**19. Оптимальное кодирование. Алгоритм Хаффмена**

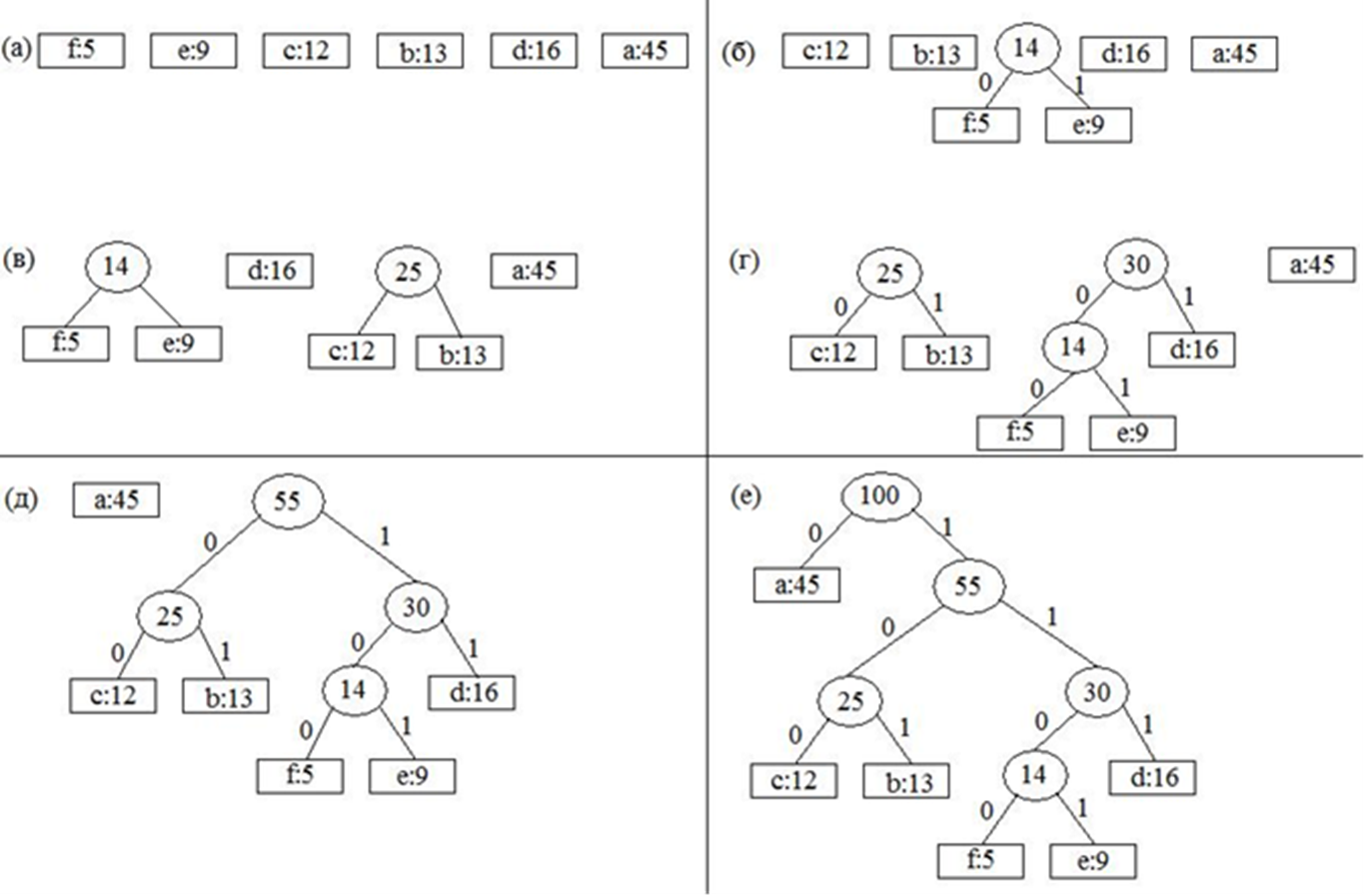
Оптимально закодированным будем считать такой код, при котором на передачу сообщений затрачивается минимальное время. Если на передачу каждого элементарного символа (0 или 1) тратиться одно и то же время, то оптимальным будет такой код, который будет иметь минимально возможную длину.

