УДК 519.72

Д. С. Ковалев

Новосибирский государственный университет
ул. Пирогова, 2, Новосибирск, 630090, Россия
E-mail: dmitriy.kovalev@gmail.com

ОБОБЩЕНИЕ ИЗВЕСТНЫХ СПОСОБОВ КОДИРОВАНИЯ СТРОК

В статье рассматривается кодирование Хаффмана. Его ключевым элементом является построение бинарного дерева. Дерево задает некоторый способ рекурсивного разбиения множества алфавита для получения префиксных кодов. Широко известно обобщение бинарного дерева на q-арное. В данной статье приводится обобщение кодирования Хаффмана на случай *произвольного* дерева. Описывается новый подход для понимания алгоритмов кодирования семейства Лемпеля – Зива. Предлагается считать, что эти алгоритмы кодируют не саму строку, а некоторое отображение, связанное с ней. При этом строка является неподвижной точкой кодируемого отображения.

Ключевые слова: кодирование Хаффмана, алгоритмы Лемпеля – Зива, энумеративное кодирование.

Введение

Сжатие данных – это процесс *экономного кодирования* информации. Можно выделить два подхода к такому кодированию. В первом случае кодируются непосредственно сами данные. Этот подход имеет широкое распространение, кодирование Хаффмана [1] является одним из его примеров. Второй подход заключается в том, что кодируются не сами данные, а некоторое преобразование, связанное с ними. При этом данные можно восстановить из преобразования и, возможно, некоторой дополнительной информации. Примером такого подхода может служить фрактальное сжатие изображений [2]. Его суть заключается в том, что вместо изображения запоминается система итерируемых функций, для которой исходное изображение является аттрактором (неподвижной точкой).

В данной работе мы рассмотрим оба подхода: обобщим кодирование Хаффмана и покажем, что кодирование Лемпеля – Зива [3; 4] можно рассматривать как кодирование отображения, для которого данные являются неподвижной точкой.

Любую информацию (текст, изображение, звук и т. п.) можно представить в виде строки символов над некоторым алфавитом, поэтому далее будем рассматривать только кодирование строк. Зафиксируем алфавит $A=\{a_1, a_2, \ldots, a_k\}$ из $k>1$ элементов. Множество строк фиксированной длины n обозначим A^n. Длину строки s будем обозначать $|s|$, а ее символ, стоящий на позиции $1 \le i \le |s|$, как $s[i]$.

Пусть $s \in A^n$ – строка, которую нужно закодировать. Будем считать, что статистика встречаемости каждого символа алфавита A в строке s известна: символ a_i встречается n_i раз, при этом $n_1 + n_2 + \ldots + n_k = n$.

Обобщение кодирования Хаффмана

Рассмотрим способ кодирования строки s. Одним из давно известных подходов является энумеративное кодирование [5]. Его суть заключается в следующем. Обозначим $A^n[n_1, n_2, \ldots, n_k]$ все строки $s \in A^n$ с фиксированной статистикой $(n_1, n_2, \ldots, n_k)$. Количество таких строк выражается мультиномиальным коэффициентом

$$\binom{n}{n_1, n_2, \ldots, n_k} = \frac{n!}{n_1! n_2! \ldots n_k!} = N .$$

Следовательно, все строки из $A^n[n_1, n_2, \ldots, n_k]$ можно перенумеровать некоторым способом числами от 0 до $N-1$. В такой ситуации кодирование является получением номера по строке, а декодирование – получением строки по ее номеру. Кодирование является оптимальным в том смысле, что каждому из N различных объектов сопоставляется уникальный номер $0, 1, \ldots, N-1$.

Определим функцию кодирования:

$$\text{index} : A^n[n_1, n_2, \ldots, n_k] \to 0, 1, \ldots, N-1,$$

и декодирования:

$$\text{index}^{-1} : 0, 1, \ldots, N-1 \to A^n[n_1, n_2, \ldots, n_k].$$

Для получения битового представления закодированной строки s достаточно записать $\text{index}(s)$ в двоичной системе счисления.

Если $H_0 = \log_2 N$, то потребуется не более $\lceil H_0 \rceil$ бит для двоичного представления каждой строки из $A^n[n_1, n_2, \ldots, n_k]$. Число H_0 также называется энтропией Хартли [6] множества $A^n[n_1, n_2, \ldots, n_k]$.

