

множества можно менять произвольным образом. Пусть a_i — i -й элемент множества A , а b_i — i -й элемент множества B после переупорядочения. Составим величину $\prod_{i=1}^n a_i^{b_i}$. Сформулируйте алгоритм, который бы максимизировал эту величину. Докажите, что ваш алгоритм действительно максимизирует указанную величину, и определите время его работы.

16.3 Коды Хаффмана

Коды Хаффмана (Huffman codes) — широко распространенный и очень эффективный метод сжатия данных, который, в зависимости от характеристик этих данных, обычно позволяет сэкономить от 20% до 90% объема. Мы рассматриваем данные, представляющие собой последовательность символов. В жадном алгоритме Хаффмана используется таблица, содержащая частоты появления тех или иных символов. С помощью этой таблицы определяется оптимальное представление каждого символа в виде бинарной строки.

Предположим, что имеется файл данных, состоящий из 100 000 символов, который требуется сжать. Символы в этом файле встречаются с частотой, представленной в табл. 16.1. Таким образом, всего файл содержит шесть различных символов, а, например, символ a встречается в нем 45 000 раз.

Таблица 16.1. Задача о кодировании последовательности символов

	a	b	c	d	e	f
Частота (в тысячах)	45	13	12	16	9	5
Кодовое слово фиксированной длины	000	001	010	011	100	101
Кодовое слово переменной длины	0	101	100	111	1101	1100

Существует множество способов представить подобный файл данных. Рассмотрим задачу по разработке *бинарного кода* символов (binary character code; или для краткости просто *кода*), в котором каждый символ представляется уникальной бинарной строкой. Если используется *код фиксированной длины*, или *равномерный код* (fixed-length code), то для представления шести символов понадобится 3 бита: $a = 000$, $b = 001$, ..., $f = 101$. При использовании такого метода для кодирования всего файла понадобится 300 000 битов. Можно ли добиться лучших результатов?

С помощью *кода переменной длины*, или *неравномерного кода* (variable-length code), удастся получить значительно лучшие результаты, чем с помощью кода фиксированной длины. Это достигается за счет того, что часто встречающимся символам сопоставляются короткие кодовые слова, а редко встречающимся — длинные. Такой код представлен в последней строке табл. 16.1. В нем символ a

представлен 1-битовой строкой 0, а символ f — 4-битовой строкой 1100. Для представления файла с помощью этого кода потребуется

$$(45 \cdot 1 + 13 \cdot 3 + 12 \cdot 3 + 16 \cdot 3 + 9 \cdot 4 + 5 \cdot 4) \cdot 1000 = 224\,000 \text{ битов.}$$

Благодаря этому дополнительно экономится 25% объема. Кстати, как мы вскоре убедимся, для рассматриваемого файла это оптимальная кодировка символов.

Префиксные коды

Мы рассматриваем только те коды, в которых никакое кодовое слово не является префиксом какого-то другого кодового слова. Такие коды называются *префиксными* (prefix codes)². Можно показать (хотя здесь мы не станем этого делать), что оптимальное сжатие данных, которого можно достичь с помощью кодов, всегда достижимо при использовании префиксного кода, поэтому рассмотрение одних лишь префиксных кодов не приводит к потере общности.

Для любого бинарного кода символов кодирование текста — очень простой процесс: надо просто соединить кодовые слова, представляющие каждый символ в файле. Например, в кодировке с помощью префиксного кода переменной длины, представленного в табл. 16.1, трехсимвольный файл `abc` имеет вид $0 \cdot 101 \cdot 100 = 0101100$, где символом “ \cdot ” обозначена операция конкатенации.

Предпочтение префиксным кодам отдается из-за того, что они упрощают декодирование. Поскольку никакое кодовое слово не выступает в роли префикса другого, кодовое слово, с которого начинается закодированный файл, определяется однозначно. Начальное кодовое слово легко идентифицировать, преобразовать его в исходный символ и продолжить декодирование оставшейся части закодированного файла. В рассматриваемом примере строка 001011101 однозначно раскладывается на подстроки $0 \cdot 0 \cdot 101 \cdot 1101$, что декодируется как `aabe`.

