Будем писать просто log вместо $\log_2$.

Сначала определим понятие информационной энтропии.

Энтропия измеряет количество информации, которую мы можем получить из данного случайного эксперимента.

Определение 1. Энтропия функции на событиях ξ --- это сумма по принимаемым значениям x $-P(\xi = x)\log P(\xi = x)$ (напомним, что символом логарифма без указания основания мы будем обозначать двоичный алгоритм).

Энтропию будем обозначать буквой H, например, энтропия случайной величины ξ обозначается $H(\xi)$.

Можно сказать, что мы рассматриваем случайную величину, которая по событию возвращает минус логарифм его вероятности, и берём её математическое ожидание. Это отражает идею, что мы получаем больше информации, узнав о событии с малой вероятностью. Если мы и так приписывали событию вероятность 1, то узнав, что оно произошло, мы только подтвердили свои прежние знания (на которые опиралась оценка вероятности).

Перечислим простейшие свойства энтропии.

Энтропия константы равна 0.

Энтропия пары независимых функций равна сумме их энтропий. Действительно, вероятность принятия ими данной пары значений равна произведению вероятностей принятия ими этих значений по отдельности; при логарифмировании произведение переходит в сумму, а ожидание суммы равно сумме ожиданий.

Теперь свяжем энтропию случайной величины и количество битов, нужных для её передачи. Пусть мы хотим передавать значение случайной величины так, чтобы просто по потоку данных было сразу понятно, закончена ли передача. Например, мы можем хотеть без дополнительных усилий записывать подряд (без разделителей) коды значений разных случайных величин.

Определение 2. Код называется префиксным, если никакой код значения не является продолжением кода другого значения. Таким образом, префиксный код можно представить в виде двоичного дерева со значениями в листьях.

Докажем, что математическое ожидание длины кода не меньше энтропии.

Обозначим длину i-го кодового слова l_i. Тогда $\sum 2^{-l_i} \leqslant 1$. Это можно доказать индукцией: для дерева кодирования из одной вершины это так (есть одно кодовое слово длины 0), а произвольное дерево кодирования состоит из корня и двух кодовых деревьев --- левого и правого. При этом длина каждого кодового слова в левом и правом поддеревьях увеличивается на 1 по сравнению с их самостоятельным рассмотрением, и интересующая на сумма оказывается равна $2 \times \frac{1}{2} = 1$.

Теперь рассмотрим разность между $\sum -p_i \log p_i$ (энтропией) и $\sum -p_i \log 2^{-l_i}$ (ожидаемой длиной кода, так как $-\log 2^{-l_i} = l_i$). Сразу воспользуемся, что разность логарифмов --- это

логарифм отношения.

$$\sum -p_i \log p_i - \sum -p_i \log 2^{-l_i} = \sum p_i \log \frac{2^{-l_i}}{p_i} \leqslant \log \sum p_i \frac{2^{-l_i}}{p_i} = \log \sum 2^{-l_i} \leqslant \log 1 = 0$$

Здесь первое неравенство использует то, что логарифм --- выпуклая вверх функция, то есть логарифм выпуклой комбинации (суммы с неотрицательными коэффициентами, в сумме дающими 1) нескольких чисел не меньше такой же выпуклой комбинации их логарифмов.

Утверждение доказано.

Теперь построим код, который имеет математическое ожидание длины кодового слова не хуже энтропии, увеличенной на 1.

Возьмём вероятности всех событий и округлим вниз до ближайших отрицательных степеней двойки. Мы занизим вероятности не более, чем в два раза. Теперь у нас имеются степени двойки с суммой не более 1. Будем выбирать минимально возможные коды для значений, идя в порядке от больших вероятностей к маленьким. Заметим, что каждому коду длины l можно сопоставить отрезок длины 2^{-l} на отрезке $[0;1]$ с началом, делящимся на 2^{-l}. Выбирая коды, мы будем просто замащивать отрезок $[0;1]$. Начало каждого из кодовых отрезков делится на его длину, так как все предыдущие отрезки не короче его, а следовательно, их длины делятся на длину текущего отрезка. Обозначая, как и раньше, длину i-го кодового слова за l_i, а вероятность i-го события за p_i, получим математическое ожидание длины кодового слова, равное

$$\sum p_i l_i = \sum -p_i \log 2^{-l_i} \leqslant \sum -p_i \log \frac{p_i}{2} = \sum -p_i (\log p_i - 1) = H + 1$$

На практике часто используют код Хаффмана. Этот код имеет минимальное ожидание длины кодового слова. Построим его, попутно доказывая, что этот код является оптимальным.