Согласно работе [7] оптимальным количеством бит для кодирования строки из $A^n[n_1, n_2, \ldots, n_k]$ является nH, где H – энтропия Шеннона:

$$H - \sum_{i=1}^{k} p_i \log_2 p_i \text{, где } p_i = \frac{n_i}{n} .$$

Число nH будем называть размером по энтропии для строки s. Энумеративное кодирование дает лучший результат (можно потратить меньше бит, чем nH), потому что используется *точная* статистика встречаемости символов, в то время как в работе [7] рассматриваются *вероятности* появления символов. Всегда выполняется

$$\frac{1}{n} H_0 < H .$$

При $n \to \infty$ имеет место следующая сходимость:

$$\frac{1}{n} H_0 = \frac{1}{n} \log_2 \binom{n}{n_1, n_2, \ldots, n_k} \to -\sum_{i=1}^{k} p_i \log_2 p_i = H(p_1, p_2, \ldots, p_k) \text{, где } p_i = \frac{n_i}{n} .$$

Сходимость доказывается непосредственно, если разложить факториал по формуле Стирлинга. Из этого следует, что энумеративное кодирование также позволяет закодировать строку не хуже размера по энтропии.

Функцию index можно задать таблично. Нужно построить таблицу, где в первой колонке будет находиться строка, а во второй – ее номер. Однако такой способ является неэффективным для реального использования на вычислительной машине. Таблица для хранения требует количество памяти пропорциональное N, которое является огромным даже при малых длинах строк n.

Рассмотрим эффективный способ построения функции index, в котором номер строки s строится только с использованием символов самой строки s. В дальнейшем этот способ будет обобщен и показана его связь с кодированием Хаффмана.

Закодируем позиции всех символов a_1, которые встречаются в строке s. Для этого сформируем битовую строку s_1 длины n, где единица стоит на месте символа a_1, а остальные нули:

$$s_1[i] = \begin{cases} 1, \text{если } s[i] = a_1 \\ 0, \text{если } s[i] \neq a_1 \end{cases} .$$

Строка s_1 содержит в точности n_1 единиц. После того, как позиции всех символов a_1 известны, аналогичным образом формируем строку s_2, чтобы запомнить позиции символов

a_2. Длина строки s_2 будет $n-n_1$, потому что позиции a_1 уже известны. Будем продолжать формировать строки s_i для остальных символов алфавита $a_3,\ldots,a_k$.

Всего придется изготовить $k-1$ строк, последняя из которых отделит символы a_{k-1} и a_k друг от друга. Исходную строку s можно однозначно восстановить по набору $(s_1,s_2,\ldots,s_{k-1})$ очевидным способом. Следовательно, имеет место взаимно однозначное соответствие

$$s \leftrightarrow (s_1,s_2,\ldots,s_{k-1})$$

и кодирование строки s эквивалентно кодированию набора битовых строк $(s_1,s_2,\ldots,s_{k-1})$.

Каждая битовая строка s_i содержит в точности n_i единиц. Количество битовых строк s_i с фиксированным числом единиц n_i выражается биномиальным коэффициентом $\binom{|s_i|}{n_i}$. Стало быть, каждой такой строке можно присвоить номер $\lambda(s_i)$, такой что

$$0 \le \lambda(s_i) < \binom{|s_i|}{n_i} = D_i,\ i=1,2,\ldots,k-1.$$

Способы построения $\lambda(s_i)$ описаны в [5], где битовая строка с заданным числом единиц называется битовой строкой с заданным весом.

Произведение всех D_i, где $i=1,2,\ldots,k-1$, дает в точности уже упомянутый мультиномиальный коэффициент:

$$\binom{n}{n_1,n_2,\ldots,n_k} = D_1D_2\ldots D_{k-1}.$$

Таким образом, набор $\lambda(s_1),\lambda(s_2),\ldots,\lambda(s_{k-1})$ можно интерпретировать цифрами в записи числа $\mathrm{index}(s)$ в системе счисления по смешанному основанию $D_1D_2\ldots D_{k-1}$. Запишем в явном виде номер строки s:

$$\mathrm{index}(s) = \lambda(s_1) + \lambda(s_2)D_1 + \lambda(s_3)D_1D_2 + \ldots + \lambda(s_{k-1})D_1D_2\ldots D_{k-2}.$$

При таком подходе функция index^{-1} реализуется тривиально.

