## 307. Range Sum Query – Mutable

数据结构：线段树segment tree

类似堆结构，利用数组来表示这颗树，索引从1开始，左子树在2i，右子树2i+1，父节点i/2，这样每个线段树需要2n个节点

构造：1初始化长度为2n的数组，2把n个复制到n+1到2n的位置作为叶子节点，3根据叶子节点一次一个个算出父节点

修改：1根据索引修改叶子节点，2根据父节点在i/2位置依次修改父节点

求和：从下至上，如果左边索引是基数，则先加上左边节点再遍历父节点，如果右边索引是偶数，则先加上右边节点再遍历父节点，本质上这两种情况就说明是每一层的边界

<https://leetcode.com/problems/range-sum-query-mutable/solution/>

## 309. Best Time to Buy and Sell Stock with Cooldown

### 121. Best Time to Buy and Sell Stock

遍历一遍，保存一个最小值，每次拿当前值和最小值去比较，如果比最小值小，则更新最小值，否则计算profit，保留过程中最大的profit即可

### 122. Best Time to Buy and Sell Stock II

允许多次交易，那么每次只要比min大都可以计算profit，每次都重置min即可

更新：更简单的办法是每次i+1的值比i打就可以算利润

### 123. Best Time to Buy and Sell Stock III

最多进行两次交易

方法1：分治加动态规划，理解为取一个点i，i左边进行一次交易，右边一次，利用第一题，复杂度是n^2。利用动态规划优化，第一个数组表示每个i左边一次交易的最大值，第二个数组表示每个i右边交易的最大值，最后每个i的左右两个数组相加最大值。对于单边数组，每次只需要保存对应边的最小值就可以用动态规划算出最大值，dp[i]为dp[i-1]，或者是当前值减去最小值。这样每个数组可以在n时间计算。总体复杂度为n

方法2：把这个看做两次投资，第一次投资等同于121题，第二次投资是在第一次投资的基础上再投资，所以要加上第一次的收益，所以现在的价格相当于原有价格减去第一次收益，这里价格将会动态变化就是反应动态考虑第一次投资各种可能的情况，接着用相同的方法（保留最小值，然后按照当前的价格算利润）算出两次的收益，最终两次收益的最大值就是答案。由于已经动态考虑了第一次的收益，所以最终第二次收益就是总收益。

代码见<https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iii/solution/>

<https://blog.csdn.net/lw_power/article/details/80800650>

### 188. Best Time to Buy and Sell Stock IV

最多允许k次交易。易知2k>n的话，退化为122题。如果2k<n

方法1：动态规划。三维数组dp[i][j][k]，i表示第i天，j表示还剩j次交易的机会。k表示是否持有股票，初始化i=0，j为k-1，则k为1，dp为-price，j为k，则k为0，dp为0。每一天你有四个选择，继续持有股票，继续不持有股票，买入股票和卖出股票。当然四种操作是有前提的。推导出递归表达式

dp[i][j][1]=max(dp[i−1][j][1],dp[i−1][j−1][0]−prices[i])

dp[i][j][0] = max(dp[i-1][j][0], dp[i-1][j][1]+prices[i])

最终结果在dp[n-1][j][0]中，其中j是从0到k。复杂度nk

方法2：合并法。思路是假设k很大，那我们每次上升都可以交易，然后缩小k，每次可以合并两次交易或者选出最小的交易删除。所以我们先算出所有的上升坡，假设有m个，再从m减少到k，每次都比较删除一个交易和合并两个交易带来的损失，选取损失小的那一个。直到最后收缩到k次交易。复杂度n(n-k)

### 309. Best Time to Buy and Sell Stock with Cooldown

冷却期一天，就是卖了之后一天内不能买，无限次交易

方法1：利用题目188的通用解法

方法2：状态机+动态规划，设置三个状态，sold，held， reset。sold就是卖，只能从held转换过来，held就是继续持有，可以从held或者reset过来， reset就是不持有，可以从sold和reset过来。利用动态规划，i状态的这三个值可以从i-1推导而来。结果在最后sold或者reset中。复杂度是n。状态机简单的解决了状态之间的转换问题，包括有条件的转换

<https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/solution/>

### 714. Best Time to Buy and Sell Stock with Transaction Fee

方法1：188题通用解法

方法2：309题方法2，只有两个状态，sold或者hold，即持有或者不持有

## 310. Minimum Height Trees

### 207. Course Schedule

有向图判断有没有环

方法1：利用DFS，添加一个数组用来表示当前的路径，如果访问到当前路径中有的点，就是有环，否则重找一个起点开始检查。用一个数组来表示每个点检查过没有。

方法2：利用拓扑排序。找到一个入度为0的点（如果是无向图，就是入度小于等于1），删除这个点，把它的后继的入度都减1，再找入度为0的点重复操作（代码实现上这个时候可以优先检查入度减少过的这些点，很可能为0）。最后如果所有的点入度都为0（所有的点都被删除了），那么没有环，否则有环。无向图同理。

<https://leetcode.com/problems/course-schedule/solution/>

<https://zhuanlan.zhihu.com/p/214747022>

### 210. Course Schedule II

找有向图的拓扑排序

方法1：利用DFS，每个点有三种状态，1没访问过，标记为白色，2正在访问但是还没有完成，标记为灰色，3访问完成，标记为黑色。每次从一个没有dfs过的点开始dfs，访问到白色点，标记为灰色，递归访问，访问灰色点，说明有环，访问黑色点，无事发生。最后结束的时候，把自己的颜色标记为黑色，并压进拓扑栈。最后拓扑栈的反序就是拓扑序

