项目说明文档

数据结构课程设计

——二叉排序树

作 者 姓 名： 曹晓慈

学 号： 2052844

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

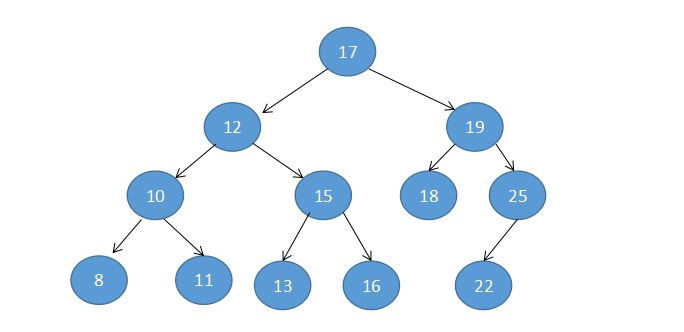
# 分析

## 功能分析

首先[二叉排序树](https://so.csdn.net/so/search?from=pc_blog_highlight&q=%E4%BA%8C%E5%8F%89%E6%8E%92%E5%BA%8F%E6%A0%91)也是一棵[二叉树](https://so.csdn.net/so/search?from=pc_blog_highlight&q=%E4%BA%8C%E5%8F%89%E6%A0%91)，所谓二叉树，就是“任何节点最多只允许两个子节点”，这两个子节点称为左右子节点。如下便是一个二叉树。

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。



# 设计

## 数据结构设计

根据对于功能分析，该系统就是要在二叉树的基础之上建立一颗二叉排序树。这里用结构体BitNode来表示树上的节点，再通过一些相应的对于树结点的操作来进行功能的实现。

## 成员与操作设计

typedef struct BiTNode

{

int data;

BiTNode\* lchild, \* rchild; //左右孩子指针

}Node, \* BST;

bool BSTInsert(Node\*& p, int element)； //数据的插入

void CreatBST(Node \*&T,vector<int> storage,int n)；//创建二叉排序树

void inOrderTraverse(BST T)； //中序遍历二叉排序树

bool BSTSearch(BST T, int element)； //在数中查找某一元素

# 实现

## 二叉排序树的建立

建立线索二叉树

结点为空

创建节点

左右孩子为NULL，输入数据

调用BSTInsert函数，比结点小进入左，比结点大进入有孩子，相同不操作

bool BSTInsert(Node\*& p, int element)

{

if (NULL == p) // 空树

{

p = new Node;

p->data = element;

p->lchild = p->rchild = NULL;

return true;

}

if (element == p->data) // BST中不能有相等的值

return false;

if (element < p->data) // 递归

return BSTInsert(p->lchild, element);

return BSTInsert(p->rchild, element); // 递归

}

void CreatBST(Node \*&T,vector<int> storage,int n)

{

T = NULL;

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

{

bool a=BSTInsert(T, storage[i]);

if (a == 0)

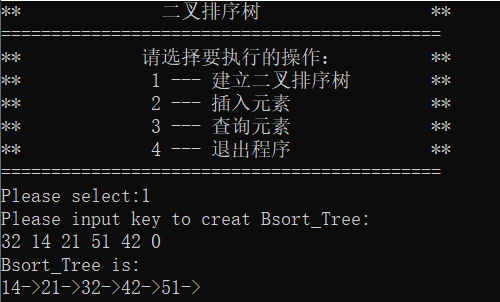
{

cout << "The input key<" << storage[i] << ">is have in!" << endl;

}

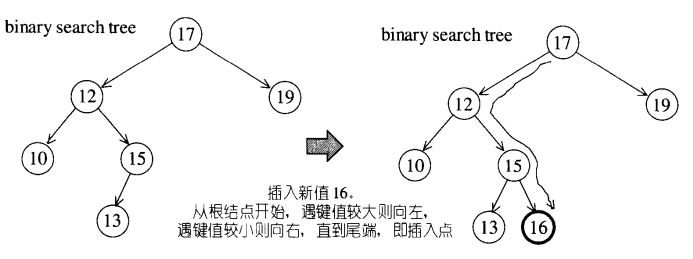
}

}



### 二叉排序树的元素插入

首先判断树是否为空，若为空则将待插入结点作为新的树的根节点，否则对树进行递归，若比根节点小则将左孩子作为根节点继续递归，比节点大则将右孩子作为根节点进行递归，直到找到一个空结点并在这里进行插入，若在递归过程中遇到与插入元素相同的值，则直接返回。



