数据结构

剑指 Offer 07. 重建二叉树

方法1：DFS

```
Map<Integer, Integer> map = new HashMap<>();

public TreeNode buildTree(int[] preorder, int[] inorder) {
    if (preorder == null || preorder.length == 0 || inorder == null || inorder.length
== 0) return null;
    for (int i = 0; i < inorder.length; i++) map.put(inorder[i], i);
    return dfs(preorder, 0, preorder.length - 1, inorder, 0, inorder.length - 1);
}


public TreeNode dfs(int[] preorder, int preStart, int preEnd, int[] inorder, int
inStart, int inEnd) {

    if (preStart > preEnd) return null;
    TreeNode root = new TreeNode(preorder[preStart]);
    int mid = map.get(preorder[preStart]);
    root.left = dfs(preorder, preStart + 1, preStart + (mid - inStart),
            inorder, inStart, mid - 1);
    root.right = dfs(preorder, preStart + (mid - inStart) + 1, preEnd,
            inorder, mid + 1, inEnd);
    return root;
}
```

98. 验证二叉搜索树

方法1：递归

- 注意边界 long类型

```
    public boolean isValidBST(TreeNode root) {
        return helper(root,Long.MIN_VALUE,Long.MAX_VALUE);
    }


    public boolean helper(TreeNode root, long down ,long up){
        if(root ==null) return true;
        if(root.val<=down||root.val>=up) return false;
        return helper(root.left,down,root.val)&&helper(root.right,root.val,up);
    }
```

方法2:中序遍历

```
TreeNode pre;

public boolean isValidBST(TreeNode root) {
    if (root == null) return true;
    if (!isValidBST(root.left)) {
        return false;
    }
    if (pre != null && pre.val >= root.val) {
        return false;
    }
    pre = root;
    if (!isValidBST(root.right)) {
        return false;
    }
    return true;
}
```

617. 合并二叉树

方法1：DFS

```
        public TreeNode mergeTrees(TreeNode root1, TreeNode root2) {
              return dfs(root1, root2);
        }

        private TreeNode dfs(TreeNode root1, TreeNode root2) {
            if (root1 == null || root2 == null) return root1 == null ? root2 : root1;
            root1.val += root2.val;
            root1.left = dfs(root1.left, root2.left);
            root1.right = dfs(root1.right, root2.right);
            return root1;
        }
```

方法2：BFS

```
public TreeNode mergeTrees(TreeNode root1, TreeNode root2) {
    if (root1 == null || root2 == null) return root1 == null ? root2 : root1;
    Queue<TreeNode> q = new LinkedList<>();
    q.offer(root1);
    q.offer(root2);
    while (!q.isEmpty()) {
        TreeNode r1 = q.poll();
        TreeNode r2 = q.poll();
        r1.val += r2.val;
        if (r1.left != null && r2.left != null) {
            q.offer(r1.left);
            q.offer(r2.left);
        } else if (r1.left == null) {
            r1.left = r2.left;
        }

        if (r1.right != null && r2.right != null) {
            q.offer(r1.right);
            q.offer(r2.right);
        } else if (r1.right == null) {
            r1.right = r2.right;
        }
    }
    return root1;
}
```

102. 二叉树的层序遍历

```
public List<List<Integer>> levelOrder(TreeNode root) {
    List<List<Integer>> res = new ArrayList<>();
    if (root == null) return res;
    Queue<TreeNode> q = new LinkedList<>();
    q.offer(root);
    while (!q.isEmpty()) {
        int size = q.size();
        List<Integer> sub = new ArrayList<>();
        for (int i = 0; i < size; ++i) {
            TreeNode curr = q.poll();
            sub.add(curr.val);
            if (curr.left != null) q.offer(curr.left);
            if (curr.right != null) q.offer(curr.right);
        }
        res.add(new ArrayList<>(sub));
    }
    return res;
}
```

429. N 叉树的层序遍历

```
public List<List<Integer>> levelOrder(Node root) {
    List<List<Integer>> res = new ArrayList<>();
    if (root == null) return res;
    Queue<Node> q = new LinkedList<>();
    q.offer(root);
    while (!q.isEmpty()) {
        int size = q.size();
        List<Integer> sub = new ArrayList<>();
        for (int i = 0; i < size; ++i) {
            Node curr = q.poll();
            sub.add(curr.val);
            if (curr.children != null && curr.children.size() != 0) {
                for (int j = 0; j < curr.children.size(); j++) {
                    q.offer(curr.children.get(j));
                }
            }
        }
        res.add(new ArrayList<>(sub));
    }
    return res;
}
```

