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

Александра Игоревна Кононова

МИЭТ

20 сентября 2024 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 бóльших, чем байт $\Longrightarrow$ слишком большой алфавит $\Longrightarrow$ 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 $A, B, \ldots$ и т. п. (маленький алфавит + визуальное отличие символа от индекса или частоты), но в программе это всё равно байты!

Сжатие

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

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

Кодирование с $|code(X)| \to I(X)$ и $|code(x)| \to I(x)$ — **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 $\left(p = \frac{1}{2^N}\right)$, он неизбыточен $\to$ 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

1. **блок** — конечная входная последовательность (произвольный доступ);
2. **поток** — с неизвестными границами (последовательный доступ).

1. блочные — статистика всего блока добавляется к сжатому блоку;
2. поточные (адаптивные) — статистика вычисляется только для уже обработанной части потока, «на лету» $\Longrightarrow$ нестационарная модель X.

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

Свойства алгоритмов сжатия:

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

Методы кодирования, коды и алгоритмы

Далее рассматриваются **методы сжатия без контекста:** Шеннона, Шеннона—Фано, Хаффмана, арифметический (AC); **методы сжатия с учётом контекста:** RLE, LZ77, LZ78 и **методы защиты от помех:** Хэмминга и Рида—Соломона.

1. из них только метод Шеннона [не используется] однозначно определяет блочный **код** и **алгоритм** его построения — но уже для поточных реализаций есть несколько вариантов;
2. для Хаффмана есть несколько вариантов блочных кодов (каждый — несколько поточных), причём для каждого кода Хаффмана есть как минимум два разных алгоритма его построения;
3. для Шеннона—Фано [не используется] — несколько блочных кодов (+ поточные, + разные алгоритмы), причём у разных блочных кодов Шеннона—Фано различается длина;
4. для AC, RLE, LZ78, Хэмминга, Рида—Соломона — десятки кодов разной длины и алгоритмов;
5. для LZ77 — сотни кодов, и для каждого кода — сотни алгоритмов.

Поэтому хотя в литературе (а также на лекциях и семинарах) употребляются для краткости названия «код Хаффмана», «алгоритм Хаффмана» и т. п. — всегда необходимо помнить, что это **метод,** порождающий **семейство кодов** и **семейство алгоритмов.**

«Энтропийное сжатие» vs «сжатие без учёта контекста»

Энтропийные методы сжатия — Хаффмана (предки: Шеннона, Шеннона—Фано) и АС: $|code(x)| \to I_X(x)$ для всех $x = c_1c_2\dots c_n$ из X; и даже $|code(c_i)| \to I_X(c_i|c_1c_2\dots c_{i-1})$

1. Канонический вариант — блочный без учёта контекста:
   - одномерный массив вероятностей $p(a_j)$ оценивается по x и сохраняется вместе с $code(x)$;
   - одно дерево Хаффмана для файла x.
2. Поточные (адаптивные) варианты без учёта контекста:
   - при кодировании первого символа c_1 считаем все a_j равновероятными (байт c_1 — байтом);
   - перед кодированием каждого следующего c_i (варианты: через ... символов; как именно) **пересчитываем** вероятности $p(a_j)$ по $c_1c_2\dots c_{i-1}$ и новое дерево Хаффмана.

   Чуть длиннее и медленнее блочного, но не требуют сохранения $p(a_j) \implies$ используются.

Теоретически **энтропийное сжатие с учётом контекста** возможно, на практике не используется:

3. Блочный вариант с учётом контекста (предыстории в N символов):
   - $(N+1)$-**мерный массив** условных вероятностей $p(a_j|a_ka_l\dots a_m)$; первые $c_1c_2\dots c_N$ — как есть;
   - далее для каждого c_i — из $p(a_j|c_{i-N}\dots c_{i-1})$ строится новое дерево Хаффмана;
4. Поточные варианты с учётом контекста — пересчитываем условные $p(a_j|a_ka_l\dots a_m)$ в процессе.

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 $\Longrightarrow$ дополняется незначащими битами, обычно нулями.

Модель, первичный алфавит, сортировка при равных частотах

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

Положение сжатия без учёта контекста:

Модель — стационарный без памяти источник $\Longrightarrow$ задаётся постоянными $(p_0, p_1, ..., p_N)$;
в алгоритмах для ЭВМ — **целочисленные частоты (ν_0,ν_1,...,ν_N):** $p_i = \dfrac{\nu_i}{\nu_0 + \nu_1 + ... + \nu_N}$,
при оценивании по файлу частота ν_i не обязательно равна количеству вхождений $count(i)$ байта i в исходном тексте, но $\nu_0 : \nu_1 : ... : \nu_N \approx count(0) : count(1) : ... : count(N)$.

