ISSN 2079-3316 **ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ** **т. 9**, № 4(39), с. 477–491

ББК 32.973.1
УДК 004.627

Ю. В. Шевчук

Vbinary: ещё раз о представлении целых чисел с переменной разрядностью

Аннотация. В статье представлен параметризованный префиксный код переменной длины для кодирования целых чисел. Код рассматривается на примерах в сравнении с существующими кодами, в том числе кодами Golomb/Rice и Elias. Предлагается система записи параметров кода в виде текстовой строки, позволяющая точно идентифицировать используемый вариант кода.

В коде Vbinary используется новый n-арный метод расширения разрядности кодовых слов, благодаря которому достигается гибкость: возможность работы с потоками битов или байтов, эффективное кодирование малых или больших чисел, согласование распределения длин кодовых слов с распределением входных данных, оптимизация для повышения эффективности кодирования и декодирования. К потенциальным применениям кода Vbinary относятся сетевые протоколы, представление данных в оперативной памяти и на диске, а также применение на финальных стадиях алгоритмов компрессии данных.

Ключевые слова и фразы: кодирование целых чисел, коды с переменной длиной, префиксный код, параметризованный код, компрессия данных.

Введение

В повседневной практике программисты регулярно занимаются описанием структур данных и сталкиваются с необходимостью выбора разрядности полей данных. Этот выбор часто оказывается непростым, особенно когда описываемая структура данных представляет собой заголовок пакета в каком-либо сетевом протоколе: если выбрать неудачно, изменение длины повлечёт необходимость обновления программного обеспечения на всех устройствах, использующих протокол.

Работа выполнена в рамках госзадания по теме AAAA-A17-117040610378-6.

10.25209/2079-3316-2018-9-4-477-491

Желание сэкономить место в памяти или объём сетевого трафика подталкивает выбрать минимально возможную разрядность поля, но приходится также думать о будущих расширениях и о том, что вообще произвольные ограничения в программе это нехорошо [1]. Выбор иногда требует совершенно непропорциональных задаче усилий, и заканчивается компромиссным результатом, который впоследствии оказывается неудачным. Предвидя эту ситуацию, программисты прибегают к кодам переменной длины.

Существует множество кодов переменной длины, например унарное кодирование, коды Голомба [2], коды Элиаса: γ, δ, ω [3]. Но при выборе кода для своего приложения программист снова сталкивается с трудностями: кодовые слова с длиной не кратной 8 могут не вписываться в определяемую структуру данных, распределение длин кодовых слов может не подходить к распределению кодируемых данных, вычислительные затраты на кодирование и декодирование кода переменной длины могут быть неприемлемы для высокоскоростного приложения.

В итоге самым популярным решением оказывается кодирование байтовой цепочкой переменной длины: в каждом байте 7 битов данных и один бит — признак конца цепочки (пример реализации: [14]). Этот код не особенно хорош как в отношении эффективности кодирования[1], так и в отношении затрат на кодирование/декодирование, но понятен и прост в реализации, и потому широко распространён.

Мы попробуем улучшить эту ситуацию при помощи нового параметризованного кода. Код Vbinary является параметризованным префиксным кодом переменной длины, который за счёт рационального выбора параметров можно адаптировать к нуждам конкретного приложения во всех трёх аспектах: эффективность кодирования, распределение длин кодовых слов и затраты на кодирование/декодирование.

1. Организация кода Vbinary

В качестве простейшего примера рассмотрим код vbinary2x (таблица 1). Как указано в его имени, этот код имеет базовую длину 2 бита — это длина самого короткого кодового слова. Из четырех значений, которые может содержать базовая бинарная группа, три используются для представления целых чисел 0..2.

[1]определяемой как число битов в кодовом слове, необходимое для представления заданного кодируемого числа

Таблица 1. Простые коды семейства Vbinary

кодируемое число	vbinary2x	vbinary2x2	vbinary2x3x
0	00	00	00
1	01	01	01
2	10	10	10
3	11 00	11 00	11 000
4	11 01	11 01	11 001
5	11 10	11 10	11 010
6	11 11 00	11 11	11 011
7	11 11 01	*none*	11 100
8	11 11 10	*none*	11 101
9	11 11 11 00	*none*	11 110
10	...	*none*	11 111 000
...	...	*none*	...

