贪心算法

核心思想：局部最优→全局最优

如果这种思想找不到反例，题目就可以用贪心算法

分发饼干

思路：把大饼干分给大胃口的孩子**（局部最优）**，从而把饼干喂给更多的孩子**（全局最优）**

*如果大饼干分给较小胃口的孩子，那么小饼干就很可能分不出去

实现：先对g和s排序（Arrays.sort），然后对这两个数组从后往前遍历，**胃口是固定遍历**（for循环），**饼干是条件遍历**（当前饼干大于等于胃口则将饼干往前一位）

*不能固定遍历饼干，条件遍历胃口，这样有可能一个符合条件的分发都没有

题目：

455. 分发饼干（简单）

摆动序列

一定要把坡度图画出来分析！

局部最优：删除单调坡度上的节点（不包括底和顶），这样这个坡上就有两个局部峰值

全局最优：序列有最多数量的局部峰值

大体思路：

考虑遍历节点的prediff（nums[i]-nums[i-1]）和curdiff（nums[i+1]-nums[i]），如果prediff>0且curdiff<0，或prediff<0且curdiff>0，就可以纳入最长序列的统计

遍历的方式是从数组头元素开始，一直到倒数第二个元素，每次计算curdiff，当前prediff为循环上一次的curdiff

然而有以下**三种特殊情况**：

（1）上下坡中有平坡：统一删除左边的所有平坡元素，仅保留最右边的元素，此时判断条件对应修改成prediff≥0且curdiff<0，或prediff≤0且curdiff>0

（2）首尾两端：

a. **首元素：**可以视作在它之前虚拟给一个同等值的平坡，此时符合（1）中的判断，将首元素纳入长度的统计，具体体现在**初始化prediff=0**

b. **尾元素：**由于遍历的尽头是数组倒数第二个元素，而尾元素根据题目规则又需要算进去，因此默认尾元素是峰值，体现在**初始化序列长度count=1**

（3）单调上坡/下坡中有平坡：如果每次循环都进行prediff和curdiff的传递，那么在这种情况下就会出问题（平坡最右边的元素会被算入结果）。为规避这种情况，仅在判断条件成立时才进行prediff=curdiff的传递

*数组元素长度为0或1时，需要返回数组长度

题目：

376. 摆动序列（中等）

最大子数组和

局部最优：当遍历到子数组的和为负数时，立刻抛弃整个子数组，以下一个元素为起点继续遍历

（如果子数组和为负数，对下一元素而言，整体的和一定会被拉低）

*换言之，保证数组里的每个元素，它前面子数组的和一定为正

全局最优：根据局部最优的方式，最终可以得到最大子数组和

实现：将结果res初始化为无穷小，子数组和count初始为0，每次遍历将count与数组元素相加，如果count比res大则将res变为count，如果count≤0则将count置为0

细节：

（1）将res初始化为无穷小，可以保证如果数组元素全为负（如[-2, -2, -2, ...]）时结果正确

（2）count比res大才更新res，可以保证如果数组出现[4,-1]时结果正确，如果-1后面还有正数则正常更新res。总而言之，这样可以保证**遍历全过程中对最大子数组和的求解都是正确的**

（3）注意区分概念：是不能让“连续和”为负数的时候加上下一个元素，而不是不让“连续和”加上一个负数

题目：

53. 最大子数组和（中等）

买卖股票的最佳时机II

问题拆解：假如说第0天购买，第3天卖出，利润为nums[3] - nums[0]，但实际上可以拆分成(nums[3] - nums[2])+(nums[2] - nums[1])+(nums[1] - nums[0])，这样便于我们确认局部最优和全局最优

