

# 一种基于自顶向下的哈夫曼编码方法

吴晨晖, 王映辉

(西安理工大学 计算机科学与工程学院, 陕西 西安 710048)

**摘 要:** 哈夫曼编码作为一种无损数据压缩编码在计算机信息压缩中有广泛的应用。但传统的哈夫曼编码的实现方式是在构造哈夫曼树的基础上, 从叶子节点向上到根节点逆向进行的。为了提高编码的效率, 给出了一种新的哈夫曼编码实现方式, 该方式通过利用队列的数据结构, 从哈夫曼树的根节点出发, 向叶子节点进行编码, 在编码过程中仅将哈夫曼树的每个节点进行一次扫描就可得到各叶子节点的哈夫曼编码。该方法不仅符合编码的思维方式, 而且解决了原先编码过程中大量指针移动的问题, 将哈夫曼编码的时间复杂度由原来的  $O(n^2)$  降为  $O(n)$ 。

**关键词:** 哈夫曼树; 哈夫曼编码; 算法

**中图分类号:** TP311.56

**文献标识码:** A

**文章编号:** 1673-629X(2009)10-0051-03

## Huffman Coding Based on a Top-Down Approach

WU Chen-hui, WANG Ying-hui

(School of Computer Science and Engineering, Xi'an University of Technology, Xi'an 710048, China)

**Abstract:** Huffman coding which is a lossless data compression coding has a wide range of using in the computer information compression. However, the traditional way to achieve Huffman coding is from leaf nodes to the root node on the basis of the Huffman tree. In order to improve coding efficiency, presents a new implementation approach of the Huffman coding, which codes from the root node to leaf nodes of the Huffman tree by using the data structure of queue. In the coding process, every node is only scanned once before get the Huffman coding. This method not only coincide with the thinking of encoding mode, and solves the problem of a large number of pointers' movement, but also reduces the time complexity of the Huffman coding from  $O(n^2)$  to  $O(n)$ .

**Key words:** Huffman tree; Huffman coding; algorithm

## 0 引言

大数据量的信息会给存储器的存储容量、通信信道带宽, 以及计算机的处理速度增加极大的压力。单纯靠增加存储器容量, 提高信道带宽以及计算机的处理速度等方法来解决这个问题并不是很现实, 这时就要考虑压缩<sup>[1]</sup>。压缩的关键在于编码, 如果在对数据进行编码时, 对于常见的数据, 编码器输出较短的码字, 而对于少见的数据则用较长的码字表示<sup>[2]</sup>, 就能够实现压缩。

哈夫曼编码<sup>[3]</sup>作为一种无损<sup>[4]</sup>数据压缩编码在计算机信息压缩中有广泛的应用。但目前的哈夫曼编码

方法是通过对构造好的哈夫曼树进行自底向上的方式实现编码的。该方式不仅与常规思维的从根节点向叶子节点的编码方式相违背; 而且在算法的复杂度上, 若定义叶子节点所在层为第1层, 其父节点为第2层, 依此类推, 则处在第 $n$ 层的节点要被扫描 $n-1$ 次, 所以在程序的运行过程中存在着大量的指针移动, 其时间复杂度为  $O(n^2)$ <sup>[5]</sup>。

针对以上问题, 提出一种新的哈夫曼编码的方式, 它不仅可实现从树的根节点向叶子节点的编码, 而且可使编码的时间复杂度降为  $O(n)$ 。

## 1 自顶向下的哈夫曼编码算法

本算法采用对二叉树进行层次遍历的方式, 利用队列对整个二叉树进行一次扫描, 即可得到节点的哈夫曼编码。

### 1.1 数据结构设计

#### 1.1.1 哈夫曼树节点数据结构

在本结构体中, 除了包含节点的被编码的信息域及其权重<sup>[6]</sup>之外, 还包含了存放节点编码的整型数组

收稿日期: 2009-02-01; 修回日期: 2009-05-27

基金项目: 中国博士点基金项目(20070700002); 陕西省科技项目(2007F51, 2008K4-11); 西安市创新支持计划重点项目(XY080030)

作者简介: 吴晨晖(1984-), 男, 浙江诸暨人, 硕士研究生, 研究方向为图像处理与算法; 王映辉, 博士, 教授, 博士生导师, 研究方向为图像处理与软件工程。

key[10], 指向其父节点的指针 \* par, 指向其左右孩子节点的指针 \* Lchild 和 \* Rchild。具体如下:

```
typedef struct Node
```

```
{
    char data;
    int weight;
    char key[10];
    struct Node * Lchild, * Rchild, * par;
}BTNode, * BT;
```

