

**数据结构课程实验报告**

**（3）**

**姓名：王秋实**

**学号：2135070824**

**专业：计科**

**完成日期：2023/4/5**

**目录**

[目录 - 1 -](#_Toc434170429)

[1 设计要求 - 2 -](#_Toc434170430)

[2 程序功能框图 - 2 -](#_Toc434170431)

[3 数据结构说明 - 2 -](#_Toc434170432)

[4 重要算法核心代码 - 2 -](#_Toc434170433)

[5 测试运行界面 - 2 -](#_Toc434170434)

[6 完整源程序 - 3 -](#_Toc434170435)

**数据结构实验三（二叉树）**

**<“哈夫曼树”的设计与实现> 设计要求**

【问题描述】

假设有一段电文由字符集 {A, B, C, D, E, F, G, H} 中的字符组成，各字符在电文中出现的频率由对应次数集 {5，29，7，8，14，23，3，11} 中的数字表示，请设计各字符的哈夫曼编码。

【基本要求】

应包含以下几方面的功能：

1. 设计哈夫曼树。具体构造方法如下：以字符集{A, B, C, D, E, F, G, H} 中的字符作为叶子结点，以各字符在次数集 {5，29，7，8，14，23，3，11} 中对应的次数作为各叶子结点的权值构造一棵哈夫曼树。
2. 设计哈夫曼编码。按照构造出来的哈夫曼树，规定哈夫曼树的左分支为0，右分支为1，则从根结点到每个叶子结点所经过的分支对应的0和1组成的序列便为该结点对应字符的哈夫曼编码。
3. 依次求出每个字符的哈夫曼编码并输出。

