目录

[阿里巴巴面试题： 2](#_Toc18796)

[字节跳动面试题： 6](#_Toc22705)

[专题总结： 16](#_Toc17212)

[0排列 组合回溯专题 17](#_Toc18015)

[1. 滑动窗口专题 27](#_Toc1350)

[2. 双指针专题 31](#_Toc23525)

[3. 链表专题 34](#_Toc29269)

[4单调队列专题 单调栈 44](#_Toc2559)

[5.图专题 48](#_Toc26442)

[6二叉树专题 55](#_Toc19477)

[7动态规划专题 65](#_Toc6995)

[8多线程编程 76](#_Toc26761)

# 阿里巴巴面试题：

#### [剑指 Offer 09. 用两个栈实现队列](https://leetcode-cn.com/problems/yong-liang-ge-zhan-shi-xian-dui-lie-lcof/)

简单 一个栈存储push 一个栈存pop值（当没有从第一个栈pop拿出来 刚好符合队列先进先出的性质）

#### [225. 用队列实现栈](https://leetcode-cn.com/problems/implement-stack-using-queues/)

一个队列出队列的顺序就是出栈的顺序

一个队列为辅助队列 入栈时把主队列的元素复制到辅助队列先 再将元素添加主队列中

把辅助队列的元素再添加回主队列

#### [165. 比较版本号](https://leetcode-cn.com/problems/compare-version-numbers/)

字符串分割比较

#### [415. 字符串相加](https://leetcode-cn.com/problems/add-strings/)

模拟字符串加法 记录进位

for循环：

取两个字符串的数字 没有数字为0

计算总和 要加上进位

更新结果 和 carry

反转返回

#### [43. 字符串相乘](https://leetcode-cn.com/problems/multiply-strings/)

大数相乘 将相乘转化为单个数字相乘后相加，注意不是个位时结果要添加多个0

**[补充题9. 36进制加法](https://mp.weixin.qq.com/s/XcKQwnwCh5nZsz-DLHJwzQ" \t "https://codetop.cc/_blank)**

<https://mp.weixin.qq.com/s/XcKQwnwCh5nZsz-DLHJwzQ>

字符串相加的变形 按36进制的加法写就行



#### [169. 多数元素](https://leetcode-cn.com/problems/majority-element/)

用哈希记录出现次数

处理数组长度为1的特殊情况

#### [200. 岛屿数量](https://leetcode-cn.com/problems/number-of-islands/)

广度优先遍历 加两个for循环

先for遍历岛屿找到为1的值 岛屿数量+1

然后广播优先遍历把岛屿1周围的岛屿改为0

for循环遍历完所有的岛屿即可

#### [695. 岛屿的最大面积](https://leetcode-cn.com/problems/max-area-of-island/)

和岛屿的个数差不多 两层循环找到岛屿

bfs四个方向遍历 消除岛屿

#### [155. 最小栈](https://leetcode-cn.com/problems/min-stack/) 简单

两个栈 一个栈维护最小值

#### [剑指 Offer 39. 数组中出现次数超过一半的数字](https://leetcode-cn.com/problems/shu-zu-zhong-chu-xian-ci-shu-chao-guo-yi-ban-de-shu-zi-lcof/)

用哈希记录出现次数

处理数组长度为1的特殊情况

#### [88. 合并两个有序数组](https://leetcode-cn.com/problems/merge-sorted-array/)

双指针

从后往前遍历

#### [151. 翻转字符串里的单词](https://leetcode-cn.com/problems/reverse-words-in-a-string/)

简单 用字符串列表存单词

双指针 标志位 一次循环划分单词的首尾 存入字符串列表 然后继续遍历更新

最后反转输出即可

#### [125. 验证回文串](https://leetcode-cn.com/problems/valid-palindrome/)

可以用双指针做

判断字符或数字可以用Character.isLetterOrDigit() 也可以自己写

#### [503. 下一个更大元素 II](https://leetcode-cn.com/problems/next-greater-element-ii/)

单调递减栈 每次存入比栈顶小的元素 碰到比栈顶大的数就找到了下一个更大的元素

若一直大 一直找到

考虑最坏情况 遍历每个元素至少两次 数组末尾的话 遍历2\*n - 1

#### [459. 重复的子字符串](https://leetcode-cn.com/problems/repeated-substring-pattern/)

枚举 O(N\*N)

字符串一定是重复的子字符串的倍数

考虑所有的子字符串的长度[1,len] n%len == 0

遍历str[len:]之后的字符

要求所有str[j] == str[j-len] 都为真 才是字符串

KMP还没理解 O(n)

#### [剑指 Offer 10- I. 斐波那契数列](https://leetcode-cn.com/problems/fei-bo-na-qi-shu-lie-lcof/)

O(log(n))的求法

矩阵快速幂来求解 可能面试进阶考

#### [剑指 Offer 62. 圆圈中最后剩下的数字](https://leetcode-cn.com/problems/yuan-quan-zhong-zui-hou-sheng-xia-de-shu-zi-lcof/)

从后往前推

关注最后一个数字的下标

最后时小标为0

#### [剑指 Offer 31. 栈的压入、弹出序列](https://leetcode-cn.com/problems/zhan-de-ya-ru-dan-chu-xu-lie-lcof/)

模拟一个栈即可

Deque<Integer>  stack = new LinkedList<Integer>();

#### [56. 合并区间](https://leetcode-cn.com/problems/merge-intervals/)

按第一个元素排序 再合并

**范式数组排序的写法：**

Arrays.sort(intervals,(a,b)->a[0] - b[0]);

Arrays.sort(intervals, new Comparator<int[]>() {

public int compare(int[] interval1, int[] interval2) {

return interval1[0] - interval2[0];

}

});

#### [73. 矩阵置零](https://leetcode-cn.com/problems/set-matrix-zeroes/)

使用标记行和标记列

#### [33. 搜索旋转排序数组](https://leetcode-cn.com/problems/search-in-rotated-sorted-array/)

使用二分法 需要知道一点 旋转后有一边是排好序的 需要判断哪一边有序 且判断target在哪一边 更新l或r 所以能够二分法来做

#### [81. 搜索旋转排序数组 II](https://leetcode-cn.com/problems/search-in-rotated-sorted-array-ii/)

有重复元素 nums[mid] == nums[l]时不好判断哪一边有序

需要在原来基础上添加

if(nums[mid] == nums[l]){

                l++;

                continue;

            }

#### [443. 压缩字符串](https://leetcode-cn.com/problems/string-compression/)

双指针

在原数组上操作 一个读 一个写

#### [179. 最大数](https://leetcode-cn.com/problems/largest-number/)

将数组内元素逐个转化为字符串后，直接通过compareTo方法比较

//通过比较器进行排序

//若字符串 ab < ba 说明 b比较大 应该排在前面

#### [剑指 Offer 45. 把数组排成最小的数](https://leetcode-cn.com/problems/ba-shu-zu-pai-cheng-zui-xiao-de-shu-lcof/)

将数组内元素逐个转化为字符串后，直接通过compareTo方法比较

若字符串 ab > ba 说明 b比较小 应该排在前面

#### [815. 公交路线](https://leetcode-cn.com/problems/bus-routes/)

比较难 多做几遍

考察广度优先遍历 这里要把公交线路看做一个节点 不同公交线路换乘看作节点的转换

在广度优先遍历中，由于便利的是公交路线 所以还要加一个层循环 遍历公交线路的节点 然后将公交线路的**节点所包含的公交线路(map实现)**加入到**未访问(set实现)**的队列中

#### [剑指 Offer 40. 最小的k个数](https://leetcode-cn.com/problems/zui-xiao-de-kge-shu-lcof/)

考察快速排序 快速排序 轴值之前的数都是小于轴值的

找到轴值索引为k 返回前k个数即可

#### [349. 两个数组的交集](https://leetcode-cn.com/problems/intersection-of-two-arrays/)

将数组转化为集合 判断集合元素是否在另一个集合上

#### [836. 矩形重叠](https://leetcode-cn.com/problems/rectangle-overlap/)

逆思路 考虑不重叠的四种情况 上下左右 再取反即可

或者用投影到x和y轴看是否都有重叠

#### [557. 反转字符串中的单词 III](https://leetcode-cn.com/problems/reverse-words-in-a-string-iii/)

双指针法 循环遍历处理即可

# 字节跳动面试题：

#### 排除在阿里巴巴面试基础上做过的

#### [129. 求根节点到叶节点数字之和](https://leetcode-cn.com/problems/sum-root-to-leaf-numbers/)

深度优先遍历

原地哈希小专题

#### [剑指 Offer 03. 数组中重复的数字](https://leetcode-cn.com/problems/shu-zu-zhong-zhong-fu-de-shu-zi-lcof/)

原地哈希法

#### [41. 缺失的第一个正数](https://leetcode-cn.com/problems/first-missing-positive/)

在原数组上做标记

我们对数组进行遍历，对于遍历到的数 x，

如果它在 [1, N]的范围内，那么就将数组中的第 x-1被打上了标记，那么答案是N+1，

否则答案是最小的没有打上标记的位置加1。

原数组交换法 记住这个版本

//将数组的值x放到下标为x-1的位置上



#### [69. Sqrt(x)](https://leetcode-cn.com/problems/sqrtx/)

x的平方根

二分法 必须用小于等于 整型

牛顿迭代法 还没掌握

当需要精确到多少位数时 也还可以用二分法 代码：

double i = 0;

double x = 2;

double j = x;

double res = 0;

while(i < j){

double mid = (i + j)/2;

if(mid \* mid <= x&&(mid + 0.0000001) \* (mid + 0.0000001) > x){

res = mid;

break;

}

if(mid \* mid <= x){

res = mid;

i = mid + 0.00000001;

}else{

j = mid - 0.00000001;

}

}

System.out.println(res);

#### [32. 最长有效括号](https://leetcode-cn.com/problems/longest-valid-parentheses/)

用栈 栈存的是每个”(”的索引 刚开始要先push(-1) 且栈不能为空 否则会出错

当为空时 把当前的”)”索引push进去

#### [678. 有效的括号字符串](https://leetcode-cn.com/problems/valid-parenthesis-string/)

用两个栈实现

一个存”(”的索引

一个存”\*”的索引

当遇到”(”或”\*”时,push

当遇到”)”从”(”栈中找或从”\*”栈中找 没找到就是false

处理完后再看两个栈的情况 \*可以用来匹配(的栈 但(一定要在前面

#### [221. 最大正方形](https://leetcode-cn.com/problems/maximal-square/)

#### [1277. 统计全为 1 的正方形子矩阵](https://leetcode-cn.com/problems/count-square-submatrices-with-all-ones/)

动态规划

定义dp[i][j]为以matrix[i][j]为右下角的正方形最大边长

递推公式

dp[i][j] = min(matrix[i][j-1],matrix[i-1][j],matrix[i-1][j-1]) + 1

根据题目要求返回相应的值即可

#### [22. 括号生成](https://leetcode-cn.com/problems/generate-parentheses/)

回溯法

注意回溯函数前面进行不符合条件的剪枝

#### [162. 寻找峰值](https://leetcode-cn.com/problems/find-peak-element/)

二分法

nums[-1] = nums[n] = -∞，这就代表着 只要数组中存在一个元素比相邻元素大，那么沿着它一定可以找到一个峰值



#### [209. 长度最小的子数组](https://leetcode-cn.com/problems/minimum-size-subarray-sum/)

 两个循环 O(n\*n)

遍历每个下标作为子数组的开始下标，对于每个开始下标，需要遍历其后面的下标得到长度最小的子数组。

#### 前缀和 + 二分查找O (nlog(n)) 没掌握

双指针 O(n)

定义两个指针 start end 表示子数组滑动窗口的开始和结束位置，维护一个sum存储子数组的总和



#### [48. 旋转图像](https://leetcode-cn.com/problems/rotate-image/)

翻转图像矩阵90度

先水平翻转 再对角翻转

#### [470. 用 Rand7() 实现 Rand10()](https://leetcode-cn.com/problems/implement-rand10-using-rand7/)

(randX() - 1)\*Y + randY() 可以等概率的生成[1, X \* Y]范围的随机数

小知识：不断random直到找到1-X的数 这个1-X的数也是均匀分布的



字节教育变形题

算法3  Leetcode 470. 用 Rand7() 实现 Rand10() 的变形 rand5变rand7



#### [93. 复原 IP 地址](https://leetcode-cn.com/problems/restore-ip-addresses/)

回溯法 多刷几遍

回溯函数的参数记得有小数点的计数 和子串开始下标start

for循环的核心代码

int i = start + 1的 因为考虑的是子串substring(start,i)



#### [468. 验证IP地址](https://leetcode-cn.com/problems/validate-ip-address/)

# **字节跳动高频题——圆环回原点问题**



Java代码

public int count(int n,int length){

//定义dp为从0走i步到达j的方案数

int[][] dp = new int[n+1][length];

dp[0][0] = 1;

for(int i = 1;i <= n;i++){

for(int j = 0;j < length;j++){

dp[i][j] = dp[i-1][(j-1 + length)%length] + dp[i-1][(j+1)%length];

}

}

return dp[n][0];

}

#### [4. 寻找两个正序数组的中位数](https://leetcode-cn.com/problems/median-of-two-sorted-arrays/)

时间复杂度

O(log(m+n))

求中位数，其实就是求第 k 小数的一种特殊情况

若两个数组长度和为奇数 则求的是第(len 1 + len2 + 1)/2 小数

若两个数组长度和为偶数，则求的是第(len 1 + len2 + 1)/2 + (len 1 + len2 + 2)/2 两个小数的平均值

可以化解为求两个有序数组第k小的数了

此题的扩展：补充题17. 两个有序数组第k小的数



#### [394. 字符串解码](https://leetcode-cn.com/problems/decode-string/)

一个栈 最近括号优先匹配

如果当前的不是’]’ 将字符存入栈中

如果当前的是’]’

找出对应的字符 存入list中 需要反转

找到对应的数字存入sb 转换为数字 循环将list中加入栈中

最后将栈的结果取出来 反转



还有一种递归解法

#### [74. 搜索二维矩阵](https://leetcode-cn.com/problems/search-a-2d-matrix/)

右上角走 类似于二叉搜索树

或者直接二分法

#### [240. 搜索二维矩阵 II](https://leetcode-cn.com/problems/search-a-2d-matrix-ii/)

从右上角 或左下角走类似于二叉搜索树

或两次二分法 行二分 列二分

#### [79. 单词搜索](https://leetcode-cn.com/problems/word-search/)

回溯法

#### [128. 最长连续序列](https://leetcode-cn.com/problems/longest-consecutive-sequence/)

集合 用于查询数组中是否存在此数

#### [402. 移掉 K 位数字](https://leetcode-cn.com/problems/remove-k-digits/)

单调递增栈 不严格递增的

循环里面有个while

当栈不为空 且没有移出完k个数 当前的数小于栈顶的元素

一直出栈

注意也是把0放入栈中，最后去掉前面的0即可

**[补充题23. 检测循环依赖](https://mp.weixin.qq.com/s/pCRscwKqQdYYN7M1Sia7xA" \t "https://codetop.cc/_blank)**

#### [8. 字符串转换整数 (atoi)](https://leetcode-cn.com/problems/string-to-integer-atoi/)

再刷一次 简单题

按照题目思路写即可

#### [剑指 Offer 54. 二叉搜索树的第k大节点](https://leetcode-cn.com/problems/er-cha-sou-suo-shu-de-di-kda-jie-dian-lcof/)

注意这里的k要设置为全局变量 所有递归函数共享的

#### [153. 寻找旋转排序数组中的最小值](https://leetcode-cn.com/problems/find-minimum-in-rotated-sorted-array/)

二分法 寻找mid的值 和最右边的值比较 判断l r的移动



#### [154. 寻找旋转排序数组中的最小值 II](https://leetcode-cn.com/problems/find-minimum-in-rotated-sorted-array-ii/)

二分法 和I差不多 再增加一种等于的情况



# **一个数组先升序再降序，用最优时间复杂度，求最大值** 例如[1,2,2,2,2,3,1]



**如果是严格单调的 没有重复元素**

二分法，先去中间的三个值，判断这三个值是否单调，如果不是单调，中间那个就是最大值，否则根据单调方向，找到下一段数组进行二分查找。复杂度O(logn)



