## 116. Populating Next Right Pointers in Each Node

1.典型bfs

```
class Solution {
    public Node connect(Node root) {
        if(root == null){
            return null;
        }
        Queue<Node> queue = new LinkedList<>();
        queue.add(root);
        while(!queue.isEmpty()){
            int size = queue.size();
            Node node = null;
            for(int i = 0; i < size; i++){
                if(i == 0){
                    node = queue.poll();
                }
                if(node.left != null){
                    queue.add(node.left);
                }
                if(node.right != null){
                    queue.add(node.right);
                }

                if(i + 1 < size){
                    Node temp = queue.poll();
                    node.next = temp;
                    node = temp;
                }
                else{
                    node.next = null;
                }
            }
        }
        return root;
    }
}
```

## 144. Binary Tree Preorder Traversal

Hint:

1. 前序遍历，dfs
2. 用stack

```
class Solution {
    public List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> list = new ArrayList<>();
        if(root == null){
            return list;
        }
        Stack<TreeNode> stack = new Stack<>();
        stack.push(root);
        while(!stack.isEmpty()){
            TreeNode node = stack.pop();
            list.add(node.val);

            if(node.right != null){
                stack.push(node.right);
            }
            if(node.left != null){
                stack.push(node.left);
            }
        }
        return list;
    }
}
```

## 105. Construct Binary Tree from Preorder and Inorder Traversal

题目：根据前序遍历和中序遍历构建二叉树

Hint：

1.DFS

2.右子节点的preorder位置pstart + index - istart + 1

```
class Solution {
    public TreeNode buildTree(int[] preorder, int[] inorder) {
        return constructTree(preorder, inorder, 0, 0, inorder.length - 1);
    }

    private TreeNode constructTree(int[] preorder, int[] inorder, int pstart, int istart, int iend){
        if(iend < istart || pstart >= preorder.length){
            return null;
        }
        TreeNode root = new TreeNode(preorder[pstart]);
        int index = 0;
        for(int i = istart; i <= iend; i++){
            if(inorder[i] == preorder[pstart]){
                index = i;
                break;
            }
        }
        TreeNode left = constructTree(preorder, inorder, pstart + 1, istart, index - 1 );
        TreeNode right = constructTree(preorder, inorder, pstart + index - istart + 1, index + 1, iend);
        root.left = left;
        root.right = right;
        return root;
    }
}
```

## 24. Swap Nodes in Pairs

**题目：linkedlist【1，2，3，4】变成【2，1，4，3】**

**Hint：**

**1.添加新的节点指向head，用来记录first的前一个节点**

**2.向后循环条件为head = head.next而不是head.next.next;**

```
class Solution {
    ListNode one = null;
    public ListNode swapPairs(ListNode head) {
        ListNode newhead = new ListNode(0);
        newhead.next = head;
        ListNode cur = newhead;
        while(head != null && head.next != null){
            ListNode first = head;
            ListNode second = head.next;
            cur.next = second;
            first.next = second.next;
            second.next = first;
            cur = first;
            head = head.next;
        }
        return newhead.next;
    }
}
```

## 92. Reverse Linked List II

题目：对特定linked list，给出起始位置，结束位置，交换起始和结束位置中间的linkedlist

```
class Solution {
    public ListNode reverseBetween(ListNode head, int m, int n) {
        if(head == null){
            return null;
        }
        ListNode newhead = new ListNode(0);
        newhead.next = head;
        ListNode first = newhead;
        ListNode second = null;
        ListNode third = null;
        ListNode forth = null;
        int index = 0;
        while(head != null){
            index++;
            if(index == m - 1){
                first = head;
            }
            if(index == m){
                second = head;
            }
            if(index == n){
                third = head;
            }
            if(index == n + 1){
                forth = head;
            }
            head = head.next;
        }
        reverse(first, second, third, forth);
        return newhead.next;
    }
    private void reverse(ListNode first, ListNode second, ListNode third, ListNode forth){
        ListNode head = second;
        ListNode cur = null;
        ListNode last = forth;
        while(head.next != forth){
            cur = head.next;
            head.next = last;
            last = head;
            head = cur;
        }
        head.next = last;
        first.next = head;
        second.next = forth;
    }
}
```