Алгоритмы сжатия информации

Д. Мастрюков

Часть 1. Сжатие по Хаффмену

Введение

Эта статья — первая в цикле, посвященном систематическому изложению алгоритмов сжатия информации. Почему эта серия статей может быть интересна для Вас?

Вы хотите знать, как работают такие популярные программы сжатия информации, как PKZIP, ARJ, LHA, STACKER, SuperStor? Те алгоритмы сжатия, которыми они пользуются, будут описаны подробно и систематично, разумеется, включая и исходные тексты работающих программ.

Вы хотите использовать сжатие информации в программах, которые Вы пишете? Вы сможете с/легкостью заимствовать и модифицировать приведенные примеры. Если Вы хотите впоследствии использовать эти программы в коммерческих целях, то ссылки на владельцев патентов на приведенные алгоритмы помогут Вам не попасть в двусмысленное положение.

Вас интересует MultiMedia? Здесь Вы сможете найти ответ на вопрос, почему алгоритмы сжатия графических образов с потерями стали ключевым фактором, сделавшим возможной эту технологическую революцию. Примеры, реализующие часть известного стандарта JPEG, дадут Вам возможность самим экспериментировать с его возможностями.

Вас интересует научная сторона вопросов сжатия информации? Систематичность изложения и короткие математические выкладки помогут Вам получить достаточно основательные знания по этой проблеме.

Магическая формула:
СЖАТИЕ = МОДЕЛИРОВАНИЕ + КОДИРОВАНИЕ

Вообще говоря, сжатие информации представляет собой процесс обработки символов некоторого сооб-

Содержание последующих статей

Сжатие без потерь

Алгоритм Хаффмена. Адаптивное кодирование Хаффмена
Арифметическое кодирование
Алгоритмы группы LZ (Lempel-Ziv)
Алгоритм LZW (Lempel-Zlv-Welch)
Комбинированные схемы в PKZIP и LHA
Алгоритмы STACKER и SuperStor

Сжатие с потерями

Сжатие речевой информации
Сжатие графических образов

Терминологический словарь

Как и любая научная дисциплина, сжатие информации использует свой своеобразный лексикон, который может показаться несколько странным для новичка. Во избежание дальнейших недоразумений приведем основные понятия:

сжатие информации —
это процесс сокращения количества битов, необходимых для хранения некоторого объема информации;

сжатие без потерь —
информация, восстановленная из сжатого состояния, в точности соответствует исходной (до начала сжатия);

сжатие с потерями -
информация, восстановленная после сжатия, только частично соответствует исходной (применяется при обработке изображений и звука).

щения и перевода этих символов в некоторые коды. Если этот процесс организован эффективно, то получающееся в результате закодированное сообщение занимает меньше места, чем исходное.

При просмотре обрабатываемого сообщения алгоритм сжатия реализует два почти независимых друг от друга процесса:

-поддерживает модель обрабатываемого сообщения;

- на основании модели кодирует очередной фрагмент сообщения;

Обычно весь процесс сжатия ошибочно отождествляется только с процессом кодирования, в то время как кодирование представляет собой только часть процесса сжатия, взаимодействующую с моделью данных. Используя один и тот же метод кодирования, можно совершенно по-разному строить модель сжимаемых данных. При этом результат сжатия будет отличаться для различных методов моделирования.

Обычно сжатие отождествляется только с кодированием, в то время как кодирование представляет собой только часть этого процесса, взаимодействующую с моделью данных.

Так что, если Вы когда-либо услышите заявления типа: «Кодирование Хаффмена дает оптимальные результаты, это доказано математически», — или наоборот: «Кодирование Хаффмена вообще не дает хороших результатов», — отнеситесь к этому спокойно. И то и другое, по крайней мере, — некорректные утверждения. В каждом случае общие результаты работы алгоритма сжатия зависят и от метода моделирования и от метода кодирования.

Терминологический словарь

Приведенные определения не претендуют на безукоризненность, но помогут лучше понять модель данных, используемую при кодировании Хаффмена.

Граф -
совокупность множества узлов и множества дуг, направленных от одного узла к другому.

Дерево -
граф, обладающий следующими свойствами:
а) ни в один из узлов не входит более одной дуги (то есть отсутствуют циклы);
б) только в один узел не входит ни одной дуги, он называется корнем дерева;
в) перемещаясь по дугам от корня, можно попасть в любой узел.