#### [20. 有效的括号](https://leetcode-cn.com/problems/valid-parentheses/)

一个栈

#### [678. 有效的括号字符串](https://leetcode-cn.com/problems/valid-parenthesis-string/)

两个栈 栈存的是字符的索引

最后要比较左括号栈 和 星号栈 索引的大小

#### [9. 回文数](https://leetcode-cn.com/problems/palindrome-number/)

将回文数取一半出来，对比即可(要根据奇数偶数的情况)



**专题总结：**

# **0排列 组合回溯专题**

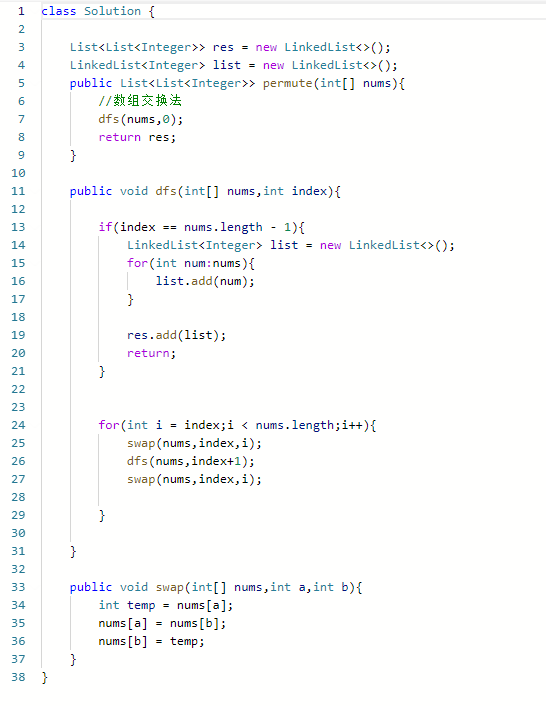
参考

<https://leetcode-cn.com/problems/subsets/solution/c-zong-jie-liao-hui-su-wen-ti-lei-xing-dai-ni-gao-/>

**全排列**

可以用回溯法做 不过去除重复的是用判断list.contains

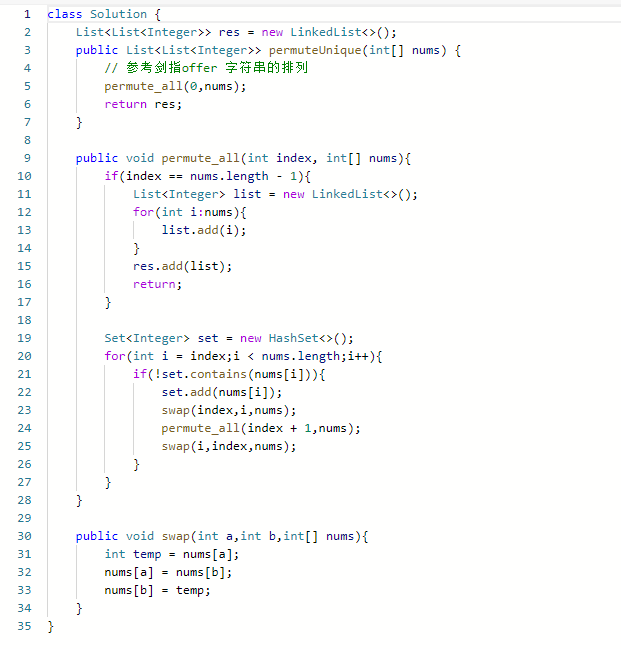
可以用**交换数组的做法 记住这个**



**全排列2**

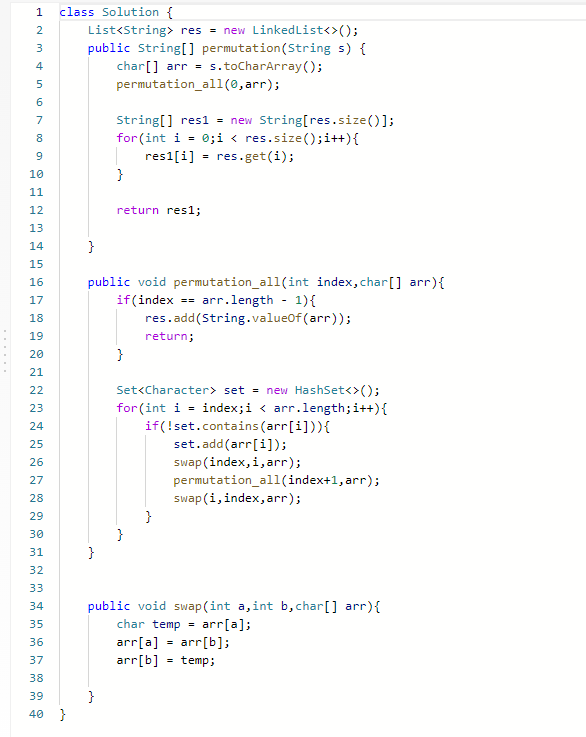
可以用剪枝的回溯法 也可以用**数组交换法(记住这个解法)** 遇到重复的元素不交换

数组交换法 在每层循环前用集合做判断



#### [剑指 Offer 38. 字符串的排列](https://leetcode-cn.com/problems/zi-fu-chuan-de-pai-lie-lcof/)

**字符数组交换法** 用集合判断是否有重复字符

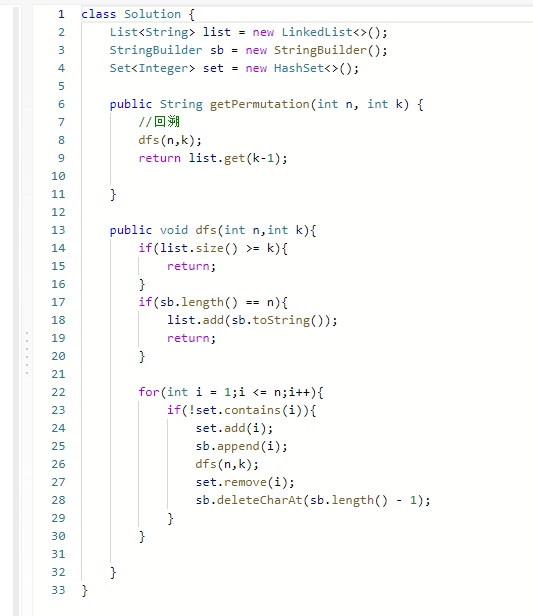


全排列2的回溯剪枝做法 和字符串的回溯剪枝做法一样的！

#### [60. 排列序列](https://leetcode-cn.com/problems/permutation-sequence/)

就是考察全排列1

用集合判断重复



#### [78. 子集](https://leetcode-cn.com/problems/subsets/)

****输入：****nums = [1,2,3]****输出：****[[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]

**利用回溯+索引进行剪枝**

**不能像全排列那样用list.contains去除重复的 这样会出现[1,2,3] [1,3,2]的**

class Solution {

    List<List<Integer>> res = new LinkedList<>();

LinkedList<Integer> list = new LinkedList<>();

    public List<List<Integer>> subsetsWithDup(int[] nums) {

        dfs(nums,0);

        return res;

}

    public void dfs(int[] nums,int index){

        res.add(new LinkedList<>(list));

        for(int i = index;i < nums.length;i++){

            list.add(nums[i]);

            dfs(nums,i+1);

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

        }

    }

}

#### [90. 子集 II](https://leetcode-cn.com/problems/subsets-ii/)

****输入：****nums = [1,2,2]****输出：****[[],[1],[1,2],[1,2,2],[2],[2,2]]

class Solution {

    List<List<Integer>> res = new LinkedList<>();

    LinkedList<Integer> list = new LinkedList<>();

    public List<List<Integer>> subsetsWithDup(int[] nums) {

        Arrays.sort(nums);

        dfs(nums,0);

        return res;

    }

    public void dfs(int[] nums,int index){

        res.add(new LinkedList<>(list));

        for(int i = index;i < nums.length;i++){

            //剪枝 画图理解

            //在当前选择列表里 有一个重复的都要去除

            if(i > index&&nums[i] == nums[i-1]){

                continue;

            }

            list.add(nums[i]);

            dfs(nums,i+1);

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

        }

    }

}

总结：排列的数组交换法 dfs(nums,index + 1)

子集的回溯法 dfs(nums,i + 1);

#### [39. 组合总和](https://leetcode-cn.com/problems/combination-sum/)

回溯法 注意如何去除重复的元素 就在第二个递归回溯不能使用以前的元素

因为可以重复利用 所以dfs(candidates,target - candidates[i],i);

不用i+1

#### 无重复元素

同一个元素可以使用多次

所以是dfs(nums,i)



即：从每一层的**第 2 个结点**开始，都不能再搜索产生同一层结点已经使用过的 candidate 里的元素。

class Solution {

    List<List<Integer>> res = new LinkedList<>();

    LinkedList<Integer> list = new LinkedList<>();

    public List<List<Integer>> combinationSum(int[] candidates, int target) {

        //回溯法

        dfs(candidates,target,0);

        return res;

    }

    public void dfs(int[] candidates,int target,int index){

        if(target < 0){

            return;

        }

        if(target == 0){

            res.add(new LinkedList<>(list));

            return;

        }

        for(int i = index;i < candidates.length;i++){

            list.add(candidates[i]);

            //去除重复值 继续递归

            dfs(candidates,target - candidates[i],i);

//不能再搜索同一层节点已经使用过的 //同一个值可以多次使用 所以是i

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

        }

    }

}

#### [40. 组合总和 II](https://leetcode-cn.com/problems/combination-sum-ii/)

元素只能使用一次 i+1

且有重复数字

去除重复值 不然后面求组合类型会出现重复

if(i > index&&candidates[i] == candidates[i-1]){

continue;

}

class Solution {

    List<List<Integer>> res = new LinkedList<>();

    LinkedList<Integer> list = new LinkedList<>();

    public List<List<Integer>> combinationSum2(int[] candidates, int target) {

        //回溯法

        Arrays.sort(candidates);

        dfs(candidates,target,0);

        return res;

}

    public void dfs(int[] candidates,int target,int index){

        if(target < 0){

            return;

        }

        if(target == 0){

            res.add(new LinkedList<>(list));

            return;

        }

        for(int i = index;i < candidates.length;i++){

            if(i > index&&candidates[i] == candidates[i-1]){

                continue;

            }

            list.add(candidates[i]);

            //去除重复值 继续递归

            dfs(candidates,target - candidates[i],i+1);

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

        }

    }

}

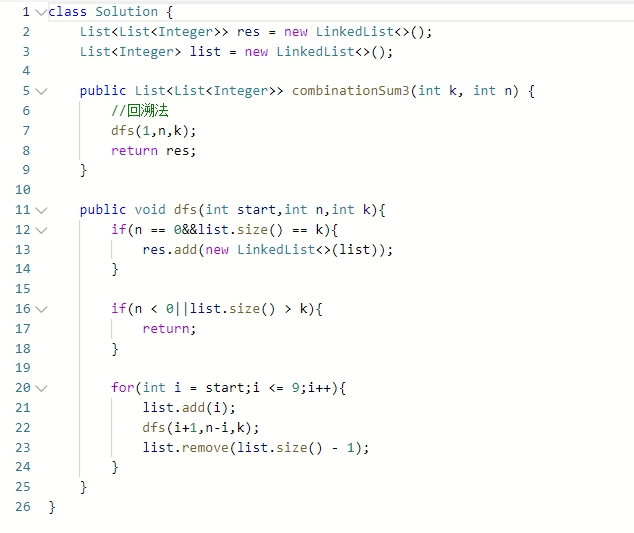
#### [216. 组合总和 III](https://leetcode-cn.com/problems/combination-sum-iii/)

回溯法

找出所有相加之和为 ****n**** 的 **k**个数的组合**。**组合中只允许含有 1 - 9 的正整数，并且每种组合中不存在重复的数字。

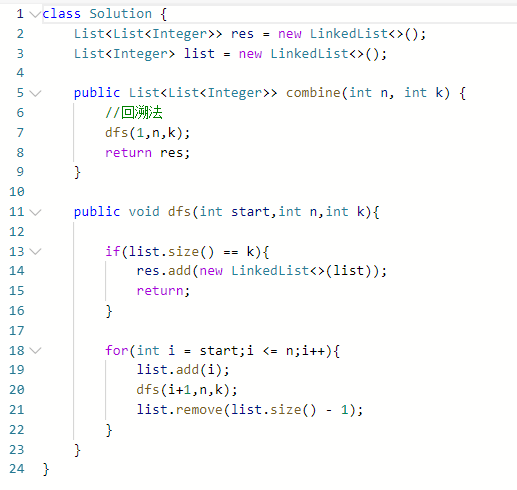
解读：每个数字只能用一次 所以是dfs(nums,i+1);

没有重复的 所以不用剪枝



#### [77. 组合](https://leetcode-cn.com/problems/combinations/)

回溯法



总结：

子集2和组合总和2的特点：

有重复元素 去重的代码都是排序，然后判断在同一层递归选择上 不选择重复的元素

//去除重复值 否则后面会出现重复

if(i > index&&candidates[i] == candidates[i-1]){

continue;

}

#### [31. 下一个排列](https://leetcode-cn.com/problems/next-permutation/)

首先从后向前查找第一个顺序对 (i,i+1)，满足 a[i] < a[i+1]。这样「较小数」即为 a[i]a[i]。此时 [i+1,n) 必然是下降序列。

如果找到了顺序对，那么在区间 [i+1,n)中从后向前查找第一个元素 j满足 a[i] < a[j]。这样「较大数」即为a[j]。

交换 a[i]与a[j]，此时可以证明区间 [i+1,n)必为降序。我们可以直接使用双指针反转区间 [i+1,n)使其变为升序，而无需对该区间进行排序。

注意用交换法去更新后面的值 这种思路在上一个排列也适用

**上一个排列**



**剑指 Offer 38. 字符串的排列**

#### 字符串 交换数组法

#### // 回溯算法 dfs(index, arr)；

#### 选择列表和路径都是数组本身 固定第index位及之前的 然后遍历交换第index位和剩下的index + 1 ... n位

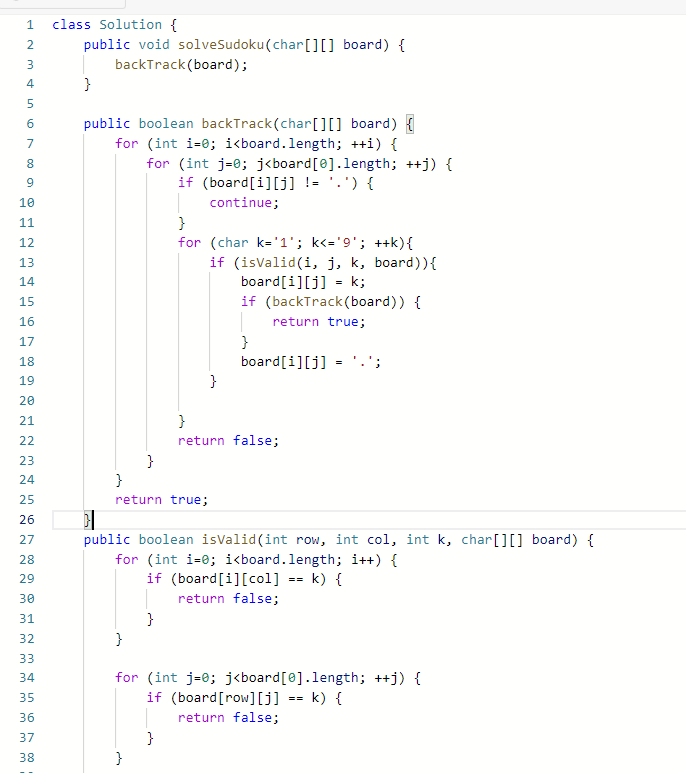
#### 若剩下的index + 1 ... n位有重复的 不交换（这里在选择路径循环定义一个集合判断）

