武汉纺织大学

数学与计算机学院

数据结构实验报告

2019 ~2020学年第一学期

|  |  |
| --- | --- |
| 专 业 班 级 | 物联网二班 |
| 学生学号 | 1802220113 |
| 学生姓名 | 梁少伟 |
| 任 课 教 师 | 叶鹏 |
| 实验指导教师 | 叶鹏 |
| 实验地点 | 实训楼 |
| 填写时间 | 2109/12/6 |
|  | |

**填写说明**

1. 报告需要学生填写的内容包括：实验要求、完整的实验结果记录、实验结果分析、实验总结。实验评价由教师填写。
2. 各部分填写内容的要求
   * **实验要求**主要填写源代码。
   * **实验结果记录**中主要反映程序运行后的效果。
   * **实验结果分析**中主要就程序运行结果是否符合预期进行分析，对于达不到预期的分析其产生的原因。
   * **实验总结**中主要阐述在实验中遇到了哪些问题，如何解决的。
3. 实验报告及源代码的提交方式
   * 报告和源代码一并发送到trustie平台。
   * 附件中应有两个文件：一是word版实验报告，**实验报告的命名规则**是：“学号\_姓名\_班级\_实验X”。另一个是压缩文件。它是将整个工程压缩成一个文件，这个**压缩文件的命名规则**是：“学号\_姓名\_班级\_实验X\_源代码”。

**实验6 二叉树及应用**

**一、实验目的**

本次实验的目的是熟悉树的各种物理表示方法及各种遍历方式 (其中以二叉树为侧重点)，了解树在计算机科学及其他工程中的应用。

**二、实验内容**

1.创建二叉树。

2.中序遍历二叉树。 (递归和非递归形式)

3.中序线索二叉树的建立和遍历。

4.赫夫曼树和赫夫曼树编码。

三、**实验要求**  
**”head.h”头文件**

#include"stdio.h"

#include"malloc.h"

#include"iostream"

using namespace std;

typedef int Elemtype;

typedef struct BiTNode{

Elemtype data;

struct BiTNode\* Lchild, \* Rchild;

}BiTNode ,\*BiTree;

void NewTree(BiTree& T, Elemtype n);/////初始化

void InsertTree(BiTree& T, Elemtype n);//////树的元素输入

void ShowTree(BiTree& T);///////树中元素的打印

void cengci(BiTree& Root);

typedef struct node {

BiTree data;

struct node\* next;

}node, \* QueuePtr;

typedef struct {

QueuePtr front;

QueuePtr rear;

int length;

}Queue;

void InitQueue(Queue& Q);//////初始化

void EnQueue(Queue& Q, BiTree e);/////入队操作

void DeQueue(Queue& Q, BiTree& e);//////出队操作

int QueueEmpty(Queue& Q);

typedef char Elemtype1;

typedef enum PointerType{Link,THread};

typedef struct BiThrNode {

Elemtype1 data;

struct BiThrNode\* Lchild, \* Rchild;

PointerType LTag, RTag;

}BiThrNode,\*BiThrTree;

BiThrTree Insert\_BiTHrTree(BiThrTree Root);//初始化

void Print\_BiTHrTree(BiThrTree Root);///打印

void InThreading(BiThrTree p, BiThrTree& pre);

void InOrderThreading(BiThrTree& head, BiThrTree Root);

void read(BiThrTree t);

typedef struct {

int weight;

int Parent;

int Lchild;

int Rchild;

}THNode,\*HuffmanTree;

typedef char\*\* HuffmanNode;

void Compare(HuffmanTree ht, int n, int\* p, int\* q);//////比较出两个最小值

void CreateHuffmanTree(HuffmanTree ht, int n, int\* w);///////哈夫曼树的建立及编码实现

**“BiTHrNode.cpp” 二叉树**

#include"head.h"

BiThrTree Insert\_BiTHrTree(BiThrTree Root)

{

char ch;

ch = getchar();

Root = (BiThrTree)malloc(sizeof(BiThrNode));

if (ch != '#')

{

Root->data = ch;

