中山大学学报 (自然科学版)
第 35卷 第 5期 ACTA SCIENTIARUM NATURALIUM Vol. 35 No. 4
1996年 9月 UNIVERSITATIS SUNYATSENI Sep. 1996

数据压缩的功能字典方法*

谭 兆 信
(中山大学岭南学院,广 州 510275)

摘 要 提出一种无损数据压缩字典方法,称为功能字典方法.该方法能较大程度上消除数据文件中多种形式的冗余,达到较好的压缩效果.

关键词 图像处理,数据压缩,计算机
分类号 TP 391.41

在计算机处理的文件和信息中(如图像文件,文本文件,可执行文件等),往往存在很多冗余,实现数据压缩,这是许多学者研究的目标.尤其是近年来,多媒体计算机的研究和信息高速公路的研究迅速发展.而如何实现更高质量的数据压缩,成了多媒体计算机和信息高速公路技术的关键.在无损数据压缩技术中,除 Huffman编码和算术编码等统计方法外,数据压缩字典方法获得了显著的成功.本文着重探讨无损数据压缩的字典方法.

现代字典压缩方法起源于 Ziv和 Lempel的论文[1],其中的算法称为 LZ 77算法,经过较长时间的改进, LZ 77后来发展成为 LZ SS算法.字典压缩的另一著名的算法是 LZ 78,随后它发展成为 LZW 算法.近年来还出现了许多成功的字典压缩算法,例如字典森林方法[3]和 FI方法[5]等.

在计算机处理的文件中,短语和词的重复是文件冗余性的主要表现之一[3].数据压缩字典方法总是力图清除文件中的这种冗余性.而不同的字典方法则又侧重于消除这种冗余性的不同表现.不同的压缩方法对不同形式的重复串有不同的压缩能力.

本文提出功能字典方法,是力图考虑源文件中多种形式的重复串的压缩,以求达到较高的压缩效果,我们将功能字典方法与 LZSS, LZW,字典森林[3]等方法进行比较,程序结果表明,功能字典具有较好的压缩效果,平均压缩比高于其它方法.

1 功能字典的结构及压缩算法 FUDC

把要压缩的数据文件称为源文件,用 sfile表示.或用 $s[m]$($m=0,1,2\cdots$)表示.$s[m]$的每个元素是 8 bits的数据,称为字符."字符"实际上可以是文件中的字符,也可以是图像中的一个像素或其它数据.压缩程序的输出称为编码文件,用 codefile表示,压缩算法是动态的无损编码.

令 L_o表示输出的码长(取为 11, 12或更大).给定正整数 h_1, h_2, h_3(例如 $h_1=2, h_2=4, h_3$

* 广东省自然科学基金资助项目
收稿日期: 1995-12-05 谭兆信,男, 55岁,副教授

(C)1994-2019 China Academic Journal Electronic Publishing House. All rights reserved. http://w

= 16;令 $max\text{-}code = (1 \ll L_0) - 1$; $lmax = h_2 + h_3$;输出的码值 $code$ 满足: $0 <= code <= max\text{-}code$.

在功能字典方法中,用到一个称为 table的线性表. table的结构为

```
struct node
    ( unstgned char str[lmax. ];      /* 表目表示的字符串* /
    int sl;                           /* 串 str的有效长度* /
    long int np;                      /* 子串 str( 0, sl- 1)在源文件中最后出现的位置* /
    int disp;                         /* disp= 0表示该表目为空,否则为非空* /
    int llink;                        /* 指向左儿子* /
    int rlink;                        /* 指向右儿子* /
    ) *  table [max_ code+ 1]
```

table表的下标对应的整数区间 $[0, max\text{-}code]$称为编码区段.把编码区段分为 4个编码分段, $seg_0, seg_1, seg_2, seg_3$.各个分段的下限,上限分别为 $lb_0, ub_0, lb_1, ub_1, lb_2, ub_2, lb_3, ub_3$.相应地, table表也分为 4个分表 $table_0, table_1, table_2, table_3$.不同分表,在文件压缩中有不同的功能.称 table为功能字典,记为 FUD.

