作者 程序员2点零 (/users/0e6b059ad57e) 2016.04.22 12:49*
写了78193字，被58人关注，获得了59个喜欢
(/users/0e6b059ad57e)

LeetCode LintCode ReviewPage 复习总结页

程序员2.0与朋友们

这份是复习总结的页面。在复习的时候，习惯用中文把想法写一下，看起来速度快，对咱们中国人肯定是顺许多的。这篇总结除去了题目，解答，而只保留了想法和Github的link，方便快速浏览。

当然，并不是所有题目都更新完。为了做这篇东西，我在github上的 .java file 都要遵守一些特定的格式，才能自动生成这个页面。先看起来吧。 程序员2.0的GitHub (https://github.com/shawnfan). 如果有谁想contribute, 就发个request啦。

关于具体的题目list, 可以参考这个页面LeetCode LintCode 干货全集 (http://www.jianshu.com/p/c458198aff5f)

Review Page

This page summarize the solutions of all problems. For thoughts,ideas written in English, refer to deach individual solution.

New problems will be automatically updated once added.

0. 2 Sum II - Input array is sorted.java (https://github.com/shawnfan/LintCode/blob/master/Java/2%20Sum%20II%20-%20Input%20array%20is%20sorted.java) Level: Medium

排序好的array. Binary Search移动start和end，核查sum。

---

1. 2 Sum II.java (https://github.com/shawnfan/LintCode/blob/master/Java/2%20Sum%20II.java) Level: Medium

LintCode的题. 注意找的是greater/bigger than target。

由于给定条件允许O(nLogn):
sort
two pointer

while里面two pointer移动。每次如果num[left]+num[right] > target，那么其中所有num[left++]的加上num[right]都>target.
也就是,num[right]不动，计算加入挪动left能有多少组，那就是: right-left这么多。 全部加到count上去。
然后right--.换个right去和前面的left部分作比较。

---

2. 2 Sum.java (https://github.com/shawnfan/LintCode/blob/master/Java/2%20Sum.java) Level: Medium

解法1：相对暴力简洁, HashMap<value, index>，找到一个value, 存一个; 若在HashMap里面 match 到结果, 就return HashMap里存的index. O(n) space && time.

解法2：Sort array, two pointer 前后++,--搜索。Sort 用时O(nlogn).

 1. 第一步 two pointer 找 value.
 2. 注意，要利用额外的空间保留original array， 用来时候找index. (此处不能用HashMap，因为以value 为key，但value可能重复)
    O(n) space, O(nlogn) time.

---

3. 3 Sum Closest.java (https://github.com/shawnfan/LintCode/blob/master/Java/3%20Sum%20Closest.java) Level: Medium

3Sum 的一种简单形式, 并且都没有找index, value, 而只是找个sum罢了.

double for loop。 2Sum只能用土办法 left/right 2 pointers。 O(n^2)

注意：check closest时候用long, 以免int不够用

4. 3 Sum.java (https://github.com/shawnfan/LintCode/blob/master/Java/3%20Sum.java) Level: Medium

用个for loop 加上 2sum 的土办法。

注意:

1. 找 value triplets, 多个结果。注意，并非找index。
2. 要升序, 第一层for loop 从最后一个元素挑起, 保证了顺序。
3. 去掉duplicate: check用过的同样的数字，都跳掉。不需要用同样的数字再计算一边已有结果。

步骤:

1. For loop 挑个数字A.
2. 2Sum 出一堆2个数字的结果
3. Cross match 步骤1里面的A.

时间 O(n^2), 两个nested loop

另外, 还是可以用HashMap来做2Sum。稍微短点。还是要注意handle duplicates.

再另外（leetcode做时写的）：先sort，然后two pointer。

---

5. 3Sum Smaller.java (https://github.com/shawnfan/LintCode/blob/master/Java/3Sum%20Smaller.java)发现j,k满足条件时候，(k – j)就是所有 sum <target的情况了。 而一旦>target, 又因为j不能后退，只能k--，那么问题就被锁定了. 这样可以做到O(n2)

---

6. 4 Sum.java (https://github.com/shawnfan/LintCode/blob/master/Java/4%20Sum.java) Level: Medium

方法1： 3Sum外面再加一层. 参考3Sum. 时间O(n^3)。 但此方法在k-sum时候，无疑过于费时间. O(n^k)

方法2： 参见 http://lifexplorer.me/leetcode-3sum-4sum-and-k-sum/ (http://lifexplorer.me/leetcode-3sum-4sum-and-k-sum/)

1. 利用2Sum的原理，把4Sum分为连个2Sum。左一个pair,右一个pair，每个pair里面放2个数字。
2. 以一个点，i，作为分界口，也要列举出所有i之前的pair,作为基础。
3. 再尝试从所有i+1后面,找合适的2nd pair。

注意：在造class Pair时候，要做@override的function: hashCode(), equals(Object d). 平时不太想得起来用。

---

7. A+B.java (https://github.com/shawnfan/LintCode/blob/master/Java/A+B.java) Level: Easy

^ 是不完全加法. 每次都忽略了进位。而 & 刚好可以算出需要的所有进位。

那么就，首先记录好进位的数字：carry. 然后 a^b 不完全加法一次。然后b用来放剩下的carry, 每次移动一位，继续加，知道b循环为0为止。

Bit Operation
Steps:
a & b: 每bit可能出得余数
a ^ b: 每bit在此次操作可能留下的值，XOR 操作
每次左移余数1位，然后存到b, 再去跟a做第一步。loop until b == 0

(http://www.meetqun.com/thread-6580-1-1.html (http://www.meetqun.com/thread-6580-1-1.html))

---

8. Add and Search Word.java (https://github.com/shawnfan/LintCode/blob/master/Java/Add%20and%20Search%20Word.java) Level: Medium

Trie结构, prefix tree的变形： '.'可以代替任何字符，那么就要iterate这个node所有的children.

节点里面有char, isEnd, HashMap<Character, TrieNode>
Build trie = Insert word:没node就加，有node就移动。
Search word:没有node就报错. 到结尾return true

这题因为'.'可以代替任何possible的字符，没一种都是一个新的path，所以recursive做比较好些。
(iterative就要queue了,麻烦点)

---

9. Add Binary.java (https://github.com/shawnfan/LintCode/blob/master/Java/Add%20Binary.java) Level: Easy

方法一:土办法没技术，把binary换成数字，加起来，再换成binary。如果input很大，那么很可能int,long都hold不住。不保险。

方法二:一般方法，string化为charArray,然后逐位加起，最后记得处理多余的一个carry on

---

10. Add Two Numbers II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Add%20Two%20Numbers%20II.java) Level: Medium

LinkedList并没有反过来，那么自己反：
方向相反。巧用stack.

做加法都一样：

1. carrier
2. carrier = (rst + carrier) / 10
3. rst = (rst + carrier) % 10

---

11. Add Two Numbers.java (https://github.com/shawnfan/LintCode/blob/master/Java/Add%20Two%20Numbers.java) Level: Easy

LinkedList都已经反转好了，直接做。

遍历两个l1,l2把carry-on处理好，每次生成一个新node，最后检查carry-on。

跟Add Binary的理解方式一模一样。

---

12. Alien Dictionary.java (https://github.com/shawnfan/LintCode/blob/master/Java/Alien%20Dictionary.java) Level: Hard

Not Done yet。 Topological sort.

---

13. Anagrams.java (https://github.com/shawnfan/LintCode/blob/master/Java/Anagrams.java) Level: Medium

1. HashMap 的做法. sort每个string, 存进HashMap, 重复的就是anagrams,最后输出。
   toCharArray
   Arrays.sort
   Stirng.valueOf(char[])
   时间nLO(logL),L是最长string的长度。

2. Arrays.toString(arr)的做法。arr是int[26], assuming only have 26 lowercase letters.
   Count occurrance, 然后convert to String，作为map的key.
   Time complexity: nO(L)

3. 另一种做法：http://www.jiuzhang.com/solutions/anagrams/ (http://www.jiuzhang.com/solutions/anagrams/)

   1. take each string, count the occurrance of the 26 letters. save in int[]count.
   2. hash the int[] count and output a unique hash value.
      hash = hash a + num
      a = a b.

3. save to hashmap in the same way as we do.

这一步把for s: strs 里面的时间复杂度降到了O(L). L = s.length().
Need to work on the getHash() function.

时间变成n*O(L). Better.

---

14. Backpack II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Backpack%20II.java) Level: Medium

做了Backpack I, 这个就如出一辙。
想法还是，选了A[i-1] 或者没选A[i].
一路往前跑不回头。就出来了。
其实这个Backpack II 还更容易看懂代码。

O(m)的做法:
想想，的确我们只care 最后一行，所以一个存value的就够了。
注意：和bakcpackI的 O(m)一样的，j是倒序的。如果没有更好的j，就不要更新。就是这个道理。

---

15. Backpack.java (https://github.com/shawnfan/LintCode/blob/master/Java/Backpack.java) Level: Medium

DP。
row是item大小: 0, A[0], A[1] ... A[A.length -1]
col是背包累积的size: 0, 1, 2, ... m.

想法：
dp[i][j]有这么i-1个item, 用他们可否组成size为j的背包？true/false.（反过来考虑了，不是想是否超过size j，而是考虑是否能拼出exact size == j）。
注意注意：虽然dp里面一直存在i的位置，实际上考虑的是在i位置的时候，看前i-1个item.

看一遍code，会发现：

1. picked A[i-1]: 如果上一个item, A[i-1],被加了上来, 用j-A[i-1]看看，是否这再前一步也true. true就好啦。
2. did not pick A[i-1]: 那就是说，不加上A[i-1], 上一行d[i-1][j]还是需要是true。

最后：
跑一边dp 最下面一个row. 从末尾开始找，最末尾的一个j (能让dp[i][j] == true)的，就是最多能装的大小 :)

时间，空间都是：O(mn)

再有：

O(m)时间的做法，具体看solution. 注意j是倒序的啊！

依然是O(mn)的空间

---

16. Balanced Binary Tree.java (https://github.com/shawnfan/LintCode/blob/master/Java/Balanced%20Binary%20Tree.java) Level: Medium

1. DFS using depth marker: 每个depth都存一下。然后如果有不符合条件的，存为-1.

   一旦有-1，就全部返回。

   最后比较返回结果是不是-1. 是-1，那就false.

   Traverse 整个tree, O(n)

2. Only calculate depth using maxDepth function. Same concept as in 1, but cost more traversal efforts.

---

17. Best Time to Buy and Sell Stock I.java (https://github.com/shawnfan/LintCode/blob/master/Java/Best%20Time%20to%20Buy%20and%20Sell%20Stock%20I.java) Level: Medium

理解意思是关键：

每天都就交易价格，n天只让买卖一次，那就找个最低价买进，找个最高价卖出。

记录每天最小值Min是多少。O(n)

每天都算和当下的Min买卖，profit最大多少。

---

18. Best Time to Buy and Sell Stock II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Best%20Time%20to%20Buy%20and%20Sell%20Stock%20II.java) Level: Medium

和Stock I 的区别：可以买卖多次，求总和的最大盈利。

找涨幅最大的区间，买卖：

找到低谷，买进:peek = start + 1 时候，就是每次往前走一步;若没有上涨趋势，继续往低谷前进。

涨到峰顶，卖出:一旦有上涨趋势，进一个while loop，涨到底, 再加个profit.

中间的：

profit += prices[peek - 1] - prices[start]; 听特别的。

当没有上涨趋势时候，peek-1也就是start, 所以这里刚好profit += 0.

O(n)

19. Best Time to Buy and Sell Stock III .java (https://github.com/shawnfan/LintCode/blob/master/Java/Best%20Time%20to%20Buy%20and%20Sell%20Stock%20III%20.java)
Level: Medium

比stock II 多了一个限制：只有2次卖出机会。也就是：找峰头；然后往下再找一个峰头。

怎么样在才能Optimize两次巅峰呢？

从两边同时开始找Max！（棒棒的想法）

leftProfit是从左往右，每个i点上的最大Profit。
rightProfit是从i点开始到结尾，每个点上的最大profit.
那么在i点上，就是leftProfit，和右边rightProfit的分割点。在i点，leftProfit+rightProfit相加，找最大值。

三个O(n),还是O(n)

---

20. Best Time to Buy and Sell Stock IV.java (https://github.com/shawnfan/LintCode/blob/master/Java/Best%20Time%20to%20Buy%20and%20Sell%20Stock%20IV.java)
Level: Hard

记得要理解： 为什么 i-1天的卖了又买，可以和第 i 天的卖合成一次交易？
因为每天交易的price是定的。所以卖了又买，等于没卖！这就是可以合并的原因。要对价格敏感啊少年。

Inspired from here:
http://liangjiabin.com/blog/2015/04/leetcode-best-time-to-buy-and-sell-stock.html   (http://liangjiabin.com/blog/2015/04/leetcode-best-time-to-buy-and-sell-stock.html)

局部最优解 vs. 全局最优解：
local[i][j] = max(global[i – 1][j – 1] + diff, local[i – 1][j] + diff)
global[i][j] = max(global[i – 1][j], local[i][j])

local[i][j]: 第i天，当天一定进行第j次交易的profit
global[i][j]: 第i天，总共进行了j次交易的profit.

local[i][j]和global[i][j]的区别是：local[i][j]意味着在第i天一定有交易（卖出）发生。
当第i天的价格高于第i-1天（即diff > 0）时，那么可以把这次交易（第i-1天买入第i天卖出）跟第i-1天的交易（卖出）合并为一次交易，即local[i][j]=local[i-1][j]+diff；
当第i天的价格不高于第i-1天（即diff<=0）时，那么local[i][j]=global[i-1][j-1]+diff，而由于diff<=0，所以可写成local[i][j]=global[i-1][j-1]。
(Note:在我下面这个solution里面没有省去 +diff）

global[i][j]就是我们所求的前i天最多进行k次交易的最大收益，可分为两种情况：

如果第i天没有交易（卖出），那么global[i][j]=global[i-1][j]；

如果第i天有交易（卖出），那么global[i][j]=local[i][j]。

---

21. Binary Representation.java (https://github.com/shawnfan/LintCode/blob/master/Java/Binary%20Representation.java) Level: Hard

首先要分两半解决，断点是'.': str.split("\.");

Integer那一半好弄，whie loop里： num%2, num/2。

Decimal那边复杂点.
bit == 1的数学条件：当下num 2 >= 1。 更新: num = num 2 - 1;
bit == 0的数学条件： num 2 < 1. 更新: num = num 2

注意：num是 double, 小数在（num = num * 2 -1）的公式下可能无限循环. 因此check: num重复性，以及binary code < 32 bit.

(所以题目也才有了32BIT的要求！)

---

22. Binary Search Tree Iterator.java (https://github.com/shawnfan/LintCode/blob/master/Java/Binary%20Search%20Tree%20Iterator.java) Level: Hard

用O(h)空间的做法：

理解binary search tree inorder traversal的规律：
先找left.left.left ....left 到底，这里是加进stack.
然后考虑parent,然后再right.

例如这题：
stack里面top，也就是tree最左下角的node先考虑,取名rst.
其实这个rst拿出来以后, 它也同时是最底层left null的parent，算考虑过了最底层的parent。
最后就考虑最底层的parent.right, 也就是rst.right.

注意:
next()其实有个while loop, 很可能是O(h).题目要求average O(1),所以也是okay的.

用O(1)空间的做法：不存stack, 时刻update current为最小值。

找下一个最小值,如果current有right child：
和用stack时的iteration类似,那么再找一遍current.right的left-most child,就是最小值了。

如果current没有right child：

那么就要找current node的右上parent, search in BinarySearchTree from root.

注意：

一定要确保找到的parent满足parent.left == current.

反而言之，如果current是parent的 right child, 那么下一轮就会重新process parent。

但是有错:binary search tree里面parent是小于right child的，也就是在之前一步肯定visit过，如此便会死循环。

---

23. Binary Tree Inorder Traversal.java (https://github.com/shawnfan/LintCode/blob/master/Java/Binary%20Tree%20Inorder%20Traversal.java) Level: Easy

法一:

Recursive: Divide and Conquer, with helper(dfs) method

法二:

Stack:

Add left nodes all the way

Print curr

Move to right, add right if possible.

注意stack.pop()在加完left-most child 的后，一定要curr = curr.right.

若不右移，很可能发生窘境：

curr下一轮还是去找自己的left-most child，不断重复curr and curr.left, 会infinite loop, 永远在左边上下上下。

---

24. Binary Tree Level Order Traversal II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Binary%20Tree%20Level%20Order%20Traversal%20II.java) Level: Medium

普通BFS，用一个queue，加上一个queue.size()来交替换行.

rst里面add(0,...)每次都add在list开头

---

25. Binary Tree Level Order Traversal.java (https://github.com/shawnfan/LintCode/blob/master/Java/Binary%20Tree%20Level%20Order%20Traversal.java) Level: Medium

方法1. 最普通,Non-recursive: BFS, queue, 用个queue.size()来end for loop:换行。

或者用两个queue. 当常规queue empty，把backup queue贴上去。

方法2. Recursive with dfs：

每个level都应该有个ArrayList. 那么用一个int level来查看：是否每一层都有了相应的ArrayList。

如果没有，就加上一层。

之后每次都通过DFS在相应的level上面加数字。

---

26. Binary Tree Longest Consecutive Sequence.java (https://github.com/shawnfan/LintCode/blob/master/Java/Binary%20Tree%20Longest%20Consecutive%20Sequence.java) Level: Medium

屌炸天的4行代码。Divide and Conquer

主要想法：

Recursive用好。首先在这个level比一比，可否成。

不成的话，另立门户, count = 1。

然后左右开弓。再把结果拿过来比较一下就好了。

---

27. Binary Tree Maximum Path Sum II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Binary%20Tree%20Maximum%20Path%20Sum%20II.java) Level: Medium

比Binary Tree Maximum Path Sum I 简单许多. 因为条件给的更多：at least 1 node + have to start from root => have to have root.

方法1：

维持一个global或者recursive里的sum。traversal entire tree via DFS. 简单明了。

方法2：

Single path: either left or right.

If the path sum < 0, just skip it.

---

28. Binary Tree Maximum Path Sum.java (https://github.com/shawnfan/LintCode/blob/master/Java/Binary%20Tree%20Maximum%20Path%20Sum.java) Level: Medium

第一次做有点难理解，复杂原因是：因为可能有负值啊。不能乱assume正数。

single path max 的计算是为了给后面的comboMax用的。

如果single path max小于0，那没有什么加到parent上面的意义，所以就被再次刷为0.

combo的三种情况：(root可能小于0)

1. 只有left
    2. 只有右边

2. root大于0，那么就left,right,curr全部加起来。

情况1和情况2取一个最大值，然后和情况三比较。做了两个Math.max(). 然后就有了这一层的comboMax

12.11.2015 recap:
So totally, 5 conditions:
(save in single)
left + curr.val OR right + curr.val
(save in combo:)
left, right, OR left + curr.val + right

---

29. Binary Tree Path Sum.java (https://github.com/shawnfan/LintCode/blob/master/Java/Binary%20Tree%20Path%20Sum.java) Level: Easy

Binary Tree的一个基本题。
遍历到底，比较sum vs. target。
注意divide的情况。要把遍历的例子写写。

LeetCode: Path Sum II

---

30. Binary Tree Paths.java (https://github.com/shawnfan/LintCode/blob/master/Java/Binary%20Tree%20Paths.java) Level: Easy

方法1：
Recursive:分叉。Helper。

方法2，Iterative:
非递归练习了一下
因为要每次切短list, 所以再加了一个Stack 来存level

---

31. Binary Tree Postorder Traversal.java (https://github.com/shawnfan/LintCode/blob/master/Java/Binary%20Tree%20Postorder%20Traversal.java) Level: Easy

最prefer 2 stack的做法：
stack1和stack2合作。倒水。记这个做法。。。挺神奇的。

Divide and Conquer 的recursive方法也非常明了！

注意，这些binary tree traversal的题目，常常有多个做法:recursive or iterative

32. Binary Tree Preorder Traversal.java (https://github.com/shawnfan/LintCode/blob/master/Java/Binary%20Tree%20Preorder%20Traversal.java) Level: Easy

Preorder 写写， stack

1. Divide and conquer
2. Stack(NON-recursive) push curr, push right, push left.
3. recursive with helper method

---

33. Binary Tree Right Side View.java (https://github.com/shawnfan/LintCode/blob/master/Java/Binary%20Tree%20Right%20Side%20View.java) Level: Medium

最右:即level traversal每一行的最末尾.

BFS，用queue.size()来出发saving result.

---

34. Binary Tree Serialization.java (https://github.com/shawnfan/LintCode/blob/master/Java/Binary%20Tree%20Serialization.java) Level: Medium

方法1: BFS. Non-recursive, using queue. 想法直观。level-order traversal. save到一个string里面就好。

方法2: DFS. Recursive. 需要一点思考。basically divide and conquer. 但是代码相对来说短。

---

35. Binary Tree Zigzag Level Order Traversal.java (https://github.com/shawnfan/LintCode/blob/master/Java/Binary%20Tree%20Zigzag%20Level%20Order%20Traversal.java) Level: Medium

简单的level traversal.根据level奇数偶数而add到不同位子.

---

36. Building Outline.java (https://github.com/shawnfan/LintCode/blob/master/Java/Building%20Outline.java) Level: Hard

又叫做skyline

看 网 上 的 解 答 做 ， 思 路 很 漂 亮 。 (http://codechen.blogspot.com/2015/06/leetcode-skyline-problem.html?_sm_au_=isVmHvFmFs40TWRt) (http://codechen.blogspot.com/2015/06/leetcode-skyline-problem.html?_sm_au_=isVmHvFmFs40TWRt%EF%BC%89)

跟scan line的approach类似:

1. 把所有点分出来， 每个点有index x, 再加上一个height.
2. 在这个list上排序，根据index和height（注意用负数标记building start point,这样保证start在end 之前。）. 叫做 heightPoints

3. 在processss时候用max-heap (reversed priorityqueue)，在ieteraete heightPoints 来存最大的height . 遇到peek,就是一个合理的解

处理1：因为start,end的height都存在了heightPoints里面，这里就是用来check end of bulding的，然后把height 从queue里面remove.

处理2：重复x 上面的许多height? priorityqueue给了我们最高，这okay了；那么其他的重复点，用一个int prev来mark之前做过的，一旦重复，跳过。

想法非常natural。 大题目，脑子乱。
看了解答再去想，挺naturally doable的。

---

37. Burst Balloons.java (https://github.com/shawnfan/LintCode/blob/master/Java/Burst%20Balloons.java)dp[i][j] = balloons i~j 之间的sum. 然后找哪个点开始burst? 设为x。
For loop 所有的点作为x ， 去burst。
每次burst都切成了三份：左边可以recusive 求左边剩下的部分的最大值 + 中间3项相乘 + 右边递归下去求最大值。

这个是momorization, 而不纯是DP
因为recursive了，其实还是搜索，但是memorize了求过的值，节省了Processing

---

38. Change to Anagram.java (https://github.com/shawnfan/LintCode/blob/master/Java/Change%20to%20Anagram.java) Level: Easy

简单的check int[26] 26个小写字母是否需要改变。若需要count+1.

主要HackerRank里要注意自己写: Scanner, import java.util, non-static method ...etc.

注意: 最后count出来要除以2：字母不同，会在不同的字母位上加减count,那么就是刚好重复计算了一遍。所以除以二。

---

39. Classical Binary Search.java (https://github.com/shawnfan/LintCode/blob/master/Java/Classical%20Binary%20Search.java)
while: start + 1 < end
mid = start + (end - start) / 2;
末尾double check start, end.

---

40. Climbing Stairs.java (https://github.com/shawnfan/LintCode/blob/master/Java/Climbing%20Stairs.java) Level: Easy

方法1: DP。爬坡到i点总共有的方法，取决于i-1点和i-2的情况。也就是DP(i-1) + DP(i-2).

还可以用滚动数组优化一点：因为用到的变量就只有i,i-1,i-2，可以被替代。
注意要写好'滚动'的代码。

方法2: DFS但是timeout

41. Clone Graph.java (https://github.com/shawnfan/LintCode/blob/master/Java/Clone%20Graph.java) Level: Medium

Use HashMap to mark cloned nodes.

先能复制多少Node复制多少。然后把neighbor 加上

---

42. Closest Binary Search Tree Value.java (https://github.com/shawnfan/LintCode/blob/master/Java/Closest%20Binary%20Search%20Tree%20Value.java) Level: Easy

Binary Search. 记录找到过的closest. 直到tree leaf, 找完return

---

43. Closest Number in Sorted Array.java (https://github.com/shawnfan/LintCode/blob/master/Java/Closest%20Number%20in%20Sorted%20Array.java) Level: Easy

跟Closest Binary Search Tree Vlaue类似：

Binary search. 考虑mid-1, mid+1.
一旦没有mid = target.index。 那么target最终就narrow down在(mid-1,mid) 或者(mid,mid+1)

---

44. Coins in a Line.java (https://github.com/shawnfan/LintCode/blob/master/Java/Coins%20in%20a%20Line.java)However, 分析过后简直屌炸天。一个 n%3就解决了。纯粹math.

---

45. ColorGrid.java (https://github.com/shawnfan/LintCode/blob/master/Java/ColorGrid.java) Level: Medium

用HashMap ， 理解题目规律，因为重复的计算可以被覆盖，所以是个优化题。

消灭重合点:
如果process当下col, 其实要减去过去所有加过的row的交接点。。。
再分析，就是每次碰到row 取一个单点, sumRow += xxx。
然后process当下col时候， sum += colValue * N - sumRow. 就等于把交叉所有row（曾经Process过的row）的点减去了。很方便。

最后read in 是O(P), process也是O(P).

---

46. Combination Sum II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Combination%20Sum%20II.java) Level: Medium

还是DFS. 和Combination Sum I 类似.
确保Helper是用i+1，下一层的数字, 不允许重复。

---

47. Combination Sum.java (https://github.com/shawnfan/LintCode/blob/master/Java/Combination%20Sum.java) Level: Medium

递归，backtracking. 非常normal。需要先sort.
记得求sum时候也pass 一个sum进去，backtracking一下sum也，这样就不必每次都sum the list了。

题目里面所同一个元素可以用n次，但是，同一种solution不能重复出现。如何handle?

1. 用一个index（我们这里用了start）来mark每次recursive的起始点。
2. 每个recursive都从for loop里面的i开始，而i = start。 也就是，下一个iteration,这个数字会有机会被重复使用。
3. 同时，确定在同一个for loop里面，不同的Index上面相同的数字，不Process两遍。用一个prev 作为checker.

假如[x1, x2, y, z], where x1 == x2， 上面做法的效果:
我们可能有这样的结果: x1,x1,x1,y,z
但是不会有:x1,x2,x2,y,z
两个solution从数字上是一样的，也就是duplicated solution, 要杜绝第二种。

---

48. Combinations.java (https://github.com/shawnfan/LintCode/blob/master/Java/Combinations.java) Level: Medium

Combination DFS。 画个图想想. 每次从1~n里面pick一个数字i

因为下一层不能重新回去 [0~i]选，所以下一层recursive要从i+1开始选。

---

49. Compare Strings.java (https://github.com/shawnfan/LintCode/blob/master/Java/Compare%20Strings.java) Level: Easy

比较一下大小, null.

然后用char[]来count chars from A. 再对照chars in B.

---

50. Complete Binary Tree.java (https://github.com/shawnfan/LintCode/blob/master/Java/Complete%20Binary%20Tree.java) Level: Easy

BFS

Use a flag . 当出现了第一次有 null children的node的时候，
说明complete tree的最低level出现了。
自此以后，queue再不该有node再有child, queue后面出现的node的左右孩子应该都是null.

