### 如何做笔记

- **易错点记录**，只要不是一次过的题，就存在易错点，比如语法错啦，边界条件少啦等等。很典型的递归错误就是忘了加退出条件。这样的错误记录了两三遍之后，就再也不会错了。

- **语法记录**，很多常用的语法，比如各种数据结构的初始化比如初始化一个list of list: List<List<Integer>> lol = new LinkedList<List<Integer>>();`，再比如数组排序，以及如何使用compare函数

- **知识点记录**，给每一类题都建一个文件，推荐Dropbox的paper或者Evernote。按前面第二步的方法做的时候将题目归类并记录。记录的时候，就用一句话记录这题的题号，用的关键知识点，和关键代码，这样的好处是下次复习的时候可以快速想起，比如明天就要面试FB了，今天晚上你不可能做新题了，你就可以看你记录的这些关键点，复习效率特别高。

- **好题分析**，上一点是每题必须做，这一点是选择性的，当你做到一道题卡了很久，或者学到很多知识点（有好几个follow up），你可以写一篇文章来分析，来记录你是如何一步一步优化做出最优解的。

2023年9月10日星期日：

1. [1359] Count All Valid Pickup and Delivery Options

动态编程/递归法/数学性质分析，重点在于利用数学归纳法或者排列组合的知识分析i->i+1的过程。此外，注意题目中要求【取模】。

1. [13] Roman To Integer

易错点是注意循环结构何时退出、if和else if的区别（if语句会持续执行，而else if在if为真后就不执行）。

2023年9月19日星期二

1. [234] Palindrome Linked List

检查一个单向链接的链表是否是回文的。

默认解法很好想，复制一遍即可。递归解法比较巧妙，但是在内存占用上并不占便宜。唯一的内存o（1）解法要在原链表中将后半段反转，反转的算法非常精妙，用两个节点跟踪当前节点和前一个节点（稍后会变成后一个节点），用一个临时节点存放后一个节点的地址。此外，这道题也提供了一个单向链表寻找重点的方法：两指针赛跑，慢的一次一个节点，快的一次两个节点，最终慢的节点会在中点（左）节点停下。

1. [383] Ransom Note

检查杂志字符串里面的字母够不够组成绑票条字符串。

这道题应该使用哈希图而不是哈希表，因为当我们用字母做键出现次数做值时，每个键只对应一个值。当且仅当需要每个键对应多个值时才使用哈希表。哈希表可以创建一个，根据另外一个字符串做减法，也可以创建两个做加法再对比，区别不大。

此外，注意Java的string中有一个函数indexOf(char)可以返回char对应的索引，没有则返回-1，可以用于检测并删除某个字母是否在另外一个字符串。

还有一个思路是将两个字符串按字母顺序排列后压入栈，分别对比两个栈的字母，根据字母先后顺序判断真伪。

最后，注意一些常用方法、结构的时间、空间复杂度，如快排O(nlogn)，哈希表的插入和搜索都是O（1）等。

2023年9月21日星期四

1. [704]Binary Search

二分查找的简单实现，只是要注意在这种简单的实现中没必要使用递归，因为递归会占用大量运行时堆栈的内存，而这道题只需要简单的循环就能实现。

2023年9月23日星期六

1. [3] length Of Longest Substring

这道题是要求找出输入字符串中无重复字符的最大子字符串的长度。这道题是典型的字符串/数组题，需要用到哈希图和滑窗法缩短运行时间。创新点在于用哈希图查询某字母是否出现，并将最后一次出现重复的字符的索引作为值存在图中。答案中对于索引的更新非常有技巧： i = Math.max(hashmap.get(s.charAt(j)) + 1, i);  
先将i更新为出现重复的字符对应的索引的值+1和当前i值中的最大者，再将结果更新为当前窗口大小和当前结果中的最大值。如此一来，既规避了某个字符在该滑窗之前就出现带来的影响，也巧妙地减少了滑窗的次数，使得运行时间进一步下降。

2023年9月23日星期六

1. [2]Add Two Numbers

给两个倒序自然数的链表，以链表的形式输出该两自然数相加的结果。

这道题的思维不难，只需要直观的采用竖式逐位计算即可。难点集中在如何把握进位问题，答案采用了将其存储在整数变量中并每次计算时更新的方法。最后注意循环结束的条件除了要判断两个链表是否循环完毕，还要检查是否还需要最后进位。

1. [4]Median of Two Sorted Arrays

给两个已排序的数组，返回这两个数组的中位数，要求运行时间在O(log(m+n))。

这段时间做的第一个hard题，其实解题思路很直观，但是特殊情况的处理太tricky了。

这道题的数学性质在于：对两个数组的中位数进行比大小，则整体的中位数一定不在具有较小中位数的数组的较小一半或者具有较大中位数的数组的较大一半，所以有一种方法可以基于这个性质进行多次二分查找锁定中位数，然而该方法仍不是最优。

给出两个数组如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 3 | 5 | 7 |  |
| 2 | 8 | 10 | 12 | 14 |

使用partition进行类二分查找法。首先根据两个数组长度可以得出中位数一定是两个数组中第k=(m+n)/2小或者第k和k+1小的数的平均。算法选择两数组中较短的那个进行类二分查找，另一个数组根据第一个数组的分位点确定自己的分位点。

选择分位点的规则是使得两个数组的较小的一半共含有(m+n+1)/2个元素，如此选择是因为这样一来最后的中位数（或者两个中位数中较小的那个，即k）将出现在左半边（较小的半边），最终只需要返回左半边两个数的最大值作为中位数（或者其一部分）即可。初始的分位点采用该数组的中位数即可。

