

一个输在起跑线的男人

昵称: Du~佛系码农
园龄: 2年
粉丝: 11
关注: 11
[+加关注](#)

< 2020年12月 >						
日	一	二	三	四	五	六
29	30	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	1	2
3	4	5	6	7	8	9

搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签
[更多链接](#)

随笔分类

HomeWork(14)
Java(26)
LeetCode(44)
Linux(1)
MySQL(2)
Sort(15)
SpringBoot(1)
spring框架(1)
场景题(1)

博客园 首页 新随笔 联系 管理 订阅 XML

随笔- 159 文章- 0 评论- 11

二叉树的四种遍历方式

二叉树的四种遍历方式：

- 二叉树的遍历 (traversing binary tree) 是指从根结点出发，按照某种次序依次访问二叉树中所有的结点，使得每个结点被访问依次且仅被访问一次。
四种遍历方式分别为：先序遍历、中序遍历、后序遍历、层序遍历。



遍历之前，我们首先介绍一下，如何创建一个二叉树，在这里用的是先建左树在建右树的方法，

首先要声明结点TreeNode类，代码如下：

```
public class TreeNode {  
    public int data;  
    public TreeNode leftChild;  
    public TreeNode rightChild;  
  
    public TreeNode(int data){  
        this.data = data;  
    }  
}
```

再来创建一颗二叉树：

动态规划(19)
链表(21)
树(6)
数据结构(3)
项目遇到问题(2)
游戏(2)

随笔档案

2020年8月(1)
2019年12月(1)
2019年11月(2)
2019年8月(8)
2019年7月(18)
2019年6月(8)
2019年5月(16)
2019年4月(22)
2019年3月(35)
2019年2月(17)
2019年1月(12)
2018年12月(10)
2018年11月(9)

最新评论

- 1. Re:判断二叉树是否是镜像对称
q.add(root); q.add(root);搞错了，楼主 应该是 q.add(root.left); q.add(root.right);...
--linghu_java
- 2. Re:二叉树的四种遍历方式
很详细，一看就懂
--一本正经的正襟危坐
- 3. Re:非递归实现快速排序
先从栈中取出的应该赋值给high吧，这里有点不明白
--化鱼
- 4. Re:二叉树的四种遍历方式
没影响阅读啊，😊觉得很ok
--蜡笔小新弟弟蜡笔无新
- 5. Re:二叉树的四种遍历方式
@欧yu 不算乱啊，写得挺清楚...
--mattzhang27

