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%20Lower 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%2 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 alllll 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%20Valu 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%2 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.java 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 .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?

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%20Sort 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%20Pointer 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](https://github.com/awangdev/LintCode/blob/master/Java/Strobogrammatic%20Number%20II.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](https://github.com/awangdev/LintCode/blob/master/Java/Reverse%20Pairs.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]`

Hard (91)

0. [Count of Smaller Number before itself.java] (https://github.com/awangdev/LintCode/blob/master/Java/Count%20of%20Smaller%20Number%20befo Level: Hard Tags: [] Is

very similar to Count of Smaller Number. The actual value is used to form the segment tree, and the leaf is stored (count of smaller number).

Trick: Query first, then modify.
Each time Query, A [i] has not been added to the Segment Tree, and A [i + 1, ... etc] has not been added yet.
Then it is naturally coutning smaller number before itself.
Tricky ah!

Also note:
In modify: Check root.start <= index and index <= root.end. It was ignored in the past. You can also write this later.
(In fact, it is Make sense, that is, the index and root.left or root.right are checked more strictly)

---

1. [Kth Smallest Sum In Two Sorted Arrays.java] (https: // github.com/awangdev/LintCode/blob/master/Java/Kth%20Smallest%20Sum%20In%20Two%20Sorted%2 Level: Hard Tags: []

Use priority queue. Each time you expand the smallest, Shift. X + 1, or y + 1:
Because x and y are the smallest in Min at the moment. So the next smallest is either (x + 1, y), or (x, y + 1).

Just poll () one every time, just put 2 new candidates in. Note that this approach will be repeated, for example the example (7,4) will appear twice. Block it with a HashSet.

Note that the uniqueness of HashSet can be solved by using an "x, y" string.

---

2. [LFU Cache.java] (https://github.com/awangdev/LintCode/blob/master/Java/LFU%20Cache.java) Level: Hard Tags: [Design, Hash Table]

Hash Table

- 具体看thoughts, 几种不同的方式使用map
- regular object map : map of <key, item>, where item : {int val; int count}
- Use a Map<frequency count, doubly-linked node> to track the frequency
- Track constant capacity, and minimum frequency
- Every get(): update all frequency map as well
- Every put(): update all frequency map as well, with optional removal (if over capacity)
- Original post: http://www.cnblogs.com/grandyang/p/6258459.html
- TODO: one doubly linked list might be good enough to replace below:
- frequency list map : map of <frequency count, List>, where the list preserves recency
- item location in frequency map : map of <key, int location index in list>:
- index relative to the item in a particular list, not tracking which list here

---

3. [Prefix and Suffix Search.java] (https://github.com/awangdev/LintCode/blob/master/Java/Prefix%20and%20Suffix%20Search.java) Level: Hard Tags : [Trie]

---

4. [Remove Node in Binary Search Tree.java] (https://github.com/awangdev/LintCode/blob/master/Java/Remove%20Node%20in%20Binary%20Search 20Tree.java) Level: Hard Tags: [BST]

方法1: Brutle一点。找到target和target的parent.
When removing the target, rearrange the child nodes of the target to form a new BST: inorder traversal, build tree based on inorder traversal list.

Method 2: Analyze the rules, find them first target and parent, then according to the nature, when removing the target, move the children nodes to ensure that it is still BST.

---

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

---

6. [K Sum.java] (https://github.com/awangdev/LintCode/blob/master/Java/k%20Sum.java) Level: Hard Tags: [DP]

DP. How do formulas come to mind , Need to be re-understood.

Dp [i] [j] [m]: # of possibilities such that from j elements, pick m elements and sum up to i. I: [0 ~ target]

dp [i] [j] [m] = dp [i] [j-1] [m] + dp [i-A [j-1]] [j-1] [m-1] (i not included) (i included)

-
    -
        7. [Copy Books.java] (https://github.com/awangdev/LintCode/blob/master/Java/Copy%20Books.java) Level: Hard Tags: [Binary Search, DP, Partition DP]

Give a series of books pages [i], k people, pages [i] represents the number of pages in each book. K people start to copy books from different points at the same time.

Ask, when can I finish the fastest?

Partition DP

-the first step, understand the problem required by the title: the first k people copy n books and find the least time; it can also be translated into: n books, let k people copy, that is, split into k segments . -Finally, dp [n] [k] is required. On: int [n + 1] [k + 1] .-Principle: -1. Consider the last step: in the book [0 ~ n-1], finally One can choose to copy 1 book, 2 books ...... n books, the results of each cutting method are different -2. Discuss the situation of the kth person, loop in j = [0 ~ i]. And loop The slowest case at j determines the result of the kth person (barrel principle): Math.max (dp [j] [k-1], sum) . -3. where: dp [j] [k-1] is the result of [k-1] personally reading the j book, which is the famous previous step . Here the loop considers the different j previous steps of the kth person:) -4. The result of the loop, Is to exist dp [i] [k] = Math.min (Math.max (dp [j] [k-1], sum [j, i]), loop over i, k, j = [i ~ 0]) -Time: O (kn ^ 2), space O (nk)

Init

-Init: dp [0] [0] = 0, 0 people 0 books -Use of Integer.MAX_VALUE:-When i = 1, k = 1 , Expression: dp [i] [k] = Math.min (dp [i] [k], Math.max (dp [j] [k-1], sum)); -the only feasible case is only one Kind: i = 0, k = 0, exactly 0 people copy 0 books, dp [0] [0] = 0. -In other cases, i = 1, k = 0, 0 people read 1 book, it is impossible to happen : So use Intege r.MAX_VALUE to break Math.max, maintaining absurd values. -When the case of i = 0, k = 0 is discussed, the above equation will calculate dp [i] [k] according to the actual situation -the init of this problem is very important and tricky

calculation Order

-k people, need a for loop; -k people, starting with copy1 book, 2, 3, ... n-1, so i = [1, n], need a second for loop -in each On i, the cutting method can be [0 ~ i], we have to calculate the worst time of each type

Scrolling array-

[k] Only related to [k-1] -Space: O (n)

Binary Search

-According to: How much time does each person spend doing binary search: How long does each person spend, can it be completed in K people with the least time? -The range of time variable is not index, It's not the page size. It's [minPage, pageSum] attention to 3 cases when validating: people are enough k> = 0, there are not enough people, so the ending is reduced to k <0, and there is a time (the time spent by each person) is less than the current page, retur n -1 -O (nLogM). N = pages.length; m = sum of pages.

---

8. [Scramble String.java] (https://github.com/awangdev/LintCode/blob/master/Java/ Scramble% 20String.java) Level: Hard Tags: [DP, Interval DP, String]-Give

two strings S, T. Check if they are scramble string.-Scramble string Definition: string can be split into binary tree Form, that is, cut into substrings;-After rotating a node that is not a leaf, a new substring is formed, which is the scramble of the original string.

Interval DP Interval-

dimensionality reduction, segmentation, dp according to length. -dp [i] [j] [k]: array S starts at index i, T starts at index j, is a substring of length k, is it a scramble string

Break down

-after everything in half, see Two cases:, or do not rotate the two halves. For these substrings, verify whether they are scrambled. -The two halves without rotate division: S [part1] corresponds to T [part1] && S [part2] corresponds to T [part2].-Rotate the two halves: S [part1] corresponds to T [part2] && S [part2] corresponds to T [part1].

Initialization

-len == 1, in fact, cannot be rotated, that is, S, Whether the corresponding index of T is equal to the characters. -Initialization is very important. Very amazing, this initailization lays the foundation for DP, and the result is calculated by mathematical expressions. -Input s1, s2 In the content, it is almost not used, but only used for initialization. -More details, see the answer

---

9. [Interleaving String.java] (https://github.com/awangdev/LintCode/blob/master/Java/Interleaving 20String.java%) Level: Hard Tags: [DP, String]

. dyad sequences DP, considered from the last point of the end of split problems, and considering the correlation between s1, s2 subsequence.

seeking existence, Boolean

-
    -

10. [Edit Distance.java] (https://github.com/awangdev/LintCode/blob/master/Java/Edit%20Distance.java) Level: Hard Tags: [DP, Double Sequence DP, Sequence DP, String]

time: O (MN) Space: O (N)

two strings, A must be B, you can insert / delete / replace, find the smallest change operation count

Double Sequence

-Consider the index of the end of two strings? S [i], t [j]: If you need to make this Two characters are the same, it is possible to use the three operations given in the title: insert / delete / replace? -First calculate the worst case, 3 operation count + 1; then compare the case of match.-Note, it is 0 in i or j When the step becomes another number, it can only be fully changed. -In the first step, space time is O (MN), O (MN) -rolling array optimization, space O (N)

Detail analysis

-insert: assume insert on s,? #ofOperation = (s [0 ~ i] to t [0 ~ j-1]) + 1; -delete: assume delete on t, #ofOperatoin = (s [0 ~ i-1] to t [0 ~ j]) + 1; -replace: replace both s and t, #ofOperatoin = (s [0 ~ i-1] to t [0 ~ j-1]) + 1; -dp [i] [j]? Represents the nature of two sequences:? S [0 ~ i]? The minimum operation count required to convert to s [0 ~ j] -init: when i == 0, dp [0] [j] = j;? + j characters each time; Similarly, when j == 0, dp [i] [0] = i; -and dp [i] [j] There are two cases to deal with: s [i] == t [j] or s [i]! = t [j]

When initialize

-this kind of judgment depends on experience: if you know that initialization can be done together in a double for loop, then you can leave it as it is -this belongs to what is needed, initialize what -When doing space optimization afterwards, you can easily do rolling array on 1st dimension

Search

-It can be done, but it is not recommended: this problem needs to find min count, not search / find all solutions, so search will It's more complicated to write, kill a chicken with a knife.