每次确定分位点后，确定分位点处前后共四个数字，若分位点在数组头或尾，则将超出数组的数字定位正负无穷(MAX/MIN\_VALUE)。确定四个数字后，检查是否满足左边两个数字小于右边两个数字这一条件。若A数组左侧大于B数组右侧，则说明A数组左侧数字不应在较小的一半，分位点应该左移，此时移动数组右指针到分位点左侧，从该点开始寻找新分位点；若A数组右侧小于B数组左侧，则说明A数组右侧数字不应该在较大的一半，分位点应该右移，此时移动数组左指针到分位点右侧，从该点开始寻找新的分位点。

若条件全部满足，则返回左侧两数中的最大值（总共奇数个元素）或者左侧最大值和右侧最小值的平均值（总共偶数个元素）。

总结：这是一道思维相对直观，但是细节非常繁杂的题目，不愧为hard题，要思考的细节很多，值得反复玩味琢磨！

2023年9月24日星期日

1. [5] Longest Palindromic Substring

给一个字符串，返回该字符串的最大回文子字符串。

该题应形成一个思维定势：遇见回文就从重心开始向两头查找，逐一核对，此方法的时间复杂度要低于从两头向中心查找。

首先写一个辅助函数，返回以某点或某两点为中心向外查找的最大回文字符串长度。接着在主函数用一个数组记录当前最大回文子字符串的索引，然后分奇偶（即单个中心点或两个相同字符为中心）调用上述辅助函数，遇到比当前结果要更长的索引则更新结果。

注意，更新时的计算需要仔细斟酌一下，此处有一个边界条件需要代入具体例子尝试。

总结：这是一道比较基础的回文练习题，要记住这种范例，并且在思考的时候养成用实例辅助思考的习惯。

1. [11] Container With Most Water

给定一个数组，代表一系列容器壁，选择其中两个容器壁，其与x轴构成容器，返回容量最大的容器的容量。



这道题的题干比较抽象，但是解法相对简单直观。最naïve的方法是遍历所有可能的容器，计算容量并更新最大容量，但是该方法时间复杂度为O(n2)。

两指针方法则可以将时间复杂度降低到O(n)。在数组头尾各创建一个指针，每次更新时将短板对应的指针向中心移动，若更新后的总容量大于之前的最大容量，则更新总容量。之所以移动短板而非长板，是因为只有移动短板带来的收益才有可能超过因移动缩小了容器宽度带了的损失。移动长板对应的损失肯定更大。

总结：这道题是一个经典的**双指针题**。双指针是一个将双层循环转换为单层循环的好方法。双指针分为快慢指针和对撞指针（甚至包括上一题的中心向两遍的反对撞指针）两种方式。

双指针将一个两层循环转化成了一层循环，时间复杂度也从n^2变成了n，那么什么时候会需要使用双指针呢？

一般来讲，当遇到需要对一个数组进行重复遍历时，可以想到使用双指针法

如何实现将普通写法到双指针写法的转换呢？

说是指针，其实是设置两个int变量分别赋值为数组的首和尾，在一层循环内，每次只对其中一个指针进行移动，找到判断指针移动的条件是双指针的核心，在该题中，是一个求和为0的问题，那么当sum>0时，显然说明整体的值太大了，应该让其更小，故对于排序后的数组将尾指针左移，显然，sum和0的大小便是此题中判断指针移动的条件

1. [14] longest Common Prefix

给一个字符串列表，返回该列表的共同字符串前缀。

该题干是典型的Trie（字典树）的性质，但既然Java没有原生Trie实现，这道题也没有涉及Trie的其他性质，故不需要实现一个Trie类。

我写了一个函数返回两个字符串的相同前缀，但是该函数的返回占用了一定内存。标答给出的方法利用了Java String类的**indexOf**方法，取得了更低的内存占用。IndexOf方法接收一个子字符串，返回其在原字符串中第一次出现的索引，无则-1.**该题中由于我们要寻找的是前缀，故要检查方法返回值是否为0**。若否，则说明当前前缀无效，应使用substring函数缩短一位，直到前缀为空或有效。结果前缀应初始化为第一个字符串。

总结：该题的思维难度不大，但是对Java自有方法的熟悉程度有一定要求。

1. [15]3 Sum

给一个数组，返回其中所有三个元素之和为0的子数组，原数组可能有重复，每个元素只能用一次。

这道题是[1]Two Sum和[167]Two Sum II的变种，解题思路也是一脉相承，只需要将前者中的target从一个给定的数变成数组中的某一个数，再遍历整个数组即可，最终时间复杂度可以达到O(N2)。

[1]采用的是哈希集。首先将数组排序以剔除重复值，给定目标数，在数组中遍历其他数，则我们需要在哈希集中寻找目标数和当前遍历数之和的相反数。若该数不存在，则将当前所遍历之数添加进哈希集；若该数存在，则说明数组中曾出现该数，也就找到了一组解，将其返回并进行下一次遍历即可。

[167]采用的是双指针法（又是双指针）。将数组排序，给定目标数，将左指针指向目标数右侧，右指针指向数组末尾，若三数之和小于0，则说明数偏小，应将左指针右移；若大于0，说明偏大，右指针左移；若为0，则找到了一组解，将其返回并同时挪动左右指针。

注意，由于要处理重复值，故在每次找到结果后挪动指针时都需要判断之后的数字是否与当前数字重复。

大循环中只需要遍历所有非正数（正数及其之后的数相加必不为0），将其设为目标数，调用上述两个方法中的一个即可。

总结：举一反三很重要，哈希和双指针法是很重要的降时间复杂度的算法。此外，Java语法也要注意：List是一个抽象类，实例化时需要指定其继承类如ArrayList；Java Arrays类中有两个很好用的方法：sort给一个数组排序，asList将一个数组转换为List类（可以快速生成新List）。