Лист дерева —
узел, из которого не выходит ни одной дуги.
В паре узлов дерева, соединенных между собой дугой, тот, из которого она выходит, называется родителем, другой же — ребенком.
Два узла называются братьями, если имеют одного и того же родителя.

Двоичное дерево —
дерево, у которого из всех узлов, кроме листьев, выходит по две дуги.

Дерево кодирования Хаффмена (далее Н-дерево) -
двоичное дерево, у которого каждый узел имеет вес, и вес родителя равен суммарному весу его детей.

Входной алфавит -
множество символов, входящих в сообщение.

Кодирование Хаффмена

В конце сороковых годов, на заре развития теории информации, идеи разработки новых эффективных способов кодирования информации носились в воздухе. Исследователи занимались вопросами энтропии, содержимого информации и избыточности. Интересно, что эти первоначальные работы в области сжатия информации велись до появления современного цифрового компьютера. Сегодня теория информации развивается параллельно с программированием, но в то время идея разработки алгоритмов, использующих двоичную арифметику для кодирования символов, была значительным шагом вперед.

Один из первых алгоритмов эффективного кодирования информации был предложен Д.А.Хаффменом в 1952 году. Идея алгоритма состоит в следующем: зная вероятности вхождения символов в сообщение, можно описать процедуру построения кодов переменной длины, состоящих из целого количества битов. Символам с большей вероятностью присваиваются более короткие коды. Коды Хаффмена имеют уникальный префикс, что и позволяет однозначно их декодировать, несмотря на их переменную длину.

Классический алгоритм Хаффмена на входе получает таблицу частот встречаемости символов в сообщении. Далее на основании этой таблицы строится дерево кодирования Хаффмена (Н-дерево). Алгоритм построения Н-дерева прост и элегантен.

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 двое его детей 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:

15	7	6	6	5
А	Б	В	Г	Д

На первом шаге из листьев дерева выбираются два с наименьшими весами — Г и Д. Они присоединяются к новому узлу-родителю, вес которого устанавливается в 5+6 = 11. Затем узлы Г и Д удаляются из списка свободных. Узел Г соответствует ветви 0 родителя, узел Д — ветви 1.

На следующем шаге то же происходит с узлами Б и В, так как теперь эта пара имеет самый меньший вес в дереве. Создается новый узел с весом 13, а узлы Б и В удаляются из списка свободных. После всего этого дерево кодирования выглядит так, как показано на рис. 1.

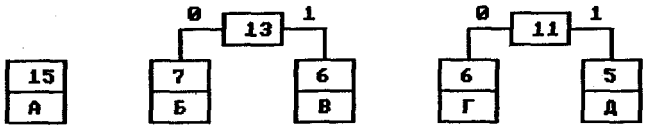

Рис. 1. Дерево кодирования Хаффмена после второго шага

На следующем шаге «наилегчайшей» парой оказываются узлы Б/В и Г/Д. Для них еще раз создается родитель, теперь уже с весом 24. Узел Б/В соответствует ветви 0 родителя, Г/Д — ветви 1.

На последнем шаге в списке свободных осталось только два узла — это А и узел (Б/В)/(Г/Д). В очередной раз создается родитель с весом 39 и бывшие свободными узлы присоединяются к разным его ветвям.

Поскольку свободным остался только один узел, то алгоритм построения дерева кодирования Хаффмена завершается. Н-дерево представлено на рис. 2.

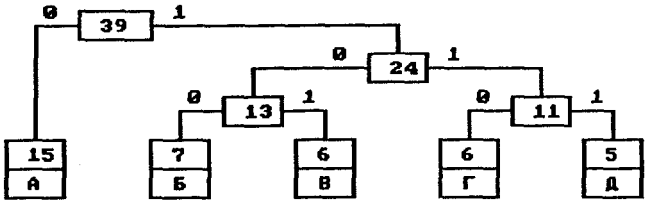

Рис. 2. Окончательное дерево кодирования Хаффмена

Чтобы определить код для каждого из символов, входящих в сообщение, мы должны пройти путь от листа дерева, соответствующего этому символу, до корня дерева, накапливая биты при перемещении по ветвям дерева. Полученная таким образом последовательность битов является кодом данного символа, записанным в обратном порядке.

Для данной таблицы символов коды Хаффмена будут выглядеть следующим образом.

А 0
Б 100
В 101
Г 110
Д 111

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

Классический алгоритм Хаффмена имеет один существенный недостаток. 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.