#### 遍历交换完 继续递归回溯 固定第index+1位 然后交换回来 撤销交换

#### 解答思路用来解决leetcode 全排列2(有重复数字的) 全排列1(就是一个回溯全排列)

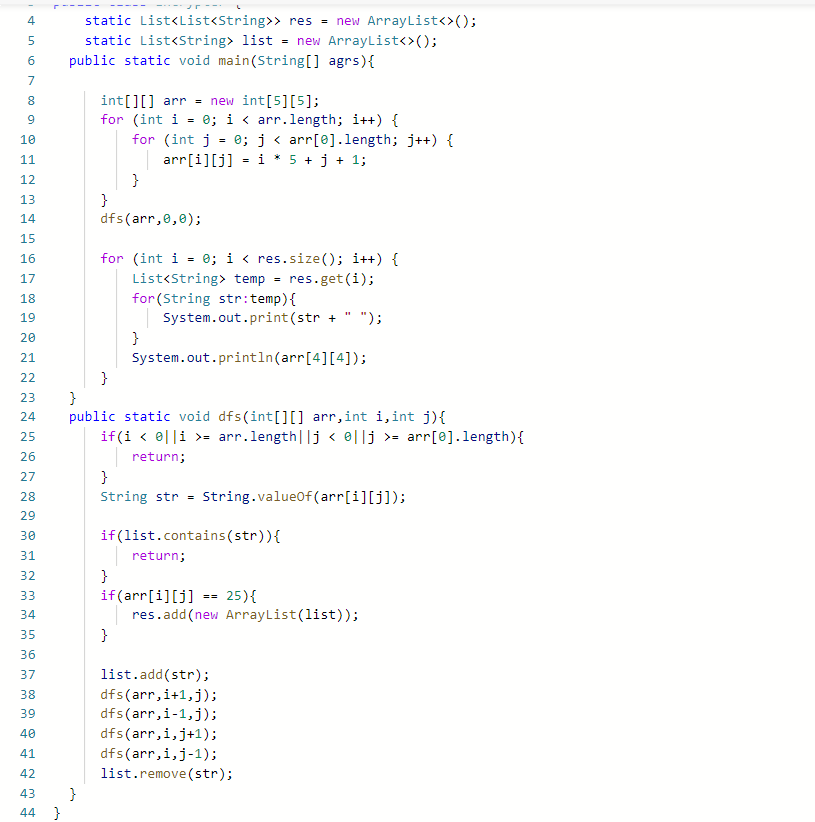
#### [37. 解数独](https://leetcode-cn.com/problems/sudoku-solver/)

回溯法



**二维数组的深度遍历 回溯**

**经过一定的路径 满足一定条件后到达终点**



# **滑动窗口专题**



#### [76. 最小覆盖子串](https://leetcode-cn.com/problems/minimum-window-substring/)

滑动窗口 双指针

定义两个Map记录滑动窗口和子串对应的字符数

WinMap记录left-right滑动窗口的字符数

从左往右每次取一个字符，更新winMap

判断是否也在needMap 若是而且winMap当前字符数满足了needMap的字符数 更新validSize

若validSize满足needMap size 再继续处理左字符串 往右边移动

而且最后一个技巧 要减少winMap里面和needMap有关的字符数



#### [438. 找到字符串中所有字母异位词](https://leetcode-cn.com/problems/find-all-anagrams-in-a-string/)

滑动窗口 双指针

和上最小覆盖子串是一个方法

#### [480. 滑动窗口中位数](https://leetcode-cn.com/problems/sliding-window-median/)



#### [剑指 Offer 48. 最长不含重复字符的子字符串](https://leetcode-cn.com/problems/zui-chang-bu-han-zhong-fu-zi-fu-de-zi-zi-fu-chuan-lcof/)

//动态规划 + 哈希表

//定义dp[i]是以s[i]结尾的最长不重复子串的长度

或//滑动窗口 + 哈希表

这道题和

[3. 无重复字符的最长子串](https://leetcode-cn.com/problems/longest-substring-without-repeating-characters/)一样

#### [498. 对角线遍历](https://leetcode-cn.com/problems/diagonal-traverse/)

先模拟正常的对角线，然后按照奇偶性进行翻转

#### [283. 移动零](https://leetcode-cn.com/problems/move-zeroes/)

双指针 从左往右交换移动非零元素

#### [224. 基本计算器](https://leetcode-cn.com/problems/basic-calculator/)

这个题目只包含（）+- 没有乘除的

栈 栈顶元素记录了当前位置所处的每个括号所「共同形成」的符号

最终元素是+- 就看经过合并后的运算符了

结果就是最终的合并后的运算符 对数字进行操作

具体看代码

#### [227. 基本计算器 II](https://leetcode-cn.com/problems/basic-calculator-ii/)

具体来说，遍历字符串s，并用变量preSign 记录每个数字之前的运算符，对于第一个数字，其之前的运算符视为加号。每次遍历到数字末尾时，根据preSign 来决定计算方式：

加号：将数字压入栈；

减号：将数字的相反数压入栈；

乘除号：计算数字与栈顶元素，并将栈顶元素替换为计算结果。

<https://www.nowcoder.com/practice/c215ba61c8b1443b996351df929dc4d4?tpId=188&&tqId=38548&rp=1&ru=/activity/oj&qru=/ta/job-code-high-week/question-ranking>

**NC137 表达式求值**

在基本计算器 II的基础上利用递归解决括号问题

https://blog.nowcoder.net/n/65fcad0be1a543359effef1228ae5d2e?f=comment



总结：关于表达式求值的话记住上面这个解法即可

#### [150. 逆波兰表达式求值](https://leetcode-cn.com/problems/evaluate-reverse-polish-notation/)

去掉括号后表达式无歧义，上式即便写成 1 2 + 3 4 + \* 也可以依据次序计算出正确结果。

适合用栈操作运算：遇到数字则入栈；遇到算符则取出栈顶两个数字进行计算，并将结果压入栈中。

1. **双指针专题**

#### [15. 三数之和](https://leetcode-cn.com/problems/3sum/)

先数组排序 双指针

两层循环 （两次去重）

先固定一个数

**去重 若固定的数之前出现过 去重重复的**

再用双指针头尾循环遍历一遍所有剩下的数 根据sum情况移动指针

Sum等于0

则把结果存入

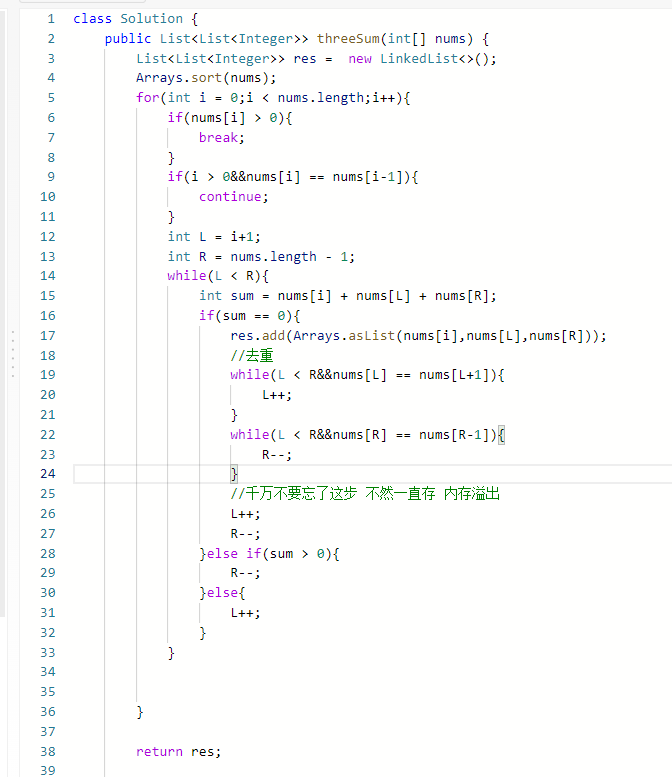
去重

若左右指针相邻的元素有相等 需要移动指针去重

不要忘了还要移动L R指针一次

Sum小于0移动左指针

Sum大于0移动右指针



#### [75. 颜色分类](https://leetcode-cn.com/problems/sort-colors/)

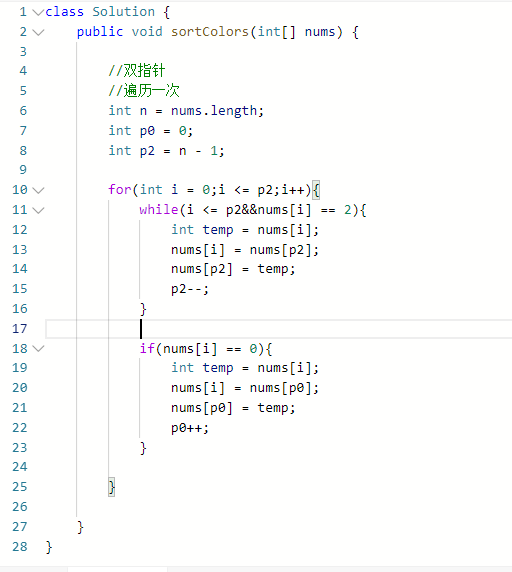
双指针

一个指向要放0的位置 p0 = 0

一个指向要放2的位置 p2 = n -1

不断找到2的nums[i] 和 nums[p2]交换 记住交换后nums[i]有可能还是2 所以要不断交换

接着若当前nums[i]为0 和nums[p0] 交换即可



还可以用刷油漆的方法

#### [11. 盛最多水的容器](https://leetcode-cn.com/problems/container-with-most-water/)

双指针

短板向内移动

可以证明的 如果长板移动 面积只会越小

#### [3. 无重复字符的最长子串](https://leetcode-cn.com/problems/longest-substring-without-repeating-characters/)

滑动窗口 双指针 遍历遇到重复的更新start指针

判断重复用哈希表



# **链表专题**

寻找链表中间节点 这个偶数时有争议 奇数时在中间

若fast = head 寻找中间节点偶数时 slow指向中间往下的一个节点 1234 3

若fast = head.next 寻找偶数中间节点偶数时 slow指向中间往上的一个节点1234 2

在排序链表中 寻找中间节点fast要初始化为head.next 不然在找中间节点 两个值时会栈溢出

在寻找环形链表2中，slow和fast要初始化为head 相交后把fast再定义为head



合并链表

#### **141 环形链表**

#### 快慢指针

#### 记得把判断放在后面 不然slow和fast开始都指向head时第一次判断就是true了(或者fast 开始定义为fast = head.next)



可以看看错误的提交

#### [142.](https://leetcode-cn.com/problems/linked-list-cycle-ii/) **[环形链表](https://leetcode-cn.com/problems/linked-list-cycle-ii/)**[II](https://leetcode-cn.com/problems/linked-list-cycle-ii/)

//快慢指针 slow和fast要初始化为head 相交后把fast再定义为head

#### // 相遇时 一个从头一步一步走 一个在相遇点一步一步走 最终相遇

这里有一个大坑 必须slow和fast初始化必须为head 不然数学推选不出来fast走的步数是slow的两倍

f = a + nb b为环大小

f = 2s

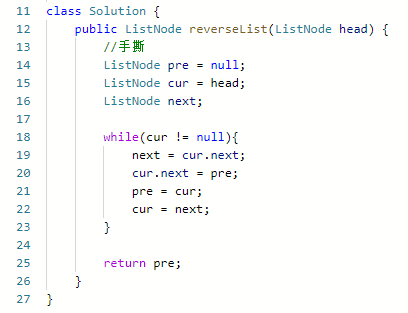
所以s = nb 由于所有环入后都为a + nb 所以s再走a步即到达环入口

由于我们不知道a 所以用双指针 再令一个指针指向head 相遇时走了a步

此时即为环入口

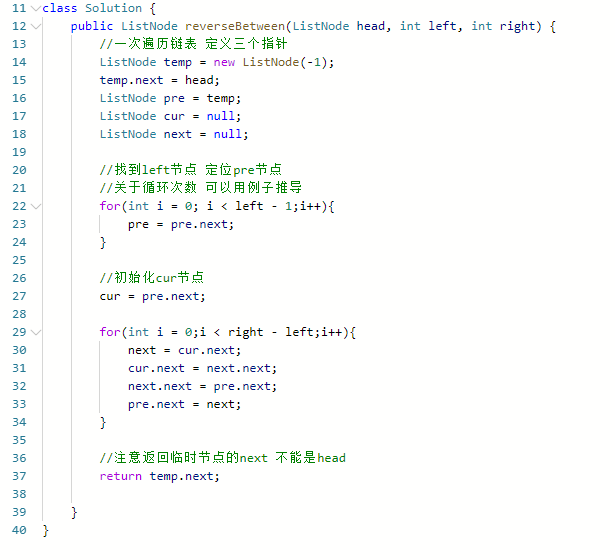
#### [206. 反转链表](https://leetcode.cn/problems/reverse-linked-list/)

三个指针



#### [92. 反转链表 II](https://leetcode-cn.com/problems/reverse-linked-list-ii/)

**记住这**种思路 边找边反转



还有一种思路

临时节点

反转区间链表 然后再连接

使用「[206. 反转链表](https://leetcode-cn.com/problems/reverse-linked-list/" \t "https://leetcode-cn.com/problems/reverse-linked-list-ii/solution/fan-zhuan-lian-biao-ii-by-leetcode-solut-teyq/_blank)」的解法，反转 left 到 right 部分以后，再拼接起来。我们还需要记录 left 的前一个节点，和 right 的后一个节点

一个需要四个指针

#### [25. K 个一组翻转链表](https://leetcode-cn.com/problems/reverse-nodes-in-k-group/)

思路： 用一个临时指针存结果头部

正常思路 先一个大循环 然后用**两个指针分别找到要反转长度的first 和 last**

然后在一个循环里反转这个区间链表

若找到的链表长度小于区间长度 不用反转 这个时候last == null 拼接起来退出大循环即可

用到的指针

ListNode temp = new ListNode(0); // 会动的

ListNode res = temp; // 储存要返回的结果指针

ListNode first = head;

ListNode last = null;



