第 27卷 第 1期 西华师范大学学报(自然科学版) 2006年 3月
Vol.27 No.1 Journal of China West Normal University (Natural Sciences) Mar.2006

文章编号:1001-8220(2006)01-0074-05

哈夫曼编码的一种基于树型模式匹配的改进型算法

刘晓锋,吴亚娟
(西华师范大学计算机学院,四川 南充 637002)

摘 要:一般情况下,哈夫曼编码所采用的存储结构及构树方法,不仅影响编码效率,而且也没充分利用存储空间.本文改顺序存储为链式存储,对叶结点和非叶结点采用不同的存储结构来降低空间复杂度.在编码时,充分利用短码字且基于树型模式匹配进行编码,提高了编码性能和传输效率.

关键词:哈夫曼树;哈夫曼编码;树型模式匹配;算法
中图分类号:TP301.6 **文献标识码**:A

1 引 言

与计算机相关的任何一个应用领域几乎都涉及到编码的概念.编码的主要目的是使数据便于计算机处理及高效率传输.哈夫曼编码是由哈夫曼于 1952年最早提出的一种基于统计模型的非等长的编码方案,它主要用在数据压缩和加密等应用场合.尤其在知道信源中各字符出现的频率的情况下,它不失为一种优秀的编码方案,如 MPEG-2的视频电视信号的压缩处理.

2 哈夫曼编码的传统算法

哈夫曼编码本身并不复杂,是一种基于贪心算法自下而上构造最优前缀码的过程.其基本思想是根据信源中待编码字符出现的频率(或概率)的大小构造一棵最优二叉树.构造一棵最优二叉树是基于如下定理.

定理 1 在变长编码中,对以较大概率出现的字符赋于较短的编码,而对较小概率出现的字符赋于较长的编码,即按照各字符出现的概率由小到大分别赋于由长到短的编码,则编码结果平均码长小于其他任何编码方式所得到的平均码长.

证明 (采用反证法)设 $f(a_i)$表示信源 S中某字符 a_i出现的频率,L_i为其编码长度,则平均码长为 $\bar{L}=\sum_i L_i * f(a_i)$.按定理中描述,对任意两个不同字符 a_i和 a_j,若有 $L_i>L_j$且 $f(a_i)<f(a_j)$成立,则平均码长 $\bar{L}=\sum_i L_i * f(a_i)$最小(如图 1(a)).现作出如下反面假设:$L_i>L_j$且 $f(a_i)>f(a_j)$时(如图 1(b)),平均码长为 $\bar{L}'$,此时有 $\bar{L}'<\bar{L}$如图 1所示.

现对两平均码长作差,有

$$\bar{L}'-\bar{L}=f(a_i) * L_i'+f(a_j) * L_j'-f(a_i) * L_i-f(a_j) * L_j.$$

由图 1(b)可得,$L_i'=L_j$, $L_j'=L_i$,所以有

$$\bar{L}'-\bar{L}=f(a_i) * L_j+f(a_j) * L_i-f(a_i) * L_i-f(a_j) * L_j=$$
$$f(a_i)(L_j-L_i)+f(a_j)(L_i-L_j)=(L_i-L_j)(f(a_j)-f(a_i))>0.$$

所以有 $\bar{L}'>\bar{L}$这与假设的结论 $\bar{L}'<\bar{L}$矛盾.所以假设不成立.

根据定理 1,整个编码过程就是,首先确定信源中所有编码信号出现的概率形成一个码表,对经常出现

收稿日期:2005-11-10
基金项目:西华师范大学校级科研基金资助项目(04A021)
作者简介:刘晓锋(1972-),男,重庆石柱人,西华师范大学计算机学院硕士研究生,研究方向为并行算法理论与研究、网格计算及基于网络的计算机应用.

