哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称：数据结构与算法

课程类型：必修

实验项目：树型结构的建立与遍历

实验题目：二叉树存储结构的建立与遍历

实验日期：2021.01.02

班级：1803501

学号：L170300901

姓名：卢兑玩

设计成绩	报告成绩	指导老师

一、 实验目的

1. 掌握树的链式存储方式及其操作实现（创建、遍历、查找等）。
2. 掌握二叉树用不同方法表示所对应的不同输入形式。
3. 掌握二叉树中各种重要性质在解决实际问题中的应用。
4. 掌握哈夫曼树的构造方法及其编码方法。
5. 掌握二叉排序树的特性及其构造方法。

二、实验要求及实验环境

实验要求：

1．编写建立二叉树的二叉链表存储结构（左右链表示）的程序，并以适当的形式显示和保存二叉树；

2．采用二叉树的二叉链表存储结构，编写程序实现二叉树的先序、中序和后序遍历的递归和非递归算法以及层序遍历算法，并以适当的形式显示和保存二叉树及其相应的遍历序列；

3．给定一个二叉树， 编写算法完成下列应用:（二选一）

（1）判断其是否为完全二叉树；

（2）求二叉树中任意两个结点的公共祖先。

3.1．在二叉树的二叉链表存储结构基础上，编写程序实现二叉树的中序线索链表存储结构（中序线索二叉树）建立的算法，并以适当的形式显示和保存二叉树的相应的线索链表；

3.2．在二叉树的线索链表存储结构上，编写程序分别实现，求二叉树任意结点的先序、中序和后序遍历的后继结点算法；

3.3．以上一条要求为基础，编写程序实现对中序线索二叉树进行先序、中序和后序遍历的非递归算法，并以适当的形式显示和保存二叉树和相应的遍历序列。

实验环境

Windows10, Codeblocks20.03

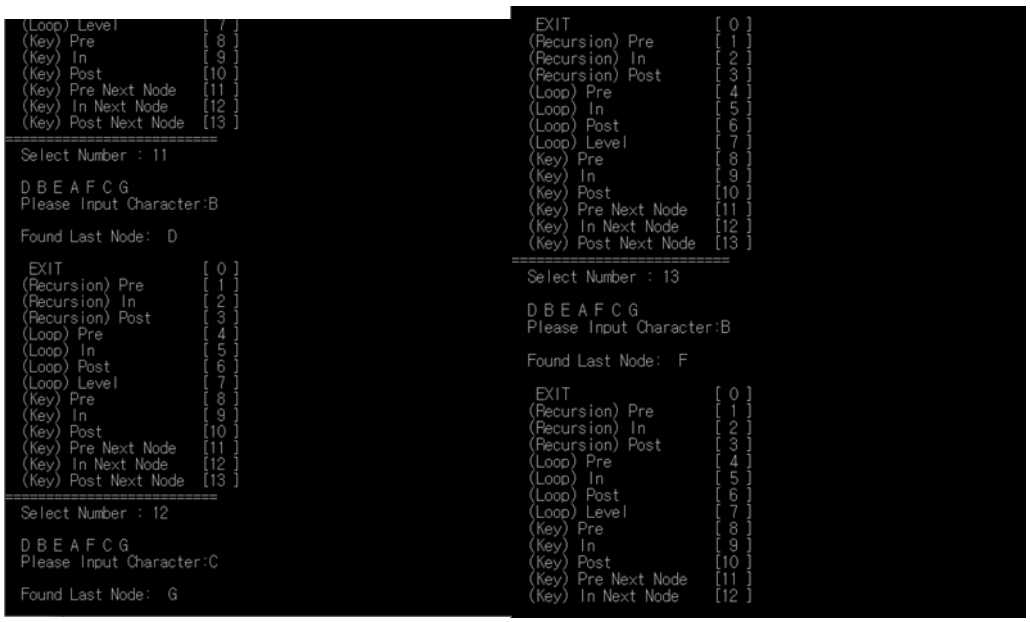
三、设计思想（本程序中的用到的所有数据类型的定义，主程序的流程图及各程序模块之间的调用关系）

```
void    SelectFunc(BTN,TBTN,BTN,TBTN);    //在 main 函数里面调用的 Select 函数,为了选择项目
BTN     MakeBTreeNode(void);              //生成 Btree 的 Node 及返回生成的 node
TBTN    MakeTBTreeNode(void);             //生成 ThreadBTree 的 Node 及返回生成的 Node
```

```
BTN     CreateBT(int,char,BTN,BTN);          //链接孩子 node 和存放内容
char*   BCreatecx();                         //层序方式读文件
char*   BCreatexx();                         //线序方式读文件
void    connectcx(char*,BTN,TBTN);           //层序方式读的内容生成一棵树
void    connectxx(char*,int,BTN,TBTN);       //线序方式读的内容生成一棵树
void    ShowData(char);  //打印内容
void    gotoxy(int,int); //gotoxy 函数，用 window 函数决定显示的位置
void    push(st, BTN);   //Save to stack
BTN     pop(st);         //Load from Stack
void    Pre_order_Tour(BTN); //用递归函数打印前序遍历
void    In_order_Tour(BTN);  //用递归函数打印中序遍历
void    Post_order_Tour(BTN);   //用递归函数打印后序遍历
void    in(BTN);             //用循环打印前序遍历
void    pre(BTN);             //用循环打印中序遍历
void    post(BTN);           //用循环打印后序遍历
void    level(BTN);           //打印层序遍历
void    InOrderTh(TBTN); //用二叉树生成中序线索二叉树。
TBTN    PreNext(TBTN);       //用中序线索二叉树找前序的后续结点而返回后续结点
void    PreO(TBTN);      //打印在中序线索二叉树的前序遍历
TBTN    InNext(TBTN);        //用中序线索二叉树找中序的后续结点而返回后续结点
void    InO(TBTN);           //打印在中序线索二叉树的中序遍历
TBTN    PostNext(TBTN,TBTN); //用中序线索二叉树找后序的后续结点而返回后续结点
void    PostO(TBTN);         //打印在中序线索二叉树的后序遍历
TBTN    FindNode(TBTN,char); //用 char 参数找 Node 及返回
```

