**中国计量大学实验报告**

实验课程：《算法与数据结构》 实验名称： 查 找

班 级： 15计算机1 / 2 实验日期：

实验角色

|  |  |  |
| --- | --- | --- |
|  | Programmer | Tester / Document Writer |
| 学号 | 1500308118 | 1500308120 |
| 姓名 | 夏佳能 | 阮启铭 |
| 实验成绩 |  |  |

**实验目的及要求：**

一、 实验目的

1、掌握顺序表和有序表的查找方法。

2、掌握动态查找表二叉排序树的构造和查找方法。

二、 实验要求

1. 构建一个静态查找表，实现折半查找；

2. 构建一个动态查找表，实现查找、插入和删除操作。

三、实验角色

1、Programmer: ADT设计、算法设计及实现

2、Document Writer: 实验所涉资料的整理、录入及排版

3、Tester: 确定测试用例并负责程序测试，测试用例不少于3种

**关键算法说明**

这一部分描述解决问题所用到的关键算法，可用伪代码、代码或框图表示，目的是让读者在短时间内清楚地理解作者解决问题的整体思路。因此，表达方式必须比源代码更通俗易懂。

ADT dynamicSearchTable {

数据对象：D是具有相同特性的数据元素的集合。各个数据元素均含有类型相同，可唯一标识数据元素的关键字。

数据关系R：数据元素同属一个集合。

基本操作：

int IterBiSearch(int data[], const int x, int beg, int last);

操作结果：在静态表中找到变量x

bool searchBinaryTree(binaryTree T, binaryTree parents, int data, binaryTree &res)；

操作结果：在动态表中查找变量data

bool deleteBinaryTree (binaryTree &T, int data)；

操作结果：在动态表中删除data

bool Delete (binaryTree &p)；

操作结果：删除节点p

void createBinaryTree(binaryTree &T, int a[], int number)；

操作结果：根据数组a创建动态表

bool binaryTreeInsert(binaryTree &p, int element)；

操作结果：向动态表中插入元素element

}

**测试用例表**

紧扣实验要求，设计（并填写）一张测试用例表。每个测试用例一般应包括下列内容：

1. 测试输入：设计一组输入数据；
2. 测试目的：设计该输入的目的在于测试程序在哪方面可能存在漏洞；
3. 预期输出：若程序正确，应该输出的结果；
4. 实际输出：程序运行后，实际得到的输出结果；
5. 错误原因：如果实际输出与预期输出不符，给出产生错误的可能原因；
6. 当前状态： “通过” 或者“待修改” 两种状态之一。

【注意】测试时，不是提供简单的几组数据让程序容易通过，从而宣称该程序是正确的，而是精心设计“刁难”的数据，尽可能让程序暴露问题，这样才能真正帮助程序员完成正确的程序。





**实验体会与收获**

**附：源代码**

编程风格应该清晰易理解，注释行应意义明确，注释行不少于程序行的15%。

【以下为源代码】

#include <iostream>

using namespace std;

//定义树的节点类型

typedef struct binaryTreeNode {

int data; //数据类型为int

struct binaryTreeNode \*leftChild,\*rightChild;

}binaryTreeNode, \*binaryTree;

//添加节点的方法

bool binaryTreeInsert(binaryTree &p, int element){

if(NULL == p) // 空树

{

p = new binaryTreeNode;

p->data = element;

p->leftChild = p->rightChild = NULL;

return true;

}

if(element == p->data) // 表中不能有相等的值

return false;

if(element < p->data) // 递归

return binaryTreeInsert(p->leftChild, element);

return binaryTreeInsert(p->rightChild, element); // 递归

}

//根据数组建立二叉树二叉树

void createBinaryTree(binaryTree &T, int a[], int number) {

T = NULL;

for(int i = 0; i < number; i++) //遍历

{

binaryTreeInsert(T, a[i]);

}

}

//删除操作

bool Delete (binaryTree &p) {

if (!p->rightChild) { //右子树为空

p = p-> leftChild;

} else if (!p->leftChild) { //左子树为空

p = p->rightChild;

} else { //左右子树都不为空

binaryTree f = p;

f = f->leftChild;

while (f->rightChild) {

f = f->rightChild;

}

binaryTree node = p -> leftChild;

delete p->leftChild;

f->rightChild = p ->rightChild;

p->data = node->data;

p->leftChild = node->leftChild;

p->rightChild = f;