### 1.1.2 用于编码的队列的数据结构

本算法采用的是循环队列, front 指向队头节点, rear 指向队尾节点, count 表示当前队列中节点的个数, data[] 是模拟队列的数组。

```
typedef struct
```

```
{
    int front, rear;
    int count;
    BT data[queuesize];
}cirqueue;
```

## 1.2 算法描述

本算法从哈夫曼树的根节点开始, 通过利用队列, 按照层次遍历的方法依次对树中除根节点以外的每一个节点进行编码。算法执行过程如下:

1) 将哈夫曼树的根节点入队。

2) 若当前队列不为空, 做以下操作:

a. 指针 p 指向当前队头节点;

b. 若当前队头节点无父节点, 即根节点, 则该根节点出队, 并让其左孩子节点和右孩子节点先后入队;

c. 若当前节点有父节点, 则将父节点的哈夫曼编码赋给其左、右孩子节点, 而后, 若此节点为其父节点的左孩子, 则在其父节点所赋给的编码后面加一个 '0', 若此节点为其父节点的右孩子, 则在其父节点所赋给的编码后面加一个 '1'; 由于根节点无编码, 所以根节点的左右孩子节点不复制根节点的编码, 直接分别得到 '0', '1' 作为自己的编码;

d. 队头节点出队; 若出队节点有左右孩子节点, 则让其左右孩子分别入队, 若出队节点没有左右孩子节点, 转向 e;

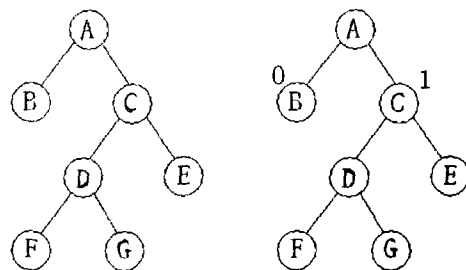
e. 判断队列是否为空。

3) 若当前队列为空, 则编码完成。

编码过程如图 1 所示。

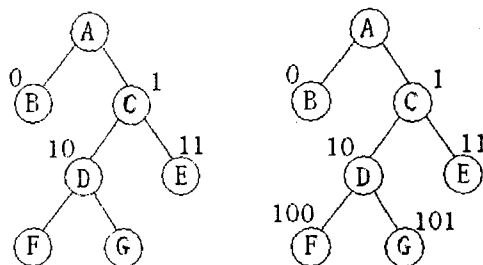
图 1(a) 表示一棵已经建好但还未进行编码的二叉树。图 1(b) 表示对根节点的孩子进行编码。图 1(c) 表示先将 C 节点的编码 '1' 赋给其孩子节点 D 和 E, 而 D 是 C 的左孩子, 故在编码 '1' 的后面加 '0', E 是 C 的右孩子节点, 故在编码 '1' 的后面加 '1'。图 1

(d) 表示先将 D 的编码 '10' 赋给其孩子节点 F 和 G, F 是 D 的左孩子, 故在编码 '10' 后面加 '0', G 是 D 的右孩子, 故在编码 '10' 后面加 '1'。



