写在前面

本系列题解讲解的是《剑指offer》第二版的题解，并且按照顺序给出了题解。

对于刷题，本人是建议大家尽量找出最优解，而不是 AC 了就完事，所以本系列的讲解，每道题都会讲解最优解代码。

由于网站播放视频可能没有 B 站好，所以本系列的讲解视频，也同步到了 B 站，直达链接：完美撒花！剑指offer题解合集(完结)｜有字幕+动画+现场手撕代码｜Java实现

由于 B 站不方便放代码以及问题描述等，所以你可以在本站看视频中的代码以及问题描述，并且每一道题，也会有链接索引到 B 站到视频讲解

> 本资料已经整理成PDF，可以在帅地的公众号「**帅地玩编程**」中回复「**剑指offer**」获取PDF版本哦。

在线阅读

为了方便对一些错误进行更新，本系列也同步到了帅地的网站：playoffer.cn

如果你们喜欢通过在线阅读的方式，那么可以看这个在线的版本，在线版本更新比较实时

在线阅读：剑指offer题解在线版

读者交流群

为了更好着方便让大家交流，帅地弄过好几种群，包括 QQ群，微信群，QQ群方便拉群，微信群需要通过好友请求比较麻烦，最终注册了**企业微信**，通过企业微信来管理群。

所以如果你想要加入交流群，可以关注帅地另外一个公众号「PlayOffer」，之后回复「**加群**」即可获取加群对应二维码，并且通过指定要求去操作，否则无法进群。

可以扫码直达：

题解

开篇：聊一聊算法面试

这个视频讲解了算法面试相关的一些事情：[开篇：聊一聊算法面试]

剑指 Offer 03. 数组中重复的数字

问题描述

找出数组中重复的数字。

在一个长度为 n 的数组 nums 里的所有数字都在 0～n-1 的范围内。数组中某些数字是重复的，但不知道有几个数字重复了，也不知道每个数字重复了几次。请找出数组中任意一个重复的数字。

示例 1：

```
输入：
[2, 3, 1, 0, 2, 5, 3]
输出：2 或 3
```

限制：

2 <= n <= 100000

视频讲解直达：本题视频讲解

```java
public int findRepeatNumber(int[] nums) {
// 遍历数组
    for(int i = 0; i < nums.length; i++) {
  // 之所以用while，是因为交换之后，该位置的元素任然没有在正确的位置
        while(i != nums[i]){
            if(nums[i] == nums[nums[i]]){
                return nums[i];
            }
            // nums[i] 正确的位置在 nums[nums[i]]
            int k = nums[nums[i]];
            nums[nums[i]] = nums[i];
            nums[i] = k;
        }
    }
    return -1;
}
```

剑指 Offer 04. 二维数组中的查找

问题描述

在一个 n * m 的二维数组中，每一行都按照从左到右递增的顺序排序，每一列都按照从上到下递增的顺序排序。请完成一个高效的函数，输入这样的一个二维数组和一个整数，判断数组中是否含有该整数。

示例 1：

```
现有矩阵 matrix 如下：

[
  [1,   4,  7, 11, 15],
  [2,   5,  8, 12, 19],
  [3,   6,  9, 16, 22],
  [10, 13, 14, 17, 24],
  [18, 21, 23, 26, 30]
]

给定 target = 5，返回 true。
给定 target = 20，返回 false。
```

限制：

0 <= n <= 1000

0 <= m <= 1000

视频讲解直达： [本题视频讲解]

```
        if(matrix == null || matrix.length <= 0 || matrix[0].length <=
0){
            return false;
        }
        // 这里 cols 表示多少行的意思（但是有人说 cols 是表示列，那我可能记错
了）
        int cols = matrix.length;
    // rows 表示多少列的意思
        int rows = matrix[0].length;
```

```
        // 左下角的位置
        int col = cols - 1;
        int row = 0;
        // 向上向右走的过程中不能出界
        while(col >= 0 && row < rows){
            if(target > matrix[col][row]){
                row++;
            } else if(target < matrix[col][row]){
                col--;
            } else {
                return true;
            }
        }
        return false;
    }
```

剑指 Offer 05. 替换空格

问题描述

请实现一个函数，把字符串 `s` 中的每个空格替换成"%20"。

示例 1：

```
输入：s = "We are happy."
输出："We%20are%20happy."
```

限制：

0 <= s 的长度 <= 10000

视频讲解直达： 本题视频讲解

```
    // Java中的常规做法
    // public String replaceSpace(String s) {
    //     // 常规
    //     StringBuilder builder = new StringBuilder();
    //     for(int i = 0; i < s.length(); i++){
```

```
    //          if(s.charAt(i) == ' '){
    //              builder.append("%20");
    //          } else {
    //              builder.append(s.charAt(i));
    //          }
    //      }
    //      return builder.toString();
    // }

    //这个做法是模拟原地扩容的，但是Java并不支持扩容，所以我们只是联系一下代码怎么写
    public String replaceSpace(String s) {
        // 统计有多少空格
        int count = 0;
        for(int i = 0; i < s.length(); i++){
            if(s.charAt(i) == ' '){
                count ++;
            }
        }

        char[] res = new char[s.length() + count * 2];
        int k = res.length - 1;
        // 从右往左遍历
        for(int i = s.length() - 1; i >= 0; i--){
            // 从右往左移动字符与替换字符
            if(s.charAt(i) == ' '){
                res[k--] = '0';
                res[k--] = '2';
                res[k--] = '%';
            } else {
                res[k--] = s.charAt(i);
            }
        }
        return new String(res);
    }
```

剑指 Offer 06. 从尾到头打印链表

问题描述

输入一个链表的头节点，从尾到头反过来返回每个节点的值（用数组返回）。

示例 1：

```
输入：head = [1,3,2]
输出：[2,3,1]
```

限制：

0 <= 链表长度 <= 10000

视频讲解直达： 本题视频讲解

```java
    // 从右边=往左填充
    public int[] reversePrint(ListNode head) {
        if(head == null)
            return new int[0];
        // 统计链表节点个数，方便创建数组
        int count = 0;
        ListNode temp = head;
        while(temp != null){
            count++;
            temp = temp.next;
        }
        int[] res = new int[count];
        int k = count - 1;
        // 从由往左填充数组
        while(head != null){
            res[k--] = head.val;
            head = head.next;
        }

        return res;
    }
```

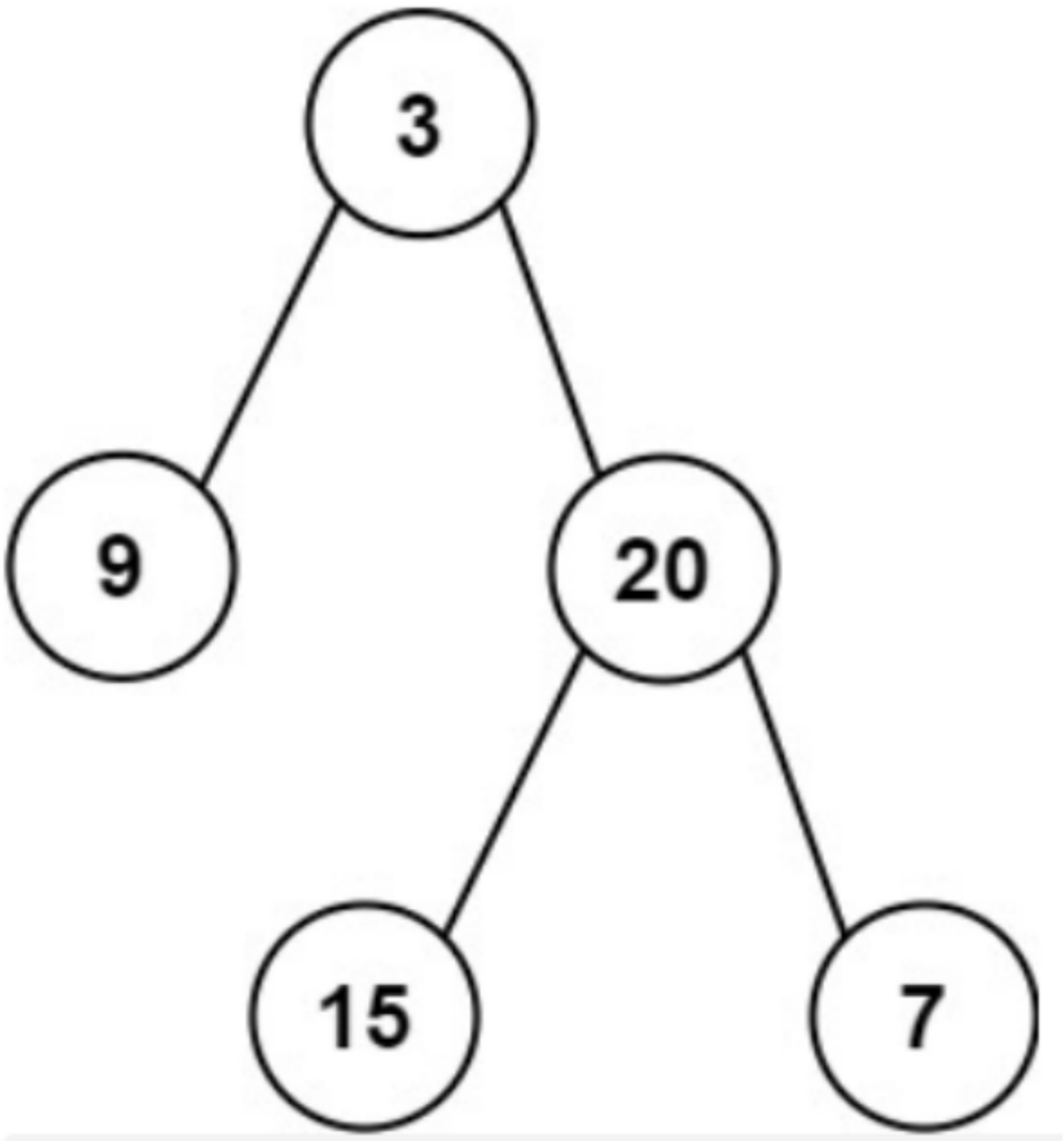
剑指 Offer 07. 重建二叉树

问题描述

输入某二叉树的前序遍历和中序遍历的结果，请构建该二叉树并返回其根节点。

假设输入的前序遍历和中序遍历的结果中都不含重复的数字。

示例 1：




```
Input: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]
Output: [3,9,20,null,null,15,7]
```

示例 2:

```
Input: preorder = [-1], inorder = [-1]
Output: [-1]
```

限制：

0 <= 节点个数 <= 5000

视频讲解直达： [本题视频讲解]

```
    Map< Integer, Integer > map = new HashMap();
    public TreeNode buildTree(int[] preorder, int[] inorder) {
        if(preorder == null || preorder.length <= 0){
            return null;
        }
        // 简历中序遍历数组的映射（就是为了快速求出某个元素的下标）
        for(int i = 0; i < inorder.length; i++) {
            map.put(inorder[i], i);
        }

        TreeNode root = f(preorder, 0, preorder.length - 1, inorder,
0, inorder.length - 1);

        return root;
    }

    TreeNode f(int[] preorder, int l1, int r1, int[] inorder, int l2,
int r2) {
        // 前序遍历或者中序遍历为空时，表示这棵树不存在，直接返回 null
        if( l1 > r1 || l2 > r2){
            return null;
        }
        // 根节点
        TreeNode root = new TreeNode(preorder[l1]);
        // 根节点在中序遍历中的下标
        int i = map.get(preorder[l1]);
        // 递归求解
        root.left = f(preorder, l1 + 1, l1 + (i - l2), inorder, l2, i
- 1);
        root.right = f(preorder, l1 + (i - l2) + 1, r1, inorder, i +
1, r2);

        return root;
    }
```

剑指 Offer 09. 用两个栈实现队列

问题描述

用两个栈实现一个队列。队列的声明如下，请实现它的两个函数 appendTail 和 deleteHead ，分别完成在队列尾部插入整数和在队列头部删除整数的功能。(若队列中没有元素，deleteHead 操作返回 -1)

示例 1:

```
输入：
["CQueue","appendTail","deleteHead","deleteHead","deleteHead"]
[[],[3],[],[],[]]
输出：[null,null,3,-1,-1]
```

示例 2:

```
输入：
["CQueue","deleteHead","appendTail","appendTail","deleteHead","deleteHead"]
[[],[],[5],[2],[],[]]
输出：[null,-1,null,null,5,2]
```

限制：

- `1 <= values <= 10000`
- 最多会对`appendTail、deleteHead`进行`10000` 次调用

视频讲解直达： [本题视频讲解]

```
class CQueue {
    Stack< Integer > stack1;
    Stack< Integer > stack2;
    public CQueue() {
        stack1 = new Stack<>();
        stack2 = new Stack<>();

    }
```

```
    public void appendTail(int value) {
        stack1.push(value);

    }

    public int deleteHead() {
        if(!stack2.isEmpty()){
            return stack2.pop();
        }

        if(!stack1.isEmpty()){
            while(!stack1.isEmpty()){
                stack2.push(stack1.pop());
            }

            return stack2.pop();
        }

        return -1;

    }
}
```

剑指 Offer 10- I. 斐波那契数列

问题描述

写一个函数，输入 n ，求斐波那契（Fibonacci）数列的第 n 项（即 F(N)）。斐波那契数列的定义如下：

$$F(0) = 0, F(1) = 1; F(N) = F(N-1) + F(N-2), \text{其中} N > 1.$$

斐波那契数列由 0 和 1 开始，之后的斐波那契数就是由之前的两数相加而得出。

答案需要取模 1e9+7（1000000007），如计算初始结果为：1000000008，请返回 1。

示例 1：

```
输入：n = 2
输出：1
```

示例 2:

```
输入：n = 5
输出：5
```

限制：

- `0 <= n <= 100`

视频讲解直达： [本题视频讲解]

```
    public int fib(int n) {
        if( n <= 1){
            return n;
        }

        int a = 0;
        int b = 1;
        int c = 0;;

        for(int i = 2; i <= n; i++){
            c = (a + b) % 1000000007;
          // a 原本指向的值不会在用到，所以让 a 保存 b，b 保存 c，之后重新计算
c，如此循环
            a = b;
            b = c;
        }

        return c;
    }
```

剑指 Offer 10- II. 青蛙跳台阶问题

问题描述

一只青蛙一次可以跳上1级台阶，也可以跳上2级台阶。求该青蛙跳上一个 n 级的台阶总共有多少种跳法。

答案需要取模 1e9+7（1000000007），如计算初始结果为：1000000008，请返回 1。

示例 1:

```
输入：n = 2
输出：2
```

示例 2:

```
输入：n = 7
输出：21
```

示例 3:

```
输入：n = 0
输出：1
```

限制：

- `0 <= n <= 100`

视频讲解直达： 本题视频讲解

```
    // 如果看不懂代码为啥这样写，可以参考上一题
  public int numWays(int n) {
        //递归公示 f(n) = f(n - 1) + f(n - 2);
        if( n <= 1)
            return 1;
        int a = 1;
        int b = 1;
        int c = 0;
        for(int i = 2; i <= n; i++){
            c = (a + b) % 1000000007;
```

```
            a = b;
            b = c;
        }

        return c;
    }
```

剑指 Offer 11. 旋转数组的最小数字

问题描述

把一个数组最开始的若干个元素搬到数组的末尾，我们称之为数组的旋转。

给你一个可能存在 重复 元素值的数组 numbers ，它原来是一个升序排列的数组，并按上述情形进行了一次旋转。请返回旋转数组的最小元素。例如，数组 [3,4,5,1,2] 为 [1,2,3,4,5] 的一次旋转，该数组的最小值为 1。

注意，数组 [a[0], a[1], a[2], ..., a[n-1]] 旋转一次 的结果为数组 [a[n-1], a[0], a[1], a[2], ..., a[n-2]] 。

示例 1：

```
输入：numbers = [3,4,5,1,2]
输出：1
```

示例 2:

```
输入：numbers = [2,2,2,0,1]
输出：0
```

限制：

- `0 <= n <= 100`
- `1 <= n <= 5000`
- `-5000 <= numbers[i] <= 5000`
- `numbers` 原来是一个升序排序的数组，并进行了 `1` 至 `n` 次旋转

视频讲解直达： [本题视频讲解]

```java
public int minArray(int[] numbers) {
    // O (n)
    // logN O1
    int l = 0;
    int r = numbers.length - 1;
    while(l < r) {
        if(numbers[l] < numbers[r]){
            return numbers[l];
        }

        int mid = (r + l) / 2;
        if(numbers[mid] > numbers[l]){
            l = mid + 1;
        } else if(numbers[mid] < numbers[l]){
            r = mid;
        } else {
            l++;
        }
    }

    return numbers[l];
}
```

