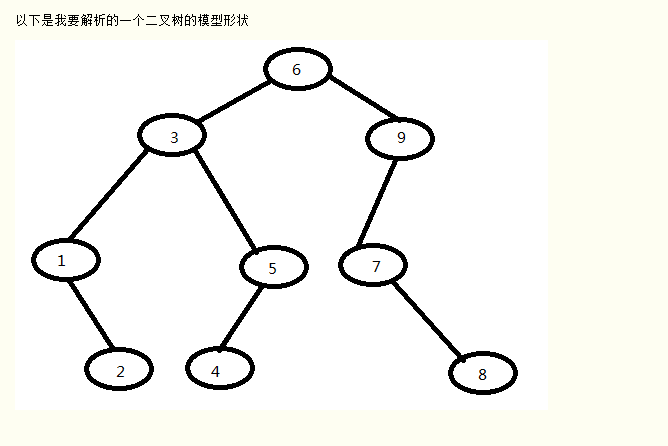
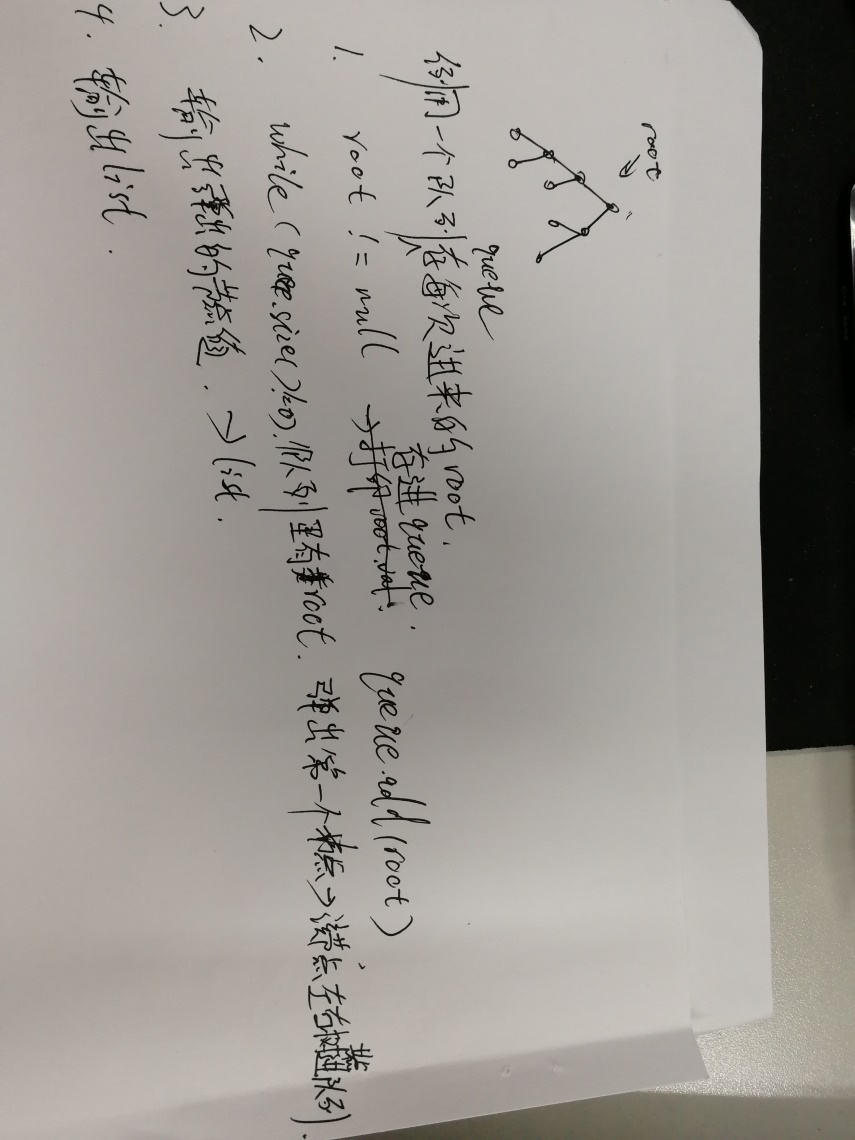
从上往下打印出二叉树的每个节点，同层节点从左至右（从右到左）打印。