#### [234. 回文链表](https://leetcode-cn.com/problems/palindrome-linked-list/)

#### 有一种简单的方法 将链表的值复制到数组中 然后双指针比较

找到中间节点 快慢指针 fast设置为head.next

翻转 记得将前面的链表末尾 null slow.next = null

比较

#### [148. 排序链表](https://leetcode-cn.com/problems/sort-list/)

归并排序 自顶向下的 正常思路

快慢指针找到中间节点

注意中间节点时 **fast的初始化为head.next** 不然剩余两个节点划分不了

划分左右两个链表 这里要将slow.next = null

归并合并左右链表 返回 正常思路

迭代排序 自底向上排序 还没掌握

快速排序 按照快速排序的思路写 框架大体相同 详细看代码

#### [147. 对链表进行插入排序](https://leetcode-cn.com/problems/insertion-sort-list/)

定义一个临时节点

具体思路看代码

定义两个指针

当前排序好的链表尾部节点

下一个要排序的节点

判断要排序的节点本来就在正确位置 不用插入了

若不在 从临时节点开始找到要插入的正确位置 进行插入即可

#### [23. 合并K个升序链表](https://leetcode-cn.com/problems/merge-k-sorted-lists/)

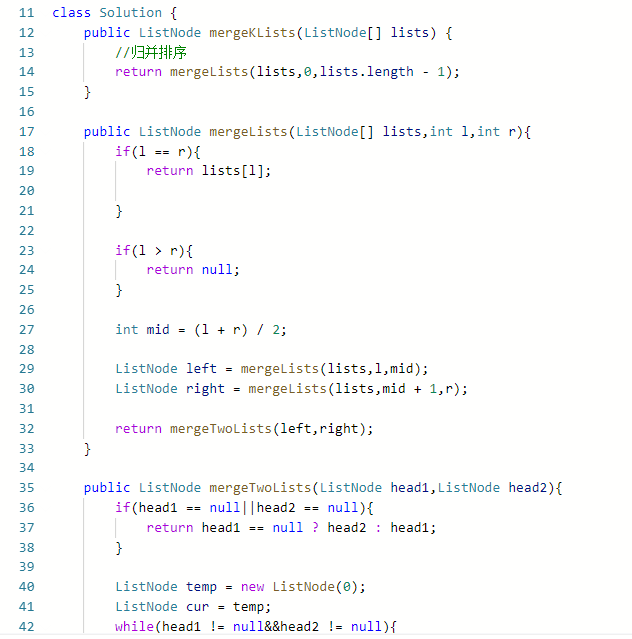
归并排序

归并函数：

从中间两两划分链表数组

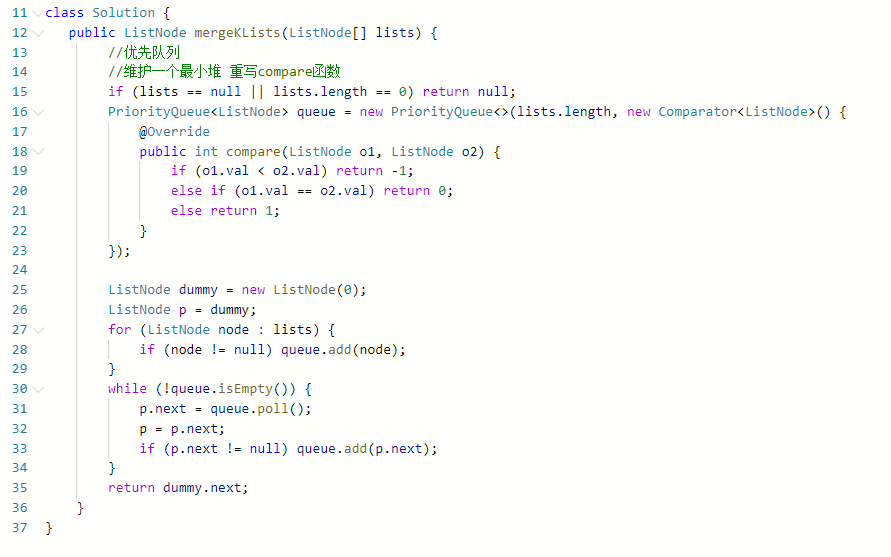
然后合并左右两边的链表 返回合并后的结果

再定义一个合并两个链表的函数 简单



记住这个 比较好写

优先队列 定一个最小堆 存每个排序链表的头





#### [24. 两两交换链表中的节点](https://leetcode-cn.com/problems/swap-nodes-in-pairs/)

建立临时节点 指向head

然后在链表中交换指针指向 可以反转两个节点 一直处理即可

#### [83. 删除排序链表中的重复元素](https://leetcode-cn.com/problems/remove-duplicates-from-sorted-list/)

遍历

#### [82. 删除排序链表中的重复元素 II](https://leetcode-cn.com/problems/remove-duplicates-from-sorted-list-ii/)

一次遍历 临时节点

定义pre cur 节点 不断更新

递归版本的

递归函数删除以 head 作为开头的有序链表中，值出现重复的节点

返回删除了值重复的节点后剩余的链表

#### [138. 复制带随机指针的链表](https://leetcode-cn.com/problems/copy-list-with-random-pointer/)

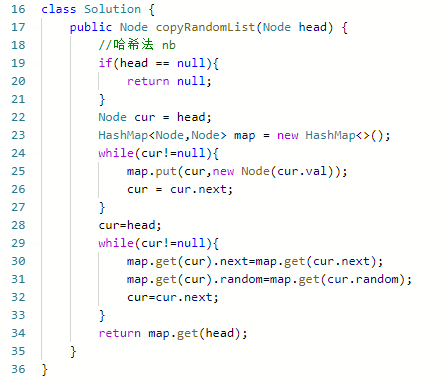
复制链表

复制随机指针

拆分链表

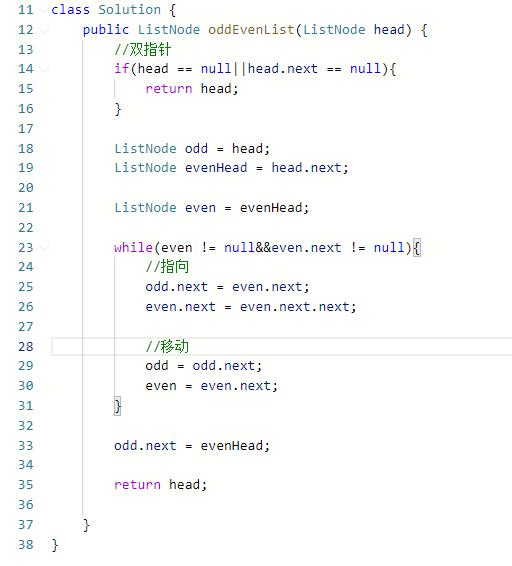
也可以用哈希法yyds

将新创建的节点和原来的节点对应起来 这样就可以找到next 和random指针了



#### [328. 奇偶链表](https://leetcode-cn.com/problems/odd-even-linked-list/)

奇数偶数双指针 可以画图理解指向 移动问题



# **字节跳动高频题——排序奇升偶降链表**

**https://mp.weixin.qq.com/s/0WVa2wIAeG0nYnVndZiEXQ**

# 分，反转，合并

# step1:leetcode328奇偶链表，去掉两链表合并的步骤； 分的过程和这道题一样 末尾odd.next = null；

# step2：leetcode206反转链表；

# step3:leetcode21合并两个有序链表；

#### [143. 重排链表](https://leetcode-cn.com/problems/reorder-list/)

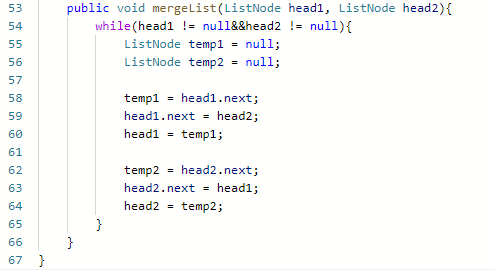
线性表

寻找中间节点 middle.next = null

翻转后面的节点 用middle.next进行翻转

合并链表 奇数个节点时 前半段要比后半段多一 所以寻找中间节点时 fast = head





#### [21. 合并两个有序链表](https://leetcode-cn.com/problems/merge-two-sorted-lists/)

临时节点 简单

#### [剑指 Offer 22. 链表中倒数第k个节点](https://leetcode-cn.com/problems/lian-biao-zhong-dao-shu-di-kge-jie-dian-lcof/)

快慢指针 快指针先走k步

#### [160. 相交链表](https://leetcode-cn.com/problems/intersection-of-two-linked-lists/)

注意循环里的写法 不相交要跳出循环的 **p == null**而不是p.next == null

#### [19. 删除链表的倒数第 N 个结点](https://leetcode-cn.com/problems/remove-nth-node-from-end-of-list/)

临时节点 快指针

一次遍历 快指针先走n-1步

快慢指针找到删除节点 用temp暂存slow前面的节点

用临时节点解决一些特殊情况

#### [450. 删除二叉搜索树中的节点](https://leetcode-cn.com/problems/delete-node-in-a-bst/)

先递归找到要删除的节点

当找到要删除的节点后，

删除的节点类型要分三种情况。

如果删除的结点不是叶子节点，并且有一个子节点为空或为空（此时节点为叶子节点），我们直接返回另一个不为空或为空的子节点即可。

如果删除的结点不是叶子节点，并且左右子树都不为空，我们可以用左子树的最大值替换掉要删除的节点或者用右子树的最小值替换掉要删除的节点都是可以的。

然后继续递归删除相应左子树的最大值 或 右子树的最小值



#### [61. 旋转链表](https://leetcode-cn.com/problems/rotate-list/)

循环旋转，但其实本质上是将尾部向前数第K个元素作为头，原来的头接到原来的尾上

去掉循环的规律

找到第n - k个节点 处理节点链接即可 注意要先把尾部链接到头部

**注意此时尾部的节点有可能为最后一个 所以先拼接到头部节点**

#### [146. LRU 缓存机制](https://leetcode-cn.com/problems/lru-cache/) 掌握1

哈希双向链表法

缓存是在双向链表里面的 每个节点存有key和value (key是为了删除哈希的值设置的)

双向链表定义虚拟的头尾节点

哈希map定义HashMap<Integer,Node>

//定义数据结构

int cap;

HashMap<Integer,Node> map = new HashMap<Integer,Node>();

DoubleList doubleList = new DoubleList();

doubleList实现 addFirst remove removeLast函数

LRU实现get put 函数

#### [460. LFU 缓存](https://leetcode-cn.com/problems/lfu-cache/) 最近最不经常使用

两个哈希

一个key node 哈希

一个 频率 双向链表哈希

（node 在LRU的基础上增加一个频率值属性）

（双链表和LRU的一样）

还要维护一个最小频率 容量满了要删除最小频率哈希双向链表的最后一个node



LFU要实现get put函数

！！还要实现一个changeFre(Node node)

（每次get 或者put的时候 节点的频率会发生变化 所以需要changFre去更新频率双向链表哈希里的双向链表）

put函数里，若容量满了 移除频率最低的链表最后一个节点 最久未使用的



#### LeetCode 热题 HOT 100

#### 关于链表的题目

和上面重复了

#### [2. 两数相加](https://leetcode-cn.com/problems/add-two-numbers/)

两个链表表示的大数相加

循环处理 计算进位 记得处理最后的进位是否不为0

**4单调队列专题 单调栈**

#### [239. 滑动窗口最大值](https://leetcode.cn/problems/sliding-window-maximum/)

单调双端队列

//单调队列维护的是滑动窗口内大于num[i]的数 的索引!

每次新增一个数字 需要将单调队列中**小于等于**该数字的索引值去掉

然后将数字加入单调队列中 判断单调队列最大的首位值 索引是否在滑动窗口范围内

最后滑动窗口长度大小大于等于k的时候 将最大的索引值存入结果中

这道题关键是索引不要用错 nums[索引]

判断单调队列最大的首位值 索引是否在滑动窗口范围内 i - k + 1 > q.peekfirst()

不能用单调队列的长度做判断



#### **[862. 和至少为 K 的最短子数组](https://leetcode-cn.com/problems/shortest-subarray-with-sum-at-least-k/)**

前缀和 双端单调递增队列

记住两个性质

要维护一个前缀和**单调严格递增**的队列 若前缀和递减，则前一个P[x]的x不可能是答案

若当前的x满足条件 y1 - x 在第一个条件下，这个x的情况可以去掉了 下一个不可能有y2 - x < y1 - x的（原因y2 > y1）



有一道题题目差不多 不是单调队列做法的

#### [560. 和为 K 的子数组](https://leetcode-cn.com/problems/subarray-sum-equals-k/)

#### 前缀和 哈希

这里用到了前缀和 利用以i结尾的子数组和等于k的个数刚好等于前缀和pre[j]== pre[i] - k 的个数

这里注意一下代码中的顺序 先求子数组和的个数再哈希map.put(pre,map.getOrDefault(pre,0) + 1);



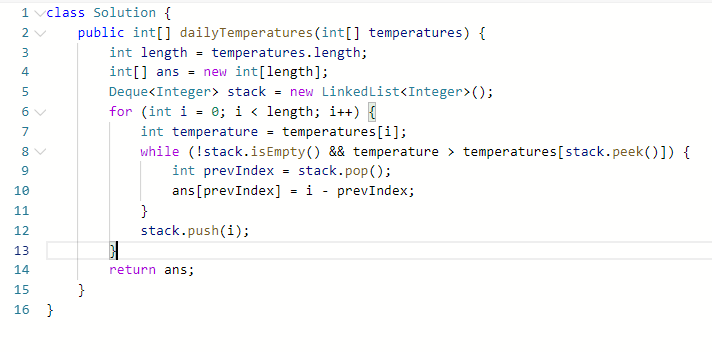
#### [739. 每日温度](https://leetcode-cn.com/problems/daily-temperatures/)

可以暴力 两个循环

单调栈

定义一个温度单调递减的栈 存温度数组下标

当前温度比栈顶元素大 则一直出栈 更新结果 然后把当前温度入栈即可



#### [316. 去除重复字母](https://leetcode-cn.com/problems/remove-duplicate-letters/)

#### 和[1081. 不同字符的最小子序列](https://leetcode.cn/problems/smallest-subsequence-of-distinct-characters/)一样

单调栈 + 字符计数器

使用布尔数组记录栈中的元素，实现去重

为了实现字典序最小 需要不断pop出后面还有的元素，若后面元素没有（指栈顶元素），不能pop出去

通过一个计数器记录后面的元素还有多少



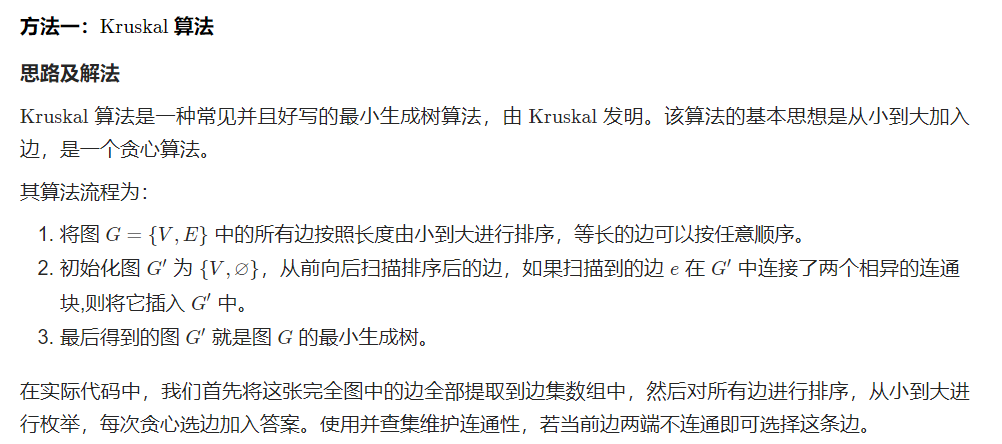
**5.图专题**

**最小生成树**

#### [1584. 连接所有点的最小费用](https://leetcode-cn.com/problems/min-cost-to-connect-all-points/)

**Kruskal算法 克鲁斯卡尔**

用到并查集



无向图

**Prim算法 普里姆**

难点在于如何找到权值最小的边 其中u为集合new的元素 v不是集合new的元素



class Solution {

    public int minCostConnectPoints(int[][] points) {

        // Prim算法

        int n = points.length;

        Queue<int[]> edges = new PriorityQueue<>((int[] o1,int[] o2)->{return o1[2] - o2[2];});

        //遍历每两个顶点构建无向图

        //第一个顶点i

        for(int i=0;i<n;i++) {

            //第二个顶点i+1

            for(int j=i+1;j<n;j++) {

                int xi = points[i][0],yi = points[i][1];

                int xj = points[j][0],yj = points[j][1];

                int weight = Math.abs(xi - xj) + Math.abs(yi - yj);

                edges.add(new int[]{i,j,weight});

            }

        }

        //最小生成树权值

        int mst = 0;

        HashSet<Integer> set = new HashSet<>();

        set.add(0);

        //暂存还没加入最小生成树的节点 构成的边集合

        // prim算法就是找到离已知节点的 边最小的节点 加入到已知节点中

        List<int[]> list = new LinkedList<>();

        while (!edges.isEmpty()&&set.size() < n){

            int[] edge = edges.poll();

            if(set.contains(edge[0])&&set.contains(edge[1])){

                continue;

            }else if(!set.contains(edge[0])&&set.contains(edge[1])){

                set.add(edge[0]);

                mst += edge[2];

                for(int[] E:list){

                    edges.add(E);

                }

                list.clear();

            }else if(set.contains(edge[0])&&!set.contains(edge[1])){

                set.add(edge[1]);

                mst += edge[2];

                for(int[] E:list){

                    edges.add(E);

                }

                list.clear();

            }else{

// 有可能最小边的没有一个节点在确定的节点集合set中

                list.add(edge);

            }

        }

        return mst;

