二叉搜索树

Part 1:二叉搜索树

主要特点：节点X的左子树中的最大关键字不超过x.key，右子树的最小关键字不小于x.key

Part 2:二叉搜索树的遍历

中序遍历：以递归的方式按序输出二叉搜索树的所有关键字（其实有迭代版本）

特点：输出的子树根的关键字位于左右子树的关键字值之间

```
inorder_tree_walk(x)
if x!=NIL
   inorder_tree_walk(x.left)
   print(x.key)
   inorder_tree_walk(x.right)
```

c++版本：

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {//向量容器用于储存已排好序的key值

        vector<int> res;
        inorder(res, root);
        return res;
    }

    void inorder(vector<int>& res , TreeNode* root){//中序遍历函数
        if(!root) return;
        inorder(res, root->left);//对左子树遍历
        res.emplace_back(root->val);//将节点值压入向量，使用emplace_back可少调用一次有参构造函数
        inorder(res, root->right);
    }
};
```

注：叫中序遍历是由二叉树的构造特点决定的

类似：先序遍历（输出根的关键字位于左右子树之前）和后序遍历（输出根的关键字位于左右子树之后）

先序遍历伪代码（递归版）

```
preorder_tree_walk(x)
if x!=NIL
   print(x.key)
   preorder_tree_walk(x.left)
   preorder_tree_walk(x.right)
```

12.1-4答案之一，后序遍历可自行尝试

复杂度分析：若x是一棵有n个结点子树的根，调用inorder-tree-walk(x)的时间复杂度是Θ(n)

证明：

对于渐近下界，算法需要遍历树的每一个的节点，所以由T(n)=Ω(n)

对于上界，使用代换法证明T(n)=O(n)

即存在常数c，当n充分大时

$$T(n) \leq cn$$

对于一颗空树，需要常数阶的时间遍历，设该常数为t

设根节点的左子树有k个元素，右子树有n-k-1个元素

给出递归式

$$\begin{aligned} T(n) & \leq T(k)+T(n-k-1)+d \\ & \leq ck+c(n-k-1)+d \\ & = c(n-1)+d \\ & = cn-(c+d) \end{aligned}$$

当c+d<=0时，有T(n)=O(n)

Part 3:查询二叉搜索树

查找：输入一个指向树根的指针和一个关键字k，若节点存在，则返回指向k的指针，若不存在则返回NIL

```
TREE-SEARCH(x,k)
if x==NIL or k==x.key
   return x
if k<x.key
   return TREE-SEARCH(x.left,k)
else return TREE-SEARCH(x.right,k)
```

复杂度分析：根据二叉搜索树性质，从树根开始的递归会形成一条向下的简单路径，最多不会超过lgn，故该算法运行时间为O(h)，其中h为树的高度

最大和最小关键字元素查找：

```
TREE-MINIMUM
while x.left!=NIL
      x=x.left
return x
```

最大查找的伪代码是对称的

二叉搜索树的性质保证了代码的正确性，且均能在O(h)的时间内执行完

递归版本

```
TREE-MAXIMUM
if x.left!=NIL
   return TREE-MAXIMUM(x.left)
else
   return x
```

后继与前驱:

```
TREE-SUCCESSOR(X)
if x.right!=NIL
   return TREE-MINIMUM(x.right)
y=x.p
while y!=NIL and x==y.right//向上回溯直到一个自己是父节点的左孩子
      x=y
      y=y.p
return y
```

```
TREE-PRESUCCESSOR(X)
if x.left!=NIL
   return TREE-MAXIMUM(x.left)
y=x.p
while y!=NIL and x==y.left//向上回溯直到自己是父节点的右孩子
      x=y
      y=y.p
return y
```

练习12.2-3

练习12.2-8

练习12.2-7

Part 4:插入和删除

插入

将一个新值v插入到一棵二叉搜索树中，设结点z，z.key=v,z.left=NIL,z.right=NIL

特点：自上而下

