УДК 621.391.15

© 1999 г. **Б. Я. Рябко, А. Н. Фионов**

ЭФФЕКТИВНЫЙ МЕТОД АДАПТИВНОГО АРИФМЕТИЧЕСКОГО КОДИРОВАНИЯ ДЛЯ ИСТОЧНИКОВ С БОЛЬШИМИ АЛФАВИТАМИ[1]

Рассматривается задача построения адаптивного арифметического кода для случая большого алфавита источника. Предлагается метод с временем кодирования и декодирования на порядок меньшим, чем у известных методов. Предлагается также реализация метода с использованием структуры данных, называемой "мнимое скользящее окно", что позволяет существенно уменьшить объем памяти кодера и декодера.

§ 1. Введение

Арифметическое кодирование известно сегодня как один из наиболее эффективных методов сжатия данных, который применим для большого класса источников иформации. Основная идея арифметического кодирования была сформулирована Элайесом еще в начале 60-х годов (см. [1]). Однако, первый шаг в направлении практической реализации этой идеи был сделан независимо Риссаненом [2] и Паско [3] в 1976 г. Вскоре после этого в работах [4–6] была изложена современная концепция арифметического кодирования, которая используется в настоящей работе. Метод получил дальнейшее развитие в ряде работ (см., например, [7]).

Преимущество арифметического кода по отношению к другим методам заключается в том, что он позволяет достичь произвольно низкой избыточности на символ источника за счет меньших затрат памяти и времени, чем любые другие известные методы (мы используем понятие избыточности, определяемой как разность между средней длиной кодового слова и энтропией источника). Кроме этого, арифметический код может легко применяться в случае источника с неизвестной статистикой, когда он объединяется с методами адаптивной оценки статистики источника в составе схемы адаптивного кодирования (см., например, [8–10]).

Основная проблема при объединении арифметического кодирования с методами адаптивного оценивания статистики (или, проще, адаптивными моделями) заключается в следующем. Все известные адаптивные модели обеспечивают получение некоторых оценок вероятностей $p(a_i)$ появления каждой буквы a_i из алфавита источника

[1]Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (номер проекта 99-01-00586).

A. Арифметическое кодирование, в свою очередь, требует задания кумулятивных вероятностей $Q(a_i) = \sum_{j<i} p(a_j)$ (термином "кумулятивная вероятность" в литературе по кодированию источников принято называть значение (кумулятивной) функции распределения вероятностей, вычисленное для некоторой буквы из алфавита источника). Так как оценки вероятностей изменяются после кодирования каждого нового символа, кумулятивные вероятности необходимо постоянно пересчитывать. Такой пересчет требует порядка $|A|/2$ операций на каждый символ источника, где $|A|$ обозначает размер алфавита, что, очевидно, оказывает влияние на скорость кодирования и декодирования. Снижение скорости становится все заметнее с ростом размера алфавита источника. В последние годы задача кодирования источников с большими алфавитами становится все более актуальной (см., например, [11]). С точки зрения воздействия на скорость кодирования алфавит из $2^8 = 256$ символов, который широко используется при сжатии компьютерных файлов, может считаться уже достаточно большим. С введением в практическое использование кодировочных таблиц Unicode размер алфавита возрастет до $2^{16} = 65536$ символов и станет настолько большим, что прямое вычисление кумулятивных вероятностей станет совершенно неприемлемым. В настоящей работе предлагается алгоритм, вычисляющий кумулятивные вероятности за время $O(\log |A|)$, что экспоненциально меньше, чем у известных методов (основная идея такого алгоритма была сформулирована ранее в работе [12]).

Для получения заданной (произвольно низкой) избыточности в случае неизвестной статистики источника недостаточно иметь хороший метод кодирования; необходимо достаточно точно оценивать вероятности порождения символов. Удобно рассматривать избыточность, состоящую из двух частей: избыточность кодера, учитывающая "несовершенство" метода кодирования, и избыточность модели, возникающая вследствие замены реальных вероятностей символов их оценками.

Чтобы обеспечить заданную (произвольно низкую) избыточность модели, мы используем адаптивную схему со скользящим окном. Идея скользящего окна хорошо известна в теории информации. Мы отдаем предпочтение этой схеме, потому что она, во-первых, позволяет достаточно точно оценивать статистику источника, во-вторых, легко адаптируется к изменяющейся статистике, и, наконец, при надлежащем выборе размера окна позволяет исключить операции деления в арифметическом кодировании, что почти вдвое увеличивает скорость работы кодера и декодера.

Единственным очевидным недостатком схемы со скользящим окном, который до сих пор мешал широкому использованию этой схемы, является необходимость хранить окно в памяти кодера и декодера. При достижении малой избыточности память, отводимая под окно, становится доминирующей компонентой сложности кодера и декодера.

Чтобы устранить указанный недостаток мы используем особый вид скользящего окна, названный "мнимым скользящим окном", предложенный в [13]. Этот подход позволяет сохранить все свойства скользящего окна и, в то же время, избавить кодер и декодер от хранения в памяти самого окна.

Мы будем измерять сложность по памяти (S), т. е. требуемый объем памяти кодера и декодера, в битах, а сложность по времени (T), т. е. время кодирования и декодирования одного символа, в битовых операциях. Будем рассматривать обе компоненты сложности как функции двух переменных — размера алфавита $|A|$ и избыточности r. Для известных методов в случае марковского источника порядка μ справедливы следующие оценки:

$$S < \text{const}\left(|A|^{\mu}\left(\frac{|A|}{r}\log|A| + |A|\log\frac{|A|}{r}\right)\right) \text{ бит},$$

$$T < \text{const}\left(\log\frac{|A|}{r}\left(|A| + \log\log\frac{|A|}{r}\log\log\log\frac{|A|}{r}\right)\right) \text{ битовых операций}$$

(здесь и далее $\log x = \log_2 x$ и const обозначает некоторую (всякий раз различную) положительную константу). Для предлагаемого метода сложность кодера и декодера ограничена сверху следующими оценками:

$$S < \text{const}\left(|A|^{\mu}|A|\log\frac{|A|}{r}\right) \text{ бит},$$

$$T < \text{const}\left(\log\frac{|A|}{r}\left(\log|A| + \log\log\frac{|A|}{r}\log\log\log\frac{|A|}{r}\right)\right) \text{ битовых операций}.$$

Заметим, что оценки для источников без памяти получаются при $\mu = 0$. При кодировании источников с большими алфавитами ($|A| \to \infty$) сложность предлагаемого метода как по памяти, так и по времени, экспоненциально ниже, чем для известных методов.