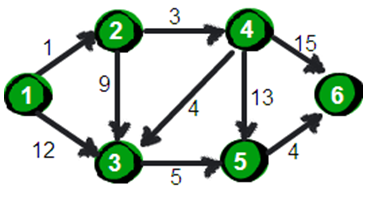
    }

}

**最短路径**

Dijkstra求点到其他节点的最短路径

<https://blog.csdn.net/lbperfect123/article/details/84281300>



**import** java.util.\*;

**class** Solution {

**public static void** main(String[] args){

// 八个节点

**int** N = 8;

*//有向图*

**int**[] A = **new int**[]{1,1,2,2,3,4,4,4,5};

**int**[] B = **new int**[]{2,3,4,3,5,3,5,6,6};

**int**[] W = **new int**[]{1,12,3,9,5,4,13,15,4};

**int**[] H = **new int**[]{2,4};

**int**[][] map = **new int**[N+1][N+1];

**for**(**int**[] ma:map){

Arrays.*fill*(ma,Integer.***MAX\_VALUE***);

}

**for**(**int** i=1;i<=N;i++)

{

map[i][i]=0;

}

**for**(**int** i = 0;i < A.**length**;i++) {

*//边权重都为1*

map[A[i]][B[i]] = W[i];

}

Solution s = **new** Solution();

**int**[] vis = **new int**[N+1];

**int**[] dis = **new int**[N+1];

Arrays.*fill*(dis,Integer.***MAX\_VALUE***);

Arrays.*fill*(vis,0);

// 1 为 起点

dis[1] = 0;

s.Dijkstra(N,1,dis,vis,map);

**for**(**int** i:dis){

System.***out***.println(i);

}

}

*/\**

*N 是节点数量*

*u是最短路径的起始点*

*dis[N + 1] 是确定的最短路径点集合*

*vis[N+1] 是最短路径集合的点是否已经访问过*

*map[N+1][N+1] 是路径图*

*\*/*

**public void** Dijkstra(**int** N,**int** u,**int**[] dis,**int**[] vis,**int**[][] map) {

**for**(**int** t = 1; t < N; ++t) {

**int** minn= Integer.***MAX\_VALUE***;

**int** temp = 0; //不能初始化为-1

**for**(**int** i=1;i<=N;i++) {

*//从没确定最短的的一个节点选择最小的*

**if**(vis[i] == 0&&dis[i]<minn)

{

minn=dis[i];

temp=i;

}

}

vis[temp]=1;

**for**(**int** i=1;i<=N;i++)

{

*//map[temp][i]+dis[temp] 溢出了*

**if**(((**long**)map[temp][i]+(**long**)dis[temp])<(**long**)dis[i])

{

dis[i]=map[temp][i]+dis[temp];

}

}

}

}

}

#### **[拓扑](https://leetcode-cn.com/problems/course-schedule-ii/)排序**

只要你会BFS，会层次遍历二叉树。

你很快就能掌握拓扑排序的写法。

就是把入度0度的节点放入队列中

然后从队列中取出节点 更新入度数组 记得把入度为0的节点加入队列即可

重复上面步骤

存在循环时，循环里的节点不存在有入度为0的，拓扑排序的节点数量不等于总的节点数，所以拓扑排序也可以用来检测环 循环

#### **[210. 课程表 II](https://leetcode-cn.com/problems/course-schedule-ii/)**

拓扑排序 入度数组+哈希集合数组邻接表

拓扑排序的一个附加效果是：****能够顺带检测有向图中是否存在环****

****思路： 广度优先遍历****

首先定义哈希集合数组，储存每个节点指向的节点集

HashSet<Integer>[] adj = new HashSet[numCourses];

定义入度数组，储存每个节点的入度

定义一个队列 存入度为0的节点（这里的节点就是节点的编号）

每次从队列中取出入度为0 的节点 从哈希数组中拿出指向节点集

将指向节点集的节点入度都减1 然后将入度为0的节点入栈

#### **[207. 课程表](https://leetcode-cn.com/problems/course-schedule/)**

方法同课程表2 修改返回值即可

# **补充题：检测循环依赖**

**华为笔试 服务启动 在下面**

#### **[743. 网络延迟时间](https://leetcode-cn.com/problems/network-delay-time/)**

**Dijkatra算法**

N 是节点数量

u是最短路径的起始点

dis[N + 1] 是确定的最短路径点集合 dis[u] = 0;

vis[N+1] 是最短路径集合的点是否已经访问过

map[N+1][N+1] 是路径图



#### **[785. 判断二分图](https://leetcode-cn.com/problems/is-graph-bipartite/)**

广度优先遍历+染色

每访问一个节点 将其相邻节点染相反色 入队列



#### [133. 克隆图](https://leetcode.cn/problems/clone-graph/)

深度优先遍历 + 哈希表



6二叉树专题

#### 剑指 Offer 32 - I. 从上到下打印二叉树

#### 剑指 Offer 32 - II. 从上到下打印二叉树 II

#### 剑指 Offer 32 - III. 从上到下打印二叉树 III

#### 广度优先搜索 二叉树

#### 广度优先遍历 队列实现 用数组存结果

#### [剑指 Offer 26. 树的子结构](https://leetcode-cn.com/problems/shu-de-zi-jie-gou-lcof/)

#### 深度优先搜索

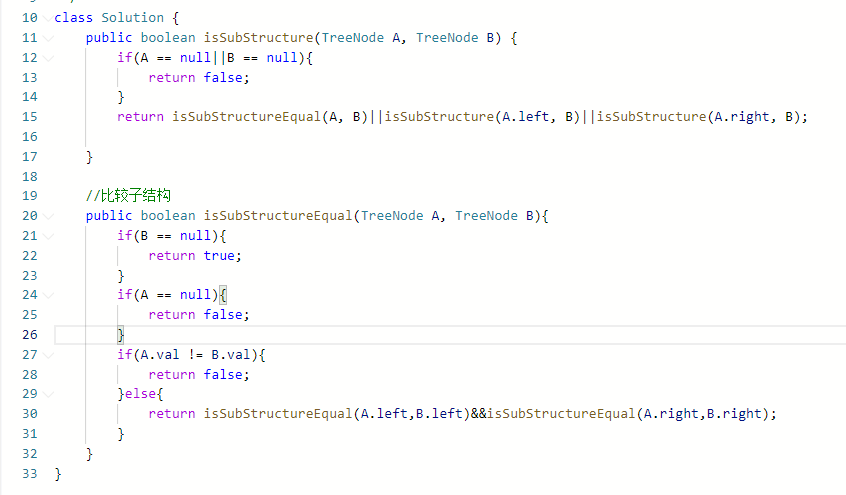
#### dfs深度遍历树的每一个节点(可以在原函数操作 )

#### 再用一个递归函数比较当前节点以root的子树和子结构是否一致

#### 比较子结构的递归函数

#### 当递归到子结构的树A为null时返回true

#### B为null或值不相等返回false



#### 

#### 剑指 Offer 27. 二叉树的镜像

#### 递归 先交换左右节点值 再递归左右子树 返回root

#### [剑指 Offer 28. 对称的二叉树](https://leetcode-cn.com/problems/dui-cheng-de-er-cha-shu-lcof/)

#### 递归 在第二个函数递归比较根节点的左右子树

#### 剑指 Offer 12. 矩阵中的路径

#### 数组 回溯 矩阵

#### 剑指 Offer 13. 机器人的运动范围

#### 深度优先搜索&广度优先搜索

#### \*\*第 15 天\*\*

#### 搜索与回溯算法（中等）

#### 剑指 Offer 34. 二叉树中和为某一值的路径

#### 深度优先搜索 回溯

#### 深度优先遍历 若当前节点为叶节点且targetSum等于当前节点 记录当前路径

#### 在递归中先将节点值加入路径 然后递归左右子树 再移出当前节点值

#### 剑指 Offer 36. 二叉搜索树与双向链表

#### 栈 树 深度优先搜索

#### //中序遍历依次递增 存头节点 当前节点 最后处理首尾节点指向即可

每次遍历对root节点进行处理即可

dfs(root.left)

root

dfs(root.right)

#### 剑指 Offer 54. 二叉搜索树的第k大节点

#### 深度优先搜索 二叉搜索树

#### \*\*第 18 天\*\*

#### 搜索与回溯算法（中等）

#### 剑指 Offer 55 - I. 二叉树的深度

#### 递归实现

#### 剑指 Offer 55 - II. 平衡二叉树

#### 深度优先搜索

#### //递归实现 遍历每一个节点 比较左右子树的高度差

#### 111 二叉树的最小深度 最小深度注意判断当左右节点有一个为null时 递归返回另一个边的深度+1

#### 104 二叉树的最大深度 和二叉树的深度一样

#### 正常返回左右子树的最大深度 +1

#### \*\*第 19 天\*\*

#### 搜索与回溯算法（中等）

#### 剑指 Offer 64. 求1+2+…+n

#### 位运算

#### 递归

#### LeetCode 热题 HOT 100

#### \*\*关于树的题目\*\*

#### 652 寻找重复的子树

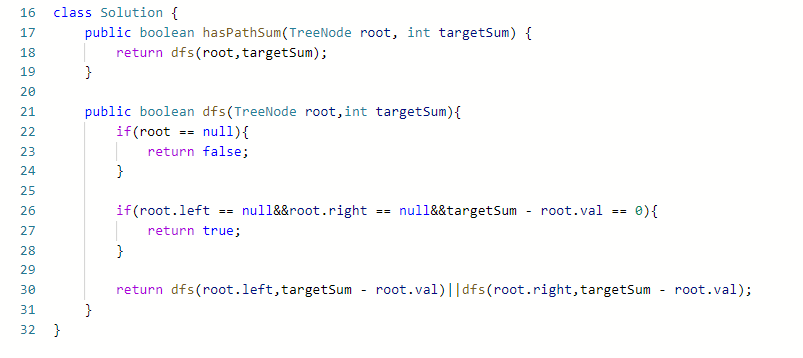
#### 序列化二叉树 只能前序或后续遍历 利用map记录每个根节点的序列化字串 若系列化字串数量大于2就是重复的子树

**112 路径总和**

#### 递归实现

#### 若当前节点为叶节点且targetSum等于当前节点 返回ture

#### 否则return 用或语句递归左右子树



**113 路径总和 II 求路径 和剑指 Offer 34. 二叉树中和为某一值的路径一样**

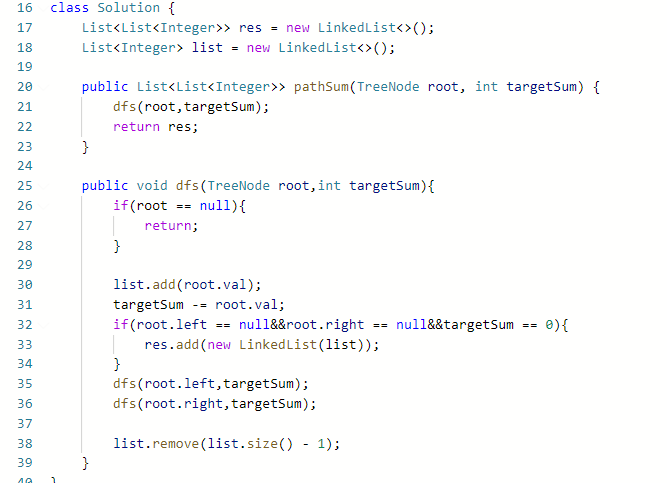
递归 类似回溯

把当前节点加入路径

判断是否为路径 当前节点为叶子节点且总和等于目标和

继续递归左右子树

移出路径加入的节点



**437 路径总和 III 求路径等于targetSum的数量**

#### 两个递归

#### 主函数递归到每一个节点 以当前节点为根节点递归寻找路径总和

#### 第二个递归函数和 112 路径总和差不多（不要求是叶节点） 当targetSum等于当前节点值 记录路径数量



动态规划之打家劫舍

#### <https://blog.csdn.net/weixin_45003125/article/details/119335848>

#### [198. 打家劫舍](https://leetcode-cn.com/problems/house-robber/)

定义dp[i]为连续0-i房屋能偷窃的最大金额

递推公式为dp[i] = Math.max(dp[i - 1], dp[i - 2] + nums[i])

变种：输出路径 字节面试

dp数组定义：dp[i][0]代表不偷第i号房屋所能获得最大金额

dp[i][1]代表偷第i号房屋所能获得最大金额

递推公式：

不偷第i号房屋所能获得最大金额就是一定得偷第i-1号房屋也就是dp[i][0] = dp[i - 1][1]

偷第i号房屋所能获得最大金额有两种情况：

偷i-2号房屋，这个好理解

不偷i-2号房屋，比如说[2，1，1，2]

dp[i][1] = Math.max(dp[i - 2][1] + nums[i], dp[i - 2][0] + nums[i])

计算路径：得出dp数组后，因为是要计算偷了那几家，因此路径一定是偷了的那家，也就是dp[i][1]，然后倒序遍历，如果偷了第i家的最大金额等于target，那就记录路径

#### [213. 打家劫舍 II](https://leetcode-cn.com/problems/house-robber-ii/)

和第一题差不多

偷了最后一家不能偷第一家【1,n-1】

不偷最后一家 第一家无所谓 所以是【0,n-2】

按照打家劫舍分别计算  取两者中的最大值

#### 337 打家劫舍 III 动态规划

二叉树

如果两个直接相连的房子在同一天晚上被打劫，房屋将自动报警

//定义一个函数返回以每个节点为根的子树最高偷窃金额 分两种情况 选节点的值或不选节点的值 函数返回值有两个

#### 以根节点为子树的最高金额有两种情况 选根节点的值 不选根节点的值

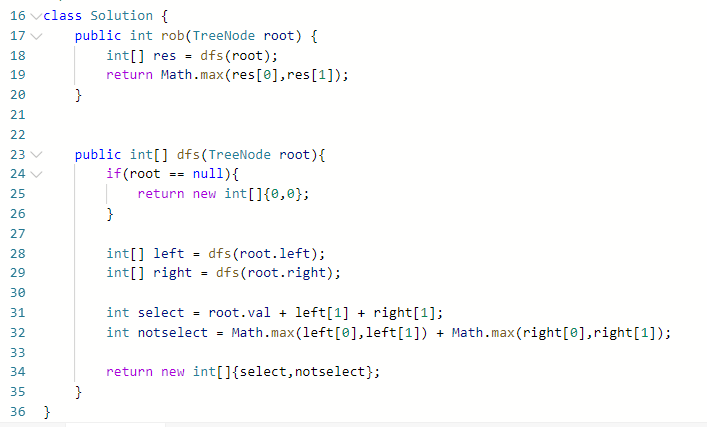
#### 选根节点的值 左右节点的值就不能选

#### 不选根节点的值 左右节点的值可选可不选 取最大值

#### 以根节点为子树的最高金额有两种情况 选根节点的值 不选根节点的值

#### 选根节点的值 左右节点的值就不能选

#### 不选根节点的值 左右节点的值可选可不选 取最大值



#### 236 二叉树的最近公共祖先

#### 面试经典题 递归实现 递归函数返回当前以root为根的子树是否包含 p q 返回root

#### 递归结束条件是root节点为p q节点中一个返回root

#### 若root为null 不存在 返回null

#### 否则递归左右子树 根据左右子树的递归情况判断

#### 递归函数返回 是否包含 p q节点的子树 根节点



#### 

#### 扩展题 二叉搜索树的最近公共祖先

#### 利用二叉树的性质 判断root.val 和 p q的大小关系 得到最近公共祖先在左右子树的哪一边

#### [剑指 Offer 37. 序列化二叉树](https://leetcode-cn.com/problems/xu-lie-hua-er-cha-shu-lcof/)

#### 297 二叉树的序列化与反序列化

#### // 深度优先遍历

#### 序列化用前序遍历处理 返回前序遍历的字符串

#### 反序列化 将字符串转化为字符队列 然后dfs处理

#### 在dfs函数中 取出一个字符 若为null返回null 然后创建字符值的节点 左右子树分别dfs剩下的字符队列

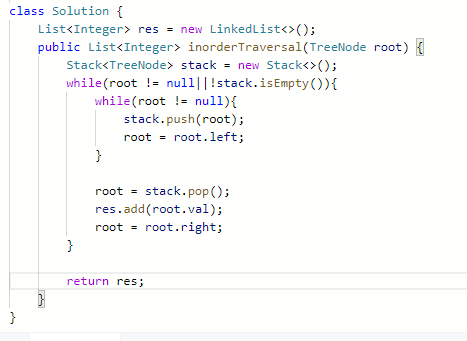


微软考到

#### 其他版本的二叉树题目：

#### [94. 二叉树的中序遍历](https://leetcode-cn.com/problems/binary-tree-inorder-traversal/)

用迭代的方式实现 用到栈



#### [144. 二叉树的前序遍历](https://leetcode-cn.com/problems/binary-tree-preorder-traversal/)

迭代实现 写法和中序差不多



#### [145. 二叉树的后序遍历](https://leetcode-cn.com/problems/binary-tree-postorder-traversal/)

迭代的方式比较难

后序遍历比较难 在中序遍历的基础上改

判断当前要访问的节点右节点是否为空 且 没有访问过

将当前节点入栈 访问右节点

否则对出栈的节点进行访问 标记当前节点为null 这样就不会重新入栈了



#### [103. 二叉树的锯齿形层序遍历](https://leetcode-cn.com/problems/binary-tree-zigzag-level-order-traversal/)

和按之字型打印二叉树一样 广度优先遍历再反转

或者用双段队列LinkedList 存结果

#### [543. 二叉树的直径](https://leetcode-cn.com/problems/diameter-of-binary-tree/)

利用二叉树的深度

可以在边求二叉树的深度时就比较直接得当前节点的二叉树直接

没必要在主函数再递归遍历每一个节点 在每一个节点再求当前节点左右子树的最大深度来求直径

#### [226. 翻转二叉树](https://leetcode-cn.com/problems/invert-binary-tree/)

可以用递归

也可以用广度优先遍历做

#### [105. 从前序与中序遍历序列构造二叉树](https://leetcode-cn.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/)

定义一个递归函数 从前序 中序中返回构造子树的根节点

#### [106. 从中序与后序遍历序列构造二叉树](https://leetcode-cn.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/)

//思路 和根据前序中序遍历还原二叉树一样思路

//从后序最后一个找到根节点

//然后遍历中序找到根节点

//递归 中序 后序的左右子树

//返回根节点

#### [889. 根据前序和后序遍历构造二叉树](https://leetcode-cn.com/problems/construct-binary-tree-from-preorder-and-postorder-traversal/)

