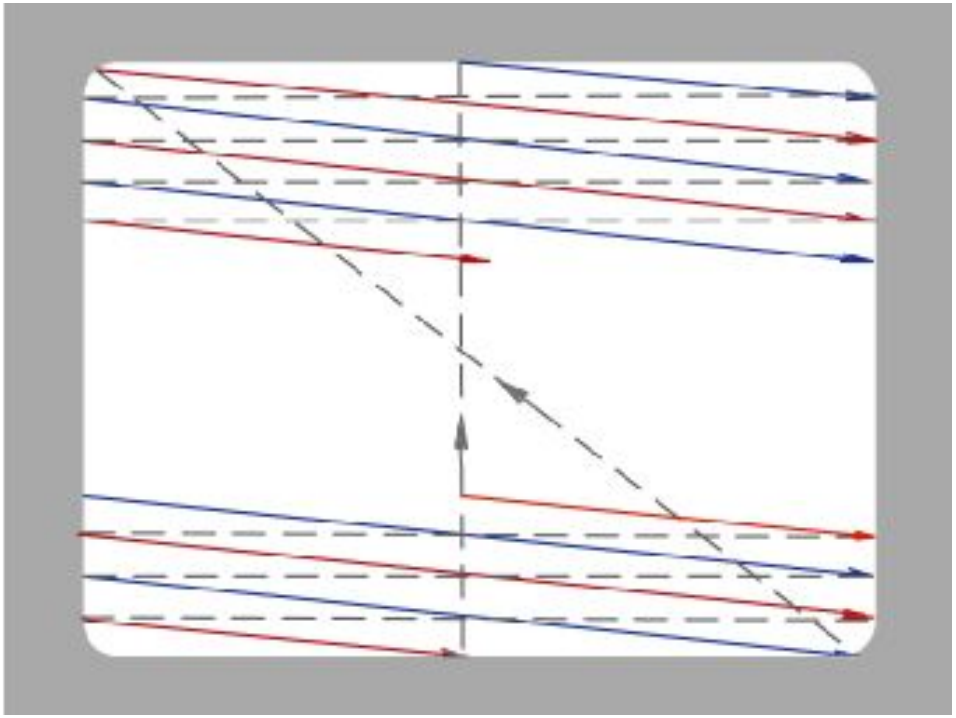
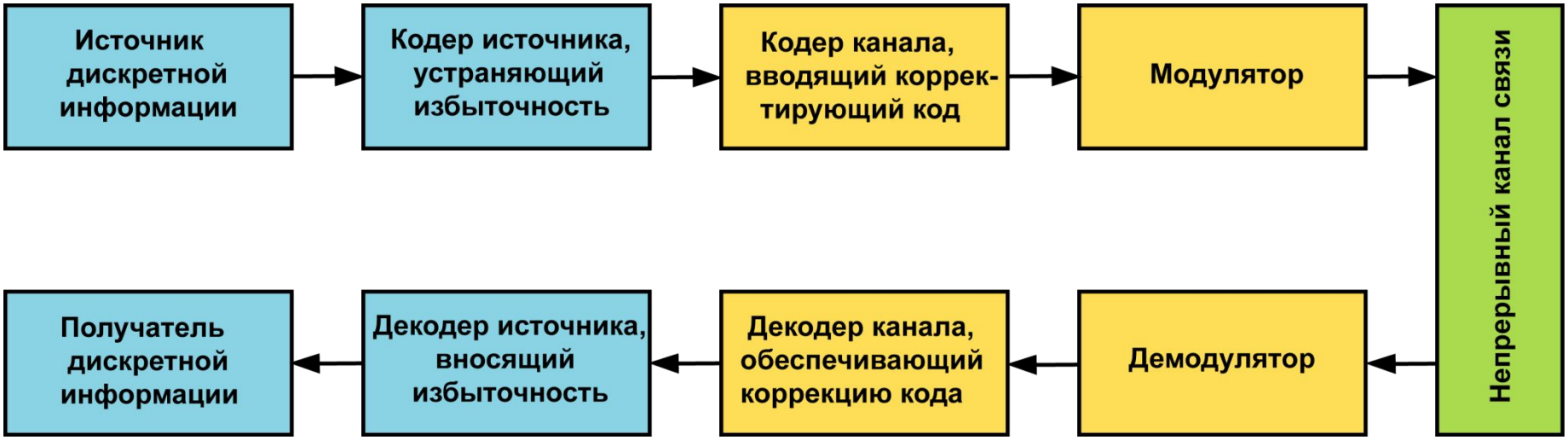
ТВ растр с чересстрочной разверткой



Система передачи данных



Приложения	Размер кадра	Модель цветности, бит на пиксел	Формат экрана	Частота кадров в сек., развертка	Скорость потока, Мбит/с	Стандарты компрессии	Скорость сжатого потока
Видеотелефония	subQCIF, QSIF, QCIF	4:2:0, 12	4:3	1 – 5 прогрессивная	0,141 – 1,45	H.261, H.263, H.264, MPEG-4	9 – 128 кбит/с
Видеоконференцсвязь	QSIF, QCIF, SIF, CIF	4:2:0, 12	4:3	5 – 30 прогрессивная	1,10 – 34,8	H.261, H.263, H.264, MPEG-4	32 кбит/с – 1,5 Мбит/с
Домашнее видео, видео по требованию	SIF	4:2:0, 12	4:3	30 прогрессивная	26,4	MPEG-1, MPEG-2, MPEG-4	0,7 – 1,5 Мбит/с
	CIF			25 прогрессивная	29,0		
Стандартное ТВ	640x480 (NTSC)	4:2:0, 12 4:2:2, 16	4:3	30 черестрочная	105 – 141	MPEG-2, MPEG-4	4 – 20 Мбит/с
	720x576 (PAL, SECAM)			25 черестрочная	119 – 158		

