Лекция 20

Тема: Сжатие данных

Конспект новой темы
Домашнее задание в конце лекции

Основные понятия

Для того чтобы сэкономить место на внешних носителях (жёстких дисках, флэш-дисках) и ускорить передачу данных по компьютерным сетям, нужно сжать данные — уменьшить информационный объём, сократить длину двоичного кода. Это можно сделать, устранив избыточность использованного кода.

Например, пусть текстовый файл объёмом 10 Кбайт содержит всего четырех различных символа: латинские буквы «A», «B», «C» и пробел. Вы знаете, что для кодирования одного из четырёх возможных вариантов достаточно 2 битов, поэтому использовать для его передачи обычное 8-битное кодирование символов невыгодно. Можно присвоить каждому из четырёх символов двухбитные коды, например, так:

A — 00, B — 01, C — 10, пробел — 11.

Тогда последовательность «ABA CABABA», занимающая 10 байтов в однобайтной кодировке, может быть представлена как цепочка из 20 битов:

00010011100001000100

Таким образом, нам удалось уменьшить информационный объём текста в 4 раза, и передаваться он будет в 4 раза быстрее. Однако непонятно, как раскодировать это сообщение, ведь получатель не знает, какой код был использован. Выход состоит в том, чтобы включить в сообщение служебную информацию — заголовок, в котором каждому коду будет сопоставлен ASCII-код символа. Условимся, что первый байт заголовка — это количество используемых символов N, а следующие N байтов — это ASCII-коды этих символов.

В данном случае заголовок занимает 5 байтов и выглядит так:

00000100_2	01000001_2	01000010_2	01000011_2	00100000_2
4 символа	A (код 65)	B (код 66)	C (код 67)	пробел (код 32)

Файл, занимающий 10 Кбайт в 8-битной кодировке, содержит 10 240 символов. В сжатом виде каждый символ кодируется двумя битами, кроме того, есть 5-байтный заголовок. Поэтому сжатый файл будет иметь объём

5 + 10240 • 2/8 байтов = 2565 байтов.

Коэффициент сжатия — это отношение размеров исходного и сжатого файлов.

В данном случае удалось сжать файл почти в 4 раза, коэффициент сжатия равен

k = 10 240/2565 ≈ 4.

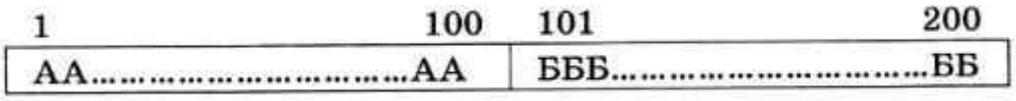
Если принимающая сторона «знает» формат файла (заголовок + закодированные данные), она сможет восстановить в точности его исходный вид. Такое сжатие называют сжатием без потерь. Оно используется для упаковки текстов, программ, данных, которые ни в коем случае нельзя искажать.

Сжатие без потерь — это такое уменьшение объёма закодированных данных, при котором можно восстановить их исходный вид из кода без искажений.

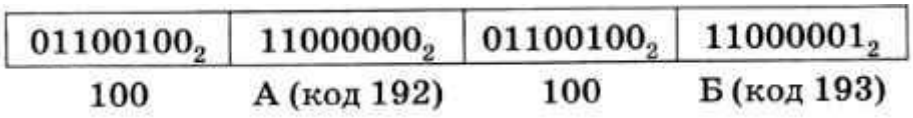
За счёт чего удалось сжать файл? Только за счёт того, что в файле была некоторая закономерность, избыточность — использовались только 4 символа вместо полного набора

Алгоритм RLE

При сжатии данных, в которых есть цепочки одинаковых кодов, можно применять ещё один простой алгоритм, который называется кодированием цепочек одинаковых символов (англ. RLE — Run Length Encoding). Представим себе файл, в котором записаны сначала 100 русских букв «А», а потом — 100 букв «Б»:



При использовании алгоритма RLE сначала записывается количество повторений первого символа, затем — сам первый символ, затем — количество повторений второго символа, затем — второй символ и т. д. В данном случае весь закодированный файл занимает 4 байта:



Таким образом, мы сжали файл в 50 раз за счёт того, что в нём снова была избыточность — цепочки одинаковых символов. Это сжатие без потерь, потому что, зная алгоритм упаковки, исходные данные можно точно восстановить из кода.

