

# Сжатие данных. Сжатие без учёта контекста. Разделимые и неразделимые коды сжатия без учёта контекста

Александра Игоревна Кононова

МИЭТ

2 ноября 2023 г. — актуальную версию можно найти на  
<https://gitlab.com/illinc/otik>

## О терминах

**Символ** — элемент качественной информации  $a \in A$  (множество  $A$  — алфавит).

**Текст** — последовательность  $m \in A^+$  таких элементов.

На практике для всех алгоритмов, где алфавит может быть произвольным, **символ кодирования = байт** (так как в большинстве ЭВМ байт 8-битен — это 00...FF), **исходный текст = любой бинарный файл**, сжатый текст — тоже бинарный файл:

- использование в программе для ЭВМ символов меньших, чем байт — неудобно;
- использование символов фиксированной разрядности больших, чем байт  $\Rightarrow$  слишком большой алфавит  $\Rightarrow$  объёмные структуры данных для восстановления.
- использование в качестве символа кодирования печатного символа ASCII или koi8r/cp1251/dos/iso/maccyrillic не позволяет рассматривать в качестве исходного текста произвольный файл и приводит к труднодиагностируемым ошибкам;
- использование печатного символа UTF-8 (144 697 символов Unicode в 2023 г.) — то же самое + проигрыш в объёме.

В книгах для наглядности используются обозначения  $A, B, \dots$  и т. п. (маленький алфавит + визуальное отличие символа от индекса или частоты), но в программе это всё равно байты!

Сжатие без учёта контекста  
Исторические коды: Шеннона и Шеннона-Фано  
Код Хаффмана

Вырожденное сообщение, адаптивные алгоритмы, арифметический код  
Семинар: подготовка к КР1

О терминах

Сжатие

Последовательность, поток, блок

Оптимальное кодирование источника  $X$

# Сжатие

**Сжатие** (компрессия, упаковка) — кодирование  $|code(X)| < |X|$ , причём  $X$  однозначно и полностью восстанавливается по  $code(X)$ .

Согласно первой теореме Шеннона  $|code(X)| \geq I(X)$  (средние!).

Кодирование с  $|code(X)| \rightarrow I(X)$  и  $|code(x)| \rightarrow I(x)$  — **оптимальное**.

- 1 Сжимается не отдельное сообщение  $x$ , а источник  $X$ .
- 2 Сжатие возможно только при наличии избыточности в изначальном кодировании  $X$  ( $|X| > I(X)$ ).

Если источник  $X$  порождает блоки длины  $N$  бит с равной вероятностью ( $p = \frac{1}{2^N}$ ), он неизбыточен  $\rightarrow$  не существует такого алгоритма сжатия, который сжимает **любой** блок длины  $N$ .

Любой алгоритм сжатия сжимает часто встречающиеся блоки данных за счёт того, что более редкие увеличиваются в размерах.

# Последовательность, поток, блок

Источник  $X$  генерирует **входную последовательность**  $C = c_1 c_2 \dots c_n \dots$ ,  $c_i \in A$  — символы пронумерованы (есть «предыдущий» и «последующий»).

## Типы входной последовательности / алгоритмы сжатия по её типу

- ❶ **блок** — конечная входная последовательность (произвольный доступ);
- ❷ **поток** — с неизвестными границами (последовательный доступ).
- ❶ блочные — статистика всего блока добавляется к сжатому блоку;
- ❷ поточные (адаптивные) — статистика вычисляется только для уже обработанной части потока, «на лету»  $\Rightarrow$  **нестационарная модель  $X$** .

$X$  неизвестен  $\Rightarrow$  строится **модель источника** по догадкам и сообщению  $x$ .

## Свойства алгоритмов сжатия:

- ❶ степень сжатия  $\frac{|X|}{|code(X)|}$ : в среднем по источнику;  $\frac{|X|}{|code(X)|} \leq \frac{|X|}{|I(X)|}$ ; модель  $X$  для оценки  $|code(x)|$  снизу для конкретной реализации может отличаться от исходной модели (ср. Хаффман блочный и адаптивный);
- ❷ **степень увеличения размера в наихудшем случае;**
- ❸ скорость сжатия и разжатия.

# Оптимальное кодирование источника $X$

Пусть  $X$  порождает последовательность из  $2^N$  возможных символов.