Допущения ниже:

- Построение дерева громоздко $\Longrightarrow$ рассматриваем на примере 3-битного байта ($2^3 = 8$-символьный алфавит $A_1 = \{0, 1, ..., 7\}$, длина файла кратна 3 битам).
- 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 $\Longrightarrow$ определим, что **при равных частотах** $0 \succ 1 \succ ... \succ N$
(здесь «$i \succ j$» = «при $\nu_i = \nu_j$ сортируем как если $\nu_i > \nu_j$»)
для Хаффмана определим **при равных частотах** $... \succ S_2 \succ S_1 \succ 0 \succ 1 \succ ... \succ N$.

Исходный текст — длина, количество информации

Исходный текст $x = 7412650044443$ (алфавит — k-битные байты, $k = 3!$):

1. длина исходного текста $n = 13$ символов, что составляет $13 \cdot k = 39$ бит;
   - сохраняется в заголовке, чтобы при декодировании отсечь незначащие биты в конце файла;
2. частоты $(\nu_0, \nu_1, ..., \nu_7) = (2, 1, 1, 1, 5, 1, 1, 1)$:
   - массив частот $(2, 1, 1, 1, 5, 1, 1, 1)$ в исходном порядке $(\nu_0, \nu_1, ..., \nu_7)$ — помещается в архив и используется для распаковки файла;
   - пары символ$^{\text{частота}}$ $4^5, 0^2, 1^1, 2^1, 3^1, 5^1, 6^1, 7^1$, отсортированные по убыванию частот (а при равных — согласно $0 \succ 1 \succ ... \succ N$) — используются для построения кодов;
3. общее (не среднее на символ!) количество информации (согласно модели без памяти):
   $I(x) = -5 \cdot \log_2 \frac{5}{13} - 2 \cdot \log_2 \frac{2}{13} - 6 \cdot 1 \cdot \log_2 \frac{1}{13} \approx 34{,}5$ 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): $a_1, a_2, ...a_{|A|}$, $\nu(a_1) \geqslant \nu(a_2) \geqslant ... \geqslant \nu(a_{|A|})$;
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	$\frac{5}{13} \approx 0{,}01100...$	1,38...	2	$0 = 0{,}00000...$	00
0	$\frac{2}{13} \approx 0{,}00100...$	2,70...	3	$\frac{5}{13} \approx 0{,}01100...$	011
1	$\frac{1}{13} \approx 0{,}00010...$	3,70...	4	$\frac{7}{13} \approx 0{,}10001...$	1000
2	$\frac{1}{13} \approx 0{,}00010...$	3,70...	4	$\frac{8}{13} \approx 0{,}10011...$	1001
3	$\frac{1}{13} \approx 0{,}00010...$	3,70...	4	$\frac{9}{13} \approx 0{,}10110...$	1011
5	$\frac{1}{13} \approx 0{,}00010...$	3,70...	4	$\frac{10}{13} \approx 0{,}11000...$	1100
6	$\frac{1}{13} \approx 0{,}00010...$	3,70...	4	$\frac{11}{13} \approx 0{,}11011...$	1101
7	$\frac{1}{13} \approx 0{,}00010...$	3,70...	4	$\frac{12}{13} \approx 0{,}11101...$	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. $\min\limits_{s_1 \leqslant s_2} |s_2 - s_1|$ — более частые символы получают более короткие коды, быстрее расчёт;
2. $\min |s_2 - s_1|$, если он достигается в одной точке; если в двух: $\min\limits_{s_1 \leqslant s_2} |s_2 - s_1|$; — короче код сообщения.