的大概率信号分配较短的比特,对不经常出现的小概率信号分配较长的比特,使得整个码流的平均码长趋于最短.在实际操作中,是根据计算出来的码表分两步进行,第一步根据码表构造一棵外部带权路径最短的二叉树,第二步对二叉树的每个分枝分别用 0和 1标注.这样从树根到叶结点的路径上字符序列就是相应叶结点的哈夫曼编码.这样得到的哈夫曼编码并非惟一,主要有两个方面的原因.其一,每次合并两个概率最小的信源符号,分别用 0和 1加以标注,这个标注是任意的.其二,当合并两个概率最小的信源之后,其概率和其它信源概率相同时,按编码方法上说,进一步合并是在这些信源概率相等的符号之间任意选择两个进行合并,并没有规定谁先谁后.这样得到的码字和相应的码长 L_i 都不一样,但平均码长 $\bar{L}$ 和编码效率一样.如有离散无记忆信源:$\begin{pmatrix} S \\ p(s_i) \end{pmatrix} = \begin{pmatrix} s_1 & s_2 & s_3 & s_4 & s_5 \\ 4/10 & 2/10 & 2/10 & 1/10 & 1/10 \end{pmatrix}$,根据前述可得到如下两种不同形式的哈夫曼编码(如图 2所示),(a)和(b)两种不同形式的哈夫曼编码具有相同的平均码长,都为 2.2,编码效率也相同,但信源符号的编码不一样.一般情况下,我们认为这两种形式没有什么差别.事实上这两者之间存在较大的差异,为找出这两者的差异而引入码字长度 L的方差(σ^2).

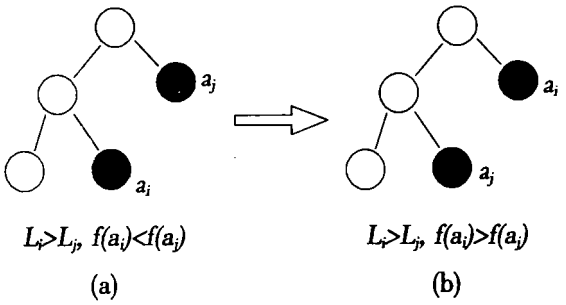

图 1　哈夫曼树的变换

Fig.1　Transfer of Huffman tree

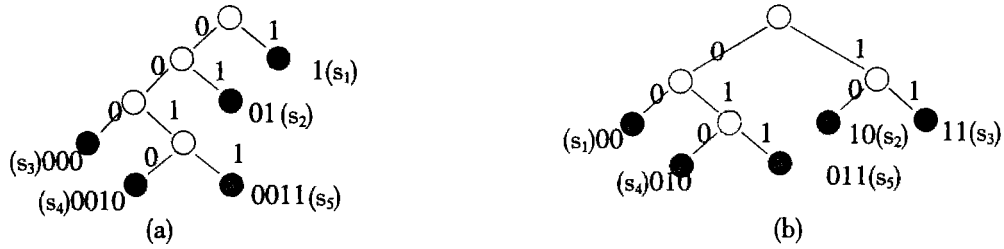

图 2　同一组数据的两种不同形式的哈夫曼树

Fig.2　Two different Huffman trees based on one group data

定义 1　码字长度 L的方差 $\sigma^2 = E[(L_i - \bar{L})^2] = \sum_i p(s_i)(L_i - \bar{L})^2$.其中 $p(s_i)$是信源符号 s_i 出现的概率,L_i 是 S_i 的码长,$\bar{L}$ 表示平均码长.

根据定义 1有 $\sigma_a^2 = 1.36$,而 $\sigma_b^2 = 0.16$.可见第二种编码的方差要小,说明该编码码长偏离平均码长的程度要小,所以我们认为(b)比(a)要好.其主要原因在于(b)中的编码充分利用了短码字.

