Lab 2 哈夫曼编码系统

姓名：阳焘

学号：2016213391

班级：2016215114

1. 问题描述：

- 设计一个利用哈夫曼算法的编码系统，能完成以下基本要求
- 基本要求
  - 键盘输入长为n的字符串，统计其中每个字符的权值（出现频次），建立哈夫曼树
  - 编码：利用建好的哈夫曼树生成哈夫曼编码
  - 输出编码（在屏幕上显示）

2. 程序结构

- 主程序函数 `main()`
- 建立哈夫曼树的函数 `void huffman(hufmtree tree[])`

  函数参数：`hufmtree tree[]`，函数返回值：`void`
- 根据哈夫曼树求哈夫曼编码的函数

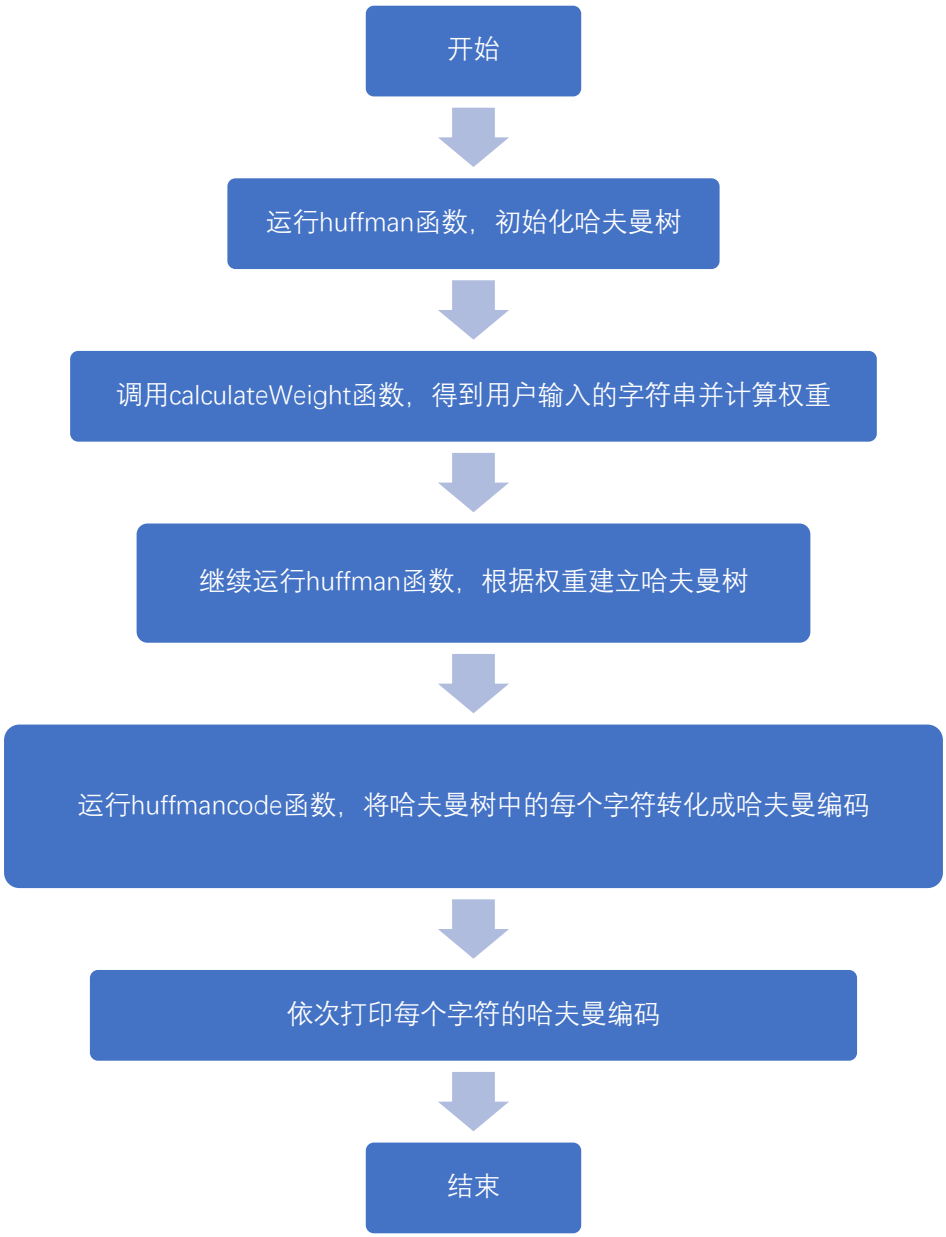
  `void huffmancode(codetype code[],hufmtree tree[])`

