Morris Traversal方法遍历二叉树（非递归，不用栈，O(1)空间）

本文主要解决一个问题，如何实现二叉树的前中后序遍历，有两个要求：

1. O(1)空间复杂度，即只能使用常数空间；

2. 二叉树的形状不能被破坏（中间过程允许改变其形状）。

通常，实现二叉树的前序（preorder）、中序（inorder）、后序（postorder）遍历有两个常用的方法：一是递归(recursive)，二是使用栈实现的迭代版本(stack+iterative)。这两种方法都是O(n)的空间复杂度（递归本身占用stack空间或者用户自定义的stack），所以不满足要求。（用这两种方法实现的中序遍历实现可以参考 这里 。）

Morris Traversal方法可以做到这两点，与前两种方法的不同在于该方法只需要O(1)空间，而且同样可以在O(n)时间内完成。

要使用O(1)空间进行遍历，最大的难点在于，遍历到子节点的时候怎样重新返回到父节点（假设节点中没有指向父节点的p指针），由于不能用栈作为辅助空间。为了解决这个问题，Morris方法用到了 线索二叉树 （threaded binary tree）的概念。在Morris方法中不需要为每个节点额外分配指针指向其前驱（predecessor）和后继节点（successor），只需要利用叶子节点中的左右空指针指向某种顺序遍历下的前驱节点或后继节点就可以了。

Morris只提供了中序遍历的方法，在中序遍历的基础上稍加修改可以实现前序，而后续就要再费点心思了。所以先从中序开始介绍。

首先定义在这篇文章中使用的二叉树节点结构，即由val，left和right组成：

```
1 struct TreeNode {
2     int val;
3     TreeNode *left;
4     TreeNode *right;
5     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
6 };
```

一、中序遍历

步骤：

1. 如果当前节点的左孩子为空，则输出当前节点并将其右孩子作为当前节点。

2. 如果当前节点的左孩子不为空，在当前节点的左子树中找到当前节点在中序遍历下的前驱节点。

  a) 如果前驱节点的右孩子为空，将它的右孩子设置为当前节点。当前节点更新为当前节点的左孩子。

  b) 如果前驱节点的右孩子为当前节点，将它的右孩子重新设为空（恢复树的形状）。输出当前节点。当前节点更新为当前节点的右孩子。

3. 重复以上1、2直到当前节点为空。

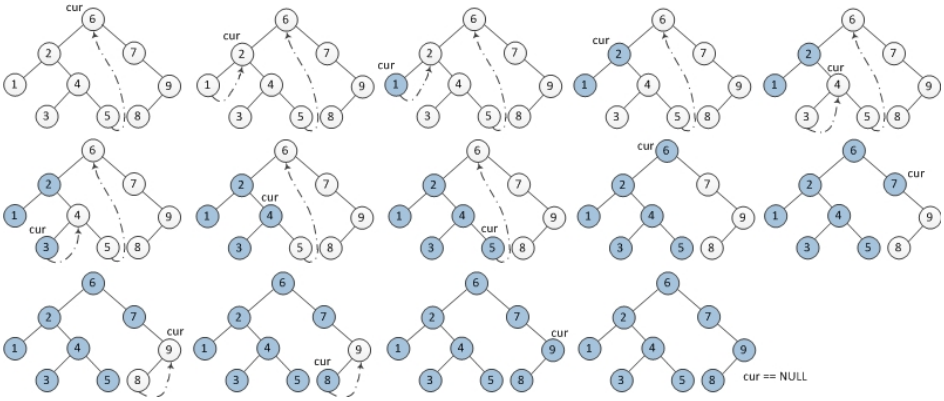
图示：

下图为每一步迭代的结果（从左至右，从上到下），cur代表当前节点，深色节点表示该节点已输出。

公告

anniekim.pku[at]gmail.com
完整LeetCode方案请参考我的GitHub→_→

昵称： AnnieKim
园龄： 8年5个月
粉丝： 93
关注： 15
+加关注



代码：

```
1 void inorderMorrisTraversal(TreeNode *root) {
2     TreeNode *cur = root, *prev = NULL;
3     while (cur != NULL)
4     {
5         if (cur->left == NULL)          // 1.
6         {
7             printf("%d ", cur->val);
8             cur = cur->right;
9         }
10        else
11        {
12            // find predecessor
13            prev = cur->left;
14            while (prev->right != NULL && prev->right != cur)
15                prev = prev->right;
16
17            if (prev->right == NULL)   // 2.a)
18            {
19                prev->right = cur;
20                cur = cur->left;
21            }
22            else                       // 2.b)
23            {
24                prev->right = NULL;
25                printf("%d ", cur->val);
26                cur = cur->right;
27            }
28        }
29    }
30 }
```

