

数据结构

2022

实验报告

实验项目名称：树和二叉树

班级：2021级6班

学号：2021302181138

姓名：伍旺旺

指导教师：沈志东

实验时间：2022 年 5 月 5 日

实验 7: 树和二叉树

一、实验要求

- 实验题 2: 实现二叉树的各种遍历算法
- 实验题 5: 构造哈夫曼树和生成哈夫曼树编码

二、实验环境

硬件: 微型计算机

软件: Windows 操作系统、Microsoft Visual Studio Code

三、实验步骤及思路

(一) 实验题 2: 实现二叉树的各种遍历算法

- 题目分析
根据题目要求, 该程序包含如下函数:

- PreOrder (BTreeNode * b): 二叉树 b 的先序遍历的递归算法。
- PreOrder1 (BTreeNode * b): 二叉树 b 的先序遍历的非递归算法。
- InOrder (BTreeNode * b): 二叉树 b 的中序遍历的递归算法。
- InOrder1 (BTreeNode * b): 二叉树 b 的中序遍历的非递归算法。
- PostOrder (BTreeNode * b): 二叉树 b 的后序遍历的递归算法。
- PostOrder1 (BTreeNode * b): 二叉树 b 的后序遍历的非递归算法。
- TravLevel (BTreeNode * b): 二叉树 b 的层次遍历算法

• 实验具体步骤

1. PreOrder (BTreeNode * b)函数编写

```
void PreOrder(BTreeNode *b)                //先序遍历的递归算法
{
    if (b!=NULL)
    {
        printf("%c ",b->data);             //访问根结点
        PreOrder(b->lchild);                //递归访问左子树
        PreOrder(b->rchild);                //递归访问右子树
    }
}
```

2. PreOrder1(BTreeNode *b)函数编写

```

void PreOrder1(BTNode *b)                                //先序非递归遍历算法
{
    BTNode *St[MaxSize],*p;
    int top=-1;
    if (b!=NULL)
    {
        top++;                                           //根结点进栈
        St[top]=b;
        while (top>-1)                                   //栈不为空时循环
        {
            p=St[top];                                   //退栈并访问该结点
            top--;
            printf("%c ",p->data);
            if (p->rchild!=NULL)                          //有右孩子,将其进栈
            {
                top++;
                St[top]=p->rchild;
            }
            if (p->lchild!=NULL)                          //有左孩子,将其进栈
            {
                top++;
                St[top]=p->lchild;
            }
        }
        printf("\n");
    }
}

```

3. InOrder(BTNode *b)函数编写

```

void InOrder(BTNode *b)                                  //中序遍历的递归算法
{
    if (b!=NULL)
    {
        InOrder(b->lchild);                             //递归访问左子树
        printf("%c ",b->data);                          //访问根结点
        InOrder(b->rchild);                             //递归访问右子树
    }
}

```

4. InOrder1(BTNode *b)函数编写

```

void InOrder1(BTNode *b)                                //中序非递归遍历算法
{
    BTNode *St[MaxSize],*p;
    int top=-1;
    if (b!=NULL)
    {
        p=b;
        while (top>-1 || p!=NULL)
        {
            while (p!=NULL)                            //扫描结点p的所有左下结点并进栈
            {
                top++;
                St[top]=p;
                p=p->lchild;
            }
            if (top>-1)
            {
                p=St[top];                            //出栈结点p并访问
                top--;
                printf("%c ",p->data);
                p=p->rchild;
            }
        }
        printf("\n");
    }
}

```

5. PostOrder(BTNode *b)函数编写

```

void PostOrder(BTNode *b)                                //后序遍历的递归算法
{
    if (b!=NULL)
    {
        PostOrder(b->lchild);                        //递归访问左子树
        PostOrder(b->rchild);                        //递归访问右子树
        printf("%c ",b->data);                      //访问根结点
    }
}

```

6. PostOrder1(BTNode *b)函数编写

```

void PostOrder1(BTNode *b)                                //后序非递归遍历算法
{
    BTNode *St[MaxSize];
    BTNode *p;
    int top=-1;                                           //栈指针置初值
    bool flag;
    if (b!=NULL)
    {
        do
        {
            while (b!=NULL)                               //将b结点的所有左下结点进栈
            {
                top++;
                St[top]=b;
                b=b->lchild;
            }
            p=NULL;                                       //p指向当前结点的
            flag=true;                                    //flag为真表示正序
            while (top!=-1 && flag)
            {
                b=St[top];                                //取出当前的栈顶元素
                if (b->rchild==p)                          //右子树不存在或已被访问,进栈
                {
                    printf("%c ",b->data); //访问b结点
                    top--;
                    p=b;                                   //p指向则被访问的
                }
                else
                {
                    b=b->rchild;                           //b指向右子树
                    flag=false;                             //表示当前不是处理左子树
                }
            }
        } while (top!=-1);
        printf("\n");
    }
}

```

7. TravLevel(BTNode *b)函数编写

```

void TravLevel(BTNode *b)                                //层次遍历
{
    BTNode *Qu[MaxSize];                                //定义环形队列
    int front,rear;                                       //定义队首和队尾指针
    front=rear=0;                                         //置队列为空队
    if (b!=NULL)
        printf("%c ",b->data);
    rear++;                                               //根结点进队
    Qu[rear]=b;
    while (rear!=front)                                  //队列不为空
    {
        front=(front+1)%MaxSize;
        b=Qu[front];                                     //出队结点b
        if (b->lchild!=NULL)                             //输出左孩子,并进队
        {
            printf("%c ",b->lchild->data);
            rear=(rear+1)%MaxSize;
            Qu[rear]=b->lchild;
        }
        if (b->rchild!=NULL)                             //输出右孩子,并进队
        {
            printf("%c ",b->rchild->data);
            rear=(rear+1)%MaxSize;
            Qu[rear]=b->rchild;
        }
    }
    printf("\n");
}

