (Lempel-Ziv-Welch) — универсальный алгоритм сжатия данных без потерь, созданный Авраамом Лемпелем, Яковом Зивом и Терри Велчем в 1984 году [4][.](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9B%D0%B5%D0%BC%D0%BF%D0%B5%D0%BB%D1%8F_%E2%80%94_%D0%97%D0%B8%D0%B2%D0%B0_%E2%80%94_%D0%92%D0%B5%D0%BB%D1%87%D0%B0#cite_note-_bcf6c4b5d55cf9ae-1) Этот метод позволяет достичь одну из наилучших степеней сжатия среди других существующих методов сжатия графических данных, при полном отсутствии потерь или искажений в исходных файлах. В настоящее время используется в файлах формата TIFF, PDF, GIF, PostScript и других, а также отчасти во многих популярных программах сжатия данных (ZIP, ARJ, LHA).

Алгоритм архивации выглядит следующим образом.

Шаг 1. Все возможные символы заносятся в словарь. Во входную фразу X заносится первый символ сообщения.

Шаг 2. Считать очередной символ Y из сообщения.

Шаг 3. Если Y — это символ конца сообщения, то выдать код для  X, иначе: Если фраза XY уже имеется в словаре, то присвоить входной фразе значение XY и перейти к Шагу 2. Иначе выдать код для входной фразы X, добавить XY в словарь и присвоить входной фразе значение YY. Перейти к Шагу 2.

Для декодирования на вход подается только закодированный текст, поскольку алгоритм LZW может воссоздать соответствующую таблицу преобразования непосредственно по закодированному тексту. Рассмотрим алгоритм разархивации.

Алгоритм генерирует однозначно декодируемый код за счет того, что каждый раз, когда генерируется новый код, новая строка добавляется в таблицу строк. LZW постоянно проверяет, является ли строка уже известной, и, если так, выводит существующий код без генерации нового. Таким образом, каждая строка будет храниться в единственном экземпляре и иметь свой уникальный номер. Следовательно, при дешифровании при получении нового кода генерируется новая строка, а при получении уже известного, строка извлекается из словаря. [4]

Рассмотрим пример. Пусть имеется сжимаемая последовательность

**abacabadabacabaeabacabadabacabae.**

Шаг 1: Тогда, согласно изложенному выше алгоритму, к изначально пустой строке добавляется символ **a** и проверяется, есть ли строка **a** в таблице. Поскольку при инициализации в таблицу были занесены все строки из одного символа, то строка **a** есть в таблице.

Шаг 2: Далее считывается следующий символ **b** из входного потока и проверяется, есть ли строка **ab** в таблице. Такой строки в таблице пока нет.

В таблицу добавляется **ab**. В поток:  ⟨0⟩;

Шаг 3:  **ba** — нет. В таблицу: **ba**. В поток: ⟨1⟩;

Шаг 4:  **ac** — нет. В таблицу: **ac**. В поток: ⟨0⟩;

Шаг 5:  **ca** — нет. В таблицу: **ca**. В поток: ⟨2⟩;

Шаг 6: **ab** — есть в таблице; **aba** — нет. В таблицу: **aba**. В поток: ⟨5⟩;

Шаг 7: **ad** — нет. В таблицу: **ad.** В поток: ⟨0⟩;

Шаг 8: **da** — нет. В таблицу: **da**. В поток: ⟨3⟩;

Шаг 9: **aba** — есть в таблице; **abac** — нет. В таблицу: **abac**. В поток: ⟨9⟩;

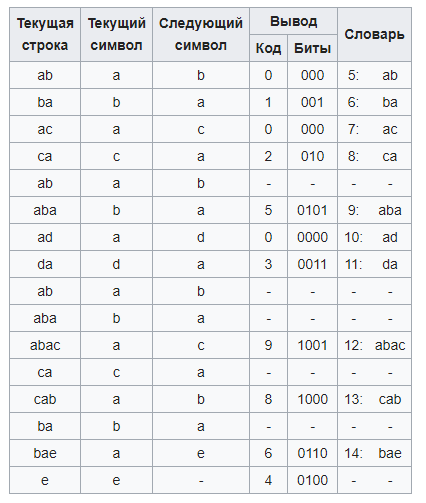
Шаг 10: **ca** — есть в таблице; **cab** — нет. В таблицу**: cab**. В поток: ⟨8⟩;

Шаг 11: **ba** — есть в таблице; **bae** — нет. В таблицу: **bae**. В поток: ⟨6⟩;

Шаг 12: И, наконец последняя строка **e**, за ней идет конец сообщения, поэтому просто выводится в поток .

Итак, мы получаем закодированное сообщение **0102503986401025039864** и его битовый эквивалент 00000100001001010000001110011000011001000000010000100101000000111001100001100100.

Описанный процесс можно представить в виде таблицы.



Каждый символ исходного сообщения был закодирован группой из трех бит, сообщение содержало 16 символов, следовательно длина сообщения составляла 3⋅16=48 бит.

Закодированное же сообщение так же сначала кодировалось трехбитными группами, а при появлении в словаре восьмого слова — четырехбитными, итого длина сообщения составила 4⋅3+7⋅4=40 бит, что на 88 бит короче исходного.

Для разархивирования прежде всего нужно знать начальный словарь, а последующие записи словаря можно реконструировать уже на ходу, поскольку они являются просто конкатенацией предыдущих записей. Кроме того, в процессе компрессии и декомпрессии коды в словарь добавляются во время обработки одного и того же символа, т.е. это происходит “синхронно”.



Для теоретической оценки эффективности данный метод сжатия сравнивают с некоторой эталонной метрикой. К сожалению, идеальная эталонная метрика сжатия данных невычислима даже приблизительно, и поэтому любой алгоритм сжатия заведомо проигрывает в сравнении с ней [4].

Достоинства: алгоритм является однопроходным; для декомпрессии не надо сохранять таблицу строк в файл для распаковки; алгоритм построен таким образом, что мы в состоянии восстановить таблицу строк, пользуясь только потоком кодов.

Недостатки: алгоритм не проводит анализ входных данных.

## 2.1.3 Алгоритм RLE

Алгоритм RLE — алгоритм сжатия, заменяющий идущие подряд одинаковые символы парой (повторяющийся символ, количество повторений). Например, строчку **aaababbcbbb** он переводит в **(a, 3) (b, 1) (a, 1) (b, 2) (c, 1) (b, 3)**. Этот алгоритм эффективен для строк, содержащих много цепочек повторяющихся символов. Распространённые форматы для упаковки данных с помощью RLE включают в себя PackBits, PCX и ILBM.

Методом кодирования длин серий могут быть сжаты произвольные файлы с двоичными данными, поскольку спецификации на форматы файлов часто включают в себя повторяющиеся байты в области выравнивания данных. Тем не менее, современные системы сжатия (например, Deflate) чаще используют алгоритмы на основе LZ77, которые являются обобщением метода кодирования длин серий и оперируют с последовательностями символов вида «BWWBWWBWWBWW».[5]

Так же стоит заметить, что если применять алгоритм кодирования повторов для тех строк, в которых повторов мало, то эффективность его снизится, потому что нет разницы (с точки зрения объёма информации) писать AA или 2A. Если же в кодируемой строке повторы вовсе отсутствуют, то подобный способ записи не только не сократит, но и увеличит её длину более чем в два раза.

Поэтому предлагаю такую реализацию данного алгоритма, где работаем с двумя видами цепочек байт: одиночными и повторяющимися байтами и рабочим байтом, который будет указывать нам на то, какая это цепочка и сколько в ней элементов.

*Рассмотрим пример*. Пусть имеется сжимаемая последовательность

00 00 00 00 00 00 04 02 00 04 04 04 04 04 04 04

50 50 50 50 00 02 02 02 02 FF FF FF FF FF 00 00

Итак, имеется два вида последовательностей: цепочки одиночных элементов (вроде «4, 2, 0») и цепочки одинаковых элементов (вроде «0, 0, 0, 0, 0, 0»). В «служебных» байтах выделяется один бит под тип последовательности: 0 — одиночные элементы, 1 — одинаковые. Для этого берется старший бит байта

В оставшихся 7 битах будут хранится длины последовательностей, т.е. максимальная длина кодируемой последовательности — 127 байт. Можно было бы выделить под служебные нужды, допустим, два байта, но в данном случае такие длинные последовательности встречаются крайне редко, поэтому проще и экономичней просто разбивать их на более короткие.

Получается, в выходной поток будут писаться сперва длина последовательности, а далее либо одно повторяемое значение, либо цепочка неповторяемых элементов указанной длины.

Не может быть последовательностей с нулевой длиной, поэтому можно увеличить максимальную длину до 128 байт, отнимая от длины единицу при кодировании и прибавляя при декодировании. Таким образом, возможно кодировать длины от 1 до 128 вместо длин от 0 до 127.

Второе, что можно заметить — не бывает последовательностей одинаковых элементов единичной длины. Поэтому, от значения длины таких последовательностей при кодировании мы будем отнимать единичку, увеличив тем самым их максимальную длину до 129 (максимальная длина цепочки одиночных элементов по-прежнему равна 128). Т.е. цепочки одинаковых элементов могут иметь длину от 2 до 129.

Происходит кодирование наших снов, но теперь уже и в понятном для компьютера виде. Служебные байты записываются как [T|L], где T — тип последовательности, а L — длина. Сразу учтём, что длины записываются в изменённом виде: при T=0 отнимаем от L единицу, при T=1 — двойку.

[1|4], 0, [0|2], 4, 2, 0, [1|5], 4, [1|2], 80, [0|0], 0, [1|2], 2, [1|3], 255, [1|0], 0

Декодированный результат:

[1|4]. T=1, значит следующий байт будет повторяться L+2 (4+2) раз: 0, 0, 0, 0, 0, 0.

[0|2]. T=0, значит просто читаем L+1 (2+1) байт: 4, 2, 0.

[1|5]. T=1, повторяем следующий байт 5+2 раз: 4, 4, 4, 4, 4, 4, 4.

[1|2]. T=1, повторяем следующий байт 2+2 раз: 80, 80, 80, 80.

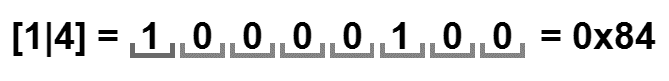
[0|0]. T=0, читаем 0+1 байт: 0.

[1|2]. T=1, повторяем байт 2+2 раз: 2, 2, 2, 2.

[1|3]. T=1, повторяем байт 3+2 раз: 255, 255, 255, 255, 255.

[1|0]. T=1, повторяем байт 0+2 раз: 0, 0.

Последний шаг: сохранение полученного результата как массива байт. Например, пара [1|4], упакованная в байт, будет выглядеть вот так:



В итоге получаем следующее:

**84 00 02 04 02 00 85 04 82 80 00 00 82 02 83 FF 80 00**

Достоинством алгоритма является то, что он не требует дополнительной памяти при архивации и разархивации, не требует хранения различных таблиц кодирования, а также факт быстрой работы.