Как произойдёт построение: мы строим кодовое дерево. На первом шаге у нас есть отдельные значения и их вероятности, мы их трактуем как деревья из одной вершины. На каждом шаге у нас есть какие-то деревья кодирования и суммарные вероятности их листьев; мы находим две минимальные вероятности и присоединяем соответствующие деревья к общему корню, получая одно дерево.

Докажем следующее утверждение: пусть есть набор деревьев кодирования для непересекающихся наборов значений случайной величины, в сумме покрывающих все принимаемые значения. Тогда среди всех деревьев кодирования, дающих минимальное математическое ожидание длины кодового слова для выпадающего события и включающих все данные , найдётся такое, что два дерева с минимальными суммами вероятностей листьев из имеющихся входят в него и их корни являются дочерними вершинами для одной и той же вершины в кодом дереве для всех значений сразу.

Доказательство 1. Рассмотрим оптимальное среди включающих данные поддеревья дерево кодирования. Рассмотрим математическое ожидание длины кодового слова. Заметим,

что мы можем влиять только на длины кодовых слов для корней данных поддеревьев; части кодов, заданные поддеревьями, дают всегда один и тот же вклад. Поэтому достаточно рассмотреть случай, когда у нас есть отдельные значения.

Рассмотрим два самых длинных кодовых слова. Можно считать, что они различаются в одном последнем символе. Ожидание длины кода не увеличится, если сопоставить их двум самым редким событиям (мы сопоставим менее вероятному значению более длинное слово) Это доказывает наш шаг.

Так как ожидание длины оптимального двоичного кода для события оказалось близко к энтропии, энтропию можно считать средним количеством битов информации, получаемой от эксперимента.

Количество информации можно использовать и для ещё одного определения невероятного.

Пусть мы кидаем симметричную монетку миллион раз, и все разы выпал орёл. Есть ли в этом что-то плохое? Конечно, после такого события невероятно, чтобы монетка была и правда симметричной --- неравенство Чебышева говорит, что такое большое отклонение количества орлов от среднего маловероятно. Но пусть монетка падает орлом только во все чётные разы, а во все нечётные --- решкой. Доля орлов такая, как должна быть, но всё равно хочется сказать, что что-то тут не так.

Применим количество информации, чтобы попытаться сформулировать, что именно здесь невероятно.

Можно определить не только количество информации в случайном эксперименте, но и попытаться определить количество информации в отдельно взятом объекте. Разумеется, количество информации, необходимой для описания объекта зависит от языка описания. Но с этим можно бороться следующим способом.

Определим такое понятие как *априорную вероятность*.

Мы будем рассматривать описания объектов, которые сами уже заданы последовательностями нулей и единиц (как в компьютере).

Определение 3. Пусть у нас есть программа с доступом к источнику независимых случайных битов. Она может никогда не остановиться или напечатать какое-то конечное слово. Рассмотрим распределение вероятностей для слов, которые она может напечатать.

Можно рассмотреть и программы, которые печатают согласованные в совокупности утверждения вида ``слово x имеет вероятность не меньше p'' и рассматривать предельные вероятности слов (остаток от единицы даст вероятность зависания программы).

Назовём получаемые таким образом распределения вероятностей на словах *перечислимыми снизу*.

Это одно и то же, так как можно просто моделировать работу программы на всех возможных последовательностях (разветвляясь при считывании случайного бита) и выписывать

оценки на вероятность --- или же раскладывать оценки на вероятность по ветвям дерева декодирования.

Например, если взять код Хаффмена для произвольного набора двоичных слов и декодировать случайную последовательность, то мы получим перечислимое снизу распределение вероятностей.

Разумеется, при этом может оказаться, что не все слова имеют положительную вероятность. Кроме того, разные методы будут приписывать большие вероятности разным наборам слов.

Теорема 1. Среди перечислимых снизу распределений вероятности есть оптимальное.

Точнее говоря, есть такое распределение, что для любого другого перечислимого снизу распределения вероятностей можно поделить все вероятности слов на одну и ту же константу и все они окажутся меньше оптимальных вероятностей.

Разумеется, константа может зависить от конкретного распределения.

Доказательство существования оптимального декодера описаний тривиально: запишем каждую программу префиксным кодом для используемого алфавита с дополнительным символом конца программы, после чего разрешим её обращаться к тому, что останется, как к данным. Скажем, что при попытке обратиться дальше конца имеющихся данных (как и при невозможности считать программу) результат не определён. Для любого способа описаний можно просто написать его программу в начале описания (это константная длина) и получать не слишком сильно худший результат.

Сложностью префиксно корректного описания некоторого слова X будем называть длину минимального описания, которое наш оптимальный декодер соглашается декодировать и выдаёт на нём X.

