

Современная теория информации

Лекция 6. Алгоритмы сжатия, используемые в архиваторах

Содержание лекции

- 1 Метод "стопка книг".
- 2 Метод скользящего словаря LZ-77.
- 3 Алгоритм LZ-78 \approx LZW;
- 4 Предсказание по частичному совпадению (PPM);
- 5 Преобразование Барроуза-Уиллера;

Метод "стопка книг"¹²

- Пусть $X = \{0, 1, \dots, M - 1\}$ – алфавит источника, на выходе которого наблюдается последовательность x_1, x_2, \dots .
 - Предполагаем, что первой букве предшествуют все буквы алфавита (в алфавитном порядке).
- 1 Интервальное кодирование.
 - Вместо буквы x_n передаётся количество букв (длина интервала) r_n между текущим и предыдущим появлением данной буквы.
 - Пример. $abc|cabbbabbac \rightarrow 0, 3, 3, 0, 0, 3, 1, 0, 2, 9$.
 - 2 "Стопка книг".
 - Вместо буквы x_n передаётся количество **различных** букв d_n между текущим и предыдущим появлением данной буквы.
 - Пример. $abc|cabbbabbac \rightarrow 0, 2, 2, 0, 0, 1, 1, 0, 1, 2$.

¹Рябко Б.Я., Сжатие информации с помощью стопки книг // Проблемы передачи информации, 1980.

²J.Bentley et al., A locally adaptive data compression scheme, 22nd Alteron Conference on Communication Control and Computing, 1984.

Метод "стопка книг"

Пример

- IF_WE_CANNOT_DO_AS_WE_WOULD_WE_SHOULD_DO_AS_WE_CAN
- В ASCII пробел это 32, а буквам A,...,Z соответствуют числа 65...90.
- Результат: 74, 72, 35, 88, 73, 3, 72, 71, 80, 1, 81, 86, 6, 76, 4, 3, 6, 86, 3, 10, 10, 3, 3, 6, 87, 83, 9, 6, 6, 7, 3, 8, 81, 9, 9, 9, 9, 7, 2, 5, 3, 10, 8, 3, 10, 10, 3, 13, 6, 13.
- Применим к $\{d_i\}$ монотонный код:

$$mon(n) \rightarrow \underbrace{111..0}_{unar(\lfloor \log_2 n \rfloor + 1)} \underbrace{bin(n - 2^{\lfloor \log_2 n \rfloor})}_{\lfloor \log_2 n \rfloor \text{ бит}}.$$

$$mon(21) \rightarrow \underbrace{11110}_{10} \underbrace{10101}_{5}.$$

- Получим 332 бита.

Метод скользящего словаря (LZ-77)³

- Кодер LZ-77 хранит в памяти скользящий словарь, объёмом W (последние W букв).
- После кодирования символов x_1, \dots, x_{n-1} кодер ищет в словаре как можно более длинную последовательность (слово) из новых букв x_n, \dots, x_{n+l} .
 - ▶ Если не нашлось, то декодеру передаётся $0|bin(x_n)$.
 - ▶ Если нашлось, то декодеру передаётся $1|bin(d)|vlc(l)$, где
 - ★ d – расстояние от текущей позиции до начала найденного в словаре слова. Кодировается равномерным кодом длины $\lceil \log W \rceil$.
 - ★ l – длина найденного слова. Кодировается неравномерным кодом, например, унарным.

³J. Ziv, A. Lempel, Compression of individual sequences via variable rate coding, IEEE Transactions on Information Theory, 1977.

Метод скользящего словаря (LZ-77)

Пример, когда $vlc(x) = mon(x)$

Шаг	Флаг	(x_1, \dots, x_n)	W	d	l	Код	Биты
0	0	I	0	-	0	0 bin(I)	9
1	0	F	1	-	0	0 bin(F)	9
2	0	_	2	-	0	0 bin(_)	9
3	0	W	3	-	0	0 bin(W)	9
4	0	E	4	-	0	0 bin(E)	9
5	1	_	5	2	1	1 010 0	5
6	0	C	6	-	0	0 bin(C)	9
7	0	A	7	-	0	0 bin(A)	9
8	0	N	8	-	0	0 bin(N)	9
9	1	N	9	0	1	1 0000 0	6
10	0	O	10	-	0	0 bin(O)	9
11	0	T	11	-	0	0 bin(T)	9
12	1	_	12	6	1	1 0110 0	6
13	0	D	13	-	0	0 bin(D)	9
14	1	O	14	3	1	1 0011 0	6
15	1	_	15	2	1	1 0010 0	6