3　树型模式匹配

在计算机科学与人工智能的许多领域中,模式匹配是最基本的运算之一.在早期的应用研究中,其主要内容集中在两个串之间的“完全匹配”和“一致匹配”.在许多更复杂的问题中,具有线性结构的串是不能刻画匹配目标和匹配模式,此时需要借助更复杂的数据结构——树.在这样的结构中,为了和串中的完全匹配或一致匹配相区别,我们引入“结构相容”匹配的概念.一棵目标子树 T_j 与一个模式 P_i“结构相容”,通常是指 T_j 具有与 P_i 中非叶子节点部分完全相同的结构,且 P_i 中的叶子节点与 T_j 中相应部分(可能是叶子节点,也可能是子树)“相容”.例如,有目标子树 T_1: $x_1 + x_2$, T_2: $y + y$, T_3: $a + b \times c$和模式 P_1: $x + y$, P_2: $x + x$.目标子树 T_1, T_2 和 T_3 都和模式树 P_1 结构相容,但只有 T_2 才能和 P_2 结构相容.

上面是关于“结构相容”的一般定义,在不同的应用环境中,“相容”的具体定义具有不同的形式.由上面的定义可知,目标子树 T_j 具有与模式树 P_i 中非叶子节点部分完全相同的结构,但在哈夫曼树中,不需要 T_j

具有与 P_i 中非叶子节点部分完全相同.因此,为了和上面"结构相容"相区别,我们引入"结构基本相容"的概念.一棵目标子树 T_j 与一个模式 P_i"结构基本相容",是指 T_j 具有与 P_i 中非叶子节点部分形状相同(但不要求对应节点的具体内容相同)的结构,且 P_i 中的叶子节点与 T_j 中相应部分(可能是叶子节点,也可能是子树)"相容".所以上例中目标 T_1, T_2 和 T_3 和模式 P: x op y(op表示任意符号)结构基本相容.

4 哈夫曼编码的改进型算法

由以上分析可得,若由权值最小的两个结点合并成新结点时,其权值大于或等于其他任意两个(或多个)结点的权时,所得到的哈夫曼树的高度最低,哈夫曼编码效果最好.另一方面,哈夫曼编码的一般作法是分两步,先构造哈夫曼树,再进行哈夫曼编码.如果在构造哈夫曼树的同时就进行哈夫曼编码,就将大大提高算法效率.为此对常规算法作出如下改进.

4.1 充分利用短编码及存储结构的改进

要降低哈夫曼树的高度,就应该充分利用短编码,在有很多与合并之后得到的权值相等时尤其如此.为此对算法的前期工作作出如下调整:

首先,按所有信源符号出现的概率由小到大排序,得到信源符号的概率序列 $\{p_1, p_2, \cdots, p_n\}$,其中 $p_1 \leqslant p_2 \leqslant \cdots \leqslant p_n$.

然后选择前面两个最小的进行合并,假设合并之后得到的概率和为 p',则在 $(p_3, p_4, \cdots, p_n)$中找到第一个大于 p'的值 p_i,现在就把 p'放在 p_i的位置,而 p_i至 p_n往后平移一个位置.通常采用顺序存储结构来存放这些概率值,这会导致大量的元素移动而降低效率,方便元素的插入和删除操作,而改为链式存储.相应结构定义如下:

```
typedef struct huffiist
    {ElemenfType weigh;
     int tag; /标志结点是叶结点或非叶结点
     struct huffnode * next;
    }HuffList;
```

另外,哈夫曼编码时,无须对所有结点进行编码,只对哈夫曼树中非叶子结点进行编码即可,因为叶结点是共享非叶结点的(如图 3所示),只对 a, b, c 和 d 4个结点编码,这样既减少了编码结点的个数又缩短编码的长度.而且在存储时对叶结点和非叶结点区别对待会降低存储空间复杂度(相应的结构定义如下所示).但是在合并时,可能是非叶结点和非叶结点(叶结点)之间进行合并,用一个队列来保存这些已被合并而得的非叶结点,由合并的顺序保证从队首到队尾的元素的权值依次递增.如果在进行合并时有非叶结点参与就从队列中取出队首元素即可.因此得到如下算法思想:对当前得到的两个权值最小的结点(来自单链表中)p_1 和 p_2,它们是否叶结点有以下 3种情况:(1)都为叶结点:(2)都为非叶结点:(3)一个叶结点和一个非叶结点.对第一种情况,分配两个 Leaf类型结点存放其权值而作为哈夫曼树的叶结点,再分配一个 NotLeaf类型结点存放其合并后的权值作为两个 Leaf类型结点的父亲而成为哈夫曼树的非叶结点,然后把该非叶结点进入队列保存(如图 4(b, d)),同时插入单链表中相应位置;对第二种情况,分配一个 NotLeaf类型结点,从队列中取出队首的两个元素分别作该 NotLeaf类型结点的左孩子和右孩子(如图 4(f, g)),然后合并所得新权值进入单链表和对应结点进入队列保存(如图 4(h));对第三种情况,叶结点按第一种情况来操作,非叶结点按第二种情况来操作即可.图 4就是用该算法思想于上例来构树的整个过程.

