**二叉树上机题：**

1. 非递归方法建立二叉树

思想：非递归建立一个二叉树，要做的就是不断的往一个树里面去插入一个新的结点  即插入法建立二叉树

实现步骤：

第一步：申请一个结点空间，并进行初始化，判断树是否为空，如果为空，返回新结点，则已经新建了一棵树

第二步：如果不为空，循环遍历，如果插入的值比左孩子孩子小，那么一直往左边找，如果大于右孩子，那么往右边开始找，最后插入即可

第三步：循环插入即可建立一个二叉树；

typedef struct Node

{

int data;

Node \*lch, \*rch;

}BiNode, \*BiTree;

BiTree Insert(BiTree root,int key)

{

BiTree pa, ptr;

BiTree p = root;

ptr = (BiTree)malloc(sizeof(BiNode));

ptr->data = key;

ptr->lch = NULL;

ptr->rch = NULL;

pa = NULL;

if(p == NULL){

return ptr;}

else

{

while(p != NULL)

{pa = p;

if(key <= p->data)

{p = p->lch;}

else

{

p = p->rch;

}}

if(key <= pa->data)

{

pa->lch = ptr;}

else

{

pa->rch = ptr;

}}

return root;

}

BiTree Creat\_Tree()

{

BiTree root = NULL;

int data;

scanf("%d", &data);

while(data != 0)

{

root = Insert(root, data);

scanf("%d", &data);

}

return root;

}

void PreOrder(BiTree root)

{

if(root != NULL)

{

printf("%d ", root->data);

PreOrder(root->lch);

PreOrder(root->rch);

}}

void InOrder(BiTree root)

{

if(root !=NULL)

{

InOrder(root->lch);

printf("%d ",root->data);

InOrder(root->rch);

}}

void PostOrder(BiTree root)

{if(root != NULL)

{PostOrder(root->lch);

PostOrder(root->rch);

printf("%d ", root->data);}}

int main()

{

BiTree root ;

root = Creat\_Tree();

PreOrder(root);

printf("\n");

InOrder(root);

printf("\n");

PostOrder(root);

return 0;

}

1. 求二叉树中节点的最大距离

（1）初始化二叉树，并将每个节点Lm和Rm初始化为0，定义二叉树中节点的最大距离Max = -1  
（2）为了计算一个节点的Lm和Rm，需要采用后序遍历策略，递归的计算出它左孩子Left的Lm,Rm，Lm = max(Left->Lm + Left->Rm) + 1；和右孩子Right的Lm,Rm，Rm = max(Right->Lm,Right->Rm) + 1；  
（3）对于每一个节点，计算Lm + Rm，如果其值大于Max，则Max = Lm + Rm，最后Max就是最大距离

#define max(a,b) a>b?a:b

typedef struct BinTreeNode

{

    int data;

    struct BinTreeNode \*pLeft;

    struct BinTreeNode \*pRight;

}\*pBinTreeNode;

int GetDeepNode(BinTreeNode\* node)

{

    int k = 0;

    if(NULL!=node->pLeft|| NULL!=node->pRight)

    {

        int left=0;

        int right =0;

        if(NULL!=node->pLeft)

            left = GetDeepNode(node->pLeft)+1;

        if(NULL!=node->pRight)

            right = GetDeepNode(node->pRight)+1;

        k= max(left,right);

    }

    return k;

}

int MaxDis(BinTreeNode\* root)

{

    int fatherMax=0;

    if(NULL!=root)

        fatherMax = GetDeepNode(root);

    int leftLen=0;

    int rightLen =0;

    if(NULL!=root->pLeft)

        leftLen = GetDeepNode(root->pLeft)+1;

    if(NULL!=root->pRight)

        rightLen = GetDeepNode(root->pRight)+1;

    int childMax = leftLen+rightLen;

    return max(fatherMax,childMax);

}

int main()

{

    int preOrder[]={10,6,4,7,1,5,8,14,12,16};

    int inOrder[]={1,7,4,5,6,8,10,12,14,16};

    int len = sizeof(preOrder)/sizeof(int);

    BinTreeNode\* root= CreateBinTree(preOrder,inOrder,len);//根据前序和中序构建二叉树

    int maxDistance = MaxDis(root);

    cout<<maxDistance<<endl;