Приложения		Размер кадра	Модель цветности, бит на пиксел	Формат экрана	Частота кадров в сек., развертка	Скорость потока, Мбит/с	Стандарты компрессии	Скорость сжатого потока
ТВ повышенной четкости		1280x720	4:2:0, 12 4:2:2, 16	16:9	25, 30, 50, 60 прогрессивная	264 – 844	MPEG-2, MPEG-4	10 – 40 Мбит/с
ТВЧ		1920x 1080	4:2:2, 16	16:9	25, 30 чересстрочная	791 – 949	MPEG-2, MPEG-4	20 – 60 Мбит/с
					25, 30, 50, 60 прогрессивная	791 – 1900		
Цифровое кино	H0	1920x 1080	4:2:2, 20	16:9	60 прогрессивная	2370	MPEG-4	60 – 80 Мбит/с
	H1	3840x 2160				9490		100 – 150 Мбит/с
	H2	5760x 3240	4:4:4, 36			38400		150 – 600 Мбит/с
	H3	7680x 4320				68300		

Статистическое кодирование

Сокращение избыточности информации без потерь

Типы избыточности

- статистическая избыточность, связанная с корреляцией и предсказуемостью данных; эта избыточность может быть устранена без потери информации, исходные данные при этом могут быть полностью восстановлены
- субъективная избыточность (или избыточность восприятия), которую можно устранить с частичной потерей данных, мало влияющих на качество восприятия информации человеком; такая избыточность характерна для звуковой и видеоинформации

Меры избыточности

- Коэффициент сжатия $C_R = N_2/N_1$
- Среднее число бит на информационный символ

- При статистическом кодировании изображений степень сжатия – 1,5-3
- При устранении визуальной избыточности отдельных изображений – 8-12 раз без видимых искажений

Статистическая избыточность дискретных данных

$$\hat{L}(B) = \sum_{i=1}^{N} L(b_i) \cdot P(b_i) \geq H(B)$$

$$H(B) = -\sum_{i=1}^{N} P(b_i) \cdot \log_2 P(b_i) \geq 0$$

$$H(B)_{\max} = \log_2 N$$

$$R(B) = 1 - \frac{H(B)}{H(B)_{\max}}$$

Классификация методов статистического кодирования

Три задачи статистического кодирования:
- построение информационной модели
- генерация кода
- хранение описания способа кодирования

Коды переменной длины

Код переменной длины C отображает каждый символ a_i алфавита $X = \{a_1, \ldots, a_N\}$ на двоичную строку $C(a_i)$, называемую кодовым словом. Количество бит в кодовом слове $C(a_i)$ называется его длиной $l(a_i)$.

Кодовые слова передаются последовательно без указания их границ. Декодер должен уметь различать границы кодовых слов, начиная с какой-то стартовой позиции. Отсюда вытекает необходимость однозначной декодируемости кодов переменной длины. При этом предполагается, что начальное положение декодирования известно (проведена начальная синхронизация).

Однозначная декодируемость требует, во-первых, чтобы $C(a_i) \neq C(a_j)$ для любых $i \neq j$. Также необходимо, чтобы кодовые слова были различимы в строке, составленной из любой их последовательности.

Коды переменной длины

По определению, код C является однозначно декодируемым, если для любой последовательности символов источника данных $x_1, x_2, \ldots, x_n$ соответствующая последовательность кодовых слов $C(x_1)C(x_2)\ldots C(x_n)$ отличается от последовательности кодовых слов $C(x'_1)C(x'_2)\ldots C(x'_n)$ любой другой последовательности символов источника $x'_1, x'_2, \ldots, x'_n$.

Рассмотрим, например, два варианта кодов для алфавита, состоящего из трех символов: $X = \{a_1, a_2, a_3\}$.

Первый вариант кодов: $C_1(x_1) = 1$; $C_1(x_2) = 00$; $C_1(x_3) = 01$. Такие коды однозначно декодируемы.

Второй вариант кодов: $C_1(x_1) = 1$; $C_1(x_2) = 0$; $C_1(x_3) = 01$. Такие коды не обладают свойством однозначной декодируемости, так как коды последовательности символов x_2, x_1 совпадают с кодом символа x_3.

Свойство однозначной декодируемости зависит от множества кодовых слов и не зависит от способа отображения символов алфавита на кодовые слова.

Методы представления целых чисел кодами переменной длины

Число	Унарные коды	
0	0	1
1	10	01
2	110	001
3	1110	0001
4	11110	00001
5	111110	000001
…	…	…

Гамма- и дельта-коды Элиаса

Для диапазона значений $[2^k, 2^{k+1}-1]$ числа *n* коды формируются:
- гамма-код: унарное представление числа k, за которым следует двоичное представление числа $(n - 2^k)$ (длина кода – 2k + 1 бит);
- дельта-код: гамма-код для числа (k + 1), за которым следует двоичное представление числа $(n - 2^k)$ (длина кода – 2L + k + 1 бит, $L = [\log_2(k + 1)]$.

Диапазон	Гамма-коды Элиаса	Длина кода, бит	Дельта-коды Элиаса	Длина кода, бит
1	1	1	1	1
2-3	01x	3	010x	4
4-7	001xx	5	011xx	5
8-15	0001xxx	7	00100xxx	8
16-31	00001xxxx	9	00101xxxx	9
32-63	000001xxxxx	11	00110xxxxx	10
64-127	0000001xxxxxx	13	00111xxxxxx	11
128-255	00000001xxxxxxx	15	0001000xxxxxxx	14

Коды Голомба и Райса

Для кодирования символа с номером n необходимо представить этот номер в виде: $n = qm + r$, где q и r - целые положительные числа, $0 \le r < m$, m – параметр кодов.
Кодируемое число разбивается на две независимо кодируемые части: частное и остаток от деления на m: $q = [n/m]$ и $r = n - mq$.
Частное q кодируется унарным кодом, а остаток r, представляющий собой число в диапазоне $[0, \ldots, m - 1]$, кодируется бинарным кодом длиной $[\log_2 m]$. Полученные двоичные последовательности объединяются в результирующее слово.