  函数参数：`codetype code[],hufmtree tree[]`

  函数返回值：`void`
- 计算输入字符串中每个字符权重的函数

  `void calculateWeight(hufmtree tree[])`

  函数参数：`hufmtree tree[]`，函数返回值：`void`


开始
运行huffman函数，初始化哈夫曼树
调用calculateWeight函数，得到用户输入的字符串并计算权重
继续运行huffman函数，根据权重建立哈夫曼树
运行huffmancode函数，将哈夫曼树中的每个字符转化成哈夫曼编码
依次打印每个字符的哈夫曼编码
结束

3. 源码

```
#include <stdio.h>
#include <string.h>
#define n 5  //叶子数目
#define m (2*n-1)    //结点总数
#define maxval 10000.0
#define maxsize 100   //哈夫曼编码的最大位数
#define N 100


struct Character
{
    char data;
    int count;
}Char[N];

char string[N]={0};
int length; //字符的长度
int number;  //字符的个数

typedef struct
{
    char ch;
    int weight;
    int lchild,rchild,parent;
}hufmtree;

typedef struct
```

```
{
    char bits[255];   //位串
    int start;      //编码在位串中的起始位置
    char ch;        //字符
}codetype;

void huffman(hufmtree tree[]);//建立哈夫曼树
void huffmancode(codetype code[],hufmtree tree[]);//根据哈夫曼树
求出哈夫曼编码



int main()
{
    hufmtree tree[m];
    codetype code[n];
    int i,j;//循环变量
    huffman(tree);//建立哈夫曼树
    huffmancode(code,tree);//根据哈夫曼树求出哈夫曼编码
    printf("输出每个字符的哈夫曼编码:\n");
    for(i=0;i<n;i++)
    {
        printf("%c: ",code[i].ch);
        for(j=code[i].start;j<n;j++)
            printf("%c ",code[i].bits[j]);
        printf("\n");
    }
}

void calculateWeight(hufmtree tree[])
{
    int i,j;
    int a,flag;
```

```
    printf("请输入一串字符:\n");
    gets(string);
    length=strlen(string);//获得字符串的长度
    printf("\n字符串长度为: %d\n", length);

    a=0;
    for(i=0;i<length;i++)   {
        flag=0;

        for(j=0;j<a;j++) {
            if(string[i]==Char[j].data) {
                flag=1;
                Char[j].count++;
                break;
            }
        }

        if(!flag)   {
            Char[a].data=string[i];
            Char[a].count++;
            a++;
        }
    }
    number=a;
    for(i=0;i<a;i++) {
        tree[i].weight = Char[i].count;
        tree[i].ch = Char[i].data;
        }
    getchar();
}
```

```
void huffman(hufmtree tree[])//建立哈夫曼树
{
    int i,j,p1,p2;//p1,p2分别记住每次合并时权值最小和次小的两个根结点的
下标
    float small1,small2,f;
    char c;
    for(i=0;i<m;i++)    //初始化
    {
        tree[i].parent=0;
        tree[i].lchild=-1;
        tree[i].rchild=-1;
        tree[i].weight=0.0;
    }
    calculateWeight(tree);


    for(i=n;i<m;i++)      //进行n-1次合并，产生n-1个新结点
    {
        p1=0;p2=0;
        small1=maxval;small2=maxval;   //maxval是float类型的最大值
        for(j=0;j<i;j++)    //选出两个权值最小的根结点
            if(tree[j].parent==0)
                if(tree[j].weight<small1)
                {
                    small2=small1;  //改变最小权、次小权及对应的位置
                    small1=tree[j].weight;
                    p2=p1;
                    p1=j;
                }
                else
                    if(tree[j].weight<small2)
                    {
                        small2=tree[j].weight;  //改变次小权及位置
```

```
                    p2=j;
                }
        tree[p1].parent=i;
        tree[p2].parent=i;
        tree[i].lchild=p1;  //最小权根结点是新结点的左孩子
        tree[i].rchild=p2;  //次小权根结点是新结点的右孩子
        tree[i].weight=tree[p1].weight+tree[p2].weight;
    }
}//huffman

void huffmancode(codetype code[],hufmtree tree[])//根据哈夫曼树求出哈夫曼编码
//codetype code[]为求出的哈夫曼编码
//hufmtree tree[]为已知的哈夫曼树
{
    int i,c,p;
    codetype cd;   //缓冲变量
    for(i=0;i<n;i++)
    {
        cd.start=n;
        cd.ch=tree[i].ch;
        c=i;       //从叶结点出发向上回溯
        p=tree[i].parent;   //tree[p]是tree[i]的双亲
        while(p!=0)
        {
            cd.start--;
            if(tree[p].lchild==c)
                cd.bits[cd.start]='0';//tree[i]是左子树，生成代码'0'
            else
                cd.bits[cd.start]='1';//tree[i]是右子树，生成代码'1'
            c=p;
            p=tree[p].parent;
        }
```

```
        code[i]=cd;    //第i+1个字符的编码存入code[i]
    }
}//huffmancode
```

4. 测试结果