5.14 用反例证明：由二叉树的先序序列和后序序列不能唯一确定一棵二叉树。

先序：ABC

后序：CBA

有两棵树

5.15 设计算法以输出二叉树中先序序列的前 k（k>0）个结点的值。

void preTraverse(btNode\* T,int k)

{

static int i = 0;

if (T&&i!=k)

{

i++;

if (i <= K)

{

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

}

preTraverse(T->lChild,k); //先序遍历左子树

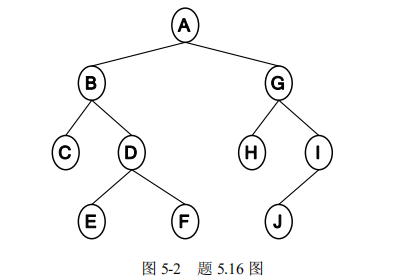
preTraverse(T->rChild,k); //先序遍历右子树

}

}

5.16 设计算法按中序次序依次输出各结点的值及其对应的序号。例如，下图中的二叉树

的输出结果是(C,1) (B,2) (E,3) (D,4) (F,5) (A,6) (H,7) (G,8) (J9) (I,10)。



void inTraverse(btNode\* T)

{

static int i = 0;

if (T)

{

i++;

inTraverse(T->lChild); //中序遍历左子树

cout <<"("<< T->data << ","<<i<<")"<<endl;

//访问根结点打印当前结点元素值

inTraverse(T->rChild); //中序遍历右子树

}

}

5.17 设计算法以输出每个结点到根结点之间的路径上的所有结点的值。

void Allpath(BtNode \*T,ElemType path[],int pathlen)

{ //先序遍历方法输出每个叶子结点到根结点的路径序列

if(T)

{ if (b->lchild==NULL && b->rchild==NULL)

{ //输出栈中所有结点值

printf("%c->",b->data);

for (int i = pathlen-1;i>0;i--)

printf("%c->",path[i]);

printf("%c\n",path[0]);

}

else

{ path[pathlen++]=b->data;

Allpath(b->lchild, path, pathlen);

Allpath(b->rchild, path, pathlen);

}

}

}

5.22 分别设计出先序、中序和后序线索化算法。

void PreOrder(BiTreeNode\* Root)

{

if (Root == NULL)

{

return;

}

BiTreeNode\* pCur = Root;

while (pCur != NULL)

{

while (pCur->pLeft != NULL && pCur->Ltag == Link)//找到最左边的节点,左标记一直为Link

{

cout << pCur->\_data << ' ';

pCur = pCur->pLeft;

}

//到这来，左边的的节点还没有访问

cout << pCur->\_data << ' ';

if (pCur->Ltag == Thread)//遇到线索 就看右节点

{

pCur = pCur->pRight;

}

while (pCur != NULL)//循环右节点

{

if (pCur->pLeft != NULL && pCur->Ltag == Link)//遇到左节点存在 ， 则访问

{

break;

}

cout << pCur->\_data << ' ';

pCur = pCur->pRight;

}

}

}

void InOrderThreading(BiTreeNode\*& Root)

{

if (Root == NULL)

{

return;

}

InOrderThreading(Root->pLeft); // 左

if (Root->pLeft == NULL) //根

{

Root->Ltag = Thread;

Root->pLeft = Prev;

}

if (Prev != NULL && Prev->pRight == NULL)

{

Prev->pRight = Root;

Prev->Rtag = Thread;

}

Prev = Root;

InOrderThreading(Root->pRight); //右

}

void InOrder(BiTreeNode\* Root)

{

if (Root == NULL)

{

return;

}

BiTreeNode\* pCur = Root;

while (pCur )

{

while (pCur->Ltag == Link) //找最左边的节点

{

pCur = pCur->pLeft;

}

cout << pCur->\_data << ' ';

while ( pCur && pCur->Rtag == Thread )//找中序后继节点

{

pCur = pCur->pRight;

cout << pCur->\_data << ' ';