Внимательнее рассмотрим приведенную схему. Мы взяли алфавит A и разделили его на две части: $A = A_0 \cup A_1$, где $A_0 = \{a_1\}$, $A_1 = \{a_2,a_3,\ldots,a_k\}$. Потом в строке s все вхождения символа a_1 обозначили единицей, а вхождения остальных символов нулями, и получили тем самым строку s_1. Далее взяли A_1 и опять разделили на две части: $A_1 = A_{10} \cup A_{11}$, где $A_{10} = \{a_2\}$, $A_{11} = \{a_3,a_4,\ldots,a_k\}$. Среди символов, обозначенных нулем на предыдущем шаге, все вхождения a_2 обозначили единицей, а остальных символов – нулями. В результате была построена строка s_2. Мы продолжали делать разбиение алфавита до тех пор, пока в последнее множество не попали последние два символа a_{k-1} и a_k. В результате получили набор рекурсивных разбиений алфавита A:

$$A = A_0 \cup \left(A_{10} \cup \left(A_{110} \cup \left(A_{\underbrace{11\ldots10}_{k-1}} \cup A_{\underbrace{11\ldots1}_{k-1}} \right) \right) \right), \text{ где}$$

$$A_0 = \{a_1\}, A_{10} = \{a_2\}, \ldots, A_{\underbrace{11\ldots10}_{k-1}} = \{a_{k-1}\}, A_{\underbrace{11\ldots11}_{k-1}} = \{a_k\}.$$

Для удобства любые рекурсивные разбиения можно представлять в виде дерева (в данном случае оно бинарное), которое в дальнейшем будем называть деревом разбиения алфавита.

Листьями в дереве являются одноэлементные подмножества или (фактически) символы алфавита A. Именно разбиение определило способ формирования всех строк s_i.

Теперь рассмотрим произвольные рекурсивные разбиения алфавита и обобщим на них описанный подход. На каждом шаге разбивать можно на произвольное число частей, но в конце должны остаться одноэлементные подмножества алфавита. Можно выбирать следующие стратегии разбиения:

- делить каждое множество на две части – бинарное дерево;
- делить каждое множество на $q > 2$ частей – q-е дерево;
- делить каждое множество на произвольное число частей – произвольное дерево.

Для мультиномиального коэффициента выполняется замечательное тождество:

$$\binom{n_1 + n_2 + \ldots + n_k}{n_1, n_2, \ldots, n_k} = \binom{n_1 + \ldots + n_t + n_{t+1} + \ldots + n_k}{n_1 + \ldots + n_t, n_{t+1} + \ldots + n_k} \binom{n_1 + \ldots + n_t}{n_1, \ldots, n_t} \binom{n_{t+1} + \ldots + n_k}{n_{t+1}, \ldots, n_k}.$$

Благодаря этому любое дерево разбиения алфавита A единственным образом соответствует разложению числа $\binom{n}{n_1, n_2, \ldots, n_k}$ на множители из других мультиномиальных коэффициентов D_i:

$$\binom{n}{n_1, n_2, \ldots, n_k} = D_1 D_2 \ldots D_l.$$

Каждый множитель D_i соответствует внутреннему узлу дерева разбиения, у которого имеется k_i дочерних узлов. В то же время каждому такому узлу соответствует строка s_i из символов $0, 1, \ldots, k_i - 1$, которой опять можно сопоставить номер $\lambda(s_i)$ такой, что $0 \le \lambda(s_i) < D_i$. Вычисляя последовательно номера $\lambda(s_i)$ каждой из строк, можно получить номер строки s как число, записанной в системе счисления по смешанному основанию.

Номер строки s может быть получен разными способами в зависимости от конкретного дерева разбиения алфавита. Тем не менее все эти способы эквивалентны и дают оптимальный результат, потому что N объектов кодируются номерами $0, 1, \ldots, N-1$.

Указанный способ вычисления $\mathrm{index}(s)$ работает для любых деревьев разбиения алфавита, но на практике почти не применяется. Это связано с тем, что подсчет мультиномиальных коэффициентов является вычислительно трудоемкой задачей даже при небольших значениях n.