Буква x в имени кода означает, что для представления следующих чисел длину кодового слова нужно расширить — поэтому последнее из четырёх значений (3, в бинарном виде 11) используется для расширения. Оно означает, что за базовыми битами следуют ещё два бита, которые называются первым уровнем расширения. Первый уровень расширения позволяет закодировать ещё три числа: 3..5, а для представления бóльших чисел добавляется следующий уровень расширения.

Код vbinary2x бесконечный: добавляя новые уровни расширения, можно закодировать сколь угодно большое целое число. Для сравнения, код vbinary2x2 (таблица 1) конечный — на это указывает отсутствие буквы x после ширины последнего присутствующего в имени уровня расширения.

Код vbinary2x2 позволяет представить всего 7 целочисленных значений. В конечных кодах последнее значение последней битовой группы не зарезервировано для расширения, а представляет очередное число. Так, в коде vbinary2x2 последнее представимое число (6) соответствует кодовому слову 1111, которое в коде vbinary2x зарезервировано для расширения (добавления уровня 2).

Имена кодов Vbinary по сути представляют собой перечисление правил расширения, хотя эти правила могут быть сложнее, чем уже встречавшиеся. Все коды в таблице 1 имеют только одно значение, зарезервированное для расширения на каждом уровне, но в принципе на каждом уровне мы можем зарезервировать до 2^{w_i} значений, где w_i это разрядность уровня. Зарезервировав более одного значения, мы получаем возможность множественных (n-арных) расширений, которая позволяет получить коды с интересными свойствами.

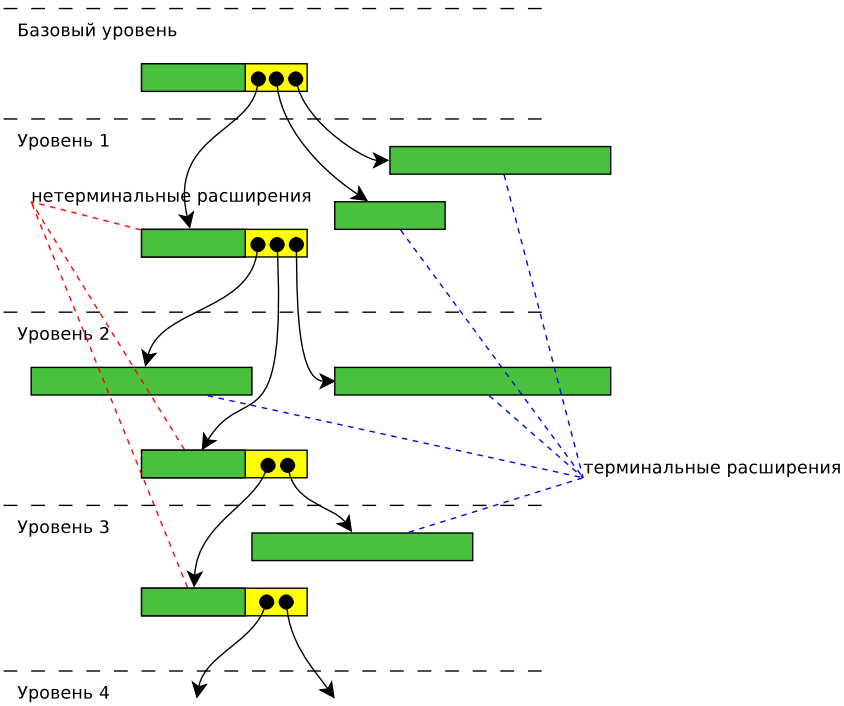

Рис. 1. Максимум одно нетерминальное расширение на каждом уровне. Код vbinary6x(6x,4,8)(8,6x,10)(6x,7).

Только одно из нескольких расширений («нетерминальное расширение») может иметь дальнейшие расширения, остальные расширения на том же уровне используются исключительно для представления кодируемых чисел («терминальные расширения») (рис. 1). Это ограничение («почти линейная структура») введено, чтобы избежать загромождения имён вложенными скобками, выражающими произвольную древовидную структуру.