剑指 Offer 32 - III. 从上到下打印二叉树 III

方法1：BFS+List索引插入

```
public List<List<Integer>> levelOrder(TreeNode root) {
    List<List<Integer>> res = new ArrayList<>();
    if (root == null) return res;
    Queue<TreeNode> q = new LinkedList<>();
    q.offer(root);
    int level = 0;
    while (!q.isEmpty()) {
        int size = q.size();
        List<Integer> sub = new ArrayList<>();
        for (int i = 0; i < size; ++i) {
            TreeNode curr = q.poll();
            if (level % 2 == 0) sub.add(curr.val);
            else sub.add(0, curr.val);
            if (curr.left != null) q.offer(curr.left);
            if (curr.right != null) q.offer(curr.right);
        }
        res.add(new ArrayList<>(sub));
        level++;
    }
    return res;
}
```

剑指 Offer 22. 链表中倒数第k个节点

方法1：快慢指针

```java
public ListNode getKthFromEnd(ListNode head, int k) {
    ListNode slow = head, fast = head;
    while (k-- > 0) fast = fast.next;
    while (fast != null) {
        slow = slow.next;
        fast = fast.next;
    }
    return slow;
}
```

450. 删除二叉搜索树中的节点

方法1：前驱结点+后继节点

> 很好的题

```java
/**
 * https://leetcode-cn.com/problems/delete-node-in-a-bst/solution/shan-chu-er-cha-sou-suo-shu-zhong-de-jie-dian-by-l/
 */

public TreeNode deleteNode(TreeNode root, int key) {
    if (root == null) return null;
    if (key > root.val) root.right = deleteNode(root.right, key);
    if (key < root.val) root.left = deleteNode(root.left, key);
    if (key == root.val) {
        if (root.left == null && root.right == null) root = null;
        else if (root.right != null) {
            root.val = successor(root);
            root.right = deleteNode(root.right, root.val);
        } else if (root.right == null) {
            root.val = predecessor(root);
            root.left = deleteNode(root.left, root.val);
        }
    }
    return root;
}


public int successor(TreeNode root) {
    root = root.right;
    while (root.left != null) root = root.left;
    return root.val;
}
```

```
public int predecessor(TreeNode root) {
    root = root.left;
    while (root.right != null) root = root.right;
    return root.val;
}
```

剑指 Offer 24. 反转链表

https://leetcode-cn.com/problems/fan-zhuan-lian-biao-lcof/solution/fan-zhuan-lian-biao-yi-dong-de-shuang-zhi-zhen-jia/

方法1：模拟

```
public ListNode reverseList(ListNode head) {
    ListNode curr = null, pre = head;
    while (pre != null) {
        ListNode next = pre.next;
        pre.next = curr;
        curr = pre;
        pre = next;
    }
    return curr;
}
```

方法2：递归

```
public ListNode reverseList(ListNode head) {
    if (head == null || head.next == null) return head;
    ListNode last = reverseList(head.next);//返回反转之后的头节点
    head.next.next = head;//head的next节点 的next指向head 形成反转
    head.next = null;//head 的next指向null
    return last;
}
```

92. 反转链表 II

方法1：常规迭代

```
/**
https://leetcode-cn.com/problems/reverse-linked-list-ii/solution/fan-zhuan-lian-biao-
ii-by-leetcode/
**/
public ListNode reverseBetween(ListNode head, int m, int n) {
    ListNode curr = head, pre = null;
    for (int i = 1; i < m; i++) {
        pre = curr;
```

```
            curr = curr.next;
        }
        System.out.printf("%d\n", curr.val);
        ListNode con = pre, tail = curr;
        for (int i = 0; i < n - m + 1; i++) {
            ListNode third = curr.next;
            curr.next = pre;
            pre = curr;
            curr = third;
        }
        System.out.printf("%d\n", pre.val);
        // System.out.printf("%d\n",con.val);
        if (con != null) con.next = pre;
        else head = pre;
        tail.next = curr;
        return head;


    }
```

方法2：进阶版迭代

```
    public ListNode reverseBetween(ListNode head, int m, int n) {
        ListNode dummy = new ListNode(0);
        dummy.next = head;
        ListNode pre = dummy;
        ListNode cur = pre.next;
        for (int i = 1; i < m; i++) {
            pre = pre.next;
            cur = cur.next;
        }
        for (int i = 0; i < n - m; i++) {
            ListNode tmp = cur.next;
            cur.next = tmp.next;
            tmp.next = pre.next;
            pre.next = tmp;
        }
        return dummy.next;
    }
```