}

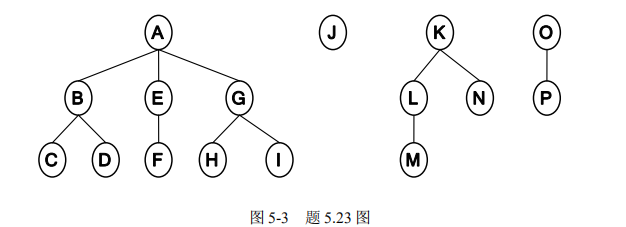
//没有后继，有右子树

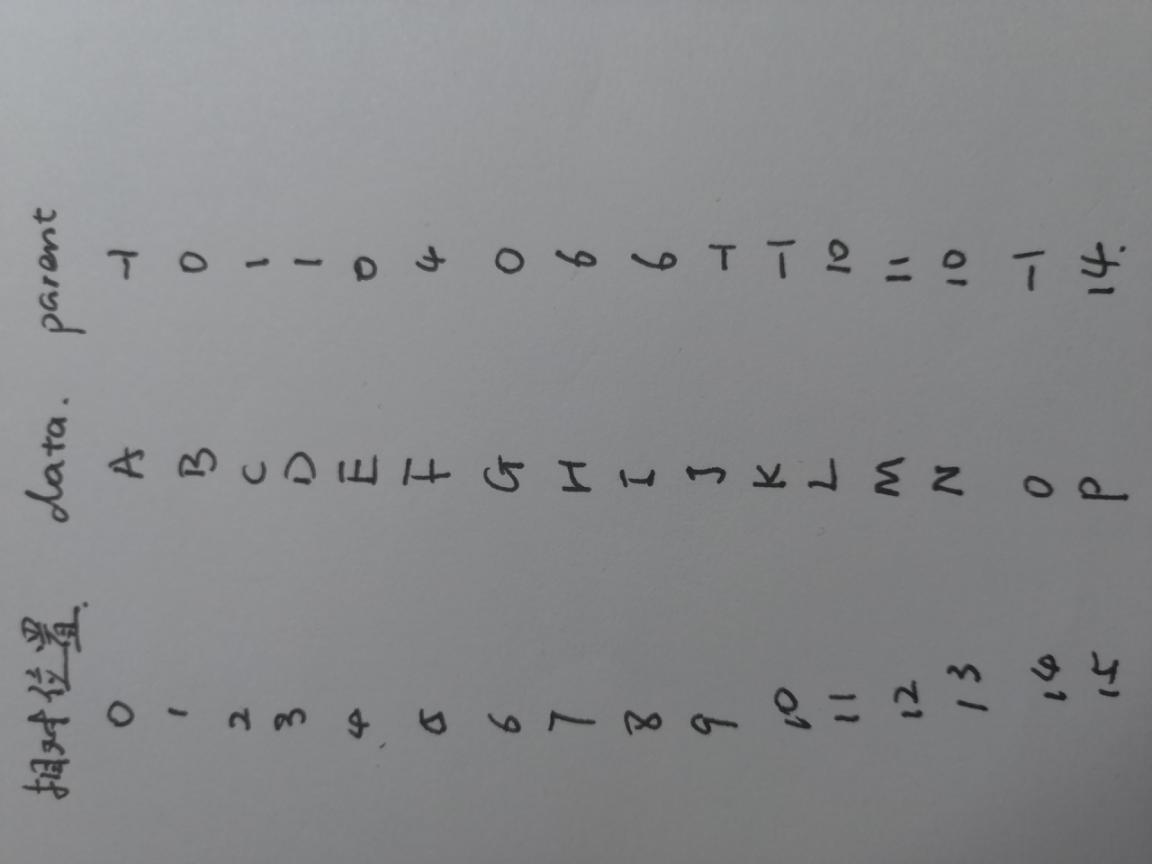
