Table of Contents *generated with DocToc*

  - 62. Connecting Graph II.java Level: Medium Tags: [Union Find]
  - 63. Connecting Graph III.java Level: Medium Tags: [Union Find]
  - 64. Number of Islands.java Level: Medium Tags: [BFS, DFS, Matrix DFS, Union Find]
  - 65. Surrounded Regions.java Level: Medium Tags: [BFS, DFS, Matrix DFS, Union Find]
  - BFS
  - 66. [Implement Trie (Prefix Tree) .java] (https://github.com/awangdev/LintCode/blob/master/Java/Implement%20Trie%20 (Prefix% 20Tree) .java) Level: Medium Tags: [Design, Trie]
  - 67. [Add and Search Word-Data structure design.java] (https://github.com/awangdev/LintCode/blob/master/Java/Add%20and%20Search%20Word%20-%20Data%20structure% 20design.java) Level: Medium Tags: [Backtracking, Design, Trie]
  - 68. Word Search.java Level: Medium Tags: [Array, Backtracking, DFS]
  - 69. Decode String.java Level: Medium Tags: [DFS, Divide and Conquer, Stack]
  - 70. Maximum Binary Tree.java Level: Medium Tags: [Stack, Tree]
  - 71. Swap Nodes in Pairs.java Level: Medium Tags: [Linked List]
  - 72. Wood Cut.java Level: Medium Tags: [Binary Search]
  - 73. Find the Duplicate Number.java Level: Medium Tags: [Array , Binary Search, Two Pointers]
  - 74. Game of Life.java Level: Medium Tags: [Array]
  - 75. Number of Airplane in the sky.java Level : Medium Tags: [Array, Interval, PriorityQueue, Sort, Sweep Line]
  - 76. Meeting Rooms II.java Level: Medium Tags: [Greedy, Heap, PriorityQueue, Sort, Sweep Line]
  - 77. Unique Path.java Level: Medium Tags: [Array, Coordinate DP, DP]
  - 78. Maximal Square.java Level: Medium Tags: [Coordinate DP, DP]
  - 79. Coins in a Line.java Level: Medium Tags: [DP , Game Theory, Greedy]
  - 80. [Coins in a Line II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Coins%20in%20a%20Line%20II. java) Level: Medium Tags: [Array, DP, Game Theory, Memoization, MiniMax]
  - 81. Binary Tree Postorder Traversal.java Level: Medium Tags: [Stack, Tree, Two Stacks]
  - 82. Compare Version Numbers.java Level: Medium Tags: [String]
  - 83. Count Complete Tree Nodes.java Level: Medium Tags: [Binary Search, Tree]
- ## Iteratively
  - 84. Course Schedule.java### Level: Medium Tags: [BFS, Backtracking, DFS, Graph, Topological Sort]
  - 85. Course Schedule II.java Level: Medium Tags: [BFS , DFS, Graph, Topological Sort]-A
  - 87. Jump Game .java Level: Medium Tags: [Array, DP, Greedy]
  - 88. [Coin Change 2.java] (https://github.com /awangdev/LintCode/blob/master/Java/Coin%20Change%202.java)### Level: Medium Tags: [Backpack DP, DP]
  - 89. [Decode Ways.java] (https://github.com/awangdev/LintCode/blob/ master / Java / Decode% 20Ways.java) Level: Medium Tags: [DP, Partition DP, String]
  - 90. Minimum Path Sum.java Level: Medium Tags: [Array, Coordinate DP , DP]
  - 91. [Counting Bits.java] (https://github.com/awangdev/LintCode /blob/master/Java/Counting%20Bits.java)### Level: Medium Tags: [Bit Manipulation, Bitwise DP, DP]
  - 92. [Continuous Subarray Sum.java] (https : //github.com/awangdev/LintCode/blob/master/Java/Continuous%20Subarray%20Sum.java) Level: Medium Tags: [Coordinate DP, DP, Math, Subarray]
  - 93. [House Robber II.java] (https://github.com/ awangdev / LintCode / blob / master / Java / House% 20Robber% 20II.java) Level: Medium Tags: [DP, Sequence DP, Status DP]
  - 94. [House Robber III.java] (https : //github.com/awangdev/LintCode/blob/master/Java/House%20Robber%20III.java) Level: Medium Tags: [DFS, DP, Status DP, Tree]
- ## DP, DFS
  - 95. Permutation in String.java Level: Medium Tags: [Two Pointers]
  - ### Two Pointer
  - 96. Permutations II.java Level: Medium Tags: [Backtracking]
  - 97. Shuffle an Array.java * * Level: Medium Tags: [Permutation]
  - 98. [Group Anagrams.java] (https://github.com/awangdev/LintCode/blob /master/Java/Group%20Anagrams.java)### Level: Medium Tags: [Hash Table, String]
  - 99. [Backpack.java] (https://github.com/awangdev/LintCode/blob/master/ Java / Backpack.java) Level: Medium Tags: [Backpack DP, DP]
  - 100. Backpack II.java Level: Medium Tags: [Backpack DP, DP]
  - 101. Backpack V.java Level: Medium Tags: [Backpack DP, DP]
- ## Backpack DP
  - 102. Evaluate Reverse Polish Notation.java Level: Medium Tags: [Stack]
  - 103. [Insertion Sort List.java] (https://github.com/awangdev/LintCode/blob/master/Java/Insertion%20Sort%20List. java) Level: Medium Tags: [Linked List, Sort]
  - 104. Interleaving Positive and Negative Numbers.java Level : Medium Tags: [Two Pointers]
  - 105. Largest Number.java Level: Medium Tags: [Sort]
  - 106. Longest Common Substring.java Level: Medium Tags: [DP, Double Sequence DP, Sequence DP, String]
  - 107. Longest Increasing Continuous subsequence II.java Level: Medium Tags : [Array, Coordinate DP, DP, Memoization]
- -O (mn) space for dp and flag. -
  - 108. Maximum Subarray II.java Level: Medium Tags: [Array, DP, Greedy, PreSum, Sequence DP, Subarray]
  - 109. [Reverse Linked List II .java] (https://github.com/awangdev/LintCode/blob/master/Java/Reverse%20Linked%20List% 20II% 20.java) Level: Medium Tags: [Linked List]
  - 110. [Lowest Common Ancestor of a Binary Tree.java] (https://github.com/awangdev/LintCode/blob/master/Java/Lowest%20Common%20Ancestor%20of%20a%20Binary%20Tree .java) Level: Medium Tags: [DFS, Tree]
  - 111. Lowest Common Ancestor of a Binary Search Tree.java Level: Medium Tags: [BST, DFS, Tree]
  - 112. Remove Duplicates from Sorted Array II. java Level: Medium Tags: [Array, Two Pointers]
  - 113. [Remove Duplicates from Sorted List II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Remove%20Duplicates% 20from% 20Sorted% 20List% 20II.java) Level: Medium Tags: [Linked List]

- ## Linked List
  - 114. QuickSort.java Level: Medium Tags: [Quick Sort, Sort]
  - 115. [MergeSort.java] (https: // github. com / awangdev / LintCode / blob / master / Java / MergeSort.java) Level: Medium Tags: [Merge Sort, Sort]
  - 116. Binary Tree Level Order Traversal.java Level: Medium Tags : [BFS, DFS, Tree]
  - 117. Binary Tree Level Order Traversal II.java Level: Medium Tags: [BFS, Tree]
- As the title, but the output must be
  - 118. Binary Tree Longest Consecutive Sequence II.java Level : Medium Tags: [DFS, Divide and Conquer, Double Recursive, Tree]
  - 119. [Combinations.java] (https://github.com/awangdev/ LintCode / blob / master / Java / Combinations.java) Level: Medium Tags: [Backtracking, Combination, DFS]
- ## DFS, Backtracking
  - 120. Combination Sum IV.java Level: Medium Tags: [Array, Backpack DP , DP]
  - 121. [Binary Tree Right Side View.java] (https: // github .com / awangdev / LintCode / blob / master / Java / Binary% 20Tree% 20Right% 20Side% 20View.java) Level: Medium Tags: [BFS, DFS, Tree]
  - 122. Binary Tree Maximum Path Sum II.java Level : Medium Tags: [DFS, Tree]
  - 123. Rotate List.java * * Level: Medium Tags: [Linked List, Two Pointers]
  - 124 [Binary Tree Longest the Consecutive Sequence.java] (https://github.com/awangdev/LintCode/blob/master/Java/. binary% 20Tree% 20Longest% 20Consecutive% 20Sequence.java) Level: Medium Tags: [DFS, Divide and Conquer, Tree]
  - 125. Number of Connected Components in an Undirected Graph.java Level: Medium Tags: [BFS, DFS, Graph, Union Find]
  - 126. Next Closest Time.java Level: Medium Tags: [Basic Implementation, Enumeration , String]
- ## String
  - 127. Partition Array.java Level: Medium Tags: [Array, Quick Sort, Sort, Two Pointers]
  - 128. Word Ladder.java Level : Medium Tags: [BFS]
  - 129. Unique Word Abbreviation.java Level: Medium Tags: [Design, Hash Table]
  - 130. [Unique Binary Search Tree II.java] (https://github.com/awangdev/LintCode/blob/master/Java/ Unique% 20Binary% 20Search% 20Tree% 20II.java) Level: Medium Tags: [BST, DP, Divide and Conquer, Tree]
- ## DP? Memoization?
  - 131. Ugly Number.java Level: Medium Tags : [Math]
  - 132. [Top K Frequent Words.java] (https://github.com/awangdev/LintCode/blob/master/Java/Top%20K%20Frequent% 20Words.java) Level: Medium Tags: [Hash Table, Heap, MaxHeap, MinHeap, PriorityQueue, Trie]
- ## PriorityQueue-Min Heap
  - 135. Segment Tree Query.java Level: Medium Tags: [Binary Tree, DFS, Divide and Conquer, Lint, Segment Tree]
  - 136. Segment Tree Modify.java Level: Medium Tags: [Binary Tree, DFS, Divide and Conquer, Lint, Segment Tree]
  - 137. Segment Tree Query II.java Level: Medium Tags: [Binary Tree, DFS, Divide and Conquer, Lint, Segment Tree]
  - 138. [ColorGrid.java] (https: // github .com / awangdev / LintCode / blob / master / Java / ColorGrid.java) Level: Medium Tags: [Design, Hash Table]
  - 139. [Container With Most Water.java] (https://github.com/awangdev/LintCode/blob/master/Java /Container%20With%20Most%20Water.java)### Level: Medium Tags: [Array, Two Pointers]
  - 140. Copy List with Random Pointer.java Level: Medium Tags : [Hash Table, Linked List]
  - 141. Encode and Decode Strings.java Level: Medium Tags: [String]
  - 142. [Fast Power.java] (https://github.com/awangdev /LintCode/blob/master/Java/Fast%20Power.java)### Level: Medium Tags: [DFS, Divide and Conquer]
  - ### Divide and Conquer
  - 143. [Find the Connected Component in the Undirected Graph.java] (https://github.com/awangdev/LintCode/blob/master/Java/Find%20the%20Connected%20Component%20in%20the% 20Undirected% 20Graph.java) Level: Medium Tags: [BFS, DFS]
  - 145. [Interval Minimum Number.java] (https: / /github.com/awangdev/LintCode/blob/master/Java/Interval%20Minimum%20Number.java)### Level: Medium Tags: [Binary Search, Divide and Conquer, Lint, Segment Tree]
  - 146. Interval Sum.java * * Level: Medium Tags: [Binary Search, Lint, Segment Tree]
  - 147. [Kth Smallest Element in a BST.java] (https://github.com/awangdev/LintCode/blob/master /Java/Kth%20Smallest%20Element%20in%20a%20BST.java)### Level: Medium Tags: [BST, DFS, Stack, Tree]
  - 148. Majority Element II.java Level: Medium Tags: [Array]
- ## Sort + count
  - 149. Partition List.java Level: Medium Tags: [Linked List, Two Pointers]
  - 150. Peeking Iterator.java Level: Medium Tags: [Design]
- ## Use concept pre cache
  - 151. Rehashing.java Level: Medium Tags: [Hash Table]
  - 153. Restore IP Addresses.java Level: Medium Tags: [Backtracking , DFS, String]
  - 154. Reverse Words in a String.java Level: Medium Tags : [String]
  - 155. Reverse Words in a String II.java * * Level: Medium Tags: [String]
  - 156. [Search a 2D Matrix.java] (https : //github.com/awangdev/LintCode/blob/master/Java/Search%20a%202D%20Matrix.java) Level: Medium Tags: [Array, Binary Search]
  - 158 [Search for A Range.java] (https://github.com/awangdev/. LintCode / blob / master / Java / Search% 20for% 20a% 20Range.java) Level: Medium Tags: [Array, Binary Search]
  - # Binary Search
  - 159. [Search Range in Binary Search Tree .java] (https://github.com/awangdev /LintCode/blob/master/Java/Search%20Range%20in%20Binary%20Search%20Tree%20.java)### Level: Medium Tags: [BST, Binary Tree]
  - 160. Sort List.java Level: Medium Tags: [Divide and Conquer, Linked List, Merge Sort, Sort]
  - 162. Topological Sorting.java Level: Medium Tags: [BFS, DFS, Topological Sort]
  - 163. [Spiral Matrix.java] (https://github.com/ awangdev / LintCode / blob / master / Java / Spiral% 20Matrix.java) Level: Medium Tags: [Array, Enumeration]
  - # DX, DY
  - 164. [Construct Binary Tree from Inorder and Postorder Traversal.java] (https://github.com/awangdev/LintCode/blob/master/Java/Construct%20Binary%20Tree%20from%20Inorder%20and%20Postorder%20Traversal .java) Level: Medium Tags: [Array, DFS, Divide and Conquer, Tree]

- 165. Generate Parentheses.java Level : Medium Tags: [Backtracking, DFS, Sequence DFS, String]
- 167. Flip Game II.java Level: Medium Tags: [Backtracking, DFS , DP]
- 168. Palindrome Partitioning.java Level: Medium Tags: [Backtracking, DFS]
- ### Complexity
- 169. [Submatrix Sum.java] (https: // github. com / awangdev / LintCode / blob / master / Java / Submatrix% 20Sum.java) Level: Medium Tags: [Array, Hash Table, PreSum]
- 170. [Longest Palindromic Substring.java] (https://github.com/awangdev/LintCode/blob/master/ Java / Longest% 20Palindromic% 20Substring.java) Level: Medium Tags: [DP, String]
- 171. Longest Palindromic Subsequence.java Level: Medium Tags: [DFS, DP, Interval DP, Memoization]
- 172. [Gas Station.java] (https://github.com/awangdev/LintCode/ blob / master / Java / Gas% 20Station.java) Level: Medium Tags: [Greedy]
- 174. Merge Intervals.java Level: Medium Tags: [Array, PriorityQueue, Sort , Sweep Line]
- 175. H-Index.java Level: Medium Tags: [Bucket Sort, Hash Table, Sort]
- 176. H-Index II. java Level: Medium Tags: [Binary Search]
- 177. Sort Colors.java Level: Medium Tags: [Array, Partition, Quick Sort, Sort, Two Pointers]
- 178. Sort Colors II.java Level: Medium Tags: [Partition, Quick Sort , Sort, Two Pointers]
- 179. Sort Letters by Case.java Level: Medium Tags: [Partition, Sort, String, Two Pointers]
- 180. [Subarray Sum Closest.java] (https: //github.com/awangdev/LintCode/blob/master/Java/Subarray%20Sum%20Closest.java)### Level: Medium Tags: [PreSum, PriorityQueue, Sort, Subarray]
- 181. [Task Scheduler.java] (https://github.com/awangdev/LintCode/blob/master/Java/Task% 20Scheduler.java) Level: Medium Tags: [Array, Enumeration, Greedy, PriorityQueue, Queue]
- 182. Exam Room.java Level: Medium Tags: [PriorityQueue, Sort]
- 183. Anagrams.java Level: Medium Tags: [Array, Hash Table]
- 184. Path Sum IV.java Level: Medium Tags: [DFS, Hash Table , Tree]
- 186. [Palindromic Substrings.java] (https://github.com/awangdev/LintCode/ blob / master / Java / Palindromic% 20Substrings.java) Level: Medium Tags: [DP, String]
- 187. Multiply Strings.java Level: Medium Tags: [Math, String]
- 188. Subsets.java Level: Medium Tags: [Array, BFS , Backtracking, Bit Manipulation, DFS]
- 189. [Subsets II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Subsets%20II. java) Level: Medium Tags: [Array, BFS, Backtracking, DFS]
- # BFS
- 190. Combination Sum.java Level: Medium Tags: [Array, Backtracking , Combination, DFS]
- 191. Combination Sum II.java Level: Medium Tags: [Array, Backtracking, Combination, DFS]
- 192. [Combination Sum III.java] (https://github.com/awangdev/LintCode/blob/master/Java/Combination% 20Sum% 20III.java) Level: Medium Tags: [Array, Backtracking, Combination, DFS]
- 194. [Total Hamming Distance.java] (https://github.com/awangdev/LintCode/blob/master/Java/Total%20Hamming%20Distance. java) Level: Medium Tags: [Bit Manipulation]
- 195. Smallest Subtree with all the Deepest Nodes.java Level: Medium Tags: [DFS, Divide and Conquer, Tree]
- 196. Subarray Sum Equals K. java Level: Medium Tags: [Array, Hash Table, PreSum, Subarray]
- 197. Simplify Path.java Level: Medium Tags: [Stack, String]
- 198. [Convert Binary Search Tree to Sorted Doubly Linked List (extra space) .java] (https://github.com/awangdev/LintCode/blob/master/Java/Convert%20Binary%20Search%20Tree%20to% 20Sorted% 20Doubly% 20Linked% 20List% 20 (extra% 20space) .java) Level: Medium Tags: [Linked List, Stack, Tree]
- 199. Binary Tree Zigzag Level Order Traversal.java Level: Medium Tags: [BFS, Stack, Tree]
- 200. [Word Break.java] (https://github.com/ awangdev / LintCode / blob / master / Java / Word% 20Break.java) Level: Medium Tags: [DP, Hash Table, Sequence DP]
- 201. Longest Increasing Subsequence.java Level: Medium Tags: [Binary Search, Coordinate DP, DP, Memoization]

- ## O (nLogN)-Maintain
  - 202. [Best Time to Buy and Sell Stock with Transaction Fee.java] (https://github.com/awangdev/LintCode/blob/master/Java/Best%20Time%20to%20Buy%20and%20Sell%20Stock % 20with% 20Transaction% 20Fee.java) Level: Medium Tags: [Array, DP, Greedy, Sequence DP, Status DP]
  - 205. Sparse Matrix Multiplication.java Level: Medium Tags: [Hash Table]
  - 206. Brick Wall.java Level: Medium Tags: [Hash Table]
  - 208. Friends Of Appropriate Ages.java Level: Medium Tags : [Array, Math]
  - 209. Target Sum.java Level: Medium Tags: [DFS, DP]
  - 211. Contiguous Array.java Level: Medium Tags: [Hash Table]
  - 212. Line Reflection.java Level: Medium Tags : [Hash Table, Math]
  - 213. [Insert Delete GetRandom O (1) .java] (https://github.com/awangdev/LintCode/blob/master/Java/Insert% 20Delete% 20GetRandom% 20O (1) .java) Level: Medium Tags: [Array, Design, Hash Table]
  - 214. Number of Longest Increasing Subsequence.java Level: Medium Tags : [Coordinate DP, DP]
  - 215. [Minimum Swaps To Make Sequences Increasing.java] (https: //github.com/awangdev/LintCode/blob/master/Java/Minimum%20Swaps%20To%20Make%20Sequences%20Increasing.java)### Level: Medium Tags: [Coordinate DP, DP, Status DP]
  - 216. Binary Tree Vertical Order Traversal.java Level: Medium Tags : [BFS, DFS, Hash Table, Tree]
  - 217. [Populating Next Right Pointers in Each Node II.java] (https://github.com/awangdev/LintCode/blob/master/Java /Populating%20Next%20Right%20Pointers%20in%20Each%20Node%20II.java)### Level: Medium Tags: [DFS, Tree]
  - ### Trick
  - 218. [Search in Rotated Sorted Array.java] (https : //github.com/awangdev/LintCode/blob/master/Java/Search%20in%20Rotated%20Sorted%20Array.java) Level: Medium Tags: [Array, Binary Search]
  - 219. [Find the Weak Connected Component in the Directed Graph.java] (https://github.com/awangdev/LintCode/blob/master/Java/Find%20the%20Weak%20Connected%20Component%20in%20the% 20Directed% 20Graph.java) Level: Medium Tags: [Union Find]
  - 220. Accounts Merge.java Level: Medium Tags: [DFS, Hash Table, Hash Table , Union Find]
  - 221. Count of Smaller Number.java Level: Medium Tags: [Binary Search, Lint, Segment Tree]

Medium (247)

0. [Evaluate Division.java] (https://github.com/awangdev/LintCode/blob/master/Java/Evaluate%20Division.java) Level: Medium Tags: [BFS, DFS, Graph, Union Find]

DFS

-build map of `x # y-> val` to store values [i] and 1 / values [i] -build map of `x-> list children` -dfs to traverse the graph

BFS

-BFS should also work: build graph and valueMap -for each starting item, add all next candidate to queue -mark visited, loop until end item is found

---

1. [Fraction to Recurring Decimal.java] (https://github.com/awangdev/LintCode/blob/master/Java/Fraction%20to%20Recurring%20Decimal.java Level: Medium Tags: [Hash Table, Math]

TODO: no need of hashMap, just use set to store the existing

It is not difficult to think of dealing with division: consider positive and negative, consider before and after the decimal point. Mainly after the decimal point, we must focus on consideration. It's easy to overlook the benefits of integer.

---

2. [Gray Code.java] (https://github.com/awangdev/LintCode/blob/master/Java/Gray%20Code.java) Level: Medium Tags: [Backtracking]

TODO:

1. backtracking, using set to perform contains ()
2. BFS: use queue to keep the mutations

The title egg hurts and currently only accepts one result.

BackTracking + DFS:
Recursive helper flips each time oneself / left / right. After Flip, it should be restored as it is. Iterate through all.

Used (not carefully verified):
The basic idea is to start from one point and go in one direction, flip a bit every time, and go back when you hit a wall.

---

3. [Majority Number II.java](https://github.com/awangdev/LintCode/blob/master/Java/Majority%20Number%20II.java) Level: Medium Tags: [Enumeration, Greedy]

Array

-Divided in three: abc consideration -If a: countA ++; or b: countB ++ -Or c: countA--, countB-- -Note: In the order of the if statement, valA && countA has priority over valB && countB -The last two a and b with count> 0 are naturally greater than 1/3. One of them is greater than 1/3. -Compare whichever countA and countB are large, then return which one.

---

4. [Majority Number III.java](https://github.com/awangdev/LintCode/blob/master/Java/Majority%20Number%20III.java) Level: Medium Tags: [Hash Table, Linked List]

TODO:

1. hash table solution not passing
2. Find O (n) solution

Hash Table

-Same as other Majority Numbers. -If the number of occurrences is more than 1 / k, divide into k count occurrences. Use HashMap. Existing +1; if it does not exist in the map, it is divided into cases:
-If map.size () == k, it means that the dates are full, and all existing ones should be -1 in the map -If map.size () <k, indicating that the new candidate is to be added, then map.put (xxx, 1); -Finally, find out the number of the highest leftance in the HashMap. -But such worst case is O (nk)

---

5. [Minimum Height Trees.java](https://github.com/awangdev/LintCode/blob/master/Java/Minimum%20Height%20Trees.java) Level: Medium Tags: [BFS, Graph]

Graph + BFS

-Build graph `map <node, list of node>` -BFS to find the shortest path: when the neibhbor has the curr node as the only one neighbor, it is leaf. -record shortest path in Map <Integer, List > as result -TODO: code it up.

Previous Solution

-removing leaf && edge

---

6. [Missing Ranges.java](https://github.com/awangdev/LintCode/blob/master/Java/Missing%20Ranges.java) Level: Medium

Tags: [Array]

Basic Implementation

-O (n) -Two pointers, calculating the part between prev and curr each time. -Then prev = curr, move forward one space -TODO: check the edge case and make sure max / min of int are checked

---

7. [Next Permutation.java](https://github.com/awangdev/LintCode/blob/master/Java/Next%20Permutation.java) Level: Medium Tags: [Array]

Need to consider: why reverse is need? Why we are looking for k?

Permutation's law:

1. Change from small numbers because all recursive traversal starts from small numbers.
2. Because of the rule of 1, find a large number of breakpoints from the end: Make sure that the permutation after swap is still the smallest when the prefix is fixed.

steps:
Find the last rising point, k
2. From back to front, find the first point larger than k, bigIndex
3. swap k && bigIndex
4. The last reversal (k + 1, end)

---

8. [Palindrome Permutation II.java](https://github.com/awangdev/LintCode/blob/master/Java/Palindrome%20Permutation%20II.java) Level: Medium Tags: [Backtracking, Permutation]

TODO: need to review permutation

Comprehensive question of permutation:

1. Can validate input do Palindromic Permutation. This is (Palindrome Permutation I)
2. By the way, save the first half of the permutation string and the single character (if any) in the middle.
3. DFS does unique permutation: given input has duplicate characters.

---

9. [Permutation Sequence.java](https://github.com/awangdev/LintCode/blob/master/Java/Permutation%20Sequence.java) Level: Medium Tags: [Backtracking, Math]

TODO: what about regular DFS / backtracking to compute the kth? Dfs (rst, list, candiate list, k)

k is a number of permutation. And permutation is regular.

That is, you can determine the character of each digit according to the size of k (starting from the largest digit, because the default factorio changes from the largest digit).

So first find n !, then k / n! Can calculate the current digit character. Then reduce factorio and k, respectively.

In addition, use a boolean [] visited to ensure that each number appears only once.

This method is much more efficient than calculating each permutation.

---

10. [Product of Array Exclude Itself.java]
(https://github.com/awangdev/LintCode/blob/master/Java/Product%20of%20Array%20Exclude%20Itself

Level: Medium Tags : [Array]

---

11. [Remove Duplicates from Unsorted List.java] (https://github.com/awangdev/LintCode/blob/master/Java/Remove%20Duplicates%20from%20Unsorted Level: Medium Tags : [Linked List]

Basic method: O (n) sapce, time Iterate. Encountered duplicate (maybe multiple), while until node.next is not duplicate. Next, since it is not duplicate, add to set

If you don't use extra memory, do it in place: Then sort linked list. Use merge sort.

Review merge sort:

1. find middle.
2. recursively: right = sort (mid.next); left = sort (head).
3. within sort (), at the end call merge (left, right)

---

12. [Rotate Image.java] (https://github.com/awangdev/LintCode/blob/master/Java/Rotate%20Image.java) Level: Medium Tags: [Array, Enumeration]

Find formulas

-Find a regular formula for the rotation angle: r = c; c = (w-r) -Use temp variable, swap in place.

---

13. [Search in Rotated Sorted Array II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Search%20in%20Rotated%20Sorted%20Array Level : Medium Tags: [Array, Binary Search]

After Allow duplicates: Because the final binary search result is also O (n) So remember this question: Since it is O (n), then a simple for loop is just fine.

Of course, talk to the interviewer about the reason. Don't come up with only for. . .