---

11. [Distinct Subsequences.java] (https://github.com/awangdev/LintCode/blob/master/Java/Distinct%20Subsequences.java) Level : Hard Tags: [DP, String]

Double Sequence DP: 0. DP size (n + 1): Find the result of the previous nth, then the dp array needs to open n + 1, because the end needs to return dp [n] [m]

1. Initialize dp [0] [j] dp [i] [0] in the for loop
2. Rolling array is optimized to O (N): If dp [i] [j] is in the for loop, it is a good replacement for curr / prev

---

12. [Ones and Zeroes.java] (https://github.com/awangdev/LintCode/blob/master/Java/Ones%20and%20Zeroes.java) Level: Hard Tags: [DP]

It is still Double Sequence, but consider the third state: the amount of string array given. So a 3-dimensional array is opened.

If you use a scrolling array to optimize space, you need to put the for loop to be scrolled to the outermost, not the innermost. Of course, this third bit of

definition is in which position of dp [] [] [], the problem is not big. In addition, pay attention to calcultete zeros and ones outside, saving time and complexity.

---

13. [Word Break II .java] (https://github.com/awangdev/LintCode/blob/master/Java/Word%20Break%20II.java) Level: Hard Tags: [backtracking, DFS, DP, Hash Table, Memoization]

find Out all word break variations, given dictionary

using memoization: Map <prefix, List <suffix variations >>

DFS + Memoization

-Realize the input s expands into a tree of possible prefixes. -We can do top-> bottom (add candidate + backtracking) OR bottom-> top (find list of candidates from subproblem, and cross-match) -DFS on string: find a valid word, dfs on the suffix. [NO backtraking in the solution]-DFS returns List : every for loop takes a prefix substring, and append with all suffix (result of dfs) -IMPORANT : Memoization: Map <prefix, List <suffix variations >>, which reduces repeated calculation if the substring has been tried. -Time O (n!). Worst case, permutation of unique letters: s = 'abcdef .. .. ', and dict = [a, b, c, d, e, f ...]

Regular DPs

-Two DPs used together, solve the problem of timeout: when a invalid case' aaaaaaaaa 'occurs, isValid [] stops dfs from occuring -1. isWord [i] [j], subString (i, j) exists in the dict? -2. Use isWord to speed up isValid [i]: Can [i ~ end] find a reasonable solution from dict?
-View i from the end: Because we need to test isWord [i] [j], j> i, and we observe the interval [i, j];
-The part of j> i also needs to be considered, we also need to know isValid [0 ~ j + 1]. So isValid [x] is a DP indicating whether [x, end] is valid this time.
-i is from the end to 0, probably because it is considered that isWord [i] [j] is within [0 ~ n], so the numbers are reversed and the coordinates are easier to figure out.
-(Looking back at Word Break I, there is also a method of coordinate inversion) -3. dfs uses isValid and isWord to make ordinary DFS.

Timeout Note

-Regarding regular solution: If you do not do memoization or dp, 'aaaaa .... aaa' will repeatedly calculate the same substring -Regarding double DP solution: Set.contains (...), In isValid, i starts from 0. However, contains () itself is O (n); intead, using an isWord [i] [j] to determine whether i ~ j is a dictionary

- O (1) - -

14. [Minimum Window Substring.java] (https://github.com/awangdev/LintCode/blob/master/Java/Minimum%20Window%20Substring.java) Level: Hard Tags: [Hash Table , String, Two Pointers] The

basic idea: use a char [] to store the frequency of the string. Then 2pointer, end go to the end, and continue to validate. If it meets the process as result candidate, the method of

---

The method of HashMap is a bit more complicated to write than char [], but more generic

15. [Longest Substring with At Most K Distinct Characters.java] (https://github.com/awangdev/LintCode/blob/master/Java /Longest%20Substring%20with%20At%20Most%20K%20Distinct%20Characters.java)### Level: Hard Tags: [Hash Table, Sliding Window, String]

Big cleaning O (nk)
map.size Once> k, you need to change longest string the very beginning (marked by pointer: start) that erase char
. Once one char to be cleared, so the char in the char between the 1st and last appearance of the Map must be cleaned from

---

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

An issue that requires rigorous thinking. Because duplicates cause constant translation, the time complexity is ultimately O (n) So it 's better to scan it directly, give The answer.

But still write a Binary Search, but the result is O (n)

---

17. [Number of Islands II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Number%20of%20Islands%20II.java) Level: Hard Tags: [Union Find]

give an island grid [] [], and list of operations to fill a particualr (x, y) position.

Count # of remaining island after each operation.

Union Find, model with int []

-put board Converted to 1D array, you can use union-find to judge.-With int [] father's unionFind, you need to convert 2D position into 1D index. This is relatively clean. -When judging, one step is taken in each of the four directions to determine whether It is the same Land.-Every time I walk the operator, it will count ++. If it is found that the same island, count--The addition and subtraction of count are all placed in the UnionFind's own function, for convenient tracking, just give a few helper functions a.

- Time: O (k * log (Mn))

  #### Note:

Union Find, model with Hashmap

-Union-find with HashMap. -

Proof of UnionFind log (n) time: https://en.wikipedia.org/wiki/Proof_of_O(log*n)_time_complexity_of_union%E2% 80% 93find

---

18. [Word Search II.java](https://github.com/awangdev/LintCode/blob/master/Java/Word%20Search%20II.java) Level: Hard Tags : [Backtracking, DFS, Trie]

Give a string of words, and a 2D character matrix. Find all the words that can be formed. Condition: 2D matrix can only be positioned next to each other.

Trie, DFS

-Compared to the previous Implementation, there are some places that can be optimized: -1. When backtracking, mark on board [] [], you don't need to open a visited [] [] -2. You don't need all the equations of the implementation trie, you can't use: here Only insert is required. -The common trie topic will let you search for a word, but here is a board, see if each letter of the board can come out of Word. -That is: the search here is written manually, not traditional trie search () funcombination -3. When there is an end in the TrieNode, the string word is stored, which means the end. When the word = null is used up, the problem of repeated search is just truncated.

Previous Notes

# About Trie

-Build Trie with target words: insert, search, startWith. Sometimes, just: `buildTree (words)` and return root. -Still need to DFS the board matrix. -no for loop on words. Directly to the board DFS:-Each
layer will have an up-to-this-point string. Check if it exists in the Trie. Use this to judge.
-If it does not exist, you do not need to continue DFS. -Trie solution time complexity, much better:
-build Trie: n * wordMaxLength -search: boardWidth * boardHeight * (4 ^ wordMaxLength + wordMaxLength [Trie Search])

Regular DFS

-for loop on words: inside, do board DFS based on each
word.-Time cpmplexity: word []. Length * boardWidth * boardHeight * (4 ^ wordMaxLength)-Big

improvement: use boolean visited on TrieNode! -Don't use rst.contains (...), because this is O (n) timeout in leetcode (lintcode can pass)! -In Trie search () method, mark any visit.

---

19. [Word Squares.java](https://github.com/awangdev/LintCode/blob/master/Java/Word%20Squares.java) Level: Hard Tags: [Backtracking, Trie]

You can open the Trie class, which uses TrieNode. Opening Trie (words) can be directly initalized with for loop TrieNode There can be a List startWith: Record all strings that can reach this point: a bit like a tree, ancestor-shaped storage .

God operation: according to the nature of the square, if the select list of words, setting int prefixIndex = list.size (). remove list all inside word [prefixedIndex], and together, the next word candidate is a prefix.

A bit of image:
list = ["ball", "area"]; prefixIndex = list.size (); ball [prefixIndex] = 'l'; area [prefixIndex] = 'e'; // then candidatePrefix = ball [prefixIndex] + area [prefixIndex] = "le"; the Trie one can be used here findByPrefix function, at each point, all the dates that can be generated by this point are stored. At this time, try all the dates: dfs

can think of this inverted structure to store prefix candidates in Trie, this idea is very worth thinking about.

---

20. [Trapping Rain Water.java](https://github.com/awangdev/LintCode/blob/master/Java/Trapping%20Rain%20Water.java) Level: Hard Tags: [Array , Stack, Two Pointers] There are

many methods for this topic.

Method 1

Array, maintaining a left-hand highest wall array, right-hand highest strength array. For each index, the largest water column that can be stored vertically is by The left and right highest walls are determined by: min (leftHighestWall, rightHighestWall)-currHeight.

Method 2

The optimization above method 1, two pointers, still find the highest left and highest right. O (1) space. The idea used in method 3 is the same: the entire structure is divided by the highest bar in the middle of the world:
left is calculated according to maxLeft, and the right is calculated according to maxRight.

Method 3

2 Pointers, double-sided pinching:

1. find the index of the highest bar in the middle
2. Swipe to the center on both sides: add (topBarIndex-currIndex each time)) * (Elevation from previous index). That is, one bar is added at a time.
3. Every time you want to subtract the height of the block itself

The

main idea of Method 4 is the same as Method 3: On the basis of the downhill slope, the stack has been used to accumulate the bottom. Before the last encounter, the bottom can be used at this time. Let's compare it with all the downhill indexes that were stacked before the stack, and calculate the stagnant water that is different from their height. The idea of using the stack to record the downhill and then dig to the end with a while loop is great.

