Table of Contents *generated with DocToc*

- 66. Complete Binary Tree.java Level: Easy Tags: [BFS, Tree]
- 67. Compare Strings.java Level: Easy Tags: [String]
- 68. Contains Duplicate.java Level: Easy Tags: [Array, Hash Table]
- 69. Contains Duplicate II.java Level: Easy Tags: [Array, Hash Table]
- 70. Nim Game.java Level: Easy Tags: [Brainteaser, DP, Game Theory]
- 71. Convert Integer A to Integer B.java Level : Easy Tags: [Bit Manipulation]
- 72. Cosine Similarity.java Level: Easy Tags: [Basic Implementation]
- 73. Count 1 in Binary.java Level: Easy Tags: [Bit Manipulation]
- 74. Count and Say.java Level: Easy Tags: [Basic Implementation, String]
- 75. Paint House.java Level: Easy Tags: [DP, Sequence DP, Status DP]
- 76. Longest Continuous Increasing Subsequence.java Level: Easy Tags: [Array , Coordinate DP, DP]
- 77. House Robber.java Level: Easy Tags: [DP, Sequence DP]
- 78. Find All Anagrams in a String.java Level : Easy Tags: [Hash Table, Sliding Window]
- 79. Count Primes.java Level: Easy Tags: [Hash Table, Math]
- 80. Delete Node in a Linked List.java Level : Easy Tags: [Linked List]
- 81. Excel Sheet Column Number.java Level: Easy Tags: [Math]
- 82. Excel Sheet Column Title.java Level: Easy Tags: [Math]
- 83. Flip Game.java Level: Easy Tags: [String]
- 84. [Implement strStr () .java] (https://github.com/awangdev/LintCode/blob/master/Java/Implement%20strStr ().java) Level: Easy Tags: [String, Two Pointers]
- 85. Last Position of Target.java Level: Easy Tags: [Binary Search]
- 86. Length of Last Word.java Level: Easy Tags: [String]
- 87. Longest Increasing Continuous subsequence.java Level: Easy Tags: [Array , Coordinate DP, DP]
- 88. Maximum Subarray.java Level: Easy Tags: [Array, DFS, DP, Divide and Conquer, PreSum, Sequence DP, Subarray]
- 89. Median.java Level: Easy Tags: [Array, Quick Select, Quick Sort]
- 90. Middle of Linked List.java Level: Easy Tags: [Linked List]
- 91. Singleton.java Level: Easy Tags: [Design]
- 92. Remove Linked List Elements.java Level: Easy Tags: [Linked List]
- 93. Fibonacci.java Level: Easy Tags: [DP, Math, Memoization]
- 94. Palindrome Linked List.java Level: Easy Tags: [Linked List, Two Pointers]
- 95. Reverse Linked List.java Level: Easy Tags: [Linked List]
- 96. Intersection of Two Linked Lists.java Level: Easy Tags : [Linked List]
- 97. Palindrome Permutation.java Level: Easy Tags: [Hash Table]
- 98. Valid Palindrome.java Level: Easy Tags: [String, Two Pointers]
- 99. Implement Stack using Queues.java Level: Easy Tags: [Design , Stack]
- 100. Implement Stack.java Level: Easy Tags: [Stack]
- 101. Invert Binary Tree.java Level: Easy Tags: [BFS, DFS, Tree]
- 102. Maximum Depth of Binary Tree.java Level: Easy Tags : [DFS, Tree]
- 103. Minimum Depth of Binary Tree.java Level: Easy Tags : [BFS, DFS, Tree]
- 104. Symmetric Tree.java Level: Easy Tags: [BFS, DFS, Tree]
- 105. Tweaked Identical Binary Tree.java Level: Easy Tags: [DFS , Tree]
- 106. Merge Two Binary Trees.java Level: Easy Tags: [DFS , Tree]
- 107. Subtree.java Level: Easy Tags: [DFS, Tree]
- 108. Lowest Common Ancestor II.java Level: Easy Tags: [Hash Table, Tree]
- 109. Hash Function.java Level: Easy Tags: [Hash Table]
- 110. Merge Two Sorted Lists.java Level: Easy Tags: [Linked List]
- 111. Missing Number.java Level: Easy Tags: [Array, Bit Manipulation, Math]
- 112. Remove Duplicates from Sorted Array.java Level: Easy Tags : [Array, Two Pointers]
- 113. Remove Duplicates from Sorted List.java Level: Easy Tags : [Linked List]
- 114. Longest Word in Dictionary.java Level: Easy Tags: [Hash Table, Trie]
- 115. Path Sum.java Level: Easy Tags: [DFS, Tree]
- 116. Path Sum II.java Level: Easy Tags: [Backtracking, DFS, Tree]
- 117. Path Sum III.java Level: Easy Tags: [DFS, Double Recursive , Tree]
- 118. Rotate String.java Level: Easy Tags: [String]
- 119. Longest Common Prefix.java Level: Easy Tags: [String]
- 120. Reverse Words in a String III.java Level : Easy Tags: [String]
- 121. Merge Sorted Array II.java Level: Easy Tags: [Array]
- 122. Nth to Last Node in List.java Level : Easy Tags: [Linked List]
- 123. Two Sum.java Level: Easy Tags: [Array, Hash Table]
- 124. Max Area of Island.java Level: Easy Tags: [Array , DFS]
- 125. Subarray Sum.java Level: Easy Tags: [Array, Hash Table, PreSum, Subarray]
- 126. Range Sum Query-Immutable.java Level: Easy Tags: [DP, PreSum]
- 127. Longest Words.java Level: Easy Tags: [Hash Table, String]
- 128. Unique Characters.java Level: Easy Tags: [Array, String]
- 129. Binary Gap.java Level: Easy Tags: [Bit Manipulation]
- 130. Maximize Distance to Closest Person.java Level: Easy Tags : [Array]
- 131. Paint Fence.java Level: Easy Tags: [DP, Sequence DP]
- 132. Best Time to Buy and Sell Stock.java Level: Easy Tags: [Array, DP, Sequence DP]
- 133. [Best Time to Buy and Sell Stock II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Best%20Time%20to%20Buy%20and%20Sell%20Stock%20II .java) Level: Easy Tags: [Array, DP, Greedy, Sequence DP, Status DP]
- 134. Minimum Subarray.java Level: Easy Tags: [Array, DP, Greedy, Sequence DP, Subarray]
- 135. Subtree of Another Tree.java Level: Easy Tags: [DFS , Divide and Conquer, Tree]

Easy (153)

0. [Hamming Distance.java] (https://github.com/awangdev/LintCode/blob/master/Java/Hamming%20Distance.java) Level: Easy Tags: []

bit: XOR, &, shift >>

---

1. [Happy Number.java] (https://github.com/awangdev/LintCode/blob/master/Java/Happy%20Number.java) Level: Easy Tags: []

Basic Implementation of the requirements.

Use HashSet to save the viewed values. If repeated, return false.

---

2. [HashWithArray.java] (https://github.com/awangdev/LintCode/blob/master/Java/HashWithArray.java) Level: Easy Tags: []

---

3. [Heaters.java] (https://github.com/awangdev/LintCode/blob/master/Java/Heaters.java) Level: Easy Tags: []

first step: Question type, it takes time to understand the meaning of the question: Literally and drawing, it is to set the housing one by one, the distance around the house needs to be enough to reach the heater. The goal is to recruit as small a radius as possible, so it is necessary for the house and heater to be close together. Set the house in the for loop, move the heater as an interval, and reach the first suitable interval. This is the smallest ideal radius at the moment. Take this value and compare it with the predetermined radius. After the comparison, continue to move the house, and then try to move the heater interval to match.

The second step: Binary Search

note! The title does not say whether the given array is sorted, we must sort to be able to move between ranges or binary search. TODO: http://www.cnblogs.com/grandyang/p/6181626.html

---

4. [IndexMatch.java] (https://github.com/awangdev/LintCode/blob/master/Java/IndexMatch.java) Level: Easy Tags: []

Ordered, suppose there is such a number: target.
The number to the left of target must not be greater than index, and the number to the right of target must be greater than index.
This allows binary search.O (logn)

---

5. [Insert Node in a Binary Search Tree .java] (https://github.com/awangdev/LintCode/blob/master/Java/Insert%20Node%20in%20a%20Binary%20Se java) Level: Easy Tags: [BST]

Add something to the Binary Search Tree and you will definitely find a suitable leaf to add.

So: That is to say, when someNode.left or someNode.right is null, it is where the insert node is.

Find that someNode according to the normal Binary Search Tree law.

---

6. [Jewels and Stones.java] (https://github.com/awangdev/LintCode/blob/master/Java/Jewels%20and%20Stones.java) Level: Easy Tags: [Hash Table]

1524017454

Give J and S two strings. The character in J is unique jewelry, the character in S contains jewelry and stones. Find how many jewelry in S

Basic HashSet

---

7. [Longest Univalue Path.java] (https://github.com/awangdev/LintCode/blob/master/Java/Longest%20Univalue%20Path.java) Level: Easy Tags: []

Figure out what path means: connect the edge of a node. To find MAX, you can define a max variable in the class scope.

Use the minimum amount of code to summarize several different situations: left == root, right == root, or left == root == right.

---

8. [Matrix Zigzag Traversal.java] (https://github.com/awangdev/LintCode/blob/master/Java/Matrix%20Zigzag%20Traversal.java) Level: Easy Tags: []

Analyze 4 steps: right, left-bottom, down, right-up
Pay attention to the index when implementing. A little patience

---

9. [Minimum Absolute Difference in BST.java] (https://github.com/awangdev/LintCode/blob/master/Java/Minimum%20Absolute%20Difference%20in% Level: Easy Tags : [BST]

BST: inorder-traversal: first left node (adding to stack till left leav), then process stack.peek (mid node), then add rightNode && dive to rightNode.left leaf

---

10. [O (1) Check Power of 2.java] (https://github.com/awangdev/LintCode/blob/master/Java/O (1)% 20Check% 20Power% 20of% 202.java) Level: Easy Tags: [Bit Manipulation]

11. [Partition Array by Odd and Even.java] (https://github.com/awangdev/LintCode/blob/master/Java/Partition%20Array%20by%20Odd%20and%2(Level : Easy Tags: [Array, Two Pointers]

-More normal start / end partition pointer is similar to: when condition meet, swap -Clean up TODO

---

12. [Pascal's Triangle II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Pascal's%20Triangle%20II.java) Level: Easy Tags: []

Simple processing of array list.

