Сделано в [Whisper AI](https://t.me/WhisperSummaryAI_bot)

Теория информации, Григорьев А.А., Лекция 05, 03.10.20

Чистая версия. Дата создания: ⁨04.10.2025⁩

О! Знать так. Закончили мы более-менее рассматривать хаффмановское кодирование. Это когда есть большой алфавит какой-нибудь, состоящий из M букв алфавита X. X первая, XMT. И это M большое. И тут содержится сообщение с разными вероятностями. Значит, X1 и так далее, XM. И у них есть вероятности P1, P1. м вот так такая обстановочка и мы хотим это закодировать скажем в маленький ухо нет ну например двоичный уфа ведь поток 0 и единица вот стало быть при этом мы хотим поступить так мы каждому из этих букв большого ухо это ставим соответствие кодовое слово двоичное и передаем их вместо буквы тот самый двоичное кодовое слово мы стал быть у этого слова есть длина и Вот у нас с вами появляется сообщение XJT. Оно передается кодовым словам длины LJT. И у этого всего есть вероятность PJT. У нас появляется такая вот мера эффективности, как средняя длина кодового слова. Дворичная сумма PJT на LJT. L большое. Ну и вот мы с вами показали... что при любом способе кодирования, с однозначным декодированием, конечно, вот эта средняя длина ограничена с двух сторон. Следующим образом, она больше или равно энтропии вот этого источника H от X. У этого источника имеет место быть энтропия. Сколько-то битов на букву. Ну вот, соответственно, длина, увы, но превышает энтропию. Ну и стало быть, существуют способы кодирования, которые дают заведомо h от x меньше, чем h от x плюс единица. Ну то есть среднюю длину мы зажали с двух сторон. Мы показали, что она заведомо больше вот этого, но можно построить префиксный код, такой, который даст среднюю длину меньше, чем вот этого. Ну и стало быть, был алгоритм построения этого самого префиксного кода, а также алгоритм кодирования и декодирования. Алгоритм построения префиксного хода он такой. Мы с вами рассматриваем набор вероятности P1, P2, PN, T. Ну и стало быть упорядоченным в порядке возрастания. И оказывается, скажем, что P1 меньше P2 меньше ранжируемых по возрастанию. А после этого берем две самые... маловероятные буквы и объединяем их в одну с вероятностью P1 плюс P2. И рисуем на верхнем узле, на верхнем ярусе нашего дерева две вершины с вероятностями P1 и P2 и объединяем их в одну вершину виртуальную с вероятностью P1 плюс P2. Вот эти две объединили в одно с вероятностью P1 плюс P2. У нас получился алфавит уже Из М минус одной буквы, одну букву мы выкинули. Ну и стало быть, далее все продолжается точно так же. Вот оставшиеся вероятности P1 плюс P2, P3 и так далее Pn. Ранжируем в порядке возрастания, выбираем две самые маленькие вероятности и их объединяем снова в одну. Ну, скажем, вот так. может быть, а может быть как-то иначе. Ну и в результате у нас вытанцовывается двоичное дерево, которое и дает Афмановский код. Это дерево состоит из какого-то вот, где-то есть концевой узел, корень этого дерева, и из него выходит по два пути, на которых написано 0,1, 0,1, 0,1, 0,1. И вот путь в каждую из вершин дает нам кодовое слово, отвечающее этому сообщению. Кодировать сообщение, кодирование осуществляется так. У нас поступает некое сообщение из источника, вываливается. Мы его отыскиваем среди вот этих концовых вершин. И видим кодовое слово, которое к нему ведет. Это кодовое слово отправляем в канал, начиная с корня. Вот 1, 0, 0, 0. Или на 1, 0, 0, 1. В зависимости от того, какое это сообщение. На приеме публика, принимающая сторона, изучает этот битовый поток. И начиная с корня, ну вот приняли из канала единицу, пошли сюда. Приняли нуля, пошли сюда. Приняли единицу, пошли сюда. Ага, это концевой узел. На нем написана буква Хжитая. Ее выдали на выход, вернулись в корень. Мы продолжаем желать входной потом битвы. Вот так происходит декодирование по Хаффману. Значит, кодирование и декодирование таким образом мы с вами посмотрели. Ну вот, первое, что тут возникает в плане обобщения, это... Повернуть этот алгоритм наоборот, то есть посмотреть ситуацию. Вот это кодирование по ТАН-СТОВА, я вам сейчас буду предлагать. ТАН-СТОВА. Первый, кто предложил некое обобщение хаффмановского кодирования. Значит, хаффмановское кодирование, которое мы сейчас рассмотрели, это очень древняя вещь. Оно известно где-то с 70-х годов. Давно и активно применяется во всяких алгоритмах сжатия данных. В общем, это такая очень развитая и популярная вещь. А вот обобщение Танстого этой процедуры. Он, этот товарищ, он повернул все наоборот. Он посмотрел ситуацию, когда у нас есть входной поток в магом алфавите, скажем, двоичном. Вот у нас идет битовый поток какой-то. А мы его хотим закодировать в большой алфавит. Алфавит «Г». Ну вот мощность Е с буквами Y1, Y3. Ну вот типичная ситуация, когда такая потребность возникает, закодировать малый эвфорид большой. Вот когда мы с вами будем рассматривать радиоканал, то мы увидим, что по эфиру мы передаем точки созвездия. И вот, скажем, у нас имеет место быть трехточечное созвездие. То есть мы по каналу можем передать одну из вот этих точек. Но это вот наша М. М равно 3 в данном случае. А входной поток у нас битовый. 0,1. 0,1. Требуется этот битовый поток перекодировать в троичный алфавит. Наиболее экономно. Вот тут нужно кодирование танцтового. Идея простая. Теперь мы будем строить дерево Хаффмана для источника. То есть мы будем из источника выбирать кодовые слова согласно некому префиксному коду, который мы построим. И, стало быть, выбрав это слово, мы будем отображать его на символу фавита. То есть фактически процедура декодирования Хаффмановского кода становится процедурой кодирования теперь. Вот у меня есть битовый поток от источника. Он вот так ко мне на вход приходит. Я отбираю 0 и 1. У меня построено какое-то дерево. Я иду по этому дереву, пока не приду в концевом узле. В концевом узле написана одна из букв этого луфавита. Например, можно построить дерево такое. Вот корень. Вот у меня 0 и 1. И вот еще 0 и 1. И вот здесь я напишу символы выходного луфавита. А, Б, А. Смотрите, как я осуществляю кодирование. Если пришла единица, я пошел налево, потом пришел 0, я попадаю в букву B. Я передаю троичную букву B. Потом у меня снова единица 0 опять букву B. А вот если 2 0 тут написать, то это будет 0, это буква A, еще 1 0, это буква A. И вот 1 0, это стало будет буква C. Ну вот я, значит, последовательно... Жуя вот с этой стороны поток битовый, выделяю в нем пути на дереве кодирования. Вот этот путь, вот этот путь, вот этот путь. Ну и, стало быть, посылаю на выход символа троичного алфавита. Вот для того, чтобы реализовать это кодирование эффективно, желательно, чтобы выходной алфавит стал равновероятным. То есть, чтобы энтропия выходных, вот этого выходного алфавита, сообщение было максимально желательно чтобы все три вероятности вероятности букв эй би и си были бы по одной трети вот так хочется построить это самое дерево ну то есть символ выходного потока будет переносить максимальную энтропию при условии что его буквы имеют равновероятно тогда каждая буква переносит максимальную энтропию поэтому дерево надо строить так чтобы все его концевые узлы были по возможности равновероятны. Ну вот на тот Анстон предложил эффективную процедуру, которая, в общем, напоминает построение кода Хаффмана, но только ведется это построение не сверху вниз, как у кода Хаффмана, а наоборот, снизу вверх. Вы начинаете с корневой вершины, у вас пока больше ничего нет. Итак, на вход вам приходит Один из символов входного лукови, то он у нас двоичный. И вы знаете, с какой вероятностью этот символ 0 является нулем, а с какой единицей. Или если это куичный символ, то вы знаете распределение вероятности всех вот этих вершин. Ну, скажем, это вот P, а это 1 минус P. Ну, это при условии, что во входном потоке вероятность 0 равна P, а вероятность 1 – 1 минус P. Ну а далее как? Я должен это, по идее, построение этого дерева продолжать до тех пор, пока я не получу нужное число M выходных вершин. У меня есть мощность выходного алфавита, вот она составляет M. Ну и стало быть, надо это дерево продолжать вверх, пока не получится нужное число концевых вершин. Ну и желательно, чтобы эти концевые вершины обладали равными вероятностями. Ну вот рецепт простой. Надо из существующих вероятностей дробить на две ту, которая самая большая. Ну вот если p