1. [20] Valid Parentheses

给一个仅包含三种括号（“(),{},[]”）的字符串，确定该字符串中的括号是否闭合（即每个括号对应）。

这道题是基础的栈的应用。需要注意有很多边界条件的判断，如空栈时怎么办。

1. [19] Remove Nth Node From End Of List

给一个链表和数字n，删除链表倒数第n个节点。

这是基础的链表练习。可以使用单指针，通过循环一次链表得出总长度，再直接访问倒数第n个节点前一个节点进行更改地址删除的操作；也可以使用**快慢指针**，让前一个指针比后一个快n+1步（因为后一个指针指到null时才能判断并停下，故应快n+1），当后一个指针停下时前一个指针所在的位置刚好是倒数第n个节点的前一个节点。

总结：遇到链表一定要创建dummy节点，这样能少讨论很多corner cases；此处又是双指针的应用，非常巧妙。

2023年10月9日星期一

首先反思一下，最近这两周借着复习quiz的名义一直在打游戏，没有刷题，很懈怠！！

1. [17] Letter Combinations Of A Phone Number

给定最长4位的一组数，返回其对应的所有可能字符串。

这是一道很精彩的回溯递归练习题。由于最长只有四位数字，故可以采用比较暴力的方法进行解答。如果只有一位数字2，则答案是平凡解[“a”,”b”,”c”]，如果再增加一位，只需固定第一位字母，然后遍历第二位数字对应的所有字母，将其加在第一位字母之后即可，以此类推可以解决所有长度的问题。

在具体实现时，首先创建一个哈希图匹配每个数字和其对应的字母，再创建一个空字符串列表用于存储结果，最后创建一个字符串用于存储目标字符串。由于Java递归时无法通过传入参数地址的方法对列表进行修改，故上述三个数据结构应该定义在整个结构的最外层使其成为全局变量。

在创建具体函数实现函数时，首先检查是否为空字符串，接着直接调用递归函数并返回结果。递归函数的参数为当前索引以及一个StringBuilder，后者中存放着上一步得到的字符串。若当前字符串的长度和目标字符串长度相等，则说明该字符串已经完成遍历，将其加入列表并退出该次递归；否则遍历哈希图中当前索引位置的数字对应的字符，将其添加到当前字符串中，调用该递归函数并使索引+1，最后删除字符串中末位字符，即刚刚添加的字符，使得字符串回到遍历前的状态以进行下一次遍历。

1. [21] Merge Two Sorted Lists

给两个非降序排列的链表，将其合并为一个非降序排列的链表。

经典链表题，有递归和循环两种解法，其中循环法更节省内存。初始化一个head node存储结果，循环遍历两链表，将当前较小的链表的node加入head。需要注意的是一个链表已经空了以后需要直接将另一个链表后面的节点直接加到结果中，并且需要另外一个指针来循环结果链表。

1. [23] merge K Lists

给定K个升序排列的链表，将其合并为一个非降序排列的链表。

根据上一题两链表排序的方案，可以很轻松的想到将K个链表逐级两两排序的方法，其中显然逐次二分的方法比用第一个链表逐一同后续链表排序的方法的时间复杂度更低，因为前者调用logK次两链表排序而后者调用K次。

该题的难点在于如何实现二分调用两两排序函数。该题给出了一个两元素问题推广到多元素问题的一般思路。假设总共k个链表，编号为0，1…k-1，初始间隔为1，每次两两排序后的结果保存至靠前的链表中，则当间隔小于k时，循环系数为(i=0,i<k-interval,i+=interval\*2)，而每次排序时使用i和i+interval这两个链表进行排序，完成循环后interval翻倍。如此一来，便能够实现逐级两两排序直至剩下最后一个链表，该链表即为结果。

总结：遇到类似分而治之的方案都可以使用这种巧妙的思路解决。

1. [22] Generate Parentheses

给定数字n，确定n组括号所有可能的组合，输出一个字符串列表。

这是一道非常典型的动态规划/回溯题，都需要用到递归，思维难度不小！

思路一：使用DFS搜索所有可能的解，遇到不合理的解将其剪枝。与[17]类似，从剩余n个左括号，n个右括号开始，只要剩余的左括号不为0，则在可能的解中增加一个左括号，继续调用递归函数并使左索引-1，再删除该次操作添加的字符将其恢复到遍历前的状态。此外，剪枝的条件是**剩余的右括号数量大于左括号数量**，成立则说明右枝合法，添加右括号并调用递归函数，使得右索引-1并删除最后一个字符；否则不添加右括号。若当前括号数量为2n，说明这条路已经到底，将其添加到结果中即可。

思路二：很玄妙的递归式。考虑如何从n-1的所有结果中推出n：假设a，b分别为n-1组括号所有可能组合的两个子列，即a+b=Sn-1，a和b可能为空，且a和b完全对称所以顺序无所谓，那么Sn=（+a+）+b，即在a的外面加一对括号，就可以得到一种n个括号的可能组合，遍历所有可能的a和b，就可以得到Sn。虽然a和b对称，但是额外增加的括号仅存在于a外，故二者对称性被打破，如此得到的所有结果都是不重复的。

得到该递归式后，递归函数的实现则非常简单。创建一个从0到n-1的索引，将这些索引传入递归函数，遍历所有递归函数的输出，再将n-1到0的索引传入递归函数，遍历所有递归函数的输出，如此便得到了两个子列，将其和“（”、“）”结合即可。

总结：思路一思维逻辑更清晰，做题时更容易想到，且实现较为顺畅，只需要注意剪枝的条件即可；思路二很难想，我最开始就试图找出递归式的规律但是失败了，如果能推导出合理的递归式，思路二的代码非常简洁。