无法得到唯一的树

#### [199. 二叉树的右视图](https://leetcode-cn.com/problems/binary-tree-right-side-view/)

广度优先遍历 索引为q.size() - 1的时候 加入结果中

深度优先遍历也可以，先遍历右子树 再遍历左子树 每一层只添加一个值(设置一个层深度参数实现)

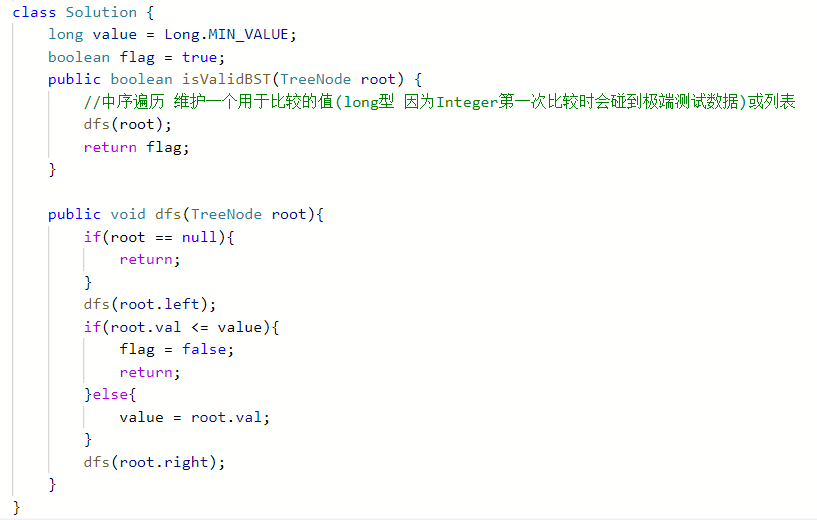
#### [958. 二叉树的完全性检验](https://leetcode-cn.com/problems/check-completeness-of-a-binary-tree/)

广度优先遍历

广度优先遍历二叉树，当出现null，后面再出现非null则不是完全二叉树 否则为完全二叉树

#### [98. 验证二叉搜索树](https://leetcode-cn.com/problems/validate-binary-search-tree/)

//中序遍历 维护一个用于比较的值(long型 因为Integer第一次比较时会碰到极端测试数据)或列表



#### [101. 对称二叉树](https://leetcode-cn.com/problems/symmetric-tree/)

递归 写一个判断函数

迭代法 队列实现

队列存左右节点，每次取出两个节点进行比较

然后把左节点的左节点 右节点的右节点

左节点的右节点 右节点的左节点放入队列中

#### [124. 二叉树中的最大路径和](https://leetcode-cn.com/problems/binary-tree-maximum-path-sum/)

定义一个函数

函数返回当前节点的最大分支(左分支或右分支)

不用再定义一个函数遍历每个节点 在返回分支最大路径和函数里已经有遍历每一个节点了

可能考最大的路径



#### [662. 二叉树最大宽度](https://leetcode-cn.com/problems/maximum-width-of-binary-tree/)

广度优先遍历

将节点的值用于保存完全二叉树的节点的位置

双端队列

**7动态规划专题**

#### 剑指 Offer 10- I. 斐波那契数列

#### 剑指 Offer 10- II. 青蛙跳台阶问题

#### 剑指 Offer 63. 股票的最大利润

#### 定义dp[i]为第i天买出的最大利润 也要维护一个最小值

#### 剑指 Offer 42. 连续子数组的最大和

#### 定义dp[i]为子数组...nums[i]连续子数组的最大和 以num[i]结尾

#### 剑指 Offer 47. 礼物的最大价值

#### 定义dp[i][j]为到达grid[i][j]最多拿到的礼物价值

#### 剑指 Offer 46. 把数字翻译成字符串

#### 定义dp[i]为字符串"num"中子串0-i的翻译种数

#### if 两个数字subtring(i-1,i+1) <=25

dp[i] = dp[i-1] + dp[i-2]

else

dp[i] = dp[i-1]

#### 剑指 Offer 48. 最长不含重复字符的子字符串

#### 第一种解法

#### 定义dp[i]是以子串...s[i]结尾的最大无重复字符串 每次用集合往回遍历 找到以s[i]结尾的最长子串 时间复杂度N\*N

记住这种

**滑动窗口 + 哈希集合**

左右指针 右指针向右移动

没有重复加入集合中 右指针向右移动 更新最大长度

遇到重复 移出左指针的元素 左指针向右移动

#### 动态规划（困难）

#### 剑指 Offer 19. 正则表达式匹配 还没做

#### 剑指 Offer 49. 丑数

#### 质数数组 是[2,3,5]

#### 313 超级丑数

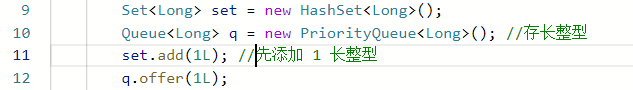
#### 质数数组primes 一般情况

#### **最小堆**

#### 简单 PriorityQueue 定义一个最小堆

#### 每次取一个最小值乘于质因子 若不重复则加入最小堆里 第n个取出的就是答案

这里有个处理细节



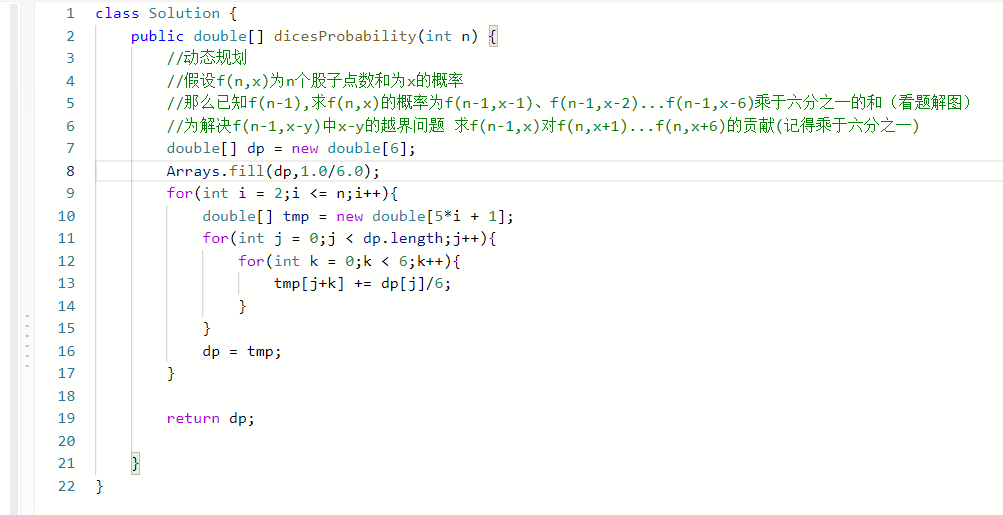
#### 剑指 Offer 60. n个骰子的点数 （着重看!!）

#### // 动态规划

#### 定义dp[n]为n个骰子的概率分布

#### 已知dp[n-1]的概率分布 根据f(n,x)的概率是 f(n-1,x-1)... f(n-1,x-6)的和除于6

#### 正向遍历 防止f(n-1,x-6)越界 就是计算f(n-1，x)的每一项对 f(n,x + k)的贡献度 k < 6



#### 剑指 Offer 14- I. 剪绳子 阿里云考过类似的题目 剪绳子的价值最大和

#### 动态规划

#### dp[i]数组表示长度为n的绳子 剪后的最大乘积

#### 剪绳子是将大段的绳子剪成小段，然后再在每小段上继续剪）

#### dp[i] = Math.max(dp[i], dp[j] \* dp[i-j]);

#### 剑指 Offer 43. 1～n 整数中 1 出现的次数 还没做

#### 递归 数学 不是动态规划做的

#### LeetCode 热题 HOT 100

#### 关于动态规划的题目

#### [322. 零钱兑换](https://leetcode-cn.com/problems/coin-change/)

凑成总金额所需的 最少的硬币个数 凑成总金额所需的 最少的硬币个数

#### 求凑成总金额所需的 最少的硬币个数

定义dp[i]凑成总金额为i所需要的最少硬币个数

if(coin <= i){

dp[i] = Math.min(dp[i],dp[i - coin] + 1);

}

#### 注意非0初始状态的定义 dp[i] = amount + 1



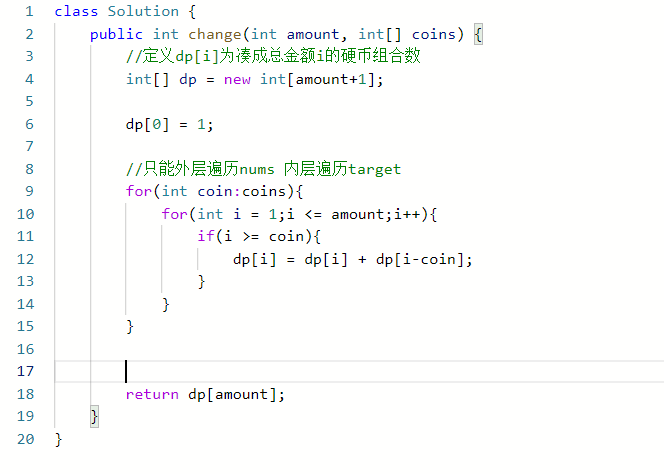
# 外层遍历nums 内层遍历target 或者相反也行

#### [518. 零钱兑换 II](https://leetcode-cn.com/problems/coin-change-2/)

#### 零钱兑换2 定义初始状态 dp[0] = 1就行

记住零钱兑换2 只能外层遍历nums 内层遍历target target就是总金额数

# 外层遍历选择nums 内层遍历目标target(就是dp数组的amount) 相反不行



# 总结以上两道题：

# **都用外层遍历nums 内层遍历target**

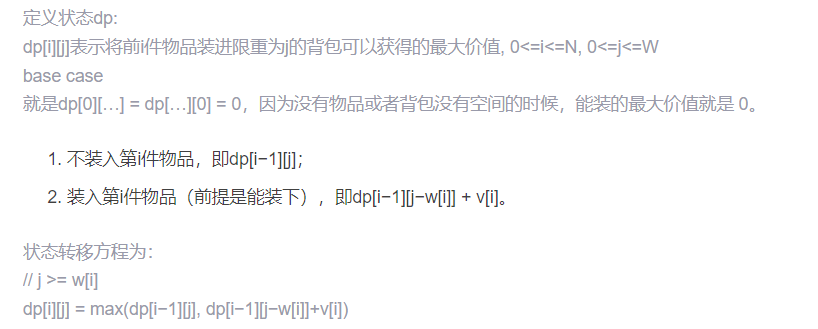
# 背包问题的第二个循环就是重量 就是目标值

上面两题都是完全背包的变形

**背包问题**

**<https://blog.csdn.net/NOTBADCODER/article/details/115373315?spm=1001.2014.3001.5501>**

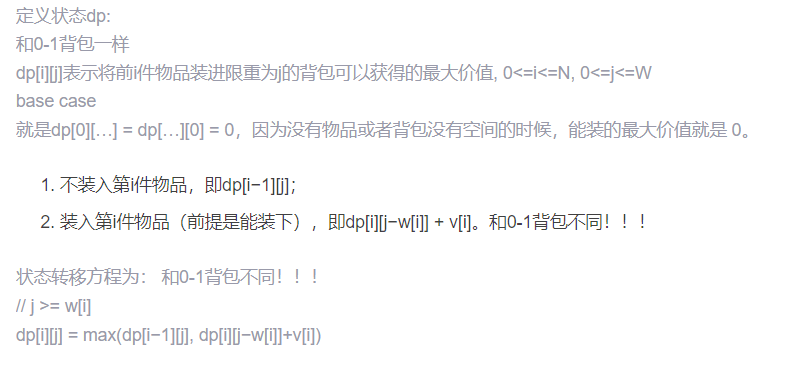
01背包



# 压缩状态 一维数组优化版

# **0-1背包 第二循环逆序**

完全背包



压缩状态 一维数组优化版

完全背包 第二循环正序

也可进行空间优化，优化后不同点在于这里的 j 只能正向枚举而01背包只能逆向枚举！！！，因为这里的max第二项是dp[i]而01背包是dp[i-1]，即这里就是需要覆盖而01背包需要避免覆盖。

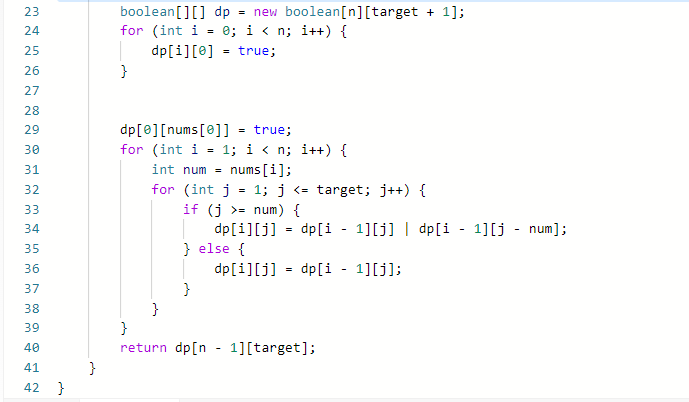
#### [416. 分割等和子集](https://leetcode-cn.com/problems/partition-equal-subset-sum/)

#### [剑指 Offer II 101. 分割等和子集](https://leetcode-cn.com/problems/NUPfPr/)

动态规划 01背包问题

传统的「0-1背包问题」要求选取的物品的重量之和不能超过背包的总容量，这道题则要求选取的数字的和恰好等于整个数组的元素和的一半。

定义dp[i][j] 表示从数组的 [0,i] 下标范围内选取若干个正整数（可以是 0 个），是否存在一种选取方案使得被选取的正整数的和等于 j



压缩版 逆序



**求划分结果版本**



#### 198 打家劫舍

#### //定义dp[i]为连续0-i房屋能偷窃的最大金额

分为偷或不偷房屋num[i]

