

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО**

**ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«Донской государственный технический университет»**

**(ДГТУ)**

**Доклад на тему**

**«Современные алгоритмы сжатия»**

(теория информации и кодирования)

Выполнил: студент ВМО21  
Оганесьянц К.П.

Проверил: Ляхницкая О.В.

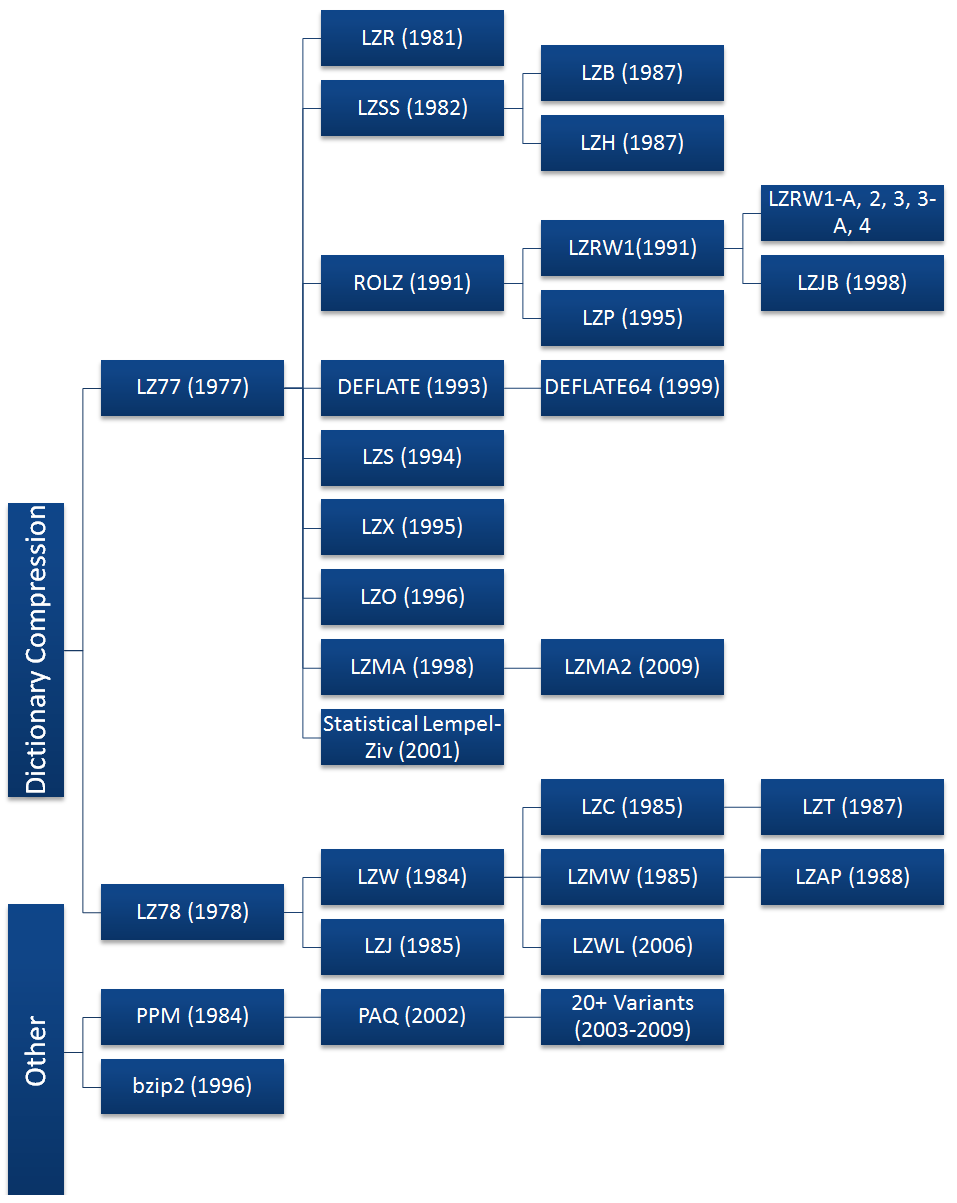
Ростов-на-Дону

2021

Существующие алгоритмы сжатия данных можно разделить на два больших класса – с потерями, и без.

Алгоритмы с потерями обычно применяются для сжатия изображений и аудио. Эти алгоритмы позволяют достичь больших степеней сжатия благодаря избирательной потере качества. Но восстановить первоначальные данные из полученного сжатого формата невозможно.