меньше 1,2, маленькая, то единица минус p, это большая вероятность. Ну вот для случая p меньше 1,2, ну стало быть вот эту вершину, она вероятная. Я ее делю на 2 и получаю тут P на 1 минус P и 1 минус P в квадрате. Это вот 1 минус P умножен на P и еще раз умножен на 1 минус P. Вот у меня получились 3 вершины, вот их вероятности. Это 0, это 1. Из этих оставшихся 3 я опять выбираю. Рецепт очень простой. Бьем, дробим на части ту вершину, которая самая вероятная из существующих. Вот у меня существует эта, эта и эта вершина. Из них какая-то вершина обгадает наибольшей среди имеющихся вероятности, и я продолжаю бить. Ну, скажем, если это вот эта вершина, то я ее побью на p, единица минус p квадрат, единица минус p куб. У меня получится 4 уже вершины. Раз, два, три, четыре. Вот их вероятности выписаны. Ну и продолжаю таким образом. Вот я, стало быть, выбираю из существующих вершин самую вероятную. Ну, скажем, вот теперь это может оказаться вот эта вершина. Я не убью на части. И так далее. Ну и после нужного числа итераций я получаю заданное число m букву. которые вот тут напишу на этих самых, на концах. И, значит, по построению окажется, что вероятности всех этих букв, они почти, ну, по возможности сделаны одинаковыми. Те вероятности, которые сильно выступают над остальными, я их уменьшаю путем деления этой вероятности на две части. Одна часть пошла сюда, другая часть пошла сюда. Сумма этих вероятностей равна вероятности этой вершины. Вот так поступаю, я получаю на верхнем ярусе, где-то там наверху, M большой вершин, у которых вероятности, нельзя сказать, что они одинаковы, но они максимально уравновешены, скажем так. Вот это код Танстога. Ну и стало быть, кодирование этим кодом ведется так, как я вам показал. Вы начинаете жевать поток источника символ за символом, И идете по этому дереву, до тех пор, пока не дойдете до концевой вершины. Дойдя до концевой вершины, вы отправляете в выходной поток ту букву, которая тут написана в этом концевом узле. И начинаете с корней снова желая рассматривать входной поток до тех пор, пока снова не попадете в концевой узле. Вот у вас получился алгоритм фактически декодирования кода Хаффмана, Но это теперь кодирование кода от Анстога. То есть это фактически мы думаем так, что этот код является кодом Хафмана для какого-то источника, и мы декодируем этот код в поток символов того самого источника. То есть все поставлено задом наперед. Теперь процедура декодирования. Это процедура декодирования кода Хафмана. Мы вот в этом... В потоке выделяем кодовые слова префиксного кода согласно выбранному дереву и каждому этому кодовому слову ставим в соответствии выходной алфавит. Вот такая полезная вещь. Значит, известно, что вот тут предложенная процедура кодирования, она оптимальна. Лучше не бывает. Она дает максимальную энтропию на букву выходного источника или максимальную длину кодового слова То есть количество битов на букву выходного алфавита максимизируется. То есть средняя длина, тут ее желательно делать не минимальная, а максимальная. То есть тут хотелось бы, чтобы буква выходного алфавита, одна из М букв, переносили как можно больше символов битов вот этого входного потока. То есть средняя длина желательно, чтобы была максимальная. Но это достигается, когда все буквы выходного алфавита равновероятны, а стремимся мы к этому путем разбиения больших вероятностей на части. Ну и тем самым мы достигаем почти равновероятных вот этих букв. Ну вот так. Значит, успех наш зависит от того, в построении этого дерева и его эффективности, зависит от количества выходных узлов. Если выходных узлов много, то есть число М большое, то, скорее всего, эти вероятности после деления на части окажутся почти одинаковыми для любой модели источника, распределения вероятности символов источника. Вот это коды Танстола, если хотите, ничего тут особо нового нет, это обращение кодов Хаффмана. Они реально применяются для перекодирования моделей. Вольных алфавитов, то есть алфавитов потоков информационных с малым размером алфавита, ну вот, в большие алфавиты. Ну вот, скажем, вот смотрите, если двоичный битовый поток, если у меня на входе двоичный битовый поток 0,1, 0,1, 1, 1, с вероятностью P 1, 2, то есть символ равновероятный. Я хочу его перекодировать в троичный алфавит ABC. Если я просто так и оставлю три выходных символа, то у меня вариантов-то будет немного. Будет так. Значит, у меня корень, две исходящие вершины с вероятностями одна вторая, каждая вот эта нуля, эта единица. Ну и дальше любую из этих вершин, у них одинаковые вероятности, я вынужден буду расщепить на два пути. Вот у меня 0 единица, тут вероятность 1 четвертая, тут вероятность 1 четвертая. Вот у меня дерево танцтова для трехсимбольного выходного алфавита. Вот тут A, вот тут B, вот тут C. Вот построенное для вот такого источника равновероятных двоичных символов. Оно плохое, потому что, видите, вероятность этого символа 1 вторая, а вероятность этих символов по 1 четвертой не сильно различается. Ну и стало быть у меня в результате будет так. 0 будет передаваться всегда буквой А, а вот 1, 0 и 1, 1 будут передаваться буквами B и C. Ну то есть вот у меня получается, что среднее количество символов на букву выходного алфавита, вот тут один символ, он с вероятностью 1, 2. И вот два символа на букву выходного алфавита, но это вероятности 1, 4 и 1, 4. Значит, два символа 1,4, два символа 1,4. Значит, три вторых символа. Один, одна вторая. Длина 1 идет с вероятностью 1,2. И длина 2 тоже идет с вероятностью 1,2 в целом. 1,4 это 1,4. Ну, значит, получается 1 умножить на 1,2 плюс 2 умножить на 1,2. Ну, то есть три вторых бита на символ. Дает мне вот эта схема кодирования. Три вторых бита, ну полтора бита на символ выходного источника в среднем. Значит, это все можно усовершенствовать, если увеличивать размер выходного алфавита. Ну то есть для того же тройничного алфавита можно рассмотреть не одиночные символы ABC, а их пары. Вот скажем, AA, AB, AC, ну и так далее. Любая первая буква и любая вторая буква. Их всего будет трижды три. Это будет B, A, C, A. Итого их будет девять штук. Можно строить дерево для девятисимбольного выходного алфавита. Он вообще-то трехзначный. Но в качестве выходных букв я взял пары букв. Их девять штук. 9 пар. Ну и стало быть, вот можно строить дерево. Оно будет уже более равновешенным. Надо продолжать эту процедуру до тех пор, пока не получите 9 концевых вершин. Чтобы на них, на этих концевых вершинах написать символы выходного алфавита. Но дальше это можно продолжать. Можно рассматривать трешки. Вот такие всевозможные. Букв ABC. Таковых трешек будет 3 в кубе. То есть 27 штук. Вот у вас уже получится код Танстога, басолкой такое дерево, с 27 концевыми вершинами. Оно уже будет еще более уравновешенное. Чтобы это не делать на доске, я давайте это вам дам в качестве домашнего задания. Значит, смотрите. Домашнее задание такое. Построить неоптимальное дерево Танстога для кодирования входного битового потока нулей в единиц с вероятностью p1,2, В 27-символьный выходной алфавит, состоящий из комбинации х0, х1, ну, у у меня было. Значит, у, у0, у1, у2, где у – это нечто из алфавита ABC. Ну, то есть, если вместо у подставить буквы ABC, то получится 27 возможных комбинаций. У вас дерево будет 27 букв в выходном алфавите. И вам надо построить оптимальное дерево и оценить среднюю длину L, число битов, среднее, на символ. Вот мы с вами построили коробку и дерево для однобуквенного алфавита. Увидели, что там эта средняя длина составляет 3 вторая. Ну вот вы сравните, во что это выльется, когда дерево вот такое более высокое. Сколько битов на символ мы можем передавать. Значит, смотрите, максимальное при трехсимбольном алфавите вот этом, ну вот количество битов на символ, оно должно быть таким. У нас 27 вариантов. Если все эти варианты равновероятны, то мы должны бы иметь логарифм по основанию 2 27 вариантов. битов на трехсимбольную комбинацию ну вот это что такое тут 27 это 3 в кубе значит 3 логарифма по основанию 2 от 3 битов на 3 символы или логарифм по основанию 2 3 битов на один символ выходного алфавита вот если поделить на 3 вот это количество битов на трехсимвольную комбинацию, а вот это количество битов на один символ. Интересно оценить, сколько битов на символ вы получите при таком кодировании в трехбукленную