四、测试结果

上边显示树的样子，显示的是按层序读的内容。（输入数字可选择读的方式）



五、经验体会与不足

通过这个实验 知道了二叉树的结构 然后知道了二叉树有几种方法 那些种方法的原理和实现原里不如说 先跟是什么，中跟是什么，后跟是什么。二叉树结构的特点是很快 为了理解这个结构 这个实验很有意

六、附录：源代码（带注释）

```
#include <stdio.h>

#include <string.h>

#include <math.h>

#include <windows.h>

//"D://xx.txt","D://cx.txt"


typedef  charBTData;

typedef  struct  _btNode  BTreeNode;

typedef BTreeNode* BTN;

struct  _btNode                                          //BTREE

{

    int num;
```

```
    BTData      data;
    BTN left;
    BTN right;
};

typedef charTBTData;
typedef struct  _threadbtNode    TBTreeNode;
typedef TBTreeNode* TBTN;
struct  _threadbtNode                                    //Thread BTREE
{
    int num;
    TBTData     data;
    TBTNleft;
    TBTNright;
    int left_thread;
    int right_thread;
};

typedef struct stack1* st;
typedef struct stack1 stack;
struct stack1                                            //Stack
{
    TBTN data;
    st next;
};

void    SelectFunc(BTN,TBTN,BTN,TBTN);                          //select
BTN     MakeBTreeNode(void);                             //MakeBTreeNode
TBTN    MakeTBTreeNode(void);                            //MakeThreadBTreeNode
```

```
BTN     CreateBT(int,char,BTN,BTN);
char*   BCreatecx();                                    //ReadFile Create
char*   BCreatexx();                                    //ReadFile First
void    connectcx(char*,BTN,TBTN);                          //cx
void    connectxx(char*,int,BTN,TBTN);                 //xx
void    ShowData(char);                                 //Print
void    gotoxy(int,int);                                //gotoxy
void    push(st, BTN);                                  //about stack
BTN     pop(st);                                        //about stack
void    Pre_order_Tour(BTN);                            //Pre(digui)
void    In_order_Tour(BTN);                             //In(digui)
void    Post_order_Tour(BTN);                           //Post(digui)
void    in(BTN);                                        //In(feidigui)
void    pre(BTN);                                       //Pre(feidigui)
void    post(BTN);                                      //Post(feidigui)
void    level(BTN);                                     //Level
void    InOrderTh(TBTN);
TBTN    PreNext(TBTN);
void    PreO(TBTN);
TBTN    InNext(TBTN);
void    InO(TBTN);
TBTN    PostNext(TBTN,TBTN);
void    PostO(TBTN);
TBTN    FindNode(TBTN,char);



int op=-1;
int front=0,rear=0;
BTN Q[100];
```

```
TBTN pree=NULL;


void main()
 {
     char xx[100],cx[100];
     int i,j,k;
     for(i=0; i<100&&BCreatexx()[i]!=NULL; i++)
     {
        xx[i]=BCreatexx()[i];
     }
     for(j=i;j<100; j++)
        xx[j]=NULL;
     for(k=0; k<100&&BCreatecx()[k]!=NULL; k++)
     {
        cx[k]=BCreatecx()[k];
     }
     for(j=k;j<100; j++)
        cx[j]=NULL;


     BTN cx_h=MakeBTreeNode();
     BTN xx_h=MakeBTreeNode();
     TBTN cx_th=MakeTBTreeNode();
     TBTN xx_th=MakeTBTreeNode();
     TBTN xx_tha=MakeTBTreeNode();
     connectxx(xx,i,xx_h,xx_tha);
     connectcx(cx,cx_h,cx_th);
     cx_h=cx_h->left;
     cx_th=cx_th;
     xx_th->left=xx_tha;
```

