哈夫曼编码的另一种实现算法

王群芳

(安徽教育学院 计算机系,安徽 合肥 230061)

[**摘要**] 传统哈夫曼编码借助树形结构构造,算法实现时使用链表或静态链表结构,空间的每个结点内有左、右子树、双亲指针。本文给出了哈夫曼编码的另一种实现算法,该算法抛开树结构,用一个数组模拟二叉树的创建过程并得到符号的深度,然后根据这一信息为每个符号分配编码。对于大型文件来说,整个编码、译码过程中需要的空间比传统哈夫曼编码要少得多。

[**关键词**] 二叉树;哈夫曼树;哈夫曼编码;哈夫曼算法

[**中图分类号**] TP314 [**文献标识码**] A [**文章编号**] 1001-5116(2006)06-0036-03

提起 Huffman 这个名字,我们会联想到二叉树和二进制编码。哈夫曼编码是哈夫曼树的一个应用,是一种常见的无失真压缩方案。

首先介绍什么是哈夫曼树。哈夫曼树又称最优二叉树,是一种带权路径长度最短的二叉树。所谓树的带权路径长度,就是树中所有的叶结点的权值乘上其到根结点的路径长度。树的带权路径长度记为 WPL =($W_1 * L_1 + W_2 * L_2 + W_3 * L_3 + \ldots + Wn * Ln$),N 个权值 W_i(i =1,2,...n)构成一棵有 N 个叶结点的二叉树,相应的叶结点的路径长度为 L_i(i =1,2,...n)。可以证明哈夫曼树的 WPL 是最小的。

怎样构造一棵哈夫曼树呢? 最具有一般规律的构造方法就是哈夫曼算法。一般的数据结构的书中都可以找到其描述:

1) 对给定的 n 个权值 $\{W_1, W_2, W_3, \ldots, W_i, \ldots, W_n\}$ 构成 n 棵二叉树的初始集合 $F = \{T_1, T_2, T_3, \ldots, T_i, \ldots, T_n\}$,其中每棵二叉树 T_i 中只有一个权值为 W_i 的根结点,它的左右子树均为空。

2) 在 F 中选取两棵根结点权值最小的树作为新构造的二叉树的左右子树,新二叉树的根结点的权值为其左右子树的根结点的权值之和。

3) 从 F 中删除这两棵树,并把这棵新的二叉树加入到集合 F 中。

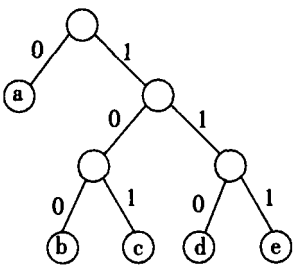
4) 重复 2、3 两步,直到集合 F 中只有一棵二叉树为止。这棵树便是哈夫曼树。

哈夫曼编码是 Huffman 在上世纪五十年代初提出的,根据字符出现的概率来构造平均长度最短的编码。它是一种变长的编码。在编码中,若各码字长度严格按照码字所对应符号出现概率的大小的逆序排列,则编码的平均长度是最小的。(注:码字即为符号经哈夫曼编码后得到的编码,其长度是因符号出现的概率而不同,所以说哈夫曼编码是变长的编码。)构造好哈夫曼树后,就可根据哈夫曼树进行编码。编码的规则是从根结点到叶结点(包含原信息)的路径,向左孩子前进编码为 0,向右孩子前进编码为 1,当然也可以反过来规定。对下面这串出现了五种字符的信息(40 个字符长)进行编码:

cabcedeacacdeddaaabaababaaabbacdebaceada

五种字符的出现次数分别 a-16,b-7,c-6,d-6,e-5。

构造出的哈夫曼树如下:



于是我们得到了此信息的编码表:

a-0 b-100 c-101 d-110 e-111

可以将例子中的信息编码为:

cabcedeacacdeddaaabaababaaabbacdebaceada

101 0 100 101 111 110 111 0 101 0 101 ……

码长共 1 * 16 + 3 * (7 + 6 + 6 + 5) = 88 位。如果用 ASCII 码表示上述信息需要 8 * 40 = 320 位,哈夫曼编码确实实现了数据压缩。

只要使用同一棵哈夫曼树,就可把编码还原成原来那组字符。显然哈夫曼编码是前缀编码,即任一个字符的编码都不是另一个字符的编码的前缀,否则,编码就不能进行翻译。

[**收稿日期**] 2006-07-10

[**作者简介**] 王群芳(1973-),女,安徽教育学院计算机系讲师。

抛开树结构，采用另一种构造哈夫曼编码的方法。该方法的基本思路是：并非只有使用二叉树建立的前缀编码才是哈夫曼编码，只要符合(1)是前缀编码(2)某一字符编码长度和使用二叉树建立的该字符的编码长度相同这两个条件的编码都可以叫做哈夫曼编码。考虑对下面六个符号的编码：

