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

Александра Игоревна Кононова

МИЭТ

25 сентября 2021 г. — актуальную версию можно найти на `https://gitlab.com/illinc/otik`

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** $C = c_1 c_2 \ldots c_n \ldots$, $c_i \in A$ — символы пронумерованы (есть «предыдущий» и «последующий»). X неизвестен $\Rightarrow$ строится **модель источника** по входной последовательности.

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

Алгоритмы сжатия по типу входной последовательности:

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

Свойства алгоритмов сжатия:

1. степень сжатия $\frac{|X|}{|code(X)|}$ (в среднем по источнику; $\frac{|X|}{|code(X)|} \leqslant \frac{|X|}{|I(X)|}$) и **степень увеличения размера в наихудшем случае;**
2. 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:

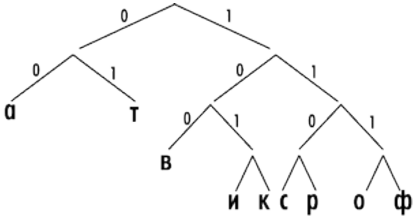
1. кодируемое сообщение — $C \in A_1^+$ (на практике символы первичного алфавита $a \in A_1$ — байты);
2. символы считаются независимыми: $p(a) = const$ (но $p(a_i) \neq p(a_j)$ в общем случае для $a_i, a_j \in A_1$);
3. их вероятности оцениваются по частотам в сообщении C;

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.

«Авиакатастрофа» — длина, количество информации

Кодируем строку $x =$«авиакатастрофа»:

- первичный алфавит — символ=тетрада (4-битный байт доски), длина — $n = 14$ тетрад (56 бит); пусть есть общепринятая «естественная» кодировка Alternative vexillum codicis inf. interpretatio (AVCII): `␣,.!?абвгикорстф` `0123456789ABCDEF`
- используется 9 различных символов, частоты а(5), т(2), в(1), и(1), к(1), с(1), р(1), о(1), ф(1);
- общее (не среднее на символ!) количество информации в тексте (согласно модели без памяти): $I(x) = -5 \cdot \log_2 \frac{5}{14} - 2 \cdot \log_2 \frac{2}{14} - 7 \cdot \log_2 \frac{1}{14} \approx 39{,}7$ бит

Построение дерева Шеннона–Фано

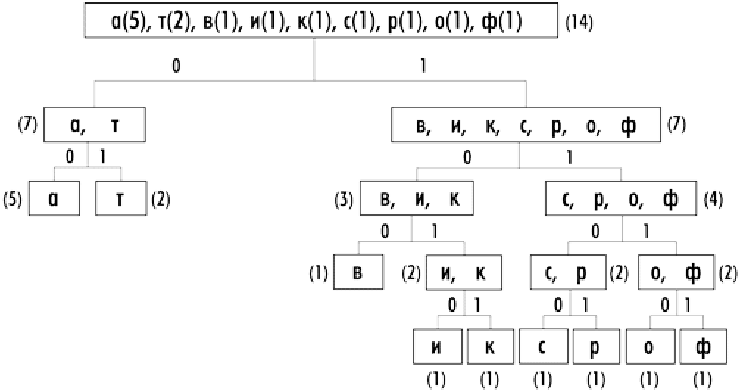
Дерево Шеннона—Фано строится **сверху вниз** (от корневого узла к листовым):

1. все символы сортируются по частоте;
2. упорядоченный ряд символов делится на две части так, чтобы в каждой из них сумма частот символов была примерно одинакова;
3. новое деление.

Исторически первый близкий к оптимальному префиксный код.