Далее предложен менее трудоемкий вариант получения $\mathrm{index}(s)$, но оптимальный (согласно энтропии) результат он будет давать только на специально построенных разбиениях алфавита.

Опять будем строить рекурсивные разбиения алфавита A так, чтобы строке s соответствовал набор строк $(s_1, s_2, \ldots, s_l)$. Пусть каждая s_i состоит из элементов $0, 1, \ldots, k_i - 1$. Изменим способ получения номера $\lambda(s_i)$ для строки s_i. Будем считать символы строки s_i цифрами числа $\lambda(s_i)$ в системе счисления с фиксированным основанием k_i. В явном виде вычисление $\lambda(s_i)$ сводится к полиному

$$\lambda(s_i) = \sum_{j=1}^{|s_i|} s_i[j] k_i^{j-1},$$

для которого выполняется ограничение

$$0 \le \lambda(s_i) < k_i^{|s_i|} = D_i.$$

Из этого следует, что

$$0 \le \mathrm{index}(s) < D_1 D_2 \ldots D_l = k_1^{|s_1|} k_2^{|s_2|} \ldots k_l^{|s_l|} = N'.$$

В общем случае такое кодирование не является оптимальным, потому что всегда $N' > N$. Оптимальным оно будет лишь в том случае, когда распределение символов $0, 1, \ldots, k_i - 1$ в каждой строке s_i является равномерным или близко к нему. При равномерном распределении энтропия Шеннона такой строки будет равна $\log_2 k_i$, а размер по энтропии – $|s_i| \log_2 k_i$. В то же время для двоичного представления $\lambda(s_i)$ всегда требуется

$$\log_2 D_i = \log_2 k_i^{|s_i|} = |s_i| \log_2 k_i$$

бит, что совпадает с размером по энтропии. Тогда для N' получаем оптимальное значение:

$$\log_2 N' = |s_1| \log_2 k_1 + |s_2| \log_2 k_2 + \ldots + |s_l| \log_2 k_l .$$

Если строить разбиения алфавита A так, чтобы все строки s_i имели распределение близкое к равномерному, то кодирование будет близко к оптимальному и конечный результат кодирования строки s будет близок к размеру по энтропии. При этом будет использоваться вычисление полиномов степени не выше n, которое на практике можно выполнять по схеме Горнера, а также распараллеливать вычисления.

Кодирование Хаффмана или Шеннона – Фано [8] является частным случаем описанной выше конструкции. В обоих случаях используется разбиение алфавита на две части на каждом шаге так, чтобы распределение было равномерным. Метод Хаффмана отличается от метода Шеннона – Фано только способом построения дерева разбиения. В методе Шеннона – Фано дерево разбиения строится «сверху-вниз»: алфавит делится на две части специальным образом, затем эти части делятся на две части и т. д. В методе Хаффмана дерево строится «снизу-вверх»: находятся два самых редких символа в строке и объединяются в новый символ, а затем процедура повторяется. Вычислять полиномы для нахождения $\lambda(s_i)$ не нужно, потому что это эквивалентно битовой записи числа, которая получается непосредственно для бинарных деревьев разбиения.

Неподвижная точка и кодирование Лемпеля – Зива

Опишем второй подход, где кодироваться будет не сама строка s, а отображение, связанное с ней. Рассмотрим некоторые автоморфизмы множества A^n. Любая перестановка π из n элементов $1, 2, \ldots, n$ задает такой автоморфизм. Действие перестановки на строку $s \in A^n$ заключается в переупорядочивании ее символов:

$$\pi(s) = s[\pi(1)]s[\pi(2)]\ldots s[\pi(n)].$$

Пусть $P(n)$ – множество всех перестановок из n элементов. Для любой строки $s \in A^n$ найдутся перестановки $\pi \in P(n)$, для которых s является неподвижной точкой или $\pi(s) = s$. Обозначим множество таких перестановок $P_s(n)$. Оно не пусто, потому что всегда содержит тождественную перестановку. Разные строки могут быть неподвижными точками у одной и той же перестановки. Разные перестановки могут иметь одну и ту же строку в качестве неподвижной точки.

