****

**Tongji University**

**离散数学实验报告**

**005-最优二元树的应用**

专 业： 软 件 工 程

指导教师： 唐 剑 锋

学 号： 2 1 5 3 0 6 1

姓 名： 谢 嘉 麒

1. **实验概述**
   1. **实验背景**

对于一组通讯符号，假设字符集合是 { C, A, S, T }，各个字符出现的频度是 W＝{ 2, 7, 4, 5 }。若给每个字符以等长编码 A 00 T 10 C 01 S 11则总编码长度为 ( 2+7+4+5 ) \* 2 = 36。若按各个字符出现的概率不同而给予不等长编码，可减少总编码长度。

由此，为了减少总编码长度，需要引入最优二元树（霍夫曼树）对报文进行数据压缩。

* 1. **实验要求**

要求用C++语言实现求通信符号前缀码程序。

用户输入一组通信符号的使用频率，利用最优二元树求解各通信符号对应的前缀码，并分行打印出来。

* 1. **实验原理**

本次实验重点在于实现最优二元树求解通信符号前缀码。

1. 用一维数组 prefixCode[2 \* N]存贮通信符号的使用频率；
2. 用求最优二元树的方法求出每个通信符号的前缀码，存入霍夫曼树中；
3. 遍历霍夫曼树，分行打印前缀码。
   1. **实验思路与分析**

根据实验原理可以得到，本次实验需求主要在于霍夫曼树的构建，此外也需要考虑输入输出流和错误处理的内容。在使用递归遍历霍夫曼树时，需要尽可能降低时间复杂度，提高程序的执行效率。

* + 1. **完善项目功能**

考虑到用户的使用体验和需求优化，故可以在程序中增加适当的问候语句和是否进行多次试验的询问语句。具体可以用输入输出流结合while循环实现，方便用户使用。

* + 1. **保证执行效率**

保证用户体验的重要环节之一就是保证代码的执行效率。本次项目为了保证程序的运行效率以及避免不必要的空间浪费，使用递归方法时，要尽可能降低循环的次数，减小时间复杂度和空间复杂度。

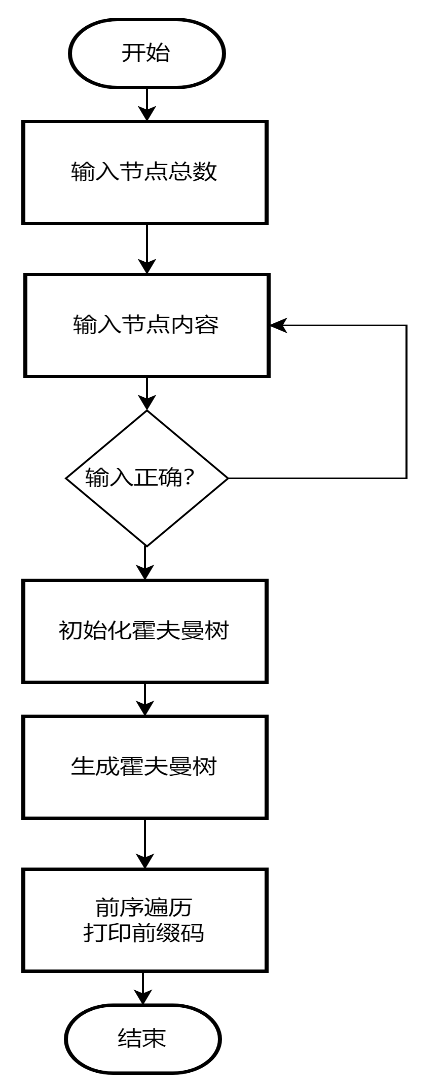
* + 1. **注重代码健壮性**

程序需要在用户输入非法字符或者不合理数据的时候进行报错提示，确保正确性，体现代码健壮性。

* + 1. **代码复用的处理**

通过增加可复用的函数来提高代码的复用能力，使整个程序简洁美观，减少重复率，降低风险，便于修改。

1. **程序设计总览**
   1. **程序流程图**

****

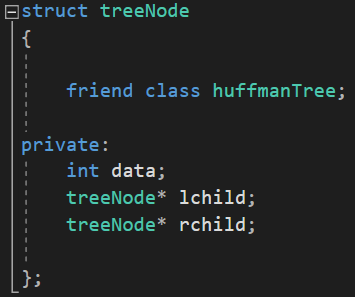
* 1. **数据结构的设计与使用**

本次实验使用的数据结构主要为树，而最优二元树则是在树的基础上考虑带权路径长度（Weight Path Length,WPL）的特殊处理。带有全职的叶结点叫做扩充二叉树的外结点，其余不带权值的分支结点叫做内结点。直观来看，带权路径长度最小的二叉树应是权值大的外结点离根结点最近的扩充二叉树，即Huffman树。