Объясним теперь, почему нам кажется невероятным получение последовательности исходов с очень маленькой сложностью описаний. Для последовательности с маленькой сложностью можно придумать относительно простой механизм, обеспечивающий её получение вместо случайной последовательности. Если мы считаем, что вероятность того, что нас исходно обманули про случайность процесса очень мала, скажем $\frac{1}{1000}$, и условная вероятность конкретного простого вида обмана тоже мала, например $\frac{1}{10000}$, то мы получаем событие с вероятностью 10^{-7}, при наступлении которого монетка падает поочерёдно то орлом, то решкой из-за некоторой неслучайности.

Но по формуле Байеса уже для ста бросаний нам надо сравнивать произведения условных вероятностей получения нашей последовательности (при предположении обмана и при предположении случайности) и вероятностей собственно обмана и случайности. Разумеется, $10^{-7} \times 1 > 1 \times 2^{-100}$. Поэтому нам кажется, что условная вероятность правды при условии, что монетка падает поочерёдно орлом и решкой, мала.

Докажем, что энтропия пары величин не больше суммы энтропий с использованием

префиксных кодов.

Энтропия N-кратного повторения пары величин не может превышать N-кратную сумму энтропий просто в силу наличия префиксного кода для N пар, который сначала описывает все первые компоненты, а потом все вторые компоненты. Разделять их нам не надо, так как это префиксные коды; получившийся код, очевидно, тоже префиксный; ошибка не превышает одного бита и делится на N.

Ожидание префиксной сложности набора из n независимых повторов одного и того же испытания равно $n \times H(\xi) + C$. Действительно, энтропийный префиксный код является одним из способов вычислимо расшифровываемого префиксно корректного описания. с другой стороны, любой способ префиксно корректного описания позволяет построить префиксный код просто выбором первого в словарном порядке из кратчайших описаний для каждого слова. Ранее мы рассматривали вероятностные пространства с заданным набором непересекающихся элементарных событий. Теперь мы рассмотрим свойства бесконечных последовательностей нулей и единиц; мы будем хотеть, чтобы про каждый начальный отрезок можно было сказать, что он порождён независимыми бросаниями симметричной монеты.

Естественными событиями для такого описания будут события вида ``последовательность начинается с $x_1 \ldots x_n$'' с вероятностью по 2^{-n}. К сожалению, просто назвать все такие события элементарными не получится, так как они пересекаются. Назовём их пока что базовыми. С другой стороны, если мы получаем такие события по одному, несложно возвращать в ответ на каждое событие некоторый набор событий так, что никакие два из возвращённых событий не будут пересекаться, а объединение полученных и возвращённых всегда будет такое же, как у полученных событий.

Разумеется, из конечного набора случайных событий можно было бы просто выкинуть все события, вложенные в большие, но для работы с бесконечными (но просто описываемыми) последовательностями событий удобнее исправлять пересечения в порядке поступления. Кроме того, заметим, что любая замена базового события на некоторое разбиение этого базового события на непересекающиеся вложенные в него базовые события даст ту же вероятность.

Вероятностью множества, заданного как объединение непересекающихся базовых множеств назовём сумму ряда вероятностей этих базовых множеств.

Бесконечную последовательность X назовём неслучайной, если существует программа, которая по любому ε печатает (возможно, бесконечное) количество базовых событий, сумма вероятностей которых не превосходит ε, и объединение которых содержит X. Остальные последовательности называются случайными.

Другое определение: последовательность называется случайной, если существует такое программно порождаемое перечисление событий суммарной вероятности 1, что данная последовательность лежит в бесконечном их количестве.

Эти два определения эквивалентны. Действительно, если мы умеем породить список базовых событий с суммой вероятностей C, который покрывает нашу последовательность бесконечно много раз, то наша последовательность в какой-то момент будет покрыта хотя бы C/ε раз --- но суммарная мера C не позволяет базовым событиям больше ε быть покрытым так много раз. В обратную сторону: пусть мы для каждого ε умеем выписывать последовательность базовых событий такой суммарной длины, хоть раз покрывающих нашу последовательность. Тогда для любого δ мы можем параллельно запустить процесс выписывания таких последовательностей базовых событий для $\varepsilon = \frac{\delta}{2}, \frac{\delta}{4}, \ldots$.

На самом деле, как и в случае с префиксной сложностью, существует одна конкретная программа, порождающая бесконечное количество событий суммарной вероятностью не больше 1 и покрывающая каждую неслучайную последовательность бесконечно много раз. Для этого достаточно перебрать все программы, давая каждой следующей вдвое меньший параметр и не позволяя выводить базовые множества сверх вероятности, заданной этим параметром. Среди программ встретятся все программы, умеющие бесконечно много раз покрыть неслучайную последовательность бесконечно много раз при сколь угодно малой суммарной вероятности.