Для упрощения процесса декодирования требуется удобное представление префиксного кода, чтобы кодовое слово можно было легко идентифицировать. Одним из таких представлений является бинарное дерево, листьями которого являются кодируемые символы. Бинарное кодовое слово, представляющее символ, интерпретируется как путь от корня к этому символу. В такой интерпретации 0 означает “перейти к левому дочернему узлу”, а 1 — “перейти к правому дочернему узлу”. На рис. 16.3 показаны такие деревья для двух кодов, взятых из нашего примера. Каждый лист на рисунке помечен соответствующим ему символом и частотой появления, а внутренний узел — суммой частот листьев его поддерева. В части *a* рисунка приведено дерево, соответствующее коду фиксированной длины, где $a = 000, \dots, f = 101$. В части *b* показано дерево, соответствующее оптимальному префиксному коду $a = 0, b = 101, \dots, f = 1100$. Заметим,

²Возможно, лучше подошло бы название “беспрефиксные коды”, однако “префиксные коды” — стандартный в литературе термин.

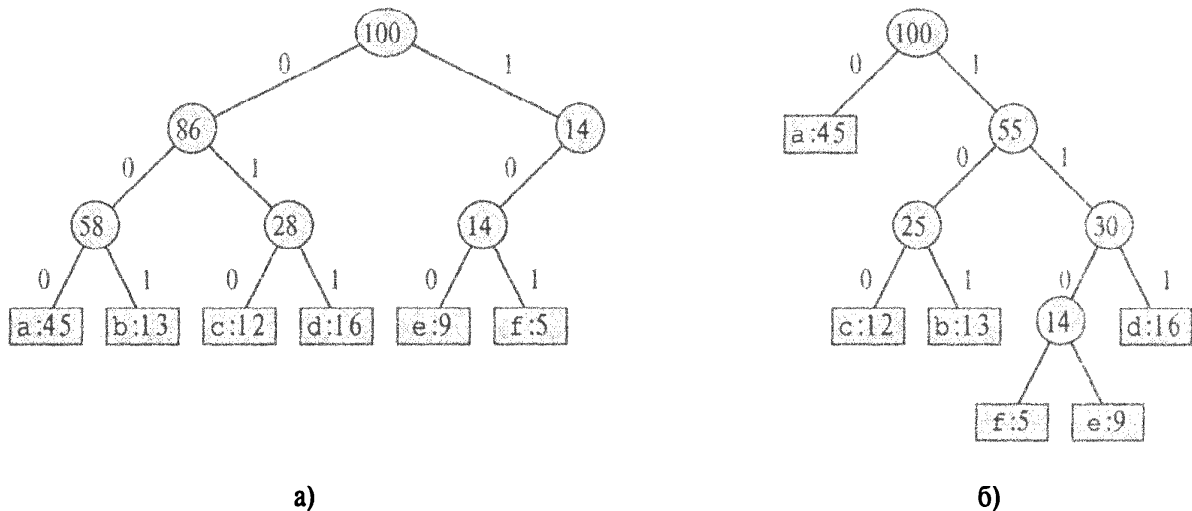


Рис. 16.3. Деревья, соответствующие схемам кодирования, приведенным в табл. 16.1

что изображенные на рисунке деревья не являются бинарными деревьями поиска, поскольку листья в них не обязательно расположены в порядке сортировки, а внутренние узлы не содержат ключей символов.

Оптимальный код файла всегда может быть представлен *полным* бинарным деревом, в котором у каждого узла (кроме листьев) имеется по два дочерних узла (см. упражнение 16.3-1). Код фиксированной длины, представленный в рассмотренном примере, не является оптимальным, поскольку соответствующее ему дерево, изображенное на рис. 16.3а, — неполное бинарное дерево: некоторые слова кода начинаются с 10..., но ни одно из них не начинается с 11.... Поскольку в нашем обсуждении мы можем ограничиться только полными бинарными деревьями, можно утверждать, что если C — алфавит, из которого извлекаются кодируемые символы, и все частоты, с которыми встречаются символы, положительны, то дерево, представляющее оптимальный префиксный код, содержит ровно $|C|$ листьев, по одному для каждого символа из множества C , и ровно $|C| - 1$ внутренних узлов (см. упражнение Б.5-3).

