**АНПОО «ВОРОНЕЖСКИЙ ИНСТИТУТ ВЫСОКИХ ТЕХНОЛОГИЙ»**

**Реферат**

**по дисциплине Информационные технологии**

**на тему «Кодирование информации»**

Выполнил студент

Андросов Антон Сергеевич

ИС-241

Воронеж 2024

### Оглавление

[Введение 2](#_Toc2017557567)

[Кодирование Хаффмана 7](#_Toc1310248393)

[Подход Шенонна-Фано 10](#_Toc1169438294)

[Кодирование Хэмминга 12](#_Toc1915007364)

[Список литературы 16](#_Toc1019932639)

### Введение

Сжатие данных — это процесс преобразования входного потока данных (исходного потока или оригинальных сырых данных) в другой поток данных (выходной, битовый поток или сжатый поток), который имеет меньший размер. Поток может быть либо файлом, либо буфером в памяти. Сжатие данных популярно по двум причинам: (1) Люди любят накапливать данные и ненавидят выбрасывать что-либо. Независимо от того, насколько велико устройство хранения, рано или поздно оно переполнится. Сжатие данных кажется полезным, потому что оно откладывает эту неизбежность. (2) Люди ненавидят долго ждать передачи данных. Когда мы сидим за компьютером, ожидая загрузки веб-страницы или файла, мы естественно чувствуем, что всё, что длится дольше нескольких секунд, — это долгое ожидание.

Область сжатия данных часто называется кодированием источника. Мы представляем, что входные символы (такие как биты, коды ASCII, байты, аудиосэмплы или значения пикселей) исходят от определенного информационного источника и должны быть закодированы перед отправкой к месту назначения. Источник может быть безпамятным или иметь память. В первом случае каждый символ независим от своих предшественников. Во втором случае каждый символ зависит от некоторых своих предшественников и, возможно, также от своих последователей, поэтому они коррелированы. Безпамятный источник также называется «независимо и идентично распределённым» или IIID.

Сжатие данных достигло своего расцвета за последние 20 лет. Как количество, так и качество литературы в этой области являются достаточным доказательством этого. Однако необходимость сжатия данных ощущалась и в прошлом, даже до появления компьютеров.

Существует множество известных методов сжатия данных. Они основаны на различных идеях, подходят для разных типов данных и дают разные результаты, но все они основаны на одном и том же принципе, а именно — они сжимают данные, удаляя избыточность из оригинальных данных в исходном файле. Любые нерегулярные данные имеют некоторую структуру, и эту структуру можно использовать для достижения более компактного представления данных, представления, в котором никакая структура не различима. В профессиональной литературе используются термины избыточность и структура, а также гладкость, согласованность и корреляция; все они относятся к одному и тому же понятию. Таким образом, избыточность является ключевым понятием в любом обсуждении сжатия данных.

Сжатие данных осуществляется путем изменения их представления с неэффективного (т.е. длинного) на эффективное (короткое). Таким образом, сжатие возможно только потому, что данные обычно представлены в компьютере в формате, который длиннее, чем абсолютно необходимо. Причина, по которой неэффективные (длинные) представления данных используются постоянно, заключается в том, что они упрощают обработку данных, а обработка данных более распространена и важна, чем сжатие данных. Код ASCII для символов является хорошим примером представления данных, которое длиннее, чем абсолютно необходимо. Он использует 7-битные коды, потому что коды фиксированного размера легко обрабатывать. Однако код переменного размера был бы более эффективным, поскольку определенные символы используются чаще других и, следовательно, могут быть назначены более короткие коды.

В мире, где данные всегда представлены в их кратчайшем возможном формате, не было бы возможности сжимать данные. Вместо написания книг о сжатии данных, авторы в таком мире писали бы книги о том, как определить кратчайший формат для различных типов данных.