方法2：利用拓扑序，见207题

<https://leetcode.com/problems/course-schedule-ii/solution/>

### 310. Minimum Height Trees

首先我们是要找一个中心，它到每个点的距离都很近，尤其是叶子节点。易知这样的中心最多有两个。如果有三个，这三个中心就会形成环。

利用拓扑排序的思想，每次都减掉叶子节点，然后最终只剩1个或2个点的时候，这一两个点就是我们要找的中心

实现上：不要用degree数组，直接读取邻接表元素的长度作为度数，否则重复删除很多

## 312. Burst Balloons

动态规划。如果在数组两端同时加1的，这个问题可以看作为求0-n-1之间打得的最大值，这里不包括0和n-1。所以将子问题设计成dp[i][j]即i到j打得的最大值，这里不包括i和j（重点！！减少了很多种情况的讨论）。由此推导出递归式

dp[i][j] = max([dp[i][k]+dp[k][j]+nums[k]\*nums[i]\*nums[j] for k in range(i+1, j)])

可以理解为假设最后打掉的是k。复杂度是O(n^3)

（i和j不包含这个思路是来自leetcode discussion，但是找不到是哪一个了，原来这玩意儿是不断刷新的，下次一定及时保存，罪过罪过。其他的思路来自自己，感觉比其他方法要简单，开心）

## 313. Super Ugly Number

### 263. Ugly Number

丑数，因子只有2,3,5。不断取余就行

### 264. Ugly Number II

找到第n个丑数，见leetcode。补充是第二种方法实现上不需要真正的数组，只需要索引表示它当前乘以的是哪个丑数即可

复杂度：第一种方法是O(nlogn)，第二种是O(n)

#### 以下内容来自leetcode2

枚举法，把n之前的丑数都枚举出来，然后查找只要O(1)的时间

方法1：用堆，每次找出当前最小的丑数加入返回数组，然后用它乘以2,3,5再放回堆里（需要判断重复）

方法2：优化了方法1，方法1的本质是把丑数序列在乘以235来找下一个丑数，可以设置三个数组，每个数组分别是丑数序列乘以235。我们用三个指针分别记录每个数组的乘数分别在丑数序列的位置，然后每次对比三个数组的下一个值来选取最小值。需要注意的是如果有重复，对应的指针位置都要后移，这样就不用判断重复了。其思路类似于合并三个有序的数组，一边计算丑数数组一边利用丑数数组。答案见https://leetcode.com/problems/ugly-number-ii/solution/，中文讲解<https://zhuanlan.zhihu.com/p/111954850>

### 313. Super Ugly Number

将264题扩展到一般的有k个因子的情况，当使用第二种办法时，类似于k个数组合并，可以使用堆进行加速。最终时间复杂度O(nlogk)

<https://leetcode.com/problems/super-ugly-number/discuss/76291/Java-three-methods-23ms-36-ms-58ms(with-heap)-performance-explained>

<https://leetcode.com/problems/super-ugly-number/discuss/277313/My-view-of-this-question-hope-it-can-help-you-understand>!!!

## 315. Count of Smaller Numbers After Self

给定一个数组，找出一个数后面有几个数比它小

方法1：线段树segment tree或者Binary Indexed Tree + bucket sort。思路是如果从后往前遍历，那么只需要一种数据结构将后面所有的数保留，然后进行0-a[i]之间的range query即可。所以使用bucket sort中的bucket思想，将每个数存在的位标记为1，不存在的标记0，那么只需要0-a[i]之间求和的range query即可。range query可以用segment tree或者binary indexed tree实现。由于bucket不能处理负数，所以要先遍历，将所有的数加上数组中的最小数，使所有数变为正数。所以总的处理顺序是先遍历预处理，然后建bucket，然后从后往前遍历，遇到的数标1，然后用segment tree进行区间求和。时间复杂度nlogm，其中m是最大数和最小数之差（理论上是无穷），空间复杂度是m

方法2：利用归并排序。注意到，归并排序合并步骤，对于第一个数组中的每一个数，如果在最终结果中，它的前面有i个数来自于第二个数组，那么在它后面就有i个数比它小。因为观察发现，每次从第二个数组拿一个数放到最终结果，那么就说明它后面出现了一个比它小的数。所以我们在归并排序的时候记录下这个数据即可。时间复杂度nlogn，空间复杂度n。实现上需要注意的是数据有重复，所以不能用字典来记录结果。用列表记录结果时，难点在于归并排序之后数据的位置发生了变化，难以根据排序后的数组推测出数据原来的位置，所以这里需要修改一下归并排序，让每次排序后的结果数组不记录具体的数据，而是记录当前数据的索引，这样就可以很容易的知道原来的索引是什么。

## 316. Remove Duplicate Letters

去除重复的字符，同时要去字母序最小

贪心法

方法1：贪心加递归。首先每次找到最左边的一个字母，最左边的字母需要满足的条件是从这个字母往后，其他的字母都出现过。然后在满足条件的字母里面选字母表最靠前的那个。这样我们把这个字母前面的字母删掉，为了避免重复，这个字母后面的出现也要删掉，然后徐递归找后面的答案，最后连起来。时间复杂度O(n)，根据算法，每次递归的时候要重新构建新的字符串，所以空间复杂度O(n)。实现上，先遍历一遍找到每个字母出现的次数。然后遍历第二遍，当每个字母的出现次数还都大于0的时候，也就是后面还有的时候，说明这个字母符合条件，记录下来。然后把相应的出现次数减一。如果减一之后为零，说明之后的位置都不符合要求，因为这个字母不会再出现了。遍历终止。这样两次遍历就找出了所需要的最左边的字符

方法2：贪心法加栈。每次我们将出现的字母和栈中的字母比较，如果栈中的字母比当前的字母大，且栈中的字母之后还会出现，那么把栈中所有这种满足条件的字母删掉。然后将当前字母入栈。最后栈中保留的就是最终结果。这个思路跟手动计算的思路类似。我考虑的时候，没有考虑到可以提前知道这个字母后面还会不会出现，导致无解

实现上，为了判断栈中的字母最后还会不会出现，需要先遍历一遍，知道每个字母最后出现的位置即可

## 318. Maximum Product of Word Lengths

找出最大的两个没有交集的单词的长度乘数

技巧1：用一个整数去存储一个单词，整数的i位表示第i个字母出现过，这样每两个单词的比较是常数时间。时间复杂度为O(n^2+L)，其中L为单词的总长度，n为数组长度

