Math-Net.Ru

Общероссийский математический портал

Б. Я. Рябко, Быстрая нумерация комбинаторных объектов, *Дискрет. матем.*, 1998, том 10, выпуск 2, 101–119

DOI: <https://doi.org/10.4213/dm426>

Использование Общероссийского математического портала Math-Net.Ru подразумевает, что вы прочитали и согласны с пользовательским соглашением
<http://www.mathnet.ru/rus/agreement>

Параметры загрузки:

IP: 3.95.254.165

26 апреля 2023 г., 15:55:28

Быстрая нумерация комбинаторных объектов

© 1998 г. Б. Я. Рябко

Проблема нумерационного кодирования представляет интерес для комбинаторики, теории информации и других разделов дискретной математики. В настоящее время известны методы нумерации перестановок, сочетаний и многих других комбинаторных объектов, не использующие экспоненциально растущую память. У этих методов нумерации скорость кодирования и декодирования, измеряемая числом операций над битовыми словами, превосходит cn , где c — постоянная, а n — длина нумеруемых слов. В работе предлагается новый метод нумерации, скорость кодирования которого равна $O(\log^c n)$, $c > 1$, для перестановок, сочетаний и других комбинаторных объектов, для которых известны методы с неэкспоненциальным объемом памяти.

Работа выполнена при поддержке Российского фонда фундаментальных исследований, грант 96-01-00052.

1. Введение

В общем виде задача нумерационного кодирования может быть сформулирована следующим образом: дан некоторый алфавит, целое число $n \geq 1$ и множество слов S , $S \subset A^n$. Нумерационный код, или метод нумерационного кодирования, должен для любого слова $x \in S$ вычислять его код $\text{code}(x)$, являющийся словом в двоичном алфавите. Причем все слова $\text{code}(x)$ должны быть различными при разных $x \in S$ и иметь одинаковую длину, которая, очевидно, не меньше $\lceil \log |S| \rceil$ бит. Здесь и ниже $\log x = \log_2 x$, $|S|$ — мощность, если S — множество, и длина, если S — слово; $\lceil x \rceil$ — ближайшее к x целое, не меньшее x . При декодировании, наоборот, по $\text{code}(x)$ находят x .

Эта задача известна в комбинаторике, теории сложности вычислений и теории информации (см., например, [1,2]). Она, в частности, возникает при случайном порождении комбинаторных объектов [1]. В теории информации задача нумерационного кодирования тесно связана с так называемым сжатием данных. Сжатие происходит тогда, когда слово $x \in S \subset A^n$, длина которого равна $n \log |A|$ бит, заменяется на слово длины $\text{code}(x)$. Чем меньше отношение $\text{code}(x)/(n \log |A|)$, тем лучше сжатие.

По-видимому, самый простой нумерационный код может быть реализован следующим образом: хранить в памяти компьютера все слова из множества S и соответствующие им слова $\text{code}(x)$, $x \in S$. В этом случае время кодирования и декодирования будет пропорционально $\log |S|$ (или $n \log |A|$), а скорость, то есть время

кодирования и декодирования на букву, не превосходит постоянной, то есть минимально по порядку роста при $n \rightarrow \infty$.

Однако у этого метода есть существенный недостаток, состоящий в том, что необходимо хранить все слова S и $\text{code}(x)$, $x \in S$, в памяти компьютера, реализующего кодирование и декодирование. Это приводит к тому, что объем памяти растет экспоненциально с ростом длины кодируемых слов. Таким образом, задача построения быстрого метода кодирования решается очень просто, если можно использовать экспоненциально растущую память для кодера и декодера. Однако, конечно, такие методы не представляют большого интереса. Одной из важнейших задач теории нумерации комбинаторных объектов и теории информации является разработка методов, не требующих экспоненциально растущего объема памяти с ростом длины нумеруемых слов. Такие методы известны для многих комбинаторных объектов (см. [1, 2]); мы упомянем здесь только две задачи, для которых известны методы с полиномиальным объемом памяти: нумерацию перестановок целых чисел от 1 до n и нумерацию всех бинарных слов длины n с заданным числом единиц. Последняя задача представляет большой интерес для теории информации, так как является одной из основных для сжатия данных. Метод, использующий полиномиально растущую память, был независимо предложен в [3-5], см. также [2].

Мы используем первую задачу в качестве примера для того, чтобы представить основную идею метода. Пусть S_n — множество всех перестановок чисел $\{1, 2, \dots, n\}$, $n \geq 2$. Для перестановки $\bar{x} = x_1 x_2 \dots x_n \in S_n$ и i , $1 \leq i \leq n$, обозначим через r_i количество чисел в этой перестановке, которые меньше x_i и расположены в $\bar{x}$ правее i -й позиции. Легко может быть показано (см. [1, 2]), что равенство

$$\text{code}(x_1 x_2 \dots x_n) = \sum_{i=1}^n r_i (n - i)! \quad (1)$$

задает нумерацию всех перестановок. Если, например, все вычисления в правой части формулы проводить в двоичной системе счисления, то результат, рассматриваемый как бинарное слово длины $\lceil \log(n!) \rceil$ бит, будет номером перестановки. Используя схему Горнера соотношение (1) можно представить в виде

$$\text{code}(x_1 \dots x_n) = (\dots (r_1(n-1) + r_2)(n-2) + r_3)(n-3) + \dots \quad (2)$$

Например, для перестановки $x_1 x_2 \dots x_8 = 32647185$,

$$r_1 = 2, \quad r_2 = 1, \quad r_3 = 3, \quad r_4 = 1, \quad r_5 = 2, \quad r_6 = 0, \quad r_7 = 1, \quad r_8 = 0$$

и в соответствии с (2) находим, что

$$\text{code}(32647185) = ((((((2 \cdot 7 + 1)6 + 3)5 + 1)4 + 2)3 + 0)2 + 1)1 = 11197.$$

Понятно, что объем памяти, требуемый для вычислений по формулам (1) или (2) растет не экспоненциально с ростом n , а полиномиально. Оценим скорость кодирования по правилу (2). Для этого заметим, что после вычисления $r_1(n-1)$ результат может быть представлен словом длины $2 \log n$ бит, после вычисления $(r_1(n-1) + r_2)(n-2)$ словом длины $3 \log n$ бит, после вычисления $((r_1(n-1) + r_2)(n-2) + r_3)(n-3)$ словом длины $4 \log n$ бит и так далее. Очевидно, при умножении необходимо по крайней мере просмотреть сомножители, следовательно, необходимо просмотреть слова длины $\log n$ бит, $2 \log n$ бит, $\dots$, $(n-1) \log n$ бит или всего

$$\log n + 2 \log n + \dots + (n-1) \log n > cn^2$$

бит, где c — положительная постоянная. Значит, время вычисления номера перестановки по формуле (2) не меньше cn^2 операций над битовыми словами. Следовательно, скорость, то есть время на букву, не меньше cn операций. Подчеркнем, что это нижняя оценка, учитывающая только время просмотра слов.

Основная идея предлагаемого метода очень проста и состоит в следующем.

Во-первых, предлагается по-новому расставить скобки и проводить вычисления по формуле

$$\begin{aligned} \text{code}(x_1 \dots x_n) = & (\dots (r_1(n-1) + r_2)((n-2)(n-3)) \\ & + (r_3(n-3) + r_4))((n-4)(n-5))((n-6)(n-7)) \\ & + (r_5(n-5) + r_6)((n-6)(n-7)) + (r_7(n-7) + r_8)) \dots \end{aligned} \quad (3)$$

Эта схема вычислений позволяет проводить большую часть операций умножения с относительно короткими числами, в то время как в (2) длина одного сомножителя постоянно возрастает, что и делает время вычислений большим.