方法3：递归

todo

剑指 Offer 25. 合并两个排序的链表

```
public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        if(l1==null||l2==null) return l1==null?l2:l1;
        ListNode dummy = new ListNode(-1);
        ListNode curr = dummy;
        while(l1!=null&&l2!=null){
            if(l1.val<=l2.val){
                curr.next = l1;
                l1= l1.next;
            }else{
                curr.next = l2;
                l2= l2.next;
            }
            curr= curr.next;
        }
         if(l1!=null){
            curr.next = l1;
        }
        if(l2!=null){
            curr.next = l2;
        }
        return dummy.next;

    }
```

23. 合并K个升序链表

https://leetcode-cn.com/problems/merge-k-sorted-lists/solution/c-you-xian-dui-lie-liang-liang-he-bing-fen-zhi-he-/

方法1：Queue

```
    public ListNode mergeKLists(ListNode[] lists) {
        Queue<ListNode> pq = new PriorityQueue<>((l1,l2)->l1.val-l2.val);
        for(ListNode l:lists){
            if(l!=null) pq.offer(l);
        }
        ListNode dummy = new ListNode(-1);
        ListNode curr = dummy;
        while(!pq.isEmpty()){
            ListNode tmp = pq.poll();
            curr.next = tmp;
            curr = curr.next;
            if(tmp.next!=null) pq.offer(tmp.next);
        }
        return dummy.next;
    }
```

方法2：成对合并 //todo

方法3：分治合并

```
 public ListNode mergeKLists(ListNode[] lists) {
        if (lists == null || lists.length == 0) return null;
        int len = lists.length;
        return mergeKLists(lists, 0, len - 1);
    }

    private ListNode mergeKLists(ListNode[] lists, int l, int r) {
        if (l == r) return lists[l];
        int mid = l + (r - l) / 2;
        ListNode l1 = mergeKLists(lists, l, mid);
        ListNode l2 = mergeKLists(lists, mid + 1, r);
        return mergeTwoSortedListNode(l1, l2);
    }

    private ListNode mergeTwoSortedListNode(ListNode l1, ListNode l2) {
        if (l1 == null) return l2;
        if (l2 == null) return l1;
        if (l1.val < l2.val) {
            l1.next = mergeTwoSortedListNode(l1.next, l2);
            return l1;
        } else {
            l2.next = mergeTwoSortedListNode(l1, l2.next);
            return l2;
        }
    }
```

25. K 个一组翻转链表

```
public ListNode reverseKGroup(ListNode head, int k) {
        ListNode dummy = new ListNode(-1);
        dummy.next = head;
        ListNode pre = dummy, end = dummy;
        while(end.next!=null){
            for(int i =0;i<k && end!=null;i++){
                end =end.next;
            }
            if(end == null) break;
            ListNode start = pre.next;
            ListNode next = end.next;
            end.next = null;
            pre.next = reverse(start);
            start.next = next;
            pre =start;
            end = pre;
        }
        return dummy.next;
    }
    private ListNode reverse(ListNode head){
        ListNode pre = null,curr = head;
        while(curr!=null){
            ListNode next = curr.next;
            curr.next =pre;
            pre = curr;
            curr = next;
        }
        return pre;
    }
```

543. 二叉树的直径

```
int ans = 1;

public int diameterOfBinaryTree(TreeNode root) {
    dfs(root);
    return ans - 1;
}

public int dfs(TreeNode root) {
    if (root == null) return 0;
    int L = dfs(root.left);
    int R = dfs(root.right);
    ans = Math.max(ans, L + R + 1);
    return Math.max(L, R) + 1;
}
```

234. 回文链表

方法1：递归

```
    ListNode left ;
    public boolean isPalindrome(ListNode head) {
        left = head;
       return recur(head);
    }
```

```java
    public boolean recur(ListNode right){
        if(right == null) return true;
        boolean ans =    recur(right.next);
        ans = ans && (left.val == right.val);
        left = left.next;
        return ans;
    }
```

226. 翻转二叉树

方法1：递归

```java
    public TreeNode invertTree(TreeNode root) {
        if(root==null) return null;
        TreeNode l = invertTree(root.left);
        TreeNode r = invertTree(root.right);
        root.left = r;
        root.right = l;
        return root;
    }
```

方法2：迭代

```java
public TreeNode invertTree(TreeNode root) {
        if (root == null) return null;
        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);
        while (!queue.isEmpty()) {
            TreeNode cur = queue.poll();
            TreeNode tmp = cur.left;
            cur.left = cur.right;
            cur.right = tmp;
            if (cur.left != null) queue.offer(cur.left);
            if (cur.right != null) queue.offer(cur.right);
        }
        return root;
    }
```