(a) 编码前的哈夫曼树

(b) 给第三层节点编码



(c) 给第二层节点编码

(d) 给第一层节点编码

图 1 编码过程描述示例

算法流程如图 2 所示。

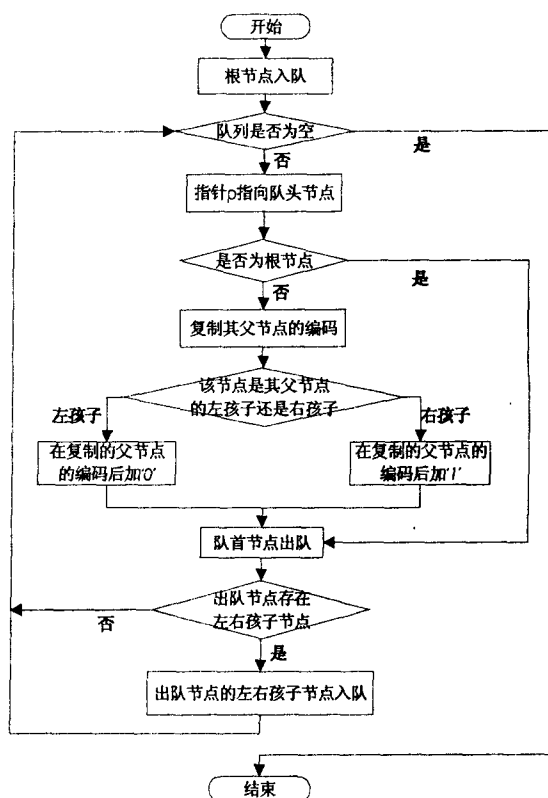


图 2 算法流程图

## 1.3 算法的具体实现(C语言)

BT lever(BT t)

```

{
int i=0;
cirqueue *q;
BT p;
p=malloc(sizeof(p));
p=t;
q=(cirqueue *)malloc(sizeof(cirqueue));
q->rear=q->front=q->count=0;
q->data[q->rear]=t;//根节点放入队列
q->count++;
q->rear=(q->rear+1)%queuesize;//重新设置队尾
while(q->count)//队列不空
{
if(q->data[q->front])
{
p=q->data[q->front];//p指向当前队头节点
if(p->par!=NULL)
{
if(p->par->Lchild=p)
{
if(p->par!=NULL)//若存在父节点,将父节点的代码给其子节点
{
for(i=0;p->par->key[i]='0' || p->par->key[i]='1';i++)
p->key[i]=p->par->key[i];
p->key[i]='0';
}
else break;
}
else if(p->par->Rchild=p)
{
for(i=0;p->par->key[i]='0' || p->par->key[i]='1';i++)
p->key[i]=p->par->key[i];
p->key[i]='1';
}
}
q->front=(q->front+1)%queuesize;//重新设置队首
q->count--;//队首元素出队
if(q->count==queuesize)
printf("队列已满");
else
{
q->count++;
q->data[q->rear]=p->Lchild;//左孩子节点入队
q->rear=(q->rear+1)%queuesize;
}
if(q->count==queuesize)
printf("队列已满");
}
}

```

```

else
{
q->count++;
q->data[q->rear]=p->Rchild;//右孩子节点入队
q->rear=(q->rear+1)%queuesize;
}
}
else//若队头节点已经出队,重新设置队头节点
{
q->front=(q->front+1)%queuesize;
q->count--;
}
}
return t;
}

```

#### 1.4 算法效率分析

若需参与编码的节点共有  $n$  个,则所建成的哈夫曼树共有  $2n-1$  个节点。由于本算法通过队列对节点进行编码,所以每个节点仅被扫描一次。故该算法在有  $2n-1$  个节点的哈夫曼树上的执行频度为  $2n-1$ ,其时间复杂度为  $O(n)$ 。

## 2 结束语

哈夫曼编码是已经被证明的一种有效的熵编码<sup>[7]</sup>方式。在诸如文本、图像、视频压缩及通信、密码等信息压缩编码标准中,哈夫曼编码被广泛使用<sup>[8]</sup>,它的执行效率影响着其它各级编码系统的执行效率,尤其是在对数据处理能力的高要求与日俱增的情况下,更是如此。而文中介绍的这种新的哈夫曼编码方法和实现过程,不仅具有一般哈夫曼编码节省存储空间的特点,而且在算法的时间复杂度上有很程度的改观,其应用优势更为显然。

#### 参考文献:

- [1] Vitter J S. Dynamic Huffman Coding[J]. ACM Transactions on Mathematical Software, 1989,15(2):158-167.
- [2] 朱怀宏,吴楠,夏黎春,等.利用优化哈夫曼编码进行数据压缩的探索[J].微机发展(现更名:计算机技术与发展),2002,12(5):1-5.
- [3] Huffman D A. A method for the construction of minimum redundancy codes[J]. Proceedings of the IRE, 1952,40(9):1098-1101.
- [4] 黄伟,龚沛曾.图像压缩中的几种编码方法[J].计算机应用研究,2003,20(8):67-72.
- [5] 刘帮涛,罗敏.改进的赫夫曼树(Huffman Tree)和赫夫曼编码(Huffman Code)构造算法[J].福建电脑,2008(9):77-91.