思路：利用一个队列存储每个节点，并且每次循环时，弹出第一个节点，并存进弹出节点的左右树节点到树最后。



////////////////////////////////////////从左到右//////////////////////////////////////

**package** facehandjava.tree;  
  
**import** java.util.LinkedList;  
**import** java.util.Queue;  
**import** java.util.Stack;  
**import** java.util.concurrent.ConcurrentLinkedQueue;  
**import** java.util.concurrent.LinkedBlockingDeque;  
  
**public class** L2RRecursiveBinaryTree {  
 */\*\*  
 \* 二叉树的先序中序后序排序（非递归版本）  
 \*/* **public** Node init() {*//注意必须逆序建立，先建立子节点，再逆序往上建立，因为非叶子结点会使用到下面的节点，而初始化是按顺序初始化的，不逆序建立会报错* Node J = **new** Node(8, **null**, **null**);  
 Node H = **new** Node(4, **null**, **null**);  
 Node G = **new** Node(2, **null**, **null**);  
 Node F = **new** Node(7, **null**, J);  
 Node E = **new** Node(5, H, **null**);  
 Node D = **new** Node(1, **null**, G);  
 Node C = **new** Node(9, F, **null**);  
 Node B = **new** Node(3, D, E);  
 Node A = **new** Node(6, B, C);  
 **return** A; *//返回根节点* }  
  
 **public static void** main(String[] args) {  
 L2RRecursiveBinaryTree tree = **new** L2RRecursiveBinaryTree();  
 Node root = tree.init();  
 System.***out***.println(**"从左到右的二叉树遍历"**);  
 tree.L2RRecursiveBinaryTree(root);  
 }  
  
  
 **private void** L2RRecursiveBinaryTree(Node root) {  
 Queue<Node> queue = **new** LinkedList<>();  
 **int** count = 0;  
 **if** (root != **null**) {  
 queue.add(root);  
 }  
 **while** (!queue.isEmpty()) {  
 count = queue.size();  
 **while** (count > 0) {  
 Node temp = queue.remove();  
 System.***out***.print(temp.getData());  
 count--;  
 **if** (temp.getLeftNode() != **null**) {  
 queue.add(temp.getLeftNode());  
 }  
 **if** (temp.getRightNode() != **null**) {  
 queue.add(temp.getRightNode());  
 }  
 }  
 }  
 }  
}

////////////////////////////////////////从右到左//////////////////////////////////////

**package** facehandjava.tree;  
  
**import** java.util.LinkedList;  
**import** java.util.Queue;  
  
**public class** R2LRecursiveBinaryTree {  
 */\*\*  
 \* 从右到左  
 \*/* **public** Node init() {*//注意必须逆序建立，先建立子节点，再逆序往上建立，因为非叶子结点会使用到下面的节点，而初始化是按顺序初始化的，不逆序建立会报错* Node J = **new** Node(8, **null**, **null**);  
 Node H = **new** Node(4, **null**, **null**);  
 Node G = **new** Node(2, **null**, **null**);  
 Node F = **new** Node(7, **null**, J);  
 Node E = **new** Node(5, H, **null**);  
 Node D = **new** Node(1, **null**, G);  
 Node C = **new** Node(9, F, **null**);  
 Node B = **new** Node(3, D, E);  
 Node A = **new** Node(6, B, C);  
 **return** A; *//返回根节点* }  
  
  
  
 **public static void** R2LRecursiveBinaryTree(Node root) {  
 Queue<Node> queue = **new** LinkedList<>();  
 Node temp = **null**;  
 queue.add(root);  
 **while** (!queue.isEmpty()) {  
 **int** i = queue.size();  
 **while** (i>0) {  
 temp = queue.remove();  
 i--;  
 System.***out***.print(temp.getData());  
 **if** (temp.getRightNode() != **null**) {  
 queue.add(temp.getRightNode());  
 }  
 **if** (temp.getLeftNode() != **null**) {  
 queue.add(temp.getLeftNode());  
 }  
 }  
 }  
  
 }  
  
  
 **public static void** main(String[] args) {  
 R2LRecursiveBinaryTree tree = **new** R2LRecursiveBinaryTree();  
 Node root = tree.init();  
 System.***out***.println(**"从右到左的二叉树遍历"**);  
 *R2LRecursiveBinaryTree*(root);  
 }  
}