Сжатие данных стало настолько важным, что некоторые исследователи предложили теорию SP (от "простота" и "мощь"), которая предполагает, что все вычисления — это сжатие! В частности, она утверждает: Сжатие данных может быть интерпретировано как процесс удаления ненужной сложности (избыточности) в информации, тем самым максимизируя простоту, сохраняя при этом как можно больше ее не избыточной описательной силы. Теория SP основана на следующих предположениях:

* Все виды вычислений и формального рассуждения могут быть полезно поняты как сжатие информации с помощью сопоставления шаблонов, унификации и поиска.
* Процесс нахождения избыточности и ее удаления всегда может быть понят на фундаментальном уровне как процесс поиска совпадающих шаблонов и объединения или унификации повторяющихся экземпляров любого шаблона в один.

Многие люди используют сжатие данных в областях науки и техники. Например, в области теории информации, сжатие данных связано с изучением того, как эффективно передавать и хранить информацию. В области компьютерных наук сжатие данных играет ключевую роль в алгоритмах и структурах данных, а также в разработке программного обеспечения.

Сжатие данных также находит применение в таких областях, как:

* Обработка изображений и видео: Сжатие изображений и видео позволяет уменьшить объем данных, что делает их более удобными для хранения и передачи. Это особенно важно для потокового видео и онлайн-стриминга.
* Аудио технологии: Сжатие аудиофайлов, таких как MP3 и AAC, позволяет уменьшить размер файлов без значительной потери качества звука, что делает их более удобными для хранения и передачи.
* Хранение данных: В системах хранения данных, таких как облачные хранилища, сжатие данных помогает экономить место и ускорять доступ к информации.
* Сетевые технологии: В сетевых протоколах сжатие данных может значительно уменьшить объем передаваемой информации, что улучшает скорость передачи и снижает затраты на трафик.
* Мобильные приложения: В условиях ограниченной пропускной способности мобильных сетей сжатие данных позволяет пользователям быстрее загружать контент и экономить трафик.

Таким образом, сжатие данных является важным аспектом современных технологий, который влияет на множество областей и способствует развитию новых решений и приложений. Исследования в этой области продолжают развиваться, открывая новые возможности для оптимизации хранения и передачи информации.

Практические проблемы стали стимулом для различных исследований в области сжатия данных. В то же время исследования в области сжатия данных также основывались на других новых предметных областях или стимулировали их развитие. Отчасти из-за своей широкой области применения сжатие данных пересекается с многими научными дисциплинами и может быть найдено в различных областях знаний. Например, вы увидите главы или разделы, посвященные сжатию данных, в книгах по:

* теории информации
* теории кодирования
* компьютерным сетям и телекоммуникациям
* цифровой обработке сигналов
* обработке изображений
* мультимедиа
* стеганографии
* компьютерной безопасности.

Язык, используемый в несвязанных дисциплинах, может существенно различаться. В этой книге слово "данные" в общем используется для обозначения информации в цифровой форме, с которой работают компьютерные программы, а "сжатие" означает процесс удаления избыточности в данных. Под "сжатием данных" мы на самом деле имеем в виду разработку техник или, более конкретно, проектирование эффективных алгоритмов для:

* представления данных в менее избыточной форме
* удаления избыточности в данных
* реализации алгоритмов сжатия, включая как сжатие, так и декомпрессию.

Интересы и цели также имеют тенденцию быть разнообразными среди людей с различными дисциплинарными фонами. Эта книга сосредоточена на алгоритмических аспектах сжатия данных. Мы рассматриваем сжатие данных как процесс разработки алгоритмических решений для задачи сжатия. Алгоритмическая задача — это общий вопрос, на который необходимо ответить с помощью упорядоченной последовательности инструкций. Эта последовательность инструкций рассматривается как последовательный алгоритм для задачи, а также как решение этой задачи. Алгоритм позволяет получить решение для любого экземпляра задачи путем выполнения алгоритма.