Во-вторых, мы предлагаем использовать для умножения метод Шенхаге-Штрассена, который является самым быстрым (его описание см., например, в [6]). Заметим сразу, что при использовании этого метода для времени $T(L)$ умножения двух L -битовых чисел справедлива оценка

$$T(L) = O(L \log L \log \log L) \quad (4)$$

Та же оценка выполняется и для времени деления числа длины $2L$ бит на L -битовое число. Во всех случаях время измеряется числом операций над однобитовыми словами [6]. Это довольно естественно в случае, когда длина чисел неограниченно возрастает.

Оценим время вычислений для предлагаемого метода, считая, что n — степень двойки. Для вычисления величин

$$\begin{aligned} (r_1(n-1) + r_2), \quad (r_3(n-3) + r_4), \quad (r_5(n-5) + r_6), \dots \\ (n-2)(n-3), \quad (n-4)(n-5), \quad (n-6)(n-7), \quad \dots \end{aligned} \quad (5)$$

требуется не более $2n$ операций умножения над числами, длина которых не превосходит $\log n$ бит каждое. Для вычисления

$$\begin{aligned} (r_1(n-1) + r_2)(n-2)(n-3) + (r_3(n-3) + r_4), \\ (r_5(n-5) + r_6)(n-6)(n-7) + (r_7(n-7) + r_8) \end{aligned}$$

и

$$((n-2)(n-3))((n-4)(n-5)), \quad ((n-6)(n-7))((n-8)(n-9)), \quad \dots$$

требуется не более $2(n/2)$ операций умножения над числами длины, не превосходящей $2 \log n$ каждое и так далее. Отсюда и из оценки времени умножения (4) мы получаем, что вычисление величин (5) требует

$$O(n \log n \log n \log \log n)$$

операций, вычисление (6) требует

$$O(n \log n \log(2 \log n) \log \log(2 \log n))$$

операций и, вообще, вычисление k -го слагаемого $k = 0, 1, \dots, [\log n] - 1$, требует

$$O(n \log n \log(2^k \log n) \log \log(2^k \log n))$$

операций. Суммируя, после алгебраических преобразований получаем, что общее время вычисления номера перестановки по схеме (3) есть

$$O(n \log^3 n \log \log n),$$

и время, затрачиваемое на кодирование одного символа, равно $O(\log^3 n \log \log n)$ операций над однобитовыми словами. Это экспоненциально меньше, чем оценка скорости sn , полученная для метода нумерации, базирующегося на схеме Горнера (2), даже если в (2) использовать метод быстрого умножения.

Рассмотрим пример кодирования по предложенной схеме (3), используя прежнюю перестановку 32647185. Получаем, что

$$\text{code}(32647185) = ((2 \cdot 7 + 1)(6 \cdot 5) + (3 \cdot 5 + 1))(4 \cdot 3)(2 \cdot 1) + (2 \cdot 3 + 0)(2 \cdot 1) + 1 + 0 = 11197.$$

Метод декодирования, также имеющий скорость $O(\log^3 n \log \log n)$ операций на букву, строится совершенно аналогично, поэтому ограничимся описанием схемы метода, используя этот же пример. Итак, дан номер перестановки 11197 и надо найти соответствующую перестановку. Для декодирования сначала вычисляем пары

$$(n - 4)(n - 5) = 4 \cdot 3 = 12, \quad (n - 6)(n - 7) = 2 \cdot 1 = 2,$$

а затем их произведение

$$((n - 4)(n - 5))((n - 6)(n - 7)) = 12 \cdot 2 = 24.$$

Затем делим $\text{code}(x_1 \dots x_8)$ на $((n - 4)(n - 5))((n - 6)(n - 7))$, то есть 11197 на 24 и получаем $11197 = 466 \cdot 24 + 13$. Это означает, что 466 кодирует первые четыре цифры перестановки, а 13 последние четыре цифры. И далее 466 и 13 декодируются независимо. Вычисляем $(n - 2)(n - 3) = 6 \cdot 5 = 30$, делим 466 на 30 и получаем $466 = 15 \cdot 30 + 16$. Значит, 15 кодирует $x_1 x_2$, а 16 кодирует $x_3 x_4$. Делим 15 на $(n - 1) = 7$ и получаем $15 = 2 \cdot 7 + 1$. Значит, $r_1 = 2, r_2 = 1$. Следовательно, $x_1 = 3, x_2 = 2$. Продолжая аналогично, находим остальные знаки.

Учитывая приведенную выше оценку (4) времени деления и умножения методом Шенхаге–Штрассена, легко получаем, что время декодирования одной буквы равно

$$O(\log^3 n \log \log n),$$

то есть совпадает со скоростью кодирования.

Пример быстрой нумерации и денумерации, или кодирования и декодирования, перестановок демонстрирует основную идею предлагаемого метода, базирующегося на подходящей расстановке скобок. Важно также отметить, что он основан на популярном в теории сложности вычислений принципе “разделяй и властвуй” (см. его описание, например, в [6]).

Ранее этот же подход был использован в работе автора [7] при построении быстрых кодов для источников информации.

В следующих двух параграфах дано описание общего метода кодирования и декодирования.

2. Быстрый метод кодирования

Пусть дан алфавит $A = \{a_1, \dots, a_m\}$, $m \geq 2$, и пусть A^n — множество всех слов длины n , $n \geq 1$. Каждое подмножество $S \subset A^n$ мы будем называть источником, или комбинаторным источником. Эта терминология восходит к теории информации. Источник порождает сообщение, которое может быть любым словом из S . Сообщение необходимо закодировать и затем декодировать. Эти операции осуществляют кодер и декодер. Каждый нумерационный код φ задается двумя отображениями: кодированием $\varphi^c: S \rightarrow \{0, 1\}^t$, где $t = \lceil \log |S| \rceil$, и декодированием $\varphi^d: \varphi^c(S) \rightarrow S$. Причем кодирование неискажающее, то есть $\varphi^d(\varphi^c(x)) = x$ для каждого слова $x \in S$.

В дальнейшем для упрощения обозначений будем отождествлять каждое двоичное слово с числом из отрезка $[0, 1]$, двоичное разложение которого совпадает с этим словом. Например, $01010 = 5/16$. Предположим также, что буквы алфавита A — целые числа, то есть $A = \{0, 1, \dots, m-1\}$, и зададим на A^n лексикографический порядок.

Для описания предлагаемого метода удобно сначала рассмотреть нумерационный код, предложенный в [8]. Для любого слова $x_1 \dots x_k \in A^k$, $k \geq 1$, обозначим через $N_S(x_1 \dots x_k)$ число элементов множества слов из S , начинающихся с $x_1 \dots x_k$. В [8] (см. также [2]), предлагается кодировать каждое слово $x_1 x_2 \dots x_n$ в соответствии с формулой

$$\text{code}(x_1 x_2 \dots x_n) = \sum_{i=1}^n \sum_{a < x_i} N_S(x_1 x_2 \dots x_{i-1} a). \quad (6)$$

Все вычисления здесь проводятся в двоичной системе счисления и полученный результат рассматривается как двоичное слово. Если его длина меньше $\lceil \log |S| \rceil$ бит, то слева дописываются нули.

Важно отметить, что для многих интересных источников (или комбинаторных задач) равенство (6) дает метод кодирования с неэкспоненциально растущей памятью (при длине слова $n \rightarrow \infty$). Например, в случае нумерации перестановок (6) превращается в (1). Другой важный пример — нумерация двоичных слов с заданным числом единиц. Точнее, пусть n и r — целые числа, $0 \leq r \leq n$, и пусть источник S — это множество слов из $\{0, 1\}^n$, у которых r единиц и $(n-r)$ нулей. Из определения $N_s(x_1 \dots x_k)$ легко находим, что в этом случае

$$N_s(x_1 \dots x_{k-1} 0) = \binom{n-k}{r-x}, \quad (7)$$

где

$$x = \sum_{i=1}^{k-1} x_i,$$

откуда с учетом (6) получаем, что

$$\text{code}(x_1 \dots x_n) = \sum_{k=1}^n x_k \binom{n-k}{r-x} \quad (8)$$

Заметим, что эта формула, дающая метод кодирования и декодирования с неэкспоненциально растущим объемом памяти (при $n \rightarrow \infty$) была предложена в работах [3,4] и [5] при построении эффективных методов сжатия данных.