技巧2：降低n^2。观察发现对于出现字母相同的单词（也就是用来表达的整数相同），我们只需要长度最长的那个，所以可以用哈希表记录每个表达数对应的最长长度。然后将哈希表里面两两无交集的数对应的长度相乘找最大值即可。当n<2^26时，复杂度不变；n>2^26时，时间复杂度为O(L)

## 319. Bulb Switcher

本质是求小于n的每个数有奇数个因子还是偶数个因子。注意这里并不需要求出有几个因子，只需要判断因子的数量是偶数还是奇数即可。数学上我们发现，因子总是成对出现，所以大部分数的因子数都是偶数。除非！这个数是平方数。平方数两个因子是重合的，导致它只有奇数个因子数。所以只需要找到小于n的平方数有几个即可。答案就是对n开方然后向下取整

<https://leetcode.com/problems/bulb-switcher/discuss/77104/Math-solution>..

## 321. Create Maximum Number

贪心加枚举。首先假设从第一个数组中取i个数，那么从第二个数组中要取k-i个数。第一个数组的长度为n1，第二个数组的长度为n2，那么i的最小值是max(0, k-n2)，最大值是min(k,n1)。然后对于这其中每个i，进行枚举，分别从对应的数组中取数然后merge。merge的过程也有点特殊，碰到相等的情况，要一直向后看，直到比出大小，然后选择大的那个的当前数值。而对于从每个数组中取数，最简单的方法就是用栈，每次循环判断当前数值和栈头的大小，如果当前数更大且还剩下足够的数，那么就把栈头删了，最后将当前的数入栈。但是这样其实有很多重复计算，可以优化。观察到第一次drop数的时候，我们是从前往后找到第一个小于后面的数把它删掉，第二次是找到第二个，最后删除的时候所有提取的数已经是一个倒序的有序数组了，这时候只要依次从最后删除即可。这样算法总的时间复杂度为n+m

<https://leetcode.com/problems/create-maximum-number/discuss/77285/Share-my-greedy-solution>

<https://leetcode.com/problems/create-maximum-number/discuss/77287/C%2B%2B-16ms-FASTEST-beats-97>.

<https://www.cnblogs.com/grandyang/p/5136749.html>

## 322. Coin Change

换钱问题，无限硬币，普通动态规划问题。递归式dp[i] = argmin(dp[i-k]+1)。由于硬币无限，这里其实并不需要维度表示硬币，合在表达式里面即可

## 324. Wiggle Sort II

### 280. Wiggle Sort

方法1：排序。排序之后从第二个元素开始，每两个元素交换一下位置即可。

方法2：遍历。如果每个元素和它后一个元素的位置关系不对，则交换这两个元素即可。可以归纳法验证当位置不对的情况只有两种，单调增和单调减。每次交换都可以满足要求

<https://leetcode.com/problems/wiggle-sort/solution/>

### 324. Wiggle Sort II

方法1：排序。从中间的数开始，每次取一个前半段末尾的数，取一个后半段末尾的数，即可满足要求。时间复杂度nlogn，空间复杂度n，也可以优化到1

<https://www.cnblogs.com/grandyang/p/5139057.html>

方法2：index\_mapping.基于方法1，可以发现，重点在于找到中位数，然后可以把数组分为三个部分，大于中位数的，等于中卫数的，小于中位数的，对于这三个部分的每个元素之间的顺序不重要，所以不需要整体的排序。如果我们把原数组顺序变为先是奇数位，后是偶数位的话就会发现这个顺序就是按照大于等于小于中位数的顺序排列的。所以我们可以设置一个新的数组，它的索引就是原数组索引按照先是奇数位后是偶数位的顺序。在新的数组上，我们进行三层的partition即可。也就是将大于中位数的往前调，小于中位数的往后调。由于这个数组跟原来的数组之间有一一对应的关系，所以我们并不真的需要这样一个数组，只需要进行索引之间的mapping即可。然后常规的调用partition函数即可。索引之间的关系式A(i)=nums[(1+2\*(i)) % (n|1)]，左边是新数组，右边是原数组。

<https://leetcode.com/problems/wiggle-sort-ii/discuss/77677/O(n)%2BO(1)-after-median-Virtual-Indexing>

## 326. Power of Three

方法1：循环相除，时间复杂度logn

方法2：将整数转换为3进制表达，如果表达式是1后面都是0的话，那就是三的倍数。可以推广到任意k，比如2进制和10进制，表达式是1后面都是0就是2或10的倍数。时间复杂度logn

方法3：直接取对数，复杂度取决于库的实现

方法4：由于整数的有限性，可以推导出整数范围内，最多有3^19，那么可以把这20个数预先算出来，然后直接看输出是不是在这里面就可以了。复杂度O(1)

## 327. Count of Range Sum

首先区间求和得到一个新数组sum，其中sum[i]表示0-i之间元素的和。题目转化为对于任意j，求j前面有多少个i，使得sum[j]-sum[i]在lower和upper之间。类比315题，因为315题是说，对于任意j，求j后面有多少个i，使得s[j]-s[i]在0和无穷之间。所以同样，我们使用归并排序，在排序的过程中找一下前面有多少个i满足条件，即使得s[i]在s[j]-upper和s[j]-lower之间。注意由于s[j]是递增的，所以区间也是递增的，所以这个区间的起始和终止的index也是递增的。所以整个查找过程只是遍历一遍前面的数组。所以总体时间复杂度不变，仍然是归并排序的时间复杂度，即nlogn。实现上，和315题一样，排序的时候存索引而不是具体的值。同时把区间索引设置成右边是开区间更利于计算总数。

<https://leetcode.com/problems/count-of-range-sum/discuss/77990/Share-my-solution>

## 328. Odd Even Linked List

把一个链表转成先是索引为奇数的值，然后是索引为偶数的值。就是有个标志位判断奇偶数，然后弄两个header，遍历完之后把奇数链表指向偶数链表即可，同时也要把偶数链表的结尾设置成null