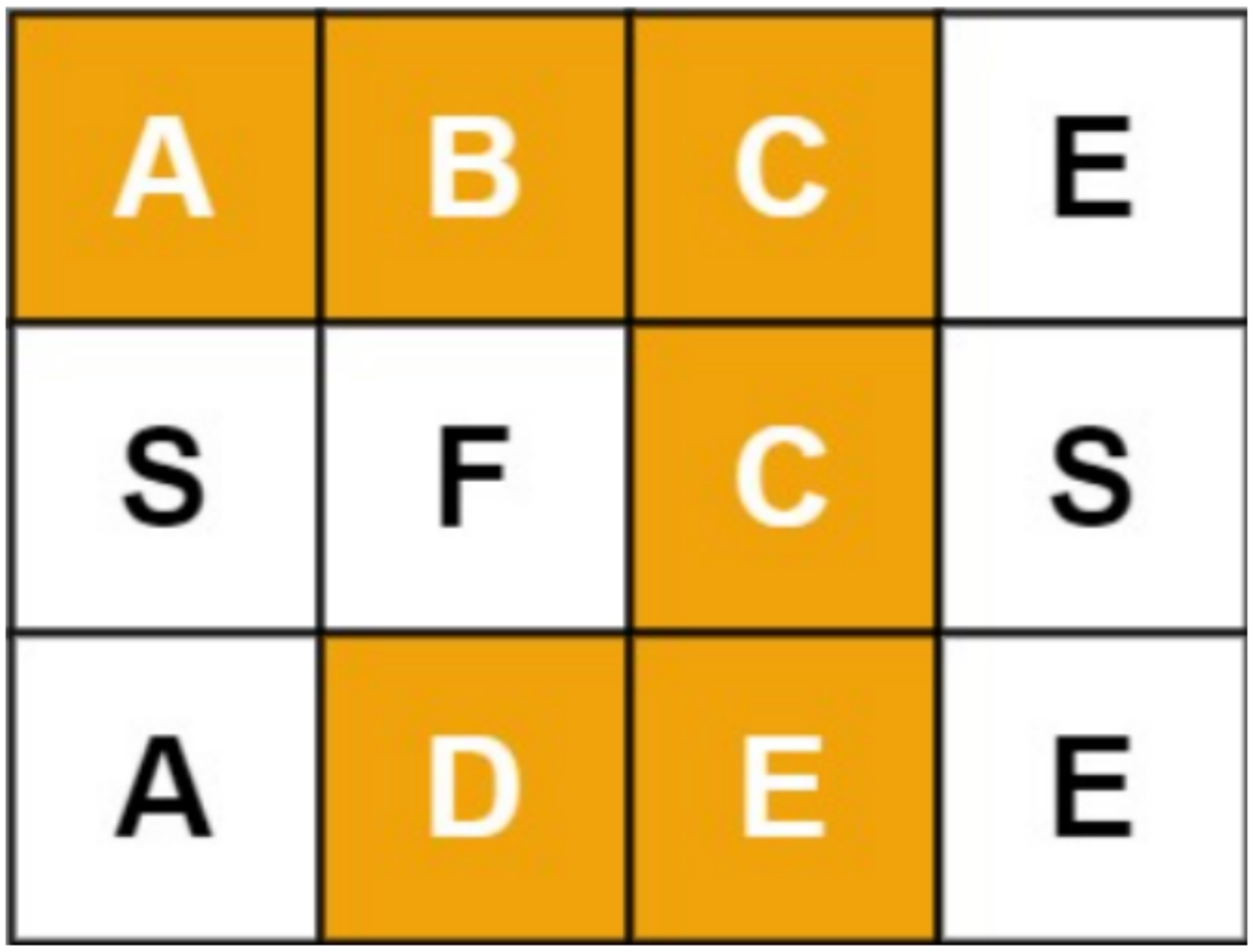
剑指 Offer 12. 矩阵中的路径

问题描述

给定一个 m x n 二维字符网格 board 和一个字符串单词 word 。如果 word 存在于网格中，返回 true ；否则，返回 false 。

单词必须按照字母顺序，通过相邻的单元格内的字母构成，其中“相邻”单元格是那些水平相邻或垂直相邻的单元格。同一个单元格内的字母不允许被重复使用。

例如，在下面的 3×4 的矩阵中包含单词 "ABCCED"（单词中的字母已标出）。



示例 1：

```
输入：board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]],
word = "ABCCED"
输出：true
```

示例 2:

```
输入：board = [["a","b"],["c","d"]], word = "abcd"
输出：false
```

限制：

- `m == board.length`
- `n = board[i].length`
- `1 <= m, n <= 6`
- `1 <= word.length <= 15`
- `board`和`word`仅由大小写英文字母组成

视频讲解直达： [本题视频讲解]

```
class Solution {
    int n;
    int m;
    int len;
```

```java
    boolean[][] visited;
    public boolean exist(char[][] board, String word) {
        this.n = board.length;
        this.m = board[0].length;
        this.len = word.length();
        visited = new boolean[n][m];

        for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++){
                if(dsf(board, i, j, word, 0)){
                    return true;
                }
            }
        }

        return false;
    }
    boolean dsf(char[][] board, int i, int j, String word, int k){
        // 判断是否越界，已经访问过或者不匹配
        if(i < 0 || i >= n || j < 0 || j >= m || board[i][j] !=
word.charAt(k)){
            return false;
        }

        if(k == len - 1){
            return true;
        }
        //visited[i][j] = true;
        board[i][j] = '\n';
        // 四个方向都搜索一下
        boolean res = dsf(board, i, j + 1, word, k + 1) ||
                      dsf(board, i + 1, j, word, k + 1) ||
                      dsf(board, i, j - 1, word, k + 1) ||
                      dsf(board, i - 1, j, word, k + 1);

        board[i][j] = word.charAt(k);
        return res;
        // 时间复杂度: mn * 3^len
    }
```

```
}
```

剑指 Offer 13. 机器人的运动范围

问题描述

地上有一个m行n列的方格，从坐标 [0,0] 到坐标 [m-1,n-1] 。一个机器人从坐标 [0, 0] 的格子开始移动，它每次可以向左、右、上、下移动一格（不能移动到方格外），也不能进入行坐标和列坐标的数位之和大于k的格子。例如，当k为18时，机器人能够进入方格 [35, 37] ，因为3+5+3+7=18。但它不能进入方格 [35, 38]，因为3+5+3+8=19。请问该机器人能够到达多少个格子？

示例 1：

```
输入：m = 2, n = 3, k = 1
输出：3
```

示例 2:

```
输入：m = 3, n = 1, k = 0
输出：1
```

限制：

- `1 <= n,m <= 100`
- `0 <= k <= 20`

视频讲解直达： [本题视频讲解]

```
class Solution {
    int m;
    int n;
    int k;
    boolean[][] visited;
    public int movingCount(int m, int n, int k) {
        this.m = m;
        this.n = n;
```

```
        this.k = k;
        visited = new boolean[m][n];

        return dfs(0, 0);
    }

    int dfs(int i, int j){
        // 是否在方格之内或者是否访问过或者是否是障碍物
        if(i < 0 || j < 0 || i >= m || j >= n || visited[i][j] || k <
sum(i) + sum(j)){
            return 0;
        }

        visited[i][j] = true;
        return 1 + dfs(i + 1, j) + dfs(i, j + 1);
    }

    int sum(int x){
        int res = 0;
        while(x != 0){
            res = res + x % 10;
            x = x / 10;
        }
        return res;
    }
}
```

剑指 Offer 14- I. 剪绳子

问题描述

给你一根长度为 n 的绳子，请把绳子剪成整数长度的 m 段（m、n都是整数，n>1并且m>1），每段绳子的长度记为 k[0],k[1]...k[m-1] 。请问 k[0]*k[1]*...*k[m-1] 可能的最大乘积是多少？例如，当绳子的长度是8时，我们把它剪成长度分别为2、3、3的三段，此时得到的最大乘积是18。

示例 1：

```
输入： 2
输出： 1
解释： 2 = 1 + 1, 1 × 1 = 1
```

示例 2:

```
输入： 10
输出： 36
解释： 10 = 3 + 3 + 4, 3 × 3 × 4 = 36
```

限制：

- `2 <= n <= 58`

视频讲解直达： [本题视频讲解]

```
    public int cuttingRope(int n) {
        if(n <= 2){
            return 1;
        }
        if(n == 3){
            return 2;
        }

        int res = n / 3;
        int mod = n % 3;

        if(mod == 0){
            return pow(3, res);
        } else if(mod == 1){
            return pow(3, res - 1) * 4;
        } else {
            return pow(3, res) * 2;
        }
    }
    // 这里多余了，其实直接调用Math.pow就可以了
    int pow(int a, int n){
        int sum = 1;
        for(int i = 1; i <= n; i ++){
```

```
            sum = sum * a;
        }
        return sum;
    }
```

剑指 Offer 14- II. 剪绳子 II

问题描述

给你一根长度为 n 的绳子，请把绳子剪成整数长度的 m 段（m、n都是整数，n>1并且m>1），每段绳子的长度记为 k[0],k[1]...k[m - 1] 。请问 k[0]*k[1]*...*k[m - 1] 可能的最大乘积是多少？例如，当绳子的长度是8时，我们把它剪成长度分别为2、3、3的三段，此时得到的最大乘积是18。

答案需要取模 1e9+7（1000000007），如计算初始结果为：1000000008，请返回 1。

示例 1：

```
输入： 2
输出： 1
解释： 2 = 1 + 1, 1 × 1 = 1
```

示例 2:

```
输入： 10
输出： 36
解释： 10 = 3 + 3 + 4, 3 × 3 × 4 = 36
```

限制：

- `2 <= n <= 1000`

视频讲解直达： 本题视频讲解

```
    public int cuttingRope(int n) {
        if(n <= 2){
            return 1;
        }
```

```
        if(n == 3){
            return 2;
        }

        int res = n / 3;
        int mod = n % 3;
        int p = 1000000007;

        if(mod == 0){
            return (int)pow(3, res);
        } else if(mod == 1){
            return (int)(pow(3, res - 1) * 4 % p);
        } else {
            return (int)(pow(3, res) * 2 % 1000000007);
        }
    }

    //求 a^n %p
    long pow(int a, int n){
        long res = 1;
        int p = 1000000007;
        for(int i= 1; i <= n; i++){
            res = (res * a) % p;
        }

        return res;
    }
```

剑指 Offer 15. 二进制中1的个数

问题描述

编写一个函数，输入是一个无符号整数（以二进制串的形式），返回其二进制表达式中数字位数为 '1' 的个数（也被称为 汉明重量).）。

提示：

- 请注意，在某些语言（如 Java）中，没有无符号整数类型。在这种情况下，输入和

输出都将被指定为有符号整数类型，并且不应影响您的实现，因为无论整数是有符号的还是无符号的，其内部的二进制表示形式都是相同的。

- 在 Java 中，编译器使用 二进制补码 记法来表示有符号整数。因此，在上面的 **示例 3** 中，输入表示有符号整数 `-3`。

示例 1:

```
输入: n = 11 (控制台输入 00000000000000000000000000001011)
输出: 3
解释: 输入的二进制串 00000000000000000000000000001011 中，共有三位为 '1'。
```

示例 2:

```
输入: n = 128 (控制台输入 00000000000000000000000010000000)
输出: 1
解释: 输入的二进制串 00000000000000000000000010000000 中，共有一位为 '1'。
```

示例 3:

```
输入: n = 4294967293 (控制台输入 11111111111111111111111111111101，部分语言中 n = -3)
输出: 31
解释: 输入的二进制串 11111111111111111111111111111101 中，共有 31 位为 '1'。
```

提示:

- `输入必须是长度为 32 的 二进制串 。`

视频讲解直达： 本题视频讲解

```java
    public int hammingWeight(int n) {
        int sum = 0;
        while(n != 0){
          // n & (n - 1)可以消除n最右边的一个 1 (二进制表示)
            n = n & (n - 1);
            sum ++;
        }
        return sum;
    }
```

剑指 Offer 16. 数值的整数次方

问题描述

实现 pow(*x*, *n*) ，即计算 x 的 n 次幂函数（即，xn）。不得使用库函数，同时不需要考虑大数问题。

示例 1:

```
输入: x = 2.00000, n = 10
输出: 1024.00000
```

示例 2:

```
输入: x = 2.10000, n = 3
输出: 9.26100
```

示例 3:

```
输入: x = 2.00000, n = -2
输出: 0.25000
解释: 2-2 = 1/22 = 1/4 = 0.25
```

提示:

- `-100.0 < x < 100.0`
- `-231 <= n <= 231-1`
- `-104 <= xn <= 104`

视频讲解直达： 本题视频讲解

```
    public double myPow(double x, int n) {
        double res = 1;
        long y = n;// 不能用 int。
        if(n < 0){
            y = -y;
            x = 1 / x;
```

```
        }
        while(y > 0){
            if(y % 2 == 1){
                res = res * x;
            }

            x = x * x;
            y = y / 2;
        }
        return res;
    }
```

剑指 Offer 17. 打印从1到最大的n位

问题描述

输入数字 `n`，按顺序打印出从 1 到最大的 n 位十进制数。比如输入 3，则打印出 1、2、3 一直到最大的 3 位数 999。

示例 1：

```
输入：n = 1
输出：[1,2,3,4,5,6,7,8,9]
```

说明：

- 用返回一个整数列表来代替打印
- n 为正整数

视频讲解直达： 本题视频讲解

```java
public int[] printNumbers(int n) {
    int sum = (int)Math.pow(10, n);
    int[] res = new int[sum - 1];
    for(int i = 1; i <= sum - 1; i++){
        res[i - 1] = i;
    }

    return res;
}
```

剑指 Offer 18. 删除链表的节点

问题描述

给定单向链表的头指针和一个要删除的节点的值，定义一个函数删除该节点。

返回删除后的链表的头节点。

注意：此题对比原题有改动

示例 1：

```
输入：head = [4,5,1,9], val = 5
输出：[4,1,9]
解释：给定你链表中值为 5 的第二个节点，那么在调用了你的函数之后，该链表应变为 4 -> 1 -> 9.
```

示例 2:

```
输入：head = [4,5,1,9], val = 1
输出：[4,5,9]
解释：给定你链表中值为 1 的第三个节点，那么在调用了你的函数之后，该链表应变为 4 -> 5 -> 9.
```

说明：

- 题目保证链表中节点的值互不相同
- 若使用 C 或 C++ 语言，你不需要 `free` 或 `delete` 被删除的节点

视频讲解直达： [本题视频讲解]

```
    public ListNode deleteNode(ListNode head, int val) {
        if(head == null){
            return null;
        }

        if(head.val == val){
            return head.next;
        }

        ListNode temp = head.next;
        ListNode pre = head;

        while(temp != null){
            if(temp.val == val){
                pre.next = temp.next;
                return head;
            }
            temp = temp.next;
            pre = pre.next;
        }

        return head;
    }
```

剑指 Offer 19. 正则表达式匹配

问题描述

请实现一个函数用来匹配包含'. '和''*的正则表达式。模式中的字符'.'表示任意一个字符，而*"表示它前面的字符可以出现任意次（含0次）。在本题中，匹配是指字符串的所有字符匹配整个模式。例如，字符串"aaa"与模式"a.a"和"ab*ac*a"匹配，但与"aa.a"和"ab*a"均不匹配。

示例 1：

```
输入：
s = "aa"
p = "a"
输出：false
解释："a" 无法匹配 "aa" 整个字符串。
```

示例 2:

```
输入：
s = "aa"
p = "a*"
输出：true
解释：因为 '*' 代表可以匹配零个或多个前面的那一个元素，在这里前面的元素就是 'a'。
因此，字符串 "aa" 可被视为 'a' 重复了一次。
```

示例 3:

```
输入：
s = "ab"
p = ".*"
输出：true
解释：".*" 表示可匹配零个或多个（'*'）任意字符（'.'）。
```

示例 4:

```
输入：
s = "aab"
p = "c*a*b"
输出：true
解释：因为 '*' 表示零个或多个，这里 'c' 为 0 个，'a' 被重复一次。因此可以匹配字
符串 "aab"。
```

示例 5:

```
输入：
s = "mississippi"
p = "mis*is*p*."
输出：false
```

说明：

- `s` 可能为空，且只包含从 `a-z` 的小写字母。
- `p` 可能为空，且只包含从 `a-z` 的小写字母以及字符 `.` 和 `*`，无连续的 `'*'`。

视频讲解直达： [本题视频讲解]

```
class Solution {
    public boolean isMatch(String s, String p) {
        if(s == null || p == null){
            return true;
        }
        int n = s.length();
        int m = p.length();
        boolean[][] dp = new boolean[n+1][m+1];
        // 初始化
        dp[0][0] = true;
        for(int j = 2; j <= m; j++){
            if(p.charAt(j - 1) == '*'){
                dp[0][j] = dp[0][j - 2];
            }
        }

        for(int i = 1; i <= n; i++){
            for(int j = 1; j <= m; j++){
                // 当j不为 *
                if(p.charAt(j - 1) != '*'){
                    // 如果不匹配
                    if(p.charAt(j-1) == '.' || p.charAt(j-1) ==
s.charAt(i - 1)){
                        dp[i][j] = dp[i-1][j-1];
                    }
                } else {
                    // 第j-1个字符不匹配
                    if(p.charAt(j - 2) != s.charAt(i - 1) &&
p.charAt(j-2) != '.'){
                        dp[i][j] = dp[i][j-2];
                    } else {
```

```
                        dp[i][j] = dp[i][j-2] || dp[i][j-1] || dp[i-1]
[j];
                    }
                }
            }
        }

        return dp[n][m];
    }
}
```