51. Construct Binary Tree from Inorder and Postorder Traversal.java (https://github.com/shawnfan/LintCode/blob/master/Java/Construct%20Binary%20Tree%20from%20Inorder%20and%20Postorder%20Traversal.java) Level: Medium

写个Inorder和Postorder的例子。利用他们分left/right subtree的规律解题。

Postorder array 的末尾， 就是当下层的root.
在Inorder array 里面找到这个root,就刚好把左右两边分割成left/right tree。

这题比较tricky地用了一个helper做recursive。 特别要注意处理index的变化, precisely考虑开头结尾

可惜有个不可避免的O(n) find element in array.

---

52. Construct Binary Tree from Inorder and Preorder Traversal.java (https://github.com/shawnfan/LintCode/blob/master/Java/Construct%20Binary%20Tree%20from%20Inorder%20and%20Preorder%20Traversal.java) Level: Medium

和Construct from Inorder && Postorder 想法一样。

写出Preorder和Inorder的字母例子，发现Preorder的开头总是这Level的root。依此写helper,注意处理index。

---

53. Container With Most Water.java (https://github.com/shawnfan/LintCode/blob/master/Java/Container%20With%20Most%20Water.java)左右两墙，往中间跑动。
另，若一面墙已经小于另外一面，就要移动，换掉矮墙（可能下一面更高，或更低）；但决不能换掉当下的高墙，因为低墙已经limit的盛水的上限，若高墙移动，导致两墙之间距离减少，就注定水量更少了。（弄啥来，不能缺心眼啊）

---

54. Convert Binary Search Tree to Doubly Linked List.java (https://github.com/shawnfan/LintCode/blob/master/Java/Convert%20Binary%20Search%20Tree%20to%20Doubly%20Linked%20List.java) Level: Medium

会iterative traverse Binary Search Tree就好（Stack && handle left-dig-down），然后create Doubly-ListNode 时候注意就好.

注意inorder traversal在check right node的事后，
不论right == null or != null, 每次都要强行move to right.

如果不node = node.right,
很可能发生窘境：
node alays = stack.top(), 然后stack.top()一直是一开始把left 全部遍历的内容。所以就会infinite loop, 永远在左边上下上下。

---

55. Convert Expression to Polish Notation.java (https://github.com/shawnfan/LintCode/blob/master/Java/Convert%20Expression%20to%20Polish%20Notation.java) Level: Hard

还是Expression Tree (Min-Tree).

根据题意，Tree出来以后，来个Pre-order-traversal.

Note: label需要是String.虽然 Operator是长度为1的char, 但是数字可为多位。

---

56.   Convert   Expression   to   Reverse   Polish   Notation.java
(https://github.com/shawnfan/LintCode/blob/master/Java/Convert%20Expression%20to%20Reverse%20Polish%20Notation.java) Level: Hard

build expression tree。

这个里面把TreeNode就当做成我们需要的node,里面扩展成有left/right child的node.

建造Expression Tree,然后根据 Reverse Polish Notation 的定义，来个post-traversal就行了。

---

57. Convert Integer A to Integer B.java (https://github.com/shawnfan/LintCode/blob/master/Java/Convert%20Integer%20A%20to%20Integer%20B.java) Level: Easy

Bit Manipulation

a^b 显示出bit format里面有不同binary code的数位.

每次 (a^b)>>i 移动i位之后， 再 & 1时其实是指留下这一位的数字.

count it up

---

58.   Convert   Sorted   Array   to   Binary   Search   Tree   With   Minimal   Height.java
(https://github.com/shawnfan/LintCode/blob/master/Java/Convert%20Sorted%20Array%20to%20Binary%20Search%20Tree%20With%20Minimal%20Height.java) Level: Easy

Binary Search的感觉. 中间一开两半, divde and conquer,左右各自recursive下去build left/right child.

---

59.   Convert   Sorted   List   to   Binary   Search   Tree.java
(https://github.com/shawnfan/LintCode/blob/master/Java/Convert%20Sorted%20List%20to%20Binary%20Search%20Tree.java) Level: Medium

Divide and Conquer
用快慢pointer

找到mid。

然后把root = mid.next

然后开始sortedListToBST(mid.next.next); //后半段
mid.next = null;//非常重要，要把后面拍过序的断掉
sortedListToBST(head); //从头开始的前半段

最后root.left, root.right merge一下。

---

60. Copy List with Random Pointer.java (https://github.com/shawnfan/LintCode/blob/master/Java/Copy%20List%20with%20Random%20Pointer.java) Level: Medium

Basic Implementation, 其中用了一下HashMap：

遍历head.next .... null.
每一步都check map里面有没有head。没有？加上。
每一步都check map里面有没有head.random。没有？加上。

---

61. Cosine Similarity.java (https://github.com/shawnfan/LintCode/blob/master/Java/Cosine%20Similarity.java) Level: Easy

basic implementation

---

62. Count 1 in Binary.java (https://github.com/shawnfan/LintCode/blob/master/Java/Count%201%20in%20Binary.java) Level: Easy

 1. 可以把integer -> string -> char array.

 2. 或者就 count += num << i & 1

---

63. Count and Say.java (https://github.com/shawnfan/LintCode/blob/master/Java/Count%20and%20Say.java) Level: Easy

Basic implementation. Count duplicates and print

---

64. Count of Smaller Number before itself.java (https://github.com/shawnfan/LintCode/blob/master/Java/Count%20of%20Smaller%20Number%20before%20itself.java) Level: Hard

与Count of Smaller Number非常类似。以实际的value来构成segment tree，leaf上存（count of smaller number）。

Trick: 先Query，再modify.

每次Query时候，A[i]都还没有加入到Segment Tree 里面，而A[i+1,...etc]自然也还没有加进去。

那么就自然是coutning smaller number before itself.

刁钻啊！

另外注意：

在modify里面：多Check了root.start <= index 和 index <= root.end。 过去都忽略了。以后可以把这个也写上。

（其实是Make sense的，就是更加严格地check了index再 root.left 或者 root.right里面的站位）

---

65. Count of Smaller Number.java (https://github.com/shawnfan/LintCode/blob/master/Java/Count%20of%20Smaller%20Number.java) Level: Medium

和平时的segment tree问题不同。 0 ~ n-1代表实际数字。是造一个based on real value的segment tree.

Modify时，把array里面的value带进去，找到特定的位子（leaf）,然后count+1.

最终在SegmentTree leaf上面全是array里面实际的数字。

trick:

在query前，给进去的start和end是： 0 ~ value-1.

value-1就是说，找比自己所在range小1的range（那么自然而然地就不包括自己了），这样就找到了smaller number.

[那么其他做过的SegmentTree是怎么样呢？]

那些构成好的SegmentTree(找min,max,sum)也有一个Array。但是构成Tree时候，随Array的index而构架。

也就是说，假如有Array[x,y,....]:在leaf,会有[0,0] with value = x. [1,1] with value = y.

[但是这题]

构成时，是用actual value.也就是比如Array[x,y,....]会产生leaf:[x,x]with value = ..; [y,y]with value =...

其实很容易看穿:

若给出一个固定的array构成 SegmentTree，那估计很简单：按照index从0~array.lengh，leaf上就是[0,0] with value = x.

若题目让构造一个空心SegmentTree, based on value 0 ~ n-1 (n <= 10000), 然后把一个Array的value modify 进去。

这样八成是另外一种咯。

---

66. Count Primes.java (https://github.com/shawnfan/LintCode/blob/master/Java/Count%20Primes.java) Level: Easy

什么是prime number: >=2的没有除自己和1以外公约数的数。

还有另外一个定义方法!!

这个n,有没有小于n的一个i,而达到： i*i + # of i = n. 如果有，那就不是 prime。

方法很牛逼也很数学。没做的时候可能想不到。做了之后就觉得，哎，我去，有道理啊。

简而言之：简历一个boolean长条，存isPrime[]。 然后从i=2， 全部变true.

然后利用这个因子的性质，非prime满足条件： selfself, self self + self ... etc.

所以就check每一个j, j+i, j+i+i, 然后把所有non-prime全部mark成false.

最后，数一遍还剩下的true个数就好了

---

67. Course Schedule II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Course%20Schedule%20II.java) Level: Medium

详细的中文分析，看Course Schedule I

---

68. Course Schedule.java (https://github.com/shawnfan/LintCode/blob/master/Java/Course%20Schedule.java) Level: Medium

有点绕，但是做过一次就明白一点。

是topological sort的题目。一般都是给有dependency的东西排序。

最终都会到一个sink node， 再不会有向后的dependency, 在那个点截止。

我就已这样子的点为map的key, 然后value是以这个node为prerequisite的 list of courses.

画个图的话，prerequisite都是指向那个sink node， 然后我们在组成map的时候，都是从sink node 发散回来到dependent nodes.

在DFS里面，我们是反向的， 然后，最先完全visited的那个node, 肯定是最左边的node了，它被mark的seq也是最高的。

而我们的sink node，当它所有的支线都visit完了，seq肯定都已经减到最小了，也就是0，它就是第一个被visit的。

最终结果：

每个有pre-requisit的node都trace上去（自底向上），并且都没有发现cycle.也就说明schedule可以用了。

---

69. Data Stream Median.java (https://github.com/shawnfan/LintCode/blob/master/Java/Data%20Stream%20Median.java) Level: Hard

把Input stream想成向上的山坡。山坡中间那点，自然就是median.

前半段，作为maxHeap,关注点是PriorityQueue的峰点，也就是实际上的median.

后半段，作为minHeap,正常的PriorityQueue。 开头是最小的。

Note:题目定义meadian = A[(n−1)/2],也就是说maxHeap需要和minHeap长度相等，或者多一个element,最后可以直接poll() and return.

---

70. Delete Digits.java (https://github.com/shawnfan/LintCode/blob/master/Java/Delete%20Digits.java) Level: Medium

数位靠前的，权值更大. 所以硬来把靠前的相对更大的（跟following digit相比）去掉。

---

71. Delete Node in the Middle of Singly Linked List.java (https://github.com/shawnfan/LintCode/blob/master/Java/Delete%20Node%20in%20the%20Middle%20of%20Singly%20Linked%20List.java) Level: Easy

Just do it. Link curr.next to curr.next.next

---

72. Distinct Subsequences.java (https://github.com/shawnfan/LintCode/blob/master/Java/Distinct%20Subsequences.java) Level: Hard

Not Done

---

73. Edit Distance.java (https://github.com/shawnfan/LintCode/blob/master/Java/Edit%20Distance.java) Level: Medium

Not Done

---

74. Encode and Decode Strings.java (https://github.com/shawnfan/LintCode/blob/master/Java/Encode%20and%20Decode%20Strings.java) Level: Medium

方法1:
用数字+"#"+string来encode.
基于我们自己定的规律, 在decode的里面不需要过多地去check error input, assume所有input都是规范的.
decode就是找"#",然后用"#"前的数字截取后面的string.

Old Solution:
Cast character into int. 串联起来, seperate by "LINE".
handle empty list [], or just null: 要把Null特别mark一下为'NULL', 这样decode时才能check到。 adminadmin

---

75. ExcelSheetColumnNumber .java (https://github.com/shawnfan/LintCode/blob/master/Java/ExcelSheetColumnNumber%20.java) Level: Easy

'A' − 'A' = 0. 所以 char − 'A' + 1 = 题目里的对应数位。
26位运算和10位一样嘛，num += 每位的digit * Math.pow(26, 数位号)。

---

76. Expression Evaluation.java (https://github.com/shawnfan/LintCode/blob/master/Java/Expression%20Evaluation.java) Level: Hard

Build Expression Tree的另外一个变形，依然Min Tree.

build好Min Tree以后，做PostTraversal. Divde and Conquer：
先recursively找到 left和right的大小， 然后evaluate中间的符号。

Note:

1. Handle数字时，若left&&right Child全Null,那必定是我们weight最大的数字node了。
2. 若有个child是null,那就return另外一个node。
3. prevent Integer overflow　during operation:过程中用个Long，最后结局在cast back to int.

---

77. Expression Tree Build.java (https://github.com/shawnfan/LintCode/blob/master/Java/Expression%20Tree%20Build.java) Level: Hard

和Max-tree一样，感谢http://blog.welkinlan.com/2015/06/29/max-tree-lintcode-java/ (http://blog.welkinlan.com/2015/06/29/max-tree-lintcode-java/)

这个题目是Min-tree， 头上最小，Logic 和max-tree如出一辙

注意treeNode,为了帮助ExpressionTreeNode 排序。它加了一个weight based on expression，协助build Min-Tree 排序。

Space: O(n)
Time on average: O(n).

---

78. Fast Power.java (https://github.com/shawnfan/LintCode/blob/master/Java/Fast%20Power.java) Level: Medium

a^n可以被拆解成(aaaa....a)， 是乘机形式，而%是可以把每一项都mod一下的。所以就拆开来take mod.

这里用个二分的方法，recursively二分下去，直到n/2为0或者1，然后分别对待.

注意1: 二分后要conquer，乘积可能大于Integer.MAX_VALUE, 所以用个long.

注意2: 要处理n%2==1的情况，二分时候自动省掉了一份，要乘一下。

---

79. Fibonacci.java (https://github.com/shawnfan/LintCode/blob/master/Java/Fibonacci.java) Level: Easy

方法1: DP array.

方法1.1: 滚动数组, 简化DP。

方法2: recursively calculate fib(n – 1) + fib(n – 2). 公式没问题, 但是时间太长, timeout.

---

80. Find Minimum in Rotated Sorted Array II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Find%20Minimum%20in%20Rotated%20Sorted%20Array%20II.java)Medium Find Minimum in Rotated Sorted Array II My Submissions

40% Accepted
Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

Find the minimum element.

The array may contain duplicates.

Example
Given [4,4,5,6,7,0,1,2] return 0

Tags Expand
Binary Search Divide and Conqueri

Thinking process:
It seems using binary search will leads to O(n), so just use a for loop with O(n)
*/

public class Solution {
/**

```
 * @param num: a rotated sorted array
 * @return: the minimum number in the array
 */
public int findMin(int[] num) {
    if (num == null || num.length == 0) {
        return -1;
    }
    int min = Integer.MAX_VALUE;
    for (int i = 0; i < num.length; i++) {
        if (min > num[i]) {
            min = num[i];
        }
    }
    return min;
}
```

}

---

81. Find Minimum in Rotated Sorted Array.java (https://github.com/shawnfan/LintCode/blob/master/Java/Find%20Minimum%20in%20Rotated%20Sorted%20Array.java)Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2).

Find the minimum element.

You may assume no duplicate exists in the array.

Example
Given [4,5,6,7,0,1,2] return 0

Tags Expand
Binary Search

Thinking process:
Understand how to use binary in this problem: compare the mid point with end point.
In this problem, because the sorted line is cut at one point then rotate, so one of the line is absolutely greater than the other line.
Situation 1:
if mid < end : that means minimum is on the end point's line. Move end to left. end = mid.
Situation 2:
if mid > end: that means there must be a mountain-jump somewhere after mid and before end, which is the minimum point. Now move start to mid.
*/

```java
public class Solution {
/**
 * @param num: a rotated sorted array
 * @return: the minimum number in the array
 */
public int findMin(int[] num) {
    if (num == null || num.length == 0) {
        return -1;
    }
    int start = 0;
    int end = num.length - 1;
    int mid = 0;
    while (start + 1 < end) {
        mid = start + (end - start) / 2;
        if (num[mid] > num[end]) {
            start = mid;
        } else {
            end = mid;
        }
    }
    if (num[start] < num[end]) {
        return num[start];
    } else {
        return num[end];
    }
}

}
```

---

82. Find Peak Element II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Find%20Peak%20Element%20II.java)There is an integer matrix which has the following features:

The numbers in adjacent positions are different.
The matrix has n rows and m columns.
For all i < m, A[0][i] < A[1][i] && A[n − 2][i] > A[n − 1][i].
For all j < n, A[j][0] < A[j][1] && A[j][m − 2] > A[j][m − 1].
We define a position P is a peek if:

A[j][i] > A[j+1][i] && A[j][i] > A[j−1][i] && A[j][i] > A[j][i+1] && A[j][i] > A[j][i−1]
Find a peak element in this matrix. Return the index of the peak.

Have you met this question in a real interview? Yes

Example

Given a matrix:

[
[1 ,2 ,3 ,6 ,5],
[16,41,23,22,6],
[15,17,24,21,7],
[14,18,19,20,10],
[13,14,11,10,9]
]

return index of 41 (which is [1,1]) or index of 24 (which is [2,2])

Note

The matrix may contains multiple peeks, find any of them.

Challenge

Solve it in O(n+m) time.

If you come up with an algorithm that you thought it is O(n log m) or O(m log n), can you prove it is actually O(n+m) or propose a similar but O(n+m) algorithm?

Tags Expand

Binary Search LintCode Copyright Matrix

*/

/

NOT DONE. Will try if have time /

class Solution {
/**

```
  * @param A: An integer matrix
  * @return: The index of the peak
  */
 public List<Integer> findPeakII(int[][] A) {
     // write your code here
 }
```

}

83. Find Peak Element.java (https://github.com/shawnfan/LintCode/blob/master/Java/Find%20Peak%20Element.java)一个特别的check condition, 和特别的move left, move right的case罢了。

---

84. Find the Connected Component in the Undirected Graph.java (https://github.com/shawnfan/LintCode/blob/master/Java/Find%20the%20Connected%20Component%20in%20the%20Undirected%20Graph.java) Level: Medium

BFS遍历，把每个node的neighbor都加进来。

一定注意要把visit过的node Mark一下。因为curr node也会是别人的neighbor，会无限循环。

Component的定义：所有Component内的node必须被串联起来via path (反正这里是undirected, 只要链接上就好)

这道题：其实component在input里面都已经给好了，所有能一口气visit到的，全部加进queue里面，他们就是一个component里面的了。

而我们这里不需要判断他们是不是Component。

---

85. Find the Weak Connected Component in the Directed Graph.java (https://github.com/shawnfan/LintCode/blob/master/Java/Find%20the%20Weak%20Connected%20Component%20in%20the%20Directed%20Graph.java) Level: Medium

Identify这是个union-find问题还挺巧妙。
看到了weak component的形式：一个点指向所有，那么所有的点都有一个公共的parent，然后就是要找出这些点。

为何不能从一个点出发，比如A，直接print它所有的neighbors呢？
不行，如果轮到了B点，那因为是directed,它也不知道A的情况，也不知道改如何继续加，或者下手。

所以，要把所有跟A有关系的点，或者接下去和A的neighbor有关系的点，都放进union-find里面，让这些点有Common parents.

最后output的想法：
做一个 map <parent ID, list>。
之前我们不是给每个num都存好了parent了嘛。
每个num都有个parent, 然后不同的parent就创造一个不同的list。
最后，把Map里面所有的list拿出来就好了。

---

86. First Bad Version.java (https://github.com/shawnfan/LintCode/blob/master/Java/First%20Bad%20Version.java) Level: Medium

Binary Search

根据isBadVersion的性质，判断还如何end=mid or start=mid.

isBadVersion 是有方向的嘛，一个点错了，后面全错。

---

87. First Missing Positive.java (https://github.com/shawnfan/LintCode/blob/master/Java/First%20Missing%20Positive.java)Given an unsorted integer array, find the first missing positive integer.

Example
Given [1,2,0] return 3, and [3,4,-1,1] return 2.

Challenge
Your algorithm should run in O(n) time and uses constant space.

Tags Expand
Array

Thoughts:
It means: after it's sorted, what's the first missing postive int counted from 1 ---> more

1. Arrays.sort();

2. count = first non-zero element in A.

3. count +1, and see if maches the current A[i]?

NOTE:
Deal with negative and positive number separately
Watch out for redundant number: ask if the list has duplicated elements
*/

public class Solution {
/**

```
     * @param A: an array of integers
     * @return: an integer
     */
    public int firstMissingPositive(int[] A) {
        if (A == null || A.length == 0) {
            return 1;
        }
        Arrays.sort(A);
        int count = -1;
        for (int i = 0; i < A.length; i++) {
            if (A[i] > 0) {
                if (count < 0) {//process 1st positive element
                    count = A[i];
                    if (count != 1) {
                        return 1;
                    }
                }
                else if (A[i] == A[i - 1]) {//watch out for duplicates
                    count--;
                }
                else if(A[i] != count) {//if not match, kick out
                    return count;
                }
                count++;
            }
        }
        if (count < 0) {//if all negative, return 1
            return 1;
        }
        return count;
    }

}
```

---

88. Flatten 2D Vector.java (https://github.com/shawnfan/LintCode/blob/master/Java/Flatten%202D%20Vector.java)Implement an iterator to flatten a 2d vector.

For example,
Given 2d vector =

[
[1,2],
[3],
[4,5,6]
]
By calling next repeatedly until hasNext returns false, the order of elements returned by next should be: [1,2,3,4,5,6].

Hint:

How many variables do you need to keep track?

Two variables is all you need. Try with x and y.

Beware of empty rows. It could be the first few rows.

To write correct code, think about the invariant to maintain. What is it?

The invariant is x and y must always point to a valid point in the 2d vector. Should you maintain your invariant ahead of time or right when you need it?

Not sure? Think about how you would implement hasNext(). Which is more complex?

Common logic in two different places should be refactored into a common method.

Tags: Design

Similar Problems: (M) Binary Search Tree Iterator, (M) Zigzag Iterator, (M) Peeking Iterator

*/

/
Thoughts:
As hint indicates: use 2 pointers to hold position.
Use hasNext to validate (x,y) and move x.
Use next() to return (x,y) and move it(regardless of correctness, which is determined by hasNext()) /

```java
public class Vector2D {
private int x;
private int y;
private List<List<Integer>> list;
public Vector2D(List<List<Integer>> vec2d) {
if (vec2d == null) {
return;
}
this.x = 0;
this.y = 0;
this.list = vec2d;
}
```

```
public int next() {
    int rst = list.get(x).get(y);
    if (y + 1 >= list.get(x).size()) {
        y = 0;
        x++;
    } else {
        y++;
    }
    return rst;
}

public boolean hasNext() {
    if (list == null) {
        return false;
    }
    while (x < list.size() && list.get(x).size() == 0) {
        x++;
        y = 0;
    }
    if (x >= list.size()) {
        return false;
    }
    if (y >= list.get(x).size()) {
        return false;
    }
    return true;
}
```

}

/**

- Your Vector2D object will be instantiated and called as such:

- Vector2D i = new Vector2D(vec2d);

- while (i.hasNext()) v[f()] = i.next();

  */

---

89. Flatten Binary Tree to Linked List.java (https://github.com/shawnfan/LintCode/blob/master/Java/Flatten%20Binary%20Tree%20to%20Linked%20List.java)
Not Done

---

90. Flattern 2D Vector.java (https://github.com/shawnfan/LintCode/blob/master/Java/Flattern%202D%20Vector.java)注意啊，一开始理解题意搞错：我以为是必须要排序正确，
所以上来就PriorityQueue+HashMap搞得无比复杂。其实，这个跟一个nxn的matrix遍历，是没区别的拉。
所有来个x,y，把2d list跑一变。

---

91. Flip Game II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Flip%20Game%20II.java)注意：不要乱改input s. recursive call 需要用原始的input s.

这个题目李特是屌炸天的。
我飞了九牛二虎之力（路子对），但是代码写的七荤八素，好长好长好长好长的。
结果正解，三四行就搞定了。真是心有不甘啊。
想法如下：
保证p1能胜利，就必须保持所有p2的move都不能赢。
同时，p1只要在可走的Move里面，有一个move可以赢就足够了。
（题目里面用一个for loop + 只要 满足条件就return true来表达 OR的意思：p1不同的路子，赢一种就行了）
p1: player1
p2: player2

---

92. Flip Game.java (https://github.com/shawnfan/LintCode/blob/master/Java/Flip%20Game.java)很郁闷的就是条件不明，原来只需要从'++'转到'--'的情况，反过来没必要关注...
搞了我半天啊

---

93. Fraction to Recurring Decimal.java (https://github.com/shawnfan/LintCode/blob/master/Java/Fraction%20to%20Recurring%20Decimal.java)很容易忽略的是integer的益处。

---

94. Game of Life.java (https://github.com/shawnfan/LintCode/blob/master/Java/Game%20of%20Life.java)According to the Wikipedia's article:
"The Game of Life, also known simply as Life, is a cellular automaton devised by the British mathematician John Horton Conway in 1970."

Given a board with m by n cells, each cell has an initial state live (1) or dead (0).
Each cell interacts with its eight neighbors (horizontal, vertical, diagonal) using the following four rules
(taken from the above Wikipedia article):

Any live cell with fewer than two live neighbors dies, as if caused by under-population.
Any live cell with two or three live neighbors lives on to the next generation.
Any live cell with more than three live neighbors dies, as if by over-population..
Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.
Write a function to compute the next state (after one update) of the board given its current state.

Follow up:
Could you solve it in-place? Remember that the board needs to be updated at the same time:
You cannot update some cells first and then use their updated values to update other cells.
In this question, we represent the board using a 2D array.
In principle, the board is infinite, which would cause problems when the active area encroaches the border of the array.

How would you address these problems?
Credits:
Special thanks to @jianchao.li.fighter for adding this problem and creating all test cases.

Hide Company Tags Google TinyCo
Hide Tags Array
Hide Similar Problems (M) Set Matrix Zeroes

*/

/
Thoughts:
https://segmentfault.com/a/1190000003819277 (https://segmentfault.com/a/1190000003819277)
http://my.oschina.net/Tsybius2014/blog/514447 (http://my.oschina.net/Tsybius2014/blog/514447)
build state machine.
take mod of 2 at the end. /

```
public class Solution {
public void gameOfLife(int[][] board) {

    }

}
```

---

95. Gas Station.java (https://github.com/shawnfan/LintCode/blob/master/Java/Gas%20Station.java)There are N gas stations along a circular route, where the amount of gas at station i is gas[i].

You have a car with an unlimited gas tank and it costs cost[i] of gas to travel from station i to its next station (i+1). You begin the journey with an empty tank at one of the gas stations.

Return the starting gas station's index if you can travel around the circuit once, otherwise return -1.

Example
Given 4 gas stations with gas[i]=[1,1,3,1], and the cost[i]=[2,2,1,1]. The starting gas station's index is 2.

Note
The solution is guaranteed to be unique.

Challenge

O(n) time and O(1) extra space

Tags Expand
Greedy

Thoughts:
Loop through the gas station, and track the possible starting index.
Start from i = 0 ~ gas.length, and use a second pointer move to track how far we are travelling
calculate: remain += gas[i] – cost[i]. (remain + gas[i] – cost[i]: the remaining gas plus i's gas, can we make it to i+1 gas station?)
if remain < 0, fail. Note: if from i ~ j can't work, even it's possible that i can make it to i+1's station, but i+1 ~ j won't work still.
Thus, once i's station failed to get to x, set index = x + 1: we are moving on to next possible starting point.

'total':simply indicates if we can make it a circle
*/

public class Solution {
/**

```
 * @param gas: an array of integers
 * @param cost: an array of integers
 * @return: an integer
 */
public int canCompleteCircuit(int[] gas, int[] cost) {
    if (gas == null || cost == null || gas.length == 0 || cost.length == 0) {
        return -1;
    }
    int start = 0;
    int remain = 0;
    int total = 0;
    for (int i = 0; i < gas.length; i++) {
        remain += gas[i] - cost[i];
        if (remain < 0) {
            remain = 0;
            start = i + 1;
        }
        total += gas[i] - cost[i];
    }
    if (total < 0) {
        return -1;
    }
    return start;
}
```

}

96. Generate Parentheses.java (https://github.com/shawnfan/LintCode/blob/master/Java/Generate%20Parentheses.java)看thought.取或者不取(,)

97. Graph Valid Tree.java (https://github.com/shawnfan/LintCode/blob/master/Java/Graph%20Valid%20Tree.java) Level: Medium

复习Union-Find的另外一个种形式。

题目类型：查找2个元素是不是在一个set里面。如果不在，false. 如果在，那就合并成一个set,共享parent.

存储的关键都是：元素相对的index上存着他的root parent.