Остановимся кратко на содержании работы. В § 2 описывается метод, обеспечивающий быстрое выполнение операций с кумулятивными вероятностями, и исследуются его свойства. В § 3 схема мнимого скользящего окна используется для построения кода, эффективного по памяти. В § 4 дается обобщение рассмотренных методов на сучай марковских источников. Описание арифметического кода вынесено в § 5.

§ 2. Быстрое кодирование с использованием скользящего окна

Пусть дан источник без памяти, порождающий буквы из алфавита $A = \{a_1, a_2, \ldots, a_n\}$ с неизвестными вероятностями $p(a_1), p(a_2), \ldots, p(a_n)$. Будем считать, что размер алфавита равен некоторой степени двойки, $n = 2^m$ (это не является существенным ограничением, т.к. если $n < 2^m$, то алфавит может быть расширен путем введения в него $2^m - n$ пустых символов, имеющих нулевые вероятности появления, при тривиальных изменениях в алгоритмах). Пусть источник порождает сообщение $x_1 \ldots x_{l-1}x_l \ldots$, $x_i \in A$ для всех i. Окно определяется как последовательность последних w символов, порожденных источником, где w обозначает размер окна. В некоторый момент времени l окно содержит символы $x_{l-w} \ldots x_{l-2}x_{l-1}$. Во время кодирования (или декодирования) окно "скользит" вдоль сообщения: как только очередной закодированный символ вводится в окно, самый "старый" символ удаляется. Каждой букве $a_i \in A$ поставлен в соответствие счетчик c_i, значение которого равно числу появлений буквы a_i в текущем окне плюс единица. Совокупность этих счетчиков позволяет определять частоту встречаемости для каждой буквы. (Прибавление единицы гарантирует, что никакая буква алфавита не будет иметь нулевую частоту встречаемости, так как любая буква может появиться на выходе источника в следующий момент времени, даже если она отсутствовала в текущем окне.) С учетом этих допущений сумма всех счетчиков равна размеру окна плюс размер алфавита источника: $\sum_{i=1}^{n} c_i = w + n$.

Удобно выбрать размер окна w таким образом, чтобы

$$w + n = 2^{\tau}, \tag{1}$$

где τ — некоторое положительное целое число. Тогда оценки вероятностей появления символов $\hat{p}(a_1)$, $\hat{p}(a_2)$, $\ldots$, $\hat{p}(a_n)$ могут быть получены как $\hat{p}(a_i) = c_i/2^\tau$ для всех i. Параметр τ в этом случае также задает размер счетчиков в битах.

Обозначим очередной символ, который должен быть закодирован, через u, а последний символ в окне, который должен быть удален, через v (в момент времени l имеем $u = x_l$ и $v = x_{l-w}$). Адаптивное кодирование символа выполняется следующим образом: 1) закодируем u в соответствии с текущим оценочным распределением вероятностей ($\hat{p}(u) = c(u)/2^\tau$); 2) уменьшим на единицу счетчик $c(v)$, соответствующий букве v; 3) удалим v из окна; 4) увеличим на единицу счетчик $c(u)$, соответствующий букве u; 5) введем u в окно. Заметим, что итоговая сумма всех счетчиков остается постоянной (равной 2^τ).

Декодер при условии, что он начинает работу с тем же самым содержимым счетчиков, что и кодер, декодирует символ в соответствии с текущими оценками вероятностей, после чего изменяет счетчики и сдвигает окно точно так же, как кодер.

Кодирование символа при заданном оценочном распределении вероятностей может быть выполнено эффективно, с произвольно низкой избыточностью, посредством арифметического кодирования (см. § 5 для уточнения деталей метода кодирования). Этот прием, однако, требует, чтобы для каждого кодируемого символа $u = a_i$ был задан интервал кумулятивных сумм счетчиков $[Q_i,\, Q_{i+1})$, где Q_* определяются следующим образом:

$$Q_1 = 0, \quad Q_i = \sum_{j<i} c_j, \quad i = 2, 3, \ldots, n+1 \tag{2}$$

(по этим кумулятивным суммам строятся кодовые слова в арифметическом коде).

Прямое вычисление Q_* по формулам (2) требует $O(n)$ операций сложения чисел размером в τ бит, или $O(\tau n)$ битовых операций. Ниже мы описываем метод, который позволяет уменьшить временную сложность до $O(\tau \log n)$ операций над битами.

Будем хранить не только счетчики $c_1, c_2, \ldots$ (обозначим это множество через C^1), но также и суммы пар счетчиков $c_1+c_2, c_3+c_4, \ldots$ (C^2), суммы четверок $c_1+c_2+c_3+c_4$, $c_5+c_6+c_7+c_8, \ldots$ (C^3) и т. д. Например, если $n = 8$, то необходимо хранить множества C^1, C^2, C^3, как показано на рисунке. Каждый элемент этих множеств есть

C^1 :

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
1	2	3	4	5	6	7	8

C^2 :

c_1+c_2	c_3+c_4	c_5+c_6	c_7+c_8
1	2	3	4

C^3 :

$c_1+c_2+c_3+c_4$	$c_5+c_6+c_7+c_8$
1	2

Структура множеств C^* для $n = 8$

число размером в τ бит, и всего требуется $(2n-2)\tau$ бит памяти, чтобы запомнить все эти числа. Важно то, что использование этих множеств позволяет существенно ускорить вычисление кумулятивных вероятностей. Например, Q_4 и Q_8 могут быть вычислены как

$$\begin{aligned} Q_4 &= c_1 + c_2 + c_3 &&= C_1^2 + C_3^1, \\ Q_8 &= c_1 + c_2 + \cdots + c_7 &&= C_1^3 + C_3^2 + C_7^1. \end{aligned}$$

Легко видеть, что вычисление любого Q_i, $i = 2, 3, \ldots, 8$, требует сложения самое большее трех чисел.