Любая перестановка π может быть записана как последовательность непересекающихся циклов (в некотором порядке):

$$\pi = c_1 c_2 \ldots c_m .$$

В данном случае используются обозначения, принятые в комбинаторике, поэтому учитываются циклы единичной длины (в отличие от алгебры). Каждый цикл c_i длины $l_i > 0$ можно записать в виде

$$c_i = \left(p_i^1 p_i^2 \ldots p_i^{l_i}\right),$$

тогда полная запись перестановки выглядит как

$$\pi = \left(p_1^1 p_1^2 \ldots p_1^{l_1}\right)\left(p_2^1 p_2^2 \ldots p_2^{l_2}\right)\ldots\left(p_m^1 p_m^2 \ldots p_m^{l_m}\right).$$

Число непересекающихся циклов в перестановке π обозначим $\tau(\pi)$, при этом, всегда выполняется $1 \le \tau(\pi) \le n$. Если $\pi(s) = s$, то обязательно $s[p_i^\alpha] = s[p_i^\beta]$ при любых $1 \le \alpha, \beta \le l_i$. Иначе говоря, каждый цикл перестановки содержит номера позиций одинаковых символов в строке s. В общем случае не все одинаковые символы из строки обязаны попадать в один и тот же цикл.

Из $P_s(n)$ выберем π_s – перестановку с минимальным числом непересекающихся циклов. Очевидно, что число циклов в π_s будет равно количеству разных символов $a_i \in A$ в стро-

ке s, которое не превышает k – размер алфавита. Позиции, на которых находятся одинаковые символы в строке s, попадают в один и тот же цикл перестановки π_s.

Если строка s содержит все символы алфавита, то длины циклов в π_s совпадают со статистикой $n_1, n_2, \ldots, n_k$. Без ограничения общности можно считать, что $l_i = n_i$, и вся перестановка записывается в виде

$$\pi_s = \left(p_1^1 p_1^2 \ldots p_1^{n_1}\right)\left(p_2^1 p_2^2 \ldots p_2^{n_2}\right)\ldots\left(p_k^1 p_k^2 \ldots p_k^{n_k}\right),$$

где p_i^j – позиции символа a_i в строке s при $1 \le j \le n_i$.

Пусть вектор C длины $\tau(\pi_s)$ задает соответствие каждого цикла перестановки определенному символу алфавита. Тогда пара (π_s, C) однозначно определяет строку s и наоборот. Следовательно, вместо исходной строки s можно попытаться закодировать пару (π_s, C). Если на кодирование (π_s, C) будет потрачено меньше, чем на кодирование s другим способом, то можно говорить о сжатии. Для кодирования перестановки π_s и вектора C можно воспользоваться энумеративным кодированием, как это было описано ранее.

Любой перестановке $\pi \in P(n)$ можно сопоставить бинарную квадратную матрицу $B(\pi)$ размера $n \times n$ со следующими элементами:

$$b(\pi)_{ij} = \begin{cases} 1, \text{если } \pi(i) = j \\ 0, \text{иначе} \end{cases}.$$

Каждая строка и каждый столбец такой матрицы содержит в точности одну единицу.

Будем считать, что символы алфавита A являются элементами кольца $\mathbb{Z}(k)$ – целых чисел по модулю k. Каждому символу a_i сопоставлено число $i-1 \in \mathbb{Z}(k)$. При такой интерпретации любой строке из A^n очевидным способом сопоставляется вектор из $\mathbb{Z}^n(k)$. Любая бинарная матрица размера $n \times n$ задает отображение $\mathbb{Z}^n(k)$ в себя.

Пусть строке $s \in A^n$ соответствует вектор $v \in \mathbb{Z}^n(k)$, тогда он является неподвижной точкой отображения $B(\pi_s)$ или $B(\pi_s)v = v$.

Для строки s, которой соответствует вектор v, рассмотрим способы построения матрицы B так, чтобы $Bv = v$. Для этого построим матрицу B' с элементами:

$$b'_{ij} = \begin{cases} 1, \text{если } v[i] = v[j]. \\ 0, \text{иначе} \end{cases}$$

Матрица B' содержит всю информацию о структуре строки, а именно: на каких позициях в строке s стоят одинаковые символы, при этом размер алфавита A неважен. Из определения следует, что матрица всегда является симметричной с единицами на главной диагонали. Она будет симметрична также относительно побочной диагонали, если строка s является палиндромом. Всего единиц в матрице будет $n_1^2 + \ldots + n_k^2$.