## 2.2 Описание функциональных требований к программному средству

Программное средство должно представлять пользователю выбор одного их двух режимов: архивирование и разархивирование. Архивирование выполняется на основе трех перечисленных алгоритмов. Соответственно может выполняться по алгоритмам:

* Хаффмана (расширение .huff);
* LZW (расширение .lzw);
* RLE (расширение .rle).

При клике по кнопке «Archive» программное средство должно проверить входную информацию, запустить процесс архивации, отобразить информацию о процессе архивации и записать файл на диск с соответствующим расширением.

Если пользователь выбрал сразу несколько алгоритмов то последовательно должны запускать все выбранные методы.

Исследуемые параметры архивации: время выполнения операций архивации/разархивации, проверки целостности архивов, степени сжатия записываются в лог-файл.

В режиме разархивации необходимо выбрать файл, в качестве входной информации и нажать на кнопку «Dearchive». Также как и при архивации процесс сопровождается параметрами, которые записываются в выходной файл.

В случае выбора неверного входного формата на экран должно выводится сообщение.

Информация, записанная в лог-файл будет использоваться в дальнейшем для анализа алгоритмов.

## 2.3 Спецификация функциональных требований

* Выбор файла посредством файлового диалога

В программе имеется кнопка, после нажатия на которую открывается окно диалога выбора файл. Полный путь к выбранному файлу устанавливается в поле ввода. По желанию, пользователь может ввести путь к файлу вручную. Имеется обработка ошибок на ввод пути к несуществующему файлу и обработка случая, когда пользователь не ввел ничего.

* Возможность выбрать один или несколько алгоритмов для архивации

Имеются три чек бокса, каждый из которых отвечает за один из трех алгоритмов, данных в задании. В зависимости от того, сколько алгоритмов выбрал пользователь, будут создаваться архивы каждым из методов после нажатия на кнопку. Имеется обработка ошибки, при которой пользователь не выбрал ни один метод.

* Создание файлов-таблиц

В двух методах из трёх необходимо создание дополнительных файлов-таблиц для успешной разархивации файла в последующем. Файлы-таблицы создаются в тех же папках, что и архивные файлы и имеют расширения: .hufftable (алгоритм Хаффмана) и .lzwtable ( LZW)

* Разархивация файла в соответствии с его расширением

В программе имеется кнопка разархивации, после нажатия на которую идет обработка имени файла, определение метода разархивации и обработка ошибок: не архивное расширение файла, несуществующий файл и если пользователь не ввел путь к файлу.

* Обработка возможных ошибок, возникающих при разархивации файла
* Создание лог-файла для анализа.

# **3. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

## 3.1 Обобщённый алгоритм работы программного средства

Схема алгоритма программы представлен в приложении 1.

## 3.2 Алгоритм архивации/разархивации по Хаффману

## 3.2.1 Проектирование структур данных для алгоритма Хаффмана

В методе архивации Хаффмана были использованы структуры, приведенные в Таблице 3.1.

Таблица 3.1

|  |  |  |
| --- | --- | --- |
| **Название** | **Назначение** | **Содержимое** |
| PNode | Элемент дерева | Symbol: Byte  Байт |
| Weight: integer  Встречаемость |
| Code: string  Код |
| left,right: TPNode  Ссылки на правую и левую ветку |
| TPNode | Дерево кодирования | ^PNode |
| BytesWithStat | Массив деревьев, используется для создания файла-таблицы | array[0..255] of TPNode |
| TWeightTable | Запись, необходимая для создания статистики и файла-таблицы с встречаемостью байтов | MainArray: BytesWithStat  Массив элементов, который содержит их количество, которые встречаются хотя бы раз |
| ByteCount: Byte  Количество значащих элементов |
| TFileName\_ | Запись, хранящая информацию о файла для его архивации | Name: String  Имя файла |
| FSize: Integer  Размер файла |
| FStat:TWeightTable  Статистика частот байт |
| Node: TPNode  Дерево кодирования |
| Signature | Сигнатура, которая записывается в шапку заархивированного файла. Используется для восстановления расширения файла и проверки его целостности при разархивации. | Name: String  Расширение входного файла |
| crc: Byte  Контрольная сумма по файлу, используется для проверки целостности разархивированного файла |
| TLog | Структура, необходимая, для формирования лог-файла, в котором хранится информация об осуществленном архивировании. | opertype:string;  Тип операции: Архивация, разархивация |
| method:string;  Метод архивирования (Хаффман,RLE,LZW) |
| Ftype:string;  Расширение архивируемого\разархивируемого файла |
| Bsize:Integer;  Размер входного файла |
| Rsize:Integer;  Размер выходного файла |
| k:Real;  Коэффициент сжатия |
| time:Cardinal  Время архивации |
| result: string;  Результат архивирования |

## 3.2.2. Алгоритм архивирования Хаффмана

Схема алгоритма архивации (рис.3.1) по Хаффману составлена на основе описания алгоритма из раздела 2.1.1.



Рисунок 3.1 – Блок-схема архивирования по Хаффману

## 3.2.3. Алгоритм разархивирования Хаффмана

Схема алгоритма разархивации (рис.3.2) по Хаффману составлена на основе описания алгоритма из раздела 2.1.1.



Рисунок 3.2 – Блок-схема разархивирования по Хаффману

## 3.3 Алгоритм архивации/разархивации LZW

## 3.3.1 Проектирование структур данных для LZW

В методе сжатия LZW были использованы структуры данных, приведенные в Таблице 3.2.

Таблица 3.2.

|  |  |  |
| --- | --- | --- |
| **Название** | **Назначение** | **Структруа** |
| TMAS | Текущая последовательность байт | array[1..4096] of byte |
| PMAS | Массив для чтения из файла по 1 Кбайт | array[1..1024] of byte |
| Dictionary | Значащие поля элемента словаря кодов | symbol:TMAS  последовательность байт |
| count:Integer  количество элементов в последовательности байт |
| used:Boolean  использован ли элемент в кодировании |
| code:integer  код элемента |
| TPDic | Словарь кодов элементов | ^TDic |
| TDic | Элемент словаря кодов | info:Dictionary  значащия поля элемента |
| AdressOfNext:TPDic  Ссылка на следующий элемент словаря |

## 3.3.2. Алгоритм архивирования LZW

Схема алгоритма архивации (рис.3.3) LZW составлена и реализована на основе словесного описания алгоритма из раздела 2.1.2.

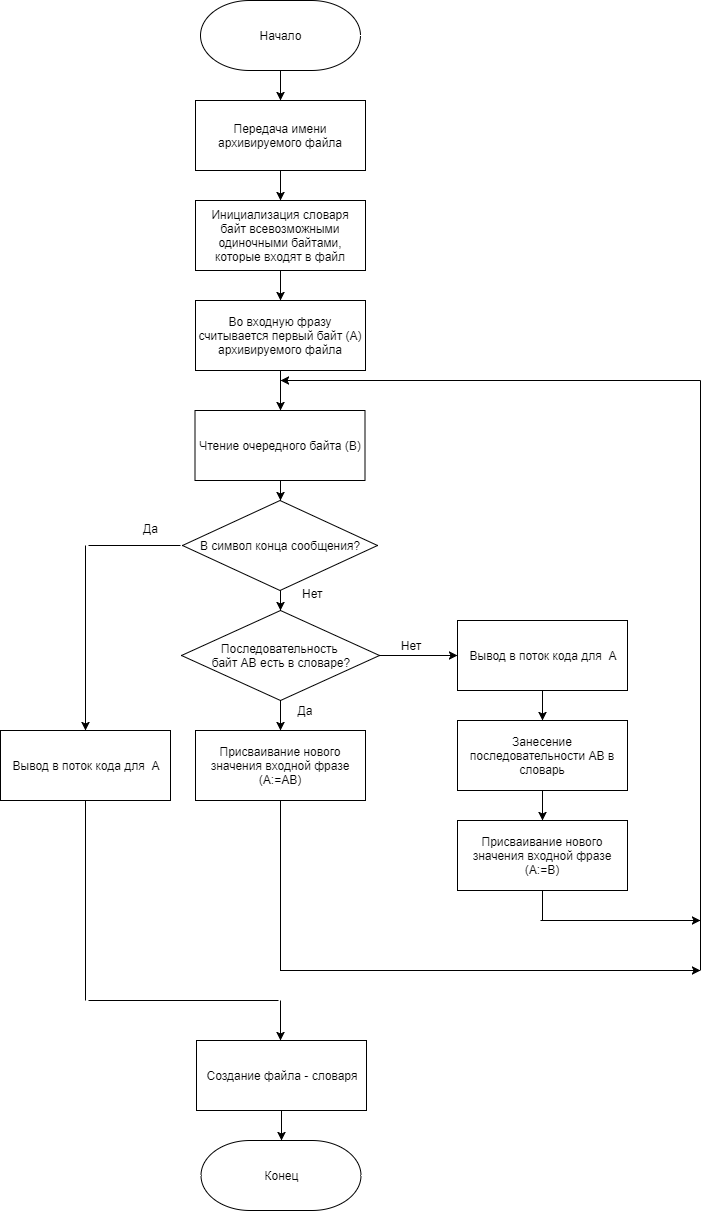


Рисунок 3.3 – Блок-схема архивирования по LZW

## 3.3.3. Алгоритм разархивирования LZW

Схема алгоритма разархивации (рис.3.4) LZW составлена на основе описания алгоритма из раздела 2.1.2.

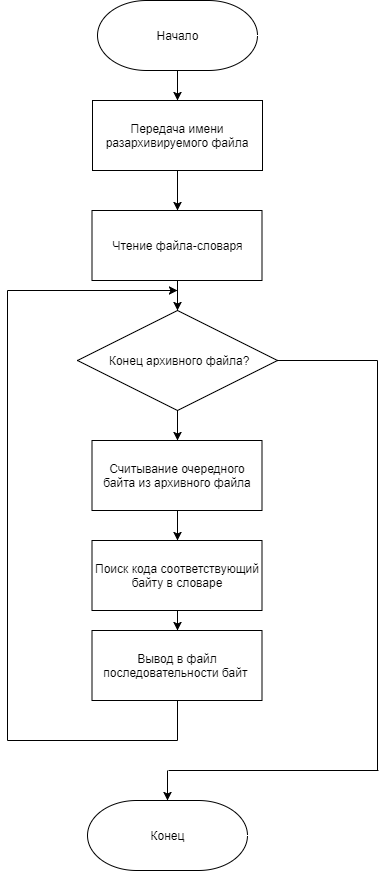


Рисунок 3.4 – Блок-схема разархивирования по LZW

## 3.4 Алгоритм архивации/разархивации RLE

## 3.4.1 Проектирование структур данных для RLE

В методе сжатия RLE были использованы структуры данных, приведенные в Таблице 3.3.

Таблица 3.3

|  |  |  |
| --- | --- | --- |
| **Название** | **Назначение** | **Структруа** |
| TMAS | Массив для чтения из файла по 1 Кбайт | array[1..1024] of byte |
| Signature | Сигнатура, которая записывается в шапку заархивированного файла. Используется для восстановления расширения файла и проверки его целостности при разархивации. | Name: String  Расширение входного файла |
| crc: Byte  Контрольная сумма по файлу, используется для проверки целостности разархивированного файла |

## 3.4.2. Алгоритм архивирования RLE

Схема алгоритма архивации (рис.3.5) RLE составлена на основе описания алгоритма из раздела 2.1.3.

