Министерство образования и науки РФ

ФГБОУ ВПО «Тамбовский государственный технический университет»

Кафедра « »

Доклад

по дисциплине «Теория информации»

Алгоритм Шеннона — Фано

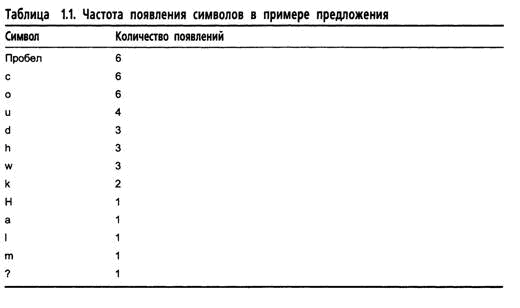
Выполнил: студент гр. −31 Д. Ю.

Проверил: Н. В.

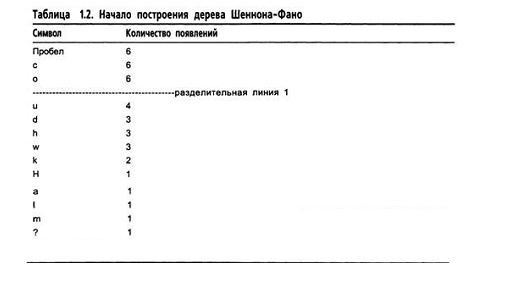
Тамбов,

Кодирование Шеннона-Фано, это название от имени двух исследователей, которые одновременно и независимо друг от друга разработали данный алгоритм: Клода Шеннона (Claude Shannon) и Р. М. Фано (R. М. Fano). Алгоритм анализирует входные данные и на их основе строит бинарное дерево минимального кодирования. Используя это дерево, затем можно выполнить повторное считывание входных данных и закодировать их.

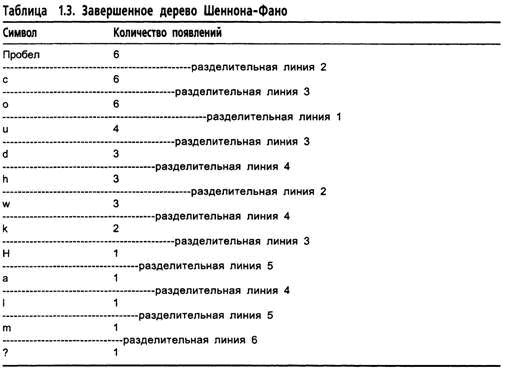
Чтобы проиллюстрировать работу алгоритма, выполним сжатие предложения "How much wood could a woodchuck chuck?" ("Сколько дров мог бы заготовить дровосек?") Прежде всего, предложение необходимо проанализировать. Просмотрим данные и вычислим, сколько раз в предложении встречается каждый символ. Занесем результаты в таблицу (см. таблицу 1.1).



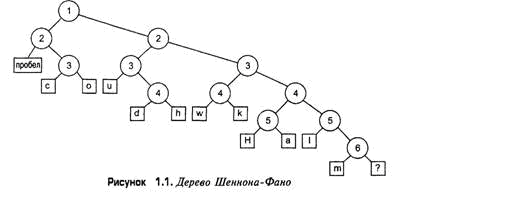
Теперь разделим таблицу на две части, чтобы общее число появлений символов в верхней половине таблицы приблизительно равнялось общему числу появлений в нижней половине. Предложение содержит 38 символов, следовательно, верхняя половина таблицы должна отражать приблизительно 19 появлений символов. Это просто: достаточно поместить разделительную линию между строкой o и строкой u. В результате этого верхняя половина таблицы будет отражать появление 18 символов, а нижняя - 20. Таким образом, мы получаем таблицу 1.2.



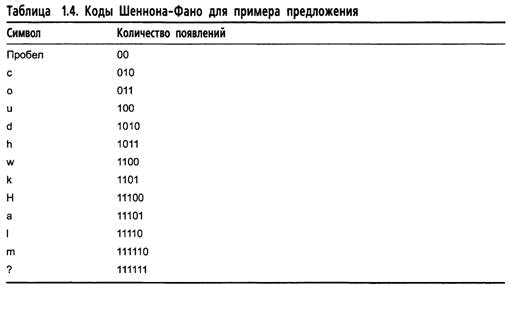
Теперь проделаем то же с каждой из частей таблицы: вставим линию между строками так, чтобы разделить каждую из частей. Продолжим этот процесс, пока все буквы не окажутся разделенными одна от другой. Результирующее дерево Шеннона-Фано представлено в таблице 1.3.