Далее мы будем рассматривать коды с множественными расширениями, имена которых имеют более сложную структуру. Полное описание синтаксиса имён кодов семейства Vbinary приведено в Приложении A.

Рассмотрим код vbinary1x(8,1x) (таблица 2). Уровень расширения 1 задан конструкцией (8,1x), означающей что уровень имеет два варианта и, соответственно, на уровне 0 два значения зарезервировано для расширения (рис. 2). Поскольку ширина базового уровня всего один бит,

Таблица 2. vbinary1x(8,1x) совпадает с кодом Golomb(256)

кодируемое число	vbinary1x(8,1x)	длина кода, бит
0	0 00000000	9
...	...	9
255	0 11111111	9
256	1 0 00000000	10
...	...	10
511	1 0 11111111	10
512	1 1 0 00000000	11
...	...	11
767	1 1 0 11111111	11
...	...	⩾12

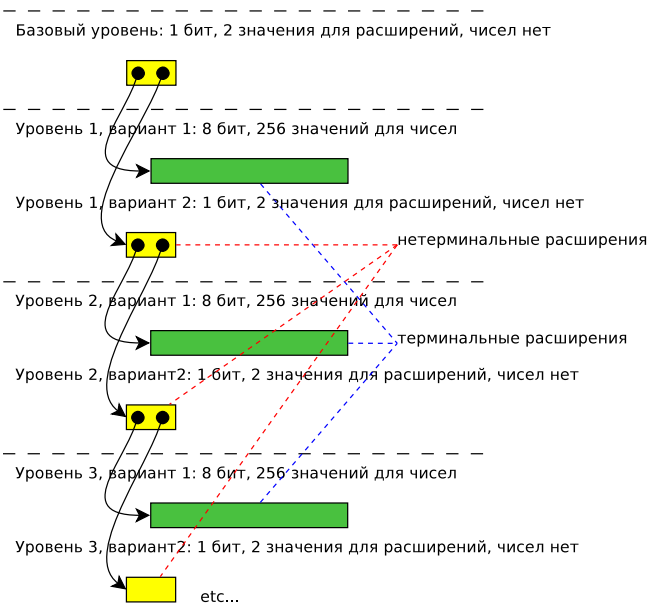

Рис. 2. Уровни расширения для кода vbinary1x(8,1x)

на нём не остаётся значений для представления чисел, оба значения используются для расширения.

Первое расширение — терминальное, позволяющее представить 2^8 чисел. Второе расширение — нетерминальное расширение с разрядностью 1 бит, которое работает в точности так же, как базовый уровень (оба значения используются для дальнейшего расширения). Поскольку спецификация последнего расширения в перечне содержит букву x, код

Таблица 3. vbinary1x2x(2,1x)(a1,a0) эквивалентен коду Elias γ

кодируемое число	vbinary1x2x(2,1x)(a1,a0)	длина кода, бит
0	0	1
1	1 00	3
2	1 01	3
3	1 10 00	5
4	1 10 01	5
5	1 10 10	5
6	1 10 11	5
7	1 11 0 000	7
8	1 11 0 001	7
9	1 11 0 010	7
10	1 11 0 011	7
11	1 11 0 100	7
12	1 11 0 101	7
13	1 11 0 110	7
14	1 11 0 111	7
15	1 11 1 0 0000	9
...	...	$\geqslant 9$

vbinary1x(8,1x) является бесконечным: последнее правило расширения применяется столько раз, сколько потребуется для представления заданного числа.

Следующий пример — код vbinary1x2x(2,1x)(a1,a0). В имени кода три спецификации уровней (базовый, уровень 1, уровень 2), а также новая конструкция — повторитель (a1,a0), который используется для генерации уровней 3, 4 и т.д. Это аддитивный повторитель, указывающий что для генерации уровня $i+1$ нужно прибавить 1 и 0 соответственно к разрядности вариантов расширения уровня i. Таким образом, все следующие имена задают один и тот же код:

vbinary1x2x(2,1x)(a1,a0)

vbinary1x2x(2,1x)(3,1x)(a1,a0)

vbinary1x2x(2,1x)(3,1x)(4,1x)(a1,a0)

vbinary1x2x(2,1x)(3,1x)(4,1x)(5,1x)(a1,a0)

...