---

14. [Single Number II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Single%20Number%20II.java) Level: Medium Tags: [Bit Manipulation]

In a string of numbers, all numbers are repeated three times, except for one number. Find this number, linear time, without extrace space (constant space)

TODO: bits

---

15. [Single Number III.java] (https://github.com/awangdev/LintCode/blob/master/Java/Single%20Number%20III.java) Level: Medium Tags: [Bit Manipulation]

TODO: wut?

---

16. [Space Replacement.java] (https://github.com/awangdev/LintCode/blob/master/Java/Space%20Replacement.java) Level: Medium Tags: [String]

---

17. [Stone Game.java] (https://github.com/awangdev/LintCode/blob/master/Java/Stone%20Game.java) Level: Medium Tags: [DP]

This DP is a bit weird. Need to consider. NOT DONE YET

---

18. [The Smallest Difference.java] (https://github.com/awangdev/LintCode/blob/master/Java/The%20Smallest%20Difference.java) Level: Medium Tags: [Array, Sort, Two Pointers]

---

19. [Total Occurrence of Target.java] (https://github.com/awangdev/LintCode/blob/master/Java/Total%20Occurrence%20of%20Target.java) Level: Medium Tags: []

The idea is simple. It's a bit long to write. Find total number of occurance. First find first occurance, then last occurance.

---

20. [Two Lists Sum.java] (https://github.com/awangdev/LintCode/blob/master/Java/Two%20Lists%20Sum.java) Level: Medium Tags: [Linked List]

Give two Linked lists, sum up and synthesize new lists

---

21. [Zigzag Iterator.java] (https://github.com/awangdev/LintCode/blob/master/Java/Zigzag%20Iterator.java) Level: Medium Tags: [BST]

This topic is relatively simple. When I do it, I first think about what to do with k. Then use a map to index and each listmark. Every time next (), the corresponding list head is removed. Then I ran in circles, swiping a list head each time. Not difficult. Just maintain a few variables clearly.

---

22. [Encode and Decode TinyURL.java] (https://github.com/awangdev/LintCode/blob/master/Java/Encode%20and%20Decode%20TinyURL.java) Level: Medium Tags: [Hash Table, Math]

In fact, I think of the entry point, which is a difficult and easy problem. The encode here is to find a way to save the URL, and then give a key. So how to make this key, simply use a map, and then count. For more complicated, you can make a random letter / number to form the key.

---

23. [Wiggle Sort.java] (https://github.com/awangdev/LintCode/blob/master/Java/Wiggle%20Sort.java) Level: Medium Tags: [Array, Sort]

method 1: Sort, nLog (n). Then change the straight uphill into cascading mountain peaks, you need to make a swap every few (every 2 digits in the title) to cause unevenness. Note: There is no relationship between every other peak, so just worry about [i], [i-1] two positions every time.

Method 2: O (n) Look at the rule of curious numbers and even digits, and then run it again according to the rule given by the question, focusing on only two positions at a time: just swap the inappropriate [i], [i-1] positions.

Method 3: Same as Law 2, but just a little more subtle: Think too much the first time. In fact, a fall-through can solve the problem, because: This fall-through cares about two elements at a time and can be done in one go, regardless of the previous elements. A special point: flags control clever changes in the peaks and valleys. Another magical scene! Such a spectacle, you must know it when you have seen it. When you haven't seen it, you are a little scratched.

---

24. [Queue Reconstruction by Height.java] (https://github.com/awangdev/LintCode/blob/master/Java/Queue%20Reconstruction%20by%20Height.ja Level: Medium Tags: [Greedy]

There is nothing else but to write the example once, find the rules, and then greedy. You need to write the rules you found, such as: starting from h large, and putting k small first. Pay attention to the correctness when writing the comparator. If you want to sort, and flexible insert: use arrayList. Make an object yourself. Finally, do the 'matchCount' where the thinking is clear, find the most correct spot, and then greedy insert.

O (n) space, O (nLog (n)) time, because of sorting.

There may be room for simplification, and the code is a bit too long. For example, try it without extra space?

---

25. [Two Sum II-Input array is sorted.java] (https://github.com/awangdev/LintCode/blob/master/Java/Two%20Sum%20II%20-%20Input%20array%20is% 20sorted.java) Level: Medium Tags: [Array, Binary Search, Two Pointers]

Ascending array, find 2SUM.

Two pointers

-Sorted array. Two pointer move start and end, check sum. -Note that sum uses long. -O (n) time

Binary Search, because it's already sorted

-Hold down a valueA, then find (target-valueB) in the remaining binary serach -for loop O (n), binary search O (logn) -overall time: O (nLogN), don't write

---

26. [2 Sum II.java] (https://github.com/awangdev/LintCode/blob/master/Java/2%20Sum%20II.java) Level: Medium Tags: [Array, Binary Search , Two Pointers]

Similar to 2sum II-input array is sorted. Both are sort array, then two pointers.

Question from LintCode. Note that you are looking for greater / bigger than target.

O (nLogn) is allowed due to given conditions:
sort two pointer

while two pointers move inside. Every time if num [left] + num [right]> target, then all num [left ++] plus num [right]> target.
That is, num [right] does not move, and calculates how many groups can be added to move left, that is: right-left so many. Add all to count.
Then right--. Change the right to compare it with the previous left part.

---

27. [Coin Change.java] (https://github.com/awangdev/LintCode/blob/master/Java/Coin%20Change.java) Level: Medium Tags: [Backpack DP, DP, Memoization]

Give a bunch of coins with different amounts, and total amount to spent. Find the minimum number of coins that can be combined into this amount. There is no limit to the number of each coin.

DP

-Find the right equation dp [x], how many coins are needed to accumulate amount x: #coin is value, and index is [0 ~ x]. -The relationship of the sub-question is: If a coin is used, then it should be #coins + 1 in the position of f [x-coinValue]

initialization

-To handle the boundary, at the beginning of 0index, use value0. -Integer.MAX_VALUE is used for comparison, initialize dp [x] -Note that once Integer.MAX_VALUE + 1 will become negative. This will happen when coin = 0.

Optimization

-Method 1: Use Integer.MAX_VALUE directly -Method 2: Use -1, a little more concise, compare dp [i] and dp [i-coin] + 1 each time, then save. It is not necessary to do multiple min comparisons.

Memoization

-dp [i] still says: min # of coints to make amount i -initialize dp [i] = Integer.MAX_VALUE -First select the last step (traversing coins), then dfs does the same -Record dp [amount] If value has already been given, do not repeat the calculation and return directly. -But there is no need to force memoization on this problem. The state and equation of ordinary DP are relatively easy to find.

---

28. [Maximum Product Subarray.java](https://github.com/awangdev/LintCode/blob/master/Java/Maximum%20Product%20Subarray.java) Level: Medium Tags: [Array, DP, Subarray]

Find a series of continuous subsequences from a set of numbers (both positive and negative), and reach the maximum product product.

DP

-Find the best value, think of DP. Time / Space O (n) -Two special places: -1. For positive and negative numbers, two DP arrays are required. -2. continuous prodct This condition determines when Math.min, Math.max, -It is compared with the current value of nums [x]. If the current value is more suitable, the previous continuous product will be discarded and restarted. -This is destined to require a global variable to hold the result.

Space optimization, rolling array

-The index i in maxProduct && minProduct can only be related to i-1, so redundant operatoins can be omitted. -Time: O (n), space: O (1)

---

29. [3 Sum Closest.java](https://github.com/awangdev/LintCode/blob/master/Java/3%20Sum%20Closest.java) Level: Medium Tags: [Array, Two Pointers]

A simple form of 3Sum, and does not find index, value, but just a sum.

double for loop. 2Sum can only use left / right 2 pointers. O (n ^ 2)

Note: use long when checking closest to avoid int being used

---

30. [Triangle Count.java](https://github.com/awangdev/LintCode/blob/master/Java/Triangle%20Count.java) Level: Medium Tags: [Array]

In fact, it is the deformation of 3sum, or the deformation of 2sum. It is mainly done with 2 pointers. Note that when selecting the index, set a [0 ~ i] one at a time, find the start, end, and i in the triangle. Note smartly: Among them, if a start / end / i match, then from this [start ~ end], anything larger than start is fine, so we count + = end-start. On the other hand, other indexes of <end may not meet nums [start] + nums [end]> nums [i]

---

31. [3Sum.java] (https://github.com/awangdev/LintCode/blob/master/Java/3Sum.java) Level: Medium Tags: [Array, Two Pointers]

sort array, for loop + two pointer. O (n ^ 2)

-Handling duplicate wthin triplets: -If the outermost moving point i is repeated, it will be used until the last one at the end.-If there are duplicates in the triplet, use the start point and move to the end.

Previous notes: note:

1. Find value triplets, multiple results. Note that you are not looking for index.
2. For ascending order, the first layer of for loop is started from the last element, ensuring the order.
3. Remove the same numbers used by duplicate: check, all skipped. You don't need to calculate the same result with the same number.

step:

1. For loop pick a number A.
2. 2Sum results in a bunch of 2 numbers
3. Cross match Step A in step 1.

Time O (n ^ 2), two nested loops

In addition, you can still use HashMap to do 2Sum. Slightly shorter. Still pay attention to handle duplicates.

---

32. [Unique Binary Search Tree.java] (https://github.com/awangdev/LintCode/blob/master/Java/Unique%20Binary%20Search%20Tree.java) Level: Medium Tags: [BST , DP, Tree]

Not quite clear. The principle is summarized according to the left and right splits.Each split, there will be a certain amount of permutation on the left and right sides.Of course, the total number of cases is multiplied. Then add each different split point: f (n) = f (0) * f (n-1) + f (1) * f (n-2) + ... + f (n-2) * f (1) + f (n-1) * f (0)

Then convert the mathematical formula into the DP equation, which is a bit metaphysical! It's hard to think.

---

33. [Unique Paths II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Unique%20Paths%20II.java) Level: Medium Tags: [Array, Coordinate DP , DP]

Like the grid of the unique path, the target goes to the bottom right corner, but there may be obstacles in the grid, which cannot be crossed. Find the count of the unique path.

Coordinate DP

-dp [i] [j]: # of paths to reach grid [i] [j] -dp [i] [j] = dp [i-1] [j] + dp [i] [j-1]; -Consider the state needed at the end: how to compose and write the formula.-In the formula, pay attention to the blocks that can be skipped, marked as 1. 'Unable to reach', which is 0 path in dp [i] [j].

---

34. [Bomb Enemy.java] (https://github.com/awangdev/LintCode/blob/master/Java/Bomb%20Enemy.java) Level: Medium Tags: [Coordinate DP, DP]

2D grid, each grid may be 'W' wall, 'E' enemy, or '0' empty.

A bomb can be blown in 4 directions. Find out how many enemies can be blown up on the grid.

Corrdinate DP

-Space, Time: O (MN) -dp [i] [j] is the maximum amount of enemy on (i, j) -dp [i] [j] needs to be added from 4 directions, that is, it has to go through 4 directions, so it is divided into UP / Down / Left / Right 4 int [] [] -Find max in the last step -The method of considering directions is easy to think of, but the code for moving in four directions is tedious. -Fry up: Think from top to bottom -Fry Down: Think Bottom Up -Skilled in writing 2D array index transformations.

There seems to be a more concise method, using a col count array: http://www.cnblogs.com/grandyang/p/5599289.html

---

35. [3Sum Smaller.java] (https://github.com/awangdev/LintCode/blob/master/Java/3Sum%20Smaller.java) Level: Medium Tags: [Array, Two Pointers]

The general O (n3) is definitely not working. Optimize on this basis. When j, k is satisfied, (k-j) is all the cases where sum <target. And once> target, because j can not go back, only k--, then the problem is locked. This can be done O (n2)

---

36. [Update Bits.java] (https://github.com/awangdev/LintCode/blob/master/Java/Update%20Bits.java) Level: Medium Tags: [Bit Manipulation]

Some tricks familiar with bits: -~ 0 = -1 = 111111 ... 11111111 (32-bit) -Create mask by shifting right >>>, and shifting left -Reverse to get 0000 ... 11110000 format mask -& 0000 = clean up; | ABC = assign ABC

---

37. [Maximum XOR of Two Numbers in an Array.java] (https://github.com/awangdev/LintCode/blob/master/Java/Maximum%20XOR%20of%20Two%20Number .java) Level: Medium Tags: [Bit Manipulation, Trie]

More difficult to think of. Using the XOR property A ^ B = C, then A = B ^ C.

1. Enumerate the possible A, 2. Then guess one by one.

2. Enumeration A: Because MAX must be the number with the largest leading-1, then the enumeration A starts from (1000000 ... 000), Take 1 or 0 for one more bit at a time

3. Because A is enumerated according to each bit, then B and C must also appear with the same digits. Here, B and C have become the form of prefix and placed in the set. Similar to the idea of 2sum using hashmap, each time using the enumerated A ^ B = C to see if the result C is already in the set. If at, it is proved that the enumerated A may be derived by B ^ C, then a case is found.

Some tricks are also used: mask = (1 << i); // i-bit mask mask = mask | (1 << i); // prefix mask

---

38. [Perfect Squares.java] (https://github.com/awangdev/LintCode/blob/master/Java/Perfect%20Squares.java) Level: Medium Tags: [BFS, DP, Math, Partition DP]

Given a number n, find out how many squares it can consist of at least.

Square number such as: 1, 4, 9, 16 ... etc

Partition DP

-Encounter the most value, think of DP. -Seeing split words, thinking of split DP. -Thinking, if j * j = 9, then j = 3 is the minimum step; but if it is 10, it will be divided into 1 + 9 = 1 + j * j -Consider the last number: if 12 is cut out of 1, how will the remaining 11 be considered? Cut out 4 and considered 8? -Partion method: When considering dp [i-x], x is not 1, but x = j * j. -Becomes dp = Min {dp [i-j ^ 2] + 1}

time complexity

-At first glance it is O (n * sqrt (n)). Actually. But how to derive it? -Consider the upper limit: turn small numbers into large derivation upper limits; consider the lower limit: integrate numbers into small ones to find the lower limit. -Consider sqrt (1) + sqrt (2) + ... sqrt (n): find the upper bound and lower bound of this. -Finally found that it is A * n * sqrt (n) <= actual time complexity <= B * n * sqrt (n) -Then O (n * sqrt (n))

BFS

-minus all possible (i * i) and calculate the remain -if the remain is new, add to queue (use a hashset to mark calculated item) -find shortest path / lowest level number

Previous Notes

-No clue at first. I checked the hint -1. The first step came to mind. From a mathematical point of view, it may start from the largest perfect square number. -2. Then the idea comes to dp. Assuming the maxSqrNum is used in the last step, then the remaining dp [i-maxSqrNum ^ 2] +1 is not good? -3. I did, and found a problem. . . Select maxSqrNum in the last step? For example, 12 is an example. -Then when prompted, think of BFS. Shun. Try everything from 1 to maxSqrNum. Find the smallest one. -Look at the example where I split 12. That's very visually BFS. -During the interview, if you are uncertain at this stage, talk to the interviewer and you may be prompted by BFS.

---

39. [Backpack VI.java] (https://github.com/awangdev/LintCode/blob/master/Java/Backpack%20VI.java) Level: Medium Tags: [Backpack DP, DP]

Give an array of nums, all positive numbers, no repeated numbers; find: # of method to spell out m.

The numbers in nums can be reused. Different orders can be counted as different spellings.

Backpack DP

-dp [i] means: # of ways to fill weight i -1 dimension: dp [w]: There are many ways to fill weigth w. There are as many possibilities as before, just as many as: -dp [w] = sum {dp [w-nums [i]]}, i = 0 ~ n

Analysis

-When fighting for a backpack, there can be duplicate items, so it doesn't make sense to think about 'which unique item was put last'. -The backpack problem is always inseparable from weight. -It's very similar to coin chagne: consider the value / weigth of the last thing put, regardless of which one.

---

40. [Binary Search Tree Iterator.java] (https://github.com/awangdev/LintCode/blob/master/Java/Binary%20Search%20Tree%20Iterator.java) Level: Medium Tags: [BST , Design, Stack, Tree]

Draw, BST in order traversal. Record the minimum value with stack and put it on top. O (h) space. Every time you consume a TreeNode, you look at the rightNode (in fact, the next smallest candidate), and stack all the left children of the rightNode in a one-stop stack.

Previous Notes: Use O (h) space:

Understand the law of binary search tree inorder traversal: First find left.left.left .... left in the end, here is added to the stack. Then consider parent, then right.

For example this question: The top in the stack, which is the node in the bottom left corner of the tree, is considered first, and it is named rst. In fact, after this rst is taken out, it is also the bottom left parent at the same time, considering the bottom parent. Finally, consider the lowest level parent.right, which is rst.right.

note: next () actually has a while loop, which is probably O (h). The title requires average O (1), so it is also okay.

Use O (1) space: there is no stack, and the update current is always the minimum value.

Find the next smallest value, if there is a right child in current:
Similar to iteration when using stack, then find the left-most child of current.right again, which is the minimum value.

If current has no right child:
Then look for the upper right parent of current node, search in BinarySearchTree from root.

note: Be sure to find the parent that meets parent.left == current. Conversely, if current is the right child of the parent, the parent will be reprocessed in the next round. But there is a mistake: the parent in the binary search tree is less than the right child, that is, it must have been visited in the previous step, so it will endlessly loop.

---

41. [Flatten Nested List Iterator.java] (https://github.com/awangdev/LintCode/blob/master/Java/Flatten%20Nested%20List%20Iterator.java) Level: Medium Tags: [Design , Stack]

Method 1: Use queue to type all the items you need Method 2: Use stack to store the required items first, and add them back to the stack every time you open a subsequence.

---

42. [Best Time to Buy and Sell Stock with Cooldown.java] (https://github.com/awangdev/LintCode/blob/master/Java/Best%20Time%20to%20Buy%20and%20Sell 20with% 20Cooldown.java) Level: Medium Tags: [DP]

Sequence DP Much like StockIII. Analyze the state of HaveStock && NoStock, and then look at the last step.

---

43. [Find Peak Element.java] (https://github.com/awangdev/LintCode/blob/master/Java/Find%20Peak%20Element.java) Level: Medium Tags: [Array, Binary Search]

binary search. Goal: find peak, where both sides are descending When the leftmost and rightmost are Integer.MIN_VALUE, it can also constitute the condition that the middle number mid is peak.

Note: There is no need to specifically check (mid-1) <0 or (mid + 1)> = n. prove:

1. Leftmost end: when start = 0, end = 2 => mid = 1, mid-1 = 0;
2. Rightmost end: when end = n-1, start = n-3; mid = (start + end) / 2 = n-2; Then mid + 1 = n-2 + 1 = n-1 <n is taken for granted

---

44. [Longest Common Subsequence.java] (https://github.com/awangdev/LintCode/blob/master/Java/Longest%20Common%20Subsequence.java) Level: Medium Tags: [DP, Double Sequence DP, Sequence DP]

Give two strings, A, B. Find the LCS in these two strings: the longest common character length (it does not need to be a continuous substring)

Double Sequence DP

-Set the length of dp to (n + 1), because dp [i] is used to represent the state of the previous i (ith), so when the length is required, i + 1 can be in the i position and hold i. -Double sequence: The relationship between two sequences, both from the end of the character, analyze 2 cases: -1. The last character of A is not in the common sequence or the last character of B is not in the common sequence. -2. The last characters of A / B are in the common sequence. Overall count + 1.

---

45. [Letter Combinations of a Phone Number.java] (https://github.com/awangdev/LintCode/blob/master/Java/Letter%20Combinations%20of%20a%20Phon Level : Medium Tags: [Backtracking, String]

Method 1: Iterative with BFS using queue.

Method 2: Recursively adding chars per digit

---

46. [Pow (x, n) .java] (https://github.com/awangdev/LintCode/blob/master/Java/Pow (x,% 20n) .java) Level: Medium Tags: [Binary Search, Math]

O (n) if you do it silly, consider O (logN) if you want to do better. Reduce double counting, everything in half.

note: -odd even number of n / 2 -positive and negative of n -Case of n == 0, case of x == 0, case of x == 1.

---

47. [Construct Binary Tree from Preorder and Inorder Traversal.java] (https://github.com/awangdev/LintCode/blob/master/Java/Construct%20Binary%20Tree%20from%20Pre .java) Level: Medium Tags: [Array, DFS, Divide and Conquer, Hash Table, Tree]

As the title

DFS

-Same idea as Construct from Inorder && Postorder. -Write out the letter examples of Preorder and Inorder, and find that the beginning of Preorder is always the root of this level. Write helpers accordingly, pay attention to the index. -Similar to Convert Sorted Array to Binary Tree, find the corresponding index, and then: -node.left = dfs (...), node.right = dfs (...) -Divide and Conquer -optimize on finding `mid node` : given value, find mid of inorder: -Instead of searching linearly, just store inorder sequence in `map <value-> index>`, O (1) -IMPORATANT: the mid from inorder sequence will become the main baseline to tell range: - `range of subTree = (mid-inStart)` -sapce: O (n), time: O (n) access

---

48. [Add Two Numbers.java] (https://github.com/awangdev/LintCode/blob/master/Java/Add%20Two%20Numbers.java) Level: Medium Tags: [Linked List, Math]

LinkedList has been reversed, just do it. Traverse the two l1, l2 to handle the carry-on, generate a new node each time, and finally check the carry-on.

It is exactly the same as Add Binary's understanding.

note: Linked List has no natural size. Use DummyNode (-1) .next to hold the result.

---

49. [Add Two Numbers II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Add%20Two%20Numbers%20II.java) Level: Medium Tags: [Linked List]

Singly-linked list requires reverse, use stack. The final result should be restored to the sequence direction like the input list, which can be done just by pop () one by one.

The addition is the same: 1.sum = carry 2.carry = sum / 10 3.sum = sum% 10;

---

50. [Balanced Binary Tree.java] (https://github.com/awangdev/LintCode/blob/master/Java/Balanced%20Binary%20Tree.java) Level: Medium Tags: [DFS, Tree]

Give a binary tree to see if it is height-balanced

DFS

-DFS using depth marker: Save every depth. Then if there are any conditions that do not meet the requirements, save it as -1. -Once there are <0 or the difference is greater than 1, all return Integer.MIN_VALUE. Integer.MIN_VALUE is extreme, to ensure the correctness of the result. -The final comparison returns whether the result is <0. If it is <0, then false. -Traverse entire tree, O (n)

DFS, maxDepth function

-Same concept as in 1, but cost more traversal efforts.

---

51. [Populating Next Right Pointers in Each Node.java](https://github.com/awangdev/LintCode/blob/master/Java/Populating%20Next%20Right%20Pointers%2 Level: Medium Tags: [DFS, Divide and Conquer, Tree]

Give a special binary tree, treeNode has a next pointer inside.

Write a function that connects all nodes to the level node. The rightmost node.next = NULL

DFS + Divide and Conquer

-The title requires DFS. I figured out how to consider several situations at the DFS level. It is easy to write. NOT BFS, because requires O (1) space -For a root, there are only a few points to avoid: root.left, root.right, root.next. -Find a way to connect the dots in these three directions: -1. `node.left.next = node.right` -2. If `node.next! = Null`, link `node.right.next = node.next.left`; -Then in dfs (root.left), dfs (root.right) -Time: visit && connect all nodes, O (n)

BFS

-It is not the same as the title, and it uses queue space, which is proportional to Input. It's too big. -BFS over Tree. Use Queue and queue.size (), old rules. -For each queue of the process, pay attention to adding the next pointer.

---

52. [Validate Binary Search Tree.java](https://github.com/awangdev/LintCode/blob/master/Java/Validate%20Binary%20Search%20Tree.java) Level: Medium Tags: [BST , DFS, Divide and Conquer, Tree]

If so, verify that it is BST.

DFS

-View each parent-child relationship: leftchild <root <rightChild; -BST has two extremes: left-most-leaf is the smallest element, and right-most-leaf is largest -imagine we know the two extreme border: Integer.MIN_VALUE, Integer.MAX_VALUE; pass node around and compare node vs. node.parent. -Method: Pass root.val as max or min, and check children

- **Note:**

- min / max long type when needed.
- If the title really gives node.val = Integer.MAX_VALUE, we need to be able to compare it with long.

---

53. [Convert Sorted List to Binary Search Tree.java](https://github.com/awangdev/LintCode/blob/master/Java/Convert%20Sorted%20List%20to%20Binary% Level: Medium Tags: [BST, DFS, Divide and Conquer, Linked List]

As the title, convert a sorted singly linked list into a height balanced BST

DFS

-Divide and Conquer
-Find the mid node -Then split the two halves, and make dfs to make two subtrees: node.left, node.right -Use the length to locate the mid, every time you find the middle point to be the root, then the first half and the second half are dfs with length. -Find the mid with fast pointer. Better: no need to traverse entire linked list

Details

-slowPointer = node; -fastPointer = node.next; -Then put root = mid.next
-Then start sortedListToBST (mid.next.next); // Last half
-mid.next = null; // Very important, you have to break the sequence
-sortedListToBST (head); // first half from the beginning
-Finally root.left, root.right merge.

---

54. [Flatten Binary Tree to Linked List.java] (https://github.com/awangdev/LintCode/blob/master/Java/Flatten%20Binary%20Tree%20to%20Linked% Level : Medium Tags: [Binary Tree, DFS]

Give a binary tree, and make the tree a linked list, in-place.

DFS

-After analyzing the intent, follow the intent: Flatten the tree, no extra space. -1. reserve right child: `reservedRightNode` -2. Connect `root.right = root.left`, DFS flatten (root.right) -3. Receiving flowers, coneect end of list-> reservedRightNode -4. flatten the rest. Root.right ...

Note

-The order must be clear, you can't write it wrong, you can see it by writing a few examples -For those nodes that move, clean up node.left = null

---

55. [Minimum Size Subarray Sum.java] (https://github.com/awangdev/LintCode/blob/master/Java/Minimum%20Size%20Subarray%20Sum.java) Level: Medium Tags: [Array , Binary Search, Subarray, Two Pointers]

time: O (n) space: O (1)

Given a string of positive integers, find the shortest subarray sum, where the sum> = s

Two pointer

-All are positive integers, so preSum must be growing. -Then actually use two pointer: `start = 0, end = 0` to keep moving forward. Strategy: -1. end ++ until a solution where sum> = s is reached -2. Then move start; record each solution, Math.min (min, end-start); -3. Then move end and look down -Note: Although it looks like a nested loop at first glance, after analysis, I found that it is actually a loop that moves according to the end pointer. start moves one space at a time. Overall, it's still O (n)

Binary Search

-O (nlogn) NOT DONE.

Double For loop

-O (n ^ 2), inefficient

---

56. [Longest Substring Without Repeating Characters.java] (https://github.com/awangdev/LintCode/blob/master/Java/Longest%20Substring%20Without%20Repeati

Level: Medium Tags : [Hash Table, String, Two Pointers]

method 1: Two Pointers Double pointer: Iterate from start, but the first step is to advance the end while loop; until the end cannot be pushed, then start ++ Ingenious point: Because end is a peripheral variable, on the start loop, end will not reset; [star ~ end] does not need to be calculated in the middle. Eventually O (n);

Previous verison of two pointers: Use two pointers, head and i. Note: head is likely to be returned to an earlier place, such as abbbbbba. When it encounters the second a, head turns into head = 0 + 1 = 1.
Of course this is wrong, so make sure that the head keeps growing and does not backtrack.