符号	出现次数	传统哈夫曼编码	另一种哈夫曼编码
符号 1	10	000	000
符号 2	11	001	001
符号 3	12	100	010
符号 4	13	101	011
符号 5	22	01	10
符号 6	23	11	11

注意到该编码的独特之处了吗？你无法使用二叉树来建立这组编码，但这组编码确实能起到和哈夫曼编码相同的作用。而且，该编码具有一个明显的特点：当我们把要编码的符号按照其频率从小到大排列时，如果把编码本身作为单词的话，也呈现出从小到大的字典顺序。

构造该编码的方法是：

1) 统计每个要编码符号的频率。

2) 根据这些频率信息求出该符号在传统哈夫曼编码树中的深度(也就是表示该符号所需要的位数-编码长度)。因为我们关心的仅仅是该符号在树中的深度，我们完全没有必要构造二叉树，仅用一个数组就可以模拟二叉树的创建过程并得到符号的深度。

3) 分别统计从最大编码长度 maxlength 到 1 的每个长度对应了多少个符号。根据这一信息从 maxlength 个 0 开始以递增顺序为每个符号分配编码。例如，编码长度为 5 的符号有 4 个，长度为 3 的有 1 个，长度为 2 的有 3 个，则分配的编码依次为：00000 00001 00010 00011 001 01 10 11。

4) 编码输出压缩信息，并保存按照频率顺序排列的符号表，然后保存每组同样长度编码中的最前一个编码以及该组中的编码个数。

现在完全可以不依赖任何树结构进行译码了。而且在整个编码、译码过程中需要的空间比传统哈夫曼编码少得多。

下面给出它的实现算法：

```
void huffman_codes(int n, unsigned int *weights)
    {  //将二叉树存放在连续空间tree 里，空间的
       //每个结点类型都和结点权值的数据类型
       //相同，空间大小为 2n，tree[0]不用。
       //初始tree[1..n ]是每个元素的权值。生
       //成 Huffman 后，tree[1..2n-1]中存储的
       //是双亲结点索引

    if (n<=1 || weights == NULL)
          {printf("参数非法");return;}

    //权值为 0 的字符不参与编码，nonzero_num 记
    //录实际参与编码的字符数量
    int nonzero_num = 0;
    int *tree = new unsigned int[2*n];
    // 0 号单元不用
    if (tree == NULL)
    {printf(("内存不足");return;}
    // 将所有元素的权值复制到tree[1..n]
    for(int i = 1;i <= n;i++)
    {
      tree[i] = weights[i - 1];
      if (weights[i - 1])
        nonzero_num++;
    }

    //flags 数组记录每个叶子结点或子树是否已
    //连入了 Huffman 树
    bool *flags = new bool[2*n];
    if (flags == NULL)
    {printf("内存不足");return;}
    memset(flags, false, sizeof(bool)*2*n);

    // 建 Huffman 树
    int s1, s2;
    for(int i = n +1;i <n +nonzero_num;i +
+)
    {
      select(tree, flags, i - 1, s1, s2);
```

```
    tree[i] = tree[s1] + tree[s2];
    tree[s1] = tree[s2] = i;
    flags[s1] = flags[s2] = true;
  }

  // 从根出发,求每个编码的码长
  // 根结点码长为0
  tree[num + nonzero_num - 1] = 0;
  for (int i = num + nonzero_num - 2; i >= 1; i--)
  // 结点码长等于双亲结点码长加1
    tree[i] = tree[tree[i]] + 1;

  // 由码长得到huffman 编码
  generate_codes(tree);
  delete tree, flags;
  }

  void select(unsigned int *tree, bool *flags,
int n, int &s1, int &s2)
  {
    s1 = s2 = 0;
    for(int i = 1; i <= n; i++)
      if (tree[i] && !flags[i])
      {
        if (tree[i] < tree[s1])
          {s2 = s1; s1 = i;}
        else if (tree[i] < tree[s2])
          {s2 = i;}
      }
  }
```


[参 考 文 献]

[1] 严蔚敏、吴伟民,数据结构[M](C 语言版),清华大学出版社,2002

[2] 钱能,C++程序设计教程[M],清华大学出版社,2005

[3] 刘祎玮、汪晓平,C 语言高级实例解析[M],清华大学出版社,2004

Another Algorithm of Huffman Coding

WANG Qunfang

(*Department of Computer Science, Anhui Institute of Education, Hefei 230061, China*)

Abstract: Traditional Huffman coding is structured in the way of tree pattern, using the chain or static chain structure in algorithm, and every node in space has left, right tree and parents pointers. This paper presents another algorithm of Huffman coding which removes tree pattern structure, using an array to simulate the creation process of binary tree so as to obtain the depth of the symbol, and then each symbol is assigned a code according to this information. For the full-length text, the space needed in the process of coding and decoding will much less than in the traditional Huffman coding.

Key Words: binary tree; Huffman tree; Huffman Coding; Huffman algorithm