局部最优：选取nums前后元素之差为正的

全局最优：选取所有nums前后元素之差为正的进行累加

*注意："问题拆解"并不反映真实情况，并不是说买3次卖3次，而是第0天买第3天卖

题目：

122. 买卖股票的最佳时机II（中等）

跳跃游戏

跳跃游戏的两个题都可以用贪心算法解决

跳跃游戏I：局部贪心，每次跳到最远距离，即跳到当前元素值的步数。这个距离用rightmost记载，它是可以到达的最右边位置。具体表现为：**当遍历位置小于等于rightmost时**，不断更新rightmost使其越来越右，若最后rightmost能到达数组最右边甚至更远，则能到终点，反之则不能。

（遍历位置小于等于rightmost这个条件很重要，遍历位置大于rightmost的统计和值的更新是无效的，因为根本就不能到达这个位置）

跳跃游戏II：局部贪心，获取当前最远覆盖位置cur和下一元素最远覆盖位置next，初始化它们为0。遍历数组元素（nums.length-2，不包括尾元素，否则结果会多1）：遍历位置小于等于next时，不断更新next到更远覆盖位置；当遍历位置等于cur时，就把结果加1，并且把cur变为next的位置，最后返回结果，其实返回的就是所有覆盖长度段的间隔数。

***这两个题都要对数组长度为1的情况进行单独说明**

题目：

55. 跳跃游戏（中等）

45. 跳跃游戏II（中等）

K次取反后最大化的数组和

局部最优：把绝对值大的负数取反变为正

全局最优：把尽可能多的负数变为正，实现和最大

步骤：

1、把数组排序

2、遍历数组进行**第一次变化，仅对负数操作**（遇到正数直接break循环），每次对负数元素取反后k要减1，第一次变化后有以下几种情况——

（1）不够把所有负数都变为正，所以需要设置遍历条件为i小于数组长度且k大于0

（2）所有负数变为正后k仍大于0（包括遍历已到尽头或绝对值最小的负数后面还有正数两种情况）

3、把更新的数组再排序（不影响步骤2的情况（1），因为此时k=0，对应下面情况b），排序后**进行第二次变化，有两种情况：**

a. k为奇数，此时需要将排序后最小的数取反

b. k为偶数，此时不做变化（相当于对同一个数操作）

4、把进行第二次变化后的数组进行for循环加和，得到最终结果

题目：

1005. K次取反后最大化的数组和（简单）

加油站

局部最优：考虑**gas[i]和cost[i]的差值**，从i=0开始遍历，如果**从起点位置start开始的各差值之和cursum**小于0，说明从cursum开始算的起点到i这个区间任何一个位置出发，都不可能走完一圈，此时把车的**起点start设为i+1且cursum置0**

全局最优：数组内**所有差值totalsum**加起来小于0，则不可能走完一圈，返回-1，否则返回start

（start，cursum，totalsum初始化为0）

题目：

134. 加油站（中等）

分发糖果

关键：右边比左边分数高，和左边比右边分数高要分开考虑

局部最优：相邻元素谁更大，则它对应的res数组值就多1

全局最优：按照全局最优的思路，获得最少分发的糖果数量

具体实现：

a. **考虑右边比左边分数高：**需要从第二个元素（ratings[1]）开始**从前往后遍历r**atings数组，先把结果数组res头元素全部置1，遇到ratings数组元素比前一个元素大时，则将res对应位置的值加1，否则不变

b. **考虑左边比右边分数高：**此时需要**从后往前遍历**（因为要依赖后面元素的值），从倒数第二个元素开始，遇到rating数组元素比它后一个元素大时，由于需要同时满足a、b两种情况，因此要取res后一个元素的值+1与自身res的最大值