В общем случае при заданных счетчиках $c_1, c_2, \ldots, c_n$, $n = 2^m$, множества C^i, $i = 1, 2, \ldots, m$, строятся следующим образом:

$$
\begin{aligned}
&C_1^1 = c_1, \quad C_2^1 = c_2, \quad \ldots, \quad C_n^1 = c_n,\\
&C_1^2 = c_1 + c_2, \quad C_2^2 = c_3 + c_4, \quad \ldots, \quad C_{n/2}^2 = c_{n-1} + c_n,\\
&C_1^3 = c_1 + \cdots + c_4, \quad C_2^3 = c_5 + \cdots + c_8, \quad \ldots, \quad C_{n/4}^3 = c_{n-3} + \cdots + c_n,\\
&\ldots\\
&C_1^m = c_1 + c_2 + \cdots + c_{n/2}, \quad C_2^m = c_{n/2+1} + c_{n/2+2} + \cdots + c_n.
\end{aligned}
$$

Отметим, что не все элементы этих множеств используются в вычислениях. Фактически, достаточно хранить в памяти только элементы с нечетными индексами. Тем не менее, мы будем пользоваться вышеприведенной конструкцией для облегчения понимания.

Покажем, как можно вычислить Q_{k+1}, $k = 1, 2, \ldots, n$ (напомним, что $Q_1 = 0$). Пусть k имеет двоичное разложение $k_m k_{m-1} \ldots k_1$, где каждая цифра k_i равна нулю или единице. Тогда

$$Q_{k+1} = k_m C^m_{\lfloor 2k/n \rfloor} + k_{m-1} C^{m-1}_{\lfloor 4k/n \rfloor} + k_{m-2} C^{m-2}_{\lfloor 8k/n \rfloor} + \cdots + k_1 C^1_k$$

(заметим, что в вычислении участвуют только те C^i_j, для которых $k_i = 1$). Данная сумма имеет не более $m = \log n$ τ-битных слагаемых, так что время вычисления кумулятивных вероятностей составляет $O(\tau \log n)$ битовых операций.

При включении и исключении элементов из скользящего окна необходимо увеличивать или уменьшать счетчики на единицу. При этом все элементы множеств, которые зависят от этих счетчиков, также должны быть увеличены или уменьшены. Более точно, увеличение (уменьшение) c_k необходимо заменить увеличением (уменьшением) $C^1_k, C^2_{\lceil k/2 \rceil}, C^3_{\lceil k/4 \rceil}, \ldots, C^m_{\lceil k/(n/2) \rceil}$, что требует также $O(\tau \log n)$ операций над битами. Например, чтобы увеличить c_3 (см. рисунок) необходимо увеличить не только C^1_3, но также C^2_2 и C^3_1.

Наконец, последняя операция, которая использует кумулятивные суммы счетчиков — это нахождение интервала $I(z) = [Q_i, Q_{i+1})$, содержащего заданное число z (и буквы $u = a_i$, которой этот интервал соответствует). Эта задача возникает при декодировании, когда число z составляется из символов кодовой последовательности (см. алгоритм декодирования в Приложении). Покажем, как можно решить эту задачу путем использования множеств C^* за время $O(\tau \log n)$.

Вначале сравним z с C^m_1. Если $z < C^m_1$, то искомый интервал находится внутри интервала $[0,\, C^m_1)$, в противном случае он принадлежит интервалу $[C^m_1,\, 2^\tau)$. В первом случае продолжим сравнение z с C^{m-1}_1, а во втором — с $(C^m_1 + C^{m-1}_3)$, что позволит выявить один из интервалов $[0,\, C^{m-1}_1)$, $[C^{m-1}_1,\, C^m_1)$, $[C^m_1,\, C^m_1 + C^{m-1}_3)$ или $[C^m_1 + C^{m-1}_3,\, 2^\tau)$, и т. д. Процедура нахождения интервала $I(z)$ может быть описана как итеративный процесс:

$k := m,\ j := 1,\ Q := 0;$
WHILE $k \neq 0$ DO
  $j := 2j - 1;$
  IF $z \geq Q + C^k_j$ THEN $Q := Q + C^k_j,\ j := j + 1;$
  $k := k - 1.$

Процесс останавливается, когда $k = 0$, и мы имеем $I(z) = [Q,\, Q + C^1_j)$, закодированный символ $u = a_j$. Так как количество итераций в этом алгоритме равно $m = \log n$, время вычислений равно $O(\tau \log n)$.

Теперь определим значение параметра τ. Из формулы (1) $\tau = \log(w+n)$, где размер алфавита n задан, а размер окна w может быть выбран так, чтобы обеспечить желаемую избыточность. Взаимосвязь между размером окна и избыточностью будет установлена нижеследующей теоремой, но предварительно необходимо уточнить используемое в ней понятие избыточности. Мы будем рассматривать верхнюю грань избыточности (т. е. разности между средней длиной кодового слова и энтропией источника), взятую по всему множеству источников без памяти, порождающих символы из алфавита A, и для краткости будем называть ее избыточностью r, как прежде.

Т е о р е м а 1. *Пусть дан источник без памяти, порождающий буквы в алфавите $A = \{a_1, a_2, \ldots, a_n\}$. Пусть для кодирования этого источника используется адаптивная схема, построенная на основе объединения скользящего окна размером в w символов, $w + n = 2^\tau$, и арифметического кодирования. Тогда избыточность r удовлетворяет неравенству*

$$r < \log e \frac{n-1}{w+1} + \frac{n+1}{w+n}. \tag{3}$$

Д о к а з а т е л ь с т в о. Избыточность r оценивается сверху суммой двух избыточностей — r_m (обусловленной заменой неизвестных вероятностей символов их оценками) и r_c (вносимой кодером):

$$r \le r_m + r_c.$$

Рассмотрим обе составляющие по отдельности.

