实验三 查找

计算机工程与科学学院 18120189 林艺珺

（一）验证性试验

1.有序表查找的验证

有三个整数数组a[]、b[]和c[]，各有aNum、bNum和cNum个元素，而且三者都已经从小到大排列。要求编写程序找出最小的共同元素在三个数组中出现的位置，若没有共同元素，请显示适当信息。

编写程序：

#include <iostream>

using namespace std;

int main()

{

int a[1000], b[1000], c[1000];

int aNum, bNum, cNum;

int i = 0, j = 0, k = 0;

int common;

cout << endl;

cout << "!!!Common smallest element!!!" << endl;

cout << endl;

cout << "Input aNum bNum cNum" << endl;

cin >> aNum >> bNum >> cNum;

if(aNum == i || bNum == j || cNum == k ||

aNum <= 0 || bNum <= 0 || cNum <= 0)

{

cout << "Illegal Num." << endl;

return 1;

}

cout << "Input a" << endl;

for (i = 0; i < aNum; i++)

{

cin >> a[i];

}

cout << "Input b" << endl;

for (j = 0; j < bNum; j++)

{

cin >> b[j];

}

cout << "Input c" << endl;

for (k = 0; k < cNum; k++)

{

cin >> c[k];

}

i = 0, j = 0, k = 0;

while(i < aNum && j < bNum && k < cNum)

{

if(a[i] < b[j]) i++;

else if(b[j] < c[k]) j++;

else if(c[k] < a[i]) k++;

else

{

common = a[i];

cout << "The common element is " << common << ". " << endl;

cout << "a at " << i+1 << ", ";

cout << "b at " << j+1 << ", ";

cout << "c at " << k+1 << "." << endl;

return 0;

}

}

if(aNum == i || bNum == j || cNum == k ||

aNum <= 0 || bNum <= 0 || cNum <= 0)

{

cout << "No common Num." << endl;

return 1;

}

}

2.二叉排序树的验证

对于在二叉排序树上删除结点的问题，教材介绍4种算法，并实现了其中第一种算法，现在要求选择后面三种算法之一完成，并用多组测试数据对这四种算法进行性能测试，并分析比较它们的查找性能。

在BinarySortTree.h的函数声明中添加：

void Delete2(BinTreeNode<ElemType> \*&p); // 删除p指向的结点

bool Delete2(const ElemType &key); // 删除关键字为key的数据元素

在BinaryTree.h的函数实现中添加：

template <class ElemType>

void BinarySortTree<ElemType>::Delete2(BinTreeNode<ElemType> \*&p)

// 操作结果: 删除p指向的结点

{

BinTreeNode<ElemType> \*tmpPtr, \*tmpF;

if (p->leftChild == NULL && p->rightChild == NULL) { // p为叶结点

delete p;

p = NULL;

}

else if (p->leftChild == NULL) { // p只有左子树为空

tmpPtr = p;

p = p->rightChild;

delete tmpPtr;

}

else if (p->rightChild == NULL) { // p只有右子树非空

tmpPtr = p;

p = p->leftChild;

delete tmpPtr;

}

else { // p左右子非空

tmpF = p;

tmpPtr = p->rightChild;

while (tmpPtr->leftChild != NULL) { // 查找p在中序序列中直接前驱tmpPtr及其双亲tmpF,直到tmpPtr右子树为空

tmpF = tmpPtr;

tmpPtr = tmpPtr->leftChild;

}

p->data = tmpPtr->data;

// 将tmpPtr指向结点的数据元素值赋值给tmpF指向结点的数据元素值

// 删除tmpPtr指向的结点

if (tmpF->leftChild == tmpPtr) // 删除tmpF的左孩子

Delete(tmpF->leftChild);

else // 删除tmpF的右孩子

Delete(tmpF->rightChild);

}

}

template <class ElemType>

bool BinarySortTree<ElemType>::Delete2(const ElemType &key)

// 操作结果: 删除关键字为key的数据元素

{

BinTreeNode<ElemType> \*p, \*f;

p = Find(key, f);

if ( p == NULL) // 查找失败, 删除失败

return false;

else // 查找成功, 插入失败

if (f == NULL) // 被删除结点为根结点

Delete2(p);

else if (key < f->data) // elem.key更小,删除f的左孩子

Delete2(f->leftChild);

else // elem.key更大, 删除f的右孩子

Delete2(f->rightChild);

return true;

}

在测试文件中，使用以下内容进行函数调用并输出结果：

BinarySortTree<int> bt;

int elem[] = {39, 68, 11, 23, 46, 34, 75, 86, 8, 71};

int n = 10;

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

bt.Insert(elem[i]); // 根据elem数组中的元素构造二叉排序树

cout << endl << "输入删除元素的值:";

cin >> x;

bt.Delete2(x);