c. 遍历res数组把结果累加

题目：

135. 分发糖果（困难）

柠檬水找零

局部最优：收到5块，不找零；收到10块，返回5块；收到20块，**优先返回10块+5块**（把更多的5块留给找零顾客的10块），再考虑返回3个5块

全局最优：按照局部最优的思路获得题目答案

具体实现：需要维护5块钱的张数sum5和10块钱的张数sum10（20块钱的张数没必要统计，派不上找零的用场），有以下几种情况：

（1）**遇到5块，**则把sum5加1；

（2）**遇到10块，**则查看sum5的张数：如果＜1，则无法找零，返回false；否则把sum5减1，sum10加1；

（3）**遇到20块，**先看sum10的张数：

a. **如果sum10张数≥1：**在这情况下**sum5的张数如果＜1**，则返回false，否则sum5和sum10均减1；

b. **如果sum10张数＜1：**需要3个5块去找零，**如果sum5＜3**，则返回false，否则sum5减3；

（4）遍历循环没有返回false，则说明能按题目条件找零，循环结束后返回true

题目：

860. 柠檬水找零（简单）

根据身高重建队列

关键：本题涉及身高h和人数k**两个维度**，需要**先确定一个维度，再确定另一个维度**

a. 如果按人数k排序，会发现k的排列并不符合条件，h的排列也不符条件且无规律，无法确定维度

b. 如果**按身高h排序**，如果**从高到低**排序，且**相同身高下按k的升序排序**，那么在重建队列时，把某个身高-人数（h-k）对**插入到索引k的位置**，能保证他前面的人比他高且数量正确

（也就是说，需要让矮个子的人掌握选择位置的主动权）

局部最优：优先按身高更高的people的k来插入

全局最优：做完所有插入操作后满足题目条件

实现细节：由于涉及插入操作，用**链表**的数据结构更加方便，最后再把链表转为数组

*注意排序和重建队列的写法，详情看leetcode代码

题目：

406. 根据身高重建队列（中等）

用最少数量的箭引爆气球

关键：重叠区间问题（此题以及往下3个题都是区间问题）

局部最优：往**重叠区间**射，可以保证数量最少，且找不出反例

全局最优：按局部最优的方式，能得到正确答案

具体实现：

（1）将气球区间**升序排序**（按左边界和右边界排都可以，本题按**左边界**）

（2）初始化结果res为1（至少需要一支箭）

（3）从第二个区间（i=1）开始遍历，如果**它的左边界大于上一个区间的右边界**，此时需要多加一支箭，res加1

***注意一定是"大于"不是"大于等于"，因为两个气球挨着没有交集（如[1,2]，[2,3]），一支箭也能搞定**

（4）否则就把当前区间的右边界更新为**自己右边界和上一区间右边界的最小值**（可以反证法讨论更新在该位置的左边和右边，会出现什么问题）

细节：为防止溢出，在左边界升序排序时用**Integer内置比较方法**，详情看Leetcode代码

题目：

452. 用最少数量的箭引爆气球（中等）

无重叠区间

局部最优：保证每个区间**往其它区间渗入尽可能小**，这样更不容易重叠

全局最优：按局部最优的方式，保证删除的区间数量最少

***数值区间问题：通常需要对区间排序（常按左边界排）**

具体实现：

（1）将区间按左边界升序排序（注意采用Integer内置比较方法）

（2）初始化结果res为0

（3）从第二个区间（i=1）开始遍历，如果它的左边界≤上一区间的右边界，**就把该区间的新右边界设置为它原来的右边界和上一区间右边界的最小值**，此时代表要删除一个区间，res加1

（用这个程序，同时实现了左边界＞上一区间右边界时不进行删除操作）

（4）最后返回结果

题目：

435. 无重叠区间（中等）

合并区间

（此题和贪心算法关系不大，但和“无重叠区间”的方法类似，可以对比总结）

采取策略：对合并结果的数组最后一个区间不断进行操作

具体实现：

（1）将区间按左边界升序排序（注意采用Integer内置比较方法）

（2）初始化将第一个区间加入结果链表res中

（3）从第二个区间（i=1）开始遍历，如果左边界≤上一区间的右边界，**就把res末尾区间的新右边界改为它的右边界和遍历区间右边界的最大值**，然后移除末尾区间，把合并后的新区间加入res（注意写法，参考leetcode代码）