Алгоритмы сжатия без потерь применяются для уменьшения размера данных, и работают таким образом, что возможно восстановить данные в точности такими, какие они были до сжатия. Они применяются в коммуникациях, архиваторах и некоторых алгоритмах сжатии аудио и графической информации.

Основной принцип алгоритмов сжатия базируется на том, что в любом файле, содержащем неслучайные данные, информация частично повторяется. Используя статистические математические модели можно определить вероятность повторения определённой комбинации символов. После этого можно создать коды, обозначающие выбранные фразы, и назначить самым часто повторяющимся фразам самые короткие коды. Для этого используются разные техники, например: энтропийное кодирование, кодирование повторов, и сжатие при помощи словаря. С их помощью 8-битный символ, или целая строка, могут быть заменены всего лишь несколькими битами, устраняя таким образом излишнюю информацию.  
  
Хотя сжатие данных получило широкое распространение вместе с интернетом и после изобретения алгоритмов Лемпелем и Зивом (алгоритмы LZ), можно привести несколько более ранних примеров сжатия. Морзе, изобретая свой код в 1838 году, разумно назначил самым часто используемым буквам в английском языке, “e” и “t”, самые короткие последовательности (точка и тире соотв.). Вскоре после появления мейнфреймов в 1949 году был придуман алгоритм Шеннона — Фано, который назначал символам в блоке данных коды, основываясь на вероятности их появления в блоке. Вероятность появления символа в блоке была обратно пропорциональна длине кода, что позволяло сжать представление данных.

Дэвид Хаффман был студентом в классе у Роберта Фано и в качестве учебной работы выбрал поиск улучшенного метода бинарного кодирования данных. В результате ему удалось улучшить алгоритм Шеннона-Фано.  
Ранние версии алгоритмов Шеннона-Фано и Хаффмана использовали заранее определённые коды. Позже для этого стали использовать коды, созданные динамически на основе данных, предназначаемых для сжатия. В 1977 году Лемпель и Зив опубликовали свой алгоритм LZ77, основанный на использования динамически создаваемого словаря (его ещё называют «скользящим окном»). В 78 году они опубликовали алгоритм LZ78, который сначала парсит данные и создаёт словарь, вместо того, чтобы создавать его динамически.

Алгоритмы LZ77 и LZ78 получили большую популярность и вызвали волну адаптаций самих алгоритмов , из которых сейчас существуют и применяются только DEFLATE, LZMA и LZX. Большинство популярных алгоритмов основаны на LZ77, потому что производный от LZ78 алгоритм LZW был запатентован компанией Unisys в 1984 году. В это время на UNIX использовали вариацию алгоритма LZW под названием LZC, и из-за проблем с правами их использование пришлось сворачивать. Предпочтение отдали алгоритму DEFLATE (gzip) и преобразованию Барроуза — Уилера, BWT (bzip2). Что с точки зрения оптимизации было правильным решением так как выбранные алгоритмы превосходят LZW   
К 2003 году срок патента истёк, но никто не решил использовать алгоритм LZW, и он сохранился только в файлах с раширением GIF сохранился,. Доминирующими являются алгоритмы на основе LZ77.  
Большие корпорации использовали алгоритмы сжатия для хранения всё увеличивавшихся массивов данных, но истинное распространение алгоритмов произошло с рождением интернета в конце 80-х. Пропускная способность каналов была чрезвычайно узкой. Для сжатия данных, передаваемых по сети, были придуманы форматы ZIP, GIF и PNG.  
Том Хендерсон придумал и выпустил первый коммерчески успешный архиватор ARC в 1985 году (компания System Enhancement Associates). ARC была популярной среди пользователей BBS, т.к. она одна из первых могла сжимать несколько файлов в архив, к тому же исходники её были открыты. ARC использовала модифицированный алгоритм LZW.

Фил Катц, вдохновлённый популярностью ARC, выпустил программу PKARC в формате shareware, в которой улучшил алгоритмы сжатия, переписав их на Ассемблере. Однако, был засужен Хендерсоном и был признан виновным. PKARC настолько открыто копировала ARC, что иногда даже повторялись опечатки в комментариях к исходному коду.  
Но Фил Катц не растерялся, и в 1989 году сильно изменил архиватор и выпустил PKZIP. После того, как его атаковали уже в связи с патентом на алгоритм LZW, он изменил и базовый алгоритм на новый, под названием IMPLODE. Вновь формат был заменён в 1993 году с выходом PKZIP 2.0, и заменой стал DEFLATE. Среди новых возможностей была функция разбиения архива на тома. Эта версия до сих пор повсеместно используется, несмотря на почтенный возраст.