- 1 Равновероятный источник ( $I(X) = N$ ) — кодирование отдельных символов кодами фиксированной ширины  $N$  бит.
- 2 Стационарный источник без памяти, порождающий символы с разными постоянными вероятностями ( $I(X) < N$ ) — кодирование отдельных символов кодами переменной ширины: коды Хаффмана, методы семейства арифметического кодирования.
- 3 Стационарный источник с памятью, порождающий символы с вероятностями, зависящими от контекста ( $I(X) < N$ ) — кодирование сочетаний символов: словарные методы семейства LZ77 (словарь=текст) и семейства LZ78 (отдельный словарь в виде дерева/таблицы).

Если изначально каждый символ записан кодом фиксированной ширины из  $N$  бит  $\Rightarrow$  сжатие для 2 и 3.

## Модель источника — стационарный без памяти

Модель источника  $X$  — стационарный источник без памяти, строится по кодируемому сообщению  $C$ :

- 1 кодируемое сообщение —  $C \in A_1^+$  (на практике символы первичного алфавита  $a \in A_1$  — байты);
- 2 символы считаются независимыми:  $p(a) = \text{const}$  (но  $p(a_i) \neq p(a_j)$  в общем случае для  $a_i, a_j \in A_1$ );
- 3 их вероятности оцениваются по частотам в сообщении  $C$ ;

---

Если  $\forall a_i, a_j \in A_1$  верно  $p(a_i) = p(a_j)$  — модель без памяти  $X$  не избыточна, энтропийное сжатие не уменьшит объёма; если вероятности символов (байтов) не равны друг другу (и  $\frac{1}{256}$ ) — энтропийное сжатие уменьшит объём данных приблизительно до  $I(X)$ .

# Алфавитное префиксное кодирование

- 1 Каждому символу  $a \in A_1$  сопоставляется код  $code(a) \in A_2^+$ , для двоичного кодирования —  $A_2 = \{0, 1\}$  и  $code(a)$  — префиксный код из 0 и 1.
- 2 Длина кода  $code(a)$  должна быть как можно ближе к  $I(a)$  (для двоичного кодирования — в битах).

Префиксный код = дерево

Оптимальный код — сбалансированное с учётом весов дерево.

---

Результат алфавитного префиксного кодирования — битовая строка произвольной длины, в общем случае некратной длине байта.

При записи битовой строки в файл последний байт может быть неполным  
 $\Rightarrow$  дополняется незначащими битами, обычно нулями.

## Модель, первичный алфавит, сортировка при равных частотах

Символы первичного алфавита = байты:  $A_1 = \{0, 1, \dots, N\}$ ,  $N = |A| - 1$  (возможно, не все).

### Положение сжатия без учёта контекста:

Модель — стационарный без памяти источник  $\implies$  задаётся постоянными  $(p_0, p_1, \dots, p_N)$ ;  
 в алгоритмах для ЭВМ — **целочисленные частоты**  $(\nu_0, \nu_1, \dots, \nu_N)$ :  $p_i = \frac{\nu_i}{\nu_0 + \nu_1 + \dots + \nu_N}$ ,  
 при оценивании по файлу частота  $\nu_i$  не обязательно равна количеству вхождений  $count(i)$   
 байта  $i$  в исходном тексте, но  $\nu_0 : \nu_1 : \dots : \nu_N \approx count(0) : count(1) : \dots : count(N)$ .

### Допущения ниже:

- Построение дерева громоздко  $\implies$  рассматриваем на примере 3-битного байта ( $2^3 = 8$ -символьного алфавита  $A_1 = \{0, 1, \dots, 7\}$ ).
- Сортировка символов — по убыванию частот. Для кодов Шеннона и АС это принципиально; для кодов Хаффмана и Шеннона—Фано — из единообразия.
- При сортировке символов по убыванию частот при  $\nu_i = \nu_j$  порядок не определён  $\implies$  определим, что при равных частотах  $0 \succ 1 \succ \dots \succ N$  (здесь « $i \succ j$ » = «при  $\nu_i = \nu_j$  сортируем как если  $\nu_i > \nu_j$ »)  
 для Хаффмана определим  $\dots \succ S_2 \succ S_1 \succ 0 \succ 1 \succ \dots \succ N$ .