---

21. [Largest Rectangle in Histogram.java] (https://github.com/awangdev/LintCode/blob/master/Java/Largest%20Rectangle%20in%20Histogram.jav Level: Hard Tags: [Array, Monotonous Stack, Stack]

Give n bars to form a histogram of histograms. Find the rectangle with the largest area that can be found in this row of histograms.

Thinking: Finding the area of a rectangle is nothing more than finding two indexes, then the length of the bottom edge * height.

# Monotonous Stack

-The main point is to maintain a monotonically increasing Stack according to the nature of the rectangle in the Histogram. -When loop over indexes:-If the height is> = previous peek (), then for that peek, it means, go down, keep going up Well, the previous peek can always continue to bottom -when can't it bottom? When there is a downward trend -this time not calculate all the previous peek, but consider all the previous peeks greater than the current height. -Put these All the rectangles from the previous grid from peek to current height are found out: stack.pop () -In the process of stack.pop (), the current height is not counted, because it needs to be retained in the next round, and the current index is added to the stack. Then again -why use stack? Because you need to know the continuous increasing peek, stack.peek () O (1), it is easy to use without stack, you can also record all heights in other ways, but you need O (n) to find peek inconvenient to

knowledge

- understanding how monotonous stack is maintained
- maintenance monotonous stack is the subject needs, rather than stack very nature, it is a means of stack.peek () O (1) Clever usage.

---

22. [Find Peak Element II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Find%20Peak%20Element%20II.java) Level: Hard Tags: [Binary Search, DFS, Divide and Conquer]

2Dmatrix, the value inside has some increasing and decreasing characteristics (the details are longer, see the original question). The goal is to find the peak element peak: the value is larger than the points in the 4 directions around

DFS

Basic principle

-We cannot accurately locate (x, y) at one go, but we can find the peak of the 1D array in another row / col. -According to this point, go in the remaining two directions Move -1. In the middle row i = midX, find the y where the peak is. -2. In the middle row j = midY, find the x where the peak is. (It is possible that the y found before the strong override overrides the line (Peak, find peak on midY)-3. According to the 4 neighbor check (x, y) of (x, y) whether (x, y) is the peak, if not, move it one block like a higher position -4. According to the previous calculation MidX, midY divide the board into 4 quadrants, and continue to find in each one -this question LintCode does not do it, so the idea is correct, but the answer has not been checked again Proof.-

Pruning / splitting quadrant

Every time I just find a peak in row / col! this point is equivalent to cutting the board in half. -Then, comparing with the remaining two adjacent positions, I know where it is more Large, go where to find the peak, that is, cut the second knife again. -When cutting the second knife, also move (x, y) to the quadrant that needs to be taken. DFS -cut according to mid row: -http : //www.jiuzhang.com/solution/find-peak-element-ii/#tag-highlight-lang-java

- http://courses.csail.mit.edu/6.006/spring11/lectures/lec02.pdf

#### Time complexity

-each level is halved -T (n) = n + T (n / 2) = n + n / 2 + n / 4 + ... + 1 = n (1 + 1 / 2 + ... + 1 / n) = 2n = O (n)

Binary Search

-TODO -O (nLogN)

---

23. [Palindrome Pairs.java] (https: // github.com/awangdev/LintCode/blob/master/Java/Palindrome%20Pairs.java)### Level: Hard Tags: [Hash Table, String, Trie]

Obvious's method is to try it all, and judge, it becomes O (n ^ 2) * O (m) = O (mn ^ 2). O (m): isPalindrome () time.

Of course it doesn't work, then it is O (nlogN), or O (n)?

Method 1: Properties of Hash Table + Palindrome. Compound.

O (mn)

Idea

-Each word can be split into front + mid + end. If this word + other words can form palindrome, it means -cut off (mid + end), front.reverse () should be stored in words [] .-cut off (front + mid), end.reverse ( ) Should exist in words [] .-We use HashMap to store all <word, index>, and then reverse, just find a match.

Corner case

-If there is an empty string "", then it is related to any palindrome word, can be paired, and transformed back and forth according to the position to make 2 distinct indexes.-This has the logic of `if (reverseEnd.equals (" ")) {...}`. -Note: Although in the The two for loops that deal with beheading / chopping are repeatedly recording according to the empty string, but because "" itself cannot be used as a starting point, overall will only be recorded once when paired by other palendrome.

Method 2: Trie

still has to do that.

---

24. [Maximal Rectangle.java] (https://github.com/awangdev/LintCode/blob/master/Java/Maximal%20Rectangle. java) Level: Hard Tags: [Array, DP, Hash Table, Stack]

Method 1: The monotonous stack is

decomposed, but it is actually 'Largest Rectangle in Histogram', but here you have to build your own model heights. The rectangle in a 2D array is also finally made with height * width. The clever thing is that each line is used as the bottom edge, and the bottom edge is calculated, and the height to the top is: -If a value == 0 on the bottom edge, then Calculated as no height (the bottom edge is used as a rectangle, value == 0 is the sky tower, cannot be used) -If the value on the bottom edge == 1, then add the above height to make a histogram

. For example, some rows seem to be calculated for nothing, but there is no way. This is a search process, and the optimal solution will eventually be compared.

Method 2: DP

Coordinate DP?

---

25. [Longest Increasing Path in a Matrix.java] (https://github.com/awangdev/LintCode/blob/master/Java/Longest%20Increasing%20Path%20in%20a%2 Level : Hard Tags: [Coordinate DP, DFS, DP, Memoization, Topological Sort]

mxn matrix, find the longest increasing sequence length. Here is the default continuous sequence.- Looping

is not possible, so visit passes (x , y) can't go anymore.-Cannot go in the oblique direction, can only go up, down, left and right -Can not do according to the coordinate DP, because the calculation order can go in 4 directions. -Finally, all nodes must be visited, so use DFS search More suitable.

DFS, Memoization

-Simple version: longest path, only allow right / down direction: - dp [x] [y] = Math.max (dp [prevUpX] [prevUpY], or dp [prevUpX] [prevUpY] + 1) ; and compare the other direction as well -This problem, just compare the direction from dfs result -DFS has too many repeated calculations; memoization (dp [] [], visited [] []) saves Double counting -initialize dp [x] [y] = 1, (x, y) One grid -dfs (matrix, x, y): check 4 neighbors (nx, ny) of (x, y) each time, if they are increasing to (x, y), then consider and compare: -Maht.max (dp [x] [y], dp [nx] [ny] + 1); where dp [n] [ny] = dfs (matrix, nx, ny) -top level: O (mn), try from each (x, y) start -O (m * n * k), where k is the longest path

Topological sort

has not been done yet

---

26. [Coins in a Line III.java] (https: / /github.com/awangdev/LintCode/blob/master/Java/Coins%20in%20a%20Line%20III.java)### Level: Hard Tags: [Array, DP, Game Theory, Interval DP, Memoization]

LeetCode: Predict The Winner

is still 2 people who take n coins, and the coins can have different values.

But this time the player can take from any side, but not restricted from one side. Will the first mover win? [i] [j] represents the sum of the maximum values that players can take in the [i, j] interval`

mover #### Memoization + Search -Like Coins in a Line II, MaxiMin's idea: find the largest value among my disadvantages -again, sum [i] [j] represents the sum of the values between [i] and [j] -the worst case of the opponent, That is the best case of the first mover: -dp [i] [j] = sum [i] [j]-Math.min (dp [i] [j-1], dp [i + 1] [j]); -You need to search here and draw a tree to see how it is segmented according to before and after.

Game + Interval DP, Interval DP

-Because it looks at the interval [i, j], it can be thought of as an interval DP -This method needs a review and is related to the inference of mathematical expressions: S (x) =-S (y) + m. Refer to the following formula to derive. -Dp [i] [j] means from index (i) to index (j), the difference between the maximum value that the first player can get and the opponent's number. That is S (x) .-One of them S (x) = dp [i] [j] = a [i]-dp [i + 1] [j] -m at the beginning and m at the end: -dp [i] [j] = max {a [i]-dp [i + 1] [j], a [j] -dp [i] [j-1]} -len = 1, the integral is values [i] -finally judged dp [0] [n]> = 0, the difference between the maximum number and the number is greater than 0, the win. -time / Space O (n ^ 2)

Formula derivation

-S (x) = X-Y, find the difference between the largest number and sum, where X and Y are the total score of player X and the total score of player Y.
-For player X: if S (x) maximum is large At 0, you win; if the maximum values are less than 0, you must lose.-Player Y: S (y) is used to indicate that for Y, the largest number and The difference. S (y) = Y-X -According to S (x), if you take a number m from the number and X, that is X = m + Xwithout (m) -S (x) = m + Xwithout (m)-Y = m + (Xwithout (m)-Y) .- If we simply remove m from the global, then S (y '') = Y-Xwithout (m)-then calculate: S (x) = m + (Xwithout (m)-Y) = m-(Y-Xwithout (m)) = m-S (y '')-In this question, when we model X and Y, they are actually both dp [i] [j], and the difference is first-hand / last-hand. -Apply the formula, a certain S (x) = a [i]-dp [i + 1] [j], which is m = a [i], And S (y '') = dp [i + 1] [j]

Note

-If you consider calculating the maximum value between [i, j] first hand, then two arrays may be needed, and finally It is used to compare the score of the first mover and the opponent => then more dimensions are needed.-The number difference we consider here just happens to make it unnecessary to calculate the total score of the first mover, which is very clever. -Trick: Use the difference formula to derive A little hard to imagine To.

