

个人公众号交流: bigsai

big

sai

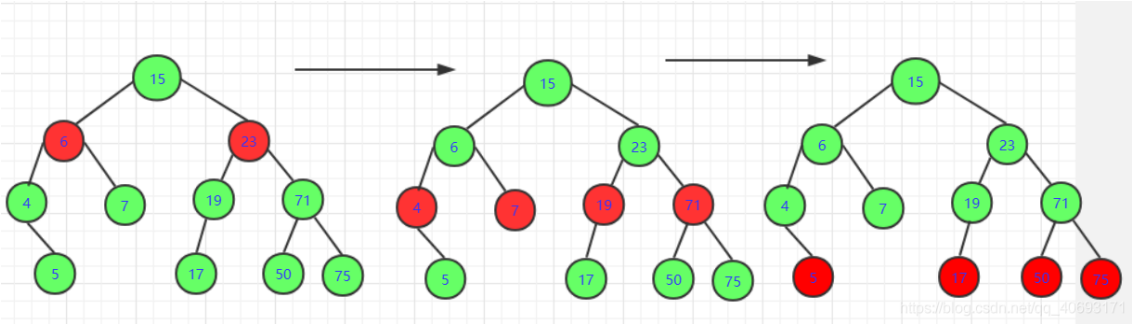
博客园 首页 新随笔 联系 订阅 管理

二叉树——前序遍历、中序遍历、后序遍历、层序遍历详解(递归非递归)

前言

- 前面介绍了二叉排序树的构造和基本方法的实现。但是排序遍历也是比较重要的一环。所以笔者将前中后序.和层序遍历梳理一遍。
- 了解树的遍历，需要具有的只是储备有队列，递归，和栈。这里笔者都有进行过详细介绍，可以关注笔者[数据结构与算法专栏](#)。持续分享，共同学习。

层序遍历



层序遍历。听名字也知道是按层遍历。我们知道一个节点有左右节点。而每一层一层的遍历都和左右节点有着很大的关系。也就是我们选用的数据结构不能一股脑的往一个方向钻，而左右应该均衡考虑。这样我们就选用队列来实现。

- 对于队列，现进先出。从根节点的节点push到队列，那么队列中先出来的顺序是第二层的左右(假设没有)。第二层每个执行的时候添加到队列，那么添加的所有节点都在第二层后面。
- 同理，假设开始pop遍历第n层的节点，每个节点会push左右两个节点进去。但是队列先进先出。它会放到队尾(下一层)。直到第n层的最后一个pop出来，第n+1层的还在队列中整齐排着。这就达到一个层序的效果。

实现的代码也很容易理解：

```
public void cengxu(node t) { //层序遍历
    Queue<node> q1 = new ArrayDeque<node>();
    if (t == null)
        return;
    if (t != null) {
        q1.add(t);
    }
    while (!q1.isEmpty()) {
        node t1 = q1.poll();
        if (t1.left != null)
            q1.add(t1.left);
        if (t1.right != null)
            q1.add(t1.right);
        System.out.print(t1.value + " ");
    }
    System.out.println();
}
```

前中后序遍历(递归)

其实这种就是一个类似dfs的思想。用递归实现。前面有很详细的介绍递归算法。我们采用的三序遍历是采用同一个递归。并且大家也都直到递归是一个有来有回的过程。三序遍历只是利用了递归中的来回过程中不同片段截取输出，而达到前(中、后序遍历的结果)。

前序递归

公告

微信公众号: bigsai
回复进群即可加入LeetCode打卡计划，多位伙伴加入。

微信搜索: bigsai



昵称: bigsai
园龄: 1年3个月
粉丝: 35
关注: 1
+加关注

aws

12个月免费套餐

新用户注册立享
11月限时专享 \$500 或 ¥2888 抵扣券

立即查看

2020年11月				
日	一	二	三	四
1	2	3	4	5
8	9	10	11	12
15	16	17	18	19
22	23	24	25	26
29	30	1	2	3
6	7	8	9	10