一旦涉及到递归问题，题目的思维将变得相对抽象，这是需要多借助例子辅助分析思考。

2023年10月10日星期二

1. [24] Swap Nodes in Pairs\

给定一个链表，将其内容两两交换，返回首个节点，要求不能改变每个节点的值。

这道题比较基础，只需要画个图弄清楚交换逻辑，将需要交换的几个节点提前存好，按照逻辑交换next即可。

1. [25] Reverse Nodes in K-Group

给定一个链表和一个数字k，在这个链表中一次交换k个节点，相当于上一题的拓展。

这道题真的非常tricky且繁琐！！！

首先，可以写出链表的反转函数：使用一个指针跟踪当前链表的头节点，另外一个指针向后遍历，每遍历一个节点就将其插到链表的开头作为新头节点，并调整前后节点的指针。微调该函数，增加一个整数参数k，使得这个反转操作仅进行k次就停止。

接下来在主函数中合理地调用上述修改过的反转函数。由于每次反转后被反转部分会与其后的部分的联系被打乱，所以需要有一个指针指向反转后的链表的末尾，另一个指向后一个部分的开头，在反转操作后将末尾的下一个指向开头，而且该操作需要判断当前是否为第一次反转。此外，还需要一个指针指向新链表的开头，也就是第一次反转后得到的头节点。然后，还需要一个节点实现计数功能，因为题目要求若剩下的节点不足k，则不反转，故该节点每向后走k步，触发一次反转操作。

总结：链表的反转函数很常用，思想是不停的将遍历到的节点插在开头。这道题的细节非常多，也很容易出错，需要跟踪的节点也很多，需要结合图像+打断点仔细调试。

2023年11月4日星期六

1. [26] remove Duplicates

给定一个非降序排列的数组，返回其中不重复的数字个数，要求在原数组修改，且原数组前k项必须为k个不重复项。

这道题是典型的双指针题，使用快慢指针。初见时思路差了一层，不应该想着交换重复项和后面的非重复项，而应该直接将非重复项覆盖重复项，如此一来前k项均为非重复项了。参考[这篇文章](https://programmercarl.com/0027.%E7%A7%BB%E9%99%A4%E5%85%83%E7%B4%A0.html#%E5%85%B6%E4%BB%96%E8%AF%AD%E8%A8%80%E7%89%88%E6%9C%AC)。

1. [31] Next Permutation

给定一个数组，找到其下一个变换。其中下一个变换的定义是该数组所有变换按字典序排列后的下个数组。

这道题需要仔细考虑“下一个”意味着什么，有点类似一道数学题，找不到破题点就完全没法做。

2024年1月12日星期五

1. [32]Longest Valid Parentheses

给定一组括号’(‘和‘)’，返回最长的完整组合的括号子串长度

这道题我用暴力解法写了出来，但是时间复杂度超了。其实较快的解法比暴力解法也就多绕了一圈。

这是一道hard题，思维难度大于代码难度。

**栈方法**：首先在一个空栈里面压入一个哨兵元素-1，其代表最早一个’)”所在的位置。压入-1是因为若第一个是左括号，第二个是右括号，则pop后应当把第一个括号也算上，所以不是0。

栈从前向后遍历，遇到左括号则压入其索引，遇到右括号则pop，如果pop后栈空了就压入当前右括号索引，若不空就更新最大长度，更新方式是当前索引减去栈顶值，即上一个没有被抵消的括号的索引。

该方法时间复杂度O（n），因为要扫描完整的字符串；空间复杂度O（n），最坏情况是一串左括号全部压入栈。

**左右指针法**：分别从左到右和从右到左扫描字符串，遇到左括号则left+1，遇到右括号则right+1，若运算后左右相等则更新最大长度为right\*2，若right大于left则二者均归零。

这个方法非常直观，如果遍历时括号正常组合，则left总是先于right且最终会相等，得到正确的长度；若出现多余左括号，则由于更新规则为right\*2而没有影响；若出现多余右括号，则二者清零后重新开始，前面的组合全部不算，字符串在此被斩断。

2024年9月4日星期三

[157] Read 4 Characters given Read4

给一个函数Read4(char[] buf4)，其将某个file读进buf4这个字符数组中并返回最终读取的字符数，要求基于这个函数实现一个read函数，其读取n位字符进字符数组buf中，并返回总共读取的字符数。

这道题的逻辑很简单，核心在于理解参数buf是读取结果的字符数组的地址，而字符来源文件并不可访问，故而可以直接调用buf中的信息来组成所需求的数组。

**双指针法系统训练：**

1. [977] 有序数组的平方

给一个非降数组，返回其各数平方的非降数组。要求O(N)

这题运用双指针法可以达到O(N)。最初的想法是找到中间最小的数字向两边遍历，比较其平方大小，从小到大填入目标数组。但实际上可以从两边开始向中间遍历，由尾至头地填入目标数组，便省去了找到最小数这一步。

1. [283] 移动零

给定一组数，把所有的0移到数组末尾且不改变其余数字的相对顺序

使用双指针法，每遇到一个0就交换其后第一个非0数字的位置就好

1. [844] Backspace String Compare

给两个字符串，其中#代表退格，在执行所有退格后比较两字符串是否相同，要求线性时间复杂度和常数空间复杂度。

这道题想到了使用倒序双指针法，但是解题思路还是有纰漏。这里的双指针不是对一个字符串使用双指针，而是两个字符串各一个指针，从两个字符串末尾向开头遍历，每遇到一个#，退格计数器加一，若退格计数器不为0则将其减一并跳过当前字符，直到退格计数器为0.