(下转第 58 页)

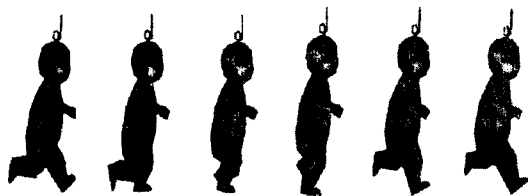


图 5 “跑”动画

算法的部分核心代码如下:

```
void MD2Model::render(void) const {
    if (!data_) return; //如果文件数据为空则返回
    if (! (MD2_ALWAYS_GL_TRIANGLES) && model() ->
numCommands > 1) { //判断文件是否有图形优化命令
        int gl_command_data_size = 4 * model() -> numCom-
mands;
        //获得图形优化命令数据的总体大小
        unsigned char * ptr = data_ + model() -> offsetCom-
mands;
        //指向第一个命令
        unsigned char * end = ptr + gl_command_data_size;
        //指向最后一个命令
        while (ptr != end) {
            int num = pCommand->num;
            GLenum PrimitiveType = GL_TRIANGLE_STRIP;
            if (num < 0) { //小于 0,改用三角面片扇模式绘制
                PrimitiveType = GL_TRIANGLE_FAN;
                num = -num;
            }
            glBegin(PrimitiveType);
            { //绘制开始
                glCommandVertex * pStart = pCommand->verts;
                glCommandVertex * pEnd = pStart + num;
                for (; pStart != pEnd; ++pStart) {
                    glTexCoord2fv(pStart->data);
                    glVertex3fv(vertices_ + 3 * pStart->vertexIndex);
                }
            }
            glEnd();
            //指针指向下一个命令
            ptr += (4 + sizeof(glCommandVertex) * num);
        } else { //如果没有图形优化命令,改用三角面片渲染
            glBegin(GL_TRIANGLES);
            unsigned short nTris = numTriangles();
            const Triangle * pt = triangles();
            for (unsigned short i=0; i!=nTris; ++i) { //遍历每个三
                角面片
```

```
for (int j=0; j!=3; ++j) { //遍历每个面片的顶点
    float * vertex = vertices_ + 3 * pt[i]. vertexIndices[j];
    Texture * uv = texCoords() + pt[i]. textureIndices[j];
    glTexCoord2sv(uv->data);
    glVertex3fv(vertex);
}
glEnd();
glMatrixMode(GL_TEXTURE);
glPopMatrix();
glMatrixMode(GL_MODELVIEW);
}
```

#### 4 结束语

文中对 MD2 文件格式进行了深入的剖析,详细论述了文件的细部结构,阐明了如何准确定位相关图形与动画数据在文件中的位置,并给出了利用编程工具解析文件数据的实例。运用阐述的文件格式分析结果,并借助 OpenGL 或 Direct3D 等,可方便地存取动画文件模型数据,实现动画的交互控制与实时绘制。相关分析结果可为三维游戏开发、交互式三维图形应用等提供支持。

#### 参考文献:

- [1] 韩桃,宋文忠.基于 VC++ 的 OpenGL 三维动画仿真系统的实现[J]. 微机发展(现更名:计算机技术与发展), 2004,14(11):52-53.
- [2] 张大强,翟素兰,程家兴. OpenGL 在视频游戏中的应用[J]. 计算机技术与发展,2006,16(2):73-75.
- [3] 盛 斌,杨景曙. MD2 文件的读取及三维动画显示[EB/OL]. 2007. [http://www.solw.cn/Article/20070319102242\\_7319.html](http://www.solw.cn/Article/20070319102242_7319.html).
- [4] Phiph E. Focus on 3D Models[M]. [s.l.]: Premier Press, 2003.
- [5] Henry D. The Quake II's MD2 file format[EB/OL]. 2002-12. <http://tfc.duke.free.fr/old/models/md2.htm>.
- [6] 禹仁贵,任 磊,王 宁. 基于 OpenGL 的纹理映射的实现与应用[J]. 西南民族大学学报,2008(10):1261-1263.
- [7] Devaney B, Kenyon J, Robertson C, et al. Modeling Aspects of the Dynasty 3D Game[C]//In: Proceedings of the International Conference on Software Engineering Research and Practice. Nevada, USA: [s. n.], 2004:283-289.

(上接第 53 页)

- [6] Larmore L L, Hirschberg D S. A Fast Algorithm for Optimal Length-Limited Huffman Codes[J]. Journal of the Association for Computing Machinery, 1990,37(3):464-473.
- [7] 杨继华,严国萍. 基于嵌入式 Linux 系统的 JPEG 压缩算法

实现[J]. 微机发展(现更名:计算机技术与发展),2005,15(3):7-10.

- [8] 李伟生,李 域,王 涛. 一种不用建造 Huffman 树的高效 Huffman 编码算法[J]. 中国图象图形学报,2005,10(3):382-387.