Method 2: HashMap <Char, Integer>: <character, last occurance index> Once there are duplicates, rest map. When there is no repetition, continue to map.put (), and then find the max value

Problem: After each reset map, it starts from the oldest index, and the worst case is O (n ^ 2): 'abcdef .... xyza'

---

57. [Remove Nth Node From End of List.java] (https://github.com/awangdev/LintCode/blob/master/Java/Remove%20Nth%20Node%20From%20End%2 Level: Medium Tags: [Linked List, Two Pointers]

O (n), one pace, no extra space Find the window, pan, and finally skip a node between pre and head.

---

58. [Linked List Cycle II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Linked%20List%20Cycle%20II.java) Level: Medium Tags: [Linked List, Math, Two Pointers]

LinkedList has cycle, find the starting point of cycle (the first repeated element).

Slow, fast Pointer

-Speed pointer, O (1) space. -1. After confirming the cycle 2. Mathematical problem: Find the beginning. -When head == slow.next, head is the cycle starting point. -In other words, when slow moves to that traceback point, the point of slow.next is exactly the point of head ...

Proof

-1. Suppose the slow pointer moves t steps, and the fast pointer doubles, which is 2t. -2. We assume that the length of the cycle is Y, and the length before entering the cycle is X. -3. Suppose that the slow pointer goes m cycles, and the fast pointer goes n cycles, and the two pointers meet. -4. Finally met at point K in the Y cycle, that is, both hands took K steps in the last lap. -Then: -t = X + mY + K -2t = X + nY + K -? Integration formula: X + K = (n-2m) Y -Here m and n are just integer number of laps, that is to say X and K are added together, it is the end cycle. X and K are complementary -Conclusion: When the slow / fast pointers meet at point K, and then take step X, the starting point of the cycle is reached, which is the starting point required by the question.

Hash Table, O (n) space

---

59. [Kth Smallest Element in a Sorted Matrix.java] (https://github.com/awangdev/LintCode/blob/master/Java/Kth%20Smallest%20Element%20in%20a%20 Level: Medium Tags: [Binary Search, Heap]

time: O (n + klogn) space: O (n)

Give a sorted matrix, find kth smallest number (not distinct)

Related: Kth Largest Element in an Array

PriorityQueue

-Similar to Merge K sorted Array / List: Use PriorityQueue. -Because the element of Array cannot find next directly, use a class node to store value, x, y positions. -Initial O (n) time, also find k O (k), sort O (logn) => O (n + klogn) -Variant: Kth Largest in N Arrays

Binary Search

-we know where the boundary is start / end are the min / max value. -locate the kth smallest item (x, y) by cutt off partition in binary fasion: -find mid-value, and count # of items <mid-value based on the ascending matrix -O (nlogn)

---

60. [Find Minimum in Rotated Sorted Array.java] (https://github.com/awangdev/LintCode/blob/master/Java/Find%20Minimum%20in%20Rotated%20Sorte Level : Medium Tags: [Array, Binary Search]

After drawing, after the minimum value is rotated, it becomes the lowest valley in the middle of the array. Also, the minimum value of the left slope is greater than the maximum value of the right slope. Use this to judge binary search.

O (nlogn)

---

61. [Connecting Graph.java] (https://github.com/awangdev/LintCode/blob/master/Java/Connecting%20Graph.java) Level: Medium Tags: [Union Find]

Haven't run this program, it is a simple implementation of UnionFind. Document describes the calculation principles / ideas of each link.

---

62. [Connecting Graph II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Connecting%20Graph%20II.java) Level: Medium Tags: [Union Find]

Lint can't run yet, all according to the intent and answer document.

---

63. [Connecting Graph III.java] (https://github.com/awangdev/LintCode/blob/master/Java/Connecting%20Graph%20III.java) Level: Medium Tags: [Union Find]

It is still a variant of UnionFind, this time it is calculated how many unions are left. In fact, it is very simple, maintain a global variable count: At the beginning count = n, because all elements are in bulk; each time union, count--.

---

64. [Number of Islands.java] (https://github.com/awangdev/LintCode/blob/master/Java/Number%20of%20Islands.java) Level: Medium Tags: [BFS, DFS, Matrix DFS, Union Find]

Give a 2Dmatrix, which is 1 and 0, find #of island.

DFS

-More or less like a graph problem: visit all nodes connected with the starting node. -top level has a double for loop, look at each point. -Whenever it encounters 1, count + 1, then the DFS helper function marks everything related to this island as '0' -This ensures that every visited island is cleaned -O (mn) time, visit all nodes

Union Find

-You can use union-find, just like Number of island II. -But this is not a Return list, but just # of islands -Union Find is independent from the problem: it models the union status of integers. -Remember the template and several variations of UnionFind (Connecting Graph I, II, III), and the final code is easier to write.

---

65. [Surrounded Regions.java] (https://github.com/awangdev/LintCode/blob/master/Java/Surrounded%20Regions.java) Level: Medium Tags: [BFS, DFS, Matrix DFS, Union Find]

Give a 2D board with 'X' and 'O'. Paint all areas surrounded by X with 'X'.

Starting from the edges of the four sides, spreading like a zombie virus, mark all the nodes on the sides, then change the 'O' to X, and finally reply marks-> 'O'

Union Find

-This time, the concept of rank is used in UnionFind, which needs review. Rank [] is the size of the union where each node is tracked. -The purpose is: always merge into the big union -note: Convert 2D coordinate (x, y) to 1D: index = x * n + y

DFS: mark all invalid 'O'

-Reversed thinking: find surrounded nodes, how about filter out border nodes && their connections? -Need to traverse all the border nodes, consider dfs, visit all. -loop over border: find any 'O', and dfs to find all connected nodes, mark them as 'M' -time: O (mn) loop over all nodes to replace remaining 'O' with 'X'

DFS: mark all valid 'O'

-More like a graph problem: traverse all 'O' spots, and mark as visited int [] [] with area count [1-> some number] -Run dfs as top-> bottom: mark area count and dsf into next level -End condition: if any 'O' reaches border, mark the global map <count, false> -keep dfs untill all connected nodes are visited. -At the end, O (mn) loop over the matrix and mark 'X' for all the true area from map. -Practice: write code to verify

BFS

-TODO

---

66. [Implement Trie (Prefix Tree) .java] (https://github.com/awangdev/LintCode/blob/master/Java/Implement%20Trie%20 (Prefix% 20Tree) .java) Level: Medium Tags: [Design, Trie]

Implement Tire, also known as Prefix Tree. Do three functions: insert, search, startWith

Trie

-HashMap builds Trie. Trie three methods: -1. Inset: add word
-2. Search: find word
-3. StartWith: find prefix

Features

-Only two children are binary tree. Then multiple children are Trie -So how do I find a child without a left / right pointer?
-Use HashMap, take child's label as Key, and value is child node. HashMap moves

other

-The char in the node is optional here. Just use the map to store the children distributed downwards in each TrieNode.
-There is another problem, such as related to other types of search. If you want to return whole string at the end, you can store an up-to-this-point String in node.

Previous Note

-If you encounter a one-word query, consider it. -Pay attention when building TrieNode: how to find children? If it's a map, it's actually pretty good. -Moreover, the char or string in each node is sometimes not very useful. -Can be empty. However, for some topics, such as what String to return at the end, a true String can be stored at the end string.

---

67. [Add and Search Word-Data structure design.java] (https://github.com/awangdev/LintCode/blob/master/Java/Add%20and%20Search%20Word%20-%20Data%20structure% 20design.java) Level: Medium Tags: [Backtracking, Design, Trie]

Trie structure, the deformation of the prefix tree: '.' Can replace any character, then iterate all children of this node.

There are char, isEnd, HashMap <Character, TrieNode> inside the node
Build trie = Insert word: Add without node, move with node.
Search word: Report an error without node. Return true to the end

This problem is because '.' Can replace any possible character. None of them is a new path, so recursive is better.
(iterative is about to be queuing, please be troublesome)

---

68. [Word Search.java] (https://github.com/awangdev/LintCode/blob/master/Java/Word%20Search.java) Level: Medium Tags: [Array, Backtracking, DFS]

DFS, Backtracking:

-Find the first letter, then recursively DFS to string the word to the end: -For each letter, go in four directions, one of them can be true. -Note: Each time you reach a letter, mark '#'. After 4 path recurse returns, mark it back.

Note: other ways of marking visited:

-Use a boolean visited [] [] -Use hash map, key = x @ y

---

69. [Decode String.java] (https://github.com/awangdev/LintCode/blob/master/Java/Decode%20String.java) Level: Medium Tags: [DFS, Divide and Conquer, Stack]

Give an expression string. It contains numbers, letters, parentheses. The number represents the content of the parentheses repeated several times.

The parentheses can be String or expression.

Purpose: Expand expression into a normal String.

Stack, Iteratively

-Process inner item first: last come, first serve, use stack. -Record number globally and only use it when '[' is met. -Stack stores the contents of [], detect brackets at the beginning and end: process inner string at the end -There are many details that need attention to get it right: -Stack can also be used, pay attention to the cast everywhere. Need to be stored is Object: String, Integer -Several type checks: instanceof String, Character.isDigit (x), Integer.valueOf (int num) -When it comes out: sb.insert (0, stack.pop ())

DFS

-Bottom-> up: find deepest inner string first and expand from inside of [] -Similar to some features to consider when stacking. Special points: Check the end of []
-Because in DFS, the substring in parentheses will be retained to enter the next level, so we need to keep track of substring at the base level. -Use int paren to track the opening and closing of parentheses, and find closure ']' when paren == 0 again -Other times, continue to append to substring

---

70. [Maximum Binary Tree.java] (https://github.com/awangdev/LintCode/blob/master/Java/Maximum%20Binary%20Tree.java) Level: Medium Tags: [Stack, Tree]

Give a string of numbers and make a maximum binary tree: the top root is the largest; the left child is also a max tree, and the right child must also be a max tree.

Monotonous Stack

-Decreasing stack using bottom-> top: the bottom root is maintained as the largest element. -In the process, once currNode.val> stack.peek () is encountered, it means that this currNode needs to be placed at the bottom of the stack. -In other words, when this condition is met, process, pop () all currNode.val> stack.peek (), and finally add currNode.

-The requirements of the maxTree problem itself are: the big one is in the middle, and the left and right subTrees must also be maxTree: -Monotonous Stack helps keep / track of max value here, but the logic of left / right tree is unique to MaxTree. -The assignment of the left / right node is based on the requirements of the title: after the middle maximum is separated, the left is the left subTree, and the right is the right subTree.

Previous notes

-Should memorize MaxTree. By analogy, you will do Min-Tree, Expression Tree -In Stack, the maximum value is below. Using this property, there are several steps:

Step1

-Pop out all the less than curr nodes, while loop, keep it going.
-The last node that popped out is smaller than Curr: it is also the largest popped out of the stack that is smaller than the curr, which is the closest to the curr size. (Because the maximum value of this stack is below)
-Put this largest node smaller than curr in curr.left.

Step2

-Then, the next stack must be greater than curr:
-That becomes the left parent of curr. Set stack.peek (). Right = curr.

Step3

-End: The bottom of the stack must be the largest one, which is the head of the max tree.

---

71. [Swap Nodes in Pairs.java] (https://github.com/awangdev/LintCode/blob/master/Java/Swap%20Nodes%20in%20Pairs.java) Level: Medium Tags: [Linked List]

enumurate

The basic principle, write it out, and then wire: pre-> A-> B-> C-> ... A virtual preNode is required as the starting node, otherwise the subsequent nodes cannot be changed to the beginning.

Note

Use dummy = pre as the previous box of the head. Use `pre.next == null && pre.next.next` to check if it is NULL. pre.next.next guarantees at least one swap.

---

72. [Wood Cut.java] (https://github.com/awangdev/LintCode/blob/master/Java/Wood%20Cut.java) Level: Medium Tags: [Binary Search]

The idea of dichotomy: the judgment is a validate () function, not a simple '=='

No sorting is needed! Why? Because we are not dicing on the given array, we are dicing on [0, max] based on the maximum value.

Overall time: O (nLogM), where M = largest wood length

---

73. [Find the Duplicate Number.java] (https://github.com/awangdev/LintCode/blob/master/Java/Find%20the%20Duplicate%20Number.java) Level: Medium Tags: [Array , Binary Search, Two Pointers]

-Be careful not to think about formulas: think mid is index -Here mid is actually a value of binary search on value [1, n]. -Use validate () function again

Time: O (nLogN)

---

74. [Game of Life.java] (https://github.com/awangdev/LintCode/blob/master/Java/Game%20of%20Life.java) Level: Medium Tags: [Array]

basic

-Simple implementation, just write the count function clearly. -time: O (mn), extra space: O (mn) -Note that the board [i] [j] copy

follow up

unlimited border? It may be necessary to split the board. Use a large frame to split, and each time you wrap a line, repeat the calculation 2 times.

improvement: do it in place!

-time: O (mn), extra space: O (1) -bit manipulation: use the second bit to store the previous value. -Because we only consider 0 and 1, we can do it this way. But it is not scalable. -If you need to store multiple states, you may need to move more bits, or use a state map -Note the details of bit manipulation: << 1, >> 1, and mast usage: |, &

---

75. [Number of Airplane in the sky.java] (https://github.com/awangdev/LintCode/blob/master/Java/Number%20of%20Airplane%20in%20the%20 Level : Medium Tags: [Array, Interval, PriorityQueue, Sort, Sweep Line]

Sweep Line

-Split Interval into points on the number line -Take off mark 1
-Landing mark -1
-Sort by PriorityQueue, loop through queue, and calculate the possible max (takeoff + landing) value.

Note

-Take off and land at the same time, which is 1-1 = 0. So there is a second while loop in the while loop,
-When coordinates x are coincident, add and subtract all x points here, and then compare max.
-This avoids wrong counts or counts

---

76. [Meeting Rooms II.java](https://github.com/awangdev/LintCode/blob/master/Java/Meeting%20Rooms%20II.java) Level: Medium Tags: [Greedy, Heap, PriorityQueue, Sort, Sweep Line]

Give a string of numbers to represent the start / end time of the meeting. Find out how many meetings happen at the same time (how many rooms are needed)

PriorityQueue

-PriorityQueue + a Class to resolve. O (nlogn) -Same question as Number of Airpline in the sky

Method 2: Tried it with a sorted Array + HashMap

It's ok, but HashMap should be careful when handle edge, because the map key of start and end will be repeated at the same time.

---

77. [Unique Path.java](https://github.com/awangdev/LintCode/blob/master/Java/Unique%20Path.java) Level: Medium Tags: [Array, Coordinate DP, DP]

2D array, count how many ways to go to the bottom right corner.

DP

-Count DP. Note that the first two positions of the equation are added together: the first two cases have no overlap, and there is no shortage of cases. -Note the initialization, return to 1. -The reason for initialize is that it is also a reminder: -1index will appear in the equation -Of course, row i = 0, or col j = 0, there is only 1 way to access -time O (mn), space O (mn)

Scrolling array

-[i] is only related to [i-1]. Use curr / prev to build a rolling array. -space O (n) optimize space

---

78. [Maximal Square.java](https://github.com/awangdev/LintCode/blob/master/Java/Maximal%20Square.java) Level: Medium Tags: [Coordinate DP, DP]

You can only go to the right and below, and find the square with the largest area. That is, find the square that has become the longest.

DP

-Square, each side needs to be the same length. -Taking the lower right corner as the point of consideration, the conditions must be met: the points of left / up / diagonal are all 1 -And, if all three points are derived in three directions, the longest square edge is the shortest edge among them (limited by the shortest edge) -dp [i] [j]: max square length when reached at (i, j), from the 3 possible directions -dp [i] [j] = Math.min (Math.min (dp [i-1] [j], dp [i] [j-1]), dp [i-1] [j-1]) + 1; -Space, time O (mn)

init

Each point may have a side length of 1, if matrix [i] [j] == '1'

Scrolling array

The relationship between [i] and [i-1], think of rolling arrays to optimize space, O (n) sapce.

---

79. [Coins in a Line.java](https://github.com/awangdev/LintCode/blob/master/Java/Coins%20in%20a%20Line.java) Level: Medium Tags: [DP , Game Theory, Greedy]

Take the chess piece game, each person can take 1 or 2 and take away the loss of the last child. Q: Based on the given chess piece loss, can I determine the winning or losing of the first mover?

Game Theory: If I want to win, the situation I get back must be 'possible to lose'.

DP, Game Theory

-To win, it must be guaranteed that when the opponent gets the board, in all cases where he can go, it is 'possible to lose', that is enough. -Design dp [i] : indicates whether I can win when i face the situation of coins, depending on whether I can lose one or two, can the opponent lose? -dp [i] =! dp [i-1] ||! dp [i-2] -Time: O (n), space O (n)-Game problems, often analyzed from the perspective of 'my first step', because the situation is the simplest at this time.

Rolling Array

space optimization O (1). Rolling array,% 2

---

80. [Coins in a Line II.java](https://github.com/awangdev/LintCode/blob/master/Java/Coins%20in%20a%20Line%20II. java) Level: Medium Tags: [Array, DP, Game Theory, Memoization, MiniMax]

Give a string of coins, represented by values []; each coin has its own value. First hand / second hand game, Only one or two pieces can be taken in order from left to right at a time, and finally who

sees the largest total value. MiniMax's thinking method is very magical, and finally The written expression is simple

DP, Game Theory The self-study process is relatively long

-not the same as Coins in a line I : The value of each coin is different. -Using the idea of MiniMax, here is actually MaxiMin. Reference: http://www.cnblogs.com/grandyang/p/5864323.html-Goal : Make the maximum value of coins that the player gets of.

- set dp [i]:. the maximum value from index i to index n so dp [0] is what we just get in [0 ~ n] of the maximum value
- the same time, you also want the biggest opponent playerB Change , and your choice has to be constrained by your opponent's choice.-Using MaxiMin's thinking, we assume a current state, imagine what the opponent playerB will react (from the opponent's perspective, how to let me lose)-in the disadvantage (Under the goal that the opponent let me lose) Find the largest coins value sum

Estimation expression

-Reference details how expressions are pushed out, in short:-If I choose i, then the opponent I can only choose two positions (i + 1) and (i + 2), and my situation under the control of the other party is min (dp [i + 2], dp [i +3]) -If I choose i and (i + 1), the opponent can only choose two positions (i + 2) and (i + 3), and my position under the opponent's control is min (dp [i + 3], dp [i + 4]) -Everyone can choose 1 or 2 coins

- The above objective is maximize the best result in the two worst-case

simplified expression

-Simplify a bit more: if I'm the first player, dp [i] represents my maximum value. -Depending on whether I take [i], or [i] + [i + 1], the opponent may be dp [i + 1] , Or dp [i + 2] -Actually dp [i] = Math.max (sum-dp [i + 1], sum-dp [i + 2]); -here sum [i] = [i ~ n] 's sum, minus dp [i + 1], the rest is the value of dp [i].

Initialization

-This method is pushed forward from the end, pay attention to initialize the value at the end of dp. -dp = new int [n + 1]; dp [n] = 0; // [n ~ n] when nothing is selected, it is 0. -sum = new int [n + 1]; sum [n] = 0; // when nothing is selected, it is naturally equal to 0 -then remember to initialize (n-1), (n-2)

Time O (n) Space O (n): dp [], sum []

---

81. [Binary Tree Postorder Traversal.java](https://github.com/awangdev/LintCode/blob/master/Java/Binary%20Tree%20Postorder%20Traversal.jav Level: Medium Tags: [Stack, Tree, Two Stacks]

As the title, POST-ORDER traversal.

LeetCode gave hard, it should be difficult to think of the stack approach.

Recursive

trivial, first add left recursively, then add right recursively, and then form the head.

Stack

-The idea of double stack, you need to draw on the drawing -the original order is: leftChild, rightChild, currNode. -Create a stack, reversely process: currNode, then rightChild, then leftChild -this way The result is reverse, so it can be reversed. -V1 uses stack1 and stack2, because it is based on the idea of this dual stack -v2 is simplified and can be placed in a stack, each time the result is recorded When: rst.add (0, item);

Take advantage of stack features

-Each time you add an element to the stack, you want to process it in bottom / after, add it first -you want to process it immediately in the next round, and finally push it into the stack.

Note

these binary tree traversal topics. Often There are multiple approaches: recursive or iterative

---

82. [Compare Version Numbers.java](https://github.com/awangdev/LintCode/blob/master/Java/Compare%20Version%20Numbers.java) Level: Medium Tags: [String]

Give two strings of version number, composed of numbers and '.'. Compare the order.

If version1> version2 return 1, if version1 <version2 return -1, otherwise return 0.

divide and conquer

-Use str.split ("\\.") To split the string -Convert to integer, and break them one by one

Note

- '1.0 ' and '0' are equal
- If you can assume that the version integers are both valid, directly Integer. parseInt () is fine
- otherwise, you can compare string

---

83. [Count Complete Tree Nodes.java] (https://github.com/awangdev/LintCode/blob/master/Java/Count%20Complete%20Tree%20Nodes.java) Level: Medium Tags: [Binary Search, Tree]

Complete Tree means that the last level may be missing nodes (not that the bottom right corner is missing nodes, don't forget!)

DFS + Optimization

-Look at the leftmost left depth and rightmost leaf each time The depth is not the same, if it is the same, directly 2 ^ h-1 is fine -if it is different, then DFS

Trick

-Direct DFS will timeout, O (n), in fact, you can optimize -to pass the test with O (h ^ 2), bit operation: Math.pow (2, h) = 2 << (h-1). Amazing! -2 << 1 is to move all bits one bit to the left, which is * 2

## Iteratively

-See details in comments inline. To understand the tree very well -binary tree one child tree nodes # = 2 ^ h-1; so a child tree + root = 2 ^ h

---

84. [Course Schedule.java] (https://github.com/awangdev/LintCode/blob/master/Java/Course%20Schedule.java)### Level: Medium Tags: [BFS, Backtracking, DFS, Graph, Topological Sort]

-A pile of lessons is represented by an int [2] pair. [1, 0] means that if you want to take lesson 1, you must first take lesson 0. -Each number is an ndoe. The question asks if you can arrange all the lessons. the

- input is numOfCourses, there is this the Prerequisites [[]]

Topological the Sort

- to Nodes of a Graph
- critical: List [] edges with ; `edges [i] = new ArrayList <> ();` expressed graph: each is the Node, the ITS to All neighbors
- the goal is based on the direction edge of this graph inside the node sort a list . - If there are cycle, this item will not be on the final list inside
- For example: if two lessons are dependent on each other, it becomes a cyclic dependency, which is not good.

BFS

-Kahn algorithem: -first build a graph map: <node, list of nodes>; or `List [] edges; edges [ i) = new ArrayList <> ();` -count in-degree: inDegree is on each node, how many edges are there
--### IMPORTANT : always initialize inDegree map / array with 0 -For those without in-coming-edge, indegree is actually equal to 0, then they should be in the final result list -For those nodes BFS with indegree == 0, add to queue. -Each node on the visit queue: count ++, also add this curr node to sorted list -Check all neighbors / edges of curr node: if visit has passed, indegree on this node --if indegree == 0, add this node to queue.

Indegree Principle

-Note: If there is a cycle, there will be more inDegree on this node, it cannot be cleared to 0, and it cannot enter the queue && sorted list. -Remember: indegree is the number of times the surrounding nodes have counted to me
-If After the connection of all the surrounding nodes is cut off, my indegree is not equal to 0, so there must be some nodes that have repeated connections indirectly, that is, cycle -Topological problem: almost always care about cycle case (if detecting cycle is not goal)

DFS

-This question does not require a final list, which is relatively simple, as long as you visit each node and finally confirm that there is no cycle -Use visited int [] to confirm whether there is a cycle. 1 represents paretNode visited, -1 represents the mark on the DFS line. -If -1 is encountered, it means that the node has gone in the same dfs path of the previous level or above. Then, it turns out that there is cycle, return false. -After all the neighbors of a node have walked, there is no fail, then backtracking, set visited [i] = 1 -Topo sort will really be at the bottom of the DFS, and the record will be placed in a stack, finally reverse, is really sort order.

Notes:

-and List [] arrayOfList = new ArrayList []; This operation, instead of map <integer, integerList>-List [] list, In fact, the default List

Previous notes is

a bit confusing, but once you do it, you will understand a little.
Is the topic of topological sort. Usually sort things that have dependencies.

In the end, it will be a sink node, there will be no backward dependency, and it will end at that point.
I already have the point as the key of the map, and the value is a list of courses with this node as the prerequisite.

When drawing a picture, prerequisites all point to the sink node, and when we compose the map, we all diverge back from the sink node to the dependent nodes.

In DFS, we are the reverse, and then, the one that is completely visited first node, it must be the leftmost node, and it has the highest mark seq.

For our sink node, when all its branches have been visited, seq must have been reduced to a minimum, which is 0, and it is the first to be visited.

The end result: every node with pre-requisit traces (bottom-up), and no cycle is found. This means that schedule can be used.

---

85. [Course Schedule II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Course%20Schedule%20II.java) Level: Medium Tags: [BFS , DFS, Graph, Topological Sort]-A

bunch of lessons are represented by int [2] pair. [1, 0] means that if you want to take lesson 1, you must first take lesson 0. -Each number is an ndoe, title Find the list of the last scheduled courses -if the ranking is not good, just leave it empty -the input is numOfCourses, and this prerequisites [[]] -The approach is very similar to Course Schedule I, you can refer to it.

Topological Sort, Indegree, BFS

-Use `List [] edges; edges [i] = new ArrayList <> ();` to represent graph: each node, to all its neighbors -each without inDegree = = 0 node, can be added to the final list. For example, those nodes with inDegree = 0 found at the beginning -Note that if prerequisites = [], it means that these lessons are independent, open an int [0 ~ n- 1] arrays and assignments are fine.-If there is a cycle, topological sort cannot be performed in a strict sense, and all nodes cannot be covered, then return []

DFS

-Modify according to DFS in Course Schedule -Maintain visited int [] Global variables -maintain sortedList int [] global variables, note that when added, add (0, node) is added at the beginning -every time after all DFS children of a node have been walked, you can add him to final In the list -if there is a cycle, that is, when dfs returns false, the problem is judged to be a failure, and return new int [] {}

. 86 [Contains Duplicate III.java] (https://github.com/awangdev/LintCode/blob/master/Java/Contains%20Duplicate%20III.java) Level: Medium Tags: [BST]

to a unsorted array, ask, if there are two elements, the difference between the value is t, and the index difference between the two elements is k.

Note: Although the title name is Contains Duplicate, the two elements you are looking for are not duplicates, but Math.abs (value1-value2) <= t.

TreeSet

-TreeSet is still a set, which we use to hold items that have been visited -if the window size exceeds K, then remove nums [i-k-1] And add the new element -here is a formula to calculate: (Math.abs (AB) <= t) = >>>>> (-t <= A-B <= t) = >>>>>> A> = B-t, A <= B + t -That is, if for B = nums [i], you can find a target A that satisfies the above formula, then you can return true. -Time O (nLogk), the size of treeSet will not exceed k, and treeSet.ceiling (), treeSet.add (), treeSet.remove () are both O (logK) -Space O (k)

Note

-Similar concept to Containers Duplicate II. TreeSet has BST so you can use it directly without building BST yourself -Simplify the important conditions in the question Math.abs (AB) <= t and infer A> = B- t, A <= B + t -and need to use TreeSet.ceiling (x): return number greater or equal to x. Remember this usage, there is no shortcut.