Пусть, например, $n = 8$, $r = 3$, $x_1 x_2 \dots x_8 = 01000101$. Тогда согласно (8)

$$\text{code}(01000101) = 0 \binom{7}{3} + 1 \binom{6}{3} + 0 \binom{5}{2} + 0 \binom{4}{2} + 0 \binom{3}{2} + 1 \binom{2}{2} + 0 \binom{1}{1} + 1 \binom{0}{1} = 21$$

Для описания предлагаемого метода введем при $k = 2, \dots, n$ величины

$$\begin{aligned} P(x_1) &= \frac{N(x_1)}{|S|}, \\ P(x_k | x_1 \dots x_{k-1}) &= \frac{N(x_1 \dots x_k)}{N(x_1 \dots x_{k-1})}, \\ q(x_1) &= \sum_{a < x_1} P(a), \\ q(x_k | x_1 \dots x_k) &= \sum_{a < x_k} P(a | x_1 \dots x_{k-1}), \end{aligned} \quad (9)$$

Очевидно,

$$\begin{aligned} \sum_{i=1}^n \sum_{a < x_i} N(x_1 \dots x_{i-a} a) &= |S| (q(x_2 | x_1) P(x_1) \\ &\quad + q(x_3 | x_1 x_2) P(x_2 | x_1) P(x_1) \\ &\quad + q(x_4 | x_1 x_2 x_3) P(x_3 | x_1 x_2) P(x_2 | x_1) P(x_1) + \dots), \end{aligned}$$

Отсюда и из (6) получаем, что

$$\text{code}(x_1 \dots x_n) = |S| (q(x_1) + q(x_2 | x_1) P(x_1) + q(x_3 | x_1 x_2) P(x_2 | x_1) P(x_1) + \dots). \quad (10)$$

Применим теперь принцип “разделяй и властвуй”, или подходящую расстановку скобок в (10). Для этого определим величины, совпадающие с выражениями, возникающими при расстановке скобок.

Для того, чтобы упростить описание, предположим, что $\log n$ — целое число, это не повлияет на приводимые ниже асимптотические оценки. Положим

$$\nu = \log n, \quad (11)$$

$$\begin{aligned} \rho_1^0 &= P(x_1), \quad \lambda_1^0 = q(x_1), \\ \rho_k^0 &= P(x_k | x_1) \dots x_{k-1}), \\ \lambda_k^0 &= q(x_k | x_1) \dots x_{k-1}), \quad k = 2, \dots, n, \end{aligned} \quad (12)$$

$$\begin{aligned} \rho_k^s &= \rho_{2k-1}^{s-1} \rho_{2k}^{s-1}, \\ \lambda_k^s &= \lambda_{2k-1}^{s-1} \lambda_{2k}^{s-1}, \quad s = 1, \dots, \nu, \quad k = 1, \dots, n/2^s. \end{aligned} \quad (13)$$

В группах формул под номерами (12) и (13) все вычисления проводятся над рациональными числами и каждое ρ_k^s и λ_k^s представлено двумя целыми числами, числителем и знаменателем рациональной дроби. Все операции умножения осуществляются по методу Шенхаге-Штрассена.

Из (12) и (13) легко видеть, что

$$\begin{aligned} \lambda_1^\nu &= (q(x_1) + q(x_2 | x_1) P(x_1)) \\ &\quad + ((q(x_3 | x_1 x_2) + q(x_4 | x_1 \dots x_3) P(x_3 | x_1 x_2)) (P(x_2 | x_1) P(x_1)) + \dots \end{aligned}$$

Отсюда с учетом (10) следует, что

$$\lambda_1^v |S| = \text{code}(x_1 \dots x_n). \quad (14)$$

Мы определим код $\alpha^c(x_1 \dots x_n)$ равенством (14), или, в общем случае, равенством

$$\alpha^c(x_1 \dots x_n) = |S| \lambda_1^{\lceil \log n \rceil}. \quad (15)$$

Здесь величина $\lambda_1^{\lceil \log n \rceil}$ вычисляется в соответствии с (12), (13) и все операции осуществляются над рациональными числами. Равенство (14) гарантирует, что после последнего умножения величина, стоящая справа в (15), будет целым числом, совпадающим с $\text{code}(x_1 \dots x_n)$ в (6).

Рассмотрим пример, иллюстрирующий описанный метод. Пусть источник — множество бинарных слов длины $n = 8$ с заданным числом единиц $r = 3$ и требуется закодировать рассмотренное в предыдущем примере слово $x_1 \dots x_8 = 01000101$. Нам понадобятся два простых свойства биномиальных коэффициентов:

$$\binom{t}{p} = \binom{t-1}{p-1} \frac{t}{p}, \quad \binom{t}{p} = \binom{t-1}{p} \frac{t}{t-p},$$

которые можно записать в виде одного равенства

$$\binom{t-1}{p-\Delta} \bigg/ \binom{t}{p} = \frac{p\Delta + (t-p)(1-\Delta)}{t},$$

где Δ принимает значения 0 и 1. Отсюда с учетом (7), (8) и (9) получаем, что при $t = 1, \dots, n$

$$\begin{aligned} P(x_t | x_1 \dots x_{t-1}) \\ = \frac{1}{n-t+1} \left(x_t \left(k - \sum_{j=1}^{t-i} x_j \right) + (1-x_t) \left(n-t+1-k + \sum_{j=1}^{t-i} x_j \right) \right), \end{aligned} \quad (16)$$

$$q(x_t | x_1 \dots x_{t-1}) = \frac{1}{n-t+1} \left(x_t \left(n-t+1-k + \sum_{j=1}^{t-i} x_j \right) \right). \quad (17)$$

Следовательно,

$$\begin{aligned} P(x_1) &= P(0) \\ &= \frac{0(3-0) + (1-0)(8-1+1-(3-0))}{8-1+1} = \frac{5}{8}, \\ P(x_2 | x_1) &= P(1 | 0) \\ &= \frac{1(3-0) + (1-1)(8-1+1-(3-0))}{8-2+1} = \frac{3}{7}, \\ P(x_3 | x_1 x_2) &= P(0 | 01) \\ &= \frac{0(3-1) + (1-0)(8-3+1-(3-1))}{8-3+1} = \frac{4}{6}, \end{aligned}$$

$$P(x_4 | x_1 x_2 x_3) = P(0 | 010) \\ = \frac{0(3-1) + (1-0)(8-4+1-(3-1))}{8-4+1} = \frac{3}{5},$$

$$P(x_5 | x_1 x_2 x_3 x_4) = P(0 | 0100) \\ = \frac{0(3-1) + (1-0)(8-5+1-(3-1))}{8-5+1} = \frac{2}{4},$$

$$P(x_6 | x_1 \dots x_5) = P(1 | 01000) \\ = \frac{1(3-1) + 0(8-6+1-(3-1))}{8-6+1} = \frac{2}{3},$$

$$P(x_7 | x_1 \dots x_6) = P(0 | 010001) \\ = \frac{0(3-2) + (1-0)(8-7+1-(3-2))}{8-7+1} = \frac{1}{2},$$

$$P(x_8 | x_1 \dots x_7) = P(1 | 0100010) \\ = \frac{1(3-2) + (1-1)(8-8+1-(3-2))}{8-8+1} = \frac{1}{1},$$

$$q(x_1) = q(0) = 0,$$

$$q = (x_2 | x_1) = q(1 | 0) \\ = \frac{1(8-2+1-(3-0))}{8-2+1} = \frac{4}{7},$$

$$q(x_3 | x_1 x_2) = q(x_4 | x_1 \dots x_3) = q(x_5 | x_1 \dots x_4) = 0, \\ q(x_6 | x_1 \dots x_5) = q(1 | 01000)$$

$$= \frac{1(8-6+1-(3-1))}{8-6+1} = \frac{1}{3},$$

$$q(x_7 | x_1 \dots x_6) = q(0 | 010001) = 0,$$

$$q(x_8 | x_1 \dots x_7) = q(1 | 0100010) \\ = \frac{1(8-8+1-(3-2))}{8-8+1} = 0.$$