## 329. Longest Increasing Path in a Matrix

在矩阵里面找到最长增长路径

方法1：dynamic programming + DFS。根据动态规划，如果邻居大于自己，那么自己的值就取决于邻居的值加一。所以用数组记录每个点的值，如果没有计算过，就递归计算，然后计算自己的值最后取最大的。时间复杂度mn，空间复杂度mn

方法2：peeling onion。根据动态规划思想，将问题转化为图问题。每个点，有有向边指向比它大的邻居。那么问题就变成求这个图中最长路径。利用peeling onion（类似于拓扑排序），我们首先找到叶子节点，然后删掉，再找新的叶子节点，不断循环。可以发现本题的答案就是层数，类似于一层一层剥洋葱，每一层我们都会删掉叶子节点。时间复杂度mn，空间复杂度mn，因为要存每个点的出度。

<https://leetcode.com/problems/longest-increasing-path-in-a-matrix/solution/>

## 330. Patching Array

贪心法。首先注意到，如果[1,miss)这个区间已经被覆盖的话，如果我们加上x，那么[x, x+miss)这个区间也会被覆盖，所以如果我们遍历数组，这个时候加上数组后面一个值x就会产生[x, x+miss)这个新的覆盖区间，为了保证覆盖区间不会断档，那么这个x势必要小于等于miss。如果数组的下一个数不是小于等于miss的话，那么就只能patch。为了保证patch的数目最少，贪心法可知这个patch就等于miss。同时这时候覆盖区间由[1,miss)变为[1,miss+miss)。所以整个过程就是初始化miss为1，然后不断的遍历数组，如果数组的数小于等于miss，则更新miss，否则加patch并更新miss，直到miss大于n。时间复杂度m+logn，因为这里每次patch的时候miss是翻倍的，所以是加logn。空间复杂度1。算法的正确性证明可以通过一般的反证法证明贪心法正确性来证明，具体见

<https://leetcode.com/problems/patching-array/solution/>

## 331. Verify Preorder Serialization of a Binary Tree

方法1：用栈，每次将节点入栈，如果栈尾是叶子节点，即一个数字加两个#号，则不断循环删除叶子节点，即将这三个符号变成一个#号，再看栈尾是不是叶子节点。时间复杂度n，空间复杂度n，因为即使一开始不用split，也需要栈存节点

方法2：注意到如果尝试把树构造出来，那么每次出现数字，会填满一个空档，同时又创造两个空档（因为有两个孩子），而每次出现#号，就是填满一个空档。所以我们可以数空档数目，初始化的时候是1，每次先用字符填空档，也就是空档数先减一，如果空档数是复数，那么树就是无效的，否则在判断是不是数字，如果是数字还要增加两个空档。最后的时候，空档数必需是0才是有效的。时间复杂度n，空间复杂度可以优化到1

<https://leetcode.com/problems/verify-preorder-serialization-of-a-binary-tree/solution/>

## 332. Reconstruct Itinerary

欧拉路径Eulerian Path：从一点出发，遍历所有边一次且仅一次的路径

欧拉路径Eulerian cycle：闭合的欧拉路径，也就是起点和终点一样的路径

欧拉图：含有欧拉环的图

半欧拉图：含有欧拉路径的图

无向图存在欧拉环的充要条件：连通图，所有点的度为偶数

无向图存在欧拉路径的充要条件：连通图，所有点的度数为偶数，或者有且仅有两个点度数为奇数，其他点度数为偶数

有向图存在欧拉环的充要条件：连通图，所有点的入度等于出度

寻找欧拉路径的算法：Hierholzer算法（逐步插入回路法）

Hierholzer算法：从一个点开始DFS，过程中删除访问过的边，如果遇到一个顶点没有边可以访问，则入栈，同时回退DFS，不断循环。最终栈中顶点的反序就是欧拉路径

Hierholzer算法时间复杂度E，空间复杂度V+E

本题时间复杂度ElogE/V，因为要对每个点对应的边进行排序，极端情况是中心图，所有的点都和中心相连，那么中心有E/V个点，排序的复杂度是E/VlogE/V，对V个点排序，乘以V得到最终的复杂度，空间复杂度V+E

实现上，可以递归，每次把点v的邻居拉出来递归并删除，最终把v推入栈即可

可以迭代，记录当前访问的点和邻居即可，向下递归就是list中添加，回退就是删除直到空

<https://leetcode.com/problems/reconstruct-itinerary/solution/>

<https://zhuanlan.zhihu.com/p/108411618>

<https://blog.csdn.net/qq_19446965/article/details/104783782>

## 334. Increasing Triplet Subsequence

方法1：自己的方法。三个指针，先找到第一个指针的位置，第二个指针是第一个指针加一，然后找第三个指针，如果第三个指针大于第二个指针，那么返回ture，如果第三个指针小于第二个指针但是比第一个指针大，那么更新第二个指针，否则更新第一个指针。最后都没找到第三个指针就返回false。时间复杂度n，空间复杂度1

方法2：答案的方法。我们的方法需要找到i j k三个索引，导致第一个索引i需要特殊处理。如果我们转为找对应的三个数，那么i指向的就是最小的数，j指向的是第二小的数，k指向的是一个比i和j指向的数大的数。那么就可以初始化这三个数为无穷大，然后不断更新。不需要特殊处理。时间复杂度n。空间复杂度1

<https://leetcode.com/problems/increasing-triplet-subsequence/solution/>

## 335. Self Crossing

## 337. House Robber III

### 198. House Robber

普通的动态规划，dp[i]=max(dp[i-1], dp[i-2]+nums[i])

### 213. House Robber II

方法1：动态规划，增加每个i有两个状态，一个是rob第0个，一个是不rob第0个。最后头尾特殊处理，中间的递归式依然简单

方法2：转化为198题，这个题1和n不能同时rob，那么就有两种情况，一种是rob第1到n-1之间的房子，一种是rob第2到n之间的房子，这就跟198题一样，之后取最大值就行