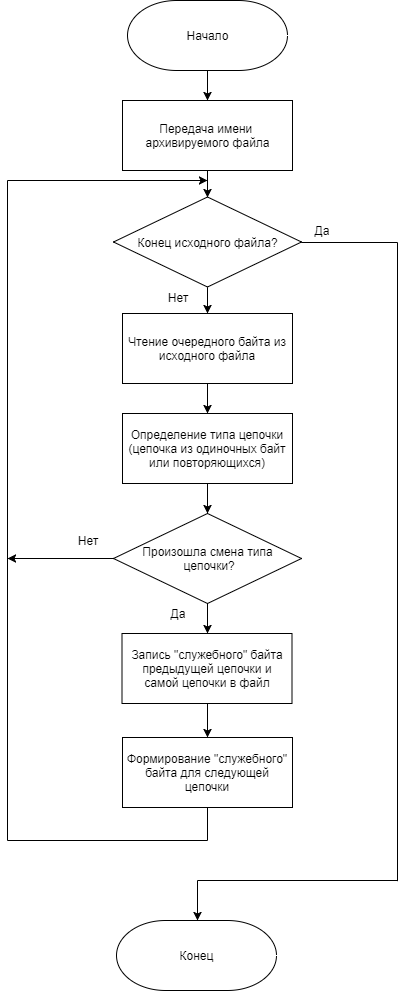


Рисунок 3.5 – Блок-схема архивирования по RLE

## 3.4.3. Алгоритм разархивирования RLE

Схема алгоритма разархивации (рис.3.6) RLE составлена на основе описания алгоритма в азделае 2.1.3.

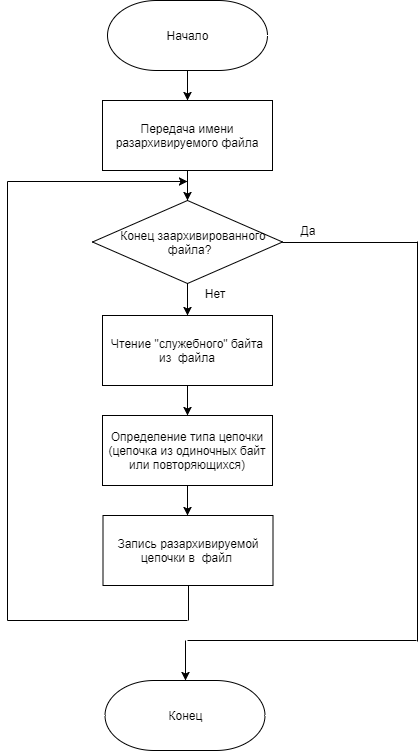


Рисунок 3.6 – Блок-схема разархивирования RLE

# **4. СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА**

## 4.1 Разработка формы программного средства

Разработка архиваторов выполнялась на основе алгоритмов разработанных и представленных в разделе 3, а также на основе спецификации функциональных требований.

Полный листинг всех подпрограмм приведен в приложении 2.

В программном средстве используется одна форма. Все основное взаимодействие происходит на ней.

В Таблице 4.1 представлены основные методы, вызываемые при обработке событий.

Таблица 4.1. – Основные подпрограммы создания интерфейса и взаимодействия с пользователем

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| TForm1.BrowseClick | Реакция программы на нажатие кнопки “Browse...”. Пользователю представится выбор файла | procedure TForm1.BrowseClick(Sender: TObject); | Sender | Объект, который сгенерировал событие |
| TForm1.ArchiveClick | Реакция программы на нажатие кнопки “Archive...”.  Будет осуществлена архивация файла одним (или несколькими) из трех методов, который (которые) выберет пользователь, или сообщено об ошибке | procedure TForm1.ArchiveClick(Sender: TObject); | Sender | Объект, который сгенерировал событие |
| TForm1.DearchiveClick | Реакция программы на нажатие кнопки “Dearchive...”.  Будет осуществлена разархивация файла одним из трех методов, который будет выбран исходя з расширения файла, или сообщено об ошибке | procedure TForm1.DearchiveClick(Sender: TObject); | Sender | Объект, который сгенерировал событие |
| TForm1.FormCreate | Создание интерфейса программного средства | procedure TForm1.FormCreate(Sender: TObject); | Sender | Объект, который сгенерировал событие |

## 4.2 Описание процедур для архивации методом RLE

Таблица 4.2. – Основные подпрограммы архивации RLE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| ByteToBin | Конверция байта в строку бит | function ByteToBin(bt:byte):String; | bt | Конвертируемый байт |
| GetNextByte | Возвращает следующий по счету байт из файла | function GetNextByte:byte; | - | - |
| WriteBlock | Запись в архивный файл цепочки (из одиночных или повторяющихся символов) и рабочий байт для этой цепочки | procedure writeBlock(symbolCount:Integer; byteTypeRepeat:boolean; blockSymbols:TMAS; blockSymbolsCounter:integer); | symbolCount | Общее количество байт в цепочке |
| byteTypeRepeat | Тип цепочки |
| blockSymbols | Массив, в котором хранится цепочка |
| blockSymbolsCounter | Количество значащих элементов в массиве |
| compressrle | Основная процедура архивации методом RLE | procedure compressrle; | - | - |
| decompressrle | Основная процедура разархивации методом RLE | procedure decompressrle; | - | - |
| WriteSign | Формирование сигнатуры для файла: его расширение и контрольная сумма | procedure WriteSign(s:string); | S | Имя файла |
| check | Проверка целостности файла после разархивации | function check(S:String):boolean; | S | Имя файла |
| RLECompress | Конечная процедура архивирования методом RLE | procedure RLECompress(FileName:string); | FileName | Имя архивируемого файла |
| RLEDecompress | Конечная процедура разархивирования методом RLE | procedure RLEDecompress(FileName:string); | FileName | Имя разархивируемого файла |
| HeadLog | Процедура открытия(создания) лог-файла и при необходимости создания шапки для него | Procedure HeadLog; | - | - |

## 4.3 Описание процедур для архивации методом LZW

Таблица 4.3. – Основные подпрограммы архивации LZW

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| Found | Функция поиска элемента в словаре | function Found(A:TMAS;c:Integer):integer; | A | Последовательность байт |
| c | Количество байт в последовательности |
| Add | Процедура добавления последовательности в словарь | procedure Add(a:TMAS;c:integer); | a | Последовательность байт |
| c | Количество байт в последовательности |
| LZWReadTable | Считывание файла-словаря | procedure ReadTable; | - | - |
| LZWDeArchiv | Процедура разархивации файла | procedure DeArchiv; | - | - |
| LZWWriteTable | Создание файла-словаря | procedure WriteTable; | - | - |
| maketrue | Пометка элемента словаря, как использованный в кодирвоании | procedure maketrue(b:integer); | b | Код элемента словаря |
| LZWArchiv | Основная процедура архивирования | procedure Archiv; | - | - |

## 4.4 Описание процедур для архивации методом Хаффмана

Таблица 4.4. – Основные подпрограммы архивации Хаффманом

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| CreateStat | Инициализация таблицы встречаемости | procedure CreateStat(var a:TFileName\_); | a | Файл, для которого необходима данная инициализация |
| Statistic | Процедура подсчета статистики встречаемости байт | procedure Statistic; | - | - |
| SortMainArray | Сортировка массива байт по убыванию весов | procedure SortMainArray(var a: BytesWithStat; LengthOfMass: byte); | a | Таблица встречаемости |
| LengthOfMass | Длина массива байт |
| CreateNode | Процедура создания дерева кодирвоания | procedure CreateNode(var Root: TPNode; MainArray: BytesWithStat; last: byte);  var | Root | Указатель на начало дерева |
| MainArray | Таблица встречаемости байт |
| last | Длина массива |
| Found | Поиск элементов в дереве кодирования | function Found(Node: TPNode; i: byte): Boolean; | Node | Дерево кодирования |
| i | Искомый байт |
| HuffCodeHelp | Функция для создания кодов Хаффмана | function HuffCodeHelp(Node: TPNode; i: Byte): string; | Node | Дерево кодирования |
| i | Байт, для которого создается код |
| HuffCode | Поиск элементов в массиве | function HuffCode(Node: TPNode; i: Byte): string; | Node | Дерево кодирования |
| i | Искомый байт |
| WriteInFile | Процедура записи строки битов в файл | procedure WriteInFile(var buffer: string); | buffer | Строка, которая равна строке байт (переведенных из бит) по завершению процедуры |
| WriteInTFileName\_( | Процедура записи остаточной строки (меньше 8) битов в файл | procedure WriteInTFileName\_(var buffer: string); | buffer | Строка, которая равна строке байт (переведенных из бит) по завершению процедуры |
| CreateTableFile | Создание файла-таблицы , который содержит все байты и их вес | procedure CreateTableFile; | - | - |
| CreateArchiv | Процедура создания сжатого файла | procedure CreateArchiv; | - | - |
| DeleteNode | Процедура очистки дерева кодирования | procedure DeleteNode(Root: TPNode); | Root | Указатель на начало дерева кодирования |
| CreateFile | Внешняя процедура создания архивного файла | procedure CreateFile; | - | - |
| Huffarchive | Конечная процедура создания архивного файла | procedure Huffarchive(FileName:string); | FileName | Имя архивируемого файла |
| ReadTable | Считывание файла-таблицы | procedure ReadTable; | - | - |
| CreateDeArc | Процедура создания разархивированного файла | procedure CreateDeArc; | - | - |
| ExtractFile | Внешняя процедура разархивирования файла | procedure ExtractFile; | - | - |
| Huffdearchive | Конечная процедура создания разархивированного файла | procedure Huffdearchive(FileName:string); | FileName | Имя разархивируемого файла |

# **5. ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА**

Тестирование - один из этапов разработки, поскольку при написании программного кода неизбежны. Тестирование – сравнение реального и ожидаемого поведения программы. Основная цель – выявление несоответствия функционирования программы спецификации и устранение их для повышения качества программного средства. Проводилось функциональное тестирование программы.

Функциональные тесты, проведенные над программой, представлены в таблице 5.1.