Формат изображений GIF (Graphics Interchange Format) был создан компанией CompuServe в 1987. Как известно, формат поддерживает сжатие изображения без потерь, и ограничен палитрой в 256 цветов. Несмотря на все потуги Unisys, ей не удалось остановить распространение этого формата. Он до сих пор популярен, особенно в связи с поддержкой анимации.  
Компания CompuServe в 1994 году выпустила формат Portable Network Graphics (PNG). Как и ZIP, она использовала новый модный алгоритм DEFLATE. Хотя DEFLATE был запатентован Катцем, он не стал предъявлять никаких претензий.  
Сейчас это самый популярный алгоритм сжатия. Кроме PNG и ZIP он используется в gzip, HTTP, SSL и других технологиях передачи данных.

**Современные архиваторы**

ZIP использовался повсеместно до середины 90-х, однако в 1993 году Евгений Рошал придумал свой формат и алгоритм RAR. Последние его версии основаны на алгоритмах PPM и LZSS. Сейчас ZIP - самый распространённый из форматов, RAR – до недавнего времени был стандартом для распространения различного малолегального контента через интернет (благодаря увеличению пропускной способности всё чаще файлы распространяются без архивации), а 7zip используется как формат с наилучшим сжатием при приемлемом времени работы. В мире UNIX используется связка tar + gzip (gzip — архиватор, а tar объединяет несколько файлов в один). В 1996 году появился вариант алгоритма BWT с открытыми исходниками bzip2, и быстро приобрёл популярность. В 1999 году появилась программа 7-zip с форматом 7z. По сжатию она соперничает с RAR, её преимуществом является открытость, а также возможность выбора между алгоритмами bzip2, LZMA, LZMA2 и PPMd.

В 2002 году появился ещё один архиватор, PAQ. Автор Мэтт Махоуни использовал улучшенную версию алгоритма PPM с использованием техники под названием «контекстное смешивание». Она позволяет использовать больше одной статистической модели, чтобы улучшить предсказание по частоте появления символов.

**LZMA2**

LZMA2 является новой версией LZMA. Отличие между данными алгорит­мами в том, что в первом реализована многопоточность и хранение одновременно сжа­тый и несжатых данных, что, в свою очередь, экономит биты. Форматом файлов, предназначенным для размещения сжатых данн^1х, который по умолчанию использует алгоритм LZMA2, является формат.„z. В настоящее время этот формат дает возможность выбора между LZMA и LZMA2.Алгоритм LZMA включает в себя:1. Дельта-кодирование.2. Алгоритм LZ77.3. Интервальное кодирование. Дельта-кодирование представляет данные как разницу между последовательными символами.

LZ77 рассматривает последовательности символов, и если какая-то последователь­ность встречается более одного раза, то он заменяет её повторные вхождения ссылкой на её первый экземпляр. Как и во всех алгоритмах семейства LZ, в LZ77 использу­ется словарь, хранящий последовательности, которые встречались ранее, т.е. он ис­пользует принцип «скользящего окна. Сам словарь хранит данные о смещении, длине серий и символ расхождения. Смещение - это расстояние фразы от начала файла. Длина серии - это количество сим - волов, которое принадлежит фразе, если считать от смещения. Символ расхождения показывает на то, что на выход пришла фраза, которая похожа на ту, которая обозначена на смещение и длиной. При интервальном кодировании выделяется некоторый диапазон символов, и оце­нивается вероятность вхождения каждого. Таким образом, этот алгоритм преобразовывает все символы сообщения в одно число. Название алгоритма происходит от «Lempel -Ziv Markov chain Algorithm».

**PAQ**

PAQ - серия свободных архиваторов с текстовым интерфейсом, которые общими усилиями разработчиков поднялись в первые места рейтингов многих тестов сжатия данных. Лучший результат в этой серии на большинстве тестов был получен архиватором PAQ8JD, созданным совместными усилиями Мэтта Махони, Александра Ратушняка, Сергея Оснача, Пшемыслава Скибиньского и Билла Петтиса, и выпущенным 30 декабря 2006 года. Однако в некоторых тестах он отстаёт от WinRK в режиме PWCM. PWCM - сторонняя проприетарная реализация алгоритма PAQ. Специально настроенные версии алгоритма PAQ выиграли призы в Приз Хаттера и Калгари Корпус Челлендж.

