

本节内容

二叉排序树
(BST)

王道考研/CSKAOYAN.COM

1

知识总览

二叉排序树

二叉排序树的定义

查找操作

插入操作

删除操作

查找效率分析

王道考研/CSKAOYAN.COM

2

二叉排序树的定义

二叉排序树可用于元素的有序组织、搜索

二叉排序树, 又称二叉查找树 (BST, Binary Search Tree)
一棵二叉树或者是空二叉树, 或者是具有如下性质的二叉树:
左子树上所有结点的关键字均小于根结点的关键字;
右子树上所有结点的关键字均大于根结点的关键字。
左子树和右子树又各是一棵二叉排序树。

左子树结点值 < 根结点值 < 右子树结点值

进行中序遍历, 可以得到一个递增的有序序列

左子树

右子树

王道考研/CSKAOYAN.COM

3

二叉排序树的查找

左子树结点值 < 根结点值 < 右子树结点值

若树非空, 目标值与根结点的值比较:
若相等, 则查找成功;
若小于根结点, 则在左子树上查找, 否则在右子树上查找。

查找成功, 返回结点指针; 查找失败返回NULL

T

例1: 查找关键字为30的结点

```
//在二叉排序树中查找值为 key 的结点
BSTNode *BST_Search(BSTree T, int key){
    while(T!=NULL && key!=T->key){
        if(key<T->key) T=T->lchild;
        else T=T->rchild;
    }
    return T;
}
```

王道考研/CSKAOYAN.COM

4

二叉排序树的查找

左子树结点值 < 根结点值 < 右子树结点值

若树非空, 目标值与根结点的值比较:
若相等, 则查找成功;
若小于根结点, 则在左子树上查找, 否则在右子树上查找。

查找成功, 返回结点指针; 查找失败返回NULL

例1: 查找关键字为30的结点

```
//二叉排序树结点
typedef struct BSTNode{
    int key;
    struct BSTNode *lchild,*rchild;
}BSTNode,*BSTree;

//在二叉排序树中查找值为 key 的结点
BSTNode *BST_Search(BSTree T,int key){
    while(T!=NULL&&key!=T->key){ //若树空或等于根结点值, 则结束循环
        if(key<T->key) T=T->lchild; //小于, 则在左子树上查找
        else T=T->rchild; //大于, 则在右子树上查找
    }
    return T;
}
```

王道考研/CSKAOYAN.COM

5

二叉排序树的查找

左子树结点值 < 根结点值 < 右子树结点值

若树非空, 目标值与根结点的值比较:
若相等, 则查找成功;
若小于根结点, 则在左子树上查找, 否则在右子树上查找。

查找成功, 返回结点指针; 查找失败返回NULL

例2: 查找关键字为12的结点

```
//二叉排序树结点
typedef struct BSTNode{
    int key;
    struct BSTNode *lchild,*rchild;
}BSTNode,*BSTree;

//在二叉排序树中查找值为 key 的结点
BSTNode *BST_Search(BSTree T,int key){
    while(T!=NULL&&key!=T->key){ //若树空或等于根结点值, 则结束循环
        if(key<T->key) T=T->lchild; //小于, 则在左子树上查找
        else T=T->rchild; //大于, 则在右子树上查找
    }
    return T;
}
```

王道考研/CSKAOYAN.COM

6

二叉排序树的查找

```
//在二叉排序树中查找值为 key 的结点
BSTNode *BST_Search(BSTree T,int key){
    while(T!=NULL&&key!=T->key){    //若树空或等于根结点值, 则结束循环
        if(key<T->key) T=T->lchild;    //小于, 则在左子树上查找
        else T=T->rchild;    //大于, 则在右子树上查找
    }
    return T;
}

//在二叉排序树中查找值为 key 的结点 (递归实现)
BSTNode *BSTSearch(BSTree T,int key){
    if (T==NULL)
        return NULL;    //查找失败
    if (key==T->key)
        return T;    //查找成功
    else if (key < T->key)
        return BSTSearch(T->lchild, key);    //在左子树中找
    else
        return BSTSearch(T->rchild, key);    //在右子树中找
}
```