Метод скользящего словаря (LZ-77)

Пример, когда $vlc(x) = mon(x)$

Шаг	Флаг	(x_1, \dots, x_n)	W	d	l	Код	Биты
16	1	A	16	8	1	1 01000 0	7
17	0	S	17	-	0	0 bin(S)	9
18	1	_WE_	18	15	4	1 01111 11000	11
19	1	W	22	2	1	1 00010 0	7
20	1	O	23	8	1	1 01000 0	7
21	0	U	24	-	0	0 bin(U)	9
22	0	L	25	-	0	0 bin(L)	9
23	1	D	26	12	1	1 01100 0	7
24	1	_WE_	27	8	4	1 01000 11000	11
25	1	S	31	13	1	1 01101 0	7
26	0	H	32	-	0	0 bin(H)	9
27	1	OULD_	33	9	5	1 001001 11001	12
28	1	DO_AS_WE_	38	24	9	1 011000 1110001	14
29	1	CAN	47	40	3	1 101000 101	10
Всего бит:							257

Метод скользящего словаря (LZ-77)

Модификации

- Использовать код Хаффмана или арифметическое кодирование для $v/c(x)$.
- Ограничить максимальное l , $W = 2^w$ для более удобной организации памяти⁴.
- Хранение словаря в виде древовидной структуры⁵.

⁴LZFG

⁵E.Fiala, Data compression with finite windows. Communication of the ACM, 1989.

Алгоритм (LZW)⁶

- На первом шаге кодер формирует словарь из *esc*-символа, который помещается в ячейку с номером $i = 0$, объём словаря $C \leftarrow 1$.
- На последующих шагах кодер ищет самое длинное слово в словаре, совпадающее с началом подлежащей кодированию последовательности x_n, \dots, x_{n+l} .

▶ Если не нашлось, то декодеру передаётся $\underbrace{00..0}_{\lceil \log(C-1) \rceil} | \text{bin}(x_n)$.

▶ Если нашлось, то

★ декодеру передаётся $\underbrace{\text{bin}(i)}_{\lceil \log(C-1) \rceil}$, где i – номер найденного слова.

★ словарь дополняется новым словом, которое получается дописыванием к слову с номером i буквы x_n , $C \leftarrow C + 1$.

⁶J. Ziv, A. Lempel, Compression of individual sequences via variable-rate coding, IEEE Transactions on Information Theory, 1980.

Алгоритм (LZW)

Пример

Шаг	Словарь	i	Код	Биты
0	esc	—	—	—
1	I	0	bin(I)	$0+8=8$
2	F	0	bin(F)	$\log(1)+8=8$
3	_	0	0bin(_)	$\lceil \log(2) \rceil + 8 = 9$
4	W	0	00bin(W)	$\lceil \log(3) \rceil + 8 = 10$
5	E	0	00bin(E)	$\lceil \log(4) \rceil + 8 = 10$
6	_ C	3	011	3
7	C	0	000bin(C)	$3+8=11$
8	A	0	000bin(A)	$3+8=11$
9	N	0	000bin(N)	$3+8=11$
10	NO	0	000bin(N)	$4+8=12$
11	O	0	0000bin(O)	$4+8=12$
12	T	0	0000bin(T)	$4+8=12$
13	_ D	3	0011	4
14	D	0	0000bin(D)	$4+8=12$
15	O _	11	1011	4

Алгоритм (LZW)

Пример

Шаг	Словарь	i	Код	Биты
16	<u>_</u> A	3	0011	4
17	A <u>S</u>	8	1000	4
18	S	0	00000bin(S)	5+8
19	<u>_</u> W	3	00011	5
20	W <u>E</u>	4	00100	5
21	E <u>_</u>	5	00101	5
22	<u>_</u> WO	19	10011	5
23	O <u>U</u>	11	01011	5
24	U	0	00000bin(U)	5+8=13
25	L	0	00000bin(L)	5+8=13
26	D <u>_</u>	14	01110	5
27	<u>_</u> WE	19	10011	5
28	E <u>_</u> S	21	10101	5
29	S <u>H</u>	18	10010	5