}

return true;

}

//删除对应的节点

bool deleteBinaryTree (binaryTree &T, int data) {

if (!T) {

return false;

} else {

if (T->data == data) {

return Delete(T);

} else if (T->data > data) {

return deleteBinaryTree(T->leftChild, data); //递归

} else {

return deleteBinaryTree(T->rightChild, data);

}

}

}

//查询二叉排序树

bool searchBinaryTree(binaryTree T, binaryTree parents, int data, binaryTree &res) {

if (!T)

{

binaryTreeInsert(T, data);

return false;

} else {

if (data == T->data)

{

res = T;

return true;

} else if (data < T->data) {

return searchBinaryTree(T -> leftChild, T, data, res); //递归

} else {

return searchBinaryTree(T -> rightChild, T, data, res);

}

}

}

//MARK: 递归方法遍历二叉树

//先序遍历二叉树

void preOrderTraverse(binaryTree T){

if(T){

cout << T -> data;

preOrderTraverse(T->leftChild);

preOrderTraverse(T->rightChild);

}

}

//中序遍历二叉树

void inOrderTraverse(binaryTree T){

if (T) {

inOrderTraverse(T -> leftChild);

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

inOrderTraverse(T -> rightChild);

}

}

//后序遍历二叉树

void postOrderTraverse (binaryTree T) {

if (T) {

postOrderTraverse(T -> leftChild);

postOrderTraverse(T -> rightChild);

cout << T -> data;

}

}

//递归法,折半查找

int IterBiSearch(int data[], const int x, int beg, int last)

{

int mid = -1;

mid = (beg + last) / 2;

if (x == data[mid])

return mid + 1;

else if (x < data[mid])

return IterBiSearch(data, x, beg, mid - 1);

else if (x > data[mid])

return IterBiSearch(data, x, mid + 1, last);

return -1;

}

//折半查找的实现

void ShowIterBiSearch ()

{

cout << "按大小输入10个数字";

int data1[10] = {0};

int siz = sizeof(data1)/sizeof(int);

for (int i = 0; i < siz; i++) {

std::cin >> data1[i];

}

cout << "The array is : " << " ";

for (int i = 0; i < siz; i++) {

cout << data1[i];

}

cout << "\n";

int searchObject = 0;

cout << "Please enter your searchObject: " << endl;

cin >> searchObject;

cout << IterBiSearch(data1, searchObject, 0, siz);

}

int main(int argc, char const \*argv[])

{

int flag;

cout << "输入1代表静态查找，输入2代表动态查找" << endl;

cin >> flag;

switch (flag) {

case 1:

ShowIterBiSearch();

break;

case 2:

cout << "请输入元素集合的数量" << endl;

int number = 0;

cin >> number;

int\* a = (int \*) malloc ( sizeof(int) \* number );

cout << "请输入元素" << endl;

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

{

cin >> a[i];

}

cout << "请确认刚刚输入的元素集合：" << endl;

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

{

cout << a[i] << " ";

}

cout << "正在创建动态表..." << endl;

binaryTree Tree;

createBinaryTree(Tree, a, number);

cout << "创建成功" << endl << "以下是中序遍历的结果" << endl;

inOrderTraverse(Tree);

cout << endl;

while (true) {

cout << "请输入要查询的数值" << endl;

int x;

cin >> x;

binaryTree searchTree;

if (!searchBinaryTree(Tree, NULL, x, searchTree)) {

cout << "动态表中没有该数值 ..." << endl << "成功添加该数值！" << endl;

binaryTreeInsert(Tree, x);

cout << "输入1可重新遍历该表 \nor press any number to pass" << endl;

int d;

cin >> d;

if (d == 1) {

inOrderTraverse(Tree); cout << endl;

}

} else {

cout << "查找成功！\n" << "如果要删除该数值请输入1 \nor press any number to pass " << endl;

int c;

cin >> c;

switch (c) {

case 1:

if (deleteBinaryTree(Tree, x))

cout << "删除成功" << endl;

else

cout << "删除失败" << endl;

break;

default:

break;

}

cout << "输入1可重新遍历该表 \nor press any number to pass" << endl;

int d;

cin >> d;

if (d == 1) {

inOrderTraverse(Tree); cout << endl;

}

}

}

break;

}

return 0;

}