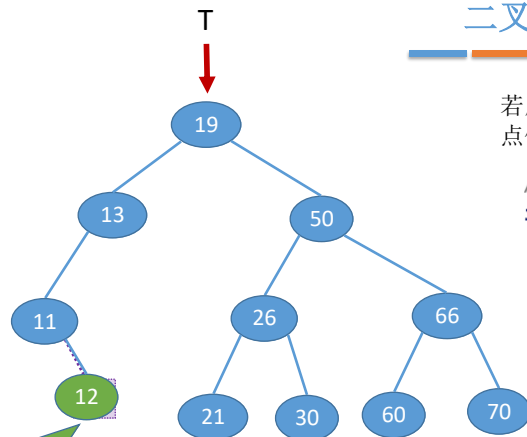
最坏空间复杂度 $O(1)$

最坏空间复杂度 $O(h)$

王道考研/CSKAOYAN.COM

7

二叉排序树的插入



例: 插入关键字为12的结点



嗨嗨, 醒醒,
敲代码了!

练习: 实现非
递归插入

若原二叉排序树为空, 则直接插入结点; 否则, 若关键字 k 小于根结点值, 则插入到左子树, 若关键字 k 大于根结点值, 则插入到右子树

//在二叉排序树插入关键字为 k 的新结点 (递归实现) 最坏空间复杂度 $O(h)$

```
int BST_Insert(BSTree T, int k){
    if(T==NULL){    //原树为空, 新插入的结点为根结点
        T=(BSTree)malloc(sizeof(BSTNode));
        T->key=k;
        T->lchild=T->rchild=NULL;
        return 1;    //返回1, 插入成功
    }
    else if(k==T->key)    //树中存在相同关键字的结点, 插入失败
        return 0;
    else if(k<T->key)    //插入到T的左子树
        return BST_Insert(T->lchild,k);
    else    //插入到T的右子树
        return BST_Insert(T->rchild,k);
}
```

王道考研/CSKAOYAN.COM

8

二叉排序树的构造

```
//按照 str[] 中的关键字序列建立二叉排序树
void Creat_BST(BSTree &T,int str[],int n){
    T=NULL;           //初始时T为空树
    int i=0;
    while(i<n){        //依次将每个关键字插入到二叉排序树中
        BST_Insert(T,str[i]);
        i++;
    }
}
```

例1: 按照序列str={50, 66, 60, 26, 21, 30, 70, 68}建立BST

例2: 按照序列str={50, 26, 21, 30, 66, 60, 70, 68}建立BST

不同的关键字序列可能得到同款二叉排序树

王道考研/CSKAOYAN.COM

9

二叉排序树的构造

例1: 按照序列str={50, 66, 60, 26, 21, 30, 70, 68}建立BST

例2: 按照序列str={50, 26, 21, 30, 66, 60, 70, 68}建立BST

例3: 按照序列str={26, 21, 30, 50, 60, 66, 68, 70}建立BST

不同的关键字序列可能得到同款二叉排序树

也可能得到不同款二叉排序树

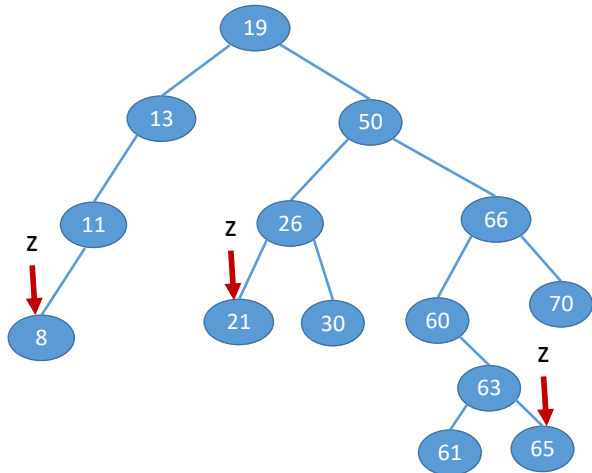
王道考研/CSKAOYAN.COM

10

二叉排序树的删除

先搜索找到目标结点:

① 若被删除结点 z 是叶结点, 则直接删除, 不会破坏二叉排序树的性质。



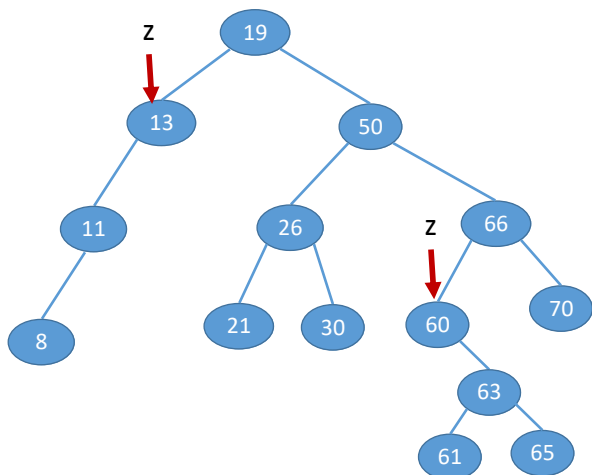
左子树结点值 < 根结点值 < 右子树结点值

王道考研/CSKAOYAN.COM

11

二叉排序树的删除

② 若结点 z 只有一棵左子树或右子树, 则让 z 的子树成为 z 父结点的子树, 替代 z 的位置。



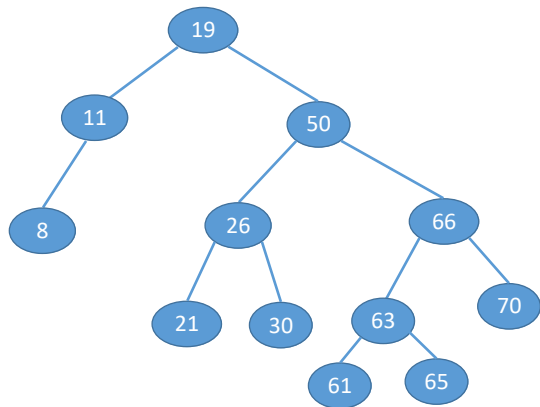
左子树结点值 < 根结点值 < 右子树结点值

王道考研/CSKAOYAN.COM

12

二叉排序树的删除

② 若结点 z 只有一棵左子树或右子树, 则让 z 的子树成为 z 父结点的子树, 替代 z 的位置。



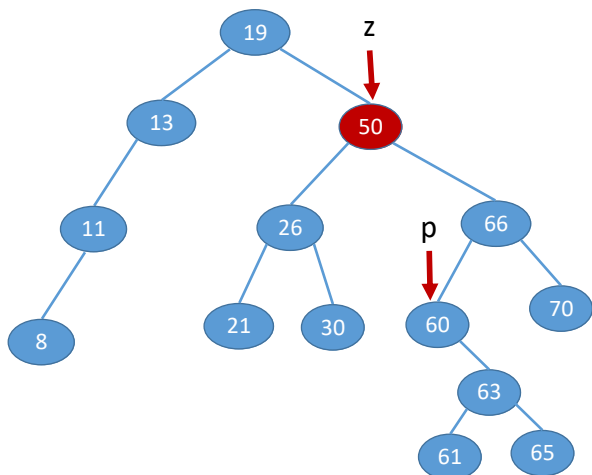
左子树结点值 < 根结点值 < 右子树结点值

王道考研/CSKAOYAN.COM

13

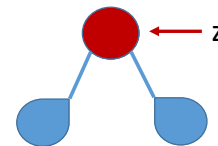
二叉排序树的删除

③ 若结点 z 有左、右两棵子树, 则令 z 的直接后继(或直接前驱)替代 z , 然后从二叉排序树中删去这个直接后继(或直接前驱), 这样就转换成了第一或第二种情况。



左子树结点值 < 根结点值 < 右子树结点值

进行中序遍历, 可以得到一个递增的有序序列



中序遍历——左 根 右

左 根 (左 根 右)

左 根 ((左 根 右) 根 右)

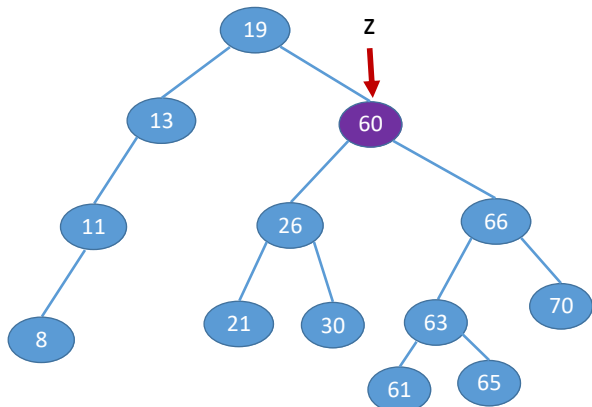
z 的后继: z 的右子树中最左下结点(该结点一定没有左子树)