Очевидно, что такой подход будет приводить к увеличению (в 2 раза) объема данных в том случае, когда в файле нет соседних одинаковых символов. Чтобы улучшить результаты RLE-кодирования даже в этом наихудшем случае, алгоритм модифицировали следующим образом. Упакованная последовательность содержит управляющие байты, за каждым управляющим байтом следует один или несколько байтов данных. Если старший бит управляющего байта равен 1, то следующий за управляющим байт данных при распаковке нужно повторить столько раз, сколько записано в оставшихся 7 битах управляющего байта. Если же старший бит управляющего байта равен 0, то надо взять несколько следующих байтов данных без изменения. Сколько именно — записано в оставшихся 7 битах управляющего байта. Например, управляющий байт 10000111_2 говорит о том, что следующий за ним байт надо повторить 7 раз, а управляющий байт 00000100_2 — о том, что следующие за ним 4 байта надо взять без изменений. Например, последовательность

10001111_2	11000000_2	00000010_2	11000001_2	11000010_2
повтор 15	А (код 192)	2	Б (код 193)	В (код 194)

распаковывается в 17 символов: АААААААААААААААБВ.

Алгоритм RLE успешно использовался для сжатия рисунков, в которых большие области закрашены одним цветом, и некоторых звуковых данных. Сейчас вместо него применяют более совершенные, но более сложные методы. Один из них (алгоритм Хаффмана) мы рассмотрим далее. Алгоритм RLE используется, например, на одном из этапов кодирования рисунков в формате JPEG. Возможность использования RLE-сжатия есть также в формате BMP (для рисунков с палитрой 16 или 256 цветов).

Алгоритм Хаффмана

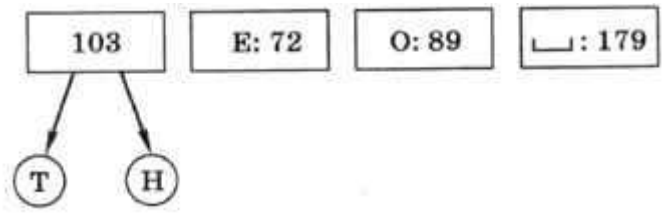
Было доказано, что в некоторых случаях кодирование Шеннона-Фано дает неоптимальное решение, и можно построить код, который ещё больше уменьшит длину кодовой последовательности. Например, в предыдущем примере (код Шеннона-Фано) пробел и буква «Т» имеют коды одинаковой длины, хотя буква «Т» встречается в 3,5 раза реже пробела. Через несколько лет Дэвид Хаффман, ученик Фано, разработал новый алгоритм кодирования и доказал его оптимальность.

Дэвид Хаффман (1925–1999)

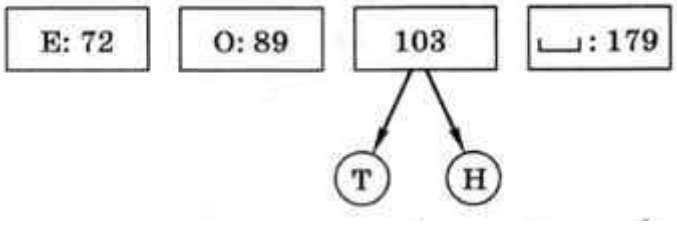
Построим код Хаффмана для того же самого примера. Сначала отсортируем буквы по увеличению частоты встречаемости:

Т: 50	Н: 53	Е: 72	О: 89	␣: 179

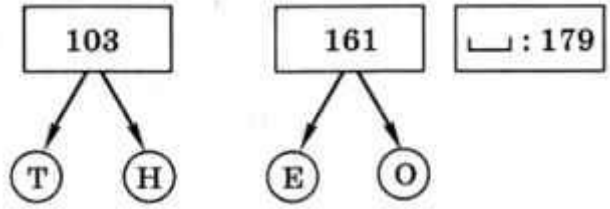
Затем берём две самые первые буквы, они становятся листьями дерева, а в узел, с которым они связаны, записываем сумму их частот:



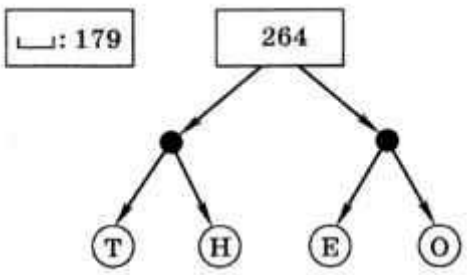
Таким образом, буквы, которые встречаются реже всего, получили самый длинный код. Снова сортируем буквы по возрастанию частоты, но для букв «Т» и «Н» используется их суммарная частота:



Повторяем ту же процедуру для букв «Е» и «О», частоты которых оказались минимальными, и сортируем по возрастанию частоты:



Теперь объединяем уже не отдельные буквы, а пары, и снова сортируем



На последнем шаге остаётся объединить символ «пробел» с деревом, которое построено для остальных символов. У каждой стрелки, идущей влево от какого-то узла, ставим код 0, а у каждой стрелки, идущей вправо — код 1 (рис. 1.4).

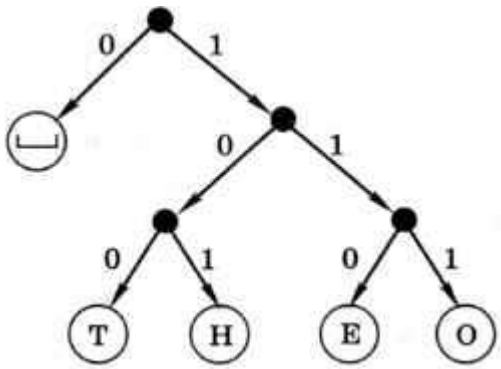


Рис. 1.4

По этому дереву, спускаясь от корня к листьям-символам, получаем коды Хаффмана, обеспечивающие оптимальное сжатие с учётом частоты встречаемости символов:

␣ — О, Т — 100, О — 111, Е — 110, Н — 101.

Легко проверить, что этот код также удовлетворяет условию Фано.

Сравним эффективность рассмотренных выше методов. Для алфавита из 5 символов при равномерном кодировании нужно использовать 3 бита на каждый символ, так что общее число битов в сообщении равно (179 + 89 + 72 + 53 + 50) • 3 = 1329 битов. При кодировании методом Шеннона-Фано получаем 179 • 2 + 89 • 2 + 72 • 3 + 53 • 3 + 50 • 2 = 1011 битов, коэффициент сжатия составляет 1329/1011 ≈ 1,31. Использование кода Хаффмана даёт последовательность длиной 179 • 1 + 89 • 3 + 72 • 3 + 53 • 3 + 50 • 3 = 971 бит, коэффициент сжатия равен 1,37. В сравнении со случаем, когда для передачи используется однобайтный код (8 битов на символ), выигрыш получается ещё более весомым: кодирование Хаффмана сжимает данные примерно в 3,65 раза. Обратим внимание, что сжать данные удалось за счёт избыточности: мы использовали тот факт, что некоторые символы встречаются чаще, а некоторые — реже.

Алгоритм кодирования Хаффмана и сейчас широко применяется благодаря своей простоте, высокой скорости кодирования и отсутствию патентных ограничений (он может быть использован свободно). Например, он применяется на некоторых этапах при сжатии рисунков (в формате JPEG) и звуковой информации (в формате MP3).

Построение бинарного дерева - ссылка

Домашнее задание

1. С помощью алгоритма RLE закодируйте сообщение «ВАААААВАААРРРРРРРРРР».
2. Постройте дерево, соответствующее коду А — 0, Б — 1, В — 00, Г — 01, Д — 10, Е — 11. Является ли этот код префиксным? Как это определить, посмотрев на дерево?