#### 337 打家劫舍 III

#### //定义一个dfs 函数返回以每个节点为根的子树 最高偷窃金额 分两种情况 选节点的值或不选节点的值

#### 注意当不选节点的值时，左右节点的值可选可不选 取最大值

#### 494 目标和 这道题用回溯法最简单 容易理解

#### 647 回文子串 动态规划

#### // 定义dp[i][j] 子字符串i-j是否为回文子串

#### 注意外循环是j 内循环是i 这个和转移方程的判断有关 dp[i+1][j-1]

#### 可以用中心扩展法做 这个好理解 记这个

#### 扩展题 5 最长回文子串 这道题是用中心扩展法做的 就是遍历找到中心位置（一个 或 两个中心） 往两边扩展看是否为回文子串

循环遍历字符串索引 然后分别调用function(i,i)和function(i,i+1)

#### [122. 买卖股票的最佳时机 II](https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-ii/)

贪心模拟 每一个波段我都要吃 不是真实的买入卖出次数

双指针 求解每一段上升序列 注意考虑最后一段上升序列

#### 309 最佳买卖股票时机含冷冻期 好难！ 还不会 卡在理解状态转移问题上

#### 338 比特位计数 技巧问题 动态规划 奇数就是前面偶数加1 偶数二进制右移 1的个数不变

#### [72. 编辑距离](https://leetcode-cn.com/problems/edit-distance/)

动态规划 记住替换 删除 插入代表什么

**定义dp数组为word1 i位置 转换到word2 j位置所需的最少步数**

那么对于dp[i][j]

if(word1[i] == word2[j]) dp[i][j] = dp[i-1][j-1]

else dp[i][j] = min(dp[i-1][j-1]替换，dp[i-1][j]删除，dp[i][j-1]插入) + 1

#### [42. 接雨水](https://leetcode-cn.com/problems/trapping-rain-water/)

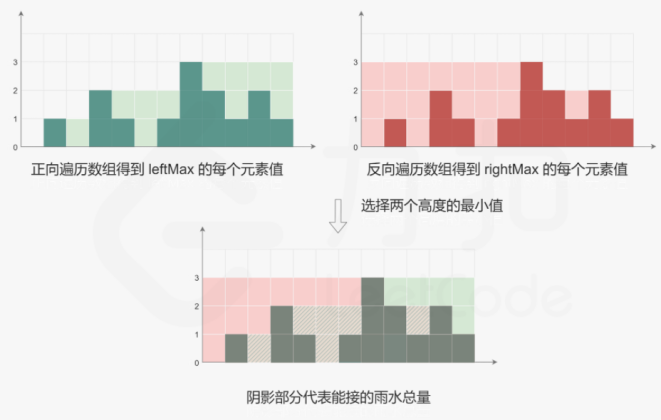
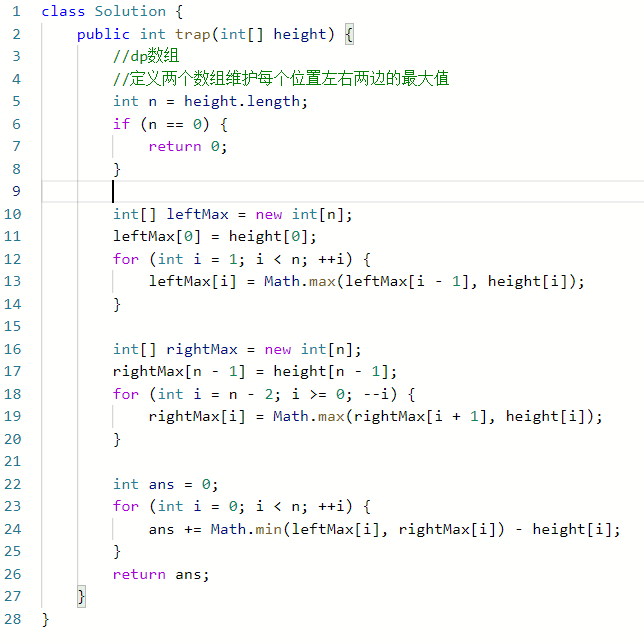
dp数组 简单

定义两个数组维护每个位置左右两边的最大值

遍历求出每个位置上面能接到的雨水

就是把左右两边的最小值 - 当前位置的高度 求和

可以将dp的解法换成简化版的双指针

**子序列问题小专题**

#### [300. 最长递增子序列](https://leetcode-cn.com/problems/longest-increasing-subsequence/)

#### 动态规划

#### 定义dp[i]为以nums[i]结尾的最长递增子序列的长度

#### 对于求dp[i] 求比nums[i]小的nums[j]中dp[j]的最大值 + 1

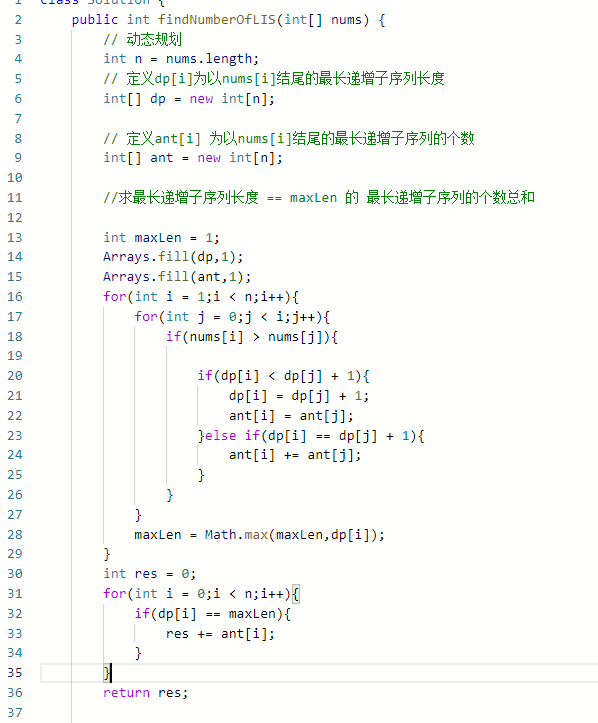


#### [673. 最长递增子序列的个数](https://leetcode-cn.com/problems/number-of-longest-increasing-subsequence/)

在最长递增子序列的基础上加一个dp

定义ant[i] 为以nums[i]结尾的最长递增子序列的个数

根据dp[i] dp[j]+1两个大小比较更新ant[i]



#### [491. 递增子序列](https://leetcode-cn.com/problems/increasing-subsequences/)

要求个数是2两个以上的

回溯法

在每层循环前用集合去重 当前层选过的数字 不能再选了 不然会出现重复

例如 4 7 7 8 9 第二个7不能选了



递增子序列个数

动态规划

注意要去重 不然第二次出现元素的时候 会出现重复

https://blog.csdn.net/u013264046/article/details/78131854?spm=1001.2101.3001.6661.1&utm\_medium=distribute.pc\_relevant\_t0.none-task-blog-2%7Edefault%7ECTRLIST%7ETopBlog-1.topblog&depth\_1-utm\_source=distribute.pc\_relevant\_t0.none-task-blog-2%7Edefault%7ECTRLIST%7ETopBlog-1.topblog&utm\_relevant\_index=1

#### [剑指 Offer 42. 连续子数组的最大和](https://leetcode-cn.com/problems/lian-xu-zi-shu-zu-de-zui-da-he-lcof/)

#### [53. 最大子序和](https://leetcode-cn.com/problems/maximum-subarray/)

动态规划 和连续子数组最大和 一样

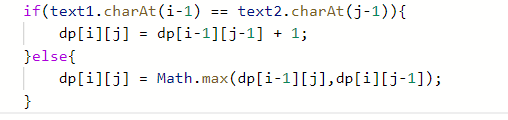
定义dp[i]为子数组...nums[i]连续子数组的最大和 以num[i]结尾

#### [1143. 最长公共子序列](https://leetcode-cn.com/problems/longest-common-subsequence/)

两个字符串的 ****公共子序列**** 是这两个字符串所共同拥有的子序列 和最长公共子串不大一样的

定义dp数组表示A[:i] 和 B[:j] 的最长公共子序列

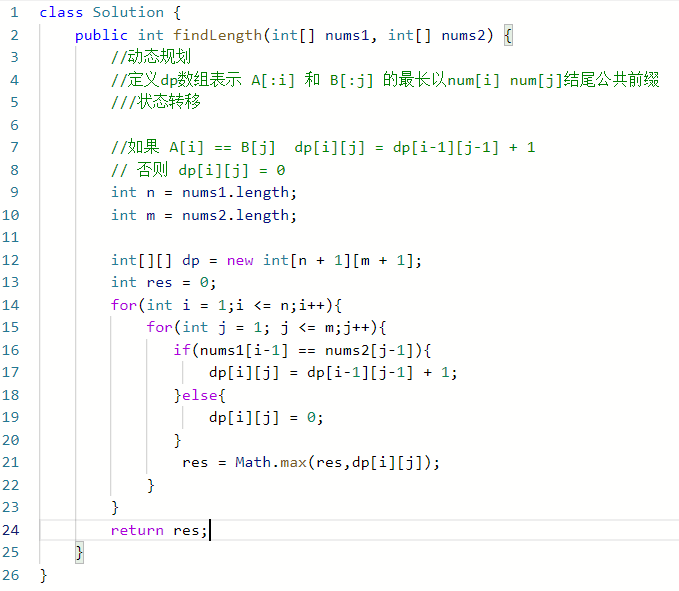
状态转移方程：



#### [718. 最长重复子数组](https://leetcode-cn.com/problems/maximum-length-of-repeated-subarray/) 就是求 最长公共子串问题

动态规划

dp数组表示 A[:i] 和 B[:j] 的最长以num[i] num[j]结尾公共前缀



#### [5. 最长回文子串](https://leetcode-cn.com/problems/longest-palindromic-substring/)

中心扩展法 这个不是动态规划的

有个小技巧

        for(int i = 0;i < n;i++){

            subString(s,i,i);

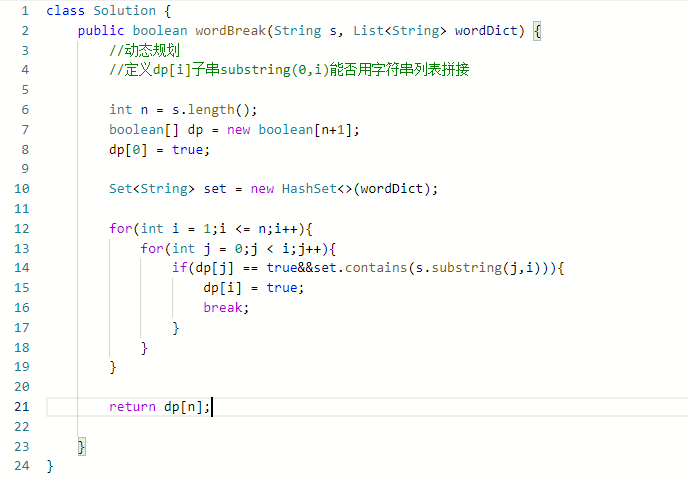
            subString(s,i,i+1);

        }

#### [139. 单词拆分](https://leetcode-cn.com/problems/word-break/)

动态规划 + 字符串哈希（集合）

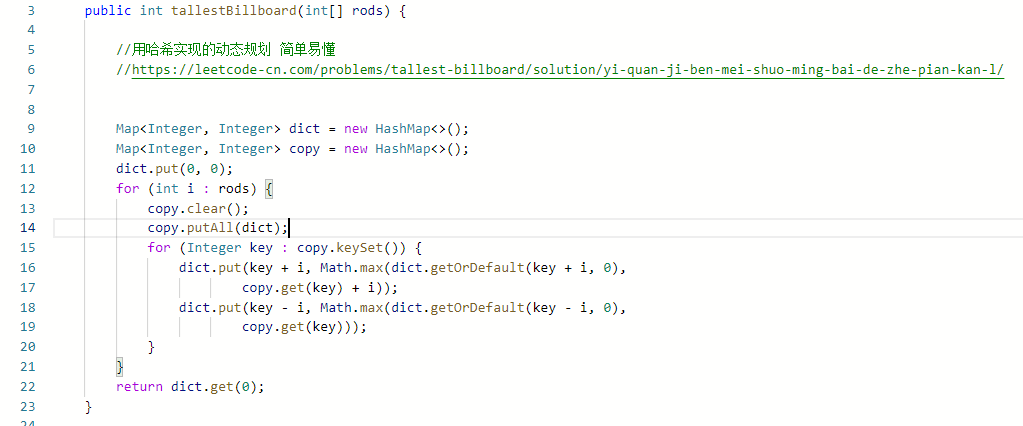
定义dp[i]子串substring(0,i)能否用字符串列表拼接



#### [956. 最高的广告牌](https://leetcode-cn.com/problems/tallest-billboard/)

将问题转化为求数组和为0时的组合

对任何一个数，可以用三种方式对待它，乘以1，-1或0，目标是求和为0时的最大正数和



#### [62. 不同路径](https://leetcode-cn.com/problems/unique-paths/)

dp

简单题 机器人移动路径

#### [64. 最小路径和](https://leetcode-cn.com/problems/minimum-path-sum/)

动态规划

求最小路径可以利用已知的dp数组从右下角往前找路径，看当前结果是从左边还是上面继承而来的

**石子合并问题**

https://blog.csdn.net/zhengjihao/article/details/77799210?spm=1001.2101.3001.6650.1&utm\_medium=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7Edefault-1-77799210-blog-101131928.pc\_relevant\_multi\_platform\_whitelistv2&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7Edefault-1-77799210-blog-101131928.pc\_relevant\_multi\_platform\_whitelistv2&utm\_relevant\_index=2

https://blog.csdn.net/weixin\_43346722/article/details/84889412?spm=1001.2101.3001.6650.1&utm\_medium=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7Edefault-1-84889412-blog-77799210.pc\_relevant\_multi\_platform\_whitelistv1&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromBaidu%7Edefault-1-84889412-blog-77799210.pc\_relevant\_multi\_platform\_whitelistv1&utm\_relevant\_index=2

1：任意版

有N堆石子，现要将石子有序的合并成一堆，规定如下：每次只能移动任意的2堆石子合并，合并花费为将的一堆石子的数量。设计一个算法，将这N堆石子合并成一堆的总花费最小（或最大）。

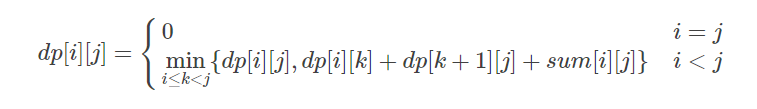
此类问题比较简单，就是哈夫曼编码的变形，用贪心算法即可求得最优解。即每次选两堆最少的，合并成新的一堆，直到只剩一堆为止。证明过程可以参考哈夫曼的证明过程。

2：链式合并

设有N堆沙子排成一排，其编号为1,2,3,…,N(N<=100)。每堆沙子有一定的数量。现要将N堆沙子并成为一堆。归并的过程只能每次将相邻的两堆沙子堆成一堆，这样经过N-1次归并后成为一堆。找出一种合理的归并方法，使总的代价最小最（最大）。