Из (12), (13) получаем, что

$$\begin{aligned} \rho_1^0 &= \frac{5}{8}, \quad \rho_2^0 = \frac{3}{7}, \quad \rho_3^0 = \frac{4}{6}, \quad \rho_4^0 = \frac{3}{5}, \\ \rho_5^0 &= \frac{2}{4}, \quad \rho_6^0 = \frac{2}{3}, \quad \rho_7^0 = \frac{1}{2}, \quad \rho_8^0 = 1, \\ \lambda_1^0 &= 0, \quad \lambda_2^0 = \frac{4}{7}, \quad \lambda_3^0 = 0, \quad \lambda_4^0 = 0, \\ \lambda_5^0 &= 0, \quad \lambda_6^0 = \frac{1}{6}, \quad \lambda_7^0 = 0, \quad \lambda_8^0 = 0, \\ \rho_1^1 &= \frac{5}{8} \frac{3}{7}, \quad \rho_2^1 = \frac{4}{6} \frac{3}{5}, \quad \rho_3^1 = \frac{2}{4} \frac{2}{3}, \quad \rho_4^1 = \frac{1}{2} \frac{1}{2} \cdot 1, \\ \lambda_1^1 &= 0 + \frac{5}{8} \frac{4}{7}, \quad \lambda_2^1 = 0 + 0, \\ \lambda_3^1 &= 0 + \frac{1}{3} \frac{2}{4}, \quad \lambda_4^1 = 0 + 0, \end{aligned}$$

$$\begin{aligned}\rho_1^2 &= \frac{5}{8} \frac{3}{7} \frac{4}{3} \frac{3}{5}, & \rho_2^2 &= \frac{2}{4} \frac{2}{3} \frac{1}{2} \frac{1}{1}, \\ \lambda_1^2 &= \frac{5}{8} \frac{4}{7} + \frac{5}{8} \frac{3}{7} \cdot 0 = \frac{5}{8} \frac{4}{7}, \\ \lambda_2^2 &= \frac{1}{3} \frac{2}{4} + \frac{2}{4} \frac{2}{3} \cdot 0 = \frac{1}{3} \frac{2}{3}, \\ \lambda_1^3 &= \frac{5}{8} \frac{4}{7} + \frac{5}{8} \frac{3}{7} \frac{4}{3} \frac{1}{2} \frac{2}{3} = \frac{20}{56} + \frac{1}{56} = \frac{21}{56}.\end{aligned}$$

Наконец, из (15) находим кодовое слово, или номер

$$\lambda^c(01000101) = 56(21/56) = 21.$$

Здесь для наглядности все вычисления проведены в десятичной системе счисления вместо двоичной.

Для исследования свойств предлагаемого метода мы определим несколько величин. Пусть, как и ранее, дан алфавит A , целое число n , $n \geq 1$, и множество сообщений $S \subset A^n$ (или источник сообщений). Обозначим через $t_{\max}$ максимальное время вычисления рациональных дробей $N(x_1 \dots x_{t+1})/N(x_1 \dots x_t)$, где максимум берется по всем $x_1 \dots x_n \in S$ и $t = 0, 1, \dots, n-1$. Обозначим через $m_{\max}$ максимальный объем памяти кодера (в битах), используемой для вычисления дробей $N(x_1 \dots x_{t+1})/N(x_1 \dots x_t)$, $x_1 \dots x_n \in S$, $t = 0, 1, \dots, n-1$. Наконец, обозначим через $q_{\max}$ максимальный знаменатель дробей $N(x_1 \dots x_{t+1})/N(x_1 \dots x_t)$, $x_1 \dots x_n \in S$, $t = 0, 1, \dots, n-1$.

Свойства предложенного метода характеризует следующая теорема.

Теорема 1. Пусть даны алфавит A , целое число $n \geq 1$ и $S \subset A^n$. Тогда метод α^c обладает следующими свойствами:

- (1) код α^c осуществляет взаимно однозначное отображение слов из S в множество целых чисел $0, 1, \dots, |S| - 1$;
- (2) скорость кодирования (то есть время кодирования на букву, измеряемое числом операций над однобитовыми словами) равна

$$t_{\max} + O(\log n \log q_{\max} \log(nq_{\max}) \log \log(nq_{\max}))$$

- (3) объем памяти в битах, используемый при кодировании, не превосходит

$$m_{\max} + O(n \log q_{\max} \log n).$$

Доказательство. Для упрощения обозначений мы предположим, что не только $\nu = \log n$, но и $\log q_{\max}$ — целые числа. Из доказательства будет видно, что это не влияет на асимптотику.

Первое утверждение теоремы следует из тождества (10) и корректности метода нумерации, базирующегося на (6).

Перейдем к доказательству второго утверждения. Из определения $q_{\max}$ и (9) мы видим, что для записи каждой дроби P или q достаточно $2 \log q_{\max}$ бит, $\log q_{\max}$ для числителя и столько же для знаменателя. Поэтому вычисление одной величины ρ_k^1 или λ_k^1 в соответствии с (13) при любом $k = 1, 2, \dots, n/2$ потребует одной операции умножения чисел, длина которых не превосходит $\log q_{\max}$ бит, а общее число

операций умножения при вычислении всех $\lambda_k^1, \rho_k^1, k = 1, \dots, n/2$, равно $5n/2$. При вычислении λ_k^1 мы используем обычное равенство

$$\frac{a}{b} + \frac{c}{d} = \frac{(ad + bc)}{bd},$$

требующее три умножения. В результате будут получены дроби, у которых для записи числителя и знаменателя требуется не более $2 \log n$ бит. Аналогично, для вычисления $\lambda_k^2, \rho_k^2, k = 1, \dots, n/4$, требуется $5n/4$ операций умножения над числами длины $2 \log n$ бит и так далее, для вычисления $\lambda_k^i, \rho_k^i, k = 1, \dots, n/2^i$, требуется $5n/2^i$ операций умножения над числами длины $2^{i-1} \log n$ бит. Из оценки времени умножения чисел по методу Шенхаге–Штрассена (4) получаем, что общее время вычислений λ_k^i, ρ_k^i по формулам (13) требует

$$\begin{aligned} & \frac{5n}{2} O(\log q_{\max} \log \log q_{\max} \log \log \log q_{\max}) \\ & + \frac{5n}{4} O(2 \log q_{\max} \log(2 \log(2 q_{\max})) \log \log(2 \log q_{\max}) + \dots \\ & + \frac{5n}{2^i} O(2^i \log q_{\max} \log(2^i \log q_{\max}) \log \log(2^i \log q_{\max}) + \dots \\ & + 5O(n \log q_{\max} \log(n \log q_{\max}) \log \log(n q_{\max})). \end{aligned}$$

В этой сумме $\log n$ слагаемых, каждое из которых есть

$$O(n \log q_{\max} \log(n \log q_{\max} \log \log(n \log q_{\max}))),$$

следовательно, сумма есть

$$O(n \log n \log q_{\max} \log(n \log q_{\max}) \log \log(n \log q_{\max}))$$

операций. Отсюда число операций на букву слова $x_1 \dots x_n \in S$ равно

$$O(\log n \log q_{\max} \log(n \log q_{\max}) \log \log(n \log q_{\max})). \quad (18)$$

Это скорость вычисления величины $\lambda_1^{\log n}$. В силу (15) для вычисления кода (или номера) требуется еще произведение $\lambda_1^{\log n} |S|$. Так как $S \subset A^n$, длина обоих сомножителей не превосходит $n \lceil \log |A| \rceil$ бит и эта операция умножения не меняет оценку (18). Следовательно, из (18) и определения $t_{\max}$ мы получаем второе утверждение теоремы.

