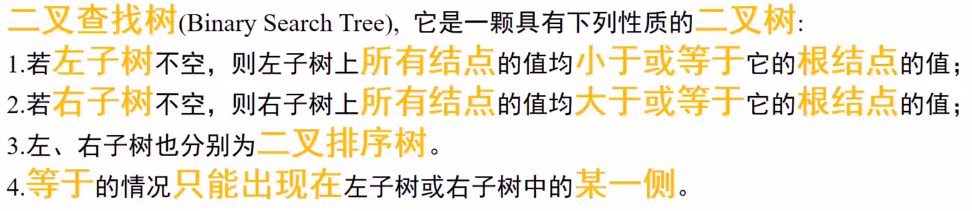
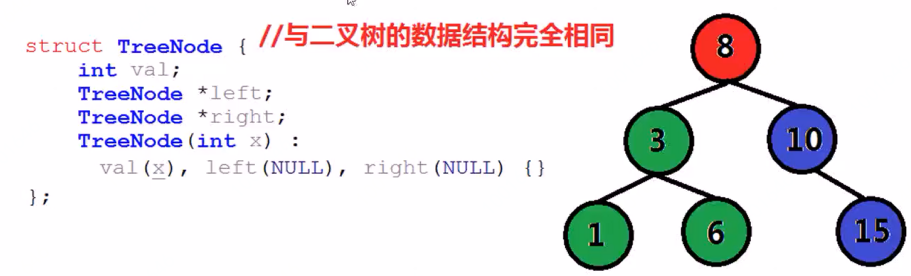
二叉查找树BinarySearchTree

# 二叉查找树BinarySearchTree

**二叉查找树BinarySearchTree又称二叉排序树BinarySortTree**；因为中序遍历输出就是从小到大排序。

## 定义





## 二叉排序树：

由于二叉查找树的**中序遍历**是从小到大的，故又称为**二叉排序树**(Binary Sort Tree)

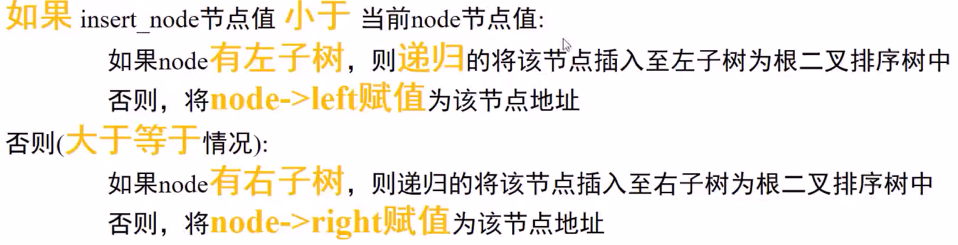
如**1->3->6->8->10->15**

# 向二叉查找树插入节点

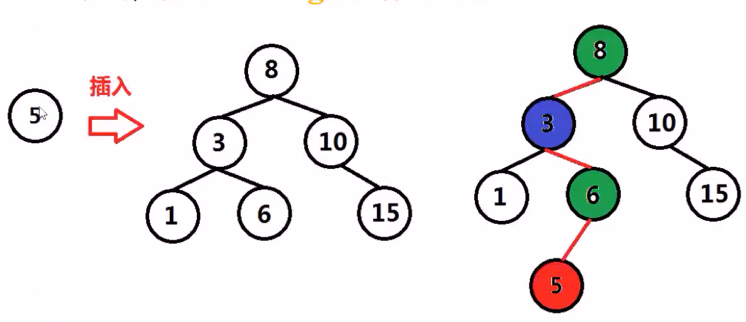
## 目标

将**某节点insert\_node**插入至以node为根二叉查找树中。

## 解决思路：



## 示例：



## 代码：

### C语言：



### Java代码

/\*\*

\* 向以root为根节点的二叉查找树中插入节点insertNode

\*/

public void BST\_insert(BSTNode root,BSTNode insertNode){

**if(insertNode.value <= root.value)**{

if(root.left != null){

BST\_insert(root.left,insertNode);//递归插入左子树

}else{

root.left = insertNode;