Таблица 5.1 – Результаты функционального тестирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер  теста | Тестируемая функциональность | Последовательность действий | Ожидаемый результат | Полученный результат |
| 1 | Некорректный выбор файла | 1. Кликнуть по кнопке “Browse” 2. Выбирать файл для архивирования, который не существует  3. Нажать кнопку “Archive” | Программа выведет сообщение о том, что такого файла не существует | Тест пройден |
| 2 | Отсутствие файла | 1. Оставить поле с именем файла для архивации пустым  2. Нажать кнопку “Archive” | Программа выведет сообщение о том, что такого файла не существует | Тест пройден |
| 3 | Некорректный выбор метода | 1. Кликнуть по кнопке “Browse” 2. Выбирать файл для архивирования  3.Не выбрать ни один из методов архивирования  4. Нажать кнопку “Archive” | Программа выведет сообщение о том, что не выбран ни один из методов | Тест пройден |
| 4 | Выбор метода | 1. Кликнуть по кнопке “Browse” 2. Выбирать файл для архивирования  3.Выбрать один метод архивирования(например метод Хаффмана)  4. Нажать кнопку “Archive” | Программа выведет сообщение о том, что архивация прошла успешно.  В папке к файлом появится архивный файл с соответствующим разрешением | Тест пройден |
| 5 | Выбор метода | 1. Кликнуть по кнопке “Browse” 2. Выбирать файл для архивирования  3.Выбрать все методы архивирования  4. Нажать кнопку “Archive” | Программа выведет сообщение о том, что архивация прошла успешно.  В папке к файлом появятся 3 архивных файл с соответствующими разрешениями, и два файла-таблицы | Тест пройден |
| 6 | Выбор файла с неверным расширением при разархивации | 1. Кликнуть по кнопке “Browse” 2. Выбирать файл для разархивирования, не имеющий расширения ‘.huff’,’.lzw’,’.rle’  4. Нажать кнопку “Dearchive” | Программа выведет сообщение о том, что выбран файл с не архивным разрешением | Тест пройден |
| 7 | Выбор разархивирования | 1. Кликнуть по кнопке “Browse” 2. Выбирать файл для разархивирования, имеющий одно из расширений ‘.huff’,’.lzw’,’.rle’ (например ’.rle’)  4. Нажать кнопку “Dearchive | Программа выведет сообщение о том, что разархивация прошла успешно, файл будет восстановлен со своим изначальным разрешением. | Тест пройден |

Программа успешно прошла все тесты, что показывает корректность работы программы и соответствие функциональным требованиям.

# **6. РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ**

Для того, чтобы начать использовать программное средство, необходимо запустить файл SkyArchiver.exe. После открытия программы появится окно, показанное на рисунке 6.1.

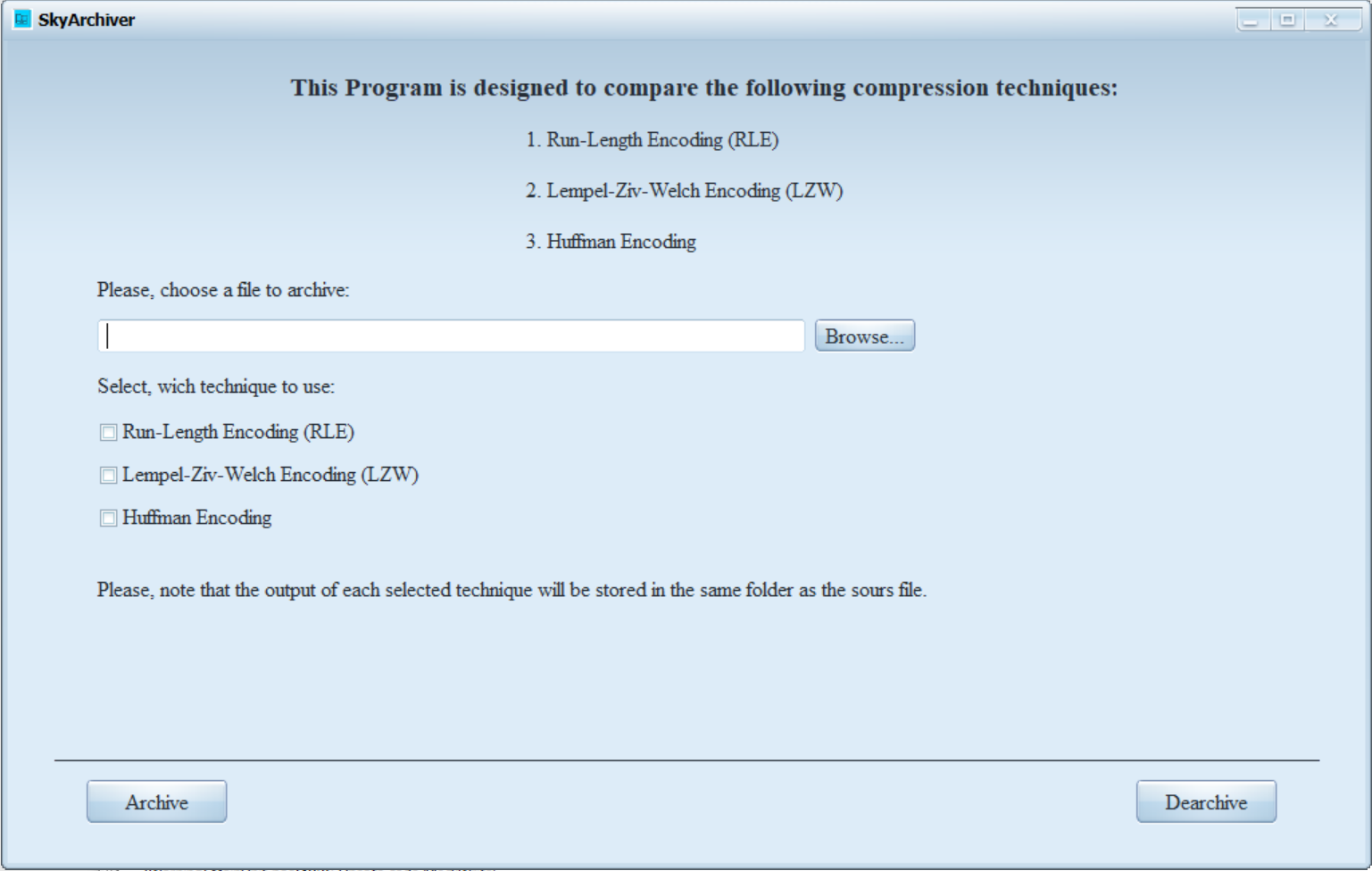


Рисунок 6.1 – Главная страница программы

Для выбора файла, пользователю необходимо нажать на кнопку “Browse..”, после чего откроется диалоговое окно выбора файла, где пользователь сможет выбрать файл. (Он так же может ввести полный путь к файлу в строку ввода) (рис.6.2)).

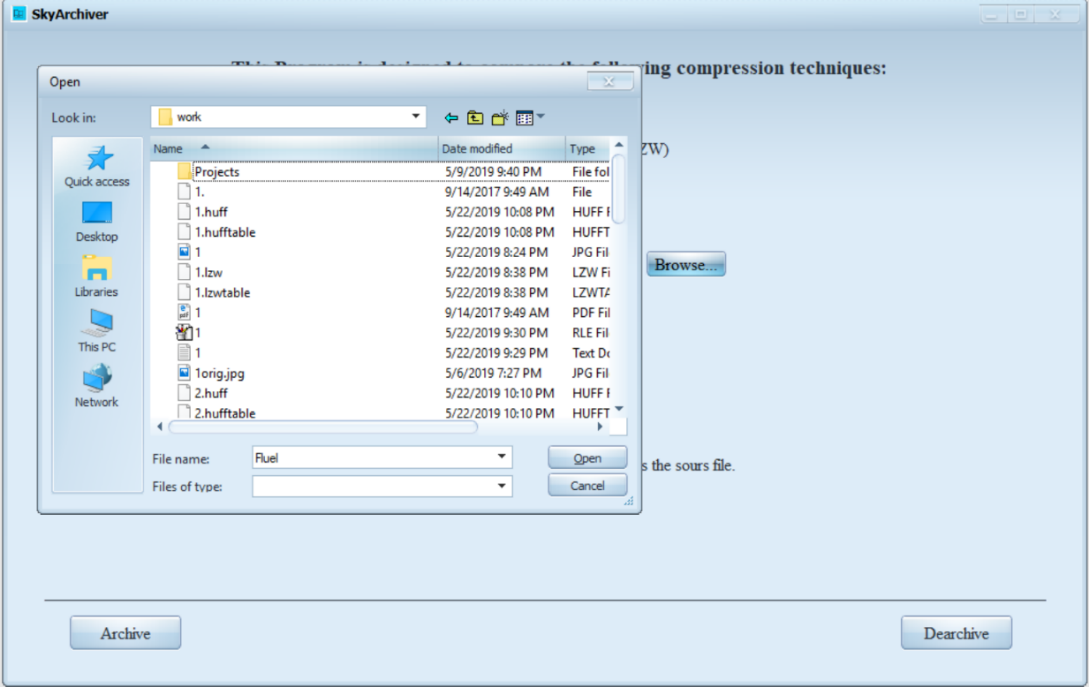


Рисунок 6.2 – Выбор файла для архивации

После выбора файла, полный путь к нему отобразится в поле ввода (рис.6.3).

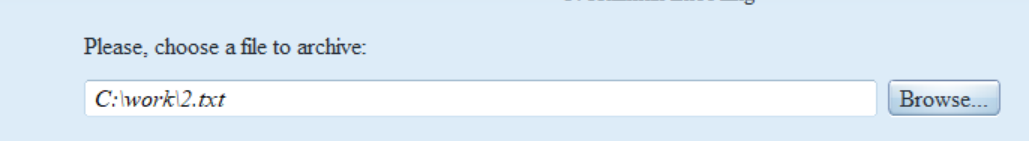


Рисунок 6.3 – Определение пути

Затем, пользователю предоставляется выбор одного (или нескольких) способов архивации файла. Выбранный чекбокс значит, что будет создан архивный файл с соответствующим расширением (рис.6.4).

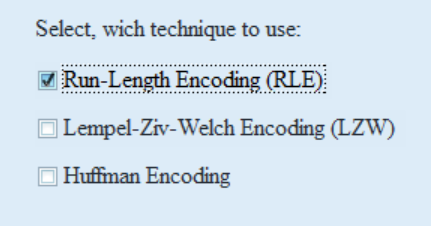


Рисунок 6.4 – Выбор алгоритма

После нажатия на кнопку “Archive” на экран выведется сообщение о том, что архивирование прошло успешно или сообщение об ошибке.

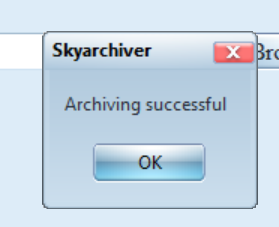


Рисунок 6.5 – Окно сообщения о результате архивации

Файл для разархивации выбирается аналогично, за одним исключением. Для разархивации не важно положение чек боксов, программа анализирует расширение входного файла, и сама выбирает метод разархивирования.

После нажатия на кнопку “Dearchive”, на экран выведется сообщение о том, что архивирование прошло успешно или сообщение об ошибке (рис.6.6).