Оценим необходимый для осуществления кодирования объем памяти. Заметим, что при вычислении λ_k^i и $\rho_k^i, i > 1, k = 1, \dots, n/2^i$, используются только величины λ_k^{i-1} и $\rho_k^{i-1}, k = 1, 2, \dots, n/2^{i-1}$. Поэтому при кодировании достаточно иметь память для хранения двух наборов $\{\lambda_k^i, \rho_k^i\}, n = 1, \dots, n/2^i$, и $\{\lambda_k^{i+1}, \rho_k^{i+1}\}, k = 1, \dots, n/2^{i+1}, k = 1, \dots, \log n - 1$. Длина каждой дроби λ_k^i и ρ_k^i не превосходит $2^{i+1} \log q_{\max}$ бит (по $2^i \log q_{\max}$ бит на числитель и знаменатель). Отсюда и из определения $m_{\max}$ получаем третье утверждение теоремы.

Замечание 1. Можно использовать в качестве кодового слова $\lceil \log |S| \rceil$ бит величины $\lambda_1^{\log n}$ вместо числа α^c (см.(15)), что позволяет не проводить одну операцию умножения. Однако это упрощение не влияет на асимптотическую оценку скорости кодирования.

Следствие 1. Если метод α^c применяется для нумерации множества двоичных слов длины n с заданным числом единиц, то скорость кодирования есть $O(\log^3 n \log \log n)$ операций на букву при объеме памяти $O(n \log^2 n)$.

Доказательство сразу следует из теоремы 1 и неравенства $q_{\max} \leq n$ (см. (16), (17)).

3. Быстрый метод декодирования

Опишем метод декодирования в два этапа: сначала общую схему, не оценивая сложность вычислений, затем полный алгоритм, учитывающий длину слов. Отметим сразу, что общее уменьшение времени декодирования по сравнению с известными методами достигается за счет проведения большей части вычислений над короткими словами. В этом случае в качестве таких слов используются старшие знаки числителей и знаменателей рациональных дробей, встречавшихся ранее при кодировании.

Пусть $A = \{0, 1, \dots, m-1\}$, $m \geq 2$, и $S \subset A^n$, $n \geq 1$, и пусть $\bar{x} = x_1 \dots x_n$ — произвольное слово из S , а $\bar{y} = \alpha^c(\bar{x})$ — кодовое слово $\bar{x}$, вычисленное выше описанным методом α^c . Для декодирования $\bar{y}$ рассмотрим рациональное $y_1 = \bar{y}/|S|$, представленное числителем и знаменателем (здесь числитель — целое число, двоичная запись которого совпадает с $\bar{y}$). После этого найдем i_1 , удовлетворяющее неравенствам

$$\lambda_{i_1}^0 \leq \bar{y}/|S| < \lambda_{i_1+1} \quad (19)$$

(будем считать, что $\lambda_m^0 = 1$ по определению). Существование и единственность такого i_1 следуют из описания λ^c (см. (12)–(15)). Выполнение (19) означает, что первая буква $x_1 = i_1$. Сразу оценим количество операций, необходимых для нахождения i_1 , удовлетворяющего (19). Для определения i_1 мы используем широко известный метод деления пополам: сначала вычисляем $\lambda_{\lceil m/2 \rceil}$. Если

$$\bar{y}/|S| < \lambda_{\lceil m/2 \rceil}, \quad (20)$$

значит, $i_1 < \lceil m/2 \rceil$, в противном случае $i_1 \geq \lceil m/2 \rceil$. После этого в случае выполнения (20) находим, к какой половине интервала $[0, \lceil m/2 \rceil)$ принадлежит i_1 , вычисляя $\lambda_{\lceil m/4 \rceil}^0$ и проверяя неравенство

$$\bar{y}/|S| < \lambda_{\lceil m/4 \rceil}^0.$$

Если (20) не выполнено, то, аналогично, делим пополам отрезок $[\lceil m/2 \rceil, m)$, вычисляя $\lambda_{\lceil 3m/4 \rceil}^0$, и так далее. Понятно, что при каждом сравнении отрезок, содержащий i_1 , уменьшается вдвое и для нахождения i_1 потребуется не более, чем $\lceil \log m \rceil$ операций сравнения и вычислений λ_s^0 .

После нахождения первой буквы мы проводим операцию, обратную кодированию (13), то есть вычисляем

$$\bar{Z} = (\bar{y}/|S| - \lambda_{i_1}^0)/\rho_{i_1}^0, \quad (21)$$

где $\bar{Z}$ — рациональное число, представленное числителем и знаменателем. Затем, используя деление пополам, находим i_2 , удовлетворяющее неравенствам

$$\lambda_{i_2}^0 \leq \bar{Z} < \lambda_{i_2+1} \quad (22)$$

Это означает, что вторая буква декодирована и $x_2 = i_2$.

Можно было бы продолжить этот путь и последовательно находить $x_3, x_4, x_5, \dots$, но такая схема не может быть сделана быстрой. Поэтому предлагается другая схема, базирующаяся на принципе "разделяй и властвуй". После декодирования первых букв x_1, x_2 мы вычисляем λ_1^1 в соответствии с (13) и затем находим

$$\bar{y}_2 = (\bar{y} - \lambda_1^1) / \rho_1^1. \quad (23)$$

Используя $\bar{y}_2$, находим буквы x_3, x_4 точно так же, как находим x_1, x_2 по слову $\bar{y}$. Затем вычисляем λ_2^1 , используя декодированные буквы x_3, x_4 и λ_1^2 и ρ_1^2 по парам букв x_1, x_2 и x_3, x_4 (см. (13)). Затем вычисляем дробь

$$(\bar{y} - \lambda_1^2) / \rho_1^2 \quad (24)$$

и по ней находим сначала x_5, x_6 , затем x_7, x_8 , вычисляем λ_1^3 и ρ_1^3 и так далее.

Основная идея алгоритма — проводить вычисления (21)–(24) и аналогичные им, используя только старшие цифры, то есть работая с короткими числами. Точнее, использовать две оценки величин λ_j^i , верхнюю и нижнюю, получаемые по старшим разрядам точных величин.

Для точного описания метода декодирования введем несколько вспомогательных функций. Пусть p/q — рациональное число, представленное как пара целых положительных чисел $p, q, p \leq q$, и пусть $t \geq 1$ — целое число. Определим функции $\varphi_t^+(p/q)$ и $\varphi_t^-(p/q)$ следующим образом. Положим $l = \lceil \log q \rceil$ и пусть $(q_l q_{l-1} \dots q_0)$ и $(p_l p_{l-1} \dots p_0)$ — двоичные представления чисел q и p . Тогда

$$\varphi_t^+(p/q) = \left(\sum_{i=l-t}^l p_i 2^i + 2^{l-t} \right) / \sum_{i=l-t}^l q_i 2^i,$$

$$\varphi_t^-(p/q) = \sum_{i=l-t}^l p_i 2^i / \sum_{i=l-t}^l q_i 2^i + 2^{l-t}$$

(отметим, что если 2^{l-t} — дробная величина, то есть $l-t < 0$, то, по определению, мы домножаем числитель и знаменатель на $2^{-(l-t)}$ для того, чтобы числитель и знаменатель были целыми числами).

Например,

$$\varphi_4^+(5/17) = 3/8, \quad \varphi_4^-(5/17) = 2/9.$$

В дальнейшем нам будет полезна следующая простая лемма.

