Кодирование Хаффмана, журнал Монитор, номер 7-8, 1993 год

Технологии Программирования

Хаффман - 1954 год. Идея алгоритма: зная вероятность вхождения символов в сообщение, можно описать процедуру построения кодов переменной длинны состоящих из целого количества битов. Символам с большей вероятностью присваиваются более короткие коды. Коды Хаффмана имеют уникальный префикс, что и позволяет однозначно их декодировать, несмотря на их переменную длину. Динамический алгоритм Хаффмана на входе получает таблицу частот встречаемости символов в сообщении. Далее на основании этой таблицы строится дерево кодирования Хаффмана (Н-дерево). Алгоритм:

1. Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который может быть равен либо вероятности, либо количеству вхождений символа в ожидаемое сообщение.
2. Выбираются 2 свободных узла дерева с наименьшими весами.
3. Создается 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 их родителя ставится в соответствие бит 1, другой - бит 0.
6. Далее пункты повторяются, начиная со второго, до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева.

Кодирование Хаффмана

Допустим у нас есть таблица частот:

15	7	6	6	5
А	Б	В	Г	Д

На первом шаге из листьев дерева выбираются 2 с наименьшими весами - Г и Д. Они присоединяются к новому узлу-родителю. вес которого устанавливаются в 5+6=11. Затем узлы Г и Д удаляются из списка свободных. Узел Г ссоответствует 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.

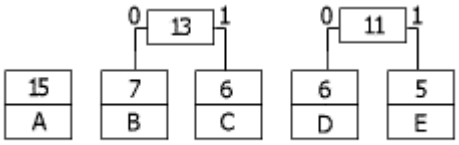

Fig. 1 Haffman Coding Tree after second step

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 остается только 2 узла - это А и узел (Б/В)/(Г/Д). В очередной раз создается родитель с весом 39 и бывшие свободные узлы присоединяются к разным его ветвям.
Поскольку свободным остается только один узел, то алгоритм построения дерева кодирования Хаффмана завершается. Н-дерево представлено на рис. 2.

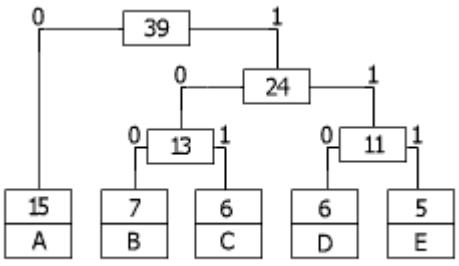

Fig. 2 Final Haffman Coding Tree

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, записанная в

обратном порядке.
Для данной таблицы символов коды Хаффмана 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 чтении их из потока. Кроме того, наиболее частый символ сообщения А закодирован наименьшим количеством битов, а наиболее редкий символ Д - наибольшим.