### 337. House Robber III

动态规划。对于每一棵树，它的最优值有两种情况，一种是包含自己，那么就只能抢自己的孙子，另一种是不包含自己，那么就是抢左右子树然后加起来

实现方法1：递归和Memoization。为了避免重复计算，用memoization将算过的节点记录下来。时间复杂度n，空间复杂度n

方法2：将树转换为数组，然后对数组直接用动态规划。这样避免了递归的调用。由于树不是完全二叉树，所以转换成的数组后，父子节点的下标没有绝对的关系，所以用map把每个节点子节点的位置记录下来即可。时间复杂度n，空间复杂度n

## 338. Counting Bits

### 191. Number of 1 Bits

方法1：每次右移，然后看看末位是不是1，是的话加一，然后接着右移，直到为0

方法2：利用n&(n-1)得到的数比n的二进制表示少一个1

### 338. Counting Bits

动态规划。利用n&(n-1)比n少一个1。得递归式dp[i] = dp[i&(i-1)]+1

## 341. Flatten Nested List Iterator

把嵌套链表展开，实现next和hasNext的判定

方法1：递归把链表展开为数组，然后实现数组的next和hasNext，优点是next和hasNext复杂度为常数级别，缺点是constructor复杂度高，空间复杂度也高

方法2：用栈，类似于dfs，每次遇到列表，则删掉列表，把链表中的内容反序入栈。直到栈头是数字，缺点是每次入栈的复杂度太高

方法3：用双栈。我的方法。一个栈记录当前的列表，一个栈记录索引，这样就减少了入栈的成本。空间复杂度是D。具体实现上要实现两个功能，一个是找到下一个值但是不移动索引，这里需要处理空链表。一个是移动索引，这里不处理空链表。而next和hasNext每次都要先调用获取下一个值但不移动索引，然后再做其他操作。因为这里任何空链表尤其是嵌套的空链表不能算作元素

方法4：仅限于支持迭代器的语言。优化了方法3，不需要分开记录，只需要一个栈，记录每个链表的迭代器即可。同样的，由于空链表和嵌套的空链表不能算，这里需要实现一个类似于Peeking Iterator的功能，因为迭代器只能向前走，不能回头，而hasNext中需要走迭代器又不移动指针。时间复杂同方法3，空间上减少了一个D，但仍然是D

方法5：仅限于支持生成器generator的语言。如Python，利用yield和yield from实现。如果当前数是整数，那么直接yield，否则就yield from这个数组。相当于生成器的迭代调用。同样由于生成器不能往回走move backward，要实现类似于Peeking Iterator的功能。空间复杂度依然是D

<https://leetcode.com/problems/flatten-nested-list-iterator/solution/>

## 342. Power of Four

以下只列出时空复杂度都是常数的方法

方法1：precomputation。最大是4的15次方，可以预先把这些数算好。实现上，算这些数的时候，可以用前一个乘以4得到后一个数，避免指数运算。

方法2：易知2的指数的条件是二进制表示时只有一个位是1，其他位都是0。也就是n&(n-1)等于0。而在2的指数的基础上是4的指数条件是这个位必须是奇数位，也就是说与0xaaaaaaaa与之后为1，所以答案num > 0 and num & (num - 1) == 0 and num & 0xaaaaaaaa == 0

方法3：结合数学知识。如果一个数是2的指数但不是4的指数，那么它模3余2，4的指数模3余1

<https://leetcode.com/problems/power-of-four/solution/>

## 343. Integer Break

把一个正整数分为k个数相加，要求这k个数乘积最大

数学上可以证明，最优因子是3，所以dp[n]=dp[n-3]\*3，但是有特殊情况，那就是n-3之后小于4，因为1,2,3的结果比自身还小，所以需要特殊处理n小于等于6的情况。正确性证明见discussion

<https://leetcode.com/problems/integer-break/discuss/80721/Why-factor-2-or-3-The-math-behind-this-problem>.

## 344. Reverse String

遍历一半就行，i对应的为n-i-1

## 345. Reverse Vowels of a String

两个指针即可，Python中str不可变，所以先转为list，再用join转回来比较省时间

## 347. Top K Frequent Elements

方法1：用堆，时间复杂度klogn，nlogk

方法2：Hoare's selection algorithm霍尔选择算法。类似于快排，先找一个数，将它放到对的位置，如果这个数刚好在n-k的位置，那么后k个数就是答案，否则根据它的位置递归放置其他的数知道找到n-k。时间复杂度平均n，最坏情况n^2

## 349. Intersection of Two Arrays

两个集合交集即可

## 350. Intersection of Two Arrays II

两个字典记录频率，然后对key取交集，对value取最小值是key在结果中的个数

## 352. Data Stream as Disjoint Intervals

方法1：我的方法。用一个有序数组表示interval，这样查找时间是logn，但是添加和删除时间复杂度是n。返回interval的时间是1

方法2：用树存储interval，这样查找和添加删除时间是logn，但是返回interval的时间是n

具体用哪种方法取决于应用场景

<https://leetcode.com/problems/data-stream-as-disjoint-intervals/discuss/82553/Java-solution-using-TreeMap-real-O(logN)-per-adding>.

## 354. Russian Doll Envelopes

### 300. Longest Increasing Subsequence

用一个数组dp存储长度为i+1的增长序列的末尾的最小值在数组的第i位（官方答案的说法有点奇怪），那么可以证明这个数组是递增的。然后对于每一个数，我们可以用它来增加数组的长度，或者是更新数组中的值。如果它比所有的数都大，那么就是说这个数可以作为现有的递增序列的基础上加一个tail，相应的LIS长度也加一。如果它在dp数组中，满足大于第j-1位，小于第j位，说明我们存在一个长度为j的递增序列末尾是之前的一个数，而这个数现在比之前的数小，但是又比长度为j-1的递增序列的末尾大，那么我们用这个数去更新长度为j的递增序列的末尾它依然是递增序列，但是末尾更小，更容易变长了。所以通过这样不断的更新，最后数组的长度就是LIS的长度，但是数组中的数不是LIS