剑指 Offer 20. 表示数值的字符串

问题描述

请实现一个函数用来判断字符串是否表示**数值**（包括整数和小数）。

数值（按顺序）可以分成以下几个部分：

1. 若干空格
2. 一个 **小数** 或者 **整数**
3. （可选）一个 `'e'` 或 `'E'` ，后面跟着一个 **整数**
4. 若干空格

   小数（按顺序）可以分成以下几个部分：

   1. （可选）一个符号字符（`'+'` 或 `'-'`）
   2. 下述格式之一：
      1. 至少一位数字，后面跟着一个点 `'.'`
      2. 至少一位数字，后面跟着一个点 `'.'` ，后面再跟着至少一位数字
      3. 一个点 `'.'` ，后面跟着至少一位数字

   整数（按顺序）可以分成以下几个部分：

   1. （可选）一个符号字符（`'+'` 或 `'-'`）
   2. 至少一位数字

部分**数值**列举如下：

- `["+100", "5e2", "-123", "3.1416", "-1E-16", "0123"]`

部分**非数值**列举如下：

- `["12e", "1a3.14", "1.2.3", "+-5", "12e+5.4"]`

示例 1：

```
输入：s = "0"
输出：true
```

示例 2:

```
输入：s = "e"
输出：false
```

示例 3:

```
输入：s = "."
输出：false
```

示例 4:

```
输入：s = "     .1  "
输出：true
```

提示：

- `1 <= s.length <= 20`
- `s` 仅含英文字母（大写和小写），数字（`0-9`），加号 `'+'` ，减号 `'-'` ，空格 `' '` 或者点 `'.'` 。

视频讲解直达： [本题视频讲解]

```java
class Solution {
    public boolean isNumber(String s) {
        //有限状态机
        // 2.小数点 3.E/e 4. 数字字符 5. -+
        if(s == null || s.length() <= 0){
```

```
            return false;
        }
        char[] res = s.trim().toCharArray();
        if(res.length <= 0) return false;

        int n = res.length;

        boolean is_dot = false;
        boolean is_e_or_E = false;
        boolean is_num = false;
        for(int i = 0; i < n; i++){
            if(res[i] >= '0' && res[i] <= '9'){
                is_num = true;
            } else if(res[i] == '.'){
                //-+ 8.  8.8  .8
                // 前面：不能有重复的小数点，也不能出现 e/E
                if(is_dot || is_e_or_E){
                    return false;
                }
                is_dot = true;
            } else if(res[i] == 'e' || res[i] == 'E'){
                // 前面必须要有一个数字 || 前面不能出现重复的 e/E
                if(is_e_or_E || !is_num){
                    return false;
                }
                is_e_or_E = true;
                is_num =false;//11E+ 11E
            } else if(res[i] == '-' || res[i] == '+'){
                if(i!=0 && res[i-1] != 'e' && res[i-1] != 'E'){
                    return false;
                }
            } else {
                return false;
            }
        }

        return is_num;
    }
}
```

剑指 Offer 21. 调整数组顺序使奇数位于偶数前面

问题描述

输入一个整数数组，实现一个函数来调整该数组中数字的顺序，使得所有奇数在数组的前半部分，所有偶数在数组的后半部分。

示例 1：

```
输入：nums = [1,2,3,4]
输出：[1,3,2,4]
注：[3,1,2,4] 也是正确的答案之一。
```

提示：

- `0 <= nums.length <= 50000`
- `0 <= nums[i] <= 10000`

视频讲解直达： 本题视频讲解

```java
class Solution {
    public int[] exchange(int[] nums) {
        if(nums == null || nums.length == 0){
            return nums;
        }
        int left = 0;
        int right = nums.length - 1;
        while(left < right){
            while(left < right && nums[left] % 2 != 0) left++;
            while(left < right && nums[right] % 2 != 1) right--;

            int temp = nums[left];
            nums[left] = nums[right];
            nums[right] = temp;
        }

        return nums;
    }
```

```
}
```

剑指 Offer 22. 链表中倒数第k个节

问题描述

输入一个链表，输出该链表中倒数第k个节点。为了符合大多数人的习惯，本题从1开始计数，即链表的尾节点是倒数第1个节点。

例如，一个链表有 6 个节点，从头节点开始，它们的值依次是 1、2、3、4、5、6。这个链表的倒数第 3 个节点是值为 4 的节点。

示例 1：

```
给定一个链表：1->2->3->4->5, 和 k = 2.

返回链表 4->5.
```

视频讲解直达： 本题视频讲解

```
class Solution {
    public ListNode getKthFromEnd(ListNode head, int k) {
        if(head == null){
            return null;
        }

        ListNode fast = head, slow = head;
        for(int i = 0; i < k; i++){
            if(fast == null){
                return null;
            }
            fast = fast.next;
        }

        while(fast != null){
            fast = fast.next;
            slow = slow.next;
        }
```

```
        return slow;
    }
}
```

剑指 Offer 24. 反转链表

问题描述

定义一个函数，输入一个链表的头节点，反转该链表并输出反转后链表的头节点。

示例 1：

```
输入： 1->2->3->4->5->NULL
输出： 5->4->3->2->1->NULL
```

限制：

- `0 <= 节点个数 <= 5000`

视频讲解直达： 本题视频讲解

```
class Solution {
    // 原地反转
    public ListNode reverseList(ListNode head) {
        if(head == null || head.next == null){
            return head;
        }

        ListNode cur = head, pre = null;

        while(cur != null){
            ListNode temp = cur.next;
            cur.next = pre;
            pre = cur;
            cur = temp;
```

```
        }

        return pre;

        // 额外空间复杂度=1
    }

// 递归
    public ListNode reverseList(ListNode head) {
        if(head == null || head.next == null){
            return head;
        }
        // 反转子链表
        ListNode temp = reverseList(head.next);
        head.next.next = head;
        head.next = null;

        return temp;
    }
}
```

剑指 Offer 25. 合并两个排序的链表

问题描述

输入两个递增排序的链表，合并这两个链表并使新链表中的节点仍然是递增排序的。

示例 1：

```
输入：1->2->4, 1->3->4
输出：1->1->2->3->4->4
```

限制：

- `0 <= 链表长度 <= 1000`

视频讲解直达： 本题视频讲解

```java
class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        ListNode merge = new ListNode(0);
        ListNode temp = merge;

        while(l1 != null && l2 != null){
            if(l1.val < l2.val){

                temp.next = l1;
                l1 = l1.next;
            } else {
                temp.next = l2;
                l2 = l2.next;
            }

            temp = temp.next;
        }

        temp.next = l1 == null ? l2 : l1;

        return merge.next;
    }
}
```

剑指 Offer 26. 树的子结构

问题描述

输入两棵二叉树A和B，判断B是不是A的子结构。(约定空树不是任意一个树的子结构)

B是A的子结构， 即 A中有出现和B相同的结构和节点值。

例如:
给定的树 A:

```
     3
    / \
   4   5
  / \
 1   2
```

给定的树 B：

```
   4
  /
 1
```

返回 true，因为 B 与 A 的一个子树拥有相同的结构和节点值。

示例 1：

```
输入：A = [1,2,3], B = [3,1]
输出：false
```

示例 2：

```
输入：A = [3,4,5,1,2], B = [4,1]
输出：true
```

限制：

- `0 <= 节点个数 <= 100000`

视频讲解直达： [本题视频讲解]

```
class Solution {
    public boolean isSubStructure(TreeNode A, TreeNode B) {
        if(A == null || B == null){
            return false;
        }

        if(isSubTree(A, B)){
            return true;
        }
```

```
        if(isSubStructure(A.left, B) || isSubStructure(A.right, B)){
            return true;
        }

        return false;
    }

    public boolean isSubTree(TreeNode Ta, TreeNode Tb){
        if(Tb == null){
            return true;
        }

        if(Ta == null){
            return false;
        }

        if(Ta.val != Tb.val){
            return false;
        }

        return isSubTree(Ta.left, Tb.left) &&
               isSubTree(Ta.right, Tb.right);
    }
}
```

剑指 Offer 27. 二叉树的镜像

问题描述

请完成一个函数，输入一个二叉树，该函数输出它的镜像。

例如输入：

```
      4
    /   \
   2     7
  / \   / \
 1   3 6   9
```

镜像输出：

```
      4
    /   \
   7     2
  / \   / \
 9   6 3   1
```

返回 true，因为 B 与 A 的一个子树拥有相同的结构和节点值。

示例 1：

```
输入：root = [4,2,7,1,3,6,9]
输出：[4,7,2,9,6,3,1]
```

限制：

- `0 <= 节点个数 <= 1000`

视频讲解直达： [本题视频讲解]

```
class Solution {
    public TreeNode mirrorTree(TreeNode root) {
        if(root == null || (root.left == null && root.right == null)){
            return root;
        }

        TreeNode left = mirrorTree(root.left);
        TreeNode right = mirrorTree(root.right);

        root.left = right;
        root.right = left;
```

```
        return root;
    }
}
```

剑指 Offer 28. 对称的二叉树

问题描述

请实现一个函数，用来判断一棵二叉树是不是对称的。如果一棵二叉树和它的镜像一样，那么它是对称的。

例如，二叉树 [1,2,2,3,4,4,3] 是对称的。

```
    1
   / \
  2   2
 / \ / \
3  4 4  3
```

但是下面这个 [1,2,2,null,3,null,3] 则不是镜像对称的:

```
    1
   / \
  2   2
   \   \
    3   3
```

返回 true，因为 B 与 A 的一个子树拥有相同的结构和节点值。

示例 1:

```
输入: root = [1,2,2,3,4,4,3]
输出: true
```

示例 2:

```
输入：root = [1,2,2,null,3,null,3]
输出：false
```

限制：

- `0 <= 节点个数 <= 1000`

视频讲解直达： [本题视频讲解]

```
class Solution {
    public boolean isSymmetric(TreeNode root) {
        if(root == null || (root.left == null && root.right == null)){
            return true;
        }

        return f(root.left, root.right);
    }
    // 判断B是不是以A阶段为根节点的子树
    public boolean f(TreeNode A, TreeNode B){
        if(A == null && B == null){
            return true;
        }

        if(A == null || B == null){
            return false;
        }

        if(A.val != B.val){
            return false;
        }

        return f(A.left, B.right) && f(A.right, B.left);
    }
}
```

剑指 Offer 29. 顺时针打印矩阵

问题描述

输入一个矩阵，按照从外向里以顺时针的顺序依次打印出每一个数字。

示例 1：

```
输入：matrix = [[1,2,3],[4,5,6],[7,8,9]]
输出：[1,2,3,6,9,8,7,4,5]
```

示例 2：

```
输入：matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
输出：[1,2,3,4,8,12,11,10,9,5,6,7]
```

限制：

- `0 <= matrix.length <= 100`
- `0 <= matrix[i].length <= 100`

视频讲解直达： [本题视频讲解]

```
class Solution {
    public int[] spiralOrder(int[][] matrix) {
        if(matrix == null || matrix.length == 0 || matrix[0].length ==
0){
            return new int[0];
        }

        int l = 0, r = matrix[0].length - 1, t = 0, b = matrix.length
- 1;
        int[] res = new int[(r+1) * (b+1)];
        int k = 0;
        while(true){
            // 从左到右
            for(int i = t,j = l; j <= r; j++){
                res[k++] = matrix[i][j];
            }
            if(++t > b) break;
```

```
            // 从下往上
            for(int i = t, j = r; i <= b; i++){
                res[k++] = matrix[i][j];
            }
            if(l > --r) break;
            // 从右往左
            for(int i = b, j = r; j >= l; j--){
                res[k++] = matrix[i][j];
            }
            if(t > --b) break;
            // 从下往上
            for(int i = b, j = l; i >= t; i--){
                res[k++] = matrix[i][j];
            }
            if(++l > r) break;
        }
        return res;
    }
}
```

剑指 Offer 30. 包含min函数的栈

问题描述

定义栈的数据结构，请在该类型中实现一个能够得到栈的最小元素的 min 函数在该栈中，调用 min、push 及 pop 的时间复杂度都是 O(1)。

示例：

```
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.min();   --> 返回 -3.
minStack.pop();
minStack.top();      --> 返回 0.
minStack.min();   --> 返回 -2.
```

限制：

- 各函数的调用总次数不超过 20000 次

视频讲解直达： [本题视频讲解]

```
class MinStack {
    Stack<Integer> stack1;
    Stack<Integer> stack2;

    /** initialize your data structure here. */
    public MinStack() {
        this.stack1 = new Stack();
        this.stack2 = new Stack();
    }

    public void push(int x) {
        stack1.push(x);
        if(stack2.isEmpty() || x <= stack2.peek().intValue()){
            stack2.push(x);
        }
    }

    public void pop() {
        if(!stack1.isEmpty()){
            //Integer, 数值 > 127 I
            if(stack1.peek().intValue() == stack2.peek().intValue()){
                stack2.pop();
            }
            stack1.pop();
        }
    }

    public int top() {
        return stack1.peek();

    }

    public int min() {
```

```
        return stack2.peek();
    }
}
```

剑指 Offer 31. 栈的压入、弹出序列

问题描述

输入两个整数序列，第一个序列表示栈的压入顺序，请判断第二个序列是否为该栈的弹出顺序。假设压入栈的所有数字均不相等。例如，序列 {1,2,3,4,5} 是某栈的压栈序列，序列 {4,5,3,2,1} 是该压栈序列对应的一个弹出序列，但 {4,3,5,1,2} 就不可能是该压栈序列的弹出序列。

示例1：

```
输入：pushed = [1,2,3,4,5], popped = [4,5,3,2,1]
输出：true
解释：我们可以按以下顺序执行：
push(1), push(2), push(3), push(4), pop() -> 4,
push(5), pop() -> 5, pop() -> 3, pop() -> 2, pop() -> 1
```

示例2：

```
输入：pushed = [1,2,3,4,5], popped = [4,3,5,1,2]
输出：false
解释：1 不能在 2 之前弹出。
```

限制：

- `0 <= pushed.length == popped.length <= 1000`
- `0 <= pushed[i], popped[i] < 1000`
- `pushed` 是 `popped` 的排列。

视频讲解直达： 本题视频讲解

```
class Solution {
```

```
    public boolean validateStackSequences(int[] pushed, int[] popped)
{
        if(pushed == null || pushed.length <= 0){
            return true;
        }
        int k = 0;
        Stack<Integer> stack = new Stack();
        for(int i = 0; i < pushed.length; i++){
            stack.push(pushed[i]);
            while(!stack.isEmpty() && stack.peek() == popped[k]){
                stack.pop();
                k++;
            }
        }
        return stack.isEmpty();
    }
}
```

剑指 Offer 32 - I. 从上到下打印二叉树

问题描述

从上到下打印出二叉树的每个节点，同一层的节点按照从左到右的顺序打印。

例如:
给定二叉树: `[3,9,20,null,null,15,7]`,

```
    3
   / \
  9  20
    /  \
   15   7
```

返回:

```
[3,9,20,15,7]
```

提示：

- `节点总数 <= 1000`

视频讲解直达： [本题视频讲解]

```
class Solution {
    public int[] levelOrder(TreeNode root) {
        if(root == null){
            return new int[0];
        }

        Queue<TreeNode> queue = new LinkedList<>();
        List<Integer> res = new ArrayList<>();

        queue.add(root);
        while(!queue.isEmpty()){
            TreeNode t = queue.poll();
            res.add(t.val);
            if(t.left != null) queue.add(t.left);
            if(t.right != null) queue.add(t.right);
        }

        int[] arr = new int[res.size()];
        for(int i = 0; i < res.size(); i++){
            arr[i] = res.get(i);
        }

        return arr;

    }
}
```

剑指 Offer 32 - II. 从上到下打印二叉树

问题描述

从上到下按层打印二叉树，同一层的节点按从左到右的顺序打印，每一层打印到一行。

例如:
给定二叉树: `[3,9,20,null,null,15,7]`,

```
  3
    / \
   9  20
     /  \
    15   7
```

返回其层次遍历结果:

```
[
  [3],
  [9,20],
  [15,7]
]
```

提示:

- `节点总数 <= 1000`

视频讲解直达： [本题视频讲解]

```java
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        if(root == null){
            return new ArrayList<>();
        }

        Queue<TreeNode> queue = new LinkedList<>();
        List<List<Integer>> res = new ArrayList<>();

        queue.add(root);
        while(!queue.isEmpty()){
```

```
            int k = queue.size();
            List<Integer> tmp = new ArrayList<>();
            for(int i = 0; i < k; i++){
                TreeNode t = queue.poll();
                tmp.add(t.val);
                if(t.left != null) queue.add(t.left);
                if(t.right != null) queue.add(t.right);
            }
            res.add(tmp);
        }

        return res;
    }
}
```

剑指 Offer 32 - III. 从上到下打印二叉树

问题描述

请实现一个函数按照之字形顺序打印二叉树，即第一行按照从左到右的顺序打印，第二层按照从右到左的顺序打印，第三行再按照从左到右的顺序打印，其他行以此类推。

例如:
给定二叉树: `[3,9,20,null,null,15,7]`,

```
    3
   / \
  9  20
    /  \
   15   7
```

返回其层次遍历结果:

```
[
  [3],
  [20,9],
  [15,7]
]
```

提示：

1. `节点总数 <= 1000`

视频讲解直达： [本题视频讲解]

```java
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        if(root == null){
            return new ArrayList<>();
        }

        Queue<TreeNode> queue = new LinkedList<>();
        List<List<Integer>> res = new ArrayList<>();
        int sum = 1;
        queue.add(root);
        while(!queue.isEmpty()){
            int k = queue.size();
            LinkedList<Integer> tmp = new LinkedList<>();
            for(int i = 0; i < k; i++){
                TreeNode t = queue.poll();
                if(sum % 2 == 1){
                    tmp.add(t.val);
                } else {
                    tmp.addFirst(t.val);
                }

                if(t.left != null) queue.add(t.left);
                if(t.right != null) queue.add(t.right);
            }
            res.add(tmp);
            sum ++;
        }
```

```
        return res;
    }
}
```

剑指 Offer 33. 二叉搜索树的后序遍历序列

问题描述

输入一个整数数组，判断该数组是不是某二叉搜索树的后序遍历结果。如果是则返回 true，否则返回 false。假设输入的数组的任意两个数字都互不相同。

参考以下这颗二叉搜索树：

```
     5
    / \
   2   6
  / \
 1   3
```

示例1：

```
输入: [1,6,3,2,5]
输出: false
```

示例2：

```
输入: [1,3,2,6,5]
输出: true
```

- **提示：**
  1. `数组长度 <= 1000`

视频讲解直达： [本题视频讲解]

```java
class Solution {
    public boolean verifyPostorder(int[] postorder) {
        if(postorder == null){
            return true;
        }

        return f(postorder, 0, postorder.length - 1);
    }

    boolean f(int[] postorder, int i, int j){
        if(i >= j){
            return true;
        }

        int root = postorder[j];
        int p = i;
        // 获取第一个大于或者等于 root 等元素的位置
        while(postorder[p] < root) p++;
        // 判断 p ~ j -1 这个范围是否存在小于root的元素
        for(int k = p; k < j; k++){
            if(postorder[k] < root){
                return false;
            }
        }

        return f(postorder, i, p - 1) && f(postorder, p, j - 1);
    }
}
```

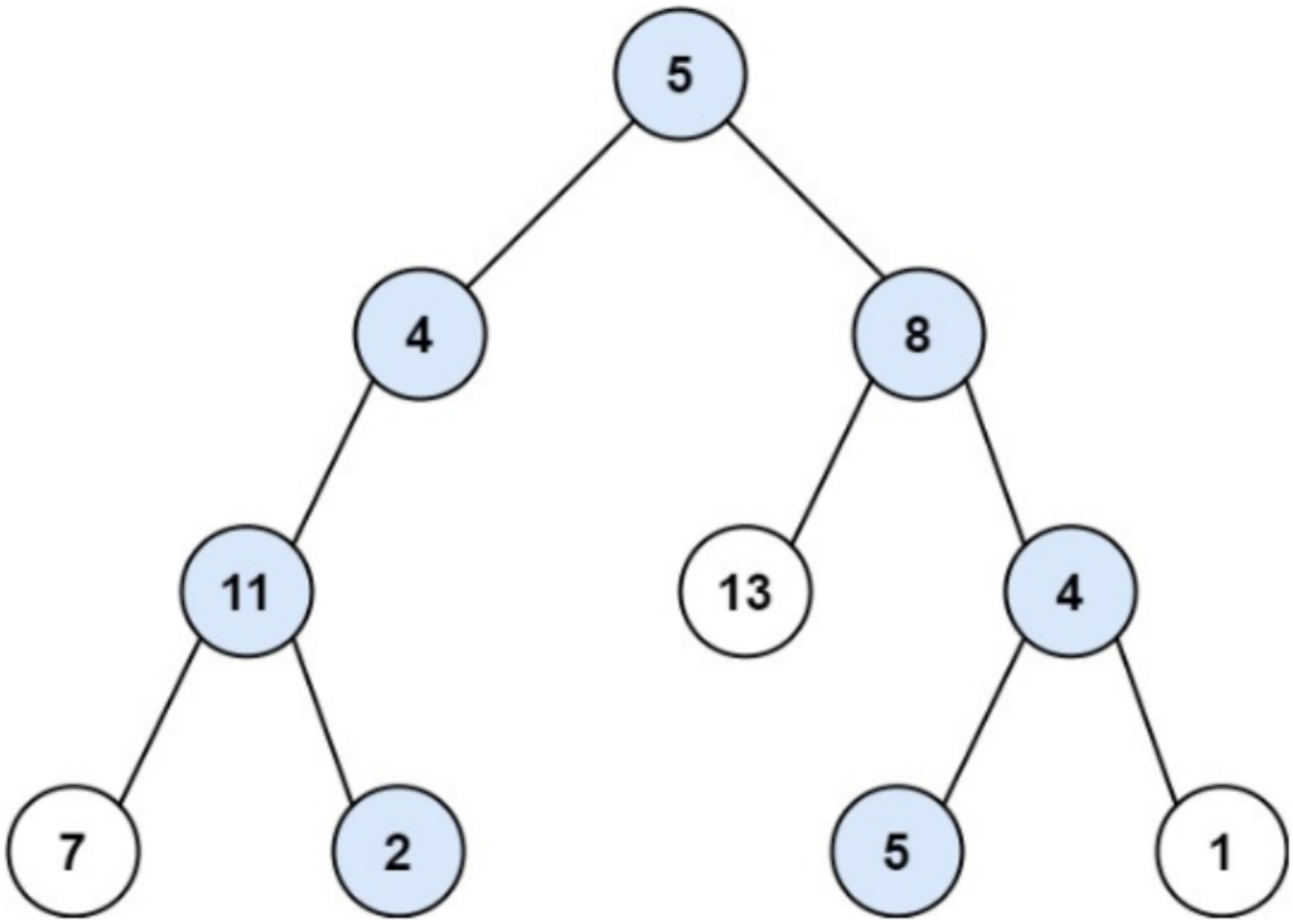
剑指 Offer 34. 二叉树中和为某一值的路径

问题描述

给你二叉树的根节点 root 和一个整数目标和 targetSum ， 找出所有 从根节点到叶子节点 路径总和等于给定目标和的路径。

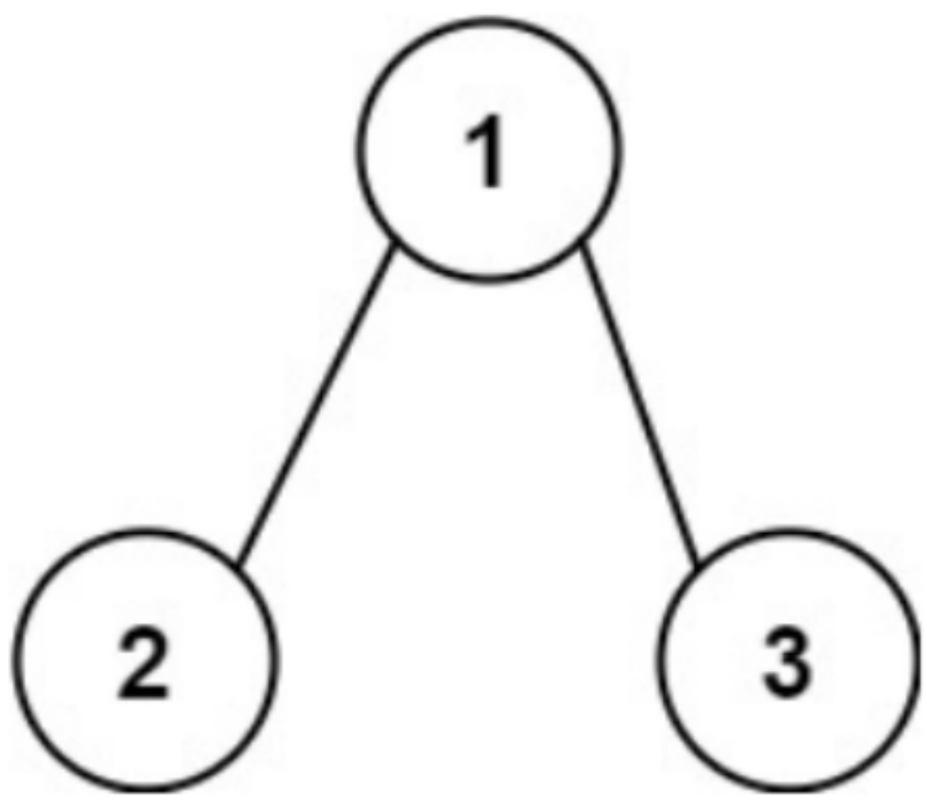
叶子节点 是指没有子节点的节点。

示例1：




```
输入：root = [5,4,8,11,null,13,4,7,2,null,null,5,1], targetSum = 22
输出：[[5,4,11,2],[5,8,4,5]]
```

示例2：




```
输入：root = [1,2,3], targetSum = 5
输出：[]
```

示例 3：

```
输入：root = [1,2], targetSum = 0
输出：[]
```

- **提示：**
  - 树中节点总数在范围 `[0, 5000]` 内
  - `-1000 <= Node.val <= 1000`
  - `-1000 <= targetSum <= 1000`

视频讲解直达： [本题视频讲解]

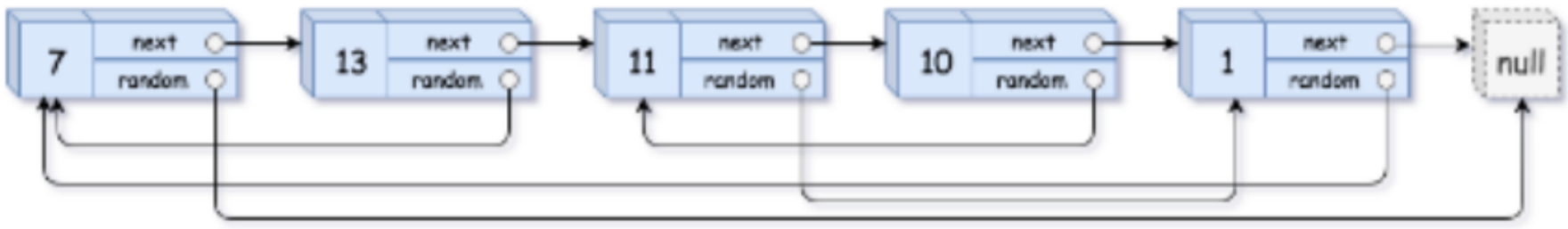
```java
class Solution {
    List<List<Integer>> res;
    List<Integer> tmp;
    public List<List<Integer>> pathSum(TreeNode root, int target) {
        res = new ArrayList<>();
        tmp = new ArrayList<>();
        dsf(root, target);
        return res;
    }
    void dsf(TreeNode root, int target){
        if(root == null){
            return;
        }

        tmp.add(root.val);
        target = target - root.val;
        if(root.left == null && root.right == null && target == 0){
            res.add(new ArrayList<>(tmp));
        }
        dsf(root.left, target);
        dsf(root.right, target);

        tmp.remove(tmp.size() - 1);

    }
}
```

剑指 Offer 35. 复杂链表的复制

问题描述

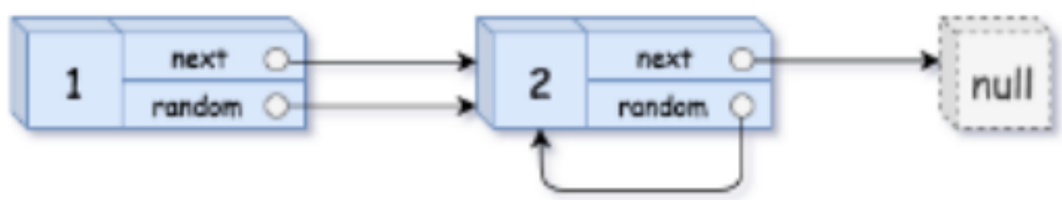
请实现 copyRandomList 函数，复制一个复杂链表。在复杂链表中，每个节点除了有一个 next 指针指向下一个节点，还有一个 random 指针指向链表中的任意节点或者 null。

示例1：



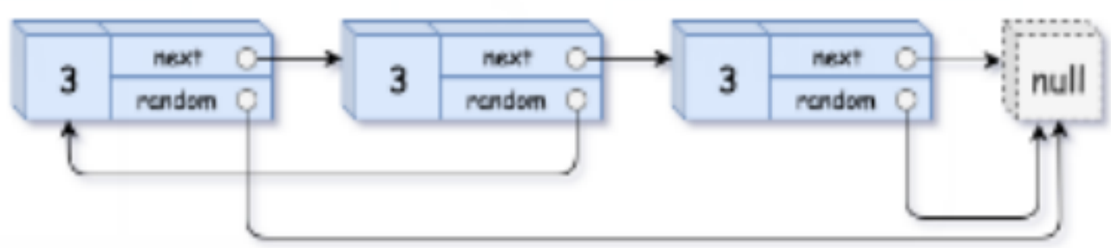

```
输入：head = [[7,null],[13,0],[11,4],[10,2],[1,0]]
输出：[[7,null],[13,0],[11,4],[10,2],[1,0]]
```

示例2：




```
输入：head = [[1,1],[2,1]]
输出：[[1,1],[2,1]]
```

示例 3:




```
输入：head = [[3,null],[3,0],[3,null]]
输出：[[3,null],[3,0],[3,null]]
```

示例 4：

```
输入：head = []
输出：[]
解释：给定的链表为空（空指针），因此返回 null。
```

- **提示：**
  - `-10000 <= Node.val <= 10000`
  - `Node.random` 为空（null）或指向链表中的节点。
  - 节点数目不超过 1000 。

视频讲解直达： [本题视频讲解]

```
class Solution {
    public Node copyRandomList(Node head) {
        if(head == null){
            return null;
        }
        // 复制链表节点
        Node cur = head;
        while(cur != null){
            Node next = cur.next;
            cur.next = new Node(cur.val);
            cur.next.next = next;
            cur = next;
        }

        // 复制随机节点
        cur = head;
        while(cur != null){
            Node curNew = cur.next;
            curNew.random = cur.random == null ? null :
cur.random.next;
            cur = cur.next.next;
        }

        // 拆分,比如把 A->A1->B->B1->C->C1拆分成 A->B->C和A1->B1->C1
        Node headNew = head.next;
```