Алгоритм (LZW)

Пример

Шаг	Словарь	i	Код	Биты
30	H	0	00000bin(H)	$5+8=13$
31	OUL	23	10111	5
32	LD	25	11001	5
33	D_D	26	11010	5
4	DO	14	001110	6
35	O_A	15	001111	6
36	AS_	17	010001	6
37	_WE_	27	011011	6
38	_CA	6	000110	6
39	AN	8	001000	6
40	N	9	001001	6
Всего бит:				291

Оценка избыточности алгоритмов LZ-77 и LZW

- Избыточность LZ-77 с увеличением W определяется как:

$$\bar{R} \approx H_{\infty}(X) + \frac{\log \log W}{\log W}$$

- Избыточность LZW с увеличением C определяется как:

$$\bar{R} \approx H_{\infty}(X) + \frac{\log \log C}{\log C}$$

Контекстное адаптивное арифметическое кодирование

Основная идея

- Если распределение вероятностей вида $\hat{p}(x_{t+1}|x_1, \dots, x_t)$ возможно оценить с достаточной точностью, то арифметический кодер может использовать эти вероятности для достижения максимального сжатия, т.е., среднюю длину кодового слова, близкую к $H(X|X^\infty)$.
- Каждое условие соответствует источнику без памяти, к которому может быть применено универсальное кодирование.
- Избыточность адаптивного кодирования приближается к нулю с ростом длины кодируемой последовательности. Поэтому в каждый контекст должно попасть достаточное количество символов, т.е., контекстов не должно быть слишком много.
- Эффективность такого кодирования сильно зависит от метода оценки вероятностей.

Предсказание по частичному совпадению

Основная идея

Определение

Строка $\mathbf{s} = \mathbf{x}_{t-d+1}^t$ длины $d \leq D$, предшествующая x_{t+1} , является контекстом для x_{t+1} , если \mathbf{s} уже появлялось в \mathbf{x}_1^{t-1} .

Пример. THE _ CAT _ IN _ THE _ CAR _ ATE _ THE _ RAT

t	буква	контекст
16	A	THE _ C
27	R	THE _

Prediction by Partial Matching (PPM)

Основные этапы кодирования символа x_{t+1}

- 1 Выполняется поиск контекста $\mathbf{s} = \mathbf{x}_{t-d+1}^t$ наибольшей длины d , не превышающей D .
- 2 Для всех возможных значений символа x_{t+1} вычисляются оценки условных вероятностей символа при известном контексте \mathbf{s} .
- 3 Значение символа x_{t+1} кодируется арифметическим кодом в соответствии с вычисленной условной вероятностью.

Prediction by Partial Matching (PPM)

Основные этапы кодирования символа x_{t+1}

- D – параметр алгоритма.
- Вероятность того, что символ $x_{t+1} = a$ после контекста $\mathbf{s} = \mathbf{x}_{t-d+1}^t$ может быть оценена как

$$\hat{p}_t(a|\mathbf{s}) = \frac{\tau_t(\mathbf{s}, a)}{\tau_t(\mathbf{s})}.$$

- Если буква для данного контекста не встречалась, т.е. $\tau_t(\mathbf{s}, a) = 0$, то арифметический кодер не сможет использовать $\hat{p}_t(a|\mathbf{s}) = 0$. Поэтому используется *esc*-символ.
- Если $\hat{p}_t(a|\mathbf{s}) = 0$, то передаётся *esc*-символ и контекст укорачивается на одну букву.