Задача сжатия данных заключается в нахождении эффективного алгоритма для удаления различных избыточностей из определенного типа данных. Общий вопрос, который следует задать в этом контексте, может быть, например, таким: дана некая строка С, какая альтернативная последовательность символов займёт меньше пространства накопительного устройства? Решения задач сжатия будут представлять собой алгоритмы сжатия, которые выведут альтернативную последовательность символов, содержащую меньшее количество бит в целом, а также алгоритмы декомпрессии для восстановления оригинальной строки.

Сколько бит будет сэкономлено? Это будет зависеть от алгоритмов, но также и от того, насколько избыточность может быть извлечена из оригинальных данных. Разные данные могут требовать различных техник для выявления и удаления избыточности. Очевидно, что это делает задачи сжатия "сложными" для решения, поскольку общий вопрос трудно легко ответить, когда он содержит слишком много экземпляров. К счастью, мы можем учитывать определенные ограничения и эвристики при проектировании алгоритмов.

Не существует универсального решения для задач сжатия данных. В исследованиях сжатия данных нам необходимо анализировать характеристики данных, которые нужно сжать, и надеяться выявить некоторые закономерности, чтобы достичь компактного представления. Это приводит к разнообразию техник моделирования и представления данных, которые лежат в основе методов сжатия.

Сжатие данных можно рассматривать как средство эффективного представления цифрового источника данных, такого как текст, изображение, звук или любое их сочетание, например, видео. Цель сжатия данных заключается в том, чтобы представить источник в цифровой форме с использованием как можно меньшего количества битов, при этом соблюдая минимальные требования для восстановления оригинала.

За каждым алгоритмом стоят идеи, математические модели или техники реализации, направленные на достижение сжатия.

При работе над задачами сжатия необходимо учитывать как аспект эффективности алгоритмов, так и эффективность самого сжатия. Интуитивно, поведение алгоритма сжатия будет зависеть от данных и их внутренней структуры. Чем больше избыточности содержится в исходных данных, тем более эффективным может быть алгоритм сжатия.

Любой алгоритм сжатия не будет работать, если не предусмотрено средство декомпрессии, что связано с природой сжатия данных. Когда обсуждаются алгоритмы сжатия в общем, слово "сжатие" на самом деле подразумевает контекст как сжатия, так и декомпрессии.

В этой книге мы иногда даже не обсуждаем алгоритмы декомпрессии, когда процесс декомпрессии очевиден или может быть легко выведен из процесса сжатия. Однако, как читатель, вы всегда должны быть уверены, что знаете решения для декомпрессии, так же как и для сжатия.

В многих практических случаях эффективность алгоритма декомпрессии вызывает большее беспокойство, чем эффективность алгоритма сжатия. Например, фильмы, фотографии и аудиоданные часто сжимаются один раз художником, а затем одна и та же версия сжатых файлов декомпрессируется множество раз миллионами зрителей или слушателей.

С другой стороны, эффективность алгоритма сжатия иногда оказывается более важной. Например, запись аудио или видео данных из некоторых программ в реальном времени может потребовать записи непосредственно на ограниченное компьютерное хранилище или передачи на удаленное место через узкий сигнальный канал.

В зависимости от конкретных задач мы иногда рассматриваем сжатие и декомпрессию как два отдельных синхронных или асинхронных процесса.

### Кодирование Хаффмана

Дэвид Альберт Хаффман (9 августа 1925 года – 7 октября 1999 года) был пионером в области компьютерных наук, известным благодаря своему кодированию Хаффмана.

В 1951 году Дэвид А. Хаффман и его одноклассники на курсе аспирантуры по электротехнике по теории информации получили выбор между курсовой работой и финальным экзаменом. Для курсовой работы профессор Хаффмана, Роберт М. Фано, задал то, что сначала казалось простой задачей. Студенты должны были найти наиболее эффективный способ представления чисел, букв или других символов с использованием двоичного кода. Кроме того, что это было ловким интеллектуальным упражнением, нахождение такого кода позволило бы сжимать информацию для передачи по компьютерной сети или для хранения в памяти компьютера.

