Жадные алгоритмы: кодирование Хаффмана

Александр Куликов

Онлайн-курс «Алгоритмы: теория и практика. Методы»
http://stepic.org/217

Сжатие данных

Вход: строка s.

Выход: бинарный код символов строки s, обеспечивающий кратчайшее представление s.

Пример

$s = \texttt{abacabad}$

коды символов: a: 00, b: 01, c: 10, d: 11

закодированная строка: 0001001000010011 (16 битов)

Коды переменной длины

- Естественная идея: присвоить более короткие коды более частым символам.

Коды переменной длины

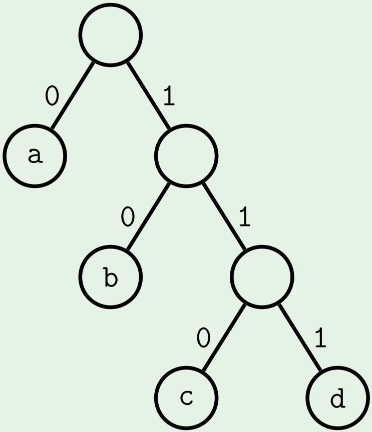
- Естественная идея: присвоить более короткие коды более частым символам.
- $s =$ abacabad
  коды символов: a: 0, b: 10, c: 110, d: 111
  закодированная строка: 01001100100111 (14 битов)

Коды переменной длины

- Естественная идея: присвоить более короткие коды более частым символам.
- s = abacabad
  коды символов: a: 0, b: 10, c: 110, d: 111
  закодированная строка: 01001100100111 (14 битов)
- Код называется беспрефиксным, если никакой код символа не является префиксом другого кода символа.