Interval-type dynamic programming

-find the properties within the interval [i, j]: dp [i] [j] The subscript indicates the interval range [i, j] -Sub-question: Beheading, tailcut, beheading

- loop should be based on Between the length
- template: consider len = 1, len = 2; i must be i <= n-len when i is set; j = len + when j is set i-1;

---

27. [Burst Balloons.java] (https://github.com/awangdev/LintCode/blob/master/Java/Burst%20Balloons.java) Level: Hard Tags: [DP, Divide and Conquer, Interval DP, Memoization]

A volleyball, each ball has a value, each time you break one, you will score: left * middle * right value. Find, how to tie, the maximum?

TODO: Need more thoughts on why using dp [n + 2] [n + 2] for memoization, but dp [n] [n] for interval DP.

Interval DP

-Because array rules will change, it's hard to find 'first A burst of balls'. On the contrary, which one is the last burst? -The last burst becomes a wall: separate the two sides, consider separately, the principle of addition; finally add the middle. -Dp [i] [j] represent max value on range [i, j) -Need to calculate dp [i] [j] incrementally, starting from range size == 3 ---> n -Use k to divide the range (i, j) and conquer each side.

Interval DP three axes:-split in the

middle -cut off the head or tail -Range interval as the basis of iteration

Print the calculation process

-use pi [i] [j] and print recursively. -Print k, using pi [i] [j]: max value taken at k

# Memoization

-In fact, I will do a DP that I think of very well afterwards -dp [i] [j] = max between balloons i ~ j. - Then which point to start burst? Set it to x. -For loop all points are taken as x, go to burst. -Each burst is cut into three parts: the left side can be recusive to find the maximum value of the remaining part on the left side + the middle 3 terms are multiplied + the right side is recursive to find the maximum value. -Note: This is Memoization, not pure DP. -Because it is recursive, it is still a search, but memorize the calculated value, saving Processing

---

28. [K Edit Distance.java] (https://github.com/awangdev/LintCode/blob/master/Java/K%20Edit%20Distance.java) Level: Hard Tags: [DP, Double Sequence DP, Sequence DP, Trie]

Give a string of String, target string, int k. Find all the dates in the string array: change K times, can become the target.

Trie

TODO

Double Sequence DP

-Follow up for Edit Distance. -Actually It is to change the function of minEditDistance and bring in K for comparison. -The main logic of writing is exactly the same as Edit Distance. -But timeout in LintCode 86% test case. -Time O (mnh), where h = words.length , If n ~ m, Time is almost O (n ^ 2), too slow.

---

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

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

a row of n houses, each house It can be painted in k colors, the price of each house is different, and it is represented by costs [] [].

Costs [0] [1] indicates that the house with index 0 is painted, and color 1.

Rules: Adjacent Two houses cannot make the same color

Find: the least cost

DP

-almost the same as Paint House I, but the paint color is more: k colors. -Consider first Simply use dp [i] to represent the minimum cost of painting the first i houses -but what colors dp [i] and dp [i-1] index will affect each other, it is difficult to discuss, so add state: the sequence DP is added The status becomes 2D. -Consider the last bit, and the previous bit i-1 is limited by the color of the i bit, so when considering min dp [i], there is another layer of iteration. -Do dp [i] [j]: # cost for i before a house, so the first pick (i-1) cost of the house, and then find the (i-2) of the house cost

- K color => O (of NK ^ 2)
- If No optimization, almost the same code as Paint House I
- Time O (NK ^ 2), space (NK)
- Rolling array: reduce space to O (K)

Note

-the sequence type dp [i] means' The first i-1 'results. So dp is best set to int [n + 1] size. -However, the color is the state here, so it remains in j: [0 ~ k) -[[8]] This edge case. Ca n't run into for loop, so special handle.

Optimization Solution

-only sell 2 times, split the sale into 5 status modules. -Time: O (NK)-If you know that you need to choose two different minimum costs from the cost each time, then the minimum two Pick out, there is no need to have a third for loop Find min -Each time in the series: find the minimum value other than yourself, use the idea of the minimum value / secondary value -maintain 2 minimum values: the minimum value / secondary value. -When calculating, if the minus value is not the minimum value The index of the value gives the minimum value; if the index of the minimum value is removed, the next smallest value is given.-Every loop: 1. calculate the two min vlaues for each i; 2. calcualte dp [i] [j] -How to think of optimization: write the expression and then see where you can optimize -In addition, you can still roll the array, reduce space complexity to O (K)

---

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

has one more restriction than stock II: only 2 sell opportunities.

DP plus status

-at status index 0, 2, 4: No shares held. 1. Always in this status, max profit unchanged; 2. just sold, dp [i] [previous state] + profit -At state index 1, 3: Holding the stock. 1. Always in this state, daily profit. 2. Just bought, state changed, but no profit yet: dp [i] [前 State]

Partial profit

-Adding daily partial profit (diff) together, the final overall profit is the same. The only thing that is better is that you don't need to record the time of the intermediate purchase. -When will the profit be accumulated?-1. Original Hold the stock, if there is no action, then the status is unchanged, and the profit diff is accumulated. -2 . The stock is sold, the status is changed, and the profit diff is accumulated. -Note: Only in the state index: 0, 2, 4, also Only when the stock is sold, you can accumulate profit

Rolling Array-

[i] Only deal with [i-1], reduce space -O (1) space, O (n) time

Find Peak

-Find the peak; then look for another peak. -How about Optimize twice? Start looking for Max from both sides at the same time! (Awesome idea) -leftProfit is the largest Profit at each i point from left to right.

-rightProfit is the maximum profit at each point starting from point i to the end.-At point i, it is the leftProfit, and the rightProfit split point. At point i, leftProfit + rightProfit is added to find the maximum value. -Three O (n), or O (n)

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

has int [] price of stock, up to k transactions. Seek maximum profit.

DP

-According to StockIII, it is not difficult to find that StockIV divides the state into 2k + 1 shares. Then the same code, transplant.

Note 1: -If

k is large, k> n / 2, then the length is n In the array, there can only be n / 2 transactions at most -then the problem is simplified to stockII, giving n arrays, unlimited transactions. -Note that the number of status is 2k + 1 -Time O (NK), Space O (2k + 1) to store the status

Note 2:

-The final status is 'no stock' should be considered, and a for loop is used to compare max. -Of course, it is also possible to make a profit variable and keep comparing.

Method 2-

(previous notes, proficient first One way to think about it is OK) -Remember to understand: Why sell on day i-1 and buy again, can I make a transaction with the sale on day i?
-Because the price of daily transactions is fixed. So if you sell and buy, you are not selling! That's why you can merge. Be sensitive to prices. -Inspired from here: http://liangjiabin.com/blog/2015/04/leetcode-best-time-to-buy-and-sell-stock.html

Local optimal solution vs. global optimal Solution:

-local [i] [j] = max (global [i – 1] [j – 1] + diff, local [i – 1] [j] + diff)
-global [i] [j] = max (global [i – 1] [j], local [i] [j])
-local [i] [j]: on the i-th day, the j-th transaction profit
-global [i] [j]: the first i day, a total of j transactions of profit.

-The difference between local [i] [j] and global [i] [j] is: local [i] [j] means there must be a transaction (sell Out) happened.
-When the price on day i is higher than day i-1 (that is, diff> 0), then this transaction (buy on day i-1 and sell on day i) can be traded with day i-1 (Sell) merge into one transaction, that is local [i] [j] = local [i-1] [j] + diff;
-When the price on day i is not higher than day i-1 (that is, diff <= 0), then local [i] [j] = global [i-1] [j-1] + diff, and due to diff <= 0, so it can be written as local [i] [j] = global [i-1] [j-1].
max number depends on the max value of the previous successful Russian doll + 1 -(Note: + diff is not omitted in this solution below)

-global [i] [j] is the maximum return we can make for a maximum of k transactions in the first i day, which can be divided into two cases:
-If there is no transaction (sell) on day i, then global [i] [j] = global [i-1] [j];
-If there is a transaction (sell) on day i, then global [i] [j] = local [i] [j].

---

32. [Russian Doll Envelopes.java](https://github.com/awangdev/LintCode/blob/master/Java/Russian%20Doll%20Envelopes.java) Level: Hard Tags: [Binary Search, Coordinate DP, DP]

Russian matryoshka, here is represented by envelope. Give a string of array, each [x, y] is the length and width of envelope. [[5,4], [6,4], [6, 7], [2,3]].

Look at these sets of dolls, you can set a few at most.

DP: 1D Coordinate

-envelopes have no order, sort first (mainly according to the first index) -then observe: After sorting, it becomes a 1D coordinate dynamic programming. -The previous index is unknown, so iterate to find the previous index. -The current state of index i depends on the state of the previous index j, so iterate through the two indexes. -O (n ^ 2) DP, n = envelopes.length;

DP: 2D Coordinate

-This method came up by myself, but the time complexity is too big, timeout -Mark the envelop on the 2D grid, and then like a robot, find the maximum count max in the bottom right corner. -Count How many Russian dolls can exist at the moment -Two cases: the current coordinate does not have a target, the current coordinate has a target -the current coordinate does not have a target: like the robot moves, Math.max (dp [i-1] [j], dp [i] [j-1])

- The current coordinate has target: dp [i-1] [j-1] + dp [i] [j]
- timeout: O (n ^ 2), n = largest coordinate.

---

33. [Expression Tree Build .java](https://github.com/awangdev/LintCode/blob/master/Java/Expression%20Tree%20Build.java) Level: Hard Tags: [Binary Tree, Expression Tree, Minimum Binary Tree, Stack]

Monotonous Stack

-Like Max-tree, https://leetcode.com/problems/maximum-binary-tree

Give a string of characters, which is the formula expression. Turn the formula into an expression tree

-use bottom-> top increasing stack: the bottom root is kept to the smallest element. -This topic is Min-tree, with the smallest head, Logic and max-tree are exactly the same
-Space: O (n) -Time on average: O (n).