Адаптивное сжатие

Адаптивное сжатие позволяет не передавать модель сообщения вместе с ним самим и ограничиться одним проходом по сообщению как при кодировании, так и при декодировании.

Практически любая форма кодирования может быть конвертирована в адаптивную. В общем случае программа, реализующая адаптивное сжатие, может быть выражена в следующей форме:

```
ИнициализироватьМодель();

Пока не конец сообщения

  Символ = ВзятьСледующийСимвол();
  Закодировать (Символ);
  ОбновитьМодельСимволом (Символ);

Конец Пока
```

Декодер в адаптивной схеме работает аналогичным образом:

```
ИнициализироватьМодель();

Пока не конец сжатой информации

  Символ = РаскодироватьСледующийСимвол();
  ВыдатьСимвол (Символ);
  ОбновитьМодельСимволом (Символ);

Конец Пока
```

Схема адаптивного кодирования/декодирования работает благодаря тому, что и при кодировании, и при декодировании используются одни и те же процедуры «ИнициализироватьМодель» и «ОбновитьМодельСимволом». И компрессор, и де-

компрессор начинают с «пустой» модели (не содержащей информации о сообщении) и с каждым просмотренным символом обновляют ее одинаковым образом.

Адаптивное кодирование Хаффмена

Следующим шагом в развитии алгоритма Хаффмена стала его адаптивная версия. Ей в основном и посвящена эта статья.

В создании алгоритма адаптивного кодирования Хаффмена наибольшие сложности возникают при разработке процедуры ОбновитьМодельСимволом(); можно было бы просто вставить внутрь этой процедуры полное построение дерева кодирования Хаффмена. В результате мы получили бы самый медленный в мире алгоритм сжатия, так как построение Н-дерева — это слишком большая работа и производить ее при обработке каждого символа неразумно. К счастью, существует способ модифицировать уже существующее Н-дерево так, чтобы отобразить обработку нового символа.

Упорядоченное дерево

Будем говорить, что дерево обладает свойством упорядоченности, если его узлы могут быть перечислены в порядке возрастания веса и в этом перечислении каждый узел находится рядом со своим братом. Пример упорядоченного дерева приведен на рис. 3. Здесь W — вес узла, N — порядковый номер в списке узлов.

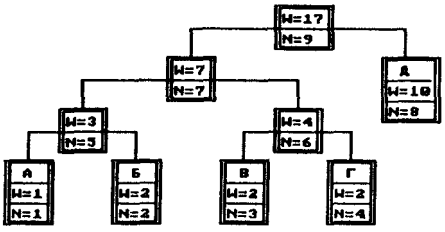

Рис. 3. Свойство упорядоченности дерева Хаффмена

Примем без доказательства утверждение о том, что двоичное дерево является деревом кодирования Хаффмена тогда и только тогда, когда оно удовлетворяет свойству упорядоченности.

Сохранение свойства упорядоченности в процессе обновления дерева позволяет нам быть уверенными в том, что двоичное дерево, с которым мы работаем, — это Н-дерево и до, и после обновления веса у листьев дерева.

Обновление дерева при считывании очередного символа сообщения состоит из двух операций.

Первая — увеличение веса узлов дерева — представлена на рис. 4. Вначале увеличиваем вес листа, соответствующего считанному символу, на единицу. Затем увеличиваем вес родителя, чтобы привести его в соответствие с новыми значениями веса у детей. Этот процесс продолжается до тех пор, пока мы не доберемся до корня дерева. Среднее число операций увеличения веса равно среднему количеству битов, необходимых для того, чтобы закодировать символ.

Вторая операция — перестановка узлов дерева — требуется тогда, когда увеличение веса узла приводит

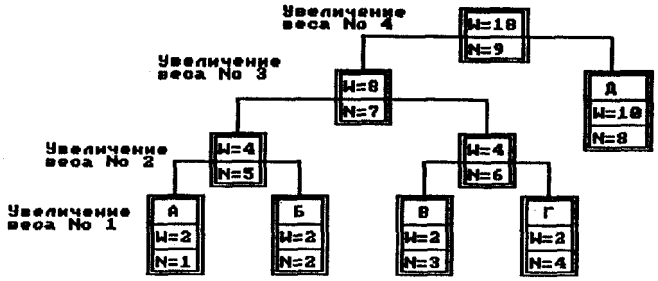

Рис. 4. Процедура обновления дерева кодирования при увеличении веса листа А

к нарушению свойства упорядоченности, то есть тогда, когда увеличенный вес узла стал больше, чем вес следующего по порядку узла (рис. 5). Если и дальше продолжать обрабатывать увеличение веса, двигаясь к корню дерева, то наше дерево перестанет быть деревом Хаффмена.