搜索

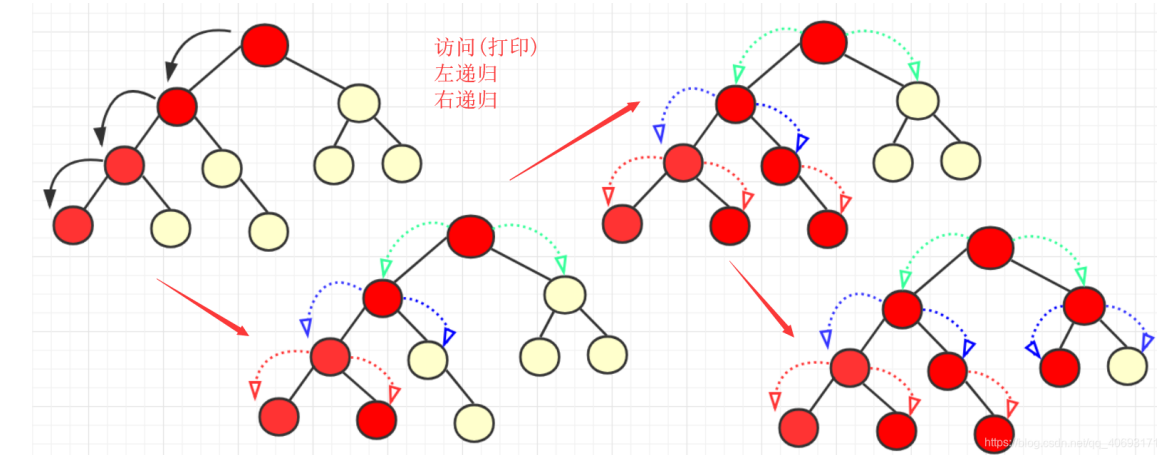
找找看

谷歌

常用链接

- 我的随笔
- 我的评论
- 我的参与

前序的规则就是根结点 ——> 左子树 ——> 右子树.我们在调用递归前进行节点操作。对于前序，就是先访问(输出)该节点。而递归左，递归右侧，会优先递归左侧。直到没有左节点。才会停止。访问次序大致为：



```
public void qianxu(node t)// 前序遍历 前序遍历：根结点 ——> 左子树 ——> 右子树
{
    if (t != null) {
        System.out.print(t.value + " ");// 当前节点
        qianxu(t.left);
        qianxu(t.right);
    }
}
```

中序递归

有了前序的经验，我们就很好利用递归实现中序遍历。中序遍历的规则是：左子树——> 根结点 ——> 右子树。所以我们访问节点的顺序需要变。

- 我们直到递归是来回的过程，对于恰好有两个子节点(子节点无节点)的节点来说。只需要访问一次左节点，访问根，访问右节点。即可。
- 而如果两侧有节点来说。每个节点都要满足中序遍历的规则。我们从根先访问左节点。到了左节点这儿左节点又变成一颗子树，也要满足中序遍历要求。所以就要先访问左节点的左节点(如果存在)。那么如果你这样想，规则虽然懂了。但是也太复杂了。那么我们借助递归。因为它的子问题和根节点的问题一致，只是范围减小了。所以我们使用递归思想来解决。
- 那么递归的逻辑为：考虑特殊情况(特殊就直接访问)不进行递归否则递归的访问左子树(让左子树执行相同函数，特殊就停止递归输出，不特殊就一直找下去直到最左侧节点。)——>输出该节点——>递归的访问右子树。

代码为：

```
public void zhongxu(node t)// 中序遍历 中序遍历：左子树——> 根结点 ——> 右子树
{
    if (t != null) {
        zhongxu(t.left);
        System.out.print(t.value + " ");// 访问完左节点访问当前节点
        zhongxu(t.right);
    }
}
```

后序递归

同理，有了前面的分析，后续就是左子树 ——> 右子树 ——> 根结点

```
public void houxu(node t)// 后序遍历 后序遍历：左子树 ——> 右子树 ——> 根结点
{
    if (t != null) {
        houxu(t.left);
        houxu(t.right);
        System.out.print(t.value + " "); // 访问完左右访问当前节点
    }
}
```

非递归前序

法一(技巧)

- 非递归的前序。我们利用栈的性质替代递归，因为递归有时候在效率方面不是令人满意的。利用栈，我们直到栈的顺序为后进先出。那么顺序如何添加？递归是左递归，右递归。但是利用栈要相反，因为如果左进栈、右进栈会出现以下后果：

最新评论
我的标签

随笔档案

- 2020年11月(3)
- 2020年10月(7)
- 2020年9月(2)
- 2020年8月(2)
- 2020年7月(4)
- 2020年6月(2)
- 2020年5月(2)
- 2020年4月(1)
- 2020年2月(1)
- 2020年1月(2)
- 2019年12月(1)
- 2019年10月(1)
- 2019年9月(4)
- 2019年8月(14)