Пример: параметр кода $m = 4$, кодируемое число $n = 13$.
$q = [n/m] = [13/4] = 3$, унарный код $a(q) = a(3) = 1110$
$r = n - m \cdot q = 13 - 4 \cdot 3 = 1$ (число в диапазоне $[0, \ldots, m - 1] = [0, \ldots, 3]$), бинарный код $b(r) = b(1) = 01$
результирующее кодовое слово - $a(q) \mid b(r) = a(3) \mid b(1) = 1110 \mid 01$.

Коды Райса – частный случай кодов Голомба, когда m - степень двойки. Коды Райса различаются параметром k, связанным со значением m соотношением $m = 2^k$ (при $k = 0$, $m = 1$, коды Голомба и Райса соответствуют стандартному унарному коду).

Омега-коды Элиаса и коды Ивен-Роде

Коды состоят из последовательности групп длинной L_1, L_2, L_3, …, L_m бит, которые начинаются с 1, а в конце последовательности следует 0.
Длина каждой следующей (n + 1)-й группы задается значением битов предыдущей n-й группы. Значение битов последней группы является итоговым значением всего кода (первые m − 1 групп служат для указания длины последней группы).
В омега-кодах Элиаса длина первой группы – 2 бита. Длина следующей группы на единицу больше значения предыдущей. Первое значение (1) задается отдельно.
В кодах Ивэн-Родэ длина первой группы – 3 бита, а длина каждой последующей группы равна значению предыдущей. Первые четыре значения (0 - 3) заданы особым образом.

Диапазон	Омега-коды Элиаса	Длина кода, бит	Коды Ивен-Роде	Длина кода, бит
0	-	-	000	3
1	0	1	001	3
2-3	xx0	3	0xx	3
4-7	10xxx0	6	xxx0	4
8-15	11xxxx0	7	100xxxx0	8
16-31	10100xxxxx0	11	101xxxxx0	9
32-63	10101xxxxxx0	12	110xxxxxx0	10
64-127	10110xxxxxxx0	13	111xxxxxxx0	11
128-255	10111xxxxxxxx0	14	1001000xxxxxxxx0	16
256-511	111000xxxxxxxxx0	16	1001001xxxxxxxxx0	17

Коды Фибоначчи

$n \setminus f_i$	1	2	3	5	8	13	21	34	55
1	1	(1)							
2	0	1	(1)						
3	0	0	1	(1)					
4	1	0	1	(1)					
5	0	0	0	1	(1)				
6	1	0	0	1	(1)				
7	0	1	0	1	(1)				
8	0	0	0	0	1	(1)			
••	••	••	••	••	••	••	••	••	••
12	1	0	1	0	1	(1)			
13	0	0	0	0	0	1	(1)		
••	••	••	••	••	••	••	••	••	••
20	0	1	0	1	0	1	(1)		
21	0	0	0	0	0	0	1	(1)	
••	••	••	••	••	••	••	••	••	••
33	1	0	1	0	1	0	1	(1)	
34	0	0	0	0	0	0	0	1	(1)
••	••	••	••	••	••	••	••	••	••

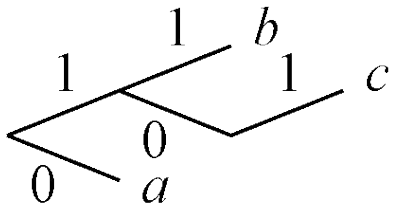
Уникально-префиксные (безпрефиксные) коды

- если существует набор однозначно-декодируемых кодов с определенным множеством длин кодовых слов, то можно легко построить уникально-префиксный код с таким же набором длин кодовых слов
- кодовое слово уникально-префиксного кода может быть декодировано сразу после получения последнего бита этого кодового слова
- при заданном распределении вероятностей символов источника данных можно легко сконструировать уникально-префиксный код минимально возможной длины

Префиксом строки $a_1a_2\ldots a_n$ называется начальная подстрока этой строки $a_1a_2\ldots a_m$, где $m < n$. Код является уникально-префиксным, если ни одно кодовое слово этого кода не является префиксом никакого другого кодового слова.

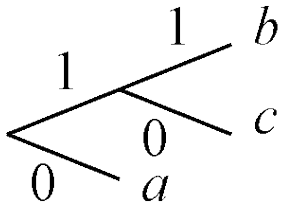
Двоичное кодовое дерево

а)



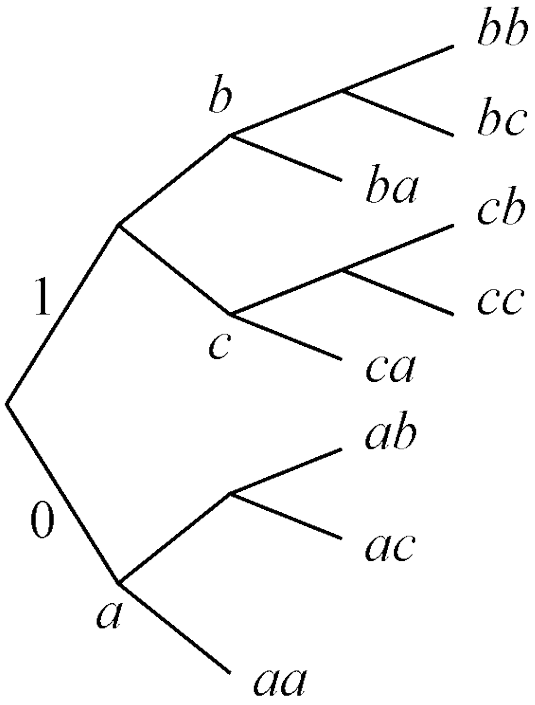
$a \to 0$
$b \to 11$
$c \to 101$

б)



$a \to 0$
$b \to 11$
$c \to 10$

в)



$aa \to 00$
$ab \to 011$
$ac \to 010$
$ba \to 110$
$bb \to 1111$
$bc \to 1110$
$ca \to 100$
$cb \to 1011$
$cc \to 1010$

Уникально-префиксные коды однозначно декодируемы, но однозначно декодируемые коды не обязательно обладают уникальными префиксами. Например, коды $C(a) = 0$; $C(b) = 01$; $C(c) = 011$ могут быть однозначно декодированы, так как 0 в данном случае означает начало нового кодового слова, но эти коды имеют одинаковые префиксы.

