Huffman 编码问题

学号：PB21000224　　姓名：陈鸿绪　　日期：11.22

实验内容：对字符串进行 01 编码，输出编码后的 01 序列，并比较其相对于定长编码的压缩率。

实验要求：对文件 orignal.txt 中所有的大小写字母、数字（0-9）以及标点符号（即：除空格换行符之外的所有字符）按照 Huffman 编码方式编码为 01 序列，输出固定格式的 table.txt 文件，并在控制台打印压缩率。注意，编码方式可能不唯一，但压缩率是确定的，字符出现频率编码。

算法设计思路：开一个长度 187 的数组作为存储哈夫曼树，前 94 位作为叶子结点存储带有关键词的结点，构造一个优先队列辅助哈夫曼算法，根据哈夫曼算法每次得到的两个结点的父节点赋值，一直到所有数组都存储满为止，最后一个位置即为哈夫曼树的根。从每一个叶子结点出发往根走即可得到每个结点的编码，最后需要将编码反转。对于压缩率可以通过对每个叶子结点总编码长度求和除以非哈夫曼编码得到的总长度求出来。

源代码解释：

```
#define N 1000;
struct node{
    int fre,pos;
    int par,lch,rch;
    char code[50];
    int code_len;
}; //哈夫曼树的结点
struct tmp{
```

```
    bool operator() (node a,node b){
        return a.fre>b.fre;
    }
};//用于快速排序的 tmp 函数
typedef struct node node;
priority_queue<node, vector<node>, tmp> q; //优先队列
node fre_char[187]; //存储哈夫曼树的数组
void get_fre_char(){
    for(int i=0;i<187;i++){
        fre_char[i].fre=0;
        fre_char[i].lch=-1;
        fre_char[i].rch=-1;
        fre_char[i].par=-1;
        fre_char[i].pos=i;
        fre_char[i].code_len=0;
    }//赋予初值
    FILE *fp=fopen("orignal.txt","r+");
    while(true){
        char s;
        fscanf(fp,"%c ",&s);
        if(feof(fp)){
            break;
        }
        fre_char[(int)(s-33)].fre+=1;
    }//读文件
    for(int i=0;i<94;i++){
        q.push(fre_char[i]);
    }//有限队列用于哈夫曼算法
}
void get_h_tree(){
//哈夫曼算法得到哈夫曼树
    int pos_i=94;
    while(true){
        node min1=q.top();
        q.pop();
        node min2=q.top();
        q.pop();
        fre_char[min1.pos].par=pos_i;
        fre_char[min2.pos].par=pos_i;
        fre_char[pos_i].lch=min1.pos;
        fre_char[pos_i].rch=min2.pos;
        node temp;
        temp.fre=min1.fre+min2.fre;
        temp.pos=pos_i++;
```

```
            q.push(temp);
            if(q.size()==1) break;
        }
}
void get_code(){
//对于每个叶子结点，从叶子到树根往上走得到每个叶子结点的反转哈夫曼编码。
        for(int i=0;i<94;i++){
            int p=i;
            while(p!=186){
                if(fre_char[fre_char[p].par].lch==p) fre_char[i].code[fre_char[i].code_len++]='0';
                else fre_char[i].code[fre_char[i].code_len++]='1';
                p=fre_char[p].par;
            }
            for(int i=0;i<=fre_char[i].code_len/2;i++){
                char temp=fre_char[i].code[i];
                fre_char[i].code[i]=fre_char[i].code[fre_char[i].code_len-i-1];
                fre_char[i].code[fre_char[i].code_len-i-1]=temp;
            }//将得到的编码反转得到正确的哈夫曼编码
        }
}
```

算法测试结果：

compress_rate:0.641287

具体字符频率以及对应编码见“table.txt”。