Root->LTag = Root->RTag = Link;

Root->Lchild = Insert\_BiTHrTree(Root->Lchild);

Root->Rchild = Insert\_BiTHrTree(Root->Rchild);

}

else {

return NULL;

}

return Root;

}

void Print\_BiTHrTree(BiThrTree Root)

{

if (Root)

{

Print\_BiTHrTree(Root->Lchild);

cout << Root->data;

Print\_BiTHrTree(Root->Rchild);

}

}

void InThreading(BiThrTree p, BiThrTree &pre)

{

if (p)

{

InThreading(p->Lchild, pre);

if (!(p->Lchild))

{

p->LTag = THread;

p->Lchild = pre;

}

if (!(pre->Rchild))

{

pre->RTag = THread;

pre->Rchild = p;

}

pre = p;

InThreading(p->Rchild, pre);

}

}

void InOrderThreading(BiThrTree &head,BiThrTree Root)

{

BiThrTree pre;

head = (BiThrTree)malloc(sizeof(BiThrNode));

pre = head;

head->LTag = Link;

head->RTag = THread;

head->Rchild = head;

if (!Root)

{

head->Lchild = head;

}

else

{

head->Lchild = Root;

InThreading(Root, pre);

pre->RTag = THread;

pre->Rchild = head;

head->Rchild = pre;

}

}

void read(BiThrTree t)

{

BiThrTree p;

p = t->Rchild;

while (p !=t)

{

cout << p->data;

while (p->LTag == THread && p->Lchild != t)

{

p = p->Lchild;

if(p!=t)

cout << p->data;

}

if (p->LTag == THread && p->Lchild == t)

break;

if (p->LTag == Link &&p->Lchild!=t)

{

p=p->Lchild;

while (p->RTag==Link)

{

p = p->Rchild;

}

}

}

}

“BiTree.cpp”二叉树的遍历及线索二叉树的建立

#include"head.h"

void NewTree(BiTree &T,Elemtype n)

{

T = (BiTree)malloc(sizeof(BiTNode));

T->data = n;

T->Lchild = T->Rchild = NULL;

}

void InsertTree(BiTree& T, Elemtype n)

{

if (T == NULL)

{

NewTree(T, n);

}

if (n > T->data)

{

InsertTree(T->Rchild, n);

}

else if (n < T->data)

{

InsertTree(T->Lchild, n);

}

}

void ShowTree(BiTree& T)

{

if (T!=NULL)

{

ShowTree(T->Lchild);

cout << " " << T->data<<" ";

ShowTree(T->Rchild);

}

}

void cengci(BiTree& Root)

{

Queue Q;

InitQueue(Q);

if (Root == NULL) return; //树为空则返回

BiTNode\* p = Root; // 临时保存树根Root到指针p中

cout << p->data; // 访问根节点

if (p->Lchild) EnQueue(Q, p->Lchild); // 若存在左孩子，左孩子进队列

if (p->Rchild) EnQueue(Q, p->Rchild); // 若存在右孩子，右孩子进队列

while (QueueEmpty(Q)) // 若队列不空，则层序遍历

{

DeQueue(Q, p); // 出队列

cout << p->data; // 访问当前节点

if (p->Lchild) EnQueue(Q, p->Lchild); // 若存在左孩子，左孩子进队列

if (p->Rchild) EnQueue(Q, p->Rchild); // 若存在右孩子，右孩子进队列

}

}

“Queue.cpp”队列的基本操作

#include"head.h"

void InitQueue(Queue& Q)//////初始化

{

Q.front = Q.rear = (QueuePtr)malloc(sizeof(node));

if (!Q.front) exit(1);

Q.front->next = NULL;

Q.length = 0;

}

void EnQueue(Queue& Q, BiTree e)/////入队操作

{

QueuePtr s;

s = (QueuePtr)malloc(sizeof(node));

s->data = e;

s->next = NULL;

Q.rear->next = s;

Q.rear = s;

Q.length++;