Синхронизация кодов переменной длины

При передаче кодов переменной длины в канале с ошибками декодер может потерять синхронизацию и допустить ошибочное декодирование одного или более символов. Поэтому важным вопросом является синхронизируемость кодов переменной длины.

Например, уникально-префиксный код {0, 10, 110, 1110, 11110} является мгновенно самосинхронизируемым, потому что каждый 0 является признаком окончания кодового слова.

Более короткий код {0, 10, 110, 1110, 1111} является вероятностно самосинхронизируемым: каждый 0 является признаком окончания кодового слова, но так как может встречаться последовательность кодовых слов 1111 произвольной длины, время до повторной синхронизации является случайной величиной.

Неравенство Крафта

Неравенство Крафта описывает условия, определяющие возможность построения уникально-префиксных кодов для заданного алфавита $X = \{a_1, \ldots, a_M\}$ при заданном множестве длин кодовых слов $\{l(a_j);\ 1 \le j \le M\}$.

Теорема о неравенстве Крафта формулируется следующим образом: любой уникально-префиксный код для алфавита $X = \{a_1, \ldots, a_M\}$ с длинами кодовых слов $\{l(a_j);\ 1 \le j \le M\}$ удовлетворяет следующему условию:

$$\sum_{j=1}^{M} 2^{-l(a_j)} \le 1$$

И наоборот, если это условие выполнено, то существует уникально-префиксный код с длинами кодовых слов $\{l(a_j);\ 1 \le j \le M\}$. Более того, любой полный уникально-префиксный код удовлетворяет неравенство Крафта при строгом равенстве, а любой неполный уникально-префиксный код удовлетворяет неравенство Крафта при строгом неравенстве.

Например, эта теорема означает, что существует полный уникально-префиксный код с длинами кодовых слов {1, 2, 2}, но не существует уникально-префиксных кодов с длинами кодовых слов {1, 1, 2}.

Неравенство Крафта

Можно представить кодовые слова в виде двоичных дробей в двоичной системе счисления. Двоичная дробь $0,y_1y_2\ldots y_l$ представляет собой запись действительного числа:

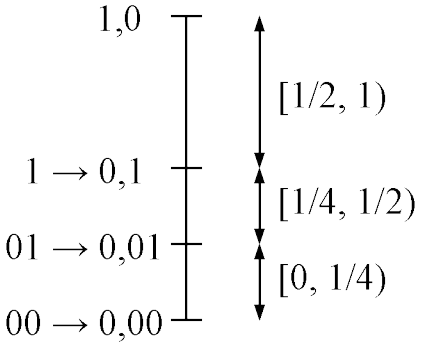
$$\sum_{m=1}^{l} y_m 2^{-m}, \quad y_m \in \{0;1\}$$

Как и в случае десятичных дробей, двоичные дроби можно использовать для аппроксимации действительных чисел с определенной точностью. Двоичная дробь $0,y_1y_2\ldots y_l$ является аппроксимацией чисел из интервала:

$$\left[\sum_{m=1}^{l} y_m 2^{-m}, \sum_{m=1}^{l} y_m 2^{-m} + 2^{-l}\right)$$

Этот интервал размером 2^{-l} включает все числа, представление которых в виде двоичной дроби начинается с $0,y_1y_2\ldots y_l$.

Любое кодовое слово $C(a_j)$ длины l можно записать в виде действительного числа на интервале [0, 1), представляющего интервал размером 2^{-l}, который включает все строки, содержащие $C(a_j)$ как префикс.



