天津大学

数据结构实验报告

实验名称：查找

学院名称 智能与计算学部

专　　业 软件工程

学生姓名 陈昊昆

学　　号 3021001196

年　　级 2021 级

班　　级 软工 3 班

时　　间 2023 年 5 月 20 日

1. 实验内容

1.实现顺序查找表和它的二分查找方法。

顺序查找表类 SSTable ，采用线性表存储，完成以下功能：

顺序查找表的建立 SSTable(int, int*) ，

参数列表为：

元素数目 int

元素列表 int* ，输入数据已按升序排列，不包含重复值

顺序查找表的二分查找方法 int binSearch(int) ，参数为要查找的元素值，若找到元素， 返回对应元素在顺序查找表中的下标（从 0 开始）；若未找到元素，返回 -1

其它类初始化、销毁、私有属性存取等基本功能

2.实现二叉排序树和它的查找、新增节点、删除节点方法。

二叉排序树节点类 BSTreeNode 、二叉排 序树类 BSTree ，完成以下功能：

二叉排序树的建立 BSTree(int, int*) ，参数列表为： 元素数目 int 元素列表 int* ，输入数据不包含重复值

二叉排序树的遍历输出 string printTree() ，返回二叉排序树的前序序列 string 类型字符 串

二叉排序树的查找 bool searchNode(int) ，参数为要查找的元素值。若找到返回 true ，未 找到返回 false

二叉排序树添加节点 bool addNode(int) ，参数为要添加的元素值。若添加成功返回 true ，若该元素与已有元素值重复，返回 false

二叉排序树删除节点 bool deleteNode(int) ，参数为要删除的元素值。若删除成功返回 true ，若该元素不存在，返回 false 其它类初始化、销毁、私有属性存取等基本功能