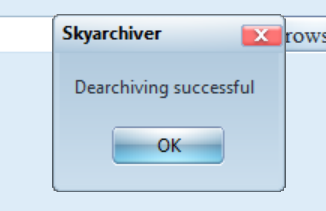


Рисунок 6.6 – Окно сообщения об ошибке

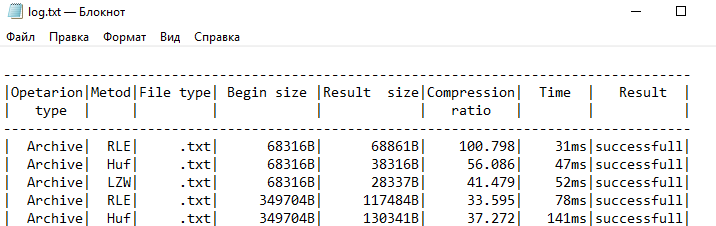
# **7. АНАЛИЗ РАБОТЫ АЛГОРИТМОВ АРХИВАЦИИ**

елью исследования является сравнение реализованных в приложении методов сжатия данных без потерь, чтобы установить следующие характеристики:

* cтепень (коэффициент) сжатия, которая может быть достигнута с помощью каждого из алгоритмов;
* уровень эффективности различных алгоритмов для разного типа информации;
* cравнение методов по времени архивации.

Тестирование программного средства проводилось на следующей вычислительной машине: Intel Core i7, оперативная память 16 ГБ. Операционная система Windows 10.

Для удобства анализа, в ходе тестирования (при выполнении архивации файлов) происходит запись информации в файл log.txt в следующем формате:



Данные, которые использовались для тестирования трех сравниваемых методов сжатия: текстовый документ (txt), документ формата pdf и изображение (jpg).

Коэффициент (степень) сжатия файлов определен как отношение размера архивного файла Vc к размеру исходного файла Vо, выраженное в процентах:

Кс=(Vc/Vo)\*100%.

После запуска тестов были получены данные по коэффициентам сжатия, размерам файлов и времени архивации представленные в таблицах 7.1 -7.3 для txt, pdf b jpg соответственно.

Таблица 7.1 – Результаты исследований для формата **txt**

|  |  |  |  |
| --- | --- | --- | --- |
|  | RLE | LZW | HUF |
| *Размер исходного файла, байт* | 68316 | | |
| *Размер архива, байт* | 68861 | 20337 | 38316 |
| *Время архивации,мс* | 31 | 50 | 47 |
| *Коэффициент сжатия* | 100,7977633 | 29,76901458 | 56,08642 |
| *Размер исходного файла, байт* | 349704 | | |
| *Размер архива, байт* | 117484 | 120565 | 130341 |
| *Время архивации,мс* | 78 | 174 | 141 |
| *Коэффициент сжатия* | 33,59526914 | 34,47629996 | 37,27181 |
| *Размер исходного файла, байт* | 1048576 | | |
| *Размер архива, байт* | 207484 | 107851 | 150341 |
| *Время архивации,мс* | 103 | 289 | 214 |
| *Коэффициент сжатия* | 19,78721619 | 10,28547287 | 14,33764 |

Таблица 7.2 – Результаты исследований для формата **pdf**

|  |  |  |  |
| --- | --- | --- | --- |
|  | RLE | LZW | HUF |
| *Размер исходного файла, байт* | 64677 | | |
| *Размер архива, байт* | 65114 | 48777 | 46682 |
| *Время архивации,мс* | 32 | 98 | 47 |
| *Коэффициент сжатия* | 100,6756652 | 75,41629946 | 72,17713 |
| *Размер исходного файла, байт* | 232915 | | |
| *Размер архива, байт* | 234676 | 198782 | 100921 |
| *Время архивации,мс* | 47 | 140 | 110 |
| *Коэффициент сжатия* | 100,7560698 | 85,34529764 | 43,32954 |
| *Размер исходного файла, байт* | 1315681 | | |
| *Размер архива, байт* | 1325968 | 1310321 | 428954 |
| *Время архивации,мс* | 203 | 482 | 359 |
| *Коэффициент сжатия* | 100,7818765 | 99,59260641 | 32,60319 |

Таблица 7.3 – Результаты исследований для формата jpg

|  |  |  |  |
| --- | --- | --- | --- |
|  | RLE | LZW | HUF |
| *Размер исходного файла, байт* | 177782 | | |
| *Размер архива, байт* | 179057 | 176876 | 177785 |
| *Время архивации, мс* | 63 | 110 | 109 |
| *Коэффициент сжатия* | 100,7171705 | 99,4903871 | 100,0017 |
| *Размер исходного файла, байт* | 2840007 | | |
| *Размер архива, байт* | 2440007 | 2587374 | 2787929 |
| *Время архивации, мс* | 187 | 677 | 875 |
| *Коэффициент сжатия* | 85,91552767 | 91,10449376 | 98,16627 |
| *Размер исходного файла, байт* | 21360644 | | |
| *Размер архива, байт* | 20360574 | 20985660 | 20972913 |
| *Время архивации, мс* | 500 | 2567 | 10016 |
| *Коэффициент сжатия* | 95,3181655 | 98,24450986 | 98,18483 |

## 7.1 Исследование коэффициента сжатия по типу архивируемой информации алгоритмом Хаффмана

Согласно полученным данным при работе алгоритма Хаффмана, хуже всего сжимается pdf, причем независимо размера файла. Больше всего коэффициент сжатия дают файлы формата txt. Как видно на рис. 7.1 чем больше размер файла тем больше коэффициент сжатия (14%).

Рисунок 7.1. Исследование коэффициента сжатия по типу и размеру исходной информации алгоритмом Хаффмана

## 7.2 Исследование коэффициента сжатия по типу архивируемой информации алгоритмом RLE

При использовании алгоритма RLE, хуже всего сжимаются файлы формата jpg, независимо от размера файла. Больше всего коэффициент сжатия дают файлы формата txt. Как видно на рис. 7.2 чем больше размер файла тем больше коэффициент сжатия (19%).

Рисунок 7.2. Исследование коэффициента сжатия по типу и размеру исходной информации алгоритмом RLE

## 7.3 Исследование коэффициента сжатия по типу архивируемой информации алгоритмом LZW

При использовании алгоритма LZW, хуже всего сжимаются файлы формата pdf, независимо от размера, и jpg. Больше всего коэффициент сжатия как и при использовании других алгоритмов дают файлы формата txt. Коэффициент сжатия (10%) увеличивается с увеличением размера файла.

Рисунок 7.3. Исследование коэффициента сжатия по типу и размеру исходной информации алгоритмом LZW

## 7.4 Исследование коэффициента сжатия для файлов формата txt

Как видно из графика анализа при работе с текстовыми файлами наиболее эффективными являются алгоритмы Хаффмана и LZW (рис.7.4). С увеличением размера файла коэффициенты сжатия становятся меньше. Однако, стоит отметить, что эффективность зависит от содержимого или контента информации.

Рисунок 7.4 Исследование коэффициента сжатия для файлов формата txt

## 7.5 Исследование коэффициента сжатия для файлов формата jpg

Как видно из графика при работе с jpg файлами алгоритмы LZW и RLE не так эффективны как алгоритм Хаффмана (рис.7.5).

Рисунок 7.5 Исследование коэффициента сжатия для файлов формата jpg

## 7.6 Исследование коэффициента сжатия для файлов формата pdf

Как видно из графика (рис.7.6) при работе с pdf файлами при увеличении размеров фалов дает алгоритм RLE, хотя коэффициент сжатия не высокий. Хуже всех работает алгоритм Хаффмана.

Рисунок 7.6 Исследование коэффициента сжатия для файлов формата pdf

## 7.7 Исследование времени архивации

Время растет с увеличением размеров файла. Если говорить о производительности алгоритмов, то быстрее работает RLE, медленнее LZW и Хаффмана.

Рисунок 7.7 Исследование времени архивации

Подводя итог, можно сделать вывод, что для сжатия тестовых документов лучше подходит LZW алгоритм. Для сжатия pdf исследуемые алгоритмы мало эффективны, так минимальный коэффициент сжатия -85% был получен при использовании алгоритма RLE. Для графической информации (jpg) лучше адаптирован алгоритм Хаффмана.

Время архивации зависит от размера данных и несколько лучше, чем остальные работает алгоритм RLE.

В дальнейшем можно было бы исследовать другие форматы – doc, html, avi , pas и т.д, а также исследовать время разархивирования и большие размеры файлов, например. от 100 Мб.

# **ЗАКЛЮЧЕНИЕ**

В процессе выполнения курсового проекта была проанализирована предметная область, рассмотрены существующие аналоги и выявлены их преимущества и недостатки. В качестве языка разработки использовался Delphi.

На этапе проектирования были разработаны: блок-схемы алгоритмов, разработан графический интерфейс пользователя. В соответствии с поставленной целью было разработано приложение предоставляющее функционал архиватора для трех алгоритмов: Хаффмана, LZW и RLE. Все операции архиватор записывает в лог-файл благодаря чему можно выполнить исследование алгоритмов.

Проведено тестирование работоспособности разработанной программной части. Поставленная цель была выполнена в полном объеме, работоспособность подтверждена тестированием программного средства. Разработано руководство пользователя.

Выполнена серия экспериментов, что позволило исследовать характеристики разработанных архиваторов по размеру и типу информации, скорости архивации.

Установлено что для сжатия тестовых файлов (txt) хорошие результаты дает LZW алгоритм. Для сжатия pdf, исследуемые алгоритмы мало эффективны, так минимальный коэффициент сжатия -85% был получен при использовании алгоритма RLE. Для графической информации (jpg) лучше адаптирована разработанная реализация алгоритм Хаффмана. Исследовано время архивации. Установлено, что время архивации зависит от размера данных и несколько лучше, чем остальные работает алгоритм RLE.

Разработанное программное средство может быть использовано при изучении дисциплин, связанных с обработкой и преобразованием информации.

# **Список использованной литературы туры**

1. Архиваторы [Электронный ресурс] – Режим доступа : <https://ru.wikipedia.org/wiki/Архиватор>
2. Архиватор 7-Zip [Электронный ресурс] – Режим доступа : <http://utilitka.com/obzory/80-obzor-i-sravnenie-winrar-winzip-i-7zip.html> Дата доступа: 20.03.19
3. Алгоритм Хаффмана [Электронный ресурс] – Режим доступа : <http://metod.vt.tpu.ru/lab/huffman/theory.html> Дата доступа: 20.03.19
4. Алгоритм LZW [Электронный ресурс] – Режим доступа : <http://neerc.ifmo.ru/wiki/LZW> Дата доступа: 20.03.19
5. Алгоритм RLE [Электронный ресурс] – Режим доступа : <https://ru.wikipedia.org/wiki/Кодирование_длинн_серий> Дата доступа: 20.03.19
6. Архиватор WinRAR [Электронный ресурс] – Режим доступа : <https://www.rarlab.com/> Дата доступа: 15.03.19
7. Архиватор 7-Zip [Электронный ресурс] – Режим доступа : <http://www.compression-pointers.ru/compress_226.html> Дата доступа: 15.03.19
8. Архиватор WinZip [Электронный ресурс] – Режим доступа : [http://www.winzip.com](http://www.compression-pointers.ru/ext_link?url=http://www.winzip.com) Дата доступа: 15.03.19
9. Архиватор PowerArchive [Электронный ресурс] – Режим доступа : <https://www.powerarchiver.com/download-powerarchiver/> Дата доступа: 15.03.19
10. Глухова, Л.А. Основы алгоритмизации и программирования: учебное пособие. В 2 Ч. / Л.А. Глухова. – БГУИР, 2006 – Ч. 1. – 195 с.
11. А.Н.Вальвачев, К.А.Сурков, Д.А.Сурков, Ю.М.Четырько. Программирование на языке Delphi. Учебное пособие[Электронный ресурс] Режим доступа: http://www.rsdn.ru/?summary/3165.xml. Дата доступа: 20.03.19