另 一 个 union-find， 用 hashmap 的 ： http://www.lintcode.com/en/problem/find-the-weak-connected-component-in-the-directed-graph/ (http://www.lintcode.com/en/problem/find-the-weak-connected-component-in-the-directed-graph/)

98. Gray Code.java (https://github.com/shawnfan/LintCode/blob/master/Java/Gray%20Code.java) Level: Medium

题目蛋疼，目前只接受一种结果。

BackTracking + DFS:
Recursive helper里每次flip一个 自己/左边/右边. Flip过后还要恢复原样.遍历所有.

曾用法（未仔细验证）：
基本想法就是从一个点开始往一个方向走，每次flip一个bit, 碰壁的时候就回头走。

99. Group Anagrams.java (https://github.com/shawnfan/LintCode/blob/master/Java/Group%20Anagrams.java) Level: Medium

方法一: 60%

和check anagram 想法一样：转化并sort char array，用来作为key。

把所有anagram 存在一起。注意结尾Collections.sort().

O(NKlog(K)), N = string[] length, k = longest word length

优化：80% ~ 97%

用固定长度的char[26] arr 存每个字母的frequency; 然后再 new string(arr).
因为每个位子上的frequency的变化，就能构建一个unique的string

错误的示范: 尝试先sort input strs[]，但是NlogN 其实效率更低. 13%

100. Group Shifted Strings.java (https://github.com/shawnfan/LintCode/blob/master/Java/Group%20Shifted%20Strings.java) Level: Easy

相同shift规则的string, 能被推算到同一个零起始点，就是共同减去一个char,最后就相等。以此作为key，用HashMap。一目了然。

记得根据题目意思，一开始要String[] sort一下。

101. H-Index II.java (https://github.com/shawnfan/LintCode/blob/master/Java/H-Index%20II.java)binary search

102. H-Index.java (https://github.com/shawnfan/LintCode/blob/master/Java/H-Index.java) 当然，搜索一遍时候可以优化，用binary search. 但是没意义，因为array.sort已经用了nlogn

o(n)也可以，用bucket. 比较巧妙。

103. Happy Number.java (https://github.com/shawnfan/LintCode/blob/master/Java/Happy%20Number.java) Level: Easy

Basic Implementation of the requirements.

用HashSet存查看过的数值。若重复，return false.

104. Hash Function.java (https://github.com/shawnfan/LintCode/blob/master/Java/Hash%20Function.java) Level: Easy

解释Hash怎么道理。Hash function例子：
hashcode("abcd") = (ascii(a) 33^3 + ascii(b) 33^2 + ascii(c) 33^1 + ascii(d)33^0) % HASH_SIZE

用到的参数比如: magic number 33, HASH_SIZE.

Hash的用法是：给一个string key, 转换成数字，从而把size变得更小。
真实的implementation还要处理collision, 可能需要design hash function 等等。

每一步都：
hashRst = hashRst * 33 + (int)(key[i]);
hashRst = hashRst % HASH_SIZE;
原因是，hashRst会变得太大，所以不能算完再%...

105. HashHeap.java (https://github.com/shawnfan/LintCode/blob/master/Java/HashHeap.java) Level: Hard

非题.是从九章找来的HashHeap implementation.

---

106. HashWithArray.java (https://github.com/shawnfan/LintCode/blob/master/Java/HashWithArray.java) Level: Easy

---

107. HashWithCustomizedClass(LinkedList).java (https://github.com/shawnfan/LintCode/blob/master/Java/HashWithCustomizedClass(LinkedList).java) Level: Medium

练习HashMap with customized class.

---

108. Heapify.java (https://github.com/shawnfan/LintCode/blob/master/Java/Heapify.java) Level: Medium

Heap用的不多. 得用一下, 才好理解。
通常default 的PriorityQueue就是给了一个现成的min-heap：所有后面的对应element都比curr element 小。

Heapify里面的siftdown的部分：
只能从for(i = n/2-1 ~ 0)，而不能从for(i = 0 ~ n/2 -1): 必须中间开花，向上跑的时候才能确保脚下是符合heap规则的

Heapify/SiftDown做了什么？
确保在heap datastructure里面curr node下面的两个孩子，以及下面所有的node都遵循一个规律。
比如在这里，若是min-heap,就是后面的两孩子都要比自己大。若不是，就要swap。

还是要记一下min-heap的判断规律:for each element A[i], we will get A[i 2 + 1] >= A[i] and A[i 2 + 2] >= A[i].

siftdown时：在curr node和两个son里面小的比较。如果的确curr < son, 搞定，break while.
但若curr 并不比son小，那么就要换位子，而且继续从son的位子往下面盘查。

---

109. House Robber II.java (https://github.com/shawnfan/LintCode/blob/master/Java/House%20Robber%20II.java) Level: Medium

和House Robber I 类似, DP.

根据dp[i-1]是否被rob来讨论dp[i]: dp[i] = Math.max(dp[i-1], dp[i - 2] + nums[i - 1]);

特别的是，末尾的last house 和 first house相连。这里就需要分别讨论两种情况:

1. 最后一个房子被rob
2. 最后一个房子没被rob

两种情况做完，综合对比一下.

110. House Robber III.java (https://github.com/shawnfan/LintCode/blob/master/Java/House%20Robber%20III.java) Level: Hard

由于无法用简单的方法构造DP array，所以采取了普通的DFS。

The catch:
判断当下的node是否被采用，用一个boolean来表示.

1. 如果curr node被采用，那么下面的child一定不能被采用。
2. 如果curr node不被采用，那么下面的children有可能被采用，但也可能略过，所以这里用Math.max() 比较一下两种可能有的dfs结果。

111. House Robber.java (https://github.com/shawnfan/LintCode/blob/master/Java/House%20Robber.java) Level: Easy

最基本的dp。
看前一个或前两个的情况，再总和考虑当下的。
思考的适合搞清楚当下的和之前的情况的关系。
滚动数组的优化，就是确定了是这类"只和前一两个位子"相关的Fn而推出的。

112. Identical Binary Tree.java (https://github.com/shawnfan/LintCode/blob/master/Java/Identical%20Binary%20Tree.java) Level: Easy

Divide, && 每种情况（左右一一对应)
注意 null states

113. Implement Queue by Two Stacks.java (https://github.com/shawnfan/LintCode/blob/master/Java/Implement%20Queue%20by%20Two%20Stacks.java)As the title described, you should only use two stacks to implement a queue's actions.

The queue should support push(element), pop() and top() where pop is pop the first(a.k.a front) element in the queue.

Both pop and top methods should return the value of first element.

Example
For push(1), pop(), push(2), push(3), top(), pop(), you should return 1, 2 and 2

Challenge
implement it by two stacks, do not use any other data structure and push, pop and top should be O(1) by AVERAGE.

Thoughts:

1. Push everything into stack2: whatever comes in last, will be on top.

2. Pop and Top: return stack1's top element.

3. Initially, when stack1 is empty, need to reverse all stack2 and put into stack: like pouring water from cup stack2 into cup stack1.

   Or:when stack1 has been top() over, pour stack2 into stack1 again: the stack2's bottom becomes stack1's top, which is correct: returning the oldest element of queue (front of queue)

Tags Expand
LintCode Copyright Stack Queue
*/

public class Solution {
private Stack<Integer> stack1;
private Stack<Integer> stack2;
public void pourS2ToS1(){
while (!stack2.empty()) {
stack1.push(stack2.peek());
stack2.pop();
}
}
public Solution() {
stack1 = new Stack<Integer>();
stack2 = new Stack<Integer>();
}

```
public void push(int element) {
    stack2.push(element);
}

public int pop() {
    if (stack1.empty()) {
        pourS2ToS1();
    }
    return stack1.pop();
}

public int top() {
    if (stack1.empty()) {
        pourS2ToS1();
    }
    return stack1.peek();
}
```

}

114. Implement Stack by Two Queues.java (https://github.com/shawnfan/LintCode/blob/master/Java/Implement%20Stack%20by%20Two%20Queues.java)用一个Temp做
swap

做法1:
逻辑在top()/pop()里, 每次换水，查看末尾项.

做法2:
逻辑在push里面:

1. x 放q2。
2. q1全部offer/append到q2.
3. 用一个Temp做swap q1, q2.
   q1的头，就一直是最后加进去的值.

---

115. Implement Stack.java (https://github.com/shawnfan/LintCode/blob/master/Java/Implement%20Stack.java)Data Structure: ArrayList
return/remove ArrayList的末尾项。

---

116. Implement strStr().java (https://github.com/shawnfan/LintCode/blob/master/Java/Implement%20strStr().java)1. border condition。 如果 haystack.length() <
needle.length() 的话，必须错。但是这个可以优化省略。

1. 当S2是""的时候，也就是能在s1的其实位置找到s2....index = 0.
2. 记得如何在s1里面找s2. 就是把遍历s1的 i，加上遍历s2的 j。

优化：

1. s1, s2长短可以不比较。因为forloop的时候： s1.length() - s2.length() + 1，如果s2长于s1,这里自然就断了。
2. if(s1.charAT(i+j) == s2.charAT(j)). 可以省略。For loop 里面就Check到这个了。

---

117. Implement Trie (Prefix Tree).java (https://github.com/shawnfan/LintCode/blob/master/Java/Implement%20Trie%20(Prefix%20Tree).java)如果是遇到一个一个字查询的
题，可以考虑一下。
构建TrieNode的时候要注意：如何找孩子？如果是个map的话，其实就挺好走位的。
而且，每个node里面的 char 或者string有时候用处不大，
可以为空。但是有些题目，比如在结尾要return一些什么String，就可以在end string那边存一个真的String。

---

118. Implement Trie.java (https://github.com/shawnfan/LintCode/blob/master/Java/Implement%20Trie.java) Level: Medium

Tire, 也即是 Prefix Tree.

HashMap构建Trie。 Trie三个Method:

1. Inset: 加 word
2. Search: 找word
3. StartWith: 找prefix

只有两条children的是binary tree. 那么多个children就是Trie。 那么没有left/right pointer怎么找孩子？
用HashMap，以child的label为Key，value就是child node。 HashMap走位

Note:
node里的char在这是optional。
另外有种题目，比如是跟其他种类的search相关，在结尾要return whole string，就可以在node里存一个up-to-this-point的String。

---

119. IndexMatch.java (https://github.com/shawnfan/LintCode/blob/master/Java/IndexMatch.java) Level: Easy

有序, 假设有这样的数字:target.
target 左边的数字，一定不比index大，target右边的数字，一定比index大。
这样可以binary search.O(logn)

---

120. Inorder Successor in Binary Search Tree.java (https://github.com/shawnfan/LintCode/blob/master/Java/Inorder%20Successor%20in%20Binary%20Search%20Tree.java) Level: Medium

画inorder图，发现规律.每个node的后继node(successor)有几种情况:

1. node.right 是个leaf到底了。那么就return.
2. set rightNode = node.right， 但发现rightNode has a lot left children to leaf.
3. 比如, node.right == null， 也就是node自己是leaf，要回头看山顶找Inorder traversal规则里的下一个。
   发现：其实就是每层都把路过的curr node放在stack里，最上面的，就是当下改return的那个successor:) Done.

---

121. Insert Interval.java (https://github.com/shawnfan/LintCode/blob/master/Java/Insert%20Interval.java) Level: Easy

方法1：Scan Line
Interval 拆点，PriorityQueue排点。
Merge时用count==0作判断点。

PriorityQueue: O(logN). 扫n点，总共：O(nLogn)

方法2：
O(n) 直接找到可以insert newInterval的位子. Insert。 这里已经给了sorted intervals by start point. 所以O(n)

然后loop to merge entire interval array

另外: 因为interval已经sort, 本想用Binary Search O(logn). 但是找到interval insert position， merge还是要用 O(n)。
比如刚好newInterval cover entire list....

---

122. Insert Node in a Binary Search Tree .java (https://github.com/shawnfan/LintCode/blob/master/Java/Insert%20Node%20in%20a%20Binary%20Search%20Tree%20.java)
Level: Easy

往Binary Search Tree里面加东西，一定会找到一个合适的leaf加上去。

那么：就是说someNode.left or someNode.right是null时，就是insert node的地方。

找到那个someNode就按照正常的Binary Search Tree规律。

---

123. Insertion Sort List.java (https://github.com/shawnfan/LintCode/blob/master/Java/Insertion%20Sort%20List.java)基本上就是正常的想法：已经有个sorted list, insert一个element进去。怎么做？
while 里面每个元素都小于 curr, keep going
一旦curr在某个点小了，加进去当下这个空隙。
这个题目也就是：把list里面每个元素都拿出来，scan and insert一遍！

---

124. Integer to English Words.java (https://github.com/shawnfan/LintCode/blob/master/Java/Integer%20to%20English%20Words.java)Convert a non-negative integer to its english words representation. Given input is guaranteed to be less than 2^31 - 1.

For example,
123 -> "One Hundred Twenty Three"
12345 -> "Twelve Thousand Three Hundred Forty Five"
1234567 -> "One Million Two Hundred Thirty Four Thousand Five Hundred Sixty Seven"
Hint:

Did you see a pattern in dividing the number into chunk of words? For example, 123 and 123000.
Group the number by thousands (3 digits). You can write a helper function that takes a number less than 1000 and convert just that chunk to words.
There are many edge cases. What are some good test cases? Does your code work with input such as 0? Or 1000010? (middle chunk is zero and should not be printed out)

Tags: Math, String
Similar Problems: (M) Integer to Roman

Thoughts:
2^31 – 1 = 2,147,483,647
Trillion, Billion, Million, Thousand, Hundred, Ninty .... Ten, Nine ... One, Zero.

 1. Break the words to up to 4 parts: new break[4] by /(1000 ^ i)
 2. For each i, deal with that 3-digit number in break[i]
 3. Append corresponding words for each break[i]

Special case:
zero
000 in one break[i]: skip the whole thing
*/

```java
public class Solution {
public String[] v1 = {"", "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Eleven", "Twelve", "Thirteen", "Fourteen", "Fifteen", "Sixteen", "Seventeen", "Eighteen", "Nineteen"};
public String[] v2 = {"", "", "Twenty", "Thirty", "Forty", "Fifty", "Sixty", "Seventy", "Eighty", "Ninety"};
public String numberToWords(int num) {
if (num < 0) {
return "";
}
if (num == 0) {
return "Zero";
}
String rst = "";
for (int i = 0; i < 4; i++) {
int partial = num - (num / 1000) * 1000;
if (partial > 0) {
rst = helper(partial, i) + " " + rst;
}
num /= 1000;
}
while (rst.charAt(rst.length() - 1) == ' ') {
rst = rst.substring(0, rst.length() - 1);
```

```
}
return rst;
}
```

```java
public String helper(int num, int i) {
    String str = "";
    if (num >= 100) {
        int hund = num / 100;
        str = v1[hund] + " Hundred ";
        num = num % 100;
    }

    if (num < 20) {
        str += v1[num] + " ";
    } else {
        int numTens = num / 10;
        int numDigit = num % 10;
        str += v2[numTens] + " ";
        str += v1[numDigit] + " ";
    }

    while (str.charAt(str.length() - 1) == ' ')  {
        str = str.substring(0, str.length() - 1);
    }

    //depending on i:
    switch (i) {
        case 1 :
            str += " Thousand";
            break;
        case 2 :
            str += " Million";
            break;
        case 3 :
            str += " Billion";
            break;
    }

    return str;
}
```

}

---

125. Interleaving Positive and Negative Numbers.java
(https://github.com/shawnfan/LintCode/blob/master/Java/Interleaving%20Positive%20and%20Negative%20Numbers.java)这里主要要特别考虑，正数多还是负数多的问题。
count一下，然后举两个小栗子就看出来端倪了。
然后Two Pointer

126. Interleaving String.java (https://github.com/shawnfan/LintCode/blob/master/Java/Interleaving%20String.java)Given three strings: s1, s2, s3, determine whether s3 is formed by the interleaving of s1 and s2.

Example
For s1 = "aabcc", s2 = "dbbca"

When s3 = "aadbbcbcac", return true.
When s3 = "aadbbbaccc", return false.
Challenge
O(n2) time or better

Tags Expand
Longest Common Subsequence Dynamic Programming

Attempt2: DP[i][j]: boolean that if first S1(i) chars and first S2(j) chars can interleavign first S3(i + j)
Match one char by one char. We have 2 conditions: match s1 or s2 char, Let's do double-for-loop on s1 and s2

 1. match s1: s3.charAt(i + j -1) == s1.charAt(i - 1) && DP[i - 1][j]; // makes sure DP[i-1][j] also works before adding s1[i-1] onto the match list
 2. match s2: s3.charAt(i + j -1) == s2.charAt(j - 1) && DP[i][j - 1]// similar as above

Note:
Need to initiate the starting conditions with just s1, or just s2
Note2:
DP ususally start i == 1, and always use (i - 1) in the loop... this is all because we are trying to get DP[i][j], which are 1 index more than length
*/
public class Solution {
public boolean isInterleave(String s1, String s2, String s3) {
if (s3 == null || (s1 == null && s2 == null) || s1.length() + s2.length() != s3.length()) {
return false;
}
boolean[][] DP = new boolean[s1.length() + 1][s2.length() + 1];
DP[0][0] = true; // empty s1 and s2 would be a working case

```
    //with just s1:
    for (int i = 1; i <= s1.length(); i++) {
        if (s3.charAt(i - 1) == s1.charAt(i - 1) && DP[i - 1][0]) {
            DP[i][0] = true;
        }
    }

    //with just s2:
    for (int j = 1; j <= s2.length(); j++) {
        if (s3.charAt(j - 1) == s2.charAt(j - 1) && DP[0][j - 1]) {
            DP[0][j] = true;
        }
    }

    for (int i = 1; i <= s1.length(); i++) {
        for (int j = 1; j <= s2.length(); j++) {
            if ((s3.charAt(i + j - 1) == s1.charAt(i - 1) && DP[i - 1][j])
                || (s3.charAt(i + j - 1) == s2.charAt(j - 1) && DP[i][j - 1])) {
                DP[i][j] = true;
            }
        }
    }

    return DP[s1.length()][s2.length()];
}

}

/*

Attempt1, Incorrect: tho, magically passed 91% of lintcode, by coincidence
This solution could goes on and on with s1, and failed at certain point when j == 0 does not fit in.
s1 = "sdfjas;dfjoisdu"
s2 = "dfnakd"
s3 = "sdfjas;dfjoisdf..." // Failed at that 'f' in s3

Thoughts:
DP[mxn]: loop through S1.length and S2.length, record DP[k] = true or false.
DP[k] = (S1(0~i) + S2(0 ~ j)) is leading S3: index of (xxx) == 0.

*/
public class Solution {
```

```
public boolean isInterleave(String s1, String s2, String s3) {
    if (s3 == null || (s1 == null && s2 == null) || s1.length() + s2.length() != s3.length()) {
        return false;
    }

    int i = 0;
    int j = 0;
    String base = "";
    for (int k = 0; k < s1.length()*s2.length() - 1; k++) {
        if (i < s1.length() || j < s2.length()) {
            if (i < s1.length() && s3.indexOf(base + s1.charAt(i)) == 0) {
                base += s1.charAt(i);
                i++;
            } else if (j < s2.length() && s3.indexOf(base + s2.charAt(j)) == 0) {
                base += s2.charAt(j);
                j++;
            } else {
                return false;
            }
        }
    }
    return true;
}
```

}

---

127. Intersection of Two Linked Lists.java (https://github.com/shawnfan/LintCode/blob/master/Java/Intersection%20of%20Two%20Linked%20Lists.java) Level: Easy

长短list，找重合点。
长度不同的话，切掉长的list那个的extra length。 那么起点一样后，重合点就会同时到达。

---

128. Interval Minimum Number.java (https://github.com/shawnfan/LintCode/blob/master/Java/Interval%20Minimum%20Number.java) Level: Medium

SegtmentTree, methods: Build, Query. 这题是在SegmentTreeNode里面存min.

类似的有存:max, sum, min

---

129. Interval Sum II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Interval%20Sum%20II.java) Level: Hard

SegmentTree大集合。记得几个Methods: Build, Query, Modify. 不难。只是要都记得不犯错:)

---

130. Interval Sum.java (https://github.com/shawnfan/LintCode/blob/master/Java/Interval%20Sum.java) Level: Medium

其实是segment tree 每个node上面加个sum。

记得Segment Tree methods: Build, Query

Note: 存在SegmentTreeNode里面的是sum. 其他题目可能是min,max ... or something else.

---

131. Invert Binary Tree.java (https://github.com/shawnfan/LintCode/blob/master/Java/Invert%20Binary%20Tree.java) Level: Easy

non-recursive: BFS with queue。 或者regular recurisve - divide and conquer.

---

132. Isomorphic Strings.java (https://github.com/shawnfan/LintCode/blob/master/Java/Isomorphic%20Strings.java) Level: Easy

HashMap 来确认match。有几种情况考虑:

 1. Match. 就是map.containsKey, map.containsValue, and char1 == char2. Perfect.

 2. Either Key not exist, or Value not exit. False;

 3. Both key and Value exist, but map.get(char1) != char2. Miss-match. False.

 4. None of Key or Value exist in HashMap. Then add the match.

---

133. Jump Game II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Jump%20Game%20II.java) Level: Hard

Greedy, 图解 http://www.cnblogs.com/lichen782/p/leetcode_Jump_Game_II.html (http://www.cnblogs.com/lichen782/p/leetcode_Jump_Game_II.html)

维护一个range, 是最远我们能走的.

index/i 是一步一步往前, 每次当 i <= range, 做一个while loop ， 在其中找最远能到的地方 maxRange

然后更新 range = maxRange

其中step也是跟index是一样, 一步一步走.

最后check的condition是，我们最远你能走的range >= nums.length - 1, 说明以最少的Step就到达了重点。Good.

---

134. Jump Game.java (https://github.com/shawnfan/LintCode/blob/master/Java/Jump%20Game.java) Level: Medium

给出步数，看能不能reach to end.

Status:
DP[i]: 在i点记录，i点之前的步数是否可以走到i点？ True of false.
其实j in [0~i)中间只需要一个能到达i 就好了
Function:
DP[i] = DP[j] && (j + A[j]), for all j in [0 ~ i)
Return:
DP[dp.length - 1];

---

135. Kth Largest Element.java (https://github.com/shawnfan/LintCode/blob/master/Java/Kth%20Largest%20Element.java) Level: Medium

用Quick Sort 里面partion的一部分。
partion的结果是那个low, 去找 low==nums.size() - k ， 也就是倒数第K个。
没找到继续partion recursively.

sort的过程是排一个从小到大的list. (同样的代码还可以好xth smallest，mid变成x就好)

Quick Sort:
每个iteration, 找一个pivot,然后从low,和high都和pivot作比较。
找到一个low>pivot, high<pivot, 也就可以swap了。
得到的low就是当下的partion point了

---

136. Kth Smallest Element in a BST.java (https://github.com/shawnfan/LintCode/blob/master/Java/Kth%20Smallest%20Element%20in%20a%20BST.java)Recursive 不难，然后稍微优化一下，确保rst.size() == k 时候，就可以return了。
Iterative 稍微难想点：先把最左边的add， pop() stack， 加上右边（如果存在）； 下一个轮回，如果又左孩子，又是一顿加。

---

137. Kth Smallest Number in Sorted Matrix.java (https://github.com/shawnfan/LintCode/blob/master/Java/Kth%20Smallest%20Number%20in%20Sorted%20Matrix.java) Level: Medium

和Merge K sorted Array/ List 类似：使用PriorityQueue。

因为Array的element无法直接找到next,所以用一个class Node 存value, x,y positions.

/*
Find the kth smallest number in at row and column sorted matrix.

Have you met this question in a real interview? Yes
Example
Given k = 4 and a matrix:

[
[1 ,5 ,7],
[3 ,7 ,8],
[4 ,8 ,9],
]
return 5

Challenge
O(k log n), n is the maximal number in width and height.

Tags Expand
Heap Priority Queue Matrix

*/

```java
//PriorityQueue store front node. (Class Node), then output the kth in queue.
public class Solution {
class Node {
int val;
int x,y;
public Node(int val, int x, int y){
this.val = val;
this.x = x;
this.y = y;
}
}
public int kthSmallest(int[][] matrix, int k) {
if (matrix == null || matrix[0] == null || matrix.length == 0
|| matrix[0].length == 0 || k <= 0) {
return -1;
}
```

```
        //Init queue
        PriorityQueue<Node> queue = new PriorityQueue<Node>(k,
            new Comparator<Node>(){
                public int compare(Node a, Node b) {
                    return a.val - b.val;
                }
            });

        for (int i = 0; i < matrix.length; i++) {
            if (matrix[i].length > 0) {
                queue.offer(new Node(matrix[i][0], i, 0));
            }
        }

        //Find kth
        while (!queue.isEmpty()) {
            Node node = queue.poll();
            if(k == 1) {
                return node.val;
            }
            int x = node.x;
            int y = node.y;
            if (y < matrix[x].length - 1) {
                queue.offer(new Node(matrix[x][y+1], x, y+1));
            }
            k--;
        }

        return -1;
    }
}
```

138. Kth Smallest Sum In Two Sorted Arrays.java
(https://github.com/shawnfan/LintCode/blob/master/Java/Kth%20Smallest%20Sum%20In%20Two%20Sorted%20Arrays.java) Level: Hard

用priority queue. 每次把最小的展开，移位。分别x+1,或者y+1:
因为当下的Min里面x,y都是最小的。所以下一个最小的不是（x+1,y），就是（x,y+1）。

每次就poll（）一个，放2个新candidate进去就好了。
注意，这样的做法会用重复，比如例子（7,4）会出现两次。用一个HashSet挡一下。

注意，HashSet的唯一性，用一个"x,y"的string就可以代为解决。

139. Largest Number.java (https://github.com/shawnfan/LintCode/blob/master/Java/Largest%20Number.java)Given a list of non negative integers, arrange them such that they form the largest number.

Example
Given [1, 20, 23, 4, 8], the largest formed number is 8423201.

Note
The result may be very large, so you need to return a string instead of an integer.

Tags Expand
Sort

Thoughts:
Use a comparator with String.comareTo, then uset Arrays.sort(...)

*/

```java
class CustomComparator implements Comparator<String> {
public int compare(String s1, String s2) {
return (s2 + s1).compareTo(s1 + s2);
}
}
public class Solution {
/* @param num: A list of non negative integers
@return: A string /
public String largestNumber(int[] num) {
if (num == null || num.length == 0) {
return "";
}
String[] strs = new String[num.length];
for (int i = 0; i < num.length; i++) {
strs[i] = num[i] + "";
}
Arrays.sort(strs, new CustomComparator());
StringBuffer sb= new StringBuffer();
for (int i = 0; i < num.length; i++) {
sb.append(strs[i]);
}
String rst = sb.toString();
```

```
if (rst.charAt(0) == '0') {
return "0";
}
return rst;
}
}
```

---

140. Largest Rectangle in Histogram.java (https://github.com/shawnfan/LintCode/blob/master/Java/Largest%20Rectangle%20in%20Histogram.java)Example
Given height = [2,1,5,6,2,3],

return 10.

Tags Expand
Array Stack

Thinking Process:
///TODO: missing thinking process for Largest Rectangle in Histogram

*/

public class Solution {
/**

```
 * @param height: A list of integer
 * @return: The area of largest rectangle in the histogram
 */
public int largestRectangleArea(int[] height) {
    if (height == null || height.length == 0) {
        return 0;
    }
    Stack<Integer> stack = new Stack<Integer>();
    int max = 0;
    for (int i = 0; i <= height.length; i++) {
        int current = (i == height.length) ? -1 : height[i];
        while (!stack.empty() && current <= height[stack.peek()]) {
            int h = height[stack.pop()];
            int w = stack.empty() ? i : i - stack.peek() - 1;
            max = Math.max(max, w * h);
        }
        stack.push(i);
    }
    return max;
}
```

}

141. Last Position of Target.java (https://github.com/shawnfan/LintCode/blob/master/Java/Last%20Position%20of%20Target.java)

142. Length of Last Word.java (https://github.com/shawnfan/LintCode/blob/master/Java/Length%20of%20Last%20Word.java)Given a string s consists of upper/lower-case alphabets and empty space characters ' ', return the length of last word in the string.