```
    SelectFunc(xx_h,xx_th,cx_h,cx_th);
}

void SelectFunc(BTN xx_h,TBTN xx_th,BTN cx_h,TBTN cx_th)
{
    int  ch,number;
    char aaa;
    BTN pbt=cx_h;
    TBTN head=cx_th;
    printf("\n  Select Way ToRead File\n   [1]:Pre\n   [Other]:Level\n   ");
    scanf("%d",&number);
    if(number==1)
    {
        pbt=xx_h;
        head=xx_th;
    }
    InOrderTh(head);
    while (1)
    {
        printf("\n");
        printf("   EXIT\t\t\t[ 0 ] \n");
        printf("  (Recursion) Pre\t[ 1 ] \n");
        printf("  (Recursion) In\t[ 2 ] \n");
        printf("  (Recursion) Post\t[ 3 ] \n");
        printf("  (Loop) Pre\t\t[ 4 ] \n");
        printf("  (Loop) In\t\t[ 5 ] \n");
        printf("  (Loop) Post\t\t[ 6 ] \n");
        printf("  (Loop) Level\t\t[ 7 ] \n");
        printf("  (Key) Pre\t\t[ 8 ] \n");
```

```
printf("  (Key) In\t\t[ 9 ] \n");
printf("  (Key) Post\t\t[10 ] \n");
printf("  (Key) Pre Next Node\t[11 ] \n");
printf("  (Key) In Next Node\t[12 ] \n");
printf("  (Key) Post Next Node\t[13 ] \n");
printf("=========================\n");
printf("  Select Number : ");
scanf("%d", &ch);
putchar('\n');

if (ch == 0)
{
    printf("ByeBye. \n");
    break;
}
else if (0 < ch &&  ch < 14)
{
    putchar(' ');
    switch (ch)
    {
    case 1:
        Pre_order_Tour(pbt);
        break;
    case 2:
        In_order_Tour(pbt);
        break;
    case 3:
        Post_order_Tour(pbt);
        break;
```

```
 case 4:
      pre(pbt);
      break;
case 5:
      in(pbt);
      break;
case 6:
      post(pbt);
      break;
case 7:
      level(pbt);
      break;
case 8:
     PreO(head);
     break;
case 9:
     InO(head);
     break;
case 10:
     PostO(head);
     break;
case 11:
     getchar();
     InO(head);
     printf("\n  Please Input Character:");
     scanf("%c",&aaa);
     printf("\n  Found Last Node: ");
     ShowData(PreNext(FindNode(head,aaa))->data);
     break;
```

```
            case 12:
                getchar();
                InO(head);
                printf("\n  Please Input Character:");
                scanf("%c",&aaa);
                printf("\n  Found Last Node: ");
                ShowData(InNext(FindNode(head,aaa))->data);
                break;
            case 13:
                getchar();
                InO(head);
                printf("\n  Please Input Character:");
                scanf("%c",&aaa);
                printf("\n  Found Last Node: ");
                ShowData(PostNext(FindNode(head,aaa),head)->data);
                break;
            default:
                break;
             }
        }
        putchar('\n');
    }
}

BTN MakeBTreeNode(void)
{
    BTN  node = (BTN)malloc(sizeof(BTreeNode));
    node->left = NULL;
    node->right = NULL;
```

```
    return node;
}

TBTN MakeTBTreeNode(void)
{
    TBTN  node = (TBTN)malloc(sizeof(TBTreeNode));
    node->left = NULL;
    node->right = NULL;
    return node;
}

BTN CreateBT(int i,char ch,BTN left,BTN right)
{
    BTN bts = MakeBTreeNode();
    bts->left=left;
    bts->right=right;
    bts->data=ch;
    bts->num=i;
    return bts;
}

char* BCreatexx()
{
    FILE* fp;
    fp = fopen("D://xx.txt", "r");