Хаффман работал над этой задачей в течение нескольких месяцев, разрабатывая ряд подходов, но ни один из них не мог доказать свою эффективность. Наконец, он отчаялся когда-либо найти решение и решил начать подготовку к финальному экзамену. Как раз в тот момент, когда он собирался выбросить свои заметки в мусор, решение пришло к нему. «Это был самый единственный момент в моей жизни», — говорит Хаффман. «Это было абсолютное озарение внезапного осознания».

Это озарение добавило Хаффмана к легиону в значительной степени анонимных инженеров, чье инновационное мышление составляет техническую основу для атрибутов современной жизни — в его случае, от факсимильных машин до модемов и множества других устройств. «Код Хаффмана — это одна из основных идей, которые люди в области компьютерных наук и передачи данных используют постоянно», — говорит Дональд Э. Кнут из Стэнфордского университета, автор многотомной серии «Искусство программирования» (Addison-Wesley, 1997).

Как и многие другие ранние прорывы, Хаффман, возможно, никогда бы не нашел это решение без помощи своего профессора Фано, который отметил, что Клод Шеннон также сталкивался с той же проблемой. «Мне повезло быть там в нужное время и также не иметь своего профессора, который бы отговаривал меня, говоря, что другие хорошие люди боролись с этой проблемой», — говорит Хаффман.

Широко используемым методом сжатия данных является кодирование Хаффмана. Этот метод служит основой для нескольких популярных программ, используемых на персональных компьютерах. Некоторые из них используют только метод Хаффмана, в то время как другие применяют его как один из этапов в многоступенчатом процессе сжатия.

Метод Хаффмана [Хаффман, 1952] несколько похож на метод Шеннона-Фано. Он, как правило, производит лучшие коды, и, как и метод Шеннона-Фано, он дает наилучшие коды, когда вероятности символов являются отрицательными степенями двойки. Основное различие между двумя методами заключается в том, что Шеннон-Фано строит свои коды сверху вниз (от самых левых до самых правых битов), в то время как Хаффман строит дерево кодов снизу вверх (создает коды справа налево). С момента его разработки в 1952 году Д. Хаффманом этот метод стал предметом интенсивных исследований в области сжатия данных..

Кодирование Хаффмана — это успешный метод сжатия, изначально использовавшийся для сжатия текста. В любом тексте некоторые символы встречаются гораздо чаще, чем другие. Например, в английском тексте буквы E, A, O, T обычно используются гораздо чаще, чем J, Q, X.

Идея Хаффмана заключается в том, чтобы вместо использования кода фиксированной длины, такого как 8-битный расширенный ASCII или DBCDIC для каждого символа, представлять часто встречающийся символ в источнике с помощью более короткого кода, а менее часто встречающийся — с помощью более длинного кода. Таким образом, общее количество битов в этом представлении значительно уменьшается для источника символов с различными частотами. Среднее количество битов, необходимых для каждого символа, уменьшается.

Метод начинается с построения списка всех символов алфавита в порядке убывания их вероятностей. Затем он строит дерево, где каждый символ находится на листьях, начиная снизу. Это делается поэтапно, где на каждом этапе выбираются два символа с наименьшими вероятностями, добавляются к вершине частичного дерева, удаляются из списка и заменяются вспомогательным символом, представляющим оба. Когда список сокращается до одного вспомогательного символа (представляющего весь алфавит), дерево считается завершенным. Затем дерево обходится для определения кодов символов

Сначала мы изложим основные идеи алгоритма сжатия Хаффмана с пропущенными деталями:

1. Постройте бинарное дерево, где листья дерева — это символы в алфавите.
2. Ребра дерева помечаются 0 или 1.
3. Получите код Хаффмана из дерева Хаффмана.