Декодирование на примере

Пример

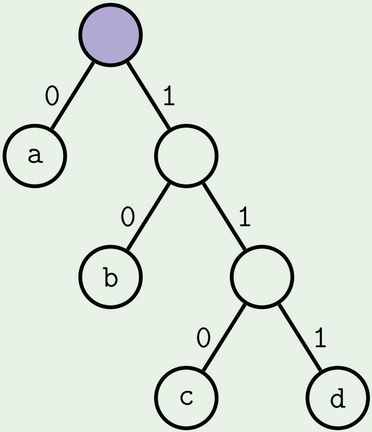


01001100100111

$s =$

Декодирование на примере

Пример

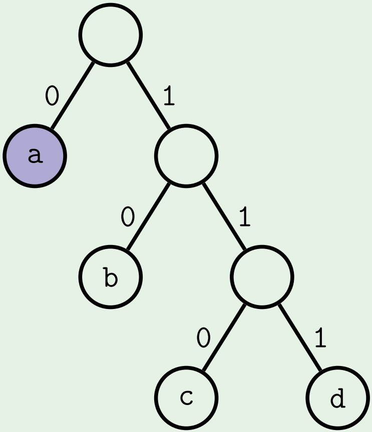


01001100100111

$s =$

Декодирование на примере

Пример

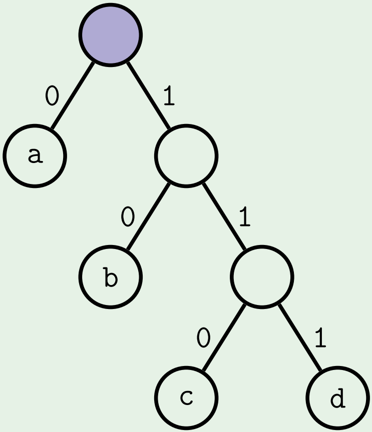


01001100100111

$s = \mathtt{a}$

Декодирование на примере

Пример

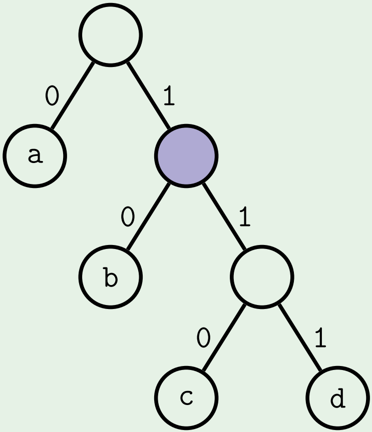


01001100100111

$s = \mathtt{a}$

Декодирование на примере

Пример

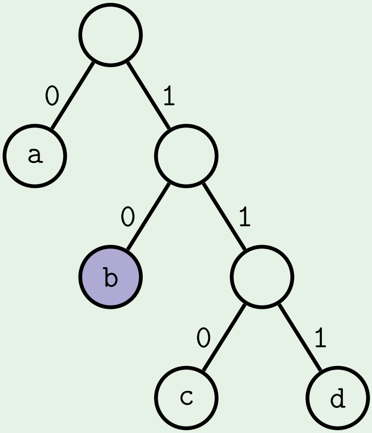


01001100100111

$s = \mathtt{a}$

Декодирование на примере

Пример

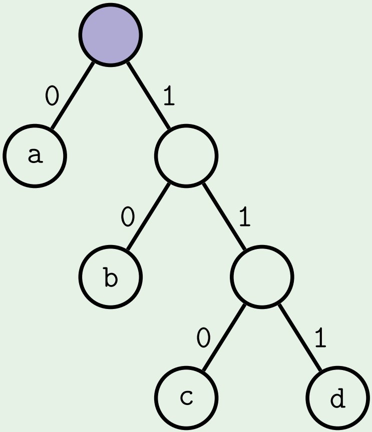


0100110010011 1

$s = \mathrm{ab}$

Декодирование на примере

Пример



010011100100111

$s = \mathrm{ab}$

Код Хаффмана

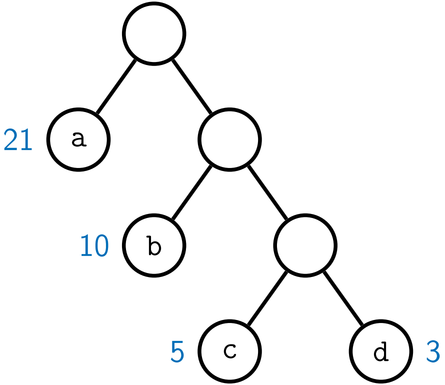
Код Хаффмана

Вход: частоты символов $f_1, \ldots, f_n \in \mathbb{N}$.

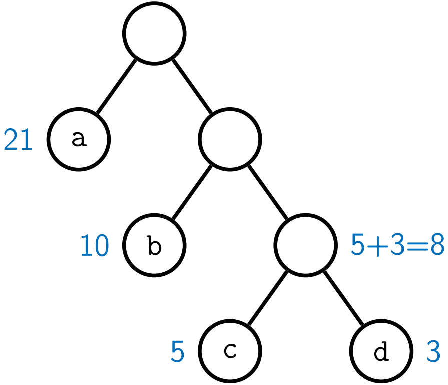
Выход: строго двоичное дерево (у каждой вершины либо ноль, либо два сына), листья которого помечены частотами $f_1, \ldots, f_n$, минимизирующее

$$\sum_{i=1}^{n} f_i \cdot (\text{глубина листа } f_i)\,.$$

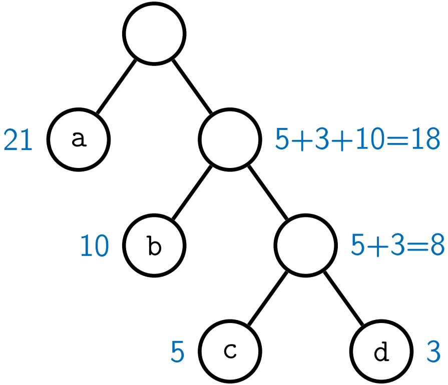
Частоты для внутренних вершин



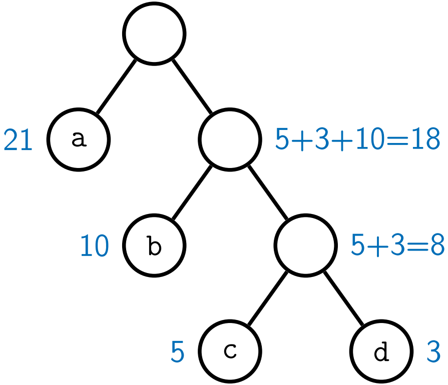
Частоты для внутренних вершин



Частоты для внутренних вершин



Частоты для внутренних вершин



Частотой (некорневой) вершины назовём количество раз, которое вершина будет посещена в процессе кодировки/декодировки.