王道考研/CSKAOYAN.COM

14

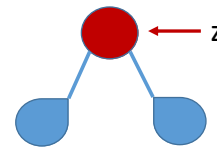
二叉排序树的删除

③ 若结点 z 有左、右两棵子树, 则令 z 的直接后继(或直接前驱)替代 z , 然后从二叉排序树中删去这个直接后继(或直接前驱), 这样就转换成了第一或第二种情况。



左子树结点值 < 根结点值 < 右子树结点值

进行中序遍历, 可以得到一个递增的有序序列



中序遍历——左 根 右

左 根 (左 根 右)

左 根 ((左 根 右) 根 右)

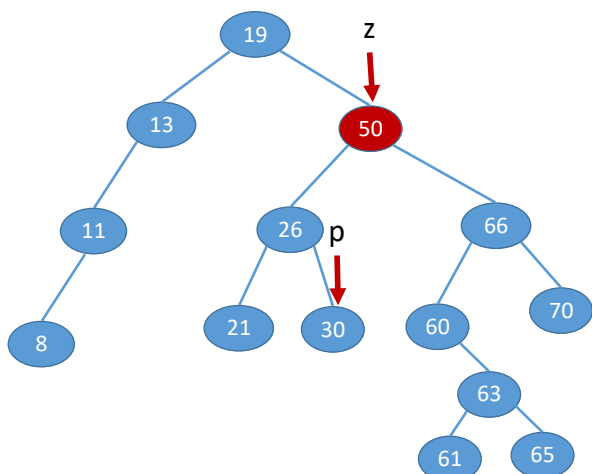
z 的后继: z 的右子树中最左下结点(该结点一定没有左子树)

王道考研/CSKAOYAN.COM

15

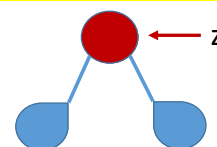
二叉排序树的删除

③ 若结点 z 有左、右两棵子树, 则令 z 的直接后继(或直接前驱)替代 z , 然后从二叉排序树中删去这个直接后继(或直接前驱), 这样就转换成了第一或第二种情况。



左子树结点值 < 根结点值 < 右子树结点值

进行中序遍历, 可以得到一个递增的有序序列



中序遍历——左 根 右

(左 根 右) 根 右

(左 根 (左 根 右)) 根 右

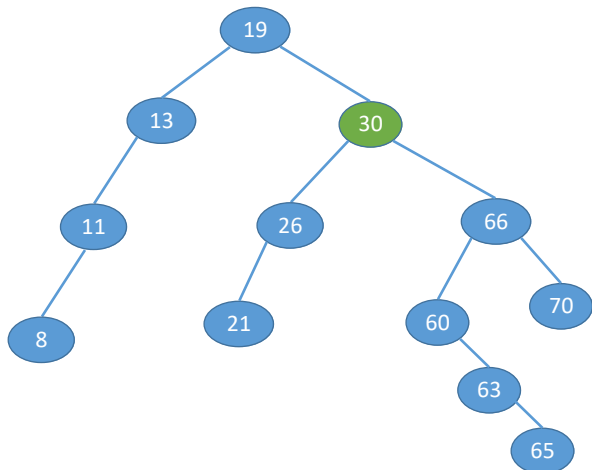
z 的前驱: z 的左子树中最右下结点(该结点一定没有右子树)

王道考研/CSKAOYAN.COM

16

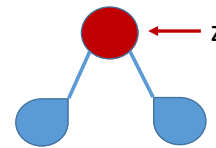
二叉排序树的删除

③ 若结点 z 有左、右两棵子树, 则令 z 的直接后继(或直接前驱)替代 z , 然后从二叉排序树中删去这个直接后继(或直接前驱), 这样就转换成了第一或第二种情况。



