P258 4

Parent(i) = (i + (d-2))/d

P296 1

template<class K,class E>

void BST<K,E>::Delete(pair<K,E>)

{

//找到要删除节点的父节点

TreeNode<pair<K,E> >\* parentNode = root;

while(parentNode)

{

bool flag = 0;

//判断当前节点的子节点是否为要删除节点

if(k==parentNode->leftChild->data.first)

{

flag = 0;

break;

}

if(k==parentNode->rightChild->data.first)

{

flag = 1;

break;

}

//向下迭代

if(k<parentNode->data.first)

parentNode = parentNode->leftChild;

else(k>parentNode->data.first)

parentNode = parentNode->rightChild;

}

//找到要删除的节点

if(flag ==0)

TreeNode<pair<K,E> >\* currentNode = parentNode->leftChild;

else

TreeNode<pair<K,E> >\* currentNode = parentNode->rightChild;

//判断要删除节点是否有左子树

if(currentNode->leftChild)

{

//判断要删除节点是否有右子树

if(currentNode->rightChild)

{

if(!flag)

{

parentNode->leftChild = currentNode->leftChild;

currentNode->leftNode->rightChild = currentNode->rightChild;

delete currentNode;

}

else

{

parentNode->rightChild = currentNode->leftChild;

currentNode->leftNode->rightChild = currentNode->rightChild;

delete currentNode;

}

}

else

{

if(!flag)

{

parentNode->leftChild = currentNode->leftChild;

delete currentNode;

}

else

{

parent->rightChild = currentNode->leftChild;

delete currentNode;

}

}

}

//判断要删除节点是否有右子树

else if(currentNode->rightChild)

{

if(!flag)

{

parentNode->leftChild = currentNode->rightChild;

delete currentNode;

}

else

{

parentNode->rightChild = currentNode->rightChild;

delete currentNode;

}

}

else

{

if(!flag)

{

parentNode->leftChild = NULL;

delete currentNode;

}

else

{

parentNode->rightChild = NULL;

delete currentNode;

}

}

}

P323 2

可以唯一确定

结合这两个遍历序列，我们可以这样重建树：

1、在后序序列中找到根节点。

2、在中序序列中找到根节点，这将中序序列分为两部分，左侧为左子树的中序序列，右侧为右子树的中序序列。

3、在后序序列中也对应地找到左子树和右子树的后序序列。

4、递归地对左右子树进行相同的处理。

P323 4

可以唯一确定

结合这两个遍历序列，可以这样重建树：

1、首先，层次遍历的第一个元素是树的根节点。

2、在中序遍历序列中找到根节点，可以将序列分为两部分：左侧是左子树的中序遍历，右侧是右子树的中序遍历。

3、使用层次遍历序列中的信息，可以确定哪些节点属于左子树，哪些属于右子树。

4、对于左子树和右子树，重复上述步骤，递归地构建整棵树。