1. 验证性实验

1．二叉链表的验证

（1）在二叉链表类模板中增加函数成员 CountLeaf ()，统计二叉树中叶子结点的数目。

**主要思路：递归判断当前结点是否有孩子，没有即为叶子结点。**

在类中增加两个成员函数

int countLeaf(const BinTreeNode<ElemType> \*p);

int countLeaf();

函数定义如下

template <class ElemType>

int BinaryTree<ElemType>::countLeaf(const BinTreeNode<ElemType> \*p)

{

if (p!=NULL)

{

int lcount=0,rcount=0;

if(p->leftChild==NULL&&p->rightChild==NULL)

{

return 1;

}

else

{

lcount=countLeaf(p->leftChild);

rcount=countLeaf(p->rightChild);

return lcount+rcount;

}

}

}

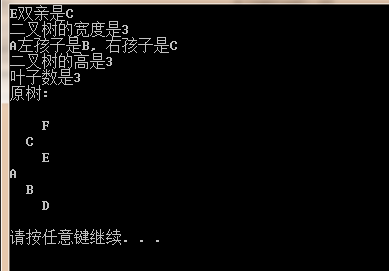
template <class ElemType>

int BinaryTree<ElemType>::countLeaf()

{

return countLeaf(root);

}



（2）在二叉链表类模板中增加函数成员 Revolute()，实现二叉树中所有结点的左右子树交换。

**主要思路：送根结点开始依次递归交换。**

在类中增加两个成员函数

void Revolute(BinTreeNode<ElemType> \*p);

void Revolute();

函数定义如下

template <class ElemType>

void BinaryTree<ElemType>::Revolute(BinTreeNode<ElemType> \*p)

{

if(p->leftChild!=NULL&&p->rightChild!=NULL)

{

BinTreeNode<ElemType> \*temp=p->leftChild;

p->leftChild=p->rightChild;

p->rightChild=temp;

Revolute(p->leftChild);

Revolute(p->rightChild);

}

}

template <class ElemType>

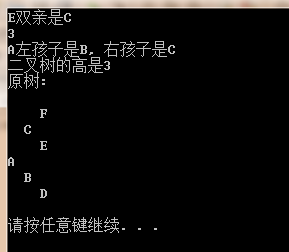
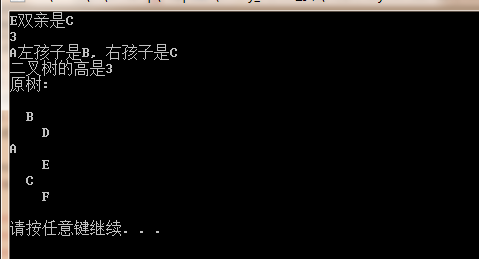
void BinaryTree<ElemType>::Revolute()

{

Revolute(root);

}

转置前的矩阵



转置后的矩阵

（3）在二叉链表类模板中增加函数成员 CountBreadth ()，统计二叉树的最大宽度（二 叉树的最大宽度是指二叉树所有层中结点个数的最大值）。

**主要思路：用一个数组记录每层的宽度，判断当前结点是否为根结点，如果是，判断其是否为空，可得出第一层的结点数。然后看当前结点是否有孩子结点，有的话将下一层的结点数+1；并且递归。最后可以得出所有层的结点数，有循环找出其最大值，即为树的宽度。**

在类中增加一个成员变量以及两个成员函数

int wide[10]={0,0,0,0,0,0,0,0,0,0};

int CountBreadth (BinTreeNode<ElemType> \*p,int floor);

int CountBreadth ();

函数定义如下

template <class ElemType>

int BinaryTree<ElemType>::CountBreadth ()

{

int max;

CountBreadth(root,0);

for(int i=0;i<=8;i++)

{

max=wide[0];

if(wide[i+1]>max) max=wide[i+1];

}

return max;

}

template <class ElemType>

int BinaryTree<ElemType>::CountBreadth (BinTreeNode<ElemType> \*p,int floor)

{

if(root==NULL) wide[0]=0;

else if(p==root) wide[0]=1;

if(p->leftChild!=NULL&&p->rightChild==NULL)

{

wide[floor++]++;

CountBreadth(p->leftChild,floor);

}

if(p->leftChild==NULL&&p->rightChild!=NULL)

{

wide[floor++]++;

CountBreadth(p->rightChild,floor);

}

else if(p->leftChild!=NULL&&p->rightChild!=NULL)

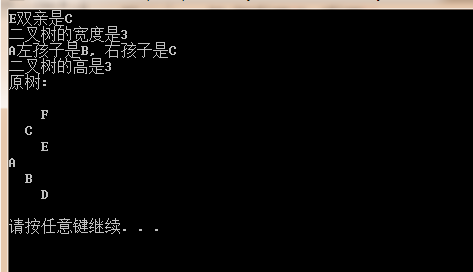
{

wide[floor]++;

CountBreadth(p->rightChild,floor);

}

}



（4）在二叉链表类模板中，增加函数成员 NonRecurringInOrder()，实现非递归中序遍历 二叉树。

**思路：用while循环和一个栈还储存孩子并进行遍历**

template <class ElemType>

void BinaryTree<ElemType>::NonRecurringInOrder(BinTreeNode<ElemType>\*r,void (\*Visit)(const ElemType &))const;

// 操作结果：中序遍历以r为根的二叉树

{

stack s;

BinTreeNode<ElemType> \*p=root;

while(p!=NULL)

{while(p!=NULL)

{

s.push(p);

p=p->leftChild;

}

if(!s.empty())

{

p=s.top();

cout<<p->data<<endl;

s.pop;

p=p->rightChild;

}

}

}