Лемма 1. Пусть p, q, t — целые положительные числа и $0 < p \leq q$ и $t > 2$. Тогда

$$0 \leq \varphi_t^+(p/q) - p/q < 2^{2-t}, \quad (25)$$

$$0 \leq p/q - \varphi_t^-(p/q) < 2^{2-t}. \quad (26)$$

Доказательство. Легко видеть, что

$$\frac{1}{1-x} < 1 + 2x, \quad \frac{1}{1+x} > x \quad (27)$$

при $x < 1/2$. Эти неравенства сразу следуют из формулы для суммы геометрической прогрессии.

Оценка (25) следует из цепочки неравенств

$$\begin{aligned}\varphi_t^+(p/q) &\leq \frac{p + 2^{l-t}}{q - 2^{l-t}} < \frac{p}{q} \left(1 + \frac{2^{l-t}}{p}\right) \left(1 + \frac{2^{l-t}}{q}\right) \\ &= \frac{p}{q} + \frac{2^{l-t}}{q} + \frac{2 \cdot 2^{l-t}}{q} + \frac{2^{l-t}}{q} \frac{2 \cdot 2^{l-t}}{q} \\ &< \frac{p}{q} + 2^{-t} + 2 \cdot 2^{-t} + 2^{-2t} < \frac{p}{q} + 4 \cdot 2^{-t}\end{aligned}$$

(здесь мы использовали (27) и очевидное соотношение $2^l < q$). Неравенство (26) получается точно так же. Лемма доказана.

Продолжим описание метода декодирования. Пусть, как прежде, $q_{\max}$ равно максимальному знаменателю рациональных чисел

$$\frac{N(x_1 \dots x_{t+1})}{N(x_1 \dots x_t)}, \quad x_1 \dots x_n \in S, \quad t = 1, 2, \dots, n-1.$$

Отсюда и из (12) и (13) сразу получаем, что знаменатели рациональных дробей λ_i^s и ρ_i^s не превосходят $q_{\max}^{2^i}$ при всех $i = 1, 2, \dots, n/2^s$ и, следовательно,

$$\rho_i^s \geq 1/q_{\max}^{2^s}. \quad (28)$$

Определим также величину

$$h = \lceil \log q_{\max} \rceil + 3. \quad (29)$$

Пусть $t = \lceil \log |S| \rceil$ — длина двоичного представления кодовых слов из S и пусть $\bar{y} = y_1 y_2 \dots y_n$ — кодовое слово, которое должно быть декодировано.

Мы опишем сначала формальный алгоритм, используя рекурсию. Заметим, что это обычный способ описания методов, базирующихся на принципе “разделяй и властвуй”.

Сначала рассмотрим декодирование для случая длины слова $n = 2$. Для каждой величины λ_j^i из (12), (13) мы определим верхнюю и нижнюю оценку $\lambda^+(i, j)$ и $\lambda^-(i, j)$ соответственно.

Для описания алгоритма удобно ввести такие оценки и для величины λ_1^1 , хотя она и точно известна при $n = 2$:

$$\lambda^+(1, 1) = \bar{y}/|S|, \quad \lambda^-(1, 1) = \bar{y}/|S| \quad (30)$$

Определим также

$$\lambda^+(0, 1) = \varphi_h^+(\lambda^+(1, 1)), \quad \lambda^-(0, 1) = \varphi_h^-(\lambda^-(1, 1)) \quad (31)$$

После этого находим i_1 , удовлетворяющее двум неравенствам

$$q(i_1) \leq \lambda^+(0, 1), \quad q(i_1 + 1) > \lambda^-(0, 1)$$

(это аналог (19); $x_1 = i_1$, но здесь приходится использовать две оценки вместо одной λ_1^0 , так как ее точное значение неизвестно). Отметим, что число операций

сравнения не превосходит $2\lceil \log m \rceil$ и каждое сравнение проводится над числами длины $h + 1$ бит.

Затем вычисляем оценки для λ_0^0

$$\lambda^+(0, 2) = \varphi_h^+((\lambda^+(1, 1) - \lambda_1^0)/\rho_1^0),$$

$$\lambda^-(0, 2) = \varphi_h^-((\lambda^-(1, 1) - \lambda_1^0)/\rho_1^0)$$

и находим i_2 , удовлетворяющее неравенствам

$$q(i_2) \leq \lambda^+(0, 2), \quad q(i_2 + 1) > \lambda^-(0, 2).$$

Это означает, что находится вторая буква i_2 , то есть $x_2 = i_2$.

Пусть теперь $n > 2$. Для реализации принципа “разделяй и властвуй” определим

$$\begin{aligned} \lambda^+(\log n, 1) &= \varphi_{nh}^+(\bar{y}/|S|), \\ \lambda^-(\log n, 1) &= \varphi_{nh}^-(\bar{y}/|S|), \end{aligned} \quad (32)$$

$$\begin{aligned} \lambda^+(\log n - 1, 1) &= \varphi_{nh/2}^+(\lambda^+(\log n, 1)), \\ \lambda^-(\log n - 1, 1) &= \varphi_{nh/2}^-(\lambda^-(\log n, 1)). \end{aligned} \quad (33)$$

Затем находим буквы $x_1, x_2, \dots, x_{n/2}$ по $\lambda^+(\log n - 1, 1)$ и $\lambda^-(\log n - 1, 1)$ и по этим буквам вычисляем $\lambda_1^{\log n - 1}$ в соответствии с алгоритмом кодирования (см. (12), (13)). Используя эту величину, находим

$$\begin{aligned} \lambda^+(\log n - 1, 2) &= \varphi_{nh/2}^+((\lambda^+(\log n, 1) - \lambda_1^{\log n - 1})/\rho_1^{\log n - 1}), \\ \lambda^-(\log n - 1, 2) &= \varphi_{nh/2}^-((\lambda^-(\log n, 1) - \lambda_1^{\log n - 1})/\rho_1^{\log n - 1}) \end{aligned} \quad (34)$$

и по этой паре оценок находим буквы $x_{n/2+1}, x_{n/2+2}, \dots, x_{n-1}, x_n$.

Таким образом, (33) и (34) сводят задачу декодирования слова $x_1 \dots x_n$ к задаче декодирования двух слов половинной длины.

Рассмотрим работу описанного метода декодирования на прежнем примере бинарных слов с заданным числом единиц. Пусть $A = \{0, 1\}$, $n = 8$, S — множество слов с 3 единицами и 5 нулями. Пусть дан номер (или код) $\alpha^c(x) = 21$ слова из S . Надо его декодировать, то есть по этому номеру восстановить $x \in S$. Ответ $\alpha^c(01000101) = 21$ известен из предыдущего параграфа.

Мы используем описанный метод декодирования α^d . В нашем случае $q = 8$ (см. (16), (17) в примере кодирования). В соответствии с (29) $h = 3 + 3 = 6$. В примере все преобразования будут проводиться над числами в двоичной системе счисления, но для удобства восприятия результаты будут представлены в десятичной системе.

