БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра РЭС

Реферат на тему:

**«Арифметическое кодирование. Кодирование длин повторений»**

МИНСК, 2009

**Арифметическое кодирование**

Пpи аpифметическом кодиpовании, в отличие от рассмотренных нами методов, когда кодируемый символ (или группа символов) заменяется соответствующим им кодом, *результат кодирования всего сообщения пpедставляется одним или парой вещественных чисел в интеpвале от 0 до 1*. По меpе кодиpования исходного текста отобpажающий его интеpвал уменьшается, а количество десятичных (или двоичных) разрядов, служащих для его пpедставления, возpастает. Очеpедные символы входного текста сокpащают величину интеpвала исходя из значений их веpоятностей, определяемых моделью. *Более веpоятные символы делают это в меньшей степени, чем менее веpоятные, и, следовательно, добавляют меньше разрядов к pезультату.*

Поясним идею арифметического кодирования на простейшем примере. Пусть нам нужно закодировать следующую текстовую строку: ***РАДИОВИЗИР.***

Пеpед началом pаботы кодера соответствующий кодируемому тексту исходный интеpвал составляет ***[0; 1).***

Алфавит кодируемого сообщения содержит следующие символы (буквы): { ***Р, А, Д, И, О, В, З*** }.

Определим количество (встречаемость, вероятность) каждого из символов алфавита в сообщении и назначим каждому из них интервал, пропорциональный его вероятности. С учетом того, что в кодируемом слове всего 10 букв, получим табл. 1

Таблица 1

|  Символ | Веpоятность | Интеpвал |
| --- | --- | --- |
| ***А*** | 0.1 | 0 – 0.1 |
| ***Д*** | 0.1 | 0.1 – 0.2 |
| ***В*** | 0.1 | 0.2 – 0.3 |
| ***И*** | 0.3 | 0.3 – 0.6 |
| ***З*** | 0.1 | 0.6 – 0.7 |
| ***О*** | 0.1 | 0.7 – 0.8 |
| ***Р*** | 0.2  | 0.8 – 1 |

Располагать символы в таблице можно в любом порядке: по мере их появления в тексте, в алфавитном или по возрастанию вероятностей – это совершенно не принципиально. Результат кодирования при этом будет разным, но эффект – одинаковым.

###

### **Процедура кодирования**

Итак, перед началом кодирования исходный интервал составляет [0 – 1).

После пpосмотpа пеpвого символа сообщения ***Р*** кодер сужает исходный интеpвал до нового - [0.8; 1), котоpый модель выделяет этому символу. Таким образом, после кодирования первой буквы результат кодирования будет находиться в интервале чисел [ 0.8 - 1).

Следующим символом сообщения, поступающим в кодер, будет буква ***А***. Если бы эта буква была первой в кодируемом сообщении, ей был бы отведен интервал [ 0 - 0.1 ), но она следует за ***Р*** и поэтому кодируется новым *подынтервалом внутри уже выделенного для первой буквы*, сужая его до величины [ 0.80 - 0.82 ). Другими словами, интервал [ 0 - 0.1 ), выделенный для буквы ***А***, располагается теперь внутри интервала, занимаемого предыдущим символом (*начало и конец нового интервала определяются путем прибавления к началу предыдущего интервала произведения ширины предыдущего интервала на значения интервала, отведенные текущему символу*). В pезультате получим новый pабочий интеpвал [0.80 - 0.82), т.к. пpедыдущий интеpвал имел шиpину в 0.2 единицы и одна десятая от него есть 0.02.

Следующему символу ***Д*** соответствует выделенный интервал [0.1 - 0.2), что пpименительно к уже имеющемуся рабочему интервалу [0.80 - 0.82) сужает его до величины [0.802 - 0.804).

Следующим символом, поступающим на вход кодера, будет буква ***И*** с выделенным для нее фиксированным интервалом [ 0,3 – 0,6). Применительно к уже имеющемуся рабочему интервалу получим [ 0,8026 - 0,8032 ).

Пpодолжая в том же духе, имеем:

вначале [0.0 - 1.0)

после пpосмотpа ***Р*** [0.8 - 1.0)

 ***А***  [0.80 - 0.82)

 ***Д*** [0.802 - 0.804)

 ***И*** [0.8026 - 0.8032)

 ***О*** [0.80302 - 0.80308)

 ***В*** [0.803032 - 0.803038)

 ***И*** [0.8030338 - 0.8030356)

 ***З*** [0.80303488 - 0.80303506)

 ***И***  [0.803034934 - 0.803034988)

 ***Р***  [0.8030349772 - 0.8030349880)

Результат кодирования: интервал [0,8030349772 – 0,8030349880]. На самом деле, для однозначного декодирования теперь достаточно знать только одну границу интервала – нижнюю или верхнюю, то есть результатом кодирования может служить начало конечного интервала - 0,8030349772. Если быть еще более точным, то любое число, заключенное *внутри* этого интервала, однозначно декодируется в исходное сообщение. К примеру, это можно проверить с числом 0,80303498, удовлетворяющим этим условиям. При этом последнее число имеет меньшее число десятичных разрядов, чем числа, соответствующие нижней и верхней границам интервала, и, следовательно может быть представлено меньшим числом двоичных разрядов.

