ALG LAB6

王世炟 PB20151796
2022/11/19

Huffman 编码问题

实验内容

编程实现 Huffman 编码问题，并理解其核心思想。

对文件 `original.txt` 中所有的大小写字母、数字（0-9）以及标点符号（即：除空格 换行等之外的所有字符）按照 `Huffman` 编码方式编码为 01 序列，输出如下格式的 `table.txt` 文件, 并在控制台打印压缩率。

算法设计思路

首先，将所有的字符以及它们出现的频率存在结点中，并将这些结点放入优先队列，形成森林。

每一次将优先队列的前两个结点（频率最小的两个结点）出队，并合并成为一个新的结点，之后再将此结点加入优先队列，并维护优先队列的性质。

当优先队列中只剩下一个结点的时候，树便建立成功。

之后从根节点开始向下遍历，依次添加编码。

源码+注释

结点结构：

```
// Huffman树的节点类
typedef struct node
{
    char value;            //节点的字符值
    double freq;           //节点字符出现的频度
    vector<int> code;      //节点字符对应的编码
    node *lchild, *rchild; //节点的左右孩子
} HFMNode, *pHFMNode;
```

优先队列比较方法：

```
struct CMP
{
    bool operator()(const pHFMNode &p, const pHFMNode &q)
    {
        return p->freq > q->freq;
    }
};
```

建立哈夫曼树：

```
void Huffman(priority_queue<pHFMNode, vector<pHFMNode>, CMP> &vctNode)
{
    while (vctNode.size() > 1)
    {
        pHFMNode first = vctNode.top(); //取vctNode森林中频度最小的树根
        vctNode.pop();
        pHFMNode second = vctNode.top(); //取vctNode森林中频度第二小的树根
        vctNode.pop();
        pHFMNode merge = new HFMNode; //合并上面两个树
        merge->freq = first->freq + second->freq;
        merge->lchild = first;  //小的放左子树
        merge->rchild = second; //大的放右子树
        vctNode.push(merge);    //向vctNode森林中添加合并后的merge树
    }
    return;
}
```

输出编码：

```
void PrintHuffman(pHFMNode node, vector<int> &vctCode, ofstream &outfile)
{
    if ((node->lchild == NULL) && (node->rchild == NULL))
    {
        node->code.assign(vctCode.begin(), vctCode.end()); //复制vctCode到code
        outfile << node->value << "    " << left << setw(8) << node->freq;
        for (vector<int>::iterator iter = node->code.begin(); iter != node->code.end(); iter++)
        {
            outfile << *iter;
        }
        outfile << endl;
        code_length += vctCode.size() * node->freq;
    }
    else
    {
        vctCode.push_back(0); //遇到左子树时给vctCode中加一个0
        PrintHuffman(node->lchild, vctCode, outfile);
        vctCode.pop_back();   //回溯，删除刚刚加进去的1
        vctCode.push_back(1); //遇到右子树时给vctCode中加一个1
        PrintHuffman(node->rchild, vctCode, outfile);
        vctCode.pop_back(); //回溯，删除刚刚加进去的0
    }
    return;
}
```

算法测试结果

可得到编码如下：

```
字符 出现频率 编码
d    6281    0000
f    2878    00010
u    3428    00011
m    3494    00100
w    3518    00101
F    188     001100000
E    95      0011000010
R    49      00110000110
P    50      00110000111
"    391     00110001
M    197     001100100
1    5       00110010100000
2    1       0011001010000100
K    1       0011001010000101
3    3       001100101000011
X    3       001100101000100
7    1       0011001010001010
5    2       0011001010001011
4    4       001100101000110
Q    4       001100101000111
U    15      0011001010010
9    4       001100101001100
6    2       0011001010011010
0    2       0011001010011011
V    9       00110010100111
;    56      00110010101
!    112     0011001011
T    430     00110011
,    1938    001101
'    233     001110000
W    120     0011100010
j    120     0011100011
N    245     001110010
I    261     001110011
k    985     0011101
b    2029    001111
r    7643    0100
s    7685    0101
i    7886    0110
h    8491    0111
e    17214   100
n    9253    1010
p    2149    101100
y    2173    101101
x    274     101110000
B    280     101110001
H    141     1011100100
?    67      10111001010
G    18      1011100101100
(    18      1011100101101
)    18      1011100101110
Y    21      1011100101111
-    292     101110011
v    1192    1011101
c    2663    101111
o    9890    1100
a    10834   1101
g    2761    111000
A    312     111001000
q    153     1110010010
C    160     1110010011
S    339     111001010
O    80      11100101100
:    40      111001011010
D    42      111001011011
z    45      111001011100
L    45      111001011101
J    93      11100101111
.    1512    1110011
l    5847    11101
t    11515   1111
```

压缩率： 0.641293