# **ПРИЛОЖЕНИE 1**

# **ПРИЛОЖЕНИE 2**

unit MainUnit;

interface

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, sSkinManager, sButton, sCheckBox, sLabel, ComCtrls,

acProgressBar, ExtCtrls;

type

TForm1 = class(TForm)

Head: TLabel;

Types: TLabel;

Label1: TLabel;

select: TLabel;

Note: TLabel;

line: TLabel;

dlgOpen: TOpenDialog;

Memo1: TMemo;

sknmngr1: TsSkinManager;

Archive: TsButton;

Dearchive: TsButton;

Browse: TsButton;

RLE: TsCheckBox;

LZW: TsCheckBox;

Huff: TsCheckBox;

procedure BrowseClick(Sender: TObject);

procedure ArchiveClick(Sender: TObject);

procedure DearchiveClick(Sender: TObject);

procedure Formcreate(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

Form1: TForm1;

implementation

{$R \*.dfm}

type

{RLE}

TMAS = array[1..1024] of byte;

A = file of Byte;

Signature = record

name:string[6];

crc:byte;

end;

TLog=record

opertype:string;

method:string;

Ftype:string;

Bsize:Integer;

Rsize:Integer;

k:Real;

time:Cardinal;

result: string;

end;

{Huffman}

TPNode=^PNode;

PNode = record

Symbol:Byte;

Weight:integer;

Code:string;

left,right:TPNode;

end;

BytesWithStat = array[0..255] of TPNode;

TWeightTable=record

MainArray:BytesWithStat;

ByteCount:byte;

end;

TFileName\_=record

Name:string;

FSize:integer;

FStat:TWeightTable;

Node: TPNode;

end;

{LZW}

TMAS = array [1..4096] of byte;

PMAS = array [1..1024] of Byte;

Dictionary=record

symbol:TMAS;

count:Integer;

used:Boolean;

code:integer;

end;

TPDic=^TDic;

TDic = record

info:Dictionary;

AdressOfNext:TPDic;

end;

const sizeofbuffer=1024;

var

{RLE}

FileIn,FileOut ,FileEx: file;

byteArray: TMAS;

onebyte:Byte;

readByte,index : Integer;

Sign:Signature;

Start,Stop:Cardinal;

{Huffman}

ArchFile:TFileName\_;

LOG: textfile;

logstr:TLog;

ex:Boolean;

Procedure HeadLog;

var

MyDIR:string;

begin

MyDIR:=ExtractFileDir(Application.ExeName);

if not DirectoryExists(MyDIR+'\LOG') then

CreateDir(MyDIR+'\LOG');

if not (fileExists(MyDIR+'\LOG\log.txt'))

then

begin

Rewrite(LOG);

writeln(LOG,'--------------------------------------------------------------------------------------');

writeln(LOG,'|Opetarion|Metod|File type| Begin size |Result size|Compression| Time | Result |');

writeln(LOG,'| type | | | | | ratio | | |');

writeln(LOG,'--------------------------------------------------------------------------------------');

end

else

begin

AssignFile(LOG,MyDIR+'\LOG\log.txt');

Append(LOG);

if (Filesize(LOG)=0) then

begin

writeln(LOG,'--------------------------------------------------------------------------------------');

writeln(LOG,'|Opetarion|Metod|File type| Begin size |Result size|Compression| Time | Result |');

writeln(LOG,'| type | | | | | ratio | | |');

writeln(LOG,'--------------------------------------------------------------------------------------');

end;

end;

end;

{---------------RLE BLOCK-------------------------}

function ByteToBin(bt:byte):String;

var

i:Integer;

b:byte;

begin

Result:='';

For i:=7 downto 0 do

begin

b:=bt shr i;

b:=b and $1;

If b=1 then

Result:=Result+'1'

else

Result:=Result+'0';

end;

end;

function GetNextByte:byte;

begin

if(index = readByte) then

begin

if not Eof(FileIn) then begin

BlockRead(FileIn, byteArray, 1024, readByte);

index := 0;

end

else readByte:=-1;

end;

if readByte<>-1 then

begin

Inc(index);

result := byteArray[index];

end;

end;

procedure writeBlock(symbolCount:Integer; byteTypeRepeat:boolean; blockSymbols:TMAS; blockSymbolsCounter:integer);

begin

if (symbolCount <> 0) AND (blockSymbolsCounter <> 0)then

begin

onebyte:= symbolCount;

if byteTypeRepeat then onebyte:= onebyte+128;

BlockWrite(FileOut, onebyte,1);

if byteTypeRepeat then BlockWrite(FileOut,blockSymbols[1],1)

else

BlockWrite(FileOut,blockSymbols,blockSymbolsCounter);

end;

end;

procedure compressrle;

var curr,prev:Byte;

byteTypeRepeat:Boolean;

blockSymbols : TMAS;

blockSymbolsCounter,symbolCount:Integer;

symbolRepeatCount: Integer;

begin

symbolCount:=1;

symbolRepeatCount:=1;

byteTypeRepeat:=False;

curr:=GetNextByte;

blockSymbolsCounter:=1;

blockSymbols[blockSymbolsCounter]:=curr;

prev:=curr;

curr:=GetNextByte;

while readByte<>-1 do

begin

try

Inc(symbolCount);

Inc(blockSymbolsCounter);

blockSymbols[blockSymbolsCounter]:=curr;

if curr = prev then

begin

Inc(symbolRepeatCount);

if (byteTypeRepeat=False) and (symbolRepeatCount > 2) then

begin

blockSymbolsCounter:=blockSymbolsCounter-symbolRepeatCount;

symbolCount:=symbolCount-symbolRepeatCount;

writeBlock(symbolCount, byteTypeRepeat, blockSymbols, blockSymbolsCounter);

symbolCount:=symbolRepeatCount;

blockSymbolsCounter:=1;

blockSymbols[blockSymbolsCounter]:=curr;

byteTypeRepeat:=True;

end

end

else

begin

if byteTypeRepeat then

begin

blockSymbolsCounter:=blockSymbolsCounter-1;

symbolCount:=symbolCount-1;

writeBlock(symbolCount, byteTypeRepeat, blockSymbols, blockSymbolsCounter);

symbolCount:=1;

blockSymbolsCounter:=1;

blockSymbols[blockSymbolsCounter]:=curr;

byteTypeRepeat:=False;

end;

symbolRepeatCount:=1;

end;

if (byteTypeRepeat=False) AND (blockSymbolsCounter > 126) then

begin

writeBlock(symbolCount, byteTypeRepeat, blockSymbols, blockSymbolsCounter);

symbolCount:=0;

blockSymbolsCounter:=0;

end;

prev:=curr;

curr:=GetNextByte;

except

ShowMessage('Error');

ex:=False;

end;

end;

writeBlock(symbolCount, byteTypeRepeat, blockSymbols, blockSymbolsCounter);

end;

procedure decompressrle;

var j,count:Integer;

s:byte;

curr:byte;

begin

curr:=GetNextByte;

while readByte<>-1 do

begin

count:=curr and 127;

s:= curr shr 7;

if s=1 then

begin

curr:=GetNextByte;

for j:=1 to count do

BlockWrite(FileOut,curr,1);

end

else

begin

for j:=1 to count do

begin

curr:=GetNextByte;

BlockWrite(FileOut,curr,1);

end;

end;

curr:=GetNextByte;

end;

end;

procedure WriteSign(s:string);

var

C,B:Byte;

begin

index:=1024;

readByte:=1024;

Sign.name:=ExtractFileExt(s);

AssignFile(FileIn, s);

Reset(FileIn, 1);

B:=GetNextByte;

C:=0;

while readbyte<>-1 do

begin

C:= C xor B;

B:=GetNextByte;

end;

Sign.crc:=C;

CloseFile(FileIn);

end;

function check(S:String):boolean;

var

C,B:Byte;

begin

AssignFile(FileIn, s);

Reset(FileIn, 1);

index:=1024;

readbyte:=1024;

B:=GetNextByte;

C:=0;

while readbyte<>-1 do

begin

C:= C xor B;

B:=GetNextByte;

end;

CloseFile(FileIn);

if C=Sign.crc then result:=True

else result:=False;

end;

procedure RLECompress(FileName:string);

var

d:Real;

begin

logstr.opertype:='Archive';

logstr.method:='RLE';

Start:=GetTickCount;

WriteSign(FileName);

AssignFile(FileIn, FileName);

try

Reset(FileIn, 1);

AssignFile(FileOut,ChangeFileExt(FileName,'.rle'));

ReWrite(FileOut, 1);

index:=1024;

readByte:=1024;

BlockWrite(FileOut,Sign.name,6);

BlockWrite(FileOut,Sign.crc,1);

compressrle;

logstr.Ftype:=Sign.name;

logstr.Rsize:=Filesize(FileOut);

logstr.Bsize:=FileSize(FileIn);

d:=((logstr.Rsize)/(logstr.Bsize))\*100;

logstr.k:=d;

CloseFile(FileIn);

CloseFile(FileOut);

except

ShowMessage('File is anavailable');

ex:=False;

DeleteFile(ChangeFileExt(FileName,'.rle'));

end;

Stop:=GetTickCount;

logstr.time:=Stop-Start;

if ex=True then

logstr.result:='successfull'

else logstr.result:='Failed';

Write(LOG,'|');

Write(LOG,logstr.opertype:9,'|');

Write(LOG,logstr.method:5,'|');

Write(LOG,logstr.Ftype:9,'|');

Write(LOG,logstr.Bsize:11,'B|');

Write(LOG,logstr.Rsize:11,'B|');

Write(LOG,logstr.k:11:3,'|');

Write(LOG,logstr.time:6,'ms|');

Writeln(LOG,logstr.result:10,'|');

end;

procedure RLEDecompress(FileName:string);

var d:real;

begin

logstr.opertype:='Dearchive';

logstr.method:='RLE';

Start:=GetTickCount;

AssignFile(FileIn,FileName);

try

Reset(FileIn, 1);

BlockRead(FileIn,Sign.name,6);

BlockRead(FileIn,Sign.crc,1);

AssignFile(FileOut,ChangeFileExt(FileName,Sign.name) );

ReWrite(FileOut, 1);

index:=1024;

readByte:=1024;

logstr.Ftype:=Sign.name;

decompressrle;

logstr.Rsize:=Filesize(FileOut);

logstr.Bsize:=FileSize(FileIn);

d:=((logstr.Rsize)/(logstr.Bsize))\*100;

logstr.k:=d;

CloseFile(FileIn);

CloseFile(FileOut);

if Check(ChangeFileExt(FileName,Sign.name)) then

begin

ShowMessage('Dearchiving successful');

ex:=true;

end

else

begin

ShowMessage('The file is damaged');

ex:=false;

end;

except

ShowMessage('File is anavailable');