## Исходный текст — длина, количество информации

Исходный текст  $x = 7412650044443$  (алфавит — трёхбитные байты!):

- ❶ длина исходного текста  $n = 13$  символов, что составляет  $13 \cdot 3 = 39$  бит;
  - сохраняется в заголовке, чтобы при декодировании отсечь незначащие биты в конце файла;
- ❷ частоты  $(\nu_0, \nu_1, \dots, \nu_7) = (2, 1, 1, 1, 5, 1, 1, 1)$ :
  - массив частот  $(2, 1, 1, 1, 5, 1, 1, 1)$  в исходном порядке  $(\nu_0, \nu_1, \dots, \nu_7)$  — помещается в архив и используется для распаковки файла;
  - пары символ<sup>частота</sup>  $4^5, 0^2, 1^1, 2^1, 3^1, 5^1, 6^1, 7^1$ , отсортированные по убыванию частот (а при равных — согласно  $0 \succ 1 \succ \dots \succ N$ ) — используются для построения кодов;
- ❸ общее (не среднее на символ!) количество информации (согласно модели без памяти):  
$$I(x) = -5 \cdot \log_2 \frac{5}{13} - 2 \cdot \log_2 \frac{2}{13} - 6 \cdot 1 \cdot \log_2 \frac{1}{13} \approx 34,5 \text{ бит}$$

35 бит — минимально возможная длина кода  $x$  любым алгоритмом без учёта контекста.

# Код Шеннона

Код Шеннона строится не как дерево [но является деревом]:

- 1 все символы сортируются по частоте (по убыванию):  $a_1, a_2, \dots, a_{|A|}$ ,  
 $\nu(a_1) \geq \nu(a_2) \geq \dots \geq \nu(a_{|A|})$ ;
- 2 код  $a_i$  — первые  $l_i = \lceil -\log_2 p_i \rceil$  двоичных цифр  $\sum_{k=0}^{i-1} p_i$ .

$a_i$	$p_i$	$I(a_i) = -\log_2 p_i$	$l_i$	$\sum_{k=0}^{i-1} p_i$	код
4	$\frac{5}{13} \approx 0,01100\dots$	1,38...	2	0 = 0,00000...	00
0	$\frac{2}{13} \approx 0,00100\dots$	2.70...	3	$\frac{5}{13} \approx 0,01100\dots$	011
1	$\frac{1}{13} \approx 0,00010\dots$	3.70...	4	$\frac{7}{13} \approx 0,10001\dots$	1000
2	$\frac{1}{13} \approx 0,00010\dots$	3.70...	4	$\frac{8}{13} \approx 0,10011\dots$	1001
3	$\frac{1}{13} \approx 0,00010\dots$	3.70...	4	$\frac{9}{13} \approx 0,10110\dots$	1011
5	$\frac{1}{13} \approx 0,00010\dots$	3.70...	4	$\frac{10}{13} \approx 0,11000\dots$	1100
6	$\frac{1}{13} \approx 0,00010\dots$	3.70...	4	$\frac{11}{13} \approx 0,11011\dots$	1101
7	$\frac{1}{13} \approx 0,00010\dots$	3.70...	4	$\frac{12}{13} \approx 0,11101\dots$	1110

$|code(x)| = 5 \cdot 2 + 2 \cdot 3 + 6 \cdot 1 \cdot 4 = 40$  бит =  $\lceil 13 \frac{1}{3} \rceil = 14$  трёхбитных байтов.

Исторически первый; не лучше Шеннона—Фано.

## Построение дерева Шеннона—Фано

Дерево Шеннона—Фано строится **сверху вниз** (от корневого узла к листовым):

- 1 все символы сортируются по частоте;
- 2 упорядоченный ряд символов в некотором месте делится на две части так, чтобы в каждой из них сумма частот символов была примерно одинакова (без пересортировки!);
- 3 новое деление.

Исторически первый близкий к оптимальному префиксный код.

Не лучше кода Хаффмана по степени сжатия и примерно аналогичен по скорости кодирования/декодирования.

# Кодирование $x$ методом Шеннона–Фано

Не определено, какая ветвь получает бит 0, а какая 1. Пусть первая подгруппа  $(s_1) — 0$ , вторая  $(s_2) — 1$ . Неточно определён алгоритм деления  $s$  на  $s_1 + s_2$ . Основные варианты уточнений:

- 1  $\min_{s_1 \leq s_2} |s_2 - s_1|$  — более частые символы получают более короткие коды, быстрее расчёт;
- 2  $\min |s_2 - s_1|$ , если он достигается в одной точке; если в двух:  $\min_{s_1 \leq s_2} |s_2 - s_1|$ ; — короче код сообщения.