Двум одинаковым подстрокам строки s будут соответствовать подряд идущие единицы на диагонали параллельной главной. Пусть $s_1 = s[i]s[i+1]\ldots s[i+\Delta]$ и $s_2 = s[j]s[j+1]\ldots s[j+\Delta]$ – подстроки s, причем $s_1 = s_2$. Тогда в матрице B' выполняется

$$b'_{ij} = b'_{i+1,j+1} = \ldots = b'_{i+\Delta,j+\Delta} = 1 \text{ и } b'_{ji} = b'_{j+1,i+1} = \ldots = b'_{j+\Delta,i+\Delta} = 1.$$

Если строка s соответствует тексту на естественном языке, то, скорее всего, она содержит много одинаковых подстрок. В этом случае в B' будут часто встречаться последовательности подряд идущих единиц на диагоналях, параллельных главной. Это можно видеть, если взглянуть на матрицу B' (рис. 1, 2). Матрица представлена как изображение размера $n \times n$ пикселей. Единице в матрице соответствует белый пиксель, а нулю – черный.

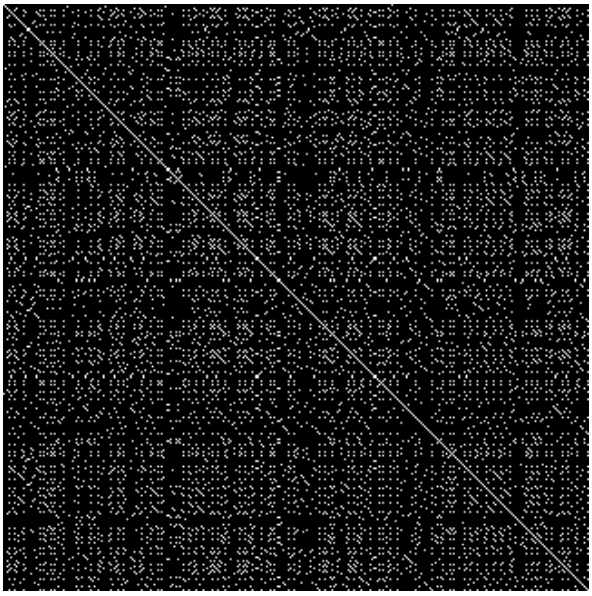

Рис. 1. Матрица B' для фрагмента текста на английском языке

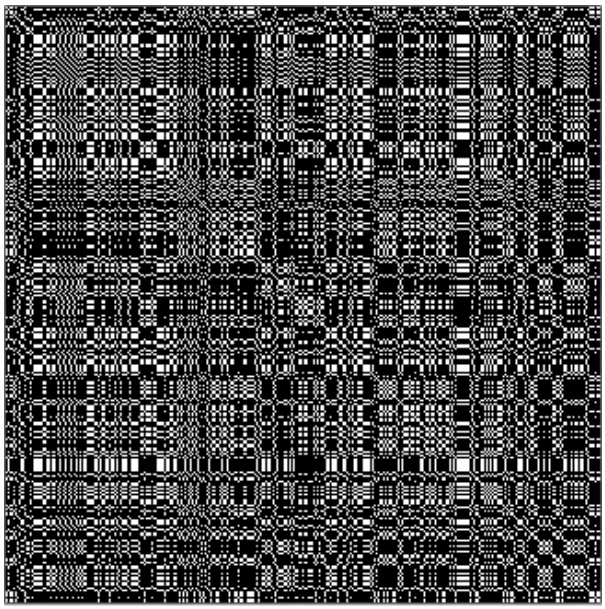

Рис. 2. Матрица B' для фрагмента последовательности ДНК

С другой стороны, сама матрица B' позволяет оценить, насколько много одинаковых подстрок содержится в строке. Если в строке s встречаются несколько одинаковых символов подряд, то им будут соответствовать квадратные блоки из единиц в B'. Если таких блоков много, то, возможно, для строки s имеет смысл применить кодирование длин серий.

Если в матрице B' в каждой строке и в каждом столбце оставить по одной единице, то будут получены всевозможные матрицы B со свойством $Bv = v$. Каждой из этих матриц будет однозначно соответствовать некоторая перестановка, для которой s будет неподвижной точкой.