}

1. 判断二叉树是不是完全二叉树

算法原理：按层次遍历二叉树，建立队列来保存节点。

如果队列不为空，从第一个根节点开始遍历。

如果节点没有子树，则判断左右子树节点是否为空，有则不是。

如果有子树，左右子树都不为空，则继续遍历。

左不空，右空，左子树压入队列，继续

若左空右不空，不是。若左右都空，则是。

bool IsCompleteBinaryTree(BiTreeNode\* &T)

{

if(T == NULL)return false;

queue<BiTreeNode\*> q;//创建队列来存储节点

q.push(T);

bool mustHaveNoChild = false;

bool result = true;

while(!q.empty())//如果队列不为空

{

BiTreeNode\* pNode = q.front();//队列头结点为指针

q.pop();//把头结点出列

if(mustHaveNoChild)//如果该节点没有子树

{

if(pNode->lchild != NULL || pNode->rchild != NULL)

{

result = false;//如果左子树或右子树不为空，则不是完全二叉树

break;

}

}

else//如果该节点有子树

{

if(pNode->lchild != NULL && pNode->rchild != NULL)//左右子树都不空

{

q.push(pNode->lchild);

q.push(pNode->rchild);

}

else if(pNode->lchild != NULL && pNode->rchild == NULL)//左子树不为空，右子树为空

{

mustHaveNoChild = true;

q.push(pNode->lchild); }

else if(pNode->lchild == NULL && pNode->rchild != NULL)//左子树为空，右子树不为空

{

result = false;

break;

}else//左右子树都为空

{ mustHaveNoChild = true;

} }}

return result;

}

1. 将二叉查找树变为有序的双向链表

算法思想：根据二叉树的拓展先序序列，使用递归方法构造二叉链表。

递归 算法的输入是二叉树的拓展先序序列，首先输入一个根节点，若输入的是空格字符，表示该二叉树为空树，即T=NULL；否则把输入的字符赋给T-〉data，之后，依次递归地建立它的左子树和右子树。

void CreatBinTree(BinTree \*T)

{

char ch;

if((ch=getchar())==' ')

\*T=NULL;

else //读入非空格

{

\*T=(BinTNode \*)malloc(sizeof(BinTNode)); //生成结点

(\*T)->data=ch;

CreatBinTree(&((\*T->lchild)); //构造左子树

CreatBinTree(&((\*T->rchild)); //构造右子树

}}

1. 判断二叉树是不是平衡二叉树

用后序遍历的方式遍历二叉树的每一个结点，在遍历到一个结点之前我们已经遍历了它的左右子树。只要在遍历每个结点的时候记录它的深度（某一结点的深度等于它到叶节点的路径的长度），我们就可以一边遍历一边判断每个结点是不是平衡的。

#include <stdio.h>

#include <stdlib.h>

typedef struct Node

{

  int data;

  struct Node\* left;

  struct Node\* right;

}TreeNode;

void buildTree(TreeNode\*\* root,int data)

{

  if(\*root==NULL)

  {

    \*root=(TreeNode\*)malloc(sizeof(TreeNode));

(\*root)->data=data;

(\*root)->left=NULL;

(\*root)->right=NULL;

return;

  }

  if((\*root)->data>=data)

  {

    buildTree(&((\*root)->left),data);

  }

  else

  {

    buildTree(&((\*root)->right),data);

  }

}

int isBalanced(TreeNode\* root,int \*depth,int \*v)

{

  int left,right,gap;

  if(root==NULL)

  {

    \*depth=0;

return 1;

  }

  if(isBalanced(root->left,&left,v)&&isBalanced(root->right,&right,v))

  {

    gap=left-right;

if(gap>=-1&&gap<=1)

{

\*depth=1+(left>right?left:right);

return 1;

}

  }

  \*v=-1;

  return -1;

}

int main()

{

  TreeNode\* root=NULL;

  int data[]={6,4,7,2,5,1};

  int i;

  int depth=0;

  int v=1;

  int len=sizeof(data)/sizeof(int);

  for(i=0;i<len;i++)

 buildTree(&root,data[i]);

  isBalanced(root,&depth,&v);

  if(v==1)

  {

    printf("OK\n");

  }

  else if(v==-1)

  {

     printf("no\n");

  }

  return 0;

}

1. 找出二叉查找树中出现频率最高的元素

struct Node

{

Node\* left;

Node\*right;

int data;

Node(int val):data(val),left(NULL),right(NULL){}

};

int maxFreqVal;//出现次数最多的值

int maxFreqCount;//最多的次数

void inorder(Node\* node,int &curFreqVal,int& curFreqCount)

{

if (node == NULL)

{

return ;