Воспользуемся 2:

$$1) \left(4^5, 0^2, 1^1, 2^1, 3^1, 5^1, 6^1, 7^1\right)^{13} \rightarrow \underbrace{\left(4^5, 0^2\right)^7}_{\text{коды начинаются с 0}} + \underbrace{\left(1^1, 2^1, 3^1, 5^1, 6^1, 7^1\right)^6}_{\text{коды начинаются с 1}}$$

$$2) \underbrace{\left(4^5, 0^2\right)^7}_{\text{коды начинаются с 0}} \rightarrow \underbrace{4^5}_{00} + \underbrace{0^2}_{01} \text{ и т. д.:}$$

$4^5$	$0^2$	$1^1$	$2^1$	$3^1$	$5^1$	$6^1$	$7^1$
0	1						
0	1	0			1		
		0	1		0	1	
			0	1		0	1
00	01	100	1010	1011	110	1110	1111

$$\text{code}(x) = 111100100\dots$$

$$|\text{code}(x)| = 5 \cdot 2 + 2 \cdot 2 + 2 \cdot 1 \cdot 3 + 4 \cdot 1 \cdot 4 = 36 \text{ бит} = 12 \text{ трёхбитных байтов}$$

# Построение дерева Хаффмана

Дерево Хаффмана строится **снизу вверх** (от листовых узлов к корневому узлу):

- 1 все символы сортируются по частоте (по убыванию);
- 2 два последних (самых редких) элемента отсортированного списка узлов заменяются на новый элемент с частотой, равной сумме исходных;
- 3 новая сортировка.

На каждом шаге число узлов сокращается на один; узел, полученные на последнем шаге — корень дерева.

Код Хаффмана имеет минимальную длину среди префиксных.

Не увеличивает размера исходных данных в худшем случае.

## Кодирование $x$ методом Хаффмана, ... $\succ S_2 \succ S_1 \succ 0 \succ 1 \succ \dots \succ 7$

Не определено, какая ветвь дерева получает бит 0, а какая 1. Пусть 0, 1 — слева направо.

1)  $4^5, 0^2, 1^1, 2^1, 3^1, 5^1, \underbrace{6^1, 7^1}_{0 \ S_1^2 \ 1}$  — последний бит кода  $6^1 = 0$ , последний бит кода  $7^1 = 1$

$$2) 4^5, S_1^2, 0^2, 1^1, 2^1, \underbrace{3^1, 5^1}_{0 \quad S_2^2 \quad 1}$$
$$3) 4^5, S_2^2, S_1^2, 0^2, \underbrace{1^1, 2^1}_{0 \quad S_3^2 \quad 1}$$
$$4) \ 4^5, S_3^2, S_2^2, \underbrace{S_1^2, 0^2}_{0 \ S_4^4 \ 1}$$

5)  $4^5, S_4^4, \underbrace{S_3^2, S_2^2}_{\substack{0 \quad S_5^4 \quad 1}}$

$$6) \ 4^5, \underbrace{S_5^4, S_4^4}_{0 \ S_6^8 \ 1}$$
$$7) \quad \underbrace{S_6^8, 4^5}_{0 \quad S_7^{13} \quad 1}$$

$4^5$	$0^2$	$1^1$	$2^1$	$3^1$	$5^1$	$6^1$	$7^1$
1	011	0000	0001	0010	0011	0100	0101

$$code(x) = 01011000000010100001101101111110010$$
$$|code(x)| = 5 \cdot 1 + 2 \cdot 3 + 6 \cdot 1 \cdot 4 = 35 \text{ бит} = \lceil 11 \frac{2}{3} \rceil = 12 \text{ трёхбитных байтов.}$$

## Код Хаффмана и архив с кодами Хаффмана

Исходный файл  $m = 7412650044443$  из  $n = 13_{10} = 15_8$  трёхбитных байтов, код Хаффмана: 01011000000010100001101101111110010 длиной 35 бит будет дополнен до  $\geq 12$  байтов:

010 110 000 000 101 000 011 011 011 111 100 100 (биты) = 260050333744 (байты)

Для декодирования нужны ещё  $n$  (запишем 12 битами = 4 байтами) и массив частот  $\vec{\nu}$ :

0015, 21115111, 260050333744

Ненормированное количество  $count(c)$  вхождений достигает  $n \implies$  нормировка:

$$\begin{cases} \nu_0 : \nu_1 : \dots : \nu_N \approx count(0) : count(1) : \dots : count(N), \\ \max(\nu_i) = \text{максимальное значение байта.} \end{cases}$$

Для  $m = 7754\ 4444\ 4444\ 4444\ 4444\ 3333\ 3333\ 3333\ 3333\ 1112$  длины  $n = 40_{10} = 50_8$   
 $\overrightarrow{count} = (0, 3, 1, 16, 17, 1, 0, 2)$ , но  $\vec{\nu} = (0, 2, 1, 7, 7, 1, 0, 1)$ .