комбинацию. Это будет заведомо меньше, чем алгоритм по основанию 2,3, но больше, чем 3,5. Стало более-менее легко понять, что если объем выходного алхавита увеличивать еще, то в принципе вот к этому пределу можно приблизиться. У нас все вершины на верхнем ярусе, все вершины на верхнем ярусе станут почти равновероятными, и их будет много. Ну вот, собственно, вот это кодирование Танстолл. Она вещь очень полезная в тех случаях, когда нужно простой источник перекодировать на большинное какое-то созвездие с каким-то количеством букв, не в некратном степени двойки. Ну или... В принципе, источник может быть посложнее, не такой, как у меня, непоследовательность равновероятных символов, а, скажем, двоичный источник, у которого одна буква имеет вероятность 3 четвертых, а второй символ вероятности 1 вторую. Вот можно строить оптимальное дерево Танстого для этой ситуации. Там уже будет иное разбиение вероятностей. Ну и, стало быть, тем не менее, вы получите некую оптимальную схему кодирования вот такого источника в трехсимвольный алфавит. Вот это схема Танстога. Ну а дальше есть великое объединение вот этих двух методов. Это так называемое арифметическое кодирование. Арифметическое, оно же символьное кодирование. или интервальная, вот его так называют, интервальная. Альфметик импорт. Интервальное кодирование. Арифметическое, оно же интервальное. Ну вот это как бы гениальная идея, которая обобщила достижения как Хаффмана, так и Танцова и объединила их в некую единую схему. Идея базируется вот на чем. Значит, смотрите, с вероятностью P тесно связан интервал на вещественной оси от 0 до 1. Это как бы область определения, распределения вероятности одномерных. Интервал от 0 до 1. Ну и, стало быть, вот смотрите, вы берете некую статистическую модель источника. То есть у вас есть источник X, который выпечатывает буквы Х0, потом Х1, потом Х2. Из некого алфавита. Вот так. Ну и, так как быть, вы должны исходить из того, что у вас, как обычно, при таких вот... Что при Хаффмановском кодировании, что при Танстоговском, важна статистическая модель источника. Она позволяет строить оптимальное дерево. Так вот, в треопедическом кодировании предполагается весьма общая модель источника. Она задается следующим. Она задается распределением вероятности первой буквы. P от X. Первая буква из какого-то алфавита, скажем, двоичная, и имеет вероятность такую и такую разных значений. Вот это вероятность однобуквенная. Потом идет вероятность буквы х1 при условии конкретной буквы х0. То есть идет распределение p от х при условии х0 конкретного. Х большое, х0 маленькое. Для всех х0. Это условная вероятность. То есть, скажем, если речь идет о каком-нибудь русском алфавите, то есть... Вероятность буквы P от А одиночной. И есть вероятность буквы от А при условии, что предыдущая буква была там B, например. Вот набор таких вероятностей. P от X при условии всех возможных X0. Потом идут вероятности P от X третьей буквы. P от X2 при условии X0, X1. Ну и так далее. Вот идет набор таких условных вероятностей, которые поддают распределение вероятностей очередной буквы изжитая при условии, что набор предыдущих букв известен. Вот набор этих вероятностей является наиболее общей моделью, статистической моделью источника. Вероятностная модель источника задается одномерным распределением и множеством условных распределений. Ну и, стало быть, когда эта модель задана, то она определяет, Разбиение отрезка 0 единица на интервалы вполне определяет. Вот смотрите. Пусть у меня имеется, скажем, три буквы. Ну, то есть P от X это P от A, P от B и P от C. Их полная сумма равна единице. И поэтому с вот этим распределением вероятности я могу связать Разбеение этого единичного интервала на подинтервалы. На вот такой подинтервал, вот такой подинтервал и вот такой подинтервал. Так что длина этого интервала есть P от A, длина этого интервала есть P от B, а длина этого интервала есть P от C. Вот, значит, первая буква. Модель задана набором вот этих вероятностей. И, значит, первая буква определила мне разбиение интервала 0,1 на подинтервалы. Подинтервалы я сознательно беру вот с этим включенным концами. Они с этой стороны замкнутые, а с этой стороны разомкнутые. Так что единица не принадлежит исходному интервалу. То есть у меня рассматривается интервал именно от 0 до 1. И со стороны единицы он открытый. Ну и стало быть, и бьется он тогда на непересекающиеся интервалы, и любая точка попадает в один из этих интервалов. Либо сюда, либо сюда, либо сюда. Ну а дальше так. На следующем этапе вы рассматриваете вот эти условные вероятности. p от x при условии x0. И вот если x0 было вот таким, то вы берете этот отрезок, он уже теперь у вас, вот какой-то интервал длины p от b, И вы его бьете уже на три подинтервала. Вот такие. Согласно вот этому распределению вероятностей. p от x плюс х0. Вот у вас интервалы первого уровня, однобуквенные. Это интервалы второго уровня, двухбуквенные. Но неизмешно говорить, что все остальные интервалы также подъются на три части. Вот на такую часть, на такую часть, на такую часть. Вот. Ну и так я могу продолжать и далее. Значит, теперь каждый из этих интервалов в соответствии с распределением вероятности P от X при условии X0, X1, а X0, X1 они у меня уже определены. Речь идет об одном из этих интервалов. Я в свою очередь бил на части, вот я его так выношу отдельненько, рисую его как интервал и разбиваю его пропорционально вероятностям, вот этим трём вероятностям. Вот этот интервал, вот этот интервал и вот этот интервал. Вот такая красота. То есть вероятностная модель источника выливается у меня в разбиение интервала от нуля до единицы на подинтервалы. Вот такие. Подинтервалы бывают первого уровня однобукленные, второго уровня. Вот первого уровня их будет три штуки в моей модели. двухбуквенных будет 9 штук, вот этих будет 27 штук, на следующем уровне 81 под интервал и так далее. И вот в этом разбиении воплощена вся статистическая модель источника. Все, что я о нем знаю, сосредоточено вот в этих условных вероятностях, а эти условные вероятности вполне определили длинные интервалы в разбиении. Вот. Ну, стало быть, после того, как я проведу или тапов вот этого разбиения, я получу свой отрезок от нуля до единицы. Вот если размер алфавита Q, то тут будет Q вольтой степени интервалов разбиения. Вот они, интервалы. Значит, этот интервал, его длина равна как раз вероятности L-буквенного сообщения источника. Сообщение из L-букв. То есть это... p от x0 на p от x1 при условии x0 на p от x2 при условии x1, x0, x1 и так далее на p от xl. xl при условии xl минус 1 и так далее и x0. Вот такая вот запись. Ну то есть длина каждого интервала это просто вероятность L-буквенного сообщения источника в нашей статистической модели. Ну и стало быть, идея, вот то есть это вот вероятность P, та самая, которая, есть вероятность того, что у меня первые буквы, первые L-буквы сообщения, ну вот такие, как они есть для этого интервала. То есть это P от X0, XL-1. Вероятности из L-буквенного сообщения. Ну а дальше как? Поскольку мы с вами модели источника согласовали, то эти разбиения у нас имеются на стороне передачи и на стороне приема. То есть вы знаете модель источника, если вы принимающая сторона, вы знаете все эти вероятности. И значит я построил вот это разбиение и имею границы интервала. Ну, то есть я, собственно, разбил отрезок 0,1 на q вельтой степени интервалов каждой определенной длины. И это длина пропорциональна вероятности того сообщения, которое приводит к этому интервалу. И вы у себя тоже построили такое же разбиение. Что остается? Я должен, если я передающая сторона, просто сообщить вам номер этого интервала. Только и всего. Сказать, что вот... Те L-букв, которые я сейчас собираюсь вам передать, они приводят в интервал номер 25. Вот в этот интервал. Вот этот интервал номер 0, вот этот интервал номер 1. То есть указать номер интервала. Или границы этого интервала, или какую-нибудь из границ этого интервала. В общем, пальцем указать на интервал, куда приводит L-символная последовательность букв источника при этом разберении. И тогда вы посмотрите у себя на это разберение и поймете, что в это место приводит вот такая L-символьная последовательность. Ну, значит, на сообщении вам, на передающей на сторону приема информации об этом интервале, я должен затратить, как