    char ch[100];
    if (fp)
    {
        fscanf(fp, "%s", &ch);
```

```
    }
    fclose(fp);
    return ch;

}

char* BCreatecx()
{
    FILE* fp;
    fp = fopen("D://cx.txt", "r");
    char ch[100];
    if (fp)
    {
        fscanf(fp, "%s", &ch);
    }
    fclose(fp);
    return ch;

}

void connectcx(char* c,BTN abt,TBTN atbt)
{
    int i,j,s=0;
    char ch[100];
    BTN bt[100];
    TBTN tbt[100];
    for(i=0; i<100; i++)
    {
        bt[i]=NULL;
```

```
    tbt[i]=NULL;
    ch[i]=c[i];
}
for(i=0; ch[i]!='\0'; i++);
for(j=i; j>0; j--)
{
    if(2*j>i)
    {
        tbt[j]=CreateBT(j,ch[j-1],NULL,NULL);
        bt[j]=CreateBT(j,ch[j-1],NULL,NULL);
    }
    else if(2*j+1>i)
    {
        tbt[j]=CreateBT(j,ch[j-1],tbt[2*j],NULL);
        bt[j]=CreateBT(j,ch[j-1],bt[2*j],NULL);
    }
    else
    {
        tbt[j]=CreateBT(j,ch[j-1],tbt[2*j],tbt[2*j+1]);
        bt[j]=CreateBT(j,ch[j-1],bt[2*j],bt[2*j+1]);
    }
}
for(j=i+1;j<100;j++)
{
    free(bt[j]);
    free(tbt[j]);
}
atbt->left=tbt[1];
abt->left=bt[1];
```

```
gotoxy(30, 1);
if (bt[++s]&&bt[s]->data!='#')
    ShowData(bt[s]->data);
for(i=24;i<40;i+=12)
{
    gotoxy(i, 3);
    if (bt[++s]&&bt[s]->data!='#')
        ShowData(bt[s]->data);
}
for(i=18;i<50;i+=8)
{
    gotoxy(i, 5);
    if (bt[++s]&&bt[s]->data!='#')
        ShowData(bt[s]->data);
}
gotoxy(12, 7);
if (bt[++s]&&bt[s]->data!='#')
    ShowData(bt[s]->data);
for(i=19;i<30;i+=4)
{
    gotoxy(i, 7);
    if (bt[++s]&&bt[s]->data!='#')
        ShowData(bt[s]->data);
}
for(i=33;i<42;i+=4)
{
    gotoxy(i, 7);
    if (bt[++s]&&bt[s]->data!='#')
```

```
        ShowData(bt[s]->data);
}
gotoxy(50, 7);
if (bt[++s]&&bt[s]->data!='#')
    ShowData(bt[s]->data);

gotoxy(6, 9);
if (bt[++s]&&bt[s]->data!='#')
    ShowData(bt[s]->data);
for(i=14;i<22;i+=3)
{
    gotoxy(i, 9);
    if (bt[++s]&&bt[s]->data!='#')
        ShowData(bt[s]->data);
}
for(i=22;i<30;i+=2)
{
    gotoxy(i, 9);
    if (bt[++s]&&bt[s]->data!='#')
        ShowData(bt[s]->data);
}
for(i=32;i<40;i+=2)
{
    gotoxy(i, 9);
    if (bt[++s]&&bt[s]->data!='#')
        ShowData(bt[s]->data);
}
for(i=40;i<49;i+=4)
{
```

```
        gotoxy(i, 9);
        if (bt[++s]&&bt[s]->data!='#')
            ShowData(bt[s]->data);
    }
    gotoxy(56, 9);
    if (bt[++s]&&bt[s]->data!='#')
        ShowData(bt[s]->data);
}

void connectxx(char *ch,int k,BTN T,TBTN AT)
{
    if(++op>k)
    {
        T->data=NULL;
        T->left=NULL;
        T->right=NULL;
        T = NULL;
        AT->data=NULL;
        AT->left=NULL;
        AT->right=NULL;
        AT =NULL;
        return;
    }
    if( ch[op] == '#' )
    {
        T->data=NULL;
        T = NULL;
        AT->data=NULL;
        AT =NULL;
```

```
    }
    else
    {
        BTN A=MakeBTreeNode();
        BTN B=MakeBTreeNode();
        TBTN C=MakeTBTreeNode();
        TBTN D=MakeTBTreeNode();
        T->left=A;
        T->right=B;
        T->num=op;
        T->data = ch[op];
        AT->left=C;
        AT->right=D;
        AT->num=op;
        AT->data = ch[op];
        connectxx( ch, k, A, C);
        connectxx( ch, k, B, D);
    }
 }

void ShowData(char dat)
{