Можно показать и связь между сложностью описания начал последовательности и случайностью последовательности. Правда, для этого нужно другое определение сложности --- (обычная) колмогоровская сложность. В отличие от префиксной сложности, при определении колмогоровской сложности программа получает на вход просто слово известной длины; на расшифровку слова и его продолжений никаких правил не накладывается.

Сравним это определение с префиксной сложностью. С одной стороны, в отличие от префиксной сложности, при описании пары мы не можем просто приписывать описания, так как границу между ними, нельзя найти. Поэтому для описания пары слов придётся потратить ещё логарифмическое количество битов на запись длины описания первого из них (причём так, чтобы граница записи длины определялась однозначно). С другой стороны, при описании одиночного слова у нас есть естественная оценка сложности ``длина слова плюс константа'': так как программа знает длину слова, то она может просто переписать его в ответ.

Заметим, что так как описаний короче, чем $n-1$ не более 2^n-1 штук, большинство слов длины n имеют кратчайшие описания не короче $n-2$.

Мы видим, что колмогоровская сложность большинства слов отличается от n не больше, чем на некоторую константу. Для неслучайных слов верно обратное.

Действительно, так как у нас есть программа, перечисляющая базовые события с суммарной вероятностью 1 и покрывающая каждую последовательность бесконечно много раз, начал длины 2^n, покрытых 2^k всего лишь $2^{2^n-2^k}$ штук; каждое описывается указанием n, k и номера среди них, что требует $C + 2\log n + 2\log k + 2^n - 2^k$ битов и заметно меньше 2^n битов. С другой стороны, для любой неслучайной последовательности и любого k достаточно

большое n найдётся.

Так же можно показать наличие коротких описаний у случайных последовательностей с точки зрения префиксной сложности. Но для того, чтобы дать определение случайности в точности совпадающее с ранее данным, пришлось бы определять монотонную сложность, которая устроена ещё чуть сложнее префиксной. Но на практике не так важно, как определять случайность: если мы видим, что миллион бросаний монетки сжался архиватором на 10%, это столь маловероятно для настоящих случайных последовательностей, что в видимой части вселенной физика не сможет описать нужное количество экспериментов (планковский объём и планковское время очень малы, а возраст и размер вселенной велики --- но не настолько).

Докажем теперь, что способ определения префиксной сложности не важен (напомним, что один способ запрещает программе распаковки давать ответ на продолжениях корректных описаний, а второй способ требует от неё давать на продолжениях корректного описания тот же ответ).

Для этого определим ещё одно понятие --- *априорную вероятность*.

Пусть у нас есть программа с доступом к источнику независимых случайных битов. Она может никогда не остановиться или напечатать какое-то конечное слово. Рассмотрим распределение вероятностей для слов, которые она может напечатать.

Можно рассмотреть и программы, которые печатают согласованные в совокупности утверждения вида ``слово x имеет вероятность не меньше p'' и рассматривать предельные вероятности слов (остаток от единицы вычислимого способа префиксно корректного декодирования оптимальный способ позволяет описание любого слова с не более чем константной разностью длин даст вероятность зависания программы).

Это одно и то же, так как можно просто моделировать работу программы на всех возможных последовательностях (разветвляясь при считывании случайного бита) и выписывать оценки на вероятность --- или же раскладывать оценки на вероятность по ветвям дерева декодирования.

Теорема: обе префиксных сложности равны логарифму априорной вероятности.

Беспрефиксную функцию (продолжение корректного описания не может быть корректным) можно просто применять ко всем началам аргумента для получения префиксно корректной (продолжение описания описывает то же самое).

По префиксно корректной функции можно построить машину, порождающую по случайной последовательности битов конечное слово, пробуя расшифровать все начала параллельно.

Пусть теперь мы узнаём оценки снизу на вероятность слова, и должны выделить беспрефиксные коды. Мы имеем право уменьшить все оценки в константу раз. Округлим запросы вниз до степеней двойки. Поделим их пополам, чтобы мы могли с каждым ε выделять и $\frac{\varepsilon}{2}, \frac{\varepsilon}{4}, \ldots$. Представим имеющееся у нас место на двоичном дереве в виде блоков попарно

разного размера. Каждое слово будем кодировать, откусывая кусок от самого маленького подходящего блока. Куски, на которые развалится испорченный кусок имеют размеры от исходного до размера запроса, которые не встречались до этого. С другой стороны, каждый запрос выполним, так как сумма всех отрицательных степеней двойки, меньших данной, всё равно меньше её.