左子树结点值 < 根结点值 < 右子树结点值

进行中序遍历, 可以得到一个递增的有序序列



中序遍历——左 根 右

(左 根 右) 根 右

(左 根 (左 根 右)) 根 右

z 的前驱: z 的左子树中最右下结点(该结点一定没有右子树)

王道考研/CSKAOYAN.COM

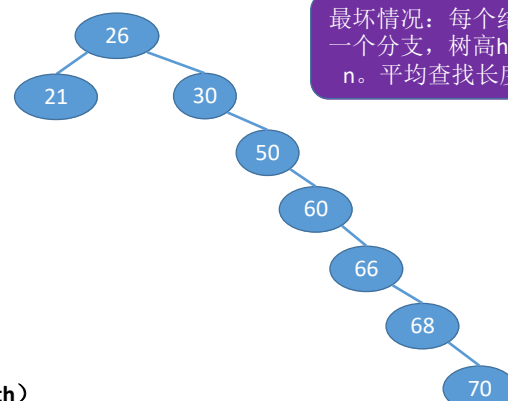
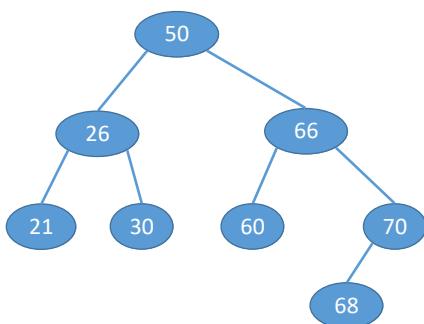
17

查找效率分析

若树高 h , 找到最下层的一个结点需要对比 h 次

最好情况: n 个结点的二叉树最小高度为 $\lfloor \log_2 n \rfloor + 1$ 。
平均查找长度 = $O(\log_2 n)$

查找长度——在查找运算中, 需要对比关键字的次数称为查找长度, 反映了查找操作时间复杂度



最坏情况: 每个结点只有一个分支, 树高 h =结点数 n 。平均查找长度= $O(n)$

查找成功的平均查找长度 ASL (Average Search Length)

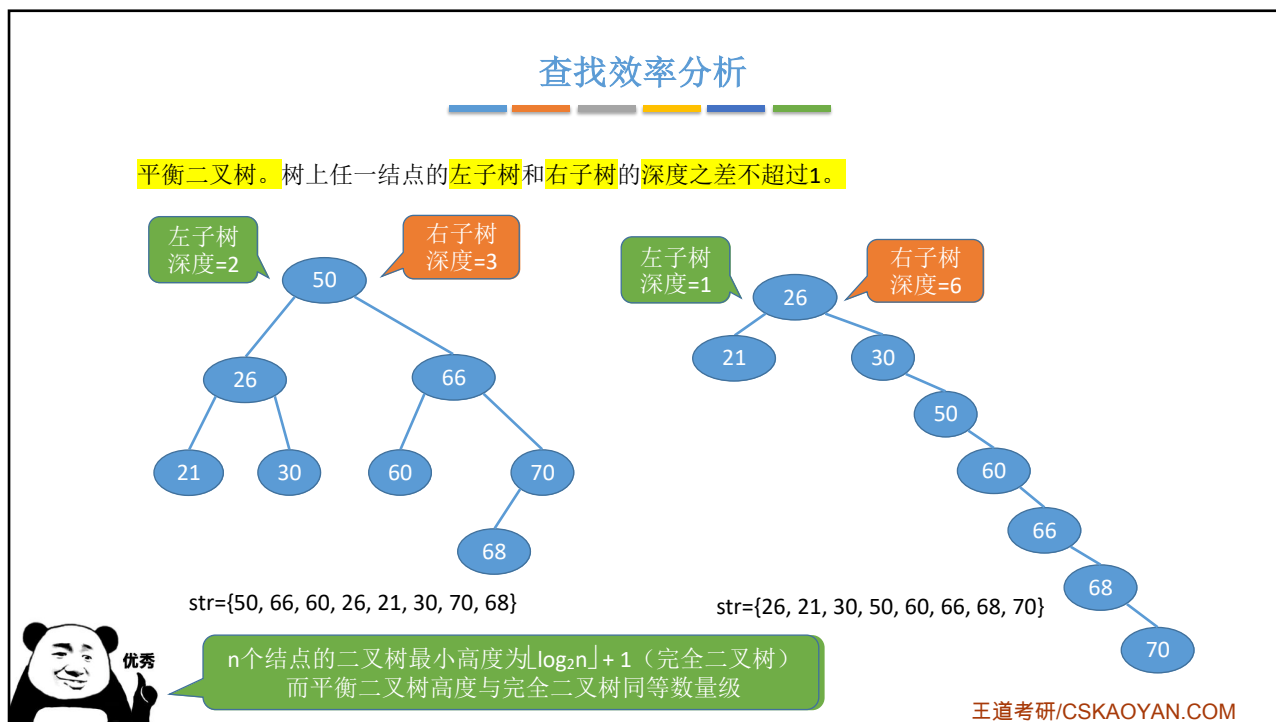
$$ASL = (1*1 + 2*2 + 3*4 + 4*1)/8 = 2.625$$