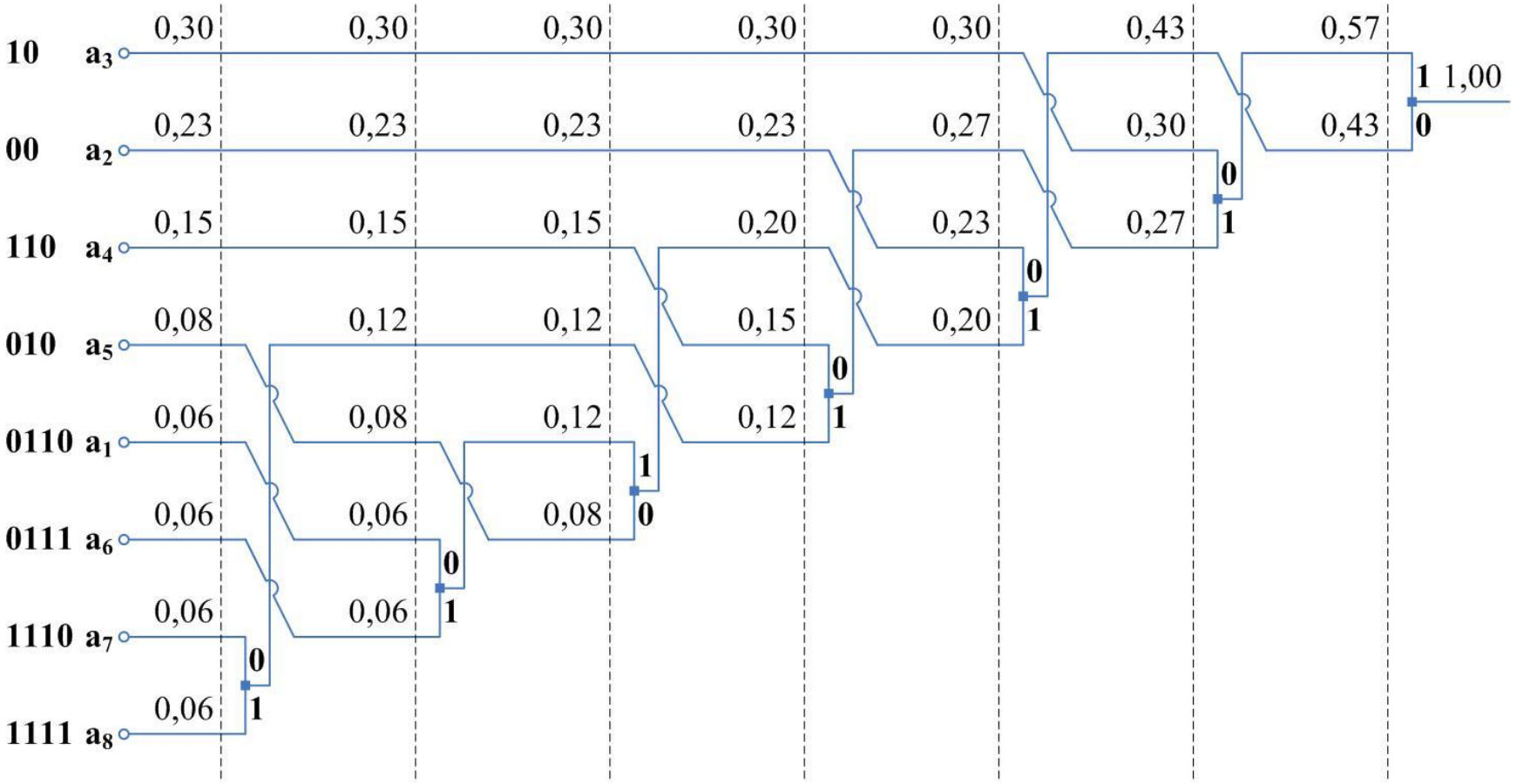
Коды Шеннона-Фано

Зная вероятности символов, строят таблицу кодов, обладающую следующими свойствами:
- различные коды имеют различное количество бит;
- коды символов, обладающих меньшей вероятностью, имеют больше бит, чем коды символов с большей вероятностью;
- хотя коды имеют различную битовую длину, они могут быть декодированы единственным образом.

Символы алфавита сортируются по убыванию вероятностей их появления, упорядоченный ряд разбивается на две части, чтобы суммы вероятностей появления символов, отнесенных к каждой части были бы примерно равны. Символам первой части присваивается код «0», а второй части – «1».

Разделенным на две составляющие первой части присваиваются коды соответственно «00» и «01», а второй части – «10» и «11» и.т. д.

Алгоритм Хаффмана



$$H(S) \le \hat{L} \le H(S)+1 \qquad \hat{L} \ge 1 \qquad \hat{L} = 2{,}71 \qquad H = 2{,}68$$

Блочное и условное кодирование

$$\vec{a} = (a_1, a_2, ..., a_N) \qquad H(\vec{A}) = -\sum_{\vec{a}} P(\vec{a}) \cdot \log_2 P(\vec{a}) \qquad \frac{1}{N} \cdot H(\vec{A}) \le H(A)$$

$$\hat{L} = \frac{1}{N} \cdot \hat{L}_N \qquad \frac{1}{N} \cdot H(\vec{A}) \le \frac{1}{N} \cdot \hat{L}_N = \hat{L} \le \frac{1}{N} \cdot H(\vec{A}) + \frac{1}{N} \le H(A) + \frac{1}{N}$$

$$H(A_N | a_1, a_2, ..., a_{N-1}) = -\sum_{a_N} P(a_N | a_1, a_2, ..., a_{N-1}) \cdot \log_2 P(a_N | a_1, a_2, ..., a_{N-1})$$

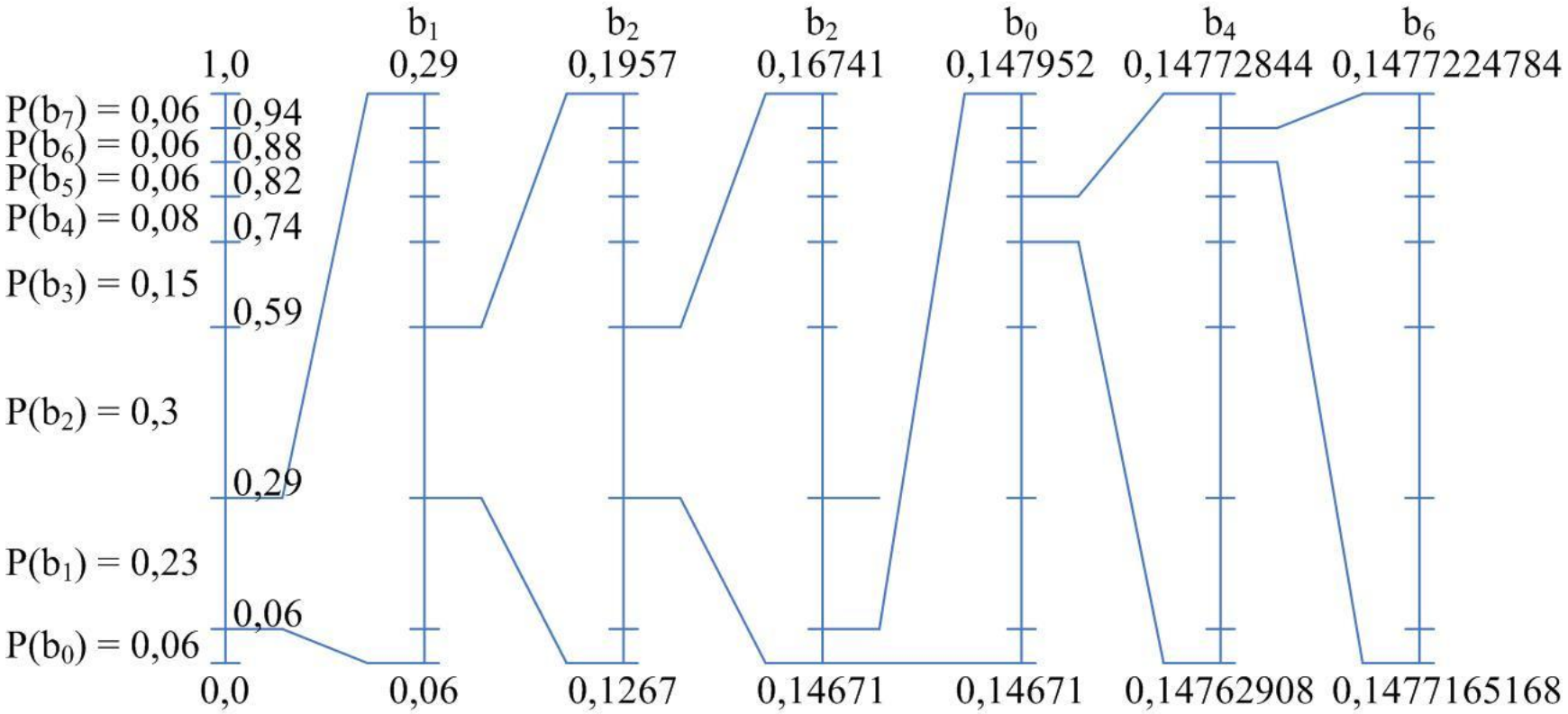
$$H(A_N | A_1, A_2, ..., A_{N-1}) = -\sum_{a_1, ..., a_{N-1}} P(a_N | a_1, a_2, ..., a_{N-1}) \cdot H(A_N | a_1, a_2, ..., a_{N-1}) =$$