压缩程序的输出由码值 $code$ 组成,对某些分段的 $code$,输出附加信息,各分段码值的附加信息及其含义如下:

编码分段	附加信息	含义 (编码表示的字符串)
seg_0	无	单字符
seg_1	无	长为 1的重复串 ($h_1 <= 1 <= h_2$)
seg_2	4 bits	长为 1的重复串 ($h_2 < 1 <= lmax$) (其中 $lmax = h_2 + h_3$)
seg_3	1 byte	长为 1的由相同字符组成的串 ($lmax < 1 <= lmax + 256$)

为了叙述方便,用 $s(i, j)$表示子串 $s[i], s[i+1], \cdots s[j]$;而用 curstr表示源文件 $s[m]$ ($m = 0, 1, 2\cdots$)中当前要处理的长为 $lmax$ 的子串,算法中采用 $s[m]$的两个下标 sp 和 $next\text{-}p$. sp 为字符 curstr[0]对应的下标,即串 curstr(0, $lmax - 1$)与源文件的字符串 $s(sp, sp + lmax - 1)$相同. $next\text{-}p$ 表示 $s[m]$中,下标 $next\text{-}p$ 以前的各字符均已编码输出,即 $s(0, next\text{-}p - 1)$已编码输出.

$table_1$ 和 $table_2$ 中的表目组成了按 str 字段排序的顺序二叉树,而字段 llink和 rlink分别指向结点的左、右儿子.用指针 root指向二叉树的根.每当将源文件中的一个字符 (例如 $s[sp]$编码后,就将以该字符为首字符,长为 $1(1= h_1, h_1+1, \cdots, h_2)$的串存入 $table_1$ 的空闲表目中.而长为 $lmax$ 的串存入 $table_2$ 的空闲表目中.对于 $h_1 = 2, h_2 = 4, h_3 = 16$, ($lmax = 20$)的情形,即是将串 curstr(0, 1), curstr(0, 2), curstr(0, 3)存入 $table_1$;串 curstr(0, $lmax - 1$)存入 $table_2$中.空闲表目的选取是分别在 $table_1$ 和 $table_2$ 中按下标由小至大选取空闲的表目.为了使 table表目的 str字段有统一的长度,当串长 $1 < lmax$ 时,在串右边补上若干个空格,而用字段 sl 表示其原来的有效长度.

在本文,我们给定常数 h_1, h_2, h_3, L_0 分别为 2, 4, 16, 11.并定义如下的常数: $max\text{-}code = (1 \ll L_0) - 1 = 2047, lb_0 = 0, ub_0 = 255, lb_1 = 256, lb_3 = ub_3 = max\text{-}code, ub_2 = lb_3 - 1, lb_2$ 为 lb_1 与 ub_2 之间的某个值 (例如 $lb_2 = 1024$), $ub_1 = lb_2 - 1, lmax = h_2 + h_3$.

功能字典压缩算法 FUDC

(C)1994-2019 China Academic Journal Electronic Publishing House. All rights reserved. http://w

step 1. 初始化. 确定 $lb_0, ub_0, \cdots, ub_3$ 等常数, 令 table[i]. $disp$= 0 ($0 <= i <= max\text{-}code$); 令 $root$= -1

step 2. 将常数 h_1, h_2, h_3, L_0 和 lb_2 输出到 codefile.

step 3. 从 sfile输入 $lmax$ 个字符到 curstr(0, $lmax$ - 1); 令 sp= 1, $next\text{-}p$= 1(若 sfile 少于 $lmax$ 个字符, 则将这些字符输出到 codefile, 算法结束).

step 4. 输出单字符 curstr[0]到 codefile; 令 $next\text{-}p$= $next\text{-}p$+ 1

step 5. 若 sfile结束, 则转 step 9.