Классический алгоритм Хаффмана имеет один существенный недостаток. Для восстановления содержимого 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 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 = РаскодироватьСледующий();
  ВыдатьСимвол(Символ);
  ОбновитьМодельСимволом(Символ);
Конец пока;
```

Схема адаптивного кодирования/ декодирования работает благодаря тому, что при кодировании, и при декодировании используются одни и те же процедуры "ИнициализироватьМодель" и "ОбновитьМодельСимволом" И компрессор, и декомпрессор начинают с "пустой" модели (не содержащей информации о сообщении) и с каждым просмотренным символом обновляют ее одинаковым образом.

Адаптивное Кодирование Хаффмана

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

В создании алгоритма адаптивного кодирования Хаффмана наибольшие сложности возникают при разработке процедуры **ОбновитьМодельСимволом();** можно было бы просто вставить внутрь этой процедуры полное построение дерева Хаффмана. 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.

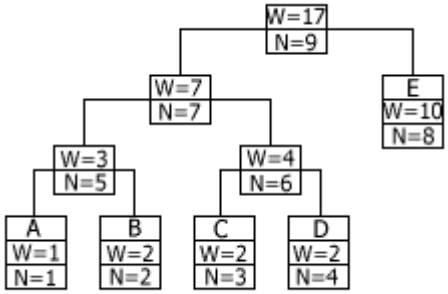

Fig. 3 Order Property of H-Tree

Примем без доказательства утверждение о том, что двоичное дерево является деревом кодирования Хаффмана тогда и только тогда, когда оно удовлетворяет свойству упорядоченности. Сохранение свойства упорядоченности в процессе обновления дерева позволяет нам быть уверенным в том, что двоичное дерево, с которым мы работаем, - это H-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

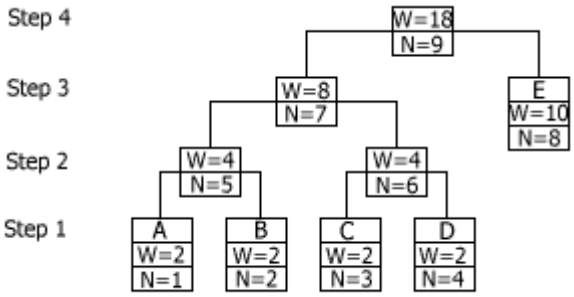

Fig. 4 H-Tree Update Procedure

к нарушению свойства упорядоченности, то есть тогда, когда увеличенный вес узла стал больше, чем вес следующего по порядку узла (рис. 5). Если и дальше

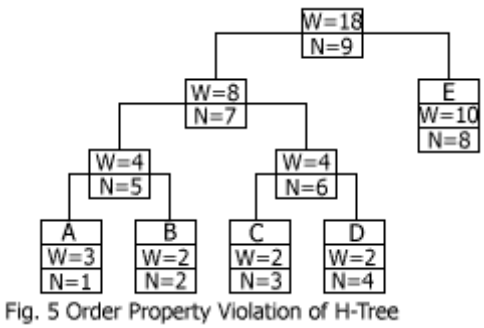

Fig. 5 Order Property Violation of H-Tree

продолжать обрабатывать увеличение веса, двигаясь к корню дерева, то наше дерево перестанет быть деревом Хаффмана.

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.

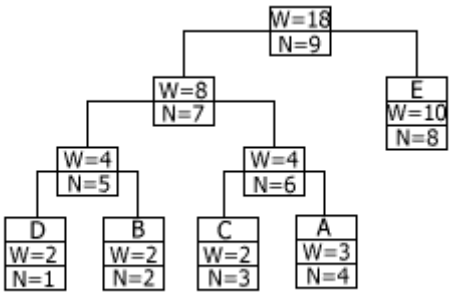

Fig. 6 H-Tree after 1st A-D exchange

(При этом родители каждого из узлов тоже изменяются.) На этом операция перестановки заканчивается.

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 A встретился в сообщении еще два раза подряд. Дерево кодирования после двукратного вызова процедуры обновления показано на рис. 7.

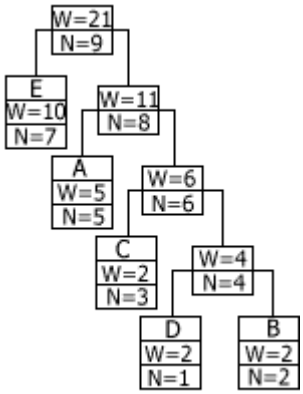

Fig. 7 H-Tree after Updating

Обратите внимание, как обновление дерева кодирования отражается на длине кодов Хаффмана 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

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

Если речь идет о классическом алгоритме Хаффмана, 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 Хаффмана так, чтобы оно имело все 256 символов алфавита (для 8-и битовых кодов) с частотой, равной 1. В начале кодирования каждый код будет иметь длинну 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 первый раз, он не может быть закодирован, так как его еще нет в дереве кодирования.
Чтобы разрешить это противоречие, введем специальный ESCAPE код, который 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 Хаффмана можно записать следующим образом.

```
ЗакодироватьСимвол(Символ)
{
 Если СимволУжеЕстьВТаблице(Символ)
   ВыдатьКодХаффманаДля(Символ);
 Иначе
 {
   ВыдатьКодХаффманаДля(ESC);
   ВыдатьСимвол(Символ);
 }
}
```

Использование специального символа ESC подразумевает определенную инициализацию дерева до начала кодирования и декодирования: в него помещаются 2 специальных символа: ESC и EOF (конец файла), с весом, равным 1 (рис. 8).

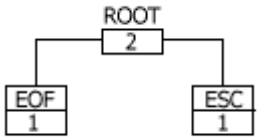

Fig. 8 H-Tree after Init()

Поскольку процесс обновления не коснется их веса, то по коду кодирования они будут перемещаться на самые удаленные ветви дерева и иметь самые длинные коды.

Заключение

С тех пор, как Д.А.Хаффман 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 Хаффмана, так и поискам его лучшего применения. Кодирование Хаффмана используется в коммерческих программах сжатия (например в PKZIP и LHA), встроено в некоторые телефаксы и даже используется в алгоритме JPEG сжатия графических изображений с потерями.