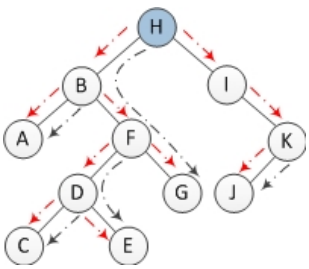
复杂度分析：

空间复杂度：O(1)，因为只用了两个辅助指针。

时间复杂度：O(n)。证明时间复杂度为O(n)，最大的疑惑在于寻找中序遍历下二叉树中所有节点的前驱节点的时间复杂度是多少，即以下两行代码：

```
1 while (prev->right != NULL && prev->right != cur)
2     prev = prev->right;
```

直觉上，认为它的复杂度是O(nlgn)，因为找单个节点的前驱节点与树的高度有关。但事实上，寻找所有节点的前驱节点只需要O(n)时间。n个节点的二叉树中一共有n-1条边，整个过程中每条边最多只走2次，一次是为了定位到某个节点，另一次是为了寻找上面某个节点的前驱节点，如下图所示，其中红色是为了定位到某个节点，黑色线是为了找到前驱节点。所以复杂度为O(n)。



这个指针的错误。 printReverse(cur->left, pre) 里面经过两次 reverse 之后，最终的确会导致 p...

--wendabei

4. Re:Morris Traversal方法…

@ yanchao107327楼代码是对的，但是说的话是错的。。您给的后序遍历思想是很不错的，但是想要逆序再输出，还是用到了额外O(n)的空间，与本文的理念不合。...

--wendabei

5. Re:Morris Traversal方法…

emmm.....楼主,很感谢整理的这篇文章，让我了解了很多。但是在中序遍历那里的，感觉有个说法有点错误，就是每条边走两遍，我觉得应该是走三遍，一次是为了定位到某个节点，一次是为了寻找上面某个节点的前...

--林学徒

阅读排行榜

1. Morris Traversal方法遍历…
2. Java调用C/C++编写的第…
3. [LeetCode(Q69)] Sqrt(x) (…
4. [LeetCode(Q41)] First Mis…
5. 恼人的函数指针（二）：…

评论排行榜

1. Morris Traversal方法遍历…
2. [LeetCode(Q69)] Sqrt(x) (…
3. Java调用C/C++编写的第…
4. 恼人的函数指针（二）：…
5. [LeetCode(Q41)] First Mis…

推荐排行榜

1. Morris Traversal方法遍历…
2. Java调用C/C++编写的第…
3. 恼人的函数指针（二）：…
4. 恼人的函数指针（一）(9)
5. const限定修饰符用法总结…

二、前序遍历

前序遍历与中序遍历相似，代码上只有一行不同，不同就在于输出的顺序。

步骤：

1. 如果当前节点的左孩子为空，则输出当前节点并将其右孩子作为当前节点。

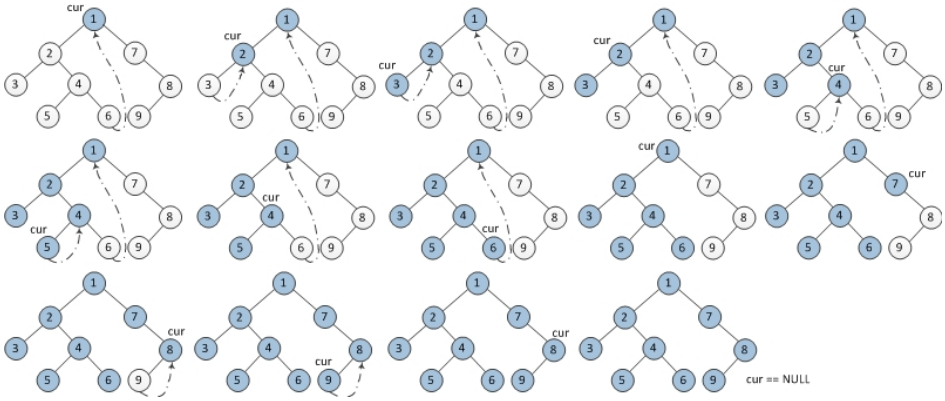
2. 如果当前节点的左孩子不为空，在当前节点的左子树中找到当前节点在中序遍历下的前驱节点。