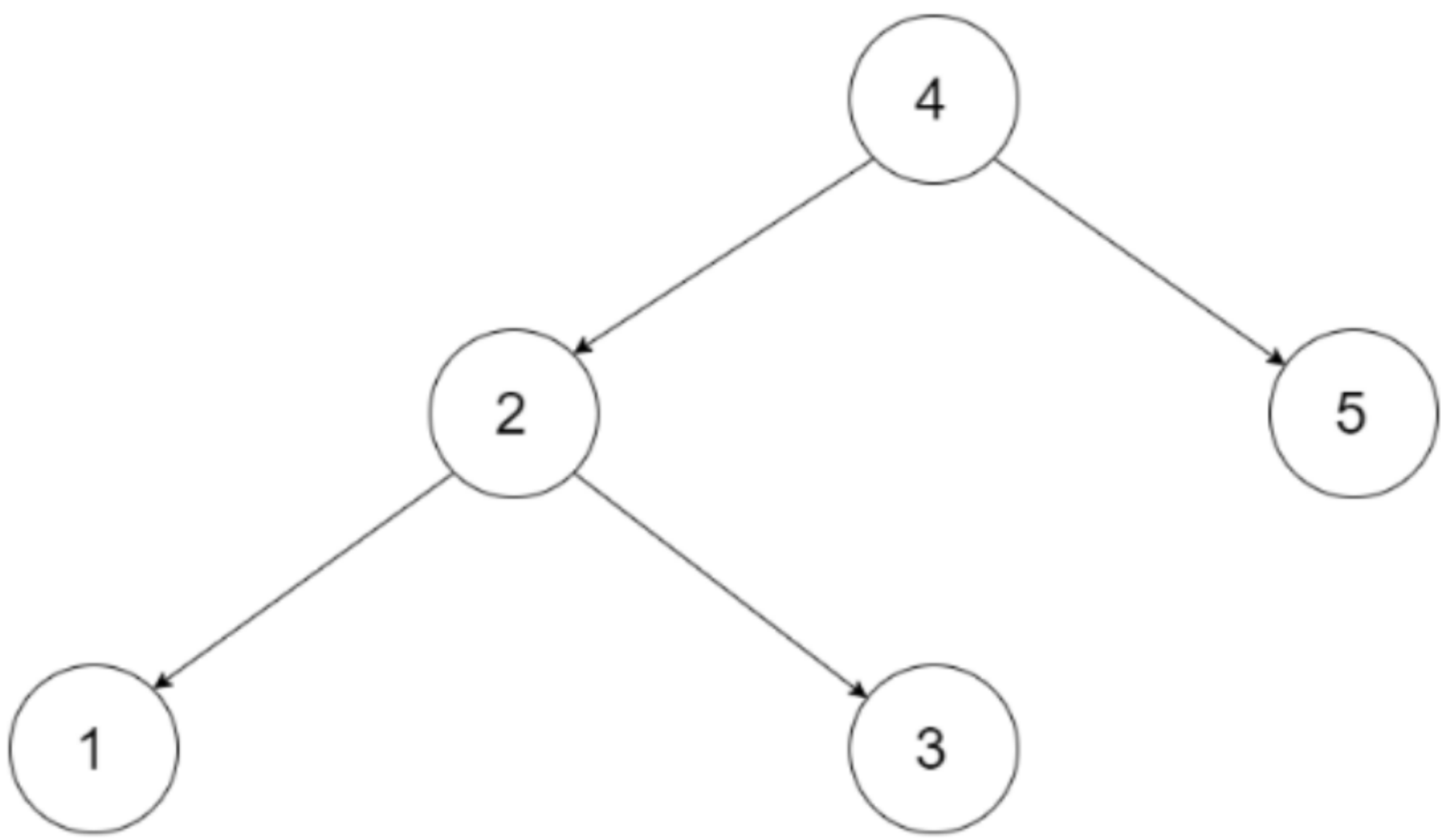
```
        cur = head;
        Node curNew = head.next;
        while(cur != null){
            cur.next = cur.next.next;
            cur = cur.next;
            curNew.next = cur == null ? null : cur.next;
            curNew = curNew.next;
        }

        return headNew;
    }
}
```

剑指 Offer 36. 二叉搜索树与双向链表

问题描述

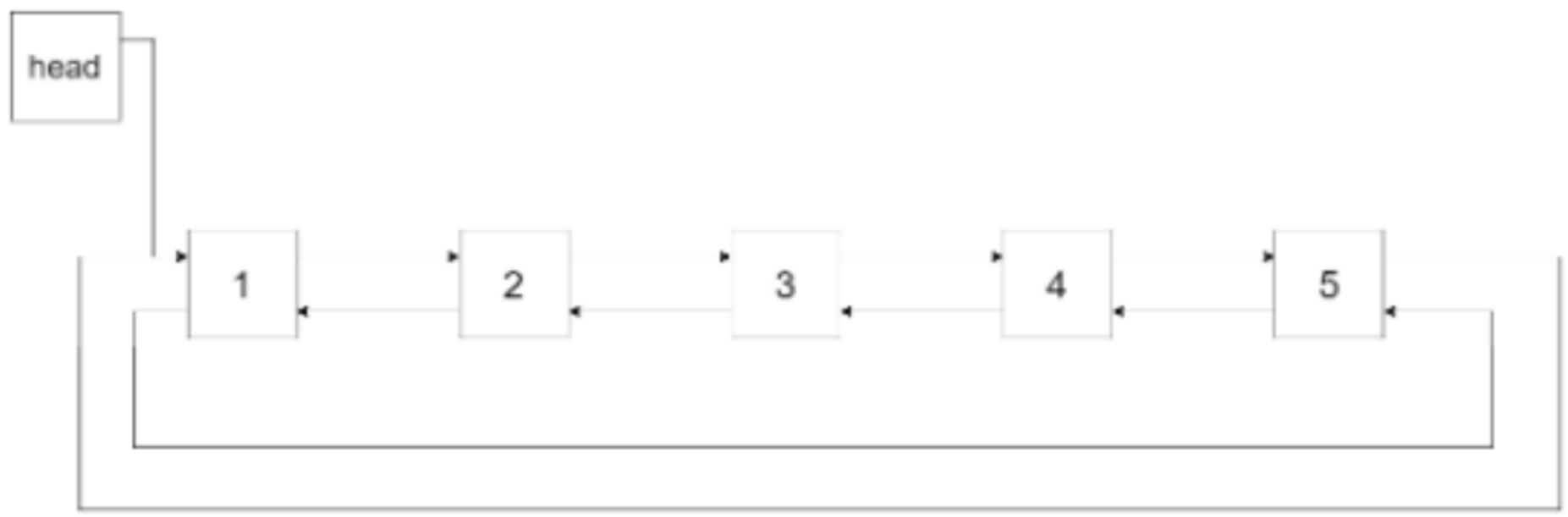
输入一棵二叉搜索树，将该二叉搜索树转换成一个排序的循环双向链表。要求不能创建任何新的节点，只能调整树中节点指针的指向。

为了让您更好地理解问题，以下面的二叉搜索树为例：



我们希望将这个二叉搜索树转化为双向循环链表。链表中的每个节点都有一个前驱和后继指针。对于双向循环链表，第一个节点的前驱是最后一个节点，最后一个节点的后继是第一个节点。

下图展示了上面的二叉搜索树转化成的链表。“head” 表示指向链表中有最小元素的节点。



特别地，我们希望可以就地完成转换操作。当转化完成以后，树中节点的左指针需要指向前驱，树中节点的右指针需要指向后继。还需要返回链表中的第一个节点的指针。

视频讲解直达： [本题视频讲解]

```
class Solution {
    public Node treeToDoublyList(Node root) {
        if(root == null){
            return null;
        }

        Queue<Node> queue = new LinkedList<>();
        inOrder(root, queue);
        Node head = queue.poll();
        Node pre = head;

        while(!queue.isEmpty()){
            Node cur = queue.poll();
            pre.right = cur;
            cur.left = pre;
            pre = cur;
        }
```

```
        pre.right = head;
        head.left = pre;

        return head;
    }

    void inOrder(Node root, Queue<Node> queue){
        if(root == null){
            return;
        }
        inOrder(root.left, queue);
        queue.add(root);
        inOrder(root.right, queue);
    }
}
```

剑指 Offer 37. 序列化二叉树

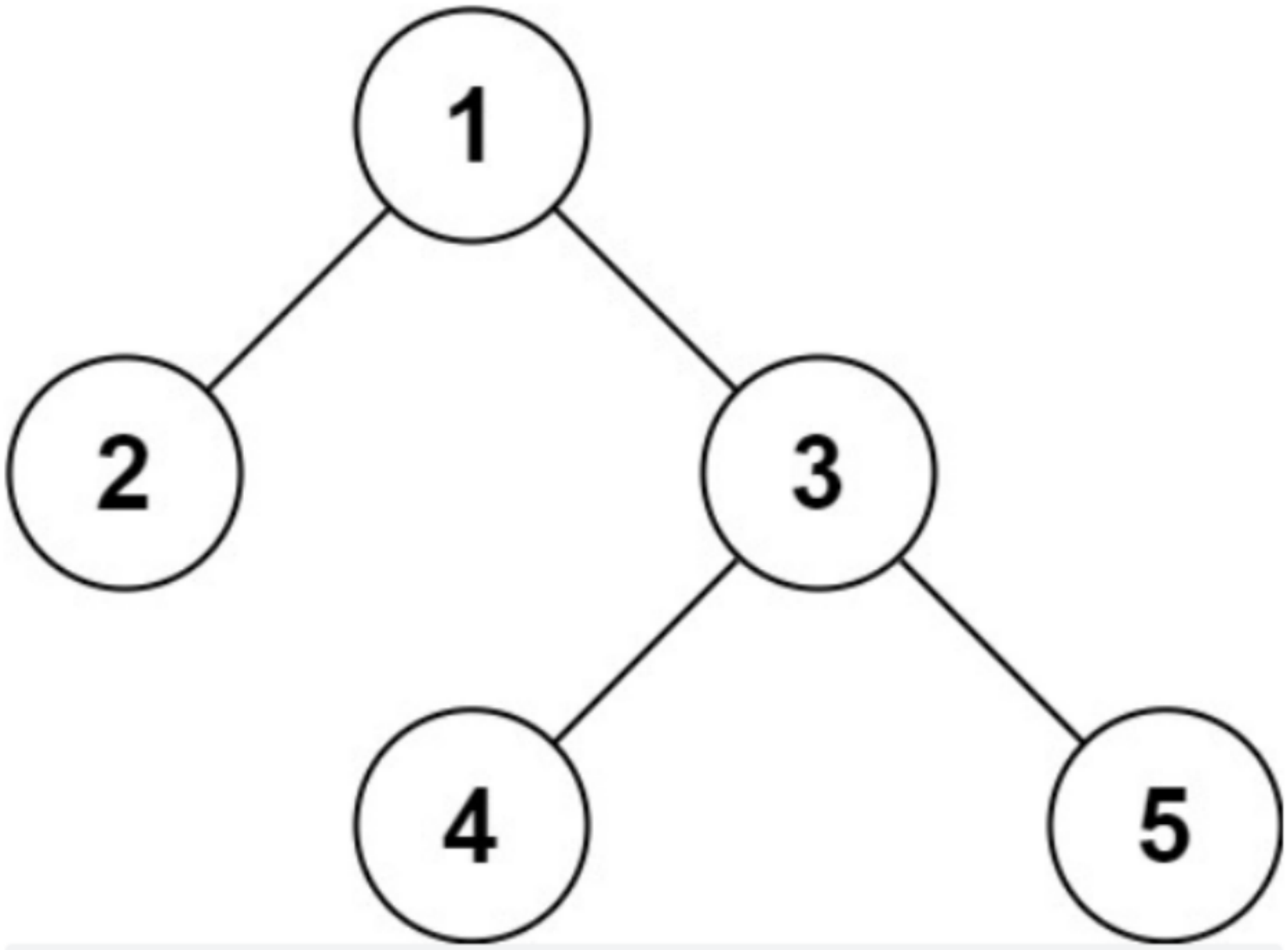
问题描述

请实现两个函数，分别用来序列化和反序列化二叉树。

你需要设计一个算法来实现二叉树的序列化与反序列化。这里不限定你的序列 / 反序列化算法执行逻辑，你只需要保证一个二叉树可以被序列化为一个字符串并且将这个字符串反序列化为原始的树结构。

提示：输入输出格式与 LeetCode 目前使用的方式一致，详情请参阅 LeetCode 序列化二叉树的格式。你并非必须采取这种方式，你也可以采用其他的方法解决这个问题。

示例：




```
输入：root = [1,2,3,null,null,4,5]
输出：[1,2,3,null,null,4,5]
```

视频讲解直达： [本题视频讲解]

```
public class Codec {

    // Encodes a tree to a single string.
    public String serialize(TreeNode root) {
        // 序列化成字符串,1,2,3,null,null,4,5,null...
        if(root == null){
            return "";
        }
        StringBuilder build = new StringBuilder();
        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);

        while(!queue.isEmpty()){
            TreeNode t = queue.poll();
            if(t != null){
                queue.add(t.left);
```

```
                queue.add(t.right);
                build.append(t.val + ",");
            }else{
                build.append("null,");
            }
        }

        return build.toString();
    }

    // Decodes your encoded data to tree.
    public TreeNode deserialize(String data) {
        // 1,2,3,null,null,4,5,null...
        if(data == null || data.length() <= 0){
            return null;
        }
        String[] s = data.split(",");
        Queue<TreeNode> queue = new LinkedList<>();
        TreeNode root = new TreeNode(Integer.parseInt(s[0]));
        queue.add(root);
        int i = 1;
        while(!queue.isEmpty()){
            TreeNode t = queue.poll();
            // 构建做左节点
            if(!s[i].equals("null")){
                TreeNode left = new TreeNode(Integer.parseInt(s[i]));
                t.left = left;
                queue.add(left);
            }
            i++;
            // 构建右节点
            if(!s[i].equals("null")){
                TreeNode right = new TreeNode(Integer.parseInt(s[i]));
                t.right = right;
                queue.add(right);
            }
            i++;
        }
        return root;
```

```
        }
    }
```

剑指 Offer 38. 字符串的排列

问题描述

输入一个字符串，打印出该字符串中字符的所有排列。

你可以以任意顺序返回这个字符串数组，但里面不能有重复元素。

示例:

```
输入：s = "abc"
输出：["abc","acb","bac","bca","cab","cba"]
```

限制：

`1 <= s 的长度 <= 8`

视频讲解直达： 本题视频讲解

```
class Solution {
    List<Character> path;
    List<String> res;
    boolean[] visited;
    public String[] permutation(String s) {
        this.path = new ArrayList<>();
        this.res = new ArrayList<>();
        this.visited = new boolean[s.length()];

        char[] arr = s.toCharArray();
        Arrays.sort(arr);
        dfs(arr, 0);

        String[] ss = new String[res.size()];
```

```
        for(int i = 0; i < res.size(); i++){
            ss[i] = res.get(i);
        }

        return ss;

    }
    // 时间复杂度: O(n*n!) 空间: N,N,递归调用最大深度也是 N, 3n,O(n)
    void dfs(char[] arr, int k){
        if(arr.length == k){
            res.add(listToString(path));
            return;
        }

        //进行N叉树搜索
        for(int i = 0; i < arr.length; i++){
            // 剪枝 aab
            if(i > 0 && arr[i] == arr[i - 1] && visited[i - 1] ==
false){
                continue;
            }
            if(visited[i] == false){
                // 递归访问
                visited[i] = true;
                path.add(arr[i]);
                dfs(arr, k + 1);
                path.remove(path.size() - 1);
                visited[i] = false;
            }
        }
    }

    String listToString(List<Character> list){
        StringBuilder b = new StringBuilder();
        for(int i = 0; i < list.size(); i++){
            b.append(list.get(i));
        }

        return b.toString();
```

```
        }
    }
```

剑指 Offer 39. 数组中出现次数超过一半的数字

问题描述

数组中有一个数字出现的次数超过数组长度的一半，请找出这个数字。

你可以假设数组是非空的，并且给定的数组总是存在多数元素。

示例 1:

```
输入：[1, 2, 3, 2, 2, 2, 5, 4, 2]
输出：2
```

限制：

`1 <= 数组长度 <= 50000`

视频讲解直达： 本题视频讲解

```
class Solution {
    public int majorityElement(int[] nums) {
        if(nums.length <= 2){
            return nums[0];
        }

        int x = nums[0];
        int sum = 1;

        for(int i = 1; i < nums.length; i++){
            if(sum == 0){
                x = nums[i];
                sum = 1;
```

```
            } else {
                if(x == nums[i]){
                    sum++;
                } else {
                    sum--;
                }
            }
        }

        return x;
    }
}
```

剑指 Offer 40. 最小的k个数

问题描述

输入整数数组 `arr` ，找出其中最小的 `k` 个数。例如，输入4、5、1、6、2、7、3、8这8个数字，则最小的4个数字是1、2、3、4。

示例 1:

```
输入：arr = [3,2,1], k = 2
输出：[1,2] 或者 [2,1]
```

示例 2:

```
输入：arr = [0,1,2,1], k = 1
输出：[0]
```

限制：

- `0 <= k <= arr.length <= 10000`
- `0 <= arr[i] <= 10000`

视频讲解直达： 本题视频讲解

```java
class Solution {
    public int[] getLeastNumbers(int[] arr, int k) {
        if(arr == null || arr.length == 0 || k == 0){
            return new int[0];
        }

        return quickFind(arr, 0, arr.length - 1, k);
    }

    int[] quickFind(int[] arr, int left, int right, int k){
        int i = partition(arr, left, right);
        // 之所以需要 i+1，是因为下标从 0 开始，0～i之间一共有 i+1个数
        if(i + 1 == k){
            return Arrays.copyOf(arr, k);
        }

        if(i + 1 > k){
            return quickFind(arr, 0, i - 1, k);
        } else {
            return quickFind(arr, i + 1, right, k);
        }
    }
    // 找出pivot的下标以及使小于等于pivot在左边，大于等于的在右边
    int partition(int[] arr, int left, int right){
        int pivot = arr[left];

        int i = left + 1;
        int j = right;

        while(i < j){
            while(i <= j && arr[i] <= pivot) i++;
            while(i <= j && arr[j] >= pivot) j--;
            if(i >= j){
                break;
            }

            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
```

```
        }

        arr[left] = arr[j];
        arr[j] = pivot;
        return j;
    }
}
```

剑指 Offer 41. 数据流中的中位数

问题描述

如何得到一个数据流中的中位数？如果从数据流中读出奇数个数值，那么中位数就是所有数值排序之后位于中间的数值。如果从数据流中读出偶数个数值，那么中位数就是所有数值排序之后中间两个数的平均值。