最新评论

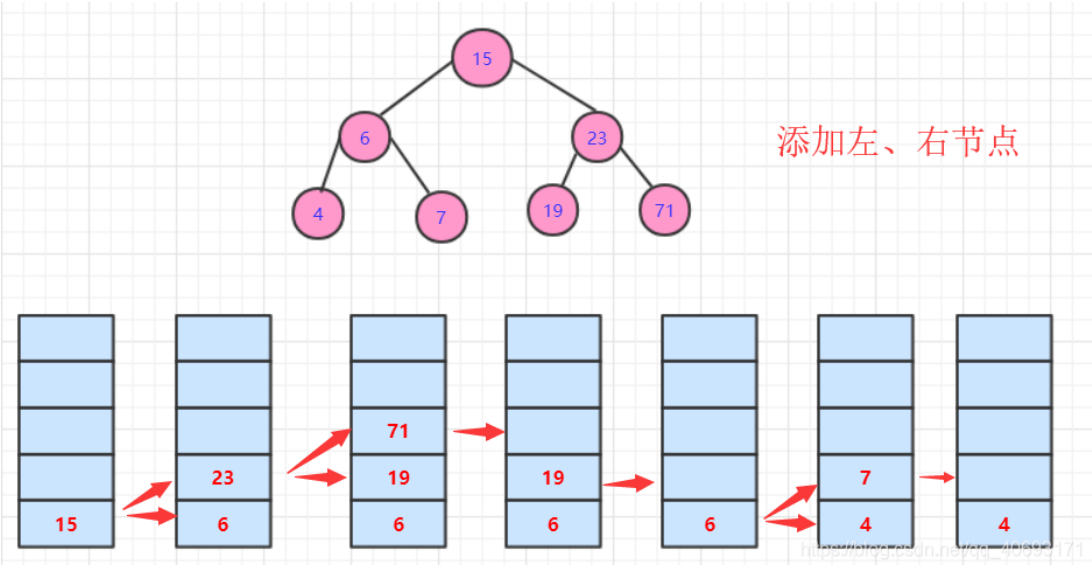
- 1. Re:「排序算法」图解双轴快排
@ToolGood 是的概率非常小。你说的个非常好优化思路。但快排最坏O(n2)重要的知识点，得提一下。...
- 2. Re:「排序算法」图解双轴快排
文章中" 如果运气肯不好遇到O(n)平方就很被啦："这句话，我不认同，因为因为通常快速排序法 最优化有三个地方是在【开始值、结尾值、1/2位置的值比较值...
- 3. Re:二叉树——前序遍历、中序遍历、层序遍历详解(递归非递归)
如果没有错别字就更好了。递归跑到最左又逐渐一层一层往上返。最后一层执行到倒数第二层去执行，倒数第二层执行数第三层。。。一直到最上层。最上层也执行完了。...
- 4. Re:JDBC+MySQL入门实战(实现C
@bigsai 好的，谢谢！ ... --51tes
- 5. Re:JDBC+MySQL入门实战(实现C
@51testing软件测试 哈哈可以哒，著号就行啦。如果公众号可以给您开白哈

阅读排行榜

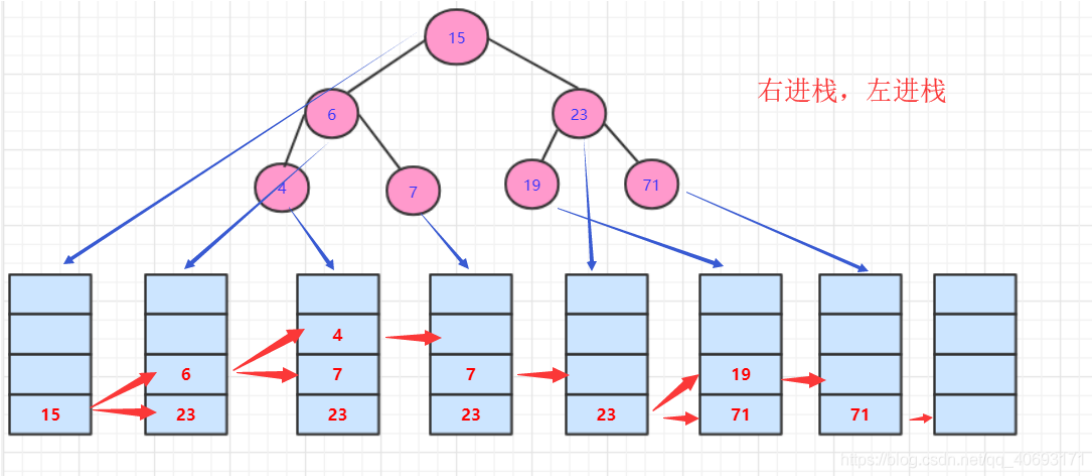
- 1. 拓扑排序详解与实现(58763)
- 2. 二叉树——前序遍历、中序遍历、层序遍历详解(递归非递归)(58691)
- 3. Dijkstra算法详细(单源最短路径算
- 4. 我用数据结构花了一夜给女朋友写了小游戏(5177)
- 5. 数据结构与算法—队列图文详解(36