 a) 如果前驱节点的右孩子为空，将它的右孩子设置为当前节点。**输出当前节点（在这里输出，这是与中序遍历唯一一点不同）**。当前节点更新为当前节点的左孩子。

 b) 如果前驱节点的右孩子为当前节点，将它的右孩子重新设为空。当前节点更新为当前节点的右孩子。

3. 重复以上1、2直到当前节点为空。

图示：



代码：

```
1 void preorderMorrisTraversal(TreeNode *root) {
2     TreeNode *cur = root, *prev = NULL;
3     while (cur != NULL)
4     {
5         if (cur->left == NULL)
6         {
7             printf("%d ", cur->val);
8             cur = cur->right;
9         }
10        else
11        {
12            prev = cur->left;
13            while (prev->right != NULL && prev->right != cur)
14                prev = prev->right;
15
16            if (prev->right == NULL)
17            {
18                printf("%d ", cur->val);  // the only difference with inorder-
traversal
19                prev->right = cur;
20                cur = cur->left;
21            }
22            else
23            {
24                prev->right = NULL;
25                cur = cur->right;
26            }
27        }
28    }
29 }
```

复杂度分析：

时间复杂度与空间复杂度都与中序遍历时的情况相同。

三、后序遍历

后续遍历稍显复杂，需要建立一个临时节点dump，令其左孩子是root。并且还需要一个子过程，就是倒序输出某两个节点之间路径上的各个节点。

步骤：

当前节点设置为临时节点dump。

1. 如果当前节点的左孩子为空，则将其右孩子作为当前节点。

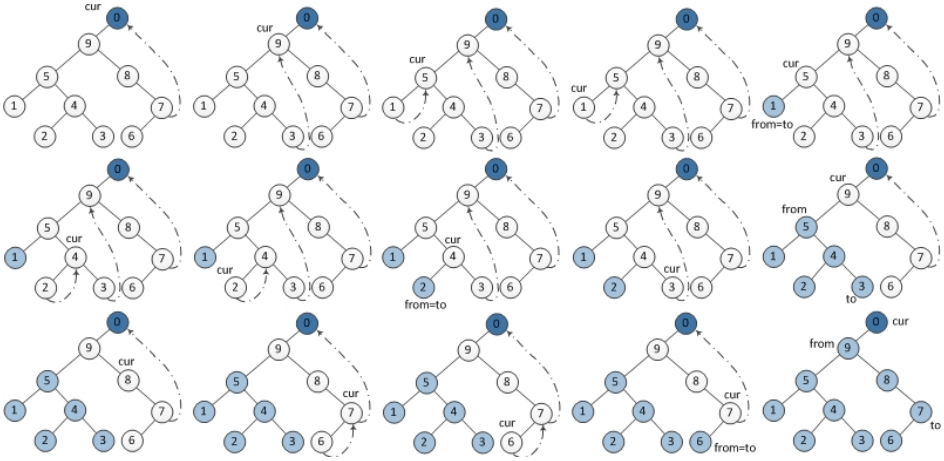
2. 如果当前节点的左孩子不为空，在当前节点的左子树中找到当前节点在中序遍历下的前驱节点。

  a) 如果前驱节点的右孩子为空，将它的右孩子设置为当前节点。当前节点更新为当前节点的左孩子。

  b) 如果前驱节点的右孩子为当前节点，将它的右孩子重新设为空。**倒序输出从当前节点的左孩子到该前驱节点这条路径上的所有节点。**当前节点更新为当前节点的右孩子。

3. 重复以上1、2直到当前节点为空。

图示：



代码：

```
 1 void reverse(TreeNode *from, TreeNode *to) // reverse the tree nodes 'from' ->
'to'.
 2 {
 3     if (from == to)
 4         return;
 5     TreeNode *x = from, *y = from->right, *z;
 6     while (true)
 7     {
 8         z = y->right;
 9         y->right = x;
10         x = y;
11         y = z;
12         if (x == to)
13             break;
14     }
15 }
16
17 void printReverse(TreeNode* from, TreeNode *to) // print the reversed tree nodes
'from' -> 'to'.
18 {
19     reverse(from, to);
20
21     TreeNode *p = to;
22     while (true)
23     {
24         printf("%d ", p->val);
25         if (p == from)
26             break;
27         p = p->right;
28     }
29
30     reverse(to, from);
31 }
32
33 void postorderMorrisTraversal(TreeNode *root) {
34     TreeNode dump(0);
35     dump.left = root;
36     TreeNode *cur = &dump, *prev = NULL;
37     while (cur)
38     {
```

```
39          if (cur->left == NULL)
40          {
41              cur = cur->right;
42          }
43          else
44          {
45              prev = cur->left;
46              while (prev->right != NULL && prev->right != cur)
47                  prev = prev->right;
48
49              if (prev->right == NULL)
50              {
51                  prev->right = cur;
52                  cur = cur->left;
53              }
54              else
55              {
56                  printReverse(cur->left, prev);  // call print
57                  prev->right = NULL;
58                  cur = cur->right;
59              }
60          }
61      }
62  }
```