Получить матрицу B из B' также можно, убрав ограничение на то, что в каждом столбце должна быть только одна единица. Таким образом, для получения матрицы B достаточно в B' оставить по одной единице в каждой строке, а количество единиц в столбце может быть произвольным. Такая матрица B в общем случае уже не будет соответствовать перестановке,

но свойство неподвижной точки $Bv = v$ сохраняется. Построенная таким образом матрица B позволяет выделить группы одинаковых символов (их позиции) в строке s. Поэтому матрицу B вместе с информацией о том, какой символ алфавита какой группе соответствует, можно использовать для кодирования строки s.

Утверждение. Всегда можно выбрать матрицу B из B' таким способом, чтобы закодировать строку s согласно размеру по энтропии.

Доказательство этого утверждения сводится к последовательному запоминанию позиций символов a_1, а затем всех остальных $a_2, \ldots, a_k$ в строке s с помощью матрицы B'. Это в точности процесс энуметативного кодирования, описанного в первой части данной статьи. Матрицу B начинаем строить с первого столбца матрицы B'. Этот первый столбец однозначно определяет позиции некоторого символа алфавита a_{i_1} в строке s. После того, как позиции a_{i_1}, известны, в B добавляем столбец, который определит все позиции элемента a_{i_2} и продолжаем так делать, пока позиции всех символов строки не попадут в матрицу B.

Матрица B' может иметь специфические свойства, которыми можно воспользоваться для получения и кодирования матрицы B. Например, матрица B' может иметь различные симметрии в зависимости от природы строки s. Однако ее главное достоинство состоит в том, что она содержит большое число подряд идущих единиц на диагоналях, параллельных главной. Именно это свойство используется в алгоритмах кодирования Лемпеля – Зива.

В основе семейства алгоритмов Лемпеля – Зива лежит простая идея: в процессе кодирования строки ее части заменяются ссылками на элементы из некоторого словаря. Различие между алгоритмами заключается в том, что процесс кодирования ссылок и словарь определены по-разному.

Одновременное построение и кодирование матрицы B из B' соответствует кодированию Лемпеля – Зива. При таком подходе единица на месте (i, j) в матрице B' интерпретируется как ссылка символа $s[i]$ на $s[j]$. Ссылаться друг на друга могут только одинаковые символы строки. Если единица находится ниже главной диагонали, то $i > j$, что означает ссылку на предшествующий символ. Если единица выше главной диагонали, то $i < j$, что является ссылкой на последующий символ. На одну и ту же позицию может ссылаться несколько символов, но каждый символ обязан ссылаться только на одну позицию. В матричном виде это выражается тем, что в каждой строке должна быть в точности одна единица. Алгоритмы кодирования Лемпеля – Зива используют ссылки только на предшествующие символы, т. е. только ту часть матрицы B', которая находится ниже главной диагонали.

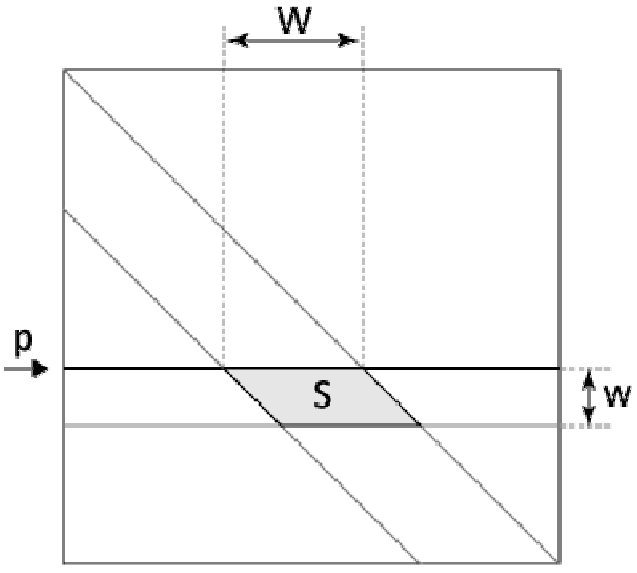

Рис. 3. Матрица B' во время кодирования LZ77

В качестве примера рассмотрим известный алгоритм LZ77 со скользящим окном [3]. Пусть окно имеет размер $N = W + w$ и состоит из двух частей: словаря длины W и буфера упреждающего просмотра длины w. На рис. 3 схематично представлена матрица B', из которой одновременно строится и запоминается матрица B. Допустим, что в процессе кодирования строки s мы находимся в позиции p. Нужно найти совпадение наибольшей длины (не превосходящей W) в словаре. Для этого в матрице нужно обследовать область S и выбрать диагональ из единиц наибольшей длины. Затем запоминается позиция диагонали, ее длина и следующий символ из строки.