Разделительные линии различны по длине, разделительная линия 1 самая длинная, разделительная линия 2 немного короче и так далее, вплоть до самой короткой разделительной линии 6. Этот подход обусловлен тем, что разделительные линии образуют повернутое на 90° бинарное дерево (чтобы убедиться в этом, поверните таблицу на 90° против часовой стрелки). Разделительная линия 1 является корневым узлом дерева, разделительные линии 2 - двумя его дочерними узлами и т.д. Символы образуют листья дерева. Результирующее дерево в обычной ориентации показано на рис.1.1



Все это очень хорошо, но как оно помогает решить задачу кодирования каждого символа и выполнения сжатия? Что ж, чтобы добраться до символа пробела, мы начинаем с коневого узла, перемещаемся влево, а затем снова влево. Чтобы добраться до символа c, мы смещаемся влево из корневого узла, затем вправо, а затем влево. Для перемещения к символу o потребуется сместиться влево, а затем два раза вправо. Если принять, что перемещение влево эквивалентно нулевому биту, а вправо - единичному, можно создать таблицу кодирования, приведенную в таблице 11.4.



Cодержит всего 131 бит. Если мы предполагаем, что исходная фраза закодирована кодом ASCII, т.е. один байт на символ, то оригинальная фраза заняла бы 256 бит, т.е. мы получаем коэффициент сжатия 54%. (131/256 \*100% = 54%)

**Для декодирования сжатого потока битов** строим то же дерево, которое было построено на этапе сжатия. Мы начинаем с корневого узла и выбираем из сжатого потока битов по одному биту. Если бит является нулевым, мы перемещаемся влево, если единичным - вправо. Мы продолжаем этот процесс до тех пор, пока не достигнем листа, т.е. символа, после чего выводим символ в поток восстановленных данных. Затем мы снова начинаем процесс с корневого узла дерева с целью извлечения следующего бита. Обратите внимание, что поскольку символы расположены только в листьях дерева, код одного символа не образует первую часть кода другого символа. Благодаря этому, неправильное декодирование сжатых данных невозможно. (Бинарное дерево, в котором данные размещены только в листьях, называется префиксным деревом (prefix tree).)

Однако при этом возникает небольшая проблема: как распознать конец потока битов? В конце концов, внутри класса мы будем объединять восемь битов в байт, после чего выполнять запись байта. Маловероятно, чтобы поток битов содержал количество битов строго кратное 8. Существует два возможных решения этой дилеммы. Первое - закодировать специальный символ, отсутствующий в исходных данных, и назвать его символом конца файла. Второе - записать в сжатый поток длину несжатых данных перед тем, как приступить к сжатию самих данных. Первое решение вынуждает нас найти отсутствующий в исходных данных символ и использовать его (это предполагает передачу этого символа в составе сжатых данных программе восстановления, чтобы она знала, что следует искать). Или же можно было бы принять, что хотя символы данных имеют размер, равный размеру одного байта, символ конца файла имеет длину, равную длину слова (и заданное значение, например 256). Однако мы будем использовать второе решение. Перед сжатыми данными мы будем сохранять длину несжатых данных, и таким образом во время восстановления будет в точности известно, сколько символов нужно декодировать.

Еще одна проблема применения кодирования Шеннона-Фано, на которую до сих пор мы не обращали внимания, связана с деревом. Обычно сжатие данных выполняется в целях экономии объема памяти или уменьшения времени передачи данных. Как правило, сжатие и восстановление данных разнесено во времени и пространстве. Однако алгоритм восстановления требует использования дерева. В противном случае невозможно декодировать закодированный поток. Нам доступны две возможности. Первая - сделать дерево статическим. Иначе говоря, одно и то же дерево будет использоваться для сжатия всех данных. Для некоторых данных результирующее сжатие будет достаточно оптимальным, для других весьма далеким от приемлемого. Вторая возможность состоит в том, чтобы тем или иным способом присоединить само дерево к сжатому потоку битов. Конечно, присоединение дерева к сжатым данным ведет к снижению коэффициента сжатия, но с этим ничего нельзя поделать.