**МИОНОБРНАУКИ РОССИИ  
федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ»  
(СПбГЭУ)**

Факультет Информатики и Прикладной Математики  
Кафедра

**КУРСОВАЯ РАБОТА**  
по дисциплине:  
**«Информатика и программирование»**

Тема: Создание программного комплекса для компрессии и декомпрессии информации в электронном виде

Направление (специальность) *09.03.03 Прикладная информатика*

Направленность (специализация) *Управление бизнес-процессами и проектами*

Обучающийся *Стерлин Павел Михайлович*

Группа *ПИ-2201* Подпись:\_\_\_\_\_\_\_\_\_\_\_\_

Проверил: Сотавов А. К.

Должность: \_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_ Дата: \_\_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург  
2023

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#__RefHeading___Toc799_3007368307)

[ВВЕДЕНИЕ 3](#__RefHeading___Toc801_3007368307)

[ОСНОВНАЯ ЧАСТЬ 4](#__RefHeading___Toc176_552501875)

[1. ИСТОРИЯ РАЗВИТИЯ И СПОСОБЫ КОМПРЕССИИ ИНФОРМАЦИИ 4](#__RefHeading___Toc140_3387658944)

[1.1 Первые алгоритмы компрессии 4](#__RefHeading___Toc105370_552501875)

[1.2 Принцип работы и развитие алгоритма Шеннона-Фано 5](#__RefHeading___Toc105372_552501875)

[1.3 Алгоритмы компрессии мультимедийной информации 6](#__RefHeading___Toc432_3878693808)

[1.4 Перспективы развития технологий сжатия информации 10](#__RefHeading___Toc434_3878693808)

[2. МОДЕЛЬ РАБОТЫ АРХИВАТОРА 12](#__RefHeading___Toc170_552501875)

[2.1 Интерфейс приложения 12](#__RefHeading___Toc172_552501875)

[2.2 Описание математической модели 13](#__RefHeading___Toc438_3878693808)

[2.3 Алгоритм Построения Дерева Хаффмана 15](#__RefHeading___Toc436_3878693808)

[2.4 Доказательство оптимальности алгоритма 18](#__RefHeading___Toc440_3878693808)

[СПИСОК ИСТОЧНИКОВ И ЛИТЕРАТУРЫ 21](#__RefHeading___Toc166_3007174596)

# ВВЕДЕНИЕ

**Актуальность темы.** Количество данных создаваемых человечеством поистине огромно, увеличиваясь по экспоненте каждый год. Часть этих данных попадет в долгосрочное хранилище — архив. Будь то обязанность компаний хранить данные о пользователях или необходимость такой услуги для функционирования организации, например соц сети, где пользователи ожидают сохранения своих данных на длительный срок, для всего этого необходимы мощность для хранения данных. Но нередко к таким данным не нужен быстрый доступ, как например к исполняемым файлам, поэтому для увеличения вместительности можно использовать компрессию данных. В таком случае мы обмениваем быстродействие на вместительность. Степень такого обмена может варьироваться, например мы можем использовать сложный алгоритм компрессии для хранения фотографий на сервисе, но при их отправке клиенту мы столкнемся с необходимостью их декомпрессировать, что вызовет повышенную нагрузку на сервер, что замедлит работу нашего клиента. Поэтому создание приложения с гибкой настройкой возможности компрессии является актуальной для сферы задачей.

**Цель исследования:** создание программного обеспечения, на основе дерева Хаффмана, позволяющего компрессировать и декомпрессировать данные.

**Для достижения цели исследования поставлены следующие задачи**:

* Разобраться в теории работы алгоритмов компрессии, математически доказать их оптимальность;
* Составить модель работы программы для работы с архивами;
* Реализовать на основе составленных в модели алгоритмов приложение.

**Объект исследования**: серверы и персональные компьютеры.

**Предмет исследования**: архивация данных, представленных в электронном виде.

# ОСНОВНАЯ ЧАСТЬ

## 1. ИСТОРИЯ РАЗВИТИЯ И СПОСОБЫ КОМПРЕССИИ ИНФОРМАЦИИ

### 1.1 Первые алгоритмы компрессии

Одним из ранних и общеизвестных примеров примеров компрессии информации является код Морзе. Разработанный в 1838 году Самюэлем Морзе, он предназначался для использования в телеграфии, в нем для обозначения более часто используемых символов применяются коды меньшей длины, жертвуя длиной менее часто используемых символов, тем самым уменьшая общую длину сообщения. Например, в русском варианте, самый популярный символ - «Е», поэтому он кодируется самым коротким кодом - «.». Самые длинные коды принадлежат знакам препинания, например восклицательный знак имеет код - «\_ \_ . . \_\_».

Но современные понятия о компрессии данных начались с создания теории информации. Отцом информационной теории считается Клауд Шеннон(1916 — 2001). Статьи Шеннона «Математическая теория связи» и «Теория связи в секретных системах» считаются основополагающими для теории информации и криптографии. Его помощником и второй по величине фигурой был Роберт Фано(1917 — 2016). Вместе они разработали код Шеннона-Фано, позволяющего определить коды для блоков текста на основе частоты их появления текста, базовый принцип работы приведен далее.

### 1.2 Принцип работы и развитие алгоритма Шеннона-Фано

1. Подсчет вероятностей. Для каждого отдельного символа необходимо подсчитать вероятность его появления в сообщении. Для этого подсчитаем сколько раз он встречается в сообщении и поделим на количество символом во всем тексте.

2. Сортировка по вероятности. Символы сортируются по вероятности в убывающем порядке, это гарантирует присвоение символа с большей вероятностью меньших по длине кодовых слов.

3. Разделение и присвоение кодов. На этом символы распределяются в 2 множества таким образом, что сумма вероятностей в каждом множестве приближается к 0,5. Одному из множеств присвоим код «0», второму «1».

4. Рекурсивное кодирование. Шаг 3 повторим для каждой группы дописывая в конец кодового слова «1» или «0», в зависимости от группы в которую они попадают. Необходимо повторить эту процедуру пока каждому индивидуальному символу не будет присвоен уникальный код.

5. Построение двоичного дерева. На основе нашего алгоритма построим двоичное дерево, для последующего декодирования.(приложение 1)

Такой метод использует особенности языка, где одни символы встречаются чаще других. Позднее оптимальный метод построения такого дерева был предложен Дэвидом Хаффманом, этот принцип мы рассмотрим во 2 разделе, так как он является основой практической части работы.

Ранние варианты компрессии использовали предзаписанные в «железе» кодовые слова. Это делалось для упрощения вычислений и коррекции ошибок. Позднее с увеличением производительности компьютеров появилась возможность динамически, то есть в процессе работы, менять кодовые слова. Такая методика получила название Адаптивное дерево Хаффмана. Это улучшало компрессию, ведь учитывались особенности данных, с которыми осуществлялась работа, но требовало дополнительных вычислительных затрат. Такой алгоритм стал популярен в 1970ых когда возможность стала появятся возможность хранить текст на локальной сети. Это привело к появлению большого количества утилит и приложений, который почти всегда использовали динамический алгоритм Хаффмана.

### 1.3 Алгоритмы компрессии мультимедийной информации

До этого рассмотренные нами алгоритмы работают с любыми данными в двоичном представлении, они могут отражать все что угодно от звуков, то текста и 3д моделей. Все они используют компрессию без потерь, то есть мы можем полностью восстановить данные без потерь информации. Но не все типы данных требуют абсолютной точности, конечно текст нельзя сжимать с потерей информации, он быстро станет труднопонимаемым, а если он будет представлять машинные команды, то абсолютно бесполезным. Для таких данных характерно сжатие в 3 раза, но для изображений сжатие может происходить и в 20 раз. Такое сжатие без потерь невозможно, но потери в таком случае не являются критичными для человека. Алгоритмы, которые мы рассмотрим далее, уже используют особенности восприятия человека, чтобы иметь возможность убрать ненужную или не заметную для человека информацию.

Первые алгоритмы для обработки звука использовались для обработки человеческой речи, для кодирования, например музыки, они не использовались, ведь они опирались на особенности работы человеческого голоса. Так называемая Линейно предсказательная модель (*Linear predictive coding*) делит звуковую дорожку на части по несколько десятых миллисекунд, назовем их сэмплами. За это время частота изменяется линейно и чтобы получить достаточный уровень детализации достаточно примерно 8 сэмплов, которые были записаны до этого, новый сэмпл будет линейной комбинацией этих 8 сэмплов, это позволяло достигать сжатия в 8 раз. Такой метод не мог быть использован для музыки, ведь он не рассчитан для неё, в отличии от формата MP3.

MP3 это формат и аудикодек, созданный в 1990ых годах Институтом интегральных схем общества Фраунгофера (*Fraunhofer-Institut für Integrierte Schaltungen*). Он быстро приобрел популярность благодаря своей способности сильно уменьшать размер фалов, почти незаметно ухудшая их качество.

Этот алгоритм сжатия данных с потерями, поэтому результат после декомпрессии отличается от оригинального файла. Сжатие может достигать 11 раз. Такое высокое сжатие и малая потеря качества обусловлена использованием достижений психоакустики — науки о восприятии музыки человеком. Благодаря особенностям слуха человека, будь то динамическое восприятие громкости или чувствительность к определенным частотам позволяет эффективно находить избыточную для слушателя информацию.

При работе звуковая дорожка делится на «кадры» - маленькие части файла, которые обрабатываются отдельно. Алгоритм затем анализирует каждый из этих кадров для выявления избыточной информации. Например многие устройства неспособны воспроизводить очень низкие и очень высокие частоты, поэтому они удаляются.

После обработки кадров они кодируются с изменяемым битрейтом (то есть количество информации, которое в секунду проигрывается), насколько он большой определяется сложностью музыки, но не только. Например после громкого кадра, следующий тихий за ним тихий, может быть более низкого качества, ведь наш слух не успел перестроится. В общем случае более монотонные и громкие кадры используют низкий битрейт.

MP3 стал сейчас почти стандартом для потребляемой музыки, он поддерживается всеми современными устройствами и приложениями, его используют не только для музыки, но и для аудиокниг, подкастов и аудио эффектов.

Для компрессии изображения также используются особенности восприятия человека. Например формат JPEG использует тот факт, что человек лучше воспринимает яркость чем цвет. Поэтому блоки одинакового размера, содержащие информацию о цвете имеют разрешение 8 на 8, а содержащие информацию о яркости — 16 на 16. Также применяются математические преобразования вроде дискретного косинусного преобразования и другие сложные операций про которые мы не будем говорить. Можно заметить тренд на увеличение сложности алгоритмов с усложнением самой информации.

Компрессия видео сейчас является актуальной проблемой с распространением потокового видео, применяемого на прямых трансляциях, видеохостингах и онлайн-конференциях. Из-за большого объёма данных в первый видеокодек — h.261, созданный в 1988, была встроена компрессия. На данный момент самая популярная версия кодека — h.264. Который использует множество техник для компрессии видео изображения. Например предикативное кодирование, которое использует информацию из прошлого кадра, чтобы предсказать следующий и хранит только разность между ними. Использование компрессии позволяет использовать видеоконтент так, как мы его используем сейчас. Без этого наши каналы связи не могли предоставлять нужной пропускной способности для возможности смотреть потоковое видео. Если средний битрейт видеокодека h.264 для видео 1080р составляет примерно 24мб/с, то для передачи нескомпенсированного видео такого же разрешение необходимо 1.98гб/с. Поэтому для просмотра потокового видео просто необходим сложный алгоритм компрессии данных.

Мы увидели что нередко алгоритмы сжатия могут требовать большое количество ресурсов, это создает дополнительную нагрузку на центральный процессор, что может вызвать невозможность выполнять какие-либо функции ещё. Для таких применяются ускорители кодеков, то есть специальное оборудование, встроенное в комплексные устройства, вроде видеокарт, или отдельно добавляемое с помощью карт расширения. Такого рода ускорители применяются в задачах, где процесс компрессии и декомпрессии идет непрерывно, как в обработке звука и видео.

На данный момент именно графические карты применяются для ускорения вычисления, для чего в них добавляются специализированные мощности. Как раньше мы видели создание узкоспециализированного инструмента для решение задачи позволяет эффективно решать её, но лишает нас возможности использовать его для чего-либо другого. Этот компромисс необходимо делать производителям видеокарт, ведь добавляя укорители кодеков они расходуют площадь кристалла процессора. Ещё пользователям более старых устройств не представляется возможность пользоваться новыми более эффективными кодеками. Например пользователи самой популярной видеокарты «Nvidia GTX 1060» не могут использовать её для работы с кодеком AV1, который по прогнозам может сменить уже достаточно старый h.264. Со всеми недостатками именно наличие таких мощностей позволяет, например, мобильным телефонам быть такими эффективными в воспроизведении видео и аудио. Современные расходуют единичные значения мощности в ваттах, воспроизводя видео в разрешение 4k, чей битрейт на самом популярном видеохостинге — YouTube составляет примерно 60 мб/с.

В последнее время повысился интерес к использованию машинного обучения и глубокого обучения в кодеках. Это позволяет понизить нагрузку на интернет соединение или даже графический процессор. Один из вариантов использования — увеличение разрешения фотографий и видео. На основе уже существующих пикселей нейросеть создаёт новые, чтобы увеличить четкость изображения. Такая техника позволяет улучшать старые фото и видео, записанные в малом разрешении. Или же использовать такую технику в реальном времени, например в играх или просмотре видео. С помощью графического процессора отрисовывается картинка в низком разрешении, которая сразу же обрабатывается нейросетью и подается на дисплей. Второй вариант использования — это отрисовка новых кадров, чтобы заполнить пробелы между отрисованными классическим способом. Оба варианта разгрузить графический процессор, но где же в таком случае происходит вычисления для увеличения разрешения, на том же графических процессоре, но в специализированных для этого ядрах — тензорных ядрах. Это тоже специализированные вычислительные устройства, как для работы с кодеками, только применяемые для нейросетей. Такой подход выгоден и потребителям ведь во многих видеокартах эти тензорные ядра уже стоят, просто чаще всего простыми пользователями они не использовались. Для мобильных игровых консолей, вроде «Steam Deck», это позволяет понизить потребление энергии, что позволит повысить время автономной работы.

### 1.4 Перспективы развития технологий сжатия информации

Исходя из вышеописанного мы можем понять, что компрессия данных стала актуальным вопросом с самого начала развития компьютерных технологий, позволяя хранить больший объём данных, на меньшем объеме физической памяти.

На данный момент это одна из самых актуальных тем, которая вбирает в себя достижения в области производства микросхем, алгоритмов и машинного обучения. Она является обязательным для работы современного интернета, ведь на ней завязана передача больших объёмов данных, потребление и производство контента. Многие сферы без данной технологии вообще не смогли бы существовать — вроде современных смартфонов с доступом в интернет, потокового видео и видеоконференций.

В будущем необходимость в сжатии данных может только расти со стагнированием технологий производства полупроводниковых модулей памяти и процессоров. Все сложнее делать более быструю память, которая могла бы поспевать за увеличивающейся скоростью передачи данных. Технологии искусственного интеллекта тоже будут толкать развитие технологий компрессии информации. Для обучения новых моделей требуются все большие массивы данных, а их возможности будут и уже применяются для работы с данными. Поэтому технологии сжатия информации в будущем, по моему мнению, будут ещё более востребованными.

## 2. МОДЕЛЬ РАБОТЫ АРХИВАТОРА

### 2.1 Интерфейс приложения

Данное ПО позволяет создавать архивы файлов или директорий, уменьшая количество места на диске, занимаемое ими.

Программа является классическим консольным приложением, взаимодействие с которым происходит с помощью командной строки.



Рис. 1 — команда создания из файла «myfile.txt» архив с именем «archive.ssaf»

Преимущество таких приложений в быстроте использования. Для использования такого ПО не нужна мышка или другое указательное устройство. Поэтому руки не отрываются от клавиатуры, что ускоряет скорость набора текста. Такими решениями часто пользуются продвинутые разработчики. Они часто пользуются редакторами кода командной строки вроде «vim» или «emacs», которые тоже не требуют для своей работы мышку. Это экономит их дорогое время, которое может быть потрачено для набора кода.

Базовое поведение программы — при указании архива (с расширением .ssaf) рабочим файлом программа распакует архив в рабочую директорию. Если указать файл с другим расширением, то программа создаст в рабочей директории архив с расширением .ssaf. Поведение программы можно модифицировать с помощью именованных аргументов командной строки.

1. -h / --help — программа не будет совершать операции с файлами, а выведет информацию о её использовании с описанием работы каждого аргумента, подобно этому списку.

2. -p / --path <file\_path> - программа создаст файл или архив по указанному пути. Путь быть указан абсолютно или относительно текущей рабочей директории.

3. -e / --encode — программа будет пытаться архивировать указанный файл вне зависимости от его расширения.

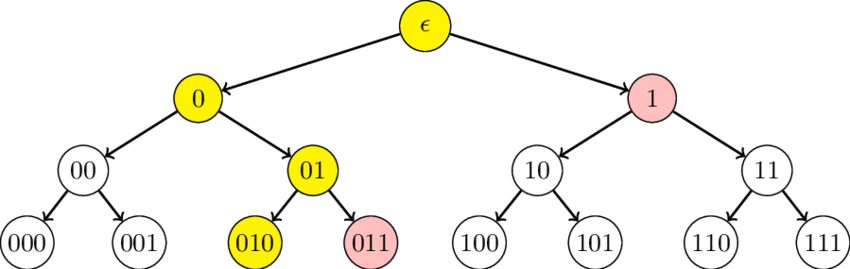
4. -d / --decode — программа будет пытаться разархивировать указанный архив вне зависимости от его расширения.

5. ---cs / --chunk\_size <uint bytes> - размер части файла в байтах, на которые файл будет разбит при обработке.

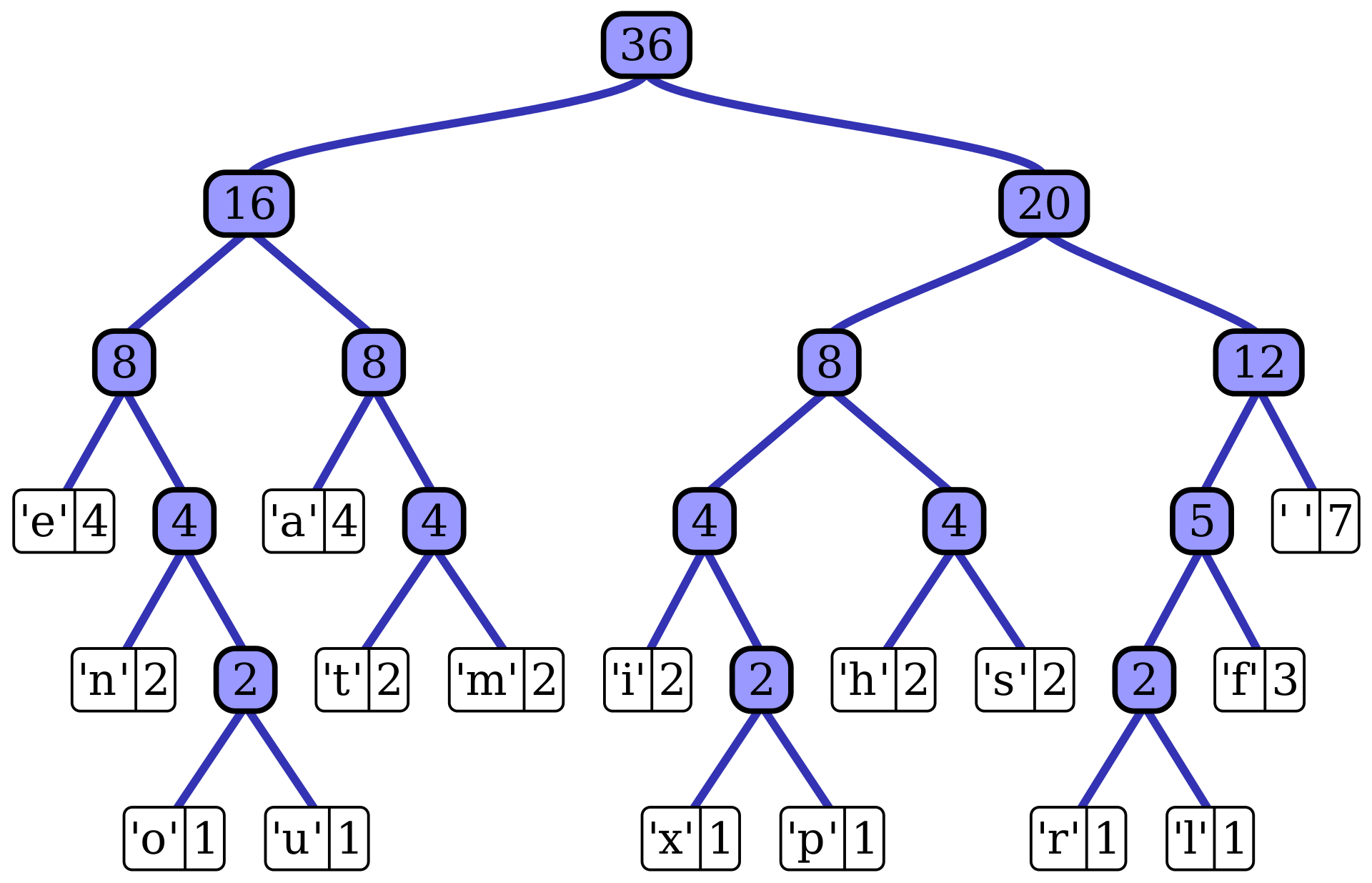
### 2.2 Описание математической модели

Приложение использует алгоритм Хаффмана для сжатия информации. Этот алгоритм является оптимальным для сжатия текста, кодируя каждый символ индивидуально. Результатом работы алгоритма является таблица кодов различной длинны, их размер определяется количеством вхождений символа в заданный текст; самому часто встречающемуся символу присваивается самый короткий код, а самому редко встречающемуся символу — самый длинный, и новым текстом, зашифрованным с помощь. Этой таблицы.

При работе алгоритма используется структура данных «двоичное дерево». Это граф одну из вершин которого называют корнем. Все точки имеют путь до корня. У каждой точки имеется один предок и максимум 2 потомка. Путь от корня до любой вершины можно представить как последовательность нулей и единиц. Путь начинается из корня, если следующим в пути будет левый потомок, то к нашему пути допишем единицу, если следующим следует правый потомок, то допишем единицу; повторяем алгоритм пока не дойдем до искомой вершины.

 Рис. 2 — пример двоичного дерева, в каждой вершине которого указан путь от корня (вершины «е») до неё самой.

В вершинах не имеющих потомков будут располагаться символы, а их пути станут кодовыми двоичными словами. Такая организация так же гарантирует условие Фано, ведь у вершин с символами не может быть потомков, а значит и кодовое слово для этого символа не может быть префиксом кодового слова другого символа.

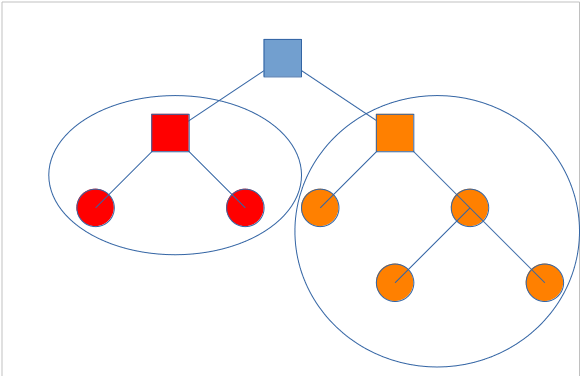
Рис. 3 — пример дерева Хаффмана(в каждой вершине указан её вес — сколько раз символ встречается в тексте для вершин с символами, для вершин без символов вес — сумма весов их потомков).

Построив дерево Хаффмана, чтобы закодировать символ будем подниматься вверх от кодируемого символа, для получения символа при прочтении текста будем двигаться от корня пока не дойдем до вершины с искомым символом.

### 2.3 Алгоритм Построения Дерева Хаффмана

В 1 главе был разобран алгоритм построения кодовых слов для кода Шеннона-Фано. Если посмотреть на него мы заметим что в результате работы мы так же получаем двоичное дерево кодов, то есть способ кодирования и декодирования при наличии дерева кодов идентичен. Отличие будет в способе построения дерева. Если использовать код Шеннона-Фано то мы начинаем построения с корня, то есть «сверху». Дерево Хаффмана же строится «снизу». Для этого применяется жадный алгоритм, принимающий оптимальное решение для текущего шага, не храня прошлые действия или предсказывая будущие.

Для построения кодового дерева мы будем объединять деревья. Объединением деревьев назовем создание новой вершины, обозначение её как нового корня, и определения корней объединяемых деревьев как его потомков.

 Рис. 4 — объединение деревьев. Квадраты обозначают корни деревьев. Обведены объединяемые деревья. Верхний квадрат — новый корень дерева.

Последовательно опишем алгоритм построения. Каждая вершина будет иметь вес и символ, если она не имеет потомков.

1. Создадим столько деревьев, сколько различных символов в кодируемом тексте. Каждое дерево будет состоять из одной вершины, с одноименным символом и весом — количеством раз, которое символ встречается в тексте. Поместим деревья во множество.

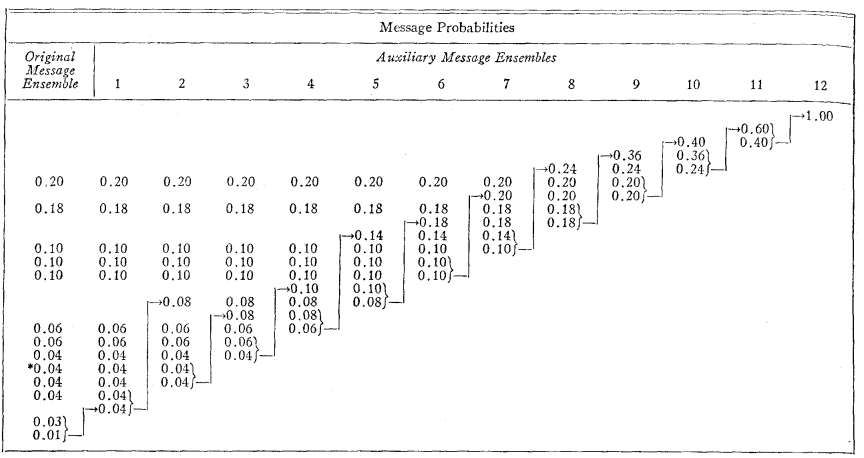
2. Выберем и уберем из множества два дерева с минимальной суммой весов вершин.

3. Объединим эти деревья, новому корню присвоим вес как сумму весов корней двух деревьев, поместим получившееся дерево во множество.

4. Если во множестве большего одного дерева вернемся к пункту 2.

5. Единственное дерево во множестве будет искомым деревом Хаффмана.

Алгоритм описан самим Дэвидом Хаффманом в его работе «*Method for the Construction of Minimum-Redundancy Codes*» (Метод построения кодов минимальной избыточности), опубликованной в сентябре 1954 года. Единственная разница с описанным выше методом использование другого определения для весов. В статье веса для вершин определены как вероятность, то есть количество вхождений символа в текст деленное на количество символов тексте. Так как делитель в вероятностях для всех элементов одинаков то определения весов в статье Хаффмана и методе, описанном выше, взаимозаменяемы.

Рис. 5 — иллюстрация из статьи «*Method for the Construction of Minimum-Redundancy Codes*»

Можем видеть процесс объединения двух минимальных весов, то есть пошаговое выполнение описанного жадного алгоритма.

### 2.4 Доказательство оптимальности алгоритма

**Лемма**: Для любого дерева Хаффмана содержащего не менее двух символов, две буквы с наименьшей частотой хранятся в родственных вершинах (то есть являющиеся потомками одного предка и находящиеся на одном уровне), глубина (дина пути от корня) которых не меньше глубины любых других вершин в дереве.

**Доказательство**: назовём два символа с наименьшей частотой L1 и L2. Они должны быть родственными, потому что мы выбираем их на первом этапе процесса построения. Предположим, что L1и L2 не являются самыми глубокими узлами в дереве. В этом случае дерево Хаффмана должно либо выглядеть так, как показано на рис. 6, либо быть симметрично ему. Для возникновения такой ситуации родитель L1 и L2, помеченный буквой V, должен иметь больший вес, чем узел с меткой X. В противном случае был бы выбран узел V, вместо узла Х, как дочерний узел узла U. Однако это невозможно, поскольку L1и L2 символы с наименьшей частотой.

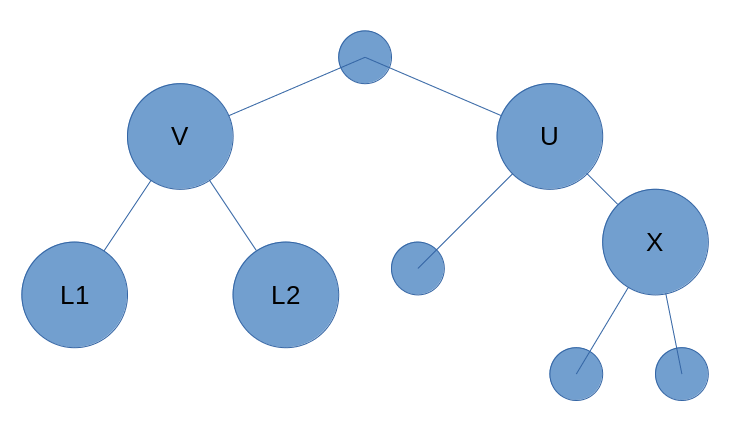


Рис. 6: Невозможное дерево Хаффмана, показывающее ситуацию, когда два узла с наименьшим весом, L1 и L2, не являются самыми глубокими узлами в дереве.

**Теорема**: Алгоритм строит дерево Хаффмана с минимальным весом внешнего пути для заданного набора букв (то есть с минимально возможной суммой произведений глубины символа в дереве на количество его вхождений в текст).

**Доказательство**. Доказательство проводится индукцией по n - количество уникальных символов.

* Базовый случай: для n = 2, дерево Хаффмана должно иметь минимальный вес внешнего пути, потому что существует только два возможных дерева, каждое с одинаковой глубиной для обоих символов.
* Гипотеза индукции: предположим, что любое дерево, содержащее n−1 вершин, имеет минимальную длину внешнего пути(то есть минимальную сумму глубин всех вершин, содержащих символ).
* Шаг индукции: дано дерево Хаффмана T с n≥2, предположим, что w1≤w2≤...≤wn, где w1 — вес самого редко встречающегося символа. Назовем родителя вершин с весами w1 и w2 — V. Из леммы мы знаем, что вершины, содержащие буквы с частотами w1 и w2 имеют глубину не меньшую, чем любые узлы в T. Если бы какие-либо другие вершины в дереве были глубже, мы могли бы уменьшить их взвешенную длину пути (произведение глубины на вес вершины), заменив их местами с w1 или w2. Но лемма говорит нам, что более глубоких узлов не существует. Назовём Т' идентичное T дерево Хаффмана, за исключением того, что узел V заменяется вершиной V ', вес которого равен w1+w2. По гипотезе индукции T′ имеет минимальную длину внешнего пути. Возвращение детей в V ′ восстанавливает дерево Т, который также должен иметь минимальную длину внешнего пути.

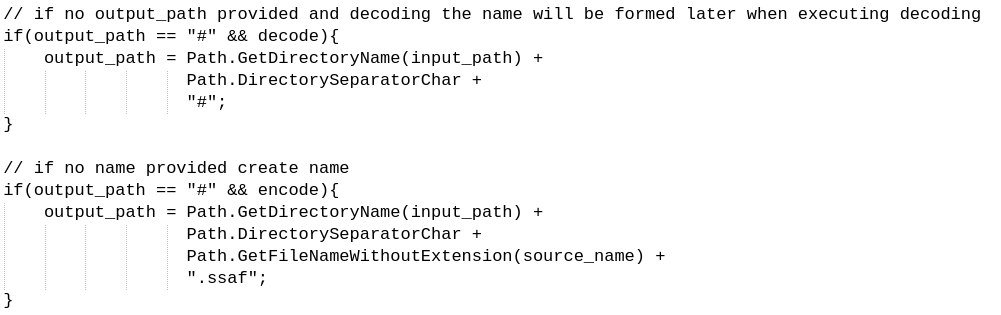
Так с помощью метода математической индукции мы доказали, что дерево Хаффмана имеет минимальный вес внешнего пути.

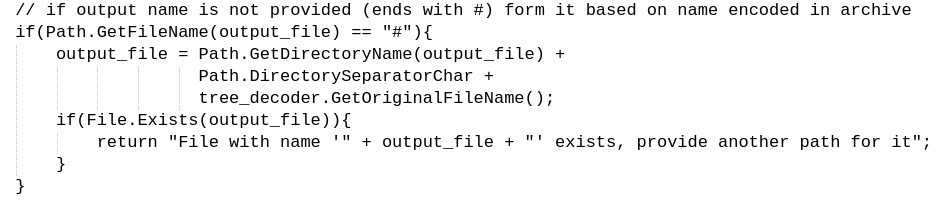
## 3. Принципы работы и тестирование архиватора

### 3.1 Работа с аргументами командной строки

Данная программа, как говорилось выше, является инструментом командной строки, то есть предназначена для работы через терминал, значит и входные данные будут передаваться через командную строку. В языке C# в стандартной библиотеке отсутствую инструменты для создания интерфейсов командной строки. Для работы с ними в программе используется класс Executor(*англ. Исполнитель*). На основе полученных аргументов формируются адреса обрабатываемого и создаваемого файла, режим работы (кодирование файла, декодирование архива), необходимость вывода сообщения со справкой о работе информации и размер части файла, обрабатываемого за один циклю. Во время формирования необходимой информации производится её валидация. Проводится проверка на существование исходного файла и отсутствие файла с таким же именем, как у создаваемого.

Для удобства пользования, если не указан путь до создаваемого файла, то программа автоматически создаст этот путь. Если исходный файл — архив, имя создаваемого файла будет восстановлено из архива, если же исходный файл не является архивом, то создаваемому файлу будет присвоено имя исходного фала с добавлением расширения «.ssaf».

Рис. 7 — фрагмент из файла lib/Executor.cs

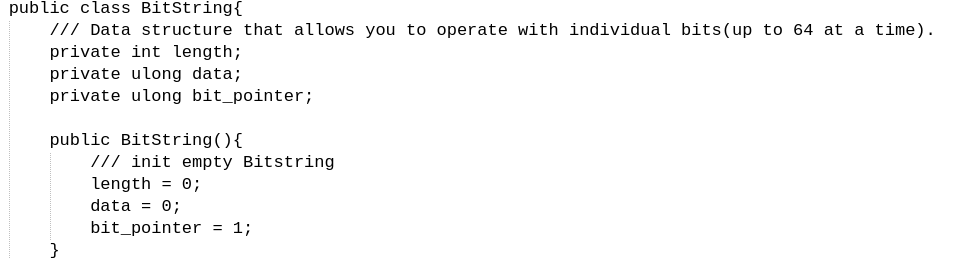
Рис 8. - фрагмент из файла lib/EncodeDecodeFiles.cs

В первом фрагменте представлены блоки, реализующие функционал представленный выше. Но можно заметить, что в случае декодирования архива вместо имени файла поставлен знак „#“. Это индикатор того, что имя для файла необходимо прочитать при обработке архива, что и представлено на 2 изображении, где представлена часть функции, декодирующей архив. Объект «tree\_decoder» считывает и декодирует информацию о файле, зашифрованном в архиве.

### 3.2 принцип хранения данных произвольной длины бит

Как описывалось во 2 главе для сжатия информации используется алгоритм Хаффмана, который создает кодирующие слова произвольной длины. Для хранения значений произвольной используется класс BitString(*англ. битовая строка*). Он позволяет хранить до 64 бит и использует битовые операции, что даёт выигрыш в производительности, если сравнивать с использованием списков. BitString использует битовые операции, которые выполняются непосредственно на процессоре, что и делает его эффективнее списков.

Класс реализует необходимый функционал для работы архиватора: добавление значений с обоих сторон, конкатенация двух битовых строк, удаление значений с обоих сторон, получение значений с обоих сторон. Для удобства также реализованы операции получения и удаления самого левого байта. Для удобства отладки реализован метод приведения объекта к виду строки.

Рис. 9 — фрагмент файла lib/BitString.cs

Состояние объекта определяется 2 переменными и 1 вспомогательной. Основными являются переменные «data» — 64 битовая целочисленная переменная без знака, в которой хранится информация о значениях объекта и «length» — целочисленное 32 битовое, обозначающее количество бит в строке. Значением строки являются первые length бит в переменной data, считая справа. Переменная «bit\_pointer» всегда имеет значение 2 в степени (length + 1). То есть бит равный единице этой переменной будет находится на бит левее самого левого значащего бита в переменной data. Переменная применяется при добавлении, удалении и получении значений слева, ведь применяя применяя битовые операции их можно выполнить в одно действие.

### 3.3 Построение дерева Хаффмана

# СПИСОК ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Stephen Wolfram - A New Kind Of Science *//* Wolfram Media - 2002

2. Shannon C. E. A Mathematical Theory of Communication // Bell System Technical Journal, июль 1948.

3. История создания формата MP3 [Электронный ресурс] // режим доступа: https://www.iis.fraunhofer.de/en/ff/amm/consumer-electronics/mp3.html

4. Возможности использования технологий Nvidia для увеличения разрешения контента в сети интернет [Электронный ресурс] // режим доступа: <https://blogs.nvidia.com/blog/2023/02/28/rtx-video-super-resolution/>

5. История развития мультимедийных кодеков [Электронный ресурс] // режим доступа: https://www.vcodex.com/historical-timeline-of-video-coding-standards-and-formats/

6. David Huffman - A Method for the Construction of Minimum-Redundancy Codes // Proceedings Of The IRE, сентябрь 1952.