}

}else{

if(root.right != null){

BST\_insert(root.right,insertNode);//递归插入右子树

}else{

root.right = insertNode;

}

}

}

# 二叉查找树的构建

首先创建一些节点Node；然后指定一个为根节点root；然后不断调用上述的BST\_insert(root,node)方法，这样就创建了一个以root为根节点的二叉查找树。若再实现一个中序遍历，即将数值从小打大排序。

@Test

public void test(){

int[] nums = {2,3,4,5,-1,34,8,2,1,4};

List<BSTNode> nodeList = new ArrayList<BSTNode>();

for(int i = 0;i<nums.length;i++){//创建节点

nodeList.add(new BSTNode(nums[i]));

}

for(int i = 1;i<nodeList.size();i++){//构建二叉查找树

BST\_insert(nodeList.get(0),nodeList.get(i));

}

//二叉搜索树若按照中序遍历输出，即从小到大顺序，因此又称为二叉树排序树

List<Integer> result = inOrderTraversal(nodeList.get(0));

**System.out.println(result.toString());//从小到大排序**

//[-1, 1, 2, 2, 3, 4, 4, 5, 8, 34]

}

/\*\*

\* 定义二叉搜索树的数据结构

\*/

public class BSTNode{

BSTNode left;

BSTNode right;

int value;

public BSTNode(int value){

this.value = value;

}

}

/\*\*

\* 向以root为根节点的二叉查找树中插入节点insertNode

\*/

public void BST\_insert(BSTNode root,BSTNode insertNode){

if(insertNode.value <= root.value){

if(root.left != null){

BST\_insert(root.left,insertNode);//递归插入左子树

}else{

root.left = insertNode;

}

}else{

if(root.right != null){

BST\_insert(root.right,insertNode);//递归插入右子树

}else{

root.right = insertNode;

}

}

}

/\*\*

\* 中序遍历

\* @param root

\* @return

\*/

public List<Integer> inOrderTraversal(BSTNode root){

List<Integer> result = new ArrayList<Integer>();

Deque<BSTNode> stack = new ArrayDeque<BSTNode>();

while(!(stack.isEmpty()&&root == null)){

if(root != null){

stack.push(root);

root = root.left;

}else{

root = stack.pop();

result.add(root.value);

root = root.right;

}

}

return result;

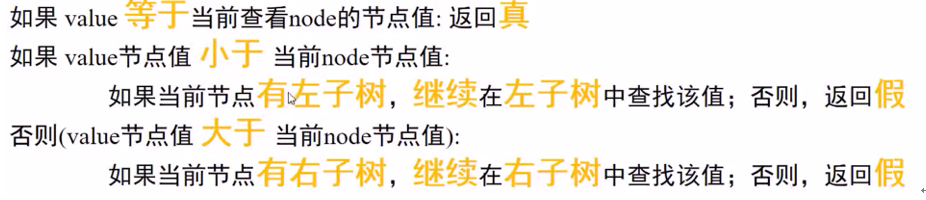
}

# 二叉查找树查找数值

## 目标

查找数值target是否在二叉查找树中出现?

## 解决思路



## 代码

### Java语言

/\*\*

\* 判断某个值是否在二叉查找树中；存在返回true，否则返回false

\*/

public boolean searchValue(BSTNode root,int value){

if(value == root.value) return true;

if(value < root.value){//小于根节点值

if(root.left != null){

**return searchValue(root.left,value);**

}else{

return false;

}

}else{//大于根节点值 (value > root.value)

if(root.right != null){

**return searchValue(root.right,value);**

}else{//相等

return false;

}

}

}

### C语言



## 改进：返回该节点或Null

### 题目：LeetCode\_700\_Search in a Binary Search Tree

难度：easy

<https://leetcode.com/problems/search-in-a-binary-search-tree/description/>

Given the root node of a binary search tree (BST) and a value. You need to find the node in the BST that the node's value equals the given value. Return the subtree rooted with that node. If such node doesn't exist, you should return NULL.

For example,

Given the tree:

4

/ \

2 7

/ \

1 3

And the value to search: 2

You should return this subtree:

2

/ \

1 3

In the example above, if we want to search the value 5, since there is no node with value 5, we should return **NULL**.