Prediction by Partial Matching (PPM)

Основные этапы кодирования символа x_{t+1}

- 1: Находим наибольшее d такое, что $\hat{p}_t(x_{t-d+1}^t) > 0$, $d \leq D$.
- 2: Выбираем контекст $\mathbf{s} = x_{t-d+1}^t$.
- 3: **while** $\hat{p}_t(x_{t+1}|\mathbf{s}) = 0$ **do**
- 4: Кодируем esc в соответствии с $\hat{p}_t(\text{esc}|\mathbf{s})$.
- 5: Уменьшаем длину контекста: $d \leftarrow d - 1$, $\mathbf{s} = x_{t-d+1}^t$.
- 6: **end while**
- 7: **if** $d > 0$ **then**
- 8: Кодируем x_{t+1} в соответствии с $\hat{p}_t(x|\mathbf{s})$.
- 9: **else**
- 10: Кодируем x_{t+1} в соответствии с равномерным распределением на не встречавшихся в x_1^t буквах.
- 11: **end if**

Prediction by Partial Matching (PPM)

Оценки вероятностей алгоритмами A и D

Алгоритм A :

$$\hat{p}_t(a|\#) = \frac{\tau_t(\mathbf{a})}{t+1}; \quad \hat{p}_t(\text{esc}|\#) = \frac{1}{t+1}, \tau_t(\mathbf{a}) > 0$$

$$\hat{p}_t(a|\mathbf{s}) = \frac{\tau_t(\mathbf{s}, a)}{\tau_t(\mathbf{s}) + 1}; \quad \hat{p}_t(\text{esc}|\mathbf{s}) = \frac{1}{\tau_t(\mathbf{s}) + 1}, \tau_t(\mathbf{s}, \mathbf{a}) > 0$$

Алгоритм D :

$$\hat{p}_t(a|\mathbf{s}) = \frac{\tau_t(\mathbf{s}, a) - 1/2}{\tau_t(\mathbf{s})}; \quad \hat{p}_t(\text{esc}|\mathbf{s}) = \frac{M_t(\mathbf{s})}{2\tau_t(\mathbf{s})}, \tau_t(\mathbf{s}, \mathbf{a}) > 0,$$

где $M_t(\mathbf{s})$ – число различных букв, появившихся в последовательности длины t вслед за контекстом \mathbf{s} .

Prediction by Partial Matching (PPM)

Принцип "исключений" (exclusions)

- Пусть $\mathbf{s} = (s_d, \dots, s_1)$ – контекст длины d .
- Если $\hat{p}(x_{t+1}|\mathbf{s}) = 0$, то передаётся *esc*-символ.
- После получения *esc* декодер всё ещё не знает x_{t+1} , но он знает, какие символы не могут быть на этой позиции. Это символы, которые следовали за контекстом \mathbf{s} , когда он появлялся ранее. Это знание позволяет уточнять вероятность $\hat{p}(x_{t+1}|\mathbf{s}' = (s_{d-1}, \dots, s_1))$.
- Список исключаемых символов растёт, если *esc* снова передаётся.
- Исключение некоторых символов увеличивает оценку вероятности для оставшихся символов, т.е., увеличивает сжатие.

Пример.

- Рассмотрим контекст $\mathbf{s} = \text{"кор"}$ за которым следует буква "т", причем "корт" ранее не встречалось.
- Предположим, что ранее после \mathbf{s} встречались "а" и "с".
- Тогда $\tau'_t(\text{ор}) = \tau_t(\text{ор}) - \tau_t(\text{ора}) - \tau_t(\text{орс})$.

Prediction by Partial Matching (PPM)

Оценки вероятностей с учетом исключений

Алгоритм A:

$$\begin{aligned}\hat{p}_t(a|\mathbf{s}) &= \frac{\tau_t(\mathbf{s}, a)}{\tau_t(\mathbf{s}) - \tau_t^*(\mathbf{s}) + 1}; \\ \hat{p}_t(\text{esc}|\mathbf{s}) &= \frac{1}{\tau_t(\mathbf{s}) - \tau_t^*(\mathbf{s}) + 1}, \tau_t(\mathbf{s}, a) > 0\end{aligned}$$

где $\tau_t(\mathbf{s})$ и $\tau_t(\mathbf{s}, a)$ – число строк \mathbf{s} и (\mathbf{s}, a) , соответственно, которые содержатся в \mathbf{x}_1^t , $\tau_t^*(\mathbf{s})$ – число букв, которые не могут следовать за \mathbf{s} .