step 6. 若 sp < $next\text{-}p$, 则转 step 7, 否则转 step 8.

step 7. 把以 curstr[0]为首字符, 长为 1的串 (1= h_1, h_1+ 1, $\cdots, h_2$)插入 table$_1$ 中; 长为 1= $lmax$ 的串插入 table$_2$ 中, 插入方法如前述. 从 sfile中读入一字符 ch, 令: curstr(0, $lmax$ - 2)= curstr(1, $lmax$ - 1); curstr[$lmax$ - 1]= ch; sp= sp+ 1; 转 step 5.

step 8. 考虑串 curstr(0, $lmax$ - 1); 采用顺序二叉树查找算法在以 root为根的树中, 找出结点 (设结点指针为 tp), 使 table[tp]. str与串 curstr(0, $lmax$ - 1)有最长的匹配长度, 设长为 ml; 则匹配的子串为 curstr(0, ml - 1); 令 $offset$= sp - table[tp]. np (必有 $offset$> 0), 按以下情形输出编码, 然后转 step 5.

(1)若 ml < h_1 (即 ml= 0或 1), 令 $code$= curstr[0]; 输出 $code$; 令 $next\text{-}p$= $next\text{-}p$+ 1.

(2)若 h_1 <= ml <= h_2 (则 lb_1 <= tp <= ub_1), 令 $code$= tp; 输出 $code$; 令 $next\text{-}p$= $next\text{-}p$+ ml.

(3)若 (h_2 < ml < $lmax$) or ((ml= $lmax$) and (table[tp]. np != sp - 1)) (则 lb_2 <= tp <= ub_2), 令 $code$= tp, 输出 $code$ 并输出附加值 $ml - h_2$ - 1(长度为 4 bits).

(4)若 (ml= $lmax$) and (table[tp]. np= sp - 1) (串 curstr(0, $lmax$ - 1)中每个字符均与 s[sp - 1]相同), 则执行:

```
ch0= curstr[0];
ch= ch0;   k= 0;
while ((not eof (sfile)) and (ch= ch0) and (k < 256))
(ch= getc(sfile));
k++);
ml= ml+ k - 1;   next-p= next-p+ ml;   sp= sp+ k;
curstr[lmax - 1]= ch;   code= lb3;
```

然后输出 $code$, 并输出附加值 $ml - lmax$.

step 9. 输出 curstr中余下的字符, 即执行:

```
i= lmax - 1+ (next-p - sp);
for(j= i; j <= 2* lmax - 2; j++)输出 curstr[j];
```

压缩算法 FUDC结束

2 解码算法

解码算法以 codefile为输入文件, 输出文件为 tfile. 为叙述方便, 我们把 tfile理解为字符序列 s[k] (k= 0, 1, 2··), 用 sp表示 s的当前空, 即 s[0], s[1], $\cdots, s$[sp- 1]已解码输出. 用 $out\text{-}len$ 表示每次解码输出的串的长度. 解码算法也使用 table表. 表的结构与编码程序的

(C)1994-2019 China Academic Journal Electronic Publishing House. All rights reserved.　http://w

table相同,为了叙述方便,我们仍以 $h_1=2, h_2=4, h_3=16$为例.