例如，

[2,3,4] 的中位数是 3

[2,3] 的中位数是 (2 + 3) / 2 = 2.5

设计一个支持以下两种操作的数据结构：

- void addNum(int num) - 从数据流中添加一个整数到数据结构中。
- double findMedian() - 返回目前所有元素的中位数。

示例 1:

```
输入：
["MedianFinder","addNum","addNum","findMedian","addNum","findMedian"]
[[],[1],[2],[],[3],[]]
输出：[null,null,null,1.50000,null,2.00000]
```

示例 2:

```
输入：
["MedianFinder","addNum","findMedian","addNum","findMedian"]
[[],[2],[],[3],[]]
输出：[null,null,2.00000,null,2.50000]
```

限制：

- 最多会对 `addNum、findMedian` 进行 `50000` 次调用。

视频讲解直达： [本题视频讲解]

```java
class MedianFinder {
    Queue<Integer> min, max;
    /** initialize your data structure here. */
    public MedianFinder() {
        min = new PriorityQueue<>(); // 小根，保存较大的
        max = new PriorityQueue<>((x, y) -> (y - x));// 大根
    }

    public void addNum(int num) {
        // 如果是偶数
        if(min.size() == max.size()){
            min.add(num);
            max.add(min.poll());
        } else {
            max.add(num);
            min.add(max.poll());
        }
    }

    public double findMedian() {
        // 如果是偶数
        if(min.size() == max.size()){
            return (min.peek() + max.peek()) / 2.0;
        } else {
            return max.peek() * 1.0;
        }
    }
}
```

剑指 Offer 42. 连续子数组的最大和

问题描述

输入一个整型数组，数组中的一个或连续多个整数组成一个子数组。求所有子数组的和的最大值。

要求时间复杂度为O(n)。

示例 1:

```
输入: nums = [-2,1,-3,4,-1,2,1,-5,4]
输出: 6
解释: 连续子数组 [4,-1,2,1] 的和最大，为 6。
```

提示:

- `1 <= arr.length <= 10^5`
- `-100 <= arr[i] <= 100`

视频讲解直达： [本题视频讲解]

```java
class Solution {
    public int maxSubArray(int[] nums) {
        int dp = nums[0];
        int max = nums[0];
        // 刷新dp之前，dp相当于是 dp[i-1],刷新之后，Dp就是dp[i]
        for(int i = 1; i < nums.length; i++){
            dp = Math.max(dp + nums[i], nums[i]);
            max = Math.max(max, dp);
        }

        return max;
    }
}
```

剑指 Offer 43. 1～n 整数中 1 出现的次数

问题描述

输入一个整数 n ， 求1～n这n个整数的十进制表示中1出现的次数。

例如，输入12，1～12这些整数中包含1 的数字有1、10、11和12，1一共出现了5次。

示例 1:

```
输入：n = 12
输出：5
```

示例 2:

```
输入：n = 13
输出：6
```

限制：

- `1 <= n < 2^31`

视频讲解直达： 本题视频讲解

```
class Solution {
    public int countDigitOne(int n) {
        // 几个变量计算：cur = (n/bit)%10, low = n % bit, high = n / bit
/ 10
        // 几个公示
        // cur > 1 => (high + 1) * bit
        // cur == 1 =>  (high * bit) + (1 + low)
        // cur = 0 => high * bit

        long bit = 1;
        long sum = 0;
        while(bit <= n){
            long cur = (n/bit)%10;
            long low = n % bit;
            long high = n / bit / 10;
```

```
            if(cur > 1){
                sum += (high + 1) * bit;
            }else if(cur == 1){
                sum += (high * bit) + (1 + low);
            }else{
                sum += high * bit;
            }
            bit  = bit * 10;
        }

        return (int)sum;

    }
}
```

剑指 Offer 44. 数字序列中某一位的数字

问题描述

数字以0123456789101112131415...的格式序列化到一个字符序列中。在这个序列中，第5位（从下标0开始计数）是5，第13位是1，第19位是4，等等。

请写一个函数，求任意第n位对应的数字。

示例 1:

```
输入：n = 3
输出：3
```

示例 2:

```
输入：n = 11
输出：0
```

限制:

- `0 <= n < 2^31`

视频讲解直达： [本题视频讲解]

```java
class Solution {
    public int findNthDigit(int n) {
        if(n == 0){
            return 0;
        }

        long bit = 1;
        int i = 1;
        long count = 9;

        while(count < n){
            n = (int)(n - count);
            bit = bit * 10;
            i++;
            count = bit * i * 9;
        }
        // 确定是在这个区间的哪个数
        long num = bit + (n - 1) / i;
        // 确定在 Num 的那个字符
        int index = (n - 1) % i + 1;
        int res = (int)(num / Math.pow(10, i - index)) % 10;

        return res;
    }
}
```

剑指 Offer 45. 把数组排成最小的数

问题描述

输入一个非负整数数组，把数组里所有数字拼接起来排成一个数，打印能拼接出的所有数字中最小的一个。

示例 1:

```
输入：[10,2]
输出："102"
```

示例 2：

```
输入：[3,30,34,5,9]
输出："3033459"
```

提示：

- `0 < nums.length <= 100`

说明：

- 输出结果可能非常大，所以你需要返回一个字符串而不是整数
- 拼接起来的数字可能会有前导 0，最后结果不需要去掉前导 0

视频讲解直达： 本题视频讲解

```java
class Solution {
    public String minNumber(int[] nums) {

        String[] strs = new String[nums.length];

        for (int i = 0; i < nums.length; i++) {
            strs[i] = String.valueOf(nums[i]);
        }

        quickSort(strs, 0, strs.length - 1);

        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < strs.length; i++) {
            sb.append(strs[i]);
        }
        return sb.toString();
    }
///时间nlogn，空间 On
    // 快速排序模版（和普通快速排序模版一样的，除了比较那里不一样）
    private void quickSort(String[] arr, int left, int right) {
        if (left > right) {
```

```
            return;
        }
        int i = partition(arr, left, right);
        quickSort(arr, left, i - 1);
        quickSort(arr, i + 1, right);

    }

    int partition(String[] arr, int left, int right){
        String pivot = arr[left]; // 中间值
         int i = left;
         int j = right;

         while (i < j) {
             while (i <= j && (arr[i] + pivot).compareTo(pivot +
arr[i]) <= 0) {
                 i++;
             }

             while (i <= j && (arr[j] + pivot).compareTo(pivot +
arr[j]) >= 0) {
                 j--;
             }
             if(i >= j){
                 break;
             }
             String temp = arr[i];
             arr[i] = arr[j];
             arr[j] = temp;
         }
         arr[left] = arr[j];
         arr[j] = pivot;

         return j;

    }
}
```

剑指 Offer 46. 把数字翻译成字符串

问题描述

给定一个数字，我们按照如下规则把它翻译为字符串：0 翻译成 “a” ，1 翻译成 “b”，……，11 翻译成 “l”，……，25 翻译成 “z”。一个数字可能有多个翻译。请编程实现一个函数，用来计算一个数字有多少种不同的翻译方法。

示例 1:

```
输入：12258
输出：5
解释：12258有5种不同的翻译，分别是"bccfi", "bwfi", "bczi", "mcfi"和"mzi"
```

提示:

- `0 <= num < 2^31`

视频讲解直达： [本题视频讲解]

```
class Solution {
    public int translateNum(int num) {
        if(num <= 9){
            return 1;
        }
        char[] arr = String.valueOf(num).toCharArray();
        int n = arr.length;
        //int[] dp = new int[n + 1];
        // f(n) = f(n-1) + f(n-2)
        int a = 1;
        int b = 1;
        int c = 1;
        // 优化 a,b,c
        for(int i = 2; i <= n; i++){
            // 计算第i和第i-1个字符的组合
            int tmp = 10 * (arr[i-2] - '0') + (arr[i-1] - '0');
            if(tmp >= 10 && tmp <= 25){
                c = a + b;
            }else{
                c = b;
```

```
            }
            //迭代
            a = b;
            b = c;
        }

        return c;
    }
}
```

剑指 Offer 47. 礼物的最大价值

问题描述

在一个 m*n 的棋盘的每一格都放有一个礼物，每个礼物都有一定的价值（价值大于 0）。你可以从棋盘的左上角开始拿格子里的礼物，并每次向右或者向下移动一格、直到到达棋盘的右下角。给定一个棋盘及其上面的礼物的价值，请计算你最多能拿到多少价值的礼物?

示例 1:

```
输入:
[
  [1,3,1],
  [1,5,1],
  [4,2,1]
]
输出: 12
解释: 路径 1→3→5→2→1 可以拿到最多价值的礼物
```

提示:

- `0 < grid.length <= 200`
- `0 < grid[0].length <= 200`

视频讲解直达： 本题视频讲解

```
class Solution {
```

```
    public int maxValue(int[][] grid) {
        //优化版本
        int n = grid.length;
        int m = grid[0].length;

        int[]dp = new int[m];
        dp[0] = grid[0][0];

        for(int j = 1; j < m; j++){
            dp[j] = dp[j-1] + grid[0][j];
        }

        for(int i = 1; i < n; i++){
            //
            dp[0] =  dp[0] + grid[i][0];
            for(int j = 1; j < m; j++){
                dp[j] = Math.max(dp[j], dp[j-1]) + grid[i][j];
            }
        }
        return dp[m-1];
    }
// 二维版本
    public int maxValue(int[][] grid) {
        int n = grid.length;
        int m = grid[0].length;

        int[][] dp = new int[n][m];
        dp[0][0] = grid[0][0];
        for(int i = 1; i < n; i++){
            dp[i][0] = dp[i-1][0] + grid[i][0];
        }
        for(int j = 1; j < m; j++){
            dp[0][j] = dp[0][j -1] + grid[0][j];
        }

        for(int i = 1; i < n; i++){
            for(int j = 1; j < m; j++){
                dp[i][j] = Math.max(dp[i-1][j], dp[i][j-1]) + grid[i]
[j];
```

```
                }
            }

            return dp[n-1][m-1];
        }
    }
```

剑指 Offer 48. 最长不含重复字符的子字符串

问题描述

请从字符串中找出一个最长的不包含重复字符的子字符串，计算该最长子字符串的长度。

示例 1:

```
输入： "abcabcbb"
输出： 3
解释： 因为无重复字符的最长子串是 "abc"，所以其长度为 3。
```

示例 2:

```
输入： "bbbbb"
输出： 1
解释： 因为无重复字符的最长子串是 "b"，所以其长度为 1。
```

示例 3:

```
输入： "pwwkew"
输出： 3
解释： 因为无重复字符的最长子串是 "wke"，所以其长度为 3。
       请注意，你的答案必须是 子串 的长度，"pwke" 是一个子序列，不是子串。
```

提示:

- `s.length <= 40000`

视频讲解直达： [本题视频讲解]

```java
class Solution {
    // public int lengthOfLongestSubstring(String s) {
    //     if(s == null || s.length() <= 0){
    //         return 0;
    //     }
    //     // 优化 => 单个变量
    //     Map<Character,Integer> map = new HashMap<>();
    //     int[] dp = new int[s.length()];
    //     dp[0] = 1;
    //     map.put(s.charAt(0), 0);
    //     int res = 1;
    //     for(int i = 1; i < s.length(); i++){
    //         if(!map.containsKey(s.charAt(i))){
    //             dp[i] = dp[i-1] + 1;
    //         }else{
    //             int k = map.get(s.charAt(i));
    //             dp[i] = i - k <= dp[i-1] ? i - k : dp[i-1] + 1;
    //         }
    //         res = Math.max(res, dp[i]);
    //         map.put(s.charAt(i), i);
    //     }

    //     return res;
    // }

    // 优化版本
    public int lengthOfLongestSubstring(String s) {
        if(s == null || s.length() <= 0){
            return 0;
        }
        // 优化 => 单个变量
        Map<Character,Integer> map = new HashMap<>();
        //int[] dp = new int[s.length()];
        int a = 1;
        map.put(s.charAt(0), 0);
        int res = 1;
        for(int i = 1; i < s.length(); i++){
            if(!map.containsKey(s.charAt(i))){
                a = a + 1;// 就是没有刷新 a 之前，a表示dp[i-1]
```

```
            }else{
                int k = map.get(s.charAt(i));
                a = i - k <= a ? i - k : a + 1;
            }
            res = Math.max(res, a);
            map.put(s.charAt(i), i);
        }

        return res;
        // 时间On,空间On
    }
}
```

剑指 Offer 49. 丑数

问题描述

我们把只包含质因子 2、3 和 5 的数称作丑数（Ugly Number）。求按从小到大的顺序的第 n 个丑数。

示例：

```
输入: n = 10
输出: 12
解释: 1, 2, 3, 4, 5, 6, 8, 9, 10, 12 是前 10 个丑数。
```

说明:

1. `1` 是丑数。
2. `n` **不超过**1690。

视频讲解直达： 本题视频讲解

```
class Solution {
    public int nthUglyNumber(int n) {
        int a = 1, b = 1, c = 1;
        int[] dp = new int[n+1];
        dp[1] = 1;
```

```
        for(int i = 2; i <= n; i++){
            dp[i] = Math.min(Math.min(dp[a] * 2, dp[b] * 3), dp[c] *
5);
            if(dp[i] == dp[a] * 2) a++;
            if(dp[i] == dp[b] * 3) b++;
            if(dp[i] == dp[c] * 5) c++;
        }

        return dp[n];
    }
}
```

剑指 Offer 50. 第一个只出现一次的字符

问题描述

在字符串 s 中找出第一个只出现一次的字符。如果没有，返回一个单空格。 s 只包含小写字母。

示例1：

```
输入：s = "abaccdeff"
输出：'b'
```

示例 2:

```
输入：s = ""
输出：' '
```

限制:

`0 <= s 的长度 <= 50000`

视频讲解直达： 本题视频讲解

```
class Solution {
    public char firstUniqChar(String s) {
```

```
        if(s == null || s.length() <= 0){
            return ' ';
        }
//一个字母出现的次数大于 1 次就不符合要求了，这个时候使用 Fasle 标记状态相对于 Integer 的不断递增更合理，也更省空间。布尔值可以用来判断，可以简化代码逻辑。
        Map<Character, Boolean> map = new LinkedHashMap<>();
        for(int i = 0; i < s.length(); i++){
            map.put(s.charAt(i), !map.containsKey(s.charAt(i)));
        }

        for(Map.Entry<Character, Boolean> m : map.entrySet()){
            if(m.getValue()){
                return m.getKey();
            }
        }

        return ' ';
    }
}
```

剑指 Offer 51. 数组中的逆序对

问题描述

在数组中的两个数字，如果前面一个数字大于后面的数字，则这两个数字组成一个逆序对。输入一个数组，求出这个数组中的逆序对的总数。

示例1：

```
输入：[7,5,6,4]
输出：5
```

限制:

`0 <= 数组长度 <= 50000`

视频讲解直达： 本题视频讲解

```java
class Solution {
    public int reversePairs(int[] nums) {
        if(nums == null || nums.length <= 1){
            return 0;
        }
        return mergeSort(nums, 0, nums.length - 1);
    }

    int mergeSort(int[] nums, int left, int right){
        if(left >= right){
            return 0;
        }
        int mid = (right - left) / 2 + left;
        int x1 = mergeSort(nums, left, mid);
        int x2 = mergeSort(nums, mid + 1, right);
        int x3 = merge(nums, left, mid, mid+1, right);

        return x1 + x2 + x3;
    }

    int merge(int[] nums, int l1, int r1, int l2, int r2){
        int[] temp = new int[r2 - l1 + 1];
        int count = 0;
        int i = l1, j = l2, k = 0;
        while(i <= r1 && j <= r2){
            if(nums[i] > nums[j]){
                count = count + (l2 - i);
                temp[k++] = nums[j++];
            }else{
                temp[k++] = nums[i++];
            }
        }
        while(i <= r1) temp[k++] = nums[i++];
        while(j <= r2) temp[k++] = nums[j++];

        // 把临时数组复制回原数组
        k = 0;
        for(i = l1; i <= r2; i++){
            nums[i] = temp[k++];
```