Если $\tau_t(\mathbf{s}, a) = 0$, но $\tau_t(\mathbf{s}', a) > 0$, то

$$\hat{p}_t(a|\mathbf{s}) = \frac{1}{(\tau_t(\mathbf{s}) - \tau_t^*(\mathbf{s}) + 1)} \hat{p}_t(a|\mathbf{s}')$$

где \mathbf{s}' на одну букву короче \mathbf{s} .

Prediction by Partial Matching (PPM)

Пример РРМА, $D = 5$

Шаг	Буква	Контекст	$\tau_t(s)$	$p_t(esc s)$	$p_t(x s)$
1	I	#	0	1	1/256
2	F	#	1	1/2	1/255
3	_	#	2	1/3	1/254
4	W	#	3	1/4	1/253
5	E	#	4	1/5	1/252
6	_	#	5		1/6
7	C	_	1,6	$1/2 \times 1/6'$	1/251
8	A	#	7	1/8	1/250
9	N	#	8	1/9	1/249
10	N	#	9		1/10
11	O	N	1,10	$1/2 \times 1/9'$	1/248
12	T	#	11	1/12	1/247
13	_	#	12		2/13
14	D	_	2,13	$1/3 \times 1/12'$	1/246
15	O	#	14		1/15
16	_	O	1,15	1/2	3/15'
17	A	_	3,16	1/4	1/14'
18	S	A	1,17	$1/2 \times 1/16'$	1/245
19	_	#	18		4/19

Prediction by Partial Matching (PPM)

Пример РРМА, $D = 5$

Шаг	Буква	Контекст	$\tau_t(s)$	$p_t(esc s)$	$p_t(x s)$
20	W	_	4		1/5
21	E	_W	1		1/2
22		_WE	1		1/2
23	W	_WE_	1,1,1,5	$1/2 \times 1' \times 1'$	$2/5'$
24	O	_W	2,2,23	$1/3 \times 1'$	$2/22'$
25	U	O	2,24	$1/3 \times 1/18'$	$1/244$
26	L	#	25	$1/26$	$1/243$
27	D	#	26		$1/27$
28		D	1,27	$1/2$	$6/25'$
29	W	_	6		$3/7$
30	E	_W	3		$2/4$
31		_WE	2		$2/3$
32	S	_WE_	2,2,2,7,31	$1/3 \times 1' \times 1' \times 1/3'$	$1/23'$
33	H	S	1,32	$1/2 \times 1/25'$	$1/242$
34	O	#	33		$3/34$
35	U	O	3		$1/4$
36	L	OU	1		$1/2$
37	D	OUL	1		$1/2$
38		OULD	1		$1/2$
39	D	OULD_	1,1,1,1,8	$1/2 \times 1' \times 1' \times 1'$	$1/4'$

Prediction by Partial Matching (PPM)

Пример РРМА, $D = 5$

Шаг	Буква	Контекст	$\tau_t(s)$ ⁷	$p_t(esc s)$	$p_t(x s)$
40	O	_D	1		1/2
41		_DO	1		1/2
42	A	_DO_	1		1/2
43	S	_DO_A	1		1/2
44		_DO_AS	1		1/2
45	W	O_AS_	1		1/2
46	E	_AS_W	1		1/2
47		_AS_WE	1		1/2
48	C	S_WE_	1,3	1/2	1/3'
49	A	_WE_C	1		1/2
50	N	_WE_CA	1		1/2

Всего бит = 250 бит (232 бит для PPMD).

⁷ $\tau_t(s)$ – число появлений контекста на предыдущих шагах

Преобразование Барроуза-Уиллера (Burrows-Wheeler)⁸

Основная идея

Пусть x – последовательность на выходе источника.

- ❶ Все циклические сдвиги последовательности x сортируются в алфавитном порядке и формируют таблицу.
- ❷ Результатом такого преобразования являются:
 - ▶ Последний столбец таблицы.
 - ▶ Номер строки, которая соответствует исходной последовательности.

