Leetcode 题集

1. 寻找两个正序数组的中位数
   要求在 O(log(m+n))

```java
class Solution {
    public double findMedianSortedArrays(int[] A, int[] B) {
        int m = A.length;
        int n = B.length;
        if (m > n) {
            return findMedianSortedArrays(B,A); // 保证 m <= n
        }
        int iMin = 0, iMax = m;
        while (iMin <= iMax) {
            int i = (iMin + iMax) / 2;
            int j = (m + n + 1) / 2 - i;
            if (j != 0 && i != m && B[j-1] > A[i]){ // i 需要增大
                iMin = i + 1;
            }
            else if (i != 0 && j != n && A[i-1] > B[j]) { // i 需要减小
                iMax = i - 1;
            }
            else { // 达到要求，并且将边界条件列出来单独考虑
                int maxLeft = 0;
                if (i == 0) { maxLeft = B[j-1]; }
                else if (j == 0) { maxLeft = A[i-1]; }
                else { maxLeft = Math.max(A[i-1], B[j-1]); }
                if ( (m + n) % 2 == 1 ) { return maxLeft; } // 奇数的话不需要考

                int minRight = 0;
                if (i == m) { minRight = B[j]; }
                else if (j == n) { minRight = A[i]; }
                else { minRight = Math.min(B[j], A[i]); }

                return (maxLeft + minRight) / 2.0; //如果是偶数的话返回结果
            }
        }
        return 0.0;
    }
}
```

2. 无重复数组的全排列

```java
class Solution {
    public List<List<Integer>> permute(int[] nums) {
        List<List<Integer>> ans = new ArrayList<List<Integer>>();
        if (nums == null || nums.length == 0) return ans;
        backtrack(ans, new ArrayList<Integer>(), nums);
        return ans;
    }
    public void backtrack(List<List<Integer>> ans, ArrayList<Integer> temp, int[] nums){
        if (temp.size() == nums.length){
            ans.add(new ArrayList<Integer>(temp));
            return;
        }
        for (int i = 0; i < nums.length; i ++){
            if (temp.contains(nums[i])) continue;
            temp.add(nums[i]);
            backtrack(ans, temp, nums);
            temp.remove(temp.size()-1);
        }
    }
```

3. LRU 缓存

```
class LRUCache {

    HashMap<Integer, Node> map;
    Node first;
    Node end;
    int cap;
    int cur;

    public LRUCache(int capacity) {
        this.map = new HashMap<>();
        this.cap = capacity;
        this.cur = 0;
    }

    public int get(int key) {
        if (!map.containsKey(key)){
            return -1;
        }else{
            Node n = map.get(key);
            nodeToEnd(n);
            return n.value;
        }
    }

    public void put(int key, int value) {
        if (map.containsKey(key)){
            Node n = map.get(key);
            nodeToEnd(n);
            n.value = value;
        }else{
            Node newNode = new Node(key, value);
            if (cur < cap){
                cur ++;
                map.put(key, newNode);
                addToEnd(newNode);
            }else{
                map.remove(first.key);
                map.put(key, newNode);
                removeFirst();
                addToEnd(newNode);
            }
        }
    }

    private void nodeToEnd(Node node){
        if (node == end) return;
        if (node == first){
            node.next.before = null;
            first = node.next;
        }else{
            node.before.next = node.next;
            node.next.before = node.before;
        }
        node.next = null;
        end.next = node;
        node.before = end;
        end = node;
    }

    private void addToEnd(Node n){
        if (end == null){
            end = n;
            first = n;
        }else{
            end.next = n;
            n.before = end;
            end = n;
        }
    }
    private void removeFirst(){
        if (first == end){
            first = null;
            end = null;
        }else{
            first.next.before = null;
            first = first.next;
        }
    }
    class Node{
        int value;
        int key;
        Node next;
        Node before;
        public Node(int key, int value){
            this.key = key;
            this.value = value;
        }
    }
```

4. 二叉树的最近公共祖先

```
class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
    if (root == null || root == p || root == q ) {
        return root;
    }
    TreeNode leftCommonAncestor =  lowestCommonAncestor(root.left, p, q);
    TreeNode rightCommonAncestor =  lowestCommonAncestor(root.right, p, q);
    //在左子树为空，那一定在右子树中
    if(leftCommonAncestor == null){
        return rightCommonAncestor;
    }
    //在右子树为空，那一定在左子树中
    if(rightCommonAncestor == null){
        return leftCommonAncestor;
    }
    //不在左子树，也不在右子树，那说明是根节点
    return root;
}
}
```

5. 二叉树的直径

```
public class Solution {
    int max = 0;
    public int diameterOfBinaryTree(TreeNode root) {
        if(root == null)
            return 0;
        max = Math.max(max, helper(root.left) + helper(root.right));
        diameterOfBinaryTree(root.left);
        diameterOfBinaryTree(root.right);
        return max;
    }
    public int helper(TreeNode root){
        if(root == null)
            return 0;
        return 1 + Math.max(helper(root.left), helper(root.right));
    }
```

6. 两个有序数组合并，nums2 到 nums1 中，nums1 空余足够
   先把 num1 的内容复制到 n 到 n+m，然后正常插入就行
7. 二叉树中最大路径和

```
int max = Integer.MIN_VALUE;

public int maxPathSum(TreeNode root) {
    helper(root);
    return max;
}
int helper(TreeNode root) {
    if (root == null) return 0;

    int left = Math.max(helper(root.left), 0);
    int right = Math.max(helper(root.right), 0);

    //求的过程中考虑包含当前根节点的最大路径
    max = Math.max(max, root.val + left + right);

    //只返回包含当前根节点和左子树或者右子树的路径
    return root.val + Math.max(left, right);
}
```