Features

-TreeNode: Use a TreeNode that is not the final result, store weight, and use it for sorting -use base weight The concept of weighing the symbols at the same level, numerical order -each character is a node, has its own weight. Use a TreeNode to store the weight value, use weight to judge :-(while loop) if node. val <= stack.peek (). nodeValue, change the current stack.peek () to the left child. -2. (if condition) If the stack is left , change the current node to stack.peek (). rightChild

---

34. [Expression Evaluation.java](https://github.com/awangdev/LintCode/blob/master/Java/Expression%20Evaluation.java) Level: Hard Tags: [Binary Tree, DFS, Expression Tree , Minimum Binary Tree, Stack]

Give a formula expression, array of strings, and evaluate the result of the expression.

DFS on Expression Tree

-Calculate the value of expression: 1. Build the expression tree. 2. DFS calculation result -Expression Tree: Minimum Binary Tree (https://lintcode.com/en/problem/expression-tree-build/)-After building the Min Tree, do PostTraversal. -Divde and Conquer: first recursively find the size of left and right, then evaluate the middle Symbol -Time, Space O (n), n = # expression nodes

Note

-1. For Handle numbers, if left && right Child is all Null, then it must be our largest number node.
-2. If one child is null, then return another node.
-3. prevent Integer overflow during operation: Use a Long during the process, and finally cast back to int.

---

35. [Convert Expression to Polish Notation.java] (https://github.com/awangdev/LintCode/blob/master/Java/Convert%20Expression%20to%20Polish%20N Level: Hard Tags : [Binary Tree, DFS, Expression Tree, Stack]

Give a string of characters to represent the formula expression. Convert this expression into Polish Notation (PN).

Expression Tree

-Expression Tree: Minimum Binary Tree (https: //lintcode.com/en/problem/expression-tree-build/)-After making the Expression Tree according to the intent: Come a Pre-order-traversal to record the Polish Notation -This question is not given to 'ExpressionTreeNode', so Think of TreeNode as the node we need, which can be expanded to have left / right child. -Note: The label needs to be String. Although Operator is a char with length 1, the number can be multiple digits

---

36. [Convert Expression to Reverse Polish Notation.java] (https://github.com/awangdev/LintCode/blob/master/Java/Convert%20Expression%20to%20Reverse%2(Level : Hard Tags: [Binary Tree, DFS, Expression Tree, Stack]

Give a string of characters to represent the formula expression. Convert this expression to Reverse Polish Notation (RPN).

Expression Tree

-Expression Tree: Minimum Binary Tree (https://lintcode.com/en/problem/expression-tree-build/)-After making the Expression Tree according to the intent: After a Post-order-traversal, you can record the Reverse Polish Notation -This question does not give 'ExpressionTreeNode', so consider TreeNode as the node we need, and it can be expanded to have left / right children.

---

37. [Decode Ways II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Decode%20Ways%20II.java) Level: Hard Tags: [DP, Enumeration, Partition DP]

gives a string of numbers to be decoded into English letters. [1 ~ 26] Corresponds to the corresponding English letters. Find out how many ways can be decoded.

The characters may be "*", which can represent [1- 9]

DP

-Multiplication principle -same as decode way I, addition principle, when cutting point: 1 digit or 2 digits is used to decode at present -define dp [i] = how many kinds of first i digits can be at most The method of decode. new dp [n + 1].-The different case is: if there is "*" in each partition, it will extend many different possibilities in itself -then: dp [i] = dp [i -1] * (#variations of ss [i]) + dp [i-2] * (#variations of ss [i, i + 1])

Features

-The ability to enumerate: specific analysis' * ' Where it appears, enumerate numbers, basic skills. -Note !! The title says * in [1, 9]. (It will be more difficult if 0 ~ 9) -Understand the reason for the MOD: the number is too large, the mod Give the final result: In fact, under the mod of 10 ^ 9 + 7, most of them Examples are pervious.

-After enumeration, in fact, the writing and thinking process of this topic are not difficult 38. [Palindrome Partitioning II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Palindrome%20Partitioning % 20II.java) Level: Hard Tags: [DP, Partition DP]

Give a String s, find out how many cuts are used at least, so that each substring that is cut out is palindrome

Partition DP

-Find minimum cut: Divide DP -dp [i]: How many knives to cut at least, so that the string of the length of the previous i, cut out are all palindrome -finally get dp [n], so int [n + 1] -move the cutter, See where to cut, index j in [0 ~ i] -consider whether [j, i-1] is a palindrome, if so, then: dp [i] = min (dp [i], d [j] + 1) .-Note: It is estimated that when traversing j, the traversal can be reversed.

Calculating the optimization of Palindrome

-Using the properties of palindrome, you can calculate the situation of boolean palindrome [i, j].- Find an arbitrary mid point :

- 1. Suppose palindrome length is odd, then the mid separate characters, The characters [mid-1], [mid + 1] on both sides should be exactly equal.
- 2. Assuming palindrome is even length, then the characters at [mid] and [mid + 1] should be equal.-Do this to palindrome [i, j]: Whether the substring from character i to character j is palindrome
- like this Give our question a reasonable dimensionality reduction, currently it is time: O (n ^ 2) .-Otherwise, find palindrome once, which is n, will become O (n ^ 3)

Previous Notes

-Double for loop A kind of substring string (i ~ j). If i, j are adjacent or at the same point, then isPal; otherwise, (i + 1, j-1) between i and j must be isPal. -It seems that when checking i, j, how can I know (i + 1, j-1) pressed in the middle first? Actually not .. When j grows up slowly, all 0 ~ j substrings are checked. So isPal [i + 1] [j-1] must already know the result. -okay. Then if any of the above is true, that is to say isPal [i] [j] == true. Then we have to judge how many ways to cut to the end point of the loop parameter j in the first layer? -The idea is smooth: we naturally think that it would be better to add the cut before i plus what happened between i ~ j.

-Anyway, j is not changed now, let 's see where i is and whether cut [i-1] is smaller / minimum; then +1 based on cut [i-1] is over.-Of course, if i == 0, and i ~ j is isPal, then there is nothing to talk about, don't cut, 0 knife. -In the end, brush to cut [s.length ()-1], which is the last point. return is right.

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

for n Different items, int [] A weight, int [] V value, each item can be used an unlimited number of times

Ask how many values can be put into a bag of size m?

DP

-You can use items infinitely and lose The meaning of last i, last unique item: Because it can be reused. -So you can convert an angle: -1. Use i kinds of items, spell out w, and meet the problem conditions (max value). Here because of item i can be used unlimited times, so consider how many times K is used. -2. Although K can be unlimited, it is also limited by k * A [i]: the maximum cannot exceed the size of the backpack. -dp [i] [w]: before i items, fill weight w backpack, what is the maximum value.-dp [i] [w] = max {dp [i-1] [w-k * A [i-1]] + kV [i-1]}, k> = 0 -Time O (nmk)-If k = 0 or 1, it is actually Backpack II: Take or not take

Optimization

-Optimize the time complexity, draw the graph and find: -The calculated (dp [i-1] [j-k * A [i-1]] + k * V [i-1]) -In fact, the grid of dp [i] [jA [i-1]] on the same line has V [i-1]-so there is no need to loop over k times every time -Simplified: dp [i] [j] One of them may be: dp [i] [j-A [i-1]] + V [i-1] -Time O (mn)

Space optimized to 1-dimensional array

-according to the previous optimization Case, draw a 2 rows grid -find that dp [i] [j] depends on: 1. dp [i-1] [j], 2. dp [i] [j-A [i-1]] -where : dp [i-1] [j] is the settlement result of the previous round (i-1), it must be already calculated, ready to be used -However, after we i ++, j ++, before row = i- 1, col <j, all of them are not needed.- Dimension reduction and simplification: We only need to keep the dimension dimension of weigth, and the dimension dimension of i can be omitted:-(i-1) row is just the old value calculated before: Each round, j = [0 ~ m], then dp [j] itself has the function of recording the old value. -It becomes a 1-bit array -The focus of the dimensionality reduction optimization: look at the left and right calculation direction of the two lines -Time (mn). Space (m)

---

40 [First Missing Positive.java] (https://github.com/awangdev/LintCode/blob/master/Java/First%20Missing%20Positive.java) Level:. Hard Tags: [Array]

to a Unordered strings, negative numbers: Find the first missing positive integer

missing positive integer in this array. It is actually [1, n] for comparison.

Array analysis, index trick

-use while loop, keep Try to send the number to where it should be placed -if index = nums [i] exceeds nums.length, of course it will not move -note: check val! = Nums [val], avoid infinitely loop -check: nums [i] Is it equal to i, if it is not right, the result is found

Edge Case

-If nums == null, in fact missing positive integer is naturally 1 -When validation, there may be no disconnected integer in this string of numbers, but the maximum The integer is in the first position (because the index exceeds the standard, it cannot be placed in the correct place) -at this time, n is placed at index 0, in fact, the next integer should be n + 1 -In the end, if the array is completely sorted, it is not lacking, and it also meets the conditions of the subscript, then the only next is the first positive outside the array range number: n

---

41. [N-Queens.java] (https://github.com/awangdev/LintCode/blob/master/Java/N-Queens.java) Level: Hard Tags: [Backtracking]

N-Queen Question , Give the numbers n, and nxn board, find all N-queens answers.