（4）左边界＞上一区间的右边界，说明不需要合并，**直接把当前区间加入res中**

（5）最后把res转为数组形式返回结果

题目：

56. 合并区间（中等）

划分字母区间

思路：两次遍历——

第一次：**统计每个字母最后一次出现的位置**，利用数组型哈希表实现；

第二次：**统计每个区间的最右边界**（取自己和当前遍历字母最后一次出现位置的最大值），**直到遍历位置和这个最右边界的值相等**，即可记录答案

（注意：记录答案需要记下区间的始末位置，才能得到长度，且需要不断更新区间初始位置）

***通过以上思路，可以保证划分的区间尽可能多且符合题意**

题目：

763. 划分字母区间（中等）

单调递增的数字

局部最优：当遇到当前数字高一位的数字比自己大时，要把高一位的数字减1，从最后一次减1的地方开始把它后面的数全部变为9

全局最优：按局部最优的方式，可以得到最大的单调递增数字

具体实现：

（1）把整数转成字符串形式，字符串每一位代表一个数字

（2）设定把后面数字全部变为9的起始位置start，初始化为字符串的长度

（3）从后往前遍历字符串，如果遇到当前数字高一位的数字比自己大，则将高一位的数字减1，且start变为当前数字的位置

（4）以start为起点向后遍历字符串，把后面所有位全部置为9

（5）最后把字符串转为整数

***遍历顺序一定是从后往前**，如果从前往后遍历会出问题（学会举例，如332）

*注意把整数转为字符串，把字符串转回整数以及对字符串的数字减1的写法（见leetcode代码）

题目：

738. 单调递增的数字（中等）

监控二叉树

局部最优：尽可能在叶节点的父节点上加监控，不要在叶节点上加监控，要求我们从下往上看

全局最优：按局部最优的方式，可以得到最少数量的摄像头

难点：

（1）二叉树的遍历——优先考虑叶节点的情况，因而采用**后序遍历**（左右中）

（2）安置摄像头的具体操作——

a. 分析每个节点存在的**所有状态**（无摄像头覆盖、自身有摄像头、有摄像头覆盖）

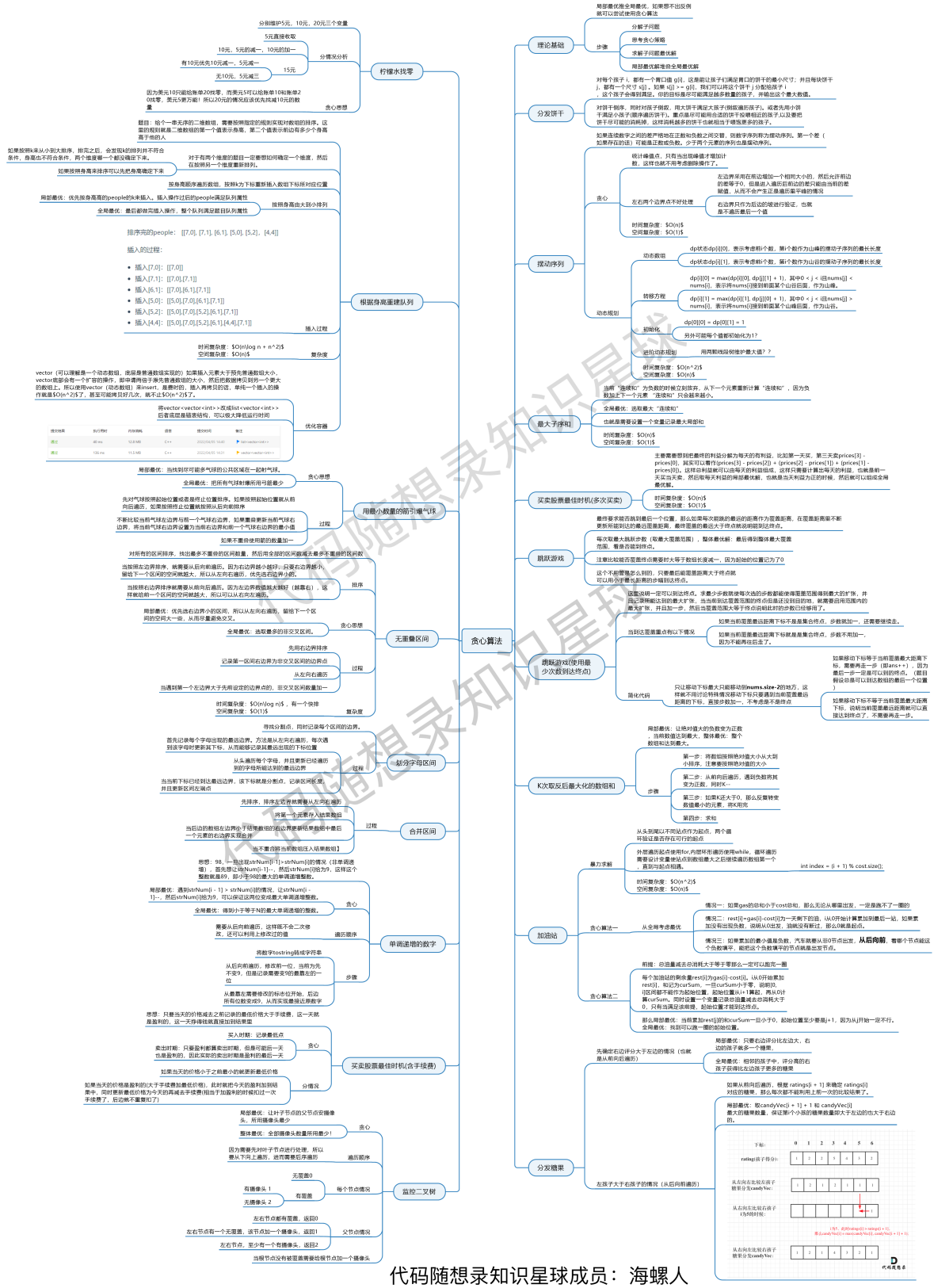
b. 善用**数字标记**（如上面三种状态分别代表0、1、2）

具体实现：参看leetcode代码，写得很详细，注意分情况讨论，以及递归函数注意对应递归三部曲

题目：

968. 监控二叉树（困难）

思路总结


贪心算法
理论基础
局部最优推全局最优，如果想不出反例就可以尝试使用贪心算法
步骤
分解子问题
思考贪心策略
求解子问题最优解
局部最优解堆叠全局最优解
分发饼干
对每个孩子 i，都有一个胃口值 g[i]，这是能让孩子们满足胃口的饼干的最小尺寸；并且每块饼干 j，都有一个尺寸 s[j]。如果 s[j] >= g[i]，我们可以将这个饼干 j 分配给孩子 i ，这个孩子会得到满足。你的目标是尽可能满足越多数量的孩子，并输出这个最大数值。
