Табличная реализация декодера кода Хаффмана

5 января 2023 г. 1:22

Рассмотрим один момент, касаемо кодера.
Предположим, что кодовое слово имеет длину 3 бита. Нельзя записать в файл 3 бита, можно записать минимум байт. Таким образом, нужен буфер длины, кратной 8. Пока буфер не заполнен целиком, записываем биты в него. Когда заполнится, передаем его в файл, очищаем и снова пишем биты в него.

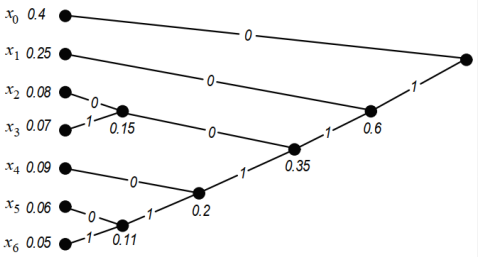
Как и кодер, декодер извлекает значения двоичных символов кодовых слов при помощи промежуточного буфера. Затем каждому кодовому слову сопоставляется сообщение из исходного алфавита.
Например, для источника с сообщениями $X = \{x_0, x_1, x_2, x_3\}$ и кодовыми словами $C = \{0, 10, 110, 111\}$ битовый поток 0 110 10 декодируется как последовательность сообщений $x_0 x_2 x_1$.

Таблица: Пример однобитной таблицы декодирования для $X = \{x_0, x_1, x_2, x_3\}$ и $C = \{0, 10, 110, 111\}$

Адрес, a	b_j	Адрес перехода, $A[a]$	$x[a]$
0	0	–	x_0
1	1	2	–
2	0	–	x_1
3	1	4	–
4	0	–	x_2
5	1	–	x_3

Как это работает:
На старте адрес = 0.
Предположим, нужно декодировать 110.
Первый бит - 1, прибавляем прибавляем 1 к адресу, получаем 1.
Видим адрес перехода 2, переходим по адресу 2.
Считываем следующий бит - снова 1.
Прибавляем к адресу, получаем 3. Переходим на адрес 3.
Видим адрес перехода 4, переходим на адрес 4.
Считываем следующий бит - это 0.
Прибавляем к адресу, он не изменяется.
Адреса перехода нет, поэтому выдаем сообщение x_2.

Пример для алфавита побольше:



a	b_j	$A[a]$	$x[a]$
0	0	–	x_0
1	1	2	–
2	0	–	x_1
3	1	4	–
4	0	6	–
5	1	8	–
6	0	–	x_2
7	1	–	x_3
8	0	–	x_4
9	1	10	–
10	0	–	x_5
11	1	–	x_6

---

j - позиция при считывании бита

b_j - j-й бит

a - Адрес

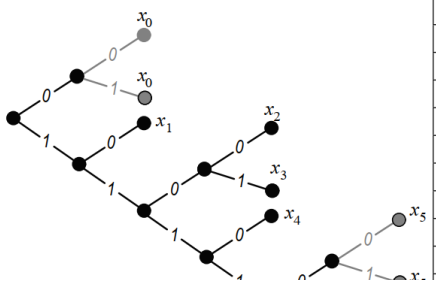
A[a] - адрес перехода

x[a] - сообщений

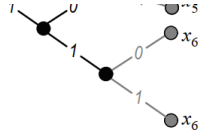
(абы что)----------

При практической реализации декодера Хаффмана используется заранее подготовленная таблица декодирования. Каждой строке таблицы сопоставляется адрес a, номер i декодированного символа x[a], адреса переходов , которые зависят от значения очередного двоичного символа на позиции j в битовом потоке, флаг, который принимает значение 0, если текущий узел дерева Хаффмана является концевым и значение 1 в противном случае, а также величина смещения в битовом потоке l[a] после обработки текущей строки.

Для уменьшения времени декодирования таблица может быть модифицирована таким образом, чтобы за один раз осуществлялся анализ более одного разряда двоичного потока. Это позволит выполнять меньшее количество переходов в таблице при декодировании очередного символа, а значит уменьшить время декодирования в целом. Модификация заключается в том, что мы достраиваем дерево, учитывая то, что длина кодового слова кратна, например, для двухбитного декодера двум. И уже согласно модифицированному дереву формируем таблицу.



a	$b_j b_{j+1}$	$l[a]$	$A[a]$	$x[a]$
0	00	1	–	x_0
1	01	1	–	x_0
2	10	2	–	x_1
3	11	–	4	–
4	00	4	–	x_2
5	01	4	–	x_3
6	10	4	–	x_4
7	11	–	8	–
8	00	5	–	x_5
9	01	5	–	x_5



10	10	5	–	x_6
11	11	5	–	x_6

сколько мы реально бит использовали

В каком случае код Хаффмана не избыточный? Т.е. когда он достигает энтропии?

Избыточность $R = \overline{l} - H = 0$

$\sum_i p_i l_i = -\sum_i p_i log\left(p_i\right)$

Ответ: в случае, если вероятности - степени двойки

Например, если вероятности 1/2, 1/4, 1/4, то избыточность = 0

Есть двоичный источник, т.е. алфавит {0,1}

Вероятность нуля 0,9

Вероятность единицы 0,1

(Максимальная энтропия была бы при вероятностях 0,5 и была бы равна 1)

Для этого случая будет код Хаффмана {0,1}

То есть по сути принимаем один бит и кодируем его так же одним битом

Это тот самый редкий случай, когда избыточность кода Хаффмана большая

Еще хуже будет, если вероятности = 0 и 1, тогда энтропия будет 0, и кодер будет вынужден тратить по 1 биту постоянно

Если сталкиваемся с подобной ситуацией, можем применить расширение алфавита

Например, от алфавита {0,1} переходим к алфавиту {1, 01, 001, 000}, считаем вероятности: {0,1; 0,9*0,1; 0,9*0,9*0,1; 0,9*0,9*0,9} и строим код Хаффмана для него