（1）

template <typename Key, typename E>

E BST<Key,E>::remove\_norec(const Key& k){

BSTNode<Key, E>\* it = root;

BSTNode<Key, E>\* pit; //it的父节点，为了修改pit的儿子指向

bool isleft = true; //root 是pit的左儿子还是右儿子，用于确定pit是setleft还是setright

//先找到要删的节点it和it的父节点

while (it && k != it->key()){

pit = it;

if (k < it->key()){

isleft = true;

it = it->left();

}

if (it && k > it->key()){

isleft = false;

it = it->right();

}

}

//找到需要被删除的点it 了

if (it == NULL) return NULL;

BSTNode<Key, E> \*temp = it; //先存下it，替换it后再删掉temp

E val = it->element(); //存下返回值

if (it->left() == NULL){

//没有左儿子，it替换为右儿子（或空）

it = it->right();

isleft ? pit->setLeft(it) : pit->setRight(it); //把pit的儿子换成新儿子

delete temp;

}

else if (it->right() == NULL){

//有左儿子没有右儿子，it替换为左儿子

it = it->left();

isleft ? pit->setLeft(it) : pit->setRight(it); //把pit的儿子换成新儿子

delete temp;

}

else{ //两个儿子都有

//修改it的key和element为右子树中最小值节点S（这样就不delete temp了），并把S父节点的左儿子设为S的右儿子（或NULL），最后删掉S

//先找到S

BSTNode<Key, E> \*PS = it; //S的父节点

BSTNode<Key, E> \*S = it->right();

while (S->left()){

PS = S;

S = S->left();

}

//找到S了，把PS的左儿子设为S的右儿子，并用S替换it

PS->setLeft(S->right());

it->setKey(S->key());

it->setElement(S->element());

//it指针没有变，因此不用改pit

//最后删掉S

delete S;

}

nodecount--;

return val;

}

（2）数学归纳法。

* n=0时显然成立;
* 假设n=k时成立，即有成立。
* 当n=k+1时，假设新增的内部节点的深度为L，

则 （先减去原来叶节点的深度，再加上新的叶节点的深度）

于是有

即

综上，原式得证。

（3）

// 给定前序遍历和中序遍历可以唯一的确定一棵树，因此只需比较两棵树的前序和中序遍历结果是否都一致即可。

template<typename E>

void preOrder(BinNode<E>\* p, char\* s){

char \*tmp = new char[1];

if(!p) return;

itoa(p->element(), tmp, 10);

strcat(s, tmp);

preOrder(p->left(), s);

preOrder(p->right(), s);

}

template<typename E>

void inOrder(BinNode<E>\* p, char\* s){

char \*tmp = new char[1];

if(!p) return;

inOrder(p->left(), s);

itoa(p->element(), tmp, 10);

strcat(s, tmp);

inOrder(p->right(), s);

}

template<typename E>

bool comp(BinNode<E>\* p1, BinNode<E>\* p2){

char s1[1024], s2[1024];

memset(s1, 0, 1024);

memset(s2, 0, 1024);

preOrder(p1, s1);

preOrder(p2, s2);

if (strcmp(s1, s2) != 0)

//前序遍历不一致

return false;

//前序遍历一致，再比较中序遍历

memset(s1, 0, 1024);

memset(s2, 0, 1024);

inOrder(p1, s1);

inOrder(p2, s2);

if (strcmp(s1, s2) != 0)

//中序遍历不一致

return false;

//中序遍历也一致，说明两棵树相同

return true;

}