Избыточность r_m определяется следующим равенством:

$$r_m = \sum_{v \in A^w} p(v) \sum_{a \in A} p(a) \log \frac{1}{\hat{p}_v(a)} - \sum_{a \in A} p(a) \log \frac{1}{p(a)}. \tag{4}$$

Здесь слово v обозначает содержимое окна, определяющее оценку вероятности $\hat{p}_v(a)$ для кодируемого символа a, $\hat{p}_v(a) = c_v(a)/2^\tau$, $p(v)$ обозначает вероятность появления v на выходе источника. Первая сумма в (4) может интерпретироваться как энтропия модели, вторая — как энтропия источника. Следующая цепочка равенств и неравенств дает оценку для r_m.

$$\begin{aligned}
r_m &= \sum_{v \in A^w} p(v) \sum_{a \in A} p(a) \log \frac{2^\tau}{c_v(a)} - \sum_{a \in A} p(a) \log \frac{1}{p(a)} \quad = \\
&= \sum_{v \in A^w} p(v) \sum_{a \in A} p(a) \log \frac{2^\tau p(a)}{c_v(a)} \quad = \\
&= \log e \sum_{v \in A^w} p(v) \sum_{a \in A} p(a) \ln \left(\frac{2^\tau p(a)}{c_v(a)} - 1 + 1 \right) \quad \le \\
&\le \log e \sum_{v \in A^w} p(v) \sum_{a \in A} p(a) \left(\frac{2^\tau p(a)}{c_v(a)} - 1 \right) \quad = \\
&= \log e \left(\sum_{a \in A} \sum_{i=0}^{w} \binom{w}{i} p(a)^{i+1} (1-p(a))^{w-i} \frac{2^\tau p(a)}{i+1} \right) - \log e \quad = \\
&= \log e \left(\sum_{a \in A} p(a) \sum_{i=0}^{w} \frac{2^\tau}{w+1} \binom{w+1}{i+1} p(a)^{i+1} (1-p(a))^{(w+1)-(i+1)} \right) - \log e \quad <
\end{aligned}$$

$$
\begin{aligned}
&< \log e\left(\frac{2^{\tau}}{w+1}\sum_{a\in A}p(a)\sum_{i=0}^{w+1}\binom{w+1}{i+1}p(a)^{i+1}(1-p(a))^{(w+1)-(i+1)}\right)-\log e \quad = \\
&= \log e\frac{2^{\tau}}{w+1}-\log e=\log e\frac{2^{\tau}-(w+1)}{w+1} \quad = \\
&= \log e\frac{n-1}{w+1}.
\end{aligned}
$$

Здесь было использовано неравенство $\ln(x+1) < x$ и свойства распределения Бернулли.

В соответствии со следствием к теореме 4, избыточность арифметического кодирования при условии, что минимальная вероятность символа равна $1/2^{\tau}$ и регистры размером в t бит используются при вычислениях, $t \geq \tau+2$, удовлетворяет неравенству

$$r_c < (n+1)(\tau+\log e)2^{-(t-2)}.$$

Можно выбрать любое $t = O(\tau)$, $\tau \to \infty$, без увеличения асимптотической сложности арифметического кодирования. Пусть

$$t = \lceil \tau + \log(\tau+\log e)\rceil + 2. \tag{5}$$

Тогда

$$r_c < (n+1)(\tau+\log e)2^{-(\tau+\log(\tau+\log e))} = \frac{(n+1)(\tau+\log e)}{(\tau+\log e)2^{\tau}} = \frac{n+1}{w+n}. \quad \blacktriangle$$

Заметим, что равенство (5) дает разумный выбор параметра t для арифметического кодирования. Параметр t фактически определяет точность арифметики, использующейся в алгоритмах арифметического кодирования и декодирования. Если взять значение t меньше значения, предписанного неравенством (5), то избыточность кодера может заметно превысить избыточность, связанную с незнанием статистики источника, и в этом случае память, отводимая для хранения окна, будет использоваться неэффективно. Если t много больше правой части в (5), то рост вычислительной сложности кодера и декодера становится неоправданным, так как не приводит к существенному снижению общей избыточности. Если значение t выбирается близким к нижней границе в (5), то избыточности кодера и модели становятся величинами одного порядка, что приводит к наиболее сбалансированному росту вычислительной сложности при уменьшении общей избыточности.

Следующая наша задача состоит в оценивании сложности предложенной адаптивной схемы.

Объем памяти кодера и декодера определяется необходимостью хранить окно, что требует $w \log n$ бит, множества C^* для поддержки операций с кумулятивными суммами счетчиков, что требует $\tau(n-1)$ бит, и внутренние регистры для арифметического кодирования, что требует $O(t)$ бит.

Время кодирования и декодирования в пересчете на символ источника определяется временем выполнения операций с множествами C^*, что требует $O(\tau \log n)$ битовых операций, и вычислением границ интервалов в арифметическом кодировании, при котором используется t-битное умножение и (при декодировании) деление, что, в свою очередь, требует $O(t \log t \log\log t)$ битовых операций при $t \to \infty$ (при использовании метода Шенхаге–Штрассена [14], который является самым быстрым из известных методов умножения и деления).

Свойства метода характеризует следующая

Т е о р е м а 2. *Пусть дан источник без памяти, порождающий буквы в некотором алфавите размера n. Пусть к этому источнику применяется адаптивная схема, основанная на скользящем окне и арифметическом кодировании и обеспечивающая заданную избыточность r. Тогда объем памяти кодера и декодера S и время кодирования и декодирования T характеризуются следующими оценками:*

$$S < \mathrm{const}\left(\frac{n}{r}\log n + n\log\frac{n}{r}\right), \tag{6}$$

$$T < \mathrm{const}\left(\log\frac{n}{r}\left(\log n + \log\log\frac{n}{r}\log\log\log\frac{n}{r}\right)\right). \tag{7}$$

Д о к а з а т е л ь с т в о. Из равенства (5), определяющего выбор параметра t для арифметического кодирования, $t = O(\tau)$, $\tau \to \infty$. Из (1) $\tau = \log(w+n)$. Из теоремы 1 может быть легко получена верхняя оценка для размера окна при заданных избыточности и размере алфавита $w < \mathrm{const}(n/r)$. Прибавляя к обеим частям этого неравенства n и логарифмируя, получаем $\log(w+n) < \mathrm{const}\log(n/r)$. Тогда $w\log n < \mathrm{const}(n/r)\log n$, $\tau(n-1) < \mathrm{const}\, n\log(n/r)$, $t < \mathrm{const}\log(n/r)$. Суммирование доминирующих сложностей всех операций приводит к утверждению теоремы. ▲

Рассмотрим два следствия из теоремы 2.