Благодаря аддитивному повторителю, каждый вновь сгенерированный уровень представляет вдвое больше чисел, чем предыдущий: $4, 8, 16, \ldots$ (таблица 3).

Полный синтаксис компонентов повторителя (mKaC) позволяет задавать увеличение разрядности полей в смешанной арифметико-геометрической прогрессии: $w_{i+1} = Kw_i + C$, где множитель K и слагаемое C могут быть целыми числами или обыкновенными дробями. Десятичные дроби не используются, поскольку их менее удобно обрабатывать на процессорах, не имеющих поддержки операций с плавающей точкой.

Таблица 4. vbinary1x совпадает с Unary и с Golomb(1)

кодируемое число	vbinary1x	длина кода, бит
0	0̲	1
1	1 0̲	2
2	1 1 0̲	3
3	1 1 1 0̲	4
4	1 1 1 1 0̲	5
5	1 1 1 1 1 0̲	6
...	...	...

Медленный рост разрядности поддерживается следующим образом. Вычисления при генерации уровней (умножение на обыкновенную дробь, сложение с обыкновенной дробью) выполняются при помощи арифметики с фиксированной точкой с весом[2] младшего разряда 1/16. Целые части результатов вычисления используются в качестве ширин вариантов расширения на сгенерированном уровне, а полный результат с дробной частью сохраняется и используется в качестве входных данных при генерации следующего уровня. Таким образом, повторитель (a1/3) задаёт увеличение длины на 1 бит на каждом третьем шаге, а повторитель (m4/3) даёт экспоненциальный рост разрядности со скоростью $(4/3)^n$.

Порядок использования значений полей на уровне с вариантами следующий:

(*1*) используются бинарные значения каждого из вариантов расширения в порядке, в котором они перечислены в имени кода. В нетерминальных вариантах пропускаются (не назначаются кодируемым числам) значения, зарезервированные для расширения;

(*2*) пропущенные зарезервированные значения используются при обработке следующего уровня в соответствии с пунктом 1.

На этом описание синтаксиса и семантики имён кодов семейства Vbinary закончено. Синтаксис задаёт обширное семейство кодов, которое мы сейчас рассмотрим в сравнении с другими кодами переменной длины.

2. Vbinary в сравнении с другими кодами

Как некоторые другие параметризованные коды [**2**, **5**], семейство Vbinary включает в себя код, совпадающий с унарным кодированием: vbinary1x (таблица 4).

[2] точность выбрана небольшая, чтобы вычисления были эффективны на 8-битных архитектурах

Таблица 5. vbinary1x(7,1x)(a7,a0) близок к коду *varint* [14]

кодируемое число	vbinary1x(7,1x)(a7,a0)	длина кода, бит
0	0 0000000	8
1	0 0000001	8
...	...	...
127	0 1111111	8
128	1 0 0000000 0000000	8
...	...	...
255	1 0 0000000 1111111	16
256	1 0 0000001 0000000	16
...	...	...
16511	1 0 1111111 1111111	16
16512	1 1 0 0000000 0000000 0000000	24
...	...	...
2113663	1 1 0 1111111 1111111 1111111	24
...	...	...

Мы видели, что код vbinary1x(8,1x) совпадает с кодом Golomb(256). Можно обобщить: vbinary1x(n,1x) совпадает с Golomb(2^n).

Мы также рассматривали код vbinary1x2x(2,1x)(a1,a0) (таблица 3), который эквивалентен коду Elias γ [3]. Отличия состоят в инверсии унарной части и сдвиге области определения на 1.

Ещё один родственный код это упомянутое во Введении кодирование цепочкой байтов переменной длины. Разновидность этого кода, используемая в системе Git [14], называется *varint*. Она эквивалентна vbinary1x(7,1x)(a7,a0) (таблица 5). Отличие состоит в порядке битов: в *varint* биты-признаки конца байтовой последовательности содержатся в каждом байте, а в vbinary1x(7,1x)(a7,a0) они собраны вместе и располагаются в начале кодового слова как унарный префикс.