If the last word does not exist, return 0.

Example
Given s = "Hello World", return 5.

Note
A word is defined as a character sequence consists of non-space characters only.

Tags Expand
String

Thoughts:

 1. Split by space
 2. return last word's length

Note: Java split: have to add '\' in order to pass the key word.
*/

public class Solution {
/**

```
 * @param s A string
 * @return the length of last word
 */
public int lengthOfLastWord(String s) {
    if (s == null || s.length() == 0) {
        return 0;
    }
    String[] arr = s.split("\\ ");
    String lastWord = arr[arr.length - 1];

        return lastWord.length();
}
```

}

---

143. Letter Combinations of a Phone Number.java (https://github.com/shawnfan/LintCode/blob/master/Java/Letter%20Combinations%20of%20a%20Phone%20Number.java) Level: Medium

方法1: Iterative with BFS using queue.

方法2: Recursively adding chars per digit

---

144. Linked List Cycle II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Linked%20List%20Cycle%20II.java)

O(1)要首先break while loop when there is a slow==fast
然后，然后就有个我不懂得地方：

当head == slow.next时候， head就是cycle starting point.
也就是说，当slow 移动到了那个回溯点，slow.next那个点就刚好是head的那个点...

这个可能要写一写，装一装，证明证明才行...不是特别清楚。

---

145. Linked List Cycle.java (https://github.com/shawnfan/LintCode/blob/master/Java/Linked%20List%20Cycle.java)那个时候其实slow.val = fast.val.

O(n):用HashMap，一直add elements. 如果有重复，那么很显然是有Cycle le

---

146. Longest Common Prefix.java (https://github.com/shawnfan/LintCode/blob/master/Java/Longest%20Common%20Prefix.java) Level: Medium

Nested loop, 每一次比较所有string 同位是否相等。

相等，append string. 不等，return.

---

147. Longest Common Subsequence.java (https://github.com/shawnfan/LintCode/blob/master/Java/Longest%20Common%20Subsequence.java)Given two strings, find the longest comment subsequence (LCS).

Your code should return the length of LCS.

Example
For "ABCD" and "EDCA", the LCS is "A" (or D or C), return 1

For "ABCD" and "EACB", the LCS is "AC", return 2

Clarification
What's the definition of Longest Common Subsequence?

```
* The longest common subsequence (LCS) problem is to find the longest subsequence common to all sequences in a set of sequences (often just two). (Note that a subsequence is different from a substring, for 

* https://en.wikipedia.org/wiki/Longest_common_subsequence_problem
```

Tags Expand
LintCode Copyright Longest Common Subsequence Dynamic Programming

Thinking process:
Using DP.
check[i][j] means: the length of longest common subsequnce between A(0 ~ i) and B(0 ~ j).
Then there are two ways to reach check[i][j]:

1. A(i-1) == B(j - 1), then check[i][j] = check[i - 1][j - 1] + 1;
2. A(i-1) != B(j - 1), then pick the max between (i-1,j) , (i,j-1) and (i, j)
   Note: check[][] is initialized with all 0's. Index (0,0) is used as starting 0.

   /
   public class Solution {
   /*

   ○ @param A, B: Two strings.
   ○ @return: The length of longest common subsequence of A and B.
     */
     public int longestCommonSubsequence(String A, String B) {
```

```
if (A == null || B == null || A.length() == 0 || B.length() == 0) {
    return 0;
}
```

```
int[][] check = new int[A.length() + 1][B.length() + 1];
for (int i = 1; i <= A.length(); i++) {
    for (int j = 1; j <= B.length(); j++) {
        if (A.charAt(i - 1) == B.charAt(j - 1)) {
            check[i][j] = check[i - 1][j - 1] + 1;
        } else {
            check[i][j] = Math.max(check[i][j], check[i - 1][j]);
            check[i][j] = Math.max(check[i][j], check[i][j - 1]);
        }
    }
}
```

```
return check[A.length()][B.length()];
}
}
```

---

148. Longest Common Substring.java (https://github.com/shawnfan/LintCode/blob/master/Java/Longest%20Common%20Substring.java)Given two strings, find the longest common substring.

Return the length of it.

Example
Given A = "ABCD", B = "CBCE", return 2.

Note
The characters in substring should occur continuously in original string. This is different with subsequence.

Challenge
O(n x m) time and memory.

Tags Expand
LintCode Copyright Longest Common Subsequence Dynamic Programming

Thoughts:

1. Compare all i X j.

2. Use a D[i][j] to mark the amount of common substring based on D[i − 1][j −1]. Could be 0.

3. track max length

NOTE1: create 2D array that's [N + 1][M + 1] because we want to hold D[n][M] in the 2d array
NOTE2: be carefule with init index 0's

*/

public class Solution {
/**

```
  * @param A, B: Two string.
  * @return: the length of the longest common substring.
  */
 public int longestCommonSubstring(String A, String B) {
     if (A == null || B == null || A.length() == 0 || B.length() == 0) {
         return 0;
     }
     int [][] D = new int[A.length() + 1][B.length() + 1];
     int max = 0;
     for (int i = 0; i <= A.length(); i++) {
         for(int j = 0; j <= B.length(); j++) {
             if (i == 0 || j == 0) {
                 D[i][j] = 0;
             } else {
                 if (A.charAt(i - 1) == B.charAt(j - 1)) {
                     D[i][j] = D[i - 1][j - 1] + 1;
                 } else {
                     D[i][j] = 0;
                 }
                 max = Math.max(max, D[i][j]);
             }
         }
     }
     return max;
 }
```

}

---

149. Longest Consecutive Sequence.java (https://github.com/shawnfan/LintCode/blob/master/Java/Longest%20Consecutive%20Sequence.java)Given an unsorted array of integers, find the length of the longest consecutive elements sequence.

For example,

Given [100, 4, 200, 1, 3, 2],

The longest consecutive elements sequence is [1, 2, 3, 4]. Return its length: 4.

Your algorithm should run in O(n) complexity.

Hide Tags Array

Thinking process:

1. This problem can be done using sorting, but time complexity of sorting is O(nlogn). This problem requires O(n).

2. Want to check if a number's left and right is consecutive to itself, but cannot do it due to the given unsorted array: think about a Hashmap.

3. HashMap(Key, Value) = (the number itself, boolean: have been counted or not). If you count a number as a consecutive, you only need to count it once.

4. How HashMap works:

   when checking a number's consecutive, look at number--, number++, see if they are in the HashMap. If exist, means consecutive.

   If a number exist in the hashmap and its value is 'true', then we need to skip this number beacuse it has been checked.

5. Track the total number consecutives of 1 perticular number, compare it with the maxL. Save the Math.max to maxL.

6. Depending on the problem, we can store a consecutive sequence or simply just its length: maxL. This problem wants the maxL.

   */

```java
public class Solution {
public int longestConsecutive(int[] num) {
if (num == null || num.length == 0) {
return 0;
}
int maxL = 1;
HashMap<Integer, Boolean> history = new HashMap<Integer, Boolean>();
for (int i : num) {
history.put(i, false);
}
for (int i : num) {
if (history.get(i)) {
continue;
}
//check ++ side
int temp = i;
int total = 1;
while (history.containsKey(temp + 1)) {
total++;
```

```java
            temp++;
            history.put(temp, true);
        }
        //check -- side
        temp = i;
        while (history.containsKey(temp - 1)) {
            total++;
            temp--;
            history.put(temp, true);
        }
        maxL = Math.max(maxL, total);
    }
    return maxL;
    }
}


/*
10.19.2015
Thougths:

 1. sort

 2. use a 'count' and 'max' to keep track of consecutive elements

 3. one-pass

Note:
Take care of equal numbers: skip/continue those


*/

public class Solution {
/**
```

```
 * @param nums: A list of integers
 * @return an integer
 */
public int longestConsecutive(int[] num) {
    if (num == null || num.length == 0) {
        return 0;
    }
    if (num.length == 1) {
        return 1;
    }
    int count = 1;
    int max = 1;
    Arrays.sort(num);
    for (int i = 1; i < num.length; i++) {
        if (num[i - 1] == num[i]) {
            continue;
        } else if (num[i - 1] + 1 == num[i]) {
            count++;
            max = Math.max(count, max);
        } else {
            count = 1;
        }
    }
    return max;
}
```

}

---

150. Longest Increasing Continuous subsequence II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Longest%20Increasing%20Continuous%20subsequence%20II.java)O(mn) runtime because each spot will be marked once visited.

这个题目的简单版本一个array的例子：从简单题目开始想DP会简单一点。每个位置，都是从其他位置（上下左右）来的dpValue + 1. 如果啥也没有的时候，init state 其实都是1，就一个数字，不增不减嘛。

---

151. Longest Increasing Continuous subsequence.java (https://github.com/shawnfan/LintCode/blob/master/Java/Longest%20Increasing%20Continuous%20subsequence.java)O(1)是用了两个int来存：每次到i点时，i点满足条件或不满足条件所有的longestIncreasingContinuousSubsequence.

特点：返跑一回，ans还是继续和left轮的ans作比较；求的所有情况的最大值嘛。

---

152. Longest Increasing Subsequence.java (https://github.com/shawnfan/LintCode/blob/master/Java/Longest%20Increasing%20Subsequence.java)每次都考虑o~i的所有情况。所以double for loop

---

153. Longest Palindromic Substring.java (https://github.com/shawnfan/LintCode/blob/master/Java/Longest%20Palindromic%20Substring.java) Level: Medium

方法1: 从中间劈开. 遍历i，从n个不同的点劈开：每次劈开都看是否可以从劈开出作为palindromic的中点延伸。
Worst case: 整个string都是相同字符，time complexity变成：1 + 2 + 3　+　. . .　+ n = O(n^2)

方法2: 穷举double for loop. O(n^2)

---

154. Longest Substring with At Most K Distinct Characters.java
(https://github.com/shawnfan/LintCode/blob/master/Java/Longest%20Substring%20with%20At%20Most%20K%20Distinct%20Characters.java) Level: Medium

大清洗 O(nk)
map.size一旦>k，要把longest string最开头（marked by pointer:start）的那个char抹掉
一旦某一个char要被清除，所以在这个char 的1st and last appearance之间的char都要被清洗from map

---

155. Longest Substring Without Repeating Characters.java
(https://github.com/shawnfan/LintCode/blob/master/Java/Longest%20Substring%20Without%20Repeating%20Characters.java) Level: Medium

方法2:用两个pointer, head和i.
HashMap<Char, Integer>: <character, last occurance index>
head从index 0 开始。若没有重复char, 每次只有for loop的i++。每次取substring[head,i]作为最新的string.
一旦有重复，那么意味着，从重复的老的那个index要往后加一格开始。所以head = map.get(i) + 1.

注意：head很可能被退回到很早的地方，比如abbbbbba,当遇到第二个a，head竟然变成了 head = 0+1 = 1.
当然这是不对的，所以head要确保一直增长，不回溯。

方法1：只要有non-existing char就count++. 一旦有重复char:
i = 新出现重复Char的位置.
重新init HashMap, count.

这个方法每次都把map打碎重来, 是可以的，也没什么不好。就是在for里面改i，自己觉得不太顺.方法二可能顺一点。

---

156. Longest Words.java (https://github.com/shawnfan/LintCode/blob/master/Java/Longest%20Words.java)Given a dictionary, find all of the longest words in the dictionary.

Example
Given

```
{
"dog",
"google",
"facebook",
"internationalization",
"blabla"
}
the longest words are(is) ["internationalization"].

Given

{
"like",
"love",
"hate",
"yes"
}
the longest words are ["like", "love", "hate"].

Challenge
It's easy to solve it in two passes, can you do it in one pass?

Tags Expand
Enumeration String LintCode Copyright

Thoughts:
Two pass: 1st, get longest length. 2nd pass, get all words.

One pass:

 1. Use hashmap: <lengthOfString, ArrayList<String>>
 2. keep track of the longest length

Review:
Map: put, get
ArrayList: add
We can get a value from map, and change directly on it, if that's an object (basically refer to the original object)
*/
```

class Solution {
/**

```
 * @param dictionary: an array of strings
 * @return: an arraylist of strings
 */
ArrayList<String> longestWords(String[] dictionary) {
    if (dictionary == null || dictionary.length == 0) {
        return null;
    }
    HashMap<Integer, ArrayList<String>> map = new HashMap<Integer, ArrayList<String>>();
    int longestLength = 0;

    for (int i = 0; i < dictionary.length; i++) {
        int strLength = dictionary[i].length();
        if (map.containsKey(strLength)) {
            map.get(strLength).add(dictionary[i]);
        } else {
            ArrayList<String> list = new ArrayList<String>();
            list.add(dictionary[i]);
            map.put(strLength, list);
        }
        longestLength = strLength > longestLength ? strLength : longestLength;
    }
    return map.get(longestLength);
}
```

};

---

157. Lowest Common Ancestor II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Lowest%20Common%20Ancestor%20II.java) Level: Easy

这个题有个奇葩的地方，每个node还有一个parent。 所以可以自底向上.

1. 曾经做的hashset的优化，找到的都存hashset. exist就return那个duplicate.

1. 普通做法：2 lists。
   自底向上。利用parent往root方向返回。

注意：无法从root去直接搜target node 而做成两个list. 因为根本不是Binary Search Tree！

---

158.   Lowest   Common   Ancestor   of   a   Binary   Search   Tree.java
(https://github.com/shawnfan/LintCode/blob/master/Java/Lowest%20Common%20Ancestor%20of%20a%20Binary%20Search%20Tree.java) Level: Medium

利用 BST的性质，可以直接搜到target node，而做成两个长度不一定相等的list。然后很简单找到LCA

---

159. Lowest Common Ancestor.java (https://github.com/shawnfan/LintCode/blob/master/Java/Lowest%20Common%20Ancestor.java) Level: Easy

普通的Binary Tree，node child 自顶向下蔓延。

方法1：O(1) sapce O(h). Recursive. 循环的截点是：
当root == null或者 A B 任何一个在findLCA底部被找到了(root== A || root == B)，那么就return 这个root.

三种情况：

 1. A,B都找到，那么这个level的node就是其中一层的parent。其实，最先recursively return到的那个，就是最底的LCA parent.
 2. A 或者 B 找到，那就还没有公共parent,return 非null得那个。
 3. A B 都null, 那就找错了没有呗, return null

//无法找到target element, 因为不是Binary Search Tree
//[Not Working]：O(n) space O(h) time。把两条线binary search出来。找第一个不同的parent. 代码长。 Iterative

---

160. LRU Cache.java (https://github.com/shawnfan/LintCode/blob/master/Java/LRU%20Cache.java) Level: Hard

timeout method, 天真的来了一个O(n) 的解法，结果果然timeout.
一个map<key,value>存数值。一个queue<key>来存排位。
每次有更新，就把最新的放在末尾；每次超过capaticity,就把大头干掉。很简单嘛，但是跑起来太久，失败了。

于是就来了第二个做法。其实还是跟方法一是类似的。
用了一个特别的双向的LinkNode，有了head和tail，这样就大大加快了速度。
主要加快的就是那个'更新排位'的过程，过去我是O(n),现在O(1)就好了。

巧妙点：

 1. head和tail特别巧妙：除掉头和尾，和加上头和尾，就都特别快。
 2. 用双向的pointer: pre和next, 当需要除掉任何一个node的时候，只要知道要除掉哪一个，
    直接把node.pre和node.next耐心连起来就好了，node就自然而然的断开不要了。

一旦知道怎么解决了，就不是很特别，并不是难写的算法:
moveToHead()
insertHead()
remove()

161. Majority Number II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Majority%20Number%20II.java) Level: Medium

分三份：a b c考虑。若a, countA++, 或b, countB++，或c，countA--,countB--.

最后出现的两个count>0的a和b,自然是potentially大于1/3的。其中有一个大于1/3.

比较a和b哪个大，就return哪一个。

---

162. Majority Number III.java (https://github.com/shawnfan/LintCode/blob/master/Java/Majority%20Number%20III.java) Level: Medium

与其他Majority Number一样。

出现次数多余1/k，就要分成k份count occurance.用HashMap。 存在的+1；不存在map里的，分情况:
若map.size() == k,说明candidate都满了，要在map里把所有现存的都-1；
若map.size() < k, 说明该加新candidate，那么map.put(xxx, 1);

最后在HashMap里找出所留下的occurance最大的那个数。

---

163. Majority Number.java (https://github.com/shawnfan/LintCode/blob/master/Java/Majority%20Number.java) Level: Easy

Majority Number是指超半数。任何超半数，都可以用0和1count：是某个number，+1；不是这个number,-1.

注意：assume valid input, 是一定有一个majority number的。否则此法不成。[1,1,1,2,2,2,3]是个invalid input,结果是3，当然也错了。

Majority Number II，超1/3，那么就分三份处理，countA, countB来计算最多出现的两个。

Majority Number III, 超1/k, 那么自然分k份。这里用到 HashMap。

---

164. Matrix Zigzag Traversal.java (https://github.com/shawnfan/LintCode/blob/master/Java/Matrix%20Zigzag%20Traversal.java) Level: Easy

分析4个step:right, left-bottom,down,right-up
implement时注意index.有点耐心

---

165. Max Tree.java (https://github.com/shawnfan/LintCode/blob/master/Java/Max%20Tree.java) Level: Hard

Should memorize MaxTree. 依次类推，会做Min-Tree, Expression Tree

Stack里，最大的值在下面。利用此性质，有这样几个step:

1

把所有小于curr node的，全Pop出来, while loop, keep it going.

最后pop出的这个小于Curr的node：它同时也是stack里面pop出来小于curr的最大的一个，最接近curr大小。（因为这个stack最大值靠下面）

把这个最大的小于curr的node放在curr.left.

2

那么，接下去stack里面的一定是大于curr：

那就变成curr的left parent. set stack.peek().right = curr.

3

结尾：stack底部一定是最大的那个，也就是max tree的头。

---

166. Maximal Square.java (https://github.com/shawnfan/LintCode/blob/master/Java/Maximal%20Square.java) Level: Medium

DP问题

从边长为2的正方形看起，看左上角的那个点。

如何确定是个正方形？首先看左上点是不是1，然后看右边，右下，下面的点是不是1.

DP就是根据这个特征想出来。dp[i,j]: 从右下往左上推算，包括当前点在内的所能找到的最大边长。

注意dp[i,j]被右边，右下，下面三点的最短点所限制。这就是fn.

Init：

把右边，下边两个边缘init一遍，存matrix在这两条边上的值，代表的意思也就是dp[i][j]在这些点上的初始值:变成为1 or 0.

---

167. Maximum Depth of Binary Tree.java (https://github.com/shawnfan/LintCode/blob/master/Java/Maximum%20Depth%20of%20Binary%20Tree.java) Level: Easy

DFS: Divide and conquer. 维持一个最大值。

---

168. Maximum Product Subarray.java (https://github.com/shawnfan/LintCode/blob/master/Java/Maximum%20Product%20Subarray.java)Find the contiguous subarray within an array (containing at least one number) which has the largest product.

Example
For example, given the array [2,3,-2,4], the contiguous subarray [2,3] has the largest product = 6.

Tags Expand

Dynamic Programming Subarray

/

/

Attempt2: Use a max array and a min array. (http://www.jiuzhang.com/solutions/maximum-product-subarray/ (http://www.jiuzhang.com/solutions/maximum-product-subarray/))

This is similar to my original attempt1, but saves a lot memory space.

1. Max array is always positive, Min array is always negative. Use these 2 arrays to keep track of largest positive number and smallest negative number
2. When current nums[i] > 0, use max[i – 1] * nums[i].
3. When current nums[i] < 0, user min[i – 1] * nums[i];
4. Don't for get to calculate both max and min for each i, for next iteration to use.

In either case, we will produce largest possible product.

Trick: depending on nums[i] is positive or negative, calculate differently ...

*/

```
public class Solution {
public int maxProduct(int[] nums) {
if (nums == null || nums.length == 0) {
return 0;
}
int[] max = new int[nums.length];
int[] min = new int[nums.length];
max[0] = nums[0];
min[0] = nums[0];
int rst = max[0];
for (int i = 1; i < nums.length; i++) {
if (nums[i] > 0) {
max[i] = Math.max(nums[i], max[i - 1] nums[i]);//the nums[i] could just be the best option
min[i] = Math.min(nums[i], min[i - 1] nums[i]);
} else {
max[i] = Math.max(nums[i], min[i - 1] nums[i]);
min[i] = Math.min(nums[i], max[i - 1] nums[i]);
}
rst = Math.max(rst, max[i]);
}
```

```
    return rst;
  }
}
```

```
/*
Attempt1 thoughts:
97% correct. However, this exceeds memory, basically the DP[][] is too large.
Draw a 2D array:
Row: Start from a number ROW[i], what contiguous value can we get:
```

```
   0   1   2   3
  ------------
   2   3   -2  4
```

```
0| 2 2 6 -12 -48
1| 3 x x -6 -24
2| -2 x x x -8
3| 4 x x x
```

Look, according to the rules of (contiguous subarray), we can't do Row[i]xRow[i], so we have to do: Row[i]xROW[i+1]xROW[i+2]...etc
Goal: find the max in DP

 1. Define DP[0][0] = nums[0];
 2. DP[i][j] = DP[i][j - 1] * nums[j]
 3. And we keep track of the max value

Note: j will always > i, so cases that i >= j are not necessary.
*/

```
public class Solution {
/**
```

```
 * @param nums: an array of integers
 * @return: an integer
 */
public int maxProduct(int[] nums) {
    if (nums == null || nums.length == 0) {
            return 0;
        }
        int[][] DP = new int[nums.length][nums.length];
        DP[0][0] = nums[0];
        int max = DP[0][0];

        for (int i = 0; i < nums.length; i++) {
            for (int j = 1; j < nums.length; j++) {
                if (i == j) {
                    DP[i][j] = nums[j];
                }
                if (j > i) {
                    if (DP[i][j - 1] == 0) {
                        DP[i][j] = nums[j];
                    } else {
                        DP[i][j] = DP[i][j - 1] * nums[j];
                    }
                    max = Math.max(max, DP[i][j]);
                }
                max = Math.max(max, nums[j]);
            }
        }
    return max;
}

}
```

---

169. Maximum Subarray III.java (https://github.com/shawnfan/LintCode/blob/master/Java/Maximum%20Subarray%20III.java)
Given an array of integers and a number k, find k non-overlapping subarrays which have the largest sum.

The number in each subarray should be contiguous.

Return the largest sum.

Have you met this question in a real interview? Yes
Example
Given [-1,4,-2,3,-2,3], k=2, return 8

Note
The subarray should contain at least one number

Tags Expand

LintCode Copyright Dynamic Programming Subarray Array

*/

/

NOT DONE /

public class Solution {

/**

```
 * @param nums: A list of integers
 * @param k: An integer denote to find k non-overlapping subarrays
 * @return: An integer denote the sum of max k non-overlapping subarrays
 */
public int maxSubArray(ArrayList<Integer> nums, int k) {
    // write your code
}
```

}

---

170. MaximumSubarray.java (https://github.com/shawnfan/LintCode/blob/master/Java/MaximumSubarray.java)然后presum[j] – presum[i– 1] 就是 (i,j)之间的和。

---

171. MaximumSubarrayII.java (https://github.com/shawnfan/LintCode/blob/master/Java/MaximumSubarrayII.java)注意：右边算prefix sum， 看上去好像是什么postfix sum？
其实不是。其实都和prefix一样。
我们需要的那部分prefix sum，其实就是一段数字的总和。
所以从右边累计上来的。也是一样可以的。

---

172. Median of two Sorted Arrays.java (https://github.com/shawnfan/LintCode/blob/master/Java/Median%20of%20two%20Sorted%20Arrays.java) Level: Hard

Not done

---

173. Median.java (https://github.com/shawnfan/LintCode/blob/master/Java/Median.java)Given a unsorted array with integers, find the median of it.

A median is the middle number of the array after it is sorted.

If there are even numbers in the array, return the N/2-th number after sorted.

Example
Given [4, 5, 1, 2, 3], return 3

Given [7, 9, 4, 5], return 5

Challenge
O(n) time.

Tags Expand
LintCode Copyright Quick Sort Array

/
/
Recap 12.09.2015.
O(n) means just run through it. It's similar to Partition array: it tries to split the list into 2 parts, and find the pivot.
*/

/*
Thoughts:
Use standard quick sort, but the goal is to look for the middle point.

 1. Get middle point: remember to -1 because we are looking for position, rather than length.

 2. Increase low pointer until find a point >= piviot

 3. Decrease high pointer until find a point < poviot

 4. Swap the low and high: this set the first value greather than pivot to the right, and first avlue less than pivot to the left.

 5. after low and high pointer meets, swap low with the piviot: simply because pivot should be the break point of low and high

 6. at the end, the low sould be the middle point, which is the point we are looking for. return corresponding recursive helper.

/
public class Solution {
/*

```java
 * @param nums: A list of integers.
 * @return: An integer denotes the middle number of the array.
 */
public int median(int[] nums) {
    if (nums == null || nums.length == 0) {
        return 0;
    }
    if (nums.length % 2 == 0) {
        return helper(nums, 0, nums.length - 1, nums.length/2 - 1);
    } else {
        return helper(nums, 0, nums.length - 1, nums.length/2);
    }
}

public void swap(int[] nums, int x, int y){
    int temp = nums[x];
    nums[x] = nums[y];
    nums[y] = temp;
}

public int helper(int[] nums, int start, int end, int mid) {
    int pivot = end;
    int num = nums[pivot];
    int low = start;
    int high = end;
    while (low < high) {
        while(low < high && nums[low] < num) {
            low++;
        }
        while(low < high && nums[high] >= num) {
            high--;
        }
        swap(nums, low, high);
    }
    swap(nums, low, pivot);
    if (low == mid) {
        return nums[low];
    } else if (low < mid) {
        return helper(nums, low + 1, end, mid);
    } else {
        return helper(nums, start, low - 1, mid);
    }
}

}
```

174. Meeting Rooms II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Meeting%20Rooms%20II.java) Level: Medium

方法1:PriorityQueue + 一个Class来解决。O(nlogn)

方法2:这里有尝试了一下用一个sorted Array + HashMap： 也还行，但是handle edge的时候,HashMap 要小心，因为相同时间start和end的map key 就会重复了。

---

175. Meeting Rooms.java (https://github.com/shawnfan/LintCode/blob/master/Java/Meeting%20Rooms.java) Level: Easy

Scan line, class Point{pos, flag}, PriorityQueue排序。计算count

注意接头点要考虑所有开会结会的情况，不要恰巧漏掉相接的点。
开会的是超人。瞬间移动接上下一个会议。

---

176. Merge Intervals.java (https://github.com/shawnfan/LintCode/blob/master/Java/Merge%20Intervals.java) Level: Easy

方法1：O(nlogn)
扫描线+Count无敌手。注意start end把interval给合起来。
count==0的时候，就是每次start end双数抵消的时候，就应该是一个interval的开头/结尾。写个例子就知道了。

空间：O(2n) -> O(n)
时间,priorityqueue: O(nlogn)

记得怎么写comparator

在 LeetCode里面，Scan line比方法2要快很多.

方法2：
Collections.sort() on interval.start之后，试着跑一遍，按照merge的需求，把需要merge的地方续好，然后减掉多余的interval就好。

(不知为何LeetCode把Merge Interval, Insert Interval 标为Hard)

Collections.sort(..., new comparator): sort by Interval.start.

画两个相连的Interval， prev, curr:
prev只有 prev.end覆盖了 curr.start， 才需要merge. 那么比较一下, marege.
记得如果merge, 一定要list.remove(i), 并且i--， 因为改变了List的大小。

若没有重合，就继续iteration: prev = curr. move on.

/*
new Comparator<Object>(){
public int compare(obj1, obj2) {