Воспользуемся 2:

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

2) $\underbrace{\left(4^5, 0^2\right)^7}_{\text{коды начинаются с 0}} \to \underbrace{4^5}_{00} + \underbrace{0^2}_{01}$ 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

$code(x) = 111100100...$

$|code(x)| = 5 \cdot 2 + 2 \cdot 2 + 2 \cdot 1 \cdot 3 + 4 \cdot 1 \cdot 4 = 36$ бит $= 12$ 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, $\ldots \succ S_2 \succ S_1 \succ 0 \succ 1 \succ \ldots \succ 7$

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

1) $4^5, 0^2, 1^1, 2^1, 3^1, 5^1, \underbrace{6^1, 7^1}_{0 \quad S_1^2 \quad 1}$ — 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) $4^5, S_3^2, S_2^2, \underbrace{S_1^2, 0^2}_{0 \quad S_4^4 \quad 1}$

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

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

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

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

$code(x) = 01011000000010100001101101111110010$

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

Код Хаффмана и архив с кодами Хаффмана

Код Хаффмана $0101100000001010000110110111111 0010$ длиной 35 бит будет дополнен до $\geqslant 12$ байтов: $010\,110\,000\,000\,101\,000\,011\,011\,011\,111\,100\,100$ (биты) $= 260050333744$ (байты)

Смещение	Размер	Описание	
0	4	Сигнатура+версия формата	всегда 0711
4	1	№ алгоритма сжатия с контекстом	0 — нет сжатия
5	1	№ алгоритма сжатия без контекста	0 — нет сжатия, 1 — Шеннона, 2 — вар-т Шеннона—Фано со с. 12, 3 — вар-т Хаффмана со с. 14
6	1	№ алгоритма шифрования	0 — нет шифрования
7	1	№ алгоритма защиты от помех	0 — нет защиты от помех
8	4	Исходная длина файла n	беззнаковое 12-битное целое
12	4	Резерв	
16	8	Массив частот $\vec{\nu} = (\nu(0), \nu(1), ..., \nu(7))$	беззнаковые 3-битные целые
24	до конца	Сжатые данные	выравнивание на 1 байт

Только Хаффман с. 14: алг. 0300; исх. длина $n = 13_{10} = 15_8$ трёхбитных байтов, порядок Intel: $n \sim 5100$; частоты $\vec{\nu} \sim 21115111$, код $260050333744 \to$ архив имеет вид $07110300510000002111511126005033 3744$

Для удобочитаемости разделяем по 4-байтовым блокам: 0711 0300 5100 0000 2111 5111 2600 5033 3744

Нормировка частот

Ненормированное $count(c)$ может превышать допустимое $\max(\nu(c)) \implies$ нормировка:

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

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 архив записываются:

- нормированные частоты 02177101;
- код, рассчитанный по нормированным $\vec{\nu} = (0, 2, 1, 7, 7, 1, 0, 1)$, а не по исходным $\overrightarrow{count}$.

Нулевые частоты и нормировка частот

1. По умолчанию байты с нулевыми ν_i отбрасываются и не получают кода. Тогда при приведении частот $count(i) \in [0, \max(count)] \ \to \ \nu_i \in [0, Max]$ необходимо, чтобы при $count(i) > 0$ было $\nu_i > 0$:
   - соотношения всех частот незначительно искажаются:
   $$\begin{cases} \nu_i = 0, & count(i) = 0, \\ \nu_i = \operatorname{round}\left(\frac{count(i)-1}{\max(count)-1} \cdot (Max-1)\right) + 1, & count(i) > 0; \end{cases} \tag{A}$$
   - для $count(i) > \frac{\max(count)}{Max}$ передаются максимально точно; для малых полностью искажаются:
   $$\begin{cases} \nu_i = 0, & count(i) = 0, \\ \nu_i = 1, & 0 < count(i) \leqslant \frac{\max(count)}{Max}, \\ \nu_i = \operatorname{round}\left(\frac{count(i)}{\max(count)} \cdot Max\right), & count(i) > \frac{\max(count)}{Max}; \end{cases} \tag{B}$$

   для октетов ($Max = 255$) и $\dfrac{\max(count)}{\min\limits_{\neq 0}(count)} \leqslant Max$ обе формулы дают приемлемый результат.

2. Если хочется $\nu_i = \operatorname{round}\left(\frac{count(i)}{\max(count)} \cdot Max\right)$ для всех (возможно $count(i) > 0 \ \to \ \nu_i = 0$), 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 $m = 4444444444444$ 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 $\vec{\nu}$:

$15, \quad 00007000$

этого достаточно для восстановления такого сообщения.

Арифметический (интервальный) код

Неалфавитное неразделимое кодирование

$C = c_0 c_1 c_2 ... c_n \to z \in [0, 1);$ $\qquad (0, 1) \underset{\sim}{\subset} \mathbb{R}$

$I(z) \approx I(C),$ $\qquad$ и чаще всего $I(z) >> 64$ бит $> I(\texttt{double})$

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

Александра Игоревна Кононова
illinc@mail.ru
gitlab.com/illinc/raspisanie