Код start-step-stop [4] это конечный параметризованный префиксный код, организованный подобно коду Elias γ, но бинарная часть в нём увеличивается не по одному биту, а на величину, заданную параметром *step*. Этот код при любом наборе параметров имеет точный аналог в семействе Vbinary: например, код start-step-stop(3,2,9) совпадает с vbinary1x(3,1x)(5,1x)(7,9). Мы не можем использовать в определении кода Vbinary повторитель, поскольку повторители в Vbinary не имеют ограничителя (*stop*). Так, код vbinary1x(3,1x)(a2,a0) похож на start-step-stop(3,2,9), но является бесконечным кодом и поэтому длиннее на один бит в интервале значений 168..679.

Код Start/Stop [5] тоже является конечным параметризованным префиксным кодом. Код параметризуется вектором длин бинарных полей, за счёт чего может быть оптимизирован под известное распределение кодируемых данных. Свойство «префиксности» кода достигается при помощи унарного кодирования числа бинарных полей, но биты унарного числа распределены по кодовому слову, как в

Таблица 6. Сравнительная эффективность кодов в диапазоне значений 0..16

Код	min	avg	median	max	макс. значение
binary	5	5	5	5	32
Unary	1	8.5	8	17	unlimited
Elias γ	1	5.94	7	9	unlimited
Elias δ	1	6.53	8	9	unlimited
Elias ω	1	5.76	7	11	unlimited
Golomb(16)	5	5.06	5	6	unlimited
Golomb(8)	4	4.59	5	6	unlimited
Golomb(4)	3	4.65	5	7	unlimited
vbinary4x1x	4	4.1	4	6	unlimited
vbinary4x2x	4	4.24	4	6	unlimited
vbinary3x(3,3x)	3	4.94	6	6	unlimited
vbinary3x(2,2,2)	3	4.41	5	5	17
vbinary2x(3,4x)	2	5.06	5	6	unlimited
vbinary1x2x(2,3x)	1	5.59	6	8	unlimited
vbinary1x2x(2,2,3x)3x	1	5.06	5	6	unlimited

упомянутом выше коде *varint*. Код Start/Stop не имеет точных аналогов в семействе Vbinary, но идеологически он ближе к Vbinary, чем все ранее рассмотренные коды.

Все ранее рассмотренные коды реализуют свойство «префиксности» с помощью той или иной формы унарного кодирования. Поскольку унарное кодирование представимо в семействе Vbinary, большинство этих кодов имеют аналоги в семействе Vbinary. Однако используемый в Vbinary метод n-арного расширения даёт гораздо больше возможностей, чем унарные префиксы.

Коды, в которых свойство «префиксности» реализуется другими методами, например код Левенштейна $W2'$ [11], коды Elias δ и ω [3], и более современные [7, 8], не имеют прямых аналогов в семействе Vbinary. Цитируя [4], «эти коды имеют хорошие асимптотические характеристики для очень больших чисел», но для представления малых чисел они не очень эффективны.

Значительная работа в области кодирования малых чисел с переменной длиной представлена в [9]. Эти коды также не имеют эквивалентов в семействе Vbinary. Они демонстрируют улучшение по сравнению с кодом Elias γ для определённых распределений кодируемых данных, но не имеют гибкости, которую дают параметризованные коды.

Благодаря параметризации и n-арному расширению, код Vbinary позволяет получить хорошие характеристики как в области малых, так и в области больших чисел. Мы рассмотрим это на примерах в следующих двух разделах.