```
TREE-INSERT(T,z)
y=NIL
x=T.root
while x!=NIL//x指针用于遍历寻找位置
      y=x//y指针用于记录本次循环的起始位置
      if z.key<x.key
         x=x.left
      else
         x=x.rigt
z.p=y
if y==NIL
```

```
   T.root=z//若T是空树，则z为根结点
elseif z.key<y.key
       y.left=z
else y.right=z
```

复杂度分析：复杂度取决于while循环中x的遍历路线，容易看出是一条向下的简单路径，故时间复杂度为O(h)

递归版本（练习12.3-1）

```
TREE-INSERT(T,z)
x=T.root
y=x
if x.key>z.key
   INSERT(T.left,z)
else INSERT(T.right,z)
z.p=y
if y==NIL
   T.root=z
elseif z.key<y.key
       z=y.left
else z=y.right
```

删除

若要从T中删除一个节点z，分为以下三种情况

1. z为叶结点，直接删除
2. z有一个孩子结点，将孩子结点的父结点修改为z的父节点，删除z
3. z有两个孩子，找到z的后继结点，修改后继结点的左右孩子，原后继节点（必然只有右孩子）的右孩子取代原后继结点的位置

定义子过程TRANSPLANT(T,u,v)，用于将u结点替换为v结点

```
TRANSPLANT(T,u,v)
if u.p==NIL
   T.root=v
elseif u==u.p.left
       u.p.left=v
else u.p.right=v
if v!=NIL
   v.p=u.p
```

利用TRANSPLANT过程删除结点z