С л е д с т в и е 1. *Если размер алфавита n растет, а избыточность r остается постоянной, то*

$$S = O(n\log n),$$
$$T = O(\log^2 n).$$

Оценка времени здесь лучше, чем для известных методов, для которых $T = O(n\log n)$.

С л е д с т в и е 2. *Если размер алфавита n фиксирован, а избыточность r стремится к нулю, то*

$$S = O(1/r),$$
$$T = O(\log(1/r)\log\log(1/r)\log\log\log(1/r)).$$

Время кодирования здесь определяется в основном сложностью арифметического кодирования (требующего умножений и делений чисел разрядностью $t = O(\log(1/r))$). В § 3 будет показано, как уменьшить объем памяти до $O(\log(1/r))$ бит путем использования схемы мнимого скользящего окна.

§ 3. Использование мнимого скользящего окна для уменьшения объема памяти кодера и декодера

В данном параграфе сначала приводится краткое описание схемы мнимого скользящего окна в форме, позволяющей совместить эту схему с методами кодирования, рассмотренными в предыдущих параграфах.

Пусть, по-прежнему, дан источник без памяти, определенный в § 2. Будем использовать те же самые понятия окна, счетчиков и способа оценивания вероятностей. Обозначим через ν_i число появлений буквы $a_i \in A$ в текущем окне, $\nu_i = c_i - 1$ для всех i. Определим ξ как случайную величину, принимающую значения из множества $1, 2, \ldots, n$ с вероятностями

$$\Pr\{\xi = i\} = \frac{\nu_i}{w}, \quad i = 1, 2, \ldots, n. \tag{8}$$

Кодирование очередного символа u с использованием мнимого скользящего окна выполняется следующим образом: 1) кодируем u в соответствии с текущим оценочным распределением вероятностей ($\hat{p}(u) = c(u)/2^\tau$); 2) генерируем случайную величину ξ; 3) уменьшаем на единицу счетчик c_ξ, т. е. счетчик, соответствующий ξ-й букве алфавита A; 4) увеличиваем на единицу счетчик $c(u)$, соответствующий букве u.

Отличительная особенность этой схемы состоит в том, что вместо счетчика, соответствующего последнему символу в окне, на единицу уменьшается случайно выбранный счетчик. Поэтому нет необходимости хранить окно, что освобождает $w \log n$ бит памяти, т. е. экспоненциально уменьшает объем памяти по сравнению с обычным (реальным) скользящим окном. В работе [13] было показано, что с ростом числа закодированных символов распределение частот, формируемое мнимым скользящим окном, сходится по вероятности к распределению частот, определяемому обычным скользящим окном, с экспоненциальной скоростью. Это означает, что мнимое скользящее окно проявляет те же свойства, что и обычное скользящее окно, а именно, оно позволяет достаточно точно оценивать статистику источника и быстро адаптируется в случае изменения статистики.

Рассмотрим задачу генерации случайной величины ξ. Будем полагать, что имеется двоичный источник, порождающий последовательность $z_1 z_2 \ldots z_k \ldots$, где каждый элемент $z_k \in \{0, 1\}$, $\Pr\{z_k = 0\} = \Pr\{z_k = 1\} = 1/2$, и все символы статистически независимы. Мы не рассматриваем здесь сложность получения случайных бит. Наша цель состоит в преобразовании последовательности случайных бит в значения случайной величины ξ с распределением вероятностей (8). Обозначим через z случайное число, построенное из τ случайных бит, т. е. двоичное разложение z есть $z_1 z_2 \ldots z_\tau$. Ясно, что z может принять любое значение из множества $\{0, 1, \ldots, 2^\tau - 1\}$ с вероятностью $1/2^\tau$. Также ясно, что для любого $w < 2^\tau$ процесс генерации z до тех пор пока не будет получено $z < w$ даст случайную величину z^*, которая будет принимать любое значение из множества $\{0, 1, \ldots, w - 1\}$ с вероятностью $1/w$.

Определим

$$\Theta_1 = 0, \quad \Theta_i = \sum_{j<i} \nu_j, \quad i = 2, 3, \ldots, n+1, \quad \Theta_{n+1} = w.$$

Рассмотрим случайную величину ξ, определенную на $\{0, 1, \ldots, w-1\}$, такую, что $\xi = i$ тогда и только тогда, когда $\Theta_i \le z^* < \Theta_{i+1}$. Легко может быть найдено, что

$$\Pr\{\xi = i\} = \Pr\{\Theta_i \le z^* < \Theta_{i+1}\} = \frac{\Theta_{i+1} - \Theta_i}{w} = \frac{\nu_i}{w}.$$

Это как раз то распределение (8), которое мы стремились получить.

Переходя к реализации этой процедуры, отметим, что нахождение Θ_i и Θ_{i+1} при заданном z^* — та же самая задача, что и нахождение интервала и соответствующего ему символа при арифметическом декодировании (как это было описано в § 2). Следовательно, эта процедура может быть эффективно выполнена при использовании структуры данных C^*, определенной в § 2. Единственное отличие здесь состоит в том, что значения ν_i на единицу меньше счетчиков c_i, участвующих в построении C^*. Это отличие может быть учтено путем вычитания величины $s = 2^{k-1}$ из C_j^k. Запишем алгоритм генерации случайной величины ξ:

а) REPEAT генерация случайного числа z UNTIL $z < 2^\tau - n$;
б) $k := m,\ i := 1,\ \Theta := 0,\ s := 2^{m-1}$;
WHILE $k \ne 0$ DO

$i := 2i - 1;$
IF $z \geq \Theta + C_i^k - s$ THEN $\Theta := \Theta + C_i^k - s,\ i := i + 1;$
$s := s/2,\ k := k - 1.$

Когда процесс останавливается, мы имеем $\xi = i$.