---

13. [Permutation Index.java] (https://github.com/awangdev/LintCode/blob/master/Java/Permutation%20Index.java) Level: Easy Tags: []

The opposite of Permutation Sequence. Thoughts are similar.

The title is Easy, after a long thinking, analysis:
Each digit number skips multiple possibilities that are less than the beginning of the number.

Take [6, 5, 2] as an example. We look for 6, 5 and 2 as the first few of the permudation.
Normal ordering, which is the first of the permutation, should be [2, 5, 6]
If you want to change from the first place, 2, to 6, how many possibilities are you going to cross?
Quite simply, just ask: how many numbers are less than 6? (2, 5). Each number becomes head, and has its own set of changes, there are (n-1)! Possibilities.

The practice of this question: each (n-1)! Add up. Note: (n-1) means, how many permutations does the leading number (2,5) bring out, which is not (n-1)!
Well. In this step, calculating the number is simple: (there are several numbers less than 6) × (how many numbers are left after removing the head)!

All the above are sacrificed in order to push 6 to the throne.

So after pushing 6 up, there are still others.

Continue to see 5, 2
6 It is determined that the variable condition of the latter permutation may be [6, 5, 2], and then it may be [6, 2, 5].

Same process, look at the second 5 of the given array, count it as follows:

1. How many numbers are less than 5?
2. Apart from 5, how many numbers can be facillary?
3. The same. Multiply the result by the second step.

Finally, it depends on the last element 2.

After seeing all 6,5,2, add up.
It is [6, 5, 2], all the lives you have stepped on!

My explanation is too vivid. Because it took so long to think ...

---

14. [Recover Rotated Sorted Array.java] (https://github.com/awangdev/LintCode/blob/master/Java/Recover%20Rotated%20Sorted%20Array.java) Level: Easy Tags: [Array]

The meaning of rotate is that there is a point break, and the array from one side is selected and placed on the other side. Rotate in three steps: first half of rotate second half of rotate rotate all

Note that the breakpoint is found first.

---

15. [Reshape the Matrix.java] (https://github.com/awangdev/LintCode/blob/master/Java/Reshape%20the%20Matrix.java) Level: Easy Tags: []

Read the examples to understand the meaning of the questions. Sort out counter case. Basic implementation

---

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

Similar to Reverse Integer. Can use StringBuffer or two pointer reverse head / tail

---

17. [Search Insert Position.java] (https://github.com/awangdev/LintCode/blob/master/Java/Search%20Insert%20Position.java) Level: Easy Tags: []

General binary search. At the end, determine which position to return.

---

18. [Shortest Word Distance.java] (https://github.com/awangdev/LintCode/blob/master/Java/Shortest%20Word%20Distance.java) Level: Easy Tags: []

Find short distance, wordB can be before and after wordA; at the same time, you only need to calculate the distance of a recent up to date. Greedy constantly changes the A / B index and then compares it.

---

19. [Single Number.java] (https://github.com/awangdev/LintCode/blob/master/Java/Single%20Number.java) Level: Easy Tags: []

Bit XOR: When two bits are different, return 1. The title is about to extinguish all recurring numbers and leave the one that appears once.

---

20. [String Permutation.java] (https://github.com/awangdev/LintCode/blob/master/Java/String%20Permutation.java) Level: Easy Tags: []

Store #of occurrences in HashMap, add the first string and subtract the second string. Finally, see if there is any judgment that is not equal to 0.

---

21. [Trailing Zeros.java] (https://github.com/awangdev/LintCode/blob/master/Java/Trailing%20Zeros.java) Level: Easy Tags: [Math]

---

22. [Two Strings Are Anagrams.java] (https://github.com/awangdev/LintCode/blob/master/Java/Two%20Strings%20Are%20Anagrams.java) Level: Easy Tags: []

Method 1: char ascii with count [256]
Pit: don't imagine this is a 26letter lowercase. May not be true.

Method 2: If it is other character encoding, not just utf16-encoding (java char)?
Then continue to do it with strings

---

23. [Valid Sudoku.java] (https://github.com/awangdev/LintCode/blob/master/Java/Valid%20Sudoku.java) Level: Easy Tags: [Enumeration, Hash Table]

Hash Set

-Store visited value with HashSet. -Validate row, col, and block in the nest for loop.
-The validate block uses the growth law of i and j.
-To put it bluntly, i && j is an index that grows from 0 to n. How to use it is flexible. This method solves all operations in the same nest for loop. - int c = 3 * (i% 3) + j% 3; // make use of how i and j increases - int r = 3 * (i / 3) + j / 3;

A bit Slower approach

-I did block validation alone: I saw 4 layers of for when I validated the block. In fact, it is n ^ 2 -Maybe the code is a little more complicated

---

24. [Word Pattern.java] (https://github.com/awangdev/LintCode/blob/master/Java/Word%20Pattern.java) Level: Easy Tags: []

Each char represents a pattern. Use HashMap <char, str>. But that's not enough. If a also matches dog, b also matches dog. For example, pattern = "abba", str = "dog dog dog dog". So the second HashMap <str, char> is the other way around. Make sure that pattern and str correspond one-to-one.

---

25. [Find Anagram Mappings.java] (https://github.com/awangdev/LintCode/blob/master/Java/Find%20Anagram%20Mappings.java) Level: Easy Tags: [Hash Table]

It is relatively simple. Use HashMap to store the index list. Finally, iterate through the array A again, and enumerate all the elements. O (n)

---

26. [Judge Route Circle.java] (https://github.com/awangdev/LintCode/blob/master/Java/Judge%20Route%20Circle.java) Level: Easy Tags: [String]

Simple character checking. In all directions, plus, minus or minus.

---

27. [Island Perimeter.java] (https://github.com/awangdev/LintCode/blob/master/Java/Island%20Perimeter.java) Level: Easy Tags: [Hash Table]

Brutle

-4 walls per grid; -2 for each shared wall (the walls are two sides, -1 * 2) -The final result is just fine.

Hash Table

-Don't think too much about using HashMap. But also think about it: -Store all the blocks connected to each block in the list with the current block as the key. Then you need to convert the 2D coordinates into an index. -At the end of the map, all key-values should have value-key reverse mapping, so it can be eliminated for a long time, and each elimination is a shared wall. -A little optimization: DFS finds all the islands. If the map of island is very large, and the island itself does not play, it is suitable for optimization. -But the overall code is too complicated. It is not recommended to write.

---

28. [Power of Three.java] (https://github.com/awangdev/LintCode/blob/master/Java/Power%20of%20Three.java) Level: Easy Tags: [Math]

method 1: Power of 3: 3 ^ x == n? It means that n / 3 is always divided, and finally it can be equal to 1, then there is n / = 3, check n% 3, and finally see if the result is the method of dividing to 1. Use while loop.

Method 2: If n is power of 3, then x of 3 ^ x must be a number smaller than n. Then you can do binary serach between 0 and n, but it is slower.

Method 3: Ingenious idea. The largest 3 ^ x integer is 3 ^ 19. Then find this number, it must be divisible by n. One step in place.

---

29. [Plus One.java] (https://github.com/awangdev/LintCode/blob/master/Java/Plus%20One.java) Level: Easy Tags: [Array, Math]

Simple implementation, add 1, carry. The only tricky place, if you want one more last, it must be 10000 ... This mode, you can take a shortcut, directly come to an array of +1 size, and then the first bit = 1. Note that converting to long is not reasonable, too much memory is used.

---

30. [Power of Two.java] (https://github.com/awangdev/LintCode/blob/master/Java/Power%20of%20Two.java) Level: Easy Tags: [Bit Manipulation, Math]

Same as powerOfThree: you can loop, check mod; you can also use binary search to find the appropriate number.

---

31. [Reverse Vowels of a String.java] (https://github.com/awangdev/LintCode/blob/master/Java/Reverse%20Vowels%20of%20a%20String.java Level: Easy Tags : [String, Two Pointers]

vowels: vowels. All reverse vowels are required.

Method 1: two pointer.

-Two pointers before and after, run inside the while loop. -Pay attention to i <j. Once you meet, break. -Find the right one, just do swap. -StringBuffer can be used with sb.setCharAt (). -O (n)

Method 2:

Take out all vowels, put them in reverse. O (n)

---

32. [Guess Number Higher or Lower.java] (https://github.com/awangdev/LintCode/blob/master/Java/Guess%20Number%20Higher%20or%20Lowe Level: Easy Tags : [Binary Search]

binary search formula

---

33. [Trim a Binary Search Tree.java] (https://github.com/awangdev/LintCode/blob/master/Java/Trim%20a%20Binary%20Search%20Tree.java) Level: Easy Tags : [BST, Tree]

method 1: Suitable for reviewing BST. Treat each node with DFS. Note the characteristics of BST: all left nodes are smaller than the current node, and all right nodes are larger than the current node.

Use [L, R] to cut according to the meaning of the question. If node.val <L, directly drop the left side of the node, and return node.right. The same is true for R. The division system is, DFS leftNode, rightNode. Then connect to node.left, node.right.

Method 2: Use iteration, not written yet.

---

34. [Array Partition I.java] (https://github.com/awangdev/LintCode/blob/master/Java/Array%20Partition%20I.java) Level: Easy Tags: [Array]

Give a string of numbers, size = 2n, find pairs, and then need to sum of min (pair) max.

(a1, b1), (a2, b2), ..., (an, bn) which makes sum of min (ai, bi) for all i from 1 to n as large as possible.

Sort, basics

-Starting from the result, you only need to find the result of the addition without emphasizing the specific match. -Just write an example -Find the rule of arranging single digits, and then consider the same rule of negative and positive numbers, then you can find the method of permutation. -sort, O (nlogn)

---

35. [1-bit and 2-bit Characters.java] (https://github.com/awangdev/LintCode/blob/master/Java/1-bit%20and%202-bit%20Characters.java) * * Level: Easy Tags: [Array]

method 1: Greedy. Counting from the first bit: If a 1 is encountered, it must be jumped by two digits; if a 0 is encountered, it must be jumped by one digit. loop to end, and see if index reaches the end.

Method 2: I did it with DP hard:

1. If the i-bit is 0, then dp [i-1] or dp [i-2] true is enough.
2. If the i bit is 1, then the i-1 bit must be 1 to satisfy the rule, and dp [i-2] needs to be true.