Нетрудно убедиться в том, что, чем шире конечный интервал, тем меньшим числом десятичных (и, следовательно, двоичных) разрядов он может быть представлен. Ширина же интервала зависит от распределения вероятностей кодируемых символов – более вероятные символы сужают интервал в меньшей степени и , следовательно, добавляют к результату кодирования меньше бит. Покажем это на простом примере.

Допустим, нам нужно закодировать следующую строку символов: ***A A A A A A A A A #***, где вероятность буквы ***А*** составляет 0,9. Процедура кодирования этой строки и получаемый результат будут выглядеть в этом случае следующим образом:

Входной символ Нижняя граница Верхняя граница

 0.0 1.0

 ***A*** 0.0 0.9

 ***A*** 0.0 0.81

 ***A*** 0.0 0.729

 ***A*** 0.0 0.6561

 ***A*** 0.0 0.59049

 ***A*** 0.0 0.531441

 ***A*** 0.0 0.4782969

 ***А*** 0.0 0.43046721

 ***А*** 0.0 0.387420489

 ***#*** 0.3486784401 0.387420489

Результатом кодирования теперь может быть, к примеру, число 0.35 , целиком попадающее внутрь конечного интервала 0.3486784401 – 0.387420489. Для двоичного представления этого числа нам понадобится 7 бит (два десятичных разряда соответствуют примерно семи двоичным ), тогда как для двоичного представления результатов кодирования из предыдущего примера – 0,80303498 – нужно 27 бит !!!

### **Декодирование**

Пpедположим, что все что декодер знает о тексте, – это конечный интеpвал [0,8030349772 - 0,8030349880]. Декодеру, как и кодеру, известна также таблица распределения выделенных алфавиту интервалов. Он сpазу же понимает, что пеpвый закодиpованный символ есть ***Р***, так как результат кодирования целиком лежит в интеpвале [0.8 - 1), выделенном моделью символу ***Р*** согласно таблице .

Тепеpь повтоpим действия кодера:

вначале [0.0 - 1.0);

после пpосмотpа [0.8 - 1.0).

Исключим из результата кодирования влияние теперь уже известного первого символа ***Р***, для этого вычтем из результата кодирования нижнюю границу диапазона, отведенного для ***Р,*** – 0,8030349772 – 0.8 = =0,0030349772 – и разделим полученный результат на ширину интервала, отведенного для ***Р,*** – 0.2. В результате получим 0,0030349772 / 0,2 = =0,015174886. Это число целиком вмещается в интервал, отведенный для буквы ***А,*** – [0 – 0,1) , следовательно, вторым символом декодированной последовательности будет ***А***.

Поскольку теперь мы знаем уже две декодированные буквы - ***РА***, исключим из итогового интервала влияние буквы ***А***. Для этого вычтем из остатка 0,015174886 нижнюю границу для буквы ***А*** 0,015174886 – 0.0 = =0,015174886 и разделим результат на ширину интервала, отведенного для буквы ***А***, то есть на 0,1. В результате получим 0,015174886/0,1=0,15174886. Результат лежит в диапазоне, отведенном для буквы ***Д***, следовательно, очередная буква будет ***Д***.

Исключим из результата кодирования влияние буквы ***Д***. Получим (0,15174886 – 0,1)/0,1 = 0,5174886. Результат попадает в интервал, отведенный для буквы ***И***, следовательно, очередной декодированный символ – ***И***, и так далее, пока не декодируем все символы:

Декодируемое Символ Границы Ширина

число на выходе нижняя верхняя интервала

0,8030349772 ***Р*** 0.8 1.0 0.2

0,015174886 ***А*** 0.0 0.1 0.1

0,15174886 ***Д***  0.1 0.2 0.1

0,5174886 ***И*** 0.3 0.6 0.3

0,724962 ***О*** 0,7 0,8 0,1

0,24962 ***В*** 0,2 0,2 0,1

0,4962 ***И*** 0,3 0,6 0,3

0,654 ***З*** 0,6 0,7 0,1

0,54 ***И*** 0,3 0,6 0,3

0,8 ***Р*** 0,6 0,8 0,2

0.0 Конец декодирования

Это основная идея арифметического кодирования, его практическая реализация несколько сложнее. Некоторые проблемы можно заметить непосредственно из приведенного примера.