**1. Алгоритм**

В основе алгоритма лежит идея контекстного моделирования. Контекст - это, говоря доступным языком, история появления символа, то есть информация о символах, предшествующих текущему в сжимаемом потоке. При этом процесс компрессии разбивается на две фазы: моделирование и кодирование. PAQ использует алгоритм смешивания контекстов. Смешивание контекстов - это техника, тесно связанная с алгоритмом PPM, но отличие состоит в том, что вероятность появления следующего символа вычисляется на основе взвешенной комбинации большого числа моделей, зависящих от разных контекстов, не обязательно следующих друг за другом. В PAQ семействе для сбора статистики и предсказания вероятности следующего символа используются в основном следующие модели:

словарные n-граммы, игнорирующие регистр и неалфавитные символы полезны в текстовых данных.

двумерные контексты полезны для изображений, табличных данных. Длина ряда определяется нахождением повторяющихся паттернов байт.

"разрежённые" контексты, например, второй и четвёртый символы перед кодируемым полезны в некоторых бинарных форматах.

специализированные модели, такие, как x86-исполняемые файлы или Windows Bitmap, TIFF, JPEG-изображения. Эти модели активируются, когда данный тип файла определяется.

"аналоговые" контексты, состоящие из верхней половины двоичного представления 8- или 16-битных слов полезны в мультимедийных форматах данных.

n-граммы - контекст; предыдущие n байт как в PPM.

Все версии PAQ предсказывают и сжимают за раз один бит, но различаются в деталях реализации того, как предсказания комбинируются и обрабатываются после. Как только предсказательно была определена вероятность появления следующего бита, бит сжимается арифметическим кодером. Существует три способа для комбинирования предсказаний моделей в зависимости от версии PAQ.

от PAQ1 до PAQ3 каждое предсказание представлено парой битовых счётчиков n 0, n 1. Эти счётчики комбинировались взвешенным суммированием, таким, что больший вес определяется более длинным контекстом.

в PAQ7 и более поздних версиях выход каждой модели есть вероятность) обрабатывают текст, просматривая его и заменяя слова из текста, содержащиеся во внешнем словаре, одно- и трёхбайтными кодами. Дополнительно слова в верхнем регистре кодируются специальным символом и переводом слова в нижний регистр. В PAQ8HP-серии словарь организован группировкой синтаксически и семантически похожих слов вместе. Это позволяет использовать модели, которые используют только верхние биты словарных кодов в качестве контекстов.

в PAQ4 до PAQ6 предсказания комбинировались, как и в первом случае, но веса, принадлежащие каждой модели, назначались так, чтобы более точные модели получали преимущество.

PAQ - это свободное программное обеспечение и распространяется на условиях GNU General Public License. Это позволяет другим авторам сделать форк PAQ и вносить такие изменения, как Графический интерфейс пользователя, либо улучшить скорость сжатия за счёт коэффициента компрессии. Наиболее известные PAQ-клоны:

WinUDA 0.291, базируется на PAQ6, но быстрее

**ZIP алгоритм сжатия.**

Формат ZIP работает на Deflate сжатии. Deflate — это алгоритм сжатия без потерь, использующий комбинацию алгоритмов LZ77 и Хаффмана.

***Сжатие Lempel-Ziv (LZ77)***

Один из способов сжатия текста заключается в создании списка частых слов или фраз с заменой разновидностей этих слов в рамках текста ссылками на словарь. Например, длинное слово «compression» в исходном тексте можно представить как #1234, где 1234 ссылается на позицию слова в списке. Это называется сжатием с использованием словаря.  
  
*Принцип скользящего окна*

Метод кодирования, согласно принципу скользящего окна, учитывает уже ранее встречавшуюся информацию, то есть информацию, которая уже известна для кодировщика и декодировщика (второе и последующие вхождения некоторой строки символов в сообщении заменяются ссылками на её первое вхождение).