каждому понятно, типа гогарифм по основанию 2 единица на p битов. Когда мы кодировали по Хаффману, у нас что было? Хорошо было, когда длина кодового слова составляла как раз логарифм по основанию двух единицы на p. Вот если вы сумеете мне так или иначе сообщить информацию об интервале с помощью вот такого количества битов, то окажется, что вы реализовали оптимальное кодирование. У вас Энтропия выходная. У вас как раз сумма P. Если это длина сообщения, то средняя длина будет сумма P на галларифм по основанию 2 единицы на P. То есть как раз энтропия нашего источника. Или символ на энтропии. То есть оптимальность кодирования таким образом обеспечивается вот чем. Вы должны передать информацию об интервале, Затратив на это, ну порядка, немногим, ну меньше не удастся, но не сильно больше, чем грифм по основанию 2 единицы на p битов, где p это длина этого интервала. Просто вся вероятность единичная, она разделилась между интервалами и сюда попала доля p. Это вероятность попадания в этот интервал. Ну и значит, она же длина этого интервала. Ну и стало быть, Количество битов должно примерно равняться логарифму этой длины. Как это делать? Публика придумала это делать так. Значит, помимо вот этих наших интервалов, единичный отрезок можно бить на цилиндры. Вот примерно таким образом. Значит, если у нас отрезок 0,1, вот он, интервал 0,1, мы его будем бить стандартным образом по дихотонии. То есть делить пополам каждый раз. Вот я его поделил пополам, и этот поделил пополам. От нуля сюда половинка, и сюда половинка. То есть тут вероятность 1,2, и тут вероятность 1,2. То есть деление на цилиндры, это деление согласно двоичному источнику, у которого... Вероятность нуля и вероятность единиц составляют одно второе. Вот на первом уровне у меня получились цилиндры длины 1,2. На втором уровне у меня окажутся цилиндры длины 1,4. Каждый. Вот так. То есть каждый из этих цилиндров я еще побью на 2. Ну и так далее. На L-1 уровне у меня окажется 2 в L-той степени цилиндров. Каждый из этих цилиндров кодирует вероятность типа единицы на 2 вольтной степени. Вот эта вероятность 1,2, это 1,4. Это единица на 2 вольтной степени. И все они полностью покрывают отрезок 0,1. Для передачи информации об этом цилиндре мне хватит L-битов. Цилиндры, в отличие от интервалов, устроены так, что у них границы лежат по степеням двойки. И поэтому вот сюда попадают для передачи вот этой границы мне достаточно L битов. Вот эта граница, это все нули. Вот эта граница, это все нули. Вот у меня имеется L битовая комбинация, но и стало быть у меня имеется L нулей, она приводит вот на эту границу. Есть комбинация из всех нулей и последней единицы, она приводит вот на эту границу. В общем, работает двоичный счетчик. Номер этого интервала, их 2 в этой степени штуки, он просто кодируется двоичным позиционным кодом. Вот этот интервал кодируется кодом 1, 0. Потом идет интервал номер 3, 1, 1. Ну и последний интервал, это когда все единицы, это вот последний цилиндр вот так итак вот мы изобрели разбиение отрезка 01 на цилиндры цилиндра замечательны тем что их концы распредел распорожены в таких удобных точках что за их передачи как раз хватает минимального требуемого количества битов 2 в не минус льтой степени и вы его передаете с помощью l битов и Ну и тогда стала быть идея дальше простая. Она заключается в том, чтобы сообщить мне номер интервала в вашей модели с помощью номера цилиндра, который находится внутри этого интервала. То есть вот смотрите, на каком-то уровне ужасном у вас в этом отрезке появился интервал с вероятностью P. Вы просто дожидаетесь, когда внутрь этого интервала попадет очередной цилиндр вот из этого разбиения. Вот такой вот. И сообщаете мне вот этот номер этого цилиндра. И я знаю, что судьба прибега вот в этот интервал. Разбиение на интервале мне известно. Если вы мне передаете цилиндр, который находится, код цилиндра, который находится целиком внутри этого интервала, то я однозначно опознаю интервал. А зная интервал, я знаю, какая символьная комбинация в него привела. Вот такая вот идея. Ну и вот тут вопрос только в следующем. Сколько битов надо в среднем, сколько битов нужно передать, чтобы цилиндр... Цилиндр же, они все время умельчаются. Каждый следующий цилиндр, он вдвое короче предыдущего. Ну и, чтобы быть... Ясное дело, что найдется цилиндр, поскольку цилиндры все более мелкие становятся. Вот для данного интервала найдется цилиндр, который лежит внутри. Но, возможно, он будет слишком короткий, и для его передачи потребуется много битов. Этого не хотелось бы. Хотелось бы, чтобы этот цилиндр был подлиннее, чтобы для его передачи мало битов потребовалось. Ну и вот тут примерно такое рассуждение. Значит, вот смотрите, берем... который длины l-1 на l-1 уровне. Его длина пусть будет больше. Ну, то есть я хочу вот эту вероятность P замкнуть с двух сторон на больше или равно 2 в степени l-1 и меньше или равно 2 в степени l-1. Вот примерно так. То есть между L уровнем и L-1 уровнем. Вот L-1 уровень это те цилиндры, длина которых превышает длину этого интервала. Значит они внутрь не попадают, они не годятся. Но стало быть тогда этот цилиндр я должен побить на 2 и посмотреть. Вот это уже уровень 2 в L степени, уровень L. Цилиндра L-тового уровня. Ну и вот смотрите, ситуация тут такая, что длина вот этого, она превышает уже и эту длину, и эту длину. Вот длина вот этого интервала, вот по условию, длина этого интервала это P. И значит, а вот у этих длина 2 в степени минус L и 2 в степени минус L. Значит, их сумма больше вот этого интервала, а каждый из них меньше вот этого интервала. Так что вот этот цилиндр не попал бы внутрь этого интервала, но по счастливой, по несчастной случайности он мог бы уже там вполне уместиться. Ну вот, значит, получается так, что вот как бы ни легла вот эта граница, у вас либо вот это расстояние больше половины, либо вот это расстояние больше половины. И поэтому на следующем этапе разделения, разбиения, у вас либо вот этот цилиндр попадет целиком внутрь, либо вот этот целиком попадет внутрь. Но отсюда следует, что для передачи номера вот этого, для передачи вот этого интервала цилиндром, вам потребуется один избыточный бит. Не 2 вольмина, не L-битовый цилиндр, а L-плюс-1 битовый цилиндр, он ляжет внутрь. Либо этот, либо этот. Тут масса вариантов надо рассматривать, но это не очень интересно. Ситуация такая, что поскольку цилиндры становятся все более короткими, то нет сомнений, что кто-то попадет внутрь интервала. Это заведомо. Вопрос только выбрать самый длинный из цилиндров, лежащих в этом интервале. Ну вот тут всякие геометрические построения показывают, что наихудший вариант реализуется вот так, как у меня показано. И требуется один избыточный бит для передачи. Ну то есть минимальное число битов L, достаточно передать L плюс 1 бит, чтобы цилиндр попал внутрь этого интервала. Ну вот это как всегда с этим самым кодированием типа Хаффмана. Ну то есть... Нужна некая избыточность. Ну вот, в результате вы получаете схему, которая позволяет передавать данные почти оптимально. Ну вот это так и выглядит. Значит, на стороне приема и передача строят модели источника, то есть строят разбиение интервала 0,1 на интервалы. Ну и, стало быть, в процессе кодирования, После обработки нужного числа битов источников попадают в некий интервал. А дальше смотрят, какой цилиндр самый большой попал внутрь этого интервала и передают получателю координату этого цилиндра. Вот его начало. Двоичный позиционный код его начала. Я получил от вас этот код, нахожу номер цилиндра, смотрю, какой цилиндр... каком интервале лежит этот мой цилиндр и знаю знаю интервал тем самым и восстанавливаю последовательность двух источников вот это декодирование вот это интервальное кодирование она фактически что делает она преобразует модель заданную модель источника в двоичный код вот так как если цилиндра двоичные то в двоичный код у которого вероятности нулей единиц Одинаковые. То есть если пренебрегать вот этими избыточными битами, то выходной код получается равновероятным. То есть вероятности и нулей, и невероятности и единиц одинаковы. То есть эта схема кодирования, так же как и хаффмановское кодирование, почти оптимальна. Значит, здесь есть проблема, как заканчивать это кодирование. То есть вот вы стали грызть символы источника, один за