Этот алгоритм легко понять. На самом деле процесс маркировки 0 и 1 не обязательно выполнять в конце построения всего дерева Хаффмана. Присвоение 0 или 1 может быть выполнено сразу после объединения двух элементов, начиная с наименее значимого бита каждого кодового слова.

Таким образом, как только два символа или поддерева объединяются, можно сразу же назначить 0 одному из них и 1 другому, что позволяет постепенно строить коды для каждого символа по мере формирования дерева. Это делает процесс более эффективным и интуитивно понятным, так как коды могут быть получены на лету, не дожидаясь завершения всего дерева.

### Подход Шенонна-Фано

Кодирование Шеннона-Фано, названное в честь Клода Шеннона и Роберта Фано, было первым алгоритмом, который строил набор лучших кодов переменной длины.

Мы начинаем с набора из n символов с известными вероятностями (или частотами) появления. Символы сначала упорядочиваются в порядке убывания их вероятностей. Затем набор символов делится на два подмножества, которые имеют одинаковые (или почти одинаковые) вероятности. Всем символам в одном подмножестве присваиваются коды, начинающиеся с 0, в то время как коды символов в другом подмножестве начинаются с 1. Каждое подмножество затем рекурсивно делится на два подсубмножества с примерно равными вероятностями, и второй бит всех кодов определяется аналогичным образом. Когда подмножество содержит всего два символа, их коды различаются добавлением еще одного бита к каждому. Процесс продолжается до тех пор, пока не останется больше подмножеств.

Алгоритм Шеннона-Фано не гарантирует генерацию оптимального кода. По этой причине коды Шеннона-Фано почти никогда не используются; кодирование Хаффмана почти так же вычислительно просто и производит префиксные коды, которые всегда достигают наименьшей возможной ожидаемой длины кодового слова, при условии, что каждый символ представлен кодом, состоящим из целого числа бит. Это ограничение часто не требуется, поскольку коды будут упакованы друг за другом в длинные последовательности. Если рассматривать группы кодов одновременно, кодирование Хаффмана по символам оптимально только в том случае, если вероятности символов независимы и представляют собой некоторую степень половины, т.е. 1 / 2^k. В большинстве ситуаций арифметическое кодирование может обеспечить большую общую компрессию, чем кодирование Хаффмана или Шеннона-Фано, поскольку оно может кодировать в дробных числах бит, что более точно приближает фактическое информационное содержание символа. Однако арифметическое кодирование не вытеснило Хаффмана так, как Хаффман вытеснил Шеннона-Фано, как из-за большей вычислительной сложности арифметического кодирования, так и из-за того, что оно охвачено несколькими патентами.

В методе Фано символы упорядочиваются от наиболее вероятных к наименее вероятным, а затем делятся на два набора, общие вероятности которых максимально близки к равным. Всем символам затем присваиваются первые цифры их кодов; символы в первом наборе получают "0", а символы во втором наборе получают "1". Пока остаются наборы с более чем одним элементом, тот же процесс повторяется для этих наборов, чтобы определить последующие цифры их кодов. Когда набор сокращается до одного символа, это означает, что код символа завершен и не будет являться префиксом кода другого символа.

Алгоритм производит довольно эффективные кодировки переменной длины; когда два меньших набора, полученные в результате разбиения, действительно имеют равные вероятности, один бит информации, используемый для их различения, используется наиболее эффективно. К сожалению, кодирование Шеннона-Фано не всегда производит оптимальные префиксные коды; набор вероятностей {0.35, 0.17, 0.17, 0.16, 0.15} является примером того, что будет присвоено неоптимальные коды с помощью кодирования Шеннона-Фано.

### Кодирование Хэмминга

Код Хэмминга — это код для исправления ошибок, используемый для обеспечения точности данных во время передачи или хранения. Код Хэмминга обнаруживает и исправляет ошибки, которые могут возникнуть при перемещении или хранении данных от отправителя к получателю. Этот простой и эффективный метод помогает повысить надежность коммуникационных систем и цифрового хранения. Он добавляет дополнительные биты к исходным данным, позволяя системе обнаруживать и исправлять ошибки с одним битом. Эта техника была разработана Ричардом Хэммингом в 1950-х годах.