<https://leetcode.com/problems/longest-increasing-subsequence/solution/>

<https://leetcode.com/problems/longest-increasing-subsequence/discuss/74824/JavaPython-Binary-search-O(nlogn)-time-with-explanation>

<https://www.cnblogs.com/grandyang/p/4938187.html>

### 354. Russian Doll Envelopes

相当于是300题的二维版本。首先我们按照第一维排序，然后就相当于在第一维递增的情况下，找第二维也要递增。这里为了处理第一维相等的情况，我们把第二维倒过来排序。这样排序之后把第二维拎出来找LIS就是最终的结果。时间复杂度nlogn

## 355. Design Twitter

我的做法是对每个user维护一个list来放所有的news，需要的时候我们只取后十个就行。优点是拿newsFeed很快，缺点是每次follow和unfollow都要大动干戈，也就是对自己的newsFeed要整个重塑。submit之后竟然意外的很快

## 357. Count Numbers with Unique Digits

数学问题，可以看做动态规划，f(n)表示长度为n的结果，注意首位不为0

那么f1 = 10\*1，f2 = 9\*9，f3 = 9\*9\*8，f4 = 9\*9\*8\*7，依次类推。本题的答案就是f1到fn的相加

<https://leetcode.com/problems/count-numbers-with-unique-digits/discuss/83041/JAVA-DP-O(1)-solution>.

## 363. Max Sum of Rectangle No Larger Than K

首先对于一维数组，如果要找到这样一个间距，使得和不大于k，那么就是先对prefix求和，然后对于每个i，在它前面找一个j，使得s[i]-s[j]<=k，那么s[j]就是大于等于s[i]-k的最小值。这样总体可以在nlogn时间内实现。那么对于二维数组，我们把i-j内的数按照行数求和，这样就转换为了一维数组。进而问题得解。时间复杂度是m^2nlogn，其中m是行数，n是列数。由于对行数是每个i-j枚举，所以是平方，而列数可以用一维数组的方法，所以是nlogn。如果行数远远大于列数，也可以按照列数枚举，行数优化，或者将矩阵转置。

## 365. Water and Jug Problem

方法1：状态机+枚举。用两个数分别表示每个桶里面的水量，首先初始状态是00，然后对于每一个状态，按照规则可以进行一定的操作转换到下一个状态，可以用bfs将所有状态都枚举出来。任意一个时刻，如果有一个状态下两个桶的水量达到想要的，就返回true，否则返回false

方法2：数学方法。首相根据裴蜀定理Bézout's identity，对于任意整数a和b，假设d是最大公约数Greatest Common Divisor(GCD)，那么对于任意整数x和y，有ax+by是d的倍数。且存在整数x,y使得ax+by=d。对应在这个问题中，如果x是负的，那么就是说我们需要empty这个桶x次，如果是正的，就是说要填满水x次。两桶相互倒水可以看做对于一个桶，是fill一定量水，另一个是放水。最终的结果总是可以用ax+by来表示的。那么由于最后我们需要z桶水，就是说z必须是d的倍数。Corner case是一个桶中的水是0，另一个桶是满的，这样也能达到要求，但是不在上述讨论范围内。最后最大公约数的求法有三种，一种是更相减损法，一种是辗转相除法，又叫欧几里得算法（Euclidean algorithm），方法3是Stein算法。

<https://leetcode.com/problems/water-and-jug-problem/discuss/83715/Math-solution-Java-solution>

<https://baike.baidu.com/item/%E8%A3%B4%E8%9C%80%E5%AE%9A%E7%90%86/5186593?fr=aladdin>

<https://blog.csdn.net/qq_34992845/article/details/53043339>

<https://www.cnblogs.com/drizzlecrj/archive/2007/09/14/892340.html>

## 367. Valid Perfect Square

方法1：二分搜索法。n开方在1-n之间，所以我们在这之间进行二分搜索即可

方法2：Newton's Algorithm。利用数学等式，对于任意一个不能使得f(x)=0的x，我们可以通过更新x’ = x – f(x)/f’(x)来使得它更接近答案。本题可以看做f(x) = x^2 – n. 时间复杂度logn

<https://leetcode.com/problems/valid-perfect-square/solution/>

## 368. Largest Divisible Subset

动态规划。首先对于任意一个已有的序列abc，如果d能被c整除，也就是d%c=0，那么abcd一定是一个满足条件的序列。所以先排序，定义dp[i]是以i结尾的最长的满足条件的序列，然后对于每个i，在前面找能被它整除的数，dp[i]就等于所有能被i整除的j的dp[j]加一。最后答案就是那个最大的dp[i]。时间复杂度n^2（只有这个方法能行，其他方法都有极端case过不了）。空间上，为了节约，不把每个i对应的序列保留（否则复杂度是n^2），我的做法是只保留指向前一位的指针，空间复杂度n，而且重构序列的时候更快

## 371. Sum of Two Integers

方法1：bit manipulation。用一个mask，逐位相加。时间复杂度是32，也就是1

方法2：bit manipulation + XOR。我们把问题总是可以转换为两个数相加，或者两个数相减。如果是两个数相加，那么没有进位的相加结果是x异或y，进位的位置是(x&y)<<1，接下来就是把这两个数相加即可，如果我们一开始就把y看做进位的话，就可以一直循环相加直到进位是0。如果是两个数相减，没有借位的结果不变，借位的位置是(~x&y)<<1，同样如果一开始把y看做借位，就可以循环相减。

<https://leetcode.com/problems/sum-of-two-integers/solution/>

## 372. Super Pow

我的方法：每次把b除以二，然后问题就变成求a^2和b/2的结果，如果有余数，还要乘上a。每次都把a去模1337使得它总是不大于1337。时间复杂度是logb，假设b的长度是n，由于b中每个位上最大是9，那么复杂度其实也是4n，也就是n。这里选择除以2，一是因为好算，二是由于1337^2不会溢出，而1337^3会溢出，所以也只能除以2。此方法与wiki上Modular Exponentiation的方法一致，见下1,。当然也可以围绕1337搞复杂的数学推导，见下2