对饼干倒序，同时对孩子倒叙，用大饼干满足大孩子(倒叙遍历孩子)。或者先用小饼干满足小孩子(顺序遍历饼干)。重点是尽可能用合适的饼干投喂相应的孩子，以及要把饼干尽可能的消耗掉，这样消耗越多的饼干也就相当于喂饱更多的孩子。
摆动序列
如果连续数字之间的差严格地在正数和负数之间交替，则数字序列称为摆动序列。第一个差（如果存在的话）可能是正数或负数。少于两个元素的序列也是摆动序列。
贪心
统计峰值点，只有当出现峰值才增加计数，这样也就不用考虑删除操作了。
左右两个边界点不好处理
左边界采用在前边增加一个相同大小的，然后允许前边的差等于0，但是进入遍历后前边的差只能由当前的差赋值，从而不会产生正是遍历最平缓的情况
右边界只作为后边的被动进行验证，也就是不遍历最后一个值
时间复杂度：$O(n)$
空间复杂度：$O(1)$
动态规则
动态数组
dp状态dp[i][0]，表示考虑前i个数，第i个数作为山峰的摆动子序列的最长长度
dp状态dp[i][1]，表示考虑前i个数，第i个数作为山谷的摆动子序列的最长长度
转移方程
dp[i][0] = max(dp[i][0], dp[j][1] + 1)，其中0 < j < i且nums[j] < nums[i]，表示将nums[i]接到前面某个山谷后面，作为山峰。
dp[i][1] = max(dp[i][1], dp[j][0] + 1)，其中0 < j < i且nums[j] > nums[i]，表示将nums[i]接到前面某个山峰后面，作为山谷。
初始化
dp[0][0] = dp[0][1] = 1
另外可能每个值都初始化为1？
进阶动态规划
用两颗线段树维护最大值？？
时间复杂度：$O(n^2)$
空间复杂度：$O(n)$
最大子序和
当前“连续和”为负数的时候立刻放弃，从下一个元素重新计算“连续和”，因为负数加上下一个元素 “连续和”只会越来越小。
全局最优：选取最大“连续和”
也就是需要设置一个变量记录最大局部和
时间复杂度：$O(n)$
空间复杂度：$O(1)$
买卖股票最佳时机(多次买卖)
主要需要想到把最终的利益分解为每天的收利益，比如第一天买，第三天卖prices[3] - prices[0]，其实可以看作[prices[3] - prices[2]] + [prices[2] - prices[1]] + [prices[1] - prices[0]]。这样总利益就可以由每天的利益组成，这样只需要计算出每天的利益，也就是前一天买当天卖。然后取每天利益的局部最优解，也就是当天利益为正的时候，然后就可以组成全局最优解。
时间复杂度：$O(n)$
空间复杂度：$O(1)$
跳跃游戏
最终要求能否跳到最后一个位置，那么如果每次能跳的最远的距离作为覆盖距离，在覆盖距离里不断更新所能到达的最远覆盖距离，最终覆盖的最远大于终点就说明能到达终点。
每次取最大跳跃步数（取最大覆盖范围），整体最优解：最后得到整体最大覆盖范围，看是否能到终点。
注意比较能否覆盖终点需要时大等于数组长度减一，因为起始的位置记为了0
这个不用需要总么到的，只要最后能覆盖距离大于终点就可以用小于最长距离的步骤到达终点。
跳跃游戏(使用最少次数到达终点)