提示：可以使用递归方法，自行添加需要用到的辅助函数，根据自己的习惯自定义实现算法的 中间函数，但请注意不要出现编译错误。

2. 程序实现

```
#include "mySearch.h"
#include <iostream>
#include <string>
#include <stack>


using namespace std;

//顺序查找表的建立
```

```
SSTable::SSTable(int length,int* origin)
{
    this->length =length;
    this->origin = origin;
}

//其它类初始化、销毁、私有属性存取等基本功能
SSTable::~SSTable()
{
    length = 0;
    origin = nullptr;
}

int SSTable::getLength()
{
    return length;
}

int* SSTable::getOrigin()
{
    return origin;
}

void SSTable::setLength(int length)
{
    this->length =length;
}

void SSTable::setOrigin(int* origin)
{
    this->origin = origin;
}

//顺序查找表的二分查找方法
int SSTable::binSearch(int val)
{
    int low = 1;
    int high = length;
    int mid= (low + high) / 2;
    while(low <= high){
        mid= (low + high) / 2;
        if(val == origin[mid]) return mid;
        else if(val < origin[mid]) {
            high = mid - 1;
```

```
        }
        else low = mid + 1;
    }
    return -1;
}


//二叉排序树节点类 BSTreeNode
BSTreeNode::BSTreeNode(int data)
{
    this->data = data;
}

BSTreeNode::BSTreeNode(int data, BSTreeNode* lchild, BSTreeNode*
rchild)
{
    this->data = data;
    this->lchild = lchild;
    this->rchild = rchild;
}

BSTreeNode::~BSTreeNode()
{
    this->data = 0;
    this->lchild = nullptr;
    this->rchild = nullptr;
}

int BSTreeNode::getData()
{
    return data;
}

BSTreeNode* BSTreeNode::getLChild()
{
    return lchild;
}

BSTreeNode* BSTreeNode::getRChild()
{
    return rchild;
}

void BSTreeNode::setData(int data)
{
```

```
    this->data = data;
}

void BSTreeNode::setLChild(BSTreeNode* lchild)
{
    this->lchild = lchild;
}

void BSTreeNode::setRChild(BSTreeNode* rchild)
{
    this->rchild = rchild;
}


//二叉排序树的建立 BSTree(int, int*)
BSTree::BSTree()
{
    num = 0;
    root = nullptr;
}

BSTree::BSTree(int num, int* data)
{
    this->num = num;
    root = new BSTreeNode(data[0], nullptr, nullptr);
    for(int i = 1; i < num; i++){
        BSTreeNode* p = root;
        BSTreeNode* q = p;
        int flag = 0;
        while(p != nullptr) {
            q = p;
            if(data[i] < p->getData()) {
                flag = 1;
                if(p->getLChild() == nullptr) p = nullptr;
                else p = p->getLChild();
            }
            else {
                flag = 2;
                if(p->getRChild() == nullptr) p = nullptr;
                else p = p->getRChild();
            }
        }
        BSTreeNode* lchild = nullptr;
        BSTreeNode* rchild = nullptr;
        BSTreeNode* bs = new BSTreeNode(data[i], lchild, rchild);
```

```
        if(flag == 1){
            q->setLChild(bs);
        }
        else if(flag == 2){
            q->setRChild(bs);
        }
    }
}

BSTree::~BSTree()
{
    num = 0;
    root = nullptr;
}

int BSTree::getNum()
{
    return num;
}

BSTreeNode* BSTree::getRoot()
{
    return root;
}

void BSTree::setNum(int num)
{
    this->num = num;
}

void BSTree::setRoot(BSTreeNode* root)
{
    this->root = root;
}


//二叉排序树的遍历输出 string printTree()
void printTemp(BSTreeNode* root, string& result)
{
    if(root != nullptr) {
            result += to_string(root->getData()) + " ";
            printTemp(root->getLChild(), result);
            printTemp(root->getRChild(), result);
        }
}
```

```
string BSTree::printTree()
{
    string result;
    printTemp(root, result);
    return result;
}


//二叉排序树的查找 bool searchNode(int)
bool BSTree::searchNode(int val){
    BSTreeNode* p = root;
    while(p != nullptr){
        if(val == p->getData()) return true;
        else if(val < p->getData()) p = p->getLChild();
        else p = p->getRChild();
    }
    return false;
}


//二叉排序树添加节点 bool addNode(int)
bool BSTree::addNode(int val)
{
    BSTreeNode* p = root;
    BSTreeNode* q = p;
    int flag = 0;
    while(p != nullptr) {
        q = p;
        if(val == p->getData()) return false;
        else if(val < p->getData()) {
            flag = 1;
            p = p->getLChild();
        }
        else {
            flag = 2;
            p = p->getRChild();
        }
    }
    BSTreeNode* lchild = nullptr;
    BSTreeNode* rchild = nullptr;
    BSTreeNode* bs = new BSTreeNode(val, lchild, rchild);
    if(flag == 1){
        q->setLChild(bs);
    }
    else if(flag == 2){
```

```
        q->setRChild(bs);
    }
    return true;
}


//二叉排序树删除节点 bool deleteNode(int)
bool BSTree::deleteNode(int val) {
    BSTreeNode* p = root;
    BSTreeNode* q = nullptr;
    bool found = false;

    while(p != nullptr && !found) {
        if(val < p->getData()) {
            q = p;
            p = p->getLChild();
        }
        else if(val > p->getData()) {
            q = p;
            p = p->getRChild();
        }
        else
            found = true;
    }

    if(!found) return false;

    else {
        BSTreeNode* s = nullptr;

        if(p->getLChild() == nullptr) {
            if(q == nullptr) root = p->getRChild();
            else if(q->getLChild() == p) q->setLChild(p->getRChild());
            else q->setRChild(p->getRChild());
        }
        else if(p->getRChild() == nullptr) {
            if(q == nullptr) root = p->getLChild();
            else if(q->getLChild() == p) q->setLChild(p->getLChild());
            else q->setRChild(p->getLChild());
        }
        else {

            s = p->getRChild();
            while(s->getLChild() != nullptr) s = s->getLChild();
```

```
            if(q == nullptr) root = s;
            else if(q->getLChild() == p) q->setLChild(s);
            else q->setRChild(s);

            s->setLChild(p->getLChild());
        }
        delete p;
        num--;
        return true;
    }
}
```

3. 实验结果

顺序表二分查找

```
2
-1
```

二叉查找树

```
3 1 2 5 4
1
0
1
0
1
0
3 1 5 4 7
0
1
```

4. 实验中遇到的问题及解决方法

学习和实践了以下知识点

二叉排序树的特性。二叉排序树左子树的值均小于根节点,右子树的值均大于根节点。这一特性在添加节点 addNode()和删除节点 deleteNode()中体现。

二叉排序树的构建过程。通过 BSTree(int num, int* data)构造函数根据数组 data 构建二叉排序树,学习了二叉排序树的插入过程。

二叉排序树的查找过程。searchNode(int val)函数实现了对二叉排序树的查找,深入理解了二叉排序树的结构。

二叉排序树的节点删除。deleteNode(int val)函数实现了对二叉排序树的删除,

需要同时维护二叉排序树的特性,比较难以理解,通过本次实现可以加深认知。
二叉树的空指针判断。在遍历、查找、删除等函数中添加空指针判断,可以避免程序崩溃,使程序更加健壮。