Обозначим через N_a среднее число итераций в а). Так как каждая следующая итерация выполняется с вероятностью $n/2^\tau$, то

$$N_a = 1 + \frac{n}{2^\tau}\left(1 + \frac{n}{2^\tau}(1 + \cdots)\right) = 1 + \frac{n}{2^\tau} + \left(\frac{n}{2^\tau}\right)^2 + \cdots = \frac{1}{1 - n/2^\tau}.$$

Каждая из $m = \log n$ итераций в б) требует $O(\tau)$ битовых операций. Отсюда получаем среднюю сложность алгоритма генерации ξ

$$T_\xi = O\left(\frac{1}{1 - n/2^\tau} + t \log n\right) = O(\tau \log n), \quad \tau \to \infty,\ n \to \infty,$$

что не увеличивает общей асимптотической сложности метода (здесь мы полагаем, что при достижении заданной избыточности 2^τ растет быстрее, чем n).

Все результаты, полученные в этом параграфе, могут быть изложены в виде следующей теоремы.

Т е о р е м а 3. *Пусть дан источник без памяти, порождающий буквы в алфавите $A = \{a_1, a_2, \ldots, a_n\}$. Пусть для кодирования сообщения источника используется адаптивная схема, основанная на мнимом скользящем окне со счетчиками размером в τ бит и на арифметическом кодировании. Тогда*

1) *избыточность r асимптотически ограничена сверху с помощью неравенства*

$$r < \log e \frac{n-1}{2^\tau - n + 1} + \frac{n+1}{2^\tau}; \tag{9}$$

2) *объем памяти кодера (декодера) S и время кодирования (декодирования) T задаются следующими оценками:*

$$S < \mathrm{const}\left(n \log \frac{n}{r}\right), \tag{10}$$

$$T < \mathrm{const}\left(\log \frac{n}{r}\left(\log n + \log\log \frac{n}{r} + \log\log\log \frac{n}{r}\right)\right). \tag{11}$$

Итак, время адаптивного кодирования при использовании мнимого скользящего окна по существу то же самое, что и в случае обычного скользящего окна, в то время как память меньше на порядок малости из-за отсутствия необходимости хранить окно (исключается терм $(n/r)\log n$ в (6)).

Дадим некоторые замечания, без строгого их рассмотрения, о путях получения случайных бит $z_1 z_2 \ldots z_k \ldots$. Для реализации мнимого скользящего окна эти биты должны быть известны как кодеру, так и декодеру. Обычный способ решения этой проблемы заключается в использовании синхронизированных генераторов псевдослучайных чисел. Мы предлагаем дополнительную возможность, а именно, использование символов кодовой последовательности (возможно, с какими-нибудь простыми рандомизирующими перестановками). Основанием для такого предложения служит тот факт, что сжатые данные почти случайны или, более точно, кодовая последовательность приближается к последовательности равномерно распределенных кодовых символов при уменьшении избыточности кода. Кодер и декодер могут использовать часть уже закодированного сообщения для получения случайных чисел. Важно, что кодовая последовательность известна как кодеру, так и декодеру, и им необходимо хранить только небольшую текущую ее часть. Такой способ получения случайных чисел может оказаться более простым, чем использование специальных генераторов.

§ 4. Кодирование марковских источников

Все результаты, полученные для источников без памяти, переносятся и на более общий случай. Рассмотрим марковский источник порядка μ, т. е. источник, в котором каждый символ зависит от μ предыдущих символов, $\mu \geq 1$. Пусть алфавит источника будет снова $A = \{a_1, a_2, \ldots, a_n\}$. Для кодирования такого источника может быть использован хорошо известный в теории информации прием, связанный с разложением марковского источника в $|A|^\mu$ источников без памяти. Будем хранить $|A|^\mu = n^\mu$ мнимых скользящих окон, каждое из которых соответствует одному слову в A^μ. Кроме этого, будем хранить небольшое действительное окно, содержащее последние μ закодированных символов. Пусть источник порождает сообщение $x_1 x_2 \ldots x_l \ldots$ Тогда перед кодированием x_l в окне запомнено слово $x_{l-\mu} \ldots x_{l-1}$. Это слово принадлежит A^μ, и существует мнимое скользящее окно, соответствующее этому слову. Очередной символ x_l может быть закодирован на основании информации, содержащейся в этом мнимом окне, после чего само окно преобразуется по тем же правилам, что и для источника без памяти (т. е. случайно выбранный счетчик уменьшается на единицу, а счетчик, соответствующий x_l, увеличивается на единицу).

Приведем простой пример. Пусть $A = \{a, b\}$, $\mu = 2$ и сообщение, которое необходимо закодировать, $x_1 x_2 x_3 x_4 \ldots = aaba \ldots$. В памяти кодера хранится четыре мнимых скользящих окна, соответствующих комбинациям букв aa, ab, ba и bb, причем каждое окно содержит по два счетчика $c(a)$ и $c(b)$. Символ x_3 кодируется на основании информации, содержащейся в окне, соответствующем комбинации букв aa, так как именно эта комбинация предшествовала символу x_3 в сообщении источника, после чего счетчики $c(a)$ и $c(b)$ этого окна изменяются в соответствии с алгоритмом мнимого скользящего окна; символ x_4 кодируется на основании информации, содержащейся в окне, соответствующем комбинации букв ab, и т. д. Короче говоря, каждая буква источника кодируется (и декодируется) на основании окна, которое представляет статистику для соответствующего источника без памяти.

Так как оценка избыточности (9) является верхней гранью для всего множества источников без памяти, кодирование марковских источников посредством описанной выше схемы гарантирует верность этой оценки для посимвольной избыточности. Временная сложность остается такой же, как (11). Сложность по памяти, однако, увеличивается в n^μ раз из-за необходимости хранить n^μ мнимых окон, так что для объема памяти становится справедливой оценка

$$S < \text{const}\left(n^\mu n \log \frac{n}{r}\right).$$

§ 5. Избыточность арифметического кодирования

Дадим оценку избыточности арифметического кодирования в форме, которая позволила бы судить о вычислительной сложности этого метода. Насколько нам известно, такой оценки до сих пор не было сделано. Мы начнем с того, что приведем алгоритм арифметического кодирования, способный работать в схеме со скользящим окном. Этот алгоритм близок к алгоритму, описанному в [7].