⁸M.Burrows, D. Wheeler, A block-sorting lossless data compression algorithm, Report 124, Digital Systems Research Center, 1994

Прямое преобразование Барроуза-Уиллера

Пример

0	A	NIF WE CANNOT DO AS WE WOULD WE SHOULD DO AS WE	C
1	A	NNOT DO AS WE WOULD WE SHOULD DO AS WE CANIF WE	C
2	A	S WE CANIF WE CANNOT DO AS WE WOULD WE SHOULD DO	—
3	A	S WE WOULD WE SHOULD DO AS WE CANIF WE CANNOT DO	—
4	C	ANIF WE CANNOT DO AS WE WOULD WE SHOULD DO AS WE	—
5	C	ANNOT DO AS WE WOULD WE SHOULD DO AS WE CANIF WE	—
6	D	O AS WE CANIF WE CANNOT DO AS WE WOULD WE SHOULD	—
7	D	O AS WE WOULD WE SHOULD DO AS WE CANIF WE CANNOT	—
8	D	DO AS WE CANIF WE CANNOT DO AS WE WOULD WE SHO	L
9	D	WE SHOULD DO AS WE CANIF WE CANNOT DO AS WE WOU	L
10	E	CANIF WE CANNOT DO AS WE WOULD WE SHOULD DO AS	W
11	E	CANNOT DO AS WE WOULD WE SHOULD DO AS WE CANIF	W
12	E	SHOULD DO AS WE CANIF WE CANNOT DO AS WE WOULD	W
13	E	WOULD WE SHOULD DO AS WE CANIF WE CANNOT DO AS	W
14	F	WE CANNOT DO AS WE WOULD WE SHOULD DO AS WE CAN	I*
15	H	OULD DO AS WE CANIF WE CANNOT DO AS WE WOULD WE	S
16	I	F WE CANNOT DO AS WE WOULD WE SHOULD DO AS WE CA	N
17	L	D DO AS WE CANIF WE CANNOT DO AS WE WOULD WE SHO	U
18	L	D WE SHOULD DO AS WE CANIF WE CANNOT DO AS WE WO	U
19	N	IF WE CANNOT DO AS WE WOULD WE SHOULD DO AS WE C	A
20	N	NOT DO AS WE WOULD WE SHOULD DO AS WE CANIF WE C	A
21	N	OT DO AS WE WOULD WE SHOULD DO AS WE CANIF WE CA	N
22	O	T DO AS WE WOULD WE SHOULD DO AS WE CANIF WE CAN	N
23	O	ULD DO AS WE CANIF WE CANNOT DO AS WE WOULD WE S	H
24	O	ULD WE SHOULD DO AS WE CANIF WE CANNOT DO AS WE	W

Прямое преобразование Барроуза-Уиллера

Пример

25	O	AS WE CANIF WE CANNOT DO AS WE WOULD WE SHOULD	D
26	O	AS WE WOULD WE SHOULD DO AS WE CANIF WE CANNOT	D
27	S	HOULD DO AS WE CANIF WE CANNOT DO AS WE WOULD WE	
28	S	WE CANIF WE CANNOT DO AS WE WOULD WE SHOULD DO	A
29	S	WE WOULD WE SHOULD DO AS WE CANIF WE CANNOT DO	A
30	T	DO AS WE WOULD WE SHOULD DO AS WE CANIF WE CANN	O
31	U	LD DO AS WE CANIF WE CANNOT DO AS WE WOULD WE SH	O
32	U	LD WE SHOULD DO AS WE CANIF WE CANNOT DO AS WE W	O
33	W	E CANIF WE CANNOT DO AS WE WOULD WE SHOULD DO AS	
34	W	E CANNOT DO AS WE WOULD WE SHOULD DO AS WE CANIF	
35	W	E SHOULD DO AS WE CANIF WE CANNOT DO AS WE WOULD	
36	W	E WOULD WE SHOULD DO AS WE CANIF WE CANNOT DO AS	
37	W	OULD WE SHOULD DO AS WE CANIF WE CANNOT DO AS WE	
38		AS WE CANIF WE CANNOT DO AS WE WOULD WE SHOULD D	O
38		AS WE WOULD WE SHOULD DO AS WE CANIF WE CANNOT D	O
40		CANIF WE CANNOT DO AS WE WOULD WE SHOULD DO AS W	E
41		CANNOT DO AS WE WOULD WE SHOULD DO AS WE CANIF W	E
42		DO AS WE CANIF WE CANNOT DO AS WE WOULD WE SHOUL	D
43		DO AS WE WOULD WE SHOULD DO AS WE CANIF WE CANN	T
44		SHOULD DO AS WE CANIF WE CANNOT DO AS WE WOULD W	E
45		WE CANIF WE CANNOT DO AS WE WOULD WE SHOULD DO A	S
46		WE CANNOT DO AS WE WOULD WE SHOULD DO AS WE CANI	F
47		WE SHOULD DO AS WE CANIF WE CANNOT DO AS WE WOUL	D
48		WE WOULD WE SHOULD DO AS WE CANIF WE CANNOT DO A	S
49		WOULD WE SHOULD DO AS WE CANIF WE CANNOT DO AS W	E