DeleteFile(ChangeFileExt(FileName,Sign.name));

ex:=false;

end;

Stop:=GetTickCount;

logstr.time:=Stop-Start;

if ex=True then

logstr.result:='successfull'

else logstr.result:='Failed';

Write(LOG,'|');

Write(LOG,logstr.opertype:9,'|');

Write(LOG,logstr.method:5,'|');

Write(LOG,logstr.Ftype:9,'|');

Write(LOG,logstr.Bsize:11,'B|');

Write(LOG,logstr.Rsize:11,'B|');

Write(LOG,'':11,'|');

Write(LOG,logstr.time:6,'ms|');

Writeln(LOG,logstr.result:10,'|');

end;

{---------------RLE BLOCK-------------------------}

{-------------HUFFMAN BLOCK----------------------}

procedure CreateStat(var a:TFileName\_);

var

i: Byte;

begin

with a.FStat do

begin

ByteCount := 255;

for i := 0 to ByteCount do

begin

New(MainArray[i]);

with MainArray[i]^ do

begin

Symbol := i;

Weight := 0;

left := nil;

right := nil;

end;

end;

end;

end;

procedure Statistic;

var

f:file;

i,j:integer;

buf:array[1..sizeofbuffer] of byte;

countbuf,lastbuf: integer;

readbyte:integer;

begin

AssignFile(f,ArchFile.Name);

try

Reset(f,1);

CreateStat(ArchFile);

ArchFile.FSize:=FileSize(f);

While not EoF(f) do

begin

BlockRead(f,buf,sizeofbuffer,readbyte);

for j:=1 to readbyte do

inc(ArchFile.FStat.MainArray[buf[j]]^.Weight);

end;

CloseFile(f);

except

ShowMessage('File is anavailable');

Ex:=false;

DeleteFile(ChangeFileExt(ArchFile.Name,'.huff'));

end;

end;

procedure SortMainArray(var a: BytesWithStat; LengthOfMass: byte);

var

i, j: Byte;

b: TPNode;

begin

if LengthOfMass <> 0 then

for j := 0 to LengthOfMass-1 do

begin

for i := 0 to LengthOfMass -1 do

begin

if a[i]^.Weight < a[i + 1]^.Weight then

begin

b := a[i];

a[i] := a[i + 1];

a[i + 1] := b;

end;

end;

end;

end;

procedure CreateNode(var Root: TPNode; MainArray: BytesWithStat; last: byte);

var

Node: TPNode;

begin

if last <> 0 then

begin

SortMainArray(MainArray, last);

new(Node);

Node^.Weight := MainArray[last - 1]^.Weight + MainArray[last]^.Weight;

Node^.left := MainArray[last - 1];

Node^.right := MainArray[last];

MainArray[last - 1] := Node;

if last = 1 then

begin

Root := Node;

end

else

begin

CreateNode(Root, MainArray, last - 1);

end;

end

else

Root := MainArray[last];

end;

function Found(Node: TPNode; i: byte): Boolean;

begin

if (Node = nil) then

Found := False

else

begin

if ((Node^.left = nil) or (Node^.right = nil)) and (Node^.Symbol = i) then

Found := True

else

Found := Found(Node^.left, i) or Found(Node^.right, i);

end;

end;

function HuffCodeHelp(Node: TPNode; i: Byte): string;

begin

if (Node = nil) then

HuffCodeHelp := '+'

else

begin

if (Found(Node^.left, i)) then

HuffCodeHelp := '0' + HuffCodeHelp(Node^.left, i)

else

begin

if Found(Node^.right, i) then

HuffCodeHelp := '1' + HuffCodeHelp(Node^.right, i)

else

begin

if (Node^.left = nil) and (Node^.right = nil) and (Node^.Symbol = i) then

HuffCodeHelp := '+'

else

HuffCodeHelp := '';

end;

end;

end;

end;

function HuffCode(Node: TPNode; i: Byte): string;

var

s: string;

begin

s := HuffCodeHelp(Node, i);

if (s = '+') then

HuffCode := '0'

else

HuffCode := Copy(s, 1, length(s) - 1);

end;

procedure WriteInFile(var buffer: string);

var

i, j: Integer;

k: Byte;

buf: array[1..2 \* SizeOfBuffer] of byte;

begin

i := Length(buffer) div 8;

for j := 1 to i do

begin

buf[j] := 0;

for k := 1 to 8 do

begin

if buffer[(j - 1) \* 8 + k] = '1' then

buf[j] := buf[j] or (1 shl (8 - k));

end;

end;

BlockWrite(FileEx, buf, i);

Delete(buffer, 1, i \* 8);

end;

procedure WriteInTFileName\_(var buffer: string);

var

a, k: byte;

begin

WriteInFile(buffer);

if Length(buffer) <> 0 then

begin

a := $FF;

for k := 1 to Length(buffer) do

if buffer[k] = '0' then

a := a xor (1 shl (8 - k));

BlockWrite(FileEx, a, 1);

end;

end;

procedure CreateTableFile;

var

TFile: File ;

s:string;

i: Byte;

writeCount: Integer;

begin

s:=ChangeFileExt(ArchFile.Name,'.hufftable');

AssignFile(TFile,s);

Rewrite(TFile,1);

BlockWrite(TFile, ArchFile.FSize, 4);

BlockWrite(TFile, ArchFile.FStat.ByteCount, 1);

for i := 0 to ArchFile.FStat.ByteCount do

begin

BlockWrite(TFile, ArchFile.FStat.MainArray[i]^.Symbol, 1);

BlockWrite(TFile, ArchFile.FStat.MainArray[i]^.Weight, 4);

end;

Close(TFile);

end;

procedure CreateArchiv;

var

buffer: string;

ArrOfStr: array[0..255] of string;

i, j,readbytea,b,a: Integer;

buf: array[1..SizeOfBuffer] of Byte;

CountBuf, LastBuf: Integer;

d:Real;

begin

AssignFile(FileIn, ArchFile.Name);

Assign(FileEx,ChangeFileExt(ArchFile.Name,'.huff'));

try

Reset(FileIn, 1);

Rewrite(FileEx, 1);

BlockWrite(FileEx,Sign.Name,6);

BlockWrite(FileEx,Sign.crc,1);

logstr.Ftype:=Sign.name;

logstr.Bsize:=FileSize(FileIn);

for i := 0 to 255 do

ArrOfStr[i] := '';

for i := 0 to ArchFile.FStat.ByteCount do

begin

ArrOfStr[ArchFile.FStat.MainArray[i]^.Symbol] := ArchFile.FStat.MainArray[i]^.Code;

end;

buffer := '';

CreateTableFile;

while not EOF(FileIn) do

begin

BlockRead(FileIn, buf, SizeOfBuffer,readbyte);

for j := 1 to readbyte do

begin

buffer := buffer + ArrOfStr[buf[j]];

if Length(buffer) > 8 \* SizeOfBuffer then

begin

WriteInFile(buffer)

end;

end;

end;

WriteInTFileName\_(buffer );

logstr.Rsize:=Filesize(FileEx);

d:=((logstr.Rsize)/(logstr.Bsize))\*100;

logstr.k:=d;

CloseFile(FileIn);

CloseFile(FileEx);

except

ShowMessage('File is anavailable');

DeleteFile(ChangeFileExt(ArchFile.Name,'.huff'));

ex:=false;

end;

Stop:=GetTickCount;

logstr.time:=Stop-Start;

if ex=True then

logstr.result:='successfull'

else logstr.result:='Failed';

Write(LOG,'|');

Write(LOG,logstr.opertype:9,'|');

Write(LOG,logstr.method:5,'|');

Write(LOG,logstr.Ftype:9,'|');

Write(LOG,logstr.Bsize:11,'B|');

Write(LOG,logstr.Rsize:11,'B|');

Write(LOG,logstr.k:11:3,'|');

Write(LOG,logstr.time:6,'ms|');

Writeln(LOG,logstr.result:10,'|');

end;

procedure DeleteNode(Root: TPNode);

begin

if Root <> nil then

begin

DeleteNode(Root^.left);

DeleteNode(Root^.right);

Dispose(Root);

Root := nil;

end;

end;

procedure CreateFile;

var

i:byte;

begin

with ArchFile do

begin

SortMainArray(FStat.MainArray,FStat.ByteCount);

i:=0;

while (i<FStat.ByteCount) and (FStat.MainArray[i]^.Weight<>0) do inc(i);

if FStat.MainArray[i]^.Weight=0 then dec(i);

FStat.ByteCount:=i;

CreateNode(Node,FStat.MainArray,FStat.ByteCount);

for i:=0 to FStat.ByteCount do

FStat.MainArray[i]^.Code:=HuffCode(Node,FStat.MainArray[i]^.Symbol);

CreateArchiv;

DeleteNode(Node);

CreateStat(ArchFile);

end;

end;

procedure Huffarchive(FileName:string);

begin

logstr.opertype:='Archive';

logstr.method:='Huf';

Start:=GetTickCount;

ArchFile.Name:=FileName;

WriteSign(FileName);

Statistic;

CreateFile;

end;

procedure ReadTable;

var

i: Byte;

TFile:file;

begin

AssignFile(TFile,ChangeFileExt(ArchFile.Name,'.hufftable'));

try

Reset(TFile,1);

BlockRead(TFile, ArchFile.FSize, 4);

BlockRead(TFile, ArchFile.FStat.ByteCount, 1);

for i := 0 to ArchFile.FStat.ByteCount do

begin

BlockRead(TFile, ArchFile.FStat.MainArray[i]^.Symbol, 1);

BlockRead(TFile, ArchFile.FStat.MainArray[i]^.Weight, 4);

end;

Close(TFile);

except

ShowMessage('No table File to dearchive');

DeleteFile(ChangeFileExt(ArchFile.Name,Sign.name));

ex:=false;

end;

end;

procedure CreateDeArc;

var

j: Integer;

k,size: Byte;

Buf: array[1..SizeOfBuffer] of Byte;

readbyte: Integer;

CurrentPoint: TPNode;

begin

CurrentPoint := ArchFile.Node;

size:=0;

while not Eof(FileIn) do

begin

BlockRead(FileIn, Buf, SizeOfBuffer,readbyte);

for j := 1 to readbyte do

begin

for k := 1 to 8 do

begin

if (Buf[j] and (1 shl (8 - k))) <> 0 then

CurrentPoint := CurrentPoint^.right

else

CurrentPoint := CurrentPoint^.left;

if ((CurrentPoint^.left = nil) or (CurrentPoint^.right = nil)) then

begin

if ArchFile.FSize<>size then

begin

BlockWrite(FileEx,CurrentPoint^.Symbol,1);

inc(size);

end;

CurrentPoint := ArchFile.Node;

end

end;

end;

end;

end;

procedure ExtractFile;

var d:Real;

begin

AssignFile(FileIn, ArchFile.Name);

Reset(FileIn, 1);

BlockRead(FileIn,Sign.name,6);

BlockRead(FileIn,Sign.crc,1);