在编码阶段,设源文件中,$s[k](k=0,1,2,\cdots sp-1)$已编码,则对任意 $j(0<=j<sp)$,以 $s[j]$为首字符的长为 2,3,4,20的串均已进入 table中.我们特别注意,当 $sp-lmax+1<j<sp$时,以 $s[j]$为首字符的长为 2,3,4,20的串(共 4* 19个串)也已进入 table中,在解码阶段,设已得到输出 $s[k](k=0,1,2,\cdots sp-1)$,但字符 $s[sp],s[sp+1]\cdots$仍未解码输出.因而,当 $sp-lmax+1<=j<sp$时,以 $s[j]$为首字符,长分别为 2,3,4,20的串中有 25个未能完全确定(长为 2,3,4,20的各有 1,2,3,19个),称为不确定串,对这些不确定串采用一种称为预分配的方法进行处理.在解码阶段,当一个不确定串要进入 table表时,我们仍分配一个表目.表目的 np, sl字段赋予正确的值;而 str字段则为任意值.且令 $disp=2$,以表示该表目的 str未确定.随着编码的继续进行,原来的不确定串变成了确定串,我们将其 str字段赋予确定值,令 $disp=1$,并按顺序二叉树插入算法令其 llink和 rlink分别指向左儿子.每当解码多一个字符,都有 4个不确定的串变为确定的串而又产生 4个不确定的串(除文件开头和结尾外),幸好这样的不确定的串只有 25个,因而可以用一个只有 25个元素的数组记下相应表目的下标,当不确定的串变为确定的串时,要找出其位置进行处理也是很省时的事,采用预分配方法,使得在编码阶段提前将一些串插入 table中,有利于提高压缩比.

为了便于处理不确定串,我们仍采用长为 2* $lmax$ 的字符缓冲区 curstr.用 p表示 curstr的当前空下标. curstr(0,$lmax-1$)存放已解码的字符中最新的 $lmax$ 个字符.在开始对一个码值 $code$解码时,$p=lmax$.对于确定的串,从 $table$中相应表目的 str字段得出解码结果;对于不确定串,从 curstr中得出解码结果.

解码算法

step 1. 从 codefile输入 h_1, h_2, h_3, L_o 和 lb_2,计算 $lb_o, ub_o, lb_1, ub_1, \cdots, lb_3, ub_3$(计算公式与压缩算法相同).

step 2 将 table初始化,令 table[i].$disp=0$ ($lb_1<=i<=ub_2$); $root=-1$.且令 sp= 1; $p=lmax$; curstr[i]= ''($0<=i<2*lmax$).

step 3. 若 codefile已结束,则转 step 6,否则按以下情形分别处理:

(1)若 $lb_o<=code<=ub_o$,令 ch为 $code$的低 8位;输出 ch;令 $sp=sp+1$;令 curstr[p]= ch; $p=p+1$; $out\_len=1$.

(2)若 $lb_1<=code<=ub_1$,令 $j=$ table[$code$].np; $out\_len=$ table[$code$].sl;调用过程 out_str输出;令 $sp=sp+out\_len$.

(3)若 $ub_1<=code<=ub_2$,令 $j=$ table[$code$].np; 从文件 codefile读入 4 bits的整数 out_len;调用过程 out_str输出;令 $sp=sp+out\_len$.

(4)若 $code=lb_3$,令 $ch_o=$ curstr[$lmax-1$];从 codefile输入 8 bits的整数 out_len;令 $out\_len=out\_len+lmax$;输出 ch_o 共 out_len 次; $sp=sp+out\_len$;令 $curstr[i]=ch_o$ ($i=lmax, lmax+1,\cdots, 2*lmax-1$)且令 $p=2*lmax$.

step 4. 对不确定串进行处理,把新的串插入 table表中,即,依次对 $j=lmax, lmax+1,\cdots, p-1$作如下处理:依次将以 curstr[$j$]为末字符,长为 2,3,4和 20的串取代 table1或 table2中对应的不确定串,修改 disp等字段(如上述);将以 curstr[j]为首字符,其余字符为空格,长依次为 2,3,4和 20的串存入 table1或 table2的空闲表目中,且令 $disp=2$.

step 5. 将 curstr中的字符左移,即,令 curstr(0,$lmax-1$)= curstr($p-lmax, p-1$);且 $p=lmax$.转 step 3.

step 6. 关闭 codefile, tfile,算法结束.