$$ASL = (1*1 + 2*2 + 3*1 + 4*1 + 5*1 + 6*1 + 7*1)/8 = 3.75$$

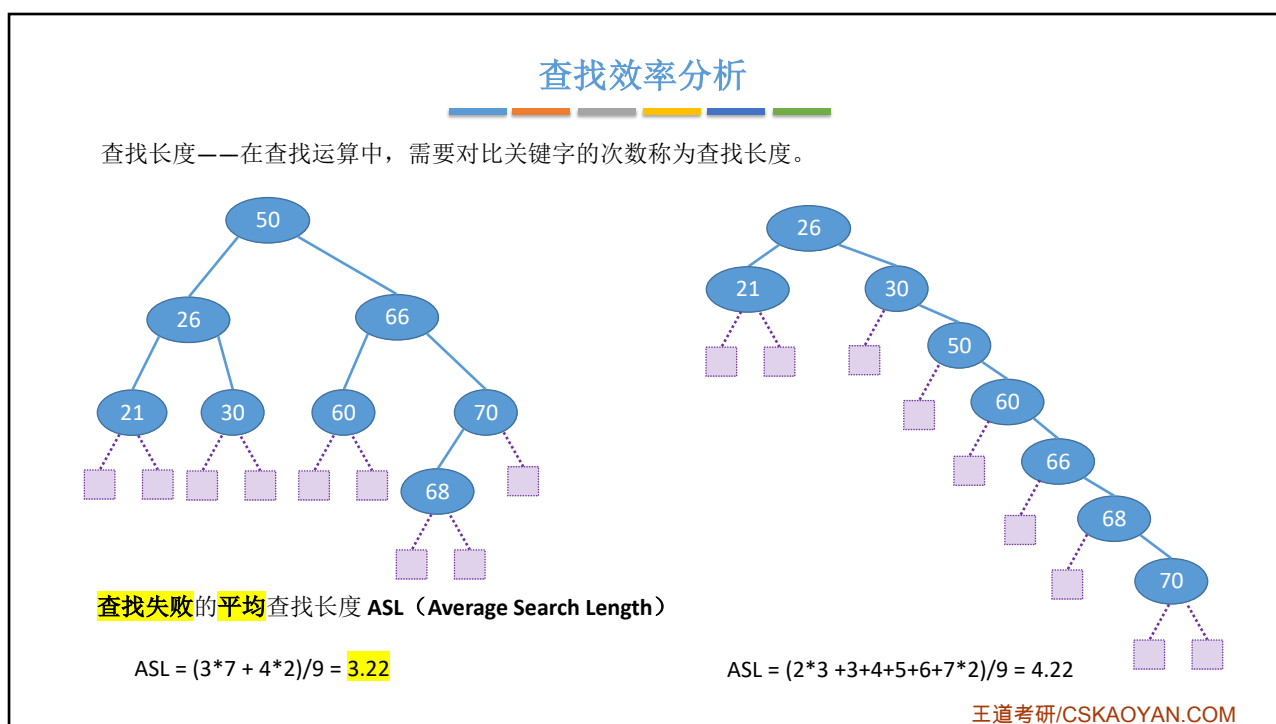


王道考研/CSKAOYAN.COM

18



19



20