Таблица 7. Коды Vbinary, упомянутые в таблице 6

integer value	vbinary 4x1x	vbinary 4x2x	vbinary 3x(3,3x)	vbinary 3x(2,2,2)	vbinary 2x(3,4x)	vbinary 1x2x(2,3x)	vbinary 1x2x(2,2,3x)3x
0	0000	0000	000	000	00	0	0
1	0001	0001	001	001	01	1 00	1 00
2	0010	0010	010	010	10 000	1 01	1 01 00
3	0011	0011	011	011	10 001	1 10 00	1 01 01
4	0100	0100	100	100	10 010	1 10 01	1 01 10
5	0101	0101	101	101 00	10 011	1 10 10	1 01 11
6	0110	0110	110 000	101 01	10 100	1 10 11	1 10 00
7	0111	0111	110 001	101 10	10 101	1 11 000	1 10 01
8	1000	1000	110 010	101 11	10 110	1 11 001	1 10 10
9	1001	1001	110 100	110 00	10 111	1 11 010	1 10 11
10	1010	1010	110 101	110 01	11 0000	1 11 011	1 11 000
11	1011	1011	110 110	110 10	11 0001	1 11 100	1 11 001
12	1100	1100	110 111	110 11	11 0010	1 11 101	1 11 010
13	1101	1101	111 000	111 00	11 0011	1 11 110 00	1 11 011
14	1110	1110	111 001	111 01	11 0100	1 11 110 01	1 11 100
15	1111 0	1111 00	111 010	111 10	11 0101	1 11 110 10	1 11 101
16	1111 1 0	1111 01	111 011	111 11	11 0110	1 11 110 11	1 11 110
17	1111 1 1 0	1111 10	111 100	*none*	11 0111	1 11 111 000	1 11 111 000
18	1111 1 1 1 0	1111 11 00	111 101	*none*	11 1000	1 11 111 001	1 11 111 001

3. Представление малых чисел

Предположим, мы разрабатываем сетевой протокол и нужно определить поле тэга, задающее 17 различных типов записей, а также предусмотреть возможность добавления тэгов в будущем. Прямолинейный подход — использовать поле с фиксированной длиной 5 бит. В этом варианте есть возможность расширения, но она ограничена 15 дополнительными тегами. Если же использовать кодирование с переменной длиной, мы получим возможность неограниченного расширения, а также возможность экономного кодирования, если распределение частот появления тегов известно и неравномерно, за счёт назначения коротких кодовых слов часто встречающимся тегам.

В таблице 6 сведены характеристики ряда кодов при использовании в описанном сценарии. Для каждого известного кода в таблице нашёлся код Vbinary, который превосходит его по крайней мере по одной из характеристик. По таблице кодовых слов (таблица 7) можно судить о распределении длин кодовых слов Vbinary и выбрать код, наиболее подходящий для известного распределения частот тегов. При необходимости можно продолжать подбор параметров Vbinary, используя свободно доступны программный инструмент [**15**].

4. Байт-ориентированные коды

Существуют приложения [**12**], для которых требуется, чтобы разрядность кодовых слов числа с переменной длиной была кратна

Таблица 8. Конечный байт-ориентированный код vbinary1x(7,15)

кодируемое число	vbinary1x(7,15)	длина кода, бит
0	0 0000000	8
...	...	...
127	0 1111111	8
128	1 000000000000000	16
...	...	...
32895	1 111111111111111	16

Таблица 9. Конечный байт-ориентированный код vbinary8x(8,16,24,56)

кодируемое число	vbinary8x(8,16,24,56)	длина кода, бит
0	00000000	8
1	00000001	8
...	...	...
251	11111011	8
252	11111100 00000000	16
...	...	...
507	11111100 11111111	16
508	11111101 0000000000000000	24
...	...	...
$508 + 2^{16} - 1$	11111101 1111111111111111	24
$508 + 2^{16}$	11111110 00000000000000000000000000	32
...	...	...
$508 + 2^{16} + 2^{24} - 1$	11111110 11111111111111111111111111	32
$508 + 2^{16} + 2^{24}$	11111111 000...(56 zero bits)...	64
...	...	...
$508 + 2^{16} + 2^{24} + 2^{56} - 1$	11111111 111...(56 one bits)...	64

8 (байт-ориентированные коды). Параметры Vbinary позволяют конструировать байт-ориентированные коды. Мы уже встречали байт-ориентированный код — vbinary1x(7,1x)(a7,a0) (таблица 5), но Vbinary позволяет конструировать коды, которые представляют больше значений в том же числе байтов (таблица 8), или допускают более эффективное кодирование/декодирование (таблица 9). Имея конкретные требования к характеристикам кода, можно подобрать и более подходящие параметры.