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

«Авиакатастрофа» — кодирование Шеннона–Фано



а 00
т 01
в 100
и 1010
к 1011
с 1100
р 1101
о 1110
ф 1111

$$code(x) = 00100101000101100010011000111011110111100$$

$$|code(x)| = 5 \cdot 2 + 2 \cdot 2 + 3 + 6 \cdot 4 = 41 \text{ бит}$$

Построение дерева Хаффмана

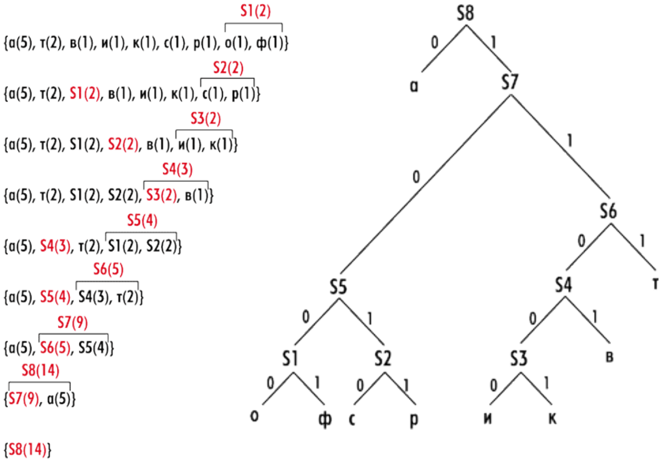
Дерево Хаффмана строится **снизу вверх** (от листовых узлов к корневому узлу):

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.

Код Хаффмана имеет минимальную длину среди префиксных.

Не увеличивает размера исходных данных в худшем случае.

«Авиакатастрофа» — кодирование Хаффмана



а 0
т 111
в 1101
и 11000
к 11001
с 1010
р 1011
о 1000
ф 1001

$$code(x) = 01101110000110010111010101111011100010010$$
$$|code(x)| = 5 \cdot 1 + 2 \cdot 3 + 5 \cdot 4 + 2 \cdot 5 = 41 \text{ бит}$$

Фактическая длина данных при префиксном кодировании

- $code(x) = 01101110000110010111010101111011100010010$, 41 бит
  но записать в файл можно только целое число байтов:
  $01101110000110010111010101111011100010010000$, 44 бита $= 11$ тетрад
- для декодирования нужно дерево/таблица кодов — если алгоритм построения детерминирован, то **массив частот ν_i:** для байта-тетрады — из 16 беззнаковых целых, для байта-октета — 256;

  ␣,.!?абвгикорстф
  в естественном порядке 0000050101111121.
  0123456789ABCDEF

  Размер ν_i — как размер поля n, либо перенормировка ν_i.

  Адаптивный (поточный) кодек: вначале считаем символы равновероятными (код=AVCII), после каждого записанного/прочитанного символа перестраиваем дерево.
- Для определения того, каким конкретно декодером пользоваться — соответствующее поле заголовка (№ алгоритма сжатия без контекста).

Нулевые частоты и нормировка частот

1. Нулевые значения частот могут быть отброшены при первой сортировке (как в примерах на слайдах), и символы с нулевыми частотами не получат кода.
   Тогда при перенормировке частот $[0, \nu_{\max}] \to [0, Max]$ необходимо, чтобы ненулевые малые частоты не перешли в нулевые:
   $$\nu_i \to \begin{cases} 0, & \nu_i = 0, \\ \mathrm{round}\left(\frac{\nu_i - 1}{\nu_{\max} - 1} \cdot (Max - 1)\right) + 1, & \nu_i \neq 0. \end{cases}$$
2. Нулевые значения частот могут обрабатываться по общему алгоритму: символы с нулевыми частотами получат коды, а код символа с наименьшей ненулевой частотой (последнего в сортировке; здесь «ф») удлинится на бит.
   Тогда при перенормировке ненулевая частота может стать нулевой: $\nu_i \to \mathrm{round}\left(\frac{\nu_i}{\nu_{\max}} \cdot Max\right)$.

Арифметический (интервальный) код

Неалфавитное неразделимое кодирование

$C = c_0 c_1 c_2 \ldots c_n \to z \in [0,1);$ $(0,1) \underline{\simeq} \mathbb{R}$

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

Спасибо за внимание!

МИЭТ
www.miet.ru

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