此时有几种情况，若两个指针此时均大于等于0，则说明此时两字符串尚不为空，比较两指针对应字符是否相等。若两指针只有一个不大于等于0，则说明只有一个为空，一定不相同，若两个均为空则说明都返回空字符串，相同。

该题不是常规的单数组双指针，而且需要考虑的条件很多，较为复杂。

**二分法系统训练**

1. [34] Find First and Last Position of Element in Sorted Array

给定一个可能有重复的非降数列，找到目标数在数组中的起止索引，若无返回[-1,-1]

这道题是一个很有意思的二分法变体题。其核心思想在于，不再单纯的寻找目标数，转而寻找目标数首次和末次出现的位置。为了避免找出所有目标数导致的近似线性运行时间，我们通过修改二分法使得仅寻找头尾。

二分法的修改也很直观，如果当前需要寻找头，而目前找到的这个目标函数满足

* 其位于当前子串的首位，或
* 其前一位不等于目标

则这就是头，返回即可，否则将搜寻范围缩小。寻找尾类似。这一操作类似于除了大小关系之外，又增加了一个头尾判断这一进一步二分的条件。

1. [69] Sqrt(x)

给定非负整数x，返回其正平方根向下取整的整数。

这道题非常直观，可以直接在2到x/2之间二分搜索，唯一tricky的地方在于判断最后输出的值，应该输出right因为在跳出while循环之前，while一定在Sqrt(x)的左边，是其向下取整。

1. [367] Valid Perfect Square

给定一个整数，如果其是完全平方数，返回true，否则false。

这道题比较简单，直接套用上一题的结构，只需要将返回值修改一下即可。

2024年1月19日星期五

1. [33] Search in Rotated Sorted Array

给定一个平移某一未知k单位的单增数组，在其中寻找给定目标数字。要求O(log N)复杂度

这道题非常有意思，可以思考的点很多，解题方法也很多。

首先为了满足复杂度要求，肯定采用二分法。**二分法对细节的要求非常高，等号和边界条件的处理可能导致完全不同的结果。**

这道题可以被拆分成两个问题，首先是寻找平移单位k，然后基于k对数组查找。而寻找k又可以被等价为寻找数组中的最小数，因为由于其单增的特性，数组的最小值所在处就是k值。

查找被平移的单增数列的最小值可以采用修改版二分法，参考[153]。观察目标数列如[4,5,6,7,0,1,2]，可以发现数列中点和数列队尾的大小比较指出了最小值所在的方位。中点较小则其在左半边，中点较大则在右半边。由此进入对应半边持续搜寻即可。**注意：这里的细节要求非常高，如究竟持续和队尾的数字比较，还是和新的right对应的数字比较；对应的left和right应该如何变？这些细节问题都需要写一个例子仔细推导。**

找到k后最直观的方法是对k的左右两边进行二分查找，因为其分别是单调数列。也可以运用平移值k创建一个平移后的二分查找。

当然，也可以利用目标值与数组首位的大小比较判断目标值在数组的哪一半，参考：

对于旋转数组 nums = [4,5,6,7,0,1,2]

首先根据 nums[0] 与 target 的关系判断 target 是在左段还是右段。

例如 target = 5, 目标值在左半段，因此在 [4, 5, 6, 7, inf, inf, inf] 这个有序数组里找就行了；

例如 target = 1, 目标值在右半段，因此在 [-inf, -inf, -inf, -inf, 0, 1, 2] 这个有序数组里找

附：[二分查找要确定区间的定义](https://programmercarl.com/0704.%E4%BA%8C%E5%88%86%E6%9F%A5%E6%89%BE.html#%E7%AE%97%E6%B3%95%E5%85%AC%E5%BC%80%E8%AF%BE)





**滑窗法系统训练**

1. [209] Minimum Size Subarray Sum

给定一个含有 n 个正整数的数组和一个正整数 s ，找出该数组中满足其和 ≥ s 的长度最小的连续子数组，并返回其长度。如果不存在符合条件的子数组，返回 0。

这道题是标准的滑窗法。其实滑窗法也可以看成一种双指针法，其中最重要的三要素是：

* 窗口是什么
* 如何移动窗口的结束位置（快指针）
* 如何移动窗口的起始位置（慢指针）

该题窗口为和 ≥ s 的长度最小的连续子数组，结束位置是遍历数组的指针，每当窗口内满足要求，就移动起始位置缩小窗口，并在每次找到符合要求的窗口时更新最小数组长度。

1. [76]Minimum Window Substring

给定两个字符串s和t，在s中寻找包含所有t中字符的最短子串，包括t的重复，不存在则返回空字符串。

该题的滑窗思想非常直观，即右端遍历s，逐窗口判断是否已经囊括t，再缩短左端寻找最小窗口。难点在于是否囊括t的判断算法的建立。此处要用到两个哈希表，一个哈希表包含t中的所有字符以及其出现的次数，另外动态维护一个哈希表，存放当前窗口的所有字符及其出现次数。每次增大窗口时，判断已成功匹配的字符数量是否与t所包含的字符数量相等，若相等，则进行缩小窗口的操作，并更新最小滑窗的长度。

最后，注意用滑窗的长度和起始索引来标记窗口位置，并用String.substring(start,end)方法提取窗口、用getOrDefault方法初始化窗口、并且在比较两个哈希表的值是否相等时要用equal来比较对象以避免整数对比的bug。

**螺旋矩阵系统训练**

[59] Spiral Matrix II

给一个正整数n，生成一个顺时针的螺旋矩阵

这道题的思路很简单，按照上右下左的顺序逐层生成即可，但是要注意每一层的判断标准要一致，也就是要统一地左闭右开（或左开右闭）。设置一个offset逐层递归进入，最后单独处理n为奇数时的中心块的值。