5. Некоторые свойства кодов Vbinary

5.1. Лексикографический порядок

Последовательность кодовых слов в кодах Vbinary упорядочена лексикографически тогда и только тогда, когда расширения на всех уровнях делаются в последней бинарной группе. Рассмотрим два конечных кода с максимальной длиной кодового слова 128 бит: vbinary8x(8,16,24,56x)64 и vbinary8x(8x,16,24,56)112. Первый код лексикографически упорядочен и имеет максимальное значение порядка 2^{64}, а второй не упорядочен, но максимальное значение у него порядка 2^{112}. Таким образом, лексикографический порядок достигается ценой эффективности кодирования. Если предпочесть эффективность, пожертвовав порядком, будет невозможно сравнивать числа в коде Vbinary, не декодировав их предварительно — но это не такая большая потеря, если код допускает эффективное декодирование.

5.2. Монотонность длин кодовых слов

Во всех кодах переменной длины, встречавшихся автору, длина кодовых слов монотонно возрастает (не убывает) с увеличением кодируемого числа: $L(v(n)) \leqslant L(v(n+1))$ для любого n, где $L(x)$ — количество бит в кодовом слове x, $v(n)$ — функция, кодирующая целое число n в коде переменной длины. Параметры Vbinary позволяют определять коды, в которых $L(v(n))$ не монотонна. Например, в vbinary8x(8,16,24,56,2) четыре 16-битных кодовых слова назначены четырём числам на верхней границе диапазона кодируемых чисел, в то время как максимальная длина кодового слова в этом коде равна 64.

6. Заключение

Vbinary это сильно параметризованный префиксный код с переменной длиной кодовых слов для кодирования неотрицательных целых чисел, пригодный как для битовых, так и для байтовых потоков данных. Для увеличения длин кодового слова используется оригинальный приём n-арного расширения, который даёт большие возможности для настройки кода на конкретное применение. Подбором параметров можно сконструировать код, эффективный для малых или больших чисел, имеющий рспределение длин кодовых слов близкое к частотному распределению кодируемых данных, допускающий эффективное кодирование/декодирование. Сравнение Vbinary с известными кодами

переменной длины оставляет впечатление, что подбором параметров Vbinary под конкретную задачу всегда можно получить более эффективное кодирование, чем при использовании любого заранее определённого кода переменной длины. Мы не пытались доказывать это утверждение.

Код Vbinary может найти применение в сетевых протоколах, в представлении данных на диске и в оперативной памяти. Возможно, код Vbinary удастся применить в алгоритмах сжатия данных, где в настоящее время используются коды Голомба/Райса или другие коды переменной длины [13] [10].

Мы определили систему записи параметров кода Vbinary, которая позволит программистам использовать коды Vbinary, ссылаясь на них по имени, вместо того чтобы изобретать для каждого случая подходящие коды переменной длины и документировать их. Программный инструмент для экспериментов с параметрами Vbinary свободно доступен [15].

Список литературы

[1] R. M. Stallman, D. Betz, J. Edwards. *BYTE Interview with Richard Stallman*, July, 1986. ↑478

[2] S. W. Golomb. "Run-length encodings", *IEEE Transactions on Information Theory*, **12**:3 (1966), pp. 399–401. ↑478,483

[3] P. Elias. "Universal codeword sets and representations of the integers", *IEEE Transactions on Information Theory*, **21**:2 (1975), pp. 194–203. ↑478,484,485

[4] E. R. Fiala, D. H. Greene. "Data compression with finite windows", *CACM*, **32**:4 (1989), pp. 490–505. ↑484,485

[5] S. Pigeon. "Start/stop codes", Data Compression Conference (Snowbird, Utah, 2001), 2001, 0511. ↑483,484

[6] A. S. Fraenkel, S. T. Klein. "Robust universal complete codes for transmission and compression", *Discrete Applied Mathematics*, **64**:1 (1996), pp. 31-–55. ↑

[7] M. Nangir, H. Behroozi, M. R. Aref. "A new recursive algorithm for universal coding of integers", *Journal of Information Systems and Telecommunication*, **3**:1 (9) (2015), pp. 1–6 10.7508/jist.2015.01.001. ↑485

[8] J. Nelson Raja, P. Jaganathan, S. Domnic. "A new variable-length integer code for integer representation and its application to text compression", *Indian Journal of Science and Technology*, **8**:24 (September 2015). ↑485