Если имеется дерево T , соответствующее префиксному коду, легко подсчитать количество битов, которые потребуются для кодирования файла. Обозначим через $f(c)$ частоту символа c из множества C в файле, а через $d_T(c)$ — глубину листа, представляющего этот символ в дереве. Заметим, что $d_T(c)$ является также длиной слова, кодирующего символ c . Таким образом, для кодировки файла необходимо количество битов, равное

$$B(T) = \sum_{c \in C} f(c) d_T(c) \quad (16.5)$$

Назовем эту величину *стоимостью* дерева T .

Построение кода Хаффмана

Хаффман изобрел жадный алгоритм, позволяющий составить оптимальный префиксный код, который получил название *код Хаффмана*. В соответствии с линией, намеченной в разделе 16.2, доказательство корректности этого алгоритма основывается на свойстве жадного выбора и оптимальной подструктуре. Вместо того чтобы демонстрировать, что эти свойства выполняются, а затем разрабатывать псевдокод, сначала мы представим псевдокод. Это поможет прояснить, как алгоритм осуществляет жадный выбор.

В приведенном ниже псевдокоде предполагается, что C — множество, состоящее из n символов, и что каждый из символов $c \in C$ — объект с определенной частотой $f(c)$. В алгоритме строится дерево T , соответствующее оптимальному коду, причем построение идет в восходящем направлении. Процесс построения начинается с множества, состоящего из $|C|$ листьев, после чего последовательно выполняется $|C| - 1$ операций “слияния”, в результате которых образуется конечное дерево. Для идентификации двух наименее часто встречающихся объектов, подлежащих слиянию, используется очередь с приоритетами Q , ключами в которой являются частоты f . В результате слияния двух объектов образуется новый объект, частота появления которого является суммой частот объединенных объектов:

HUFFMAN(C)

```

1   $n \leftarrow |C|$ 
2   $Q \leftarrow C$ 
3  for  $i \leftarrow 1$  to  $n - 1$ 
4      do Выделить память для узла  $z$ 
5           $left[z] \leftarrow x \leftarrow \text{EXTRACT\_MIN}(Q)$ 
6           $right[z] \leftarrow y \leftarrow \text{EXTRACT\_MIN}(Q)$ 
7           $f[z] \leftarrow f[x] + f[y]$ 
8          INSERT( $Q, z$ )
9  return EXTRACT_MIN( $Q$ )           ▷ Возврат корня дерева
```

Для рассмотренного ранее примера алгоритм Хаффмана выводит результат, приведенный на рис. 16.4. На каждом этапе показано содержимое очереди, элементы которой рассортированы в порядке возрастания их частот. На каждом шаге работы алгоритма объединяются два объекта (дерева) с самыми низкими частотами. Листья изображены в виде прямоугольников, в каждом из которых указана буква и соответствующая ей частота. Внутренние узлы представлены кругами, содержащими сумму частот дочерних узлов. Ребро, соединяющее внутренний узел с левым дочерним узлом, имеет метку 0, а ребро, соединяющее его с правым дочерним узлом, — метку 1. Слово кода для буквы образуется последовательностью меток на ребрах, соединяющих корень с листом, представляющим эту букву. По-