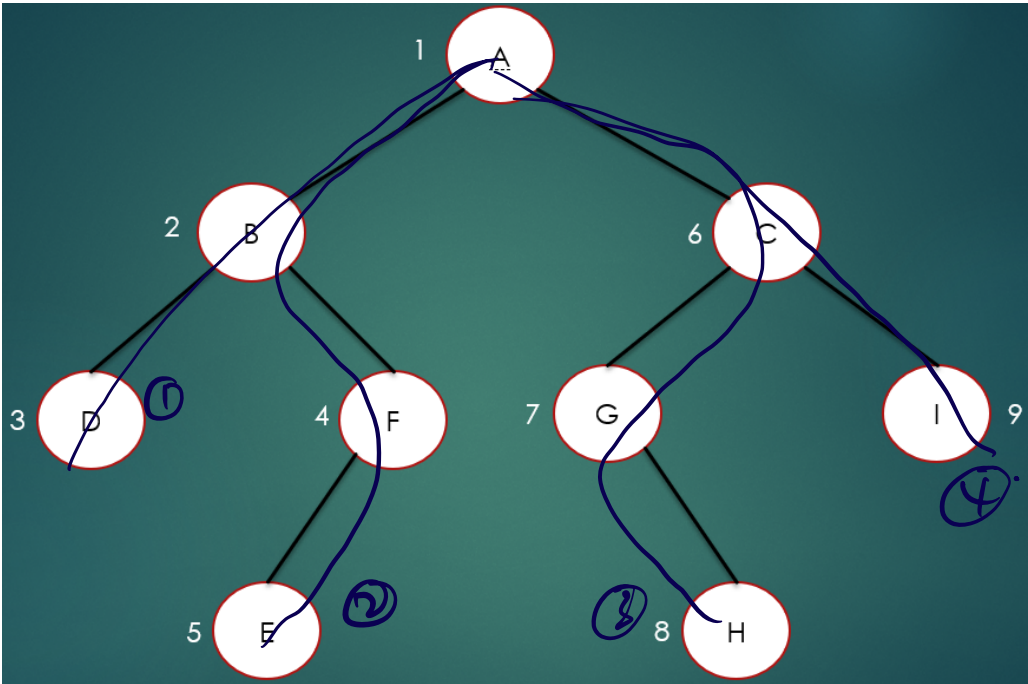
阅读排行榜

- 1. 二叉树的四种遍历方式(52528)
- 2. 如何将Map对象转换为一个实体类对象(8719)
- 3. 排序链表(4400)
- 4. 假设你正在爬楼梯。需要 n 阶你才能到达楼顶。 每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢？ (2739)
- 5. 快乐数（编写一个算法来判断一个数是不是“快乐数”。一个“快乐数”定义为：对于一个正整数，每一次将该数替换为它每个位置上的数字的平方和，然后重复这个过程直到这个数变为 1，也可能是无限循环但始终变不到 1。如果可以变为 1，那么这个数就是快乐数。） (2671)

评论排行榜

```
/**
 * 构建二叉树
 * @param list 输入序列
 * @return
 */
public static TreeNode createBinaryTree(LinkedList<Integer> list){
    TreeNode node = null;
    if(list == null || list.isEmpty()){
        return null;
    }
    Integer data = list.removeFirst();
    if(data!=null){
        node = new TreeNode(data);
        node.leftChild = createBinaryTree(list);
        node.rightChild = createBinaryTree(list);
    }
    return node;
}
```

接下来按照上面列的顺序——讲解，
首先来看前序遍历，所谓的前序遍历就是先访问根节点，再访问左节点，最后访问右节点，



如上图所示，前序遍历结果为：ABDFECGHI

实现代码如下：

```
/**
 * 二叉树前序遍历 根->左->右
 * @param node 二叉树节点
 */
public static void preOrderTraverl(TreeNode node){
    if(node == null){
        return;
    }
    System.out.print(node.data+" ");
    preOrderTraverl(node.leftChild);
    preOrderTraverl(node.rightChild);
}
```

out
L
R.

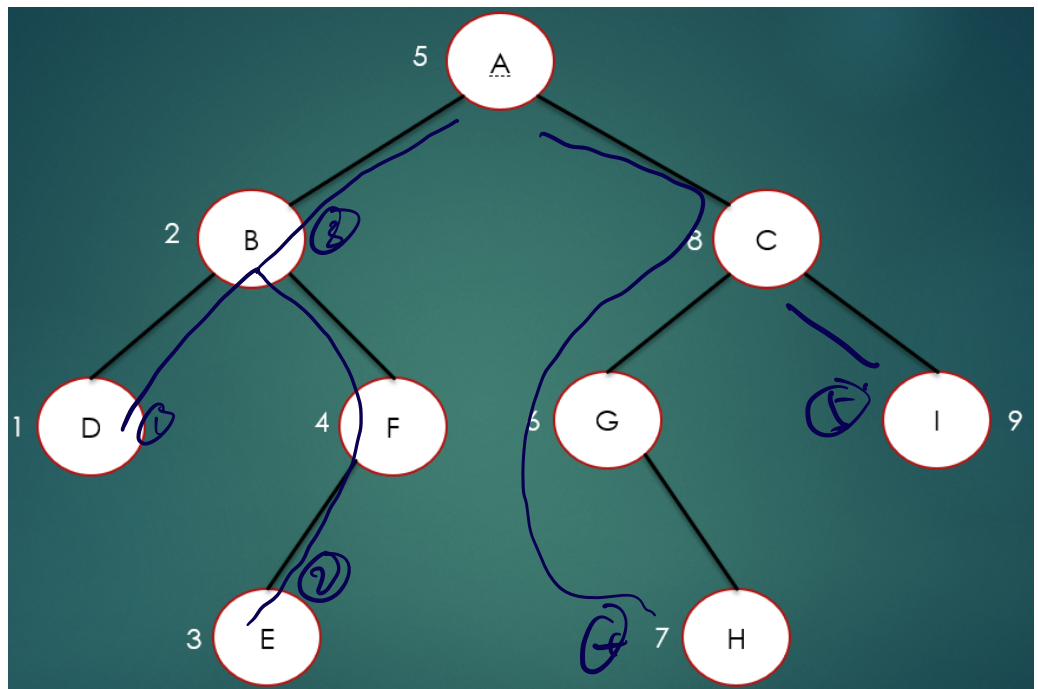
1. 二叉树的四种遍历方式(7)
2. 给定一个字符串，找到它的第一个不重复的字符，并返回它的索引。如果不存在，则返回 -1。(2)
3. 判断二叉树是否是镜像对称(1)
4. 非递归实现快速排序(1)