}

inorder(node->left,curFreqVal,curFreqCount);

if (curFreqVal != node->data)//如果考察的值不等于遍历到的节点的值

{

curFreqCount = 1;

curFreqVal = node->data;

}

else

{//如果考察的值等于遍历到的节点的值

curFreqCount++;

if (curFreqCount>maxFreqCount)

{//更新最大的频率

maxFreqCount = curFreqCount;

maxFreqVal = curFreqVal;

}

}

inorder(node->right,curFreqVal,curFreqCount);

}

int getFreq(Node \* root)

{

if (root = NULL)

{

return -1;

}

int curFreqVal = -1;//不一定要为-1 只要未在树中出现的值就行

int curFreqCount = -1;

inorder(root,curFreqVal,curFreqCount);

return maxFreqVal;

}

模板

#include<stdio.h>  
#define maxsize 10  
typedef struct node{  
char data;  
struct node \*lchild,\*rchild;  
} node;  
typedef node \*bitree;  
typedef struct {  
    bitree a[maxsize];  
    int top;  
}seqstack;  
  
bitree buildtree()  
{  
    char c;  
    node \*p;  
    c=getchar();  
    if(c=='0')  
    {  
        p=NULL;  
    }  
    else  
    {  
           p=new(node);  
           p->data=c;  
           p->lchild=buildtree();  
           p->rchild=buildtree();  
    }  
    return(p);  
}  
  
int leaf (bitree t)  
{  
   int d;  
    if(t==NULL)  
{  
return 0;  
}  
    else if(t->lchild!=NULL||t->rchild!=NULL)  
   {  
   d=leaf(t->lchild)+leaf(t->rchild);  
  
   }  
  else   
  d=1;  
return(d);  
}  
void preorder(bitree t)  
{  
    seqstack s;  
    s.top = -1;//置栈空  
    while((t)||(s.top!=-1))  
    {  
        while(t)  
        {  
            printf("%c", t->data);  
            s.top++;  
            s.a[s.top] = t;  
            t = t->lchild;  
        }  
        if (s.top>-1) {  
            t = s.a[s.top];  
            s.top--;  
            t = t->rchild;  
        }  
    }  
}  
  
void DLR( node \*root )  
{  
    if (root!=NULL) //非空二叉树  
    {   printf("%c",root->data); //访问D  
            DLR(root->lchild); //递归遍历左子树  
            DLR(root->rchild); //递归遍历右子树  
    }  
}  
  
void midorder(bitree t)  
{  
    seqstack s;  
    s.top = -1;//置栈空  
    while((t)||(s.top!=-1))  
    {  
        while(t)  
        {  
            s.top++;  
            s.a[s.top] = t;  
            t = t->lchild;  
        }  
        t=s.a[s.top];  
        printf("%c", t->data);  
        s.top--;  
        t=t->rchild;  
    }  
}  
  
void LDR(node \*root)  
{  
    if(root !=NULL)  
    {  
            LDR(root->lchild);  
            printf("%c",root->data);  
            LDR(root->rchild);  
    }  
}  
  
void postorder(bitree t)//这是非递归后序实现  
{  
    bitree lastvist;  
    seqstack s;  
    lastvist=0;  
    s.top = -1;//置栈空  
    while(t||s.top != -1)  
    {  
       while(t)  
        {  
            s.top++;  
            s.a[s.top]=t;  
            t=t->lchild;  
        }  
        t=s.a[s.top];  
        if(t->rchild==lastvist||t->rchild==NULL)  
        {  
            printf("%c", t->data);  
            s.top--;  
            lastvist=t;  
            t=0;  
        }  
        else t=t->rchild;  
    }  
}  
  
void LRD (node \*root)  
{  
    if(root !=NULL)  
    {  
        LRD(root->lchild);  
        LRD(root->rchild);  
        printf("%c",root->data);  
    }  
}  
  
void main()  
{  
    bitree t;  
    t=buildtree();   //  测试用例：abc00d00e0fg000  
    printf("\n非递归前序遍历：\n");  
    preorder(t);  
    printf("\nthe leaves of the tree is %d \n",leaf(t));  
    printf("\n递归前序遍历：\n");  
    DLR(t);  
    printf("\n非递归中序遍历：\n");  
    midorder(t);  
    printf("\n递归中序遍历：\n");  
    LDR(t);  
    printf("\n非递归后序遍历：\n");  
    postorder(t);  
    printf("\n递归后序遍历：\n");  
    LRD(t);  
    printf("\n");  
}