---

87. [Jump Game .java] (https://github.com/awangdev/LintCode/blob/master/Java/Jump%20Game.java) Level: Medium Tags: [Array, DP, Greedy]

Give the number of steps, see if you can jump to end.

Greedy-start from index = 0

-Keep track of farest can go -Once farest> = nums.length-1, that is, when it reaches the end, you can stop, return true.- Once farest <= i, that is, at point i, I have already taken steps and ca n't jump forward, so return false -This can be done using DP. However, greedy algorithm is fast in this particular problem.

Greedy-start from index = n-1

-greedy: start from end, and mark last index -loop from i = [n-2-> 0], where i + nums [i] should always> = last index -check if last == 0 when returning. It means: can we jump from index = 0 to the end? -Time: O (n), beat 100%

DP

-DP [i]: at point i Record, can the number of steps before point i go to point i? True of false. -In fact, only one of j in [0 ~ i) can reach i -Function: DP [i] = DP [j] && (A [j]> = i-j), for all j in [0 ~ i) -Return: DP [dp.length-1]; -Time: O (n ^ 2)

---

88. [Coin Change 2.java] (https://github.com /awangdev/LintCode/blob/master/Java/Coin%20Change%202.java)### Level: Medium Tags: [Backpack DP, DP]

Give a string of numbers, target amount, how many ways can you reach the amount.

DP

-O (MN): M, total target amount; N: size of coins -Similar to: 2 ways in grid dp, unique path: top to bottom, left to right -state : dp [i]: sum of ways that coins can add up to i. -Function: dp [j] + = dp [j-coins [i]]; -Init: dp [0] = 1 for ease of calculation; other dp [i] = 0 by default -note: Avoid repeated counts, so j = coins [i] as start -Note that coins need to be placed outside the for loop, and dominate the process of changing coins. Each coin can be used countless times, so Try to use each coin on each sum value

knapsack problem: backpack problem

---

89. [Decode Ways.java] (https://github.com/awangdev/LintCode/blob/ master / Java / Decode% 20Ways.java) Level: Medium Tags: [DP, Partition DP, String]

time: O (n) space: O (n)

Given a string of numbers, it should be decoded into English letters. [1 ~ 26] Corresponding English letters. Find out how many methods can be decoded.

Partition DP

-Addition principle: According to the intent, there is range [1, 9] of = 1 and [10 ~ 26] of range = 2 as the partition. -Determine the two states at the end: single letter or combos. Then calculate the case of a single letter, and the case of a double number -Definition `dp [i] = How many decoding methods are available for the first i digits? new dp [n + 1] .-Addition principle: add up the different cases, single-digit, double-digit cases -dp [i] + = dp [i-x], where x = 1, 2 -note: calculate number from characters, need to-'0' to get the correct integer mapping. -Note: check value! = '0', because '0' is not in the condition (AZ) -Space, Time O (n)

Extension

-There are only two cases of range = 1, range = 2. If there are more types of partitions, there may be more A layer of for loop to do the loop

---

90. [Minimum Path Sum.java] (https://github.com/awangdev/LintCode/blob/master/Java/Minimum%20Path%20Sum.java) Level: Medium Tags: [Array, Coordinate DP , DP]

DP

-Time, Space O (MN) -Go to the bottom right corner and calculate the shortest path sum. Typical coordinate type. -Note: When init the first line, accumulate dp [0] [j-1] + grid [i] [j], not just assign grid [i] [j]

Rolling Array

-Time O (MN), Space O (1) -need to be in the same for loop Complete initialization and use dp [i] [j] -Reason: dp [i% 2] [j] is calculated almost immediately in the next round; it is not used until it is overwritten. White calculation -If you follow the first method, initializing the dp at the beginning, it looks simple, but it is not convenient for space optimization

---

91. [Counting Bits.java] (https://github.com/awangdev/LintCode /blob/master/Java/Counting%20Bits.java)### Level: Medium Tags: [Bit Manipulation, Bitwise DP, DP]

Give an array, calculate how many bits there are

Bitwise DP

-For each number, it is actually very simple to calculate: every time >> 1, then & 1 can count 1s. Time: a number can >> 1 O (logN) times -Now calculate all [0 ~ num], that is N numbers, time complexity: O (nLogN).- Use DP to optimize, find 1s count of the number, store Go down in dp [number].- Calculate your order from 0-> num, count can be reused. -Bit title uses the value of num itself to indicate the status of DP.-Here, dp [i] is not and dp [i-1] has a logical relationship; instead, dp [i] and dp [i >> 1] have a direct relationship from the binary representation.

---

92. [Continuous Subarray Sum.java] (https : //github.com/awangdev/LintCode/blob/master/Java/Continuous%20Subarray%20Sum.java) Level: Medium Tags: [Coordinate DP, DP, Math, Subarray]

gives a non-negative sequence and number k (can be positive or negative, can be 0). Find continuous subsequences (length greater than 2), so that the sum of this subarray is a multiple of k. Q: Is it possible?

DP

-O (n ^ 2) -Sum, coordinate type dynamic programming needs to be recorded at 0 ~ i (including nums [i], ending with nums [i].) -Dp [i] = dp [i-1] + nums [i]; -Finally Move, make comparison

Calculate results directly

-from sum = all cases of [i ~ j] each time -verification

---

93. [House Robber II.java] (https://github.com/ awangdev / LintCode / blob / master / Java / House% 20Robber% 20II.java) Level: Medium Tags: [DP, Sequence DP, Status DP]

Similar to House Robber I, search for houses, and neighbors cannot move. The characteristic is : Now nums are arranged in a circle, end to end.

Sequence DP

-dp [i] [status]: under status = [0,1], the max rob gain obtained by the first i house. Status = 0, 1st house robbed; status = 1, 1st house skipped -dp [i]: dp [i] = Math.max (dp [i-1], dp [i -2] + nums [i-1]); -In particular, the last house at the end is connected to the first house. Here we need to discuss two cases: the first house is searched, or the first house is not searched.

- Edge Case BE careful with the nums = [0],. 1 only with Element.
- Time, space: O (n)

Two states

-whether the first house has been searched and two branches are divided, which can be regarded as two states. -You can consider using two DP arrays; you can also add a dp dimension to supplement this state.-Two dimensions represent two states (1st house being robbed or not); these two states are two states of the parallel world , Not related to each other.

Rolling array

-Like House Robber I, you can use% 2 to operate rolling array, space reduced to O (1)

---

94. [House Robber III.java] (https : //github.com/awangdev/LintCode/blob/master/Java/House%20Robber%20III.java) Level: Medium Tags: [DFS, DP, Status DP, Tree]

Houses have been transformed into binary trees, The rules are still the same, consecutively connected houses cannot be copied at the same time.

Find out how much Binary Tree neighbor max can copy.

DFS

-Determine whether the current node is adopted and use a boolean.
it.-If the curr node is adopted, the child below must not be adopted. -If the curr node is not adopted, then The following children may be used, but they may also be skipped, so use Math.max () to compare the two possible dfs results. -dfs repeated calculation: each root has 4 possibilities of dive in, assuming level height is h, then time O (4 ^ (h)), where h = logN, which is O (n ^ 2)

## DP, DFS

-Not just DP, but after finding that DFS is strenuous, can you replace some repeated calculations? -The basic idea is that the dfs solution is the same: take root to find the maximum value, or not take root to find the maximum value -DFS on root, Do not fork before dfs enters; each level stores the corresponding value according to the state: dp [0] root not picked, dp [1] root picked.-Optimization: In DP, find leftDP [] in one breath and dfs to the lowest level , And then do the calculation from the bottom up -in this process, because there is no dfs () forking outside, the calculations will not overlap, and you will not have to go back to visit most-left-leaf, it will be done after one calculation. -However, Ordinary dfs without dp, after calculating visited dfs, you need to dfs again! Visited case. -Space O (h), time O (n), or O (2 ^ h), where h = log (n)

DP Features

-Forking dfs without state-Modeling different states into dp
-Each dfs a dp array based on status. -equal to one-time dfs calculation to the end, and then back track to calculate each layer at the top. -DP does not Be sure to use n as the base. It can also go to the memory state-> value.

---

95. [Permutation in String.java] (https://github.com/awangdev/LintCode/blob/master/Java/Permutation%20in%20String.java) Level: Medium Tags: [Two Pointers]

### Two Pointer

-If you do s1's permudation, the time complexity is O (n!) Definitely not possible -here is the practice of HashTable (because of 26 letters, so use int [26] to simplify) to record the character count in the window

- If the character in the window is equal to COUNT, then that permudation
- further optimization: find two map correspond to each other, as with a int [26]: s1 make additions to character encountered, s2 subtraction of the character encountered
- Two pointers are used in the control of n1, n2; and the step of s2.charAt (i-n1)

Backtracking

96. [Permutations II.java](https://github.com/awangdev/LintCode/blob/master/Java/Permutations%20II.java) Level: Medium Tags: [Backtracking]

Give a list of arrays, Find all permutation arrays. Note: The given nums have duplicate numbers, and the results of permutation need to be non-repeating.

-Sorting, -Mark visited. Check whether the duplicate results are discharged by the permutation rule -and check the previous recursive level Have you skipped repeating element -time O (n!)

Background 1

-There is a for loop in the recursive call, each time starting from i = 0, try to add each of the nums to the current list .
-Since i = 0, every nums will be recursive in turn:-For example, if i = 2, it must be accessed before i = 3. That is: the list permutation with i = 2 must be discharged first.

Background 2

-repeated example: Given Input [x, y1, y2], assuming that the value of y is the same. Then, {x, y1, y2} and {x, y2, y1} are the same result.

Note

-In summary, y1 must be accessed before y2, and {x, y1, y2} comes out first. Immediately after that, in another recursive loop, {x, y2 ...} y2 is accessed first, skipping y1.
-Important: The rule is here. If you skip y1, that is, visited [y1] == false, and num [y2] == num [y1], then this is a duplicate result. There is no need to do it. -Result: Then, we need to input values like {x, y1, y2} and put them together, then we must sort.

Non-recursive, manuall swap

-Idea from: https://www.sigmainfy.com/blog/leetcode-permutations-i-and-ii.html-use sublist sort -Use swap function to adjust the new permutation on the original array -Note: Every time you get a new candidate, you must sort the digits without permutate, and then start swap. -This is to ensure that [j] and [j -1] When repeating, do not need to re-record.

Queue

-give a visited queue -populate with queue in all places. -And then visited indexes are stored in visited. (Not efficient code. Check again)

---

97. [Shuffle an Array.java](https://github.com/awangdev/LintCode/blob/master/Java/Shuffle%20an%20Array.java) * * Level: Medium Tags: [Permutation]

Like shuffle music, make a set of shuffle array functions:

shuffle () gives random permutation -O (n)

reset () gives the initial nums.

Permutation

-Permutation is actually changing the position of the elements on the list / array / ...- hard position, each time the position is different, use random to find the one to be replaced index -Maintain the same random seed

Note

-compute all permutations is too slow to work.

---

98. [Group Anagrams.java] (https://github.com/awangdev/LintCode/blob/master/Java/Group%20Anagrams.java)### Level: Medium Tags: [Hash Table, String]

Give a string, return list of list, put anagram together.

Hash Table, key is character frequency

-store anagram -use character frequency to make unique key -use fixed-length char [26] arr to store the frequency of each letter; then new string (arr).- because the frequency of each seat changes, you can Construct a unique string -O (nk), k = max word length

Hash Table, key is a sorted string

-the same idea as check anagram: transform and sort char array to use as key. -Store all anagrams together. Notice the end of Collections.sort ().-O (NKlog (K)), N = string [] length, k = longest word length

---

99. [Backpack.java] (https://github.com/awangdev/LintCode/blob/master/ Java / Backpack.java) Level: Medium Tags: [Backpack DP, DP]

for i book, each book has its own weight int [] A, backpack has its own size M, see how much weight can be put at most Book?

Backpack DP 1

-Simple and straightforward thinking dp [i] [m]: For the previous book, when the size of the backpack is M, can you hold a variety of books? -### Note : The backpack problem, the weight must be one-dimensional. -Dp [i] [j] = Math.max (dp [i] [j], dp [i-1] [j-A [i-1]] + A [i-1]); -maximum value for each step -last return dp [n] [m] -time and space O (mn) -rolling array, space O (m)

Backpack DP 2

-true / false solution, a little curve to save the country: the point is, finally, according to the weight from large to small, the first one that encounters true, the index is the maximum value.
-consider: use I item (skipable), can it be loaded to weight w?-It needs to be considered from the perspective of 'possibility', not to make a single maximum problem. -1 . The size and total load of the items that can be loaded in the backpack related. -2. Do not look for the maximum total weight of the first i items in dp [i] , not this one. Dp [i] Find the maximum sum that can be put in time, but i + 1 may have a better value, making dp [i + 1] larger and more suitable.

Practice

-boolean [] [] dp [i] [j] means: there are the first i items, can they be used to form a backpack of size j? true / false.- (Considering it, we do n't want to exceed the size j, but consider whether we can spell the exact size == j) -### Note : Although the position of i always exists in the dp, the actual consideration is that at the i position, look at the first i-1 items.

Polynomial law

-1. picked A [i -1]: A [i-1] has been used, weight j should be subtracted from A [i-1]. Then dp [i] [j] depends on dp [i-1] [jA [i-1]] .-2. did not pick A [i-1]: That is to say, A [i-1] has not been used, then dp [i] [j] depends on the previous line d [i-1] [j] -dp [i] [j] = dp [i-1] [j] || dp [i-1] [j-A [i-1]]

End

-run over dp The bottom row. Find it from the end. The last j (which allows dp [i] [j] == true) is the largest one that can be installed. :)

- time and space are: O (Mn)

---

100. [Backpack II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Backpack%20II.java) Level: Medium Tags: [Backpack DP, DP]

for i Books, each book has its own weight int [] A, each book has a value int [] V

backpack has its own size M, how many value books can I see?

### Backpack DP

-Made Backpack I, this is exactly the same, except that dp does not store the maximum weight, but the maximum value. -The idea is still different when A [i-1] is selected or A [i-1] is not selected. -Time and Space O (mn) -Rolling Array, Space O (m)

Previous DP Solution

-If w cannot be reached, it should be marked as impossible. A simple way is to mark as -1 in dp. -If there is Negative value, this is not the case, but to open an array of can [i] [w], which is the prototype of backpack I. -This seems to require more code, it does not seem to be very necessary

---

101. [Backpack V.java] (https://github.com/awangdev/LintCode/blob/master/Java/Backpack%20V.java) Level: Medium Tags: [Backpack DP, DP]

## Backpack DP

-Different from Backpack 1: This is not the check possibility (OR) or the maximum size that can fit; it is the calculation of how many kinds of exactly fill possibilities. -Dp [i] [w]: Before use I book, the possibility of filling up to w weight. -For the end, there are two cases: -1. i-1 position is not added with a bag -2. i-1 position is added with a bag -two cases can fill w Add up the situation, which is the result we want.-As usual: dp [n + 1] [w + 1] -Important: dp [0] [0] means 0 books filled with packages with weight = 0, here we must dp [0] [0] = 1, do base -Space, time: O (MN) -Rolling array, space optimization, rolling array for the following dp function . Space: O (M)

dimensionality reduction, ultimate optimization

- analysis row (i-1) of the law, found that the value of all the row (i), the related row (i-1) related element of the left and the right element is useless.
- it may be the override.
- Space: O (M), really one-dimensional!
- Time: O (MN) inserting -take out each element in the list, scan and insert it again

---

102. [Evaluate Reverse Polish Notation.java] (https://github.com/awangdev/LintCode/blob/master/Java/Evaluate%20Reverse%20Polish%20Notation.j Level: Medium Tags: [Stack]

Give an RPN string list, according to this list, calculate the result.

Stack

-stack stores numbers -each time an operator is encountered, the first 2 numbers are calculated -the calculation results are stored back in the stack, which is convenient for the next step Round use.- Time, Space O (n)

---

103. [Insertion Sort List.java](https://github.com/awangdev/LintCode/blob/master/Java/Insertion%20Sort%20List. java) Level: Medium Tags: [Linked List, Sort]

input a string of numbers, which needs to be sorted output. Each time a number is inserted, it must be placed in the correct sorted position

. Subtract this number inside

Linked List

-Time O (n ^ 2), worst case, each time the element in n numbers is placed, it just happens to be the largest -so traverse n nodes each time, then walk n times

Thinking method

-if There is already a sorted list, insert an element into it. How to do it? -Each element in while is less than curr, keep going -Once the curr is small at a certain point, add it to the current gap.

---

104. [Interleaving Positive and Negative Numbers.java](https://github.com/awangdev/LintCode/blob/master/Java/Interleaving%20Positive%20and%20Negative Level : Medium Tags: [Two Pointers]

Give a string of arrays with positive and negative numbers. Rearrange them so that the positive and negative numbers in the array are separated. The original order does not matter

Two pointer

-You need to know the positive and negative positions, so Sort O (nlogN)-Consider : the problem of more positive or negative numbers, you can see it by lifting chestnuts -then Two Pointer, swap -Time O (nlogn), space O (n)

extra space

-use extra O (n) space, split positive and negative into two lists -Then fill it back according to the index -time O (n). Space O (n) it is so useful to use Two pointer

---

105. [Largest Number.java](https://github.com/awangdev/LintCode/blob/master/Java/Largest%20Number.java) Level: Medium Tags: [Sort]

Give a string of numbers, Non-negative numbers, concatenate all numbers to form the largest number.

Because the result is very large, so use string

Sort, Comparator

-Consider more significant spot should get a larger value -if sort number, comparator will be more difficult Write: The weight of each digit is different, and single-digit and multi-digit numbers should be discussed separately. -Goal: Let the larger combination number come first, and let the smaller combination number come after. -Inferior: Combination of two cases, Use String to compare the size (you can also use integer to compare the number of combinations, but to ensure that it does not exceed Integer.MAX_VALUE, compare String here) -String.compareTo () is arranged in lexicographically, lexicographic order -use compareTo to sort the strings in reverse order , Just get the result we want.-O (nlogn), sort

---

106. [Longest Common Substring.java] (https://github.com/awangdev/LintCode/blob/master/Java/Longest%20Common%20Substring.java) Level: Medium Tags: [DP, Double Sequence DP, Sequence DP, String]

Double Sequence DP

-two strings, find the highest value: longest common string length -sequence type, and is a double sequence, find two sequences (some property of two dimensions) -dp [i] [j]: For the first i letters of A and for the first j letters of B, find the length of the longest common substring -dp = new int [m + 1] [n + 1] -dp [i] [j] = dp [i-1] [j-1] + 1; only if A.charAt (i-1) == B.charAt (j-1)-note track max, finally return -space O (n ^ 2), time (n ^ 2)

Rolling array

-space optimization, [i] is only related to [i-1], space optimization is O (n)

String

-find all A's substring, then B.contains () -track max substring length -O (n ^ 2) time

---

107. [Longest Increasing Continuous subsequence II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Longest%20Increasing%20Continuous%20sub Level: Medium Tags : [Array, Coordinate DP, DP, Memoization]

Coordinate DP

-due to access permission, not test -dp [i] [j]: longest continuous subsequence length at coordinate (i, j) -dp [i] [j] should come from (i-1, j) and (i, j-1) .-dp [0] [0] = 1 -condition: from up / left, must be increasing -return dp [m-1] [n-1]

Memoization

-O (mn) space for dp and flag. -

- O(mn) runtime because each spot will be marked once visited.
- An example of an array of the simple version of this question: Thinking about DP from a simple question will be easier. Each position is a dpValue + 1 from the other positions (up, down, left, right). If there is nothing, the init state is actually 1, which is a number. It does not increase or decrease.

108. [Maximum Subarray II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Maximum%20Subarray%20II.java) Level: Medium Tags: [Array, DP, Greedy, PreSum, Sequence DP, Subarray]

give a string of arrays, find the maximum of the sum of the numbers of the two non-interactive subarrays in the middle of the array

DP

-consider dp [i] in both directions: including i The subarray max sum. -dp [i] is characterized by: If the previous dp [i-1] + nums [i-1] is less than nums [i-1], then discard it before starting again: -dp [i] = Math.max (dp [i-1] + nums.get (i-1), nums.get (i-1)); -Disadvantage: Unable to track global max, need to record max. -Because we now need to consider from All max from the left / right, so record maxLeft [] and maxRight [] -maxLeft [i]: What is the maximum sum of the first i elements (continuously increasing); maxRight otherwise, from right to left -finally compare maxLeft [i] + maxRight [i] maximum -Space , Time O (n) -Rolling array, reduce some space, but can not reduce maxLeft / maxRight

- basic reverse linked list in the above multi-layer: found front node, the next [m ~ n] node needs to be reverse

preSum, minPreSum

-preSum is the value of each sum of [0, i]-if a minPreSum is maintained, the minimum value of [0, i] sum is recorded (because there may be negative numbers) -preSum-minPreSum is at [0, i], The maximum sum value of subarray-record this maximum subarray sum in array, left []-right [] is the same principle -enumerate the order of the elements, and finally max = Math.max (max, left [i] + right [i + 1])

---

109. [Reverse Linked List II .java](https://github.com/awangdev/LintCode/blob/master/Java/Reverse%20Linked%20List% 20II%20.java) Level: Medium Tags: [Linked List]

reverse A part of [m ~ n] in a linked list.

Reverse linked list

-Only the middle part of the reverse is required. -When Reverse: Use a dummyNode. In this problem, actually use nodeFront, then dummy.next is the entire reversed list.

Note

-Be sure to use the mth node that begins with Mark, and use it to connect the remaining node tail. Otherwise The subsequent nodes will be broken

Previous notes

-traverse to M, -save that point, -start from M, for loop, reverse [m ~ n]. Then link the three paragraphs together.

---

110. [Lowest Common Ancestor of a Binary Tree.java](https://github.com/awangdev/LintCode/blob/master/Java/Lowest%20Common%20Ancestor%20of%20a .java) Level: Medium Tags: [DFS, Tree]

Give a Binary Tree root, and two nodes p, q. Find the lowest common ancestor

DFS

-because it is a binary tree, so directly Blind search search path is not efficient, use extra space and waste time -Use DFS to find the common ancestor of each
-Need the assumption: 1. unique nodes across tree; 2. must have a solution node.-When root == null or pq is found at the bottom of findLCA (root == A || root == B), then the root is returned. -Three cases: -1.
A and B are found, then the node at this level is one of the ancestors: In fact, the one that is recursively returned is the bottom LCA parent.
-2. If A or B is found, then there is no public parent, and the return is not null.
-3. AB is null, then you find the wrong one, return null -Worst case, visit all nodes to find pq at last level, last two leaves: time / space O (n)

---

111. [Lowest Common Ancestor of a Binary Search Tree.java](https://github.com/awangdev/LintCode/blob/master/Java/Lowest%20Common%20Ancestor%20of%20a Level: Medium Tags: [BST, DFS, Tree]

for binary search tree root, q node, p node. Find the lowest common ancestor of pq

Find path with BST

-Use the nature of BST to directly search the target node And make two lists that are not necessarily equal in length -and then easily find LCA -O (n) space, O (logn) time

DFS

-Brutly find common ancestor of p and q, then recursively drive left / right -very clever, But it is also more limited; it is difficult to recursive if you change the conditions slightly.-Several cases: -1. one of p, q in leaf, then the root at this time is actually the lowest common ancestor -2. If p, q are on the left and right sides of root, this is the fork, then root is the lowest common ancestor -3. If p, q are on the same side of the root (left, right), then continue with dfs -O (1) extra space, O (logn) time

---

112. [Remove Duplicates from Sorted Array II. java] (https://github.com/awangdev/LintCode/blob/master/Java/Remove%20Duplicates%20from%20Sorted%2 Level: Medium Tags: [Array, Two Pointers]

to a Sorted array, remove duplicates: that is, paste the non-repeating in order, the extra position at the end of the array does not matter. The

number of elements that can be repeated is not more than 2. The length of the return unique item.

Two Pointers

-sorted array, repeating elements are all together -Almost the same as `Remove Duplicates from Sorted Array`, except that unique index can now validate 2 digits -The rest is exactly the same, use index to track unique item; skip if duplicated for more than 2 times -O (n) time, O (1) space -You can really write a while loop with 2 pointers here, but it is not necessary, just Simply walking a for loop is actually enough.

---

113. [Remove Duplicates from Sorted List II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Remove%20Duplicates% 20from% 20Sorted% 20List% 20II.java) Level: Medium Tags: [Linked List]

Remove duplicate elements from the Linked list: As long as they are repeated, all are deleted; One element that repeatedly appears is not left.

## Linked List

-sorted list, duplicate elements are all together -using dummyHead: If you want to remove all duplicate elements, you must have a dummyHead as an outsider to match at the beginning -as long as you find a node.val == node.next.val, then Make a note of this duplicated val, move forward,
-thought: and remove all the duplicated elements-use a second inner while loop, process all the duplicated elements, and then move forward -Advantages: outter while loop does not need to consider too many cases, and the main business logic is solved in the inner loop.

Note the use of DummyHead

-When we have DummyHead as an external line head of the Linked List, it can actually Every time I encounter duplicate, I assign the later elements to dummyHead.next forcibly -I also tried one method below: But there are too many edge cases to consider: keep moving the node, know not to repeat, assign prev.next = node. -This method is relatively straightforward, but it needs to consider many edge cases, and it does not make good use of the dummy head. Be careful to avoid it.

Previous Note

-Cut off the roots. -Multiple nodes, check node.next? = Node.next.next

---

114. [QuickSort.java] (https://github.com/awangdev/LintCode/blob/master/Java/QuickSort.java) Level: Medium Tags: [Quick Sort, Sort]

implement quick sort. -then split Two halves

Quick Sort

-First partition. Return the position of the middle point of a partition: At this time, all less than nums [partitionIndex] should be to the left of partitionIndex -Quick sort before and after, respectively, recursively -Note: In partition, when comparing nums [start] < pivot, nums [end]> pivot, if written as <= will stack overflow. -Time O (nlogn), Space: O (1)

---

115. [MergeSort.java] (https: // github. com / awangdev / LintCode / blob / master / Java / MergeSort.java) Level: Medium Tags: [Merge Sort, Sort]

Merge Sort

-Divide and conquer, recursively -segment from the middle first, merge sort On the left (dfs), merge sort on the right -finally merge it up -since the merge is done as int [], there is no way to use O (n) space -Time O (nlogn), Space O (n)

---

116. [Binary Tree Level Order Traversal.java] (https://github.com/awangdev/LintCode/blob/master/Java/Binary%20Tree%20Level%20Order%20Traver Level: Medium Tags : [BFS, DFS, Tree]

BFS

Such as the title.

BFS

-the most common, Non-recursive: BFS, queue, use queue.size () to end for loop: newline.
-Or use two queues. When the regular queue is empty, paste the backup queue

DFS

-Each level should have an ArrayList. Then use an int level to see if each level has a corresponding ArrayList .
-If not, add a layer.
-Each time after that, the number is added to the corresponding level through DFS.

