function\* preOrder(root){ // 前序遍历 if(root){ yield root.mid; yield\* preOrder(root.left); yield\* preOrder(root.right); } } function\* inOrder(root){ // 中序遍历 if(root){ yield\* inOrder(root.left); yield root.mid; yield\* inOrder(root.right); } } function\* postOrder(root){ // 后序遍历 if(root){ yield\* postOrder(root.left); yield\* postOrder(root.right); yield root.mid; } } function Node(left, mid, right){ // 二叉树构造函数 this.left = left; this.mid = mid; this.right = right; } function binaryTree(arr){ // 生成二叉树 if(arr.length == 1){ return new Node(null, arr[0], null); } return new Node(binaryTree(arr[0]), arr[1], binaryTree(arr[2])); } // 完全二叉树节点 let bTree = binaryTree([[['d'], 'b', ['e']], 'a', [['f'], 'c', ['g']]]); // 遍历结果 var preResult = []; for(let node of preOrder(bTree)){ // 前序遍历结果 preResult.push(node); } console.log(preResult); // (7) ["a", "b", "d", "e", "c", "f", "g"] var inResult = []; for(let node of inOrder(bTree)){ // 中序遍历结果 inResult.push(node); } console.log(inResult); // (7) ["d", "b", "e", "a", "f", "c", "g"] var postResult = []; for(let node of postOrder(bTree)){ // 后序遍历结果 postResult.push(node); } console.log(postResult);