Backtracking

-find all situations with dfs, each iteration, pick the appropriate point from the line, dfs -add the selected point to the candidate In the list, remember to backtracking. -Each candidate needs validation, check if row, col, 2 diagnal is queen

validate n queue at certain (x, y) -1. There

must be no target row in the array #

-
    2. diagnal. Remember the formula:-row1
- row2 == col1-col2. Diagnal elelment.fail
- row1-row2 ==-(col1-col2). Diagnal element. Fail-
- Draw a 3x3 board to test the 2 scanarios: (0,0) and (3,3) are diagnal- (0,2) and (2,0) are diagnal

---

42. [N-Queens II.java] (https://github.com/awangdev /LintCode/blob/master/Java/N-Queens%20II.java)### Level: Hard Tags: [backtracking]

with N-Queens, like, not looking for all the results, but how many results COUNT.

backtracking

- When list.size () == n, it means that a solution was found.
-
    1. dfs function (List , n) -2. validate function

---

43. [LRU Cache.java] (https://github.com/awangdev/LintCode/blob/master/Java/ LRU% 20Cache.java) Level: Hard Tags: [Design, Hash Table, Linked List]

Double Linked List

-A special bidirectional ListNode is used, with head and tail, which greatly speeds up speed.
-The main thing is to speed up the process of updating the ranking, find the item hashmap O (1), and perform subtraction and transposition are O (1) -Overall O (1)

Ingenious point

Kinda, Tree DP

-1. head And tail are particularly clever: removing heads and tails, and adding heads and tails, are particularly fast.
-2 . Use two-way pointers: pre and next. When you need to remove any node, you just need to know which one to remove.-Just connect patient.pre and node.next patiently, and the node will be natural. Do not disconnect.
-Once you know how to solve it, it is not very special and not difficult to write: -moveToHead ()
-insertHead ()-remove
()

O (n) Check for duplicates

-timeout method, naive O (n) solution, and the result is indeed timeout.
-A map <key, value> stores the value. A queue to hold the rank.
-Every time there is an update, put the latest one at the end; every time you exceed the capaticity, kill the big head. Very simple, but it took too long to run and failed.

---

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

Find max path sum, from any treeNode to any treeNode.

-Two cases: 1. combo sum: left + right + root; 2. single path sum -Note1: the path needs to be continuous, curr node cannot be skipped -Note2: what about I want to skip curr node: handled by lower level of dfs (), where child branch max was compared. -Note3: skip left / right child branch sum, by comparing with 0. If it is less than 0, it is not necessary to record the idea of #### DP -tree gives us 2 branch, each branch is similar to dp [i-1], here is similar to dp [left], dp [right] this way -after finding dp [left], dp [right], combine with curr node. -because it is looking for max sum, and can skip nodes, so the global variable max is required -each time dfs () returns must be a path that can continue continuously link ', so return one single path sum + curr value`.

DFS, PathSum object

-that just solves everything

---

45. [Basic Calculator.java] (https://github.com/awangdev/LintCode/blob/master/Java/Basic%20Calculator.java) Level: Hard Tags: [Binary Tree, Expression Tree, Math , Minimum Binary Tree, Stack]

Give an expression String, and evaluate the value of the expression. The expression

string includes +,-, integers, opening and closing brackets, and space.

Expression Tree

-Expression Tree is a weight- based min-tree-tree based on arithmetic symbols + numbers: the numbers are always in the leaf, then the symbol is the tree node, the brackets do not appear in the tree -use monotonuous stack to build this tree

Thinking points

-Understand Expression Tree -Use stack to build the expression tree + understand the weight system -Use post-order traversal to evaluate the tree -Note that the number in the input will not be a single digit, so a buffer is needed to store the number string -For the practice of the entire topic, you can refer to Expression Evaluation

---

46. [Longest Consecutive Sequence.java] (https://github.com/awangdev/LintCode/blob/master/Java/Longest%20Consecutive%20Sequence.java) Level: Hard Tags: [Array, Hash Table, Union Find]

Give a string of numbers, unsorted, find the length of the sequence of consecutive elements in the string (consecutive sequence, it is a continuous number, not to say that the original order)

### HashSet

-To see continuous elements, you must use num ++, num-- search like this -1. need O (1) to find the element -2. need to quickly and easily find num-1, num + 1. -if you use min, max open array, consume space -use HashSet to save, use set.contains () to find whether num-1, num + 1 exists or not -for loop. O (n) -while loop in the general will not have O (n); Once O (n), it also means that the set is cleared, and there will be no more inner while derivatives in the for loop.-Overall O (n) time complexity

Union Find

- The final element is linked should count the total length, in fact, is to group elements together, group connected together, so think UnionFind
- this uses a int [] size to help when dealing with merger of parent which issues: forever large group of union go
- main function inside, there is a map to track, each element handles only 1 times.
- Contents of union: current number-1, current number + 1
- https:

//www.jianshu.com/p/e6b955ca208f ##### Features -Union Find seems easier to do on index -Other union find function: boolean connected (a, b) {return find (a) == find (b)}

---

47. [Serialize and Deserialize Binary Tree.java] (https://github.com/awangdev/LintCode/ blob / master / Java / Serialize% 20and% 20Deserialize% 20Binary% 20Tree.java) Level: Hard Tags: [BFS, DFS, Deque, Design, Divide and Conquer, Tree]

Serialize and Deserialize Binary Tree

DFS , Divide and Conquer

Serilize

-Divide and conquer: Pre-order traversal to link all nodes together -build the string data: use '#' to represent null child.

- the preorder string, can be parsed apart by `split(',')`

Deserialize

- Use a list (here we use `Deque` for the ease of get/remove in 1 function: remove())
- to take all parts of the parsed sring data: dfs on the Deque
- first node from the list is always the head
- '#' will be a null child: this should break dfs
- Deque is a global variable, so dfs(right child) will happen after dfs(left child) completes

DFS, Recursive [previous note]

- serilize: divide and conquer, pre-order traversal
- deserialize: 稍微复杂, 用dfs. 每次要truncate input string:
- 一直dfs找left child, 接着right child until leaf is found.
- Use a StringBuffer to hold the string, because string is primitive, we need to pass reference
- traverse nums [] from the end i = n-1

BFS, Non-recursive

-using queue. The idea is intuitive. level-order traversal. Save to a string. -When encountering a null child, instead of ignoring it directly, you assign an Integer.MIN_VALUE, and then mark as '#' -BFS needs track queue size, each time only a specific number of nodes

---

48. [Count of Smaller Numbers After Self.java] (https://github.com/awangdev/LintCode/blob/master/Java/Count%20of%20Smaller%20Numbers%20Afte Level: Hard Tags: [BST, Binary Indexed Tree, Binary Search, Divide and Conquer, Segment Tree]

gives a string of numbers nums [], find a new array result, where result [i] = # of smaller items on right of nums [i]

Binary Search

-sort and insert into a new list, the new list is sorted -each time insert nums [i] enters the list, it is # of smaller items on right side of nums [i]-each time results [i] is recorded

- 问题: 这里的binary search 是用 `end = list.size(); while(start<end){...}`做的, 可否换成用`end=list.size() - 1`?

Segment Tree based on actual value

- Build segment tree based on min/max values of array: set each possible value into leaf
- query(min, target - 1): return count # of smaller items within range [min, target - 1]
- Very similar to `Count of Smaller Number`, where segment tree is built on actual value!!
- IMPORTANT: goal is to find elements on right -> elements processed from left-hand-side can be removed from segment tree
- Use `modify(root, target, -1)` to remove element count from segment tree. Reuse function
- time: `n * log(m)`, where m = Math.abs(max-min). log(m) is used to modify() the leaf element

Segment Tree solution - tricky part:

-negative nubmer works oddly with mid and generates endless loop in build (): [-2, -1] use case -build entire segment tree based on [min, max], where min must be> = 0. -we can do this by adding Math.abs (min) onto both min / max, as well as + diff during accessing nums [i]

Binary Indexed Tree

-TODO, have code

---

49. [Remove Duplicate Letters.java] (https://github.com/awangdev/LintCode/blob/master/Java/Remove%20Duplicate%20Letters.java) Level: Hard Tags: [Greedy, Hash Table, Stack]

Hash Table, Greedy

- count[] = int[256], 不需要 c-'a'
- boolean visited []: Once a letter has fixed its position, when it meets again, it skips the used character directly
- if the tail letter can become smaller, then delete the tail and reconnect with the new letter (Prerequisites: The removed letter will appear again, set visited [tail] = false)
- Space: O (1) count [], visited [].
- Time: Go through all letters O (n)

Stack

-Use stack instead of stringBuffer: keep append / remove last added item -However, stringBuffer appears to be faster than stack.