---

36. [Non-decreasing Array.java] (https://github.com/awangdev/LintCode/blob/master/Java/Non-decreasing%20Array.java) Level: Easy Tags: [Array]

When comparing ascending order, three digits i-1, i, i + 1 must be estimated. Write down the relationship between i-1, i + 1, and then make a reasonable fix.

You need to really fix the array, because loop through will use the number after the fix for comparison.

---

37. [Max Consecutive Ones.java] (https://github.com/awangdev/LintCode/blob/master/Java/Max%20Consecutive%20Ones.java) Level: Easy Tags: [Array]

Basic. Math.max track results. Remember to clear the result object after there is a loop for external operations.

---

38. [Find All Numbers Disappeared in an Array.java] (https://github.com/awangdev/LintCode/blob/master/Java/Find%20All%20Numbers%20Disappeared%20

Level: Easy Tags: [Array]

method 1: Change to the correct position. You need to be careful with handle i, because you don't know what is changed to nums [i], so you must clean it up in place to move on.

Method 2: to mark! Very clever use of the mark method, marked as a negative number, proves visit. Preserve the negative number of the original number, so you can continue to use the absolute value of this negative number to find the position where the original number should be set. Very clever!

Method 3: to mark! Similar to method 2, it is also marked. This time, a number greater than n is added (because the title gives n a border), and finally check it. To not exceed Integer.MAX_VALUE, take the remainder before adding n each time.

Although the method of marking is fast, it is relatively hacky. It is probably not used in regular code.

---

39. [Maximum Average Subarray I.java] (https://github.com/awangdev/LintCode/blob/master/Java/Maximum%20Average%20Subarray%20I.java) Level: Easy Tags: [Array , Subarray]

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

Simply find sum of fixed window k, and at the same time max avg, and the remainder at the end.

---

40. [Largest Number At Least Twice of Others.java] (https://github.com/awangdev/LintCode/blob/master/Java/Largest%20Number%20At%20Least%20Twice Level: Easy Tags: [Array]

Find the maximum value, and the second largest value, and see if it fits the question. Analyze the meaning of the question, the simplest method, you can loop twice: find the most value; compare. But in fact, as a counterexample: if one is not satisfied, it is enough to oppose this 'at least twice of allll others'.

---

41. [Toeplitz Matrix.java] (https://github.com/awangdev/LintCode/blob/master/Java/Toeplitz%20Matrix.java) Level: Easy Tags: [Array]

It seems that there are no algorithmic features, that is, the basic operation of array, and then split into a helper function to do repeated calculations and cut the code. Pay attention to the boundary of check MxN.

---

42. [Sum of Two Integers.java] (https://github.com/awangdev/LintCode/blob/master/Java/Sum%20of%20Two%20Integers.java) Level: Easy Tags: [Bit Manipulation]

a ^ b is: incomplete addition. a & b is: all possible rounds. a & b << 1 is the form of rounding to the left.

Goal: first add a ^ b naked, calculate the carry; then add the result and carry naked, and then calculate the carry of this round; then: naked price, calculate the carry ... until the carry == 0.

So, first record the number of carry: carry. Then a ^ b is not completely added once. Then b is used to put the remaining carry, move one bit at a time, and continue adding until b cycles to 0.

After the first round a ^ b, the meaning of b itself disappeared. The parameter should be renamed next. sum = a ^ b; // sum without adding carries nextCarry = (a & b) << 1;

Substitute a, b for other variable names, which is better understood.

Bit Operation
Steps: a & b: the number of rounds that can occur per bit
a ^ b: the value that each bit may leave in this operation, XOR operation
Each time, the remainder is shifted by 1 digit, and then stored in b. loop until b == 0

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

---

43. [Swap Bits.java] (https://github.com/awangdev/LintCode/blob/master/Java/Swap%20Bits.java) Level: Easy Tags: [Bit Manipulation]

Simple, but a lot of knowledge:

1. Hex 0xaaaaaaaa is 1010101 .... 1010; 0x55555555 is 01010101 0101
2. You can use these two hex to get singular and negative numbers. If you need to take other patterns, you can also do it.
3. x is likely to be a negative number, so right-shift should use logic shift, >>> to avoid leading negative complements.

---

44. [Intersection of Two Arrays II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Intersection%20of%20Two%20Arrays%20II.ja Level: Easy Tags : [Binary Search, Hash Table, Sort, Two Pointers]

method 1: Use HashMap: store a nums1, then check against map with nums2. Time / space: O (n)

Method 2: Binary search? Requires array sorted. Otherwise time O (nlogn) is not worth it. [Not done, wrong]

---

45. [Majority Element.java] (https://github.com/awangdev/LintCode/blob/master/Java/Majority%20Element.java) Level: Easy Tags: [Array, Bit Manipulation, Divide and Conquer]

Vote count

-vote ++, vote--the rest is winner. Time O (n), Space O (1) -Majority Number means more than half. The number of more than half will have at least vote> = 1: match current majority number, vote ++; if not, vote--. -Note: there must be a majority number for the assert valid input. Otherwise this method will not work. [1,1,1,2,2,2,3] is an invalid input, the result is 3, of course it is wrong.

HashMap count occurance

-Time, Space: O (n)

Bit manipulation

-TODO

Related Problems

-Majority Number II, over 1/3, then divided into three parts, countA, countB to calculate the two that appear at most. -Majority Number III, over 1 / k, then naturally divided into k parts. HashMap is used here.

---

46. [Nested List Weight Sum.java] (https://github.com/awangdev/LintCode/blob/master/Java/Nested%20List%20Weight%20Sum.java) Level: Easy Tags: [BFS , DFS]

Give a list of integers, the list may have a nest list. Calculate the total sum. The rule, if it is a nested list, each depth is a depth, sum must be multiplied by depth.

DFS

-New interface to understand: object contains integer or object -Visit all && sum, consider dfs. -bottom-> up is easier: pick nested object and execute dfs, which returns sum of it, add with (level value * weight). -Simple processing of nested structure, dfs increases depth. -time: visit all nodes eventually, O (n), space O (n) -Note1: not multiplying on overall level sum. Only multiply level with single value at this level. -Note2: top-> bottom is not necessary: there is not need of passing added object into next level.

BFS

-bfs, queue, handle queue.size (). -use a level variable to track levels

---

47. [Same Tree.java] (https://github.com/awangdev/LintCode/blob/master/Java/Same%20Tree.java) Level: Easy Tags: [DFS, Tree]

Give two binary trees to see if the two trees are identified.

DFS

-DFS. Determine leaf condition, && with all dfs (sub1, sub2). -I have to walk through all the nodes anyway, so dfs is more suitable and easy to write.

BFS

-Two queues store all current level nodes of each tree. Check equality, check queue size. -Populate next level by nodes at current level.

---

48. [Convert Sorted Array to Binary Search Tree.java] (https://github.com/awangdev/LintCode/blob/master/Java/Convert%20Sorted%20Array%20to%20Binary Level: Easy Tags: [DFS, Divide and Conquer, Tree]

As the title, build balanced BST from sorted array

DFS

-Binary Search Tree Features: The nodes on the left are smaller than the nodes on the right. -height balance, subtree height difference <1, the left and right sub trees must be shared equally. DFS (num, start, end) -At each level, find the middle point, then divide 2 halves, continue with dfs -Divide and Conquer - time / space: O (n), visit all nodes, no redundant visits.

---

49. [Add Digits.java] (https://github.com/awangdev/LintCode/blob/master/Java/Add%20Digits.java) Level: Easy Tags: [Math]

Method 1: The common practice is to add the numbers according to the intent, double-while loop. The first layer of loop is O (n), and then the second layer of loop is a lot less digits, overall O (n)

Method 2: Find the mathematical rule. Every 9 digits, the mod will start to repeat, so you can find the answer indirectly by taking the mod for all numbers. O (1)

---

50. [Valid Anagram.java] (https://github.com/awangdev/LintCode/blob/master/Java/Valid%20Anagram.java) Level: Easy Tags: [Hash Table, Sort]

HashMap

---

51. [Binary Tree Paths.java] (https://github.com/awangdev/LintCode/blob/master/Java/Binary%20Tree%20Paths.java) Level: Easy Tags: [Backtracking, Binary Tree , DFS]

Give a binary tree, return all root-to-leaf path

DFS, backtracking

-Find all paths, bfs / dfs all works. Dfs will be simplier to write -Recursive: Forked.dfs. -top-> bottom: enumerate current node into the list, carry to next level, and backtrack -top-> bottom is trivial to consider: path flows from top-> bottom

DFS, bottom-> up

-We can also take current node.left or node.right to generate list of results from the subproblem -let dfs return list of string candidates, and we can run pair the list with currenet node, once they come back. -TODO: can write code to practice

Iterative

-Iterative, a non-recursive exercise -Because the list needs to be shortened each time, a Stack is added to store the level -This problem is simpler with dfs, because the path from beginning to end is found, which is the pattern of dfs

---

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

Two Pointer: Slow Fast Pointer

-O (1) sapce: use fast and slow pointers. One runs .next, one runs .next.next. Once, fast will catch up with slow because of cycle. -At that time, slow.val = fast.val.

Hash Table

-O (n) space: Use HashMap, always add elements. If there are duplicates, then obviously there is Cycle

---

53. [Min Stack.java] (https://github.com/awangdev/LintCode/blob/master/Java/Min%20Stack.java) Level: Easy Tags: [Design, Stack]

Double Stack: One normal stack, and the other minStack stores the current minimum level. Note the maintenance of changes in minStack

In addition, if you want maxStack, it is similar

---

54. [Implement Queue using Stacks.java] (https://github.com/awangdev/LintCode/blob/master/Java/Implement%20Queue%20using%20Stacks.jav Level: Easy Tags: [Design , Stack]

Double Stack

Draw a picture, know that the last stack is the reverseStack: pop (), peek (), empty () are all on this stack, no transformation is required. Push () does the stack and reverseStack dumps back and forth. Compared to the old code, dumping in PUSH is easier to read.

Previous notes

Double Stack. One is equal to queue and the other is backfillStack. Tricky: It is backfilled during pop () and peek (), and waits until the stack is used up before backfilling. Write an example to know that if you backfill early, stack.peek () is not the head of the queue.