评论排行榜

- 1. 我用数据结构花了一夜给女朋友写了小游戏(30)
- 2. 二叉树——前序遍历、中序遍历、层序遍历详解(递归非递归)(6)



所以，我们要利用递归的思路，需要先放右节点进栈，再放左节点进栈，这个下次再取节点取到左节点，这个节点再右节点进栈，左节点进栈。然后循环一直到最后会一直优先取到左节点。达到和递归顺序相仿效果。



每pop完添加右左节点直接输出(访问)即可完成前序非递归遍历。

```
public void qianxu3(node t)// 非递归前序 栈 先左后右 t一般为root
{
    Stack<node> q1 = new Stack<node>();
    if (t == null)
        return;
    if (t != null) {
        q1.push(t);
    }
    while (!q1.empty()) {
        node t1 = q1.pop();
        if (t1.right != null) {
            q1.push(t1.right);
        }
        if (t1.left != null) {
            q1.push(t1.left);
        }
        System.out.print(t1.value + " ");
    }
}
```

法二(传统)

方法二和非递归中序遍历的方法类似，只不过需要修改输出时间，在进栈时候输入访问节点即可。具体参考中序遍历分析。

```
public void qianxu2(node t) {
    Stack<node> q1 = new Stack();
    while(!q1.isEmpty()||t!=null)
    {
        if (t!=null) {
```

- 3. 数据结构与算法—递归(阶乘、斐波塔)(6)
- 4. 毕业季疫情下的校园生活是咋样的？
- 5. 哪吒票房逼近30亿，从豆瓣短评看哪吒的态度(4)

推荐排行榜

- 1. 我用数据结构花了一夜给女朋友写小游戏(21)
- 2. 拓扑排序详解与实现(14)
- 3. 二叉树——前序遍历、中序遍历、后序遍历详解(递归非递归)(10)
- 4. 毕业季疫情下的校园生活是咋样的？
- 5. SpringBoot+MongoDB实现物流

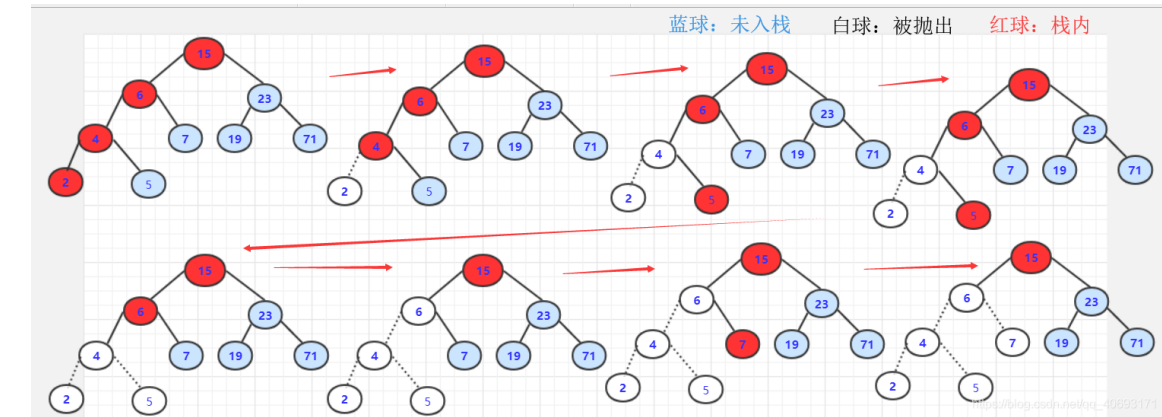
```
System.out.print(t.value+" ");
q1.push(t);
t=t.left;
}
else {
    t=q1.pop();
    t=t.right;
}
}
```

非递归中序

非递归中序和前序有所区别。
我们直到中序排列的顺序是：左节点，根节点，右节点。那么我们在经过根节点的前面节点 不能释放， 因为后面还需要用到它。所以要用栈先储存。
它的规则大致为：

- 栈依次存入左节点所有点，直到最左侧在栈顶。
- 开始抛出栈顶并访问。(例如第一个抛出2)。如果有右节点。那么将右节点加入栈中，然后右节点一致左下遍历直到尾部。（这里5和7没有左节点，所以不加）但是如果抛出15，右节点加入23,再找23的左侧节点加入栈顶。就这样循环下去直到栈为空。

可行性分析：中序是左一中一右的顺序。访问完左侧。当抛出当前点的时候说明左侧已经访问完(或者自己就是左侧)，那么需要首先访问当前点的右侧。那么这个右节点把它当成根节点重复相同操作（因为右节点要满足先左再右的顺序）。这样其实就是模拟了一个递归的过程，需要自己思考。