推荐排行榜

1. 二叉树的四种遍历方式(5)
2. 利用前序遍历和中序遍历构造二叉树(2)
3. 为什么重写equals还要重写hashCode呢? (1)
4. 如何在百万级数据中找到第K大的数据(1)
5. 三角形最小路径和(1)



再者就是中序遍历，所谓的中序遍历就是先访问左节点，再访问根节点，最后访问右节点，



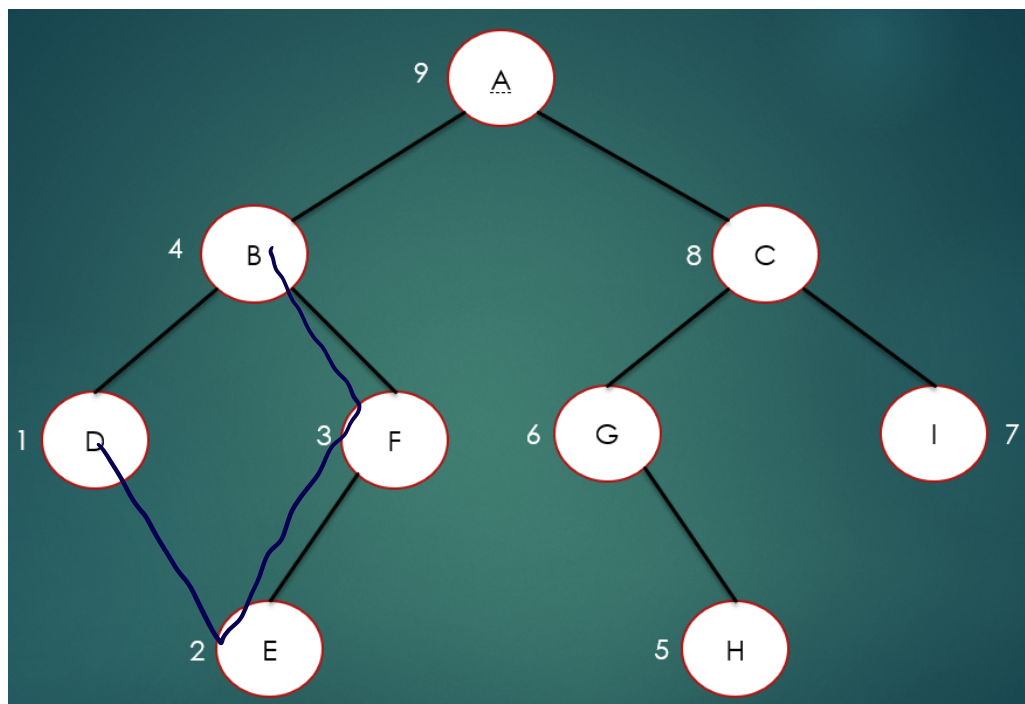
如上图所示，中序遍历结果为：DBEFAGHCI

实现代码如下：

```
/**
 * 二叉树中序遍历 左-> 根-> 右
 * @param node 二叉树节点
 */
public static void inOrderTravalal(TreeNode node){
    if(node == null){
        return;
    }
    inOrderTravalal(node.leftChild);
    System.out.print(node.data+" ");
    inOrderTravalal(node.rightChild);
}
```

先到达最左，然后往上一级。
 如果没有左节点但有右节点时向右。

最后就是后序遍历，所谓的后序遍历就是先访问左节点，再访问右节点，最后访问根节点。



如上图所示，后序遍历结果为：DEFBHGICA

实现代码如下：

```

/**
 * 二叉树后序遍历 左-> 右-> 根
 * @param node 二叉树节点
 */
public static void postOrderTravalal(TreeNode node){
    if(node == null){
        return;
    }
    postOrderTravalal(node.leftChild);
    postOrderTravalal(node.rightChild);
    System.out.print(node.data+" ");
}

```

2
R
one.

讲完上面三种递归的方法，下面再给大家讲讲非递归是如何实现前中后序遍历的
还是一样，先看非递归前序遍历

1. 首先申请一个新的栈，记为stack；
2. 声明一个结点treeNode，让其指向node结点；
3. 如果treeNode的不为空，将treeNode的值打印，并将treeNode入栈，然后让treeNode指向treeNode的右结点，
4. 重复步骤3，直到treenode为空；
5. 然后出栈，让treeNode指向treeNode的右孩子
6. 重复步骤3，直到stack为空.