---

55. [Reverse Integer.java] (https://github.com/awangdev/LintCode/blob/master/Java/Reverse%20Integer.java) Level: Easy Tags: [Math]

method 1

Add x% 10 each time, then x keeps decreasing ~ 0 Pay attention to handling MAX_VALUE, MIN_VALUE The symbol is not important, it is processed directly, and it is also retained.

Method 2

Convert to String and then reverse Space O (n), time O (n)

---

56. [Sqrt (x) .java] (https://github.com/awangdev/LintCode/blob/master/Java/Sqrt (x) .java) Level: Easy Tags: [Binary Search, Math]

s- qrt (int x)

-Understand the meaning of the question, find a value that can be m * m = x from [0, x]. -Note, if you ca n't find it, ask the examiner what value to return: It makes sense because return int will be rounded, so returning a square up to x is fine. -Note the use of long for mid, as it is likely to exceed the maximum int.

sqrt (double x)

-Bisection float number, the end should be defined by precision. -Still dichotomy, but the judgment condition becomes: while (end-start> eps) -eps = 1e-12, i.e. accuracy to 1e-12

---

57. [First Bad Version.java] (https://github.com/awangdev/LintCode/blob/master/Java/First%20Bad%20Version.java) Level: Easy Tags: [Binary Search]

Binary Search

According to the nature of isBadVersion, determine how to end = mid or start = mid.
isBadVersion is directional. One point is wrong, and the other is wrong.

---

58. [Meeting Rooms.java] (https://github.com/awangdev/LintCode/blob/master/Java/Meeting%20Rooms.java) Level: Easy Tags: [PriorityQueue, Sort, Sweep Line]

-Pay attention to the joint points to take into account the situation of all meetings, do not accidentally miss the joint points -The meeting was Superman. Move instantly to the next meeting

method 1:

Find if there is an overlap. PriorityQueue After sorting according to start time, compare current and peek: current.end> peek.start?

Method 2: Sweep line

-class Point {pos, flag}, PriorityQueue sort. Count -Is a type of problem with Number of Airplanes in the Sky

---

59. [Binary Tree Inorder Traversal.java] (https://github.com/awangdev/LintCode/blob/master/Java/Binary%20Tree%20Inorder%20Traversal.java) Level: Easy Tags: [Hash Table, Stack, Tree]

Inorder traverse Binary Tree

Recursive

-Recursive on your own, without helper function -Divide and Conquer, with helper (dfs) method -O (n) time, no extra space

Iterative: Stack

-Add left nodes all the way
-Print curr
-Move to right, add right if possible -O (n) time, O (h) space

Note that stack.pop () must add curr = curr.right after adding the left-most child.

Without moving right, a dilemma is likely to occur: The next round of curr is to find its left-most child, and repeating curr and curr.left repeatedly will infinite loop, always up and down on the left.

HashMap

How?

---

60. [Change to Anagram.java] (https://github.com/awangdev/LintCode/blob/master/Java/Change%20to%20Anagram.java) Level: Easy Tags: [String]

Random title in HackerRank: Give a string, everything in half, see how many characters the two halves will change, can become anagram.

-Cut the two halves into two Strings A and B. Int count [26] respectively, ++,-.-Record the frequency of 26 lower case letters. If all offset, it is anagram.-Note: In the end, count should be divided by 2: different letters, and count will be added to and subtracted from different letter positions, then the calculation is just repeated. So divide by two

-Note: Write your own in HackerRank: Scanner, import java.util, non-static method ... etc.

---

61. [Classical Binary Search.java] (https://github.com/awangdev/LintCode/blob/master/Java/Classical%20Binary%20Search.java) Level: Easy Tags: [Binary Search]

Binary Search Template

-while: start + 1 <end -mid = start + (end-start) / 2; -Compare by mid -Double check start, end.

---

62. [Climbing Stairs.java] (https://github.com/awangdev/LintCode/blob/master/Java/Climbing%20Stairs.java) Level: Easy Tags:

[DP, Memoization, Sequence DP]

Each step can take 1 or 2 steps, find out how many ways to climb the ladder in total.

Recursive + Memoization

-Recursion is well written, but repeated calculations, timeout. Time: O (2 ^ n) -O (2 ^ n): each n can spawn 2 dfs child, at next level, it will keep spawn. Total 2 ^ n nodes will spawn. -Use global variable int [] memo to help reduce double counting -O (n) time, space

DP

-The principle of addition, the last step is determined by the first two moves: dp [i] = dp [i-1] + dp [i-2] -Basic sequence DP, int [] dp = int [n + 1]; -DP [] is stored as 1-based index -dp [i]: count # of ways to finish -Need to know the status of dp [n], but the maximum coordinate is [n-1], so int [n + 1] -dp [0] often has special status -O (n) space, time

Sequence DP, scrolling array

-[i] only associates with [i-2], [i-1].

- %2
- O (1) space

---

63. [Closest Binary Search Tree Value.java] (https://github.com/awangdev/LintCode/blob/master/Java/Closest%20Binary%20Search%20Tree%20Val
Level: Easy Tags : [BST, Binary Search, Tree]

Give a BST, and a double target, and find the closest number.

Recursive

-when less than curr val, consider left -when greater than curr val, consider right -dfs to the end, then compare each layer, then return

Binary Search

-Records found closest -Binary Search, according to current node position, -Find node.val == target, or finish walking, return closest

---

64. [Binary Tree Preorder Traversal.java] (https://github.com/awangdev/LintCode/blob/master/Java/Binary%20Tree%20Preorder%20Traversal.java
Level: Easy Tags: [BFS , DFS, Stack, Tree]

Recursive

-Add root, left, then right. Obvious -Divide and conquer -Actually no helper function is needed

Iterative

-First add root, then push the bottom of the stack (root.right) at the end of the process, then push root.left -Stack: push curr, push right, push left.

---

65. [Closest Number in Sorted Array.java] (https://github.com/awangdev/LintCode/blob/master/Java/Closest%20Number%20in%20Sorted%20Arra

Level: Easy Tags : [Binary Search]

-A variant of Binary Search, LintCode can't run any further.-Consider mid-1, mid + 1.-Once there is no mid = target.index. Then the target will eventually narrow down at (mid-1, mid) or (mid, mid + 1)

---

66. [Complete Binary Tree.java] (https://github.com/awangdev/LintCode/blob/master/Java/Complete%20Binary%20Tree.java) Level: Easy Tags: [BFS, Tree]

A complete binary tree is a binary tree in which every level, except possibly the last,

is completely filled, and all nodes are as far left as possible

BFS

-When a node with null children appears for the first time, the leaf level is reached, mark flag = true; -From now on, queues should no longer have nodes and children; left / right children of nodes appearing behind the queue should all be null -Otherwise there is a problem, return false;

---

67. [Compare Strings.java] (https://github.com/awangdev/LintCode/blob/master/Java/Compare%20Strings.java) Level: Easy Tags: [String]

See if StringA includes all StringB characters.

Basic Implementation

-Compare sizes, null.-Then use int [] to count chars from A, count [x] ++. Then compare chars in B, count [x]--If count [c] <0, then false.-O (n)

---

68. [Contains Duplicate.java] (https://github.com/awangdev/LintCode/blob/master/Java/Contains%20Duplicate.java) Level: Easy Tags: [Array, Hash Table]

Unordered array, find if there are duplicate elements, return true / false.

HashSet

-No brain: HashSet.-Time O (n), Space O (n)

Sort, Binary Search

-Arrays.sort (x): Time O (nLogN), Space O (1) -After sorting, the repeating numbers will be sorted together, then binary search

---

69. [Contains Duplicate II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Contains%20Duplicate%20II.java) Level: Easy Tags: [Array, Hash Table]

Unsorted array, find out if there are duplicate elemenets: the necessary condition is that the size of the index i, j of these two elements differ by at most k.

HashSet

-Very cleverly control the value in HashSet to [i-k, i] according to the conditions of k range -Each time you add new elements to the set, subtract the element at the end of the index from the set -Set.add (x) will return false if it encounters duplicates. -Once there is a repetition in this range of length k, the conditions are met. -Time O (n)

HashTable <value, List of duplicates>

-Record the index of each element value in the list -Once there are duplicate element repeats, put the entire list of indexes out, and check if there are any matching conditions: (index-i) <= k -Time O (nm), m = # of duplicates

The difference between these two approaches is artistic

-Method 1 is to limit the selection of selected dates, and remove them if they are not qualified, then once there are duplicates, then it must be certain, and the rest will not be viewed. -Method 2 is to find the index that meets the conditions and process it centrally, but all candidates will be selected -It 's like recruiting people: one is to stop when you are good; the second is to see everyone and choose the best one. Obviously the first is faster.

---

70. [Nim Game.java] (https://github.com/awangdev/LintCode/blob/master/Java/Nim%20Game.java) Level: Easy Tags: [Brainteaser, DP, Game Theory]

Brainteaser

-Famous Nim games -Write some and find that the situation after n = 4,5,6,7,8 ... etc is regular: Whoever gets 4 first loses. -It is very simple in the end n% 4! = 0, time, space O (1)

DP

-Formally and regularly, just like coins in a line, do it first hand first -Can roll array to optimize space -Time O (n), of course, this question will timeout, you can use braineaser to write the result.

---

71. [Convert Integer A to Integer B.java] (https://github.com/awangdev/LintCode/blob/master/Java/Convert%20Integer%20A%20to%20Integer%

Level : Easy Tags: [Bit Manipulation]

How many bits do I need to change to convert Integer A to Integer B?

Bit Manipulation

-a ^ b shows the digits with different binary codes in the bit format. -Each time (a ^ b) >> i moves by i, then & 1 actually means that the number is left. -count -It's practical ^ find different bits, >> shift, & 1 mask

---

72. [Cosine Similarity.java] (https://github.com/awangdev/LintCode/blob/master/Java/Cosine%20Similarity.java) Level: Easy Tags: [Basic Implementation]

According to the formula of Cosine Similarity, basic implementation

---

73. [Count 1 in Binary.java] (https://github.com/awangdev/LintCode/blob/master/Java/Count%201%20in%20Binary.java) Level:

Easy Tags: [Bit Manipulation]

count how many 1 in a 32-bit number binary format

Bit Manipulation

-shift >> i -apply mask & 1

Convert to string O (n) space

You can put integer-> string-> char array.

