《离散数学》课程实验报告

5-最优 2 元树的应用

一、题目背景与简介

在普通树中，一个结点到另一个结点之间的分支为这两个结点间的路径，而路径长度指的是这条路径上的分支数目。一般不带权时每个路径长度默认为 1，所以结点数为 n 的树路径长度为 n-1（树根到每一结点的路径之和）。

在许多应用中，常常将一个有某种意义的实数赋值给某个结点，称此实数就是该结点的权，于是就有了带权路径。结点的带权路径长度=根结点到该结点之间的路径长度与该结点权的乘积，树中所有叶子的带权路径长度之和即为该树的带权路径长度。

而哈夫曼（Huffman）树又称最优二叉（搜索）树，作为一种带权路径的树，是带权路径长度最短的二叉树，在很多现实问题中得到了广泛的应用。其基本思想为：权值大的结点用短路径，权值小的结点用长路径。若对一棵哈夫曼树中的每个左分支赋 0，右分支赋 1，则从根到每个叶子的路径上形成了一个 0-1 串，该二进制串就称为哈夫曼编码。

本题要求利用最优二元树（哈夫曼树）解决通信编码：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。

二、原理与方法

（1）用一维数组 frequency[N]存储通信符号的使用频率，用求最优 2 元树的方法求出每个通信符号的前缀码：通过结点存放数据（即权值），再根据树的遍历方法求出对应节点的哈夫曼编码。

（2）用链表保存最优 2 元树，输出前缀码时可以用树的遍历方法。

三、解题思路与核心算法

3.1 涉及的主要变量与数据结构

结构体 tree 存储该结点的左指针 Left、右指针 Right 以及该结点本身的值 value，并命名一个 fp[N]的指针以保存结点。另外，前缀码的存放通过 bool 型一维数组 code[2*N]（因为编码由 0、1 组成，所以定为 bool 型以减少内存占用，而大小 2*N 是因为 N 个结点最终构造出的哈夫曼树一共有 2*N 结点），用一维数组 frequecy[N]存贮通信符号的使用频率，int 型变量 node_num 存储节点个数。

3.2 哈夫曼树的创建——哈夫曼算法

用函数 create_Huffman 创建哈夫曼树：

（1）由给定的 n 个权值$\{W_1, W_2, \cdots\cdots W_n\}$，构造具有 n 棵二叉树的集合 F=$\{T_1, T_2, \cdots\cdots T_n\}$，其中每棵二叉树$T_i$中只有一个带权为$W_i$的根结点。

（2）在 F 中选取两棵根权值最小的树作为左、右子树构造一棵新的二叉树，且该二叉树根结点的权值为其左、右子树上根结点的权值之和。

（3）在 F 中删除这两棵权值最小的树，同时将新得到的二叉树加入 F 中。

（4）重复步骤 2 和 3，直到 F 只含一棵树为止，此时 F 即为哈夫曼树。

3.3 前序遍历存储前缀码

函数 preorder 进行递归式的前序遍历，存储前缀码。由根遍历到左子树，再遍历右子树，根据前序遍历的结果可知第一个访问的必定是 root 结点。这里需要通过 bool 型变量 judge 来判断正在遍历的是左/右子树，从而决定该位置的前缀码是 0。在函数体中，该位置的前缀码始终为 judge 的值。

3.4 错误输入的处理

本题涉及到的输入有节点个数的输入和结点权值的输入。对于结点个数，是一个 int 型

变量，操作起来比较方面，当 cin.fail()或 node_num<0 时，清空缓冲区，重新输入。而结点权值的输入就有一些复杂，因为是在循环体内部依次进行输入，所以每当输入错了一个值，就得将循环控制变量 i 置 0，从 frequency 数组的头部重新输入。