    if (dat != '#')
    printf(" %c", dat);
}

void gotoxy(int x, int y)//내가 원하는 위치로 커서 이동
{
    COORD pos = { x - 1, y - 1 };//커서가 X 좌표에서 -1 한값. Y 좌표에서 -1 한 값으로
```

```
이동
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE),    pos);//    WIN32API
함수입니다. 이건 알필요 없어요
}

void push(st stack_h, BTN data)
{
    st p = stack_h;
    st stack_s = (st)malloc(sizeof(stack));
    if (stack_h->next == NULL)
    {
        stack_s->next = NULL;
        stack_h->next = stack_s;
        stack_s->data = data;
    }
    else
    {
        stack_s->data = data;
        stack_s->next = p->next;
        p->next = stack_s;
    }

}

BTN pop(st stack_h)
{
    BTN ch;
    st p = stack_h->next;
    if (stack_h->next)
```

```
    {
        ch = p->data;
        stack_h->next = p->next;
        p->next = p;
        free(p);
        return ch;
    }
    else
        return NULL;
}

void Pre_order_Tour(BTN pbt)
{
    if (pbt == NULL||pbt->data==NULL)
        return;
    ShowData(pbt->data);
    Pre_order_Tour(pbt->left);
    Pre_order_Tour(pbt->right);
}

void In_order_Tour(BTN pbt)
{
    if (pbt == NULL||pbt->data==NULL)
        return;
    In_order_Tour(pbt->left);
    ShowData(pbt->data);
    In_order_Tour(pbt->right);
}
```

```
void Post_order_Tour(BTN pbt)
{
    if (pbt == NULL||pbt->data==NULL)
        return;
    Post_order_Tour(pbt->left);
    Post_order_Tour(pbt->right);
    ShowData(pbt->data);
}

void in(BTN bth)
{
    st stack_0 = (st)malloc(sizeof(stack));
    stack_0->next=NULL;
    BTN p =bth;
    while(1)
    {
        for(;p;)
        {
            push(stack_0,p);
            p=p->left;
        }
        if(!stack_0->next)
            return;
        p=pop(stack_0);
        if(p->data)
            ShowData(p->data);
        p=p->right;
    }
}
```

```
void pre(BTN bth)
{
    st stack_0 = (st)malloc(sizeof(stack));
    stack_0->next=NULL;
    BTN p =bth;
    while(1)
    {
        for(;p;)
        {
            push(stack_0,p);
            if(p->data)
            ShowData(p->data);
            p=p->left;
        }
        if(!stack_0->next)
            return;
        p=pop(stack_0);
        p=p->right;
    }
}

void post(BTN bth)
{
    st stack_0 = (st)malloc(sizeof(stack));
    stack_0->next=NULL;
    int abc[100],i;
    for(i=0; i<100; i++)
        abc[i]=1;
```

```
    BTN p =bth;
    push(stack_0,p);
    while(1)
    {
        if(!stack_0->next)
            return;
        else
        {
            p=pop(stack_0);
            push(stack_0,p);
            if(p!=NULL)
            {
                if(p->left&&p->left->data!=NULL&&abc[p->left->num]==1)
                    push(stack_0,p->left);
                else
                {
                    if(p->right&&p->right->data!=NULL&&abc[p->right->num]==1)
                        push(stack_0,p->right);
                    else
                    {
                        ShowData(p->data);
                        abc[p->num]=-1;
                        pop(stack_0);
                    }
                }
            }
        }
    }
}
```

```
void level(BTN bth)
{

    BTN p=bth;
    ShowData(p->data);
    if(bth->left)