---

74. [Count and Say.java] (https://github.com/awangdev/LintCode/blob/master/Java/Count%20and%20Say.java) Level: Easy Tags: [Basic Implementation, String]

Introduce a method of counting numbers, and then read the result of the previous line for each line, and calculate it line by line. Ask what is the nth line?

Basic Implementation

-Mainly because the meaning of the question is difficult to understand, very misleading. When the question is understood, there are actually no algorithm requirements. -Count duplicates and print

---

75. [Paint House.java] (https://github.com/awangdev/LintCode/blob/master/Java/Paint%20House.java) Level: Easy Tags: [DP, Sequence DP, Status DP]

time: O (nm), m = # of colors space: O (nm)

To paint n houses, and the cost [] [] of nx3. Find the minimum cost to paint all houses.

Sequence DP

-Find the min cost of dp [i], but do not know what color the last house chooses, then iterate through the colors of the last house (i-1) -While selecting the color of the last house, find the lowest cost according to the color of dp [i-1] / cost + cost [i-1] -Consider the last position of DP (color selection): color status needs to be attached to DP [i]: define a two-dimensional array, one of which is status -dp [i] [j]: the minimum cost of painting the j color in the first i houses. -dp [0] [j] = 0: 0th house, no cost -Calculation order: Counting from each house [0 ~ n], first for loop -Then select the color of the ith house, and then select the color of the (i-1) th house. Double for loop, skip same color

Rolling Array

-Observe that index [i] is only related to [i-1], so 2 digits are sufficient,% 2

---

76. [Longest Continuous Increasing Subsequence.java] (https://github.com/awangdev/LintCode/blob/master/Java/Longest%20Continuous%20Increasing%20Sub Level: Easy Tags: [Array , Coordinate DP, DP]

Find the length of the continuous continuous rising subsequence.

Coordinate DP

-1D coordinate, the subscript of dp, is the state of index i -Find the maximum value, dp [i] = longest subsequence at index i -If nums [i]> nums [i-1], dp [i] = dp [i-1] + 1 -If it does not continue to rise, then dp [i] = 1, repeat -maintain max

Basic

-Use a number to store current count, maintain max

---

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

time: O (n) space: O (n) or rolling array O (1)

Search for houses, the adjacent ones cannot touch. Each house has value, find max.

Sequence DP

-dp [i]: max gain from the first i house -Look at the previous one or two of the last ending state, and then consider the current situation -Find out the relationship between the current [i] and the previous [ix] situation: it is not possible to connect the house, then consider the situation of dp [i-2] directly - Sequence DP, new dp [n + 1];

Rolling Array

-[i] 'is only relevant for the first two seats [i-1], [i-2]' -Mark [i], [i-1], [i-2] with% 2. -Others use curr / prev to represent coordinates when scrolling. Here% 2 is more abstract, but more practical.

---

78. [Find All Anagrams in a String.java] (https://github.com/awangdev/LintCode/blob/master/Java/Find%20All%20Anagrams%20in%20a%20Stri Level : Easy Tags: [Hash Table, Sliding Window]

Much like Permutation in String. Give short string p, long string s.

Find the starting index of all p's anagram (permutation) in s.

HashTable

-count character apperance -Note the tricks of countS, countP: only O (26) for comparison -Overall timeO (n) -Be careful not to use an int [] count to technically check 0, the complexity is O (n)

---

79. [Count Primes.java] (https://github.com/awangdev/LintCode/blob/master/Java/Count%20Primes.java) Level: Easy Tags: [Hash Table, Math]

Count: all prime numbers less than n.

Prime number definition

-> = 2 has no common divisors other than itself and 1.
-There is another way to define: this n, is there an i less than n, and achieve: i * i + # of i = n. If there is, it is not prime

Steps

-A boolean bar, isPrime []. Then from i = 2, all become true. -hash key: the number itself -Then use the nature of this factor, non-prime meets the conditions: self * self, self * self + self ... etc.
-So check every j, j + i, j + i + i, and mark all non-prime as false.
-Finally, just count the remaining true numbers.

---

80. [Delete Node in a Linked List.java] (https://github.com/awangdev/LintCode/blob/master/Java/Delete%20Node%20in%20a%20Linked%20Li Level : Easy Tags: [Linked List]

Given Singlely linked list, delete an arbitrary node (cannot be a head node)

Basic

-update node.val -Link curr.next to curr.next.next

---

81. [Excel Sheet Column Number.java] (https://github.com/awangdev/LintCode/blob/master/Java/Excel%20Sheet%20Column%20Number.java) Level: Easy Tags: [Math]

Math

-26-bit operation, thinking based on 10-bit operation -'A'-'A' = 0. So char-'A' + 1 = corresponding digits in the title -Or: 26-bit operation is the same as 10-bit: num + = digit per digit * Math.pow (26, number number)

---

82. [Excel Sheet Column Title.java] (https://github.com/awangdev/LintCode/blob/master/Java/Excel%20Sheet%20Column%20Title.java) Level: Easy Tags: [Math]

Basic Conversion

-26 bits -From the end, mod% 26 can get the last number remain = n% 26 -Special: When remain = 0, which means n is a multiple of 26, the end should be 'Z' -After recording 'Z', n--

---

83. [Flip Game.java] (https://github.com/awangdev/LintCode/blob/master/Java/Flip%20Game.java) Level: Easy Tags: [String]

String

-You can use sb.replace (i, j, "replacement string") -Simply press window = 2 to scan -Turned from '++' to '-' -O (n)

---

84. [Implement strStr () .java] (https://github.com/awangdev/LintCode/blob/master/Java/Implement%20strStr () .java) Level: Easy Tags: [String, Two Pointers]

Give two strings A, B, find one B at the beginning of A.

Two Pointer

-Find the starting position of B in A, and see if the substring from this point is equal to B.-Quite a lot of pits, these can help optimize: -1. When B is "", that is, B can be found in the actual position of A .... index = 0.-2. edge condition: if haystack.length () <needle.length (), it must be wrong, return -1 -3. If the remaining length after a certain index, A is shorter than the length of B, it is also a misunderstanding, return -1

---

85. [Last Position of Target.java] (https://github.com/awangdev/LintCode/blob/master/Java/Last%20Position%20of%20Target.java) Level: Easy Tags: [Binary Search]

Give a sorted integer array, find the last index where the target appears. There are duplicate numbers in the array

There are duplicates, not the end point, continue binary search

---

86. [Length of Last Word.java] (https://github.com/awangdev/LintCode/blob/master/Java/Length%20of%20Last%20Word.java) Level: Easy Tags: [String]

Give a String with lower case character and ''. Find the length of the last single word

basics

-Look for '' from the end and find the calculated length -Remember to s.trim (), remove the leading and trailing spaces

---

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

https://leetcode.com/problems/longest-continuous-increasing-subsequence/description/

O (n) runs for 2 times. O (1) uses two ints to store: every time it reaches the point i, the point i meets the condition or does not satisfy all the longestIncreasingContinuousSubsequence. Features: One run back, the ans will continue to be compared with the left ans; what is the maximum value in all cases.

---

88. [Maximum Subarray.java] (https://github.com/awangdev/LintCode/blob/master/Java/Maximum%20Subarray.java) Level: Easy Tags: [Array, DFS, DP, Divide and Conquer, PreSum, Sequence DP, Subarray]

time: O (n) space: O (n), O (1) rolling array

Give a list of arrays, unsorted, can have negative / positive num. Find the maximum of the sum of the numbers in the subarray in the middle of the array

Sequence DP

-dp [i]: the maximum sum of the first i elements, including last element (i-1), the possible subarray.-init: dp = int [n + 1], dp [0]: first 0 items, does not have any sum -Because of the continuous sequence, when the condition is not met, it will break. That is: need to take curr num, regardless => can drop prev max in dp [i] -track overall max -init dp [0] = 0; max = MIN_VALUE because there are negative numbers -Time, space O (n) -Rolling array, space O (1)

Divide and Conquer, DFS

-Find a mid piont, consider 3 cases: only the left, only the right, cross-mid -left / rigth case, direct dfs -corss-mid case: continuous sum max from left + continous sum max from right + mid -continuous sum max from one direction:

---

89. [Median.java] (https://github.com/awangdev/LintCode/blob/master/Java/Median.java) Level: Easy Tags: [Array, Quick Select, Quick Sort]

Given an unordered array, find median (the number in the middle after sort).

Quick Select

-Same as the template for kth largest element in an Array. -Different from quickSort, it only needs to recurring in half of the list each time, so the time complexity of O (logn) is reduced to O (n) -quickSelect finds the smallest kth element -Using this principle, find the minimum value of kth, then if == target index, we find our median -Quick select template to be familiar with, you may want it all at once, but you ca n't write it -Main steps: partition, dfs, only recur on one part of the array

---

90. [Middle of Linked List.java] (https://github.com/awangdev/LintCode/blob/master/Java/Middle%20of%20Linked%20List.java) Level: Easy Tags: [Linked List]

Find the middle node of the Linked List

-Fast and slow hands -Don't care if slow is in the end, because fast must come first. -Make sure fast, fast.next is not Null

---

91. [Singleton.java] (https://github.com/awangdev/LintCode/blob/master/Java/Singleton.java) Level: Easy Tags: [Design]

Let a class be a singleton

---

92. [Remove Linked List Elements.java] (https://github.com/awangdev/LintCode/blob/master/Java/Remove%20Linked%20List%20Elements.java) Level: Easy Tags: [Linked List]

Remove all targets from the linked list

Basics

-If match: node.next = head.next; -If not match, node and head move together

---

93. [Fibonacci.java] (https://github.com/awangdev/LintCode/blob/master/Java/Fibonacci.java) Level: Easy Tags: [DP, Math, Memoization]

Memoization

-fib [n] = fibonacci (n-1) + fibonacci (n-2);

DP array.

-Scrolling array, simplified DP

recursively calculate

-recursively calculate fib (n-1) + fib (n-2). The formula is fine, but the time is too long, timeout.

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

Reverse Linked List

-The Palindrome concept is very simple, but it is difficult to get random access coordinates in the Linkde List: so half of the ListNode needs to be flipped -reverse linked list: traverse the beginning -Use the speed to find the mid point -Time O (n), and does not require additional space (just reverse the internal order of half a list), so space O (1)

Previous Note

-Palindrome must be equal on both sides -linkedlist cannot reverse iterating, then reverse the list, bloom from the middle for comparison.