Согласно (32), первый шаг при декодировании — вычисление $\lambda^+(\log n, 1)$ и $\lambda^-(\log n, 1)$:

$$\lambda^+(3, 1) = \varphi_{48}^+(21/56) = 21 \cdot 2^{42} + 1/56 \cdot 2^{42},$$

$$\lambda^-(3, 1) = \varphi_{48}^-(21/56) = 21 \cdot 2^{42}/56 \cdot 2^{42} + 1.$$

Затем по формулам (33) находим

$$\lambda^+(2, 1) = \varphi_{21}^+(21 \cdot 2^{42} + 1/56 \cdot 2^{42}) = 21 \cdot 2^{23} + 1/56 \cdot 2^{23},$$

$$\lambda^-(2, 1) = 21 \cdot 2^{23}/56 \cdot 2^{23} + 1.$$

По этим величинам декодер должен найти буквы x_1, x_2, x_3, x_4 . В соответствии с алгоритмом, находим сначала

$$\begin{aligned}\lambda^+(1, 1) &= \varphi_{12}^+(21 \cdot 2^{23} + 1/56 \cdot 2^{23}) = 21 \cdot 2^7 + 1/56 \cdot 2^7, \\ \lambda^-(1, 1) &= 21 \cdot 2^7 / 56 \cdot 2^7 + 1.\end{aligned}$$

Эта пара кодирует x_1, x_2 . Наконец, находим числа

$$\begin{aligned}\lambda^+(0, 1) &= \varphi_6^+(21 \cdot 2^7 + 1/56 \cdot 2^7) = 22/56, \\ \lambda^-(0, 1) &= \varphi_6^-(21 \cdot 2^7 + 1/56 \cdot 2^7) = 21/57,\end{aligned}$$

кодирующие x_1 . Для рассматриваемого источника $q(0) = 0, q(1) = 5/8$ (см. (16), (17)) и в соответствии с алгоритмом декодирования находим i_1 , для которого

$$q(i_1) \leq \lambda^+(0, 1) = 22/56, \quad 21/57 = \lambda^-(0, 1) < q(i_1 + 1).$$

Очевидно, обоим неравенствам удовлетворяет только $i_1 = 0$. Следовательно, $x_1 = 0$ и $\lambda_1^0 = 5/8$. Далее по формулам (34) вычисляем

$$\begin{aligned}\lambda^+(0, 2) &= \varphi_6^+((21 \cdot 2^7 + 1/56 \cdot 2^7 - 0)/(5/8)) = 169/280, \\ \lambda^-(0, 2) &= 168/281.\end{aligned}$$

Из (16) находим, что $q(0/0) = 0, q(1/0) = 4/7$. Так как

$$\lambda^+(0, 2) = 169/280 \geq 4/7, \quad \lambda^-(0, 2) = 168/281 < 1,$$

получаем, что $i_2 = 1$ и $x_2 = 1$. Далее нужно вычислить $\lambda_1^1 = 5/14$ и $\rho_1^1 = 15/56$, найти $\lambda^+(1, 2)$ в соответствии с (34), определить $x_3 = 0, x_4 = 0$ и так далее.

Оценим теперь сложность предложенного алгоритма. Сначала отметим, что одной операции умножения при вычислении λ_j^i в (13) соответствует одна операция деления, причем при вычислении оценок $\lambda^+(i, j), \lambda^-(i, j)$ длина числителя и знаменателя делимого не превосходит $2^{i+1}h$ бит, а делителя $2^i \lceil \log q_{\max} \rceil$ бит. Из определения h в (29) следует, что длина делимого и делителя пропорциональна длине сомножителей, используемых при вычислении λ_j^i . Следовательно, и время вычисления $\lambda^+(i, j)$ и $\lambda^-(i, j)$ при использовании методов деления и умножения Шенхаге-Штрассена будет пропорционально времени вычисления λ_j^i (см.(4)). Кроме операций деления декодер вычисляет те же величины λ_j^i и ρ_j^i , которые используются при кодировании и для этого используется столько же времени. Следовательно, время, затрачиваемое на выполнение операций, необходимых на вычисление оценок

$$\lambda^+(i, j), \quad \lambda^-(i, j), \quad i = 0, \dots, \log n, \quad j = 1, \dots, n/2^i$$

пропорционально времени кодирования

$$nt_{\max} + O(n \log n \log q_{\max} \log(nq_{\max}) \log \log(nq_{\max})). \quad (35)$$

Кроме этого надо учесть время, затрачиваемое на определение букв $x_1 = i_1, x_2 = i_2$ методом деления пополам. Всего букв n и, как показано выше, для нахождения одной буквы требуется не более $2 \lceil \log m \rceil$ сравнений слов, длина каждого из

которых не превосходит $h + 1$ бит. Отсюда получаем, что общее число операций над однокбитовыми словами, затрачиваемых на этом этапе, есть

$$O(n \log m \log q_{\max}). \quad (36)$$

Из (35) и (36) находим, что общее время декодирования равно

$$nt_{\max} + O(n \log q_{\max} (\log m + \log n \log(nq_{\max}) \log \log(nq_{\max}))),$$

а скорость (то есть время на букву) дается выражением

$$t_{\max} + O(\log q_{\max} (\log m + \log n \log(nq_{\max}) \log \log(nq_{\max}))). \quad (37)$$

Остановимся теперь кратко на оценке объема используемой при декодировании памяти. Из (32) и (33) видно, что декодирование левой половины $x_1 \dots x_{n/2}$ и правой половины $x_{n/2}, \dots, x_n$ слова проводится независимо и может осуществляться при использовании одной и той же памяти. Рассуждая так же, как при анализе кодирования, легко получить, что объемы памяти, используемой при кодировании и декодировании асимптотически совпадают.

Покажем теперь, что описанный метод кодирования α^d корректен, то есть, что для любого $x \in S$

$$\alpha^d(\alpha^c(x)) = x \quad (38)$$

(x сначала закодировали методом α^c и получили кодовое слово (или номер) $\alpha^c(x)$, затем его декодировали и, естественно, после этого должны получить исходное слово x).

Для доказательства (38) оценим сверху величины

$$|\lambda^+(0, j) - q(x_j)|, \quad |q(x_j) - \lambda^-(0, j)|,$$

так как j -я буква декодируемого слова при декодировании полагается равной i_j , если

$$q(i_j) \leq \lambda^+(0, j), \quad q(i_j + 1) \geq \lambda^-(0, j). \quad (39)$$

Из (12) следует, что (39) эквивалентно неравенствам

$$\lambda_{i_j}^0 \leq \lambda^+(0, j), \quad \lambda_{i_j}^0 \geq \lambda^-(0, j), \quad (40)$$

которые должны выполняться для единственного i_j в случае $x_j = i_j$ при любых

$$x_1, x_2, \dots, x_{j-1}, x_{j+1}, \dots, x_n$$

при условии, что $x_1 x_2 \dots x_j \dots x_n \in S$. Выполнение этих условий гарантирует единственность и корректность декодирования.

По определению, $q_{\max}$ — максимальный знаменатель рациональных дробей q_i , $i = 0, \dots, m - 1$. Следовательно, при $i \neq k$

$$|q_i - q_k| \geq 1/q_{\max}.$$

В силу (12) отсюда следует, что

$$|\lambda_i^0 - \lambda_k^0| \geq 1/q_{\max}.$$

Из этого неравенства и (40) сразу следует, что единственность и корректность декодирования эквивалентна выполнению неравенств

$$\begin{aligned} 0 &\leq \lambda^+(0, j) - \lambda_j^0 < 1/q_{\max}, \\ 0 &\leq \lambda_j^0 - \lambda^-(0, j) < 1/q_{\max}. \end{aligned} \quad (41)$$

Перейдем к их доказательству, ограничившись оценками для $\lambda^+(0, j) - \lambda_j^0$, так как другая пара неравенств доказывается совершенно аналогично. Оценим величину ошибки $\lambda^+(0, n) - \lambda_n^0$, возникающей при декодировании последней буквы x_n . Из построения оценки будет следовать, что она пригодна и для всех других букв, так как число приближенных операций, проводимых при вычислении x_n , больше, чем для других букв.

Неравенство $\lambda^+(0, j) - \lambda_j^0 \geq 0$ сразу следует из определения операции $\varphi^+(\cdot)$ и $\lambda^+(\cdot)$ в (31), (32), (33).

Обозначим через $\varepsilon(i, j)$ погрешность, возникающую при вычислении величины $\lambda^+(i, j)$ из-за использования функции $\varphi^+(\cdot)$:

$$\varepsilon(i, j) = \lambda^+(i, j) - (\lambda^+(i+1, j/2) - \lambda_{j-1}^i)/\rho_{j-1}^i, \quad (42)$$

$i = 0, \dots, \log n$, $j = n/2^i$. Напомним, что в соответствии с алгоритмом декодирования

$$\lambda^+(i, j) = \varphi_{2^i h}^+((\lambda^+(i+1, j/2) - \lambda_{j-1}^i)/\rho_{j-1}^i), \quad (43)$$

что и объясняет введение $\varepsilon(i, j)$.

