**LeetCode 449. Serialize and Deserialize BST**

2017年01月26日 16:36:46

阅读数：312

**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.

**思路：**

* 使用BFS，把所有空节点保存成null，对于非BST应该是唯一做法。BST只需要保存前序遍历就可以。
* 使用DFS，迭代的编码解码。利用BST左边>中间>右边的特性，可以不保存null，在解码时使用lower和upper判断是否是自己的左右节点。（这个递归我写了一下午，还是需要多做题啊）

**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.

public String serialize(TreeNode root) {

StringBuffer sb = new StringBuffer();

serializeDFS(root,sb);

return sb.toString();

}

private void serializeDFS(TreeNode root,StringBuffer sb){

if(root==null) return;

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

serializeDFS(root.left, sb);

serializeDFS(root.right, sb);

}

// Decodes your encoded data to tree.

public TreeNode deserialize(String data) {

if(data.length() == 0) return null;

String[] strArr = data.split(" ");

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

list.add(0);

return deserializeDFS(strArr,list,Integer.MIN\_VALUE,Integer.MAX\_VALUE);

}

private TreeNode deserializeDFS(String[] data,List<Integer> list,int lower,int upper){

//获取位置

int index = list.get(0);

//如果值不在当前范围返回null

int val = Integer.parseInt(data[index]);

if(val<lower||val>upper) return null;

//如果位置在最后直接返回

if(index==data.length-1) return new TreeNode(val);

//新建点,位置+1,可能有左右孩子

TreeNode node = new TreeNode(val);

list.set(0,++index);

// 下一节点在左边

if(Integer.parseInt(data[list.get(0)])<node.val){

node.left = deserializeDFS(data,list,lower,node.val);

}

// 下一节点在右边

if(Integer.parseInt(data[list.get(0)])>node.val){

node.right = deserializeDFS(data,list,node.val,upper);

}

return node;

}

}

// Your Codec object will be instantiated and called as such:

// Codec codec = new Codec();

// codec.deserialize(codec.serialize(root));

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46
* 47
* 48
* 49
* 50
* 51
* 52
* 53
* 54
* 55
* 56
* 57
* 58
* 59
* 60
* 61
* 62
* 使用队列把BST的前序遍历恢复，小的数放到新队列，大的数放剩在原来的队列里，递归调用。利用了一个新队列，复杂度略高。

// some notes:

// 5

// 3 6

// 2 7

private TreeNode getNode(Queue<Integer> q) { //q: 5,3,2,6,7

if (q.isEmpty()) return null;

TreeNode root = new TreeNode(q.poll());//root (5)

Queue<Integer> samllerQueue = new LinkedList<>();

while (!q.isEmpty() && q.peek() < root.val) {

samllerQueue.offer(q.poll());

}

//smallerQueue : 3,2 storing elements smaller than 5 (root)

root.left = getNode(samllerQueue);

//q: 6,7 storing elements bigger than 5 (root)

root.right = getNode(q);

return root;

}