基础题

151. 翻转字符串里的单词

方法1：基础版

```
        public String reverseWords(String s) {
            if(s==null||s.length()==0) return null;
            char[] chas = s.toCharArray();
            reverse(chas,0,chas.length-1);
            String t = String.valueOf(chas);
            String[] arr = t.trim().split("\\s+");
            StringBuilder ans =new StringBuilder();
            for(String a: arr){
                char[] tmp =a.toCharArray();
                reverse(tmp,0,a.length()-1);
                ans.append(tmp);
                ans.append(" ");
            }
            return ans.toString().trim();
        }


        private void reverse(char[] chas, int l, int r) {
            while (l < r) {
                char tmp = chas[l];
                chas[l++] = chas[r];
                chas[r--] = tmp;
            }
        }
```

方法2：O(1)空间

```
/**
 * https://leetcode.com/problems/reverse-words-in-a-string/discuss/47720/Clean-Java-two-pointers-solution-(no-trim(-)-no-split(-)-no-StringBuilder
 */
public String reverseWords(String s) {
    char[] chas = s.toCharArray();
    int n = chas.length;
    reverse(chas, 0, chas.length - 1);
    reverseWords(chas, n);
    return cleanSpaces(chas, n);
}
```

```java
public String cleanSpaces(char[] chas, int n) {
    int l = 0, r = 0;
    while (r < n) {
        while (r < n && chas[r] == ' ') r++;
        while (r < n && chas[r] != ' ') chas[l++] = chas[r++];
        while (r < n && chas[r] == ' ') r++;
        if (r < n) chas[l++] = ' ';
    }
    return new String(chas).substring(0, l);
}

public void reverseWords(char[] chas, int n) {
    int l = 0, r = 0;
    while (l < chas.length) {
        while (l < r || l < n && chas[l] == ' ') l++;
        while (r < l || r < n && chas[r] != ' ') r++;
        reverse(chas, l, r - 1);
    }
}


public void reverse(char[] chas, int l, int r) {
    while (l < r) {
        char t = chas[l];
        chas[l++] = chas[r];
        chas[r--] = t;
    }
}
```

54. 螺旋矩阵

- 做个d来进行方向调换方向

```java
int R, C;
int[][] dirs = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};

public List<Integer> spiralOrder(int[][] matrix) {
    R = matrix.length;
    C = matrix[0].length;
    int r = 0, c = 0, d = 0;
    List<Integer> res = new ArrayList<>();
    for (int i = 0; i < R * C; i++) {
        res.add(matrix[r][c]);
        matrix[r][c] = -101;
        int nr = r + dirs[d][0], nc = c + dirs[d][1];
        if (!inArea(nr, nc) || matrix[nr][nc] == -101) {
            d = (d + 1) % 4;
            nr = r + dirs[d][0];
            nc = c + dirs[d][1];
        }
        r = nr;
        c = nc;

    }
    return res;
}


private boolean inArea(int r, int c) {
    return r >= 0 && r < R && c >= 0 && c < C;
}
```

剑指 Offer 53 - II. 0～n-1中缺失的数字

方法1：模拟

```
public int missingNumber(int[] nums) {
    for (int i = 0; i < nums.length; ++i) {
        if (i != nums[i]) return i;
    }
    return nums.length;
}
```

方法2：二分

剑指 Offer 03. 数组中重复的数字

方法1：暴力

```
public int findRepeatNumber(int[] nums) {
    Set<Integer> set = new HashSet<>();
    for (int n : nums) {
        if (set.contains(n)) return n;
        set.add(n);
    }
    return 0;
}
```

方法2

```
class Solution {
  public int findRepeatNumber(int[] nums) {
        if (nums == null || nums.length == 0) return -1;
        int n = nums.length;
        for (int i = 0; i < n; i++) {
            while (nums[i] != i) {
                if (nums[i] == nums[nums[i]]) {
                    return nums[i];
                }
                swap(nums, nums[i], i);
            }
        }
        return -1;
    }

    public void swap(int[] arr, int m, int n) {
        int temp = arr[m];
        arr[m] = arr[n];
        arr[n] = temp;
    }
}
```

89. 格雷编码

方法1：DP

```
        public List<Integer> grayCode(int n) {
            List<Integer> res = new ArrayList<>();
            res.add(0);
            for (int i = 0; i < n; i++) {
                int add = 1 << i;
                for (int j = res.size() - 1; j >= 0; j--) {
                    res.add(res.get(j)+add);
                }
            }
```

```
//              for (int g : res) {
//
System.out.println(PrintUtils.addZeroForNum(Integer.toBinaryString(g), 3));
//              }
//              System.out.println("------------------");
            return res;
```

