**《数据结构》**

**课程设计报告**

|  |  |
| --- | --- |
| 课程名称： | 《数据结构》课程设计 |
| 课程设计题目： | 哈夫曼树 |
| 姓 名： | 陈剑 |
| 院 系： | 计算机学院 |
| 专 业： | 软件工程 |
| 班 级： | 19052714 |
| 学 号： | 19051711 |
| 指导教师： | 张灵均 |

2020年12月8日

1. **需求分析**

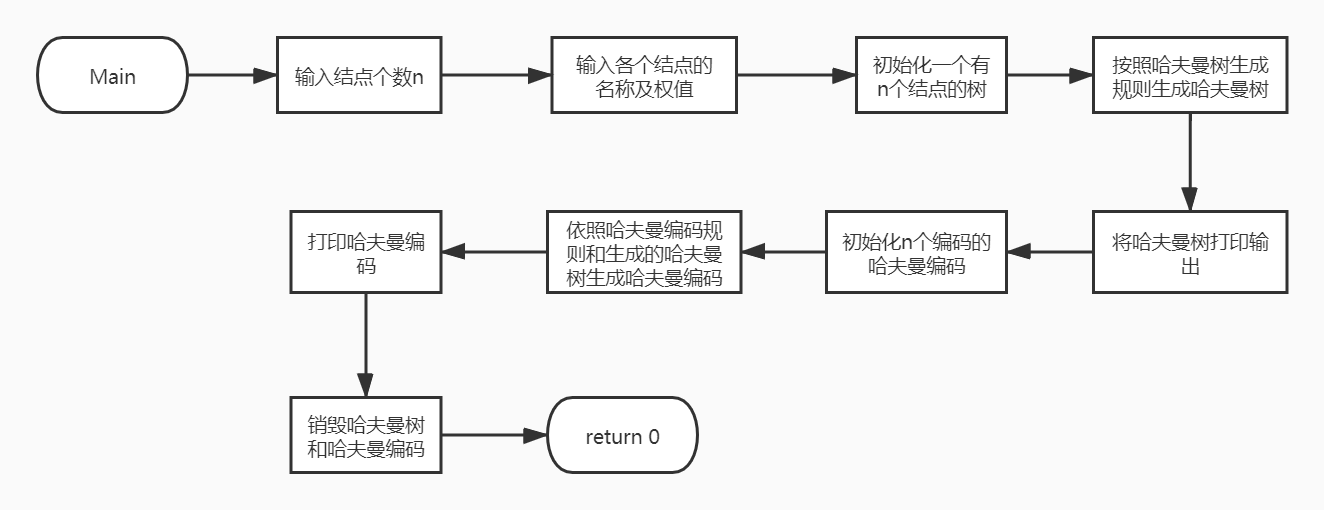
#### 功能需求：

模拟哈夫曼树及其哈夫曼编码的生成过程

#### 界面需求：

打印出哈夫曼树的左子树、右子树、双亲、权值，以及每一个结点的哈夫曼编码

1. **概要设计**



#### 接口设计

void InitHuffmantree(Huffmantree &HT, int n) //初始化哈夫曼树

void DestroyHuffmantree(Huffmantree &HT) //销毁哈夫曼树

int MinVal(Huffmantree &HT, int i) //在前i个结点找到双亲为0且权值weight最小的结点编号

void Select(Huffmantree &HT, int i, int &s1, int &s2) //找到两个weight值最小的结点中编号小的结点

void CreateHaffmantree(Huffmantree &HT, int n, char ch[], int weight[]) //根据字符表和权值，建立哈夫曼树

void InitHuffmanCoder(HuffmanCoder &HC, int n) //初始化编码

void DestroyHuffmanCoder(HuffmanCoder &HC) //销毁哈夫曼编码

void CreateHuffmanCoder(HuffmanCoder &HC, Huffmantree &HT) //建立n个字符的哈夫曼编码

**界面设计**

void Display(Huffmantree &HT) //打印哈夫曼树

#### 数据结构设计

    char ch[256]; //存储结点名称

    int weight[256]; //存储各个结点的权值

1. **详细设计**

#ifndef \_Huffmantree\_

#define \_Huffmantree\_

#include <iostream>

#include <fstream>

#include <iomanip>

using namespace std;

#define max 10000

//哈夫曼树结点结构

struct HufftreeNode

{

    char ch;

    int weight, parent, lchild, rchild;

};

//哈夫曼树结构

struct Huffmantree

{

    HufftreeNode \*ht; //静态树地址

    int htsize;       //树的结点个数

};

//初始化哈夫曼树

void InitHuffmantree(Huffmantree &HT, int n)

{

    HT.ht = new HufftreeNode[2 \* n - 1];

    HT.htsize = 2 \* n - 1;

}

//销毁哈夫曼树

void DestroyHuffmantree(Huffmantree &HT)

{

    delete[] HT.ht;

    HT.htsize = 0;

}

//在前i个结点找到双亲为0且权值weight最小的结点编号

int MinVal(Huffmantree &HT, int i)

{

    int j, k, min = max;

    for (j = 0; j < i; j++)

        if (HT.ht[j].parent == -1 && HT.ht[j].weight < min)

        {

            min = HT.ht[j].weight;

            k = j;

        }

    HT.ht[k].parent = max;

    return k;

}

//找到两个weight值最小的结点中编号小的结点

void Select(Huffmantree &HT, int i, int &s1, int &s2)

{

    int s;

    s1 = MinVal(HT, i);

    s2 = MinVal(HT, i);

    if (s1 > s2)

    {

        s = s1;

        s1 = s2;

        s2 = s;

    }

}

//根据字符表和权值，建立哈夫曼树

void CreateHaffmantree(Huffmantree &HT, int n, char ch[], int weight[])