AssignFile(FileEx,ChangeFileExt(ArchFile.Name,Sign.name) );

try

Rewrite(FileEx, 1);

CreateStat(ArchFile);

ReadTable;

CreateNode(ArchFile.Node, ArchFile.FStat.MainArray, ArchFile.FStat.ByteCount);

logstr.Ftype:=Sign.name;

CreateDeArc;

logstr.Rsize:=Filesize(FileEx);

logstr.Bsize:=FileSize(FileIn);

d:=((logstr.Rsize)/(logstr.Bsize))\*100;

logstr.k:=d;

DeleteNode(ArchFile.Node);

Closefile(FileIn);

Closefile(FileEx);

except

ShowMessage('File is anavailable');

DeleteFile(ChangeFileExt(ArchFile.Name,Sign.name));

ex:=false;

end;

Stop:=GetTickCount;

logstr.time:=Stop-Start;

if ex=True then

logstr.result:='successfull'

else logstr.result:='Failed';

Write(LOG,'|');

Write(LOG,logstr.opertype:9,'|');

Write(LOG,logstr.method:5,'|');

Write(LOG,logstr.Ftype:9,'|');

Write(LOG,logstr.Bsize:11,'B|');

Write(LOG,logstr.Rsize:11,'B|');

Write(LOG,'':11,'|');

Write(LOG,logstr.time:6,'ms|');

Writeln(LOG,logstr.result:10,'|');

end;

procedure Huffdearchive(FileName:string);

begin

logstr.opertype:='Dearchive';

logstr.method:='Huf';

Start:=GetTickCount;

ArchFile.name := FileName;

ExtractFile;

if Check(ChangeFileExt(ArchFile.Name,Sign.name)) then

ShowMessage('Dearchiving successful')

else

begin

ShowMessage('The file is damaged');

DeleteFile(ChangeFileExt(ArchFile.Name,Sign.name));

end;

end;

{-------------HUFFMAN BLOCK----------------------}

{----------------LZW BLOCK-----------------------}

function Found(A:TMAS;c:Integer{;b:byte}):integer;

var request:TMAS;

i:integer;

flag:Boolean;

Current:TPDic;

begin

result:=-1;

{ Inc(c);

A[c]:=b; }

Current := Head;

while (Current^.AdressOfNext <> Nil) and (result=-1) do

begin

Current := Current^.AdressOfNext;

//flag:=True;

if (c = Current^.Info.count) and (CompareMem(@Current^.Info.symbol, @a, c)) then

result:=Current^.Info.code;

{

for i:=1 to c do

if (Current^.Info.symbol[i] <> A[i]) then

flag:=False;

if flag=True then

result:=Current^.Info.code;

}

end;

//if result=-1 then

// Add(a, c);

end;

procedure SingleSymbols;

var

i,g:Byte;

s:TMAS;

begin

index:=1024;

readbyte:=1024;

New(Head);

Curr := Head;

Curr^.AdressOfNext := Nil;

i:=0;

AssignFile(FileIn,FileName);

Reset(FileIn,1);

// g:=GetNextByte;

s[1]:=GetNextByte;

while readbyte<>-1 do

begin

// if Found(s,0,g)=-1 then

if Found(s,1)=-1 then

begin

New(Curr^.AdressOfNext);

Curr^.AdressOfNext^.AdressOfNext := Nil;

Curr^.AdressOfNext^.info.symbol[1]:=s[1];

Curr^.AdressOfNext^.info.count:=1;

Curr^.AdressOfNext^.info.used:=false;

Curr^.AdressOfNext^.info.code:=i;

Curr:=Curr^.AdressOfNext;

inc(i);

end;

s[1]:=GetNextByte;

end;

CloseFile(FileIn);

end;

procedure Add(a:TMAS;c:integer);

var i:Integer;

begin

while Curr^.AdressOfNext<>nil do

begin

Curr:=Curr^.AdressOfNext;

end;

New(Curr^.AdressOfNext);

Curr^.AdressOfNext^.AdressOfNext := Nil;

Curr^.AdressOfNext^.info.used:=false;

Curr^.AdressOfNext^.Info.Count:=c;

//Move(Curr^.AdressOfNext^.Info.Symbol, a, c);

for i:=1 to c do

begin

Curr^.AdressOfNext^.Info.Symbol[i]:=a[i];

end;

{ if (Curr^.AdressOfNext = Head) then

begin

Curr^.AdressOfNext^.Info.code:= 0;

end

else }

Curr^.AdressOfNext^.Info.code := Curr^.Info.code + 1;

Curr:=Curr^.AdressOfNext;

end;

{procedure CreateTableFile;

TFile:file;

begin

AssignFile(TFile,'Table.LZW');

Reset(TFile,1);

end; }

procedure ReadTable;

var

Current: TPDic;

i,c: integer;

b,a:byte;

begin

AssignFile(FileIn,'C:\work\Table.LZW');

Reset(FileIn,1);

index:=1024;

readbyte:=1024;

New(Head);

Current := Head;

Current^.AdressOfNext := Nil;

while readbyte<>-1 do

begin

New(Current^.AdressOfNext);

Current := Current^.AdressOfNext;

a:=GetNextByte;

b:=GetNextByte;

c:=10\*b+a;

Current^.Info.code:=c;

Current^.Info.count:=GetNextByte;

for i:=1 to Current^.Info.count do

begin

Current^.Info.symbol[i]:=GetNextByte;

end;

Current^.AdressOfNext := Nil;

end;

CloseFile(FileIn);

end;

procedure DeArchiv;

var a,b:Byte;

Current:TPDic;

i,c:integer;

begin

ReadTable;

AssignFile(FileEx,FileName);

AssignFile(FileIn,'C:\work\UK.jpg.LZW');

Rewrite(FileEx,1);

Reset(FileIn,1);

index:=1024;

readbyte:=1024;

a:=GetNextByte;

b:=GetNextByte;

while readbyte<>-1 do

begin

c:= 10 \* b + a ;

Current := Head;

while (Current^.AdressOfNext <> Nil)

do

begin

Current := Current^.AdressOfNext;

if Current^.Info.code=c then

begin

// BlockWrite(FileEx,Current^.Info.code,1);

// BlockWrite(FileEx,Current^.Info.count,1);

for i:=1 to Current^.Info.count do

BlockWrite(FileEx,Current^.Info.symbol[i],1);

end;

end;

// BlockWrite(FileEx,c,1);

a:=GetNextByte;

b:=GetNextByte;

end;

end;

procedure WriteTable;

var

Current:TPDic;

i:integer;

begin

AssignFile(TFile,'C:\work\Table.LZW');

Rewrite(TFile,1);

Current := Head;

while (Current^.AdressOfNext <> Nil) do

begin

Current := Current^.AdressOfNext;

if Current^.Info.used then

begin

BlockWrite(TFile,Current^.Info.code,2);

BlockWrite(TFile,Current^.Info.count,1);

for i:=1 to Current^.Info.count do

BlockWrite(TFile,Current^.Info.symbol[i],1);

end;

end;

end;

procedure maketrue(b:integer);

var

Current:TPDic;

i:integer;

flag:Boolean;

begin

Current := Head;

flag:=False;

while (Current^.AdressOfNext <> Nil) and (flag<>true) do

begin

Current := Current^.AdressOfNext;

if Current^.Info.code=b then

begin

Current^.Info.used:=True;

flag:=True;

end;

end;

end;

procedure Archiv;

var i:Integer;

a,b:integer;

begin

// SingleSymbols;

New(Head);

Curr := Head;

Curr^.AdressOfNext := Nil;

AssignFile(FileIn,FileName);

AssignFile(FileEx,FileName+'.LZW');

Reset(FileIn,1);

Rewrite(FileEx,1);

index:=1024;

readbyte:=1024;

//P:=0;

P[1]:=GetNextByte;

//C:=GetNextByte;

count:=1;

//Add(P,count);

a:=-1;//Found(P, count);

while readByte<>-1 do

begin

b:=Found(P, count);

if b=-1 then

begin

if a <> -1 then

begin

BlockWrite(FileEx, a, 2);

MakeTrue(a);

end;

Add(P,count);

P[1]:=P[count];

count:=1;

a:=Found(P, count);

end

else a:=b;

if a<>-1 then

begin

inc(count);

P[count]:=GetNextByte;

end;

end;

a:=Found(P,count-1);

BlockWrite(FileEx,a,2);

MakeTrue(a);

WriteTable;

Close(FileIn);

Close(FileEx);

end;

{----------------LZW BLOCK-----------------------}

procedure TForm1.BrowseClick(Sender: TObject);

begin

if dlgOpen.Execute then

begin

Memo1.Text:=dlgOpen.FileName;

end;

end;

procedure TForm1.ArchiveClick(Sender: TObject);

begin

HeadLog;

ex:=True;

if FileExists(Memo1.Text) then

begin

if RLE.Checked then

RLECompress(Memo1.Text);

if Huff.Checked then

Huffarchive(Memo1.Text);

if LZW.Checked then

LZWarchive(Memo1.Text);

if (RLE.Checked=false) and (Huff.Checked=False) and (LZW.Checked=false) then

ShowMessage('You haven''t select any technique.')

else if ex=True then

ShowMessage('Archiving successful')

else ShowMessage('Something went wrong, archiving is anavailable')

end

else ShowMessage('Such file does not exists.');

CloseFile(LOG);

end;

procedure TForm1.DearchiveClick(Sender: TObject);

begin

HeadLog;

ex:=True;

if FileExists(Memo1.Text) then

begin

if ExtractFileExt(Memo1.Text)='.rle' then

RLEDecompress(Memo1.Text)

else

if ExtractFileExt(Memo1.Text)='.huff' then

Huffdearchive(Memo1.Text)

Else

if ExtractFileExt(Memo1.Text)='.lzw' then

LZWdearchive(Memo1.Text)

else ShowMessage('Non archive file extension');

end

else ShowMessage('Such file does not exists.');

CloseFile(LOG);

end;

procedure TForm1.FormCreate(Sender: TObject);

begin

Form1.Height:= 650;

Form1.Width:= 1024;

sknmngr1.SkinDirectory:=extractfilepath(application.ExeName);

sknmngr1.SkinName:='Topaz';

sknmngr1.Active:=true;

Memo1.Text:='';

end;

end.

ВЕДОМОСТЬ

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | Дополнительные сведения | | | |
|  | | | | Текстовые документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| БГУИР КР 1–40 01 01 226 ПЗ | | | | Пояснительная записка | | | | 75 с. | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | | Графические документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| ГУИР 851002 226 ПД | | | | "Анализ алгоритмов архивации данных"  А1, схема программы, чертеж | | | | **Формат А1** | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  |  |  |  |  | **БГУИР КР 1-40 01 01 226 Д1** | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | "Анализ алгоритмов архивации данных"  Ведомость курсовой  работы |  | | | | Лист | Листов |
| Разраб. | | Цыбулько К.Д. |  | 20.05.09 | Т |  | |  | 75 | 75 |
| Пров. | | Фадеева Е.Е. |  | 20.05.09 | Кафедра ПОИТ  гр. 851002 | | | | | |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |