Министерство образования и науки Республики Тыва
Государственное бюджетное профессиональное образовательное учреждение
Республики Тыва
«Тувинский техникум народных промыслов»

ДИПЛОМНАЯ РАБОТА

на тему

«Методы сжатия цифровой информации»

Выполнил: студентка 1 курса
Хертек А.М.
Специальности: мастер по обработке цифровой информации
Мастер п.о.: Саая С-Х.В.
Оценка_____

Кызыл 2020

Содержание

Введение

С ростом использования компьютеров в мире бизнеса и персональных компьютеров объём данных, хранящихся в Интернете, значительно вырос. Данное явление привело к необходимости сжатия данных. Передача информации с помощью Интернета имеет решающее значение для всех типов бизнес-структур на всех уровнях. По мере развития технологий скорость передачи данных также может быть достигнута путём улучшения алгоритмов сжатия, через которое передаются данные, или путём изменения формата данных, чтобы данные могли передаваться с низкой стоимостью и максимальной скоростью. Сжатие данных также полезно, поскольку оно помогает снизить потребление дорогостоящих ресурсов, таких как пространство на жёстком диске или пропускная способность передачи данных.

Сжатие данных является очень важной темой для многих приложений. Многие методы сжатия данных были изучены в течение 40 лет. Хотя алгоритмы изменились и улучшились по эффективности, они играют важную роль в повседневной обработке компьютерного бизнеса и в компьютерной науке в целом. Поскольку мы продолжаем использовать компьютеры практически для всех наших задач, потребность в сжатии данных растет с каждым днем. В последние годы компании начали предлагать услуги резервного копирования пользователям персональных компьютеров. Компьютерные данные сжимаются для всего, что мы делаем, от банковских операций до отправки текстовых сообщений. А без технологии сжатия данных все это было бы невозможно. Но поскольку сжатие данных достигло значительных успехов, эта услуга возможна без использования большого пространства на жестком диске.

Объектом данной дипломной работы является методы сжатия информации.

Предмет - анализ и определение основных методов сжатия цифровой информации.

Целью данной дипломной работы является определение и подробное изучение основных методов сжатия цифровой информации.

В соответствии с поставленной целью в работе были выявлены следующие задачи:

— рассмотрение теоретических основ понятия «информация»;

— определение основных типов методов сжатия информации;

— исследование методов сжатия без потерь и с потерями.

Работа состоит из введения, двух глав, заключения и списка литературы.

1. Теоретические основы понятия «информация»

1.1. Понятие информации и ее свойства

Ключевым понятием информатики является понятие информации, с которым мы сталкиваемся ежедневно, однако единого се определения до сих пор не существует. Поэтому вместо определения обычно используют понятие об информации. Первоначально под информацией (от дат. infbrmatio — разъяснение. изложение, сообщение, осведомление) понимались сведения, передаваемые людьми различными способами: устно, с помощью сигналов или технических средств.

Мы часто слышим и используем термин «информация», но редко задумываемся, что же это такое на самом деле.В науке есть такое понятие, как энтропия — мера неопределенности. Фактически информация уменьшает энтропию, так как увеличивает объем наших знаний. Для того чтобы это уяснить, введем понятие сообщения. Вот два сообщения:

1. В данный момент вы читаете учебник.

2. Основным запоминающим элементом компьютера является релаксационная электронная схема, называемая триггером.

Первое сообщение не несет какой-либо новизны, не сообщает ничего нового. Во втором случае сообщение содержит новизну, так как раньше вы этого не знали. А если второе сообщение передать инженеру по компьютерной технике, будет ли оно содержать для него новизну? Конечно, нет. Для разных приемников сообщений одно и то же сообщение может содержать, а может и не содержать элемент новизны.

Федеральный закон от 27.07.2006 № 149-ФЗ «Об информации, информатизации, информационных технологиях и о защите информации» определяет информацию следующим образом.

Информация — сведения (сообщения, данные) независимо от формы их представления [2].

Информатизация — организационный социально-экономический и научно-технический процесс создания оптимальных условий для удовлетворения информационных потребностей и реализации прав граждан, органов государственной власти, органов местного самоуправления, организаций, общественных объединений на основе формирования и использования информационных ресурсов.

Основные понятия, определения и термины информатики также определяются ГОСТ 15971—90 «Системы обработки информации.

Термины и определения», согласно которому, информация — это сведения о фактах, концепциях, объектах, событиях и идеях, которые в данном контексте имеют вполне определенное значение. Отметим, что информация — это не просто сведения, а сведения нужные, имеющие значение для лица, обладающего ими. В этих определениях информации отражены основные важные свойства понятия информации.

Во-первых, информация не является материальным объектом, ее передают от одного человека к другому, при этом первый ее не утрачивает. В результате передачи оба эти человека будут владеть переданной информацией. Информация — единственный ресурс, который при передаче не уменьшается, а только увеличивается.

Во-вторых, для передачи информация должна быть представлена на каком-нибудь материальном носителе.

В-третьих, содержание информации должно быть неизменным при ее переносе с одного носителя информации на другой.

Сообщение от источника к приемнику передается в материально-энергетической форме (электрический, световой, в виде звуковых сигналов и т.д.). В зависимости от вида сигнала, определяемого свойствами передающего устройства, различают непрерывную (аналоговую) и дискретную (цифровую) информацию.

Источником аналоговой информации обычно являются различные природные объекты (например, температура, давление и влажность воздуха),

объекты технологических производственных процессов (например, нейтронный поток в активной зоне, давление и температура теплоносителя в контурах ядерного реактора) и др.

Информационные сообщения, используемые человеком, имеют характер дискретных сообщений, например сигналы тревоги, передаваемые посредством световых сообщений, телеграфные сигналы, языковые сообщения, передаваемые в письменном виде или с помощью звуковых сигналов и др.

Человек воспринимает сообщения при помощи органов чувств, и, как правило, в основном это непрерывная информация, а вот логическое мышление человека имеет, скорее, дискретный характер.

войства информации. При работе с информацией и разработке информационных систем и технологий важно оценить свойства поступающей, хранимой и передаваемой информации

Сформулируем следующие определения свойств информации.

Адекватность— свойство информации однозначно соответствовать отображаемому объекту или явлению.

Достоверность — свойство информации не иметь скрытых ошибок.

Полнота — свойство информации исчерпывающе характеризовать отображаемый объект или процесс. ^

Доступность — свойство информации, характеризующее возможность се получения данным пользователем.

Релевантность — способность информации соответствовать запросам пользователя.

Качество информации — обобщенная положительная характеристика информации, отражающая степень ее полезности.

Актуальность информации — степень соответствия информации текущему моменту времени. Нередко с актуальностью, как и с полнотой, связывают коммерческую ценность информации,

Существуют еще и другие, менее существенные свойства информации.

Краткость и четкость информации — отсутствие в информации ненужных сведении.

Ценность — степень важности информации для решения задачи.

Понятность — выражение информации на языке, понятном чем, кому она предназначена.

Своевременность — актуальность информации и наличие в ней сведений, необходимых в данный момент для понимания и принятия решения [5].

1.2. Виды информации и ее кодирование

Виды информации. Обычно для классификации объектов одной природы используется то или иное свойство либо набор свойств объектов. Нас интересует классификация информации в плане автоматизации основных информационных процессов.

Первоначально вычислительные машины применялись только для обработки числовой информации, однако довольно быстро выяснилось, что их возможности не ограничиваются только работой с числами.

Далеко не вся информация окружающего нас мира может быть обработала компьютером, ведь пока не придумали такого компьютера, который мог бы чувствовать или наслаждаться произведениями искусства. Поэтому, говоря об информации, необходимо выделить тс се вилы, которые компьютер воспримет и позволит человеку использовать свои ресурсы для обработки, хранения и передачи такой информации.

Компьютер может работать с текстовой, числовой, табличной, графической информацией, а также со звуковой, анимационной и видеоинформацией. Также компьютер воспринимает специальную двоичную информацию.

В настоящее время практически все компьютерные технологии ограничиваются обработкой перечисленных видов информации. С развитием компьютерной техники увеличиваются объемы перерабатываемой

информации. И хотя современные компьютеры могут делать очень много, все же их возможности не безграничны. Наибольший эффект от применения компьютера будет там, где оправдано его применение.

Кодирование информации — это преобразование одной последовательности сигналов в другую. Под кодированием данных понимается выражение данных одного типа через данные другого типа. Для автоматизации работы с данными, относящимися к различным типам, очень важно унифицировать их форму представления, поэтому обычно используется прием кодирования [7].

Человеческий язык — это система кодирования понятий для выражения мыслей посредством речи. Азбуки — системы кодирования компонентов языка с помощью графических символов.

Своя системы существует и в вычислительной технике. Она называется двоичным кодированием и основана на представлении данных последовательностью всего двух цифр: 0 и 1.

Для представления дискретной информации в компьютере применяется алфавитный способ, основанный на использовании фиксированного конечного набора символов (алфавита). Примерами алфавитов могут служить алфавиты естественных человеческих языков, совокупность десятичных цифр, любая другая упорядоченность знаков, предназначенная для образования и передачи сообщений. Символы из набора алфавита называются буквами, а любая конечная последовательность букв — словом в этом алфавите. При этом не требуется, чтобы слово обязательно имело языковое смысловое значение.

Процесс преобразования информации часто требует представлять буквы одного алфавита средствами (буквами, словами) другого алфавита. Такое представление и называется кодированием. Процесс обратного преобразования информации относительно ранее выполненного кодирования называется декодированием.

Предыстория кодирования информации. Люди общаются в основном с помощью сказанных или написанных слои. Эта система норм&1ьно работает, когда пес участники находятся поблизости друг от друга (в пределах слышимости или видимости), А если мы хотим связаться с удаленным собеседником? С древних времен до XIX в. для этой цели использовались курьеры с устными или письменными сообщениями. Такая связь работала неплохо, хотя часто слишком медленно; к тому же сообщение или курьер до адресата порой не доходили.

Шло время, развивались технологии, и люди изобретали различные коммуникационные приспособления. В доиндустриальную эпоху для передачи сообщений на большие расстояния использовали устройства наподобие маяков. Индейцы Северной Америки применяли дымовые сигналы, в армиях для передачи сообщений использовали флаги и зеркала. Создавались и хитроумные механизмы для передачи сообщений на все увеличивающиеся расстояния.

Техническая революция сопровождалась распространением электричества и телеграфа, позволявшего мгновенно передавать сообщения на очень большие расстояния по одному проводу. Теперь уже не нужно было видеть человека на другом конце провода или посылать к нему посредника-почтальона. Телеграф и дымовые сигналы имеют одно общее свойство — им требуется некоторый код, чтобы перевести человеческий язык в форму, которую мог бы передать механизм или телеграфный аппарат. На принимающем конце этот код необходимо перевести обратно па человеческий язык. Уже в ранних коммуникационных устройствах сформировались две идеи, которые легли в основу современных компьютеров [12]:

1)цифровой (digital), т.е. дискретный, код, основанный па двух состояниях (включено—выключено, или 0 и I);

2)специализированный машинный язык (обычно цифровой), используемый машиной для обработки данных.

Телеграф и первые радиостанции применяли для передачи сообщений специальный кол — азбуку Морзе, названную но имени ее создателя Сэмюэла Ф. Б. Морзе. В ней с каждой буквой алфавита сопоставлена комбинация точек (коротких импульсов) и тире (длинных импульсов). Импульсы передаются по проводам в определенной последовательности, которую оператор на принимающем устройстве переводит обратно в буквы и слова. Как правило, оператор использует справочник по кодам, но опытные операторы знают код настолько хорошо, что могут расшифровывать каждый символ по памяти.

Современные компьютеры похожи на ранний телеграф, ведь они передают информацию по проводам в цифровой форме, используя специальный код. Но если основная задача телеграфа передавать информацию на далекие расстояния, то компьютер перелает данные внутри себя. При этом компьютер использует другой кодовый язык и несколько проводов, а не один, как телеграф.

Кодирование данных двоичным кодом. На современном языке телеграф можно назвать устройством для цифровой последовательной связи. Связь является цифровой, потому что в ней используется дискретный (включено—выключено) код; последовательной, потому что элементы языка (точки и тире) отправляются последовательно один за другим. Если мы разработаем код, в котором каждая буква алфавита будет представлена комбинацией из восьми элементов (0 или 1), и будем отправлять их один за другим, то мы создадим цифровое последовательное устройство. При наличии единственного провода такой способ связи работает прекрасно, но медленно (ведь нам приходится посылать но очереди восемь единиц информации, чтобы передать одну букву). А если вместо одного у нас было бы восемь проводов? Тогда мы могли бы передать все восемь элементов сразу, или параллельно. Именно так данные передаются в компьютере.

Кодирование может производиться без потери и с потерями информации. Так, преобразование принципиально различных видов информации — непрерывной в дискретную (аналого-цифровое преобразование (АЦП)) и дискретной в непрерывную (цифро-аналоговое преобразование (ЦАП)) — возможно только с потерей информации.

К кодированию можно отнести и сжатие (архивацию) информации. Сжатие — это устранение избыточности информации, например за счет упрощения кодов путем исключения из них постоянных битов.

Другой разновидностью кодирования является введение избыточной информации, что широко применяется в криптографии. Примерами такого кодирования могут служить электронный сертификат, цифровая подпись и шифрование [15].

1. Методы сжатия информации

2.1. Методы сжатия информации без потерь

Все известные алгоритмы сжатия сводятся к шифрованию входной информации, а принимающая сторона выполняет дешифровку принятых данных.

Существуют методы, которые предполагают некоторые потери исходных данных, другие алгоритмы позволяют преобразовать информацию без потерь. Сжатие с потерями используется при передаче звуковой или графической информации, при этом учитывается несовершенство органов слуха и зрения, которые не замечают некоторого ухудшения качества, связанного с этими потерями.

Все известные алгоритмы сжатия сводятся к шифрованию входной информации, а принимающая сторона выполняет дешифровку принятых данных.

Сжатие без потерь — метод сжатия информации представленной в цифровом виде, при использовании которого закодированная информация может быть восстановлена с точностью до бита. При этом оригинальные данные полностью восстанавливаются из сжатого состояния. Этот тип сжатия принципиально отличается от сжатия данных с потерями. Для каждого из типов цифровой информации, как правило, существуют свои оптимальные алгоритмы сжатия без потерь.

Сжатие данных без потерь используется во многих приложениях. Например, оно используется во всех файловых архиваторах. Оно также используется как компонент в сжатии с потерями [20].

Сжатие без потерь используется, когда важна идентичность сжатых данных оригиналу. Обычный пример — исполняемые файлы и исходный код. Некоторые графические файловые форматы, такие как PNG , используют только сжатие без потерь; тогда как другие (TIFF, MNG) или GIF могут использовать сжатие как с потерями, так и без.

Сжатие информации без потерь осуществляется статистическим

кодированием или на основе предварительно созданного словаря. Статистические алгоритмы (например схема кодирования Хафмана) присваивают каждому входному символу определенный код. При этом, наиболее часто используемому символу присваивается наиболее короткий код, а наиболее редкому - более длинный. Таблицы кодирования создаются заранее и имеют ограниченный размер. Этот алгоритм обеспечивает наибольшее быстродействие и наименьшие задержки. Для получения высоких коэффициентов сжатия статистический метод требует больших объемов памяти.

Альтернативой статистическому алгоритму является схема сжатия, основанная на динамически изменяемом словаре. Данный метод предполагает замену потока символов кодами, записанными в памяти в виде словаря (таблица перекодировки). Соотношение между символами и кодами меняется вместе с изменением данных. Таблицы кодирования периодически меняются, что делает метод более гибким. Размер небольших словарей лежит в пределах 2-32 килобайт, но более высоких коэффициентов сжатия можно достичь при заметно больших словарях до 400 килобайт [19].

2.1.1. Кодирование длин сессий

Кодирование длин серий (англ. Run-length encoding, RLE) или Кодирование повторов — простой алгоритм сжатия данных, который оперирует сериями данных, то есть последовательностями, в которых один и тот же символ встречается несколько раз подряд. При кодировании строка одинаковых символов, составляющих серию, заменяется строкой, которая содержит сам повторяющийся символ и количество его повторов.

Рассмотрим изображение, содержащее простой чёрный текст на сплошном белом фоне. Здесь будет много серий белых пикселей в пустых местах, и много коротких серий чёрных пикселей в тексте. В качестве примера приведена некая произвольная строка изображения в черно-белом

варианте. Здесь B представляет чёрный пиксель, а W обозначает белый:

WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWW
WWWWWWWWWWWWWWBWWWWWWWWWWWWWW

Если мы применим простое кодирование длин серий к этой строке, то получим следующее:

12W1B12W3B24W1B14W

Последняя запись интерпретируется как «двенадцать W», «одна B», «двенадцать W», «три B» и т. д. Таким образом, код представляет исходные 67 символов в виде всего лишь 18.

Однако, в случае, если строка состоит из большого количества неповторяющихся символов, её объем может вырасти.

ABCABCABCABCDDEFFFFFFFFF

1A1B1C1A1B1C1A1B1C1A1B1C2D1E8F

Проблема решается достаточно просто. Алфавит, в котором записаны длины серий, разделяется на две (обычно равные) части. Алфавит целых чисел можно, например, разделить на две части: положительные и отрицательные. Положительные используют для записи количества повторяющихся одинаковых символов, а отрицательные — для записи количества неодинаковых.

-12ABCABCABCABC2D1E8F

Так как численные типы данных на компьютере всегда имеют некоторый предел, возникает еще одна проблема. Предположим, мы используем signed char для записи длин серий. Тогда мы не можем записать серию длиннее 127 символов одной парой "длина-символ". Если подряд записано 256 символов A, их разделяют на минимальное количество групп:

127A127A2A

Запись на некотором языке программирования алгоритма RLE с учетом этих ограничений нетривиальна.

Конечно, кодирование, которое используется для хранения изображений, оперирует с двоичными данными, а не с символами ASCII, как

в рассмотренном примере, однако принцип остаётся тот же.

Очевидно, что такое кодирование эффективно для данных, содержащих большое количество серий, например, для простых графических изображений, таких как иконки и графические рисунки. Однако это кодирование плохо подходит для изображений с плавным переходом тонов, таких как фотографии.

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» [1].

2.1.2. Алгоритм LZ78-LZW84

Алгоритм LZ78, предложенный в 1978 г. Лемпелом и Зивом, нашел свое практическое применение только после реализации LZW84, предложенной Велчем в 1984 г.

Словарь является расширяющимся (expanding). Первоначально в нем содержится только 256 строк длиной в одну букву-все коды ASCII. В процессе работы словарь разрастается до своего максимального объема $|V_{max}|$ строк (слов). Обычно, объем словаря достигает нескольких десятков тысяч слов. Каждая строка в словаре имеет свою известную длину и этим похожа на привычные нам книжные словари и отличается от строк LZ77, которые допускали использование подстрок. Таким образом, количество слов в словаре точно равно его текущему объему. В процессе работы словарь пополняется по следующему закону:

1. В словаре ищется слово str, максимально совпадающее с текущим кодируемым словом в позиции pos исходного текста. Так как словарь первоначально не пустой, такое слово всегда найдется;

2. В выходной файл помещается номер найденного слова в словаре position и следующий символ из входного текста B (на котором обнаружилось различие) - <position,B>. Длина кода равна |position|+|B|| =[logVmax]+8 (бит);

3. Если словарь еще не полон, новая строка strB добавляется в словарь по адресу last_position, размер словаря возрастает на одну позицию;

4. Указатель в исходном тексте pos смещается на |strB|=|str|+1 байт дальше к символу, следующему за B.

В таком варианте алгоритм почти не нашел практического применения и был значительно модернизирован. Изменения коснулись принципов управления словарем (его расширения и обновления) и способа формирования выходного кода:

так как словарь увеличивается постепенно и одинаково для кодировщика и декодировщика, для кодирования позиции нет необходимости использовать $[\log V_{max}]$ бит, а можно брать лишь [logV] бит, где V-текущий объем словаря.

Самая серьезная проблема LZ78-переполнение словаря: если словарь полностью заполнен, прекращается его обновление и процесс сжатия может быть заметно ухудшен (метод FREEZE). Отсюда следует вывод-словарь нужно иногда обновлять. Самый простой способ как только словарь заполнился его полностью обновляют. Недостаток очевиден кодирование начинается на пустом месте, как бы с начала, и пока словарь не накопится сжатие будет незначительным, а дальше-замкнутый цикл опять очистка словаря!.. Поэтому предлагается словарь обновлять не сразу после его заполнения, а только после того, как степень сжатия начала падать (метод FLUSH). Более сложные алгоритмы используют два словаря, которые заполняются синхронно, но с задержкой на |V|/2 слов один относительно

другого. После заполнения одного словаря, он очищается, а работа переключается на другой (метод SWAP). Еще более сложными являются эвристические методы обновления словарей в зависимости от частоты использования тех или иных слов (LRU, TAG).

Выходной код также формируется несколько иначе (сравните с предыдущим описанием):

1. В словаре ищется слово str, максимально совпадающее с текущим кодируемым словом в позицииpos исходного текста;

2. В выходной файл помещается номер найденного слова в словаре <positior>. Длина кода равна |position|=[logV] (бит);

3. Если словарь еще не полон, новая строка strB добавляется в словарь по адресу last_position, размер словаря возрастает на одну позицию;

4. Указатель в исходном тексте pos смещается на |str| байт дальше к символу B [6].

2.1.3. Алгоритм LZW

Алгоритм Лемпеля — Зива — Велча (LZW) — это универсальный алгоритм сжатия данных без потерь, созданный Абрахамом Лемпелем, Якобом Зивом и Терри Велчем. Он был опубликован Велчем в 1984 году, в качестве улучшенной реализации алгоритма LZ78, опубликованного Лемпелем и Зивом в 1978 году. Алгоритм разработан так, чтобы его можно было быстро реализовать, но он не обязательно оптимален, поскольку он не проводит никакого анализа входных данных.

Данный алгоритм при сжатии (кодировании) динамически создаёт таблицу преобразования строк: определённым последовательностям символов (словам) ставятся в соответствие группы бит фиксированной длины (обычно 12-битные). Таблица инициализируется всеми 1-символьными строками (в случае 8-битных символов — это 256 записей). По мере кодирования, алгоритм просматривает текст символ за символом, и сохраняет каждую новую, уникальную 2-символьную строку в таблицу в

виде пары код/символ, где код ссылается на соответствующий первый символ. После того как новая 2-символьная строка сохранена в таблице, на выход передаётся код первого символа. Когда на входе читается очередной символ, для него по таблице находится уже встречавшаяся строка максимальной длины, после чего в таблице сохраняется код этой строки со следующим символом на входе; на выход выдаётся код этой строки, а следующий символ используется в качестве начала следующей строки.

Алгоритму декодирования на входе требуется только закодированный текст, поскольку он может воссоздать соответствующую таблицу преобразования непосредственно по закодированному тексту.

Алгоритм:

1. Инициализация словаря всеми возможными односимвольными фразами. Инициализация входной фразы w первым символом сообщения.
2. Считать очередной символ K из кодируемого сообщения.
3. Если КОНЕЦ_СООБЩЕНИЯ, то выдать код для w, иначе
4. Если фраза wK уже есть в словаре, присвоить входной фразе значение wK и перейти к Шагу 2, иначе выдать код w, добавить wK в словарь, присвоить входной фразе значение K и перейти к Шагу 2.

На момент своего появления алгоритм LZW давал лучший коэффициент сжатия, для большинства приложений, чем любой другой хорошо известный метод того времени. Он стал первым широко используемым на компьютерах методом сжатия данных.

Алгоритм был реализован в программе compress, которая стала более или менее стандартной утилитой Unix-систем приблизительно в 1986 году. Несколько других популярных утилит-архиваторов также используют этот метод или близкие к нему [11].

В 1987 году алгоритм стал частью стандарта на формат изображений GIF. Он также может (опционально) использоваться в формате TIFF. В настоящее время, алгоритм содержится в стандарте PDF.

2.1.4. Алгоритм FLAC

FLAC — популярный свободный кодек, предназначенный для сжатия аудиоданных без потерь. В отличие от аудио-кодеков, обеспечивающих сжатие с потерями (MP3, AAC, WMA, Ogg Vorbis) FLAC не удаляет никакой информации из аудиопотока и подходит как для прослушивания музыки на высококачественной звуковоспроизводящей аппаратуре, так и для архивирования аудиоколлекции.

На сегодня формат FLAC поддерживается множеством аудиоприложений, а также имеет большое количество аппаратных реализаций.

Основными частями потока являются:

1. Строка из четырёх байтов «fLaC»
2. Блок метаданных STREAMINFO
3. Другие необязательные блоки метаданных
4. Аудио фреймы

Первые четыре байта идентифицируют поток FLAC. Следующие за ними метаданные содержат информацию о потоке, затем идут сжатые аудиоданные.

По состоянию на 10.03.2010 в libflac-1.2.1 определены следующие типы блоков: StreamInfo, Padding, Application, SeekTable, VorbisComment, CueSheet, Picture, Unknown. Блоки метаданных могут быть любого размера, новые блоки могут быть легко добавлены. Декодер имеет возможность пропускать неизвестные ему блоки метаданных. Блок STREAMINFO является обязательным. В нём содержатся данные, позволяющие декодеру настроить буферы, частота дискретизации, количество каналов, количество бит на семпл и количество семплов. Также в блок записывается подпись MD5 несжатых аудиоданных. Это полезно для проверки всего потока после его передачи [17].

Другие блоки предназначены для резервирования места, хранения таблиц точек поиска, тегов, список разметки аудиодисков, а также данных для конкретных приложений. Опции для добавления блоков PADDING или точек поиска приведены ниже. FLAC не нуждается в точках поиска, однако они позволяют значительно увеличить скорость доступа, а также могут быть использованы для расстановки меток в аудио редакторах. Точное описание структур стандартных блоков можно найти в файле format.h библиотеки libflac, доступной с сайта формата.

За метаданными следуют сжатые аудиоданные. Метаданные и аудиоданные не чередуются. Как и большинство кодеков, FLAC делит входной поток на блоки и кодирует их независимо друг от друга. Блок упаковывается во фрейм и добавляется к потоку. Базовый кодер использует блоки постоянного размера для всего потока, однако формат предусматривает наличие блоков разной длины в потоке.

Размер блока — очень важный параметр для кодирования. Если он очень мал, то в потоке будет слишком много заголовков фреймов, что уменьшит уровень сжатия. Если размер большой, то кодер не сможет подобрать эффективную модель сжатия. Понимание процесса моделирования поможет Вам увеличить уровень сжатия для некоторых типов входных данных. Обычно при использовании линейного прогнозирования на аудиоданных с частотой дискретизации 44.1 кГц оптимальный размер блока лежит в диапазоне 2-6 тысяч семплов.

Если на вход поступают стерео аудиоданные, они могут пройти через стадию межканальной декорреляции. Правый и левый канал преобразуются к среднему и разностному по формулам: средний = (левый + правый)/2, разностный = левый — правый. В отличие от joint stereo, используемом в lossy кодерах, в lossless кодировании этот процесс не приводит к потерям. Для данных с аудио компакт-дисков это обычно приводит к значительному увеличению уровня сжатия.

На следующем этапе кодер пытается аппроксимировать сигнал такой

функцией, чтобы полученный после её вычитания из оригинала результат (называемый разностью, остатком, ошибкой) можно было закодировать минимальным количеством битов. Параметры функций тоже должны записываться, поэтому они не должны занимать много места. FLAC использует два метода формирования аппроксимаций [9]:

- подгонка простого полинома к сигналу
- общее кодирование с линейными предикторами (LPC).

Во-первых, постоянное полиномиальное предсказание (-l 0) работает значительно быстрее, но менее точно, чем LPC. Чем выше порядок LPC, тем медленнее, но лучше будет модель. Однако с увеличением порядка выигрыш будет все менее значительным. В некоторой точке (обычно около 9) процедура кодера, определяющая наилучший порядок, начинает ошибаться и размер получаемых фреймов возрастает. Чтобы преодолеть это, можно использовать полный перебор, что приведёт к значительному увеличению времени кодирования.

Во-вторых, параметры для постоянных предикторов могут быть описаны тремя битами, а параметры для модели LPC зависят от количества бит на семпл и порядка LPC. Это значит, что размер заголовка фрейма зависит от выбранного метода и порядка и может повлиять на оптимальный размер блока.

Когда модель подобрана, кодер вычитает приближение из оригинала, чтобы получить остаточный (ошибочный) сигнал, который затем кодируется без потерь. Для этого используется то обстоятельство, что разностный сигнал обычно имеет распределение Лапласа и есть набор энтропийных кодов, называемый кодами Райса, позволяющий эффективно и быстро кодировать эти сигналы без использования словаря.

Кодирование Райса состоит из нахождения одного параметра, отвечающего распределению сигнала, а затем использования его для составления кодов. При изменении распределения меняется и оптимальный параметр, поэтому имеется метод позволяющий пересчитывать его по

необходимости. Остаток может быть разбит на контексты или разделы, у каждого из которых будет свой параметр Райса. FLAC позволяет указать, как нужно производить разбиение. Остаток может быть разбит на 2^n разделов.

Аудиофрейму предшествует заголовок, который начинается с кода синхронизации и содержит минимум информации, необходимой декодеру для воспроизведения потока. Сюда также записывается номер блока или семпла и восьмибитная контрольная сумма самого заголовка. Код синхронизации, CRC заголовка фрейма и номер блока/семпла позволяют осуществлять пересинхронизацию и поиск даже в отсутствие точек поиска. В конце фрейма записывается его шестнадцатибитная контрольная сумма. Если базовый декодер обнаружит ошибку, будет сгенерирован блок тишины [3].

2.1.5. Код Хаффмана

Алгоритм Хаффмана — адаптивный жадный алгоритм оптимального префиксного кодирования алфавита с минимальной избыточностью. Был разработан в 1952 годуаспирантом Массачусетского технологического института Дэвидом Хаффманом при написании им курсовой работы. В настоящее время используется во многих программах сжатия данных.

В отличие от алгоритма Шеннона — Фано, алгоритм Хаффмана остаётся всегда оптимальным и для вторичных алфавитов m_2 с более чем двумя символами.

Этот метод кодирования состоит из двух основных этапов:

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

Один из первых алгоритмов эффективного кодирования информации был предложен Д. А. Хаффманом в 1952 году. Идея алгоритма состоит в следующем: зная вероятности символов в сообщении, можно описать процедуру построения кодов переменной длины, состоящих из целого

количества битов. Символам с большей вероятностью ставятся в соответствие более короткие коды. Коды Хаффмана обладают свойством префиксности, что позволяет однозначно их декодировать.

Классический алгоритм Хаффмана на входе получает таблицу частот встречаемости символов в сообщении. Далее на основании этой таблицы строится дерево кодирования Хаффмана (Н-дерево).

1. Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который может быть равен либо вероятности, либо количеству вхождений символа в сжимаемое сообщение.

2. Выбираются два свободных узла дерева с наименьшими весами.

3. Создается их родитель с весом, равным их суммарному весу.

4. Родитель добавляется в список свободных узлов, а два его потомка удаляются из этого списка.

5. Одной дуге, выходящей из родителя, ставится в соответствие бит 1, другой — бит 0.

6. Шаги, начиная со второго, повторяются до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева [10].

Допустим, у нас есть следующая таблица частот (см.табл.1) :

Таблица 1

15	7	6	5	5
А	Б	В	Г	Д

Этот процесс можно представить как построение дерева, корень которого — символ с суммой вероятностей объединенных символов, получившийся при объединении символов из последнего шага, его n_0 потомков — символы из предыдущего шага и т. д.

Чтобы определить код для каждого из символов, входящих в сообщение, мы должны пройти путь от листа дерева, соответствующего

этому символу, до корня дерева, накапливая биты при перемещении по ветвям дерева. Полученная таким образом последовательность битов является кодом данного символа, записанным в обратном порядке.

Для данной таблицы (табл.2) символов коды Хаффмана будут выглядеть следующим образом.

Таблица 2

А	Б	В	Г	Д
0	100	101	110	111

Поскольку ни один из полученных кодов не является префиксом другого, они могут быть однозначно декодированы при чтении их из потока. Кроме того, наиболее частый символ сообщения А закодирован наименьшим количеством бит, а наиболее редкий символ Д — наибольшим.

При этом общая длина сообщения, состоящего из приведённых в таблице символов, составит 87 бит (в среднем 2,2308 бита на символ). При использовании равномерного кодирования общая длина сообщения составила бы 117 бит (ровно 3 бита на символ). Заметим, что энтропия источника, независимым образом порождающего символы с указанными частотами составляет ~2,1858 бита на символ, т.е. избыточность построенного для такого источника кода Хаффмана, понимаемая, как отличие среднего числа бит на символ от энтропии, составляет менее 0,05 бит на символ.

Классический алгоритм Хаффмана имеет ряд существенных недостатков. Во-первых, для восстановления содержимого сжатого сообщения декодер должен знать таблицу частот, которой пользовался кодер. Следовательно, длина сжатого сообщения увеличивается на длину таблицы частот, которая должна посылаться впереди данных, что может свести на нет все усилия по сжатию сообщения. Кроме того, необходимость наличия полной частотной статистики перед началом собственно кодирования

требует двух проходов по сообщению: одного для построения модели сообщения (таблицы частот и Н-дерева), другого для собственно кодирования. Во-вторых, избыточность кодирования обращается в ноль лишь в тех случаях, когда вероятности кодируемых символов являются обратными степенями числа 2. В-третьих, для источника с энтропией, не превышающей 1 (например, для двоичного источника), непосредственное применение кода Хаффмана бессмысленно.

Адаптивное сжатие позволяет не передавать модель сообщения вместе с ним самим и ограничиться одним проходом по сообщению как при кодировании, так и при декодировании.

В создании алгоритма адаптивного кодирования Хаффмана наибольшие сложности возникают при разработке процедуры обновления модели очередным символом. Теоретически можно было бы просто вставить внутрь этой процедуры полное построение дерева кодирования Хаффмана, однако, такой алгоритм сжатия имел бы неприемлемо низкое быстродействие, так как построение Н-дерева — это слишком большая работа и производить её при обработке каждого символа неразумно. К счастью, существует способ модифицировать уже существующее Н-дерево так, чтобы отобразить обработку нового символа.

Обновление дерева при считывании очередного символа сообщения состоит из двух операций.

Первая — увеличение веса узлов дерева. Вначале увеличиваем вес листа, соответствующего считанному символу, на единицу. Затем увеличиваем вес родителя, чтобы привести его в соответствие с новыми значениями веса потомков. Этот процесс продолжается до тех пор, пока мы не доберемся до корня дерева. Среднее число операций увеличения веса равно среднему количеству битов, необходимых для того, чтобы закодировать символ.

Вторая операция — перестановка узлов дерева — требуется тогда, когда увеличение веса узла приводит к нарушению свойства

упорядоченности, то есть тогда, когда увеличенный вес узла стал больше, чем вес следующего по порядку узла. Если и дальше продолжать обрабатывать увеличение веса, двигаясь к корню дерева, то дерево перестанет быть деревом Хаффмана.

Чтобы сохранить упорядоченность дерева кодирования, алгоритм работает следующим образом. Пусть новый увеличенный вес узла равен W+1. Тогда начинаем двигаться по списку в сторону увеличения веса, пока не найдем последний узел с весом W. Переставим текущий и найденный узлы между собой в списке, восстанавливая таким образом порядок в дереве (при этом родители каждого из узлов тоже изменятся). На этом операция перестановки заканчивается.

После перестановки операция увеличения веса узлов продолжается дальше. Следующий узел, вес которого будет увеличен алгоритмом, — это новый родитель узла, увеличение веса которого вызвало перестановку [13].

В процессе работы алгоритма сжатия вес узлов в дереве кодирования Хаффмана неуклонно растет. Первая проблема возникает тогда, когда вес корня дерева начинает превосходить вместимость ячейки, в которой он хранится. Как правило, это 16-битовое значение и, следовательно, не может быть больше, чем 65535. Вторая проблема, заслуживающая ещё большего внимания, может возникнуть значительно раньше, когда размер самого длинного кода Хаффмана превосходит вместимость ячейки, которая используется для того, чтобы передать его в выходной поток. Декодеру все равно, какой длины код он декодирует, поскольку он движется сверху вниз по дереву кодирования, выбирая из входного потока по одному биту. Кодер же должен начинать от листа дерева и двигаться вверх к корню, собирая биты, которые нужно передать. Обычно это происходит с переменной типа «целое», и, когда длина кода Хаффмана превосходит размер типа «целое» в битах, наступает переполнение.

Можно доказать, что максимальную длину код Хаффмана для сообщений с одним и тем же входным алфавитом будет иметь, если частоты

символов образует последовательность Фибоначчи. Сообщение с частотами символов, равными числам Фибоначчи до Fib (18), — это отличный способ протестировать работу программы сжатия по Хаффману.

Принимая во внимание сказанное выше, алгоритм обновления дерева Хаффмана должен быть изменен следующим образом: при увеличении веса нужно проверять его на достижение допустимого максимума. Если мы достигли максимума, то необходимо «масштабировать» вес, обычно разделив вес листьев на целое число, например, 2, а потом пересчитав вес всех остальных узлов.

Однако при делении веса пополам возникает проблема, связанная с тем, что после выполнения этой операции дерево может изменить свою форму. Объясняется это тем, что мы делим целые числа и при делении отбрасываем дробную часть.

Правильно организованное дерево Хаффмана после масштабирования может иметь форму, значительно отличающуюся от исходной. Это происходит потому, что масштабирование приводит к потере точности нашей статистики. Но со сбором новой статистики последствия этих «ошибок» практически сходят на нет. Масштабирование веса — довольно дорогостоящая операция, так как она приводит к необходимости заново строить все дерево кодирования. Но, так как необходимость в ней возникает относительно редко, то с этим можно смириться.

Выигрыш от масштабирования

Масштабирование веса узлов дерева через определенные интервалы дает неожиданный результат. Несмотря на то, что при масштабировании происходит потеря точности статистики, тесты показывают, что оно приводит к лучшим показателям сжатия, чем если бы масштабирование откладывалось. Это можно объяснить тем, что текущие символы сжимаемого потока больше «похожи» на своих близких предшественников, чем на тех, которые встречались намного раньше. Масштабирование приводит к уменьшению влияния «давних» символов на статистику и к увеличению

влияния на неё «недавних» символов. Это очень сложно измерить количественно, но, в принципе, масштабирование оказывает положительное влияние на степень сжатия информации. Эксперименты с масштабированием в различных точках процесса сжатия показывают, что степень сжатия сильно зависит от момента масштабирования веса, но не существует правила выбора оптимального момента масштабирования для программы, ориентированной на сжатие любых типов информации.

Сжатие данных по Хаффману применяется при сжатии фото- и видеоизображений (JPEG, стандарты сжатия MPEG), в архиваторах (PKZIP, LZH и др.), в протоколах передачи данных MNP5 и MNP7.

2.1.6. Алгоритм PPM

Алгоритм PPM (prediction by partial matching) - это метод контекстно-ограниченного моделирования, позволяющий оценить вероятность символа в зависимости от предыдущих символов. Строку символов, непосредственно предшествующую текущему символу, будем называть контекстом. Модели, в которых для оценки вероятности используются контексты длиной не более чем N, принято называть моделями порядка N.

Вероятность символа может быть оценена в контекстах разных порядков. Например, символ "o" в контексте "to be or not t" может быть оценен в контексте первого порядка «t», в контексте второго порядка «_t», в контексте третьего порядка «t_t» и так далее. Он также может быть оценен в контексте нулевого порядка, где вероятности символов не зависят от контекста, и в контексте минус первого порядка, где все символы равновероятны. Контекст минус первого порядка используется для того, чтобы исключить ситуацию, когда символ будет иметь нулевую вероятность и не сможет быть закодирован. Это может случиться, если вероятность символа не будет оценена ни в одном из контекстов (что возможно, если символ в них ранее не встречался).

Существуют два основных подхода к вычислению распределения вероятностей следующего символа на основе вероятностей символов в контекстах. Первый подход называется «полное перемешивание». Он предполагает назначение весов контекстам разных порядков и получение суммарных вероятностей сложением вероятностей символов в контекстах, умноженных на веса этих контекстов. Применение такого подхода ограничено двумя факторами. Во-первых, не существует быстрой реализации данного алгоритма. Во-вторых, не разработан эффективный алгоритм вычисления весов контекстов. Примитивные же подходы не обеспечивают достаточно высокой точности оценки и, как следствие, степени сжатия.

Второй подход называется «методом исключений». При этом подходе сначала делается попытка оценить символ в контексте самого высокого порядка. Если символ кодируется, алгоритм переходит к кодированию следующего символа. В противном случае кодируется «уход» и предпринимается попытка закодировать символ в контексте меньшего порядка. И так далее, пока символ не будет закодирован [12].

2.1.7. Алгоритм BWT

BWT-компрессор (Преобразование Барроуза – Уиллера) - сравнительно новая и революционная техника для сжатия информации (в особенности-текстов), основанная на преобразовании, открытом в 1983 г. и описанная в 1994 г.. BWT является удивительным алгоритмом. Во-первых, необычно само преобразование, открытое в научной области, далекой от архиваторов. Во-вторых, даже зная BWT, не совсем ясно, как его применить к сжатию информации. В-третьих, BW преобразование чрезвычайно просто. И, наконец, сам BWT компрессор состоит из "магической" последовательности нескольких рассмотренных ранее алгоритмов и требует, поэтому, для своей реализации самых разнообразных программных навыков.

BWT не сжимает данные, но преобразует блок данных в формат,

исключительно подходящий для компрессии. Рассмотрим его работу на упрощенном примере. Пусть имеется словарь V из N символов. Циклически переставляя символы в словаре влево, можно получить N различных строк длиной N каждая. В нашем примере словарь-это слово V="БАРАБАН" и N=7. Отсортируем эти строки лексикографически и запишем одну под другой:

F L

АБАНБАР

АНБАРАБ

АРАБАНБ

БАНБАРА

БАРАБАН

НБАРАБА

РАБАНБА

Далее нас будут интересовать только первый столбец F и последний столбец L. Оба они содержат все те же символы, что и исходная строка (словарь). Причем, в столбце F они отсортированы, а каждый символ из L является префиксом для соответствующего символа из F.

Фактический "выход" преобразования состоит из строки L="РББАНАА" и первичного индекса I, показывающего, какой символ из L является действительным первым символом словаря V (в нашем случае I=2). Зная L и I можно восстановить строку V [18].

2.1.8. Алгоритм арифметического кодирования

Арифметическое сжатие - достаточно изящный метод, в основе которого лежит очень простая идея. Мы представляем кодируемый текст в виде дроби, при этом строим дробь таким образом, чтобы наш текст был представлен как можно компактнее. Для примера рассмотрим построение такой дроби на интервале [0, 1) (0 - включается, 1 - нет). Интервал [0, 1) выбран потому, что он удобен для объяснений. Мы разбиваем его на

подынтервалы с длинами, равными вероятностям появления символов в потоке. В дальнейшем будем называть их диапазонами соответствующих символов.

Пусть мы сжимаем текст "КОВ.КОРОВА" (что, очевидно, означает "коварная корова"). Распишем вероятности появления каждого символа в тексте (в порядке убывания) и соответствующие этим символам диапазоны:

Таблица 3

Символ	Частота	Вероятность	Диапазон
О	3	0.3	[0.0; 0.3)
К	2	0.2	[0.3; 0.5)
В	2	0.2	[0.5; 0.7)
Р	1	0.1	[0.7; 0.8)
А	1	0.1	[0.8; 0.9)
“.”	1	0.1	[0.9; 1.0)

Будем считать, что эта таблица известна в компрессоре и декомпрессоре. Кодирование заключается в уменьшении рабочего интервала. Для первого символа в качестве рабочего интервала берется [0, 1). Мы разбиваем его на диапазоны в соответствии с заданными частотами символов (см. таблицу диапазонов). В качестве следующего рабочего интервала берется диапазон, соответствующий текущему кодируемому символу. Его длина пропорциональна вероятности появления этого символа в потоке. Далее считываем следующий символ. В качестве исходного берем рабочий интервал, полученный на предыдущем шаге, и опять разбиваем его в соответствии с таблицей диапазонов. Длина рабочего интервала уменьшается пропорционально вероятности текущего символа, а точка начала сдвигается вправо пропорционально началу диапазона для этого символа. Новый построенный диапазон берется в качестве рабочего и т. д. Используя исходную таблицу диапазонов, кодируем текст "КОВ.КОРОВА":

Исходный рабочий интервал [0,1).

Символ "К" [0.3; 0.5) получаем [0.3000; 0.5000).

Символ "О" [0.0; 0.3) получаем [0.3000; 0.3600).

Символ "В" [0.5; 0.7) получаем [0.3300; 0.3420).

Символ "." [0.9; 1.0) получаем [0,3408; 0.3420).

Таким образом, окончательная длина интервала равна произведению вероятностей всех встретившихся символов, а его начало зависит от порядка следования символов в потоке. Можно обозначить диапазон символа с как [a[c]; b[c]), а интервал для i-го кодируемого символа потока как $[l_i, h_i)$.

Большой вертикальной чертой на рисунке выше обозначено произвольное число, лежащее в полученном при работе интервале $[/_i, h_i)$. Для последовательности "КОВ.", состоящей из четырех символов, за такое число можно взять 0.341. Этого числа достаточно для восстановления исходной цепочки, если известна исходная таблица диапазонов и длина цепочки.

Рассмотрим работу алгоритма восстановления цепочки. Каждый следующий интервал вложен в предыдущий. Это означает, что если есть число 0.341, то первым символом в цепочке может быть только "К", поскольку только его диапазон включает это число. В качестве интервала берется диапазон "К" - [0.3; 0.5) и в нем находится диапазон [a[c]; b[c]), включающий 0.341. Перебором всех возможных символов по приведенной выше таблице находим, что только интервал [0.3; 0.36), соответствующий диапазону для "О", включает число 0.341. Этот интервал выбирается в качестве следующего рабочего и так далее [7].

2.2. Методы сжатия информации с потерями

Сжатие с потерями применяется в основном для графики (JPEG), звука (MP3), видео (MPEG), то есть там, где в силу огромных размеров файлов степень сжатия очень важна, и можно пожертвовать деталями, не существенными для восприятия этой информации человеком. Особые возможности для сжатия информации имеются при компрессии видео. В ряде случаев большая часть изображения передается из кадра в кадр без

изменений, что позволяет строить алгоритмы сжатия на основе выборочного отслеживания только части «картинки». В частном случае изображение говорящего человека, не меняющего своего положения, может обновляться только в области лица или даже только рта — то есть в той части, где происходят наиболее быстрые изменения от кадра к кадру.

В целом ряде случаев сжатие графики с потерями, обеспечивая очень высокие степени компрессии, практически незаметно для человека. Так, из трех фотографий, показанных ниже, первая представлена в TIFF-формате (формат без потерь), вторая сохранена в формате JPEG с минимальным параметром сжатия, а третья с максимальным. При этом можно видеть, что последнее изображение занимает почти на два порядка меньший объем, чем первая.Однако методы сжатия с потерями обладают и рядом недостатков.

Первый заключается в том, что компрессия с потерями применима не для всех случаев анализа графической информации. Например, если в результате сжатия изображения на лице изменится форма родинки (но лицо при этом останется полностью узнаваемо), то эта фотография окажется вполне приемлемой, чтобы послать ее по почте знакомым, однако если пересылается фотоснимок легких на медэкспертизу для анализа формы затемнения — это уже совсем другое дело. Кроме того, в случае машинных методов анализа графической информации результаты кодирования с потерей (незаметные для глаз) могут быть «заметны» для машинного анализатора.

Вторая причина заключается в том, что повторная компрессия и декомпрессия с потерями приводят к эффекту накопления погрешностей. Если говорить о степени применимости формата JPEG, то, очевидно, он полезен там, где важен большой коэффициент сжатия при сохранении исходной цветовой глубины. Именно это свойство обусловило широкое применение данного формата в представлении графической информации в Интернете, где скорость отображения файла (его размер) имеет первостепенное значение. Отрицательное свойство формата JPEG —

ухудшение качества изображения, что делает практически невозможным его применение в полиграфии, где этот параметр является определяющим.

2.2.1. Алгоритм JPEG

Алгоритм JPEG используется для сжатия статических изображений.