одним, у вас появляется все более и более мелкое разбиение в соответствии со статистикой этого источника, с его моделью. Ну вот на каком-то этапе надо остановиться и сообщить номер цилиндра. Здесь, возможно, разные стратегии. Можно договориться так, что вот обработал 16 букв и передал то, что получилось. Потом обработал еще 16 букв и тоже передал. Можно 128 букв, а можно 256. Значит, чем больше букв вы будете передавать за одну попытку, чем мельче у вас разбиение, тем у вас вот эта плюс единица, которая лишняя. У вас будет много битов на кодирование цилиндра. Ну вот при длине, когда у меня 256 интервалов на верхнем уровне, у меня на кодирование этого пути будет байт затрачиваться, 8 битов. Ну тогда вот лишний бит стоит 1,9. А если вы 32 тысячи сделаете интервалов, то есть перейдете на 16 разрядные, на 16 итераций, то у вас этот бит будет стоить 1,030. Совсем мало в этом деле. Ну, вот 1,16. Ну, то есть, чем дольше вы ведете кодирование, тем меньше влияние вот этой неидеальности, связанной с необходимостью уложить цилиндр внутреннего интервала. Можно этот процесс не заканчивать вообще никогда. То есть если представить себе, что у меня источник работает непрерывно и постоянно поставляет символа для передачи, то я могу эту процедуру просто никогда не обрывать, не задумываться об обрыве, а работать непрерывно. То есть строить все более и более мелкие разбиения и поступать так. Вот красивая идея, связанная с интервальным кодированием. Вот смотрите, в процессе кодирования вы устраиваете разбиение интервала 0,1 на все более и более мелкие интервалы. И вот когда у вас вот этот очередной интервал, где вы находитесь, попал внутрь какого-то цилиндра, уже не цилиндр внутрь интервала, а интервал внутрь цилиндра, Вы можете мне сообщить номер этого цилиндра на сторону приема, не дожидаясь конца передачи. Сказать, что вот попали в цилиндр номер 35. И с этого момента я перестану рассматривать все остальное как несущественное. Для меня вот этот цилиндр станет интервалом 0-1, в котором я и буду теперь дальше продолжать жить. Ну и, стало быть, по мере уменьшения этого интервала он будет попадать... внутрь все более мелких цилиндров и он не будете вываливать битовые биты которые маркируют начала вот этого цилиндра ну то есть тут по возникает некое запаздывание вы строите интервал а сообщаете мне номер цилиндра внутри которого на ху заведомо находится этот интервал ну и значит тут есть как некий геп вот этот интервал он уменьшается уменьшается от цилиндра может поменьшаться не так быстро и Ну и вот, возможно, некие тут неприятности. Неприятности вот какого рода. Вот может быть так, что вот в этом месте у меня проходит граница двух цилиндров. Цилиндр номер M и цилиндр номер N плюс 1. Вот это bound между ними. Ну и, стало быть, вот бывает, может оказаться так, что при последовательном дроблении вот этих самых интервалов у вас некий интервал, представляющий для вас интерес, Он никогда не попадает внутрь ни одного цилиндра. Получается так. Вот у вас этот интервал. Он лежит на границе двух цилиндров. Следовательно, ни один цилиндр внутри него не находится. Я его укорачиваю, бью пополам. Бью, допустим, на части. Вот на такую часть, вот на такую часть и на такую часть. И опять у меня возник интервал, который лежит на границе цилиндра. Теоретически это может продолжаться бесконечно долго. То есть... Вот смотрите, я могу и этот интервал побить на подинтервалы, и у 5 у меня образовался интервал, который лежит на границе двух цилиндров. Вот этот интервал попадет внутрь какого-то тонкого цилиндра, сделанного из этого. С этим и все в порядке. А вот где bound, где граница, ну там она так и остается границей. Так что вот в этих местах возникают ситуации, когда я... Очень долго не могу сообщить получателю ни одного бита. Я буду ждать и ждать и ждать, пока это все умельчается. И теоретически это уходит в бесконечности, вот для неудачного сложения цепс. Ну, правда, вероятность этого события равна нулю, это же все вероятностные вещи. То есть вероятность этого события уменьшается с ростом уменьшения длины этого интервала, естественно. Ну, это раз. А во-вторых, можно поправить модель. Так, чтобы вот такие сингулярные ситуации исключить. Для этого надо просто граница, при очередном разбиении, границы деления, ну, чуть-чуть сместить. Ну, вот сделать так, чтобы у вас граница интервала вот тут проходит, граница цилиндров. Вот был такой интервал. Вот он. И вот если его грамотно разбить, согласно правилам, то... Вот этот цилиндр попадает вот сюда. Но надо просто вот эту границу взять и немножко переставить вот в это место. И станет хорошо. У нас с вами вот этот цилиндр попадет внутрь вот этого... Вот этот интервал попадет внутрь этого цилиндра. А вот этот интервал попадет внутрь этого цилиндра. Вот так. То есть поправки модели позволяют избежать этой ситуации, а так она в принципе неприятная, имеет место. И при реализации этого дела программной, ну об этом по крайней мере надо знать. Что вот с такими вещами, если вы задумаете кодировать бесконечные потоки, не обрывая процесс кодирования, то вот с такими неприятностями приходится сталкиваться. Проявляется это в том, что вы на ход кодера подаете бит за битом, а он все время умельчает разбиение, и только на выход ничего не передается. По той причине, что он не может указать номер цилиндра, содержащего интервал. Интервал, который получается, все время оказывается на границе цилиндров. Границу цилиндров двигать нельзя, это же степень двойки. У нас же все бьется напополам. Одна вторая, одна четвертая. Границу цилиндра я двигать не могу. А границу интервала я могу двигать, потому что модель в моем распоряжении, она же граница интервала, она определяется моделью. Модель можно немножко поправить. Вот примерно так. Вот это символьное кодирование. Это на самом деле самый популярный в наше время способ кодирования источника. Ну вот он, в частности, применяется для отображения всяких двоичных потоков, ламычные, алфавиты и прочее, и прочее, и прочее. Значит, замечательная особенность этого алгоритма состоит в том, что при правильной реализации, значит, он, к сожалению, вот на него невозможно почти придумать задачу, потому что, видите, этот алгоритм, этот, ну, Алгоритм кодирования предполагает программную реализацию. Надо писать программу, которая бьет отрезок 0,1 на интервалы. Там возникнут заведомо проблемы, связанные с точностью представления чисел. Вот просто предвижу. Ну вот скажем, если у вас 32-разрядная арифметика, то она довольно быстро исчерпается. Потому что если вы длину интервала каждый раз делите пополам, то на 32 шаге длина будет 2 в минус 32. Это предельная точность вашего компьютера. Там уже о разбиениях говорить не придется. Значит, проблема точности представления чисел возникает. Вот эта вот проблема неприятная, она сопряжена с точностью представления чисел. Короче говоря, это некий такой изощренный алгоритм. Но такие вот процедуры имеются, которые по заданной модели источника, набору условных вероятностей, перекодируют этот источник в поток независимых выходных символов в алфавите произвольной мощности, скажем, в троичном алфавите, либо в двоичном алфавите. Значит, техническое задание на этот кодер такое. Набор условных вероятностей источника, Ну, на худой конец может быть источник независимых символов. Тогда это проще всего. Когда все вероятности p от x одинаковые. Ну, то есть p от x при условии x0 равно p от x независимых символов. Тогда это проще всего. Ну, то есть та или иная статистическая модель источника и объем выходного алфавита, который определяет цилиндры. Вот эти самые разбиения отрезка 0,1 на цилиндры. Ну вот мы смотрели двоичный алфавит, тогда это двоичный цилиндр по степени АН двойки, иначе троичный и так далее. Ну и, стало быть, на выходе получается поток равновероятных и почти независимых символов в выходном алфавите. Вот смотрите, вот этим алгоритмом кодирования и декодирования нашлось неожиданно полезное применение вот какое. Вот смотрите, если вы устраиваете некодирование, вот у вас есть кодер, нарисую картинку, вот так. Вот есть у вас кодор, который преобразует поток символов X, вот согласно статистической модели, в поток нулей и единиц равновероятных и независимых. Ну, почти вот такой белый шум. А вот у вас есть декодор, который вот этот поток преобразует