### 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。

**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

**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)