实现代码1：

```
public void zhongxu2(node t) {
    Stack<node> q1 = new Stack();
    while(!q1.isEmpty()||t!=null)
    {
        if (t!=null) {
            q1.push(t);
            t=t.left;
        }
        else {
            t=q1.pop();
            System.out.print(t.value+" ");
            t=t.right;
        }
    }
}
```

实现代码2: (个人首次写的)

```
public void zhongxu3(node t)// 先储藏所有左侧点，抛出一个点，访问该点右节点，对右节点在储存所有子左节点
{
    Stack<node> q1 = new Stack();
    if (t == null)
        return;
    if (t != null) {
        q1.push(t);
    }
    node t1 = q1.peek();// 不能抛出，要先存最左侧
    while (t1.left != null) {
        t1 = t1.left;
        q1.push(t1);
    }
```

```
while (!q1.isEmpty()) {
    node t2 = q1.pop();
    System.out.print(t2.value + " ");
    if (t2.right != null) {
        t2 = t2.right;
        q1.push(t2);
        while (t2.left != null) {
            t2 = t2.left;
            q1.push(t2);
        }
    }
}
```

非递归后序※

非递归后序遍历有两种方法
一种方法是利用和前面中序、前序第二种方法类似的方法进入压栈出栈，但是要借助额外的标记次数，一个节点访问第二次才能输出。(这个访问第一次是入栈，第二次是子树解决完毕自己即将出栈（先不出栈））。

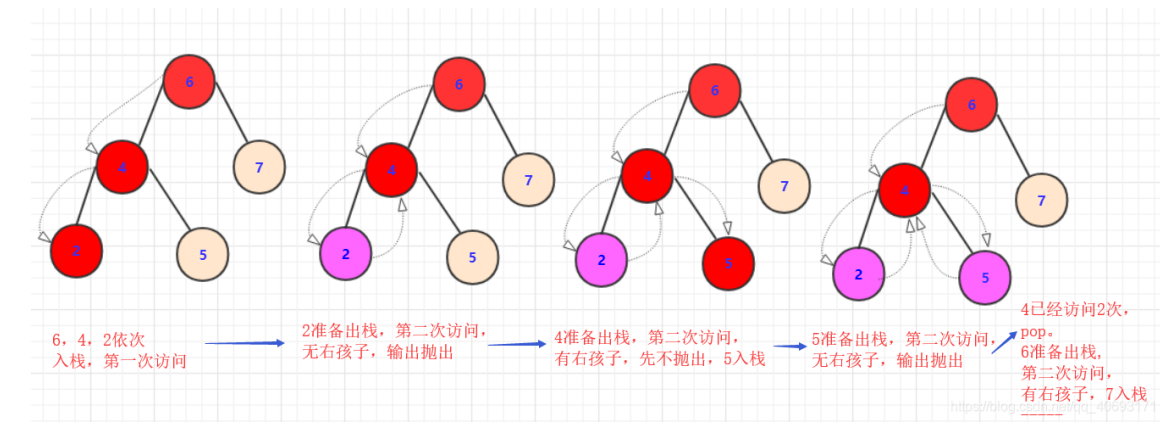
法1(传统方法)

在前面的前序和中序先到最左侧压入栈的时候，两种顺序依次是

- 前序：中入栈——>左入栈——>左出栈——>中出栈——>右入栈——>右孩子入出——>右出栈 在入栈时候操作即可前序
- 中序：中入栈——>左入栈——>左出栈——>中出栈——>右入栈 ——>右孩子入出——>右出栈按照出栈顺序即可完成中序

