二叉排序树

问题描述

依次输入关键字并建立二叉排序树，实现二叉排序数的插入和查找功能。

项目要求

二叉排序树就是指将原来已有的数据根据大小构成一棵二叉树，二叉树中的所有结点数据满足一定的大小关系，所有的左子树中的结点均比根结点小，所有的右子树的结点均比根结点大。

二叉排序树查找是指按照二叉排序树中结点的关系进行查找，查找关键自首先同根结点进行比较，如果相等则查找成功；如果比根节点小，则在左子树中查找；如果比根结点大，则在右子树中进行查找。这种查找方法可以快速缩小查找范围，大大减少查找关键的比较次数，从而提高查找的效率。

功能实现及代码分析

1、结构体和数据结构设计

二叉排序树的结点，包括存储的数据、左右子节点。

```
typedef struct Node {
    int data;
    struct Node *lchild, *rchild;
} Node, *Tree;
```

2、进入项目

进入项目，引导用户输入操作，使用功能，包括建立二叉排序树、插入元素、查询元素和退出程序。

```
int main() {
    int choice, key;
    Tree T = NULL;
    cout << "**              二叉排序树             **" << endl;
    cout << "====================================" << endl;
    cout << "**         1 --- 建立二叉排序树         **" << endl;
    cout << "**         2 --- 插入元素              **" << endl;
    cout << "**         3 --- 查询元素              **" << endl;
    cout << "**         4 --- 退出程序              **" << endl;
    cout << "====================================" << endl;

    do {
        cout << "\n\nPlease select:\t";
        cin >> choice;
        switch (choice) {
            case 1:
                T = create();
                cout << "Bsort_Tree is:\n" << endl;
                show(T);
                cout << endl;
                break;
            case 2:
                cout << "Please input key which inserted:\t";
                cin >> key;
                T = insert(T, key);
                cout << "\nBsort_Tree is:\n" << endl;
                show(T);
                cout << endl;
                break;
            case 3:
                cout << "Please input key which searched:" << endl;
                cin >> key;
                if (search(T, key)) {
                    cout << "Search success!" << endl;
                } else {
                    cout << key << "not exist!" << endl;
```

```
                }
                break;
            case 4:
                return 0;
            case 5:
                cout << "Please input key which removed:";
                cin >> key;
                remove(T, key);
            default:
                cout << "Wrong input!Please select again!" << endl;
                break;
        }
    } while (choice != 0);
    cout << "\n----------End--------" << "\n";
    return 0;
}
```

3、建立二叉排序树

获取用户数据，通过反复将数据插入二叉树，建立起二叉排序树。

```
Tree create()         //创建二叉排序树
{
    int key;
    Tree T = NULL;
    cout << "Please input key to create Bsort_Tree:" << endl;
    cin >> key;
    while (key != 0) {
        T = insert(T, key);
        cin >> key;
    }
    return T;
}
```

4、插入元素

插入元素之前，先遍历二叉排序树检查元素是否已经存在，若已存在输出相关信息并返回，若不存在则将新元素插入二叉树。

此时，若二叉排序树为空，元素作为根节点插入空树；非空时，将数据与树根数据进行比较，若相等则无需插入，然后按照较小往左较大往右的规则递归下降直至结点插入二叉排序树中或找到相等结点。

```
Tree insert(Tree T, int key) {  //插入节点
    Tree p = T;
    Tree f = NULL, s;
    while (p != NULL) {
        f = p;
        if (key == p->data) {
            cout << "The input key<" << key << ">is have in! " << endl;
            return T;
        }
        p = (key > p->data) ? p->rchild : p->lchild;
    }
    s = new Node;
    s->data = key;
    s->lchild = NULL;
    s->rchild = NULL;
    if (T == NULL) return s;
    if (key < f->data) {
        f->lchild = s;
    } else {
        f->rchild = s;
    }
    return T;
}
```

5、查询元素

从根节点开始，沿某一分支逐层向下进行比较判断，若查询值更小，继续递归查询左子树结点，若更大，则递归查询右子树结点。

```
bool search(Tree T, int key) {  //搜索指定节点
    Tree p = T;
```

```
    while (p) {
        if (key == p->data) {
            return true;
        } else {
            p = (key < p->data) ? (p->lchild) : (p->rchild);
        }
    }
    return false;
}
```

6、删除元素

删除元素时要判断当前结点的子节点情况。

```
void remove(Tree T, int key)      //删除指定的节点
{
    Tree p = T, q, f = NULL, s;
    while (p) {                   //先寻找到要删除的节点
        if (key == p->data) {
            break;
        }
        f = p;
        p = (key < p->data) ? p->lchild : p->rchild;
    }
    if (!p) {
        cout << "The key which removed is not exist!" << "\n";
    } else if (!p->lchild && !p->rchild) { //若当前节点没有孩子节点
        if (p == T) T = NULL;
        if (p == f->lchild) {
            f->lchild = NULL;
        } else {
            f->rchild = NULL;
        }
        delete p;
        p = NULL;
    } else if (!p->lchild && p->rchild) { //若当前节点只有右子节点
        if (f->lchild == p) {
            f->lchild = p->rchild;
        } else {
            f->rchild = p->rchild;
        }
        delete p;
        p = NULL;
    } else if (!p->rchild && p->lchild) { //若当前结点只有左子节点
        if (f->lchild == p) {
            f->lchild = p->lchild;
        } else {
            f->rchild = p->lchild;
        }
        delete p;
        p = NULL;
    } else {
        q = p;
        s = p->lchild;
        while (s->rchild) {
            q = s;
            s = s->rchild;
        }
        p->data = s->data;
        if (q != p) {
            q->rchild = s->lchild;
        } else {
            q->lchild = s->lchild;
        }
        delete s;
        s = NULL;
    }
}
```

7、输出二叉树

此时二叉树中数据已按顺序排列，可以直接输出。若二叉树为空，则无需操作，若不为空，按照左子树、根节点、右子树，即中序遍历的方式递归遍历输出二叉树中结点数据。

```
void show(Tree T)     //中序遍历并显示
{
    if (T) {
        show(T->lchild);
        cout << T->data << "->";
        show(T->rchild);
    }
    return;
}
```

用例演示

```
**             二叉排序树             **
======================================
**        1 --- 建立二叉排序树         **
**        2 --- 插入元素               **
**        3 --- 查询元素               **
**        4 --- 退出程序               **
======================================

Please select:  1
Please input key to create Bsort_Tree:
12 34 67 48 19 44 21 30 19 7 4 24 9 88 100 100 0
The input key<19>is have in!
The input key<100>is have in!
Bsort_Tree is:
4->7->9->12->19->21->24->30->34->44->48->67->88->100->

Please select:  2
Please input key which inserted:    90
Bsort_Tree is:
4->7->9->12->19->21->24->30->34->44->48->67->88->90->100->

Please select:  3
Please input key which searched:    90
Search success!

Please select:  110
Wrong input!Please select again!

Please select:  4

Process finished with exit code 0
```