<https://en.wikipedia.org/wiki/Modular_exponentiation>

<https://leetcode.com/problems/super-pow/discuss/84466/Math-solusion-based-on-Euler's-theorem-power-called-only-ONCE-C%2B%2BJava1-line-Python>

## 373. Find K Pairs with Smallest Sums

堆+遍历。假设对于第一个数组，前i位分别对应是与第二个数组中ai相加，那么下一次的取值就是在前i位与ai+1相加，和i+1位与0相加中取得，用堆存这个数值并进行选取。时间复杂度klogn，但是最坏情况很难达到，一般情况下堆很小。

## 374. Guess Number Higher or Lower

二分法。或者可以用三分法，四分法，五分法等等。可以证明都不如二分法。二分法复杂度是，三分法复杂度是。可以证明后者更大。所以不能总是用O看复杂度。

<https://leetcode.com/problems/guess-number-higher-or-lower/solution/>

## 375. Guess Number Higher or Lower II

动态规划。观察发现dp[i][j] = dp[i][k-1] + k + dp[k+1][j]。可以理解为先猜k，如果不中，问题就变为两个部分了。时间复杂度n^3。ij型动态规划实现上先控制j，然后让i从0-j遍历。

## 376. Wiggle Subsequence

贪心法。最后结果就是要看有多少次上升和下降，所以多次上升可以看做一个，多次下降也是。只有符号有变化才把长度加一。特殊情况是相等，不作数。时间复杂度n，空间复杂度1

## 377. Combination Sum IV

### 39. Combination Sum

DFS+枚举，每个序列的第一个数是i，那么下一个数可以是i，或者i+1，i+2等等。注意这里用DFS甚至是递归更节省时间。回溯法的话时间复杂度O(n^(t/m+1))，t是target，m是list中最小数。可以理解为树高是这么高，所以数节点数这么多，但是实际上要比这个少很多。时间辅助度是t/m，也就是树高。复杂度分析见下

<https://leetcode.com/problems/combination-sum/solution/>

### 40. Combination Sum II

DFS+枚举。先要对数字进行特殊处理，相同的数看做同一个，用counter计数即可。枚举的时候也要有node变量表示当前数还剩几个。时间复杂度2^n，最坏情况就是每个数都要看看取还是不取，宏观求得法。分析见

<https://leetcode.com/problems/combination-sum-ii/>

### 216. Combination Sum III

DFS+枚举。因为k和n都是1-9，可以进行更多的预处理，比如提前判断这个list不行。

### 377. Combination Sum IV

动态规划。直接枚举会超时，因为复杂度是n!。利用动态规划，对于每一个值i，它的可能值来源于每次从列表中取一个数aj，用aj作为开头，看剩下可能性，也就是dp[i]=sumj{dp[i-aj]}。时间复杂度tn，t是target，n是列表长度，空间复杂度t

## 378. Kth Smallest Element in a Sorted Matrix

方法1：堆。类似于合并n个递增的序列，维持一个size为n的堆，每次从中拿一个，直到k个。时间复杂度是n+klogn。还是挺高的，因为k最高是n^2。而且没有利用行和列同时增长的条件

方法2：二分搜索。易知最小值一定是第一个数，而最大值一定是最后一个数。我们可以对值进行二分搜索。每次搜索后计算有多少个数小于等于这个值，如果等于k，说明这个值就是答案，如果大于k，说明值大了，反之同理。通过对行的正向遍历和列反向的遍历，我们可以在2n的时间里找到有多少个数小于等于value，因为行永远递增，列永远递减，所以最多是2n次比较。为了加速，我们同时记录不大于value的最小值和不小于value的最大值。这样每次更新start和end会很快。同时也是为了处理重复。总体时间复杂度O(nlog(max-min))。与值有关，但是实际很快。

<https://leetcode.com/problems/kth-smallest-element-in-a-sorted-matrix/solution/>

## 380. Insert Delete GetRandom O(1)

hashmap+list。首先，要实现插入和删除是O(1)的，用hashmap。那么如何实现随机获得元素，需要用list把所有的key记一记。但是list删除不是O(1)的，这里的技巧在于，我们每次都把最后一个元素和要删除的元素换位置，然后就可以通过删掉最后一个元素实现删除。

也就是说，hashmap+list可以实现一个增删改查都是常数时间的数据结构，但是元素是无序的

## 381. Insert Delete GetRandom O(1) - Duplicates allowed

hashmap+list。跟380题一样，只不过这次每个value可能对应多个key，所以hashmap中每个value对应的key的集合

## 382. Linked List Random Node

方法1：直接把list转成array，然后随机选取

方法2：Reservoir Sampling水塘抽样。算法用于从未知n个sample中等概率选k个sample。本题中是选一个，k=1。算法为先用前k个元素填满水库，然后对于第x个元素，x被选中的概率是k/x，如果x被选中，那么就用x替代水库中随机一个元素，一直循环直到数据流结束。实际实现有更多技巧。算法优点是可以对事先不知道size的sample取样，而且不需要把n个sample都存到内存中。本题时间复杂度n，空间复杂度1

<https://leetcode.com/problems/linked-list-random-node/solution/>

<https://www.cnblogs.com/strugglion/p/6424874.html>

<https://en.wikipedia.org/wiki/Reservoir_sampling>

## 383. Ransom Note

第一个单词中的每个字母都要在第二个单词出现且次数更多

方法1：用一个hashmap构造第二个，然后遍历第一个来重构，重构成功就是true

方法2：排序，然后用两个指针遍历，相同同时后移，不同一个后移，如果第一个到终点就是true

<https://leetcode.com/problems/ransom-note/solution/>

## 384. Shuffle an Array

Fisher-Yates Algorithm（费雪耶兹算法，洗牌算法）。反向遍历，对于第i个数，每次从0-i中找一个数和它换位置。时间复杂度n。有点类似380题

<https://leetcode.com/problems/shuffle-an-array/solution/>

## 385. Mini Parser

方法1：递归；方法2：用栈迭代

## 386. Lexicographical Numbers

每次拿出栈的最后一个元素，下一个元素有三种情况，一是在后面补零，二是这个数加一，三是这个数退位加一。每种都要考虑溢出和有9的情况(corner case)