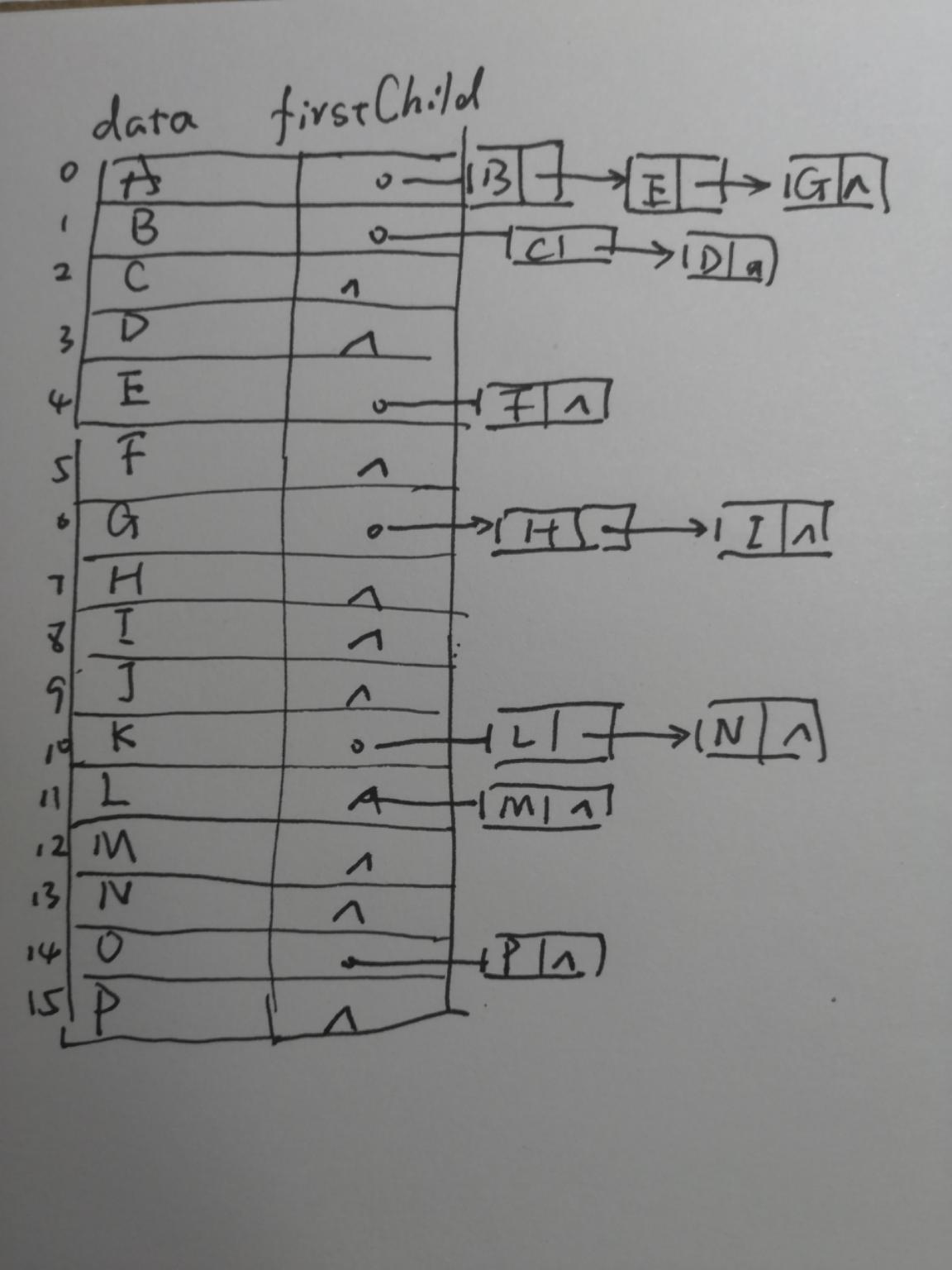
pCur = pCur->pRight;