8. 从前序，中序遍历构造二叉树

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

private TreeNode buildTreeHelper(int[] preorder, int p_start, int p_end, int[] inorder, int
i_start, int i_end) {
    // preorder 为空，直接返回 null
    if (p_start == p_end) {
        return null;
    }
    int root_val = preorder[p_start];
    TreeNode root = new TreeNode(root_val);
    //在中序遍历中找到根节点的位置
    int i_root_index = 0;
    for (int i = i_start; i < i_end; i++) {
        if (root_val == inorder[i]) {
            i_root_index = i;
            break;
        }
    }
    int leftNum = i_root_index - i_start;
    //递归的构造左子树
    root.left = buildTreeHelper(preorder, p_start + 1, p_start + leftNum + 1, inorder, i_start,
i_root_index);
    //递归的构造右子树
    root.right = buildTreeHelper(preorder, p_start + leftNum + 1, p_end, inorder, i_root_index
+ 1, i_end);
    return root;
}
}
```

9. 最长连续序列

   Hashset 装，从头开始，如果 a[i]-1 不在 set，就开始遍历计数

10. 二叉搜索树与双向链表

```
public class Solution {
    private TreeNode pre = null; //保存当前节点的前一个节点
    private TreeNode head = null;//保存链表的头结点
    public TreeNode Convert(TreeNode pRootOfTree) {
        if(pRootOfTree==null) return null;
        inOrder(pRootOfTree);
        return head;
    }
    private void inOrder(TreeNode node) {
        if (node == null) return;
        inOrder(node.left);
        node.left = pre;
        if (pre != null)  pre.right = node;
        pre = node;
        if (head == null) head = node;
        inOrder(node.right);
    }
```

11. 二叉树的前序，中序遍历

```
public List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> result=new ArrayList<Integer>();
        Stack<TreeNode> stack=new Stack<>();
        if(root==null)
                return result;
        stack.push(root);
        while(!stack.isEmpty()){
                TreeNode pNode=stack.pop();
                result.add(pNode.val);
                if(pNode.right!=null)
                        stack.push(pNode.right);
                if(pNode.left!=null)
                        stack.push(pNode.left);
        }
        return result;
}
```

```
public List<Integer> inorderTraversal(TreeNode root) {
    List<Integer> list=new ArrayList<Integer>();
    Stack<TreeNode> s=new Stack<>();
    TreeNode top=root;
    while(top!=null || !s.isEmpty()){
            while(top!=null){
                    s.push(top);
                    top=top.left;
            }
            top=s.pop();
            list.add(top.val);
            top=top.right;
    }
    return list;
}
```

```
public List<Integer> postOrderTrace(Tree root) {
                List<Integer> list = new ArrayList<Integer>();
                Stack<Tree> s = new Stack<>();
                Tree top = root;
                Tree pre=null;
                while (top != null || !s.isEmpty()) {
                        while (top != null) {
                                s.push(top);
                                top = top.left;
                        }
                        top = s.peek();
                        if (top.right != null && pre != top.right) {
                                top = top.right;
                        } else {
                                top = s.pop();
                                list.add(top.val);
                                pre = top;
                                top = null;
                        }

                }
                return list;
        }
```

12. 合并 k 个有序链表

```
public ListNode mergeKLists(ListNode[] lists) {
    int min_index = 0;
    ListNode head = new ListNode(0);
    ListNode h = head;
    while (true) {
        boolean isBreak = true;//标记是否遍历完所有链表
        int min = Integer.MAX_VALUE;
        for (int i = 0; i < lists.length; i++) {
            if (lists[i] != null) {
                //找出最小下标
                if (lists[i].val < min) {
                    min_index = i;
                    min = lists[i].val;
                }
                //存在一个链表不为空，标记改完 false
                isBreak = false;
            }

        }
        if (isBreak) {
            break;
        }
        //加到新链表中
        ListNode a = new ListNode(lists[min_index].val);
        h.next = a;
        h = h.next;
        //链表后移一个元素
        lists[min_index] = lists[min_index].next;
    }
    h.next = null;
    return head.next;
}
```

13. 合满足要求的连续最短子序列

```java
public int minSubArrayLen(int s, int[] nums) {
    int n = nums.length;
    if (n == 0) {
        return 0;
    }
    int left = 0;
    int right = 0;
    int sum = 0;
    int min = Integer.MAX_VALUE;
    while (right < n) {
        sum += nums[right];
        right++;
        while (sum >= s) {
            min = Math.min(min, right - left);
            sum -= nums[left];
            left++;
        }
    }
    return min == Integer.MAX_VALUE ? 0 : min;
}
```

14. 从(0,0)点出发走 n 步，上下左右都可，有几个落点

```java
public int findCount(int step){
    if (step < 0) return 0;
    if (step == 0) return 1;
    int[][] flag = new int[step+1][step+1];
    return helper(step, i: 0, j: 0, flag);
}
private int helper(int step, int i, int j, int[][] flag){
    if (i < 0 || j < 0) return 0;
    if (step == 0){
        if (flag[i][j] == 0){
            flag[i][j] = 1;
            return 1;
        }else {
            return 0;
        }
    }
    return helper( step: step-1, i: i+1, j, flag) +
           helper( step: step-1, i: i-1, j, flag) +
           helper( step: step-1, i, j: j+1, flag) +
           helper( step: step-1, i, j: j-1, flag);
}
public int solution2(int step){
    int index = step;
    int count = 0;
    while (index >= 0){
        count += index + 1;
        index = index - 2;
    }
    return count;
}
```

15.