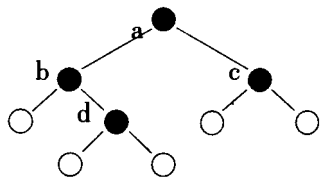

图 3 只对非叶结点编码

Fig.3 Huffman coding of Non-leaf nodes

```
typedef struct Notleaf                    typedef struct leaf
    {ElemenfType weigh;                       {ElemenfType weigh;
```

```
    char Huffcode[Max];                NotLeaf * parent;
    struct Notleaf * lch, * rch;       } Leaf; /叶结点的定义
} NotLeaf; /非叶结点的定义
```

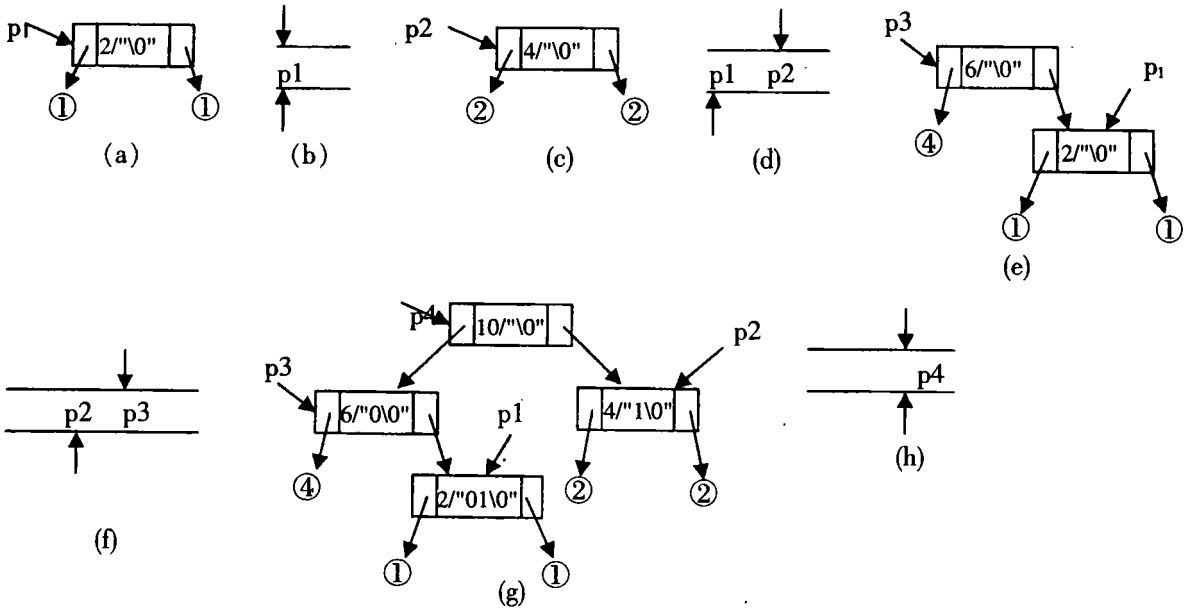

图 4　建立哈夫曼树的过程及队列的状态变化

(非叶结点中斜扛前表示权值,斜扛后表示该结点的哈夫曼编码)