```
    return obj1.x - obj2.x;
}


    }
```

*/

---

177. Merge k Sorted Arrays.java (https://github.com/shawnfan/LintCode/blob/master/Java/Merge%20k%20Sorted%20Arrays.java) Level: Medium

由Merge k sorted list启发。用PriorityQueue,存那k个首发element。

PriorityQueue需要存储单位。自己建一个Class Node 存val, x,y index。
因为array里没有 'next' pointer，只能存x,y来推next element

---

178. Merge k Sorted Lists.java (https://github.com/shawnfan/LintCode/blob/master/Java/Merge%20k%20Sorted%20Lists.java) Level: Medium

用Priorityqueue来排列所有list的leading node.

记得k lists 需要是已经sort好的。

时间：n*O(logk)
PriorityQueue: logk

这个题目可以有好几个衍生：
比如，如果k很大，一个机器上放不下所有的k list怎么办？
比如，如果Merge起来的很长，一个机器上放不下怎么办？

---

179. Merge Sorted Array II.java

(https://github.com/shawnfan/LintCode/blob/master/Java/Merge%20Sorted%20Array%20II.java)

180. Merge Sorted Array.java (https://github.com/shawnfan/LintCode/blob/master/Java/Merge%20Sorted%20Array.java) Level: Easy

A够长，那么可以从A的尾部开始加新元素。
注意，从尾部，是大数字优先的。

---

181. Merge Two Sorted List.java (https://github.com/shawnfan/LintCode/blob/master/Java/Merge%20Two%20Sorted%20List.java) Level: Easy

小的放前。每次比head大小。
while过后，把没完的list一口气接上。

一开始建一个node用来跑路, 每次都存node.next = xxx。存一个dummy。用来return dummy.next.

---

182. Merge Two Sorted Lists.java (https://github.com/shawnfan/LintCode/blob/master/Java/Merge%20Two%20Sorted%20Lists.java)Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

Tags: Linked List
Similar Problems: (H) Merge k Sorted Lists, (E) Merge Sorted Array, (M) Sort List, (M) Shortest Word Distance II

*/

/
Thouhts:
Loop throgh both list. Make sure to check the border cases /

/**

- Definition for singly-linked list.
- public class ListNode {
- int val;
- ListNode next;
- ListNode(int x) { val = x; }
- }
   */
   public class Solution {
   public ListNode mergeTwoLists(ListNode l1, ListNode l2) {

```
        if (l1 == null && l2 == null) {
            return null;
        }
        ListNode node = new ListNode(0);
        ListNode dummy = node;
        while (l1 != null || l2 != null) {
            if (l1 == null) {
                node.next = l2;
                break;
            } else if (l2 == null) {
                node.next = l1;
                break;
            } else {
                if (l1.val < l2.val) {
                    node.next = l1;
                    l1 = l1.next;
                } else {
                    node.next = l2;
                    l2 = l2.next;
                }
                node = node.next;
            }
        }//end while
        return dummy.next;

    }
}
```

183. Middle of Linked List.java (https://github.com/shawnfan/LintCode/blob/master/Java/Middle%20of%20Linked%20List.java)

不在乎slow是不是到底，因为fast肯定先到。
确保fast, fast.next不是Null就好

return slow

184. Min Stack.java (https://github.com/shawnfan/LintCode/blob/master/Java/Min%20Stack.java) Level: Easy

双Stack：一个正常stack，另一个minStack存当下level最小值. 注意维护minStack的变化

另外. 如果要maxStack，也是类似做法

185. Minimum Height Trees.java (https://github.com/shawnfan/LintCode/blob/master/Java/Minimum%20Height%20Trees.java)For a undirected graph with tree characteristics, we can choose any node as the root. The result graph is then a rooted tree. Among all possible rooted trees, those with minimum height are called minimum height trees (MHTs). Given such a graph, write a function to find all the MHTs and return a list of their root labels.

Format

The graph contains n nodes which are labeled from 0 to n − 1. You will be given the number n and a list of undirected edges (each edge is a pair of labels).

You can assume that no duplicate edges will appear in edges. Since all edges are undirected, [0, 1] is the same as [1, 0] and thus will not appear together in edges.

Example 1:

Given n = 4, edges = [[1, 0], [1, 2], [1, 3]]

```
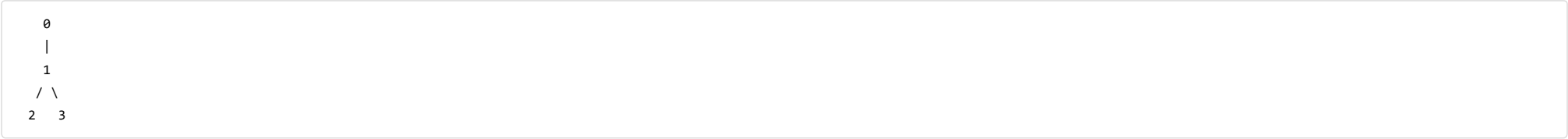    0
    |
    1
   / \
  2   3
```

return [1]

Example 2:

Given n = 6, edges = [[0, 3], [1, 3], [2, 3], [4, 3], [5, 4]]

```
 0  1  2
  \ | /
    3
    |
    4
    |
    5
```

return [3, 4]

Show Hint
Note:

(1) According to the definition of tree on Wikipedia: "a tree is an undirected graph in which any two vertices are connected by exactly one path. In other words, any connected graph without simple cycles is a tree."

(2) The height of a rooted tree is the number of edges on the longest downward path between the root and a leaf.

Credits:
Special thanks to @peisi for adding this problem and creating all test cases.

*/

/*
Starting from leaf with depth == 1,

remove all leaf, and the edge

Till the end, whatever node left, should be the root.

```
* WHen there is only 1,2 nodes remaining. that's the rst.

 Put Node in  HahsMap<node, list of neighbor node>

 Iterative over map till map.size() <= 2

 border n == 2,1, just returl rst.
 edges == null, return null.
 edges.length == 1, reutrn list
```

*/

public class Solution {
public List<Integer> findMinHeightTrees(int n, int[][] edges) {
List<Integer> rst = new ArrayList<Integer>();
if (n == 1) {
rst.add(0);
return rst;
}else if (n == 0 || edges == null || edges.length == 0 || edges.length != n - 1) {
return rst;
}

```java
        //populate map
        boolean[] nodes = new boolean[n];
        HashMap<Integer, ArrayList<Integer>> map = new HashMap<Integer, ArrayList<Integer>>();
        for (int i = 0; i < n; i++) {
            map.put(i, new ArrayList<Integer>());
            nodes[i] = true;
        }
        for (int i = 0; i < edges.length;  i++) {
            if (!map.get(edges[i][0]).contains(edges[i][1])) {
                map.get(edges[i][0]).add(edges[i][1]);
            }
            if (!map.get(edges[i][1]).contains(edges[i][0])) {
                map.get(edges[i][1]).add(edges[i][0]);
            }
        }


        //Remove list with leng == 1
        Queue<Integer> queue = new LinkedList<Integer>();
        while (n > 2) {
            for (Map.Entry<Integer, ArrayList<Integer>> entry : map.entrySet()) {
                if (entry.getValue().size() == 1) {
                    queue.offer(entry.getKey());
                }
            }
            while (!queue.isEmpty()) {
                n--;
                Integer key = queue.poll();
                nodes[key] = false;
                int from = map.get(key).get(0);
                map.get(from).remove(key);
                map.get(key).remove(0);

            }
        }


        //Put remaining into rst
        for (int i = 0; i < nodes.length; i++) {
            if (nodes[i]) {
                rst.add(i);
            }
        }


    }

    return rst;
}


}
```

186. Minimum Path Sum.java (https://github.com/shawnfan/LintCode/blob/master/Java/Minimum%20Path%20Sum.java)Given a m x n grid filled with non-negative numbers, find a path from top left to bottom right which minimizes the sum of all numbers along its path.

Note
You can only move either down or right at any point in time.

Example
Tags Expand
Dynamic Programming

Thinking process:

1. Check null, lenght == 0
2. Min Sum = sum of array. Initialization is a bit different, for example: each row element is added up from previous elemenet. (Not simple value assign from given grid)

   - Assign (0,0) to grid[0][0]
   - Row 1st row and 1st col, add up values

3. f(x,y) = sum of path value. f(x,y) = Math.Min(f(x-1,y), f(x, y-1))
4. return f(r-1)(c-1)

*/

public class Solution {
/**

```java
 * @param grid: a list of lists of integers.
 * @return: An integer, minimizes the sum of all numbers along its path
 */
public int minPathSum(int[][] grid) {

    if (grid == null || grid.length == 0 || grid[0].length == 0) {
        return 0;
    }
    int row = grid.length;
    int col = grid[0].length;
    int[][] matrix = new int[row][col];
    matrix[0][0] = grid[0][0];
    //Add up for 1st row && 1st col
    for (int i = 1; i < row; i++) {
        matrix[i][0] = matrix[i - 1][0] + grid[i][0];
    }
    for (int j = 1; j < col; j++) {
        matrix[0][j] = matrix[0][j - 1] + grid[0][j];
    }
    //Evaluate
    for (int i = 1; i < row; i++) {
        for (int j = 1; j < col; j++) {
            matrix[i][j] = Math.min(matrix[i - 1][j], matrix[i][j - 1])
                            + grid[i][j];
        }
    }
    return matrix[row - 1][col - 1];

}
```

}

---

187. Minimum Size Subarray Sum.java (https://github.com/shawnfan/LintCode/blob/master/Java/Minimum%20Size%20Subarray%20Sum.java) Level: Medium

2 pointer, O(n). 找subarray, start 或 end pointer，每次一格这样移动.

好的策略: 先找一个solution, 定住end, 然后移动start; 记录每个solution if occurs； 然后再移动end，往下找。

Note: 虽然一眼看上去是nested loop.但是分析后，发现其实就是按照end pointer移动的Loop。start每次移动一格。总体上，还是O(n)

Note done the O(nlogn) yet

---

188. Minimum Subarray.java (https://github.com/shawnfan/LintCode/blob/master/Java/Minimum%20Subarray.java)Given an array of integers, find the subarray with smallest sum.

Return the sum of the subarray.

Example
For [1, -1, -2, 1], return -3

Note
The subarray should contain at least one integer.

Tags Expand
Greedy LintCode Copyright Subarray Array

Thoughts:
Note: sub-array has order. It's not sub-set

1. On each index: decide to add with nums.get(i), to use the new lowest value nums.get(i). That means:

   If the new value is negative (it has decresing impact on sum) and the sum is larger than new value, just use the new value.

   In another case, if sum has been nagative, so sum + new value will be even smaller, then use sum.
2. Every time compare the currMin with the overall minimum value, call it minRst.

Note: remember to pre-set init value for curMin, minRst.
*/

public class Solution {
/**

```
 * @param nums: a list of integers
 * @return: A integer indicate the sum of minimum subarray
 */
public int minSubArray(ArrayList<Integer> nums) {
    if (nums == null || nums.size() == 0) {
        return 0;
    }
    int curMin = nums.get(0);
    int minRst = nums.get(0);
    for (int i = 1; i < nums.size(); i++) {
        curMin = Math.min(nums.get(i), curMin + nums.get(i));
        minRst = Math.min(curMin, minRst);
    }
    return minRst;
}
```

}

189. Minimum Window Substring.java (https://github.com/shawnfan/LintCode/blob/master/Java/Minimum%20Window%20Substring.java) Level: Hard

LeetCode Hard
LintCode M, 测试有问题，即使做错也能过.

基本思想: 用个map存target的<char, int frequency>。 然后在搜索s的时候，遇到Match， frequency--.
一旦map里面的frequency都被减为0, 就说明找到candidate.

有好几个trick：考虑start，前指针怎么移动；考虑start在candidate首字母没有多余前，不能移动；考虑candidate出现的情况...

复习时，回去看别人网站和自己的thoughts

---

190. MinimumDepthOfBinaryTree.java (https://github.com/shawnfan/LintCode/blob/master/Java/MinimumDepthOfBinaryTree.java) Level: Easy

Divide and Conquery一个最小值. 注意处理Leaf的null, 用Integer.MAX_VALUE代替，这样可以避免错误counting.

---

191. Missing Ranges.java (https://github.com/shawnfan/LintCode/blob/master/Java/Missing%20Ranges.java)自己做的时候，想的太复杂，做起了binarysearch,企图节省时间。
下次要算清楚，是否有意义。
binarySearch的确logn,但是在lower 和upper之间的数字，很可能还是O(n).
因此一开始就for一遍也是O(n), 而code会相对来说简单许多。

想法：
两个pointer， 每次计较prev和curr之间的部分。
然后prev = curr，向前移动一格。

---

192. Multiply Strings.java (https://github.com/shawnfan/LintCode/blob/master/Java/Multiply%20Strings.java) Level: Medium

想法不难。turn into int[], 然后每个位子乘积，然后余数carrier移位。

但是做起来有很多坑。适合面试黑。

1. 数字'123'，在数组里面， index == 0 是 '1'。 但是我们平时习惯从最小位数开始乘积，就是末尾的'3'开始。
   所以！翻转两个数字先！我去。这个是个大坑。

2. 乘积product，和移动Carrier都很普通。

3. ！！最后不能忘了再翻转。

4. 最后一个看坑。要是乘积是0，就返回'0'。 但是这个其实可以在开头catch到没必要做到结尾catch。

用到几个StringBuffer的好东西:
reverse（）；
sb.deleteCharAt(i)

找数字，或者26个字母，都可以：
s.charAt(i) – '0'; //数字
s.charAt(i) – 'a'; //字母

---

193. Next Permutation.java (https://github.com/shawnfan/LintCode/blob/master/Java/Next%20Permutation.java) Level: Medium

需斟酌。

Permutation的规律:

1. 从小的数字开始变化因为都是从小的数字开始recursive遍历。
2. 正因为1的规律，所以找大的断点数字要从末尾开始： 确保swap过后的permutation依然是 前缀固定时 当下最小的。

steps:

1. 找到最后一个上升点，k
2. 从后往前，找到第一个比k大的点, bigIndex
3. swap k &&   bigIndex
4. 最后反转 (k+1，end)

---

194. Nim Game.java (https://github.com/shawnfan/LintCode/blob/master/Java/Nim%20Game.java)写一些，发现n=4,5,6,7,8...etc之后的情况有规律性。
最终很简单n%4!=0就可以了

---

195. NQueens.java (https://github.com/shawnfan/LintCode/blob/master/Java/NQueens.java)index就是col number
值就是row number.

validate n queue的时候 target row#

1. array 里面不能有 target row#
2. diagnal. 记得公式：
   row1 – row2 == col1 – col2. Diagnal elelment.fail

row1 – row2 == –(col1 – col2). Diagnal element. fail

---

196. NQueensII.java (https://github.com/shawnfan/LintCode/blob/master/Java/NQueensII.java)直接add 一个什么乱起八糟的东西进rst都可以。

然后最后要的是 rst.size() = # of solutons

---

197. Nth to Last Node in List.java (https://github.com/shawnfan/LintCode/blob/master/Java/Nth%20to%20Last%20Node%20in%20List.java)然后head开始跑。

node 到底，而head ~ node刚好是 n 距离。所以head就是要找的last nth

---

198. Number of Airplane in the sky.java (https://github.com/shawnfan/LintCode/blob/master/Java/Number%20of%20Airplane%20in%20the%20sky.java) Level: Medium

把Interval拆分成数轴上的Point：
起飞mark 1
降落mark –1
用PriorityQueue排序，loop through queue, 计算(起飞+降落)值可能有的max。

注意:
同时起飞和降落，就是 1 – 1 = 0. 所以在while loop里面有第二个while loop，
当坐标x重合时，在这里做完所有x点的加减，然后再比较 max。
这避免了错误多count，或者少count

---

199. Number of Islands II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Number%20of%20Islands%20II.java) Level: Hard

用HashMap的Union-find.

把board转换成1D array，就可以用union-find来判断了。 判断时，是在四个方向各走一步，判断是否是同一个Land.

每走一次operator，都会count++. 若发现是同一个island, count--

---

200. Number of Islands.java (https://github.com/shawnfan/LintCode/blob/master/Java/Number%20of%20Islands.java) Level: Medium

方法1: 两个for loop brutle force。 DFS把每个跟1相关的都Mark一遍.生成一个island.

方法2：（暂时没有写union-find的解）
可以用union-find，就像Number of island II 一样。
只不过这个不Return list, 而只是# of islands

201. Number Triangles.java (https://github.com/shawnfan/LintCode/blob/master/Java/Number%20Triangles.java)Given a triangle, find the minimum path sum from top to bottom. Each step you may move to adjacent numbers on the row below.

Note
Bonus point if you are able to do this using only O(n) extra space, where n is the total number of rows in the triangle.

Example
For example, given the following triangle

[
[2],
[3,4],
[6,5,7],
[4,1,8,3]
]
The minimum path sum from top to bottom is 11 (i.e., 2 + 3 + 5 + 1 = 11).

Tags Expand
Dynamic Programming

Thinking process:

1. Bottom-up

   - Start from the bottom row, get all values of this row. Note: in triangle, height = cols at each row. So row X has X numbers.
   - Start from (n − 1)th row and run up: calculate min from lower level + current node value.
   - Depending what is wanted, here we use a 2D int arraya and return the min sum.
     */

public class Solution {
/**

```
 * @param triangle: a list of lists of integers.
 * @return: An integer, minimum path sum.
 */
//Bottom - up
public int minimumTotal(ArrayList<ArrayList<Integer>> triangle) {
    if (triangle == null || triangle.size() == 0) {
        return 0;
    }
    int n = triangle.size();
    int[][] sum = new int[n][n];
    for (int i = 0; i < n; i++) {
        sum[n - 1][i] = triangle.get(n - 1).get(i);
    }
    for (int i = n - 2; i >= 0; i--) {
        for (int j = 0; j <= i; j++) {
            sum[i][j] = Math.min(sum[i + 1][j], sum[i + 1][j + 1]) + triangle.get(i).get(j);
        }
    }
    return sum[0][0];
}
```

}

/*

1. Memorize Search

- Go through all nodes and initialize with Integer.MAX_VALUE;

- Search from top: thislevel-current = Math.min(nextlevel-current, nextlevel-next) + thislevel-current

- During the Search Helper, when a node has been set previously, just return this value because this min value has been pre-calculated.

  If row is >= triangle.size(), return 0.

- This method can actually calculate the min sum from bottom to any point in the triangle.

  */

public class Solution {
/**

```
 * @param triangle: a list of lists of integers.
 * @return: An integer, minimum path sum.
 */

//Version 2 : Memorize Search
private int n;
private ArrayList<ArrayList<Integer>> triangle;
public int minimumTotal(ArrayList<ArrayList<Integer>> triangle) {
    if (triangle == null || triangle.size() == 0) {
        return 0;
    }
    this.n = triangle.size();
    this.triangle = triangle;
    int[][] sum = new int[n][n];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j <= i; j++) {
            sum[i][j] = Integer.MAX_VALUE;
        }
    }
    return searchHelper(0, 0, sum);
}

public int searchHelper(int r, int c, int[][] sum) {
    if (r >= this.n) {
        return 0;
    }
    if (sum[r][c] != Integer.MAX_VALUE) {
        return sum[r][c];
    }
    sum[r][c] = Math.min(searchHelper(r + 1, c, sum), searchHelper(r + 1, c + 1, sum)) + this.triangle.get(r).get(c);
    return sum[r][c];
}

}
```

---

202. O(1) Check Power of 2.java (https://github.com/shawnfan/LintCode/blob/master/Java/O(1)%20Check%20Power%20of%202.java)Using O(1) time to check whether an integer n is a power of 2.
Example
For n=4, return true

For n=5, return false

Challenge
O(1) time

Tags Expand
Binary

Thinking process:
Any integer that's power of 2, follows one pattern. They are all: 1000000000....000 format.
so (n – 1) becomes: 01111111111...111.
If bit-and them togeter, it will be 0.

*/

class Solution {
/*

```
 * @param n: An integer
 * @return: True or false
 */
public boolean checkPowerOf2(int n) {
    if (n <= 0) {
        return false;
    }
    return (n & (n - 1)) == 0;
}
```


};

---

203. One Edit Distance.java (https://github.com/shawnfan/LintCode/blob/master/Java/One%20Edit%20Distance.java) Level: Medium

理解Edit: 就是删除，增加，和替换。
换完之后，理论上换成的String 就应该全等
一旦找到不一样的char，就判断那三种可能性

---

204. Paint Fence.java (https://github.com/shawnfan/LintCode/blob/master/Java/Paint%20Fence.java)设定T（n）的做法，最后题目化简以后就跟Fibonacci number一样一样的。
详细分析如下。
做完，还是觉得如有神。本来是个Easy题，想不到，就是搞不出。

12.13.2015再看了一下：
因为最多2个fence 颜色相同。
假设i是和 i-1不同，那么结果就是 (k-1)dp[i – 1]
假设i是何 i-1相同，那么根据条件，i-1和i-2肯定不同。那么所有的结果就是(k-1)dp[i-2]
加在一起就有了。

205．Palindrome Linked List.java (https://github.com/shawnfan/LintCode/blob/master/Java/Palindrome%20Linked%20List.java)linkedlist 不能 reverse iterating， 那么就 reverse the list, 从中间开花作比较。

206．Palindrome Partitioning II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Palindrome%20Partitioning%20II.java)看上去，在检查i,j的时候，中间按的（i+1, j-1）怎么可能先知道？其实不然..在j慢慢长大的时候，所有的0~j的substring都检查过。所以isPal[i+1][j-1]一定是已经知道结果的。

okay.那么假如以上任意一种情况成立，也就是说isPal[i][j] == true。那就要判断，切到第一层循环参数j的末尾点时，有多少种切法？
想法很顺：我们naturally会想到，把i之前的cut加上i~j之间发生的不就好了。
反正现在j不变，现在就看吧i定在哪里，cut[i - 1]是否更小/最小； 再在cut[i-1]基础上+1就完了。
当然，如果i==0，而 i~j又是isPal,那没啥好谈的，不必切，0刀。

最终，刷到cut[s.length() - 1] 也就是最后一点。 return的理所应当。

207. Palindrome Partitioning.java (https://github.com/shawnfan/LintCode/blob/master/Java/Palindrome%20Partitioning.java)在遍历str的时候，考虑从每个curr spot 到 str 结尾，是能有多少种palindorme? 那就从curr spot当个字符开始算，开始back tracing.
如果所选不是palindrome， 那move on.
若所选的确是palindrome， 加到path里面，DFS去下个level，等遍历到了结尾，这就产生了一种分割成palindrome的串。
每次DFS结尾，要把这一层加的所选palindrome删掉，backtracking嘛。

208. Palindrome Permutation II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Palindrome%20Permutation%20II.java) Level: Medium

permutation的综合题：

1. validate Input 是不是可以做palindromic permutation. 这个就是（Palindrome Permutation I）
2. 顺便存一下permutation string的前半部分和中间的single character(if any)
3. DFS 做unique permutation: given input有duplicate characters。

209. Palindrome Permutation.java (https://github.com/shawnfan/LintCode/blob/master/Java/Palindrome%20Permutation.java) Level: Easy

注意，条件里面没说是否全是lower case letter

210. Partition Array by Odd and Even.java (https://github.com/shawnfan/LintCode/blob/master/Java/Partition%20Array%20by%20Odd%20and%20Even.java)Partition an integers array into odd number first and even number second.

Example
Given [1, 2, 3, 4], return [1, 3, 2, 4]

Challenge
Do it in-place.

Tags Expand
Two Pointers Array

Thougths:
Use two pointers: nextOddPt, firstEvenPt

1. Whenever nextOddPt > firstEvenPt, swapt them
2. Incrase nextOddPt in a for loop
   Note:
   After each swap, have to start checking again from beginning-switching point, which will be firstEvenPt. Need to set i = firstEvenPt.
   However, since for loop will do i++, we need to set i = firstEvenPt - 1;
   And firstEvenPt only needs to be update once so use -1 to check if it's set.
   */

public class Solution {
/**

```
 * @param nums: an array of integers
 * @return: nothing
 */
public void partitionArray(int[] nums) {
    if (nums == null || nums.length == 0){
        return;
    }
    int nextOddPt = -1;
    int firstEvenPt = -1;
    for (int i = 0; i < nums.length; i++) {
        if (nums[i] % 2 == 1) {
            nextOddPt = i;
        } else {
            if (firstEvenPt == -1) {
                firstEvenPt = i;
            }
        }
        if (nextOddPt > firstEvenPt && firstEvenPt != -1) {
            int temp = nums[nextOddPt];
            nums[nextOddPt] = nums[firstEvenPt];
            nums[firstEvenPt] = temp;
            i = firstEvenPt - 1;
            firstEvenPt = -1;
        }
    }
}
```

}

211. Partition Array.java (https://github.com/shawnfan/LintCode/blob/master/Java/Partition%20Array.java) 从 array 两边开始缩进。 while loop 到遍历完。非常直白的 implement。
注意low/high,或者叫start/end不要越边界
O(n)

Quick sort的基础。

212. Partition List.java (https://github.com/shawnfan/LintCode/blob/master/Java/Partition%20List.java)
那就最普通的，建造两个list

把满足条件（<x, >=x）的数字分别放到两个list里面

记得用dummyNode track head.
最终pre.next = post链接起来。

213. Pascal's Triangle II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Pascal's%20Triangle%20II.java) Level: Easy

简单处理array list.

---

214. Peeking Iterator.java (https://github.com/shawnfan/LintCode/blob/master/Java/Peeking%20Iterator.java)
回到原题，其实不难。找一个cache来存next()的值，然后每次next()里面维护这个cache就好。

---

215. Perfect Squares.java (https://github.com/shawnfan/LintCode/blob/master/Java/Perfect%20Squares.java)
1. 第一步想到了，从数学角度，可能是从最大的perfect square number开始算起。
2. 然后想法到了dp，假设最后一步用了最大的maxSqrNum, 那么就在剩下的 dp[i - maxSqrNum^2] + 1　不就好了？
3. 做了，发现有个问题．．．最后一步选不选maxSqrNum? 比如12就是个例子。
然后就根据提示，想到BFS。顺的。 把1～maxSqrNum 都试一试。找个最小的。
看我把12拆分的那个example. 那很形象的就是BFS了。
面试时候，如果拆分到这个阶段不确定，那跟面试官陶瓷一下，说不定也就提示BFS了。

---

216. Permutation Index.java (https://github.com/shawnfan/LintCode/blob/master/Java/Permutation%20Index.java) Level: Easy

和Permutation Sequence相反的题目。思想类似。

题目为Easy，琢磨很久，分析：
每个数位的数字，都是跳过了小于这数字开头的多种可能。

举例【6，5，2】吧。我们找6，5，2是permudation里面的第几个。
正常排序，也就是permutation的第一个，应该是【2，5，6】
如果要从首位，2，变成6，要跨过多少可能性呢？
很简单，就是问：小于6的数字有多少个呢？（2，5）.每个数字变成head，都有各自的一套变化，都有(n-1)!种可能。

本题做法：每个（n-1）!加起来。　Note:(n-1) means, 开头的数字(2,5)各带出多少种排列，也就是不就是(n-1)!嘛。
这一步，计算数量很简单: (有几个小于6的数字) ×(除去head剩下有多少个数字)!

以上，都是为了把6推上皇位，而牺牲的条数。

那么把6推上去以后，还有接下去的呢。

接下去要看5，2.
6确定，后面ｐｅｒｍｕｄａｔｉｏｎ可变的情况有可能是【6，5，2】，那还可能是【6，2，5】呢。

Same process, 看 g i v e n　数组的第二位 5，算它接下去：

1． 有几个数字小于 5 呢？

2． 除去 5，还有几个数字可以　f a c t o r i a l 呢？

3．一样的。第一步就结果乘以第二步。

最后接下去要看最后一个元素2了。

6,5,2全看过了以后，加起来。

就是【6，5，2】上位，所踏过的所有小命啊！

我这解释太生动了。因为耗费了好长时间思考...

---

217. Permutation Sequence.java (https://github.com/shawnfan/LintCode/blob/master/Java/Permutation%20Sequence.java) Level: Medium

k是permutation的一个数位。而permutation是有规律的。

也就是说，可以根据k的大小来判断每一个数位的字符（从最大数位开始，因为默认factorio从最大数位开始变化）。

于是先求出n!，然后 k/n!就可以推算出当下这一个数位的字符。然后分别把factorio 和 k减小。

另外, 用一个boolean[] visited来确保每个数字只出现一次。

这个方法比计算出每个permutation要efficient许多。

---

218. Permutations II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Permutations%20II.java) Level: Medium

方法1：
Mark visited. 并且要检查上一层recursive时有没有略过重复element. 并且要排序，通过permutation规律查看是否排出了重复结果。

背景1:在recursive call里面有for loop, 每次从i=0开始, 试着在当下list上加上nums里面的每一个。
从i=0开始，所以会依次recursive每一个nums：因此，例如i=2,肯定比i=3先被访问。也就是:取i=2的那个list permutation肯定先排出来。

背景2:重复的例子：给出Input[x, y1, y2], 假设y的值是一样的。那么，{x,y1,y2}和{x,y2,y1}是相同结果。

综上，y1肯定比y2先被访问,{x,y1,y2}先出。 紧随其后，在另一个recursive循环里，{x,y2...}y2被先访问，跳过了y1。
重点:规律在此，如果跳过y1，也就是visited[y1] == false, 而num[y2] == num[y1]，那么这就是一个重复的结果，没必要做，越过。

结果:那么，我们需要input像{x,y1,y2}这样数值放一起，那么必须排序。

方法2:

一个办法就是给一个visited queue。 和queue在所有的地方一同populate. 然后visited里面存得时visited indexes。 (Not efficient code. check again)

---

219. Permutations.java (https://github.com/shawnfan/LintCode/blob/master/Java/Permutations.java) Level: Medium

Recursive： 取，或者不取。

Iterative: 用个queue，每次poll()出来的list, 把在nums里面能加的挨个加一遍。 However, code is a bit massive.

---

220. Plus One.java (https://github.com/shawnfan/LintCode/blob/master/Java/Plus%20One.java)Given a non-negative number represented as an array of digits, plus one to the number.

The digits are stored such that the most significant digit is at the head of the list.

Example
Given [1,2,3] which represents 123, return [1,2,4].

Given [9,9,9] which represents 999, return [1,0,0,0].

Tags Expand
Array

*/