Рис. 5. При увеличении веса листа А нарушается свойство упорядоченности

Чтобы сохранить упорядоченность дерева кодирования, алгоритм работает следующим образом. Пусть новый увеличенный вес узла равен W+1. Тогда начинаем двигаться по списку в сторону увеличения веса, пока не найдем последний узел с весом W. Переставим текущий и найденный узлы между собой в списке (рис. 6), восстанавливая таким образом порядок в дереве. (При этом родители каждого из узлов тоже изменятся.) На этом операция перестановки заканчивается.

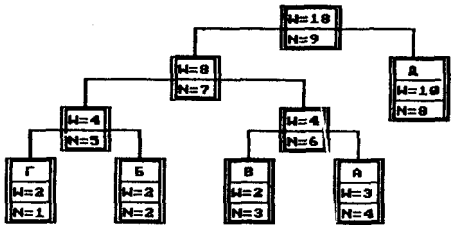

Рис. 6. Дерево кодирования после первой перестановки узлов (Г и А)

После перестановки операция увеличения веса узлов продолжается дальше. Следующий узел, вес которого будет увеличен алгоритмом, — это новый родитель узла, увеличение веса которого вызвало перестановку.

Предположим, что символ А встретился в сообщении еще два раза подряд. Дерево кодирования после двукратного вызова процедуры обновления показано на рис. 7.

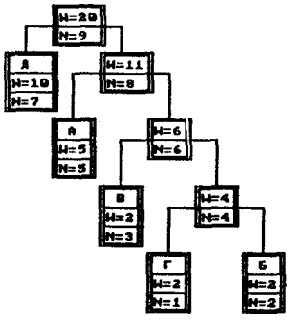

Рис. 7. Дерево после завершения процедуры обновления

Обратите внимание, как обновление дерева кодирования отражается на длине кодов Хаффмена для входящих в данное сообщение символов.

В целом алгоритм обновления дерева может быть записан следующим образом:

```
ОбновитьМодельСимволом (Символ)
{
  ТекущийУзел = ЛистСоответствующий (Символ) Всегда
      УвеличитьВес (ТекущийУзел) Если
      ТекущийУзел = КореньДерева
          Выход; Если   Вес
      (ТекущийУзел) >
              Вес (СледующийЗа (ТекущийУзел))
          Перестановка ();
    ТекущийУзел = Родитель (ТекущийУзел);
Конец Всегда }
```

Проблемы адаптивного кодирования

Инициализация

Хорошо было бы, чтобы кодер не тратил зря кодовое пространство на символы, которые не встречаются в сообщении.

Если речь идет о классическом алгоритме Хаффмена, то те символы, которые не встречаются в сообщении, уже известны до начала кодирования, так как известна таблица частот и символы, у которых частота встречаемости равна 0. В адаптивной версии алгоритма мы не можем знать заранее, какие символы появятся в сообщении. Можно проинициализировать дерево Хаффмена так, чтобы оно имело все 256 символов алфавита (для 8-битовых кодов) с частотой, равной 1. В начале кодирования каждый код будет иметь длину 8 битов. По мере адаптации модели наиболее часто встречающиеся символы будут кодироваться все меньшим и меньшим количеством битов. Такой подход работоспособен, но он значительно снижает степень сжатия, особенно на коротких сообщениях.

Лучше начинать моделирование с пустого дерева и добавлять в него символы только по мере их появления в сжимаемом сообщении. Но это приводит к очевидному противоречию: когда символ появляется в сообщении в первый раз, он не может быть закодирован, так как его еще нет в дереве кодирования.

Чтобы разрешить это противоречие, введем специальный ESCAPE код, который для декодера будет означать, что следующий символ закодирован вне контекста модели сообщения. Например, его можно передать в поток сжатой информации как есть, не кодируя вообще. Метод «ЗакодироватьСимвол» в алгоритме адаптивного кодирования Хаффмена можно записать следующим образом.

```
ЗакодироватьСимвол (Символ)
{ Если СимволУжеЕстьВТаблице (Символ)
ВыдатьКодХаффменаДляСимвола (Символ)
  Иначе
{
ВыдатьКодХаффменаДляСимвола (ESCAPE)
  ВыдатьСимвол (Символ) } }
```

Использование специального символа ESCAPE подразумевает определенную инициализацию дерева до начала кодирования и декодирования: в него помещаются 2 специальных символа: ESCAPE и EOF (конец файла), с весом, равным 1 (рис. 8). Поскольку процесс обновления дерева не коснется их веса, то по ходу кодирования они будут перемещаться на самые удаленные ветви дерева и иметь самые длинные коды.