Арифметическое кодирование предполагает приближенное вычисление интервала кумулятивных вероятностей для блока символов (или сообщения) и выбор некоторого числа, принадлежащего этому интервалу, в качестве кодового слова (см., например, [7]). Допустим, что в нашем распоряжении имеются три регистра l, h и d размером в t бит, в которых содержатся, соответственно, нижняя (или правая) и верхняя (или левая) границы интервала и размер интервала. Во избежание некоторых осложнений, будем предполагать, что $1/2^\tau$ есть минимальная ненулевая вероятность символа, и $t \geq \tau + 2$. Нам понадобится также счетчик s размером в σ бит, в котором будет запоминаться количество “сжатий” интервала при масштабировании. Обозначим через $\mathrm{msb}(\cdot)$ старший (наиболее значимый) бит регистра, а через $\mathrm{pmsb}(\cdot)$ — бит, предшествующий старшему. Арифметическое кодирование применяется к блокам символов (или к целым сообщениям) $u_1 u_2 \ldots u_L$, где L обозначает длину блока (или сообщения). С учетом этих допущений и обозначений алгоритм арифметического кодирования записывается следующим образом.

Начальная установка

Выполняется перед кодированием каждого блока из L символов

$l := 0,\ h := 2^t - 1,\ s := 0.$

Кодирование очередного символа

На входе имеем символ $u = a_i$ при заданном интервале кумулятивных вероятностей $[Q_i,\ Q_{i+1})$

$d := h - l + 1,$
$h := l + \lfloor dQ_{i+1}/2^\tau \rfloor - 1,$
$l := l + \lfloor dQ_i/2^\tau \rfloor;$
WHILE $\mathrm{msb}(l) = \mathrm{msb}(h)$ DO
  передать на выход кодера $\mathrm{msb}(l)$,
  s раз передать бит $1 - \mathrm{msb}(l)$, $s := 0$,
  $l := 2l,\ h := 2h + 1 \pmod{2^t};$
WHILE $\mathrm{pmsb}(l) = 1$ AND $\mathrm{pmsb}(h) = 0$ DO
  $s := s + 1,$
  $l := 2l,\ h := 2h + 1 \pmod{2^t};$
$\mathrm{msb}(l) := 0\ \mathrm{msb}(h) := 1.$

Завершение блока (или сообщения)

Выполняется после кодирования последнего символа блока (или сообщения)

$s := s + 1;$
IF $\mathrm{pmsb}(l) = 0$ THEN $b := 0$ ELSE $b := 1;$
передать бит b,
s раз передать бит $1 - b$.

Для полноты описания метода приведем также алгоритм декодирования арифметического кода. Алгоритм декодирвания строится в целом симметрично по отношению к алгоритму кодирования. Главная его особенность состоит во введении дополнительного регистра v, содержащего текущие t бит кодовой последовательности. При декодировании нет необходимости в использовании счетчика s.

Начальная установка

Выполняется только один раз перед декодированием первого блока

$v :=$ первые t бит кодовой последовательности; $l := 0, \quad h := 2^t - 1.$

Декодирование очередного символа

$d := h - l + 1,$
$z := (2^\tau(v - l + 1) - 1)/d,$
найти символ a_i, для которого $Q_i \le z < Q_{i+1}$;
$h := l + \lfloor dQ_{i+1}/2^\tau \rfloor - 1,$
$l := l + \lfloor dQ_i/2^\tau \rfloor;$
WHILE msb(l) = msb(h) DO
  $v := 2v+$ очередной бит кодовой последовательности,
  $l := 2l, \quad h := 2h + 1 \qquad (\mathrm{mod}\ 2^t);$
$b := \mathrm{msb}(v);$
WHILE pmsb(l) = 1 AND pmsb(h) = 0 DO
  $v := 2v+$ очередной бит кодовой последовательности,
  $l := 2l, \quad h := 2h + 1 \qquad (\mathrm{mod}\ 2^t);$
$\mathrm{msb}(v) := b, \quad \mathrm{msb}(l) := 0, \quad \mathrm{msb}(h) := 1.$

В результате $u = a_i$ — закодированный символ.

Следующие леммы определяют некоторые важные черты алгоритма, а теорема дает оценку избыточности.

Л е м м а 1. *Длина L кодируемого блока удовлетворяет неравенству*

$$L \le \frac{2^\sigma - 2}{\tau - 1}.$$

Д о к а з а т е л ь с т в о. При кодировании любого символа, появляющегося с вероятностью не менее $1/2^\tau$, значение s может увеличиться максимум на $\tau - 1$. При кодировании L символов счетчик s не может переполниться, если

$$L(\tau - 1) + 1 \le 2^\sigma - 1,$$

откуда следует утверждение леммы. ▲

Л е м м а 2. *После кодирования любого символа размер d интервала удовлетворяет неравенству*

$$d \ge 2^{t-2} + 2.$$

Д о к а з а т е л ь с т в о. По определению $d = h - l + 1$. Рассматривая комбинации двух старших бит в l и h после кодирования символа, получаем три возможных типа интервалов: $[00*, 11*]$, $[00*, 10*]$ и $[01*, 11*]$, где $*$ обозначает $t - 2$ произвольные двоичные цифры. Можно видеть, что в первом случае длина интервала больше, чем во втором и в третьем, которые эквивалентны. Для второго случая

$$l \leq 2^{t-2} - 1, \ h \geq 2 \cdot 2^{t-2}.$$

Отсюда

$$d \geq 2 \cdot 2^{t-2} + 1 - (2^{t-2} - 1) = 2^{t-2} + 2. \quad \blacktriangle$$

Т е о р е м а 4. *Избыточность на символ r_c арифметического кодирования удовлетворяет неравенству*

$$r_c < n(\tau + \log e) 2^{-(t-2)} + 2/L, \tag{12}$$

где n — размер алфавита, τ — количество бит в представлении вероятностей, t — размер регистров l и h (границ интервала), $t \geq \tau + 2$, L — длина кодируемого блока или сообщения.