в поток символов X. согласно вашей статистической модели. Вот это все стали использовать для формирования символов, потоков символных с заданной вероятностной моделью. Ну, то есть, вот задача стоит так. Сделать ящик, который вырабатывает на выходе символов с алфавитом каким-то, ну вот с заданными условными вероятностями, ну, то есть с заданной моделью. Как его реализовать технически? Делают это дело так. берут декодер вот этого самого арифметического кода и подают именно вход в поток независимых равновероятных символов. Он выдает на выходе символный поток с заданными вероятностями. Ну, то есть реализует вот эту процедуру преобразования кодированного. Ну, то есть поток случайных символов, которые вы подаете сюда, воспринимается как результат кодирования нету потока, ну, вот, неким кодером. Вот на самом деле он просто... получен от источника случайных чисел. Если его подать на декодер, то на выходе получится декодированный символный поток с заданной статистикой, скажем, с заданной вероятностью нуля-единица, с заданными переходными вероятностями. Вот это применяется для формирования сложных шумов, не просто белых шумов с независимыми символами, а шумов с заданными корреляциями. Символные декодеры. Ну вот так. Вот до чего, так сказать, дошла техника символьного кодирования. То есть это вот, ну, как-то так, передовой край, вот это символьное интервальное кодирование, оно же арифметическое кодирование. Вещь совершенно универсальная. Вы можете перекодировать любой источник в источник независимых символов над любым алфавитом с помощью этого алгоритма. И вот почему оно арифметическое, понятно, потому что тесто связано с арифметикой, с разлиянием интервала 0,1 под интервалом. Ну и стало быть, вот так. Значит, ясное дело, что Хаффман и Панстол, они утаптываются в эту схему как частные случаи. Хафмановское кодирование – это просто кодирование большого алфавита, двоичный алфавит выходной. Если вы станете строить это дерево, то вы построите оптимальный код Хафмана. То есть арифметический код для независимого алфавита, для алфавита, у которого буква независимая, и выходной код – это и есть оптимальный код Хафмана. И наоборот, вот этот декодер – это код Танстола. Ну, то есть, это на самом деле, если хотите, кодер-декодер – это каскадное соединение Хаффмановского кодирования и Танстоловского декодирования. Вот это на символы выдает кодовые слова. Номера цилиндров – это кодовые слова. А этот на номера цилиндров выдает результаты декодирования, то есть большой алфавит. Вот это Танстола – это фактически это две половинки одного и того же значительной мере но не совсем так но тем не менее близко по смыслу вот так что это вот некая универсальная схема символьного кодирования вот она такая есть жалению и обсуждать можно только чисто умозрительно потому что писать соответствующий по элборитму непросто это некое искусство некая кухня вокруг этого всего. В общем, они имеются. Они в библиотеках по обработке данных, такие альтметические кодеки, давно реализованы, и можно ими пользоваться готовыми. Значит, блеск и нищета всех этих символьных методов кодирования опирается на простую, связанную с простой вещью. К сожалению, для того, чтобы этот код был оптимальным, нужна хорошая статистическая модель источника. Все построено на статистической модели. Все ваше сжатие. То есть для того, чтобы построить хорошее дерево кода Хаффмана правильное, вы должны знать вероятности букв. Когда этих букв... И букв должно быть много. Ну вот, и значит, вот вопрос, где вы возьмете эту статистику. Ну вот тут то же самое. Для того, чтобы реализовать кодер-декодер, вы должны знать статистическое описание своего источника, Потому что это описание заложено внутрь этого, в обоях этих процедур, в виде последовательности разбиения интервала 0,1 на подинтервал. Ну и дальше так. Если у вас на самом деле статистика источника, который вы кодируете, не соответствует вот этому статистике, по которой вы построили свой кодер, то его характеристики, естественно, быстро деградируют. То есть если, не дай бог, начать сжимать, неправильный источник с другими вероятностями букв, то выходной поток будет ничего, не имеет ничего общего с равновероятным, и, стало быть, вы получите плохую степень сжатия. Вот этот вопрос о том, как быстро деградируют характеристики при нарушении статистики источника, он исследован, довольно быстро деградирует, поэтому тут все неутешительно. То есть статистика должна быть хорошая. общепринято. Ну и, чтобы быть, в этой связи все это становится не очень практически значимым, по той причине, что при больших объемах алфавитов хорошую статистику взять негде. Ведь статистика, она подлежит измерению. Ну, чтобы быть, вот если взять обычный текст, то распределение вероятностей отдельных букв еще можно найти, распределение вероятностей пары букв тоже можно найти, А вот уже распределение вероятностей четверых букв идти проблематично экспериментально. По той причине, что нужно очень много вариантов. Это же вот у вас каждая буква это порядка 40 вариантов и 44 степень. Это довольно небольшой алфавит. То есть вы должны рассмотреть очень большой текст, чтобы надежную оценку вероятностей этих символов создать. Поэтому... Вот это все как бы нищета. Облеск, конечно, то, что это просто эффективно. Так что при наличии какой-то априорной вероятности о свойствах источника применять это нужно, а в отсутствии не очень интересно, потому что сложно. В этой связи люди на этом не остановились в символьном кодировании и стали придумывать схемы универсального кодирования. Постановка, на первый взгляд, звучит дико. То есть требуется придумать универсальный компрессор, штуковину, которая сжимала бы все на свете, независимо от статистики, статистических свойств того, что вы ей подаете на вход. Когда первый раз формулируешь постановку, кажется, что это невозможно. А это возможно. Это модный нынче машин-ленинг, машинное обучение. Чем эта штуковина занимается, компрессор, сжимальщик данных? Он занимается сжатием данных. Кому лучше знать, что ему подсовывают на вход? Он же много-много этого видел. Он может собрать эту статистику и хранить ее внутри себя. Ну, то есть он должен самообучаться. Вы ему подсовываете на вход новые-новые данные для сжатия. Он смотрит за тем, что вы ему подсовываете, и адаптируется так, чтобы сжимать то, что вы ему подаете. Хорошо. Вот и все. Вот это идея универсального кодирования. Ну, вот... Она тоже довольно старая, она уже реализована во всех архиваторах. Ну, что бы быть, она началась все с идеи лембеля. Это, ребятки, еще знаете когда? Это черти когда, когда делались компрессоры для телефонных модемов. Вот были времена, когда интернет работал по телефонным проводам. витой паре со скоростью 38 терабит в секунду. Такие скорости никто уже и не поверит. И там требовалось сжатие данных. Ну и вот Лемпель начал задумываться над этой проблемой. Как сжимать символьные потоки при передаче по модернным каналам. И предавал он простую довольно вещь. Значит, надо на сторонах приема и передача составлять сговори и передавать. Не буквы, а слова. В виде номеров слов в словаре. Ну вот смотрите, какая картина. Вот вы передачи. Я передатчик. У меня есть некий локбук. Словарь. ABC, DDD и так далее. Какие-то последовательности букв. Мой словарик. А вот ваш словарик. И они тождественные. И я вместо того, чтобы передавать вам вот эту комбинацию букв в виде трех словарек, букв подряд, я просто передам номер этого слова в словаре. Вот по каналу. Номер слова в словаре. Вы его получите, гляньте в свой словарик и вытащите оттуда слово под этим номером. Ну и, стало быть, если длина слова сильно больше, чем число битв, требуемых для кодирования номера, то есть, если слов относительно мало, а их длинно большие, то вы получите возможность сильно сэкономить. Вы будете передавать, как правильно, номера слов, которые относительно короткие. Ну и вот, стало быть, тут обыгрывать это все можно бесконечно. Первое, что предложил Лемпель, это вот чего. Он предложил передавать вот так. Значит, скобочка открывается, номер в сговоре, скобочка закрывается, и еще одна буква. Это, значит, пока ногу гуляет. Ну вот, условные скобки, никаких скобок там на самом деле нет, а просто начинается посылка с номера кодового слова в словаре и потом удлиняющая это слово буква. Значит, смотрите, на