Note that an empty tree is represented by NULL, therefore you would see the expected output (serialized tree format) as [], not null.

### 解题思路

只需要在刚才基础上，方法返回BSTNode类型，将return语句改变一下，false变成null;true改为root即可。

### Java代码

/\*\*

\* 判断某个值是否在二叉查找树中；存在返回该节点，否则返回NULL

\* LeetCode\_700\_Search in a Binary Search Tree

\* 难度：easy

\* <https://leetcode.com/problems/search-in-a-binary-search-tree/description/>

\* 注意：需要将BSTNode中的value改为val；BSTNode 换成TreeNode

\*/

public **BSTNode** **searchBST**(BSTNode root,int value){

if(root == null) return null;

if(value == root.value) return root;

if(value < root.value){//小于根节点值

if(root.left != null){

return searchBST(root.left,value);

}else{

return null;

}

}else{//大于根节点值 (value > root.value)

if(root.right != null){

return searchBST(root.right,value);

}else{//相等

return null;

}

}

}

# 二叉查找树编码与解码(序列化与反序列化)

**LeetCode**\_449\_Serialize and Deserialize BST

<https://leetcode.com/problems/serialize-and-deserialize-bst/description/>

难度：Medium

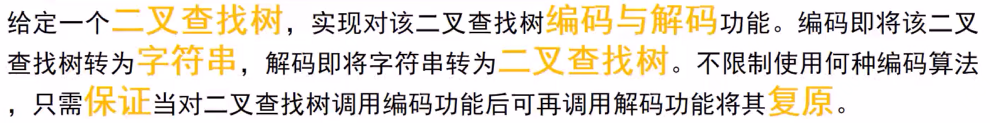
## 题目

**Serialization** is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link to be **reconstructed** later in the same or another computer environment.

Design an algorithm to **serialize and deserialize a binary search tree**. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that a binary search tree can be serialized to a string and this string can be deserialized to the original tree structure.

The encoded string should be as compact as possible.

Note: Do not use class member/global/static variables to store states. Your serialize and deserialize algorithms should be stateless.



/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

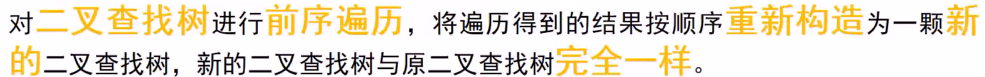
## 问题分析

二叉查找树遍历之后，能否复原？

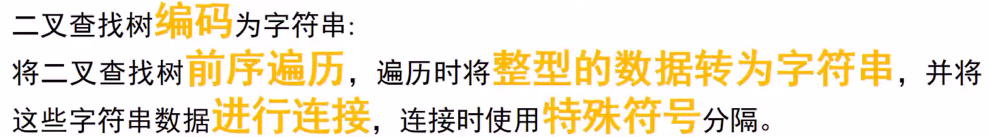
**前序遍历是可以的：首先前序遍历，然后依次插入节点，可以复原。**

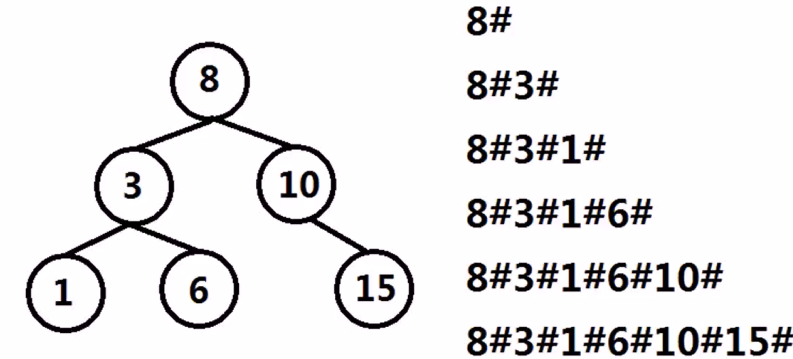
中序遍历和后续遍历都是不可以的。(仔细分析下,因为第一个元素就不是原来的root了)