**Построение кода Хаффмена**

Коды Хаффмена широко используются при сжатия информации ( ситуации выигрыш может составить от 20% до 90% в зависимое файла). Алгоритм Хаффмена находит оптимальные коды символов ( исходя из частоты использования этих символов в сжимаемом тексте) с помощью жадного выбора.

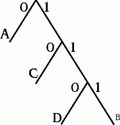
Хаффмен построил жадный алгоритм, который строит оптимальный префиксный код. Этот код называется кодом Хаффмена (Huffman соde). Алгоритм строит дерево Т, соответствующее оптимальному коду, снизу вверх, начиная с множества из |С| листьев и делая |С| - 1 «слияний». Мы предполагаем, что для каждого символа с ϵ С задана его частота f[с]. Для нахождения двух объектов подлежащих слиянию, используется очередь с приоритетами Q, использующая частоты f в качестве рангов — сливаются два объекта с наименьшими частотами. В результате слияния получается новый объект (внутренняя вершина), частота которого считается равной сумме частот двух сливаемых объектов.

Работа этого алгоритма для нашего примера показана на рисунке. Поскольку имеется всего 6 букв, первоначально очередь имеет размер n=6, и для построения дерева нужно сделать 5 слияний. Префиксный код соответствует дереву, полученному в результате всех этих слияний.



Оценим время работы алгоритма, считая, что очередь Q реализована в виде двоичной кучи. Инициализацию Q в строке 2 можно провести за O(n) операций с помощью процедуры Build-Heap. Обращения к куче исполняется ровно n-1 раз; поскольку каждая операция с кучей требует времени O(lg(n)), общее время будет O(nlg(n)). Стало быть, время работы алгоритма Хаффмена для алфавита из n символов будет O(nlg(n)).

**Алгоритм Хаффмана**

Проще всего рассмотреть алгоритм Хаффмана на простейшем примере представленном на рисунке 1. Предположим, что нам надо заархивировать следующую символьную последовательность: "AAABCCD". Без архивации эта последовательность занимает 7 байт. С архивацией по методу RLE она бы выглядела бы так: 3,"A",1,"B",2,"C",1,"D" то есть возросла бы до 8-ми байтов. А алгоритм Хаффмана может сократить ее почти до двух байтов, и вот как это происходит.

Прежде всего, отметим, что разные символы встречаются в нашем тексте по-разному. Чаще всего присутствует буква "A". Можно составить таблицу частот:

Символ 'A','B','C,'D'

Количество повторений '3','1','2','1'

Затем эта таблица используется для построения так называемого двоичного дерева (рис 1). Именно это дерево используется для генерации нового сжатого кода.

Левые ветви дерева помечены кодом 0,а правые-1. Имея такое дерево, легко найти любого символа, если идти от вершины к нужному символу. Итак в нашем случае алгоритм выглядит так:

Если символ равен "A" то ему присваивается двоичный ноль, в противном случае - единица и рассматривается следующий бит, если символ - "C", то он получит код 10, в противном случае 11. Если символ "D", то его код - 110, в противном случае-111.

Обратите внимание на то, что в алгоритме Хаффмана разные символы будут иметь разную битовую длину, и не надо иметь ни какого разделителя между символами. Например, если вы попробуете декодировать с помощью рисунка 1 последовательность:

0001111111010110

то получите текст:"AAABBCCD"

0 0 0 111 111 10 10 110A A A B B C C D

а, рассмотренным нами выше текст "AAABCCD" займет всего 13 бит (а это меньше двух байтов).

A A A B C C D0 0 0 111 10 10 110