Избыточные биты — это дополнительные двоичные биты, которые генерируются и добавляются к информационным битам передачи данных, чтобы гарантировать, что во время передачи данных не было потеряно ни одного бита. Количество избыточных битов можно рассчитать с помощью следующей формулы: 2r ≥ m + r + 1

где m — количество бит во входных данных, а r — количество избыточных бит.

Предположим, что количество бит данных равно 7, тогда количество избыточных бит можно рассчитать по формуле: = 2 4 ≥ 7 + 4 + 1. Таким образом, количество избыточных бит равно 4.

Бит четности — это бит, добавляемый к данным двоичных битов, чтобы гарантировать, что общее число единиц в данных четное или нечетное. Биты четности используются для обнаружения ошибок. Существует два типа битов четности:

* Бит четности: В случае [четности](https://www.geeksforgeeks.org/how-even-parity-is-different-from-odd-parity/" \t "/home/aandrosov/Documents\\x/_blank) для заданного набора бит подсчитывается количество единиц. Если это количество нечетное, значение бита четности устанавливается равным 1, что делает общее количество появлений единиц четным числом. Если общее количество единиц в заданном наборе бит уже четное, значение бита четности равно 0.
* Бит нечетности: В случае нечетности для заданного набора битов подсчитывается количество единиц. Если это количество четное, значение бита четности устанавливается равным 1, что делает общее количество появлений единиц нечетным числом. Если общее количество единиц в заданном наборе битов уже нечетное, значение бита четности равно 0.

Код Хэмминга — это просто использование дополнительных битов четности, позволяющих идентифицировать ошибку.

Шаг 1: Запишите позиции битов, начиная с 1, в двоичной форме (1, 10, 11, 100 и т. д.).

Шаг 2: Все битовые позиции, являющиеся степенью числа 2, помечаются как биты четности (1, 2, 4, 8 и т. д.).

Шаг 3: Все остальные позиции битов помечаются как биты данных.

Шаг 4: Каждый бит данных включается в уникальный набор битов четности, определяемый его позицией бита в двоичной форме:

* **а)** Бит четности 1 охватывает все позиции битов, двоичное представление которых включает 1 в наименее значимой позиции (1, 3, 5, 7, 9, 11 и т. д.).
* **б)** Бит четности 2 охватывает все позиции битов, двоичное представление которых включает 1 во второй позиции от наименее значимого бита (2, 3, 6, 7, 10, 11 и т. д.).
* **в)** Бит четности 4 охватывает все позиции битов, двоичное представление которых включает 1 в третьей позиции от наименее значимого бита (4–7, 12–15, 20–23 и т. д.).
* **г.** Бит четности 8 охватывает все позиции битов, двоичное представление которых включает 1 в четвертой позиции от наименее значимых битов (8–15, 24–31, 40–47 и т. д.).
* **е.** В общем случае каждый бит четности охватывает все биты, где побитовое И позиции четности и позиции бита не равно нулю.

Шаг 5: Поскольку мы проверяем четность, установите бит четности в 1, если общее количество единиц в проверяемых им позициях нечетно. Установите бит четности в 0, если общее количество единиц в проверяемых им позициях четно.

Биты избыточности размещаются в позициях, соответствующих степени числа 2. Как в приведенном выше примере:

* Количество бит данных = 7
* Количество избыточных бит = 4
* Общее количество бит = 7+4=>11
* Избыточные биты размещаются в позициях, соответствующих степени числа 2, то есть 1, 2, 4 и 8.