```

(二) 实验题 5：构造哈夫曼树和生成哈夫曼树编码

- 题目分析

该程序包含如下函数：

- CreateHT (HTNode ht int n)：由含有 n 个叶子结点的 ht 构造完整的哈夫曼树。
- reateHCode (HTNode ht , HCode hed [, int n)：由哈夫曼树 ht 构造哈夫曼编码 hcd
- DispHCode (HTNode ht . HCode hcdintn)：出哈夫曼树 ht 利哈夫曼编码 hcd 中 n 个叶子结点的哈夫曼编码

- 实验具体步骤

1. CreateHT(HTNode ht[],int n0)函数编写

```

void CreateHT(HTNode ht[],int n0)          //构造哈夫曼树
{
    int i,k,lnode,rnode;
    double min1,min2;
    for (i=0;i<2*n0-1;i++)                //所有节点的相关域置初值-1
        ht[i].parent=ht[i].lchild=ht[i].rchild=-1;
    for (i=n0;i<=2*n0-2;i++)              //构造哈夫曼树的n0-1个节点
    {
        min1=min2=32767;                  //lnode和rnode为最小权重的两个节点位置
        lnode=rnode=-1;
        for (k=0;k<=i-1;k++)              //在ht[0..i-1]中找权值最小的两个节点
            if (ht[k].parent== -1)        //只在尚未构造二叉树的节点中查找
            {
                if (ht[k].weight<min1)
                {
                    min2=min1;rnode=lnode;
                    min1=ht[k].weight;lnode=k;
                }
                else if (ht[k].weight<min2)
                {
                    min2=ht[k].weight;rnode=k; }
            }
        ht[i].weight=ht[lnode].weight+ht[rnode].weight;
        ht[i].lchild=lnode;ht[i].rchild=rnode; //ht[i]作为双亲节点
        ht[lnode].parent=i;ht[rnode].parent=i;
    }
}

```

2. CreateHCode(HTNode ht[],HCode hcd[],int n0)函数编写

```

void CreateHCode(HTNode ht[],HCode hcd[],int n0)    //构造哈夫曼树编码
{
    int i,f,c;
    HCode hc;
    for (i=0;i<n0;i++)                            //根据哈夫曼树求哈夫曼编码
    {
        hc.start=n0;c=i;
        f=ht[i].parent;
        while (f!= -1)                             //循环直到无双亲节点即到达树根节点
        {
            if (ht[f].lchild==c)                    //当前节点是双亲节点的左孩子
                hc.cd[hc.start--]='0';
            else                                     //当前节点是双亲节点的右孩子
                hc.cd[hc.start--]='1';
            c=f;f=ht[f].parent;                      //再对双亲节点进行同样的操作
        }
        hc.start++;
        hcd[i]=hc;
    }
}

```

3. DispHCode(HTNode ht[],HCode hcd[],int n0)函数编写


```


void DispHCode(HTNode ht[],HCode hcd[],int n0) //输出哈夫曼树编码
{
    int i,k;
    double sum=0,m=0;
    int j;
    printf("  输出哈夫曼编码:\n"); //输出哈夫曼编码
    for (i=0;i<n0;i++)
    {
        j=0;
        printf("      %s:\t",ht[i].data);
        for (k=hcd[i].start;k<=n0;k++)
        {
            printf("%c",hcd[i].cd[k]);
            j++;
        }
        m+=ht[i].weight;
        sum+=ht[i].weight*j;
        printf("\n");
    }
    printf("\n  平均长度=%g\n",1.0*sum/m);
}

```

四、实验结果及分析

(一) 实验题 2：实现二叉树的各种遍历算法

运行结果如下：

 选择 C:\Windows\system32\cmd.exe

二叉树b:A(B(D,E(H(J,K(L,M(,N))))),C(F,G(,I)))

层次遍历序列:A B C D E F G H I J K L M N

先序遍历序列:

递归算法:A B D E H J K L M N C F G I

非递归算法:A B D E H J K L M N C F G I

中序遍历序列:

递归算法:D B J H L K M N E A F C G I

非递归算法:D B J H L K M N E A F C G I

后序遍历序列:


递归算法:D J L N M K H E B F I G C A

非递归算法:D J L N M K H E B F I G C A

请按任意键继续. . .

(二) 实验题 5: 构造哈夫曼树和生成哈夫曼树编码

运行结果如下:

 选择 C:\Windows\system32\cmd.exe

输出哈夫曼编码:

The:	01
of:	101
a:	001
to:	000
and:	1110
in:	1101
that:	11110
he:	11001
is:	11000
at:	10011
on:	10010
for:	10001
His:	10000
are:	111111
be:	111110

平均长度=3.56208

请按任意键继续. . .

五、总结

- 通过实验题 2 加深了对二叉树遍历过程以及遍历算法设计的理解。
- 通过实验题 5 加深了对哈夫曼树构造过程以及哈夫曼编码生成过程的理解。