```java
public class Solution {
public int[] plusOne(int[] digits) {
if(digits.length==0) return digits;
```

```
        digits[digits.length-1] += 1;
        //Check index digit.length-1 to 1
        for(int i = digits.length-1; i>0; i--){
            if(digits[i] == 10){
                digits[i]=0;
                digits[i-1]+=1;
            }
            else return digits;
        }

        //Check index 0. If ==0, set it to 0 and carry over 1
        if(digits[0]==10){
            int[] output = new int[digits.length+1];
            output[0] = 1;
            output[1] = 0;
            for(int i=2; i<output.length-1; i++){
                output[i]=digits[i-1];
            }
            return output;
        }
        else return digits;
    }
```

}

/* Trivial solution
create a secondary method func(int index, int[]digits).
add check index from digits.length−1 to 0: digits[index]+1==10? 0 : digits[index]+1;
if add up to 10, push into another level; if not ,return digits.
if index==0, check if add up to 10. If ==10, create a new array and put 1 infront. else return digits.

*/

/*

Thoughts: Old soluton .will fail LeetCode
It looks I should convert array to int, then add, and then convert back to array.

 1. Convert to string: Arrays.toString(xxx);

 2. Integer.parseInt(str)

 3. add

 4. split to int array

Note:

Int may not hold the rst since it could exceed 32 bits, so use Long.

But … What if long does not work neither?
*/

public class Solution {
/**

```
 * @param digits a number represented as an array of digits
 * @return the result
 */
public int[] plusOne(int[] digits) {
    if (digits == null || digits.length == 0) {
        return null;
    }

    String str = "";
    for (int i = 0; i < digits.length; i++) {
        str += digits[i];
    }
    long digit = Long.parseLong(str);
    digit += 1;
    str = digit + "";
    int[] rst = new int[str.length()];
    for (int i = 0; i < str.length(); i++) {
        rst[i] = Character.getNumericValue(str.charAt(i));
    }
    return rst;
}
```

}

---

221. Populating Next Right Pointers in Each Node II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Populating%20Next%20Right%20Pointers%20in%20Each%20Node%20II.java) Level: Hard

非perfect tree, 也就是有random的null children. DFS + BFS

Populating Next Right Pointers in Each Node I 里面依赖parent.next.left来作链接，但现在这个parent.next.left很可能也是Null.

1. 于是需要移动parent去找children level的next node。
2. 并且每次在一个level, 要用BFS的思想把所有parent 过一遍，也就是把parent 正下方的children全部用.next链接起来
   原因: 到下一层children变成parent, 他们需要彼此之间的connection, grand children才可以相互连接。

Note: runtime O(n * 2^log(n)) = O(n^2), not good.

---

222. Populating Next Right Pointers in Each Node.java (https://github.com/shawnfan/LintCode/blob/master/Java/Populating%20Next%20Right%20Pointers%20in%20Each%20Node.java) Level: Medium

方法1：
题目要求DFS。
其实basic implementation. 每次处理node.left.next = node.right; node.right.next = node.next.left;

方法2:
不和题意，用了queue space，与Input成正比。太大。

BFS over Tree。 用Queue 和 queue.size(), 老规矩。
process每层queue时, 注意把next pointer加上去就好.

---

223. Pow(x,n).java (https://github.com/shawnfan/LintCode/blob/master/Java/Pow(x,n).java)n的正负。
n == 0的情况。

---

224. Power of Three.java (https://github.com/shawnfan/LintCode/blob/master/Java/Power%20of%20Three.java) Level: Easy

Power of 3: 3 ^ x == n ?

做出发. 查%.

---

225. Product of Array Exclude Itself.java (https://github.com/shawnfan/LintCode/blob/master/Java/Product%20of%20Array%20Exclude%20Itself.java)Given an integers array A.

Define B[i] = A[0] ... A[i-1] A[i+1] ... * A[n-1], calculate B WITHOUT divide operation.

Example
For A = [1, 2, 3], return [6, 3, 2].

Tags Expand
Forward-Backward Traversal LintCode Copyright

Thought:
Trivial way would be first calculate the zigma(A[0]* ... A[n-1]) then divide by B[i]. However, not allowed in this question.

The other way: do for loop again and again? that will be n^2 time.

*/

public class Solution {
/**

```
 * @param A: Given an integers array A
 * @return: A Long array B and B[i]= A[0] * ... * A[i-1] * A[i+1] * ... * A[n-1]
 */
 public ArrayList<Long> productExcludeItself(ArrayList<Integer> A) {
     if (A == null || A.size() == 0) {
         return null;
     }
     ArrayList<Long> rst = new ArrayList<Long>();
     for (int i = 0; i < A.size(); i++) {
         long num = 1;
         for (int j = 0; j < A.size(); j++) {
             if (j != i) {
                 num *= A.get(j);
             }
         }
         rst.add(num);
     }
     return rst;
 }
```

}

---

226. QuickSort.java (https://github.com/shawnfan/LintCode/blob/master/Java/QuickSort.java) Level: Easy

代码是不难的.

首先partition. 返还一个partition的那个中间点的位置。
然后劈开两半。
前后各自 quick sort, recursively

注意：在partition里面, 比较的时候nums[start] < pivot, nums[end]>pivot, 如果写成了 <= 会 stack overflow.

但是： 在partition array那个题目里面，第二个 nums[end] >= pivot, 是要去加上这个 '='的

227. Recover Rotated Sorted Array.java (https://github.com/shawnfan/LintCode/blob/master/Java/Recover%20Rotated%20Sorted%20Array.java)Rotate三步：

rotate前半

rotate后半

rotate全部

注意先找到断点。

---

228. Rehashing.java (https://github.com/shawnfan/LintCode/blob/master/Java/Rehashing.java) Level: Medium

---

229. Remove Duplicates from Sorted Array.java (https://github.com/shawnfan/LintCode/blob/master/Java/Remove%20Duplicates%20from%20Sorted%20Array.java)
LinkedList里面我们是最好不要动node.val的，直接把node去掉。
而array我们很难直接把node去掉，又不能用新array，那么就要：

把不重复的element一个个放到最前面。

这个思想跟merge two sorted array （其中一个后续非常长的array可以放下arr1,arr2） 类似。
就是找个不会事后mess up，不会去动得index,把满足条件的element 填进去。这样保证了in place.

- 有个反向思维：remove duplicate,实际上也是找unique elements, and insert into original array

---

230. Remove Duplicates from Sorted List II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Remove%20Duplicates%20from%20Sorted%20List%20II.java)多
个node，check node.next ?= node.next.next

---

231. Remove Duplicates from Sorted List.java

(https://github.com/shawnfan/LintCode/blob/master/Java/Remove%20Duplicates%20from%20Sorted%20List.java)

232. Remove Duplicates from Unsorted List.java (https://github.com/shawnfan/LintCode/blob/master/Java/Remove%20Duplicates%20from%20Unsorted%20List.java) 遍
历。
遇到duplicate(可能多个), while直到node.next不是duplicate。
接下去,既然不是duplicate,那就add 进 set

如果不用extra memory, do it in place:
那就要sort linked list. 用merge sort.

复习merge sort:

1. find middle.

2. recursively: right = sort(mid.next); left = sort(head).

3. within sort(), at the end call merge(left, right)

---

233. Remove Linked List Elements.java (https://github.com/shawnfan/LintCode/blob/master/Java/Remove%20Linked%20List%20Elements.java) 如果不match, parent 和 node 一起移动

---

234. Remove Node in Binary Search Tree.java (https://github.com/shawnfan/LintCode/blob/master/Java/Remove%20Node%20in%20Binary%20Search%20Tree.java) Level: Hard

方法1: Brutle一点。找到target和target的parent.
把target remove时，把target的children nodes 重新排列组成新的BST: inorder traversal, build tree based on inorder traversal list.

方法2: 分析规律,先找到target和parent, 然后根据性质，把target remove时，移动children nodes, 保证还是BST。

---

235. Remove Nth Node From End of List.java (https://github.com/shawnfan/LintCode/blob/master/Java/Remove%20Nth%20Node%20From%20End%20of%20List.java)Given a linked list, remove the nth node from the end of list and return its head.

Note
The minimum number of nodes in list is n.

Example
Given linked list: 1->2->3->4->5->null, and n = 2.
After removing the second node from the end, the linked list becomes 1->2->3->5->null.
Challenge
O(n) time

Tags Expand
Two Pointers Linked List

Thinking process:
Very similar to 'Nth to last node'. Except, have a pre pointer to keep track of the previous node of 'nth to last'.
Also have a dummy.next to store the beginning of the list;
/
/*

- Definition for ListNode.
- public class ListNode {
- int val;
- ListNode next;
- ListNode(int val) {
- this.val = val;
- this.next = null;
- }
- }

```
/
public class Solution {
    /*
```

- @param head: The first node of linked list.
- @param n: An integer.
- @return: The head of linked list.

```
    */
    ListNode removeNthFromEnd(ListNode head, int n) {
    if (head == null || n < 0) {
```

```
        return null;
```

```
    }
    int count = 0;
    ListNode dummy = new ListNode(0);
    ListNode pre = new ListNode(0);
    pre.next = head;
    dummy = pre;
    ListNode node = head;
    while (node != null && count < n) {
```

```
        node = node.next;
        count++;
```

```
    }
    while (node != null) {
```

```
        node = node.next;
        head = head.next;
        pre = pre.next;
```

    }
    pre.next = head.next;
    return dummy.next;
    }
    }

---

236. Reorder List.java (https://github.com/shawnfan/LintCode/blob/master/Java/Reorder%20List.java)24% 通过
Given a singly linked list L: L0→L1→…→Ln-1→Ln,
reorder it to: L0→Ln→L1→Ln-1→L2→Ln-2→…

You must do this in-place without altering the nodes' values.

样例
For example,
Given 1->2->3->4->null, reorder it to 1->4->2->3->null.

标签 Expand
Linked List

Thinking Process:
Similar to sort list:
find middle.
reverse last section
merge(head, mid) alternatively by using index % 2.
Append whatever left from the 2 lists
Note: re-order in place, does not necessarily mean you can create any variable. As long as the variable is O(1), it should be fine.
*/

/**

- Definition for ListNode.
- public class ListNode {
- int val;
- ListNode next;

- ListNode(int val) {
- this.val = val;
- this.next = null;
- }
- }

*/

public class Solution {

private ListNode reverse(ListNode head) {

```
ListNode reversedList = null;
while (head != null) {
    ListNode temp = head.next;
    head.next = reversedList;
    reversedList = head;
    head = temp;
}
return reversedList;
```

}

private void merge(ListNode head1, ListNode head2) {

```
ListNode dummy = new ListNode(0);
int index = 0;
while (head1 != null && head2 != null) {
    if (index % 2 == 0) {
        dummy.next = head1;
        head1 = head1.next;
    } else {
        dummy.next = head2;
        head2 = head2.next;
    }
    dummy = dummy.next;
    index += 1;
}
if (head1 != null) {
    dummy.next = head1;
} else if (head2 != null) {
    dummy.next = head2;
}
```

}

private ListNode findMiddle(ListNode head) {

```
        ListNode slow = head;
        ListNode fast = head.next;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }
```

```
public void reorderList(ListNode head) {
    if (head == null || head.next == null) {
        return;
    }

    ListNode mid = findMiddle(head);
    ListNode tail = reverse(mid.next);
    mid.next = null;

    merge(head, tail);
}
```

}

---

237. Restore IP Addresses.java (https://github.com/shawnfan/LintCode/blob/master/Java/Restore%20IP%20Addresses.java)递归在一个index上面（具体问题，具体分析的情况）
validate string要注意leading '0'

注意： 递归的时候可以用一个start/level/index来跑路
但是尽量不要去改变Input source， 会变得非常confusing.

---

238. Reverse Integer.java (https://github.com/shawnfan/LintCode/blob/master/Java/Reverse%20Integer.java)Reverse digits of an integer. Returns 0 when the reversed integer overflows (signed 32-bit integer).

Example
Given x = 123, return 321

Given x = -123, return -321

Tags Expand
Integer

Thoughts:

1. Use long to capture the result. If > Integer.MAX_VALUE,return 0;

2. Use string to reverse, the conver to long

3. use string builder to reverse string

*/

public class Solution {
/**

```
 * @param n the integer to be reversed
 * @return the reversed integer
 */
public int reverseInteger(int n) {
    long num = (long)n;
    int sign = n > 0 ? 1 : -1;
    String rst = new StringBuilder(Math.abs(num)+"").reverse().toString();
    num = Long.parseLong(rst) * sign;

    if (num > Integer.MAX_VALUE || num < Integer.MIN_VALUE) {
        return 0;
    } else {
        return (int)num;
    }
}
```

}

---

239. Reverse Linked List II .java (https://github.com/shawnfan/LintCode/blob/master/Java/Reverse%20Linked%20List%20II%20.java)存一下那个点，
从M开始，for loop，reverse [m~n]。 然后把三段链接在一起。

---

240. Reverse Linked List.java (https://github.com/shawnfan/LintCode/blob/master/Java/Reverse%20Linked%20List.java) Level: Easy

建立新list。每次把newList append 在current node的后面。
用head来循环所有node。

---

241. Reverse Words in a String II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Reverse%20Words%20in%20a%20String%20II.java) Level: Medium

In-place reverse.

reverse用两回. 全局reverse。局部:遇到空格reverse。

注意：结尾点即使没有' '也要给reverse一下最后一个词。

---

242. Reverse Words in a String.java (https://github.com/shawnfan/LintCode/blob/master/Java/Reverse%20Words%20in%20a%20String.java) Level: Medium

几种不同的方法flip：
坑：1. 结尾不能有空格。 2. 注意，如果Input是' '的话，split以后就啥也没有了。check split以后 length == 0

另个题目Reverse Words in String (char[]) 可以in-place，因为条件说char[]里面是没有首尾空格,好做许多哟.

---

243. reverseInteger.java (https://github.com/shawnfan/LintCode/blob/master/Java/reverseInteger.java)Reverse Integer

Reverse digits of an integer.

Example1: x = 123, return 321
Example2: x = -123, return -321
//input = 1534236469

Thinking process:
Make sure of operators.
Note: check for overflow using long. When integer is > Integer.MAX_VALUE, then it's overflow.
Initialize long : long x = 1234L;
Convert using (int)

/
public class Solution {
public int reverse(int x) {
if (x == 0) {
return x; //123
}
boolean sign = x > 0; //sign = true
long rst = 0L;
x = Math.abs(x); // 123
while (x != 0) { //x = 123, 12, 1
rst = rst 10 + x % 10; //rst = 3, 30 + 2 = 32, 320 + 1 = 321
x = x / 10; //x = 12; 1; 0
}

```
if (rst < 0 || rst > Integer.MAX_VALUE) {
return 0;
}
return sign ? (int)rst : -(int)rst;
}
}
```

---

244. Roman to Integer.java (https://github.com/shawnfan/LintCode/blob/master/Java/Roman%20to%20Integer.java) Level: Easy

熟悉罗马字母规则

1. 'I V X L C D M' 分别代表的数字
2. 'IV, IX'减1, 'XL, XC'减10, 'CD, CM'减100.

---

245. Rotate Image.java (https://github.com/shawnfan/LintCode/blob/master/Java/Rotate%20Image.java) Level: Medium

找到个转角度的规律公式。用一个temp。in place.

---

246. Rotate List.java (https://github.com/shawnfan/LintCode/blob/master/Java/Rotate%20List.java)/* Given a list, rotate the list to the right by k places, where k is non-negative.

Example
Given 1->2->3->4->5->null and k=2
return 4->5->1->2->3->null
Tags Expand
Basic Implementation Linked List

Thining process:
Two pointers.
First pointer move k steps.
Then 2 pointers start moving together. When 1st pointer reaches the end, then 2nd pointer should be in middle.
Let 2nd pointer be head, and move original head to tail of the list

/
/*

- Definition for singly-linked list.
```

```
public class ListNode {
    int val;
    ListNode next;
    ListNode(int x) {
        val = x;
        next = null;
    }
}
/
public class Solution {
    /*
     * @param head: the List
     * @param k: rotate to the right k places
     * @return: the list after rotation
     */
    public ListNode rotateRight(ListNode head, int k) {
        if (head == null || k == 0) {
            return head;
        }
        //Check length
        int length = 0;
        ListNode dummy = head;
        while(dummy != null) {
            dummy = dummy.next;
            length++;
        }
        k = k % length;
        //Store dummy as 1 node before tail
        dummy = new ListNode(0);
        dummy.next = head;
        head = dummy;
        for (int i = 0; i < k; i++) {
```

```
    head = head.next;
```

}

//Move 2 pointers. When head reaches end, tail.next will be at the newHead

ListNode tail = dummy;

while (head.next != null) {

```
    head = head.next;
    tail = tail.next;
```

}

head.next = dummy.next;//Link old Head to the end, form circle

dummy.next = tail.next;//Link tail.next as new head. tail should be end point.

tail.next = null;//add null to end point tail

return dummy.next;

}

}

---

247. Rotate String.java (https://github.com/shawnfan/LintCode/blob/master/Java/Rotate%20String.java)有个坑：offset可能很长，那么要%length，才能得到真正需要rotate的部分。
Note: rotate 一个 full length之后，是string 不变

---

248. Search a 2D Matrix II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Search%20a%202D%20Matrix%20II.java)每次删掉一行，或者一列

---

249. Search a 2D Matrix.java (https://github.com/shawnfan/LintCode/blob/master/Java/Search%20a%202D%20Matrix.java)2D转1D。
Binary Search

---

250. Search for a Range.java

(https://github.com/shawnfan/LintCode/blob/master/Java/Search%20for%20a%20Range.java)

251. Search Insert Position.java (https://github.com/shawnfan/LintCode/blob/master/Java/Search%20Insert%20Position.java)在结尾判断该return 哪个position。

---

252. Search Range in Binary Search Tree .java (https://github.com/shawnfan/LintCode/blob/master/Java/Search%20Range%20in%20Binary%20Search%20Tree%20.java)
Level: Medium

等于遍历了所有k1<= x <= k2的x node。

如果是用Binary Search Tree搜索，那么一般是if (...) else {...}，也就是一条路走到底，直到找到target.

这里, 把 left/right/match的情况全部cover了，然后把k1,k2的边框限制好，中间就全部遍历了。

---

253. Search Rotated in Sorted Array II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Search%20Rotated%20in%20Sorted%20Array%20II.java) 因为最终
binary search的结果也是O(n)
所以这道题要记得： 既然是O(n), 那来个简单的for loop 也就好了。

当然，要跟面试官提起来原因。别一上来就只有for。。。

---

254. Search Rotated in Sorted Array.java (https://github.com/shawnfan/LintCode/blob/master/Java/Search%20Rotated%20in%20Sorted%20Array.java) Level: Hard

方法1：O(logn)
还是把它先当做正常的sorted list开始搜。
但是在比较的时候，多比较一个A[start] < A[mid]?
在1 和 2 里面分别讨论 target 的位置

```
1. A[start] < A[mid] ?
      说明在前半段
    - start<target<mid
    - target > mid
2. A[start] > A[mid]
      说明 start 还在前半段，而mid在后半段
    - mid < target < end
    - target < mid
```

方法2：O(logn)

```
1. binay search break point
2. binary search target
注意等号，在判断target在前半段还是后半段：if (A[p1] <= target && target <= A[breakPoint])
```

---

255. Segment Tree Build II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Segment%20Tree%20Build%20II.java) Level: Medium

给的是Array。注意找区间内的max, assign给区间。 其余和普通的segment tree build一样

给了array,但是并不根据array里的内容排位，而是依然根据index in [0, array.length - 1]割开区间，break到底，
最终start==end。同时assign max=A[start] or A[end]

往上,parent一层的max:就是比较左右孩子,其实都是在两个sub-tree里面比较sub-tree的max。

这就好做了：
先分，找到left/right，比较max,在create current node,再append到当前node上面。

实际上是depth-first, 自底向上建立起的。

---

256. Segment Tree Build.java (https://github.com/shawnfan/LintCode/blob/master/Java/Segment%20Tree%20Build.java) Level: Medium

按定义：
左孩子：（A.left, (A.left+A.rigth)/2）
右孩子：（(A.left+A.rigth)/2 + 1， A.right）

---

257. Segment Tree Modify.java (https://github.com/shawnfan/LintCode/blob/master/Java/Segment%20Tree%20Modify.java) Level: Medium

Recursively 在segment tree里面找index, update it with value.

每个iteration，很可能（要么左手，要么右手）max就变了。所以每次都left.max and right.max compare一下。
最后轮回到头顶，头顶一下包括头顶，就全部都是max了。

Divde and Conquer

---

258. Segment Tree Query II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Segment%20Tree%20Query%20II.java) Level: Medium

和 Segment Tree Query I 以及其他Segment Tree问题没啥区别。这个就是return个count。

这个题目考了validate input source：input 的start,end可能超出root[start,end]。
那么第一步就要先clear一下。完全不在range就return 0. 有range重合就规整到root的range.

---

259. Segment Tree Query.java (https://github.com/shawnfan/LintCode/blob/master/Java/Segment%20Tree%20Query.java) Level: Medium

给了segment Tree, node里面有Max value, 找[start,end]里面的max

[start,end]跟mid相比，可能：
全在mid左
全在mid右
包含了mid：这里要特别break into 2 query method

按定义：
mid = (root.start + root.end)/2

---

260. Serilization and Deserialization Of Binary Tree.java
(https://github.com/shawnfan/LintCode/blob/master/Java/Serilization%20and%20Deserialization%20Of%20Binary%20Tree.java)Design an algorithm and write code to serialize and deserialize a binary tree. Writing the tree to a file is called 'serialization' and reading back from the file to reconstruct the exact same binary tree is 'deserialization'.

There is no limit of how you deserialize or serialize a binary tree, you only need to make sure you can serialize a binary tree to a string and deserialize this string to the original structure.

Example
An example of testdata: Binary tree {3,9,20,#,#,15,7}, denote the following structure:

```
3
```

/ \
9 20
/ \
15 7
Our data serialization use bfs traversal. This is just for when you got wrong answer and want to debug the input.

You can use other method to do serializaiton and deserialization.

Tags Expand
Binary Tree

Thinking process:

1. Carefully turn the binary tree into a string: use pre-order in this example.
2. Use a global variable to track the data(data string will be cut in different levels of recursion).
   The concept is very easy tho, just need to carefully code it up.

```
*/

/**
```

- Definition of TreeNode:
- public class TreeNode {
- public int val;
- public TreeNode left, right;
- public TreeNode(int val) {
- this.val = val;
- this.left = this.right = null;
- }
- }

```
/
class Solution {
    /*
```

- This method will be invoked first, you should design your own algorithm
- to serialize a binary tree which denote by a root node to a string which
- can be easily deserialized by your own "deserialize" method later.

```
    */
    public String serialize(TreeNode root) {
    if (root == null) {
```

```
        return "#,";
```

```
    }
    String mid = root.val + ",";
    String left = serialize(root.left);
    String right = serialize(root.right);
    mid += left + right;
    return mid;
    }

    private String data = "";
    /**
```

- This method will be invoked second, the argument data is what exactly
- you serialized at method "serialize", that means the data is not given by
- system, it's given by your own serialize method. So the format of data is
- designed by yourself, and deserialize it here as you serialize it in
- "serialize" method.

```
*/
public TreeNode deserialize(String data) {
this.data = data;
return desHelper();
}


public TreeNode desHelper() {
if (this.data.indexOf("#,") == 0) {
```

```
this.data = this.data.substring(this.data.indexOf(",") + 1);
return null;
```

```
}
String midVal = this.data.substring(0, this.data.indexOf(","));
TreeNode mid = new TreeNode(Integer.parseInt(midVal));
this.data = this.data.substring(this.data.indexOf(",") + 1);
TreeNode left = desHelper();
TreeNode right = desHelper();
mid.left = left;
mid.right = right;
return mid;
}
}
```

---

261. Single Number II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Single%20Number%20II.java)Given 3*n + 1 numbers, every numbers occurs triple times except one, find it.
Example
Given [1,1,2,3,3,3,2,2,4,1] return 4

Challenge

One-pass, constant extra space

Thinking process:

Still using bit manipulation. We need to erase all of the 3-appearance number and leave the single number out. A few steps:

Store the final result by continuously bit OR with the result variable.

Want to XOR the 3 numbers, but can't erase them as if only 2 duplicate numbers:Consider the number as 3-based number, so XOR can be understand this way when add 3 numbers together, add each individual bit. If the sum is 3, then set it as 0. If not 3, leave as is.

1. Store the bits in a integer array, which simulates a binary version of the integer

2. When each bit's XOR process finishes, bit OR it with result

    */

```java
public class Solution {
public int singleNumberII(int[] A) {
if (A == null || A.length == 0) {
return -1;
}
//present the XOR results in binary format
int[] bits = new int[32];
int rst = 0;
for (int i = 0; i < 32; i++) {
for (int j = 0; j < A.length; j++){
//XOR the numbers in a 3-base fashion. Whenever bit[i] has a number 3, set it back to 0.
bits[i] += A[j] >> i & 1;
bits[i] %= 3;
}
//OR it to the result. However, each time only the i - spot is updated with the bits[i].
rst |= bits[i] << i;
}
return rst;
}
}
```

---

262. Single Number III.java (https://github.com/shawnfan/LintCode/blob/master/Java/Single%20Number%20III.java)Given 2*n + 2 numbers, every numbers occurs twice except two, find them.

Example

Given [1,2,2,3,4,4,5,3] return 1 and 5

Challenge

O(n) time, O(1) extra space.

Thinking Process:

The 2 exception must have this feature: a ^ b != 0, since they are different

Still want to do 2n + 1 problem as in Single Number I, then we need to split a and b into 2 groups and deal with two 2n+1 problems

Assume c = a^b, there mush be a bit where a and b has the difference, so that bit in c is 1.