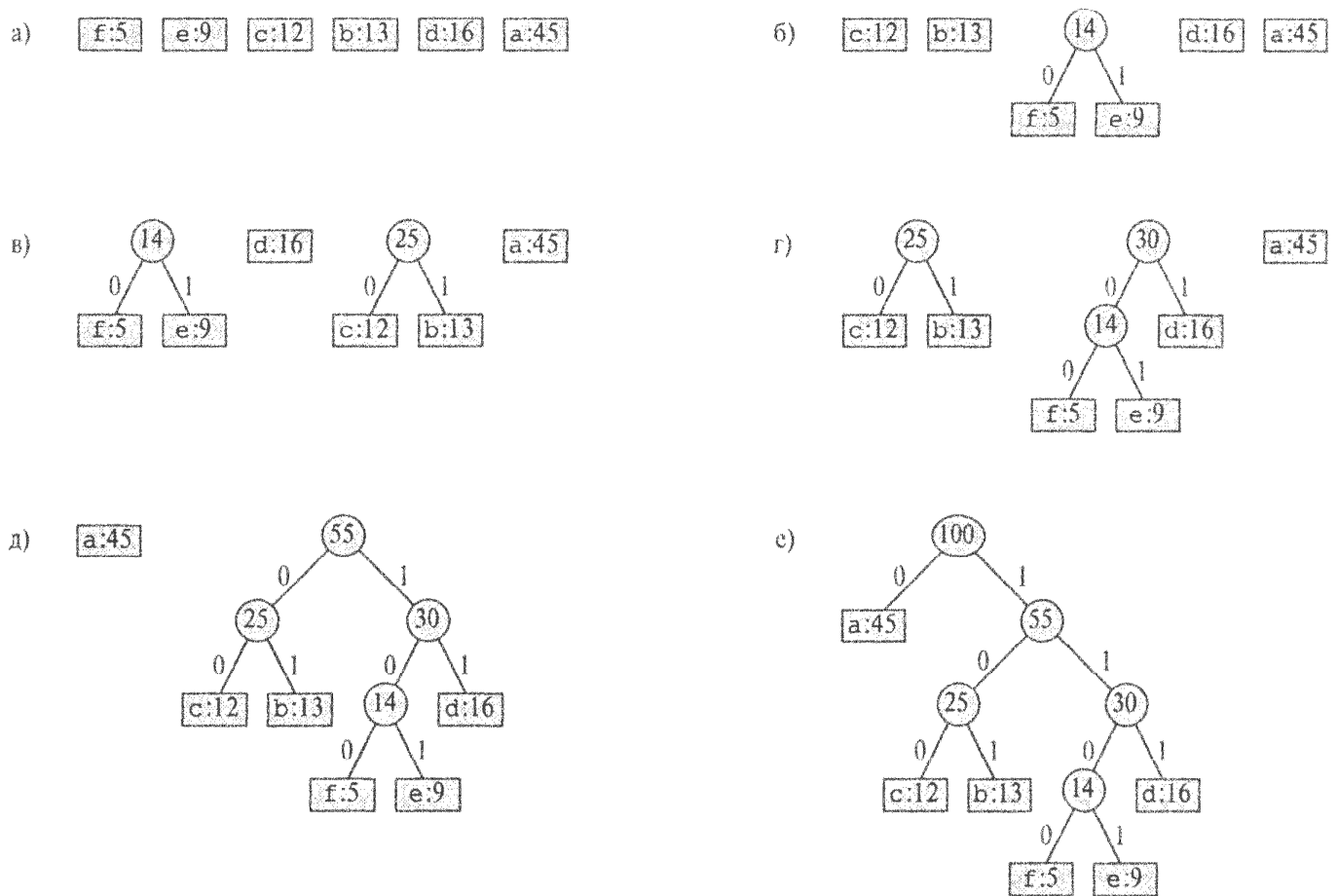


Рис. 16.4. Этапы работы алгоритма Хаффмана для частот, заданных в табл. 16.1

сколько данное множество содержит шесть букв, размер исходной очереди равен 6 (часть *a* рисунка), а для построения дерева требуется пять слияний. Промежуточные этапы изображены в частях *б–д*. Конечное дерево (рис. 16.4*е*) представляет оптимальный префиксный код. Как уже говорилось, слово кода для буквы — это последовательность меток на пути от корня к листу с этой буквой.

В строке 2 инициализируется очередь с приоритетами Q , состоящая из элементов множества C . Цикл **for** в строках 3–8 поочередно извлекает по два узла, x и y , которые характеризуются в очереди наименьшими частотами, и заменяет их в очереди новым узлом z , представляющим объединение упомянутых выше элементов. Частота появления z вычисляется в строке 7 как сумма частот x и y . Узел x является левым дочерним узлом z , а y — его правым дочерним узлом. (Этот порядок является произвольным; перестановка левого и правого дочерних узлов приводит к созданию другого кода с той же стоимостью.) После $n - 1$ объединений в очереди остается один узел — корень дерева кодов, который возвращается в строке 9.