复杂度分析：

空间复杂度同样是O(1)；时间复杂度也是O(n)，倒序输出过程只不过是加大了常数系数。

注：

以上所有的代码以及测试代码可以在 我的Github 里获取。

参考：

http://www.geeksforgeeks.org/inorder-tree-traversal-without-recursion-and-without-stack/
http://www.geeksforgeeks.org/morris-traversal-for-preorder/
http://stackoverflow.com/questions/6478063/how-is-the-complexity-of-morris-traversal-on
http://blog.csdn.net/wdq347/article/details/8853371
Data Structures and Algorithms in C++ by Adam Drozdek

以前我只知道递归和栈+迭代实现二叉树遍历的方法，昨天才了解到有使用O(1)空间复杂度的方法。以上都是我参考了网上的资料加上个人的理解来总结，如果有什么不对的地方非常欢迎大家的指正。

分类: Algorithm

标签: Morris Traversal , Morris Inorder Traversal , without stack , 二叉树遍历 , 常数空间中序遍历 , 线索二叉树中序遍历

AnnieKim
关注 - 15
粉丝 - 93
+加关注

23 0

« 上一篇： [Google Code Jam] 2013-1A-C Good Luck 解法

posted @ 2013-06-15 18:22 AnnieKim 阅读(70508) 评论(34) 编辑 收藏

评论列表

#1楼 2013-06-15 19:14 dark_dream

求问楼主画图是用什么画的?

支持(0) 反对(0)

#2楼 [楼主] 2013-06-15 19:36 AnnieKim

@ dark_dream
是用visio画的，导出到jpeg:)

支持(0)　反对(0)

#3楼　2013-06-15 22:11 dark_dream

@ AnnieKim
谢！！

支持(0)　反对(0)

#4楼　[楼主] 2013-06-15 22:58 AnnieKim

@ dark_dream
不客气~

支持(0)　反对(0)

#5楼　2013-06-16 01:48 C4ISR

这个博文说的很清楚

支持(0)　反对(0)

#6楼　2013-06-24 13:47 啊超~

线索二叉树的确能解决遍历的问题，很充分的利用了叶子节点的空链域，但是我我觉得你的描述有点冗余了，我们可否这样描述，就拿中序来说，比如当前节点的左子树不为空，这个时候只要找到当前节点左子树中最右的叶子节点，即是当前节点的直接前驱，后继的话也是同样的描述（后继就是最左叶子节点了）

支持(4)　反对(0)

#7楼　[楼主] 2013-06-24 17:48 AnnieKim

@ 啊超~
中序遍历下的前驱节点是左子树中最右的叶子节点，我这里并没有用到后继，只是令前驱节点指向当前节点。
我是按照我将来回忆起来比较方便的方法描述，表达可能显得有些拖沓，以后会慢慢改进:)

支持(0)　反对(0)

#8楼　2013-06-24 18:04 啊超~

@ AnnieKim
恩，是的，理解最重要
给你的只是一个小小的建议，具体的你也可以看下严蔚敏的数据结构，讲的还是比较清晰的，虽然大部分是伪码

支持(0)　反对(0)

#9楼　[楼主] 2013-06-24 18:45 AnnieKim

@ 啊超~
你说看严蔚敏的数据结构，是看线索二叉树吧。
这里关键还是应用。
不过还是谢谢你的建议。

支持(0)　反对(0)

#10楼　2013-07-07 17:00 sing1ee

还是不太理解，查找每一个节点的中序遍历前驱的时候，为何平均是O(1)，而不是O(logn)。

支持(0)　反对(0)

#11楼　[楼主] 2013-07-07 18:58 AnnieKim

@ sing1ee
我认为对于单个节点来讲，平均情况下查找复杂度是O(lgn)，最好时是O(1)。

考虑所有节点总的查找复杂度的话，是O(n)（文章里有解释）。而且这并不代表对于单个节点是O(1)啊，有些节点是O(1)，有些是O(lgn)。

支持(0) 反对(0)