Благодаря этому принципу алгоритмы LZ\* иногда называются методами сжатия с использованием скользящего окна. Скользящее окно можно представить в виде буфера (или более сложной динамической структуры данных), который организован так, чтобы запоминать «сказанную» ранее информацию и предоставлять к ней доступ. Таким образом, сам процесс сжимающего кодирования согласно LZ77 напоминает написание программы, команды которой позволяют обращаться к элементам «скользящего окна», и вместо значений сжимаемой последовательности вставлять ссылки на эти значения в «скользящем окне». Размер скользящего окна может динамически изменяться и составлять 2, 4 или 32 килобайта. Следует также отметить, что размер окна кодировщика может быть меньше или равен размеру окна декодировщика, но не наоборот.

Приведенное выше сравнение процесса кодирования с «программированием» может натолкнуть на преждевременный вывод о том, что алгоритм LZ77 относится к методам контекстного моделирования. Поэтому следует отметить, что алгоритм LZ77 принято классифицировать как метод словарного сжатия данных, когда вместо понятия «скользящего окна» используется термин «динамического словаря».

Механизм кодирования совпадений

Перед тем, как перейти к рассмотрению механизма кодирования, уточним понятие совпадения (от англ. match). Рассмотрим последовательность из N элементов. Если все элементы последовательности уникальны, то такая последовательность не будет содержать ни одного повторяющегося элемента, или, иначе говоря, в последовательности не найдется хотя бы двух равных друг другу или совпадающих элементов.

В стандартном алгоритме LZ77 совпадения кодируются парой:

длина совпадения (match length)

смещение (offset) или дистанция (distance)

В продолжение уже приведенной аналогии с программированием отметим, что в большинстве статей, посвященных алгоритму LZ77, кодируемая пара трактуется именно как команда копирования символов из скользящего окна с определенной позиции, или дословно как: «Вернуться в буфере символов на значение смещения и скопировать значение длины символов, начиная с текущей позиции».

Хотя для приверженцев императивного программирования такая интерпретация может показаться интуитивно понятной, она мало говорит о сущности алгоритма LZ77 как метода сжатия. Особенность данного алгоритма сжатия заключается в том, что использование кодируемой пары длина-смещение является не только приемлемым, но и эффективным в тех случаях, когда значение длины превышает значение смещения.

Пример с командой копирования не совсем очевиден: «Вернуться на 1 символ назад в буфере и скопировать 7 символов, начиная с текущей позиции». Каким образом можно скопировать 7 символов из буфера, когда в настоящий момент в буфере находится только 1 символ? Однако следующая интерпретация кодирующей пары может прояснить ситуацию: каждые 7 последующих символов совпадают (эквивалентны) с 1 символом перед ними.

Это означает, что каждый символ можно однозначно определить, переместившись назад в буфере — даже если данный символ ещё отсутствует в буфере на момент декодирования текущей пары длина-смещение. Такая кодируемая пара будет представлять собой многократное (определяемое значением смещения) повторение последовательности (определяемой значением длины) символов, что представляет собой более общую форму RLE.

***Кодирование Хаффмана***

Один из первых алгоритмов эффективного кодирования информации был предложен Д. А. Хаффманом в 1952 году. Идея алгоритма состоит в следующем: зная вероятности появления 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 (то есть ни одно кодовое слово не является префиксом другого), 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.

Выбираются два свободных узла дерева с наименьшими весами.

Создается их родитель с весом, равным их суммарному весу.

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

Одной дуге, выходящей из родителя, ставится в соответствие бит 1, другой — бит 0. Битовые значения ветвей, исходящих от корня, не зависят от весов потомков.

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.

Допустим, у нас есть следующая таблица частот:

Символ А Б В Г Д

Частота 15 7 6 6 5

Этот процесс можно представить, как построение дерева, корень которого — символ с суммой вероятностей объединенных символов, получившийся при объединении символов из последнего шага, его n0 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 текущему символу, до его корня, накапливая биты при перемещении по ветвям дерева (первая ветвь в пути соответствует младшему биту). Полученная таким образом последовательность битов является кодом данного символа, записанным в обратном порядке.

Построение дерева для данного примера

Для данной таблицы символов коды Хаффмана будут выглядеть следующим образом.

Символ А Б В Г Д

Код 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, то есть 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.

Сам Deflate алгоритм работает следующим образом:

1. Сначала идет замена повторяющихся строк на указатели с помощью алгоритма LZ77
2. Вторым пунктом идет замена символов на новые, на основании статистических данных по частоте их использования с помощью алгоритма Хаффмана