Предположим, что данные, которые необходимо передать, имеют вид 1011001 от отправителя к получателю, тогда биты будут размещены следующим образом:

Особенности кода Хэмминга такие:

* **Обнаружение и исправление ошибок:** Код Хэмминга предназначен для обнаружения и исправления однобитовых ошибок, которые могут возникнуть во время передачи данных. Это гарантирует, что получатель получит те же данные, которые были переданы отправителем.
* **Избыточность:** Код Хэмминга использует избыточные биты для добавления дополнительной информации к передаваемым данным. Эта избыточность позволяет получателю обнаруживать и исправлять ошибки, которые могли возникнуть во время передачи.
* **Эффективность:** Код Хэмминга — относительно простая и эффективная техника исправления ошибок, не требующая больших вычислительных ресурсов. Это делает его идеальным для использования в сетях связи с низким энергопотреблением и низкой пропускной способностью.
* **Широкое применение:** Код Хэмминга — широко используемый метод исправления ошибок, используемый в различных приложениях, включая телекоммуникации, компьютерные сети и системы хранения данных.
* **Исправление одиночных ошибок:** код Хэмминга способен исправлять однобитовые ошибки, что делает его идеальным для использования в приложениях, где существует вероятность возникновения ошибок из-за внешних факторов, таких как электромагнитные помехи.
* **Ограниченное исправление множественных ошибок:** Код Хэмминга может исправить только ограниченное количество множественных ошибок. В приложениях, где вероятно возникновение множественных ошибок, могут потребоваться более продвинутые методы исправления ошибок.

Преимущества кодирования Хэмминга:

* Код Хэмминга может обнаруживать и исправлять однобитовые ошибки, повышая надежность данных при передаче и хранении.
* Он добавляет минимальное количество избыточных битов к исходным данным, поддерживая хороший баланс между целостностью данных и накладными расходами. Алгоритм генерации и проверки кода Хэмминга прост и может быть легко реализован как в аппаратном, так и в программном обеспечении.
* Обнаруживая и исправляя ошибки, код Хэмминга обеспечивает точность полученных данных, снижая вероятность их повреждения.
* Код Хэмминга широко используется в различных областях, таких как компьютерная память (ОЗУ), устройства хранения данных и системы связи.
* По сравнению с более сложными кодами исправления ошибок код Хэмминга обеспечивает экономически эффективное решение для приложений, где достаточно исправления однобитовых ошибок.

Недостатки кодирования Хэмминга:

* Код Хэмминга может исправлять только однобитовые ошибки. Он не способен исправлять многобитовые ошибки, что ограничивает его эффективность в средах с высоким уровнем ошибок.
* Хотя код Хэмминга может обнаруживать однобитовые и некоторые двухбитовые ошибки, он не может обнаружить все многобитовые ошибки. Это снижает его надежность в некоторых приложениях.
* Хотя он использует меньше избыточных битов по сравнению с некоторыми другими методами исправления ошибок, добавление этих битов все равно увеличивает общий размер данных, что может стать недостатком в средах с ограниченной полосой пропускания.
* Реализация кода Хэмминга требует дополнительных аппаратных или программных ресурсов для обнаружения и исправления ошибок, что может стать ограничением в системах с ограниченными ресурсами.

### Список литературы

1. FB, Алгоритм Шеннона-Фано: эффективное кодирование информации (https://fb.ru/article/511227/2023-algoritm-shennona-fano-effektivnoe-kodirovanie-informatsii) - 2024
2. Хабр, Ещё раз про алгоритм сжатия Хаффмана (https://habr.com/ru/amp/publications/771572/) - 2023
3. Хабр, Код Хэмминга. Пример работы алгоритма (https://habr.com/ru/articles/140611/). - 2012
4. Team Ques10, Data Compression and Encryption. - 2020 May.
5. Sky McKinley, The Hamming Codes and Delsarte’s Linear Programming Bound, Professor John S. Caughman - 2003 May
6. O’REILLY, Understanding Compression / DATA COMPRESSION FOR MODERN DEVELOPERS, Colt McAnlis & Aleks Haecky. - 2016 July
7. Springer, Data Compression / The Complete Reference / Third Edition. - 2004