而在后序遍历中：它有这样的规则：

- 入栈，第一次访问
- 即将出栈。第二次访问，
- 如果有右孩子，先不出栈把右孩子压入栈第一次访问，如果没有右孩子。访问从栈中弹出。
- 循环重复，直到栈为空



实现代码为(用map记录节点出现次数):

```
public void houxu2(node t) {
    Stack<node> q1 = new Stack();
    Map<Integer,Integer> map=new HashMap<>();
    while(!q1.isEmpty() || t!=null)
    {
        if (t!=null) {
            q1.push(t);
            map.put(t.value, 1); //t.value标记这个值节点出现的次数
            t=t.left;
        }
        else {
            t=q1.peek();
            if(map.get(t.value)==2) { //第二次访问，抛出
                q1.pop();
                System.out.print(t.value+" ");
                t=null; //需要往上走
            }
            else {
                map.put(t.value, 2);
            }
        }
    }
}
```

```
        t=t.right;
    }

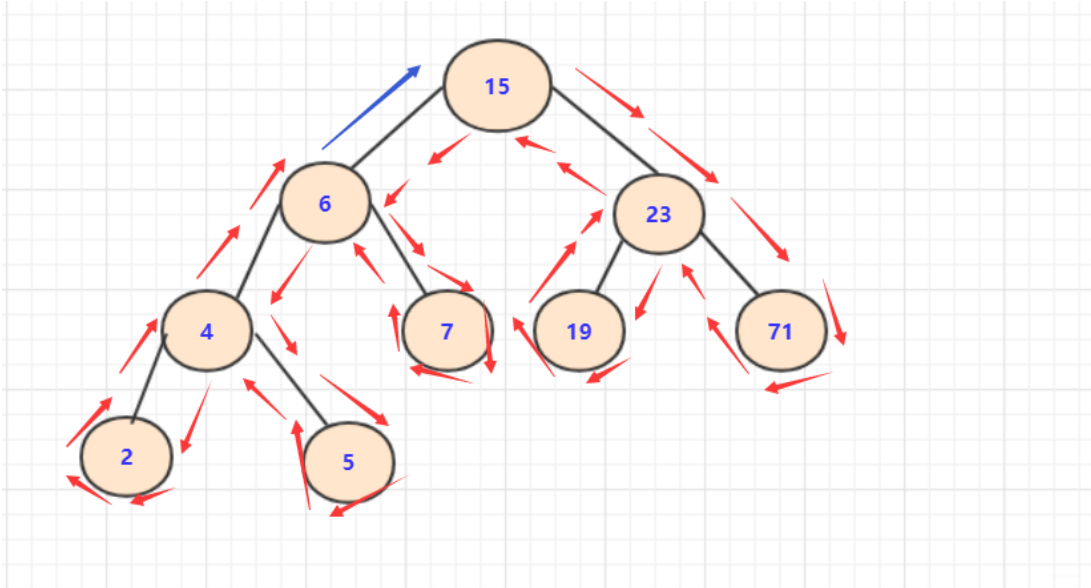
    }

}
```

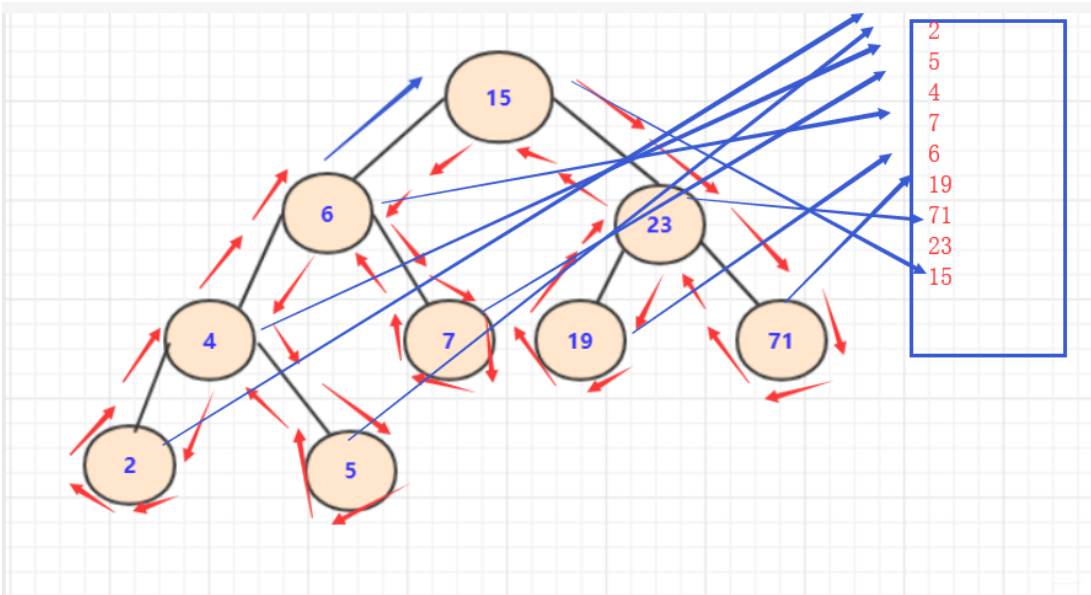
法2(双栈):

另一种方法是借助双栈进行处理。我们曾在前序方法一借助一个栈右压，左压。持续让达到一个前序遍历的效果。但是这个方法很难实现后续。

- 分析相同方法，如果我们先压左，再压右，那么我们获得的顺序将是和前序完全相反的顺序（顺序为：中间，右侧，左侧。倒过来刚好是左侧、右侧、中间的后续）对称看起来的前序。即用另一个栈将序列进行反转顺序！