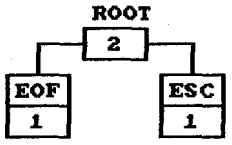

Рис. 8. Дерево кодирования после инициализации

Переполнение

В процессе работы алгоритма сжатия вес узлов в дереве кодирования Хаффмена неуклонно растет. Первая проблема возникает тогда, когда вес корня дерева начинает превосходить вместимость ячейки, в которой он хранится. Как правило, это 16-битовое значение и, следовательно, не может быть больше, чем 65535. Вторая проблема, заслуживающая еще большего внимания, может возникнуть значительно раньше, когда размер самого длинного кода Хаффмена превосходит вместимость ячейки, которая используется для того, чтобы передать его в выходной поток. Декодеру все равно, какой длины код он декодирует, поскольку он движется сверху вниз по дереву кодирования, выбирая из входного потока по одному биту. Кодер же должен начинать от листа дерева и двигаться вверх к корню, собирая биты, которые нужно передать. Обычно это происходит с переменной типа «целое», и, когда длина кода Хаффмена превосходит размер типа «целое» в битах, наступает переполнение.

Можно доказать, что максимальную длину код Хаффмена для сообщений с одним и тем же входным алфавитом будет иметь, если частоты символов образует последовательность Фибоначчи (рис. 9).

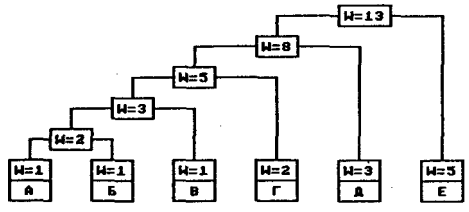

Рис. 9. Дерево Хаффмена на числах Фибоначчи

Функция Фибоначчи определяется следующим образом.

```
Int Fib (int n)
{ If (n <= 1)
      return 1;
  else
      return (fib (n — 1) + fib (n — 2)); }
```

Если вес корневого узла в дереве Хаффмена равен Fib (i), то наиболее длинный код, возможный для этого дерева, имеет длину i-1.

Это означает, что если «целые», используемые для представления весов в дереве Хаффмена, имеют длину 16 битов, то вес корня, равный 4181 (Fib(18)), может привести к переполнению. (Вообще говоря, сообщение с частотами символов, равными числам Фибоначчи до Fib (18), — это отличный способ протестировать работу программы сжатия по Хаффмену.)

Масштабирование весов узлов дерева Хаффмена

Принимая во внимание сказанное выше, алгоритм обновления дерева Хаффмена должен быть изменен следующим образом: при увеличении веса нужно проверять его на достижение допустимого максимума. Если мы достигли максимума, то необходимо «масштабировать» вес, обычно разделив вес листьев на целое число, например, 2, а потом пересчитав вес всех остальных узлов.

Однако при делении веса пополам возникает проблема, связанная с тем, что после выполнения этой операции дерево может изменить свою форму. Объясняется это тем, что мы делим целые числа и при делении отбрасываем дробную часть (рис. 10-12).

Правильно организованное дерево Хаффмена после масштабирования может иметь форму, значительно отличающуюся от исходной. Это происходит потому, что масштабирование приводит к потере точности нашей статистики. Но со сбором новой статистики последствия этих «ошибок» практически сходят на нет. Масштабирование веса — довольно дорогостоящая операция, так как она приводит к необходимости заново строить все дерево кодирования. Но, так как необходимость в ней возникает относительно редко, то с этим можно смириться.

