**Код Хаффмана**

**Алгоритм Хаффмана** — [жадный алгоритм](https://ru.wikipedia.org/wiki/%D0%96%D0%B0%D0%B4%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) оптимального [префиксного](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D1%84%D0%B8%D0%BA%D1%81%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) [кодирования](https://ru.wikipedia.org/wiki/%D0%AD%D0%BD%D1%82%D1%80%D0%BE%D0%BF%D0%B8%D0%B9%D0%BD%D0%BE%D0%B5_%D0%BA%D0%BE%D0%B4%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) алфавита с минимальной [избыточностью](https://ru.wikipedia.org/wiki/%D0%98%D0%B7%D0%B1%D1%8B%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C). Был разработан в [1952 году](https://ru.wikipedia.org/wiki/1952_%D0%B3%D0%BE%D0%B4) аспирантом [Массачусетского технологического института](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B0%D1%87%D1%83%D1%81%D0%B5%D1%82%D1%81%D0%BA%D0%B8%D0%B9_%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%81%D1%82%D0%B8%D1%82%D1%83%D1%82) [Дэвидом Хаффманом](https://ru.wikipedia.org/wiki/%D0%A5%D0%B0%D1%84%D1%84%D0%BC%D0%B0%D0%BD,_%D0%94%D1%8D%D0%B2%D0%B8%D0%B4) при написании им курсовой работы. В настоящее время используется во многих программах сжатия данных.

Кодирование Хаффмана.

Один из первых алгоритмов эффективного кодирования информации был предложен Д. А. Хаффманом в 1952 году. Идея алгоритма состоит в следующем: зная вероятности символов в сообщении, можно описать процедуру построения кодов переменной длины, состоящих из целого количества битов. Символам с большей вероятностью ставятся в соответствие более короткие коды. Коды Хаффмана обладают свойством [префиксности](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D1%84%D0%B8%D0%BA%D1%81%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) (то есть ни одно кодовое слово не является префиксом другого), что позволяет однозначно их декодировать.

Классический алгоритм Хаффмана на входе получает таблицу частот встречаемости символов в сообщении. Далее на основании этой таблицы строится дерево кодирования Хаффмана (Н-дерево).[[1]](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%A5%D0%B0%D1%84%D1%84%D0%BC%D0%B0%D0%BD%D0%B0#cite_note-1)

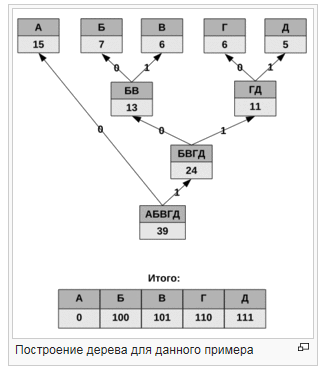
1. Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который может быть равен либо вероятности, либо количеству вхождений символа в сжимаемое сообщение.
2. Выбираются два свободных узла дерева с наименьшими весами.
3. Создается их родитель с весом, равным их суммарному весу.
4. Родитель добавляется в список свободных узлов, а два его потомка удаляются из этого списка.
5. Одной дуге, выходящей из родителя, ставится в соответствие бит 1, другой — бит 0.
6. Шаги, начиная со второго, повторяются до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева.

Допустим, у нас есть следующая таблица частот:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Символ** | **А** | **Б** | **В** | **Г** | **Д** |
| **Частота** | 15 | 7 | 6 | 6 | 5 |

Этот процесс можно представить как построение [дерева](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_(%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%BE%D0%B2)), корень которого — символ с суммой вероятностей объединенных символов, получившийся при объединении символов из последнего шага, его n0 потомков — символы из предыдущего шага и т. д.

Чтобы определить код для каждого из символов, входящих в сообщение, мы должны пройти путь от листа дерева, соответствующего текущему символу, до его корня, накапливая биты при перемещении по ветвям дерева (первая ветвь в пути соответствует младшему биту). Полученная таким образом последовательность битов является кодом данного символа, записанным в обратном порядке.



Для данной таблицы символов коды Хаффмана будут выглядеть следующим образом.