Надёжный шаг

- Таким образом, мы ищем строго двоичное дерево с минимальной суммой пометок в вершинах, в котором листья помечены входными частотами, а внутренние вершины — суммами пометок их детей.

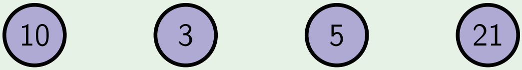
Надёжный шаг

- Таким образом, мы ищем строго двоичное дерево с минимальной суммой пометок в вершинах, в котором листья помечены входными частотами, а внутренние вершины — суммами пометок их детей.
- Двумя наименьшими частотами помечены листья на нижнем уровне.

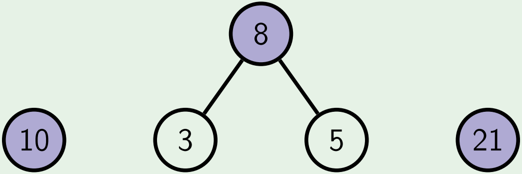
Надёжный шаг

- Таким образом, мы ищем строго двоичное дерево с минимальной суммой пометок в вершинах, в котором листья помечены входными частотами, а внутренние вершины — суммами пометок их детей.
- Двумя наименьшими частотами помечены листья на нижнем уровне.
- Надёжный жадный шаг: выбрать две минимальные частоты f_i и f_j, сделать их детьми новой вершины с пометкой $f_i + f_j$; выкинуть частоты f_i и f_j, добавить $f_i + f_j$.

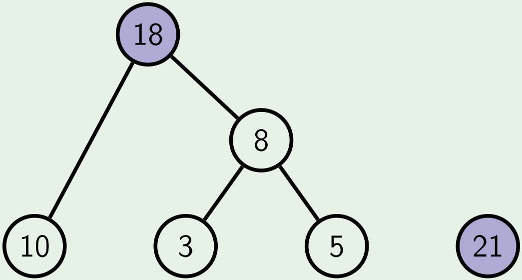
Пример



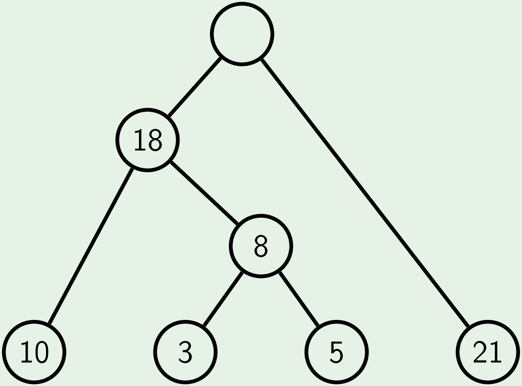
Пример



Пример



Пример



Очередь с приоритетами

INSERT(p) добавляет новый элемент с приоритетом p

EXTRACTMIN() извлекает из очереди элемент с минимальным приоритетом

Алгоритм

процедура $\textsc{Huffman}(F[1 \dots n])$

$H \leftarrow$ очередь с приоритетами
для i от 1 до n:
  $\textsc{Insert}(H, (i, F[i]))$
для k от $n+1$ до $2n-1$:
  $(i, F[i]) \leftarrow \textsc{ExtractMin}(H)$
  $(j, F[j]) \leftarrow \textsc{ExtractMin}(H)$
  создать вершину k с детьми i, j
  $F[k] = F[i] + F[j]$
  $\textsc{Insert}(H, (k, F[k]))$

Алгоритм

процедура $\textsc{Huffman}(F[1 \dots n])$

$H \leftarrow$ очередь с приоритетами
для i от 1 до n:
  $\textsc{Insert}(H, (i, F[i]))$
для k от $n+1$ до $2n-1$:
  $(i, F[i]) \leftarrow \textsc{ExtractMin}(H)$
  $(j, F[j]) \leftarrow \textsc{ExtractMin}(H)$
  создать вершину k с детьми i, j
  $F[k] = F[i] + F[j]$
  $\textsc{Insert}(H, (k, F[k]))$

Время работы: $O(n^2)$, если очередь с приоритетами реализована на базе массива, $O(n \log n)$ — если на базе кучи (разберём в следующей лекции).