---

117. [Binary Tree Level Order Traversal II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Binary%20Tree%20Level%20Order%20Traver Level: Medium Tags: [BFS, Tree]

As the title, but the output must be

-Same as Binary Tree Level Order Traversal, except that there are always 0 bits in the result.

DFS

-Append each list according to level -add (0, ...) in rst is added at the beginning of the list each time

118. [Binary Tree Longest Consecutive Sequence II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Binary%20Tree%20Longest%20Consecutive% Level : Medium Tags: [DFS, Divide and Conquer, Double Recursive, Tree]

find the longest consecutive sequence in the binary tree. Sequence can be incremented and decremented, Sequence sequence can be traced back to parent.

DFS, Divide and Conquer

-Similar to Binary Tree Longest Consecutive Sequence I -It can only be incremented and decremented, as well as the direction of the parent. -For any node, it is possible: -1. Link itself with two children to become a sequence -2. Left child, right child Each is a consecutive sequence, but not connected to root -The main function divides into these three parts at the beginning, and then dfs -dfs take diff == 1, diff == -1, to do an incrementally decreasing proofreading.- -Handle the root node in two recursive ways dfs rules: if node == null, leaf depth = 0 -2. if not consecutive, reset the depth = 0 (same for both left child, and right child) -3. compare the leftDepth && rightDepth to find the maximum -4. diff is the same in the same dfs loop to maintain consistant increase / decrease

Note

-the result of dfs is likely to be 0, if there is no result, then the caller depth of the previous layer = dfs () + 1 = 1 -then return to root, the result of dfs is likely to be 1. -May ask: Then the partial sequence (not connected to root) in the tree is ignored? -Here `longestConsecutive (root.left)` is very important -this step specifically ignores the root, and then walks Next level: Because it is recursive, it will continue to divde && conquer -Finally, children at any level will be taken care of.

Double Recursive functions

-Recursive using dfs (), basically build child + parent -Recursive using main function, but with value of child node: skipping root

---

119. [Combinations.java] (https://github.com/awangdev/ LintCode / blob / master / Java / Combinations.java) Level: Medium Tags: [Backtracking, Combination, DFS]

Given two integers n and k, return all possible combinations of k numbers out of 1 ... n.

## DFS, Backtracking

-for loop, recursive (dfs) .-Use once for each item, next level dfs (index + 1) -Combination DFS. Draw a picture and think. Pick a number from 1 ~ n each time i -Because the next layer can't go back to [0 ~ i] to choose, so the next layer of recursive should be chosen from i + 1.

---

120. [Combination Sum IV.java] (https://github.com/awangdev/LintCode/blob/master/Java/Combination%20Sum%20IV.java) Level: Medium Tags: [Array, Backpack DP , DP]

-To find overall dp [i], make a for loop: dp [i] = sum {dp [i-num]}, where for (num: nums)

gives a list of dates dates (no duplicates), and a target.

Find all unique combinations int [], requiring the sum of each combination = target.

Note: The same candidate integer can be used any number of times.

Backpack DP

-counting problem, you can think of DP. In fact, it is Backpack VI. -Find candidate from x numbers (you can use the same number repeatedly) to sum up to target. Find: # of ways to form the sequence. -Backpack VI: give an array of nums, all positive numbers, none Repeat the numbers; find: # of the method of spelling out m -dp [i]: # of ways to build up to target i -consider last step: if the previous step was candidate A, then it should be added to dp [i] : -dp [i] + = dp [i-A] -Time: O (mn). m = size of nums, n = target -If we optimize dp for loop, requires Sort nums. O (mlogm). will efficient If m is constant or relatively small. Overall: O (n)

DFS, backtracking

-although the way of thinking is right, but times out -When numbers can be reused, for example, 1 is used to spell 999. Here, 1 can go to 999 dfs level, not efficient

---

121. [Binary Tree Right Side View.java] (https: // github .com / awangdev / LintCode / blob / master / Java / Binary% 20Tree% 20Right% 20Side% 20View.java) Level: Medium Tags: [BFS, DFS, Tree]

Give a binary tree, see it from the right, return all visible nodes

BFS

-rightmost: the end of each line of level traversal.
-BFS, queue to store the content of each line, save end node into list

DFS

-Use Map <Level, Integer> Store the results of each level
-in the dfs function, if the input depth does not exist, add to map. -dfs function first: dfs (node.right), then dfs (node.left)-because always depth search on right side, the map will be populated by right branch; then leftChild.right

---

122. [Binary Tree Maximum Path Sum II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Binary%20Tree%20Maximum%20Path%20Su Level : Medium Tags: [DFS, Tree]

find max path sum from root. Condition: There is at least one node.

DFS

-Much simpler than Binary Tree Maximum Path Sum I. Because conditions give more: at least 1 node + have to start from root -root must be used -3 cases: curr node, curr + left, curr + right -because it must include root, say starting with dfs (root, sum = 0) , each First add root, sum + = root.val

---

123. [Rotate List.java] (https://github.com/awangdev/LintCode/blob/master/Java/Rotate%20List.java) * * Level: Medium Tags: [Linked List, Two Pointers]

give a single linked list, move k steps to the right. K non-negative.

Linked List basics

-Remember to use dummy.next to store the head. -Special: Here k may be greater than the total length of the list. Write the number of steps the linked node moves, then k = k% n-.

- found newTail, newHead, then use dummy, Commutator

---

124 [Binary Tree Longest the Consecutive Sequence.java] (https://github.com/awangdev/LintCode/blob/master/Java/. binary% 20Tree% 20Longest% 20Consecutive% 20Sequence.java) Level: Medium Tags: [DFS, Divide and Conquer, Tree]

to find the binary tree in the Sequence longest consecutive.

the DFS

- Divide and Conquer the DFS.
- Look left / right separately
- If the left meets the continuous increasing rule, dfs (depth + 1), if the rule is not satisfied, dfs (depth = 1)-the same is true on the right -compare the result with max, return

-the ---

125. [Number of Connected Components in an Undirected Graph.java](https://github.com/awangdev/LintCode/blob/master/Java/Number%20of%20Connected%20Components Level: Medium Tags: [BFS, DFS, Graph, Union Find]

Give a number n for n nodes, marked from 1 ~ n, and a string of undirected edge int [] [].

Count how many independent components there are in this graph.

Union Find

-almost the same as Graph Valid Tree -build simple parent [] union find -every edge is union .-### Note When union, only union if rootA! = RootB

DFS

-build graph as adjacent list: Map <Integer, List > -dfs for all nodes of the graph, and mark visited node -count every dfs trip and that will be the total unions

---

126. [Next Closest Time.java](https://github.com/awangdev/LintCode/blob/master/Java/Next%20Closest%20Time.java) Level: Medium Tags: [Basic Implementation, Enumeration , String]

Give a time string "12:09", use the 4 integers in it to combine other time strings, and find the smallest next time.

If the combined time string is before input time, the default is + 24 hours.

## String

-enumerate all candidates and filter to keep the correct ones -String.compareTo (string)-> gives lexicographical comparision

---

Two Pointer

127. [Partition Array.java](https://github.com/awangdev/LintCode/blob/master/Java/Partition%20Array.java) Level: Medium Tags: [Array, Quick Sort, Sort, Two Pointers]

gives a string of numbers, and int k. According to the value of k partition array, find the first i, nums [i]> =

k.-The basis of Quick sort. -Partition Array divides the array into two halves according to pivot. -Indent from both sides of the array. while loop to iteration. Very straightforward implementation. -Note that low / high, or start / end, do not cross the boundary. -O (n) -Note: Here the second inner while `while (low <= high && nums [high]> = pivot) {..}` uses `Nums [high]> = pivot` -the reason is that the problem is to find the first nums [i]> = k, that is, even nums [i] == k should be swapped to the front -this is the same as quick The sort original title is slightly different.

---

128. [Word Ladder.java](https://github.com/awangdev/LintCode/blob/master/Java/Word%20Ladder.java) Level : Medium Tags: [BFS]

For a string of string [], you need to find the shortest distance to change from wordA-> wordB. (See the original title for the details of the restrictions)

BFS

-Usually, give a graph (this question can Think of beginWord as the starting node of a graph), find the shortest path using BFS -based on the start string, each letter of the string traverses all 26 letters -Visited removed from wordList -time: word length m, there can be n candidates => O (mn) -but always exceed time limit on LeetCode. However, it passes LintCode: -The reason is that LeetCode gives a list , list.contains (), list.remove () are O (logn) time !!! -convert to set first.

Trie

-timeout, overkill

---

129. [Unique Word Abbreviation.java] (https://github.com/awangdev/LintCode/blob/master/Java/Unique%20Word%20Abbreviation.java) Level: Medium Tags: [Design, Hash Table]

give a string [] dict, and a word.

Each word can be abbreviated to a fixed abbreviation <first letter> <number> <last letter> (see the original question in detail) to

check whether the input word meets unique

HashMap <string, Set>

-Simple calculation abbreviatioin -Check if abbr exists; if so, is it the input word itself.

---

130. [Unique Binary Search Tree II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Unique% 20Binary% 20Search% 20Tree% 20II.java) Level: Medium Tags: [BST, DP, Divide and Conquer, Tree]

Give a number n, and find all unique BSTs with (1 ... n) as the node .

BST

- according to the rules of BST, Divide and Conquer
- take a value, then two and a half (start, value - 1), (value + 1, end) DFS respectively
- and both sides of the match results Cross

## DP? Memoization?

---

131. [Ugly Number.java] (https://github.com/awangdev/LintCode/blob/master/Java/Ugly%20Number.java) Level: Medium Tags : [Math]

LeetCode: Determine whether the number is ugly number. (Definition: factor only have 2, 3, 5) -See if it is divisible.

Math

-See if the final result of the division is == 1

LintCode: Find the kth ugly number, which should be the same as Ugly Number II

-Method 1: PriorityQueue sorting. Use ArrayList to check if the new ugly Number has appeared. -Method 1-1: (Unexplained, not desirable) Sort by PriorityQueue. Magical 3, 5, and 7 positions: According to the starting point of the answer, determine the rules for 3, 5, and 7 to appear. But the title is not specifically stated. -Method 2: DP. Not Done yet.

132. [Top K Frequent Words.java](https://github.com/awangdev/LintCode/blob/master/Java/Top%20K%20Frequent% 20Words.java) Level: Medium Tags: [Hash Table, Heap, MaxHeap, MinHeap, PriorityQueue, Trie]

time: O (nlogk) space: O (n)

gives a string of Strings. Find top k frequent words.

## PriorityQueue-Min Heap

-O (n) space of map, O (nlogk) to build queue. -limit minHeap queue size to k: add to queue if found suitable item; always reduce queue if size> k

PriorityQueue-Max Heap

-Use HashMap to store frequency, ArrayList to store lists of words -Create a Node class, and then use PriorityQueue.
-PriorityQueue uses String.compareTo (another String). -time: PQ uses O (nlogn), overall O (nlogn) -slower, because the maxHeap needs to add all candidates

Trie && MinHeap 屌 炸 天-Can

do it -http: //www.geeksforgeeks.org / find-the-k-most-frequent-words-from-a-file /

HashMap + collections.sort ()

-Use HashMap to store frequencies and ArrayList to store lists of words. Finally return k from the tail forward.
-Note that the cost of Collection.sort () is O (nLogk) when sorting -not efficient

---

133. Segment Tree Build.java### Level: Medium Tags: [Binary Tree, Divide and Conquer, Lint, Segment Tree]

给一个区间[startIndex, endIndex], 建造segment tree structure, return root node.

Segment Tree definition

- Recursively build the binary tree
- 左孩子： (A.left, (A.left+A.rigth)/2)
- 右孩子： ((A.left+A.rigth)/2＋1， A.right)

---

134. Segment Tree Build II.java### Level: Medium Tags: [Binary Tree, Divide and Conquer, Lint, Segment Tree]

给一个array, 建造segment tree structure,

每个treeNode 里面存这个range里的 max value, return root node.

Segemnt Tree

-Array is given. Pay attention to find the max in the interval, assign to the interval. The rest is the same as the ordinary segment tree build
-Note that the segment tree is ranked according to the array index range: according to index in [0, array.length-1] cut the interval, break to the end -finally start == end to the end -trackmax is required for this problem, then in leaf node assign max = A [start] or A [end]-go up, parent layer Max: It is to compare the left and right children. In fact, the max of the sub-tree is compared in the two sub-trees.

-Devide and Conquer -Divide first, find left / right, compare max, then create current node, then append to the current node. -It's actually depth-first, built from the bottom up.

---

135. [Segment Tree Query.java](https://github.com/awangdev/LintCode/blob/master/Java/Segment%20Tree%20Query.java) Level:

Medium Tags: [Binary Tree, DFS, Divide and Conquer, Lint, Segment Tree]

gave the Segment Tree, the node has Max value, find the max in [start, end]

Segment Tree, Divide and Conquer

-Compare [start, end] with mid of (root.start, root.end): -Simple 2 cases: [start, end] are all to the left of mid, or [start, end] is all to the right of mid -a slightly more complicated 3rd case: [start, end] contains mid, then break into 2 queries-[start, node.left.end], [node.right.start, end]

---

136. [Segment Tree Modify.java] (https://github.com/awangdev/LintCode/blob/master/Java/Segment%20Tree%20Modify.java) Level: Medium Tags: [Binary Tree, DFS, Divide and Conquer, Lint, Segment Tree]

Give max to a segmentTree, node. Write a modify function: modify (node, index, value).

Segment Tree, Divide and Conquer

-Recursively in In the segment tree, look for index, update it with value.
-For each iteration, it is possible (either left-handed or right-handed) that max has changed. So each time left.max and right.max compare end of the round, the top of the head, including the top of the head, are all max -Use HashMap to understand the rules of the problem, because repeated calculations can be used Cover, so it is an optimization problem. There is not much suspense and meaning.

---

137. [Segment Tree Query II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Segment%20Tree%20Query%20II.java) Level: Medium Tags: [Binary Tree, DFS, Divide and Conquer, Lint, Segment Tree]

Segment Tree

-Similar to Segment Tree Query I and other Segment Trees: This SegmentTreeNode returns count of elements in range -This topic has a valid input source: input start, end may exceed root [start, end].
-Then the first step is to clear it first: 1. Return 0 if it is not in range at all. 2. Regularize to the root range when range overlaps.

---

138. [ColorGrid.java] (https: // github .com / awangdev / LintCode / blob / master / Java / ColorGrid.java) Level: Medium Tags: [Design, Hash Table]

basic implementation

-Eliminate the coincidence point:
-If the current col is actually reduced, Been to the junction of all the added rows. . .
-Analyze again, that is, take a single point each time you encounter row, sumRow + = xxx.
-Then when the current process is col, sum + = colValue * N-sumRow. Is equivalent to subtracting the point that crosses all rows (rows that have been processed). very convenient. -The last read in is O (P), and the process is also O (P).

---

139. [Container With Most Water.java] (https://github.com/awangdev/LintCode/blob/master/Java/Container%20With%20Most%20Water.java)### Level: Medium Tags: [Array, Two Pointers]

Two Pointers

- Wooden barrel theory. The highest level of water depends on the lowest wall.
- Run left and right to the middle.

- Another: If one wall is already smaller than the other, move it and replace the low wall (maybe the next higher or lower)
- but you must not replace the current high wall because the low wall The upper limit, if the high wall moves, causing the distance between the two walls to decrease, it is doomed to less water. (What's the matter, don't be indifferent)

---

140. [Copy List with Random Pointer.java] (https://github.com/awangdev/LintCode/blob/master/Java/Copy%20List%20with%20Random%20Pointe Level: Medium Tags : [Hash Table, Linked List]

time: O (n) space: O (1)

deep copy linked list.

Linked list has random pointer to other nodes. #### HashMap, Linked List -Basic Implementation of copy linked list : -use node and dummy to hold new list, traverse head.next .... null.
-Map is used here: 1. avoid creating same node; 2. return the item if existing -map keys are all old objects , The new keys are all newly created objects -check whether there is a head in the map at each step. No? Plus -check whether there is a head.random in the map at each step. No? Plus

---

141. [Encode and Decode Strings.java] (https://github.com/awangdev/LintCode/blob/master/Java/Encode%20and%20Decode%20Strings.java) Level: Medium Tags: [String]

如 题.

String

-'word.length () # word' This encoding can avoid encountering #-Based on our own rules, there is no need to check error input too much in decode, assume all The input is
normal.-Decoding is to find "#", and then use the number before "#" to intercept the subsequent string.

---

142. [Fast Power.java] (https://github.com/awangdev /LintCode/blob/master/Java/Fast%20Power.java)### Level: Medium Tags: [DFS, Divide and Conquer]

As the title: Calculate the a ^ n% b where a, b and n are all 32bit integers.

### Divide and Conquer

-a ^ n can be disassembled into (a * a * a * a .... * a), which is an opportunity form, and% can mod each item. So take apart to take mod.-Here we use a dichotomous method, recursively dice until n / 2 is 0 or 1, and then treat them separately.

DFS

-Note 1: After the dichotom is conquered, the product may be greater than Integer.MAX_VALUE, so use a long.-Note 2: To deal with the case of n% 2 == 1, a part of a is automatically saved at two points, and it needs to be multiplied.

---

143. [Find the Connected Component in the Undirected Graph.java] (https://github.com/awangdev/LintCode/blob/master/Java/Find%20the%20Connected%20Component%2 20Undirected% 20Graph.java) Level: Medium Tags: [BFS, DFS]

Give an undirected graph, return all the components. (This question is not found)

BFS

-BFS traversal, All neighbors are added. -Be sure to mark the visited nodes. Because curr nodes will also be neighbors of others, they will loop indefinitely.
-Definition of Component: All nodes in Component must be connected in series via path (anyway here is undirected, as long as the link is fine)
-This question: In fact, the component is already given in the input, all can be visited in one go All of them are added to the queue, and they are in a component.
-And we don't need to judge whether they are Component

-DFS should also be able to visit all nodes, mark visited.

---

144. HashWithCustomizedClass(LinkedList).java### Level: Medium Tags: [Hash Table]

练习HashMap with customized class. functions: get(), put(), getRandom()

Hash Table

- store map as array: Entry<K,V>[] table;
- store entry as linked list: public Entry(K key, V value, Entry<K,V> next)
- compute hashKey: Math.abs(key.hashCode()) % this.capacity
- Handle collision:
- 1. Check if duplicate (matching key), if so, replace and return
- 2. Check through the linked list, find find duplicate (matching key), replace and return.
- 3 . If no duplicate, add the entry to the tail -Find item: compute hashKey-> find linked list-> iterate over list to find a matching key.

---

145. [Interval Minimum Number.java] (https: / /github.com/awangdev/LintCode/blob/master/Java/Interval%20Minimum%20Number.java)### Level: Medium Tags: [Binary Search, Divide and Conquer, Lint, Segment Tree]

Give a string of numbers int [] , And then a query Interval [], each interval is [start, end], find the minimum value in the query interval.

Segment Tree

-SegtmentTree, methods: Build, Query. This problem is to store min in SegmentTreeNode. -Similar existence: max, sum, min

---

146. [Interval Sum.java] (https://github.com/awangdev/LintCode/blob/master/Java/Interval%20Sum.java) * * Level: Medium Tags: [Binary Search, Lint, Segment Tree]

Give a string of numbers int [], then a query Interval [], each interval is [start, end], find the sum in the query range.

Segment Tree + Binary Search

-In fact, the segment tree adds a sum to each node.-Remember Segment Tree methods: Build, Query -Note: The sum stored in SegmentTreeNode is sum . Other topics may be min, max, count ... or something else.

---

147. [Kth Smallest Element in a BST.java] (https://github.com/awangdev/LintCode/blob/master /Java/Kth%20Smallest%20Element%20in%20a%20BST.java)### Level: Medium Tags: [BST, DFS, Stack, Tree]

Iterative + stack: inorder traversal

-I can think of Inorder-binary -search-tree Traversal -Iterative Slightly harder to think about: first add the leftmost add, pop () stack, plus the right (if it exists); the next reincarnation, if there is a left child, add another meal.

Recursive + DFS

-Then optimize it a bit to ensure that rst.size () == k, you can return -check leaf => dfs left => add root => dfs right

---

148. [Majority Element II.java](https://github.com/awangdev/LintCode/blob/master/Java/Majority%20Element%20II.java) Level: Medium Tags: [Array]

## Sort + count

-O (nlogN)

Two counters

-O (n), count and track valueA, valueB -count overall apperance at the end for the two items -save to result -Note: According to the if statement In order, valA && countA has priority over valB && countB

---

149. [Partition List.java](https://github.com/awangdev/LintCode/blob/master/Java/Partition%20List.java) Level: Medium Tags: [Linked List, Two Pointers]

Linked List

-linked list cannot be traversed from both sides like partitioin array -add less than value in the first half and add> = value in the second half -The method is very common: build two lists, midTail pointer, post pointer -Put the numbers that meet the conditions (<x,> = x) into the two lists separately -Remember to use dummyNode track head. -Finally midTail.next = post Link up.

---

150. [Peeking Iterator.java](https://github.com/awangdev/LintCode/blob/master/Java/Peeking%20Iterator.java) Level: Medium Tags: [Design]

## Use concept pre cache

-find a cache to store the value of next (), that is: the value of next value is stored in the cache in advance -so when peek (), you can directly return the cache without using itt.next ( ) -Then every time next () is really taken, take an itt.next () to maintain this cache

Previous notes

-understand the wrong topic again. Peek () is overhead, but not necessarily the largest It's worth it. -Always think of PEEK as the maximum value, then use 2 STACK to make the maximum value cache, and practice a good dual stack, but it is a mistake.

---

151. [Rehashing.java] (https://github.com/awangdev/LintCode/blob/master/Java/Rehashing.java) Level: Medium Tags: [Hash Table]

For a Hash Table, use Linked list does. The problem is: if the capacity is too small, if there are too many collisions, you need double capacity and then rehash.

Hash Table

-Understand the meaning of hashCode () function: When you get the hashKey, use hashKey% capacity Let's do hash code -hashcode is the index in the hash map -understand the way of collision handling, and how to double capacity and rehashing -are basic operations, concept implementation

---

152. [Reorder List.java] (https: / /github.com/awangdev/LintCode/blob/master/Java/Reorder%20List.java)### Level: Medium Tags: [Linked List]

for a Linked list, reorder: proceed from the head / tail direction to the middle, re-order like: one node at a time,

Linked List

-reverse list, find mid of list, merge two list -find mid first, then reverse mid.next, and finally merge two paragraphs. -Note that after using mid.next, be sure to mid.next = null, otherwise merge There will be a problem

---

153. [Restore IP Addresses.java] (https://github.com/awangdev/LintCode/blob/master/Java/Restore%20IP%20Addresses.java) Level: Medium Tags: [Backtracking , DFS, String]

Give a string of numbers, check if it is a valid IP, and if it is reasonable, give all valid IP combinations.

Backtracking

-End point of recursion: list.zie () == 3, solve the last paragraph -Recursively on an index, pass s.toCharArray ()-validate string should pay attention to leading ' 0' -Note: When recursing , you can use a start / level / index to run the route -but try not to change the input source, it will change It is very confusing. -Note: The code is a bit messy, because the validity of the IP must be considered -the 'remainValid' is actually a judgment optimization for the remain substring. If it is not true, it will not be dfs

---

154. [Reverse Words in a String.java] (https://github.com/awangdev/LintCode/blob/master/Java/Reverse%20Words%20in%20a%20String.java Level: Medium Tags : [String]

In-place reverse

Break by space, then flip

-No space at the end -trim () output -If Input is "", there will be nothing after split -Another topic Reverse Words in String (char []) can be in -place, provided that there are no leading and trailing spaces in char [].- Time, Space: O (n)

Other methods

-flip entire string, then flip each individual string (there is a lot of code, this question cannot be made)

-
  -

155. [Reverse Words in a String II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Reverse%20Words%20in%20a%20String%20 * * Level: Medium Tags: [String]

-reverse is used twice. Global reverse. Partial: encounter a space reverse -note the ending index: `i == str.length-1` , even if there is no '' at the end point, give the last word to the reverse

156. [Search a 2D Matrix.java] (https : //github.com/awangdev/LintCode/blob/master/Java/Search%20a%202D%20Matrix.java) Level: Medium Tags: [Array, Binary Search]

for 2D matrix, sorted by each row, each row The first position is greater than the end of the previous line. Goal: find target from matrix

2D matrix to 1D array

-line by line from small to large, sorted, continuous, can be regarded as 1D sorted array -Binary Search

---

-
  -
    157. [Search a 2D Matrix II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Search%20a%202D%20Matrix%20II.java) Level: Medium Tags: [Binary Search, Divide and Conquer]

For matrix, each row is sorted, each column is sorted from top to bottom, find if the target exists

### Binary Search

-According to the given properties, in fact, click the extreme point: x = bottom row, y = the smallest left position in the current row. (X, y) in the lower left corner -in this case, it can only run in one direction: if less than target, y ++; If it is greater than target, then only x--‑For each operation, delete a row, or a column, no need to look back -while (x> = 0 && y <col) {} to ensure that it will not run away -the same way: you can from the upper right corner (0, col - 1), the code follows a slightly altered

Divide and Conquer?

- TODO

---

158 [Search for A Range.java] (https://github.com/awangdev/. LintCode / blob / master / Java / Search% 20for% 20a% 20Range.java) Level: Medium Tags: [Array, Binary Search]

For sorted array, there are duplicate numbers, find the range where the target coincides.

# Binary Search

-2 while loop -find first / last occurance -TODO: Can the code be simplified?

---

159. [Search Range in Binary Search Tree .java] (https://github.com/awangdev /LintCode/blob/master/Java/Search%20Range%20in%20Binary%20Search%20Tree%20.java)### Level: Medium Tags: [BST, Binary Tree]

Give a BST, integer range (k1, k2), find all integers in the range.

BST

-equals dfs to traverse all k1 <= x <= k2 x node. -dfs left, process root, then dfs right -Here, we have covered all the left / right / match cases, and then limited the borders of k1 and k2, and traversed all in the middle.

---

160. [Sort List.java] (https://github.com/awangdev/LintCode/blob/master/Java/Sort%20List.java) Level: Medium Tags: [Divide and Conquer, Linked List, Merge Sort, Sort]

Merge sort

-1. find middle. Speed pointer -2. Sort: cut the two halves, sort the first half first, if sort first mid.next ~ end, sort, middle Point mid.next == null, then sort the first half -3. Merge: Assume that given list A, B is already sorted, and then mix according to size. -Recursively call sortList () on partial list.

Quick sort

-If you want to do it, please read the handout: http://www.jiuzhang.com/solutions/sort-list/-But quick sort is not recommended for list . -Sort list, merge sort may be more feasible and reasonable. The reason analysis is as follows: http://www.geeksforgeeks.org/why-quick-sort-preferred-for-arrays-and-merge-sort-for-linked-lists/

---

161. [Summary Ranges.java] (https://github.com/awangdev/LintCode/blob/master/Java/Summary%20Ranges.java) Level: Medium Tags: [Array]

Give a list of sorted lists, with missing numbers in the middle, return all number range string (example see title)

Basic implementation

-use a list as the buffer to store candidates -when: 1. end of nums; 2. not continuous integer => convert list to result

---

162. [Topological Sorting.java] (https://github.com/awangdev/LintCode/blob/master/Java/Topological%20Sorting.java) Level: Medium Tags: [BFS, DFS, Topological Sort]

Topological Sort BFS

-indegree tracking: Track all neighbors / childrens. Store all children in inDegree <label, indegree count>-Process with a queue: First add all the roots (indegree == 0), there may be multiple roots. And all added to the queue. -BFS with Queue: -Only when map.get (label) == 0, add into queue && rst. (Indegree is cut, it is root) -inDegree count down indegree here, make sure that the nodes that appear later, must Finally process.