Можно предложить другие способы кодирования матрицы B из B'. Идея также заключается в том, чтобы воспользоваться наличием большого количества подряд идущих единиц на диагоналях матрицы B'. Пусть B' содержит n диагоналей $d_0, d_2, \ldots, d_{n-1}$. Каждая диагональ d_i состоит из элементов $b'_{t+1,(i+t) \bmod k+1}$, где $t = 0, 1, \ldots, n-1$.

Суть алгоритма кодирования заключается в последовательном кодировании диагоналей d_i. На каждом шаге в матрице B' выбирается диагональ с максимальным числом единиц и кодируется согласно энтропии. Это можно сделать с помощью арифметического [9] или энумеративного кодирования. Те единицы, которые оказались на закодированной диагонали, автоматически попадают в матрицу B. Строки и столбцы с единицами на выбранной диагонали больше не учитываются в B'. Затем выбирается следующая диагональ с максимальным количеством единиц и кодируется согласно энтропии. Процесс продолжается до тех пор, пока матрица B не будет полностью построена и закодирована.

Заключение

В статье рассмотрено обобщение кодирования Хаффмана для случая произвольного рекурсивного разбиения множества алфавита. Показано, как получить это обобщение исходя из простых идей энумеративного кодирования. Результаты обобщения были использованы на практике для построения алгоритма быстрого сжатия видеоданных [10].

Обобщение кодирования Лемпеля – Зива позволяет единообразно взглянуть на существующие алгоритмы этого класса. Это может вызвать новые исследования в этой области. Все такие алгоритмы кодируют одно и то же отображение (его матрицу), но делают это разными способами. Подробно разобран пример того, как алгоритм LZ77 кодирует матрицу отображения, построенного по строке.

Список литературы

1. *Huffman D. A.* A Method for the Construction of Minimum-Redundancy Codes // Proceedings of the I.R.E. 1952. P. 1098–1102.

2. *Fisher Y.* Fractal Image Compression: Theory and Application. Springer Verlag, 1995.

3. *Ziv J., Lempel A.* A universal algorithm for sequential data compression // IEEE Transactions on Information Theory, IT. 1977.Vol. 23 (3). P. 337–343.

4. *Ziv J., Lempel A.* Compression of Individual Sequences via Variable-Rate Coding // IEEE Transactions on Information Theory, IT. 1978. Vol. 24 (5). P. 530–536.

5. *Cover T. M.* Enumerative Source Encoding // IEEE Transactions on Information Theory. 1973. Vol. IT-19. P. 73–77.

6. *Hartley R. V. L.* Transmission of Information // Bell System Technical Journal. July 1928.

7. *Shannon C. E.* A Mathematical Theory of Communication // Bell System Technical Journal. 1948. Vol. 27. P. 379–423; 623–656.

8. *Fano R. M*. The transmission of information // Technical Report. 1949. No. 65.

9. *Rissanen J. J*. Generalized Kraft Inequality and Arithmetic Coding // IBM J. Res. Dev. May 1976. P. 198–203.

10. *Ковалев Д. С*. Алгоритм быстрого кодирования видеоданных // Материалы XLVI МНСК «Студент и научно-технический прогресс». Новосибирск, 2008. С. 8–9.

Материал поступил в редколлегию 17.05.2010

D. S. Kovalev

GENERALIZATION OF KNOWN METHODS OF STRING CODING

The first part of this paper is about Huffman coding. Its key feature is a binary tree construction. This tree defines recursive partition of alphabet set to construct prefix codes. Generalization of binary tree to q-ary tree is well known. This paper gives Huffman coding generalization for *any* trees. The second part of this paper defines another approach to understanding Lembel-Ziv family of algorithms. The suggestion is that these algorithms encode a mapping associated with a string, not a string itself. The string is a fixed point of encoded mapping in this case.

Keywords: Huffman coding, Lempel-Ziv algorithms, LZ77, LZ78, enumerative coding.