离散数学

项目说明文档

**最优二元树的应用**

同济大学

Tongji University

姓名： 林觉凯

学号： 2253744

指导老师： 李冰

学院专业： 软件学院 软件工程

**1.实验背景介绍**

输入一组通信符号的使用频率，求各通信符号对应的前缀码。在通信编码中，最优二元树经常用于构建哈夫曼编码。哈夫曼编码是一种变长编码，它根据数据的频率来分配更短的编码给出现频率较高的字符，从而实现数据的压缩。

**2.实验要求**

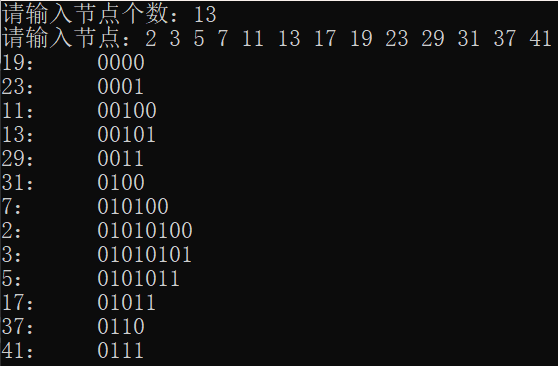
**2.1 输入要求**

要求用户输入结点的个数和对应的权值。

**2.2 输出要求**

打印出通过构造哈夫曼树生产的通信编码序列。

**2.3 项目示例**



1. **数据结构设计**

最优二元树（也称为哈夫曼树）在通信编码中有广泛的应用。它是一种用于数据压缩的技术，可以将数据转换为更短的编码，以达到减少存储空间和传输时间的目的本程序采用哈夫曼树这一数据结构，构造了两个类：哈夫曼树结点类（Treenode）和哈夫曼树类（Huffmantree），两个类之间的耦合关系可以采用嵌套、继承等多种关系。本程序也构造了这两个基本的哈夫曼树类来进行操作。

**4.项目实现**

**4.1输入部分的实现**

输入部分我额外写了一个函数用来检查错误的输入并给出提示。在输入结点个数和结点权值的时候，考虑到了输入的数据类型错误和输入的数据数值大小范围是否在符合的范围内。

int Get\_int(int Min = INT\_MAX, int Max = INT\_MIN, string errortips = "")

{

int retint;

while (1)

{

cin >> retint;

if (cin.good() == 0 || retint < Min || retint > Max)

{

cin.clear();

cin.ignore(1024, '\n');

cout << errortips << endl;

continue;

}

break;

}

return retint;

}

**4.2 哈夫曼树构建的实现**

哈夫曼树的构建是本题的难点，主要是要一直去找当前叶子结点中的最小权值并不断循环下去。leaf是一个二维数组的指针，第一层循环n-1次，每次循环都要在叶子结点数组中寻找权重最小的两个结点，如果找到了最小的结点，那就把第二小的值赋值为原先最小的值，将最小的值赋值为该节点的权重；如果找到了第二小的结点，那就直接重置第二小的权值即可。一轮循环之后，将找到的最小和第二小的结点相加构建出新的父节点，原先的两个结点其中一个指向父节点，另外一个置为NULL，NULL的标志就是之后循环不再遍历到此节点。所以最后其他结点都为NULL，唯一一个不为NULL的是父结点，将根结点最后指向这个父节点即可以完成了哈夫曼树的构建。

Huffmantree::Huffmantree(int\* Array,int number)

{

root = NULL;

this->number = number;

leaf = new Treenode \* [number]; //为叶子数组动态分配空间

for (int i = 0; i < number; i++)

leaf[i] = new Treenode(Array[i]);

for (int i = 0; i < number - 1; i++) //找两个权重最小的叶子节点

{

int min1 = INT\_MAX, min2 = INT\_MAX;

int index1 = -1, index2 = -1;

for (int j = 0; j < number; j++)

{

if (leaf[j] != NULL)

{

if (leaf[j]->weight < min1) //找到最小权值

{

min2 = min1;

index2 = index1;

min1 = leaf[j]->weight;

index1 = j;

}

else if (leaf[j]->weight < min2) //找到第二小权值

{

min2 = leaf[j]->weight;

index2 = j;

}

}

}

Treenode\* newnode = new Treenode(min1 + min2, NULL, leaf[index1], leaf[index2]);

//构造新的哈夫曼树结点

leaf[index1]->parent = newnode; //指向父节点

leaf[index2]->parent = newnode; //指向父节点

leaf[index1] = newnode; //使1结点重置为新节点（父节点）

leaf[index2] = NULL; //另外一个为NULL

}

for (int i = 0; i < number; i++)//所以到最后其他结点都为NULL，唯一一个不为NULL的是根结点

{

if (leaf[i] != NULL)

{

root = leaf[i];

break;

}

}

}

**4.3打印通信编码的实现**

本程序利用递归打印通信编码，首先找到最底层（没有左右孩子的结点），然后一直向上找父节点，这里要注意，如果这个结点是其父节点的左孩子，每往上走一层，通信编码前缀需要加一个“0”；如果这个结点是其父节点的右孩子，每往上走一层，通信编码前缀需要加一个“1”。最后再打印出通信编码即可。

void Huffmantree::Printcode(Treenode\* node)