这里说明一定可以到达终点。求最少步数就使每次选的步数都能使得覆盖范围得到最大的扩张，并且记录所能达到的最大扩张，当当前到达覆盖范围的终点但是还没到目的地，就需要启用范围内的最大扩张，并且加一步，然后当覆盖范围大于终点但却赶时的步数已经够用了。
当到达覆盖最点有以下情况
如果当前覆盖最远距离下标不是是集合终点，步数就加一，还需要继续走。
如果当前覆盖最远距离下标就是是集合终点，步数不用加一，因为不能再往后走了。
简化代码
只让移动下标最大只能移动到nums.size-2的地方，这样就不用讨论特殊情况移动下标只要遇到当前覆盖最远距离的下标，直接步数加一，不考虑是不是终点
如果移动下标等于当前覆盖最大距离下标，需要再走一步（即ans++），因为最后一步一定是可以到的终点。（题目假设总是可以到达数组的最后一个位置）
如果移动下标不等于当前覆盖最大距离下标，说明当前覆盖最远距离是可以直接达到终点了，不需要再走一步。
K次取反后最大化的数组和
局部最优：让绝对值大的负数变为正数
当前数值达到最大，整体最优：整个数组和达到最大。
步骤
第一步：将数组按照绝对值大小从大到小排序，注意要按照绝对值的大小
第二步：从前向后遍历，遇到负数将其变为正数，同时K--
第三步：如果K还大于0，那么反复转变数值最小的元素，将K用完
第四步：求和
加油站
暴力求解
从头到尾以不同起点作为起点，两个循环验证是否存在可行的起点
外部遍历起点使用for,内层环形遍历使用while，循环遍历需要设计变量使起点到数组最大之后继续遍历数组第一个，直到与起点相遇。
int index = (i + 1) % cost.size();
时间复杂度：$O(n^2)$
空间复杂度：$O(n)$
贪心算法一
从全局考虑最优
情况一：如果gas的总和小于cost总和，那么无论从哪里出发，一定是跑不了一圈的
情况二：rest[i]=gas[i]-cost[i]为一天剩下的油，从0开始计算累加到最后一站，如果累加没有出现负数，说明从0出发，油就没有断过，那么0就是起点。
情况三：如果累加的最小值是负数，汽车就要从非0节点出发，从后向前，看哪个节点能这个负数填平，能把这个负数填平的节点就是出发节点。
贪心算法二
前提：总油量减去总消耗大于等于零那么一定可以跑完一圈
每个加油站的剩余量rest[i]为gas[i]-cost[i]。i从0开始累加rest[i]，和记为curSum，一旦curSum小于零，说明[0, i]区间都不能作为起始位置，起始位置从i+1算起，再从0计算curSum。同时设置一个变量记录总油量减去总消耗大于0，只有当满足这条规则，起始位置才能到达终点。
那么局部最优：当前累加rest[i]的和curSum一旦小于0，起始位置至少要是i+1，因为从i开始一定不行。
全局最优：找到可以跑一圈的起始位置。
分发糖果
先确定右边评分大于左边的情况（也就是从前向后遍历）
局部最优：只要右边评分比左边大，右边的孩子就多一个糖果。
全局最优：相邻的孩子中，评分高的右孩子获得比左边孩子更多的糖果
左孩子大于右孩子的情况（从后向前遍历）