如果再这个过程，我们利用另一个栈进行储存，将它的首次入栈用一个栈存入，相当于起到一个反转的作用。



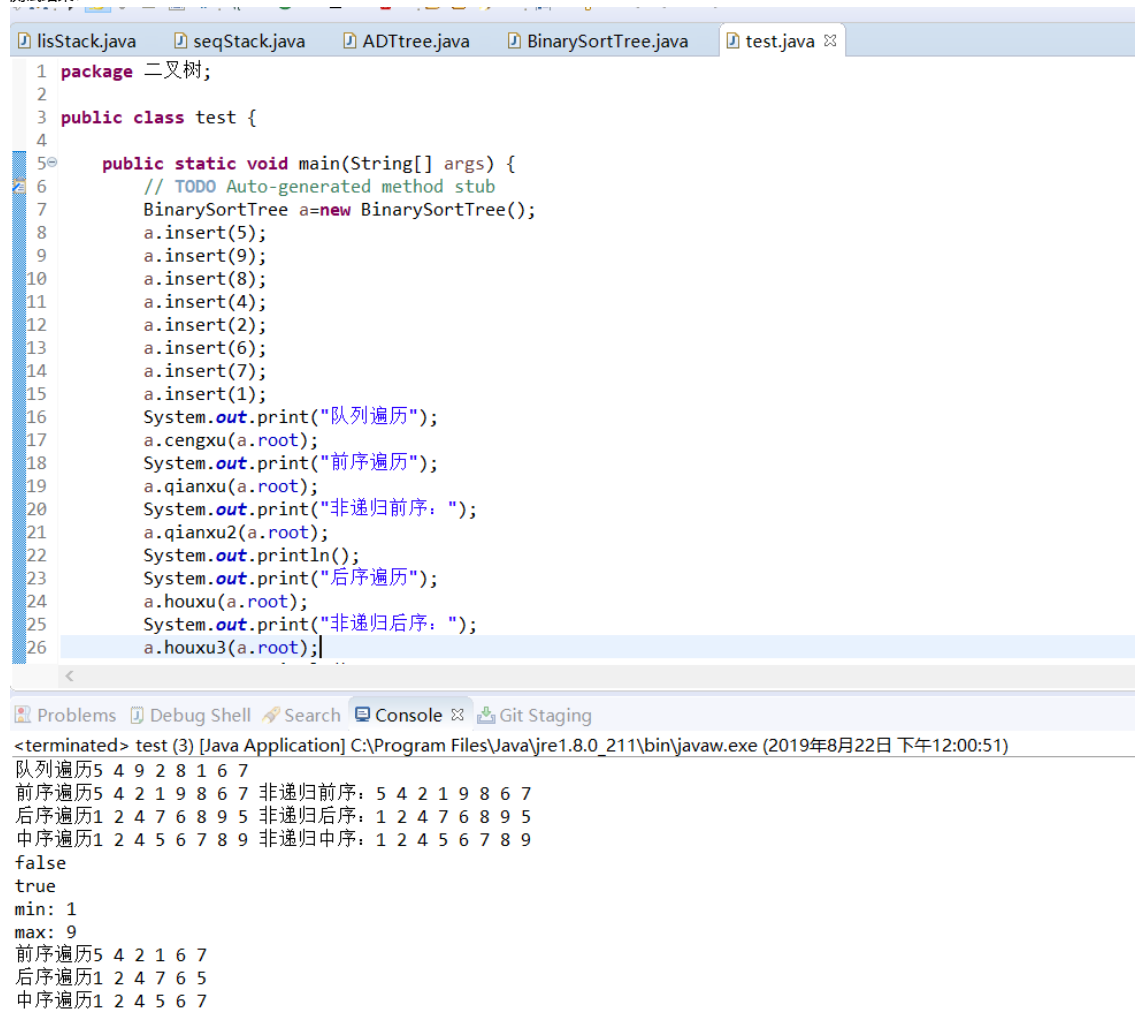
实现代码为:

```
public void houxu3(node t)// q1和q2 q1要先右后左，先遍历右侧，q1先装右侧就把右侧放到前面，左侧放在上面（栈顶）
{
    Stack<node> q1 = new Stack();
    Stack<node> q2 = new Stack();
    if (t == null)
        return;
    if (t != null) {
        q1.push(t);
    }
    while (!q1.isEmpty()) {
        node t1 = q1.pop();
        q2.push(t1);
    }
}
```

```
        if (t1.left != null) {
            q1.push(t1.left);
        }
        if (t1.right != null) {
            q1.push(t1.right);
        }
    }
    while (!q2.isEmpty()) {
        node t1 = q2.pop();
        System.out.print(t1.value + " ");
    }
}
```