Basics about graph

-several graph conditions:
-1. There may be multiple roots -2. directed node, you can direct backwards.

TODO: -build `Map <DirectedGraphNode, Integer> inDegree = new HashMap <> ();` and include the root itself -that is more traditional indegree building

---

163. [Spiral Matrix.java] (https://github.com/ awangdev / LintCode / blob / master / Java / Spiral% 20Matrix.java) Level: Medium Tags: [Array, Enumeration]

From (0,0) coordinates, walk through the spiral matrix, and store the result in the list.

# DX, DY

-Basic implementation, array, enumeration -Write the direction of position forward: RIGHT-> DOWN-> LEFT-> UP -Use a direction status to determine the direction -Write a compute direction function to change the direction `(direction + 1) % 4` - `boolean [] [] visited` where the track went

---

164. [Construct Binary Tree from Inorder and Postorder Traversal.java] (https://github.com/awangdev/LintCode/blob/master/Java/Construct%20Binary%20Tree%20from%20Ino .java) Level: Medium Tags: [Array, DFS, Divide and Conquer, Tree]

DFS, Divide and Conquer

-Write an example of Inorder and Postorder. Use them to divide left / right subtrees to solve problems. -The end of the Postorder array is the root of the current layer.-When you find this root in the Inorder array, you just split the left and right sides into a left / right tree. -This question is tricky using a helper for recursive. Pay special attention to handling changes in the index, precisely considering the beginning and end -runtime: O (n), visit && build all nodes

Improvement

-`findMid (arr)` can be replaced with a map <value, index>, no need execute O (n) search at runtime

---

165. [Generate Parentheses.java](https://github.com/awangdev/LintCode/blob/master/Java/Generate%20Parentheses.java) Level : Medium Tags: [Backtracking, DFS, Sequence DFS, String]

DFS

-start with empty string, need to go top-> bottom -take or not take `(, )`

- rule: open parentheses >= close parentheses
- Note: 在DFS时 pass a reference (StringBuffer) and maintain, instead of passing object (String) and re-create every time
- time: O(2^n), pick/not pick, the decision repat for all nodes at every level
- T(n) = 2 * T(n - 1) + O(1)

bottom->up DFS

- figure out n=1, n=2 => build n=3, and n=4
- dfs(n-1) return a list of candidates
- add a pair of `()` to the candidates: either in front, at end, or contain the candidates

---

166. [Strobogrammatic Number II.java]()### Level: Medium Tags: [DFS, Enumeration, Math, Sequence DFS]

TODO:

1. use list, iterative? Keep candidates and populating
2. clean up the dfs code, a bit messy
3. edge case of "0001000" is invalid, right?

DFS

-A bit like BFS solution: find inner list , and then combine with outter left / right sides. -find all solutions, DFS will be easier to write than iterative / BFS -when n = 1, there can be list of candidates at bottom of the tree, so bottom-> up is better -bottom-> up, dfs till leaf level, and return candidates.

- each level, pair with all the candidates
- In fact, it is peeling, one layer at a time, is a central-depth-first. When drilling to the end, return n = 1, or n = 2. And then start backtracking.
- Difficult case without handle first. After that, come to an overall scan.
- Every level have 5 choices of digital pairs to add on sides. Need to do for n-2 times.
- Time complexity: O (5 ^ n)

-
    -

167. [Flip Game II.java](https://github.com/awangdev/LintCode/blob/master/Java/Flip%20Game%20II.java) Level: Medium Tags: [Backtracking, DFS , DP]

String contains only +,-two symbols. Two people take turns turning consecutive consecutive ++, into `--`.

If one of them can no longer be turned over, the other one wins. If the string is out, can you win first?

Backtracking

-curr player Every step you take, a new situation is generated, dfs on this -Wait until the dfs is over, whether it is successful or not, you must backtracking -curr level: put " ++ "changed to"-"; backtrack, change back to '-' -Replaced by boolean [] is much faster than string / stringBuilder, because there is no need to regenerate the string.- ++ can take (n-1) positions: -T (N) = (N-2) * T (N- 2) = (N-4) * (N-2) * T (N-4) ... = O (N!)

iterate based on "++"

-make a replica of String s: string or stringBuilder

- After each dfs, then replace the characters "+" => "-"
- the purpose is just the index that Mark has used
- The real dfs is still expanded on the original input string s
- the substring is regenerated every time , not very efficient

Game theory

-To ensure that p1 can win, you must keep all the moves of p2 not win -In other words, knowing all the situations of chess, as long as p2 has a way to lose, p1 will definitely be able to go the opposite way and win. -At the same time, as long as p1 is in a move that can move, it is enough to have a move to win . -p1: player1, p2: player2

DP of O (N ^ 2) -Requires

Game Theory skills, Nim game. https://www.jiuzhang.com/qa/941/-http : // www .1point3acres.com / bbs / thread-137953-1-1.html -TODO: https://leetcode.com/problems/flip-game-ii/discuss/73954/Theory-matters-from-Backtracking(128ms)-to-DP-(0ms)

---

168. [Palindrome Partitioning.java](https://github.com/awangdev/LintCode/blob/master/Java/Palindrome%20Partitioning.java) Level: Medium Tags: [Backtracking, DFS]

giving string s, partition (segmentation), make sure that every All partitions are palindrome.

Find all combinations of partition palindrome. List < list <string >>

DFS

-can top-> bottom: traverse str, validate substring (start, i); if valid, add as candidate, and dfs; backtrack by remove candidate. -can also bottom-> up: traverse str, validate substring (start, i); if valid, dfs (remaining str), return list of suffix; cross match with curr candidate.

DFS Top-> Bottom

-When traversing str, consider From each curr spot to the end of str, how many kinds of palindorme can there be?-Then start counting from the curr spot as a character and start backtracking. selection is not palindrome, then move -If the selection is indeed palindrome, add to path, DFS Go to the next level and wait until the end of the traversal, which results in a string split into palindrome. -At the end of each DFS, delete the selected palindrome added in this layer, backtracking.

Optimization

-You can calculate isPalindrome (S) for each dfs level, but you can calculate boolean [] [] isPalin first Come out, every time O (1) is used -Note: isPalin [i] [j] is inclusive, so you need to find the coordinates when you use it -Calculate isPalin [i] [j]: pick mid point [0 ~ n] -expand and validate palindrome at these indexes: [mid, mid + 1] or [mid-1] [mid + 1]

### Complexity

-Overall Space O (n ^ 2): isPlain [] []-Time O (2 ^ n), each layer is doing pick / not pick index i selection, so worst case 2 ^ n . -Because our isPalin [] [] optimizes Palindrome's judgment O (1), so overall Time: O (2 ^ n)

---

169. [Submatrix Sum.java] (https: // github. com / awangdev / LintCode / blob / master / Java / Submatrix% 20Sum.java) Level: Medium Tags: [Array, Hash Table, PreSum]

give an int [] [] matrix, find a sub matrix, where the sum == 0.

The idea of PreSum

-Calculate the size of a lower right corner point (i, j) to (0,0): previous block + left block + curr node-overlap area -preSum [i] [j] : sum from (0,0) to (i-1, j-1) -same approach as subarray sum : use hashmap to store diff-> index; if diff re-appears, that means sum of 0 has occurred -sequence of calculation: 1. iterate over start row. 2. iterate over end row. 3. iterate over col number (this is where hashmap is stored based on) -the iteration over col is like a screening: find previous sum and determine result -Note: Actually, I didn't really find the answer of == 0 , But judging by the characteristics of the rest / later must be 0

---

170. [Longest Palindromic Substring.java] (https://github.com/awangdev/LintCode/blob/master/ Java / Longest% 20Palindromic% 20Substring.java) Level: Medium Tags: [DP, String]

Give a string to find the longest

Palindrome substring. Related: Longest Palindromic Subsequence, Palindrome Partioning II

O (n ^ 2) is not too hard to think of. How about O (n)?

String, Palindrome definition

-split from the middle, traverse i: from n Different points of splitting: each time we see whether we can extend from the splitting as the midpoint of palindromic-palindrome two cases: odd, even palindrome -Worst case: the entire string is the same character, time complexity becomes: 1 + 2 +3 +. . . + n = O (n ^ 2)

DP: isPalin [] []-Exhaustive

double for loop. O (n ^ 2) -boolean isPalin [i] [j], it is recorded every time Palindrome is confirmed true / false -exhaustive for loop calculation order: end point j, and stat point i = [0, j] -when calculating isPalin [i] [j], isPalin [i + 1] [j-1] Should have been calculated.- double for loop: O (n ^ 2). Slower, because it guarantees O (n ^ 2) due to the for loop

O (n)

-TODO Https://www.felix021.com/blog/read.php?2040 - -the same way as model dp [i] [j]: range [i, j] Max palindromic length

---

171. [Longest Palindromic Subsequence.java] (https://github.com/awangdev/LintCode/blob/master/Java/Longest%20Palindromic%20Subsequence.java Level: Medium Tags: [DFS, DP, Interval DP, Memoization]

Give a string s, find the longest sub-sequence which is also palindrome.

Attention! subsequence is not a substring, but can be skip letter / non-continuous character sequence

Interval DP

-use [i] [j] to represent the beginning and end of the interval -consider 3 cases: behead, end, end and end (Considering the head-to-tail relationship) -Iteration must be viewed in terms of len between i ~ j. -Len = j-i + 1; then inverse, if len is known, j = len + i -1;-pay attention to consideration len == 1, len == 2 is a special case.- time / space: O (n ^ 2)

Memoization

-Three cases: -1. End -to -end match followed by dfs [i + 1, j-1] -2. Do not match, dfs [i + 1, j] -3. Do not match, dfs [i, j-1] -Note: init dp [i] [j] =-1, check if dp [i] [j] is counted when dfs -more about dfs: bottom-up, first dive deep into dfs (i + 1, j-1) till the base cases. -time / space: O (n ^ 2) -prepare dp [n] [n]: O (n ^

2); dfs: visit all combinations of [i, j]: O (n ^ 2)

---

172. [Gas Station.java] (https://github.com/awangdev/LintCode/ blob / master / Java / Gas% 20Station.java) Level: Medium Tags: [Greedy]

Give a string of gas station array, each index has a certain amount of gas.

Give a string of cost array, each index has a value , Is the fuel consumption of the next gas station in the reach.

At the end of the array, the next point is the beginning, forming a circle route.

Find an index, as the starting point: let the car go from this point, get oil, drive out, but also Drive back to this starting point

Greedy

-No matter where you start, you can record the total fuel consumption, total = {gas [i]-cost [i]}. Finally, if total <0, no matter where you start, you must not walk back -Bottom-top: first dfs to the deepest path, then gradually return online -you can record the accumulation of fuel consumption at each step, remain + = gas [i]-cost [i] -Once remain <0, it means that the previous starting point is not suitable, that is, the initial point must be at the following index. Reset: start = i + 1 -single for loop. Time: O (n)

NOT DP

-Seems a bit like House Robber II, but the question asks: the index of a starting point -instead of asking: can the last point be completed / maximum value / count

---

173. [Triangles.java] (https : //github.com/awangdev/LintCode/blob/master/Java/Triangles.java) Level: Medium Tags: [Array, Coordinate DP, DFS, DP, Memoization]

give a list <list > triangle , Details of the original question. Find min path sum from root.

DFS + Memoization

-Actually there is no difference to giving a 2D matrix, you can do dfs, memoization. -Initialize memo: pathSum [i] [j] = MAX_VALUE; Calculated path omitted - OR principle: min (pathA, pathB) + currNode -waste a little space, pathSum [n] [n]. Space: O (n ^ 2), where n = triangle height -Time: O (n ^ 2). Visit all nodes once: 1 + 2 + 3 + ... n = n ^ 2

DP

-Much like the principle of dfs, OR principle: min (pathA , pathB) + currNode -init dp [n-1] [j] = node values -build from bottom-> top: dp [i] [j] = Math.min (dp [i + 1] [j], dp [i + 1] [j + 1]) + triangle.get (i) .get (j); -Different from the traditional coordinate dp, the inner for loop needs to calculate j <= i, the reason is triangle nature.

- space: DP [n-] [n-] space: O (^ n-2).
- time:. O (n ^ 2) Visit all nodes once: 1 + 2 + 3 + ... n = n ^ 2

DP + O (n) space

-Based on the DP solution: the calculation always depend on next row for col at j and j + 1 -since only depend on next row, you can use Rolling array to deal with: reduce to O (n) space. -Further: You can reduce the dimension, remove the first dimension completely, and it will become dp [n] -Same double for loop, but only care about column changes: dp [j] = Math.min (dp [j], dp [j + 1]) + triangle.get (i) .get (j);

-
    -

174. [Merge Intervals.java] (https://github.com/awangdev/LintCode/blob/master/Java/Merge%20Intervals.java) Level: Medium Tags: [Array, PriorityQueue, Sort , Sweep Line]

give a string of int [Interval] (unsorted), merge all Intervals.

Sweep Line with Priority Queue

-O (nlogn) time (PriorityQueue), O (n) space
-Scan line + Count Invincible. Note that the start end closes the interval.
-When count == 0, it means the start / end of an interval every time the start end doubles off. Just write an example.
-Remember how to write comparator. New way: new PriorityQueue <> (Comparator.comparing (p-> p.val)); -In LeetCode, Sweep Line is much faster than method 2.

Sort Interval

-After Sort by interval.start, try to run it again, according to the needs of merge, continue the place where you need to merge, and then subtract the extra interval.

- Sort by Interval.start: `intervals.sort (Comparator.comparing (interval The -> interval.start)); // O (nlogn)`
- Related Example: the Insert Interval
- Interval connected with two: Curr, Next
- If curr.end covers next.start: merge is required. Then compare curr.end vs. next.end
- once merge, remove the next interval that needs to be overridden: `list.remove (i + 1)` -if there is no overlap , Continue iteration
- time O (nlogn), space O (1)

Sort Intervals and append end logically

-Sort intervals: O (nlogn), extra space O (n) when creating rst list -find the ending interval, Satisfy the conditions to save -if the conditions for return are not met, continue to extend interval.end

---

175. [H-Index.java] (https://github.com/awangdev/LintCode/blob/master/Java/H-Index.java) Level: Medium Tags: [Bucket Sort, Hash Table, Sort]

finds the h-index, and the citation int [] is not sorted. The definition of h-index depends on the subject.

Sort, find h from end

-The example is written out, and it can be sorted and searched according to the definition later. nlogn. -can be optimized when searching again, use binary search. But it doesn't make sense, because array.sort already uses nlogn -rules given by the title, after sorting from small to large: the remaining paper `nh`, all must be <= h Citation. -Time O (nlogn), search O (n)

Forward thinking

-start with i = 0 and find the first `citations [i]> = h`, which is the first one that matches h- index rule paper, return h

Thinking backwards

-if h = n, every time h--; then `x = n-h` is the first one starting from `(0 ~ n)` `dictations [x]> = h`, which is the result -at the same time, `dictations [x-1]` is the last (most dictation) remaining paper.

Bucket count / Bucket Sort

-O (n) -The idea of Bucket sort (more like counting sort?): After inputting again, use dictation value as index, and distribute it on bucket [index] ++ -bucket [x] is count when # of citation = = x. -If x is greater than n, it is beyond the index range, but this problem can be tolerated. Just record this situation in bucket [n] -clever: `sum + = bucket [h]` where `h = [n ~ 0]` uses the definition of h-index: -#of papers (sum of bucket [n] ... bucket [0]) has more than h cidations -the idea of bucket sort is used here, But it is not sorting, and the definition of h-index is clever. -Read more about actual bucket sort: https://en.wikipedia.org/wiki/Bucket_sort

---

176. [H-Index II. java] (https://github.com/awangdev/LintCode/blob/master/Java/H-Index%20II.java) Level: Medium Tags: [Binary Search]

find h-index, give citation int [ ] The sorted. H-index definition depends on the topic.

Binary Search

-A simple version of H-index, sorted (from small to large), find target value -By definition, find the last dictations [mid]> = h , where h = n-mid -O (logn)

---

177. [Sort Colors.java] (https://github.com/awangdev/LintCode/blob/master/Java/Sort%20Colors.java) Level: Medium Tags: [Array, Partition, Quick Sort, Sort, Two Pointers]

gives a string of numbers nums, the numbers represent the color [0,1,2]; requires sort nums, the numbers are finally arranged according to size.

Although called sort color, it is actually sort these numbers, but it is abstract look.

Array partition, the Base of Quick Sort

- the partition Array Pivot by K = {0,. 1, 2}
- are each partition Starting Point of the Current return partition
- and according to the next color, but also to There is no clean part of the sort, just sort it again
- time O (kn), where k = 0 => O (n)
- here is just a partion, there is no need to recursively quick sort, so the result is simple O (n)

One pass

- have two pointers, left/right
- start tracks red, end tracks blue. Swap red/blue to right position, and left++ or right--.
- leave white as is and it will be sorted automatically
- be very careful with index i: when swapping with index right, we do not know what is nums[right], so need to re-calculate index i .
- O(n)
- Note: this one pass solution does not work if there are more than 3 colors. Need to use the regular quick sorty.

Counting sort

- TODO: count occurance and reassign array

---

178. [Sort Colors II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Sort%20Colors%20II.java) Level: Medium Tags: [Partition, Quick Sort , Sort, Two Pointers]

Normal version of Sort Color, sort all k colors in colors array.

Details see: https://github.com/awangdev/LintCode/blob/master/Java/Sort%20Color.java

Quick Sort

-O (nk)

---

179. [Sort Letters by Case.java] (https://github.com/awangdev/LintCode/blob/master/Java/Sort%20Letters%20by%20Case.java) Level: Medium Tags: [Partition, Sort, String, Two Pointers]

Give a string of characters (ASCII uppercase, lowercase letters), require sort lowercase letters, before the uppercase letters.

The order of the letters is not important, and it is not necessary preserve original order.

Similar to sort color split.

Partition + Two pointers

-In fact, it is a simplified version of the partition function in quick sort -Two pointers, find a pivot 'a' to distinguish uppercase and lowercase letters -ASCII code in which uppercase letters precede lowercase letters, and the numbers are smaller -then while, move start ++, end- -, -Swap in each round time: O (nlogn)

Two pointers

-Directly mark the beginning and end with two pointer left / right -Each time you encounter > = 'a' is a lowercase letter, swap (chars, i, left); -Each time you encounter <' a' is uppercase Letter, swap (chars, i, right); -Note: Left swap is processed every time, let for loop i ++, because it is determined that [0 left] is accurate -Right swap is processed every time, we are not sure from right The index is not correct, so i--, offset from i ++ of the for loop.-It is easier to understand the solution of the while loop.

---

180. [Subarray Sum Closest.java] (https: //github.com/awangdev/LintCode/blob/master/Java/Subarray%20Sum%20Closest.java)### Level: Medium Tags: [PreSum, PriorityQueue, Sort, Subarray]

space: O (n)

gives a string of numbers , Find the beginning and end index of the subarray, Condition: The subarray is closest to 0.

PreSum + index in class

-Can be a 2D array, or a `class Point` to store preSum + index -Sort preSum: smaller (possibly negative) Forward, big numbers backward -Compare the two nodes connected by? PreSum, find the difference min; because the difference between the two closest preSum nodes must be the smallest -the index of the two nodes where min is, is the? Result candidate: these two indexes May be very different in the original nums -time O (nlogn), sort -space: O (n)

Why didn't you use map <preSum, index>?

-Because map can store preSum + index, but it is not effective Sorting -So use a class to store these two information, and then sort them reasonably

---

181. [Task Scheduler.java] (https://github.com/awangdev/LintCode/blob/master/Java/Task%20Scheduler.java) Level: Medium Tags: [Array, Enumeration, Greedy, PriorityQueue, Queue]

Array, count frequency, enumerate

-Enumerate to understand: 1. we can module the tasks in module / section; 2. Only need sum the intervals / slots, not return actual layout -Perfect condition, all letters appear identical # times: just line them up separate in order.

- Real case: task appears different times
- 1. Place maxCount task as header followed with n slots: define (maxCount-1) sections
- 2. For tasks with less # than maxCount# can fill the (maxCount-1) sections; what about the tail section?
- 3. Any task with same maxTask#, of if prior sections all filled, will fill the tail section
- To count overall slots/intervals, come up with this equation:
- 1. Fixed sections: `(maxCount - 1) * (n + 1)`
- 2. Plus all repeating maxCount tasks: calculate by couting identical maxCount of them
- 3. Exception: if the first (max-1) sections are all filled completely, and we still have extra task (ex: when n is not large enough), then just return tasks.length
- time O (1), space O (1)

PriorityQueue

- 正面去做:
- summerize the number of times each task occurs, and then qp sort Task object, count large forward
- start each section: k slots = n + 1
- the goal is to exhaust k, or exhaust pq (poll k times , but will save it back to queue if Task #> 0)
- if qp is really exhausted, break, return count
- otherwise, count + remain of k
- extra space O (x), time O (n) + constant time O (xlogx), where x = 26

---

182. [Exam Room.java] (https://github.com/awangdev/LintCode/blob/master/Java/Exam%20Room.java) Level: Medium Tags: [PriorityQueue, Sort]

PriorityQueue

-Use priority queue to sort by customized class interval {int dist; int x, y;} .- Sort by larger distance and then sort by start index -seat (): pq.poll () to find interval of largest distance. Split and add new intervals back to queue. -leave (x): one seat will be in 2 intervals: remove both from pq, and merge to a new interval. -The main equation is actually very easy to write, which is split + add interval, then find + delete interval. Most The hard part is building the data structure -seat (): O (logn), leave (): O (n)

Trick: Constructing a virtual boundary

-if it is the beginning of the seat, or the end of the seat, it is more difficult to handle : When sitting at seat = 0, there is no interval! -Trick is, we define a virtual seat seat = -1 , seat = N -At first, there is an interval [-1, N] Then boundary was established. -From now on, every time the split becomes a small interval: -When encountering interval [-1, y] , distance is (y-0) -When encountering interval [x, N] , distance is (N-1-x) -Of course, the normal interval dist is (y-x) / 2

TreeSet

distance

-Interval.dist What we actually do is the middle point of distance (y-x) / 2 -Here dist is distance from both sides , not distance between x, y. Pay special attention

here.-Https : //leetcode.com/problems/exam-room/discuss/139885/Java-Solution-based-on-treeset/153588

Map

-how? -TODO , not sure.

---

183. [Anagrams.java] (https://github.com/awangdev/LintCode/blob/master/Java/Anagrams.java) Level: Medium Tags: [Array, Hash Table]

Find and output anagram

HashMap

-There is int [26], Arrays.toString (arr) is string key: character frequency map -anagram has the same key, stored in hashmap <string, list of anagrams> -output anagrams

HashMap + Sort

-HashMap The method is to sort each string and store it in HashMap, the duplicate is anagrams, and finally output.
-toCharArray

- Arrays.sort
- Stirng.valueOf (char [])

- time n * L * O (logL) , L is the longest string length.

Previous Notes

-Arrays.toString (arr). arr is int [26], assuming only have 26 lowercase
letters.-Count occurrance, and then convert to String as the key of the map. -Time complexity: nO (L) -Another way: 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. -This step reduces the time complexity in for s: strs to O (L). L = s.length ().
-Need to work on the getHash () function. -Time becomes n * O (L). Better.

---

184. [Path Sum IV.java](https://github.com/awangdev/LintCode/blob/master/Java/Path%20Sum%20IV.java) Level: Medium Tags: [DFS, Hash Table , Tree]

gives a string of 3-digit arrays. Each number represents a TreeNode, and 3 digits represent: depth.position.value

This string has been arranged from small to large. Seek: All possible root-> leaf path The sum of all possible path sums.

DFS, Hash Table

-Because the first two digits can uniquely identify a node, the first two digits can be used as keys to locate the node. -Features: For example, consider root, there is n leafs, n roots will be added, because there are n unique paths. -Implementation: Each node, first add the curr value to the sum; as long as there is a child, the path sum to the position of this node will be added Repeat it again.- format: depth.position.value. (On same level, position may not be continuous) -approach: map each number into: <depth.position, value>, and dfs. -Start from dfs (map, rootKey, sum): -1. add node value to sum

- 2. compute potential child.
- 3. check child existence, if exist, add sum to result (for both left/right child). Check existence using the map.
- 4. also, if child exist, dfs into next level
- Space, time O(n)

---

185. Number Of Corner Rectangles.java### Level: Medium Tags: [DP, Math]

具体看题目: count # of valid rectangles (four corner are 1) in a grid[][].

basic thinking + Math

- Fix two rows and count matching columns
- Calculate number rectangles with `combination` concept:
- total number of combinations of pick 2 points randomly: count * (count - 1) / 2

DP

-TODO . HOW? #### Brutle -O (m ^ 2 * n ^ 2), times out

---

186. [Palindromic Substrings.java] (https://github.com/awangdev/LintCode/ blob / master / Java / Palindromic% 20Substrings.java) Level: Medium Tags: [DP, String]

According to the intent, count # of palindromic substring. (Substrings extracted from different indexes are different)

isPalin [] []-build

boolean [] [] to check isPalin [i] [j] with DP concept-? check all candidates isPalin [] [] -O (n ^ 2)

odd / even split check

https://leetcode.com/problems/palindromic-substrings/discuss/105689/Java-solution-8-lines-extendPalindrome

---

187. [Multiply Strings.java] (https://github.com/awangdev/LintCode/blob/master/Java/Multiply%20Strings.java) Level: Medium Tags: [Math, String]

for two integer String, Find product

String calculation, basic implementation

-let num1 = multipier, num2 = base.-mutiply and save into int [m + n], without carry. Loop over num1, each row num1 [x] * num2 -move carry to the correct index and direclty save result -calculate carry on rst []: sb.insert (0, c) such that no need to reverse () later -remove leading '0', but do not delete string " 0 " -time, space O (mn)

Previous notes.

- and so! Flip two numbers first! I go. This is a big pit.
- Bad solution: reversing makes it complicated, no need to reverse.
-
    1. The number '123'. In the array, index == 0 is '1'. But we usually used to start the product from the minimum number of digits, which is the beginning of the '3'.
-
    2. The product product is very common with moving Carrier.
- 3.! !! Finally, don't forget to flip it again.
-
    4. One last look at the pit. If the product is 0, it returns '0'. But this can actually catch from the beginning to the end without having to catch.
- A few good things about StringBuffer: reverse (), sb.deleteCharAt (i)-Find
  numbers, or 26 letters: s.charAt (i)-'0'; s.charAt (i)- 'a';

---

188. [Subsets.java] (https://github.com/awangdev/LintCode/blob/master/Java/Subsets.java) Level: Medium Tags: [Array, BFS , Backtracking, Bit Manipulation, DFS]

time: O (2 ^ n) space: O (2 ^ n)

gives a string of unique integers, and finds all possible subsets. There must be no duplicates in the result.