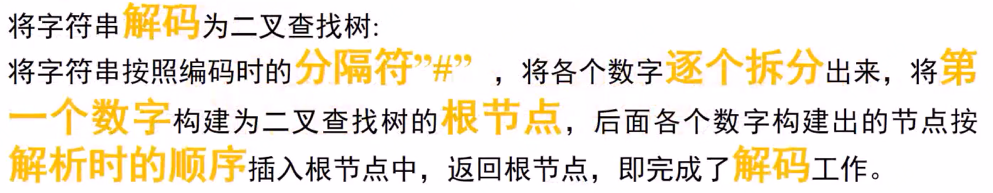
因此，**解题思路**就是

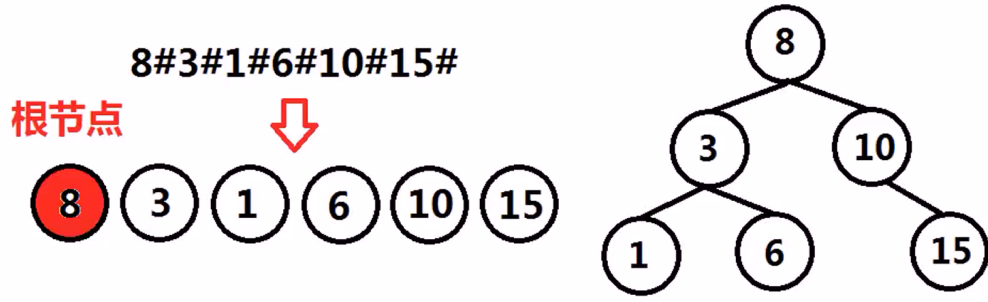


当然这里还存在序列化和反序列化的问题，因此需要将**节点的int值**拼接成字符串，然后还需要将字符串转换为**节点**。(具体分析见后面)这里直接使用Integer.parse方法。









**重点掌握**：**只有前序遍历之后序列化，然后反序列化才能恢复原来的结构**，对于中序、后序都是不行的。

## 完整的Java代码

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode(int x) { val = x; }

\* }

\*/

public class Codec {

/\*\*

\* Encodes a tree to a single string.

\* 利用前序遍历将二叉查找树的值拼接成字符串

\* 注意：根节点为null时返回空字符串

\*/

public String **serialize**(TreeNode root) {

if(root == null) return "";

StringBuilder sb = new StringBuilder();

Deque<TreeNode> stack = new ArrayDeque<TreeNode>();

while(!(stack.isEmpty()&&root == null)){

if(root != null){

stack.push(root);

sb.append(root.val+"#");

root = root.left;

}else{

root = stack.pop().right;

}

}

return sb.toString().substring(0,sb.length()-1);

}

/\*\*

\* Decodes your encoded data to tree.

\* 利用二叉查找树的节点插入方法重新构建二叉查找树

\* 经过验证：按照前序遍历的顺序回复二叉查找树与原来的二叉查找树是一致的

\* 中序遍历与后序遍历的顺序肯定不行，因为第一个节点就不是原来的根节点root

\* 注意：空字符情况对应的是二叉查找树为空，返回null即可

\*/

public TreeNode **deserialize**(String data) {

if(data == null|| data.length() == 0) return null;//对空字符串处理

String[] values = data.split("#");

List<TreeNode> nodes = new ArrayList<TreeNode>();

for(String value:values){

nodes.add(new TreeNode(Integer.parseInt(value)));//转换成节点

}

for(int i = 1;i< nodes.size();i++){//重建二叉查找树(按照前序遍历获取的顺序逐个插入，可以恢复(经过实践证明))

BST\_insert(nodes.get(0),nodes.get(i));

}

return nodes.get(0);

}

/\*\*

\* 向以root为根节点的二叉查找树中插入节点insertNode

\*/

public void **BST\_insert**(TreeNode root, TreeNode insertNode){

if(insertNode.val <= root.val){

if(root.left != null){

BST\_insert(root.left,insertNode);//递归插入左子树

}else{

root.left = insertNode;

}

}else{

if(root.right != null){

BST\_insert(root.right,insertNode);//递归插入右子树

}else{

root.right = insertNode;