#12楼 2013-07-07 19:00 sing1ee

嗯，后来仔细看了你的解释。
明白了。
多谢。

支持(0) 反对(0)

#13楼 [楼主] 2013-07-07 19:05 AnnieKim

@ sing1ee
嗯嗯，你太客气了哈~:)

支持(0) 反对(0)

#14楼 2013-07-20 16:32 云戈

请问博主，后序遍历的时候reverse函数的思想是怎样的？x、y、y->right的颠倒我看了好久都没看懂，可否详加解释？多谢

支持(0) 反对(0)

#15楼 [楼主] 2013-07-20 17:12 AnnieKim

@ 云戈
给你举一个例子吧。
调用reverse(1, 3)之前和之后分别如下(这里的->都是指向右孩子的指针)：
前：1->2->3 后：3->2->1
调用reverse主要是为了倒序输出，输出完之后需要再调用reverse恢复原来的树的形状。
个人觉得morris的后续遍历太复杂了，所以网上也很少有讲后续的方案，还是中序和前序比较多。

支持(1) 反对(0)

#16楼 2013-07-20 19:16 云戈

@ AnnieKim
嗯，多谢。reverse函数的功能我是理解的，主要不是很理解以下四句：
z = y->right;
y->right = x;
x = y;
y = z;
刚才在纸上又画了画，已经弄懂了，多谢

支持(0) 反对(0)

#17楼 2013-09-19 20:16 rannn

其实后续遍历一个更简单的方法就是把前序遍历的代码原本access left的地方access right，原本right的地方取left， 并且将原本需要输出的地方入栈。全部走完后最后再出栈print

```
static void MorrisPostOrder(Node root)
{
    if (root == null) return;
    var current = root;
    var stack = new Stack<int>();
    while (current != null)
    {
        if (current.Right != null)
        {
            var t = current.Right;
            while (t.Left != null && t.Left != current)
            {
```

```
                t = t.Left;
            }

            if (t.Left == null)
            {
                stack.Push(current.D);
                t.Left = current;
                current = current.Right;
            }
            else
            {
                t.Left = null;
                current = current.Left;
            }
        }
        else
        {
            stack.Push(current.D);
            current = current.Left;
        }
    }

    foreach (var i in stack)
    {
        Console.Write(i + " ");
    }
}
```

支持(0) 反对(4)

#18楼 [楼主] 2013-09-22 17:31 AnnieKim

@ rannn
您的方法给了我一点提示，就是空的左指针也可以利用起来，到目前为止我利用的都是叶子节点内部闲置的右指针。
按照您的方法，如果要把后续遍历从头到尾打印出来，程序里的stack还是要占用O(n)空间，这并不是我这篇文章想要实现的。
谢谢您的评论。

支持(1) 反对(0)

#19楼 2014-03-13 17:24 peimin.org

博主 想问下 你博客中 那个类似于CSDN的代码 格式化 是怎么做的 谢谢

支持(0) 反对(0)

#20楼 2014-03-13 17:24 peimin.org

@ rannn
想问下 你这个类似于CSDN的代码格式化 怎么做的 谢谢

支持(0) 反对(0)

#21楼 [楼主] 2014-03-13 17:48 AnnieKim

@ peimin_outlook
编辑的时候有“插入代码”的按钮啊。

支持(0) 反对(0)

#22楼 2014-12-31 12:14 mm豆mm

"整个过程中每条边最多只走2次，一次是为了定位到某个节点，另一次是为了寻找上面某个节点的前驱节点"
每条边是不是应该最多走三遍？一次是定位，两次是找前驱节点（一次前驱节点right连接到当前节点，一次前驱节点right为空）？

支持(1) 反对(0)

#23楼 2015-01-23 22:38 laoyixia

博主问一下，用栈迭代的方法空间复杂度应该是O(logn)吧(树的高度)，不是O(n)?不知道对不对？谢谢。

支持(0) 反对(0)

#24楼 2015-02-21 00:51 CodingBug

@AnnieKim

Your inorderMorrisTraversal() is wrong.