}

void DeQueue(Queue& Q, BiTree& e)//////出队操作

{

QueuePtr p;

p = Q.front->next;//////头指针为空

e = p->data;

Q.front->next = p->next;

if (Q.rear == p)

Q.rear = Q.front;

free(p);

Q.length--;

}

int QueueEmpty(Queue& Q)////////层次遍历

{

if (Q.front == Q.rear)

return 0;

else

{

return 1;

}

}

“main.cpp”主函数

#include"head.h"

int main()

{

int option;

int sign=1;

Elemtype n;

BiTree T=NULL;

while (sign)

{

cout << "--------------------菜单--------------------" << endl;

cout << " 1.输入树中元素 2.中序序列输出树中元素 " << endl;

cout << " 3.层次遍历 4.退出 " << endl;

cout << "请输入功能（1-2）： ";

cin >> option;

switch (option)

{

case 1:

cout << "请输入元素： ";

cin >> n;

InsertTree(T, n);

break;

case 2:

cout << "树中按照中序遍历为——>" ;

ShowTree(T);

cout << endl;

break;

case 3:

cengci(T);

break;

case 4:

sign=0;

break;

}

}

/\*

BiThrTree Root;

BiThrTree head;

Root = (BiThrTree)malloc(sizeof(BiThrNode));

head = (BiThrTree)malloc(sizeof(BiThrNode));

Root = Insert\_BiTHrTree(Root);

Print\_BiTHrTree(Root);

InOrderThreading(head,Root);

read(head);

\*/

/\*

HuffmanTree ht = NULL;

int w[5] = { 1,2,3,4,5 };

CreateHuffmanTree(ht,5, w);\*/

return 0;

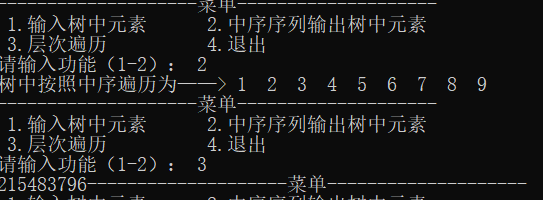
}

1. **完整的实验结果记录**

**测试一：输入树中元素258147369**

**并将它们分别按照中序输出，和层次输出**

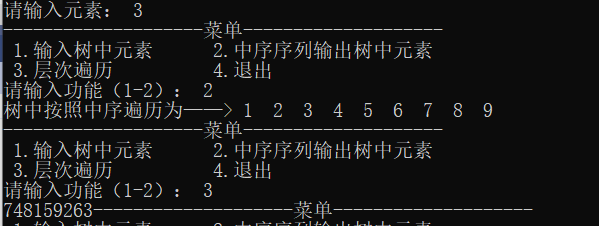
**结果正确；**



测试二：输入树中元素789456123

**并将它们分别按照中序输出，和层次输出**

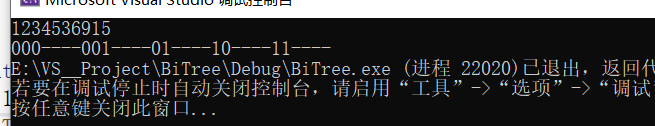
**结果正确；**



测试一：输入哈夫曼树的权值为12345

编码应为000-001-01-10-11

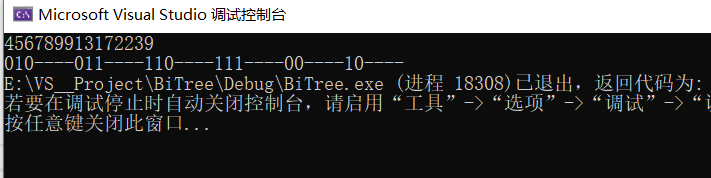
结果正确：



测试二：输入哈夫曼树的权值为456789

编码应为010-011-110-111-00-10

结果正确：



1. **实验结果分析**

**结果都正确**

**六、实验总结**

**在进行哈夫曼树编码是总是显示栈溢出，进行一步步调试才发现末尾”\0”没有分配这个空间，最后多分配了一个空间才正确；**

**七、实验评价**（教师）