---

50. [Expression Add Operators.java] (https://github.com/awangdev/LintCode/blob/master/Java/Expression% 20Add% 20Operators.java) Level: Hard Tags: [Backtracking, DFS, Divide and Conquer, String]

Give a number String, the numbers come from `0-9`, give 3 operators`+`,`-`,`*`, See how to piece together, can produce results target. Output

all expression

string dfs, use list to track steps (backtracking)

-related to string, it may be a little tedious to write -numbers have dfs ([1,2,3 ...]) combination methods -operator has [`+`, `-`,`*`] 3 combination methods -Note 1: The multiplication sign must be specially processed, and the numbers along the multiplication are passed. When calculating the next product, sum-preProduct + product -Interval teardown point, PriorityQueue point -Note 2: '01' is a skip number.-Note 3: The first selected number does not need to be added, it is added directly -Time: O (4 ^ n), Space: O (4 ^ n) -T (n) = 3 * T (n-1) + 3 * T (n-2) + 3 * T (n-3) + ... + 3 * T (1); -T (n-1) = 3 * T (n-2) + 3 * T (n-3) + ... 3 * T (1); -Thus T (n) = 4T (n-1) = 4 ^ 2 * T (n-1) = ... O (4 ^ n)

String dfs, use string as buffer

-the logic is the same, the code is shorter, but instead of doing a list, pass `buffer +" + "+ curr` directly -because a new string is created each time, it is slightly slower One point. Same time complexity

---

51. [Insert Interval.java] (https://github.com/awangdev/LintCode/blob/master/Java/Insert%20Interval.java) Level: Hard Tags: [Array, PriorityQueue, Sort]

Sweep Line

-count == 0 is used as the judgment point during Merge -Note, be sure to compare curr `px == queue.peek (). X` to ensure that all overlapping points are processed. : `count + = px` -PriorityQueue: O (logN). Scan n points, total: O (nLogn)

Basic Implementation

-### sorted intervals have been given here by start point. -Directly find a place where insert newInterval can be inserted.-Insert -then loop to merge entire interval array -Because it is given as a list, it is convenient for `intervals.remove (i)` -Before remove, assgin `pre.end` will be reasserted to ensure it is removed The node.end is captured -O (n)

Also

-because interval has been sorted, I wanted to use Binary Search O (logn).- But to find the interval insert position, the final merge still uses O (n), so not Required binary Search

---

52. [Shortest Palindrome.java](https://github.com/awangdev/LintCode/blob/master/Java/Shortest%20Palindrome.java) Level: Hard Tags: [KMP , String]

Divide by mid point, Brutle

-check (mid, mid + 1), or (mid-1, mid + 1).

- If the two position matches, that is a palindrome candidate
- 比较front string 是否是 end string 的substring
- O(n^2)
- timeout on last case: ["aaaaaa....aaaacdaaa...aaaaaa"]

KMP

- TODO

---

53. [K Empty Slots.java](#)### Level: Hard Tags: [Array, BST, TreeSet]

题目解析后: find 2 number, that: 1. k slots between the 2 number, 2. no slots taken between the two number.

BST

- BST structure not given, use TreeSet to build BST with each node
- Every time find last/next inorder element
- `treeSet.lower(x)`, `treeSet.higher(x)`
- Once the positions are separated by (k + 1), the title conditions are met
- O (nlogn), good enough

Track slots of days

-Reverse the array, save days index into days [], where the new index is slot. -Days [i]: at slot i, which day a flower will be planted -O (n) -Needs to understand: http://www.cnblogs.com/grandyang/p/8415880.html

---

54. [Count of Range Sum.java](https://github.com/awangdev/LintCode/blob/master/Java/Count%20of%20Range%20Sum.java) Level: Hard Tags: [BST, Divide and Conquer, Merge Sort, PreSum]

TODO : Write the code + merge function

Divide and Conquer + PreSum + MergeSort

-The algorithm is very powerful: presum [], then sum range [i, j] is equal to preSum [j + 1]-preSum [i] -Divide and conquer: Consider the results in [start, mid] range, and then the results in [mid, end] range. (MergeSort separately) -Finally consider the overall result of [low, high] -The premise of the n, m method above is feasible: preSum [] has two range [low, mid], [mid, high] before and after sorted -Tip: PreSum is done (n + 1) length, then finding range sum [i, j] can be simplified to preSum [j]-preSum [i] -NOTE: should write merge () function, but that is minor, just use `Arrays.sort (nums, start, end)`, OJ passed -Every mergeSort () has a for loop => O (n log n)

#### How to count range?

-Here is a special method: find a i in [low, mid], and compare the preSum after mid (explain from: https://blog.csdn.net/qq508618087/ article / details / 51435944)-find two boundaries in the right-hand array, set to `m, n`,-where m is the first in the right-hand array such that `sum [m]-sum [i]> = lower` Position, -n is the first position that makes `sum [n]-sum [i]> upper`, -so that `nm` is the interval in the range of [[lower, upper] `formed by the left element i . the number

magical focus: Why the Sort and Merge

- border [lower, higher] good for comparison sorted array, once the borders, we can stop the calculation, reducing unnecessary computing.
- in other words, When recursively mergeSort (), you really need to merge sorted 2 partitions
- you may ask: Can you sort it? Sort will soon disrupt the order? Yes, the order of preSum [] is disrupted.
- But it doesn't matter: very clever, when dividing and conquering, the first half / the second half are separated and the process is completed with the original order retained, and finally merged.
- When doing the range of m, n, the principle is as follows, such as preSum Divided into two sections: `[A, B, C]`, `[D, E, F]`
- each preSum value `A` when comparing with preSum [i] `A-preSum <lower`, All are single comparisons, not involving B, C
- so it doesn't matter whether `[A, B, C]` retains the order of the initial preSum at this time-the most important thing at this time is that [[A, B, C] `and sorting are good, then when the` lower` boundary is compared, once the boundary is crossed, the calculation can be stopped (reduce unnecessary calculations)

BST

-TODO?

---

55. [Max Sum of Rectangle No Larger Than K.java](https://github.com/awangdev/LintCode/blob/master/Java/Max%20Sum%20of%20Rectangle%20No%20 Level : Hard Tags: [Array, BST, Binary Search, DP, Queue, TreeSet]

Given a non-empty two-dimensional matrix matrix and an integer k, find the sum of the largest rectangle in the matrix that is not greater than k.

BST, Array, preSum

- Reduce the problem to: row of values, find 1st value> = target. -

1. loop over startingRow; 2. loop over [startingRow, m-1]; 3. Use TreeSet to track areas and find boundary defined by k.

-When building more rows / cols the rectangle, total sum could be over k: -when it happens, just need to find a new starting row or col, -where the rectangle area can reduce / remain <= k -find the starting point of the excess area: extraArea = treeSet.ceiling (totalSum-k). That is, find the starting / left area after subtracting k. -remove these left starting areas, the rest Just <= k. (Num-extraArea)-Why use TreeSet: the size of the area is irregular, and find the first value of> = any value. Give a list of non-sorted numbers, find the number of> = target, if Do not write binary search, then BST is most suitable -O (m ^ 2 * nlogn)

Idea

-From the most basic O (m ^ 2 * n ^ 2), consider: traversing startingRow / startingCol -rectangle? Layer by layer? You can think of the idea of Presum, when the sum is greater than the required sum, subtract the excess part -how to find the excess Area? So is search: save the content you need to search, you can think of using BST (TreeSet), or write Binary Search yourself.

---

56. [Perfect Rectangle.java] (https://github.com /awangdev/LintCode/blob/master/Java/Perfect%20Rectangle.java)### Level: Hard Tags: [Design, Geometry, Hash Table]

See if the list of coordinates can form a perfect rectangle, and overlap area is not allowed.

### Drawing Features

-Feature 1: All the given points (and then find the diagonal points without the specification), if the perfect rectangle is formed at the end, they should be eliminated from each other, and finally 4 corners are left -Feature 2: Find The min / max (x, y) in all points, and the final maxArea, should be equal to the area accumulate in the process -Feature 1 Make sure that there is no hollow part in the middle, ensure that all coincident points will be eliminated from each other, and 4 are left Vertex -Feature 2 ensures no coincidence: The coincident areas will eventually exceed maxArea

---

57. Max Points on a Line.java### Level: Hard Tags: [Array, Geometry, Hash Table, Math]

给list of (x,y) coordinates. Determine # of points on the same line

Observation

- If given n points, we can calculate all possible slopes. O(n^2) times
- For the two dots that generates the same slope, these dots could be on parallel### slopes
- figure out how to prune the parallel dots

Trick: prune parallel dots using greatest common divider

- GCD: greatest common divider
- Devide the x and y by their greatest common divider, such that x and y can be reduced to minimum value
- All other x and y can be reduced to such condition as well
- track the final reduced (x,y) in a map: they are the key to the count
- No need to use Map<Integer, Map<Integer, Integer>> to perform 2 level mapping; just `map<String, Integer>`, where the key is "x@y"

---

58. Number of Digit One.java### Level: Hard Tags: [Math]

Pure math problem, not quite representative

Explanation https://leetcode.com/problems/number-of-digit-one/discuss/64381/4+-lines-O(log-n)-C++JavaPython

---

59. Binary Representation.java### Level: Hard Tags: [Bit Manipulation, String]

String

-First, we need to solve it in two halves. The breakpoint is '.': Str.split ("\.");

Observation

-The half of Integer is easy to handle, in the whie loop: num% 2, num / 2. Make a `parseInteger ()` function -Decimal is more complicated. Make a `parseDecimal ()` function:-bit == 1 math Condition: Now num * 2> = 1. Update: num = num * 2-1; -bit == 0 Mathematical condition: num * 2 <1. Update: num = num * 2

Note

-num is double, decimals are in `num = num * 2 -1` formula may be infinite loop -so check: num repeatability, and binary code <32 bit. -So the problem also has 32BIT requirements!

---

60. [Recover Binary Search Tree.java] (https://github.com/awangdev/LintCode/blob/master/Java/Recover%20Binary%20Search%20Tree.java)## Level: Hard Tags: [BST, DFS, Tree]

There are 2 node misplace in BST , To be classified as: Requirement: O (1) extra space

-BST inorder traversal should give small-> large sequence

- misplaced means: a large->small item would occur, and later a large>small### would occur.
- The first large && second small item are the 2 candidates. Example
- [1, 5, 7, 10, 12, 15, 18]
- [1, 5, 15, 10, 12, 7, 18]

dfs, O(1) extra space

- traverse, and take note of the candidate
- at the end, swap value of the 2 candidates

O(n) space

- inorder traversal the nodes and save in array, find the 2 items misplanced and swap them
- But O(n) space should not be allowed

---

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

- index / i is a forward step by step, each time when i <= range, do a while loop, in which the furthest to go to find a place maxRange

Giving a string of numbers is the distance that can be jumped. Goal: the minimum number of jumps to the last index possible.