如果从前向后遍历，根据 ratings[i + 1] 来确定 ratings[i] 对应的糖果，那么每次都不能利用上前一次的比较结果了。
局部最优：取candyVec[i + 1] + 1 和 candyVec[i]最大的糖果数量，保证第i个小孩的糖果数量即大于左边的也大于右边的。
下标:
ratings(孩子得分):
从左向右比较左孩子糖果分发(candyVec):
从右向左比较右孩子i为结的时候:
从右向左比较右孩子糖果分发(candyVec):
根据身高重建队列
柠檬水找零
分别维护5元，10元，20元三个变量
5元直接收取
分情况分析
10元，5元的减一，10元的加一
15元
有10元优先10元减一，5元减一
无10元，5元减三
因为美元10只能给账单20找零，而美元5可以给账单10和账单20找零，美元5更万能！所以20元的情况应该优先找减10元的数量
贪心思想
题目：给个一串无序的二维数组，需要按照指定的规则实现对数组的排序。这里的规则就是二维数组的第一个值表示身高，第二个值表示前边有多少个身高高于他的人
如果按照k来从小到大排序，排完之后，会发现k的排列并不符合条件，身高也不符合条件，两个维度哪一个都没确定下来。
对于有两个维度的题目一定要想如何确定一个维度，然后在按照另一个维度重新排列。
如果按照身高来排序可以先把身高确定下来
按身高顺序遍历数组，按照k为下标重新插入数组下标所对应位置
局部最优：优先按身高高的people的k来插入。插入操作过后的people满足队列属性
按照身高由大到小排列
全局最优：最后都做完插入操作，整个队列满足题目队列属性
排序完的people：[[7,0], [7,1], [6,1], [5,0], [5,2], [4,4]]
插入的过程：
插入[7,0]：[[7,0]]
插入[7,1]：[[7,0],[7,1]]
插入[6,1]：[[7,0],[6,1],[7,1]]
插入[5,0]：[[5,0],[7,0],[6,1],[7,1]]
插入[5,2]：[[5,0],[7,0],[5,2],[6,1],[7,1]]
插入[4,4]：[[5,0],[7,0],[5,2],[6,1],[4,4],[7,1]]
插入过程
时间复杂度：$O(n\log n + n^2)$
空间复杂度：$O(n)$
复杂度
vector（可以理解是一个动态数组，底层是普通数组实现的）如果插入元素大于预先普通数组大小，vector底层会有一个扩容的操作，即申请两倍于原先普通数组的大小，然后把数据拷贝到另一个更大的数组上。所以使用vector（动态数组）来insert，是费时的。插入两终贝的话，单纯一个插入的操作就最$O(n^2)$了，甚至可能拷贝好几次，就不止$O(n^2)$了。
将vector<vector<int>>改成list<vector<int>>后者底层是链表结构，可以极大降低运行时间
优化容器
用最小数量的箭引爆气球
贪心思想
局部最优：当找到尽可能多气球的公共区域在一起射气球。
全局最优：把所有气球射爆所用弓箭最少
过程
