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();

}

}

//it 是需要被删除的点

if (it == NULL) return NULL;

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

}