Fig.4　The process of creating Huffman tree and status of queue

4.2　建树与编码同时进行

为了进一步提高编码效率,在构建哈夫曼树的同时进行哈夫曼编码.仔细分析哈夫曼编码的全过程,就会发现左右孩子是共享其父亲的编码,所以只要知道其父亲编码就很容易得到左右孩子的编码.其次,在构树过程中会产生很多子树,当产生一个新的子树时,其哈夫曼编码可通过这样的方法来确定.以新子树为目标子树,以队列中旧子树为模式树,如果目标子树与某棵模式树 P_i(可能有多棵)结构基本相容,则将模式树 P_i 的哈夫曼编码直接赋给目标子树;如果目标子树不与任何模式树结构基本相容,就以该新结点为根先序遍历二叉树即可(算法描述如下).

```
NotLeaf HuffmanCode_preorder(NotLeaf * root)
{   NotLeaf  * ptr, * stack[Max];
    int  top=0;
    ptr=root;
    while(ptr不是叶结点 ‖ 栈不为空)
      if(ptr不是叶结点)
      {   if(ptr->lch不是叶结点)
      ptr->lch->Huffcode=ptr->Huffcode+"0";
      if(ptr->rch不是叶结点)
      {ptr->rch->Huffcode=ptr->Huffcode+"1";
      stack[top++]=ptr->rch;
      }
    ptr=ptr->lch;
    } else if(栈不空) ptr=stack[--top];
```

```
    return root;
}
```

整个哈夫曼编码过程就是,通过合并得到新子树,如果存在与该子树结构基本相容的模式树,则将模式树的哈夫曼编码赋给新子树,否则就调用上面过程对其进行哈夫曼编码.于是,哈夫曼编码可用下面形式算法予以描述.

```
NotLeafHuffmanCode(NotLeaf * root)
{   if(HuffmanCode_treematch(root, pattern)) //HuffmanCode_treematch(root, pattern)是判
      Huffmancode—Copy(pattern, root);          断 root和 pattern是否结构基本相容
      else HuffmanCode_preorder(root);
}
```

现在就可方便的知道每个叶结点(信源符号)的哈夫曼编码.首先通过它知道其父亲(由 Leaf类型的数据域 parent得到)的编码,再根据它是其父亲的左孩子还是右孩子分别定出它的编码.

5 结束语

通过这样的改进,构造的哈夫曼树在任何时候都是惟一的,而且其高度都是最低,保证长度方差最小.另一方面,改顺序存储为链式存储.这避免了在插入和删除时大量元素的移动则提高了时间效率.叶结点和非叶结点的不同对待,这大大降低了空间复杂度.在对非叶结点编码,缩短编码长度和减少编码结点数,而且在编码时利用树型模式匹配而提高编码效率.因此,改进算法在实际应用中有较大性能的改善.

参考文献:

[1] 严蔚敏,吴伟民.数据结构(C语言版)[M].北京:清华大学出版社,1997.
[2] 傅祖芸.信息论——基础理论与应用[M].北京:电子工业出版社,2001.
[3] 王晓东.算法设计与分析[M].北京:清华大学出版社,2003.
[4] SARA B, ALLEN V G.计算机算法——设计与分析导论(影印版)[M].北京:高等教育出版社,2001.
[5] 孙家骕,欧阳民,陈文科.C语言程序设计[M].北京:北京大学出版社,1998.
[6] 杨学良,张占军.分布式多媒体计算机系统教程[M].北京:电子工业出版社,2002.
[7] 刘 峰,袁春风.基于 MathML的数学表达式等价性的研究[J].计算机应用研究,2004,11:54—56.

Improved Algorithm of Huffman Coding Based on Tree Pattern Matching

LIU Xiao-feng, WU Ya-juan

(College of Computer, China West Normal University, Nanchong 637002, China)

Abstract: Generally, the storage of Huffman coding and the way of creating Huffman tree, which not only influence the coding efficiency, but also do not fully make use of the storage space. In this paper, sequential storage is changed into chain storage for reducing the space complexity, and two different storage structures are applied to store leaf node and non-leaf node. Also, in order to improve the efficiency of coding and transmitting, short coded keys and the tree pattern matching have been fully applied to the Huffman coding.

Key words: Huffman tree; Huffman coding; tree pattern matching; algorithm