[189. 旋转数组]

方法1：O(n)空间

```
public void rotate(int[] nums, int k) {
    int n = nums.length;
    int[] tmp = new int[n];
    for (int i = 0; i < n; i++) tmp[(i + k) % n] = nums[i];
    for (int i = 0; i < n; i++) nums[i] = tmp[i];
}
```

方法2：O(1)空间

```
public void rotate(int[] nums, int k) {
    int n = nums.length;
    k %= n;
    reverse(nums, 0, n - 1);//整体反转
    reverse(nums, 0, k - 1);//0~k-1反转
    reverse(nums, k, n - 1);//k~n-1反转
}

public void reverse(int[] A, int l, int r) {
    while (l < r) {
        int m = A[l];
        A[l++] = A[r];
        A[r--] = m;
    }
}
```

[面试题 17.14. 最小K个数]

方法1：小根堆

```
public int[] smallestK(int[] arr, int k) {
    PriorityQueue<Integer> pq = new PriorityQueue<>((o1, o2) -> o2 - o1);
    for (int a : arr) {
        if (k > 0) pq.offer(a);
        if (!pq.isEmpty()) {
            if (pq.size() > k && pq.peek() > a) {
                pq.poll();
            }
        }
    }
    while (!pq.isEmpty() && pq.size() > k) {
        pq.poll();
    }
    int[] res = new int[k];
    int idx = k;
    while (!pq.isEmpty()) {
        res[--idx] = pq.poll();
    }
    return res;
}
```

468. 验证IP地址

方法1：验证

```
public String validIPAddress(String IP) {
    String[] arr = IP.split("\\.");
    if (arr.length <= 1) {
        if (IP.endsWith(":")) return "Neither";
        arr = IP.split(":");
        boolean ipv6 = isIPV6(arr);
        if (ipv6) return "IPv6";
    } else {
        if (IP.endsWith(".")) return "Neither";
        boolean ipv4 = isIPV4(arr);
        if (ipv4) return "IPv4";
    }
    return "Neither";
}


private boolean isIPV4(String[] arr) {
    if (arr.length != 4) return false;
    for (String a : arr) {
        if (a.length() == 0 || a.length() > 3) return false;
        if (a.startsWith("0") && a.length() > 1) return false;
        if (!isDigit(a)) return false;
        if (Integer.parseInt(a) >= 256 || Integer.parseInt(a) < 0) return false;
    }
    return true;
}
```

```java
private boolean isIPV6(String[] arr) {
    if (arr.length != 8) return false;
    for (String a : arr) {
        if (a.length() == 0 || a.length() > 4) return false;
        if (!checkIPV6(a)) return false;
    }
    return true;
}


private boolean isDigit(String a) {
    char[] chas = a.toCharArray();
    for (char c : chas) {
        int ic = (int) c - (int) '0';
        if (!(ic >= 0 && ic <= 9)) return false;
    }
    return true;
}


private boolean checkIPV6(String a) {
    for (int i = 0; i < a.length(); ++i) {
        char c = a.charAt(i);
        if (!Character.isDigit(c) && !('a' <= c && c <= 'f') && !('A' <= c && c <=
'F')) {
            return false;
        }
    }
    return true;
}
```

53. 最大子序和

方法1：贪心

```java
public int maxSubArray(int[] nums) {
    int currSum = nums[0], maxSum = currSum;
    for (int i = 1; i < nums.length; i++) {
        currSum = Math.max(nums[i], currSum + nums[i]);
        maxSum = Math.max(maxSum, currSum);
    }
    return maxSum;
}
```

50. Pow(x, n)

方法1：快速幂+递归

```
public double myPow(double x, int n) {
    long N = n;
    return N >= 0 ? helper(x, N) : 1.0 / helper(x, -N);
}

public double helper(double x, long N) {
    if (N == 0) return 1.0;
    double y = helper(x, N / 2);
    double res;
    if (N % 2 == 0) res = y * y;
    else res = y * y * x;
    return res;
}
```

方法2：快速幂+迭代

```
public double myPow(double x, int n) {
    long N = n;
    return N >= 0 ? helper(x, N) : 1.0 / helper(x, -N);
}

public double helper(double x, long N) {
    double res = 1.0;
    double x_c = x;
    while (N > 0) {
        if (N % 2 == 1) {
            res *= x_c;
        }
        x_c *= x_c;
        N /= 2;
    }
    return res;
}
```