первом этапе у меня и у вас словари пустые. Поэтому приходит на вход какая-то буква, я смотрю в свой словарь, ничего нет, и я отправляю вам пустые скобки без ничего, пустое множество и букву а которая мне поступила на вход при этом слова а я заношу в словарь и вы получил это сообщение вы даете букву а на выход и заносите ее в свой словарь под номером 1 первое слово вот теперь если мне понадобится еще одну букву а передать то есть за первой буквой передам вторую вот эту то я гляну в словарик И увижу, что буква А там уже есть. Вот это. Значит, я ее поэтому... Значит, буква А уже есть, поэтому я как поступлю? Я даже не буду ее передавать в виде номера. Я дождусь следующей буквы, вот она, скажем, В. И передам вам номер А, вот такую комбинацию, номер 1 и удлиняющая буква В. Значит, вы, получив это от меня... Полезете в свой словарь, выберите оттуда слово под номером 1, это буква А, и пришпандорите к ней букву Б. Занесете АБ в свой словарь под номером 2, и я занесу АБ в свой словарь под номером 2. Вот так мы и будем продолжать действовать. То есть всякий раз вы мне передаете номер слова из имеющегося словаря и удлиняющую его букву, новую букву. И мы каждый раз заносим в срыварь нечто новое при каждой передаче. Ну и вот посмотрите, что тут будет со временем, с годами, так сказать. Со временем у нас с вами накопится срыварь здесь и на приеме, и на передаче. Он будет тождественным. Вот смотрите, суть в том, что мы на стороне передачи и приема над срыварем выполняем одинаковые операции. Если я что-то заношу, вот я что-то заношу, то и вы что-то заносите. Поэтому сровари у нас всегда тождественные. Их не надо передавать по каналу. Мы их просто строим по одинаковым правилам. Правила эти однозначны, поэтому они получаются одинаковые. Ну и стало быть, с годами в этих сроварях накопятся часто употребляемые срова. Ну и стало быть, я, как правило, буду передавать их номера по каналу. Они будут длинные, они будут удлиняться со временем. Их будет количество расти. Ну вот, стало быть... И как только у меня для передачи номера слова окажется, что требуется значительно меньше битов, чем для комбинации букв, входящих в это слово, ну вот, скажем, для передачи комбинации букв ABCD мне нужно, вот, скажем, по 6 битов на каждую букву. 2 в 6 — это 64 варианта. Ну, по 7 битов, как правило, вот. при кодировке ASCII. А для передачи номера этого слова мне может понадобиться десяток битов. Ну вот на этом экономия. Ну а дальше как? Дальше проблема работа с этими словарями. Тут совершенно ясно, что при таком алгоритме построения словаря проблема будет следующим, что у нас рано или поздно будут появляться в этом словаре появляться все более длинные слова, Вот у меня после буквы А появилось слово А, Б. Со временем там появятся вообще все возможные слова А, Б, А, С, А, Д, А, Е, А, З. После этого я букву А могу выбросить. Она уже не нужна. Дело в том, что для нее есть все возможные удлинения. То есть никогда я ее больше не передам. Потому что все возможные варианты удлинения уже есть в словаре. Я буду передавать теперь только эти слова. Поэтому с словарик можно из словаря, можно вот такие варианты исключать. Дальше. При правильном подходе надлежит считать частоту использования слов и отдавать оценки вероятностей этих слов. И использовать какой-нибудь код Хаффмана, строить коды Хаффмана и использовать его для передачи вот этих номеров кодовых слов. Хорошая идея. При этом, поскольку алгоритмы на стороне приема и передачи мы используем одинаковые, то и коды Хаффмана мы будем строить одинаковые всегда. Так что вот есть масса вариантов для улучшений. Ну и, стало быть, вот так выглядит универсальное кодирование лемпелевское. Проблема тут об одном. Она кончается в том, что со временем объем, сговоря, становится слишком большим. его надо чистить, грубо говоря. То есть тут нужна какая-нибудь теористика. То есть вот в том алгоритме, который я изражил, Сгварий непрерывно растет по объему, за исключением вот этих вариантов, когда для какого-то слова есть всевозможные продолжения в Сгварий. Тогда короткое слово можно убрать. То есть он понемногу начинает сокращаться. Но в целом объем растет. И это приводит к проблемам с памятью. Просто этот словарь надо хранить, в нем надо искать, с ним надо возиться, добавлять новую, исключать существующее. Это некие проблемы. Поэтому вот в первых реализациях Лемпелевского алгоритма там было так. Когда число битов, нужных при декодировании номера в словаре, превышает, скажем, 7, то словарь просто чистится тупо и начинается все по-новому. Вот такая вот схема. То есть, чтобы слов в словаре не накапливалось слишком много. Когда, скажем, если я для передачи бита загожил 7 битов, то у меня 128 слов в словаре максимум. Как только там 128 накопилось, я просто ее почищу в ноль и начинаю процедуру сжатия по новой. Конец сеанса, так сказать. Вот это самая простая схема. А вот гениальная идея, совсем уже гениальная, которая была предложена чуть позднее, чем лемпелем, она используется без исключения во всех архиваторах. ЗИП лемпель предложили. Очень простая в реализации. С рука хеот буфером. Значит, идея безумно простая. Берется буфер, очень длинный. Кодируем байты. то есть ASCII код 8-битный, на вход поступает поток байтов. Вот у меня идут байты на вход, вот так вот. Каждый байт, это 8 битов, переносит код ASCII какой-нибудь. Вот, а это буфер довольно здоровый. Буфер берут где-нибудь на 64 кигабайта, вот типичная длина, двух кофеат буфер. Значит, этот буфер содержит то, что я передал раньше. Вначале он пустой. Потом сюда начинают запихиваться вот эти байты. Ну и, стало быть, вот смотрите, какая идея. В этом буфере лежит что-то, что было передано раньше. Вот оно вот сюда запихнуто, и вот я собираюсь передавать нечто новое. Вот как это начинается, мы давайте обсудим потом. А сейчас так, вот мы рассматриваем середину этого процесса. Процесс начался далеко. Вот это те байты, которые мне предстоит передать. А вот это те, которые я уже передал, так или иначе, они были раньше. Они вот просто тупо вот сюда вот так запихиваются, а отсюда выпихиваются и теряются. И тут вот у меня лежат ранее переданные байты. Это, ребята, мой словарь в рамках вот той идеологии, что я обсуждал. Вот набор символных комбинаций вот здесь, всех возможных, это мой типущий словарь. И вот смотрите, чего я делаю. Вот у меня пошли тут какие-то буквы ABCD. Я ищу в этом словаре самое длинное слово, которое совпадает вот с этим словом, что мне предстоит передать. Вот я просматриваю все эти позиции и ищу там ABCD подряд. И так вот пока оно не совпадает. Вот эта процедура называется Longest Match. От буфера она применяется при заданном буфере и при заданном input. Input это вот это, а буфер это вот это. Вукофе от буфера. Значит, процедура делает следующее. Она проверяет все позиции в буфере на максимальное совпадение с тем, что предстоит передать. То есть она ищет тут подстроку, которая совпадает с вот этой строкой. И выдает ответ. Ответ она выдает в виде вот этой координаты position и в виде вот этой координаты len. То есть результатом работы этой процедуры является пара position и len. Положение совпадения и длина совпадения. То есть процедура обнаружена, начиная с этого места номер 35. У меня есть совпадение на 50 символов вот с этим. Вот оно мне вернулось. 50 позиции. 