```
TREE-DELETE(T,z)
if z.left==NIL
   TRANSPLANT(T,z,z.right)
elseif z.right==NIL
   TRANSPLANT(T,z,z.left)
else y=TREE-MINIMUM(z.right)//让y作为z的后继
     if y.p!=z
        TRANSPLANT(T,y,y.right)
        y.right=z.right
        y.right.p=y
     TRANSPLANT(T,z,y)
     y.left=z.left
     y.left.p=y
```

其他删除策略

1. 以待删除结点z的前驱作为替代

   与寻找后继的策略镜像，找到z的前驱结点，修改前驱结点的左右孩子，原前驱结点（必然只有左孩子）的左孩子取代原前驱结点的位置

   ```
   TREE-DELETE(T,Z)
   if z.left==NIL
      TRANSPLANT(T,z,z.right)
   elseif z.right==NIL
      TRANSPLANT(T,z,z.left)
   else y=TREE-MAXIMUM(z.left)//让y作为z的后继
        if y.p!=z
           TRANSPLANT(T,y,y.right)
           y.right=z.right
           y.right.p=y
        TRANSPLANT(T,z,y)
        y.left=z.left
        y.left.p=y
   ```

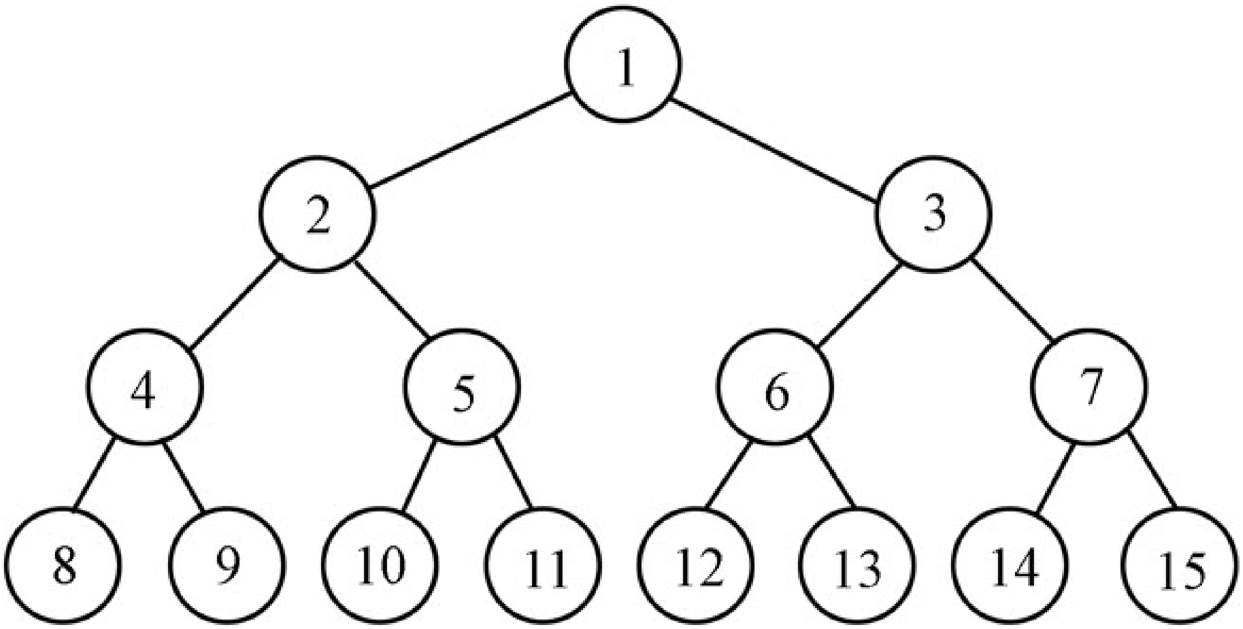
2. 公平删除策略

   赋予前驱和后继相同的优先级，每次删除随机选择后继或者前驱

Part 5:随机构建二叉搜索树

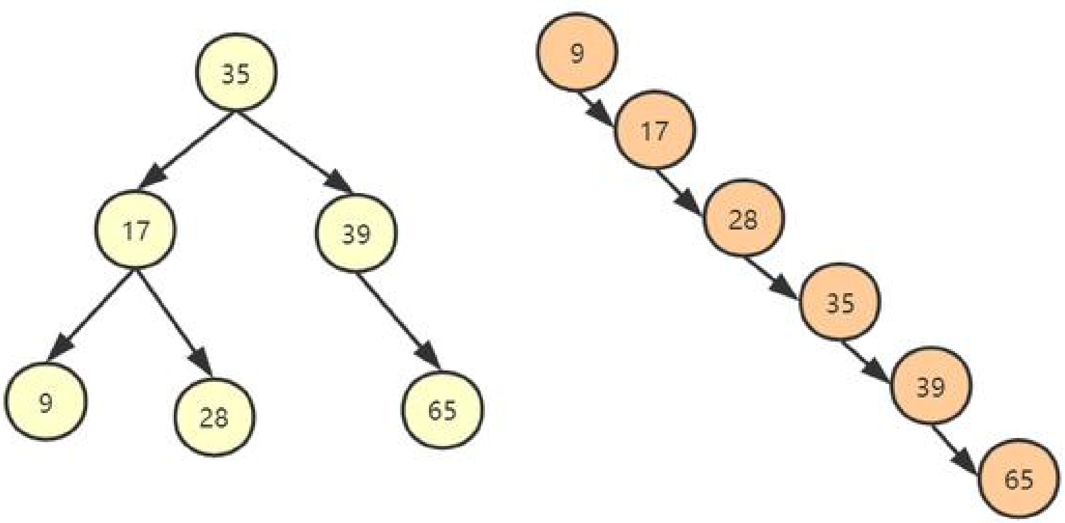
如果由一个数组来构建二叉树

最好情况：二叉树是一棵满二叉树



时间复杂度为Ω(nlgn)

最坏情况：数组有序排列有序插入



时间复杂度为Θ(n^2)

我们希望一个由n个元素组成的二叉树是随机插入元素的，其中n个元素的n!种排列是等可能出现的

引理1:Jensen不等式

对于一个凹函数f(x),在其定义域上，有

$$f(E[X_i]) \leq E[f(X_i)]$$

证明：

由凹函数定义，若f(x)定义域上存在任意两点x1,x2。0<=t<=1

有

$$tf(x) + (1-t)f(x) \geq f(tx_1 + (1-t)x_2)$$

考虑点集[λ_i]，满足

$$\lambda_i > 0 \qquad \sum_{i=1}^{n} \lambda_i = 1$$

现用数学归纳法证明

$$\sum_{i=1}^{n} \lambda_i f(x_i) \geq f(\sum_{i=1}^{n} \lambda_i x_i)$$

对于n=1以及n=2的情况，由凹函数的定义得证

当n>2时，运用数学归纳法

假设上式对于n>=2成立，只需证明对于n+1也成立