四、代码与运行结果

```
/*2151133 孙韩雅*/
#include <iostream>
#include <iomanip>
#include <stdio.h>
#include <stdlib.h>
using namespace std;

#define N 13//宏定义结点个数
#define MAX_F 80//宏定义使用频率数组的大小
struct tree {
	int value = 0;
	tree* Left =NULL;//左节点
	tree* Right =NULL;//右节点
}*fp[N];	//保存结点
bool code[N];  //放前缀码

void init_leaf(int f[], int n) {//生成叶子节点
	tree* leaf;
	for (int i = 0; i < n; i++) {
		leaf = (tree*)malloc(sizeof(tree));  //生成叶子节点
		if (leaf == NULL) {//确保空间开辟成功
			cout << "生成叶子结点失败" << endl;
			return;
		}
		leaf->value = f[i];
		leaf->Left = NULL;
		leaf->Right = NULL;
		fp[i] = leaf;		//空间开辟成功则将node 存入 fp[]中
	}
}
void sort(tree* array[], int n) {//将第 N-n 个结点插入到已排好序的序列中
	tree* tmp;
	for (int i = N - n; i < N - 1; i++) {
		if (array[i]->value > array[i + 1]->value) {//序列从小到大进行排序
			tmp = array[i + 1];
			array[i + 1] = array[i];
			array[i] = tmp;
		}
	}
}
tree* create_Huffman(int f[], int n) {//建立树
	tree* parent;
	for (int i = 1; i < N; i++) {
		parent = (tree*)malloc(sizeof(tree));  //生成非叶子结点（父结点）
		if (parent == NULL) {//保证空间开辟成功
			cout << "生成非叶子结点失败" << endl;
			return NULL;
		}
		parent->value = fp[i - 1]->value + fp[i]->value;//最小的两个数上添一个父结点，其值为两数之和
		parent->Left = fp[i - 1];
		parent->Right = fp[i];
		fp[i] = parent;  //w1+w2
		sort(fp, N - i);//将此时第 i 个点进行排序
	}
	return fp[N - 1];
}
void preorder(tree* p, int k, bool judge/*用以判别左右子树*/) {//递归式前序遍历求哈弗曼编码
	if (p != NULL) {
		code[k] = judge;//judge 为 true 时，是右子树，标 1；否则为左子树，标 0
		if (p->Left == NULL) {//P 指向叶子
			cout << setw(6) << setiosflags(ios::left) << p->value << ": ";
			for (int j = 1; j <= k; j++) //从 j=1 开始打印，code[0]为多出的一位‘0’
				cout << code[j];
			cout << endl;
		}
		//递归遍历树
		preorder(p->Left, k + 1, false);
		preorder(p->Right, k + 1, true);
	}
}
```

```
int main() {
	int node_num = 0;
PART1:
	cout << "请输入结点个数(必须是正整数):";
	cin >> node_num;
	if (cin.fail() || node_num < 0) {
		cin.clear();
		cin.ignore(100, '\n');
		cout << "输入错误，请重新输入！" << endl << endl;
		goto PART1;
	}
	int frequency[MAX_F] = { 0 };
	cout << "请输入结点（以空格分隔）：" << endl;
	for (int i = 0; i < node_num; i++) {
		cin >> frequency[i];
		if (cin.fail()) {
			cout << "输入错误，请重新输入所有节点：" << endl;
			cin.clear();
			cin.ignore(100, '\n');
			i = 0;
		}
	}
	tree* head = new tree;//分配空间
	init_leaf(frequency, N);  //初始化结点
	head = create_Huffman(frequency, N);  //生成最优树
	code[0] = 0;
	preorder(head, 0, false);  //遍历树
	return 0;
}
```

运行结果：

```
请输入结点个数(必须是正整数):13
请输入结点（以空格分隔）：
2 3 5 7 11 13 17 19 23 29 31 37 41
19     : 000
23     : 001
11     : 0100
13     : 0101
29     : 011
31     : 100
7      : 10100
2      : 1010100
3      : 1010101
5      : 101011
17     : 1011
37     : 110
41     : 111
```

五、体会与心得

在哈夫曼树这块由于没跟上进度，我对其的了解可谓是极其浅薄，并不知道它应该如何应用。在看到本题的要求时，我的脑袋是空的，但还是硬着头皮跟着示例代码走了一遍，明白了大致的流程。在后期我对代码进行修改与完善的过程中，我对哈夫曼树以及哈夫曼算法有了彻底的了解，并且惊叹于这样编码的巧妙性。

在代码的编写过程中，我新增了几点心得。

第一个是初始化与分配空间的问题。示例代码通过使用 init_leaf()函数对结点进行初始化，是一个细节的操作，这就让我思维打开、放眼整个代码，在每一个声明语句中进行检查，把所有未初始化变量的声明语句都进行了修改，这样使得代码更加的健壮。另外，树的应用就不得不使我们对结点进行空间的分配，可以使用 new 或者 malloc，但都需要进行内存空间是否成功开辟的判断，做到了这一点，也同样增加了程序的健壮性和完备性。

第二个是要正确选择变量的类型。我在阅读示例代码时，发现有很多可以用 bool 型来定义的变量反而用了 int 型，或者与数字有关的变量用了 char 型。与实际意义不匹配的变量类型很大程度上会增加代码的运行量，降低代码的效率。所以我在此基础上对相关变量进行了修改。

第三个是每个代码要形成自己的风格。比如我会喜欢用英文全拼来定义变量，而非一个

字母，这样不仅增加了代码的可读性，而且自己在编写、更进的同时不会引起混淆。所以我对示例代码的变量名进行了大规模的修改，虽然工序繁复，但对后期的检查提供了很大的帮助，提高了效率。