Д о к а з а т е л ь с т в о. Избыточность кодирования r_c состоит из двух частей, обозначаемых через r_t и r_L и соответствующих двум слагаемым в (12): r_t возникает вследствие отсечений дробной части при вычислении границ интервала, а r_L обусловлена ограничением длины блока.

Пусть кодируемый символ u появляется с вероятностью $p(u)$, $p(u) \geq 1/2^{\tau}$. Вследствие отсечения дробной части вероятность $p(u)$ фактически заменяется на приближение $\hat{p}(u)$, и

$$\begin{aligned} r_t &= \sum_{u \in A} p(u) \log \frac{1}{\hat{p}(u)} - \sum_{u \in A} p(u) \log \frac{1}{p(u)} \quad = \\ &= \sum_{u \in A} \left(p(u) \log p(u) - p(u) \log \hat{p}(u) \right). \end{aligned}$$

Для $\hat{p}(u)$ имеем

$$p(u) = \hat{p}(u) \pm q(u), \ q(u) < 1/d.$$

Предположим (пессимистически), что $q(u)$ берется с плюсом для каждой буквы. В соответствии с леммой 2 $d \geq 2^{t-2} + 2$, $d > 2^{t-2}$. Принимая во внимание эти оценки, найдем верхнюю границу для $p(u) \log p(u) - p(u) \log \hat{p}(u)$ (опустим (u) для краткости).

$$\begin{aligned} & p \log p - p \log \hat{p} = p \log p - \hat{p} \log \hat{p} - q \log \hat{p} \quad < \\ & < \ p \log p - (p - q) \log(p - q) - \frac{1}{d} \log \frac{\lfloor 2^{-\tau} d \rfloor}{d} \quad < \\ & < \ 1 \cdot \log 1 - (1 - q) \log(1 - q) - \frac{1}{d} \log \frac{\lfloor 2^{-\tau} d \rfloor}{d} \quad < \\ & < \ \frac{d-1}{d} \log \frac{d}{d-1} + \frac{1}{d} \log \frac{d}{\lfloor 2^{-\tau} d \rfloor} \quad \leq \\ & \leq \ \log e \frac{d-1}{d} \left(\frac{d}{d-1} - 1 \right) + \frac{1}{d} \log \frac{d}{\lfloor 2^{-\tau} d \rfloor} \quad = \\ & = \ \frac{1}{d} \left(\log \frac{d}{\lfloor 2^{-\tau} d \rfloor} + \log e \right) \quad < \\ & < \ \frac{1}{2^{t-2}} \left(\log \frac{2^{t-2}}{2^{t-2} 2^{-\tau}} + \log e \right) \quad = \\ & = \ \frac{\tau + \log e}{2^{t-2}}. \end{aligned} \tag{13}$$

Суммируя (13) по $u \in A$, получаем

$$r_t < n(\tau + \log e) 2^{-(t-2)}.$$

Завершение блока добавляет к кодовой последовательности самое большее два дополнительных бита (см. алгоритм). Отсюда

$$r_L < 2/L.$$

Объединяя r_t и r_L, получаем утверждение теоремы. ▲

В интересах основной части статьи можно получить более удобный результат, рассмотрев следующее

С л е д с т в и е. *Если* $\sigma = t$ *и*

$$L = 2^t/\tau, \tag{14}$$

то

$$r_c < (n+1)(\tau + \log e) 2^{-(t-2)}.$$

Д о к а з а т е л ь с т в о. Отметим вначале, что при $t \geq \tau + 2$

$$\frac{2^t - 2}{\tau - 1} > \frac{2^t}{\tau},$$

и (14) удовлетворяет лемме 1. Подстановка (14) в (12) дает утверждение следствия. ▲

СПИСОК ЛИТЕРАТУРЫ

1. *Jelinek F.* Probabilistic information theory. New York: McGraw-Hill, 1968. P. 476–489.
2. *Rissanen J. J.* Generalized Kraft inequality and arithmetic coding // IBM J. Res. Dev. 1976. V. 20. P. 198–203.
3. *Pasco R.* Source coding algorithm for fast data compression. Ph. D. thesis, Dept. Elect. Eng. Stanford Univ. Stanford, CA, 1976.
4. *Rubin F.* Arithmetic stream coding using fixed precision registers // IEEE Trans. Inform. Theory. 1979. V. 25. №6. P. 672–675.
5. *Rissanen J. J., Langdon G. G.* Arithmetic coding // IBM J. Res. Dev. 1979. V. 23. №2. P. 149–162.
6. *Guazzo M.* A general minimum-redundancy source-coding algorithm // IEEE Trans. Inform. Theory. 1980. V. 26. №1. P. 15–25.
7. *Witten I. H., Neal R., Cleary J. G.* Arithmetic coding for data compression // Comm. ACM. 1987. V. 30. №6. P. 520–540.
8. *Cleary J. G., Witten I. H.* Data compression using adaptive coding and partial string matching // IEEE Trans. Commun. 1984. V. 32. №4. P. 396–402.
9. *Moffat A.* A note on the PPM data compression algorithm. Res. Rep. 88/7, Dep. Comput. Sci. Univ. of Melbourne. Australia, 1988.

10. *Willems F. M. J., Shtarkov Y. M., Tjalkens T. J.* The context-tree weighting method: Basic properties // IEEE Trans. Inform. Theory. 1995. V. 41. №3. P. 653–664.

11. *Moffat A., Turpin A.* Efficient construction of minimum-redundancy codes for large alphabets // IEEE Trans. Inform. Theory. 1998. V. 44. №4. P. 1650–1657.

12. *Рябко Б. Я.* Быстрый алгоритм адаптивного кодирования // Пробл. передачи информ. 1990. Т. 26. №4. С. 24–37.

13. *Рябко Б. Я.* Сжатие данных с помощью “мнимого скользящего окна” // Пробл. передачи информ. 1996. Т. 32. №2. С. 22–30.

14. *Ахо А., Ульман Дж., Хопкрофт Дж.* Построение и анализ вычислительных алгоритмов. М.: Мир, 1979.