[54] Spiral Matrix

给一个mxn矩阵，顺时针返回其所有值

这题有一个相对简单的解法。

**链表系统训练**

[203] Remove Linked List Elements

给定链表头和整数val，删除链表中所有值为val的节点，返回新节点

这道题比较简单的方法就是设置一个dummy头在前面，然后while遍历检查val即可，写的时候要搞清楚目前在检查哪一个节点，别搞混了。

另外也可以用递归，若遇到需要当前头的值相等就创建一个新头并执行递归，否则直接递归

Backtracking系统训练：

Subsets

Subsets II

Permutations

Permutations II

Combinations

Combination Sum

Combination Sum III

Palindrome Partition

1. [39]Combination Sum

给定一个不重复数列和一个目标整数，返回可以组成目标数的所有数字组合，数字可以重复使用

标准的回溯法，有一系列相关习题。对所有可能的结果进行树状DFS，并舍弃不可能的解（超过目标数）。Java写这道题时有一些麻烦，由于其无法同时返回多个值，故需要在递归时传递结果数组，同时在添加新数字时要创建deep copy。

1. [216] Combination Sum III

给定目标数n和总数k，在1-9中用k个不同的数相加得到n，要求每个数用一次，每个组合仅返回一次

注意回溯法中返回到result的数列要创建deep copy，此外该题为了避免组合重复，需要在递归时额外传递一个整数，指示当前遍历到了1-9中的哪个数字

1. [40] Combination Sum III

给定一个可重复数列和一个目标整数，返回可以组成目标数的所有数字组合，数字不能重复使用，且组合不能重复

该题在上一题的基础上又要更复杂一些，为了正确地处理candidates中的重复数字（这是前两题中不存在的，前两题即使重复也是结果数列可能重复，而不是candidates数列中的重复），需要首先将candidates排序，再在上一题基础上传递当前数字索引的同时增加一个if语句，当遇到和我们前一个已经添加的数字相同的数字时跳过后续的步骤。

1. [78] Subsets

给定一组不重复数列，返回其所有可能的子集（幂集）

该题用回溯法时检测合规并加入结果列表的标准是当前combination的长度，即一个长为n的数列的所有子集可能的长度是0-n，在主函数中循环对所有长度调用回溯法递归函数即可。

1. [90] Subsets II

给一组可能重复的数列，返回所有可能的子集（每个数用一次，重复的数可以重复用）

这道题的核心和[40]相似，在于如何避免重复，需要注意的是，通过和前一个数比较来得出是否重复的方法的前提是数组是有序的，所以需要先排序

1. [46] Permutations

给一组不重复整数，返回所有可能的重排

这题回溯的返回条件是当前数组长度等于总数组长度，遍历时的核心是检查当前数字是否在当前数组中。

1. [47] Permutations II

给一组包含重复整数，返回所有可能的重排，每个数用一次

这题仅修改一个条件，难度增大不少。首先不能再使用当前数组是否包含当前数字来判断是否重复，因为可能有重复数字，故使用一个index数组记录已走过的数字索引，再根据index是否包含当前索引判断是否重复。其次，仅使用index判断也会导致出现重复数组，于是还需要将原数组排序，遇到重复数字时跳过，以避免重复。

**Meta专题训练**

[1249] Minimum Remove to make Valid Parentheses

给定一串包含“（”和“）“的小写英文字符，移除最小数量个左或者右括号使得这个字符串合规，即左右互相匹配。

该题思维难度不大，主要是熟悉python语法。注意：不要在for循环中使用O（N）操作，这样会超时，所以记录删除索引的数据结构要选择set而不是list，这样查找时间是O（1）。遍历字符串，遇到左括号就在stack记下索引，遇到右括号且stack为空就将其置入丢弃集，不为空就stack pop，最后把stack中剩余的内容和丢弃集做并集后重新遍历丢掉。

Python中enumerate方法返回index，内容（character）。

Union方法可以将set和另一个可迭代对象中的内容合并

“a”.join（s）方法可以将s以a为分隔符组成字符串，如””.join(string\_builder)

[408] Valid Word Abbreviation

给定一个字符串和一个缩写，返回二者是否相符合。如word = "internationalization", abbr = "i12iz4n"。

这道题的逻辑非常简单，采用双指针一个指向原词，另一个指向缩写，遇到数字就往后挪原词指针就好。主要用来熟悉python语法，字符串可以直接用+拼接，然后再用int（）将其转换为整数即可。

[680] Valid Palindrome II

给定一个字符串，检查能否在删除至多一个字符后形成回文。

这题相对也比较简单，写一个辅助函数检查回文字符串，然后在主函数中遇到头尾字符不相同的情况时尝试删除头或尾的字符（用辅助函数检查跳过对应字符后是否回文）。

若需要删除n个字符检查是否形成回文，可以创建一个计数器加入递归辅助函数中。

[215] Kth Largest Element in an Array

返回数组中第k大的数。数组可能重复，第k大是指排序后位于k位置的数。

这道题是一个比较基础但是锻炼思考能力的题目。最naïve的方法是排序，但是面试肯定不会只让排序。

1. 用min Heap，heap大小为k，每次超出k后就pop堆顶的数，维持heap中一直是k个较大的数，遍历所有数之后位于堆顶的就是第k大的数。
2. Count sort，适用于最大和最小值差距不大的数组。遍历所有数找到最大最小数，创建一个[Min,Max]的数列，然后再次遍历数组，为遇到的每个数字计数，最后根据数字的计数排序或从后往前数k个数返回。
3. Quick select：平均用时最快的方法，专门用于找第k大或小的数。和quick sort思想类似，随机选取一个pivot，将所有数分为比pivot大和比pivot小的两组，根据每组长度和k的关系只处理对应组，这样递归地处理整个数组。本题中若较大组长度大于k则说明要找的数落在了较大组，否则落在了较小组。



[314] Binary Tree Vertical Order Traversal

给定一个二叉树，返回其垂直顺序遍历。

这道题没见过真的很难想，基本只能记住答案。二叉树中节点有一个潜在的坐标，如果以根节点为（0，0），则左子节点横坐标-1，右+1；每向下一层纵坐标+1。如此一来，创建一个哈希表，其键为列索引，值为该列从上至下的节点中所保存的值。由于采用BFS逐层递归遍历树，故纵坐标从上至下的顺序自动满足。如此一来，只需要创建一个双端队列，对树进行BFS，将遇到的节点何其对应的列索引放进哈希表，再将其子节点更新进双端队列即可。