实现代码如下：

```
public static void preOrderTravalWithStack(TreeNode node){
    Stack<TreeNode> stack = new Stack<TreeNode>();
    TreeNode treeNode = node;
    while(treeNode!=null || !stack.isEmpty()){
        //迭代访问节点的左孩子，并入栈
        while(treeNode != null){
            System.out.print(treeNode.data+" ");
            stack.push(treeNode);
            treeNode = treeNode.leftChild;
        }
        //如果节点没有左孩子，则弹出栈顶节点，访问节点右孩子
        if(!stack.isEmpty()){
            treeNode = stack.pop();
            treeNode = treeNode.rightChild;
        }
    }
}
```

中序遍历非递归，在此不过多叙述具体步骤了，

具体过程：

1. 申请一个新栈，记为stack，申请一个变量cur，初始时令treeNode为头节点；
2. 先把treeNode节点压入栈中，对以treeNode节点为头的整棵子树来说，依次把整棵树的左子树压入栈中，即不断令treeNode=treeNode.leftChild，然后重复步骤2；
3. 不断重复步骤2，直到发现cur为空，此时从stack中弹出一个节点记为treeNode，打印node的值，并让treeNode= treeNode.right，然后继续重复步骤2；
4. 当stack为空并且cur为空时结束。

```
public static void inOrderTravalWithStack(TreeNode node){
    Stack<TreeNode> stack = new Stack<TreeNode>();
    TreeNode treeNode = node;
    while(treeNode!=null || !stack.isEmpty()){
        while(treeNode != null){
            stack.push(treeNode);
            treeNode = treeNode.leftChild;
        }
        if(!stack.isEmpty()){
            treeNode = stack.pop();
            System.out.print(treeNode.data+" ");
            treeNode = treeNode.rightChild;
        }
    }
}
```

后序遍历非递归实现，后序遍历这里较前两者实现复杂一点，我们需要一个标记位来记忆我们此时节点上一个节点，具体看代码注释

```

public static void postOrderTravalWithStack(TreeNode node){
    Stack<TreeNode> stack = new Stack<TreeNode>();
    TreeNode treeNode = node;
    TreeNode lastVisit = null;    //标记每次遍历最后一次访问的节点
    while(treeNode!=null || !stack.isEmpty()){//节点不为空，结点入栈，并且指向下一个左孩子
        while(treeNode!=null){
            stack.push(treeNode);
            treeNode = treeNode.leftChild;
        }
        //栈不为空
        if(!stack.isEmpty()){
            //出栈
            treeNode = stack.pop();
            /**
             * 这块就是判断treeNode是否有右孩子，
             * 如果没有输出treeNode.data，让lastVisit指向treeNode，并让treeNode为空
             * 如果有右孩子，将当前节点继续入栈，treeNode指向它的右孩子，继续重复循环
             */
            if(treeNode.rightChild == null || treeNode.rightChild == lastVisit) {
                System.out.print(treeNode.data + " ");
                lastVisit = treeNode;
                treeNode = null;
            }else{
                stack.push(treeNode);
                treeNode = treeNode.rightChild;
            }
        }
    }
}

```

最后再给大家介绍一下层序遍历

具体步骤如下：

1. 首先申请一个新的队列，记为queue；
2. 将头结点head压入queue中；
3. 每次从queue中出队，记为node，然后打印node值，如果node左孩子不为空，则将左孩子入队；如果node的右孩子不为空，则将右孩子入队；
4. 重复步骤3，直到queue为空。

实现代码如下：

```

public static void levelOrder(TreeNode root){
    LinkedList<TreeNode> queue = new LinkedList<>();
    queue.add(root);
    while(!queue.isEmpty()){
        root = queue.pop();
        System.out.print(root.data+" ");
        if(root.leftChild!=null) queue.add(root.leftChild);
        if(root.rightChild!=null) queue.add(root.rightChild);
    }
}

```