Первая состоит в том, что декодеру нужно обязательно каким-либо образом дать знать об окончании процедуры декодирования, поскольку остаток 0,0 может означать букву а или последовательность аа, ааа, аааа и т.д. Решить эту проблему можно двумя способами.

Во-первых, кроме кода данных можно сохранять число, представляющее собой размер кодируемого массива. Процесс декодирования в этом случае будет прекращен, как только массив на выходе декодера станет такого размера.

Другой способ – включить в модель источника специальный символ конца блока, например #, и прекращать декодирование, когда этот символ появится на выходе декодера.

Вторая проблема вытекает из самой идеи арифметического кодирования и состоит в том, что окончательный результат кодирования – конечный интервал – станет известен только по окончании процесса кодирования. Следовательно, нельзя начать передачу закодированного сообщения, пока не получена последняя буква исходного сообщения и не определен окончательный интервал? На самом деле в этой задержке нет необходимости. По мере того, как интервал, представляющий результат кодирования, сужается, старшие десятичные знаки его (или старшие биты, если число записывается в двоичной форме) перестают изменяться (посмотрите на приведенный пример кодирования). Следовательно, эти разряды (или биты) уже могут передаваться. Таким образом, передача закодированной последовательности осуществляется, хотя и с некоторой задержкой, но последняя незначительна и не зависит от размера кодируемого сообщения.

И третья проблема – это вопрос точности представления. Из приведенного примера видно, что точность представления интервала (число десятичных разрядов, требуемое для его представления) неограниченно возрастает при увеличении длины кодируемого сообщения. Эта проблема обычно решается использованием арифметики с конечной разрядностью и отслеживанием переполнения разрядности регистров.

## Кодирование длин повторений

Кодирование длин участков (или повторений) может быть достаточно эффективным при сжатии двоичных данных, например, черно-белых факсимильных изображений, черно-белых изображений, содержащих множество прямых линий и однородных участков, схем и т.п. Кодирование длин повторений является одним из элементов известного алгоритма сжатия изображений JPEG.

Идея сжатия данных на основе кодирования длин повторений состоит в том, что вместо кодирования собственно данных подвергаются кодированию числа, соответствующие длинам участков, на которых данные сохраняют неизменное значение.

Предположим, что нужно закодировать двоичное (двухцветное) изображение размером 8 х 8 элементов, приведенное на рис. 1.

Рис. 1

Просканируем это изображение по строкам (двум цветам на изображении будут соответствовать 0 и 1), в результате получим двоичный вектор данных

***X=*** (0111000011110000000100000001000000010000000111100011110111101111)

длиной 64 бита (скорость исходного кода составляет 1 бит на элемент изображения).

Выделим в векторе ***X*** участки, на которых данные сохраняют неизменное значение, и определим их длины. Результирующая последовательность длин участков - положительных целых чисел, соответствующих исходному вектору данных ***X, -*** будет иметь вид ***r*** = (1, 3, 4, 4, 7, 1, 7, 1, 7, 1, 7, 4, 3, 4, 1, 4, 1, 4).

Теперь эту последовательность, в которой заметна определенная повторяемость (единиц и четверок гораздо больше, чем других символов), можно закодировать каким-либо статистическим кодом, например, кодом Хаффмена без памяти, имеющим таблицу кодирования (табл. 2)

Таблица 2

|  |
| --- |
| Кодер |
| Длина участка | Кодовое слово |
| 4 | 0 |
| 1 | 10 |
| 7 | 110 |
| 3 | 111 |

Для того, чтобы указать, что кодируемая последовательность начинается с нуля, добавим в начале кодового слова префиксный символ 0. В результате получим кодовое слово ***B (r)*** = (0100011010110101101011001110100100) длиной в 34 бита, то есть результирующая скорость кода ***R*** составит 34/64, или немногим более 0,5 бита на элемент изображения. При сжатии изображений большего размера и содержащих множество повторяющихся элементов эффективность сжатия может оказаться существенно более высокой.

Ниже приведен другой пример использования кодирования длин повторений, когда в цифровых данных встречаются участки с большим количеством нулевых значений. Всякий раз, когда в потоке данных встречается “*ноль*”, он кодируется двумя числами. Первое - 0, являющееся флагом начала кодирования длины потока нулей, и второе – количество нулей в очередной группе. Если среднее число нулей в группе больше двух, будет иметь место сжатие. С другой стороны, большое число отдельных нулей может привести даже к увеличению размера кодируемого файла:

7 8 54 0 0 0 97 5 16 0 45 23 0 0 0 0 0 3 67 0 0 8 ...

7 8 54 0 3 97 5 16 0 1 45 23 0 5 3 67 0 2 8 ...

Еще одним простым и широко используемым для сжатия изображений и звуковых сигналов методом неразрушающего кодирования является метод дифференциального кодирования.