---

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

Reverse List

-Basic operation of Linked List: every insert at the beginning -Use head to cycle through all nodes -No additional space required -Time O (n), Space O (1)

---

96. [Intersection of Two Linked Lists.java] (https://github.com/awangdev/LintCode/blob/master/Java/Intersection%20of%20Two%20Linked%20List Level: Easy Tags : [Linked List]

For two linked lists, ask which node starts, and the two linked lists start to overlap?

Basics

-Length list, find overlap -If the length is different, cut the extra length of the long list -After the starting point is the same, the coincidence point will arrive at the same time -Time O (n) * 2, constant space

---

97. [Palindrome Permutation.java] (https://github.com/awangdev/LintCode/blob/master/Java/Palindrome%20Permutation.java) Level: Easy Tags: [Hash Table]

For String, see if the permutation can be Palindrome

Hash, or ASCII array

-count occurrance -Only one odd # appearance can be accepted. -Consider all 256 ASCII codes, if you want to expand, use HashMap <Character, Integer> -Note, cannot assum lower case letter. It should be at least all ASCII codes

---

98. [Valid Palindrome.java] (https://github.com/awangdev/LintCode/blob/master/Java/Valid%20Palindrome.java) Level: Easy Tags:

[String, Two Pointers]

Verify that the string is palendrome. Only alphanumeric is considered, other characters can be ignored

Check Palindrome

-Two pointers before and after, move to the middle to see if the letters overlap

filter alphanumeric

-You can use ASCII code to filter manually, as long as it is between '0' ~ '9', 'a' ~ 'z', 'A'-'Z' -You can also use regular expression: match all these letters, which is [a-zA-Z0-9] -Any match of these letters is the opposite: "[^ a-zA-Z0-9]". Test: https://regex101.com/

---

99. [Implement Stack using Queues.java] (https://github.com/awangdev/LintCode/blob/master/Java/Implement%20Stack%20using%20Queues.jav Level: Easy Tags: [Design , Stack]

As the title.

Queue, pouring water

-Two Queues, pouring water interactively -Swap with a Temp

Practice 1

-The logic is in push: -1. x put q2. -2. q1 all offer / append to q2. -3. Use a Temp for swap q1, q2. -The head of q1 is always the last value added.

Practice 2

-The logic is in top () / pop (), every time you change the water, check the last item.

---

100. [Implement Stack.java] (https://github.com/awangdev/LintCode/blob/master/Java/Implement%20Stack.java) Level: Easy Tags: [Stack]

Just use a data structure, implement stack.

Stack, first in, last out

-ArrayList: return / remove the last item of the ArrayList. -2 Queues

---

101. [Invert Binary Tree.java] (https://github.com/awangdev/LintCode/blob/master/Java/Invert%20Binary%20Tree.java) Level: Easy Tags: [BFS, DFS, Tree]

DFS

-Simple handling of swap -recursively swap children

BFS

-BFS with Queue -Process one node at a time, swap children; then add child to queue -Until the queue process is complete

---

102. [Maximum Depth of Binary Tree.java] (https://github.com/awangdev/LintCode/blob/master/Java/Maximum%20Depth%20of%20Binary%20Tree Level: Easy Tags : [DFS, Tree]

Give a binary tree, find the deepest depth

DFS

-I have to walk through all the nodes here, so dfs is very suitable -Divide and conquer. -Maintain a maximum value: Math.max (maxDepth (root.left), maxDepth (root.right)) + 1; -Note check root == null

Note

-BFS is doable as well, but a bit more code to write: tracks largest level we reach

---

103. [Minimum Depth of Binary Tree.java] (https://github.com/awangdev/LintCode/blob/master/Java/Minimum%20Depth%20of%20Binary%20Tree Level: Easy Tags : [BFS, DFS, Tree]

BFS

-Shortest path; minimum depth: Think of BFS, check level by level, BFS can make sure you find results faster -depth definition: reach to a leaf node, where node.left == null && node.right == null -BFS using queue, track level.

DFS

-Divide and Conquery a minimum. -Pay attention to processing Leaf's null: when the leaf appears, the leaf is ignored, and the direct return counts as a leaf - Another way to count: use Integer.MAX_VALUE instead of null leaf, this can avoid wrong counting. (Can't directly recursive) -This will take all nodes anyway, so dfs should be more suitable.

---

104. [Symmetric Tree.java] (https://github.com/awangdev/LintCode/blob/master/Java/Symmetric%20Tree.java) Level: Easy Tags: [BFS, DFS, Tree]

Check if tree is symmetric

Note the example and definition of Symmetric Binary Tree: mirror-like symmetry. It is not that the left and right sub-trees are equal.

DFS

-Recursively check symmetrically corresponding Node.
-The children of each node and the children of the node opposite to the other side of the mirror are exactly mirror reflection positions.

Stack

-stack1: Left-hand sub-tree is added first, then right child; -stack2: Right-hand sub-tree is added with right child first, then left child.
-During the process, if symmetric, all the nodes in the stack will correspond one by one.

105. [Tweaked Identical Binary Tree.java] (https://github.com/awangdev/LintCode/blob/master/Java/Tweaked%20Identical%20Binary%20Tree.java Level: Easy Tags: [DFS , Tree]

Check if the binary tree is identified.

Features: If the subtree has rotation, as long as the tree node values are equal, it can be considered as an identifier.

DFS

-Based on DFS, compare left and right, left and right, left and right

---

106. [Merge Two Binary Trees.java] (https://github.com/awangdev/LintCode/blob/master/Java/Merge%20Two%20Binary%20Trees.java) Level: Easy Tags: [DFS , Tree]

DFS

-Basic binary tree traversal. Pay attention to the judgment of null child

---

107. [Subtree.java] (https://github.com/awangdev/LintCode/blob/master/Java/Subtree.java) Level: Easy Tags: [DFS, Tree]

Give a binary tree s, and a binary tree t, check if t is a subtree of s.

DFS

-It's very similar to the identification of binary tree -Compare same tree is required only when current s.val = t.val. -In other cases, continue to recursively isSubtree -Note: Even if T1 == T2 is found, it is likely that the numbers are the same (here is not a binary search tree !!), and children are different -So continue to recursively isSubtree (T1.left, T2) ... etc.

---

108. [Lowest Common Ancestor II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Lowest%20Common%20Ancestor%20II.java) Level: Easy Tags: [Hash Table, Tree]

Give a Binary Tree root, and two nodes A, B. Features: The parent pointer is stored in the node. Find the lowest common ancestor

Hash Set

-This question has a strange place, each node also has a parent, so it can be bottom-up. -save visited in hashset. The first duplicate is the lowest common ancestor of AB

Save in lists

-Bottom-up. Use parent to return to root -Save all parents, then find the last common node in the two lists

Note

-Can't search target node directly from root to make two lists. Because it's not Binary Search Tree at all!

109. [Hash Function.java] (https://github.com/awangdev/LintCode/blob/master/Java/Hash%20Function.java) Level: Easy Tags: [Hash Table]

Hash Function

-Explain how Hash does it. -Hash function example:
-hashcode ("abcd") = (ascii (a) * 33 ^ 3 + ascii (b) * 33 ^ 2 + ascii (c) * 33 ^ 1 + ascii (d) * 33 ^ 0)% HASH_SIZE -Parameters used: magic number 33, HASH_SIZE.

-The meaning of Hash is: give a string key, convert it to a number, so that the size becomes smaller.
-Real implementations also need to deal with collision, may require design hash function, etc.

Reason for% HASH_SIZE at each step

-hashRst = hashRst * 33 + (int) (key [i]);
-hashRst = hashRst% HASH_SIZE;
-The reason is that hashRst will become too big, so it can't be counted and% ...

---

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

As the title

Basics

-Put it small before. Every time than head size -After the while, connect the endless list in one breath.
-At the beginning, a node is built to run, and each time node.next = xxx is stored. Save a dummy. Used to return dummy.next.

---

111. [Missing Number.java] (https://github.com/awangdev/LintCode/blob/master/Java/Missing%20Number.java) Level: Easy Tags: [Array, Bit Manipulation, Math]

Give a string of unique numbers, the numbers are taken from [0 ~ n], unordered, find the first skipped number.

Swap

-Much like First Missing Positive, only one line of code is different. -Swap all numbers to their correct position -The last for loop finds the misplaced index, which is the missing number.

Bit Manipulation

-XOR will only retain bits that are different 1 ^ 0 = 1, but 0 ^ 0, 1 ^ 1 == 0 -Use that feature, XOR all values with index -The remaining excess numbers are actually the index that cannot be eliminated by XOR, that is, the missing number value. -Note: The title tells the number is [0 ~ n], but missing a number, then in [0 ~ n-1], the largest number (regardless of whether it is missing) must be n = nums.length.

HastSet

-Save all, looking for missing -O (n) space, unsuitable

sorting

-sort, find 1st missing -O (n log n) is too slow

---

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

Give a sorted array and remove the duplicates: that is, paste the non-repeating in order, the extra positions at the end of the array don't matter.

return unique item length.

Two Pointers

-sorted array, repeating elements are all together -Two pointers can actually be a for loop pointer, another dynamic variable. -track unique index -skip duplicated items -O (n)

Thinking Mode:

-Remove Duplicate from Array is different from remove from linked list. -In LinkedList, it is better not to move node.val, and just remove node. -For the array, it is difficult to remove the node directly, and we cannot use the new array, so we must: -Put non-repeating elements one by one. -This idea is similar to merge two sorted array (one of the following very long array can put arr1, arr2). -Just find an element that will not mess up afterwards, will not move the index, and fill in the elements that meet the conditions. This guarantees in place. -* Reverse thinking *: remove duplicate, actually find unique elements, and insert into original array

---

113. [Remove Duplicates from Sorted List.java] (https://github.com/awangdev/LintCode/blob/master/Java/Remove%20Duplicates%20from%20Sorted%2
Level: Easy Tags : [Linked List]

Remove duplicate elements from the Linked list, leaving only unique elements.

Linked List

-sorted list, duplicate elements are all together -Know how to build a Linked List. -If there is a duplicate element at one point: node.val == node.next.val, remove it. -Run with a dummy node -Note: -Node = node.next only if there is no duplication; -When there is a repetition, after the third element is brought up, it may still be the same as the current element, so it cannot be moved forward. -ex: A-> A-> A -check node and node.next in the while loop are better, so the ending position will be very clear

---

114. [Longest Word in Dictionary.java] (https://github.com/awangdev/LintCode/blob/master/Java/Longest%20Word%20in%20Dictionary.java)
Level: Easy Tags: [Hash Table, Trie]

Give the string word [], find the longest Word, meet the conditions: This Word can be built from word [] letter by letter.