有一些比较tricky的python语法，第一个是使用defaultdict类创建哈希表，如此可以省去为空节点赋空列表这一步。其次是要用popleft这一deque特有的方法才能实现队列的LIFO特性。最后就是在return时为了保持横坐标从小到大地输出且避免排序，需要在遍历时记录记录列索引的范围。

[1650] Lowest Common Ancestor of a Binary Tree III

给定一个二叉树的两节点，返回他们公共最小祖先。所有节点保存的值不同。

这题思维不难，利用保存的parent节点从一个节点出发回溯到根节点，将其存到哈希表中。由于其值都不同，将节点值作为键，节点地址作为值。再从另一个节点出发开始回溯，并不停地检查值是否在哈希表出现过，直到回溯到某个共享节点，则该节点就是公共最小祖先。注意，由于认为自己是自己的一个祖先，所以要把这两个节点最先加入哈希表。

[236] Lowest Common Ancestor of a Binary Tree

给定一个二叉树的两节点，返回他们公共最小祖先。所有节点保存的值不同。这次节点中没有父节点信息。

这题比上一题要复杂一些，因为没有父节点信息，无法直接从pq开始回溯。

方法一思维较为直接，但是实现略tricky。对树做DFS，若到达叶节点还是没遇到pq则沿途返回false，反之若遇到了则沿途返回true。若有某节点左右均返回true，则该节点就是LCA。

实现时若没遇到pq，则返回None，所以若左右recursion都返回非none值，则说明当前节点是LCA，否则直接返回left或right。

方法二思维和实现都比较复杂，但是更快，可以遇到答案立刻返回，不需要遍历其他节点。面试时可能需要回答这个方法。用一个stack记录遇到的节点以及当前节点的遍历情况，是左右都未遍历还是遍历了左还是都已遍历。再用一个flag记录目前是否已经找到一个目标节点。此外，用LCAindex标记当前LCA索引，当找到一个p或者q时，将LCA index更新为stack末尾，当两个都找到时，直接返回LCAindex对应节点。

[227] Basic Calculator II

给定一个代表算式的字符串，完成计算，仅包含加减乘除

这道题乍一看很好理解，但是思维细节很复杂，非常tricky，基本需要记住答案，不记住的话面试遇到了基本想不出来。为了达到最好的时间和空间复杂度，不能用stack记录加减法的数值，而是要用一个整数记录当前的计算结果，还需要记录上一个运算符号以判断当前计算的优先级。

此外，为了正确处理最后一个数字，需要在字符串末尾加一个“+”，而此处需要用python内置包 itertools.chain而非直接+，因为这样会直接生成一个迭代器，从而只需要O（1）时间而非生成一个新的字符串将二者复制进去，导致O（N）。

整体思路是，迭代整个字符串，遇到数字就更新当前数字num，遇到符号就用curr\_res暂存结果，若前一个算符是加减，则将其更新到最终结果，否则不更新，最后重置当前算符和数字num，进入新的循环。

[339] Nested List Weight Sum

给定一个整数嵌套列表，返回每个数和其深度的乘积和。

这道题有DFS和BFS两个思路，都比较好想。DFS就遇到list就进去计算一个结果返回即可，BFS就创建一个deque，遇到列表就把它加到列表最前面，像剥洋葱一样一层层进到最内侧列表即可。

注意python语法，要用题目定义的getInter()和getList()才能正确识别，否则type无法对应。

[88] Merge Sorted Array

给两个非降数组，长度分别m，n，将他们按非降顺序合并到num1中，num1总长度m+n，末尾都是0.

这道题之前刷过，这次是第二次遇到，基本思路还能想起来，从所有数组的末尾开始，用三个指针分别跟踪两个数组和num1的末尾，将每次遇到的较大值插入num1的末尾即可，注意边界条件和退出条件即可。

[71] Simplify Path

给一个Unix风格路径，将其转换为简化规范路径。

这道题乍一看比较复杂，需要分类讨论很多，但其实很简单，只需要创建一个stack记录所有遇到的文件夹或文件名，用split函数将初始字符串以”/”分割并遍历，若遇到”..”且stack非空则pop，否则若不是”.”、“..”或空字符串，则将其加入stack。最后使用之前用过的join方法，以”/”作为分隔符将stack中所有内容连接起来，再在最前面在一个斜杠即可。

[938] Range Sum of BST

给一个BST根节点和两个整数low和high，返回[low, high]范围内所有节点值的和。

很简单的DFS题，唯一问题是递归的时候注意排除掉范围外的子节点，否则会超时。

[1091] Shortest Path in Binary Matrix

给一个nxn二元矩阵，返回其从左上到右下的最短路长度，无则返回-1.其中0是通路，1是障碍，8向。

这道题乍一看可以用DP，但其实不行，因为DP无法判断向上找寻的路径。该题是标准的BFS最短路问题。最直观的思路就是直接修改原矩阵，将所有遍历过的位置都改为1，并创建一个新的cell类或使用数组跟踪当前行列和当前长度；或直接将遍历过的位置都改为当前长度，这样仅需跟踪当前行列。需要注意，BFS使用的FIFO的queue结构，由于不涉及向queue中间增删，故linkedlist和ArrayDeque实现都可以。还要注意如何遍历每个节点的所有邻居，用两层for循环从-1到1作为行列的偏移量是最简单的方法。当遇到为遍历过的节点时将其加入queue并标记已访问即可。