Find this bit position and use it to split the group: shift number in the array by 'bit-position' indexes. If the shifted number has 1 at the 'bit-position', set it to one group; otherwise to another group.

*/

public class Solution {
/**

```
  * @param A : An integer array
  * @return : Two integers
  */
public List<Integer> singleNumberIII(int[] A) {
    if (A == null || A.length == 0) {
        return null;
    }
    List<Integer> rst = new ArrayList<Integer>();
    int xor = 0;
    for (int i = 0; i < A.length; i++) {
        xor ^= A[i];
    }
    int bitOnePos = 0;
    for (int i = 0; i < 32; i++) {
        if ((xor >> i & 1) == 1) {
            bitOnePos = i;
        }
    }
    int rstA = 0;
    int rstB = 0;
    for (int i = 0; i < A.length; i++) {
        if ((A[i] >> bitOnePos & 1) == 1) {
            rstA ^= A[i];
        } else {
            rstB ^= A[i];
        }
    }
    rst.add(rstA);
    rst.add(rstB);
    return rst;
}
```

}

263. Single Number.java (https://github.com/shawnfan/LintCode/blob/master/Java/Single%20Number.java)62% Accepted

Given 2*n + 1 numbers, every numbers occurs twice except one, find it.

Example

Given [1,2,2,1,3,4,3], return 4

Challenge

One-pass, constant extra space

Tags Expand

Greedy

Manipulate bits:

Thinking process:

One-pass and constant extra space.

since all numbers appears twice, consider them as in bits format. Two identical number XOR will be zero. If we XOR everything double-numbers together, it will be zero. At the end, we use o XOR our target number, the result is actually the target number.

Very smart trick to use bits.

In order to compare from index 0 to the end, we need to extract index 0 first as result before for loop. And start for loop at i = 1.

*/

```java
public class Solution {
/* @param A : an integer array
return : a integer /
public int singleNumber(int[] A) {
if (A == null || A.length == 0) {
return 0;
}
int rst = A[0];
for (int i = 1; i < A.length; i++) {
rst = rst ^ A[i];
}
return rst;
}
}
```

264. Singleton.java (https://github.com/shawnfan/LintCode/blob/master/Java/Singleton.java)Singleton is a most widely used design pattern. If a class has and only has one instance at every moment, we call this design as singleton. For example, for class Mouse (not a animal mouse), we should design it in singleton.

You job is to implement a getInstance method for given class, return the same instance of this class every time you call this method.

Example
In Java:

A a = A.getInstance();
A b = A.getInstance();
a should equal to b.

Challenge
If we call getInstance concurrently, can you make sure your code could run correctly?

Tags Expand
LintCode Copyright OO Design

Thoughts:
...
Was not clear to me. Need to loop up more on synchronized/volatile
Good reference:
http://www.cnblogs.com/EdwardLiu/p/4443230.html (http://www.cnblogs.com/EdwardLiu/p/4443230.html)

*/

class Solution {
public static volatile Solution solution = null;
/**

```
  * @return: The same instance of this class every time
  */
public static Solution getInstance() {
    if (solution == null) {
        synchronized (Solution.class) {
            // Double check
            if (solution == null) {
                solution = new Solution();
            }
        }
    }
    return solution;
}
```

};

265. Sliding Window Maximum.java (https://github.com/shawnfan/LintCode/blob/master/Java/Sliding%20Window%20Maximum.java)每次把小于当前node的，全部剔除，剩下的，自然就是:最大的>第二大的>第三大的...ETC.

为啥可以不管不无地剔除？

因为我们只在乎最大值的存在；而任何小于当前（正要新就加进去的）值的，反正以后也成不了最大值，于是扔掉！

---

266. Sliding Window Median.java (https://github.com/shawnfan/LintCode/blob/master/Java/Sliding%20Window%20Median.java)移动窗口2step：

1. 加一个数。

2. 减一个数。

   加减时看好，是从前面的maxheap里面抽，还是从后面的minHeap里面抽。

   抽完balance一下。

记得:
左边的maxHeap总有 x+1或者x个数字。
后边minHeap应该一直有x个数字。

---

267. Sort Color.java (https://github.com/shawnfan/LintCode/blob/master/Java/Sort%20Color.java)Given an array with n objects colored red, white or blue, sort them so that objects of the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.

Example
Note
You are not suppose to use the library's sort function for this problem.

Clarification
Follow up:
A rather straight forward solution is a two-pass algorithm using counting sort.
First, iterate the array counting number of 0's, 1's, and 2's, then overwrite array with total number of 0's, then 1's and followed by 2's.

Could you come up with an one-pass algorithm using only constant space?

Tags Expand
Two Pointers Sort Array

Thoughts;
A easier version of Sort ColorII. Using the exact same code with different k number. Note, now k is start from 0.

```
*/

class Solution {
/**
  * @param nums: A list of integer which is 0, 1 or 2
  * @return: nothing
  */
public void sortColors(int[] colors) {
    if (colors == null || colors.length == 0) {
        return;
    }
    int end = colors.length - 1;
    int k = 2; // 3 different colors
    for (int i = 0; i < k; i++) {
        end = helper(colors, 0, end, k - i - 1);
    }
}

public void swap(int[] colors, int x, int y){
    int temp = colors[x];
    colors[x] = colors[y];
    colors[y] = temp;
}

public int helper(int[] colors, int start, int end, int pivot) {
    int low = start;
    int high = end;
    while (low <= high) {
        while(low < high && colors[low] <= pivot) {
            low++;
        }
        while(high > 0 && colors[high] > pivot) {
            high--;
        }
        if (low <= high) {
            swap(colors, low, high);
            low++;
            high--;
        }
    }
    return low - 1;
}

}
```

268. Sort Colors II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Sort%20Colors%20II.java)Given an array of n objects with k different colors (numbered from 1 to k), sort them so that objects of the same color are adjacent, with the colors in the order 1, 2, ... k.

Example
GIven colors=[3, 2, 2, 1, 4], k=4, your code should sort colors in-place to [1, 2, 2, 3, 4].

Note
You are not suppose to use the library's sort function for this problem.

Challenge
A rather straight forward solution is a two-pass algorithm using counting sort. That will cost O(k) extra memory.

Can you do it without using extra memory?

Tags Expand
Two Pointers Sort

Thoughts (Need to revist and think about this, very interesting)
Doing quick sort partition for K -1 times.

 1. Use K - 1 value as pivot
 2. Starting from 0, whenever low<high && less or equal to pivot, low++
 3. starting from end, whenever high >0, and greater than pivot, high--
 4. Result: only swap when low and high have disagreement on the pivot value.

*/

class Solution {
/**

```
 * @param colors: A list of integer
 * @param k: An integer
 * @return: nothing
 */
public void sortColors2(int[] colors, int k) {
    if (colors == null || colors.length == 0 || k <= 0) {
        return;
    }
    int end = colors.length - 1;
    for (int i = 0; i < k - 1; i++) {
        end = helper(colors, 0, end, k - i - 1);
    }
}


public void swap(int[] colors, int x, int y){
    int temp = colors[x];
    colors[x] = colors[y];
    colors[y] = temp;
}


public int helper(int[] colors, int start, int end, int pivot) {
    int low = start;
    int high = end;
    while (low <= high) {
        while(low < high && colors[low] <= pivot) {
            low++;
        }
        while(high > 0 && colors[high] > pivot) {
            high--;
        }
        if (low <= high) {
            swap(colors, low, high);
            low++;
            high--;
        }
    }
    return low - 1;
}
```

}

---

269. Sort Letters by Case.java (https://github.com/shawnfan/LintCode/blob/master/Java/Sort%20Letters%20by%20Case.java)Given a string which contains only letters. Sort it by lower case first and upper case second.

Example
For "abAcD", a reasonable answer is "acbAD"

Note

It's not necessary to keep the original order of lower-case letters and upper case letters.

Challenge

Do it in one-pass and in-place.

Tags Expand

String Two Pointers LintCode Copyright Sort

Thoughts:

Another two pointer sorting.

Difference: use a ASCII code 'a' as the pivot. all the letters that from a ~ z

have bigger integer values, and A~Z have small integer values.

This problem requires lowcase+upperCase, so we'd sort the list from high to low.

NOTE: in the 2 while loop, the it's always having ">='

*/

```
public class Solution {
/* @param chars: The letter array you should sort by Case
@return: void /
public void sortLetters(char[] chars) {
if (chars == null || chars.length == 0) {
return;
}
char pivot = 'a';
int start = 0; int end = chars.length - 1;
while (start <= end) {
while (start <= end && chars[start] >= pivot) {
start++;
}
while (start <= end && chars[end] < pivot) {
end--;
}
if (start <= end) {
char temp = chars[end];
chars[end] = chars[start];
chars[start] = temp;
start++;
```

```
end--;
}
}
}
}
```

270. Sort List.java (https://github.com/shawnfan/LintCode/blob/master/Java/Sort%20List.java) 1. find middle. 快慢指针

```
2. Merge:  假设given list A, B 已经是sorted，然后按照大小，混合。
3. Sort: 切开两半，先sort前半，如果先sort了mid.next~end, sort后，中间点mid.next == null，再sort前半段。
   然后mege.
   要recursively call itself.
```

Quick sort:

想做可以看讲义：http://www.jiuzhang.com/solutions/sort-list/ (http://www.jiuzhang.com/solutions/sort-list/)

但是quick sort不建议用在list上面。

排列 list, merge sort可能更可行和合理。原因分析在下面， 以及： http://www.geeksforgeeks.org/why-quick-sort-preferred-for-arrays-and-merge-sort-for-linked-lists/ (http://www.geeksforgeeks.org/why-quick-sort-preferred-for-arrays-and-merge-sort-for-linked-lists/)

271. Space Replacement.java (https://github.com/shawnfan/LintCode/blob/master/Java/Space%20Replacement.java)Write a method to replace all spaces in a string with %20. The string is given in a characters array, you can assume it has enough space for replacement and you are given the true length of the string.

Example
Given "Mr John Smith", length = 13.

The string after replacement should be "Mr%20John%20Smith".

Note
If you are using Java or Python , please use characters array instead of string.

Challenge
Do it in-place.

Tags Expand
String Cracking The Coding Interview

Thoughts:

Overriding the array from the back to front.

This is because as we re-writing the string from the back, stuff at head of the string does not change yet.

This is wonderful:)

*/

public class Solution {
/**

```
  * @param string: An array of Char
  * @param length: The true length of the string
  * @return: The true length of new string
  */
 public int replaceBlank(char[] string, int length) {
     if (string == null || string.length == 0) {
         return 0;
     }
     int count = 0;
     for (char c : string) {
         if (c == ' ') {
             count += 2;
         }
     }
     int lastIndex = length + count - 1;
     //from back to front:
     for (int i = length - 1; i >= 0; i--) {
         if (string[i] == ' ') {
             string[lastIndex--] = '0';
             string[lastIndex--] = '2';
             string[lastIndex--] = '%';
         } else {
             string[lastIndex--] = string[i];
         }
     }
     return length + count;
 }
```

}

---

272. Sqrt(x).java (https://github.com/shawnfan/LintCode/blob/master/Java/Sqrt(x).java)Implement int sqrt(int x).

Compute and return the square root of x.

Example

sqrt(3) = 1

sqrt(4) = 2

sqrt(5) = 2

sqrt(10) = 3

Challenge

O(log(x))

Tags Expand

Binary Search

Thinking process:

Binary search. While loop until the head and tail meets.
*/

class Solution {
/**

```java
 * @param x: An integer
 * @return: The sqrt of x
 */
public int sqrt(int x) {
    long start = 0;
    long end = x;
    while (end >= start) {
        long mid = start + (end - start) / 2;
        if (mid * mid > x) {
            end = mid - 1;
        } else if (mid * mid < x) {
            start = mid + 1;
        } else {
            return (int)mid;
        }
    }
//When start > end, while loop ends. That means, end must be the largest possible integer that end^2 is closest to x.
    return (int)end;
}
```

}

273. Stone Game.java (https://github.com/shawnfan/LintCode/blob/master/Java/Stone%20Game.java)NOT DONE YET

---

274. String to Integer(atoi).java (https://github.com/shawnfan/LintCode/blob/master/Java/String%20to%20Integer(atoi).java) Level: Easy

方法1: 问清情况，一点一点把case都涉及到。

方法2: 用regular expression。if (!str.matches("[+-]?(?:\d+(?:\.\d*)?|\.\d+)")). 猛了一点

---

275. Strobogrammatic Number II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Strobogrammatic%20Number%20II.java)难的case先不handle.到底之后来一次O(n) scan.
总共的时间起码是O(n/2) + O(n), 所以还是O(n)

---

276. Strobogrammatic Number.java (https://github.com/shawnfan/LintCode/blob/master/Java/Strobogrammatic%20Number.java)A strobogrammatic number is a number that looks the same when rotated 180 degrees (looked at upside down).
Write a function to determine if a number is strobogrammatic. The number is represented as a string.
For example, the numbers "69", "88", and "818" are all strobogrammatic.
Tags: Hash Table Math
Similar Problems: (M) Strobogrammatic Number II, (H) Strobogrammatic Number III
*/

/*
OPTS 11.04.2015
Thoughts:
Because the symmetric pairs are:
1-1, 8-8,0-0,6-9,9-6, we make a hashmap of it.
Create left/right pointer, where each compare has to match the pair in hashmap.

Note:
On map.containsKey() line, need to check (right), or whichever item that map is going to map.get(...) afterwards.
If containsKey fails, return false; only when it passes through, then proceed to mpa.get()
*/
public class Solution {
public boolean isStrobogrammatic(String num) {
if (num == null || num.length() == 0) {
return true;
}
HashMap<Character, Character> map = new HashMap<Character,Character>();

```
map.put('0','0');
map.put('1','1');
map.put('8','8');
map.put('6','9');
map.put('9','6');
int left = 0;
int right = num.length() - 1;
while (left <= right) {
if (!map.containsKey(num.charAt(right)) || num.charAt(left) != map.get(num.charAt(right))) {
return false;
}
left++;
right--;
}
return true;
}
}


/
Thoughts:
Compare digits to the symmetric postion; special care for (6,9) pair, mark it after comparision.
Elimite the cases before the for-loop run through(can do it in or as well, but that just make the code a bit complex)
Note:
Didn't use HashMap. I beleive hash map is used to mark the spot? /
public class Solution {
public boolean isStrobogrammatic(String num) {
if (num == null || num.length() == 0) {
return true;
}
//Any non-strobogrammatic
if (num.indexOf("2") >= 0 || num.indexOf("3") >= 0 ||
num.indexOf("4") >= 0 || num.indexOf("5") >= 0 ||
num.indexOf("7") >= 0) {
return false;
}
//If only 6 or 9 exist:
if ((num.indexOf("6") >= 0 && num.indexOf("9") < 0) ||
(num.indexOf("9") >= 0 && num.indexOf("6") < 0)) {
return false;
```

```
}
//Check if (6,9) or other strobogrammatic # are appearing at symmetric position
char[] arr = num.toCharArray();
int leng = num.length();
for (int i = 0; i < leng; i++) {
if (arr[i] == '6' || arr[i] == '9') {
if ((arr[i] == '6' && arr[leng – i – 1] != '9') ||
(arr[i] == '9' && arr[leng – i – 1] != '6')) {
return false;
}
arr[i] = arr[leng – i – 1] = 'M';//marker
} else if (arr[i] != 'M' && arr[i] != arr[leng – i – 1]) {
return false;
}
}
return true;
}
}
```

---

277. StrStr.java (https://github.com/shawnfan/LintCode/blob/master/Java/StrStr.java)StrStr:
strStr My Submissions

19% Accepted
strstr (a.k.a find sub string), is a useful function in string operation. You task is to implement this function.

For a given source string and a target string, you should output the "first" index(from 0) of target string in source string.

If target is not exist in source, just return -1.

Example
If source="source" and target="target", return -1.

If source="abcdabcdefg" and target="bcd", return 1.

Challenge
O(n) time.

Clarification
Do I need to implement KMP Algorithm in an interview?

Tags Expand
Basic Implementation String

Thinking process:
Two Pointer.
String Null case.
Break Statement.

Check position i+j of source and position j of target. If not match, break out.
If j matches target.length(), means target is fully embedded in source.
return start point of target in source: i
*/
public int strStr(String source, String target) {
//Check Null
if(source == null || target == null){
return -1;
}
//Two Pointer check for target
int i,j;
for (i = 0; i < source.length() - target.length() + 1; i++){
for (j = 0; j < target.length(); j++){
if (source.charAt(i+j) != target.charAt(j)){
break;
}
}
if(j == target.length()){
return i;
}
}
//'target' not found:
return -1;
}

---

278. Subarray Sum Closest.java (https://github.com/shawnfan/LintCode/blob/master/Java/Subarray%20Sum%20Closest.java) Level: Medium

?

279. Subarray Sum.java (https://github.com/shawnfan/LintCode/blob/master/Java/Subarray%20Sum.java) Level: Easy

分析出，如果sum[0~a]=x, 然后sum[0~b]=x, 说明sum(a ~ b) = 0.

这样理解后，用`hashMap`存每个`sum[0~i]`的值和`index i`. 如果有重复，就找到了一组sum为`0`的数组。

280. Subset.java (https://github.com/shawnfan/LintCode/blob/master/Java/Subset.java) Level: Medium

最基本的递归题目。
坑：记得一开头sort一下 nums。 因为要升序。那么整体就是O(nlogn)

注意：用level/index来track到哪一步。最后一level就add into rst

方法1: subset的概念，取或者不取,backtracking. 当level/index到底，return 一个list.

方法2: 用for loop backtracking. 记得：每个dfs recursive call是一种独特可能，先加进rst。

recap:时间久了忘记dfs的两种路子. for loop dfs/backtracking vs. regular dfs

281. Subsets II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Subsets%20II.java) Level: Medium

递归：找准需要pass along的几个数据结构。

和SubsetI类似，先sort input, 然后递归。但是input可能有duplicates.

Using for loop approach: 每个dfs call是一种可能性，直接add into result.
为了除去duplicated result, 如果在递归里面用rst.contains(),就是O(n), which makes overall O(n^2).

这里有个基于sorted array的技巧：
因为我们有mark index。 一旦for loop里面的i!=index，并且nums[i] == nums[i-1],说明x=nums[i-1]已经用过，不需要再用一次：
[a,x1,x2] , x1==x2
i == index -> [a,x1]
i == index + 1 -> [a,x2]. 我们要skip这一种。

如果需要[a,x1,x2]怎么办？ 其实这一种在index变化时，会在不同的两个dfs call 里面涉及到。

Iterative: 写一写，用个Queue. Not recommended, Again, rst.contains() cost too much.

282. Subtree.java (https://github.com/shawnfan/LintCode/blob/master/Java/Subtree.java) Level: Easy

找到potential subtree, 比较Children.

一点注意：即使找到T1 == T2, 但很可能只是数字相同（这里不是binary search tree!!），而children不同。所以同时要继续recursively isSubtree(T1.left, T2) ...etc.

---

283. Summary Ranges.java (https://github.com/shawnfan/LintCode/blob/master/Java/Summary%20Ranges.java)Given a sorted integer array without duplicates, return the summary of its ranges.

For example, given [0,1,2,4,5,7], return ["0->2","4->5","7"].

Tags: Array
Similar Problems: (M) Missing Ranges

```
/
/
Thoughts: basic implementation, use a arraylist to catch candidates.
Detect condition, and return results.
*/
public class Solution {
public List<String> summaryRanges(int[] nums) {
List<String> rst = new ArrayList<String>();
if (nums == null || nums.length == 0) {
return rst;
}
ArrayList<Integer> list = new ArrayList<Integer>();
for (int i = 0; i < nums.length; i++) {
list.add(nums[i]);
if (i + 1 == nums.length || nums[i] + 1 != nums[i + 1]) {
if (list.size() == 1) {
rst.add(list.get(0) + "");
} else {
rst.add(list.get(0) + "->" + list.get(list.size() - 1));
}
list = new ArrayList<Integer>();
}
}
```

```
    return rst;
    }
}


//O(n)
```

---

284. Surrounded Regions.java (https://github.com/shawnfan/LintCode/blob/master/Java/Surrounded%20Regions.java)Given a 2D board containing 'X' and 'O', capture all regions surrounded by 'X'.

A region is captured by flipping all 'O's into 'X's in that surrounded region.

For example,
X X X X
X O O X
X X O X
X O X X
After running your function, the board should be:

X X X X
X X X X
X X X X
X O X X
Hide Tags Breadth-first Search

Thinking Process:
Since dfs does not work, try bfs.
Very similar to DFS, however, when checking the 4 bounaries:

 1. chcek the curruent point.

 2. Add surrounding points into a queue.

 3. Deal with the queue immediately via a while loop

```
/
public class Solution {
private char[][] board;
private int row;
private int col;
private char target;
private char mark;
```

```java
private Queue<Integer> queue = new LinkedList<Integer>();
public void solve(char[][] board) {
if (board == null || board.length == 0) {
return;
}
this.board = board;
row = board.length;
col = board[0].length;
target = 'O';
mark = 'M';
//Check the board
for (int i = 0; i < row; i++) {
for (int j = 0; j < col; j++) {
if (i == 0 || i == row - 1 || j == 0 || j == col - 1) {
check(i,j);
}
}
}
//Replacement
for (int i = 0; i < row; i++) {
for (int j = 0; j < col; j++) {
if (board[i][j] == target) {
board[i][j] = 'X';
}
if (board[i][j] == mark) {
board[i][j] = target;
}
}
}
}
//BFS
public void check(int i, int j) {
fill(board, i, j);
while(!queue.isEmpty()) {
int val = queue.poll();
int x = val / col;
int y = val % col;
fill(board, x - 1, y);
fill(board, x + 1, y);
```

```
fill(board, x, y - 1);
fill(board, x, y + 1);
}
}
public void fill(char[][] board, int i, int j) {
if (i < 0 || i >= row || j < 0 || j >= col || board[i][j] != target) {
return;
}
board[i][j] = mark;
queue.offer(i col + j);
}
}
```

```
/*
Thinking process:
Using DFS.

 1. Whenever the edge has an 'O', all touching point with 'O' will be non-surrounded by 'X'. SO check the 4 bounds first. Mark all non-surrounded point as M.
 2. Replace all remaining 'O' with 'X'
 3. Replace 'M' with 'O'
    However, in the LeetCode test, DFS gives stack overflow. So we'd use BFS instead.
    */
```

```
/*

//The following is using DFS, but gives Stackoverflow.
public class Solution {
private char[][] board;
private int row;
private int col;
private char target;
private char mark;
public void solve(char[][] board) {
if (board == null || board.length == 0) {
return;
}
this.board = board;
target = 'O';
```

```
mark = 'M';
row = board.length;
col = board[0].length;
```

```
    //check bound
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            if (i == 0 || j == 0 || i == row - 1 || j == col - 1) {
                check(i, j);
            }
        }
    }
    //1. replace remaining target with 'x'
    //2. replace all mark with 'O'
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            if (board[i][j] == target) {
                board[i][j] = 'X';
            } else if (board[i][j] == mark) {
                board[i][j] = 'O';
            }
        }
    }
}

public void check(int i, int j) {
    if (i < 0 || j < 0 || i > row - 1 || j > col - 1) {
        return;
    }
    if (board[i][j] == target) {
        board[i][j] = mark;
        check(i - 1, j);
        check(i + 1, j);
        check(i, j - 1);
        check(i, j + 1);
    }
}
```

}

*/

---

285. Swap Nodes in Pairs.java (https://github.com/shawnfan/LintCode/blob/master/Java/Swap%20Nodes%20in%20Pairs.java)画三个block, 1,2,3. 连线。

---

286. Symmetric Binary Tree.java (https://github.com/shawnfan/LintCode/blob/master/Java/Symmetric%20Binary%20Tree.java) Level: Easy

注意Symmetric Binary Tree的例子和定义: 是镜面一样的对称. 并不是说左右两个sub-tree相等。

方法1: Recursively check symmetrically相对应的Node. 每个node的children都和镜面另外一边相对的node的children刚好成镜面反射位置。

方法2: 用stack. 左手边sub-tree先加left,再加right child; 右手边sub-tree先加right child, 再加left child。
process时，若symmetric，所有stack里面出来的node会一一对应。

---

287. The Smallest Difference.java (https://github.com/shawnfan/LintCode/blob/master/Java/The%20Smallest%20Difference.java)Given two array of integers(the first array is array A, the second array is array B), now we are going to find a element in array A which is A[i], and another element in array B which is B[j], so that the difference between A[i] and B[j] (|A[i] - B[j]|) is as small as possible, return their smallest difference.

Example
For example, given array A = [3,6,7,4], B = [2,8,9,3], return 0

Challenge
O(n log n) time

Tags Expand
Two Pointers LintCode Copyright Sort Array
*/

/
Thoughts:
Sort A, B. O(nLogn)
Use smaller array to binarh in longer array. n logn

*/

public class Solution {
/**

```java
 * @param A, B: Two integer arrays.
 * @return: Their smallest difference.
 */
public int smallestDifference(int[] A, int[] B) {
    if (A == null || A.length == 0 || B == null || B.length == 0) {
        return 0;
    }
    if (A.length > B.length) {
        int[] temp = A;
        A = B;
        B = temp;
    }
    Arrays.sort(A);
    Arrays.sort(B);
    int diff = Integer.MAX_VALUE;

    for (int i = 0; i < A.length; i++) {//10
        int start = 0;
        int end = B.length - 1;
        int mid;
        //Small enhancement
        if (B[start] >= A[A.length - 1]) {
            return B[start] - A[A.length - 1];
        }
        if (A[start] >= B[B.length - 1]) {
            return A[start] - B[B.length - 1];
        }
        while (start + 1 < end) {
            mid = start + (end - start)/2;
            if (B[mid] == A[i]) {
                return 0;
            } else if (mid - 1 >= 0 && B[mid - 1] < A[i] && A[i] < B[mid]) {
                diff = Math.min(diff, Math.min(A[i] - B[mid - 1], B[mid] - A[i]));
                break;
            } else if (mid + 1 < B.length - 1 && B[mid] < A[i] && A[i] < B[mid + 1]) {
                diff = Math.min(diff, Math.min(A[i] - B[mid], B[mid + 1] - A[i]));
                break;
            } else if (B[mid] > A[i]) {
                end = mid;
            } else {
                start = mid;
            }

        }//end while
        if (start + 1 >= end) {
            int min = Math.min(Math.abs(A[i] - B[start]), Math.abs(A[i] - B[end]));
            diff = Math.min(diff, min);
        }
    }//end for
    return diff;
}
```

}

---

288. Top K Frequent Words.java (https://github.com/shawnfan/LintCode/blob/master/Java/Top%20K%20Frequent%20Words.java) Level: Medium

方法1：Brutle force用HashMap存frequency, 用ArrayList存lists of words。最后返回从尾部向前数的k个。
注意排序时Collection.sort()的cost是O(nLogk)

方法1-1: 还是用HashMap,但create一个Node class, 然后用PriorityQueue.
PriorityQueue里面用到了 String.compareTo(another String).巧妙。

方法2: Trie && MinHeap屌炸天
http://www.geeksforgeeks.org/find-the-k-most-frequent-words-from-a-file/ (http://www.geeksforgeeks.org/find-the-k-most-frequent-words-from-a-file/)

---

289. Topological Sorting.java (https://github.com/shawnfan/LintCode/blob/master/Java/Topological%20Sorting.java) Level: Medium

比较特别的BFS.

几个graph的condition：

1. 可能有多个root
2. directed node, 可以direct backwards.

Steps:
Track all neighbors/childrens. 把所有的children都存在map<label, count>里面
先把所有的root加一遍，可能多个root。并且全部加到queue里面。

然后以process queue, do BFS:
Only when map.get(label) == 0, add into queue && rst.
这用map track apperance, 确保在后面出现的node, 一定最后process.