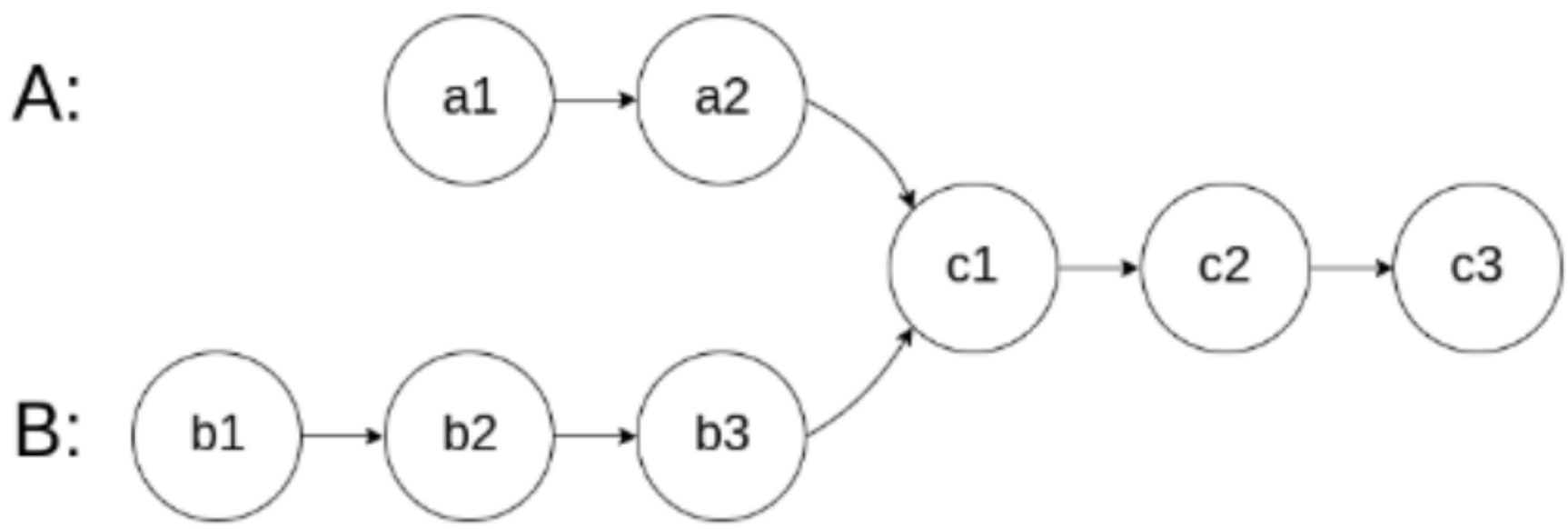
```
        }

        return count;
    }
}
```

剑指 Offer 52. 两个链表的第一个

问题描述

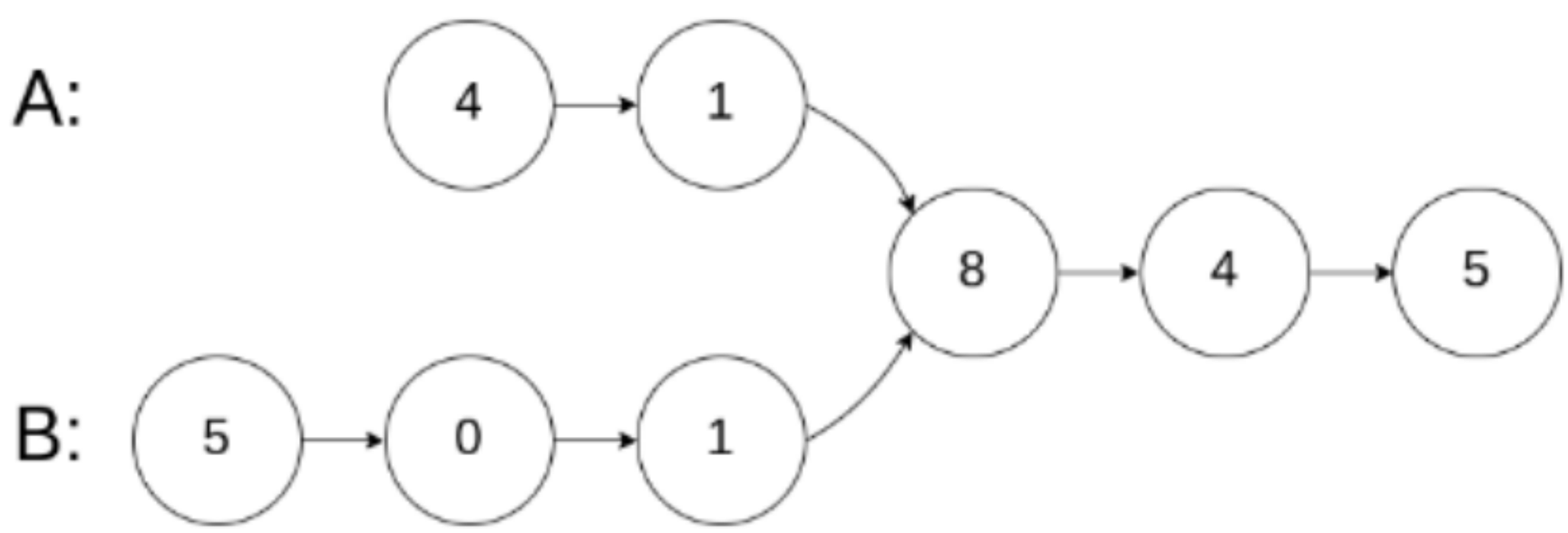
输入两个链表，找出它们的第一个公共节点。

如下面的两个链表：



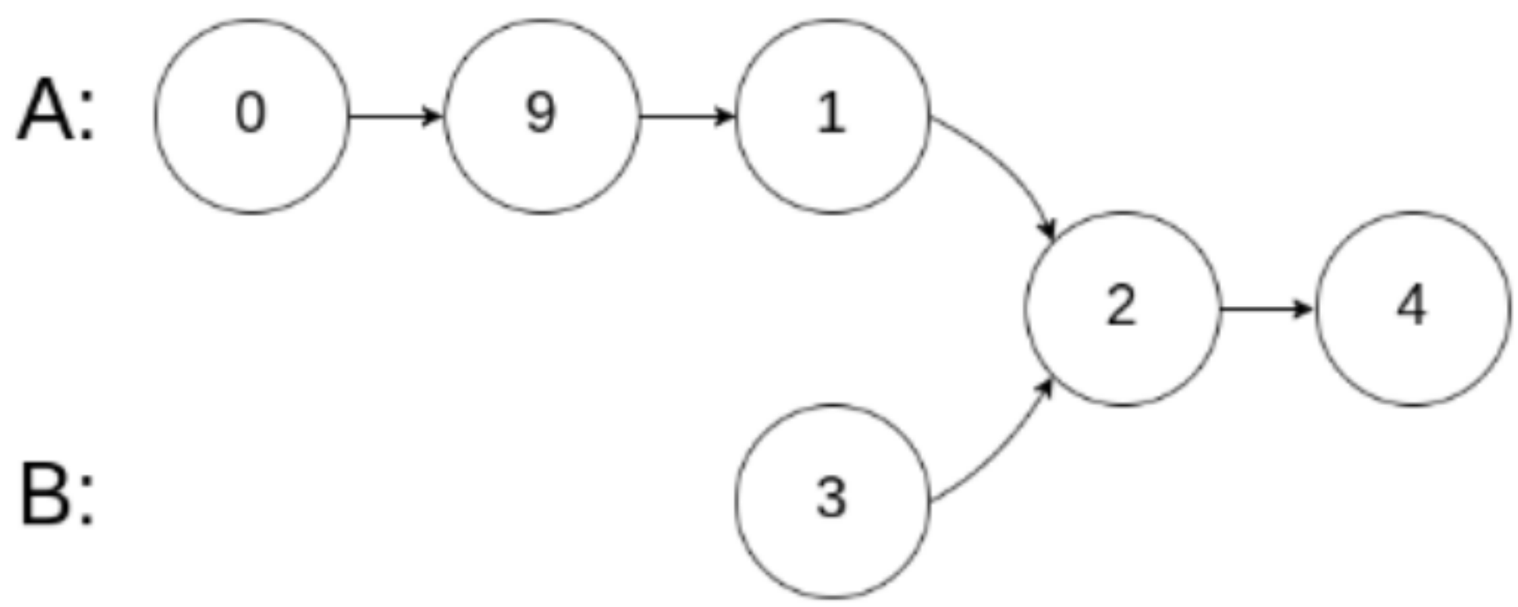
在节点 c1 开始相交。

示例1：



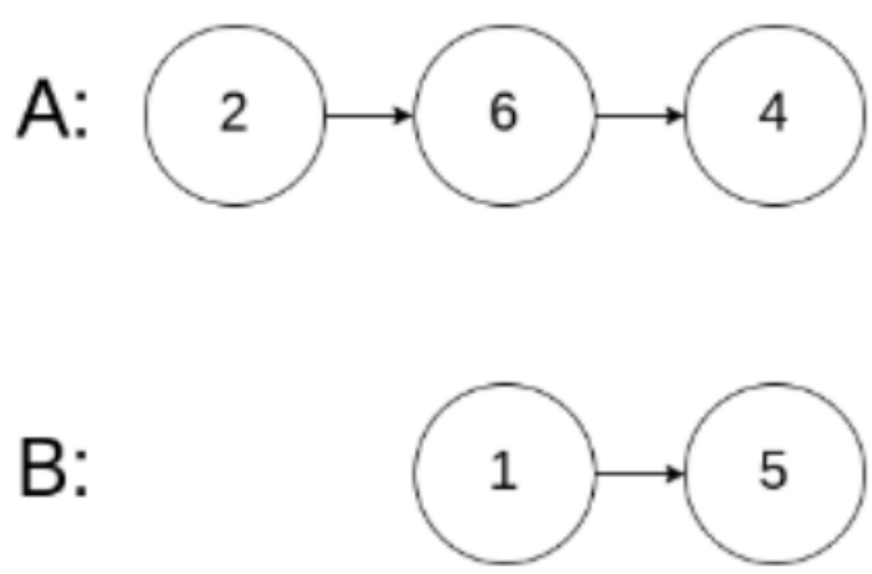
```
输入：intersectVal = 8, listA = [4,1,8,4,5], listB = [5,0,1,8,4,5],
skipA = 2, skipB = 3
输出：Reference of the node with value = 8
输入解释：相交节点的值为 8 （注意，如果两个列表相交则不能为 0）。从各自的表头开始算
起，链表 A 为 [4,1,8,4,5]，链表 B 为 [5,0,1,8,4,5]。在 A 中，相交节点前有 2
个节点；在 B 中，相交节点前有 3 个节点。
```

示例 2：




```
输入：intersectVal = 2, listA = [0,9,1,2,4], listB = [3,2,4], skipA =
3, skipB = 1
输出：Reference of the node with value = 2
输入解释：相交节点的值为 2 （注意，如果两个列表相交则不能为 0）。从各自的表头开始算
起，链表 A 为 [0,9,1,2,4]，链表 B 为 [3,2,4]。在 A 中，相交节点前有 3 个节点；
在 B 中，相交节点前有 1 个节点。
```

示例 3：



```
输入：intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3,
skipB = 2
输出：null
输入解释：从各自的表头开始算起，链表 A 为 [2,6,4]，链表 B 为 [1,5]。由于这两个链表不相交，所以 intersectVal 必须为 0，而 skipA 和 skipB 可以是任意值。
解释：这两个链表不相交，因此返回 null。
```

注意:

- 如果两个链表没有交点，返回 `null`.
- 在返回结果后，两个链表仍须保持原有的结构。
- 可假定整个链表结构中没有循环。
- 程序尽量满足 O(*n*) 时间复杂度，且仅用 O(*1*) 内存。

视频讲解直达： [本题视频讲解]

```java
class Solution {
    ListNode getIntersectionNode(ListNode headA, ListNode headB) {
        if(headA == null || headB == null) return null;
        ListNode A = headA;
        ListNode B = headB;

        while(A != B){
            A = A == null ? headB : A.next;
            B = B == null ? headA : B.next;
        }

        return A;
    }
}
```

剑指 Offer 53 - I. 在排序数组中查找

问题描述

统计一个数字在排序数组中出现的次数。

示例1：

```
输入: nums = [5,7,7,8,8,10], target = 8
输出: 2
```

示例 2:

```
输入: nums = [5,7,7,8,8,10], target = 6
输出: 0
```

提示:

- `0 <= nums.length <= 105`
- `-109 <= nums[i] <= 109`
- `nums` 是一个非递减数组
- `-109 <= target <= 109`

视频讲解直达： [本题视频讲解]

```
class Solution {
    public int search(int[] nums, int target) {
        //常见题型:
        // 1.寻找第一个大于等于 target 的数   2.寻找第一个等于 target 的数
        // 3.寻找最后一个大于等于 target 的数 4.寻找最后一个等于 target 的数
        //PS: 其实这里是不需要写两个查找函数的, 可以把代码优化放在同一个方法里滴
        int left = search2(nums, target);
        int right = search4(nums, target);

        if(left < 0 || right < 0) return 0;
        return right - left + 1;




    }
    //2.寻找第一个等于 target 的数的下标
    int search2(int[] nums, int target){
```

```
        if(nums == null || nums.length <= 0){
            return -1;
        }
        int left = 0, right = nums.length - 1;
        while(left < right){
            // 向下取整
            int mid = (right - left) / 2 + left;
            if(nums[mid] >= target){
                right = mid;
            }else{
                left = mid + 1;
            }
        }
        // 判断该数是否存在
        if(nums[left] != target) return -1;
        return left;
    }

    //4.寻找最后一个等于 target 的数下标
    int search4(int[] nums, int target){
        if(nums == null || nums.length <= 0){
            return -1;
        }
        int left = 0, right = nums.length - 1;
        while(left < right){
            //向上取整
            int mid = (right - left + 1) / 2 + left;
            if(nums[mid] <= target){
                left = mid;
            }else{
                right = mid - 1;
            }
        }
        // 判断该数是否存在
        if(nums[left] != target) return -1;
        return left;
    }

}
```

剑指 Offer 53 - II. 0～n-1中缺失的数字

问题描述

一个长度为n-1的递增排序数组中的所有数字都是唯一的，并且每个数字都在范围0～n-1之内。在范围0～n-1内的n个数字中有且只有一个数字不在该数组中，请找出这个数字。

示例1：

```
输入：[0,1,3]
输出：2
```

示例 2:

```
输入：[0,1,2,3,4,5,6,7,9]
输出：8
```

限制:

- 1 <= 数组长度 <= 10000

视频讲解直达： 本题视频讲解

```java
class Solution {
    public int missingNumber(int[] nums) {
        int l = 0, r = nums.length - 1;
        while(l < r){
            int mid = (r - l) / 2 + l;
            if(nums[mid] == mid) l = mid + 1;
            else r = mid;
        }

        return nums[l] == l ? l + 1 : l;
    }
}
```

剑指 Offer 54. 二叉搜索树的第k大节点

问题描述

给定一棵二叉搜索树，请找出其中第 `k` 大的节点的值。

示例1：

```
输入: root = [3,1,4,null,2], k = 1
   3
  / \
 1   4
  \
   2
输出: 4
```

示例 2:

```
输入: root = [5,3,6,2,4,null,null,1], k = 3
       5
      / \
     3   6
    / \
   2   4
  /
 1
输出: 4
```

限制:

- 1 ≤ k ≤ 二叉搜索树元素个数

视频讲解直达： 本题视频讲解

```
class Solution {
    int k = 0;
    int target = 0;
    public int kthLargest(TreeNode root, int k) {
        // 中序遍历：有序的序列 左 根 右 从小到大排序的
        //                      右 根 左 从大到小的排序
```

```
        this.k = k;
        right_root_left(root);
        return target;
    }

    void right_root_left(TreeNode root){
        if(root == null || k <= 0) return;

        right_root_left(root.right);
        // 打印当前根节点
        k--;
        if(k == 0){
            target = root.val;
        }
        right_root_left(root.left);

    }
}
```

剑指 Offer 55 - I. 二叉树的深度

问题描述

输入一棵二叉树的根节点，求该树的深度。从根节点到叶节点依次经过的节点（含根、叶节点）形成树的一条路径，最长路径的长度为树的深度。

例如：

给定二叉树 [3,9,20,null,null,15,7],

```
    3
   / \
  9  20
    /  \
   15   7
```

返回它的最大深度 3 。

提示：

1. `节点总数 <= 10000`

视频讲解直达： [本题视频讲解]

```
class Solution {
    public int maxDepth(TreeNode root) {
        if(root == null) return 0;
        int left = maxDepth(root.left);
        int right = maxDepth(root.right);

        return Math.max(left, right) + 1;
    }
}
```

剑指 Offer 55 - II. 平衡二叉树

问题描述

输入一棵二叉树的根节点，判断该树是不是平衡二叉树。如果某二叉树中任意节点的左右子树的深度相差不超过1，那么它就是一棵平衡二叉树。

示例 1:

给定二叉树 `[3,9,20,null,null,15,7]`

```
    3
   / \
  9  20
    /  \
   15   7
```

返回 `true` 。

示例 2:

给定二叉树 `[1,2,2,3,3,null,null,4,4]`

```
        1
       / \
      2   2
     / \
    3   3
   / \
  4   4
```

返回 `false` 。

限制：

- `0 <= 树的结点个数 <= 10000`

视频讲解直达： [本题视频讲解]

```java
class Solution {
    // nlogn 空间 On   方法2:时间 On,空间O
    public boolean isBalanced(TreeNode root) {
        if(root == null) return true;
        if(maxDepth(root) == -1) return false;
        return true;

    }

    public int maxDepth(TreeNode root) {
        if(root == null) return 0;
        int left = maxDepth(root.left);
        if(left == -1) return -1;
        int right = maxDepth(root.right);
        if(right == -1) return -1;
        // 返回-1表示不符合条件了
        if(Math.abs(left - right) > 1) return -1;

        return Math.max(left, right) + 1;
    }
}
```

