水中的鱼

zhang lei

关注者列表未公开

2012年12月30日星期日

[LeetCode] Recover Binary Search Tree 解题报告

Two elements of a binary search tree (BST) are swapped by mistake.

Recover the tree without changing its structure.

Note:
A solution using O(*n*) space is pretty straight forward. Could you devise a constant space solution?

confused what `"{1,#,2,3}"` means? > read more on how binary tree is serialized on OJ.

» Solve this problem

[解题报告]
O(n)空间的解法比较直观，中序遍历一遍以后，重新赋值一遍即可，这个解法可以面向n个元素错位的情况。但是对于O(1)空间的解法，最开始的想法是，可以考虑采用类似最大堆的调正过程的算法，但是这样又可能会破坏树的原有结构。暂未想出来解法。

只能给个O(n)空间的解法。

[Code]

```
1:    void recoverTree(TreeNode *root) {
2:      // Start typing your C/C++ solution below
3:      // DO NOT write int main() function
4:      vector<TreeNode*> list;
5:      vector<int > vals;
6:         InOrderTravel(root, list, vals);
7:            sort(vals.begin(), vals.end());
8:            for(int i =0; i< list.size(); i++)
9:            {
10:                 list[i]->val = vals[i];
11:           }
12:   }
13:      void InOrderTravel(TreeNode* node, vector<TreeNode*>& list, vector<int>& vals)
14:      {
15:           if(node == NULL) return;
16:           InOrderTravel(node->left, list, vals);
17:           list.push_back(node);
18:           vals.push_back(node->val);
19:           InOrderTravel(node->right, list, vals);
20:      }
```

Updates: 3/16/2013
Searched in the web. Actually, there is a smart solution to travel the tree with two node.
The link is http://discuss.leetcode.com/questions/272/recover-binary-search-tree

```
1:  void Solution::recoverTree(TreeNode *h) {
2:    TreeNode *f1=NULL, *f2=NULL;
3:    bool found = false;
4:    TreeNode *pre, *par = 0; // previous AND parent
5:    while(h) { // Morris Traversal
6:      if(h->left == 0) {
7:        if(par && par->val > h->val) { // inorder previous is: par
8:          if(!found) {
9:            f1 = par;
10:           found = true;
11:         }
12:         f2 = h;
13:       }
14:       par = h;
```

我的简介

zhang lei

关注 0

小磊哥

查看我的完整个人资料

Follow @codingtmd

Visit Map

标签

- Algorithm (138)
- BFS (1)
- binary tree (7)
- Bit Operation (1)
- Brute-force (2)
- design (1)
- DFS (3)
- DP (16)
- facebook (1)
- google (1)
- hash (1)
- LeetCode (121)
- Math (1)
- recursion (4)
- review (22)
- sort (1)
- SRM (6)
- TopCoder (7)
- two-pointer (5)
- Ubuntu (1)
- Yahoo (1)
- 二分搜索 (7)
- 二叉树 (15)
- 动态规划 (20)
- 双指针 (16)
- 图 (1)
- 字符串处理 (8)
- 实现题 (7)
- 数组 (8)
- 模拟 (23)
- 解题报告 (22)
- 贪心 (1)
- 递归 (32)

```
15:       h = h->right;
16:       continue;
17:     }
18:     pre = h->left;
19:     while(pre->right != 0 && pre->right != h)
20:       pre = pre->right;
21:     if(pre->right == 0) {
22:       pre->right = h;
23:       h = h->left;
24:     } else {
25:       pre->right = 0;
26:       if(pre->val > h->val) { // inorder previous is: pre
27:         if(!found) {
28:           f1 = pre;
29:           found =true;
30:         }
31:         f2 = h;
32:       }
33:       par = h;
34:       h = h->right;
35:     }
36:   }
37:   if(found)
38:     swap(f1->val, f2->val);
39: }
```

Update: 3/21/2013 上一个解法不容易看清楚，添加分析。
O(1)的解法就是
Inorder traverse, keep the previous tree node,
Find first misplaced node by
if (current.val < prev.val)
  Node first = prev;

Find second by
if (current.val < prev.val)
  Node second = current;

After traversal, swap the values of first and second node. Only need two pointers, prev and current node. O(1) space.