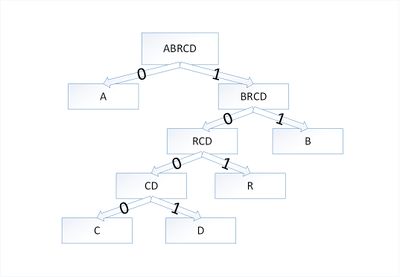
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Символ** | **А** | **Б** | **В** | **Г** | **Д** |
| **Код** | 0 | 100 | 101 | 110 | 111 |

Поскольку ни один из полученных кодов не является префиксом другого, они могут быть однозначно декодированы при чтении их из потока. Кроме того, наиболее частый символ сообщения А закодирован наименьшим количеством бит, а наиболее редкий символ Д — наибольшим.

При этом общая длина сообщения, состоящего из приведённых в таблице символов, составит 87 бит (в среднем 2,2308 бита на символ). При использовании равномерного кодирования общая длина сообщения составила бы 117 бит (ровно 3 бита на символ). Заметим, что [энтропия](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%8D%D0%BD%D1%82%D1%80%D0%BE%D0%BF%D0%B8%D1%8F) источника, независимым образом порождающего символы с указанными частотами, составляет ~2,1858 бита на символ, то есть [избыточность](https://ru.wikipedia.org/wiki/%D0%98%D0%B7%D0%B1%D1%8B%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%B8) построенного для такого источника кода Хаффмана, понимаемая как отличие среднего числа бит на символ от энтропии, составляет менее 0,05 бит на символ.

Классический алгоритм Хаффмана имеет ряд существенных недостатков. Во-первых, для восстановления содержимого сжатого сообщения декодер должен знать таблицу частот, которой пользовался кодер. Следовательно, длина сжатого сообщения увеличивается на длину таблицы частот, которая должна посылаться впереди данных, что может свести на нет все усилия по сжатию сообщения. Кроме того, необходимость наличия полной частотной статистики перед началом собственно кодирования требует двух проходов по сообщению: одного для построения модели сообщения (таблицы частот и Н-дерева), другого для собственно кодирования. Во-вторых, избыточность кодирования обращается в ноль лишь в тех случаях, когда вероятности кодируемых символов являются обратными степенями числа 2. В-третьих, для источника с энтропией, не превышающей 1 (например, для двоичного источника), непосредственное применение кода Хаффмана бессмысленно.

Пример.

[](http://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:Huffman_abracadabra.jpg)

Дерево Хаффмана для слова abracadabra

Закодируем слово abracadabra. Тогда алфавит будет A= \{a, b, r, c, d\}, а набор весов (частота появления символов алфавита в кодируемом слове) W=\{5, 2, 2, 1, 1\}:

В дереве Хаффмана будет 5 узлов:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Узел** | **a** | **b** | **r** | **с** | **d** |
| Вес | 5 | 2 | 2 | 1 | 1 |

По алгоритму возьмем два символа с наименьшей частотой — это c и d. Сформируем из них новый узел cd весом 2 и добавим его к списку узлов:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Узел** | **a** | **b** | **r** | **cd** |
| Вес | 5 | 2 | 2 | 2 |

Затем опять объединим в один узел два минимальных по весу узла — r и cd:

|  |  |  |  |
| --- | --- | --- | --- |
| **Узел** | **a** | **rcd** | **b** |
| Вес | 5 | 4 | 2 |

Еще раз повторим эту же операцию, но для узлов rcd и b:

|  |  |  |
| --- | --- | --- |
| **Узел** | **brcd** | **a** |
| Вес | 6 | 5 |

На последнем шаге объединим два узла — brcd и a:

|  |  |
| --- | --- |
| **Узел** | **abrcd** |
| Вес | 11 |

Остался один узел, значит, мы пришли к корню дерева Хаффмана (смотри рисунок). Теперь для каждого символа выберем кодовое слово (бинарная последовательность, обозначающая путь по дереву к этому символу от корня):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Символ** | **a** | **b** | **r** | **с** | **d** |
| Код | 0 | 11 | 101 | 1000 | 1001 |

Таким образом, закодированное слово abracadabraбудет выглядеть как 01110101000010010111010. Длина закодированного слова — 23 бита. Стоит заметить, что если бы мы использовали алгоритм кодирования с одинаковой длиной всех кодовых слов, то закодированное слово заняло бы 33 бита, что существенно больше.