DFS

-dfs Two ways: 1. pick && skip dfs, 2. for loop dfs -1. pick && skip dfs: take or not + backtracking. When the level / index reaches the end, return a list. Bottom-up, reach the bottom, only the first solution is produced. -2. for loop dfs: for loop + backtracking. Remember: when doing a subset, each dfs recursive call is a unique possibility, add it to rst first. Top-bottom: If there is a solution, add it first.-Time && space : subset means independent choice of either pick && not pick. You pick n times: O (2 ^ n) , 3ms

Bit Manipulation

-n = nums.length, then at each index, it is pick / not pick : 0/1 -Consider bit index of subset index 0/1: range is [0000 ... 00 ~ 2 ^ n-1] -Each bitmap can show the contents of a subset: all the 1 represents picked indexes -Method: -1 . Find the Range -2. Traverse each bitmap candidate -3 . Traverse the bit representation of each integer, if it is 1, add to list -time: O (2 ^ n * 2 ^ n) = O (4 ^ n), still 3ms, fast.

Iterative, BFS

-Regular BFS, pay attention to consider if one level to generate next level -1 . Use queue to store the candidate indexes every time-2. Each time you open a layer of candiates, add them all to result -3. And use each round of dates, populate next level, back into queue. -Should be same O (2 ^ n), but actual run time 7ms, slower

---

189. [Subsets II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Subsets%20II. java) Level: Medium Tags: [Array, BFS, Backtracking, DFS]

time: O (2 ^ n) sapce: O (2 ^ n)

gives a list of integers (may have duplicates), find all possible subsets. There can be no duplicates in the result.

DFS

-DFS, find the data structures that need to pass along. First `sort input`, then DFS -sort O (nlogn), subset: O (2 ^ n) -Using for loop approach: each dfs call Is a possibility, directly add into result.
-To remove duplicated results, skip used item at current level: `if (i> depth && nums [i] == nums [i-1]) continue;` -space O (2 ^ n), save results

# BFS

-Regular BFS, pay attention to consider if one level to generate next level -skip duplicate: `if (i> endIndex && nums [i] == nums [i-1]) continue;` -1 . Use queue to save Candidate indexes each time-2. Candiates each time, add them all to result -3. And candidates, populate next level, back into queue.-Srot O (nlogn), subset: O (2 ^ n) -should be same O (2 ^ n). Slower than dfs

Previous notes:

-Skip duplicate candidates in DFS, based on the sorted array technique:-Once
i! = Index in the for loop, And nums [i] == nums [i-1], -Explain that x = nums [i-1] has been used at the curr level and does not need to be used again: [a, x1, x2], x1 == x2
-i == index-> [a, x1]
-i == index + 1-> [a, x2]. We want to skip this one -what if [a, x1, x2] is needed? In fact, when the index changes, it will be involved in two different dfs calls.

Note

-you cannot use result.contains (), which is very costly O (nlogn) -using list.toString () several times is actually O (n) iteration, which actually increases the check time, no Suggestion

---

190. [Combination Sum.java] (https://github.com/awangdev/LintCode/blob/master/Java/Combination%20Sum.java) Level: Medium Tags: [Array, Backtracking , Combination, DFS]

time: O (n!) Space: O (n!)

Gives a string of numbers dates (no duplicates), and a target.

Find all unique combinations (combination) int [], requiring each combination of And = target.

Note: The same candidate integer can be used any number of times.

DFS, Backtracking

- considering input: no duplicate, not Need sort
- Consider the rules of reuse: Can be reused, then when using dfs in the for loop, use curr index i
- the result is trivial, save success list into result.

Time complexity for Combination (reuse-candidate)

-At each level dfs, we have the index as starting point:

- if we are at `index = 0, we can have n child dfs () options via for loop`;-if at `index = 1, we will have (n-1) dfs options via for loop`.
- Consider it as the `pick / not-pick` problem, where the difference is you can pick `x` times at each index rather than only 2 times.
- Overall, we will multiply the # of possibilities: n * (n-1) * (n-2) ... * 1 = n! => `O (n!)`

Combination DFS idea

-at each index: `pick / not pick choice`, use for loop over index + backtracking to implement picks. -After each pick, a new routine is generated, from this index -The next level of dfs starts from this index, and the same pick / not pick is selected for the later (or the current / if allow index reuse)
-Note 1: In each level dfs, there will be an end condition in the for loop: the dfs is no longer necessary. -Note 2: Backtracking must be done after success case && dfs case, because backtrack is For the previous dfs.

---

191. [Combination Sum II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Combination%20Sum%20II.java) Level: Medium Tags: [Array, Backtracking, Combination, DFS]

Give a string of numbers dates (can have duplicates), and a target.

Find all unique combinations (combination) int [], requiring the sum of each combination = target.

Note: The same candidate integer can only be used once.

DFS, Backtracking

-when the input has duplicates, and want to skip redundant items? -1. sort. 2. in for loop, skip same neighbor. -Consider input: duplicate, must be sorted -rules for reuse are considered: cannot be reused -1 For dfs in the for loop, use curr index + 1-2 -2. In a loop, the same level and the same number cannot be reused: `(i> index && candidates [i] == candidates [i-1]) continue` -because the numbers are repeated in the same level It will be taken into account in the next dfs level, here must be skipped (remember this) -the result is trivial, save success list into result.

Time complexity

-Which one? -Time: every level has 1 less element to choose, worst case is: cannot find any solution over all combinations: O (m!) -Time: Same as `subsetII`, pick / not = pick an item as we go, no reuse of item. Worst case: all unique items in the set. O (2 ^ n)

---

192. [Combination Sum III.java] (https://github.com/awangdev/LintCode/blob/master/Java/Combination% 20Sum% 20III.java) Level: Medium Tags: [Array, Backtracking, Combination, DFS]

gives an integer k, and a target n.

From the positive numbers [1 ~ 9], find all unique combinations (combination) int [], size = k, requires the sum of each combination = n.

(Hidden condition, requires declare): the same candidate integer [1-9] can only be used once.

the DFS, backtracking

- with Combination Sum I, II is not much difference, but be sure to use k digits, which is a special condition inside the loop for
- Consider input: no duplicate numbers [1 ~ 9]
- consider candidate reuse: no reuse, next level dfs, curr index + 1
- the result is trivial, save success list into result.

Time Complexity

-Which one? -Worst case: tried all numbers and cannot find: O (m!), M = 9, all possible integers in [1 ~ 9] -C (n, k), n choose k problem: `n! / (K! * (Nk)!)` => Ends up being `O (min (n ^ k, n ^ (nk)))`

- -

-
    - 193. [Product of Array Except Self.java] (https://github.com/awangdev/LintCode/blob/master/Java/Product%20of%20Array%20Except%20Self.java) Level: Medium Tags: [Array, PreProduct]

time: O (n) space: O (1)

gives a string of numbers, output rst [n], each index is the product of all itemds except nums [i].

Array, PreProduct

-Analyze common practices, and finally use O (n) from left to right, and O (n) from right to left. -Pay attention to the maintenance of carry -see the second answer, further? Simplified the code -PreProduct, and preSum feels a bit like, but it's one bit worse.

---

194. [Total Hamming Distance.java](https://github.com/awangdev/LintCode/blob/master/Java/Total%20Hamming%20Distance. java) Level: Medium Tags: [Bit Manipulation]

time: O (n) Space: O (1), 32-bit Array

gives Hamming Distance definition (how much binary diff is in bit format), find the sum of the Hamming distance of a series of numbers.

Bit Manipulation

- Bit title: test bit >>, mask & 1, as well as understanding of the subject
- of Put integers in binary, and the Compare the each column:
- for each 1 , ask: how many are different from me? all the 0 - # of diffs at each bit-column = #ofZero * #ofOne
-
    1. countZero [], countOne []; 2. loop over nums and populate the two array

Pay attention to the thunder point

-ask clearly: 10 ^ 9 <2 ^ 31, we are okay with 32 bits- ? Final Hamming Distance? Which bit to [1 ~ 32] Starting to count ? Depends on the longest binary format: but don't need to find the bit length first -when doing countZero, countOne, all do 32-bit; when doing the final product, if 1 or 0 The number is zero, the product is naturally 0.

---

195. [Smallest Subtree with all the Deepest Nodes.java](https://github.com/awangdev/LintCode/blob/master/Java/Smallest%20Subtree%20with%20all%20the% Level: Medium Tags: [DFS, Divide and Conquer, Tree]

time: O (n) space: O (n)

for a tree,? Find the most node according to the intent? Satisfy: 1.? The subtree of this node Covers all leaves at the deepest level. 2. This node must be the deepest one that can be found ? The requirement of condition 2 is because: root itself is a node that meets condition 1, and there are many higher-level nodes, so find that deepest.

DFS on tree

-analyze the topic, the idea is: see all the leaves in the tree, find their deepest common ancestor -Maintain a map <Node, maxChildDepth>-Recursively dfs: return deepest node that has all leaves by these comparisons: -1. If left, right child same depth, return root: they need common ancestor
-2. If not same depth, return the one with larger depth -Transferred to the previous level, always in the subtree: 1. the node containing all leaf nodes -Visit all nodes once O (n), space O (n)

BFS

-Find all leaves at deepest level -Use map to track each node-parent -Backtrack all nodes to find common ancestor

---

196. [Subarray Sum Equals K. java](https://github.com/awangdev/LintCode/blob/master/Java/Subarray%20Sum%20Equals%20K.java)

Level: Medium Tags: [Array, Hash Table, PreSum, Subarray]

time: O (n) space: O (n)

gives a string of numbers, find # of subarray where subararySum == k.

Hash Table + PreSum

-Hash Table two sum thought, but `save frequency of current preSum` -map.get (priorSum) = the # of possible ways to reach k -Keep counting -O (n) time, O (n) space

Detailed explanation

- From the orignal presum solution: `target = preSum[j] - preSum[i - 1]`. Here: `k = sum - priorSum`, and reversely, `priorSum = sum - k`
- priorSum is just previously calcualted sum; track its frequency using `map<preSumValue, frequency>`
- map.get(priorSum): # ways to sum up to priorSum.
- Also, to get `priorSum + k = sum`: each unique way of building priorSum will append later elements to reach sum (the later elemnts will sum up to k)
- Therefore # ways to build `k = map.get(priorSum)`

PreSum, O(n^2)

- move from starting point i = [0 ~ n -1] and define range = [i ~ j]
- use presum to verify k: `preSum [j]-preSum [i-1]`
- O (n ^ 2): `1 + 2 + 3 + 4 ... + n ~ = O (n ^ 2)`

---

197. [Simplify Path.java] (https://github.com/awangdev/LintCode/blob/master/Java/Simplify%20Path.java) Level: Medium Tags: [Stack, String]

time: O (n) space: O (n)

gives a path, simplify to the simplest form. Note that consider edge case

Stack

-understand the situation of unix path, do not know how to ask: -1. `.` stands for current directory, It can be ignored. -2. `../` means the previous level. -3. Double slash can be ignored. -4. The empty string is output `/` -In the end, the stack (`item added to the previous is used to prepare Select pop () out`), when you encounter `../` pop () drop the last added item, and add the rest to the stack -finally use '/' to connect all items.

---

198. [Convert Binary Search Tree to Sorted Doubly Linked List (extra space) .java] (https://github.com/awangdev/LintCode/blob/master/Java/Convert%20Binary%20Search%20Tree%20to% 20Sorted% 20Doubly% 20Linked% 20List% 20 (extra% 20space) .java) Level: Medium Tags: [Linked List, Stack, Tree]

time: O (n) space: O (n)

to a BST, convert into sorted doubly DoublyListNode.

Inorder Traversal, Linked List

-will iterative traverse Binary Search Tree (Stack && handle left-dig-down) -create Doubly-ListNode, note the use of a dNode as the tail node of the list

# Iterative inorder traversal

-After checking the right node,
-regardless of right == null or! = Null, you must move to right every time.-If not node = node.right,-a
dilemma is likely to occur:
-node always = stack.top (), then stack.top () has always been the first to traverse all the left. So it will infinite loop, always up and down on the left.

---

199. [Binary Tree Zigzag Level Order Traversal.java] (https://github.com/awangdev/LintCode/blob/master/Java/Binary%20Tree%20Zigzag%20Level%20Order Level: Medium Tags: [BFS, Stack, Tree]

time: O (n) space: O (n)

Queue

-Simple level traversal. Add to different positions according to level odd and even.-Option1: based on level% 2, insert to front / end of list -Option2: based on level, insert right / left of node into queue

---

200. [Word Break.java] (https://github.com/ awangdev / LintCode / blob / master / Java / Word% 20Break.java) Level: Medium Tags: [DP, Hash Table, Sequence DP]

time: O (n ^ 2) space: O (n)

Give a String word, and a dictionary, check whether the word can be split, and all substrings should be the words in the dictionary.

Sequence DP

-true / false problem, think about dp -sub-problems: the first i Letters, is there a valid break -check dp [j] && if substring (j, i) valid, for all j = [0 ~ i]-dp = new boolean [n + 1]; dp [0] = to true;

- Goal: IF there IS AJ, DP [J] == && to true Word [J, n-] in the iterate over dict Need I = [0 ~ n-], Also J = [0, I].
- Note Use set instead of list because contains () is used.

Previous notes

Method 2 (attempt4 code)

-Use the same DP as Word BreakII. -valid [i]: record if i is valid from i to the end of the valid array.

Method 1: (attempt3 code)-function

, rst [i]: Can the inclusive string from [0 ~ i] be in the dict? Break to find it?
: rst [i] = true if (rst [i-j] && set.contains (s.substring (i-j, i))); j in [0 ~ i]
-1. rst [i-j] records whether [0, ij] can be found in dict after break.
-2. If true, plus all remaining [ij, i] can be found in dict, then rst [i] = rst [0, i-j] && rst [ij, i] == true -optimization: Find the longest string in dict and limit the increase of j.

---

201. [Longest Increasing Subsequence.java] (https://github.com/awangdev/LintCode/blob/master/Java/Longest%20Increasing%20Subsequence.java) Level: Medium Tags: [Binary Search, Coordinate DP, DP, Memoization]

time: O (n ^ 2) dp, O (nLogN) binary search space: O (n)

unordered array, find the length of the longest rising (no continuous) array. First Do O (n ^ 2), and then O (nLogN)?

DP, double for loop, O (n ^ 2) -When

subsequence: no continuous, you can skip candidate considering the end of nums [i], at [0, i), dp [How many counts in i-1] are less than nums [i] -dp [i]: up to i (for all j in [0, i], record the max length of increasing subsequence -max needs to be maintained globally: nums is out of order, nums [i] is also It may be a small value, so the end dp [i] is not the global max, but only the max for nums [i] .-Because of this, each nums [i] must work with each nums [j] Compare, j < i.-Dp [i] = Maht.max (dp [i], dp [j] + 1); j = [0, i-1]-time complexity O (n ^ 2)

## O (nLogN)-Maintain

a list of increasing sequence -This list is actually a base-line, recording the lowest increasing sequence. -When we go through all nums, if it happens to rise, directly append -if No rise, you should go to the list, find the smallest number that is just larger than new num, and replace it with num -this completes the baseline. For example, for example, the replacement is just at the last element of the list, which is equivalent to putting the peak It has fallen, then the other numbers may continue to rise.- The proof of 'maintaining the baseline is an increasing number series', I haven't thought about it carefully.

---

202. [Best Time to Buy and Sell Stock with Transaction Fee.java] (https://github.com/awangdev/LintCode/blob/master/Java/Best%20Time%20to%20Buy%20and%20Sell% 20with% 20Transaction% 20Fee.java) Level: Medium Tags: [Array, DP, Greedy, Sequence DP, Status DP]

time: O (n) space: O (n), O (1) rolling array and

Stock Like II, the sale of infinite, you must first buy sell additional conditions: each sell transaction to add a sum Fee..

Sequence DP

- and StockII the same, dp [i]: i days ago represents the greatest Profit.
- When sell is completed, a transaction is completed and fee is deducted. Fee is not deducted when buying.
- model sell on dp [i] day (which depends on dp [i-1]) and each day can be sell / buy = > add status to dp [i] [status]
- status [0] buy on this day, status [1] sell on this day
- dp [i] [0] = Math.max (dp [i-1] [0], dp [i-1] [0]-prices [i]);
- dp [i] [1] = Math.max (dp [i-1] [1], dp [i-1] [1] + prices [i]-fee);
- init: dp[0][0,1] = 0; dp[1][1] = 0; dp[1][0] = - prices;
- return dp[n][1]

---

203. Random Pick Index.java### Level: Medium Tags: [Reservior Sampling]

time: O(n) space: O(n) for input int[], O(1) extra space used

Reservior sampling

- Random choose: think about reservoir sampling. https://www.youtube.com/watch?v=A1iwzSew5QY
- Use random generator rd.nextInt(x) pick integer between [0, x)
- try all numbers, when target is met, we want to model reservoir sampling:
- item was chosen out of i samples, and all other samples are failed.
- where we can use 'count' to represent the denominator/base to choose.
- HAVE TO finish all samples### to make sure equal opportunity
- we can pick that last matched item as result
- `rd.nextInt(count++) == 0` make sure we are always picking num == 0 to meet definition of reservoir sampling.

Knowledge

- If multiply these probablities together to get the probability of one item being chosen with reservior sampling:
- probability = 1/i * (1 - 1/i+1) * (1 - 1/i+2) ....(1 - 1/n) = 1/n

---

204. Find the Celebrity.java### Level: Medium Tags: [Array, Greedy]

time: O(n) space: O(1)

有n个人, 其中有个人是celebrity, 满足条件 Celeb knows nobody; Everyone else knows the celeb. 找到celeb

Understand the property

-We can greedy Ground, once fail one, immediately assume the next one is celeb candidate -If brutly find celeb by comparing all possible pair: take complete O (n ^ 2) handshakes. -Instead, we can perform pruning, or like survival mode: -1. Assume a celeb = 0, and compare with all i = [1 ~ n-1] -2. If `celeb candidate know i`, `set celeb = i` as the next candidate (ex: prev canddiate invalid when he knows i) -3. For last standing celeb candidate: compare with all for validation -Why performing the last run of validation? There could be someone dropped out before we execute `know (celeb, i)`.

Thinking logic

-write it first [0 ~ n-1], the simplest way O (n ^ 2) check and record the status of each person.- Gradually find out that because celeb will not know anyone, then when any candidate knows anyone, he is not celeb. -In the end, it is necessary to check it again to avoid mistakes and omissions.-In the -Think about the happy case: if celeb = 0, then know (celeb, i) is always false, and then celeb remains 0, and persists until verify everyone.

---

205. [Sparse Matrix Multiplication.java] (https://github.com/awangdev/LintCode/blob/master/Java/Sparse%20Matrix%20Multiplication.java) Level: Medium Tags: [Hash Table]

time: O (mnk), where `m = A.row`,`n = B.col`, `k = A.col = B.row` space: O (1) extra

gives two matrices, do the product. Note, yes sparse matrix (features: many 0).

Hash Table

-Recall matric multiplication rules: result [i] [j] = sum (A-row [i] * B-col [j]) -`sparse matric: lots positions are zero` -It doesn't make sense to write matric multiplication plainly, the point is optimization: -`optimization`: for A-zero-row, and B-zero-col, there is no need to calculate, just return 0.

Hash Table

-1. Find A-zero-rows and store in setA, same for setB -2 . during multiplication, reduce time complexity. -Base: O (mnk), where `m = A.row`,`n = B.col`, `k = A.col = B.row`

the Matrices

- product rule: Result [I] [J] = SUM (A-Row [I] * B-COL [J])
- A column size and size == B Row: A calculation procedure is column size over the iterate
-
    -

206. [Brick Wall.java] (https://github.com/awangdev/LintCode/blob/master/Java/Brick%20Wall.java) Level: Medium Tags: [Hash Table]

time: O (mn) space: O (X), X = max wall width

for a wall, each line is a line of bricks. Scanning with a vertical line will cut the bricks vertically. Find the x index of the line that cuts the least brick .

-Find the vertical line (x-coordinate of the grid), where most gaps are found. -Each gap has (x, y) coordinate

- Create `map<x-coordinate, #occurrance>`, and maintain a max occurance.
- 计算: x-coordinate: `x = 0; x += brick[i] width`
- Eventually: min-crossed bricks = wall.lenght - maxOccurrance

思想

- 分析题意, 找到题目的目标
- 虽然Map自己不能有sort的规律, 但是可以maintain global variable

---

207. Exclusive Time of Functions.java### Level: Medium Tags: [Stack]

Stack

- 1. later function always appears after prior fn: 1 is called by 0
- 2. Not mentione in the question : a function can be started multiple times
- 3. Not mentione in the question : a fn cannot start if children fn starts
- 4. Use stack to keep id
- TODO: what leads to the choice of stack? stacking fn id

---

208. [Friends Of Appropriate Ages.java] (https://github.com/awangdev/LintCode/blob/master/Java/Friends%20Of%20Appropriate%20Ages.java) Level: Medium Tags : [Array, Math]

Array, Math

-This problem lies in the analysis of the problem itself (and there are redundant conditions); the final for loop is also less standard. -People younger than 15 cannot make requests due to the first rule. -From the age of 15, people can make requests to the same age: a [i] * (a [i]-1) requests.

- People can make requests to younger people older than 0.5 * i + 7: a[j] * a[i] requests.
- The third rule is redundant as the condition is already covered by the second rule .
- TODO: the approach.

---

209. [Target Sum.java] (https://github.com/awangdev/LintCode/blob/master/Java/Target%20Sum.java) Level: Medium Tags: [DFS, DP]

// How to think of initialize from the middle

---

210. Maximum Size Subarray Sum Equals k.java### Level: Medium Tags: [Hash Table, PreSum, Subarray]

time: O(n) space: O(n)

Map<preSumValue, index>

- use Map<preSum value, index> to store inline preSum and its index.
- 1. Build presum incline
- 2. Use map to cache current preSum value and its index: Map<preSum value, index>
- 3. Each iteration: calculate possible preSum candidate that prior target sequence. ex: (preSum - k)
- 4. Use the calculated preSum candidate to find index
- 5. Use found index to calculate for result. ex: calculate range.

---

211. [Contiguous Array.java] (https://github.com/awangdev/LintCode/blob/master/Java/Contiguous%20Array.java) Level: Medium Tags: [Hash Table]

TODO: how aout without chaning the input nums?

---

212. [Line Reflection.java] (https://github.com/awangdev/LintCode/blob/master/Java/Line%20Reflection.java) Level: Medium Tags : [Hash Table, Math]

time: O (n) space: O (n) . When processing left == right, it is treated as two points. -4. There is no sort in the set, but the check is done at the end When you need a sort list

Give a list of points, find if there is a middle line in the middle of all the points, parallel to the y-axis.

Hash Table

-1. store in `Map <y, set <x >>`, 2. iterate over map , Check head, tail Against the MID point

- nice detail Title:
- 1 divided by 2, need to save Double
-
    2. (ask the interviewer) may duplicate points so SET `Track <X>`!
- time : visit all nodes twice, O (n)

---

213. [Insert Delete GetRandom O (1) .java] (https://github.com/awangdev/LintCode/blob/master/Java/Insert% 20Delete% 20GetRandom% 20O (1) .java) Level: Medium Tags: [Array, Design, Hash Table]

time: O (1) avg space: O (n)

Hash Table

-Use `map < value, index> to track value-> index`, use `list track index-> value` -map to see if value exists -list maintain is used for insert / remove / random operations. -Features: Once , switch to the end of the list and (list.size () -1) `, so the cost of remove is lower.- list.remove (object) should be a search by O (logn)

.---

214. [Number of Longest Increasing Subsequence.java] (https://github.com/awangdev/LintCode/blob/master/Java/Number%20of%20Longest%20Increasing%20 Level: Medium Tags : [Coordinate DP, DP]

time: O (n ^ 2) time: O (n)

gives a string of unsorted sequence and finds the number of long increasing subsequences!

Coordinate DP

-Need to be able to judge comprehensive questions and be clear Situation and routine: combination of `longest subsequence` and `ways to do`, as well as global variable. -Len [i] (our usual dp [i]): in the first i elements, the longest incident subsequence length;
-count [i]: in the first i elements, and the count of the subsequence based on the length of len [i]. Or: in the first i elements, ways to reach longest increasing subsequence.- `len [i] == len [j] + 1` : same length, but different sequence, so add all `count [i] + = count [j]` - `len [i] <len [j ] + 1` : This is where the longer case is found, then there are as many count [j] as there are count [i]. Think about sequence: the length has increased, but the way to reach i has not increased.-The same judgment needs to be used on maxLen and maxFreq:-If maxLen is not increased, maxFreq needs + = count [i] (the same length, more ways) -If maxLen becomes longer, maxFreq is also Count [i] = count [j] -TODO : Is rolling array possible?

Related -Both are the originator of Coordiate DP, DP:-Longest

Increasing Subsequence (exactly the same as part of this question) -Longest Continuous Increasing Subsequence (Continuous, only check dp [i-1]) -Longest Increasing Continuous Subsequence I, II (Lintcode, II is matrix)

---

215. [Minimum Swaps To Make Sequences Increasing.java] (https: //github.com/awangdev/LintCode/blob/master/Java/Minimum%20Swaps%20To%20Make%20Sequences% Level: Medium Tags: [Coordinate DP, DP, Status DP]

DP

-Features: The previous step may be swaped or fixed -Consider the current situation between A and B: A [i]> A [i-1] && B [i]> B [i-1] or A [i]> B [i-1] && B [i]> A [i-1] -Question: How to turn this state into a reasonable strick-increasing state?- A [i]> A [i-1] && B [i]> B [i- 1] : 1. It's reasonable and doesn't move. 2. [i], [i-1] All swap - A [i]> B [i-1] && B [i]> A [i- 1] , staggered, so change [i], or [i-1]: 1. Change [i-1]. 2. Change [i] -Note that since min is calculated, the init value should be Integer.MAX_VALUE ;

---

for a Binary Tree, traverse all nodes, arranged in output order according to vertical order: List

216. [Binary Tree Vertical Order Traversal.java] (https://github.com/awangdev/LintCode/blob/master/Java/Binary%20Tree%20Vertical%20Order%20Trav Level: Medium Tags : [BFS, DFS, Hash Table, Tree]

time: O (n) space: O (n) The

key point is: there is sorting in col, it is ranked at the higher level; if node encounters collision in the same position: according to Their relative position is left first, then right

BFS

-it should be more imaginative: naturally level-traverse all nodes, add node to appropriate col list -Use min / max to track map keys, since the keys are continous -Map does not provide random access; unless map key is marked with sequence i = [min, max]

DFS

-It is easy to think at first: enumerate it, put curr node.val first, then node.left.val , node.right.val. is very simple -But the easiest way is wrong: assume that all left subtrees are ranked in the right subtree. However: right subtree may have a lower-left-branch, appear in a column first. -So also reserve column list of the Order.