5.第k大元素

```
    public int kthLargestElement(int n, int[] nums) {
        // write your code here
        Queue<Integer> pq = new PriorityQueue<>();
        for (int a : nums) {
            while (!pq.isEmpty() && (pq.size()>=n && pq.peek() < a)) {
                pq.poll();
            }
            pq.offer(a);
        }
        if (!pq.isEmpty() && pq.size() > n) {
            pq.poll();
        }
        // System.out.printf("%d\n",pq.size());
        return pq.isEmpty() ? -1 : pq.peek();
    }
```

17. 电话号码的字母组合

```
    int N ;
     Map<Character,List<String>> dict = new HashMap<>();
     List<String> res = new ArrayList<>();
     char[] chas;


     public List<String> letterCombinations(String digits) {
         if(digits ==null||digits.length() ==0) return res;
          N = digits.length();
          chas= digits.toCharArray();
         dict.put('2', Arrays.asList("a", "b", "c"));
         dict.put('3', Arrays.asList("d", "e", "f"));
         dict.put('4', Arrays.asList("g", "h", "i"));
         dict.put('5', Arrays.asList("j", "k", "l"));
         dict.put('6', Arrays.asList("m", "n", "o"));
         dict.put('7', Arrays.asList("p", "q", "r", "s"));
         dict.put('8', Arrays.asList("t", "u", "v"));
         dict.put('9', Arrays.asList("w", "x", "y", "z"));
        dfs(new ArrayList<>(),0);
        return res;
     }

     private void  dfs(List<String> sub ,int idx ){
         if(N == sub.size()){
             StringBuilder sb = new StringBuilder();
             for(String s : sub)  sb.append(s);
             res.add(sb.toString());
             return;
         }
         List<String> seed = dict.get(chas[idx]);
         for(String s: seed){
             sub.add(s);
             dfs(sub,idx+1);
             sub.remove(sub.size()-1);
         }
     }
```

131. 分割回文串

```
  List<List<String>> res = new ArrayList<>();

    public List<List<String>> partition(String s) {
        if(s==null||s.length()==0) return res;
        dfs(s,new ArrayList<>(),0);
        return res;
    }

    private void dfs(String s ,List<String> sub , int start){
        if(start ==s.length()){
            res.add(new ArrayList<>(sub));
            return ;
        }

        for(int end = start;end<s.length();end++){
            if(isPalindrome(s,start,end)){
                sub.add(s.substring(start,end+1));
                dfs(s,sub,end+1);
                sub.remove(sub.size()-1);
            }
        }

    }



    public boolean isPalindrome(String str, int l, int r) {
        while (l < r) {
            if (str.charAt(l) != str.charAt(r)) return false;
            l++;
            r--;
        }
        return true;
    }
```

图类问题

1162. 地图分析

方法1：BFS

//todo

方法2：DP

//todo

动态规划

剑指 Offer 46. 把数字翻译成字符串

方法1：基础版DP

- 考察当前的数是否在10 和 25 之间

```
    public int translateNum(int num) {
        String s = String.valueOf(num);
        int N = s.length();
        int[] f = new int[N + 1];
        f[0] = 1;
        f[1] = 1;
        for (int i = 2; i <= N; i++) {
            String curr = s.substring(i - 2, i);
            boolean flag = curr.compareTo("10") >= 0 && curr.compareTo("25") <= 0;
            if (flag) f[i] = f[i - 2] + f[i - 1];
            else f[i] = f[i - 1];
        }
        return f[N];
    }
```

方法2：O(1)空间压缩DP

```
public int translateNum(int num) {
    String s = String.valueOf(num);
    int N = s.length();
    int pre = 1, curr = 1;
    for (int i = 2; i <= N; i++) {
        String tmp = s.substring(i - 2, i);
        boolean flag = tmp.compareTo("10") >= 0 && tmp.compareTo("25") <= 0;
        int next;
        if (flag) next = curr + pre;
        else next = curr;
        pre = curr;
        curr = next;
    }
    return curr;
}
```

5. 最长回文子串

方法1：DP

```
public String longestPalindrome(String s) {
    if (s == null || s.length() == 0) return "";
    int n = s.length();
    boolean[][] f = new boolean[n][n];
    String ans = "";
    int maxL = 0;
    for (int j = 0; j < n; ++j) {
        for (int i = 0; i <= j; i++) {
            if (i == j) f[i][i] = true;
            if (s.charAt(i) == s.charAt(j) && ((j - i) <= 2 || f[i + 1][j - 1])) {
                f[i][j] = true;
            }
            if (f[i][j] && (j - i + 1) > maxL) {
                maxL = j - i + 1;
                ans = s.substring(i, j + 1);
            }
        }
    }
    return ans;
}
```