If multiple answers, respect smallest lexicographical order.

Sort, HashSet

-Sort first, you can see if the partial string already exists after sorting -Use set.contains (substring (0, n-1)) to see if the substring in the previous step exists - If found, because it has been sorted alphabetically, the one found must be the most suitable answer in this length. -Then brutally find the next bigger one. - Sort O (n log n), O (n) set space

Trie

-You can sort words Array first: 1. Long string first; 2. Equal length, sort by dictionary order -Put all in Trie. Trie.insert () -For sorted words array, find Trie.startWith from the longest start. -Once found, it is in line with the intent, return directly. -Note: startWith must be isEnd for each node in order to meet the condition of 'spell out letter by letter'. -Time: build Trie O (mn) + sort: O (nlogn) => O (nlogn) -Space: O (mn)

-Sort by size-> compare contains () from the largest-> sort the result in lexicographically. -But Collections.sort () is twice, and list.contains () is slower

---

115. [Path Sum.java] (https://github.com/awangdev/LintCode/blob/master/Java/Path%20Sum.java) Level: Easy Tags: [DFS, Tree]

Give an inputSum, then dfs, find if there is a path, and the resulting path sum is the same as inputSum.

DFS

-Determine the conditions for a good ending: is leaf && val == sum -Minus node.val for each layer, then dfs. -Write a note: The effect of root == null => false on parent nodes. It is found that it has no effect, so it can be simplified to use 1 functionDFS.

---

116. [Path Sum II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Path%20Sum%20II.java) Level: Easy Tags: [Backtracking, DFS, Tree]

Give an inputSum, then dfs, find all paths, satisfy: path sum is the same as inputSum.

DFS, Backtracking

-Use remaining sum to check if input path sum condition is met -Add to result list when satisfied -Two kinds of backtracking: -1. backtrack the current node, add it to the list, and then dfs. After dfs ends, delete the previously added elements. Very clean. -2. Backtrack the next increased dfs level value. After dfs return, delete the last element in the list: but delete the remaining value of dfs. -The first kind of backtrack is better mastered.

Previous Notes:

-A basic problem of Binary Tree: find all paths that meet the conditions -Traverse to the end, compare sum vs. target -Pay attention to divide. Write the traversal example

---

117. [Path Sum III.java] (https://github.com/awangdev/LintCode/blob/master/Java/Path%20Sum%20III.java) Level: Easy Tags: [DFS, Double Recursive , Tree]

Count all existing path sum == target sum. It can start at any point. But only parent-> child.

DFS

-Subtract the given input sum until the sum reaches a target value -Because it can start from any point, when the sum reaches the standard, it needs to continue to recursive, so as to find all cases (with positive and negative numbers, sum may continue to increase / decrease) -Classic helper dfs recursive + self recursive -1. helper dfs recursive handles cases including root -2. self recursive to lead the situation of skip root.

Features

-It is similar to Binary Tree Longest Consecutive Sequence II in recursive approach: -Use dfs for recursive computation including root -Use this function yourself, do recursive computation that does not include root

---

118. [Rotate String.java] (https://github.com/awangdev/LintCode/blob/master/Java/Rotate%20String.java) Level: Easy Tags: [String]

Give two Strings to see if A rotates into B

LeetCode

-Basics -StringBuffer.deleteCharAt (xx), StringBuffer.append (xx) -O (n)

LintCode

-Different problem: give a char [], rotate offset times. -* Three steps rotate * -There is a pit: the offset may be long, so% length is needed to get the part that really needs rotate. -Note: After rotating a full length, the string is unchanged

---

119. [Longest Common Prefix.java] (https://github.com/awangdev/LintCode/blob/master/Java/Longest%20Common%20Prefix.java) Level: Easy Tags: [String]

Find the longest public prefix in a string.

Sort, compare string

-Sort O (nlogn) -first and last string should share common prefix -This assumes that the title requires a common prefix for all strings, not some strings

Brutle

-Nested loop, each time compares all strings for equality -Equal, append string. Not equal, return. -O (mn)

---

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

Give a String, the Word inside is separated by single space, the purpose is to reverse all Word, but preserve Word and space order.

Reverse function

-Downgrade of Reverse Words in a String II, just remove the first overall reverse

---

121. [Merge Sorted Array II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Merge%20Sorted%20Array%20II.java) Level: Easy Tags: [Array]

As the title, merge two sorted array into new sorted array

-Length is fixed. Basic Implementation -If an array is large enough, merge into this array, then merge from the end.

---

122. [Nth to Last Node in List.java] (https://github.com/awangdev/LintCode/blob/master/Java/Nth%20to%20Last%20Node%20in%20List.ja Level : Easy Tags: [Linked List]

Linked List

-Find nth node first -Then head started running -node to the end, and head ~ node is exactly n distance away. So head is the last nth

---

123. [Two Sum.java] (https://github.com/awangdev/LintCode/blob/master/Java/Two%20Sum.java) Level: Easy Tags: [Array, Hash Table]

HashMap <value, index>

-Relatively violent and concise: find a value, store an index -If the result is matched in the HashMap, the index stored in the HashMap is returned. -O (n) space && time.

Sort array, two pointer

-Before and after ++, --Search. Sort takes O (nlogn).
-1. The first two pointers look for value.
-2. Note that you must use the extra space to reserve the original array and use it to find the index. (HashMap cannot be used here because the value is used as the key, but the value may be duplicated)
-O (n) space, O (nlogn) time.

---

124. [Max Area of Island.java] (https://github.com/awangdev/LintCode/blob/master/Java/Max%20Area%20of%20Island.java) Level: Easy Tags: [Array , DFS]

DFS

-Although Easy, the basic idea of DFS is used. -1. dive deep -2. mark VISITED -3.sum it up -Time: worst O (mn), traverse all possible nodes

-Pay more attention to starting dfs from the place where the value == 1 is met. -For situations where there is no island, area should be 0, not Integer.MIN_VALUE. Ask the examiner's guy, don't write it down.

---

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

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

To a string of numbers, to find a subarray therein [start, end] index, condition: subarary sum == 0.

Hash Table

-Simple version of subarray sum equals k: k = 0 -Find preSum, then keep checking map.containsKey (preSum-k). -If priorSum = preSum-k == 0, it means that [priorSum.index + 1, curr index] is the paragraph we are looking for

Previous notes, same preSum + map solution

-Analyze that if sum [0 ~ a] = x, then sum [0 ~ b] = x, then sum [a + 1 ~ b] == 0 -Use hashMap to store the value of each sum [0 ~ i] and index i. If there are duplicates, find a set of sum 0 arrays.

---

126. [Range Sum Query-Immutable.java] (https://github.com/awangdev/LintCode/blob/master/Java/Range%20Sum%20Query%20-%20Immutable.java) Level: Easy Tags: [DP, PreSum]

Given a string of numbers, find sumRange.

PreSum

-Is the definition of pre sum -preSum is also the simplest form of dp []. -dp [i], preSum [i]: the sum of the first (i-1) elements.

---

127. [Longest Words.java] (https://github.com/awangdev/LintCode/blob/master/Java/Longest%20Words.java) Level: Easy Tags: [Hash Table, String]

Give a string of Strings, find the longest length, and return the longest Strings all

Hash Table

-<Integer, List > -Store the longest value, and finally map.get (max)

---

128. [Unique Characters.java] (https://github.com/awangdev/LintCode/blob/master/Java/Unique%20Characters.java) Level: Easy Tags: [Array, String]

determine if characters are unique in string

HashSet

-space O (n), time O (n)

char []

-space O (n), time O (nlogn)

no additional data structure

-double for loop: O (n ^ 2)

---

129. [Binary Gap.java] (https://github.com/awangdev/LintCode/blob/master/Java/Binary%20Gap.java) Level: Easy Tags: [Bit Manipulation]

time: O (n), n = # of bits space: O (1)

Bit Manipulation

-Understand the description of Binary Gap -Simple >>, & 1, track start and end point just fine

---

130. [Maximize Distance to Closest Person.java] (https://github.com/awangdev/LintCode/blob/master/Java/Maximize%20Distance%20to%20Closest%20I
Level: Easy Tags : [Array]

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

For a row of seats, sit alone: find the farthest place (middle point) from the people on both sides, return the maximum distance from the person next to you

It's the same concept of Exam Room, to simplify the problem: just consider one person here.

####? Basic Implementation, track start / end -start / end point, then compare size records dist -Note 1: If there is no one in the first seat, special treatment, dist = [0 ~ end] -Note 2:? If there is no one in the last seat, special treatment: dist = [n-1-start]; -The rest: dist = Math.max (dist, (end-start) / 2) -Related topics: Almost the same concept Binary Gap, upgrade complex version Exam Room

---

131. [Paint Fence.java] (https://github.com/awangdev/LintCode/blob/master/Java/Paint%20Fence.java)
Level: Easy Tags: [DP, Sequence DP]

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

DP

-Up to 2 fences with the same color -Assuming that i is different from i-1, the result is (k-1) * dp [i-1] -Assuming i is the same as i-1, then depending on the conditions, i-1 and i-2 must be different. Then all the results are (k-1) * dp [i-2] -dp [i]: count # of ways to paint -Principle of addition -time, space: O (n) -rolling array: space O (1)

Previous Notes

-This topic is very interesting. The analysis was too complicated at the beginning, and finally I followed the idea of this buddy (http://yuanhsh.iteye.com/blog/2219891), but it was much simpler. -The method of setting T (n) is the same as the Fibonacci number after simplification. The detailed analysis is as follows. -After finishing, I still feel like a god. It was an Easy question, but I couldn't think of it.

---

132. [Best Time to Buy and Sell Stock.java] (https://github.com/awangdev/LintCode/blob/master/Java/Best%20Time%20to%20Buy%20and%20Sell
Level: Easy Tags: [Array, DP, Sequence DP]

Give an array of stock prices, limit one round of trading (buy / buy), and ask how to find the maximum profit.

Understanding meaning is key

-The price is traded every day. Only buy and sell once in n days, then find the lowest price to buy and the highest price to sell. -Record daily minimum value of Min. O (n) -Buy and sell with Min every day, how much profit?