{

    int i, s1, s2;

    if (n > 1)

    {

        for (i = 0; i < n; i++)

        {

            HT.ht[i].ch = ch[i];

            HT.ht[i].weight = weight[i];

            HT.ht[i].parent = -1;

            HT.ht[i].lchild = -1;

            HT.ht[i].rchild = -1;

        }

        for (; i < HT.htsize; ++i)

        {

            Select(HT, i, s1, s2);

            HT.ht[s1].parent = HT.ht[s2].parent = i;

            HT.ht[i].lchild = s1;

            HT.ht[i].rchild = s2;

            HT.ht[i].weight = HT.ht[s1].weight + HT.ht[s2].weight;

            HT.ht[i].parent = -1;

            HT.ht[i].ch = ' ';

        }

    }

    cout << "哈夫曼树建立完毕!" << endl;

}

//显示哈夫曼树

void Display(Huffmantree &HT)

{

    int i;

    cout << "所建哈夫曼树的静态链表表示如下:" << endl

         << endl;

    cout << "下标位置"

         << "  字符"

         << "  权值  "

         << " 左孩子 "

         << " 右孩子 "

         << " 双亲 " << endl;

    for (i = 0; i < HT.htsize; i++)

    {

        cout << setw(6) << i << setw(6) << HT.ht[i].ch << setw(8) << HT.ht[i].weight << setw(9) << HT.ht[i].lchild << setw(8) << HT.ht[i].rchild << setw(6) << HT.ht[i].parent << endl;

    }

    cout << endl;

}

//哈夫曼编码结构

struct HCnode

{

    char ch;

    char \*pstring;

};

//编码表结构

struct HuffmanCoder

{

    HCnode \*hc;

    int hcsize;

};

//初始化编码

void InitHuffmanCoder(HuffmanCoder &HC, int n)

{

    HC.hc = new HCnode[n];

    HC.hcsize = n;

}

//销毁哈夫曼编码

void DestroyHuffmanCoder(HuffmanCoder &HC)

{

    for (int i = 0; i < HC.hcsize; i++)

        delete[] HC.hc[i].pstring;

    delete[] HC.hc;

}

//建立n个字符的哈夫曼编码

void CreateHuffmanCoder(HuffmanCoder &HC, Huffmantree &HT)

{

    int i, j, c, f, start;

    char \*cd = new char[HC.hcsize];

    cd[HC.hcsize - 1] = '\0';

    cout << "各个字符的哈夫曼编码为：" << endl

         << endl;

    for (i = 0; i < HC.hcsize; i++)

    {

        start = HC.hcsize - 1; //编码结束符的位置

        HC.hc[i].ch = HT.ht[i].ch;

        //从叶子结点开始逆向求编码

        for (c = i, f = HT.ht[i].parent; f != -1; c = f, f = HT.ht[f].parent)

        {

            if (HT.ht[f].lchild == c)

            {

                cd[--start] = '0';

            }

            else

            {

                cd[--start] = '1';

            }

        }

        HC.hc[i].pstring = new char[HC.hcsize - start];

        cout << "第" << i + 1 << "个字符" << HT.ht[i + 1].ch << "编码是： ";

        for (j = start; j < HC.hcsize; j++)

        {

            cout << cd[j];

            HC.hc[i].pstring[j - start] = cd[j];

        }

        cout << endl;

    }

    cout << endl;

    delete[] cd;

}

#endif

#include<iostream>

#include<iomanip>

#include"Haffmantree.h"

using namespace std;

int main(){

    int n;

    cout << "请输入树叶结点的个数：" << endl;

    cin >> n;

    char ch[256];

    int weight[256];

    for (int i = 0; i < n;i++){

        cout << "请输入第" << i + 1 << "个字符及权值:";

        cin >> ch[i] >> weight[i];

    }

    Huffmantree HT;

    InitHuffmantree(HT, n);

    CreateHaffmantree(HT, n, ch, weight);

    Display(HT);

    HuffmanCoder HC;

    InitHuffmanCoder(HC, n);

    CreateHuffmanCoder(HC, HT);

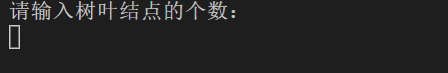
    DestroyHuffmantree(HT);

    system("pause");

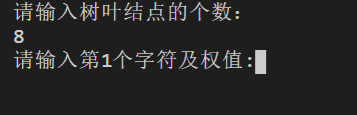
    return 0;

}

1. **调试分析**
2. 本次程序要实现的是哈夫曼树及其对应的哈夫曼编码的生成，核心是用先序和中序遍历的方法生成哈夫曼树，再依照哈夫曼树来实现哈夫曼编码。
3. 本程序的一个难点在于如何运用哈夫曼树的生成规律来生成哈夫曼树。如何利用哈夫曼树来生成哈夫曼编码。
4. **用户手册**
5. 本程序的执行文件为：HuffmanTree.exe
6. 进入演示程序后，将显示如下的界面



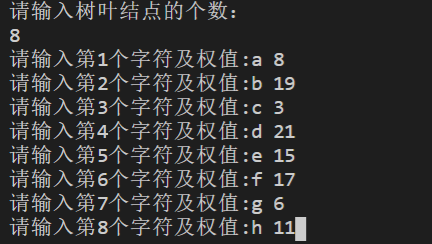
输入叶子结点个数后将显示如下界面：



在输入各个字符的名称及权值后，将生成对应的哈夫曼树及哈夫曼编码

1. **测试结果**

下面演示哈夫曼树的程序实现：





PS：详细的运行结果请打开HuffmanTree.exe文件

1. **附录**

源程序文件名清单：HuffmanTree.cpp,HuffmanTree.exe,Haffmantree.h