## 387. First Unique Character in a String

哈希表记录一下出现次数

## 388. Longest Absolute File Path

以换行为分割，用\t数量来看等级，用stack存前面级数的长度

## 389. Find the Difference

哈希表或者排序

<https://leetcode.com/problems/find-the-difference/>

## 390. Elimination Game

二分法，每次都会去掉一半的数。把握开始的数和间距即可，初始时间距是1，每次间距翻倍，而第一个数有时删掉，那么下一个数成为第一个数，也就是原来的第一个数加间距，如果不删掉就保留。最终的这第一个数也是最后一个数。时间复杂度logn，空间复杂度1

## 391. Perfect Rectangle

方法1：堆heap。先遍历找到四个角的值，接下来就是用长方形把这之间的平面填上，我们用堆每次找到最小的长方形和最小的需要填充的区域（这里的最小是按列表进行坐标排序，不是面积），如果左下角对不上，就返回false。否则分情况讨论，取决于把要填充的区域切块还是把长方形切块。新的区域放入堆中，然后从堆中每次拿最小的循环。如果最后长方形用完了，要填的区域也没有，都填满了，就返回true，否则false，时间复杂度nlogn

方法2：分析法。分析发现，要满足拼成完美的长方形，有两个条件，一是所有长方形的面积之和等于最终长方形的面积，二是除了最终长方形的四个角之外，每个长方形的四个角都要在整个长方形的角集合中出现偶数次，就是保证长方形都是挨着的。时间复杂度n

<https://leetcode.com/problems/perfect-rectangle/discuss/87181/Really-Easy-Understanding-Solution(O(n)-Java)>

## 392. Is Subsequence

两个指针，相同同时移动，不同只移动第二个

follow up：如果有很多个s要和一个t比较，那么用hashmap把t字符的位置记住，对于s中任意一个字符，都去t中找最小的位置，然后s整体的位置要单调增

<https://leetcode.com/problems/is-subsequence/solution/>

## 393. UTF-8 Validation

分类讨论即可，由于这里数字太固定了，不用位操作，直接比大小更快

## 394. Decode String

迭代过去，用栈处理嵌套即可

## 395. Longest Substring with At Least K Repeating Characters

方法1：Divide And Conquer分治。对于每个字符，如果它总共的出现次数不超过k，那么它就将字符串分为两个部分，最长串就在其中一个部分。递归即可。最坏情况时间复杂度n^2，就是每次都只能去掉一个字符，不过一般复杂度大概在nlogn，空间复杂度n

方法2：Sliding Window滑动窗口。For each h, apply two pointer technique to find the longest substring with at least K repeating characters and the number of unique characters in substring is h.具体为如果unique的字符超出了，那么移左pointer，如果字符不够，移右pointer。用数记录unique characters和有几个repeat了k次。每次移动完都判断是不是符合要求，如果符合就把结果重置。相当于把问题转化为求一个window里面有n个不同的字符，且每个重复k次，由于最终的答案必然也是这样一种window，所以答案就在其中，且非答案的同种window都比答案小。时间复杂度26n，也就是n，空间复杂度1

<https://leetcode.com/problems/longest-substring-with-at-least-k-repeating-characters/solution/>

<https://leetcode.com/problems/longest-substring-with-at-least-k-repeating-characters/discuss/87739/Java-Strict-O(N)-Two-Pointer-Solution>

## 396. Rotate Function

找规律，fn+1 = fn+sum-N\*nums[-i]

## 397. Integer Replacement

方法1：BFS，时间复杂度logn，空间复杂度logn

方法2：数学分析。重点在于n是奇数的情况。可证如果n模4余1，那么n应该减一，如果余3，那么加一，3除外。简单证明如下，n是奇数，那么n是2k+1，n+1=2k+2，n-1=2k，

(n+1)/2=k+1，(n-1)/2=k，那么k和k+1谁是偶数，对应的操作一定更少，反推可得n模4

的情况。

<https://leetcode.com/problems/integer-replacement/discuss/87928/Java-12-line-4(5)ms-iterative-solution-with-explanations.-No-other-data-structures>.

<https://leetcode.com/problems/integer-replacement/discuss/87948/Python-O(log-n)-time-O(1)-space-with-explanation-and-proof>

## 398. Random Pick Index

方法1：用hashmap记录位置list即可

方法2：Reservoir Sampling水塘抽样。样本size依然是1，这里target的数量未知，适合使用

## 399. Evaluate Division

转换为图问题，每一个equation，都是一条正向边，值不变，一条反向边，值相反。最后query，就是看两点能不能连通，结果就是路径上值的积

方法1：BFS。时间复杂度mn，缺点是重复计算多，每一个query都要遍历图

方法2：union-find结构。union-find结构可以很快判断是不是连通。我们给每一点设一个相对于根节点的权重，这样如果连通，可以很快计算结果。同时我们使用lazy update的方法，每次不更新其相对根的权重，等到查询的时候，因为要接到根节点，在同时更新权重。union-find每个操作的时间复杂度是log\*n（近似于O(1)），所以总体是(m+n)\*log\*n

<https://leetcode.com/problems/evaluate-division/>

<https://blog.csdn.net/guoziqing506/article/details/78752557>

<https://zhuanlan.zhihu.com/p/30293447>

[https://en.wikipedia.org/wiki/Disjoint-set\_data\_structure#Proof\_of\_O(log\*(n))\_time\_complexity\_of\_Union-Find](https://en.wikipedia.org/wiki/Disjoint-set_data_structure#Proof_of_O(log*(n))_time_complexity_of_Union-Find)

<https://en.wikipedia.org/wiki/Iterated_logarithm>

## 400. Nth Digit

数学分析。先确定n个数位是几位数，然后再找第几个数，再找第几位