[9] P. Fenwick. "A note on variable-length codes with constant Hamming weights", *Journal of Universal Computer Science*, **21**:9 (2015), pp. 1136–1142. ↑485

[10] P. Fenwick. “Burrows-Wheeler compression with variable length integer codes”, *Software Practice and Experience*, **32**:13 (2002), pp. 1307–1316. ↑489

[11] В. И. Левенштейн. «Об избыточности и замедлении разделимого кодирования натуральных чисел», *Проблемы кибернетики*, **20** (1968), с. 173–179 (in Russian). ↑485

[12] F. Scholer, H. E. Williams, J. Yiannis, J. Zobel. “Compression of inverted indexes For fast query evaluation”, *SIGIR ’02 Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval* (Tampere, Finland, August 11–15, 2002), ACM, 2002, pp. 222–229. ↑486

[13] Josh Coalson. *FLAC documentation: format overview.* ↑489

[14] J. C. Hamano. *git/varint.c*, 2012. ↑478,484

[15] Yu. V. Shevchuk. *vbinary/vbinary-eval.pl*, 2018. ↑486,489

[16] D. Crocker, P. Overell, *Augmented BNF for Syntax Specifications: ABNF*, RFC 5234, 2008. ↑

Поступила в редакцию 09.11.2018
Переработана 04.12.2018
Опубликована 30.12.2018

Рекомендовал к публикации *д.ф.-м.н. С. В. Знаменский*

Пример ссылки на эту публикацию:

Ю. В. Шевчук. «Vbinary: ещё раз о представлении целых чисел с переменной разрядностью». *Программные системы: теория и приложения*, 2018, **9**:4(39), с. 477–491. 10.25209/2079-3316-2018-9-4-477-491
http://psta.psiras.ru/read/psta2018_4_477-491.pdf

Об авторе:

Юрий Владимирович Шевчук

Заведующий лабораторией телекоммуникаций и мультипроцессорных систем Института программных систем им, А,К. Айламазяна РАН, к.т.н. Сфера интересов: системное программирование, цифровая электроника, компьютерные сети, сенсорные сети, глобальный мониторинг и управление, распределенное программирование

0000-0002-2327-0869
e-mail: sizif@botik.ru

Приложение А. Синтаксис имён кодов Vbinary в нотации ABNF

```
vbinary-name  = "vbinary" basewidth
vbinary-name /= "vbinary" basewidth repeat-rule
vbinary-name /= "vbinary" basewidth "x" [*midlevel lastlevelx]
vbinary-name /= "vbinary" basewidth "x" [*midlevel] lastlevel repeat-rule
midlevel      = width "x"
      ; 1-ary non-terminal extension
midlevel     /= "(" 1*(width ",") width "x" ")"
midlevel     /= "(" *(width ",") width "x," *(width ",") width ")"
      ; n-ary non-terminal extensions
lastlevelx    = width "x"
      ; 1-ary non-terminal extension
lastlevel     = width
      ; 1-ary terminal extension
lastlevel    /= "(" 1*(width ",") width ")"
      ; n-ary terminal extension
lastlevel    /= "(" 1*(width ",") width "x" ")"
lastlevel    /= "(" *(width ",") width "x," *(width ",") width ")"
      ; n-ary non-terminal extensions
repeat-rule   = addmul
      ; for 1-ary lastlevel
repeat-rule  /= "(" 1*(addmul ",") addmul ")"
      ; for n-ary lastlevel
addmul        = add / mul / mul add
add           = "a" increment
mul           = "m" factor
rational      = 1*DIGIT
rational     /= 1*DIGIT "/" 1*DIGIT
basewidth     = width
width         = 1*DIGIT
increment     = rational
factor        = rational
```

Sample citation of this publication:

Yury Shevchuk. “Vbinary: variable length integer coding revisited”. *Program Systems: Theory and Applications*, 2018, **9**:4(39), pp. 477–491. (*In Russian*).
10.25209/2079-3316-2018-9-4-477-491
http://psta.psiras.ru/read/psta2018_4_477-491.pdf

The same article in English: 10.25209/2079-3316-2018-9-4-239-252