# Постановка задачи.

Вход программы: некоторый текст.

Результаты работы программы:

1) таблица частот.

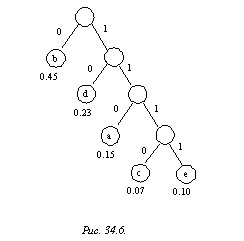
2) код каждого символа.

3) закодированный текст.

4) раскодировать последовательность кодов, которая вводится, используя построенное дерево.

Помощь к заданию.

Для символов с вероятностью появления a(0.15), b(0.45), c(0.07), d(0.23), e(0.10), построим двоичное дерево по алгоритму Хаффмана.



Коды символов как путь в двоичном дереве:  
a - 110  
b - 0  
c - 1110  
d - 10  
e – 1111

*Выбор структур данных*

1. Для представления двоичных деревьев выберем массив, элементами которого является запись. Для каждого узла дерева в записи сохраним информацию о его левом, правом сыне и родителе. Количество элементов в массиве соответствует количеству узлов дерева.

function tree (lson, rson,parent, stroca) {

this.lson=lson

this.rson=rson

this.parent=parent

this.stroca=stroca

}

T=new Array()

В исходном состоянии каждый узел представляет собой изолированное дерево, состоит из одного узла и соответственно не имеет родителей и сыновей. В начальном состоянии такой массив записей содержит нули во всех полях для каждого элемента. Результирующее состояние массива для дерева из рассмотренного выше примера:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| lson | 0 | 0 | 0 | 0 | 0 | 3 | 1 | 4 | 2 |
| rson | 0 | 0 | 0 | 0 | 0 | 5 | 6 | 7 | 8 |
| parent | 7 | 9 | 6 | 8 | 6 | 7 | 8 | 9 | 0 |
| stroca | a | b | c | d | e | ce | ace | dace | bdace |

Индексам от 1 до 5 соответствуют узлы с метками «a», «b», «c», «d», «e», индексы 6-9 обозначают внутренние узлы дерева.

2. Для динамического представления объединяющихся деревьев используем массив, в котором для каждого узла-корня дерева создана запись с информацией о корне дерева и его весе (вероятности).

function forest(wes, root) {

this.wes=wes

this.root=root

}

F=new Array()

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| i | 1 | 2 | 3 | 4 | 5 |
| wes | 0.15 | 0.45 | 0.07 | 0.23 | 0.10 |
| root | 1 | 2 | 3 | 4 | 5 |

 Реализация алгоритма Хаффмана:

n - количество узлов-символов

//Заполнение массивов узлов-корней деревьев F, T.

for i=1 to n do begin  
T[i] =new tree(0,0,0,символ)

F[i]= new forest(частота символа,i)

end;

lastusel =n //индекс-номер последнего элемента в массиве Т

lasttree=n //индекс-номер последнего элемента в массиве F

while lasttree >1 do

begin

// присваивает переменным first, second индексы массива F , соответствующие деревьям с наименьшими весами.

//инициализация переменных first, second рассматриваются первые два дерева

if F[1].wes<=F[2].wes  
then begin first=1; second=2 end  
else begin first=2; second=1 end;  
for i= 3 to lasttree  do  
if F[i].wes < F[first].wes then begin second=first; first=i end  
else if F[i].wes < F[second].wes then second=i;

//

// создаем новый узел   
lastusel:=lastusel+1 //ячейка T[lastusel] для нового узла  
T[lastusel] =new tree (F[first].root, F[second].root,0, T[F[first].root].stroca+T[F[second].root].stroca)

T[F[first].root].parent=lastusel;  
T[F[second]. root].parent=lastusel;  
//

//Замена в дереве F, массив F уменьшается на одну запись

F[first].wes=F[first].wes+F[second].wes;  
F[first].root= lastusel;

F[second]=F[lasttree]  
lasttree:=lasttree - 1;  
end;

//определение кода каждого символа в дереве Хаффмана  
for i=1 to n do begin  
k=i; p=k;

str=’’  
do  
p=T[p].parent;  
if T[p].lson=k then str=’0’+str;  
if T[p].rson=k then str=’1’+str;  
k=p;  
while T[p].parent<>0;  
Вывести ('символ ', T[i].stroca,' ',str)  
end;