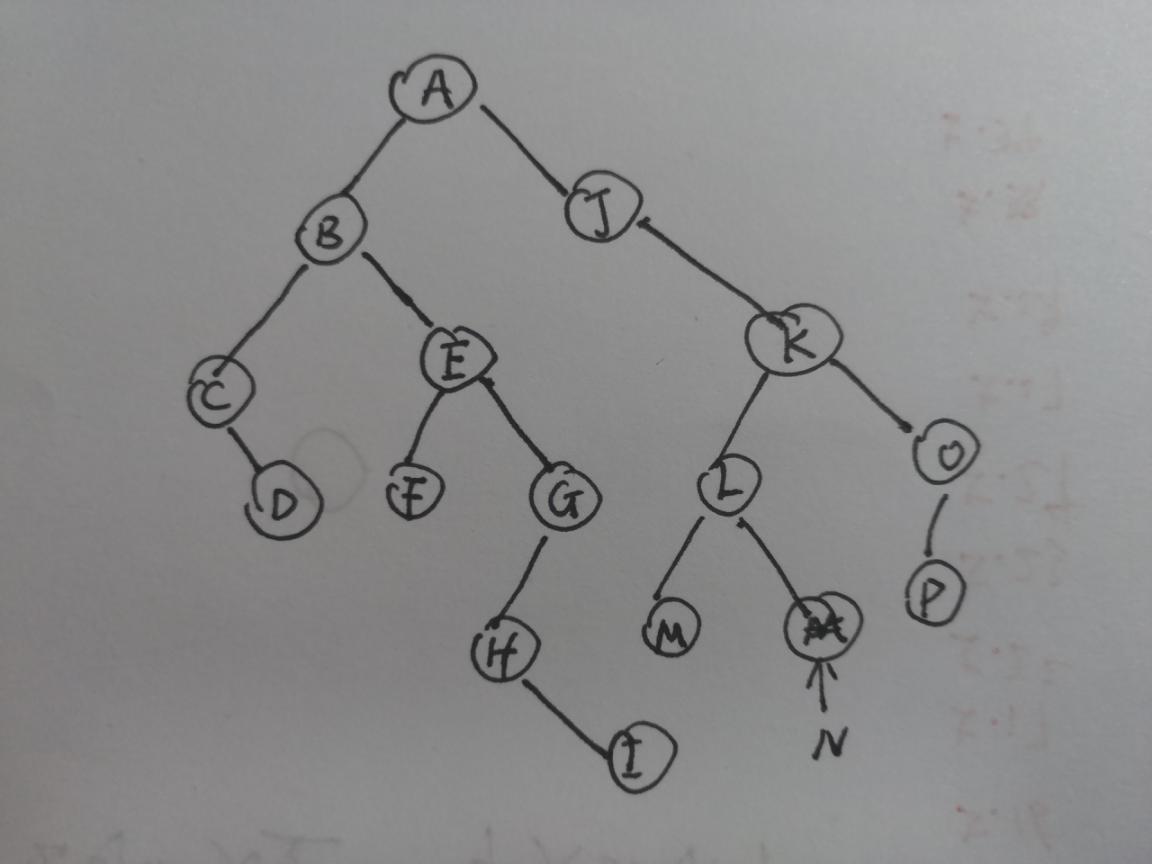
}

}

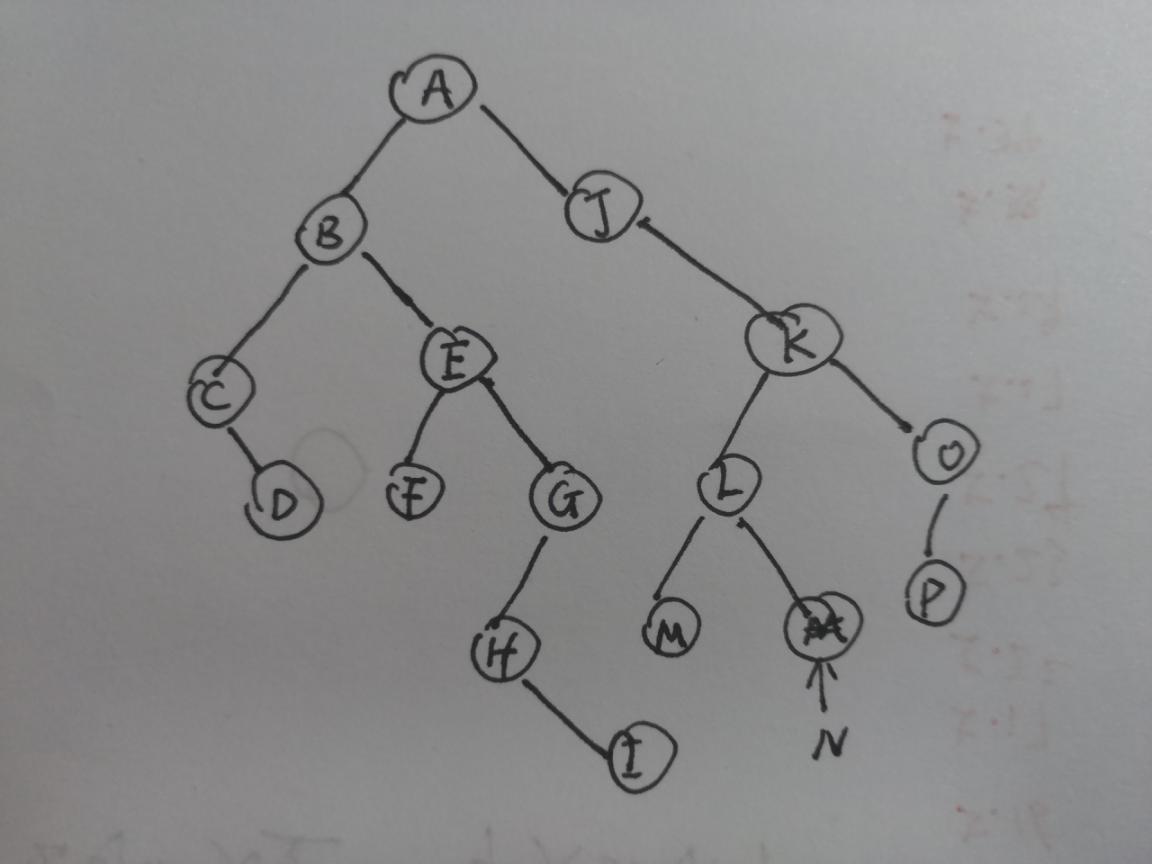
5.23 分别画出下图所示的森林的双亲表示形式、孩子链表表示形式和二叉链表表示形式。







5.24 将图 5-3 中的森林转换为对应的二叉树。



5.27\*设计算法将一个以孩子链表形式表示的森林转换为二叉链表形式。

void treeToBtree(Tree \*tree,BtNode \*&btnode,int i)

{//树转二叉树

if(tree != NULL)

{

BtNode node1 = tree[i];

btnode = node1;//二叉树的头结点等于树的头结点

listnode \*list = node1.firstChild;//该结点的孩子结点

BtNode \*node2 = bt->lChild;//二叉树的左孩子结点

while(node3)//把该层的树全部转为二叉树

{

treeToBtree(tree,q,list->data);//若该结点以及该结点的子孙结点转为二叉树

list = list->next;//指向树结点下一个孩子结点

q = q->rChild;//指向二叉树的右孩子结点

}

}

}

void forestToBtree(Tree \*Forest[MAX],int n,BtNode \*&fn)

{

//森林转二叉树

int i = 0;

BtNode \*q = bt;//指向二叉树的每一个右孩子结点

while(i < fn)

{

treeToBtree(Forest[i],q,0);//将每一个森林转为二叉树

i++;

q = q->rChild;//连接各二叉树

}

}

5.29 设计算法按先序次序输出森林中每个结点的值及其对应的层次数。

5.34 以数据集合{4,6,8,10,12,15,18,20,22}中的元素为叶子结点的权值构造一棵哈夫曼树，

并计算其带权路径长度。