但是这个解法的前提是Traverse Tree without Stack. 中序遍历如何才能不使用栈。这里就要引入一个概念， Threaded Binary Tree。So, we first create links to Inorder successor and print the data using these links, and finally revert the changes to restore original tree.

```
1. Initialize current as root
2. While current is not NULL
   If current does not have left child
      a) Print current's data
      b) Go to the right, i.e., current = current->right
   Else
      a) Make current as right child of the rightmost node in current's left subtr
ee
      b) Go to this left child, i.e., current = current->left
```

代码如下：

```
/* Function to traverse binary tree without recursion and without stack */
vector<int> inorderTraversal(TreeNode *root)
{
	vector<int> result;
	TreeNode  *current,*pre;

	if(root == NULL)
	return result;

	current = root;
	while(current != NULL)
	{
		if(current->left == NULL)
		{
			result.push_back(current->val);
			current = current->right;
		}
		else
```

```
		{
			/* Find the inorder predecessor of current */
			pre = current->left;
			while(pre->right != NULL && pre->right != current)
				pre = pre->right;

			/* Make current as right child of its inorder predecessor */
			if(pre->right == NULL)
			{
				pre->right = current;
				current = current->left;
			}

			/* Revert the changes made in if part to restore the original
		    tree i.e., fix the right child of predecssor */
			else
			{
				pre->right = NULL;
				result.push_back(current->val);
				current = current->right;
			} /* End of if condition pre->right == NULL */
		} /* End of if condition current->left == NULL*/
	} /* End of while */

	return result;
}
```

那么，基于这个双指针遍历，可以把错置节点的判断逻辑加进去，就可以完美的在O(1)空间内，完成树的重构。

[Code]
改动代码如红字所示。增加了一个pointer – parent来记录上一个访问节点。整个遍历过程中，使用（parent->val > current->val）来寻找违规节点，但是区别是，要获取第一次violation的parent和第二次violation的current，然后交换。

```
void recoverTree(TreeNode *root)
{
	TreeNode *f1=NULL, *f2=NULL;
	TreeNode  *current,*pre, *parent=NULL;

	if(root == NULL)
		return;
	bool found = false;
	current = root;
	while(current != NULL)
	{
		if(current->left == NULL)
		{
			if(parent && parent->val > current->val)
			{
				if(!found)
				{
					f1 = parent;
					found = true;
				}
				f2 = current;
			}
			parent = current;
			current = current->right;
		}
		else
		{
			/* Find the inorder predecessor of current */
			pre = current->left;
			while(pre->right != NULL && pre->right != current)
				pre = pre->right;

			/* Make current as right child of its inorder predecessor */
			if(pre->right == NULL)
			{
				pre->right = current;
				current = current->left;
			}

			/* Revert the changes made in if part to restore the original
			tree i.e., fix the right child of predecssor */
			else
			{
				pre->right = NULL;
				if(parent->val > current->val)
				{
					if(!found)
```

in Histogram 解题报告
[LeetCode] Jump Game II 解题报告
[LeetCode] Jump Game 解题报告
[LeetCode] Interleaving String 解题报告
[LeetCode] Integer to Roman 解题报告
[LeetCode] Insert Interval 解题报告
[LeetCode] Implement strStr() 解题报告
[LeetCode] Gray Code 解题报告
[LeetCode] Generate Parentheses 解题报告
[LeetCode] Flatten Binary Tree to Linked List 解题报告...
[LeetCode] First Missing Positive 解题报告
[LeetCode] Edit Distance 解题报告
[LeetCode] Divide Two Integers 解题报告
[Leetcode] Distinct Subsequences 解题报告
► 2007 (1)

```
                        {
                                f1 = parent;
                                found = true;
                        }
                        f2 = current;
                    }
                    parent = current;
                    current = current->right;
                } /* End of if condition pre->right == NULL */
            } /* End of if condition current->left == NULL*/
        } /* End of while */

        if(f1 && f2)
            swap(f1->val, f2->val);
}
```

发帖者 zhang lei 时间： 下午5:46 +4 在 Google 上推荐

标签： Algorithm, LeetCode, review, 二叉树

3 条评论

以“Zerg Do”的身份发表评论

热门评论

Xuezhi Yan 1 周前 - 公开分享

如果把stack占用的空间也考虑进来的话，threaded binary tree的建立阶段寻找rightmost node of left subtree的过程就已经用了 O(log n)的space了吧？

1 · 回复

Wuzhenni hu 5 个月前 - 公开分享

这个解法就是inorder traverse变体一下，并没有考虑到那个thread binary tree吧，你可以看看。。

```
class Solution {
public:
  void find(TreeNode *cur, TreeNode *&p1, TreeNode *&p2, TreeNode *&pre)
```

展开（23行）

+1 1 · 回复

Guochao Ren 4 个月前

中序遍历需要在栈上分配至少O(logN)的空间，最坏是O(n)，所以不符合题目要求，题目中要求O(1)

貌似只能用thread binary tree

发表评论

较新的帖子 主页 较早的帖子

订阅： 帖子评论 (Atom)