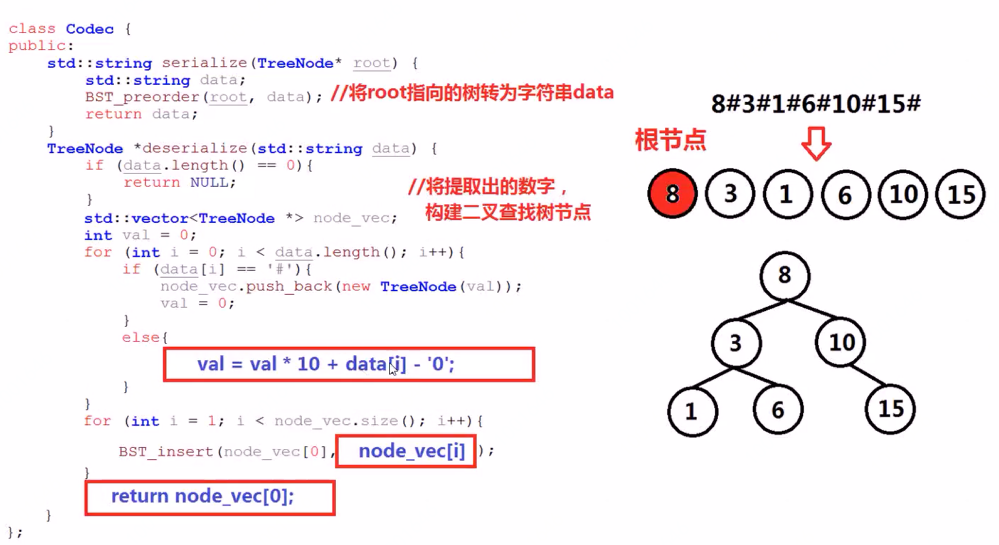
}

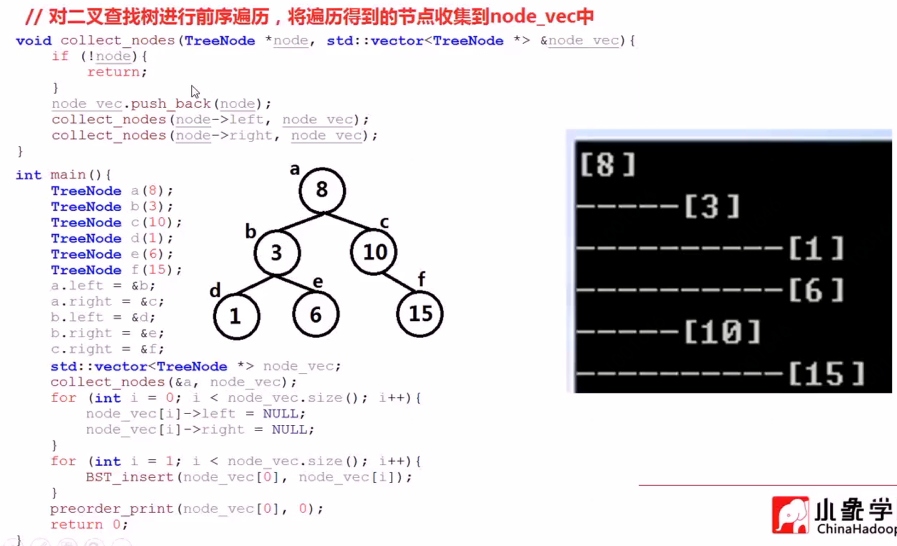
}

}

}

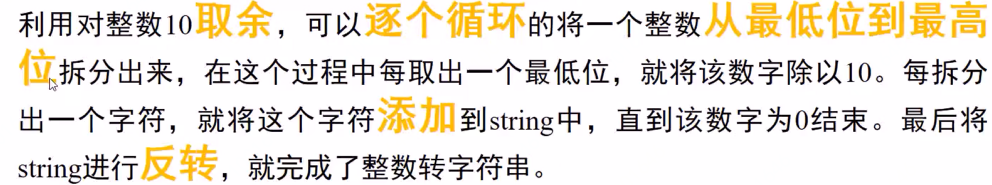
## C语言代码





# 整型与字符串之间的转换

## 整型转为字符串





C语言代码



## 字符串拆分为整数