(C)1994-2019 China Academic Journal Electronic Publishing House. All rights reserved. http://w

过程 out_str的算法如下:

```
if(j - sp+ out_len - 1) < p- lmax /* 输出确定串 * /
    (for (i= 0; i <= out_len - 1;i+ + )
        (ch= table[code].str[i];输出 ch;
          curstr[p ]= ch; p+ + ;
    )
)
else    /* 输出不确定串 * /
    (for (i= 0; i <out_len - 1; i+ + )
      (ch= curstr[j - sp+ lmax+ i];输出 ch;
        curstr[p ]= ch;   p+ + ;
          )
    )
```

过程 out_str结束

3 实验结果

为了分析 FUDC的压缩效能,我们选择了字典方法中较有代表性的 LZSS, LZW,经改进的字典森林方法[3]与 FUDC进行比较. LZSS和 LZW是当前数据压缩商品软件中使用最广泛的数据压缩字典方法;为了与统计方法比较,我们选择了算术编码方法.下表列出了对 5个文本文件(包括 C语言源程序和 PASCAL语言源程序)、5个可执行文件和 5个图像文件(. BMP文件)的平均压缩比以及对 15个文件的总平均压缩比,其中, ratio= (源文件长度 - 编码文件长度) /源文件长度.

表 1 实验结果

Tab. 1 Results of experiment

算　法	ratio/%			
	文本文件	可执行文件	图象文件	总平均值
FUDC	63. 52	50. 12	41. 52	51. 69
LZW	57. 30	39. 82	41. 80	46. 31
LZSS	65. 74	44. 80	36. 88	49. 14
FOREST	54. 06	22. 84	13. 58	30. 16
ARITH	36. 56	29. 56	22. 88	29. 67
源文件平均长度	12514	26145	142697	60461

4 结　语

上述数据表明,对文本文件, LZSS有最高压缩比, FUDC次之;对可执行文件, FUDC有最高压缩比, LZSS次之;对图像文件(. BMP文件) LZW有最高压缩比, FUDC次之. 3种文件的总平均压缩比, FUDC最高, LZSS次之, LZW再次之.

由此可见, FUDC对各种文件有较均衡的压缩比;而对三种文件,在总体上,有最高的压缩能力.因而, FUDC是一种有价值的压缩方法.但 FUDC算法较为复杂,这是为得到较好的压缩比而付出代价.从算法复杂性理论的角度分析, FUDC算法的主要时间花费是对顺序二叉树的查找和插入.幸好,人们对二叉树的查找和插入已深入研究,已有成熟的高效的算法,其平均时间复杂性为 $O(\lg n)$, n 为二叉树结点数目.对于 FUDC, $\lg(n) <= l_0$ (l_0 为编码长度(例如 l_0= 11或 l_0= 12).因此,压缩算法或解码算法的平均时间复杂性可认为是 $O(l_0 * \text{filelength})$,其中 file_length是源文件的长度.

(C)1994-2019 China Academic Journal Electronic Publishing House. All rights reserved.　http://w

参 考 文 献

1 Ziv J, Lempel A. A universal algorithm for sequential data compression. IEEE Transactions on Information Theory. 1977, IT-23, (3): 337~343

2 Ziv J, Lempel A. Compression of Individual sequences via variable-Rate Coding. IEEE Transactions on Information Theory, 1978, IT-24, (5): 530~536

3 Jakobsson M. Compression of character strings by an adaptive dictionary. Bit, 1985, 25, (4): 593~603

4 Bailey R L, Mukkamala R. Pipelining data compression algorithms. The Computer Journal, 1990, 33, (4): 308~313

5 Fisher P S, Cleopas A. Text Compression Using FI Sequences. Proceedings of 9th Computing in Aerospace, 1993

Function Dictionary for Data Compression

Tan Zhaoxin[*]

Abstract A new method to data compression is presented. The method is based on the use of a dictionary called function dictionary. The method can reduce several kinds of redundancy in data files and attains high compression ratio.

Keywords image processing, data compression, computer

[*] Department of Computer Science, Zhongshan University, Guangzhou 510275

(C)1994-2019 China Academic Journal Electronic Publishing House. All rights reserved. http://w