        Q[front++]=bth->left;
    if(bth->right)
        Q[front++]=bth->right;

    for(;front!=rear;)
        level(Q[rear++]);
}

void InOrderTh(TBTN p)
{
    if(p&&p->data){
        InOrderTh(p->left);
        p->left_thread=(p->left&&p->left->data)?1:0;
        p->right_thread=(p->right&&p->right->data)?1:0;
        if(pree)
        {
            if(pree->right_thread==0)
                pree->right=p;
            if(p->left_thread==0)
                p->left=pree;
        }
        pree=p;
```

```
        InOrderTh(p->right);
    }
}

TBTN PreNext(TBTN p)
{
    TBTN q;
    if(!p->left_thread)
    {
        q=p;
        for(;!q->right_thread;)
            q=q->right;
        q=q->right;
    }
    else
        q=p->left;
    return q;
}

void PreO(TBTN head)
{
    TBTN tmp;
    tmp=head->left;
    ShowData(tmp->data);
    for(;tmp->right!=head;)
    {
        tmp=PreNext(tmp);
        if(tmp!=head)
        {
```

```
            if(tmp->data)
            ShowData(tmp->data);
        }
    }
}

TBTN InNext(TBTN p)
{
    if(!p)
        return NULL;
    if(!p->right_thread)
        return p->right;
    TBTN q = p->right;
    for(;q->left_thread;)
        q = q->left;
    return q;
}

void InO(TBTN head)
{
    TBTN p=head->left;
    while(p!=head)
    {
        while(p->left_thread)
            p=p->left;
        if(p->data)
            ShowData(p->data);
        while(!p->right_thread&&p->right!=head)
        {
```

```
            p=p->right;
            if(p->data)
                ShowData(p->data);
        }
        p=p->right;
    }
}

TBTN PostNext(TBTN p,TBTN head)
{
    TBTN q=p,r;
    int flag = 0;
    if(!p->right_thread)
    {
        if(p->right->left == p)
        {
            flag = 1;
            r = p->right;
        }
    }
    else if(p->right_thread)
    {
        q = p->right;
        for(;q->right_thread;)
        {
            q = q->right;
        }
        q = q->right;
        if(q->left == p)
```

```
    {
         flag = 1;
         r = q;
    }
    else if(q == head && q->left == p)
        return head;
}
if (flag&&r->right_thread)
{
    q = r->right;
    while(q->left_thread)
    {
        q = q->left;
    }
    r = q;
    return r;
}
else if(flag&&!r->right_thread)
    return r;
q = p;
if(p->left_thread)
{
    q = q->left;
    for(;q->left_thread;)
    {
        q = q->left;
    }
    q = q->left;
    if(q->right == p)
```

```
            return q;
    }
    else if(!p->left_thread)
    {
        if(p->left->right == p)
            return p->left;
    }
}

void Post0(TBTN p)
{
    if(p!=NULL)
    {
        if(p->left_thread==1)
            Post0(p->left);
        if(p->right_thread==1)
            Post0(p->right);
        if(p->data)
            ShowData(p->data);
    }

}

TBTN FindNode(TBTN head,char abc)
{
    TBTN p=head->left;
    while(p!=head)
    {
        while(p->left_thread==1)
```

```
            p=p->left;
        if(p->data==abc)
            return p;
        while(p->right_thread==0&&p->right!=head)
        {
            p=p->right;
            if(p->data==abc)
            return p;
        }
        p=p->right;
    }
    return NULL;
}
```