Выигрыш от масштабирования

Масштабирование веса узлов дерева через определенные интервалы дает неожиданный результат. Несмотря на то, что при масштабировании происходит потеря точности статистики, тесты показывают, что оно приводит к лучшим показателям сжатия, чем если бы масштабирование откладывалось. Это можно объяснить тем, что текущие символы сжимаемого по-

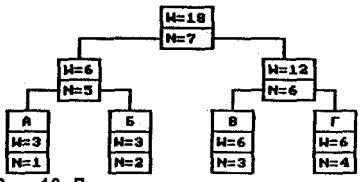

Рис. 10. Дерево кодирования перед масштабированием весов

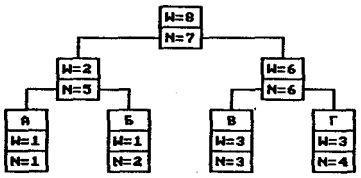

Рис. 11. Дерево кодирования после деления веса каждого листа на 2. Свойство упорядоченности

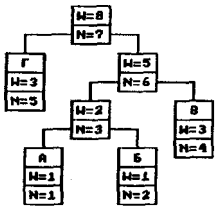

Рис. 12. Правильная форма дерева после масштабирования

тока больше «похожи» на своих близких предшественников, чем на тех, которые встречались намного раньше. Масштабирование приводит к уменьшению влияния «давних» символов на статистику и к увеличению влияния на нее «недавних» символов. Это очень сложно измерить количественно, но, в принципе, масштабирование оказывает положительное влияние на степень сжатия информации. Эксперименты с масштабированием в различных точках процесса сжатия показывают, что степень сжатия сильно зависит от момента масштабирования веса, но не существует правила выбора оптимального момента масштабирования для программы, ориентированной на сжатие любых типов информации.

Заключение

С тех пор, как Д.А.Хаффмен опубликовал в 1952 году свою работу «Метод построения кодов с минимальной избыточностью», его алгоритм кодирования стал базой для огромного количества дальнейших исследований в этой области. По сей день в компьютерных журналах можно найти большое количество публикаций, посвященных как различным реализациям алгоритма Хаффмена, так и поискам его лучшего применения. Кодирование Хаффмена используется в коммерческих программах сжатия (например, в PKZIP и LHA), встроено в некоторые телефаксы и даже используется в алгоритме JPEG сжатия графи-ческих изображений с потерями.

Приведенная далее программа, реализующая алгоритм адаптивного кодирования Хаффмена, написана на языке С и может быть собрана компиляторами фирм Borland, Microsoft и Zortech. Она не была сколько-нибудь оптимизирована по скорости и дает не сравнимые с PKZIP, например, результаты по сжатию (что абсолютно закономерно, так как PKZIP использует совершенно другой способ моделирования данных для кодирования Хаффмена). Автор надеется, что приведенные комментарии, а также возможность «поиграть» с исходным текстом в отладчике или

профайлере помогут Вам получить достаточно полное представление о том, как работает схема сжатия Хаффмена.■

В следующем номере

Следующая статья этой серии будет посвящена арифметическому кодированию. Этот алгоритм является таким же краеугольным камнем в сжатии информации, как и алгоритм Хаффмена, в отличие от которого арифметическое кодирование позволяет превратить один байт входного потока в менее, чем один бит кода (в некоторых случаях). О том, как это делается, читайте в следующем номере «Монитора».

Рекомендуемая литература

1. **Huffman D.A. A Method for the construction of minimum-redundancy codes// IRE.- September 1952,-Vol. 40.- № 9. Русский перевод: Д.Л. Хаффмен. Метод построения кодов с минимальной избыточностью// Кибернетический сборник.-1961.- Вып. 3.**
2. **Storer J.A. Data compression// Computer Science Press.-1988.**
3. **Nelson M. The Data Compression Book.-MAT Publishing.-1991.**
4. **Gallager R.G. Variations on a theme by Huffman// IEEE Transactions on Information Theory,-1978.- Vol. 24,-№6.**
5. **Knuth D.E. Dynamic Huffman Coding// Journal of Algorithms.-1985,-№ 6.**
6. **Gormak G.V. and Horspool R.N. Algorithms for adaptive Huffman codes// Information Processing Letters.-1984.-№18.**
7. **Jacobsson M. Huffman coding in Bit-Vector Compression// Information Processing Letters.-1978.- Vol. 7.-№6.**
8. **Johnsen O. On the redundancy of binary Huffman codes// IEEE Transactions on Information Theory.-1980.-Vol. 26.-№2.**
9. **D.R. McIntyre and M.A. Pechura. Data Compression Using Static Huffman Code-Decode Tables// Journal of the ACM-1985- Vol. 28.-№ 6.**
10. **Д. Арапов. Пишем упаковщик// Монитор.-1993.-№1.**