В файл записываются 0050, 02177101 и код, рассчитанный по  $\vec{\nu} = (0, 2, 1, 7, 7, 1, 0, 1)$ .

# Нулевые частоты и нормировка частот

- ❶ По умолчанию байты с нулевыми  $\nu_i$  отбрасываются и не получают кода. Тогда при приведении частот  $count(i) \in [0, \max(count)] \rightarrow \nu_i \in [0, Max]$  необходимо, чтобы при  $count(i) > 0$  было  $\nu_i > 0$ :

- соотношения всех частот незначительно искажаются:

$$\begin{cases} \nu_i = 0, & count(i) = 0, \\ \nu_i = \text{round} \left( \frac{count(i)-1}{\max(count)-1} \cdot (Max - 1) \right) + 1, & count(i) > 0; \end{cases} \quad (A)$$

- для  $count(i) > \frac{\max(count)}{Max}$  передаются максимально точно; для малых полностью искажаются:

$$\begin{cases} \nu_i = 0, & count(i) = 0, \\ \nu_i = 1, & 0 < count(i) \leq \frac{\max(count)}{Max}, \\ \nu_i = \text{round} \left( \frac{count(i)}{\max(count)} \cdot Max \right), & count(i) > \frac{\max(count)}{Max}; \end{cases} \quad (B)$$

для октетов ( $Max = 255$ ) и  $\frac{\max(count)}{\min(count)} \leq Max$  обе формулы дают приемлемый результат.

- ❷ Если хочется  $\nu_i = \text{round} \left( \frac{count(i)}{\max(count)} \cdot Max \right)$  для всех (возможно  $count(i) > 0 \rightarrow \nu_i = 0$ ), то:

- необходимо модифицировать алгоритм, чтобы байты с  $\nu_i = 0$  получили коды (возможно для Хаффмана и Шеннона—Фано, невозможно для арифметического и Шеннона);
- тогда коды получат и байты с  $count(i) = 0$ , а коды  $count(i) > 0$  удлинятся.

⏪ ⏩ ⏴ ⏵ ⏶ ⏷ ⏸ ⏹ ⏺ ⏻ ⏼ ⏽ ⏾ ⏿ 🔍 ↺



## Вырожденный случай

Для  $m = 44444444444444$  из  $n = 13_{10} = 15_8$  байтов  $I(m) = 0$ :

- длина кода Шеннона символа 4 равна нулю, так как  $I(m) = 0$ ;
- длина кода Хаффмана и Шеннона—Фано символа 4 равна нулю, так как дерево состоит из одного узла (корня 4) и нуля ветвей.

Длина кода (Хаффмана, Шеннона—Фано или Шеннона) всего сообщения из  $n$  одинаковых символов 4 также **нулевая**.

Файл архива должен содержать  $n$  и массив частот  $\vec{\nu}$ :

0015, 00007000

этого достаточно для восстановления такого сообщения.

## Адаптивный (поточный) кодек

Вначале считаем символы равновероятными (код=значение байта как есть), после каждого записанного/прочитанного символа перестраиваем дерево:

- собственно код  $code(x)$  длиннее блочного кодека:  
оценка  $|code(x)|$  снизу — не стационарный источник без памяти, а нестационарный (набор вероятностей на каждом шаге меняется);
- не требует для декодирования частот  $(\nu_0, \nu_1, \dots, \nu_N) \implies$   
общий размер файла может быть меньше, чем у блочного.

# Арифметический (интервальный) код

Неалфавитное неразделимое кодирование

$$C = c_0 c_1 c_2 \dots c_n \rightarrow z \in [0, 1); \quad (0, 1) \simeq \mathbb{R}$$

$$I(z) \approx I(C), \quad \text{и чаще всего } I(z) \gg 64 \text{ бит} > I(\text{double})$$

# Семинар: подготовка к КР1

Для сообщения  $m = 3242\ 5675\ 2067\ 5462$

- 1 оцените суммарное количество информации согласно модели «источник без памяти» (вероятности символов оцениваются по сообщению);
- 2 закодируйте методами: Хаффмана, Шеннона—Фано, Шеннона (укажите порядок сортировки по умолчанию — при равных частотах);
- 3 сравните длины кодов друг с другом и с количеством информации.

В байте три бита; символы первичного алфавита — байты.

# Спасибо за внимание!

МИЭТ

[www.miet.ru](http://www.miet.ru)

Александра Игоревна Кононова

[illinc@mail.ru](mailto:illinc@mail.ru)

[gitlab.com/illinc/raspisanie](https://gitlab.com/illinc/raspisanie)