总结

测试结果:



```
lisStack.java  seqStack.java  ADTtree.java  BinarySortTree.java  test.java x
1 package 二叉树;
2
3 public class test {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         BinarySortTree a=new BinarySortTree();
8         a.insert(5);
9         a.insert(9);
10        a.insert(8);
11        a.insert(4);
12        a.insert(2);
13        a.insert(6);
14        a.insert(7);
15        a.insert(1);
16        System.out.print("队列遍历");
17        a.cengxu(a.root);
18        System.out.print("前序遍历");
19        a.qianxu(a.root);
20        System.out.print("非递归前序: ");
21        a.qianxu2(a.root);
22        System.out.println();
23        System.out.print("后序遍历");
24        a.houxu(a.root);
25        System.out.print("非递归后序: ");
26        a.houxu3(a.root);
27    }
28 }
```

<terminated> test (3) [Java Application] C:\Program Files\Java\jre1.8.0_211\bin\javaw.exe (2019年8月22日 下午12:00:51)

队列遍历5 4 9 2 8 1 6 7
前序遍历5 4 2 1 9 8 6 7 非递归前序: 5 4 2 1 9 8 6 7
后序遍历1 2 4 7 6 8 9 5 非递归后序: 1 2 4 7 6 8 9 5
中序遍历1 2 4 5 6 7 8 9 非递归中序: 1 2 4 5 6 7 8 9
false
true
min: 1
max: 9
前序遍历5 4 2 1 6 7
后序遍历1 2 4 7 6 5
中序遍历1 2 4 5 6 7

这部分内容比较多，也可能比较杂，希望大家好好吸收，也可能笔者写的大意或者错误。还请大佬指正。!

- 另外，完整代码还请关注公众号(bigsai)。笔者认真更新数据结构与算法。有兴趣可以关注一波学一起学习。回复数据结构或者爬虫有精心准备学习资料赠送。



好文要顶

关注我

收藏该文

bigsai

关注 - 1

粉丝 - 35

+加关注

« 上一篇：[数据结构与算法—二叉排序树\(java\)](#)

» 下一篇：[AVL树\(二叉平衡树\)详解与实现](#)

104

posted @ 2019-08-22 12:05 bigsai 阅读(58709) 评论(6) 编辑 收藏

评论列表

- #1楼 2020-03-21 19:12 蟹老板爱吃炒蟹

写得很好啊，为什么有两个人投反对呢

支持(0) 反对(1)
- #2楼 2020-04-30 11:55 MadHuang

写的很详细，受教受教！法一前序里有处错误，栈应该是先进后出才是

支持(0) 反对(0)
- #3楼 [楼主] 2020-04-30 14:17 bigsai

@MadHuang
已修改啦！

支持(0) 反对(0)
- #4楼 2020-08-03 10:55 Apple2012

• 支持

支持(0) 反对(0)
- #5楼 2020-08-03 10:55 Apple2012

感谢博主

支持(0) 反对(0)
- #6楼 2020-08-23 12:16 super超人

如果没有错别字就更好了。
递归跑到最底层，然后又逐渐一层一层往上返。最后一层执行完，然后又跑到倒数第二层去执行，倒数第二层执行完，又跑到倒数第三层。。。一直到最上层。最上层执行完，递归也执行完了。

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

登录后才能发表评论，立即 [登录](#) 或 [注册](#)， [访问 网站首页](#)

博客园派送云上免费午餐，AWS注册立享12个月免费套餐

- 【推荐】News：大型组态、工控、仿真、CADGIS 50万行VC++源码免费下载
- 【推荐】博客园 & 陌上花开HIMMR 给单身的程序员小哥哥助力脱单啦~
- 【推荐】了不起的开发者，挡不住的华为，园子里的品牌专区
- 【推荐】未知数的距离，毫秒间的传递，声网与你实时互动

【福利】AWS携手博客园为开发者送免费套餐与抵扣券

【推荐】阿里云折扣价格返场，错过再等一年

相关博文：

- SQL与，或，非
- javascript逻辑非 (! /! !)
- Emmet (非重要)
- 墨问非名
- java非空判断
- » 更多推荐...

专为场景设计的API
让你最快速码出实时互动

点击进入
声网专区 »

最新 IT 新闻：

- 联想中国区总裁刘军：今年服务业务目标10亿美元
- 蛋壳公寓深圳租金兑付困难持续 员工称未准时发工资
- 台积电明年基础薪资上调20%
- 内含氮化镓 GaN 晶体管 / 系统集成电路终端产品已批量生产
- 5G SA网络到底对iPhone 12有什么影响?
- » 更多新闻...

Copyright © 2020 bigsai
Powered by .NET 5.0.0 on Kubernetes

个人公众号交流: bigsai