---

290. Total Occurrence of Target.java (https://github.com/shawnfan/LintCode/blob/master/Java/Total%20Occurrence%20of%20Target.java)找total number of occurance. 首先找first occurance, 再找last occurance.

---

291. Trailing Zeros.java (https://github.com/shawnfan/LintCode/blob/master/Java/Trailing%20Zeros.java)Write an algorithm which computes the number of trailing zeros in n factorial.

Example

11! = 39916800, so the out should be 2

Challenge

O(log N) time

Tags Expand

Mathematics

Thoughts:

Attempt1:

Can this problem be converted to : how many 10's to I have?

Loop through n, and check how many 2s, 5s do we have.

For each i, do while loop and count the number of 2s, and 5s in that particular i.

Note:

5 and 2 makes 10. So don't worry about 10.

Some values will be checked redundantly, so record the ones checked, return the hash value directly.

Attempt2:

Don't even need to worry about 2's because 2 is definitely more than 5's. Only need to care about 5's.

How many 5's? n/5. loop (1 ~ n)

However, some number within (1 ~ n) may give more 5's, which for example is: 25 = 5 * 5, double 5's. And 125 = triple 5's.

In fact count = n / 5 + n / 25 + n / 125 + ...

/

class Solution {

/

```
 * param n: As desciption
 * return: An integer, denote the number of trailing zeros in n!
 */
public long trailingZeros(long n) {
    if ( n < 5) {
        return 0;
    }
    long count = 0;
    for (long i = 5; n / i != 0; i *= 5) {
        count += n / i;
    }
    return count;
}

}


/*
//Attempt 1:
//This solution exceed time limit, and it's over-complex. 滚粗。
class Solution {
```

```java
    private HashMap<Long, Long> mapTwo = new HashMap<Long, Long>();
    private HashMap<Long, Long> mapFive = new HashMap<Long, Long>();
    public long trailingZeros(long n) {
        if (n < 5) {
            return 0;
        }
        long countFive = 0;
        long countTwo = 0;
        for (int i = 1; i <= n; i++) {
            if (i % 2 == 0) {
                countTwo += countExistance(i, 2, mapTwo);
            }
            if (i % 5 == 0) {
                countFive += countExistance(i, 5,mapFive);
            }
        }
        return (countFive < countTwo) ? countFive : countTwo;
    }
    public long countExistance(long n, long m, HashMap<Long, Long> map) {
        long temp = n;
        long count = 0;
        double num = (double)n;
        while (num / m == n / m) {
            count++;
            n = n / m;
            num = (double)n;
            if (map.containsKey(n)) {
                count += map.get(n);
                break;
            }
        }
        if (!map.containsKey(temp)) {
            map.put(temp, count);
        }
        return count;
    }
```

};

*/

---

292. Trapping Rain Water II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Trapping%20Rain%20Water%20II.java) Level: Hard

用PriorityQueue把选中的height排序。为走位，create class Cell {x,y, height}.

注意几个理论：

1. 从matrix四周开始考虑，发现matrix能Hold住的水，取决于height低的block。
2. 必须从外围开始考虑，因为水是被包裹在里面，外面至少需要现有一层。

以上两点就促使我们用min-heap: 也就是natural order的PriorityQueue<Cell>.

process的时候，画个图也可以搞清楚，就是四个方向都走走，用curr cell的高度减去周围cell的高度。 若大于零，那么就有积水。

每个visited的cell都要mark. 去到4个方向的cell,加进queue里面继续process.

这里，有一点，和trapping water I 想法一样。刚刚从外围，只是能加到跟外围cell高度一致的水平面。往里面，很可能cell高度变化。
这里要附上curr cell 和 move-to cell的最大高度。

---

293. Trapping Rain Water.java (https://github.com/shawnfan/LintCode/blob/master/Java/Trapping%20Rain%20Water.java) Level: Medium

2 Pointers ， 双面夹击：

1. 找中间最高bar的index
2. 两面往中心扫：每次加上（topBarIndex - currIndex）* (elevation from previous index).也就是每次加一个横条。
3. 每次还要减去block自身的height。

---

294. Triangle Count.java (https://github.com/shawnfan/LintCode/blob/master/Java/Triangle%20Count.java)Given an array of integers, how many three numbers can be found in the array, so that we can build an triangle whose three edges length is the three numbers that we find?

Example
Given array S = [3,4,6,7], return 3. They are:

[3,4,6]
[3,6,7]
[4,6,7]
Given array S = [4,4,4,4], return 4. They are:

[4(1),4(2),4(3)]
[4(1),4(2),4(4)]
[4(1),4(3),4(4)]
[4(2),4(3),4(4)]
Tags Expand
Two Pointers LintCode Copyright

*/

```
/*
Thoughts:
Pick 3 integers that fits the condition:
A + B > C
B + C > A
A + C > B
If we sort the input, then we know A <= B <= C, so we can remove 2 conditoins above and only have:
A + B > C
That is, Pick one C, and pick two integers A,B in front. Similar to TWO SUM II.
Have a fixed C as target, and find A + B > target in the remaining array on left of C.
How about just use 2 pointers left, right, and compare with a C (s[i] in for loop)
Time: O(n^2)

Note: don't forget to sort
*/

public class Solution {
/**
```

```java
 * @param S: A list of integers
 * @return: An integer
 */
public int triangleCount(int S[]) {
    if (S == null || S.length == 0) {
        return 0;
    }
    Arrays.sort(S);
    int count = 0;
    for (int i = 0; i < S.length; i++) {
        int left = 0;
        int right = i - 1; //at least 1 step left from C
        while (left < right){
            if (S[left] + S[right] > S[i]) {
                count += (right - left);
                right--;
            } else {//(S[left] + S[right] <= S[i])
                left++;
            }
        }
    }
    return count;
}
```

```
}
```

295. Tweaked Identical Binary Tree.java (https://github.com/shawnfan/LintCode/blob/master/Java/Tweaked%20Identical%20Binary%20Tree.java) Level: Easy

Recursive 比对左左,左右,右左，右右

---

296. Two Lists Sum.java (https://github.com/shawnfan/LintCode/blob/master/Java/Two%20Lists%20Sum.java)You have two numbers represented by a linked list, where each node contains a single digit.The digits are stored in reverse order, such that the 1's digit is at the head of the list.Write a function that adds the two numbers and returns the sum as a linked list.

Example
Given two lists, 3->1->5->null and 5->9->2->null, return 8->0->8->null

Tags Expand
Linked List Backtracking
//TODO: check 9chapter solution
Thinking process:
Simply add 2 lists' values together.
Handle the carrier
Use dummy node at beginning.
*/

/**

- Definition for singly-linked list.
- public class ListNode {
- int val;
- ListNode next;
- ListNode(int x) {
- val = x;
- next = null;
- }
- }
    /
    public class Solution {
    /*

    ○ @param l1: the first list
    ○ @param l2: the second list

- @return: the sum list of l1 and l2

```
*/
public ListNode addLists(ListNode l1, ListNode l2) {
ListNode rst = new ListNode(0);
ListNode dummy = rst;
int carrier = 0;
//while
while (l1 != null || l2 != null) {
```

```
if (l1 != null) {
    carrier += l1.val;
    l1 = l1.next;
}
if (l2 != null) {
    carrier += l2.val;
    l2 = l2.next;
}
rst.next = new ListNode(carrier % 10);
carrier = carrier / 10;
rst = rst.next;
```

```
}
//check the carrier
if (carrier == 1) {
```

```
rst.next = new ListNode(1);
```

```
}
return dummy.next;
}
}
```

---

297. Two Strings Are Anagrams.java (https://github.com/shawnfan/LintCode/blob/master/Java/Two%20Strings%20Are%20Anagrams.java) Level: Easy

方法1:char ascii 用count[256]
坑：不要想象这个是个26letter lowercase. may not be true.

方法2: 若是其他字符encoding, 而不只是utf16-encoding (java char)?
那么就继续用string去做

---

298. Ugly Number II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Ugly%20Number%20II.java)每次把dp[i-1]拿出来，不管三七二十一，分别乘以2,3,5. 出来的结果放进priority queue做比较。

最后时间是nlog(n3)

注意：
Long
HashSet确保没有重复。

---

299. Ugly Number.java (https://github.com/shawnfan/LintCode/blob/master/Java/Ugly%20Number.java) Level: Medium

方法1: PriorityQueue排序。用ArrayList check 新的ugly Number是否出现过。

方法1-1：(解释不通，不可取)用PriorityQueue排序。神奇的3，5，7走位：按照题目答案的出发，定了3，5，7以什么规律出现。但是题目并没有特殊表明。

方法2: DP . Not Done yet.

---

300. Unique Binary Search Tree II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Unique%20Binary%20Search%20Tree%20II.java)Given n, generate all structurally unique BST's (binary search trees) that store values 1...n.

Example
Given n = 3, your program should return all 5 unique BST's shown below.

```
1 3 3 2 1
 \ / / / \ \
 3 2 1 1 3 2
 / / \ \
 2 1 2 3
```
Tags Expand
Dynamic Programming Depth First Search

Thinking process:

- For a BST, root can be any node from node(1) to node(n).
- For each root, left nodes has mutiple forms of BST, and right node has mutiple forms of BST.
- For each root node, divide and conquer left / right

   */

/**

- Definition of TreeNode:
- public class TreeNode {
- public int val;
- public TreeNode left, right;
- public TreeNode(int val) {
- this.val = val;
- this.left = this.right = null;
- }
- }

```
/
public class Solution {
/*
```

  - @paramn n: An integer
  - @return: A list of root

```
*/
public List<TreeNode> generateTrees(int n) {
    return generate(1, n);
}
public ArrayList<TreeNode> generate(int start, int end) {
    ArrayList<TreeNode> rst = new ArrayList<TreeNode>();
    if (start > end) {
```

```
        rst.add(null);
        return rst;
```

```
    }
    for (int i = start; i <= end; i++){
```

```
        ArrayList<TreeNode> left = generate(start, i - 1);
        ArrayList<TreeNode> right = generate(i +  1, end);
        for (TreeNode l : left) {
            for (TreeNode r : right) {
                TreeNode root = new TreeNode(i);
                root.left = l;
                root.right = r;
                rst.add(root);
            }
        }
```

```
    }
```

```
        return rst;
    }
}
```

---

301. Unique Binary Search Tree.java (https://github.com/shawnfan/LintCode/blob/master/Java/Unique%20Binary%20Search%20Tree.java)Given n, how many structurally unique BST's (binary search trees) that store values 1...n?

Example
Given n = 3, there are a total of 5 unique BST's.

```
1 3 3 2 1
\ / / / \ \
3 2 1 1 3 2
/ / \ \
2 1 2 3
```
Tags Expand
Catalan Number Dynamic Programming

Thinking proces:
Knowing what is Catalan number.
C(n+1) = SUM(C(i)C(n-i))
OR: C(n) = SUM(C(i)C(n-i-1)).
*/

```
public class Solution {
    /**
```

```
 * @paramn n: An integer
 * @return: An integer
 */
public int numTrees(int n) {
    if (n <= 1) {
        return 1;
    }
    int[] count = new int[n + 1];
    count[0] = 1;
    count[1] = 1;
    for (int i = 2; i < n + 1; i++) {
        for (int j = 0; j < i; j++) {
            count[i] += count[j] * count[i - j - 1];
        }
    }
    return count[n];
}
```

}

---

302. Unique Characters.java (https://github.com/shawnfan/LintCode/blob/master/Java/Unique%20Characters.java)用hashSet, space O(n), time O(n)

---

303. Unique Path.java (https://github.com/shawnfan/LintCode/blob/master/Java/Unique%20Path.java) Level: Medium

---

304. Unique Paths II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Unique%20Paths%20II.java)Follow up for "Unique Paths":

Now consider if some obstacles are added to the grids. How many unique paths would there be?

An obstacle and empty space is marked as 1 and 0 respectively in the grid.

Note
m and n will be at most 100.

Example
For example,
There is one obstacle in the middle of a 3x3 grid as illustrated below.

[
[0,0,0],
[0,1,0],

[0,0,0]
]
The total number of unique paths is 2.

Tags Expand
Array Dynamic Programming

Thinking process:

1. Still use an extra matrix to count possible paths.

2. When initializing, skip block if it's obstacle (break the for loop, basically skip this row/col)

3. When evaluating paths, skip block if it's obstacle (save current spot's path as 0, means no path through this point).

4. Note: At evaluating double-for loop, we cannot use break, because we still need to evaluate using upper/left block. Hence we set the obstacle = 0.

   */

public class Solution {
/**

```java
 * @param obstacleGrid: A list of lists of integers
 * @return: An integer
 */
public int uniquePathsWithObstacles(int[][] obstacleGrid) {
    if (obstacleGrid == null || obstacleGrid.length == 0 || obstacleGrid[0].length == 0) {
        return 0;
    }
    int row = obstacleGrid.length;
    int col = obstacleGrid[0].length;
    int[][] matrix = new int[row][col];
    for (int i = 0; i < row; i++) {
        if (obstacleGrid[i][0] == 1) {
            break;
        } else {
            matrix[i][0] = 1;
        }
    }
    for (int j = 0; j < col; j++) {
        if (obstacleGrid[0][j] == 1) {
            break;
        } else {
            matrix[0][j] = 1;
        }
    }
    for (int i = 1; i < row; i++) {
        for (int j = 1; j < col; j++) {
            if (obstacleGrid[i][j] == 1) {
                matrix[i][j] = 0;
            } else {
                matrix[i][j] = matrix[i - 1][j] + matrix[i][j - 1];
            }
        }
    }
    return matrix[row - 1][col - 1];
}
```

}

---

305. Unique Word Abbreviation.java (https://github.com/shawnfan/LintCode/blob/master/Java/Unique%20Word%20Abbreviation.java)An abbreviation of a word follows the form <first letter><number><last letter>. Below are some examples of word abbreviations:

a) it --> it (no abbreviation)

```
1
```

b) d|o|g --> d1g

```
        1    1  1
  1---5----0----5--8
```

c) i|nternationalizatio|n --> i18n

```
        1
  1---5----0
```

d) l|ocalizatio|n --> l10n

Assume you have a dictionary and given a word, find whether its abbreviation is unique in the dictionary. A word's abbreviation is unique if no other word from the dictionary has the same abbreviation.

Example:
Given dictionary = ["deer", "door", "cake", "card"]

isUnique("dear") -> false
isUnique("cart") -> true
isUnique("cane") -> false
isUnique("make") -> true

Tags: Hash Table, Design
Similar Problems: (E) Two Sum III - Data structure design

/
/
Thought:
Originally, used a hashset to store all existing pattern. If checked word exist in dict hashset, then return false.
However, there is a case that: the word existed in the dict only for once, which is by accident the same as the checked work, then return true.
Therefore, we need to keep track of what word has been catagrize into pattern. SO, use a HashMap<String, ArrayList> instead.

Note: Dealing with char, integer, string. Be careful if char are turnning int integers.
*/
public class ValidWordAbbr {
HashMap<String, ArrayList<String>> map;
public ValidWordAbbr(String[] dict) {
if (dict == null || dict.length == 0) {
return;
}
map = new HashMap<String, ArrayList<String>>();
```

```java
for (String s : dict) {
String str = "";
if (s.length() <= 2) {
str = s;
} else {
str += s.charAt(0) + (s.length() - 2 + "") + s.charAt(s.length() - 1);
}
if (!map.containsKey(str)) {
ArrayList<String> list = new ArrayList<String>();
list.add(s);
map.put(str, list);
} else {
if (!map.get(str).contains(s)) {
map.get(str).add(s);
}
```

```java
        }
    }
 }

 public boolean isUnique(String word) {
     if (map == null || map.size() == 0) {
         return true;
     }
     String str = "";
     if (word.length() <= 2) {
         str =  word;
     } else {
         str += word.charAt(0) + (word.length() - 2 + "") + word.charAt(word.length() - 1);
     }
     if (map.containsKey(str) && map.get(str).size() == 1 && map.get(str).get(0).equals(word)) {
         return true;
     }
     return !map.containsKey(str);
 }
```

```java
}
```

```java
// Your ValidWordAbbr object will be instantiated and called as such:
// ValidWordAbbr vwa = new ValidWordAbbr(dictionary);
// vwa.isUnique("Word");
// vwa.isUnique("anotherWord");
```

306. Update Bits.java (https://github.com/shawnfan/LintCode/blob/master/Java/Update%20Bits.java)Given two 32-bit numbers, N and M, and two bit positions, i and j. Write a method to set all bits between i and j in N equal to M (e g , M becomes a substring of N located at i and starting at j)

Example
Given N = (10000000000)2, M = (10101)2, i = 2, j = 6

return N = (10001010100)2

Challenge
Minimum number of operations ?

Tags Expand
Cracking The Coding Interview Bit Manipulation Binary Representation

Thinking process:
Create a mask: xxxx000000xxxx.
Trick part: when it encounters negative number or dealing with index at edge index = 31, it starts having issue. Interesting fix: use long for masks.
*/

```java
class Solution {
/* @param n, m: Two integer
@param i, j: Two bit positions return: An integer
*/
public int updateBits(int n, int m, int i, int j) {
//Create mask: xxx00000xxx
long rightMask = ~0 >> i;
rightMask = ~(rightMask << i);// 00000xxx
long leftMask = ~0 >> (j + 1);
leftMask = leftMask << (j + 1);//xxxxx00000000
long mask = leftMask | rightMask;//xxx00000xxx
n = (int) (n & mask);
n = (int) (n | (m << i));
return n;
}
}
```

---

307. Valid Anagram.java (https://github.com/shawnfan/LintCode/blob/master/Java/Valid%20Anagram.java)Given two strings s and t, write a function to determine if t is an anagram of s.

For example,

s = "anagram", t = "nagaram", return true.

s = "rat", t = "car", return false.

Note:

You may assume the string contains only lowercase alphabets.

Follow up:

What if the inputs contain unicode characters? How would you adapt your solution to such case?

Tags: Hash Table, Sort

Similar Problems: (M) Group Anagrams, (E) Palindrome Permutation

```
/
/
Thoughts:
Anagram: reorder of letters.
Use HashMap<charactor, count> to store the frequency of chars of 1st string, and check aginst 2nd string.
s character: +1;
t character: -1;
check count of each index in the map; they should all be 0
*/

public class Solution {
public boolean isAnagram(String s, String t) {
if (s == null || t == null) {
return s == null && t == null;
} else if (s.length() != t.length()) {
return false;
}
```

```java
        HashMap<Character, Integer> map = new HashMap<Character, Integer>();
        for (int i = 0; i < s.length(); i++) {
            if (!map.containsKey(s.charAt(i))) {
                map.put(s.charAt(i), 1);
            } else {
                map.put(s.charAt(i), map.get(s.charAt(i)) + 1);
            }
            if (!map.containsKey(t.charAt(i))) {
                map.put(t.charAt(i), -1);
            } else {
                map.put(t.charAt(i), map.get(t.charAt(i)) - 1);
            }
        }//END for

        for (int i = 0; i < s.length(); i++) {
            if (map.get(s.charAt(i)) != 0) {
                return false;
            }
        }
        return true;
    }
```

}

---

308. Valid Palindrome.java (https://github.com/shawnfan/LintCode/blob/master/Java/Valid%20Palindrome.java) Level: Easy

过滤alphanumeric，其他字母掠过

---

309. Valid Parentheses.java (https://github.com/shawnfan/LintCode/blob/master/Java/Valid%20Parentheses.java) Level: Easy

剥皮过程。解铃还须系铃人
左边的外皮'{['在stack底部
右边的外皮应该和stack顶上的左外皮一一对应

---

310. Valid Sudoku.java (https://github.com/shawnfan/LintCode/blob/master/Java/Valid%20Sudoku.java) Level: Easy

用HashSet存visited value.

方法1: 在nest for loop里面validate row,col,and block.
validate block要利用i 和 j 增长的规律。
说白了，i && j是按照0~n增长的index, 具体怎么用是可以flexible的。这个方法在同一个nest for loop解决所有运算。

方法2: 单独做block validation: validate block的时候虽然看到了4层for.其实也就是n^2.

---

311. Validate Binary Search Tree.java (https://github.com/shawnfan/LintCode/blob/master/Java/Validate%20Binary%20Search%20Tree.java) Level: Medium

查看每个parent-child关系。同时把root level上面传下来max,min界限定住。

---

312. Wiggle Sort.java (https://github.com/shawnfan/LintCode/blob/master/Java/Wiggle%20Sort.java)这样的fall-through每次在乎两个element，可以一口气搞定，无关乎再之前的elements。
特别的一点：flag来巧妙的掌控山峰和低谷的变化。又是神奇的一幕啊！

这样子的奇观，见过就要知道了，没见过的时候有点摸不着头脑。

---

313. Wood Cut.java (https://github.com/shawnfan/LintCode/blob/master/Java/Wood%20Cut.java)Given n pieces of wood with length L[i] (integer array). Cut them into small pieces to guarantee you could have equal or more than k pieces with the same length. What is the longest length you can get from the n pieces of wood? Given L & k, return the maximum length of the small pieces.

Note
You couldn't cut wood into float length.

Example
For L=[232, 124, 456], k=7, return 114.

Challenge
O(n log Len), where Len is the longest length of the wood.

Tags Expand
Binary Search

Thinking process:
Take the largest item.
Priorities:

 1. Have to get calculatedK >= givenK

 2. Meanwhile, want to maximize the smal piece.

One thing not clear: do we have to use the given small piece? If we have to, we need to concern about the shortest wood piece. See commentted-out part
In this problem, however, we can abandon the small pieces, as long as the max_small_pieces can allow calculatedK >= givenK.

Use binary search on the largest item:

 1. if calculatedK < givenK: end = mid;

 2. If calculated >= givenK, move start = mid as much as possible, which gives maximized small piece.

*/

```java
public class Solution {
/* @param L: Given n pieces of wood with length L[i]
@param k: An integer return: The maximum length of the small pieces.
*/
public int woodCut(int[] L, int k) {
if (L == null || L.length == 0 || k < 0) {
return 0;
}
if (L.length == 1) {
return L[0] / (L[0] / k);
}
Arrays.sort(L);
int start = 0;
int end = L[L.length - 1];
int mid = 0;
int max = 0;
// int min = L[0];
while (start + 1 < end) {
mid = start + (end - start) / 2;
//if (mid > min) {
// end = mid;
// } else {
int count = 0;
for (int i : L) {
count += i / mid;
}
if (count < k) {
end = mid;
} else {
start = mid;
max = mid;
}
```

```
//}
}//end while
return max;
}
}
```

---

314. Word Break II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Word%20Break%20II.java) Level: Hard

两个DP一起用.解决了timeout的问题

1. isWord[i][j], subString(i,j)是否存在dict中？

2. 用isWord加快 isValid[i]: [i ~ end]是否可以从dict中找到合理的解？
   从末尾开始查看i：因为我们需要测试isWord[i][j]时候，j>i, 而我们观察的是[i,j]这区间；
   j>i的部分同样需要考虑，我们还需要知道isValid[0~j+1]。 所以isValid[x]这次是表示[x, end]是否valid的DP。
   i 从 末尾到0, 可能是因为考虑到isWord[i][j]都是在[0~n]之内，所以倒过来数，坐标比较容易搞清楚。
   (回头看Word Break I， 也有坐标反转的做法)

3. dfs 利用 isValid 和isWord做普通的DFS。

Note:
在Word Break里面用了set.contains(...), 在isValid里面，i 从0开始。 但是，contains()本身是O(n).
在这道题里面应该是因为word dictionary太大，加上nest for, 变成O(n^3)所以timeout.

istead,用一个isWord[i][j]，就O(1)判断了i~j是不是存在dictionary里面。

---

315. Word Break.java (https://github.com/shawnfan/LintCode/blob/master/Java/Word%20Break.java) Level: Medium

DP

方法1:（attempt3 code）
state,rst[i]: 从[0~i] inclusive的string是否可以在dict中break开来找到？
function: rst[i] = true if (rst[i - j] && set.contains(s.substring(i - j, i))); j in[0~i]

1. rst[i - j] 记录的是[0, i-j]这一段是否可以break后在dict找到。
2. 若true，再加上剩下所有[i-j, i]都能在dict找到，那么rst[i] = rst[0, i - j] && rst[i-j, i] == true

优化：找dict里面最长string, 限制j的增大。

(attempt4 code)
与Word BreakII用同样的DP。
valid[i]: 记录从i到valid array末尾是否valid.

---

316. Word Ladder II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Word%20Ladder%20II.java) Level: Hard

---

317. Word Ladder.java (https://github.com/shawnfan/LintCode/blob/master/Java/Word%20Ladder.java) Level: Medium

BFS Brutle: 在start string基础上，string的每个字母都遍历所有26个字母，换换。

方法2:
用Trie。 理应更快. However implementation可能有点重复计算的地方，LeetCode timeout. 需要再做做。

---

318. Word Pattern.java (https://github.com/shawnfan/LintCode/blob/master/Java/Word%20Pattern.java) Level: Easy

每个char代表一个pattern。用HashMap<char, str>.
但不够，如果a也match dog, b也match dog, 纠错了。比如pattern = "abba", str = "dog dog dog dog".
因此第二个HashMap<str, char> 反过来。
确保pattern和str一一对应。

---

319. Word Search II.java (https://github.com/shawnfan/LintCode/blob/master/Java/Word%20Search%20II.java) Level: Hard

Big improvement: use boolean visited on TrieNode!
不要用rst.contains(...), 因为这个是O(n) 在leetcode还是会timeout（lintcode竟然可以pass）！
在Trie search() method 里面，凡是visit过的，mark一下。

Regular:
for loop on words: inside, do board DFS based on each word.
Time cpmplexity: word[].length boardWidth boardHeight * (4^wordMaxLength)

Build Trie with target words: insert, search, startWith.
依然要对board matrix做DFS。

no for loop on words. 直接对board DFS:

每一层,都会有个up-to-this-point的string. 在Trie里面check它是不是存在。以此判断。

若不存在，就不必继续DFS下去了。

Trie solution time complexity, much better:

build Trie: n wordMaxLength

search: boardWidth boardHeight * (4^wordMaxLength + wordMaxLength[Trie Search])

---

320. Word Search.java (https://github.com/shawnfan/LintCode/blob/master/Java/Word%20Search.java) Level: Medium

Backtracking:

比较 Brutle。找到开头的字母，然后投入一个recursive找字母的工程：每到一个字母，朝四个方向走。他们之中，有一个true就可以。

Note:每次到一个字母，mark一下'#'. 4个path recurse回来后，mark it back.

Backtracking方法2:

用一个boolean visited[][]

---

321. Zigzag Iterator.java (https://github.com/shawnfan/LintCode/blob/master/Java/Zigzag%20Iterator.java)每次next(), 相应的list的头拿下来就好。

然后就跑圈呗，每次刷一个list头。不难。只要把几个variable维护清楚就行。

如果觉得我的文章对您有用，请随意打赏。您的支持将鼓励我继续创作！

¥ 打赏支持

♡ 喜欢 | 1

分享到微博
分享到微信
更多分享

0条评论（ 按时间正序 · 按时间倒序 · 按喜欢排序 ）

添加新评论 (/sign_in)

被以下专题收入，发现更多相似内容：

程序员 (/collection/NEt52a)

如果你是程序员，或者有一颗喜欢写程序的心，喜欢分享技术干货、项目经验、程序员日常囧事等等，欢迎投稿《程序员》专题。 专题主编：小...
(/collection/NEt52a)
20648篇文章 (/collection/NEt52a) · 139026人关注

添加关注 (/sign_in)

李特抠得LeetCode (/collection/09147fcb1399)

小李的李，特别的特。 李特，特别抠，抠得天荒地老。
(/collection/09147fcb1399)
26篇文章 (/collection/09147fcb1399) · 8人关注

添加关注 (/sign_in)

烧火棍 (/collection/ee0fb33889ed)

一些技术类文章，读书分享等
(/collection/ee0fb33889ed)
3篇文章 (/collection/ee0fb33889ed) · 0人关注

添加关注 (/sign_in)