{

if (node == NULL)

return;

if (node->left == NULL && node->right == NULL)

{

cout << node->weight << "：\t";

Treenode\* current = node;

string code = "";

while (current->parent != NULL) //不断向上找父节点

{

if (current->parent->left == current)

code = "0" + code; //如果这个节点是其父节点的左孩子，则加“0”

else

code = "1" + code; //如果这个节点是其父节点的右孩子，则加“1”

current = current->parent;

}

code = "0" + code;

cout << code << endl;

return;

}

else

{

Printcode(node->left);

Printcode(node->right);

}

}

**5.实验心得**

本次项目和上次的最小生成树项目相对于之前几次项目来说较为复杂。在此次项目的完成过程中，主要是构建哈夫曼树和打印通信编码这两个重要的部分。首先在哈夫曼树构建的过程中需要考虑比较个权值的大小并且创建新的树节点；在打印通信编码的过程中我们需要考虑打印当前的结点是其父节点的左孩子还是右孩子，根据不同输出不一样的效果。通过此次项目，我从原理上更加深入地理解了哈夫曼树的创建过程，同时造价的代码编写思路上也有了一定的提升。

附件(源代码)：

#include <iostream>

#include <cstring>

#include <climits>

using namespace std;

//代码输入的健壮性保证

int Get\_int(int Min = INT\_MAX, int Max = INT\_MIN, string errortips = "")

{

int retint;

while (1)

{

cin >> retint;

if (cin.good() == 0 || retint < Min || retint > Max)

{

cin.clear();

cin.ignore(1024, '\n');

cout << errortips << endl;

continue;

}

break;

}

return retint;

}

//哈夫曼树节点类

class Treenode

{

public:

int weight; //权重

Treenode\* left; //左指针

Treenode\* right; //右指针

Treenode\* parent; //指向父节点的指针

Treenode(int Number, Treenode\* Parent = NULL, Treenode\* Left = NULL, Treenode\* Right = NULL)

{

weight = Number;

left = Left;

right = Right;

parent = Parent;

}

};

//哈夫曼树类

class Huffmantree

{

public:

Huffmantree(int\* Array,int number); //构造函数

~Huffmantree() {}; //默认析构函数

void Printcode(Treenode\* node); //打印通信码

Treenode\* root;

Treenode\*\* leaf;

int number;

};

//构造函数

Huffmantree::Huffmantree(int\* Array,int number)

{

root = NULL;

this->number = number;

leaf = new Treenode \* [number]; //为叶子数组动态分配空间

for (int i = 0; i < number; i++)

leaf[i] = new Treenode(Array[i]);

for (int i = 0; i < number - 1; i++) //找两个权重最小的叶子节点

{

int min1 = INT\_MAX, min2 = INT\_MAX;

int index1 = -1, index2 = -1;

for (int j = 0; j < number; j++)

{

if (leaf[j] != NULL)

{

if (leaf[j]->weight < min1)

//找到最小权值 {

min2 = min1;

index2 = index1;

min1 = leaf[j]->weight;

index1 = j;

}

else if (leaf[j]->weight < min2) //找到第二小的权值

{

min2 = leaf[j]->weight;

index2 = j;

}

}

}

Treenode\* newnode = new Treenode(min1 + min2, NULL, leaf[index1], leaf[index2]); //构造新的哈夫曼树结点

leaf[index1]->parent = newnode; //指向父节点

leaf[index2]->parent = newnode; //指向父节点

leaf[index1] = newnode; //使1结点重置为新节点（父节点）

leaf[index2] = NULL; //另外一个为NULL

}

for (int i = 0; i < number; i++) //所以到最后其他结点都为NULL，唯一一个不为NULL的是根结点

{

if (leaf[i] != NULL)

{

root = leaf[i];

break;

}

}

}

//打印通信编码

void Huffmantree::Printcode(Treenode\* node)

{

if (node == NULL)

return;

if (node->left == NULL && node->right == NULL)

{

cout << node->weight << "：\t";

Treenode\* current = node;

string code = "";

while (current->parent != NULL) //不断向上找父节点

{

if (current->parent->left == current)

code = "0" + code; //如果这个节点是其父节点的左孩子，则加“0”

else

code = "1" + code; //如果这个节点是其父节点的右孩子，则加“1”

current = current->parent;

}

code = "0" + code;

cout << code << endl;

return;

}

else

{

Printcode(node->left);

Printcode(node->right);

}

}

int main()

{

cout << "请输入节点个数：";

int number = Get\_int(1, INT\_MAX, "输入错误！请重新输入！"); //输入节点个数

int\* Array = new int[number];

cout << "请输入节点：";

for (int i = 0; i < number; i++)

Array[i] = Get\_int(1, INT\_MAX, "输入错误！请重新输入！"); //输入节点权重

Huffmantree Coding\_tree(Array, number); //实例化哈夫曼树Coding\_tree

Coding\_tree.Printcode(Coding\_tree.root);

delete[] Array;

return 0;

}