$$-\sum_{\vec{a}} P(\vec{a}) \cdot \log_2 P(a_N | a_1, a_2, ..., a_{N-1}).$$

$$H(A_N | A_1, A_2, ..., A_{N-1}) \le \hat{L}_C \le H(A_N | A_1, A_2, ..., A_{N-1}) + 1, \quad \hat{L}_C \ge 1.$$

$$H(A_N | A_1, A_2, ..., A_{N-1}) \le \frac{1}{N} \cdot H(\vec{A})$$

Арифметическое кодирование



Словарные методы кодирования

- Словарные методы энтропийного кодирования вместо вероятностного используют следующий подход: кодовые схемы используют коды только тех информационных последовательностей, которые реально порождаются информационным источником
- Словарные модели опираются на информационную структуру, реализуемую словарем; словарь включает в себя части уже обработанной информации, на основе которых осуществляется кодирование
- Составляющие последовательности символов информационного источника кодируются посредством ссылок на идентичные им элементы словаря (совпадения)
- Словарные методы отличаются друг от друга способом организации словаря, схемой поиска совпадений и видом ссылки на найденное совпадение
- Впервые эти методы были описаны в работах А. Лемпела (A. Lempel) и Я. Зива (J. Ziv), первый вариант словарного алгоритма был описан в 1977 году и был назван LZ-77 по первым буквам фамилий авторов

Словарное кодирование: LZ77

В скользящем окне помещается *N* символов, причем часть из них *M* = *N* - *n* – уже закодированные символы, являющиеся словарем. Предположим, к текущему моменту закодировано *m* символов: s_0, s_1, s_2, …, s_{m-1}, часть из которых s_{m-M}, …, s_{m-2}, s_{m-1} записаны в словаре, а в буфере записаны ожидающие кодирования символы s_m, s_{m+1}, …, s_{m-1+n}.

Шаг	Скользящее окно		Совпадающая фраза	Закодированные данные		
	Словарь	Буфер		*i*	*j*	*s*
1	-	*на_дворе*	-	1	0	*«н»*
2	н	*а_дворе_*	-	1	0	*«а»*
3	на	*_дворе_т*	-	1	0	*«_»*
4	на_	*дворе_тр*	-	1	0	*«д»*
5	на_д	*воре_тра*	-	1	0	*«в»*
6	на_дв	*оре_трав*	-	1	0	*«о»*
7	на_дво	*ре_трава*	-	1	0	*«р»*
8	на_двор	*е_трава_*	-	1	0	*«е»*
9	на_дворе	*_трава_н*	_	6	1	*«т»*
10	на_дворе_т	*рава_на_*	р	4	1	*«а»*
11	на_дворе_тра	*ва_на_тр*	в	8	1	*«а»*
12	на_дворе_трава	*_на_трав*	_	6	1	*«н»*
13	на_дворе_трава_н	*а_траве*	а_	15	2	*«т»*
14	на_дворе_трава_на_т	*раве_дро*	рав	9	3	*«е»*
15	на_дворе_трава_на_траве	*_дрова_н*	-д	21	2	*«р»*
16	на_дворе_трава_на_траве_др	*ова_на_д*	о	21	1	*«в»*
17	на_дворе_трава_на_траве_дров	*а_на_дво*	а_на_	15	5	*«д»*
18	на_дворе_трава_на_траве_дрова_на_д	*воре_тра*	воре_тра	30	8	*«в»*
19	…. _на_траве_дрова_на_дворе_трав	*а_на_тра*	а_на_тра	30	8	*«в»*
20	…. _дрова_на_дворе_трава_на_трав	*е_дрова*	е_дрова	30	7	*«end»*