支持(2) 反对(1)

#25楼 2015-04-05 03:24 -东方不败-

Morris 后序遍历，翻转链表有问题：
没有处理头结点的右孩子指针，结果会改变二叉链表的结构，影响其他操作。
应该加一句：
from->right = NULL;

支持(1) 反对(1)

#26楼 2015-04-28 02:53 Joyce-Lee

讲解好清晰，图画的很清楚，转载一下，多谢啦

支持(0) 反对(0)

#27楼 2016-02-26 09:44 ohuohuo

@AnnieKim @CodingBug

annie你的inorderMorrisTraversal的第2个判断错了，整个code应该是：

```
vector<int> inorderTraversal_3(TreeNode *root) {
vector<int> res;
TreeNode *cur = root;
while (cur)
{
if (cur->left)
{
TreeNode *prev = cur->left;
while (prev->right && prev->right != cur)
prev = prev->right;

if (prev->right == cur) //这里判断是不是current，而不是是不是null
{
res.push_back(cur->val);
cur = cur->right;
prev->right = NULL;
}
else
{
prev->right = cur;
cur = cur->left;
}
}
else
{
res.push_back(cur->val);
cur = cur->right;
}
}
return res;
}
```

支持(0) 反对(4)

#28楼 2017-04-23 21:36 emoheizi

@ AnnieKim
时间复杂度分析时，每个边虽然只访问2次但是每个边长平均值为logn吧。
这样也就是2*n*logn的复杂度

支持(0)　反对(1)

#29楼　2017-05-23 22:47 YuzheLi

@ ohuohuo
请不要误导他人。你只不过交换了这个if else的顺序而已，原作中的else语句即为你的if，你的else即为原作的if。仅此而已。留言前请验证程序的正确性，以及请对代码良好排版，以供他人阅读.

支持(3)　反对(0)

#30楼　2017-09-25 10:03 yanchao1073

@ AnnieKim
27楼说的是对的，后序遍历和前序遍历其实是左右对称的，只用把前序遍历的代码稍作修改就可以了。不需要使用栈，最后返回结果的时候，将vector逆序就可以了。这样还是O(1)的空间复杂度。

代码如下：

```
class Solution {
public:
vector<int> postorderTraversal(TreeNode* root) {
vector<int> nums;
TreeNode* cur = nullptr;

while (root) {
if (root->right) {
cur = root->right;
while (cur->left && cur->left != root) {
cur = cur->left;
}

if (cur->left == root) {
cur->left = nullptr;
root = root->left;
} else {
nums.push_back(root->val);
cur->left = root;
root = root->right;
}

} else {
nums.push_back(root->val);
root = root->left;
}
}
return vector<int>(nums.rbegin(), nums.rend());
}
}
```

具体分析可以看这个博客（blog.csdn.net/yc461515457/article/details/78082042）

支持(1)　反对(2)

#31楼　2018-01-11 06:00 吕大大的小世界

楼主你这个与一个问题不对。考虑一下这个树：[3,null, 1,2,null]，根节点3只有右子树没有右子树，根据分支：curr.left == null 进入的时候pre是3，然而下一次时你上来就把pre更新成了1的left也就是2，你直接把3跳过去了后面1，2的顺序发现不对但你已经丢失了3的信息了。正确的做法应该是用一个temp先进行右向traverse，不能先修改pre。

支持(0)　反对(0)

#32楼　2018-12-31 15:39 林学徒

emmm.....楼主,很感谢整理的这篇文章，让我了解了很多。但是在中序遍历那里的，感觉有个说法有点错误，就是每条边走两遍，我觉得应该是走三遍，一次是为了定位到某个节点，一次是为了寻找上面某个节点的前驱节点，并将其右孩子节点指向当前节点，一次是为了寻找上面某个节点的前驱节点，并将其前驱节点的右孩子节点指向null，即恢复二叉树的结构。虽然时间复杂度最后仍然是O(n)，但是会让整篇博文更严谨些。

支持(0)　反对(0)

#33楼　2019-04-15 13:57 wendabei

@ yanchao1073
27楼代码是对的，但是说的话是错的。。
您给的后序遍历思想是很不错的，但是想要逆序再输出，还是用到了额外O(n)的空间，与本文的理念不合。

支持(0)　反对(1)

#34楼　2019-04-15 14:08 wendabei

@ -东方不败-
先说结论，在整个 Morris 后续遍历代码中不需要处理这个指针的错误。
printReverse(cur->left, pre) 里面经过两次 reverse 之后，最终的确会导致 pre 的右指针有问题，指向了 pre 的父节点。但是执行完 printReverse 之后还会将 pre.right 置为 null，所以对于树的结构是没有影响的。
但是从函数式编程的思想来看，reverse 函数确实存在 from 右指针指向未处理的问题，单独使用会造成非预期后果，所以还是建议楼主加上对 from 右指针的处理。

支持(0)　反对(0)