【扩展功能】

从键盘上分别输入哈夫曼编码字符的个数以及每个字符对应的权值，程序执行中请一步一步依次显示出哈夫曼树的构造过程，最后输出每个权值对应的哈夫曼编码。

【实现方法】

选择合适的存储结构，要便于从叶子结点找到双亲结点。

**2、程序功能框图：**

输入格式：第一行一个整数n，接下来的i行输入一个字符Ci和一个整数Wi，表示Ci这个字符的权重是Wi

样例输入：

8

A 5

B 29

C 7

D 8

E 14

F 23

G 3

H 11

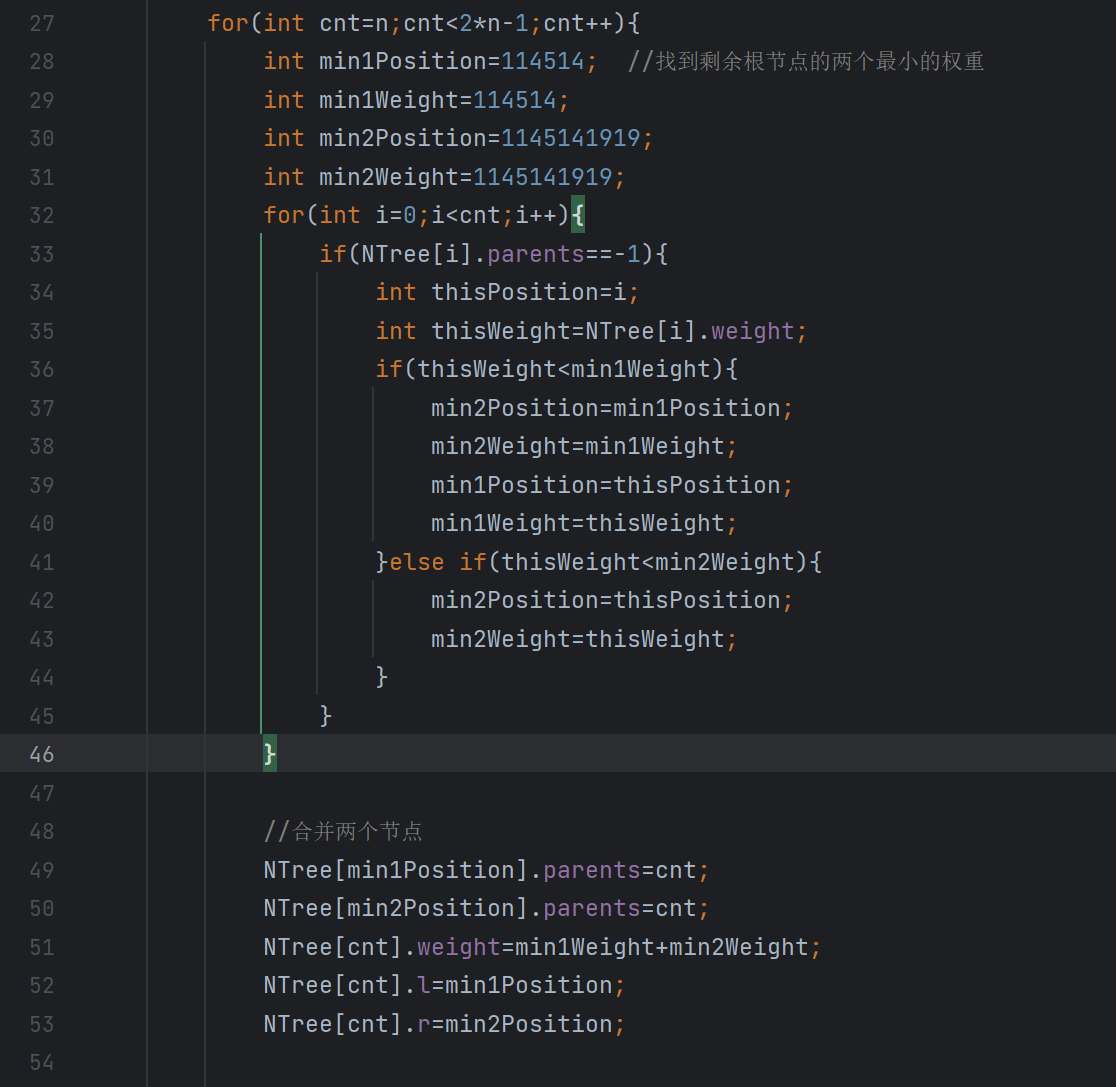
输出格式：第一部分输出每一步剩余结点的权重，第二部分输出字符和对应的哈夫曼编码

**3、数据结构说明**

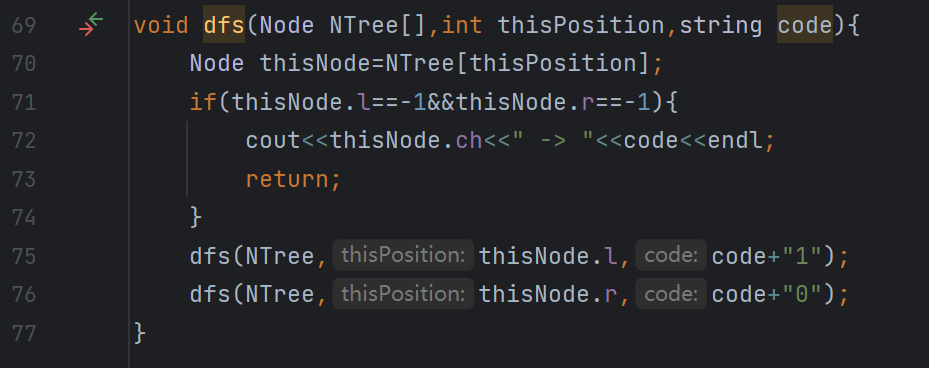
利用贪心算法，每次选择权重最小的两个结点合并，从而构造一棵权重最小的二叉树，然后从根节点开始，左孩子编码为1，右孩子编码为0，叶子结点是包含字符的结点，此时的编码即是该字符的哈夫曼编码。