思路：动态规划

假设dp[i][j] 表示从第i个到第j个合并时的最小（最大）代价，则可以得到递推公式



**8多线程编程**

**<https://leetcode-cn.com/problems/fizz-buzz-multithreaded/solution/chang-you-duo-xian-cheng-zhi-jiao-ti-da-eeurc/>**

**10 字典树**

#### [5302. 加密解密字符串](https://leetcode-cn.com/problems/encrypt-and-decrypt-strings/)

这道题用逆向思维做很简单

字典树 + 回溯

字典树定义



**11 原地哈希**

#### [剑指 Offer 03. 数组中重复的数字](https://leetcode-cn.com/problems/shu-zu-zhong-zhong-fu-de-shu-zi-lcof/)

原地哈希法

#### [41. 缺失的第一个正数](https://leetcode-cn.com/problems/first-missing-positive/)

在原数组上做标记

我们对数组进行遍历，对于遍历到的数 x，

如果它在 [1, N]的范围内，那么就将数组中的第 x-1被打上了标记，那么答案是N+1，

否则答案是最小的没有打上标记的位置加1。

原数组交换法 记住这个版本

//将数组的值x放到下标为x-1的位置上



**[442. 数组中重复的数据](https://leetcode-cn.com/problems/find-all-duplicates-in-an-array/)**

原地哈希法

这里数组的范围在1-n

所以索引所在的值减去1，取余 然后将该值索引所在的数组值加数组长度l

当数组中元素大于2\*l时，元素为重复元素



#### [448. 找到所有数组中消失的数字](https://leetcode-cn.com/problems/find-all-numbers-disappeared-in-an-array/)

原地哈希

和442数组中重复的元素差不多

都是将索引所在的值-1 然后当成索引 把对应的值加数组长度

#### [剑指 Offer 03. 数组中重复的数字](https://leetcode-cn.com/problems/shu-zu-zhong-zhong-fu-de-shu-zi-lcof/)

原地哈希 一样的做法 数组中的元素本来就是1-(n-1)的范围

**2022.2.9开始刷的题目**

#### **[136. 只出现一次的数字](https://leetcode-cn.com/problems/single-number/)**

数组中的全部元素的异或运算结果即为数组中只出现一次的数字

对数组所有的元素异或 结果就是[只出现一次的数字](https://leetcode-cn.com/problems/single-number/)

#### **[14. 最长公共前缀](https://leetcode-cn.com/problems/longest-common-prefix/)**

纵向扫描

从前往后遍历所有字符串的每一列，比较相同列上的字符是否相同，如果相同则继续对下一列进行比较，如果不相同则当前列不再属于公共前缀，当前列之前的部分为最长公共前缀。



#### [912. 排序数组](https://leetcode-cn.com/problems/sort-an-array/) 堆排序

两步

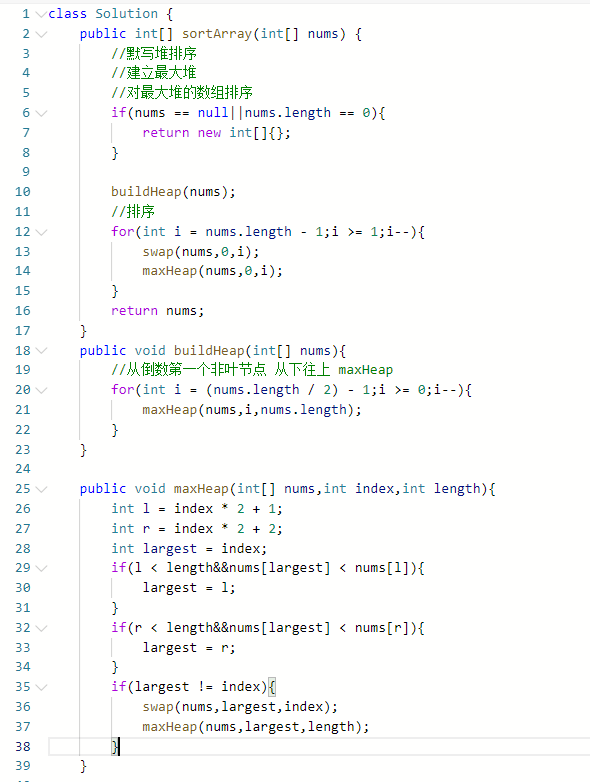
建最大堆

从完全二叉树的倒数第一个非叶子节点（序号为n/2 - 1）开始(循环是从n/2 - 1 到 0 MaxHeap) 自上往下递归MaxHeap（自上往下）构建最大堆

将数组排序

从最后的元素开始，将其和第一个元素（根节点交换） 然后处理根节点MaxHeap重新构建最大堆

从最后处理完第2个元素即可排好序了



#### [152. 乘积最大子数组](https://leetcode-cn.com/problems/maximum-product-subarray/)

动态规划

遍历数组时计算当前最大值，不断更新

令imax为当前最大值，则当前最大值为 imax = max(imax \* nums[i], nums[i])

由于存在负数，那么会导致最大的变最小的，最小的变最大的。因此还需要维护当前最小值imin，imin = min(imin \* nums[i], nums[i])

当负数出现时则imax与imin进行交换再进行下一步计算

#### [剑指 Offer 54. 二叉搜索树的第k大节点](https://leetcode-cn.com/problems/er-cha-sou-suo-shu-de-di-kda-jie-dian-lcof/)

中序遍历倒序

注意这里的k要设置为全局变量 所有递归函数共享的

#### [54. 螺旋矩阵](https://leetcode-cn.com/problems/spiral-matrix/)

模拟四个方向遍历即可

#### [59. 螺旋矩阵 II](https://leetcode-cn.com/problems/spiral-matrix-ii/)

和螺旋矩阵一样做法

四个方向螺旋填入数字

#### **[7. 整数反转](https://leetcode-cn.com/problems/reverse-integer/)**

需要判断是否溢出

每次需要对当前的结果判断

如果

res \* 10 < Integer.MIN\_VALUE 或者

res \* 10 > Integer.MAX\_VALUE

注意 res \* 10 可能溢出 要换成除法

res < Integer.MIN\_VALUE / 10

#### **[9. 回文数](https://leetcode-cn.com/problems/palindrome-number/)**

将数字右半边数字反转

然后和剩余的左半边数字对比

注意 数字长度为奇数的时候



#### **[8. 字符串转换整数 (atoi)](https://leetcode-cn.com/problems/string-to-integer-atoi/)**

判断溢出就是先保存结果temp

然后将  res= res\* 10 + number;

res / 10 是否等于 temp

#### **[148. 排序链表](https://leetcode-cn.com/problems/sort-list/)**

归并排序

快慢指针找到中间节点

排序左边 右边

合并左右边排序好的链表 即合并有序链表

返回归并结果

#### **[21. 合并两个有序链表](https://leetcode-cn.com/problems/merge-two-sorted-lists/)**

正常思路 定义临时节点

#### **[23. 合并K个升序链表](https://leetcode-cn.com/problems/merge-k-sorted-lists/)**

归并排序 还需要用到合并两个排序链表

**优先队列**

维护一个最小堆 重写compare函数

对象排序Comparator比较器写法



返回正数就要逆转 顺序是 o2 o1

**集合，数组排序的构造写法**

Comparable 接口 重写compareTo(Object obj) 对象实现这个接口，重写方法

在对象里自定义排序 只能用Comparable 重写compareTo(Object obj)方法

**记得以后用这个**

**Comparator接口** compare(Object obj1, Object obj2) 是外比较器 不能写进类里面的 必须重新写一个类比较器 将MyComparator传入 Arrays.sort() 或者 Collections.sort()







#### **[146. LRU 缓存](https://leetcode-cn.com/problems/lru-cache/)** 复习

哈希双向链表法

缓存是在双向链表里面的 每个节点存有key和value (key是为了删除哈希的值设置的)

双向链表定义虚拟的头尾节点

哈希map定义HashMap<Integer,Node>

//定义数据结构

int cap;

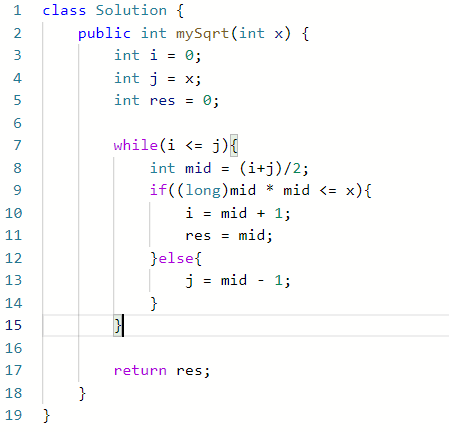
HashMap<Integer,Node> map = new HashMap<Integer,Node>();

DoubleList doubleList = new DoubleList();

#### [69. Sqrt(x)](https://leetcode-cn.com/problems/sqrtx/)

x的平方根

二分法 必须用小于等于 整型



当需要精确到多少位数时 也还可以用二分法 代码：

double i = 0;

double x = 2;

double j = x;

double res = 0;

while(i < j){

double mid = (i + j)/2;

if(mid \* mid <= x&&(mid + 0.0000001) \* (mid + 0.0000001) > x){

res = mid;

break;

}

//记住必须用小于等于

if(mid \* mid <= x){

res = mid;

i = mid + 0.00000001;

}else{

j = mid - 0.00000001;

}

}

System.out.println(res);

#### [739. 每日温度](https://leetcode-cn.com/problems/daily-temperatures/)

#### **[210. 课程表 II](https://leetcode-cn.com/problems/course-schedule-ii/)**

拓扑排序 入度数组+哈希集合数组邻接表

#### 否则比较当前两个树的左右根值 继续递归比较

拓扑排序的一个附加效果是：****能够顺带检测有向图中是否存在环****

****思路： 广度优先遍历****

首先定义哈希集合数组，储存每个节点指向的节点集

HashSet<Integer>[] adj = new HashSet[numCourses];

定义入度数组，储存每个节点的入度

定义一个队列 存入度为0的节点（这里的节点就是节点的编号）

每次从队列中取出入度为0 的节点 从哈希数组中拿出指向节点集

将指向节点集的节点入度都减1 然后将入度为0的节点入栈

#### [剑指 Offer II 076. 数组中的第 k 大的数字](https://leetcode-cn.com/problems/xx4gT2/)

利用快速排序 轴值的特点

#### [912. 排序数组](https://leetcode-cn.com/problems/sort-an-array/)

快速排序

微软考了递归和非递归的版本 非递归用栈的方法

堆排序

两步

建最大堆

从完全二叉树的倒数第一个非叶子节点（序号为n/2 - 1）开始 自上往下递归MaxHeap构建最大堆

将数组排序

从最后的元素开始，将其和第一个元素（根节点交换） 然后处理根节点MaxHeap重新构建最大堆

从最后处理完第2个元素即可排好序了

归并排序

记住创建一个临时节点 保存merge的数组 然后赋值回原数组

**数组中的第K个最大元素**

**注意快速排序里的**

if(l > r){

Return

}

不是>=

不然输入为[1]时 直接返回了

快速排序都可以

#### [剑指 Offer 40. 最小的k个数](https://leetcode-cn.com/problems/zui-xiao-de-kge-shu-lcof/)

利用快排

记住这里没有

if(l >= r){

Return

}

#### 剑指 Offer 60. n个骰子的点数 （着重看!!） 需要多看

要做的 先列出来

最长上升子序列的 二分+动态规划版的

[4. 寻找两个正序数组的中位数](https://leetcode-cn.com/problems/median-of-two-sorted-arrays)

还有LC295

三数之和

k个一组反转

**2022.2.28**

复习二叉树专题：

复习链表专题：

复习排列子集组合专题：

复习动态规划专题

每日一题：

#### [762. 二进制表示中质数个计算置位](https://leetcode-cn.com/problems/prime-number-of-set-bits-in-binary-representation/)

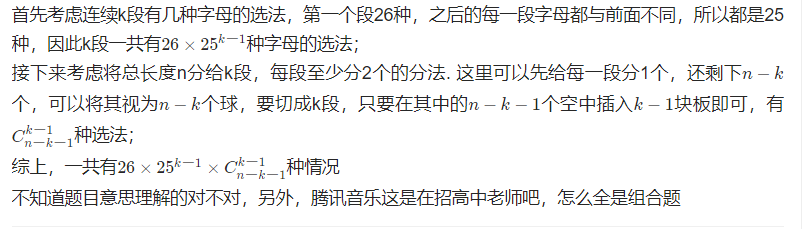
Integer.bitCount()

判断质数

腾讯qq音乐笔试：

第一道k-size字符串

每次运算都要取模

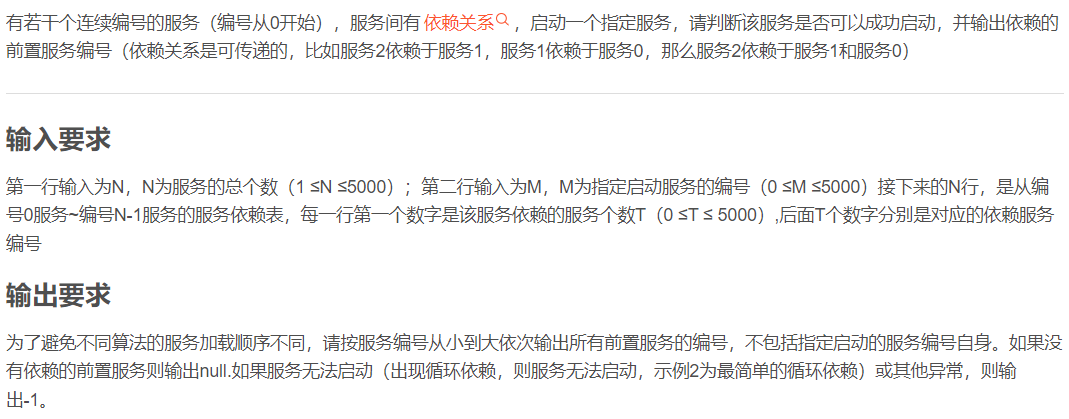


华为笔试：

4.6

统计词频 排序求topk

服务启动



将要启动的服务入队列 然后判断前置服务之间是否存在依赖 就是发现前置条件循环了 之前已经需要

单调栈

华为4.13.2022笔试

硬件最佳分配策略

优先队列 排序 两个策略排序

工单选择方案

<https://www.codeleading.com/article/42425810425/>

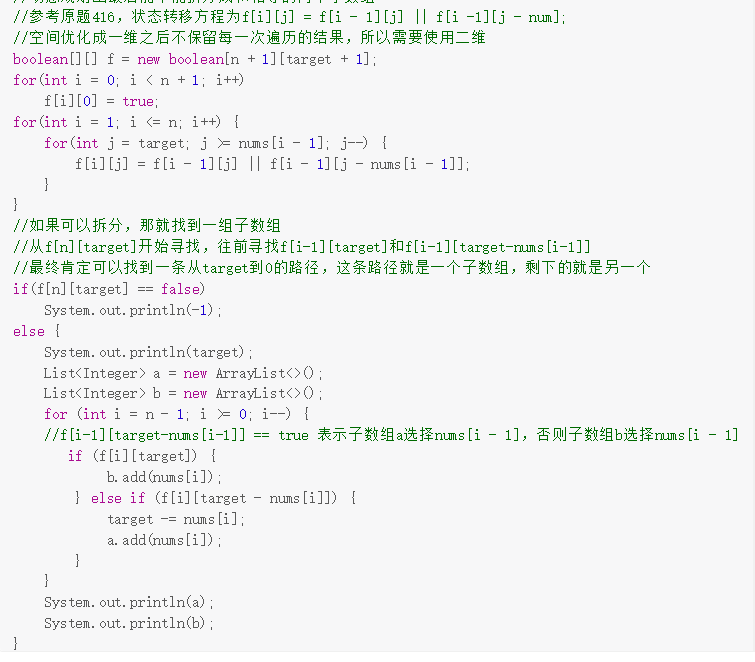
假设这些工单应该是同时产生的，SLA的计时从某个时间点开始是一直要计时的。每个工单修复花费的都是一个小时，这样的话都是考虑整数就可以了。优先处理SLA小的，SLA相同的话，由于在该一个小时内只能处理一个工单，所以优先选择积分大的工单进行处理，舍弃其余的工单。

分配糖果

#### [剑指 Offer II 101. 分割等和子集](https://leetcode-cn.com/problems/NUPfPr/)

<https://leetcode-cn.com/circle/discuss/eNm2zD/>

掌握这个求路径的方法



#### 腾讯光子面试：

用定长数组实现队列

<https://www.cnblogs.com/bowenqianngzhibushiwo/p/11620777.html>

阿里二面：

三个线程，分别打印x, y, z。

顺序打印xyz十次

字节二面：

K 个一组翻转链表

最小覆盖子串

网易互娱春招笔试EBWK

数独