剑指 Offer 56 - I. 数组中数字出现的次数

问题描述

一个整型数组 `nums` 里除两个数字之外，其他数字都出现了两次。请写程序找出这两个只出现一次的数字。要求时间复杂度是O(n)，空间复杂度是O(1)。

示例 1:

```
输入：nums = [4,1,4,6]
输出：[1,6] 或 [6,1]
```

示例 2:

```
输入：nums = [1,2,10,4,1,4,3,3]
输出：[2,10] 或 [10,2]
```

限制：

- `2 <= nums.length <= 10000`

视频讲解直达： 本题视频讲解

```
class Solution {
    public int[] singleNumbers(int[] nums) {
        int z = 0;
        for(int i = 0;i < nums.length; i++){
            z = z ^ nums[i];
        }

        int m = 1;
        while((m & z) == 0){
            m = m << 1;
        }

        int x = 0, y = 0;

        for(int i = 0;i < nums.length; i++){
            if((nums[i] & m) == 0){
                //结果为 0 的子数组，一边统计用异或统计x
```

```
                x = x ^ nums[i];
            } else {
                //结果为 1 的子数组，一边统计用异或统计y
                y = y ^ nums[i];
            }
        }

        return new int[]{x, y};
    }
}
```

剑指 Offer 56 - II. 数组中数字出现的次数 II

问题描述

在一个数组 `nums` 中除一个数字只出现一次之外，其他数字都出现了三次。请找出那个只出现一次的数字。

示例 1:

```
输入：nums = [3,4,3,3]
输出：4
```

示例 2:

```
输入：nums = [9,1,7,9,7,9,7]
输出：1
```

限制：

- `1 <= nums.length <= 10000`
- `1 <= nums[i] < 2^31`

视频讲解直达： 本题视频讲解

```
class Solution {
    public int singleNumber(int[] nums) {
        int[] res = new int[32];
```

```
        int m = 1;
        int sum = 0;
        for(int i = 0; i < 32; i++){
            for(int j = 0; j < nums.length; j++){
                if((nums[j] & m) != 0){
                    res[i]++;
                }
            }

            res[i] = res[i] % 3;
            sum = sum + res[i] * m;
            m = m << 1;
        }

        return sum;
    }
}
```

剑指 Offer 57. 和为s的两个数字

问题描述

输入一个递增排序的数组和一个数字s，在数组中查找两个数，使得它们的和正好是s。如果有多对数字的和等于s，则输出任意一对即可。

示例 1:

```
输入：nums = [2,7,11,15], target = 9
输出：[2,7] 或者 [7,2]
```

示例 2:

```
输入：nums = [10,26,30,31,47,60], target = 40
输出：[10,30] 或者 [30,10]
```

限制：

- `1 <= nums.length <= 10^5`

- `1 <= nums[i] <= 10^6`

视频讲解直达： 本题视频讲解

```
class Solution {
    public int[] twoSum(int[] nums, int target) {
        if(nums == null || nums.length < 2){
            return new int[0];
        }

        int i = 0, j = nums.length - 1;

        while(i < j){
            if(nums[i] + nums[j] > target){
                j--;
            }else if(nums[i] + nums[j] < target){
                i++;
            }else{
                return new int[]{nums[i], nums[j]};
            }
        }

        return new int[0];
    }
}
```

剑指 Offer 57 - II. 和为s的连续正数序列

问题描述

输入一个正整数 target ， 输出所有和为 target 的连续正整数序列（至少含有两个数）。

序列内的数字由小到大排列，不同序列按照首个数字从小到大排列。

示例 1:

```
输入：target = 9
输出：[[2,3,4],[4,5]]
```

示例 2:

```
输入：target = 15
输出：[[1,2,3,4,5],[4,5,6],[7,8]]
```

限制：

- `1 <= target <= 10^5`

视频讲解直达： [本题视频讲解]

```java
class Solution {
    public int[][] findContinuousSequence(int target) {
        List<int[]> res = new ArrayList<>();
        int i = 1, j = 1;
        int sum = 1;
        while(i <= target/2){
            if(sum < target){
                j++;
                sum = sum + j;
            } else if(sum > target){
                sum = sum - i;
                i++;
            } else {
                int[] temp = new int[j - i + 1];
                int index = 0;
                for(int k = i; k <= j; k++){
                    temp[index++] = k;
                }
                sum = sum - i;
                i++;
                j++;
                sum = sum + j;
                res.add(temp);
            }
        }
```

```
        return res.toArray(new int[res.size()][]);
    }
}
```

剑指 Offer 58 - I. 翻转单词顺序

问题描述

输入一个英文句子，翻转句子中单词的顺序，但单词内字符的顺序不变。为简单起见，标点符号和普通字母一样处理。例如输入字符串"I am a student. "，则输出"student. a am I"。

示例 1:

```
输入："the sky is blue"
输出："blue is sky the"
```

示例 2:

```
输入："  hello world!  "
输出："world! hello"
解释：输入字符串可以在前面或者后面包含多余的空格，但是反转后的字符不能包括。
```

示例 3:

```
输入："a good   example"
输出："example good a"
解释：如果两个单词间有多余的空格，将反转后单词间的空格减少到只含一个。
```

说明：

- 无空格字符构成一个单词。
- 输入字符串可以在前面或者后面包含多余的空格，但是反转后的字符不能包括。
- 如果两个单词间有多余的空格，将反转后单词间的空格减少到只含一个。

视频讲解直达： 本题视频讲解

```java
class Solution {
    public String reverseWords(String s) {
        // split
        if(s == null || s.length() <= 0){
            return s;
        }
        s = s.trim();
        StringBuilder build = new StringBuilder();
        int i = s.length() - 1, j = i;

        while(i >= 0){
            while(i >= 0 && s.charAt(i) != ' ') i--;
            //[i+1, j];
            build.append(s.substring(i+1, j+1) + " ");
            while(i >= 0 && s.charAt(i) == ' ') i--;
            j = i;
        }

        return build.toString().trim();
    }
}
```

剑指 Offer 58 - II. 左旋转字符串

问题描述

字符串的左旋转操作是把字符串前面的若干个字符转移到字符串的尾部。请定义一个函数实现字符串左旋转操作的功能。比如，输入字符串"abcdefg"和数字2，该函数将返回左旋转两位得到的结果"cdefgab"。

示例 1:

```
输入：s = "abcdefg", k = 2
输出："cdefgab"
```

示例 2:

```
输入: s = "lrloseumgh", k = 6
输出: "umghlrlose"
```

限制：

- `1 <= k < s.length <= 10000`

视频讲解直达： [本题视频讲解]

```java
class Solution {
    public String reverseLeftWords(String s, int n) {
        StringBuilder build = new StringBuilder();
        int len = s.length();
        for(int i = n; i < len + n; i++){
            build.append(s.charAt(i%len));
        }

        // for(int i = 0; i < n; i++){
        //     build.append(s.charAt(i));
        // }

        return build.toString();
    }
}
```

剑指 Offer 59 - I. 滑动窗口的最大值

问题描述

给定一个数组 `nums` 和滑动窗口的大小 `k`，请找出所有滑动窗口里的最大值。

示例：

```
输入: nums = [1,3,-1,-3,5,3,6,7], 和 k = 3
输出: [3,3,5,5,6,7]
解释:

  滑动窗口的位置                最大值
---------------               -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
 1  3  -1  -3 [5  3  6] 7       6
 1  3  -1  -3  5 [3  6  7]      7
```

提示：

你可以假设 *k* 总是有效的，在输入数组 **不为空** 的情况下，`1 ≤ k ≤ nums.length`。

视频讲解直达： [本题视频讲解]

```java
class Solution {
    public int[] maxSlidingWindow(int[] nums, int k) {
        if(nums == null || nums.length <= 1){
            return nums;
        }

        LinkedList<Integer> queue = new LinkedList<>();
        int[] res = new int[nums.length - k + 1];
        int index = 0;

        //K

        for(int i = 0; i < nums.length; i++){
            while(!queue.isEmpty() && nums[queue.peekLast()] <=
nums[i]){
                queue.pollLast();
            }

            queue.add(i);
```

```
            if(queue.peekLast() - k == queue.peek()){
                queue.poll();
            }

            if(i + 1 >= k){
                res[index++] = nums[queue.peek()];
            }
        }

        return res;
    }
}
```

剑指 Offer 60. n个骰子的点数

问题描述

把n个骰子扔在地上，所有骰子朝上一面的点数之和为s。输入n，打印出s的所有可能的值出现的概率。

你需要用一个浮点数数组返回答案，其中第 i 个元素代表这 n 个骰子所能掷出的点数集合中第 i 小的那个的概率。

示例 1:

```
输入： 1
输出： [0.16667,0.16667,0.16667,0.16667,0.16667,0.16667]
```

示例 2:

```
输入： 2
输出：
[0.02778,0.05556,0.08333,0.11111,0.13889,0.16667,0.13889,0.11111,0.083
33,0.05556,0.02778]
```

限制：

- 1 <= n <= 11

视频讲解直达： [本题视频讲解]

```
class Solution {
    public double[] dicesProbability(int n) {
        int[][] dp = new int[n+1][6*n+1];

        for(int i = 1; i <= 6; i++){
            dp[1][i] = 1;
        }

        for(int i = 2; i <= n; i++){
            for(int j = i; j <= 6 * i; j++){
                for(int k = 1; k <= 6; k++){
                    if(j < k) break;
                    dp[i][j] += dp[i-1][j-k];
                }
            }
        }

        double[] res = new double[5*n + 1];
        int index = 0;
        double sum = Math.pow(6, n);

        for(int i = n; i <= 6 * n; i++){
            res[index++] = dp[n][i] / sum;
        }

        return res;
    }
}
```

剑指 Offer 61. 扑克牌中的顺子

问题描述

从若干副扑克牌中随机抽 5 张牌，判断是不是一个顺子，即这5张牌是不是连续的。2～10为数字本身，A为1，J为11，Q为12，K为13，而大、小王为 0 ，可以看成任意数字。A 不能视为 14。

示例 1:

```
输入: [1,2,3,4,5]
输出: True
```

示例 2:

```
输入: [0,0,1,2,5]
输出: True
```

限制:

数组长度为 5

数组的数取值为 [0, 13] .

视频讲解直达： 本题视频讲解

```java
class Solution {
    public boolean isStraight(int[] nums) {
        // 排序法 nlogn

        // 集合 n, n
        // 如果要成为一个顺子，则需要 满足两个条件
        // 1、不存在有重复的数，大小王除外
        // 2、最大值 - 最小值 < 5,大小王除外
        Set<Integer> set = new HashSet<>();
        int max = -1, min = 20;
        for(int i = 0; i < nums.length; i++){
            if(nums[i] == 0) continue;
            if(set.contains(nums[i]))return false;
            set.add(nums[i]);
```

```
            max = Math.max(nums[i], max);
            min = Math.min(nums[i], min);
        }

        return max - min < 5;
    }
}
```

剑指 Offer 62. 圆圈中最后剩下的数字

问题描述

0,1,…,n-1这n个数字排成一个圆圈，从数字0开始，每次从这个圆圈里删除第m个数字（删除后从下一个数字开始计数）。求出这个圆圈里剩下的最后一个数字。

例如，0、1、2、3、4这5个数字组成一个圆圈，从数字0开始每次删除第3个数字，则删除的前4个数字依次是2、0、4、1，因此最后剩下的数字是3。

示例 1:

```
输入： n = 5, m = 3
输出： 3
```

示例 2:

```
输入： n = 10, m = 17
输出： 2
```

限制：

- `1 <= n <= 10^5`
- `1 <= m <= 10^6`

视频讲解直达： [本题视频讲解]

```
class Solution {
    // 时间n 空间 1
    public int lastRemaining(int n, int m) {
```

```
        if( n == 0) return n;
        //return (lastRemaining(n - 1, m) + m) % n;
        // 优化
        int res = 0;

        for(int i = 1; i <= n; i++){
            res = (res + m) % i;
        }
        return res;
    }
}
```

剑指 Offer 63. 股票的最大利润

问题描述

假设把某股票的价格按照时间先后顺序存储在数组中，请问买卖该股票一次可能获得的最大利润是多少?

示例 1:

```
输入：[7,1,5,3,6,4]
输出：5
解释：在第 2 天（股票价格 = 1）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，最大利润 = 6-1 = 5 。
      注意利润不能是 7-1 = 6，因为卖出价格需要大于买入价格。
```

示例 2:

```
输入：[7,6,4,3,1]
输出：0
解释：在这种情况下，没有交易完成，所以最大利润为 0。
```

限制：

- 0 <= 数组长度 <= 10^5

视频讲解直达： [本题视频讲解]

```java
class Solution {
    public int maxProfit(int[] prices) {
        int min = Integer.MAX_VALUE;
        int max = 0;

        for(int i = 0; i < prices.length; i++){
            if(prices[i] < min){
                min = prices[i];
            }else{
                max = Math.max(max, prices[i] - min);
            }
        }
        return max;
    }
}
```

剑指 Offer 64. 求1+2+...+n

问题描述

求 `1+2+...+n` ，要求不能使用乘除法、for、while、if、else、switch、case等关键字及条件判断语句（A?B:C）。

示例 1:

```
输入： n = 3
输出： 6
```

示例 2:

```
输入： n = 9
输出： 45
```

限制：

- `1 <= n <= 10000`

视频讲解直达： [本题视频讲解]

```
class Solution {
    int sum = 0;
    public int sumNums(int n) {

        // 与或门
        boolean flag = n >= 1 && sumNums(n - 1) < 1;
        sum = sum + n;
        return sum;
    }
}
```

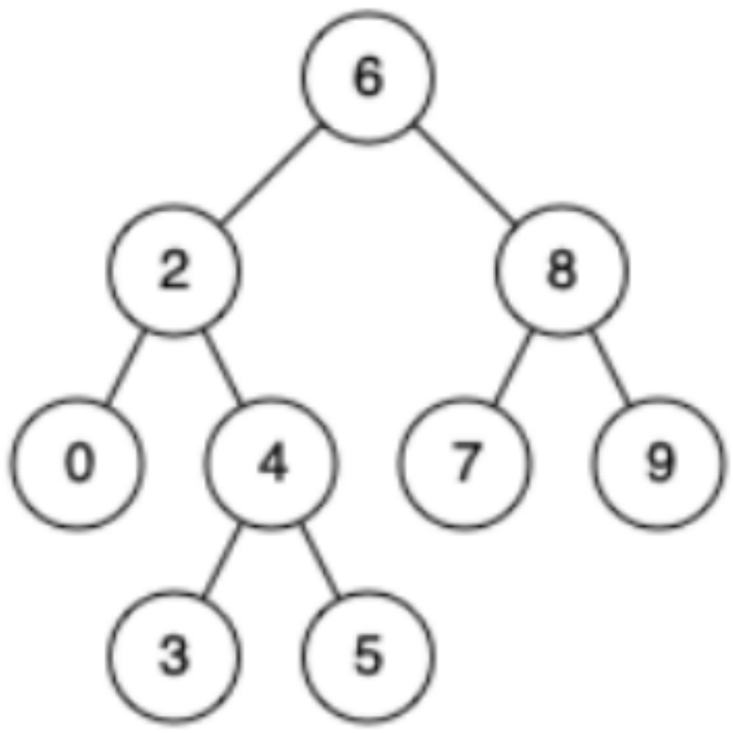
剑指 Offer 68 - I. 二叉搜索树的最近公共祖先

问题描述

给定一个二叉搜索树, 找到该树中两个指定节点的最近公共祖先。

百度百科中最近公共祖先的定义为：“对于有根树 T 的两个结点 p、q，最近公共祖先表示为一个结点 x，满足 x 是 p、q 的祖先且 x 的深度尽可能大（一个节点也可以是它自己的祖先）。”

例如，给定如下二叉搜索树：root = [6,2,8,0,4,7,9,null,null,3,5]



示例 1:

```
输入：root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8
输出：6
解释：节点 2 和节点 8 的最近公共祖先是 6。
```

示例 2:

```
输入：root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4
输出：2
解释：节点 2 和节点 4 的最近公共祖先是 2，因为根据定义最近公共祖先节点可以为节点本身。
```

说明:

- 所有节点的值都是唯一的。
- p、q 为不同节点且均存在于给定的二叉搜索树中。

视频讲解直达： [本题视频讲解]

```java
class Solution {
    // On,O1
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        while(root != null){
            if(root.val > p.val && root.val > q.val){
                root = root.left;
            } else if(root.val < p.val && root.val < q.val){
                root = root.right;
            } else {
                return root;
            }
        }

        return null;
    }
}
```