Словарное кодирование: LZ78

Шаг	Добавляемая в словарь фраза		Буфер	Совпадающая фраза	Закодированные данные	
	Фраза	Ее номер			*N*	*S*
1	н	2	*на_дворе*	-	1	*«н»*
2	а	3	*а_дворе_*	-	1	*«а»*
3	_	4	*_дворе_т*	-	1	*«_»*
4	д	5	*дворе_тр*	-	1	*«д»*
5	в	6	*воре_тра*	-	1	*«в»*
6	о	7	*оре_трав*	-	1	*«о»*
7	р	8	*ре_трава*	-	1	*«р»*
8	е	9	*е_трава_*	-	1	*«е»*
9	_т	10	*_трава_н*	_	4	*«т»*
10	ра	11	*рава_на_*	р	8	*«а»*
11	ва	12	*ва_на_тр*	в	6	*«а»*
12	_н	13	*_на_трав*	_	4	*«н»*
13	а_	14	*а_траве_*	а	3	*«_»*
14	т	15	*траве_др*	-	1	*«т»*
15	рав	16	*раве_дро*	ра	11	*«в»*
16	е_	17	*е_дрова*	е	9	*«_»*
17	др	18	*дрова_на*	д	5	*«р»*
18	ов	19	*ова_на_д*	о	7	*«в»*
19	а_н	20	*а_на_дво*	а_	14	*«н»*
20	а_д	21	*а_дворе*	а_	14	*«д»*
21	во	22	*воре_тра*	в	6	*«о»*
22	ре	23	*ре_трава*	р	8	*«е»*
23	_тр	24	*_трава_н*	_т	10	*«р»*
24	ав	25	*ава_на_т*	а	3	*«в»*
25	а_на	26	*а_на_тра*	а_н	20	*«а»*
26	_тра	27	*_траве_д*	_тр	24	*«а»*
27	ве	28	*ве_дрова*	в	6	*«е»*
28	_д	29	*_дрова*	_	4	*«д»*
29	ро	30	*рова*	р	8	*«о»*
30	ва	31	*ва*	ва	12	«end»

Словарное кодирование: LZW

Шаг	Входная строка	Закодированные данные	Фраза в словарь	Код
1	на_дворе_трава_на_траве_дрова_на_дворе_.....	*«н»*	на	256
2	а_дворе_трава_на_траве_дрова_на_дворе_т……..	*«а»*	а_	257
3	_дворе_трава_на_траве_дрова_на_дворе_тр……..	*«_»*	_д	258
4	дворе_трава_на_траве_дрова_на_дворе_тра……..	*«д»*	дв	259
5	воре_трава_на_траве_дрова_на_дворе_трав……..	*«в»*	во	260
6	оре_трава_на_траве_дрова_на_дворе_трава	*«о»*	ор	261
7	ре_трава_на_траве_дрова_на_дворе_трава	*«р»*	ре	262
8	е_трава_на_траве_дрова_на_дворе_трава_.....	*«е»*	е_	263
9	_трава_на_траве_дрова_на_дворе_трава_н…….	*«_»*	_т	264
10	трава_на_траве_дрова_на_дворе_трава_на…….	*«т»*	тр	265
11	рава_на_траве_дрова_на_дворе_трава_на_.....	*«р»*	ра	266
12	ава_на_траве_дрова_на_дворе_трава_на_т…….	*«а»*	ав	267
13	ва_на_траве_дрова_на_дворе_трава_на_тра……..	*«в»*	ва	268
14	а_на_траве_дрова_на_дворе_трава_на_трав……..	*«257»* (а_)	а_н	269
15	на_траве_дрова_на_дворе_трава_на_траве_дрова	*«256»*(на)	на_	270
16	_траве_дрова_на_дворе_трава_на_траве_дрова	*«264»*(_т)	_тр	271
17	раве_дрова_на_дворе_трава_на_траве_дрова	*«266»*(ра)	рав	272
18	ве_дрова_на_дворе_трава_на_траве_дрова	*«в»*	ве	273
19	е_дрова_на_дворе_трава_на_траве_дрова	*«263»*(е_)	е_д	274
20	дрова_на_дворе_трава_на_траве_дрова	*«д»*	др	275
21	рова_на_дворе_трава_на_траве_дрова	*«р»*	ро	276
22	ова_на_дворе_трава_на_траве_дрова	*«о»*	ов	277
23	ва_на_дворе_трава_на_траве_дрова	*«268»*(ва)	ва_	278
24	_на_дворе_трава_на_траве_дрова	*«_»*	_н	279
25	ра_дворе_трава_на_траве_дрова	*«270»*(на_)	на_д	280
26	дворе_трава_на_траве_дрова	*«259»*(дв)	дво	281
27	оре_трава_на_траве_дрова	*«261»*(ор)	оре	282
28	е_трава_на_траве_дрова	*«263»*(е_)	е_т	283
29	трава_на_траве_дрова	*«265»*(тр)	тра	284
30	ава_на_траве_дрова	*«267»*(ав)	ава	285
31	а_на_траве_дрова	*«269»*(а_н)	а_на	286
32	а_траве_дрова	*«257»*(а_)	а_т	287
33	траве_дрова	*«284»*(тра)	трав	288
34	ве_дрова	*«273»*(ве)	ве_	289
35	_дрова	*«258»*(_д)	_др	290
36	рова	*«276»*(ро)	ров	291
37	ва	*«268»*(ва)		«end»

Ассоциативное кодирование Буяновского

Предположим, что уже закодирована последовательность
..1111010111000010100100010101001011 10
и предстоит закодировать строку
0100010111.., обозначенную далее как «пакуемая строка».

Закодированная последовательность представляет предысторию, пакуемая строка – постисторию:
..1111010111000010100100010101001011 10|0100010111..

Выпишем все подпоследовательности из предыстории, которые совпадают с первым и более начальными битами предыстории (отсчет бит ведется справа налево, в данном случае первый бит равен «0»):

Ассоциативное кодирование Буяновского

..11110|10111000010100100010101001011110
..1111010|11100001010010001010100101110
..11110101110|00010100100010101001011110
..11110101110|00101001000101010010111100
..11110101110|01010010000101010010111000
..11110101110|10100100010101001011110000
..11110101110|10010001010100101110
..11110101110|01000101010010111000010
..11110101110|10001010100101110000010100
..11110101110|00101010010010111000010100
..11110101110|01010100100111000010100
..11110101110|10101001000111000010100
..11110101110|01001000101110000010100
..11110101110|00100010111000010100
..11110101110|10001011100010100
..11110101110|01011100010100
..11110101110|10111000101010010|1110
..11110101110000101001000101010010111000010100..

Максимальная длина предыстории ограничивается некоторым числом N.

