二叉查找树（Binary Search Tree），（又：二叉搜索树，二叉排序树）它或者是一棵空树，或者是具有下列性质的二叉树： 若它的左子树不空，则左子树上所有结点的值均小于它的根结点的值； 若它的右子树不空，则右子树上所有结点的值均大于它的根结点的值； 它的左、右子树也分别为二叉排序树。（百度百科）

ps:其实就在二叉树上面加多了一些限制条件

**树节点的结构**

**节点类API设计：**

| **类名** | **Node<key,value>** |
| --- | --- |
| 构造方法 | Node(Key key,Value value,Node left,Node right) |
| 成员变量 | 1.public Key key :存储键 2.public Value value:存储值 3.public Node right:右子节点 4.public Node left:左子节点 |

代码实现：

private class Node {

public Key key;

private Value value;

public Node left;

public Node right;

public Node(Key key, Value value, Node left, Node right) {

this.key = key;

this.value = value;

this.left = left;

this.right = right;

}

}

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

**二叉查找树API的设计**

| **类名** | **Binary<Key extends Comparable，Value value>** |
| --- | --- |
| 构造方法 | BinaryTree():创建BinaryTree对象 |
| 成员变量 | 1.private Node root :根节点 2.private int N：记录树中节点个数 |
| 成员方法 | 1.public int size():获取树中节点个数 2.public void put(Key key,Value value):向树中插入一个键值对 3.private Node put(Node root,Key key ,Value value)： 向指定树root中插入键值对，返回新树 4.public Value get(Key key):根据键获取值 5.private Value get(Node root,Key key):从指定的树x中，找出key对应的值 6.public void delete(Key key):根据指定键值删除Value 7.private Node delete(Node root,Key key):删除指定树中键为key的value |

显示详细信息

**下面代码实现：**

(1)、获取树中节点个数 public int size()

public int size(){

return N;

}

(2)、向树中插入一个键值对 public void put (Key key,Value value)

public void put (Key key,Value value){

root = put(root,key,value);

}

(3)、向指定树中插入键值对public Node put(Node node,Key key ,Value value)

private Node put (Node root,Key key ,Value value){

//如果root为空

if(root == null){

N++;

return new Node(key,value,null,null);

}

//如果树不为空，就要找到一个合适的插入点

//跟当前点解比较大小

int i = key.compareTo(root.key)

//如果i大于当前的key，就从右子节点遍历

if(i > 0 ){

root.right = put(root.right,key,value);

}else if( i < 0){

//如果i小于当前key,就从左子节点遍历

root.left = put (root.left,key,value);

}else{

//在树中找到了key相同的值，就替换它的value

root.value = value;

}

return root;

}

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

4、根据键获取对应的值 public Value get(Key key)

public Value get(Key key){

return get(root,key);

}

5、从指定的树x中，找出key对应的值 public Value get(Node root,Key key)

private Value get(Node root,Key key){

//root为空的情况下

if( root == null){

return null;

}

//root 不为空的情况下

int i = key.compareTo(root.key);

if( i < 0 ){

//key小于当前的root.key，则继续从左子树找

return get(root.left,key);

}else if( i > 0 ){

//key大于当前root.key，则继续从右子树找

return get(root.right,key);

}else{

//找到了

return root.value;

}

}

https://csdnimg.cn/release/blogv2/dist/pc/img/newCodeMoreWhite.png

6、根据指定键删除值 public void delete(Key key)

public void delete(Key key){

root = delete(root,key);

}

7、在指定树中，根据指定键值删除值 public Node delete(Node root,Key key,Value value)

/\*\* 删除方法delete得闲的方法

\*1.找到被删除的节点

\*2.找到被删除节点中右子树的最小节点minNode;

\*3.删除被删除节点中右子树中的最小节点minNode

\*4.让被删除节点的左子树成为最小节点minNode的左子节点，被删除节点的右子树

\*成为最小节点的右子树

\*5.让被删除的节点的父节点指向最小节点minNode

\*\*/

private Node delete(Node root,Key key){

//root为null

if( root == null ){

return null ;

}

//root不为null

//找到要被删除的节点

int i = key .compareTo(root.key);

//这里跟上面查找的思想差不多

if(i < 0){

root.left = delete(root.left,key);

}else if( i > 0 ){

root.right = delete(root.right,key);

}else{

//找到了被删除的节点了

//树中节点数减少一

N--;

if(root.right == null){

return root.left;

}

if(root.left == null){

return root.right;

}

//记录树中最少的节点

Node minNode = x.left;

while(minNode.left != null){

minNode = minNode.left;

}

//删除被删除节点的右子树最少节点

Node n = root.right;

while(n.left != null){

if(n.left.left == null){

n.left = null;

}else{

n = n.left;

}

}

//让被删除节点的左子树成为最小节点minNode的左子节点

minNode.left = root.left;

//让被删除节点的右子树成为最小节点的右子树

minNode.right = root.right;

//让被删除节点的父节点指向minNode

root = MinNode;

}