Оказывается, что оценка интересующей нас величины $\lambda^+(0, n) - \lambda_n^0$ может быть выражена через величины $\varepsilon(i, j)$. Точнее, справедлива следующая цепочка равенств, соответствующая этапам декодирования от последнего к первому:

$$\begin{aligned} \lambda^+(0, n) &= \varphi_h^+((\lambda^+(1, n/2) - \lambda_{n-1}^0)/\rho_{n-1}^0) \\ &= \varepsilon(0, n) + (\lambda^+(1, n/2) - \lambda_{n-1}^0)/\rho_{n-1}^0 \\ &= \varepsilon(0, n) + (\varphi_{2h}^+((\lambda^+(2, n/4) - \lambda_{n/2-1}^1)/\rho_{n/2-1}^1) - \lambda_{n-1}^0)/\rho_{n-1}^0 \\ &= \varepsilon(0, n) + ((\varepsilon(1, n/2) + (\lambda^+(2, n/4) - \lambda_{n/2-1}^1)/\rho_{n/2-1}^1) - \lambda_{n-1}^0)/\rho_{n-1}^0 \\ &= \varepsilon(0, n) + \varepsilon(1, n/2)/\rho_{n-1}^0 + ((\varphi_{4h}^+(\lambda^+(3, n/8) - \lambda_{n/4-1}^2)/\rho_{n/4-1}^2) - \lambda_{n/2-1}^1)/\rho_{n/2-1}^1 - \lambda_{n-1}^0/\rho_{n-1}^0 \\ &= \varepsilon(0, n) + \varepsilon(1, n/2)/\rho_{n-1}^0 \\ &\quad + \varepsilon(2, n/4)/(\rho_{n-1}^0 \rho_{n/2}^1) + (\dots (\varphi_{8h}^+(\lambda^+(4, n/2^4) - \lambda_{n/8-1}^3)/\rho_{n/8-1}^3) - \lambda_{n/4-1}^2)/\rho_{n/4-1}^2 - \lambda_{n/2-1}^1/\rho_{n/2-1}^1 - \lambda_{n-1}^0/\rho_{n-1}^0 = \dots \\ &= \varepsilon(0, n) + \varepsilon(1, n/2)/\rho_{n-1}^0 + \varepsilon(2, n/4)/(\rho_{n-1}^0 \rho_{n/2}^1) \\ &\quad + \varepsilon(3, n/8)/(\rho_{n-1}^0 \rho_{n/2}^1 \rho_{n/4}^2) + \dots + \varepsilon(\log n, 1)/(\rho_{n-1}^0 \rho_{n/2}^1 \dots \rho_1^{\log n-1}) \\ &\quad + ((\dots (\lambda_1^{\log n} - \lambda_1^{\log n-1})/\rho_1^{\log n-1} - \lambda_3^{\log-2})/\rho_3^{\log n-2} - \lambda_7^{\log n-3})/\rho_7^{\log n-3} - \dots)/\rho_{n-1}^0. \end{aligned}$$

Таким образом, мы получили, что

$$\lambda^+(0, n) = \sum_{i=0}^{\log n} \varepsilon(i, n2^{-i}) / \prod_{j=0}^{i-1} \rho_{n2^{-j}-1}^j + (\dots (\lambda_1^{\log n} - \lambda_1^{\log n-1}) / \rho_1^{\log n-1} - \lambda_3^{\log n-2}) / \rho_3^{\log n-2} - \dots) / \rho_{n-1}^0.$$

Из определения λ_j^i в (13) мы видим, что второе слагаемое в правой части равно λ_n^0 и все равенство можно представить в виде

$$\lambda^+(0, n) = \sum_{i=0}^{\log n} \varepsilon(i, n2^{-i}) / \prod_{j=0}^{i-1} \rho_{n2^{-j}-1}^j + \lambda_n^0.$$

Отсюда

$$\lambda^+(0, n) - \lambda_n^0 = \sum_{i=0}^{\log n} \varepsilon(i, n2^{-i}) / \prod_{j=0}^{i-1} \rho_{n2^{-j}-1}^j. \quad (44)$$

Из этого неравенство и (28) получаем, что

$$\lambda^+(0, n) - \lambda_n^0 \leq \varepsilon(0, n) + \varepsilon(1, n/2)q_{\max} + \varepsilon(2, n/4)q_{\max}^3 + \varepsilon(3, n/8)q_{\max}^7 + \dots + \varepsilon(\log n, 1)q_{\max}^{2^n-1}.$$

Из (42), (43) и леммы получаем неравенство

$$\varepsilon(i, j) < 2^{2-2^i h}.$$

Из двух последних неравенств и определения (29) находим, что

$$\begin{aligned} \lambda^+(0, n) - \lambda_n^0 &< 4(1/(8q_{\max}) + q_{\max}/(8q_{\max})^2 \\ &+ q_{\max}^3/(8q_{\max})^4 + \dots + q_{\max}^{2^i-1}/(8q_{\max})^{2^i} + \dots + q_{\max}^{2^n-1}/(8q_{\max})^{2^n} \\ &< \frac{1}{2q_{\max}} \frac{8}{7} < \frac{1}{q_{\max}}. \end{aligned}$$

Таким образом, показано, что

$$\lambda^+(0, n) - \lambda_n^0 < 1/q_{\max},$$

это завершает доказательство (41), что, как было показано выше, гарантирует корректное декодирование (38).

Сформулируем теперь все полученные свойства метода декодирования α^d .

Теорема 2. Пусть даны алфавит A , целое число $n \geq 1$ и множество (источник сообщений) $S \subset A^n$. Пусть сообщения из S кодируются и декодируются методами α^c и α^d , соответственно. Тогда объем памяти (в битах) и время декодирования одной буквы (в операциях над однобитовыми словами) при использовании α^d , равны, соответственно,

$$\begin{aligned} m_{\max} + O(n \log q_{\max} \log n), \\ t_{\max} + O(\log q_{\max} (\log m + \log n \log(nq_{\max}) \log \log(nq_{\max}))). \end{aligned}$$

где $m_{\max}$ и $t_{\max}$ те же величины, что и в методе кодирования.

Следствие 2. При использовании метода α^d для декодирования множества двоичных слов длины n с заданным числом единиц, скорость декодирования есть $O(\log^3 n \log \log n)$ операций на букву при объеме памяти $O(n \log^2 n)$ бит.

Доказательство прямо следует из теоремы 2 и равенства $q_{\max} = n$.

Список литературы

1. Рейнольд Э., Нивергельт Ю., Део Н. *Комбинаторные алгоритмы. Теория и практика*. Мир, Москва, 1980.
2. Кричевский Р. Е. *Сжатие и поиск информации*. Радио и связь, Москва, 1989.
3. Lunch T. J. Sequence time coding for data compression. *Proc. IEEE* (1966) **54**, 1490–1491.
4. Davisson L. D. Comments on “Sequence time coding for data compression”. *Proc. IEEE* (1966) **54**, 2010.
5. Бабкин В. Ф. Метод универсального кодирования неэкспоненциальной трудоемкости. *Проблемы передачи информации* (1974) **7**, №4, 13–21.
6. Ахо А., Хопкрофт Дж., Ульман Дж. *Построение и анализ вычислительных алгоритмов*. Мир, Москва, 1976.
7. Ryabko B. Ya. Fast and effective coding of information sources. *IEEE Trans. Inform. Theory* (1994) **40**, №1, 52–58.
8. Cover T. M. Enumerative source encoding. *IEEE Trans. Inform. Theory* (1973) **19**, №1, 73–77.

Статья поступила 24.02.1997.