Ассоциативное кодирование Буяновского

Полученные последовательности упорядочиваются по возрастанию лексикографического порядка предыстории. При этом анализируется последовательность символов справа налево начиная от символа «|». Т. е. последовательность ..110000| лексикографически меньше последовательности ..001000|. Слева проставлен номер строки. Пакуемой строке (с ее предысторией) присваивается номер 0, от нее номера вверх возрастают, а вниз убывают:

Ассоциативное кодирование Буяновского

```
15                                  ..11110101110000|1010010001010100101110
14                              ..1111010111000010100100|1010100101110
13                                  ..1111010111000|0101001000101010010111O
12                              ..1111010111000010100100|01010100101110
11                                  ..1111010111000010100|1000101010010111O
10                          1110101110000101001000101010010|101110
 9                                  ..111101011100|0010100100010101001011O
 8                                  ..1111010111000010|100100010101001011O
 7                              ..1111010111000010100100010|10100101110
 6                                  ..1111010111000010100010|001010100101110
 5                      ..1111010111000010100100010101010010|1110
 4                                  ..1111010111000010100|0100010101001011O
 3                          ..1111010111000010100100010100|100101110
 2                       ..1111010111000010100100010101010|0101110
 1                                              ..1111010|1110000101001000101010010111O
 0  ..1111010111000010100100010101001011O|0100010111..
-1                                          ..11110101110|0001010010001010100101110
-2                                              ..11110|1011100001010010001010100101110
```

Этот список называют левосторонним ассоциативным списком или «воронкой аналогий» в терминологии автора алгоритма.

Ассоциативное кодирование Буяновского

Затем этот список переупорядочивается в лексикографическом порядке постистории (в противоположном направлении - слева направо после «|», т.е. последовательность |110000.. лексикографически больше последовательности |001000..):

-1 ..11110101110|000101001000101010010111 0
9 ..111101011100|001010010001010100101110
6 ..1111010111000010100 10|001010100101110
4 ..111101011100001010|010001010100101110
0 ..11110101110000101001000101010010111 0|0100010111..
13 ..1111010111000|010100100010101001011 10
12 ..11110101110000101001 00|01010100101110
2 ..1111010111000010100100010101 0|0101110
11 ..111101011100001010 0|100010101001011 10
8 ..11110101110000 10|100100010101001011 10
3 ..1111010111000010100100010 10|100101110
15 ..11110101110000|101001000101010010111 0
7 ..1111010111000010100100 010|10100101110
14 ..1111010111000010100 1000|1010100101110
10 ..111101011100001010010001010 100|101110
-2 ..11110|101110000101001000101010010111 0
1 ..1111010|111000010100100010101001011 10
5 ..1111010111000010100100010101 0010|1110

Ассоциативное кодирование Буяновского

В дальнейшем используются только две строки, примыкающие к пакуемой строке:
4 ..1111010111000010 10|0100010101001011 10
0 ..1111010111000010100100010101001011 10|0100010111..
13 ..1111010111000|0101001000101010010111 0
Код пакуемой строки состоит из трех частей.

1. Номер строки, максимально совпадающей с пакуемой строкой. В данном случае – 4. Этот номер кодируется любым эффективным префиксным или арифметическим кодом. В качестве эквивалента вероятности автор алгоритма предложил использовать длину совпадения строки предыстории пакуемой строки со строками предысторий из рассмотренного списка с соответствующей нормировкой.

2. Бит различия – первый бит пакуемой строки, отличающий ее от строки, максимально с ней совпадающей. В данном случае – 1 (отмечен подчеркиванием):
- в пакуемой строке – **01000101<u>1</u>**,
- в максимально близкой к ней 4-й строке – 01000101<u>**0**</u>100101110.
Этим битом определяется положение максимально близкой строки: если бит 0 – то эта строка выше, если 1, то ниже. В данном случае пакуемая строка ниже.

Ассоциативное кодирование Буяновского

3. Длина «вытяжки» (в терминологии автора алгоритма) – количество нулей или единиц в той части пакуемой строки (ниже отмечено подчеркиванием значений **0101**), которая находится между битом различия максимально совпадающей строки (бит отмечен подчеркиванием в нижней строке) и битом различия второй строки (бит отмечен подчеркиванием в верхней строке). Если пакуемая строка лексикографически старше строки, максимально совпадающей с ней, то передается количество нулей, а если младше – то количество единиц. В данном случае подсчитывается количество нулей, которое равно 2.

010**1**0010001010100101110
0100010111..
01000101**0**100101110

Длина «вытяжки» также кодируется адаптивным статистическим кодером.

Таким образом, упакована строка **010001011** (до бита различия включительно). Операция восстановления строки осуществляется следующим образом:

Ассоциативное кодирование Буяновского

Операция восстановления строки осуществляется следующим образом:
1. Декодируется номер строки максимального совпадения (в данном случае 4).
2. По биту различия определяется положение запакованной строки (и соседней строки) – если бит «0», то она выше (лексикографически младше, длина «вытяжки» - количество единиц), в противном случае – ниже (лексикографически старше, длина «вытяжки» - количество нулей) (в данном случае номер соседней строки – 13, она лексикографически старше).
3. Копируется в искомую строку совпадающая часть строки максимального совпадения (4, 010001010100..) и соседней (13, 010100100010..), а также следующий бит из строки максимального совпадения (в данном случае совпадающая часть - 010, а следующий бит – 0).
4. Декодируется длина «вытяжки» (2).
5. Восстанавливается следующая часть искомой строки по «вытяжке» - копируются из строки максимального совпадения следующие биты, включающие соответствующее длине вытяжки количество нулей или единиц (в данном случае два нуля, так как искомая строка лексикографически старше; строка максимального совпадения – 010001010100.., из нее уже скопировано 4 бита, 0100, значит, копируется 010). Далее копируется из строки максимального совпадения все биты до следующего нуля или единицы (в данном случае – до следующего нуля, то есть 1). Таким образом, восстановленная по «вытяжке» часть строки - 0101.
6. Дописывается в искомую строку бит различия (в данном случае – 1, а в строке максимального совпадения следующий бит, естественно, 0).
7. Таким образом, восстановлена строка **010001011**, что и требовалось.