先对气球按照起始位置或者是终止位置排序。如果按照起始位置就从前向后遍历，如果按照终止位置就按照从后向前排序
不断比较当前气球左边界与前一个气球右边界，如果重叠更新当前气球右边界，将当前气球右边界设置为当前右边界和前一个气球右边界的最小值
如果不重叠使用箭的数量加一
无重叠区间
对所有的区间排序，找出最多不重叠的区间数量，然后用全部的区间数减去最多不重叠的区间数
排序
当按照左边界排序，就需要从后向前遍历。因为右边界越小越好，只要右边界越小，留给下一个区间的空间就越大，所以从左向右遍历，优先选右边界小的。
当按照右边界排序就需要从前向后遍历，因为左边界越靠越大越好（越靠右），这样就给前一个区间的空间就越大，所以可以从右向左遍历。
贪心思想
局部最优：优先选右边界小的区间，所以从左向右遍历，留给下一个区间的空间大一些，从而尽量避免交叉。
全局最优：选取最多的非交叉区间。
过程
先用右边界排序
记录第一区间右边界为非交叉区间的边界点
从左向右遍历
当遇到第一个左边界大于先前设定的边界点的，非交叉区间数量加一
复杂度
时间复杂度：$O(n\log n)$，有一个快排
空间复杂度：$O(1)$
划分字母区间
寻找分割点，同时记录每个区间的边界。
过程
首先记录每个字母出现的最远边界。方法是从左向右遍历，每次遇到该字母时更新其下标，从而能够记录其最远出现的下标位置
从头遍历每个字母，并且更新已经遍历到的字母所能达到的最远边界
当当前下标已经到达最远边界，该下标就是分割点，记录区间长度，并且更新区间左端点
合并区间
过程
先排序，排序左边界就需要从左向右遍历
将第一个元素存入结果数组
当后边的数组左边界小于结果数组的右边界更新结果数组中最后一个元素的右边界实现合并
当不重合将当前数组压入结果数组】
单调递增的数字
思想：98，一旦出现strNum[i-1] > strNum[i]的情况（非单调递增），首先想让给Num[i-1]--，然后strNum[i]给为9，这样这个整数就是89，即小于98的最大的单调递增整数。
贪心
局部最优：遇到strNum[i - 1] > strNum[i]的情况，让strNum[i - 1]--，然后strNum[i]给为9，可以保证这两位变成最大单调递增整数。
全局最优：得到小于等于N的最大单调递增的整数。
遍历顺序
需要从后向前遍历，这样既不会二次修改，还可以利用上修改过的值
步骤
将数字tostring转成字符串
从后向前遍历，修改前一位，当前为先不变9，但是记录需要变9的最靠左的一位
从最靠左需要修改的标志位开始，后边所有位数变成9，从而实现最接近原数字
监控二叉树
贪心
局部最优：让补子节点的父节点安摄像头，所用摄像头最少
整体最优：全部摄像头数量所用最少！
遍历顺序
因为需要先对叶子节点进行处理，所以要从下向上遍历，进而需要后序遍历
每个节点情况
无覆盖0
有覆盖
有摄像头 1
无摄像头 2
父节点情况
左右节点都有覆盖，返回0
左右节点有一个无覆盖，该节点加一个摄像头，返回1
左右节点，至少有一个有摄像头，返回2
当根节点没有被覆盖需要给根节点加一个摄像头
代码随想录知识星球成员：海螺人