DP

-Find min value for first i items, new dp [n + 1]. -dp [i]: What is the minimum price for the previous i days: min cost of first i days -Then use the price of the day to subtract dp [i] to calculate max profit. -Time, Space: O (n) -Further, use min to represent min [i], because only the current min is needed in the calculation.

Rolling array

-index i only depend on [i-2] -Space O (n)

Brutle Failed

-Try to buy every day, then try to sell every day thereafter. Double for loop, O (n ^ 2). Timeout. -Many of them are unnecessary calculations: [7, 1, 5, 3, 6, 4]. if we know buyin with 1 is cheapest, we don't need to buyin at 5, 3, 6, 4 later on; we'll only sell on higher prices.

---

133. [Best Time to Buy and Sell Stock II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Best%20Time%20to%20Buy%20and%20Sell%20Stock%20II.java) Level: Easy Tags: [Array, DP, Greedy, Sequence DP, Status DP]

time: O (n) space: O (1) greedy, O (n) dp

Difference from Stock I: You can buy and sell multiple times, and find the maximum profit of the sum.

Several other different ideas:

-Greedy, every time there are adjacent diffs that meet the profit criteria, they are sold, and finally all the diffs are added together. Calculating the delta is actually simple and rude, which is not bad. -See below, find peek from the trough, sell. -DP. (Old dp solution BuyOn [], SellOn []) -DFS calculates all (timeout). Improvement on DFS-> DP-> calculate sellOn [i] and buyOn [i], and then return buyOn [i]. It is a bit hard to imagine, but the code is simple and O (n)

Greedy

-Draw, because you can buy and sell indefinitely, as long as there is a rise, there will be profit -All the sales, the translations add up, which is actually overall best profit -O (n)

Find the range with the largest increase, buy and sell:

-Find the trough and buy: when peek = start + 1, you take a step forward each time; if there is no upward trend, continue to the trough. -Rise to the peak and sell: Once there is an upward trend, enter a while loop, go to the end, and add a profit. -profit + = prices [peek-1]-prices [start]; quite special. -When there is no uptrend, peek-1 is also start, so here is exactly profit + = 0.

DP, sequence dp + status

-Want to know the maximum profit of the previous i day, then use sequence DP: -dp [i]: represents the maximum profit for the previous i days -Whether the day can be sold depends on whether it was bought yesterday, that is, the state of yesterday's purchase or sale: plus state, dp [i] [0], dp [i] [1] -The status of buying dp [i] [0] = 1. Buy today, dp [i-1] [1] sold yesterday-result [price] i; 2. Do not buy today, as yesterday Buy status dp [i-1] [0] and compare results. -The status of selling dp [i] [1] = 1. sold today, dp [i-1] [0] result bought yesterday + price [i]; 2. not sold today, compared with yesterday Comparison of sold status dp [i-1] [1]. -Note init: -dp [0] [0] = dp [0] [1] = 0; // 0 days, -dp [1] [0] = 0; // sell on 1st day, haven't bought, so just 0 profit. -dp [1] [0] = -prices [0]; // buy on 1st day, with cost of prices [0]

Rolling Array

-[i] is associated with [i-1], roll

---

134. [Minimum Subarray.java] (https://github.com/awangdev/LintCode/blob/master/Java/Minimum%20Subarray.java) Level: Easy Tags: [Array, DP, Greedy, Sequence DP, Subarray]

time: O (m) space: O (1)

Give a list of arrays, unsorted, can have negative / positive num. Find the minimum of the sum of the numbers in the subarray in the middle of the array

DP

-See min value, at least consider dp: -Consider last num: min sum will be (preMinSum + curr, or curr) -Use preMinSum to cache previouly calcualted min sum, also compare with + curr. -Have a global min to track: because the preMinSum can be dis-continuous. -Can also be written as dp [i] but not necessary

135. [Subtree of Another Tree.java] (https://github.com/awangdev/LintCode/blob/master/Java/Subtree%20of%20Another%20Tree.java) Level: Easy Tags: [DFS , Divide and Conquer, Tree]

Tree

-Traverse tree: left, right -Concept of partial compare vs. whole compare

---

136. [Two Sum IV-Input is a BST.java] (https://github.com/awangdev/LintCode/blob/master/Java/Two%20Sum%20IV%20-%20Input%20is%20a% 20BST.java) Level: Easy Tags: [Tree]

HashSet to store visited items. Same old 2 sum trick.

---

137. [Read N Characters Given Read4.java] (https://github.com/awangdev/LintCode/blob/master/Java/Read%20N%20Characters%20Given%20Read4 Level: Easy Tags : [Enumeration, String]

Read4 title. Understanding title: There is an input object buff, which will be populated with data.

String in char [] format

-Understanding the topic: Actually it is track How many bytes can be read by read4 () response -Another useful function System.arraycopy (src, srcIndex, dest, destIndex, length)

---

138. [Merge Sorted Array.java] (https://github.com/awangdev/LintCode/blob/master/Java/Merge%20Sorted%20Array.java) Level: Easy Tags: [Array, Two Pointers]

Give two sorted arrays, merge. One of the arrays nums1 has extra positions

Basics

-A is long enough, then you can add new elements from the end of A.
-Note that from the end, the large number comes first.

---

139. [Valid Palindrome II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Valid%20Palindrome%20II.java) Level: Easy Tags: [String]

Palindrome String

-delete an index = jump over the index -Note that boolean chance can use a helper function

---

140. [Moving Average from Data Stream.java] (https://github.com/awangdev/LintCode/blob/master/Java/Moving%20Average%20from%20Data%20Str Level: Easy Tags : [Design, Queue, Sliding Window]

Give an interface, design a structure, and be able to calculate the moving window average.

Queue

-Understand the problem, pay attention to the handling of average and window. -Simple queue.size () comparison

---

141. [Move Zeroes.java] (https://github.com/awangdev/LintCode/blob/master/Java/Move%20Zeroes.java) Level: Easy Tags: [Array, Two Pointers]

Move non-zero elements to front of array; preseve order.

Two Pointers

-Outside pointer that moves in certain condition. -Save appropirate elements

---

142. [Flood Fill.java] (https://github.com/awangdev/LintCode/blob/master/Java/Flood%20Fill.java) Level: Easy Tags: [DFS]

Same as MS Paint

DFS

-track `boolean [] [] visited`, validate before dfs

---

143. [Diameter of Binary Tree.java] (https://github.com/awangdev/LintCode/blob/master/Java/Diameter%20of%20Binary%20Tree.java) Level: Easy Tags: [Tree]

Find longest path (include or not include root)

Same idea as Binary Tree Maximum Path Sum: handle single path, or combined path (do not include curr root)

Singlepath, combined path

- `int [] {combinedPath, singlePath}` ; -pick single path + 1: `singlePath = Math.max (left [1], right [1]) + 1` ; -complete left / right child, or join curr root: `combinedPath = Math.max (Math.max (left [0], right [0]), left [1] + right [1] + 1)` ;

---

144. [Backspace String Compare.java] (https://github.com/awangdev/LintCode/blob/master/Java/Backspace%20String%20Compare.java) Level: Easy Tags: [Stack, Two Pointers]

---

145. [Roman to Integer.java] (https://github.com/awangdev/LintCode/blob/master/Java/Roman%20to%20Integer.java) Level: Easy Tags: [Math, String]

String

-Familiar with Roman alphabet rules
-1. 'IVXLCDM' numbers
-2. To enumerate the case of combo, you need to subtract the extra part from the original sum: 'IV, IX' minus 2, 'XL, XC' minus 20, and 'CD, CM' minus 200. - Leading I (1 * 2), X (10 * 2), C (100 * 2) causes double counting

https://en.wikipedia.org/wiki/Roman_numerals

---

146. [Intersection of Two Arrays.java] (https://github.com/awangdev/LintCode/blob/master/Java/Intersection%20of%20Two%20Arrays.java) Level: Easy Tags: [Binary Search, Hash Table, Sort, Two Pointers]

-Method 1: Use hashset to find unique && duplicate: O (m + n) -Method 2: You can use binary search to find numbers. Note: binary search must require array sorted: nLog (m)

---

147. [Strobogrammatic Number.java] (https://github.com/awangdev/LintCode/blob/master/Java/Strobogrammatic%20Number.java) Level: Easy Tags: [Enumeration, Hash Table, Math]

Enumerate according to the topic, and then follow the basic implementation rules

Alter input

HashTable + Two Pointer

---

148. [Valid Parentheses.java] (https://github.com/awangdev/LintCode/blob/master/Java/Valid%20Parentheses.java) Level: Easy Tags: [Stack, String]

Peeling process. The trouble should end it
The outer skin '{[' at the bottom of the stack
The right skin should correspond to the left skin on the top of the stack.

---

149. [First Unique Character in a String.java] (https://github.com/awangdev/LintCode/blob/master/Java/First%20Unique%20Character%20in%20a%2(Level : Easy Tags: [Hash Table, String]

Method 1: According to the meaning of the title, find the first letter of first index == last index.

Method 2: Use a hashmap to store the index of the letter, and the index of some duplicate letters will be a list. Find a single index, combine into a list, sort, return list.get (0)

---

150. [Add Binary.java] (https://github.com/awangdev/LintCode/blob/master/Java/Add%20Binary.java) Level: Easy Tags: [Math, String, Two Pointers]

Two pointers

-Use two pointers i, j to track the 2 strings -Add when i and j are applicable. While (i> = 0 || j> = 0) -StringBuffer.insert (0, x); -handle carry

wrong: convert to int

-Native method is not technical, replace binary with numbers, add them up, and then use binary -If the input is large, it is likely that both int and long cannot be held. Not insurance.

---

151. [Isomorphic Strings.java] (https://github.com/awangdev/LintCode/blob/master/Java/Isomorphic%20Strings.java) Level: Easy Tags: [Hash Table]

HashMap

-two failture cases: -same key, value not matching -two key maps to same value

Previous note

1. Match. Is map.containsKey, map.containsValue, and char1 == char2. Perfect.
2. Either Key not exist, or Value not exit. False;
3. Both key and Value exist, but map.get (char1)! = Char2. Miss-match. False.
4. None of Key or Value exist in HashMap. Then add the match.

---

152. [Next Greater Element I.java] (https://github.com/awangdev/LintCode/blob/master/Java/Next%20Greater%20Element%20I.java) Level: Easy Tags: [Hash Table, Stack]

stack?