При анализе времени работы алгоритма Хаффмана предполагается, что Q реализована как бинарная неубывающая пирамида (см. главу 6). Для множества C , состоящего из n символов, инициализацию очереди Q в строке 2 можно выполнить

за время $O(n)$ с помощью процедуры BUILD_MIN_HEAP из раздела 6.3. Цикл `for` в строках 3–8 выполняется ровно $n - 1$ раз, и поскольку для каждой операции над пирамидой требуется время $O(\lg n)$, вклад цикла во время работы алгоритма равен $O(n \lg n)$. Таким образом, полное время работы процедуры HUFFMAN с входным множеством, состоящим из n символов, равно $O(n \lg n)$.

Корректность алгоритма Хаффмана

Чтобы доказать корректность жадного алгоритма HUFFMAN, покажем, что в задаче о построении оптимального префиксного кода проявляются свойства жадного выбора и оптимальной подструктуры. В сформулированной ниже лемме показано соблюдение свойства жадного выбора.

Лемма 16.2. Пусть C — алфавит, каждый символ $c \in C$ которого встречается с частотой $f[c]$. Пусть x и y — два символа алфавита C с самыми низкими частотами. Тогда для алфавита C существует оптимальный префиксный код, кодовые слова символов x и y в котором имеют одинаковую длину и отличаются лишь последним битом.

Доказательство. Идея доказательства состоит в том, чтобы взять дерево T , представляющее произвольный оптимальный префиксный код, и преобразовать его в дерево, представляющее другой оптимальный префиксный код, в котором символы x и y являются листьями с общим родительским узлом, причем в новом дереве эти листья находятся на максимальной глубине.

Пусть a и b — два символа, представленные листьями с общим родительским узлом, которые находятся на максимальной глубине дерева T . Предположим без потери общности, что $f[a] \leq f[b]$ и $f[x] \leq f[y]$. Поскольку $f[x]$ и $f[y]$ — две самые маленькие частоты (в указанном порядке), а $f[a]$ и $f[b]$ — две произвольные частоты, то выполняются соотношения $f[x] \leq f[a]$ и $f[y] \leq f[b]$. Как показано на рис. 16.5, в результате перестановки в дереве T листьев a и x получается дерево T' , а при последующей перестановке в дереве T' листьев b и y получается дерево T'' . Согласно уравнению (16.5), разность стоимостей деревьев T и T' равна

$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) = \\ &= f[x] d_T(x) + f[a] d_T(a) - f[x] d_{T'}(x) - f[a] d_{T'}(a) = \\ &= f[x] d_T(x) + f[a] d_T(a) - f[x] d_T(a) - f[a] d_T(x) = \\ &= (f[a] - f[x]) (d_T(a) - d_T(x)) \geq 0, \end{aligned}$$

поскольку величины $f[a] - f[x]$ и $d_T(a) - d_T(x)$ неотрицательны. Величина $f[a] - f[x]$ неотрицательна, потому что x — лист с минимальной частотой, величина $d_T(a) - d_T(x)$ неотрицательна, потому что a — лист на максимальной глубине

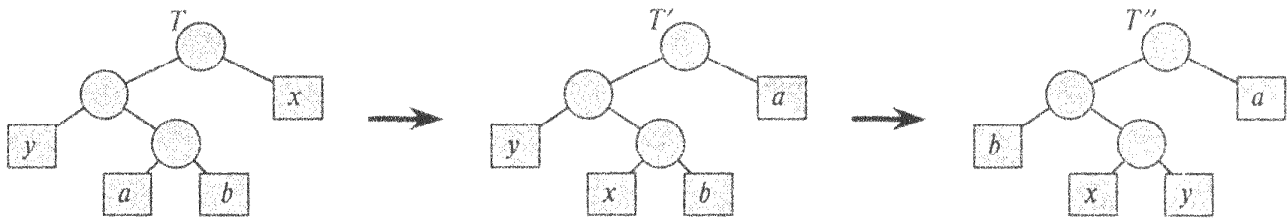


Рис. 16.5. Иллюстрация ключевых этапов доказательства леммы 16.2

в дереве T . Аналогично, перестановка листьев y и b не приведет к увеличению стоимости, поэтому величина $B(T') - B(T'')$ неотрицательна. Таким образом, выполняется неравенство $B(T'') \leq B(T)$, и поскольку T — оптимальное дерево, то должно также выполняться неравенство $B(T) \leq B(T'')$, откуда следует, что $B(T'') = B(T)$. Таким образом, T'' — оптимальное дерево, в котором x и y — находящиеся на максимальной глубине дочерние листья одного и того же узла, что и доказывает лемму. \square