Сжатие JPEG осуществляется в несколько этапов: сперва цвета пикселей переводятся из RGB- представления в YCbCr-представление (в данной модели цвет представляется компонентами «яркость» Y, «цветоразность зеленый-красный» Cr и «цветоразность зеленый-синий» Cb). Затем в каждой второй строке и каждом втором столбце матрицы пикселей информация о цветовых компонентах Cb и Cr просто удаляется, что мгновенно уменьшает объем данных вдвое. Оставшиеся данные подвергаются специальной процедуре «сглаживания», при которой объем данных не изменяется, но потенциальная степень их сжимаемости резко увеличивается (на этом этапе учитывается коэффициент сжатия). Затем данные сжимаются алгоритмом Хаффмана.

Алгоритм JPEG способен упаковывать графические изображения в несколько десятков раз, при этом потери качества становятся заметными только при очень высоких коэффициентах сжатия.

2.2.2. Алгоритм MP3

Сжатие MP3 является частью стандарта MPEG и применяется для сжатия аудиоинформации. Помимо сжимаемой информации алгоритму передается желаемый битрейт – количество бит, используемых для кодирования одной секунды звука.Этот параметр регулирует долю информации, которая будет удаляться.

Сжатие MP3 также осуществляется в несколько этапов: звуковой фрагмент разбивается на небольшие участки — фреймы, а в каждом фрейме звук разлагается на составляющие звуковые колебания, которые в физике

называют гармониками. Затем начинается психоакустическая обработка — удаление маловажной для человеческого восприятия звуковой информации. Желаемый битрейт определяет, какие эффекты будут учитываться при сжатии, а также какое количество информации будет удалятся. На следующем этапе оставшиеся данные сжимаются алгоритмом Хаффмана.

Алгоритм MP3 позволяет сжимать звуковые файлы в несколько раз. Даже при самом «плохом» раскладе обеспечивается четырехкратное сжатие аудиоинформации [14].

2.2.3. Алгоритмы MPEG

MPEG — это целое семейство методов сжатия видеоданных. В них используется очень большое количество приемов сжатия.

Первый прием - использование «опорного кадра» — заключается в том, чтобы сохранять не целиком кадры, а только изменения кадров. Например, в фильме есть сцена беседы героев в комнате. При этом от кадра к кадру меняются только выражения лиц, а большая часть изображения неподвижна. Закодировав первый кадр сцены и отличия остальных ее кадров от первого, можно получить очень большую степень сжатия.

Следующий прием заключается в том, чтобы быстро сменяемые участки изображения кодировать с качеством, которое намного ниже качества статичных участков, — человеческий глаз не успевает рассмотреть их детально.

Кроме того, формат MPEG позволяет сохранять в одном файле несколько так называемых потоков данных. Так, в основном потоке можно сохранить фильм, в другом — логотип (храниться один раз, а не в каждом кадре), в третьем — субтитры (как текст), и т. д. Потоки данных накладываются друг на друга только при воспроизведении.

Разновидности формата MPEG отличаются друг от друга по возможностям, качеству воспроизводимого изображения и максимальной степени сжатия:

• MPEG-1 — использовался в первых Video CD (VCD-I);

• MPEG-2 — используется в DVD и Super Video CD (SVCD, VCD-II);

• MJPEG — формат сжатия видео, в котором каждый кадр сжимается по методу JPEG;

• MPEG-4 — усовершенствованный формат сжатия видео;

• DivX, XviD — улучшенные модификации формата MPEG-4 [15].

Заключение

В данной выпускной работе были рассмотрены основные методы сжатия информации. Изучены соответствующие источники и необходимая литература. Был проведён анализ нескольких изученных алгоритмов кодирования без потерь и с потерями.

Конечная цель в работе – изучение основных методов сжатия информации - была достигнута. Все алгоритмы сжатия можно условно разбить на два основных класса – это алгоритмы сжатия без потерь качества при восстановлении закодированной информации, и алгоритмы с некоторой потерей качества восстановления. В первом случае гарантируется точное соответствие между исходным и восстановленным и информацией, но при этом достигается, как правило, невысокая степень сжатия, обычно 2-3. При сжатии с некоторой потерей качества. При этом выбор в пользу того или иного алгоритма сжатия следует делать в зависимости от конкретной прикладной задачи.

В настоящий момент на рынке программных продуктов и серверах программного обеспечения можно встретить достаточно большое число сжимающих утилит, большинство из которых доступны для некоммерческого использования. Такая доступность, прежде всего, связана с тем, что каждая, даже коммерческая, программа нуждается в обширном рынке пользователей. А поскольку форматы файлов архиваторов и паковщиков, даже использующих один и тот же алгоритм, не одинаковы, то такие программы невольно конкурируют за рынок пользователей.

Индустрия и теория сжатия информации постоянно развиваются. Поэтому не за горами появление еще более мощных и удобных в использовании программ, и алгоритмов.

Список использованных источников

1. Д. Ватолин, А. Ратушняк, М. Смирнов, В. Юкин. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео. — Диалог-МИФИ, 2002.
2. Д. Сэломон. Сжатие данных, изображения и звука. — М.: Техносфера, 2004.
3. Ватолин Д.С. Алгоритмы сжатия изображений – М.: Диалог-МГУ, 1999.
4. Климов А.С. Форматы графически файлов С.-Пб.: ДиаСофт. 1995
5. Александров В.В., Горский Н.Д. Представление и обработка изображений/ М.: Наука, 2002.
6. Кривошеев М.И. Основы телевизионных измерений. 3- изд., доп. и перераб. М.: Радио и связь, 1989.
7. Кадач А.В. Эффективные алгоритмы неискажающего сжатия текстовой информации – Институт систем информатики им. А.П.Ершова, Новосибирск, 1997.
8. Ковалгин Ю.А., Вологдин Э.И. Цифровое кодирование звуковых сигналов — М.: Корона-Принт, 2004 г.
9. Артюшенко В.М., Шелухин О.И., Афонин М.Ю. Цифровое сжатие видеоинформации и звука — М.: Дашков и Ко, 2004 г.
10. Ричардсон Я. Видеокодирование. H.264 и MPEG-4 - стандарты нового поколения. Техносфера, 2005 г.
11. Дж. Миано. Форматы и алгоритмы сжатия изображений в действии. Из-во Триумф, 2003.
12. . Ахметов К. С. Курс молодого бойца.- М.: Компьютер- пресс, 1995.
13. Глушаков С. В., Мельников И. В. Персональный компьютер. –М.: АСТ,2001.
14. Дьяконов В. П. MathCAD 2000: Учебный курс.- СПб.: Питер, 2000.
15. Илюшечкин В. М., Костин А. Е. Системное программное обеспечение.- М.: Высшая школа, 1991.
16. Карпов Ю. . Теория автоматов: Учеб. Пособие.- СПб.: «Питер», 2002.

17. Ковтанюк Ю.С., Соловьян С.В. Самоучитель работы на персональном компьютере - К.:Юниор, 2001.- 560с.
18. Леонтьев В.П. Учимся работать с Windows XP.- М.: Олма- Пресс, 2004.
19. Маейрс Г. Надежность программного обеспечения.- М.: Мир, 1980.
20. Меженный О. А. Windows XP. Самоучитель.- М.: Диалектика, 2002.