Прямое преобразование Барроуза-Уиллера

Пример

Результат преобразования:

- Последний столбец:
CC_____LLWWWW|SNUUAANNHWDD_AA000_____OOEEDTESFDSE
- Номер строки = 16.

Свойства преобразования:

- В результате преобразования символы сгруппировались так, что они стали содержать серии одинаковых символов.
- Это свойство может быть использовано для увеличения степени сжатия.
- Необходимо передавать номер строки.

Обратное преобразование Барроуза-Уиллера

Общая идея

- Находим первый столбец таблицы путём сортировки последнего столбца.
- Восстанавливаем исходную строку символ за символом, с учетом того, что символу, находящемуся в последнем столбце предшествует символ в первом столбце (из-за лексикографического упорядочивания).

Обратное преобразование Барроуза-Уиллера

Пример

t	First	Last	t	First	Last	t	First	Last
0	A	C	17	L	U	34	W	—
1	A	C	18	L	U	35	W	—
2	A	—	19	N	A	36	W	—
3	A	—	20	N	A	37	W	—
4	C	—	21	N	N	38	—	O
5	C	—	22	O	N	39	—	O
6	D	—	23	O	H	40	—	E
7	D	—	24	O	W	41	—	E
8	D	L	25	O	D	42	—	D
9	D	L	26	O	D	43	—	T
10	E	W	27	S	—	44	—	E
11	E	W	28	S	A	45	—	S
12	E	W	29	S	A	46	—	F
13	E	W	30	T	O	47	—	D
14	F	I*	31	U	O	48	—	S
15	H	S	32	U	O	49	—	E
16	I	N	33	W	—			

- $(N,16) \rightarrow I, (I,14) \rightarrow F, (F,46) \rightarrow _$. Пробелов много. Что делать?
- Так как на строке 46 пробел является 9-м по счёту, ищем 9-ю строку с пробелом в последнем символе, получаем $(34,W), \dots$

Преобразование Барроуза-Уиллера

Кодирование последнего столбца "стопкой книг"

- `CC_____LLWWWISNUUAANNHWDD_AA000_____OOEEDTESFDSE`
- Будем писать `esc` каждый раз, когда появляется новый символ и число различных букв между предыдущим и текущим появлением буквы.
- Получим: `esc 0 esc 0 0 0 0 0 esc 0 esc 0 0 0 esc esc esc esc 0 esc 0 2 0 esc 6 esc 0 9 5 0 esc 0 0 2 0 0 0 0 1 0 esc 0 4 esc 2 10 esc 4 2 3`
- Применяем нумерационное кодирование:
 - ▶ Кодирование композиции: 35 бит.
 - ▶ Кодирование последовательности: 94 бита.
- Передача символов, соответствующих `esc`: $8 \times 15 = 120$ бит.
- Передача индекса исходной строки в преобразовании: 6 бит.

Итого: 255 бит.