1. **重要算法核心代码**

构造哈夫曼树：



先找到剩余结点的两个最小权重的结点，然后将它们合并。



dfs前序遍历整个哈夫曼树，碰到叶结点返回该字符的哈夫曼编码

1. **测试运行界面：**

样例输入：

8

A 5

B 29

C 7

D 8

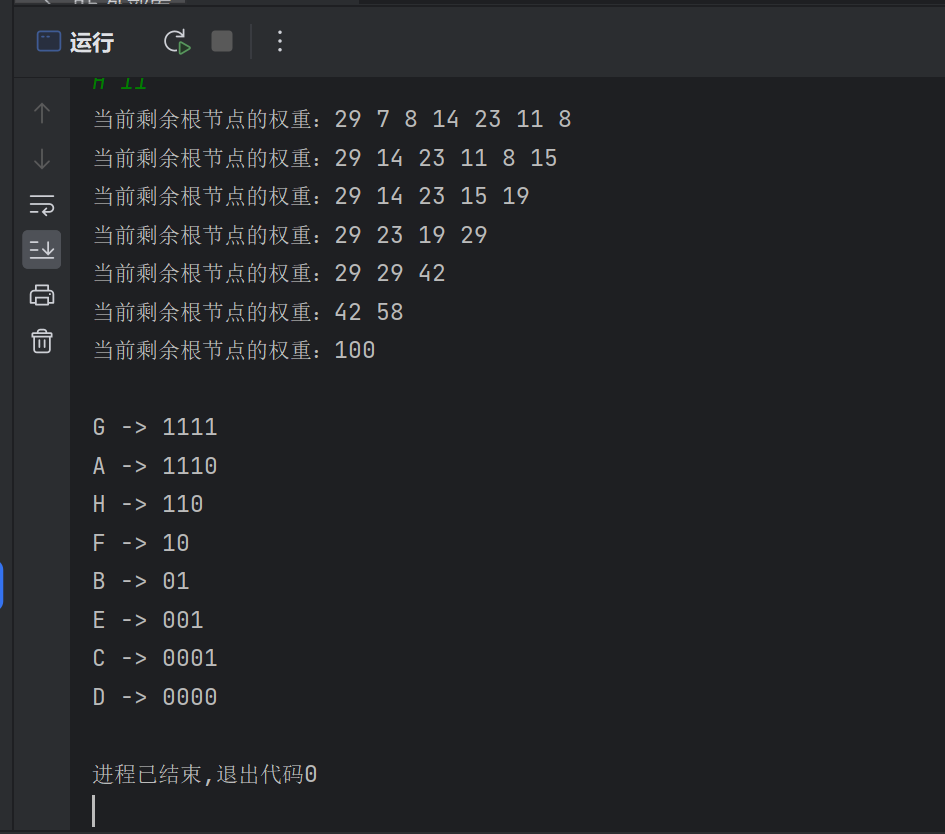
E 14

F 23

G 3

H 11

样例输出：



1. **完整源程序：**

#include "iostream"

using namespace std;

struct Node{

int id=-1;

int weight;

char ch;

int parents=-1;

int l=-1;

int r=-1;

};

void dfs(Node NTree[],int thisNode,string code);

int main(){

int n;

cin>>n;

Node NTree[2\*n-1];

for(int i=0;i<n;i++){

char thisChar;

int weight;

cin>>thisChar>>weight;

NTree[i].id=i;

NTree[i].ch=thisChar;

NTree[i].weight=weight;

}

for(int cnt=n;cnt<2\*n-1;cnt++){

int min1Position=114514; //找到剩余根节点的两个最小的权重

int min1Weight=114514;

int min2Position=1145141919;

int min2Weight=1145141919;

for(int i=0;i<cnt;i++){

if(NTree[i].parents==-1){

int thisPosition=i;

int thisWeight=NTree[i].weight;

if(thisWeight<min1Weight){

min2Position=min1Position;

min2Weight=min1Weight;

min1Position=thisPosition;

min1Weight=thisWeight;

}else if(thisWeight<min2Weight){

min2Position=thisPosition;

min2Weight=thisWeight;

}

}

}

//合并两个节点

NTree[min1Position].parents=cnt;

NTree[min2Position].parents=cnt;

NTree[cnt].weight=min1Weight+min2Weight;

NTree[cnt].l=min1Position;

NTree[cnt].r=min2Position;

cout<<"当前剩余根节点的权重：";

for(int i=0;i<=cnt;i++){

if(NTree[i].parents==-1){

cout<<NTree[i].weight<<" ";

}

}

cout<<endl;

}

cout<<endl;

dfs(NTree,2\*n-2,"");

}

void dfs(Node NTree[],int thisPosition,string code){

Node thisNode=NTree[thisPosition];

if(thisNode.l==-1&&thisNode.r==-1){

cout<<thisNode.ch<<" -> "<<code<<endl;

return;

}

dfs(NTree,thisNode.l,code+"1");

dfs(NTree,thisNode.r,code+"0");

}