

## 0.1 Оптимальность кода Хаффмана

**Лемма 1.** Существует оптимальный префиксный код, для которого наиболее редко встречающиеся два символа  $x$  и  $y$ :

1. Самые глубокие листья
2. Братья

*Доказательство.* Рассмотрим самый глубокий лист в дереве Хаффмана  $a$ ,  $f_a \leq x$ .

Т.к. дерево оптимально, у  $a$  есть брат  $b$ ,  $f_b \leq y$ .

Поменяем местами  $a, b$  с  $x, y$ . Тогда  $\sum l_i f_i = A - l_a f_a - l_b f_b - l_y f_y + l_a f_x + l_x f_a + l_b f_y + l_y f_b = A + (l_a - l_x)(f_x - f_a) + (l_b - l_y)(f_y - f_b)$ . Первая и третья скобка  $\geq 0$ , вторая и четвертая  $\leq 0 \Rightarrow \sum l_i f_i \leq A$ , т.е. новое дерево оптимально.  $\square$

Заменим  $x$  и  $y$  на  $z$ , так что  $f_x + f_y = f_z$

$$\sum f_i l_i = A - f_z l_z + f_x l_x + f_y l_y = A - f_x l_z - f_y l_z + f_x (l_z + 1) + f_y (l_z + 1) = A + f_x + f_y$$

Таким образом, оптимизация дерева с  $z$  вместо  $x$  и  $y$  оптимизирует и дерево с  $x$  и  $y$ , поэтому код Хаффмана оптимален.

Но это не значит, что нельзя сжимать лучше, чем Хаффман.

## 1 Арифметическое кодирование

Пусть  $a$  встречается 1000 раз, а  $b - 1$ . Тогда скорее всего оптимально в первый бит писать 0, если в строке только  $a$ , иначе 1, а дальше — по Хаффману.

### 1.1 Кодирование

Пусть  $a$  встречается 3 раз,  $b - 2$ ,  $c - 1$ . Тогда закодируем строку  $ababac$ . Рассмотрим отрезок  $[0, 1]$  и поделим его в отношении частот символов, т.е. в точках  $\frac{1}{2}, \frac{5}{6}$ . Теперь проведем то же самое для отрезка  $[0, \frac{1}{2}]$ , т.е. разделим его в точках  $\frac{1}{4}$  и  $\frac{5}{12}$ . Теперь то же самое для  $[\frac{1}{4}, \frac{5}{12}]$ , т.к. он соответствует  $b$ , которая соответствует второму символу. В итоге получим некий отрезок  $[l, r]$ . Найдём в нем число вида  $\frac{p}{2^a}$ . Запишем  $p$  в двоичном виде с дополнением слева нулями до длины  $a$ .

## 1.2 Декодирование

Из длины кодового слова можно понять  $a$  — длина слова, и  $p$  — перевод из двоичного представления кода. В таком случае известна дробь  $\frac{p}{2^q}$  на отрезке  $[0, 1]$ . Как и в кодировании, делим этот отрезок на соответствующие части и спускаемся в часть, в которую попал  $\frac{p}{2^q}$ . Чтобы остановить построение, необходимо знать длину исходного слова.

$$\frac{f_a}{\sum f_i} \frac{f_b}{\sum f_i} \frac{f_c}{\sum f_i} = \frac{\prod_{i=1}^L f_{s[i]}}{L^L} = \frac{\prod_{i=1}^k f_i^{f_i}}{L^L}$$

Можно заметить, что алгоритм не работает, когда

$$\frac{1}{2^q} > len = r - l$$

Поэтому возьмем  $\frac{1}{2^q} \leq len$ .

$$\frac{1}{2^q} \leq \frac{\prod_{i=1}^k f_i^{f_i}}{L^L}$$

$$-q \leq \sum_{i=1}^k f_i \log_2 f_i - L \log_2 L = \sum_{i=1}^k f_i (\log_2 f_i - \log_2 L)$$

$$q \geq -L \sum_{i=1}^k \frac{f_i}{L} \log_2 \frac{f_i}{L} = -L \sum_{i=1}^k p_i \log_2 p_i \quad \boxed{\phantom{p_i \log_2 p_i}} \text{ — Энтропия Шеннона} = -LH(p_1, p_2 \dots p_k)$$

Оптимальность кодирования с учетом зависимостей между символами не определена, поэтому все рассматриваемые далее методы являются эвристиками.

## 2 Словарное кодирование

### 2.1 LZ

Используется zip.

Существуют следующие виды токенов:

1. символ

2. ссылка:  $abacaba \rightarrow abac(4, 3)$  (сдвинуться на 4 символа назад и вывести три символа),  $ab(2, 6) = ababab$

Оптимальное построение: Для каждого символа находим длиннейшую подстроку, начинающуюся с него. Если записать ссылку на неё, то делаем это, иначе пишем символ без оптимизации. Для оптимального времени построения нужны суффиксные деревья.

## 2.2 LZW

Каво

## 2.3 BWT

$\angle abacaba\$$ . Отсортируем все её циклические сдвиги.

$\$abacaba$   
 $a\$bacaba$   
 $aba\$caba$   
 $abacaba\$$   
 $acaba\$ab$   
 $ba\$abaca$   
 $bacaba\$a$   
 $caba\$aba$

Последний столбец — преобразование BWT.

Из того, что  $x$  часто встречается как подстрока, получаем много одинаковых символов подряд.

## 2.4 MTF — move to front

Исходно код символа равен его номеру в алфавите. Когда символ встречается, его код выводится и приравнивается к 0.

$aaaaaaaazzzz \rightarrow ?00000000?00$

Полученную строку можно эффективно кодировать.

## 2.5 bzip2