* + 1. **Huffman树类的设计**

为了使代码完整且具有封装的安全性，本次实验设计类作为最优二元树的数据结构载体。

1. treeNode结构体：
2. 定义treeNode结构体来作为树的基本结点单元；
3. 定义data变量存储结点内容；
4. 递归定义treeNode\* lchild和treeNode\* rchild作为左右孩子的指针；
5. 将huffmanTree类声明为友元类，方便其访问；



1. huffmanTree类：
2. 定义三个私有变量：int sumNode；int weights[N]；treeNode\* fp[N]；

分别存储节点总数；节点权重；霍夫曼树；

1. 定义一个私有函数：void sort(int n)；

作为排序算法辅助霍夫曼树的建立；

1. 定义五个公有函数：

用户输入节点权重：

void get\_weights();

用户输入节点总数：

void get\_sumNode();

霍夫曼树节点初始化：

void init();

建立霍夫曼树：

treeNode\* HuffmanTree();

前序遍历霍夫曼树并打印前缀码：

void preorder(treeNode\* p, int k, char c);

1. **核心算法的使用与分析**

为了构造权值集合为{w1,w2,…,wn}的Huffman树，需要采用Huffman算法，基本思路如下：

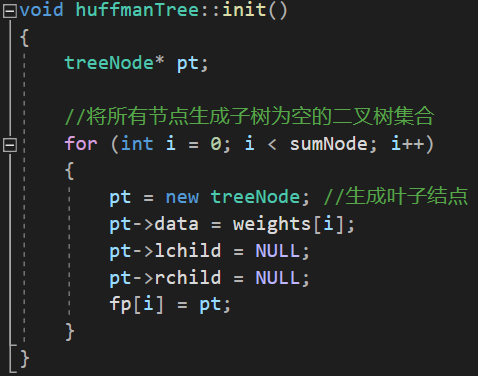
1. 根据给定的n个权值，构造具有n棵扩充二叉树的森林F={T1,T2,…,Tn}，其中每棵扩充二叉树Ti只有一个带权值wi的根结点，左右孩子树均为空。
2. 重复以下步骤，直到F中只剩下一棵树为止：
3. 在F中选取两棵根结点权值最小的扩充二叉树，作为左、右子树构造一棵新的二叉树。置新的二叉树的根结点的权值为其左右子树上根结点权值的和；
4. 在F中删除这两棵二叉树；
5. 把新的二叉树加入F；

最后得到的就是Huffman树。

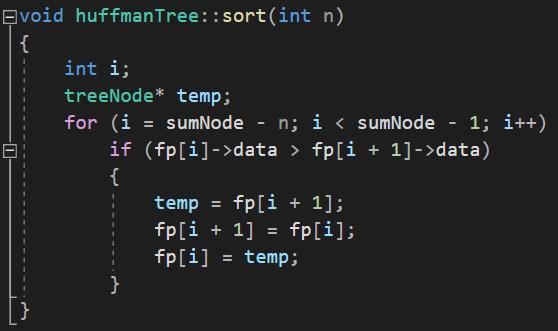
* + 1. **最优二元树的代码实现**

本实验将霍夫曼算法分为三部分，分别为：初始化霍夫曼树结点；霍夫曼树生成；排序算法。

1. 利用init()函数初始化霍夫曼树结点时，先定义treeNode\*指针，并为指针动态申请空间，将其作为根节点，左右孩子置为NULL。完成初始化后将指针存储在fp指针数组中留待后用。循环sumNode次，即完成所有结点的初始化。



1. 利用sort()函数对fp指针数组中的根结点权值进行比较和排序，利用中间变量temp完成排序，将较小权值的根结点上浮至指针数组顶端，为霍夫曼树的建立提供条件（类似于最小堆的思想）