该元素作为二叉排序树根节点

根节点不存在

左结点递归直到找到空结点插入元素

右结点递归直到找到空结点插入元素

返回

大于

等于

小于

待插入元素与根节点

bool BSTInsert(Node\*& p, int element)

{

if (NULL == p) // 空树

{

p = new Node;

p->data = element;

p->lchild = p->rchild = NULL;

return true;

}

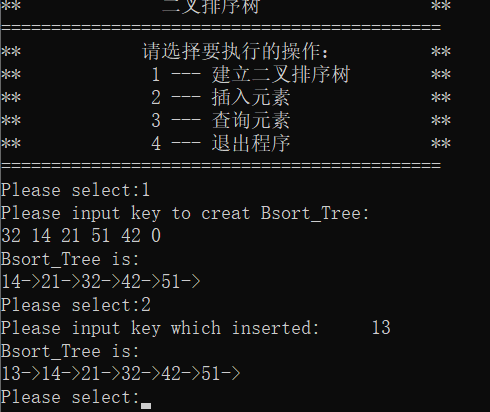
if (element == p->data) // BST中不能有相等的值

return false;

if (element < p->data) // 递归

return BSTInsert(p->lchild, element);

return BSTInsert(p->rchild, element); // 递归

}

## 二叉排序树的查找

针对二叉排序树的查找同样可以使用递归的操作，当等于当前结点时返回true,小于当前元素则去往左子树递归查找，大于当前子树则去右子树递归查找。

bool BSTSearch(BST T, int element)

{

bool a=0, b=0;

if (T == NULL)

{

return false;

}

if (T->data == element)

return true;

else if (element > T->data)

return BSTSearch(T->rchild, element);

else

return BSTSearch(T->lchild, element);

}

## 主要系统实现

主要系统的实现则是用了while循环和switch语句对choice的输入进行判断，接受到对应信息后进入对应的case进行操作。

cin >> choice;

while (choice!=4)

{

switch (choice)

{

case 1:

{

（建立二叉排序树）

}

case 2:

{

（调用Insert函数进行对二叉排序树的插入）

}

case 3:

{

（调用Search()函数查找元素）

}

default:

{

cout << "请输入正确的命令";

break;

}

}

cout << "Please select:";

cin >> choice;

}

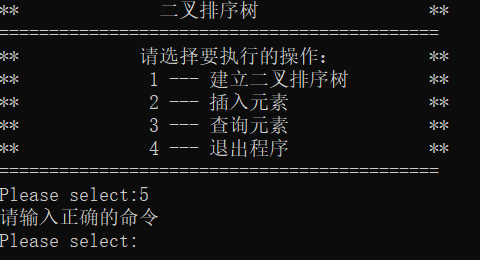
## 测试

### 1. 选择序号不存在

**测试用例：**输入5

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**

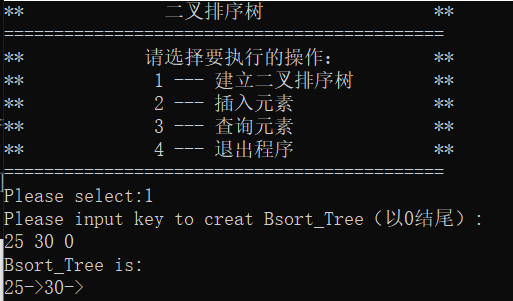
****

### 2.建立二叉树功能

**测试用例：**1 25 30

**预期结果：**程序正常运行给出正确答案。

**实验结果：**



## 多次创建二叉排序树

**测试用例：**1 25 30 0 1 21 35 0

**预期结果：**程序正常运行，后面的二叉排序树覆盖前面的二叉排序树。

**实验结果：**



## 查找功能

**测试用例：**1 21 35 0 3

**预期结果：**程序正常运行，正确和错误均有对应的提示。

**实验结果：**



## 插入功能

**测试用例：**1 21 35 0 2 25 2 21

**预期结果：**程序正常运行，正确和错误均有对应的提示。

**实验结果：**