方法2：中心扩展法

```
/**
 * 5. 最长回文子串 LeetCode Medium
 *
 * @param s
 * @return
 */
public String longestPalindrome(String s) {
    if (s == null || s.length() == 0) {
        return "";
    }
    int n = s.length();
    int start = 0, end = 0;
    for (int i = 0; i < n; i++) {
        //获取到当前点i 的奇回文和偶回文的最大长度
        int len1 = expandBySeed(s, i, i);
        int len2 = expandBySeed(s, i, i + 1);
        //取最大长度，然后扩展
        int len = Math.max(len1, len2);
        if (len > (end - start)) {
            start = i - (len - 1) / 2;
            end = i + len / 2;
        }
    }
    return s.substring(start, end + 1);
}


/**
 * 由中心往两边扩散，返回满足最大回文的长度
 *
 * @param s
 * @param start
 * @param end
 * @return
 */
private int expandBySeed(String s, int start, int end) {
    int n = s.length();
    while (start >= 0 && end < n && s.charAt(start) == s.charAt(end)) {
        start--;
        end++;
    }
    return end - start - 1;
}
```

22. 括号生成

方法1：DFS

```
public List<String> generateParenthesis(int n) {
    List<String> ans = new ArrayList<>();
    dfs("", n, n, ans);
    return ans;
}
```

```java
private void dfs(String currStr, int l, int r, List<String> ans) {
    if (l == 0 && r == 0) {
        ans.add(currStr);
        return;
    }
    if (l > r) return;//左边剩余的括号数量不能比右边的剩余的多，多了，说明先用了右边括号
    if (l > 0) dfs(currStr + "(", l - 1, r, ans);
    if (r > 0) dfs(currStr + ")", l, r - 1, ans);
}
```

221. 最大正方形

- `f[i][j]`是以最大`[i,j]`结尾的正方形的最大边长

```java
    public int maximalSquare(char[][] matrix) {
        if(matrix==null||matrix.length==0||matrix[0].length==0) return 0;
        int R = matrix.length,C = matrix[0].length;
        int[][] f = new int[R][C];
        int maxL = 0;
        for(int r = 0;r<R;r++){
            for(int c = 0;c<C;c++){
                f[r][c] = matrix[r][c] =='1' ?1:0;
                maxL = Math.max(maxL,f[r][c]);
            }
        }

        for(int r=1;r<R;r++){
            for(int c=1;c<C;c++){
                if(matrix[r][c]=='1'){
                    // System.out.printf("%d,%d\n",r,c);
                    f[r][c] = Math.min(Math.min(f[r-1][c],f[r][c-1]),f[r-1][c-1])+1;
                    //注意+1
                    // System.out.printf("%d\n", f[r][c]);
                    maxL = Math.max(maxL,f[r][c]);
                }
            }
        }
        return maxL*maxL;
    }
```

300. 最长递增子序列

todo

结构设计

307. 区域和检索 - 数组可修改

todo

面试题 16.25. LRU 缓存

```
class LRUCache {

    class Node{
        int key,val;
        Node pre,next;

        public Node(int key, int val){
            this.key = key;
            this.val = val;
        }
    }


    int capacity;
    int size;
    Node head,tail;
    Map<Integer,Node> map = new HashMap<>();

    public LRUCache(int capacity) {
        head = new Node(-1,-1);
        tail = new Node(-1,-1);
        head.next = tail;
        tail.pre = head;
        this.capacity = capacity;
        size = 0;
    }
```

```java
    public void addHead(int key,int val){
        Node node = new Node(key,val);
        Node next = head.next;
        head.next = node;
        node.pre = head;
        node.next = next;
        next.pre = node;
        size ++;
        map.put(key,node);
        if(size>capacity){
            Node preTail = tail.pre;
            remove(preTail.key);
        }
    }

    public void remove(int  key){
        Node curr = map.get(key);
        Node next = curr.next;
        Node pre = curr.pre;
        pre.next = next;
        next.pre = pre;
        size --;
        map.remove(key);
    }



    public int get(int key) {
        if(!map.containsKey(key)) return -1;
        else{
            Node curr = map.get(key);
            remove(key);
            addHead(curr.key,curr.val);
            return curr.val;
        }

    }

    public void put(int key, int value) {
        if(map.containsKey(key)){
            remove(key);
        }
        addHead(key,value);

    }
}

/**
```

```
 * Your LRUCache object will be instantiated and called as such:
 * LRUCache obj = new LRUCache(capacity);
 * int param_1 = obj.get(key);
 * obj.put(key,value);
 */
```