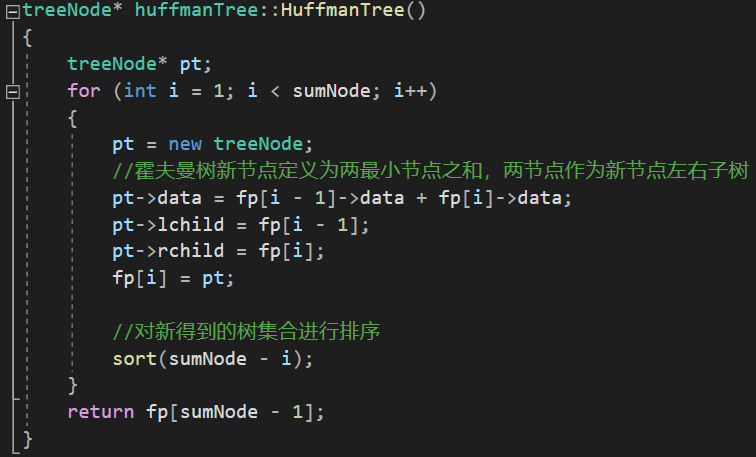


1. 设定treeNode\*作为函数HuffmanTree()的返回值，方便\*Node变量接收霍夫曼树首结点，为前序遍历函数提供参数；

在函数中声明treeNode\* pt作为新的根结点，将根结点权值最小的两棵子树（即指针数组fp顶端的两棵树）加入pt，作为pt的左右孩子，而pt->data则更新为两根结点权值的和；

将pt存入指针数组fp，调用排序函数，将权值最小的两树上浮；

返回fp[sumNode - 1]，递归生成Huffman树。

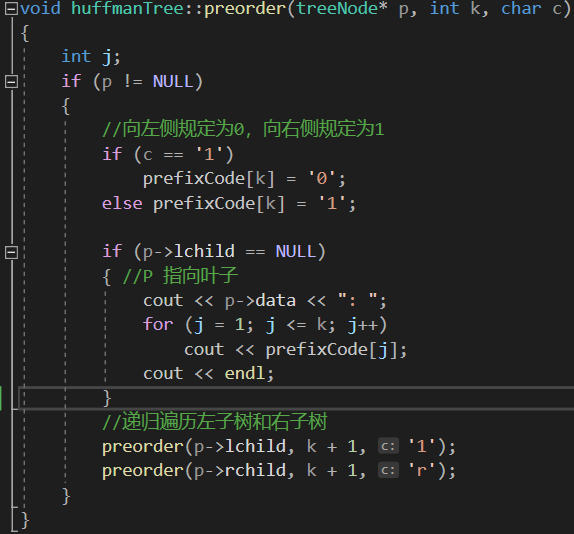


* + 1. **前序遍历打印前缀码**

树的遍历一般可以采用前序，中序和后序遍历，本实验考虑到前缀码需要从根结点进行读取，故选择采用前序遍历的方法，即：先访问根结点，在依次访问左右孩子，递归调用直到结束。

打印前缀码时，将向左递归行进视作“0”，将向右递归行进视作“1”。

递归前进到左子树为NULL时，则将存储在prefixCode中的前缀码都逐个打印出来，最后换行，继续递归。



1. **心得体会**
   1. **巩固练习相关基础知识**

本次实验主要巩固学习了最优二元树的相关知识，并且通过前缀码求解的实际应用进一步实现了将理论与实际相结合。

Huffman树最经典的应用在通信领域。经Huffman编码的信息消除了冗余数据，极大程度上提高了通信信道的传输效率。目前，Huffman树是数据压缩的重要方法。

由实验可以得出，设计变长编码时，为出现概率较高的字符指定较短的码字，而为出现概率较低的字符指定较长的码字，可明显地提高传输的平均性能。

因此，变长编码的效率高于定长编码。

* 1. **提高编程能力和数据结构的处理能力**

本次实验通过设计与使用Huffman树类强化了对于树结构的理解和应用，同时也学会了分步调试代码的方法。

实验中由于个别参数有误，如循环次数设置等导致了预料之外的错误，通过分步调试逐过程、逐语句来对程序进行修正，提高了Debug的能力。

1. **程序测试**

测试用例：

1. 节点个数：13
2. 节点 ：2 3 5 7 11 13 17 19 23 29 31 37 41

预期结果：

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