Сравнение алгоритмов для примера

Алгоритм	Затраты (бит)
Без сжатия	400
Двухпроходное кодирование, код Хаффмана	302
Нумерационное кодирование	283
Арифметическое кодирование, алгоритм A	293
Арифметическое кодирование, алгоритм D	283
LZ77	280
LZW	291
PPMA	250
PPMD	232
Преобразование Барроуза-Уиллера и метод "стопка книг"	255
Преобразование Барроуза-Уиллера и кодирование расстояний	235

Сравнение алгоритмов сжатия

Calgary corpus⁹

Size (bytes)	File name	Description
111,261	BIB	ASCII text in UNIX "refer"format – 725 bibliographic references.
768,771	BOOK1	unformatted ASCII text – Thomas Hardy: Far from the Madding Crowd.
610,856	BOOK2	ASCII text in UNIX "troff"format – Witten: Principles of Computer Speech.
102,400	GEO	32 bit numbers in IBM floating point format – seismic data.
377,109	NEWS	ASCII text – USENET batch file on a variety of topics.
21,504	OBJ1	VAX executable program – compilation of PROGP.
246,814	OBJ2	Macintosh executable program – "Knowledge Support System".
53,161	PAPER1	UNIX "troff"format – Witten, Neal, Cleary: Arithmetic Coding for Data Compression.
82,199	PAPER2	UNIX "troff"format – Witten: Computer (in)security.
513,216	PIC	1728×2376 bitmap image (MSB first): text in French and line diagrams.
39,611	PROGC	Source code in C – UNIX compress v4.0.
71,646	PROGL	Source code in Lisp – system software.
49,379	PROGP	Source code in Pascal – program to evaluate PPM compression.
93,695	TRANS	ASCII and control characters – transcript of a terminal session.

⁹<http://www.data-compression.info/Corpora/CalgaryCorpus/>

Сравнение алгоритмов сжатия

Результаты для Calgary corpus

Архиватор	Настройки	Сжатие ¹⁰ , байт
Uncompressed		3,141,622
compress		1,272,772
Info-ZIP	2.32 -9	1,020,781
gzip 1.3.5	-9	1,017,624
bzip2 1.0.3	-9	828,347
7-zip 9.12b		848,687
ppmd Jr1	-m256 -o16	740,737
ppmonstr J		675,485
ZPAQ v7.15	-method 5	659,709

¹⁰Все файлы сжимаются по-отдельности

Варианты заданий

№	Алгоритм	Параметры	Баллы
1	Код Хаффмана	Двухпроходный	10
2	Код Хаффмана	Двухпроходный, блоками по 2 символа	15
3	Код Хаффмана	Двухпроходный, с учетом предыдущего символа	15
4	Нумерационное кодирование	На основе арифметического кодирования	20
5	Арифметическое кодирование	Оценка вероятности A	15
6	Арифметическое кодирование	Оценка вероятности D	15
7	"Стопка книг"	Кодирование монотонным кодом	10
8	LZ77	$vlc(x)$ – монотонный код	10
9	LZ77	$vlc(x)$ – код Хаффмана	15
10	LZW		10
11	PPMA		20
12	PPMD		20
13	Барроуза-Уиллера	Сжатие методом "стопки книг"	20
14	DeepZIP	Оценка вероятностей при помощи нейронной сети и арифметическое кодирование	30
15	Range Coder	Huawei Global Data Compression Competition	20

Требования к программе и отчету

Требования к программе:

- 1 Написать кодер и декодер, которые работают как отдельные программы.
- 2 В качестве параметра на вход кодера подаётся имя сжимаемого файла. На выходе программа выдаёт файл со сжатыми данными.
- 3 На вход декодера подаётся файл со сжатыми данными. На выходе декодер выдаёт файл, который идентичен исходному (бит в бит).
- 4 Алгоритмы, относящиеся непосредственно к кодированию и декодированию должны быть реализованы без использования сторонних библиотек.

Требования к отчету:

- 1 Отчет в pdf формате, который включает в себя:
 - ▶ ФИО студента, номер группы и задания.
 - ▶ Описание алгоритма кодирования и декодирования.
 - ▶ Для каждого из 14 файлов Calgary corpus вычислить оценки $H(X)$, $H(X|X)$, $H(X|XX)$, а также указать значения средних затрат на символ для реализованного алгоритма (в битах) и суммарный размер сжатого файла в байтах. Данные представить в виде таблицы.
 - ▶ Суммарное значение сжатого размера всех файлов в байтах.
- 2 Исходный код кодера и декодера.
- 3 Исполняемые программы кодера и декодера, готовыми к запуску на Windows 10.