50 символов. Я на сторону приема не буду передавать это совпадение, а передам только вот это. Position Length. Представьте себе, что у них там точно такой же буфер. Вот, с ровно теми же самыми байтами. Ну, то есть, содержимое буферов тождественное. Они, получив от меня вот эту пару Position Length, встанут вот в это место, position, и отсчитают вот эту длину, Лен. И вот этот кусок кода они просто выбросят вот сюда. Вот ровно вот так, копировали. И потом задвинут это в буфер. И я задвину это в буфер. Я буду считать, что я это уже передал, и они будут... Они вот этот кусок кода отдадут получателю и будут считать, что они его отработали. В результате того, когда я это совпадение задвину в буфер, и они выкинутые совпадения задвинут в буфер, у нас содержимое буферов снова одинаково. И я могу искать следующий ленгри-снэч в виде позишн-длина и передавать вам вот эти координаты. Вот эта гениальная идея, она хороша своей простотой. То есть не надо сложных алгоритмов обработки словаря. Словарь составляется автоматически на буфере памяти. Словарь содержит то, что было передано ранее. Но если к практическим реализациям обращаться, то так. Вначале словарь просто пустой. И у вас передаются прямо буквы. Вот как есть. То есть Процедура ленгисмейч возвращает нулевую длину, то есть нет совпадений, и вы передаете просто букву А и заносите ее в буфер. Она там уже появилась. Ну и так далее. Потом, по мере того, как буфер наполняется, начинают возникать совпадения, начинают возникать совпадения, и вы начинаете экономить. Ну то есть первые 64К сжимаются не так уж хорошо, но пока не заполнится буфер. Ну вот, вот это алгоритм Зива Лемпеля. Значит, он начинается с того, что когда вы предъявляете для сжатия файл, проводится, как говорят, его парсинг. Ну, то есть, вот процедура такая. Вначале буфер очищен, и применяются процедуры Lengis Match к инкуту и буферу. Ну, и стало быть, она возвращает позишины лемп, А если лин равен нулю, то она возвращает, то вы передаете вот этот литерал. То есть вы в канал передаете либо букву, либо пару позишн лин. Ну и стало быть весь файл, который вы предъявили для сжатия, обрабатывается таким образом и формируется последовательность литералов. Иногда это нужно. Дело в том, что может оказаться так, то что вы предъявить должны передать сейчас этого в буфере еще никогда не было ну то есть прошла вот эта процедура лен без мяч по всей длине и буквы а тут ни одной не нашла дать буква придется передавать отдельно вот для этого вот здесь предусмотрена такая возможность она называется передача литерал и и позиция длина. Ну и вот, чтобы быть, файл преобразуется в последовательность литералов и позиция длины. Вот этих пар. Потом может идти опять литерал, потом опять позиция длина и так далее. Вот результат парсинга выглядит вот таким образом. Поток. Ну, как правило, если алгоритм работает давно, то, скорее всего, литералов уже нет, а есть только позиция и длина. То есть вы Всякий раз обнаруживаете некое совпадение и его там фиксируете. Ну а дальше так. Получателю передается вот этот результат парсинга. Результаты парсинга для передачи сжимаются по Хаффману. Ну то есть составляется статистика распределения вероятностей позиций, распределения вероятностей длин. Вот тут оказывается, что распределение вероятности позиций, вот если долго поработать, то вот у вас позиция, позишн, она, как известно, от 0 до 2 в 64 гигабайта, ну, то есть 2 в 16. И, в общем, распределение позиций, оно равномерное почти оказывается. Ну, вот если совпадение возникло, то оно как-то с равной вероятностью возникает в разных местах. А вот распределение длин имеет ярко выраженный экстремум. Обычно вот эти совпадения высоко вероятно, а очень длинные совпадения мало вероятно. Стало быть строится код Хаффмана. Строятся эмпирические оценки распределения вероятности по частотам. Строится гистограмма распределения чисел, сколько раз встретилась какая позиция, и на этом основе строится распределение вероятностей, строится распределение вероятностей длин. На этой основе строятся оптимальные коды Хаффмана для кодирования позиций длин, строится оптимальный код Хаффмана для кодирования литерагов. Все это передается на стороны получателя и дальше передаются сами эти позиции Литераго согласно коду Хаффмана. Вот так оно довольно сложно работает. Те архиваторы, с которыми вы сталкиваетесь в компьютерной технике, началось все с JZIP и от него пошли всякие WinRAR, WinZIP и прочее. Это разные варианты вот этого Зива-Лемкеля, алгоритма Зива-Лемкеля. Здесь много вариантов, много эвристики. Не надо думать, что это так-то реально. Ну, то есть, на самом деле, эта программа сложная. Тут есть много подводных камней. Дело в том, что вот эта процедура, Lengis Match, она очень сложная в реализации. Она требует время емкое. Вы должны вот эту строку сравнить со всеми подстроками, коих имеется 64 тысячи штук на каждом этапе. Это очень длинная процедура. То есть вы должны вот эту строку приложить в этом месте, посмотреть, какое совпадение. Потом приложить в другом месте, посмотреть длину совпадения. Вот это оказывается довольно времяемкой процедурой. Поэтому тут публика сильно экономит. Строит деревья поиска, строит хеш-функции. То есть тут есть февристические методы ускорения вот этой процедуры Leng-Bismatch. Дальше довольно быстро было обнаружено, что не надо искать очень длинные совпадения. Вот эти Leng. Их бывает мало. Искать их не стоит. Поэтому имеет смысл ограничить длину самого большого совпадения как-нибудь разумно. Скажем, на 256 символов, если совпало, то и хватит. Потому что более длинные совпадения все равно крайне маловероятны, а для их передачи нужны длинные кодовые слова. Маловероятные символы, длинные кодовые слова. То есть это распределение сильно отлично от равномерного. Вот это распределение почти равномерное. Тут простой код Хаффмана, почти полное дерево. Вот смотрите, когда у вас все вероятности почти одинаковые, у вас код Хаффмана, ну вот он такой вот хороший, он почти вот так обстрежен на дерево, почти равномерные вероятности. А когда есть малые вероятности, то у вас в коде Хаффмана появляются длинные пути, длинные пути, которые кодируют вот эти малые вероятности. становится такой вот нехороший с длинными кодовыми словами и это на самом деле приводит не к улучшению а снижению эффективности то есть если вы хорошо ищите очень длинное совпадение маловероятное то вы на передачу этих совпадений будете затрачивать столько битов что общем экономия не оправдает себя в общем тут довольно много и в ристик все это довольно непросто Но тем не менее, вот такие алгоритмы разработаны, они существуют, и ими все пользуются. Ну и стало быть, вот это вопрос о сжатии данных по Зиму Лемпеля. Мало кто догадывается о том, как работает этот архиватор. В наше время уже почти никто и не знает, как он работает, он уже стал вещью себе. Ну то есть работает и работает, а кого волнует, как он работает? А он работает вот так, в основе там лежит вот этот рухахиат буфер, далее вот эту идею универсального кодирования которые как бы возникли это все 80-е годы прошлого века когда вот этой занимались этими делами а потом имеющим перестали заниматься вот в тем не менее интерес к этому кодированию универсальному он не совсем и сяк работы какие-то ведутся продолжаются но не в плане сжатия компьютерных файлов и Там уже, по-моему, все достигнут потолок. Ну, то есть, как всегда, в области ширпотреба, то, что нужно, именно рассчитано на широкий рынок, оно оптимизируется очень быстро. Находится много людей, которые там хорошие деньги крутятся, поэтому вот так. А вот в каких-то мелких приложениях, где нет широкого рынка, все эти методы пока остаются актуальными. Ну, вот это алгоритм сжатия данных. Ну, стало быть... Ну, пожалуй, и все, что я вам хотел сказать на эту тему. Значит, со следующего раза начнем заниматься передачей данных по каналу. Обратимся, наконец, к настоящей теории вероятной информации. Это теория сжатия данных. Вот она как бы разбиралась и выродилась вот в это. Меня в этой связи, я был непосредственным участником разработки вот этого алгоритма в Музип, я работал в команде, мы там это все тестировали, Меня удивляет, я с вами поделюсь, вот какая вещь. Вот как целые простые знания, они уходят в компьютерный код, и после этого этот код становится вещью себе. Никто в нем уже ничего не понимает, а он работает. Ну то есть на гнузии, на гно вывяжу на эти вот исходник гнузик носи, нами когда-то написан, мы там принимали участие в ее написании. И все его просто тупо встраивают к себе, и он работает. И если начнешь сам писать, вот сам, с нуля, вот напишешь, оно получится, но будет сильно, ну не сильно, а процентов на 20 хуже, чем у них. Потому что у них это все отработано, там много юристических правильных решений принято, которые вот так в бухте-барах ты не примешь правильно. Например, вот эту длину совпадения на каком уровне обрезать? Покороче можно, а можно подлиннее. От этого будет зависеть степень сжатия конечная. И вот это требует экспериментальной проверки, потому что никакими формулами этих явлений не опишешь. Дело в этом, что тут как бы теории-то нет как таковой, тут голая эмпирика начинается. Какие решения рациональны, какие нет. Ну вот так, ладно, все, спасибо. закричать ладно приходите еще на все лекции какой-нибудь физикой.