- here we use the `map <col, Node>` to track col, Node inside with a `node.level` to track level (in fact, then a map can be)
- so in the end you want to sort, it will be very Slow: Visit all nodes O (n) + O (logK) + O (KlogM), K = # of cols, M = # of items in col
- It should also be possible to optimize map keys. Anyway, they are continuous keys

---

217. [Populating Next Right Pointers in Each Node II.java] (https://github.com/awangdev/LintCode/blob/master/Java /Populating%20Next%20Right%20Pointers%20in%20Each%20Node%20II.java)### Level: Medium Tags: [DFS, Tree]

time: O (n) space: O (1)

for a binary tree, use constant space link all all nodes.next to the same level next node.

DFS

-using constant space is not BFS, but it is fine to use dfs stack space (mention!) -1. link leftChild-> rightChild -2. resolve root.rightMost child-> first possible root.next.left / right child -3. dfs connect (rightChild), connect (leftChild)-Each level should be fully linked from left side, so every reach to parent will have valid path or

Binary Search

### Trick

-1. Handle the case of nextNode-> next-> next ...: find the first next node with child. This case is easy to miss -2. Our assumption is that all nodes at the previous level are It should be linked, then in dfs, you should connect (root.right) first. After the right child is completely processed, then trick1 can be implemented.

---

218. [Search in Rotated Sorted Array.java] (https : //github.com/awangdev/LintCode/blob/master/Java/Search%20in%20Rotated%20Sorted%20Array.java) Level: Medium Tags: [Array, Binary Search]

time: log (n) space: O (1)

-The key point is to find the continuous increasing subarray of [mid] on the left / right: compare A [start] <A [mid] -discuss the position of the target in two sections
-1. nums [start] <nums [mid] : start starts at index = 0, which means that mid is in the first half - start <target <mid : target is in this section, end = mid;

- target> mid : start = mid; half -2. nums [start]> nums [mid] : start starts at index = 0, That means mid is in the second half - mid <target <end`: start = mid;
- target <mid : end = mid;

binary search break point, then continue to binary search target

-1 binay search break point
-2. binary search target
-pay attention to the equal sign, in determining whether the target is in the first half or the second half: if (A [p1] <= target && target <= A [breakPoint])

---

219. [Find the Weak Connected Component in the Directed Graph.java] (https://github.com/awangdev/LintCode/blob/master/Java/Find%20the%20Weak%20Connected%20Com 20Directed% 20Graph.java) Level: Medium Tags: [Union Find]

Iterates over weak connected graph and stores the results in List <List > .-

Union Find

-Two differences from the traditional UnionFind:

1. Use Map <Integer, Integer> instead of int [], because The boundary of the graph node label is not given.-2. When find (x), I did not update parent [x] /map.put (x, ..) . Because we eventually need to find this path.

-Cannot use traditional dfs: directed node cannot point to the previous point; all nodes must be traversed by using the method of "storing parent"

Identify this is a union-find problem

-see the form of weak component: one point points to all, then all All points have a common parent, and then these points are to be found.
-Why ca n't we start from a point, such as A, and print all its neighbors directly?-If it is B's turn, then because it is directed, it does not know the situation of A, nor does it know how to continue to add, or start.
-So, put all points that are related to A, or points that are related to A's neighbor, into the union-find, so that these points have Common parents.
-The final output idea:
-Make a map <parent ID, list>.
-We didn't save parent for each num before.
-Each num has a parent, and then different parents create a different list.
-Finally, just take out all the lists in the map.
-init with <email, email> for all emails

---

220. [Accounts Merge.java] (https://github.com/awangdev/LintCode/blob/master/Java/Accounts%20Merge.java) Level: Medium

Tags: [DFS, Hash Table, Hash Table , Union Find]

Give a string of account in format [[name, email1, email2, email3], [name2, email, ..]].

Require all accounts to be merged (maybe multiple records record the same person,? by common email)

Union Find

-build Map <email, email parent>, and then integrate backwards: parent-> list of email -because different accounts may string emails, then when you combine all emails, Emails of different accounts will also be chained -eventually: all emails are unionized , pointing to a parent email of each union -ParentFind of UnionFind can output all child under parents in reverse. -Also maintain an < email- > account name> map, which is ultimately used for output.

Hash Table solution, passed but very slow

output.-Definitely need iterate over accounts: merge them by email. -Account object {name, list of email}

- map <email, account>
- 1. iterate over accounts
- 2. find if 'account' exist; if does, add emails
- 3. if not, add account to list and to map. map all emails to accounts.
- output- > all accounts, and sort emails
- space O (mn): m row, n = emails
- time O (mn)

---

221. [Count of Smaller Number.java] (https://github.com/awangdev/LintCode/blob/master/Java/Count%20of%20Smaller%20Number.java) Level: Medium Tags: [Binary Search, Lint, Segment Tree]

gives a string of numbers, array size = n. Gives a string of queries: each query is a number, the purpose is to find count # items smaller than query element.

Segment Tree

-The segment tree problem is different in peacetime. [0 ~ n] represents the actual number: segment tree based on real value.-When modifying , bring the value in the array to find a specific seat, and then count + 1. -Finally on the SegmentTree leaf is the actual array inside Numbers. -node.count: how many numbers are there in the node range

right use of modify () -build

() is only an empty segment tree, no property -modify () requires: 1. find left, update count + = 1; 2. aggregate all parent when after returning -so each modify is on all nodes on the entire path + count

query trick

-Before the query, the start and end given are: 0 ~ value-1.
- Value-1 : find a range that is 1 less than the range in which you are (so naturally you are not included), so you find it smaller number.

About other basic segment tree setup

setup-[The other SegmentTree I have done is how is it?]
-Those well-formed SegmentTrees (find min, max, sum) also have an Array. However, when constructing a Tree, it is structured with the index of the Array.
-that is, if there is Array [x, y, ...]: in leaf, there will be [0,0] with value = x. [1 , 1] with value = y.- [But this question]
-When constructing, you use actual value. That is, for example, Array [x, y, ...] will produce leaf: [x, x] with value =. .; [y, y] with value =
...- In fact, it is easy to see through: -If a fixed array is formed to form SegmentTree, it is estimated to be simple: according to the index from 0 to array.lengh, the leaf is [0, 0] with value = x.-If the title asks to construct a hollow SegmentTree, based on value 0 ~ n-1 (n <= 10000), then modify the value of an Array

into it.
-This 80% is another.

---

222. [My Calendar I.java](https://github.com/awangdev/LintCode/blob/master/Java/My%20Calendar%20I.java) Level: Medium Tags: [Array , TreeMap]

Given a list of interval as calendar items. Check if newly added calendar item is overlapping.

Understand it is only checking time, but not requiring to insert into right spot. No need to overthink.

Simply O(n) check on array

- number of test cases is small, like 1000, so less concern about the time complexity
- simply loop over the list of intervals, and check if any overlapping.
- where to insert does not really matter: every time we are just checking for overlaopping, not merging any range
- IMPORTANT: if interval over lapping, they will have this property `Math.max(s1, s2) < Math.min(e1, e2)`. This will help detect the overlapping very easily.
- O(n^2) runtime, with simple code. But somehow this approach is faster than the TreeMap solution: maybe the test cause causes avg O(n)?

TreeMap

- One constraint from the simply array solution: it always cost O(n) to find the potential overlapping interval
- We can manually sort and always manually try to find the correct element via binary search, or we could store the interval in a treeMap<startKey, endValue>, where the intervals are sorted by `start`.
- As result, all we need to do for book(start, end) is to find the next element ceiling(start), last element floor(start), and check for overlapping
- This approach also saves the custom data structure
- Overall cost O(nlogn)

About TreeMap

- always with key sorted ascendingly
- more costly than regular HashMap because of the sorting. Building treemap of n items: O(nlogn)

Sweep line

- use `Point{int start, end; boolean start}` to mark start/end of class. Add to pq.
- Adding new item to pq, sort, and check if overlapping occurs by counting started classes
- If started classes > 1, that means we overlapped.
- Every time it could consume all classes to find the overlap, O(n^2).
- Not quite need to sort or insert at correct point, and this solution requires longer code. Not quite worthy it for a simple problem.

---

223. Reverse Pairs.java### Level: Medium Tags: [Binary Indexed Tree, Binary Search Tree, Divide and Conquer, Merge Sort, Segment Tree]

给一串数字, count total reverse pair `nums[i] > 2*nums[j]`, i < j

This problem can be solved with Merge sort concept, BST, Segment Tree and Binary Indexed Tree. Good for learning/review.

Merge Sort

- Using merge sort concept, not exaclty merge sort implementation.
- One very simply concept: if we want to know how many elements between [i, j] are meeting requirements of `nums[i] > 2*nums[j]`, it would be really helpful, if the entire range is sorted:
- then we just need to keep one i index, and keep j++ for all elements meeting requirement `j<=e && nums[i]/2.0 > nums[j]`
- Then it comes to the sorting part: we cannot just directly sort entire array, because the restriction is `all elements on right side of curr element`. BUT, it is okay to sort `right side range` and compare with left side elements :)
- 灵感: use merge sort concept, divide and conquer:
- divide the elements from mid, compare each subarray
- sort once sub-array is completed (so that it can be used recursively at parent level)
- use classic while loop `while(j<=e && nums[i]/2.0 > nums[j])` to count pairs

Segment tree

- TODO
- split the array into index-based segment tree, where each element is at leaf
- store min of range: use max to determine if certain range is needed for further query
- query for each element right side range (i + 1, end), where it recursively query&aggregate sub-range if meeting requirement nums[i] > 2*nums[j]
- only when target > subRange.min * 2: there are possible candidates, query further
- worst case O(n^2) when all tailing elements are meeting requirement.

BST

- TODO
- Build the BST based on node value. It will be not applicable if we search after entire tree is built (our goal is right range), so we need to build right elements, and search/count right after the elements is added
- Worst case is still O(n^2), if all added nodes are meeting requirement
- search(tree, curr / 2.0)

O(n^2)

- check each one of them

---

224. Kth Largest Element in an Array.java### Level: Medium Tags: [Divide and Conquer, Heap, MinHeap, PriorityQueue, Quick Sort]

kth largest in array

PriorityQueue, MinHeap

-Need to maintain k large elements, where the smallest will be compared and dropped if applicable: -Find a low> pivot, high <pivot, and you can swap. -Maintain k elements with min value: consider using minHeap -add k base elements first -Maintain MinHeap: only allow larger elements (which will squzze out the min value)-Remove peek () of queue if over size -O (nlogk)

### Quick Sort

-Use part of the Quick Sort partion -after sorting is ascending, then n-k is the kth largest.-The result of the partion is that low, find low == nums.size ()-k , Which is the penultimate K.
-Did not find continued partial recursively. -The process of sorting is to sort a list from small to large. (The same code can also be xth smallest, mid becomes just x) -Steps: -For each iteration, find a pivot, then From low, and high are compared with pivot.
-The obtained low is the current partion point -Overall O (nlogN), average O (n) for this problem.

---

225. [Merge k Sorted Lists.java] (https://github.com/awangdev/LintCode/blob/master/Java/Merge%20k%20Sorted%20Lists.java)### Level: Medium Tags: [Divide and Conquer, Heap, Linked List, PriorityQueue]

Give an array of ListNode, and connect all the nodes into one according to size. .

PriorityQueue

- the Iterative, the PQ to align the leading node list all.
- k lists need to remember the sort that has been well
- time: n * O (logk), where n = total node number, and PriorityQueue: logk,
- Note:
-
    1. Don't forget that customized priority requires a customized new Comparator ()
- 2 . Given node may also have a null node, don't forget to check.

Divide and Conquer -always merge 2 list at a time

-3 branches: -1. start == end -2. start + 1 == end -3. or start + 1 <end (recursive and keep merging) -T (k) = 2T (k / 2) + O (mk), where m = longest list length - time complexity: O (nklogk)-TODO : write the recursive code.

Followup

-If k is large What if I can't fit all k lists on one machine?

- If the Merge up very long, a fit how to do on a machine?

---

226. [Merge k Sorted Arrays.java](https://github.com/awangdev/LintCode/blob/master/Java/Merge%20k%20Sorted%20Arrays.java) Level: Medium Tags : [Heap, MinHeap, PriorityQueue]

Same as merge k sorted list, use priorityQueue

Priority Queue

-Inspired by Merge k sorted list. Use PriorityQueue to store the k first element -PriorityQueue needs to store units: Build a Class Node yourself to store val, x, y
-Because there is no 'next' pointer in the array, only x, y can be stored to push the next element index.-Not sure why `new PriorityQueue <> (Comparator.comparing (a-> a.val))` ; is slower

---

227. [Heapify.java] (https://github.com/awangdev/LintCode/blob/master/Java/Heapify.java) Level: Medium Tags: [Heap, MinHeap]

Turn unsorted array into a min -heap array, where for each A [i],

A [i * 2 + 1] is the left child of A [i] and A [i * 2 + 2] is the right child of A [i].

## Heap

-Heap is not used much. You have to use it to understand it. Usually, the PriorityQueue of default gives a ready-made min-heap: -All the corresponding elements are smaller than the curr element. -The siftdown part of Heapify:-Only from for (i = n / 2-1 ~ 0), but not from for (i = 0 ~ n / 2 -1): Must bloom in the middle and upward When running, can I ensure that my feet are in line with the rules of

What does Heapify / SiftDown do?-For

Min-heap's judgment rules:

-for each element A [i], we will get A [i * 2 + 1]> = A [i] and A [i * 2 + 2]> = A [i]. sure that the two children under the curr node in the heap datastructure and all the nodes below follow a rule -For example here, if it is min-heap, then the next two children will be older than themselves. If not, swap is required.

-In siftdown: small comparison between curr node and two children. If it is true that curr <child, get it, break while.
-But if curr is not smaller than child, then change the seat, and continue to check from the child's seat down.

---

228. [Top K Frequent Elements.java](https://github.com/awangdev/LintCode/blob/master/Java/Top%20K%20Frequent%20Elements.java) Level: Medium Tags : [Hash Table, Heap, MaxHeap, MinHeap, PriorityQueue]

time: O (n) space: O (n)

gives a string of numbers, finds the top k frequent element, and the time complexity is better than nLogN

HashMap + bucket List []

-Use HashMap to store <num, freq> -Reverse mapping <count, list unique element with that count> in a `bucket = new List [n]`. -Size of the data structure will be m <= n -The bucket [count] preserves order from end of the array. -then priorityQueue, (mLog (m)), where m is the total number of unique numbers - Simply loop over the reversed map, we can find the top k items.

- Solid O (n)

PriorityQueue, MinHeap -Use regualr priorityQueue to sort by frequency ascendingly

- the queue.peek () record has lowest frequency, which is replacable
- Always only maintain k elements in the queue, so sorting is O (logk)
- IMPORTANT: remember to `rst.add (0, x)` for desired ordering
- time faster than maxHeap: O (nlogk)

## PriorityQueue, MaxHeap

-The title has a reminder: it must be better than O (nLog (n)), which means that it must be O (n) -the first thought is PriorityQueue, and it cannot be queue.offer on the fly -then count, O (n), using HashMap -eventually find top k, O (k) -Overall time: O (n) + O (mLogm) + O (k) => O (n), if m is small enough

---

229. [Ugly Number II.java](https://github.com/awangdev/LintCode/blob/master/Java/Ugly%20Number%20II.java) Level: Medium Tags: [DP, Enumeration, Heap, Math, PriorityQueue]

time: O (n) space: O (n)

DP

-curr index is based on previous calculation: the min of all 3 previous factors -O (n)

PriorityQueue, DP

-very brute. -Each time you take out dp [i-1], regardless of whether it is thirty-seven or twenty-one, multiply it by 2,3,5. The result will be put in the priority queue for comparison. -The last time is n * log (n * 3) -Note: use long, use HashSet to ensure that there are no duplicates -O (nlogn)

---

230. [Inorder Successor in BST.java](https://github.com/awangdev/LintCode/blob/master/Java/Inorder%20Successor%20in%20BST.java) Level: Medium Tags: [BST , Tree]

find the next one in the Inorder traversal rule. The

main idea is to consider: 1. If node.right == null, find the previous unprocessed node alone the inorder traversal path 2. If node.right! = Null, successor must be in The node.right subtree can be reduced to a few lines at the end, a very comprehensive BST problem: search, understanding of inorder traversal, and pits.

Short Recursive and Iterative without Stack

-Previous solution, we use stack to hold previous cached / unprocessed items: but do we need use catch to hold them? -If moving left: `p.val <root.val`, then root (parent of left child) is a successor candidate, so save `rst = root`. -If moving right or equal: `p.val> = root.val`, the successor has nothing to do with curr node, so just directly dive into root.right. -Both iterative and recursive solution can be simplified as such.

Previous Iterative + stack

-Iteratively search -Still need stack to store previously unprocessed items along the path

Previous Recursive + Stack

-Draw an inorder graph and discover the rules. There are several cases for the successor (successor) of each node:
-1. node.right is a leaf In the end. Then
return.-2. set rightNode = node.right, but found that rightNode has a lot left children to
leaf.-3. For example, node.right == null, that is, node itself is a leaf, you need to look back at the top of the mountain to find Inorder The next in the traversal rule.
-Discovery: In fact, each layer puts the passing curr node in the stack. The top one is the successor that changed the return now :) Done.

---

231. [Walls and Gates.java] (https: //github.com/awangdev/LintCode/blob/master/Java/Walls%20and%20Gates.java)### Level: Medium Tags: [BFS, DFS]

Give a room 2D grid. Inside there is wall-1, door 0 , And empty space INF (Math.MAX_VALUE).

For each empty space, fill it with dist to nearest gate.

DFS

- Form empty room: it can reach different gate, but each shortest length will be determined by the 4 directions.
- Option1(NOT applicable). DFS on INF, mark visited, summerize results of 4 directions.
- hard to resue: we do not know the direction in cached result dist[i][j]
- Option2. DFS on gate, and each step taken to each direction will +1 on the spot: distance from one '0';
- Through dfs from all zeros, update each spot with shorter dist
- Worst time: O (mn), where entre rooms [] [] are gates. It takes O (mn) to complete the iteration. Other gates be skipped by `if (rooms [x] [y] <= dist) return ;`

BFS

-Exact same concept. Init with `Queue <int []> queue = new LinkedList <int []> ()`

---

232. [Convert Binary Search Tree to Sorted Doubly Linked List.java] (https://github.com/awangdev/LintCode/blob/master/Java/Convert%20Binary%20Search%20Tree%20to% Level: Medium Tags: [BST, DFS , Divide and Conquer, Linked List, Tree]

time: O (n) space: O (1) The

title is a bit complicated to describe, in short: convert BST into a sorted doubly linked list. (In-place)

## Tree, In-order traversal

-I usually do convert BST to sored list: I can understand it by drawing, it is actually in-order traversal -Just doubly link them when you do it.

Topic special features

-use the Node {val, left, right} `from beginning to end, instead of opening a new doubley linked list class

- After understanding, it is simple, traverse all nodes, DFS is easy to do: `left, curr, right` -The problem of extra space is because it requires create new DoublyLinkedNode class: different from `Convert Binary Search Tree to Sorted Doubly Linked List (extra space)` -requires in-place: cannot recreate new node

---

233. [String to Integer (atoi) .java] (https: // github .com / awangdev / LintCode / blob / master / Java / String% 20to% 20Integer% 20 (atoi) .java) Level: Medium Tags: [Math, String]

String

-check sign, leading-0 , overall size> 11, check max / min in Long format -if passed all tests, parseInt ()

regular expression

-if (! str.matches ("[+-]? (?: \ d + (? : \. \ d *)? | \. \ d +) ")). It's a bit

tougher ---

. 234 [Clone Graph.java] (https://github.com/awangdev/LintCode/blob/master/Java/Clone%20Graph.java) Level: Medium Tags: [BFS, DFS, Graph]

to A graph node, each node has a list of neighbors. Copy the entire graph, return new head node. It is

implemented like crawl urls.

Idea

-Use HashMap to mark cloned nodes. -How many nodes can be copied first and how many Then add neighbor -Use `map <oldNode, newNode>` to mark visited

DFS

-Given graph node obj `{val, list of neighbor}`: copy the node and all neighbors -Mark visited using map <oldNode, newNode>-for loop on the each one of the neighbors: map copy, record in map, and further dfs
-once dfs completes, add newNeighbor as neighbor of the new node (get to it via map) -The main idea is: once copied, there is no need to re-copy

BFS

-Copy the root node , then copy all the neighbors. -Mark copied node in map. -Use queue to contain the newly added neighbors. Need to work on them in the future.

---

235. [Permutations. java] (https://github.com/awangdev/LintCode/blob/master/Java/Permutations.java) Level: Medium Tags: [Backtracking, DFS, Permutation]

Recursive: Backtracking

-Given a remaining list: take, or not take -always iterate over full `nums []`, use list.contains () to check if item has been added.- Improvement: maintain list (add / remove elements) instead of 'list.contains'

- Time O (n-!): Visit All Possible outcome
- T (n-) = n-T * (. 1-n-) O + (. 1)

the Iterative: insertion

- insertion method:
-
  1. an element added to the list a
-
  2. Each time you take out each list in rst, create a new list, and then select the position and add the new element
-
  3. When adding a new element, you must insert at each position of the list, and eventually you have to insert the original Add new element to the end of the list
- still O (n!), Because rst insert O (n!) Permutations
- but faster than dfs, because it is less # of checks: no need to check list.size (), No need to maintain remaining list.

Previous Notes

-Use a queue, each time the list of the poll (), add each one that can be added in nums -Time O (n!) -A bit slower, possibly because of the polling and saving the entire list every time

236. [One Edit Distance.java] (https://github.com/awangdev/LintCode/blob/master/Java/One%20Edit%20Distance.java) Level: Medium Tags: [String]

If S, T can become equal with only one operation, return true.

Edit: Delete, add, and replace

-after the replacement , theoretically replaced by String should be congruent -For loop, once different chars are found, judge the three possibilities: insert / delete / replace -insert / delete For 2 strings, the effect is similar -O (n) -See #### Based on 3Sum

---

237. [4Sum.java] (https://github.com/awangdev/LintCode/blob/master/Java/4Sum.java) Level: Medium Tags: [Hash Table]

Based on 2sum

-1. Using the principle of 2Sum, 4Sum is divided into 2Sum. A pair on the left and a pair on the right. Put 2 numbers in each pair.
-2. Take a point, i, as the boundary, and also list all pairs before i as the basis.
-3. Then try to find the appropriate 2nd pair from behind all i + 1.
-Time: O (n ^ 2 * x), where x = # of candidates, still slow -You can use HashSet , you can directly compare each element in the list to ensure that the set is not duplicated. -Previous Notes: Creating classes When pairing, you need to do @ override's function: hashCode (), equals (Object d). Usually I don't think of it.
http://lifexplorer.me/leetcode-3sum-4sum-and-k-sum/-Add

-Add another layer outside 3Sum. See 3Sum. Time O (n ^ 3). But this method is undoubtedly too time-consuming in k-sum. O (n ^ k)

238. [Redundant Connection.java] (https://github.com/awangdev/LintCode/blob/master/Java/Redundant % 20Connection.java) Level: Medium Tags: [BFS, DFS, Graph, Tree, Union Find]

unionFind

-keyword: tree has no `cycle`. -Once two nodes appear in the edge, and The parent is the same, indicating that the two nodes are not union and are also in the same tree, so you can break them.

Graph, DFS

-Add graph using adjacent list, and verify cycle alone the way -IMPORTANT: use `pre` node in dfs to prevent backward dfs -similar to `Graph Valid Tree` where it validates cycle and also needs to validate if all nodes are connected

BFS

-same concept as DFS, find first redundant edge that alreay exists in graph map .

-Another union-find, using hashmap :

239. [Graph Valid Tree.java] (https://github.com/awangdev/LintCode/blob/master/Java/Graph%20Valid%20Tree.java) Level: Medium Tags: [BFS, DFS, Graph, Union Find]

Give a number n for n nodes, marked from 1 ~ n, and a string of undirected edge int [] [].

Check if these edges can form a valid tree

Union Find

-Review Union-Find Another form of the track union size: tree does not have cycle, so eventually union size should == 1-1 . Find if 2 elements are in a union. If it is not, false. If it is, then merge into a set and share the parent.
-2. Verify cycle: `find (x) == find (y) => cycle` : new index has been visited before -the key of storage are: elements relative index kept on his parent root.

- Note: to check the end, if only one union: Tree must be connected to all given the Node.
- 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/ #### DFS -Very similar to `Redundant Connection` -Create adjacent list graph: Map <Integer, List > -Check: -1. Whether there is cycle using dfs, check boolean [] visited -2. Whether all nodes are linked: validate if all edge connected: # of visited node should match graph size -IMPORTANT: use `pre` node to avoid linking backward / infinite loop such as (1)-> (2), and (2)-> (1)

BFS-

(Not done yet, you can (Write and write) -also check: 1. whether there is a cycle, 2. whether all nodes are linked

---

240. [The Maze.java] (https://github.com/awangdev/LintCode/blob/master/Java/The%20Maze.java)### Level: Medium Tags: [BFS, DFS]

BFS

-BFS on coordinates -always attempt to move to end of border

- use boolean[][] visited to alingn with BFS solution in Maze II, III, where it uses Node[][] to store state on each item.

---

241. The Maze II.java### Level: Medium Tags: [BFS, DFS, PriorityQueue]

BFS

- if already found a good/shorter route, skip
- `if (distMap[node.x][node.y] <= node.dist) continue;`
- This always terminates the possibility to go return to original route, because the dist will be double/higher

---

242. Predict the Winner.java### Level: Medium Tags: [DP, MiniMax]

Detailed in `Coins in a Line III`

Priority Queue

---

243. [Group Shifted Strings.java] (https://github.com/awangdev/LintCode/blob/master/Java/Group%20Shifted%20Strings.java) Level: Medium Tags: [Hash Table, String]

Convert to orginal string

-shit by offset. `Int offset = s.charAt (0)-'a';` -increase if less than 'a': `if (newChar <'a') newChar + = 26;`

Previous notes

- Strings with the same shift rule can be calculated to the same zero starting point, that is, subtracting a char together, and finally equal. With this as the key, use HashMap. At a glance.

- Remember to sort String [] at the beginning according to the meaning of the title.

---

244. [Delete Digits.java] (https://github.com/awangdev/LintCode/blob/master/Java/Delete%20Digits.java) Level: Medium Tags: [Greedy, Priority Queue]

-TODO: parse into node (index, digitValue)-find the top k, and remove from char array -O (nlogn) time

Greedy

-The higher the number, the greater the weight. So it is hard to remove the relatively higher one (compared to the following digit).

---

245. [Flatten 2D Vector.java] (https://github.com/awangdev/LintCode/blob/master/Java/Flatten%202D%20Vector.java) Level: Medium Tags: [Design]

Implement an iterator to flatten a 2d vector.

Just move pointers carefully with next (), hashNext ()

Basic Implementation using x, y corrdinate

-Iterates over all elements in the 2D list. -Traversing with an nxn matrix is the same as pulling it; all x, y, change the 2d list.

Always return item at index 0, and remove from list?

-List is convenient for remove, consider reduce input vector (as if it were a linked list)

---

246. [The Spiral Matrix II.java] (https://github.com/awangdev/LintCode/blob/master/Java/The%20Spiral%20Matrix%20II.java) Level: Medium Tags: [Array]

Move forward till end

-Similar concept as `The Maze` : keep walking until hit wall, turn back -fix direction `dx [direction% 4]`