若要进一步避免修改原矩阵，只需要额外增加一个二维矩阵跟踪已访问过的节点，并在判断是否添加新节点时进行额外的判断即可，仅需略微修改上面的代码。

[199] Binary Tree Right Side View

给一个二叉树，返回其右视图，从顶到底。

这道题比较直观的方法是BFS，用某种方式跟踪当前层的节点，以返回最右边的节点。可以用双队列，将子节点都加入下一层对应的队列中，也可以用哨兵节点标记每一层的结束，还可以跟踪每一层的节点个数。或者用DFS，每次优先遍历右子节点并跟踪深度。

[162] Find Peak Element

给一个整数数组，找到它其中的peak元素，即严格比邻居大的元素，返回任意一个即可。

这道题题干花里胡哨的，但其实就是最简单的二分法找最大即可。

2024年11月17日星期日

[528] Random Pick With Weight

给一个整数数组，对其加权随机采样，概率是对应值占数列所有值的和的比例。实现这个函数。

这道题的描述非常抽象，但是可以转化为求前缀和数组和二分查找问题。根据数组创建一个前缀和数组，然后生成0到totalsum的随机数，使用二分法判断落在哪个区间，返回区间对应的索引即可。

[1570] Dot Product of Two Sparse Vectors.

计算两个稀疏向量的点积。

这题我直接用hash做出来了，但是follow up问如果只有一个向量稀疏怎么办？另外meta有可能问如果hash在大规模数据时效率太低怎么办？

若我们已知哪个稀疏则很简单，若不知道也可以直接比较两个压缩后向量的长度，用短的匹配长的。如果希望避免hash，则需要用List<int[]>把键值对存在List中，且查找时用二分查找，在长向量中查询短向量对应索引是否存在。

[125] Valid Palindrome

给定一字符串，检查其是否回文，回文定义是仅看字母和数字（alphanumeric）是否前后对称。无视大小写。

这题有两个细节，一是alphanumeric，二是需要转成小写字母再比较。注意，创建新的字符串不仅浪费时间，还会超时。最简单的方案是构建一个for循环，用Character.isLetterOrDigit方法检查是否符合要求，跳过不符的字符。注意跳过时的条件检查。

[50] Pow(x,n)

计算x^n

这道题数学思维比较多，主要在于高效地拆分幂函数，如果每次将其分解为或者，遇到n小于0直接返回倒数，则可以将运行时间从N降至log（N）。想明白这一层，实现代码很简单。

[1004] Max Consecutive Ones III

给一个二元数组和整数k，至多反转k个0，返回最多多少个连续的1。

一遍过。滑窗法，右端点for循环，跟踪当前最大长度和窗口内数字和，当窗口长度减去窗口数字和大于k时，左端点前进。注意移动窗口时更新数字和即可。

[426] Convert Binary Search Tree to Sorted Doubly Linked List

In place地将一个BST转换为双端链表。

这道题直观上很容易想到用DFS in order，但是要想写出in place的代码比较麻烦。若能够创建新数据结构，则可以使用一个List将遍历的节点按顺序放入，再逐个修改其子节点即可。

最优的方法是跟踪当前双端链表的头尾，将其初始化为null，直接中序遍历，若头null则说明我们达到了整个BST的最小值，即链表头，将其设置为头，否则将当前节点与链表尾相互连接。接着将当前节点标记为链表尾，进入右子节点循环。在DFS全部执行完毕后，在主函数再将头尾节点链接。

2024年11月18日星期一

[543] Diameter of Binary Tree

返回一个二叉树的直径。

这道题可以转换为寻找左右子树深度和最大的节点，如此一来便可以用DFS，每次递归时更新当前直径和左右深度和的更大者，递归返回值是左右深度中较大者加一。

[129] Sum Root to Leaf Numbers

给一个仅包含0-9的二叉树，返回其从根到叶组成的数字之和。

[二叉树遍历方法](https://programmercarl.com/%E4%BA%8C%E5%8F%89%E6%A0%91%E7%9A%84%E9%80%92%E5%BD%92%E9%81%8D%E5%8E%86.html#%E7%AE%97%E6%B3%95%E5%85%AC%E5%BC%80%E8%AF%BE)。注意递归函数返回值问题：**子树问题返回int，遍历问题返回void。**

面试时最好将迭代法步骤记个笔记贴在旁边。

这道题本身并不难，前序遍历，每次更新当前数字，遇到叶节点就更新总和即可。

此外，还需要了解一个Morris遍历法。

Morris遍历优点：

不需要递归或额外栈，空间效率高。

时间复杂度为 O(n)。

缺点：

代码实现较复杂。

对树结构进行了临时修改，可能不适合多线程环境或对树的完整性要求较高的场景。

[146] LRU Cache

实现Least Recently Used cache. 要求get和put都是O（1）时间。

LRU的核心是维护一个双端队列，队尾是最近使用节点，队首是最久没使用节点。若使用Java内置Deque类，其remove函数是O(N)时间复杂度，超出了要求，故需要自己实现一个双端队列，该双端队列记录队首队尾两个dummy node，实现O1的添加和删除操作。此外，哈希表维持key-Node这一键值对，并在Node中记录value，这样从哈希表删除某节点是O1时间，且基于该节点直接将其从Deque删除也是O1时间。需要注意，初始化Deque的时候要将两个dummy相互连接，否则会有空指针报错。

2024年11月19日星期二

[31] next Permutation

给定一个数组，找到其下一个变换。其中下一个变换的定义是该数组所有变换按字典序排列后的下个数组。

之前做过一次，但是没什么印象了。这道题的核心是观察到“下一个变换”意味着什么。

若数组单减，则下一个变换一定是反转整个数组。若数组从后往前看增长，但在某处下降，即a[i-1]<a[i]，则a[i-1]就是需要交换的值。重新从队尾向前遍历，将a[i-1]与遇到的第一个大于a[i-1]的数交换，再将从a[i]到队尾的子数组翻转，即可得到新的反转数组。

**这道题第一次遇到比较难想，需要记住。**

[]