Из леммы 16.2 следует, что процесс построения оптимального дерева путем объединения узлов без потери общности можно начать с жадного выбора, при котором объединению подлежат два символа с наименьшими частотами. Почему такой выбор будет жадным? Стоимость объединения можно рассматривать как сумму частот входящих в него элементов. В упражнении 16.3-3 предлагается показать, что полная стоимость сконструированного таким образом дерева равна сумме стоимостей его составляющих. Из всевозможных вариантов объединения на каждом этапе в процедуре HUFFMAN выбирается тот, в котором получается минимальная стоимость.

В приведенной ниже лемме показано, что задача о составлении оптимальных префиксных кодов обладает свойством оптимальной подструктуры.

Лемма 16.3. Пусть дан алфавит C , в котором для каждого символа $c \in C$ определены частоты $f[c]$. Пусть x и y — два символа из алфавита C с минимальными частотами. Пусть C' — алфавит, полученный из алфавита C путем удаления символов x и y и добавления нового символа z , так что $C' = C - \{x, y\} \cup \{z\}$. По определению частоты f в алфавите C' совпадают с частотами в алфавите C , за исключением частоты $f[z] = f[x] + f[y]$. Пусть T' — произвольное дерево, представляющее оптимальный префиксный код для алфавита C' . Тогда дерево T , полученное из дерева T' путем замены листа z внутренним узлом с дочерними элементами x и y , представляет оптимальный префиксный код для алфавита C .

Доказательство. Сначала покажем, что стоимость $B(T)$ дерева T можно выразить через стоимость $B(T')$ дерева T' , рассматривая стоимости компонентов из уравнения (16.5). Для каждого символа $c \in C - \{x, y\}$ выполняется соотношение

$d_T(c) = d_{T'}(c)$, следовательно, $f[c] d_T(c) = f[c] d_{T'}(c)$. Поскольку $d_T(x) = d_T(y) = d_{T'}(z) + 1$, получаем соотношение

$$\begin{aligned} f[x] d_T(x) + f[y] d_T(y) &= (f[x] + f[y]) (d_{T'}(z) + 1) = \\ &= f[z] d_{T'}(z) + (f[x] + f[y]), \end{aligned}$$

из которого следует равенство

$$B(T) = B(T') + f[x] + f[y],$$

или

$$B(T') = B(T) - f[x] - f[y].$$

Докажем лемму методом от противного. Предположим, дерево T не представляет оптимальный префиксный код для алфавита C . Тогда существует дерево T'' , для которого справедливо неравенство $B(T'') < B(T)$. Согласно лемме 16.2, x и y без потери общности можно считать дочерними элементами одного и того же узла. Пусть дерево T''' получено из дерева T'' путем замены элементов x и y листом z с частотой $f[z] = f[x] + f[y]$. Тогда можно записать:

$$B(T''') = B(T'') - f[x] - f[y] < B(T) - f[x] - f[y] = B(T'),$$

что противоречит предположению о том, что дерево T' представляет оптимальный префиксный код для алфавита C' . Таким образом, дерево T должно представлять оптимальный префиксный код для алфавита C . \square

Теорема 16.4. Процедура HUFFMAN дает оптимальный префиксный код.

Доказательство. Справедливость теоремы непосредственно следует из лемм 16.2 и 16.3. \square

Упражнения

- 16.3-1. Докажите, что бинарное дерево, которое не является полным, не может соответствовать оптимальному префиксному коду.
- 16.3-2. Определите оптимальный код Хаффмана для представленного ниже множества частот, основанного на первых восьми числах Фибоначчи.

a:1 b:1 c:2 d:3 e:5 f:8 g:13 h:21

Попытайтесь обобщить ответ на случай первых n чисел Фибоначчи.

- 16.3-3. Докажите, что полную стоимость дерева, представляющего какой-нибудь код, можно также вычислить как сумму комбинаций частот двух дочерних узлов по всем внутренним узлам.