460. LFU 缓存

```
class LFUCache {




    class Node{
        int key,val;
        int cnt;
        Node pre,next;

        public Node(int key,int val){
            this.key =key;
            this.val = val;
            this.cnt =1;
        }
    }

    class DLList{
        Node head,tail;
        int len;

        public DLList(){
            head = new Node(-1,-1);
            tail = new Node(-1,-1);
            head.next = tail;
            tail.pre = head;
            len = 0;
        }

        public void addHead(Node node){
            Node next = head.next;
            node.next = next;
            node.pre = head;
            head.next = node;
            next.pre = node;
            map.put(node.key,node);
            len++;
        }


        public void remove(Node node){
            Node pre =node.pre;
            Node next = node.next;
            pre.next = next;
            next.pre = pre;
            len--;
            map.remove(node.key);
        }

        public void removeTail(){
            Node preTail = tail.pre;
            remove(preTail);
        }
    }

    Map<Integer,Node> map;
    Map<Integer,DLList> freq;
    int size ;
    int capacity;
    int maxFreq ;

    public LFUCache(int capacity) {
        map = new HashMap<>();
        freq= new HashMap<>();
        this.size = 0;
        this.capacity = capacity;
        maxFreq = 0;
    }

    public int get(int key) {
        if(map.get(key)==null) return -1;
        Node node = map.get(key);
        int preFreq = node.cnt;
        DLList preList = freq.get(preFreq);
        preList.remove(node);
        int currFreq = preFreq+1;
        maxFreq = Math.max(maxFreq,currFreq);
        DLList currList = freq.getOrDefault(currFreq,new DLList());
        node.cnt++;//要修改cnt
        currList.addHead(node);
        freq.put(preFreq ,preList);
        freq.put(currFreq,currList);
        return node.val;
    }
```

```java
    public void put(int key, int value) {
        if(capacity ==0 ) return;
        if(map.get(key)!=null){
            map.get(key).val = value;
            get(key);
            return ;
        }
        Node node = new Node(key,value);
        DLList currList = freq.getOrDefault(1,new DLList());
        currList.addHead(node);
        size++;
        if(size>capacity){
            if(currList.len>1){
                currList.removeTail();
            }else{
                for(int i=2;i<=maxFreq;i++){
                    DLList tmpList = freq.get(i);
                    if(tmpList!=null&&tmpList.len>0){
                        tmpList.removeTail();
                        break;
                    }
                }
            }
            size--;
        }
        freq.put(1,currList);
    }
}

/**
 * Your LFUCache object will be instantiated and called as such:
 * LFUCache obj = new LFUCache(capacity);
 * int param_1 = obj.get(key);
 * obj.put(key,value);
 */
```

384. 打乱数组

todo

剑指 Offer 41. 数据流中的中位数

```
static class MedianFinder {

    PriorityQueue<Integer> maxHeap;//大根堆，保存较小的一半，始终多一个，如果有的多的话
    PriorityQueue<Integer> minHeap;//小根堆，保存较大的一半
    int count;

    /**
     * initialize your data structure here.
     */
    public MedianFinder() {
        count = 0;
        maxHeap = new PriorityQueue<>((o1, o2) -> o2 - o1);
        minHeap = new PriorityQueue<>();
    }

    public void addNum(int num) {
        count++;
        maxHeap.offer(num);
        minHeap.offer(maxHeap.poll());
        if (count % 2 == 1) maxHeap.offer(minHeap.poll());
    }

    public double findMedian() {
        double ans;
        if (count % 2 == 1) ans = (double) maxHeap.peek();
        else ans = (double) (maxHeap.peek() + minHeap.peek()) / 2;
        return ans;
    }
}
```

经典题

42. 接雨水

> Hard

//todo

过题

[25. K 个一组翻转链表]

> Hard

https://leetcode-cn.com/problems/reverse-nodes-in-k-group/solution/tu-jie-kge-yi-zu-fan-zhuan-lian-biao-by-user7208t/

方法1：迭代

```
    public ListNode reverseKGroup(ListNode head, int k) {
        ListNode dummy = new ListNode(-1);
        dummy.next = head;
        ListNode pre = dummy, end = dummy;
        while(end.next!=null){
            for(int i =0;i<k && end!=null;i++){
                end =end.next;
            }
            if(end == null) break;
            ListNode start = pre.next;
            ListNode next = end.next;
            end.next = null;
            pre.next = reverse(start);
            start.next = next;
            pre =start;
            end = pre;
        }
        return dummy.next;
    }

    private ListNode reverse(ListNode head){
        ListNode pre = null,curr = head;
        while(curr!=null){
            ListNode next = curr.next;
            curr.next =pre;
            pre = curr;
            curr = next;
        }
        return pre;
    }
```