$$\begin{aligned} f(\sum_{i=1}^{n+1} \lambda_i x_i) &= f(\sum_{i=1}^{n} \lambda_i x_i + \lambda_{n+1} x_{n+1}) \\ &= f((1-\lambda_{n+1}) \sum_{i=1}^{n} \frac{1}{1-\lambda_{n+1}} \lambda_i x_i + \lambda_{i+1} x_{i+1}) \\ &\leq \end{aligned}$$

引理2：取幂函数

设一个随机构建二叉树的期望高度为X(n)，定义结点高度的取幂函数Y(n)为2^X(n)

对任意结点，其高度的取幂函数为左右孩子结点中较大者高度取幂函数的2倍，假设左子树有k个元素，可给出递推式

$$Y(n) = 2max(Y(k), Y(n-k-1))$$

定义指示器变量Z_nk，其取值为

$$Z_{nk} = \begin{cases} 1 & n = k \\ 0 & n \neq k \end{cases}$$

考虑到对于i∈{1，2，......，n}，二叉树的结点是等可能取值的，且每个结点取值相互独立

则有Pr{i=k}=Pr{Z_nk}，两边取期望有

$$E[Z_{nk}] = \frac{1}{n}$$

结合每个结点的高度取幂函数，可以给出一个基本的递推式

$$Y(n) = \sum_{i=1}^{n} Z_{ni} 2max(Y(i-1), Y(n-i-1))$$

两边同时取期望，有

$$
\begin{aligned}
E(Y(n)) \quad &= 2\sum_{i=1}^{n} E[Z_{ni}]E[max(Y(i-1), Y(n-i-1))] \\
&= \frac{2}{n}\sum_{i=1}^{n} E[max(Y(i-1), Y(n-i-1))]
\end{aligned}
$$

显然，

$$
\begin{aligned}
E[max(Y(i-1), Y(n-i-1))] \quad &\le E[Y(i-1) + Y(n-i-1)] \\
&\le E[Y(i-1)] + E[Y(n-i-1)]
\end{aligned}
$$

$$Then \qquad E(Y(n)) = \frac{2}{n}\sum_{i=1}^{n}(E[Y(i-1)] + E[Y(n-i-1)])$$

对于任意i∈{0，1，2，……，n-1}，在求和式中都会出现两次，一次作为E[Y(i-1)]，一次作为E[Y(n-i-1)]，有

$$E[Y(n)] = \frac{4}{n}\sum_{i=0}^{n-1} E[Y(i)]$$

使用替换法，证明

$$
\begin{aligned}
E(Y(n)) \quad &= O(n^3) \\
exist \quad c, E(Y(n)) \quad &\le cn^3 \\
\frac{4}{n}\sum_{i=0}^{n-1} E[Y(i)] \quad &\le \frac{4}{n}\sum_{i=0}^{n-1} cn^3 \\
&\le \frac{4}{n}\frac{cn^4}{4} \\
&\le cn^3
\end{aligned}
$$

由Jensen不等式，

$$2^{E[X(n)]} \le E[2^{X(n)}] = E[Y(n)] = O(n^3)$$

两边取对

$$E[X(n)] = O(lgn)$$