Greedy

-always aiming for the `farest can go` -if the `farest can go` breaches the end , return steps -otherwise, send `start = end + 1` , `end = farest` and keep stepping from here -though trying with 2 loops, worst case [1,1,1, ... 1,1] could have O (n ^ 2) -But on average should be jumpping through the array without looking back -time: average O (n)

Previous Notes, Greedy

-Maintaining a range is the farthest we can go.- Then Update range = maxRange -where step is also the same as index, step by step. - The condition of the final check is that we can range as far as you can go> = nums.length-1, indicating that the focus has been reached with the fewest steps. Good.

Even simpler Greedy

-Graphic http://www.cnblogs.com/lichen782/p/leetcode_Jump_Game_II.html-track the farest point -whenver curr index reaches t the farest point, that means we are making a nother move, so count ++ -there seems to have one assumption: must have a solution. Otherwise, count will be wrong number. -In fact, it is exactly the same thinking pattern as the first greedy.

DP

-DP [i]: Record at point i, go to point i The minimum number of jumps -dp [i] = Math.min (dp [i], dp [j] + 1); -condition (j + nums [j]> = i)-Note the use of dp [i] = Integer. MAX_VALUE as the starting value, come to find min -time: O (n ^ 2), slow, and timesout

---

62. [Longest Valid Parentheses.java](https://github.com/awangdev/LintCode/blob/master/Java/Longest%20Valid%20Parentheses.java) Level: Hard Tags: [Coordinate DP, Stack , String]

gives a string of strings with only `( , )` in it. Find the length of the longest valid parentheses.

1D Coordinate DP

-use dp [i] track local max, maintain global max -int [] dp. dp [i]: longest valid string that ends on i.-ends with ')', 2 cases: 1. exactly s [i-1] is '('; 2. s [i] 's') An earlier start (') corresponds -note, if the end is' (' is unreasonable, ignore it.-Init: dp [0] = 0, a single char cannot be formed. -Calculation order: left to right , Find local max, maintain global max -O (n) space, O (n) runtime

Stack

-Store all open / close parentheses in the Stack.-If you encounter stack.top () just open and close , Just stack.pop ().-The rest are unreasonable elements.-A bit like negatively looking for a solution: endIndex-The last failedIndex (stack.pop ())-1 , it should be the length of the last succeded string -Each time you update the endIndex to stack.top (), then continue to find from the stack A failedIndex -compare all the lengths , you can find the longest length -O (n) stack space, O (n) runtime. It should be a bit slower than dp, because O (n) is done 2 times

---

63. [Rearrange String k Distance Apart.java] (https://github.com/awangdev/LintCode/blob/master/Java/Rearrange%20String%20k%20Distance%20A Level: Hard Tags: [Greedy, Hash Table, Heap]

Give a string, all lowercase letters, and ask for rearrangement: Then each unique character must have a distance of k. It is

a bit like Task Scheduler, but there are other methods in Task Find count, this question requires the ranking result

PriorityQueue + HashTable

-A classic usage of PriorityQueue ordering + distribution ordering. -Count frequency and store in pq.-Consume element of pq for k rounds, each time pick one element from queue. -Exception: if k still has content but queue is consumed: cannot complete valid string, return ""; -space, O (n) extra space in sb , O (26) constant space with pq. -Time: O (n) to add all items

---

64. [Valid Number.java] (https://github.com/awangdev/LintCode/blob/master /Java/Valid%20Number.java)### Level: Hard Tags: [Enumeration, Math, String]

time: O (n)

analyzes the edge case, and various cases, and then determines whether it is a valid number

Summary of the situation

-Encountered several different cases of `.`,`e`, +/-, and int -The order of the encounters is different, and the results are different. -This question is more about analyzing the situation, and then edge case enumerate out, meaning algorithm is relatively small.

---

65. [Bricks Falling When Hit.java] (https://github.com/awangdev/LintCode/blob/master/Java/Bricks%20Falling%20When%20Hit.java) Level: Hard Tags: [Union Find]

Give a matrix of 1 and 0, `1` stands for brick. The brick connected to ceiling will not drop. Give a series of coordinate hits [] [], record how many drops will be taken after each take down 1 brick.

UnionFind

-1. We know that most of the bricks may be connected to ceiling, so every forward check is traverse all and timeout -2. Can I use union, install the connections together, and then drop brick When I drop all the connected ones? Difficult: because I still need to check all the current status of the brick. -Inspired by the answers of others, because it is counting, we can think backwards: -mark all hit-bricks = 2 (just discard it), observe the last step of the whole situation, first calculate the total of all bricks still connected to ceiling, and count all the counts in unionFind in count [0].-The remaining ones are not connected The ceilings are also isolated islands -method: add hit-brick one by one, and then do unio again n, see how many are eventually connected to ceiling. The increased count is the number of dropped bricks when thinking forward!

Variation of Union Find

-still use the number index to make the union find, but +1 each index, shift right by one, and [0] is reserved for special purposes: -use union at 0 to count the total remain count of ceiling-connected bricks, where `x = 0`. -If it is on other topics, the condition may not be `x = 0`, but you can also use this union index = 0 to make a root count -the key : Add the last hit brick back, and then re-union around this hit-brick: The increase in count is not the number of drops when the hit-brick is missing.

DFS (timeout)

-consider each hit All around the traverse, all without the ceiling:-For example, a 200 x 200 matrix of all 1s, and the traverse must reach the top every time; the calculation is repeated, so the timeout -the algorithm is correct, but not efficient. -Think To reduce repeated calculations, but not in advance: the grid is constantly changing. So if you can group all connected ceilings, you can O (1) quickly check?

---

66. Interval Sum II.java### Level: Hard Tags: [Binary Search, Lint, Segment Tree]

SegmentTree大集合. Methods: `build`, `query`, `modify`. 不难。只是要都记得不犯错.

Segment Tree

- build: recursively build children based on index-mid and link to curr node
- query: recursively try to find `node.start == targetStart && node.end == targetEnd`. Compare with node.mid
- modify: recursively try to find `node.start == targetStart && node.end == targetEnd`; modify and recursively assign upper interval with updated interval property.

---

67. HashHeap.java### Level: Hard Tags: [HashHeap, Heap]

Non-question. It is the HashHeap implementation found from Chapter 9.

---

68. [Trapping Rain Water II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Trapping%20Rain % 20Water% 20II.java) Level: Hard Tags: [BFS, Heap, MinHeap, PriorityQueue]

Give a 2Dmap, each position has a height. Find Trapping water sum.

Min Heap

-Use PriorityQueue to select the selected Height sorting, for position, create class Cell (x, y, height).

Note a few theories

-1. Consider from the matrix, and find that the water that the matrix can hold depends on the low-height block -2. It must be considered from the outside, because the water is wrapped inside, and at least one layer is required on the outside -the above two points prompt us to use min-heap: that is, PriorityQueue . Of natural order.

Steps

-1. During the process, you can draw a picture to make it clear: that is to walk in all four directions, subtract the height of the surrounding cells from the height of the curr . -2. If it is greater than zero, then the surrounding cells have a product Water: Because the cell is already the lowest on the periphery, so the interior is lower, there must be water accumulation. -3. Each visited cell must be marked, avoid revisit -4. Create a new cell and add it to the queue according to the 4 directions of the position (mX, mY) `: cell (mX, mY) has calculated the accumulated water Later, when the peripheral wall is small,? `(MX, mY)` will become a wall.-5. Because the new fence wall is reduced by one circle, height = Math.max (cell.h, neighbor.h);

- Same idea as trapping water I. Just from the periphery, it can only be added to the horizontal plane with the same height as the peripheral cell. Going inside, it is likely that the cell height changes.
- Attach the maximum height of curr cell and move-to cell here.

Why think of Heap (min-heap-priorityQueue)

-To find the shortest board of the bucket -Each time you need to process the shortest one first (on top)

Why traverse from outside to inside

-wood Bucket theory, water, is from the outside bread to live inside -peel the onion, throw it away

---

69. [Find Median from Data Stream.java] (https://github.com/awangdev/LintCode/blob/ master / Java / Find% 20Median% 20from% 20Data% 20Stream.java) Level: Hard Tags: [Design, Heap, MaxHeap, MinHeap]

Principle

-Think of the input stream as an upward slope. The middle point of the slope is naturally median.-The first half, as maxHeap, the focus is the peak of PriorityQueue, which is the actual
median.-The second half, as minHeap, is normal PriorityQueue. The beginning is minimal.

Attention

-first set here, which queue stores one more element. Here select maxHeap: maxHeap.size () == minHeap.size () + 1 || minHeap.size ()-must be maintained first There is an element in maxHeap, otherwise null will cause problems when comparing sizes.

---

70. [Sliding Window Median.java] (https://github.com/awangdev/LintCode/blob/master/Java/ Sliding% 20Window% 20Median.java) Level: Hard Tags: [Design, Heap, MaxHeap, MinHeap, Sliding Window]

The same problem of Data Stream Median: Not only the increasing sequence, but also remove item (keep a window size)

MaxHeap, MinHeap

-Median still uses min-heap and max-heap. Time (logN) -Add / Subtract: prioirtyQueue, log (n) -findMedian: O (1) -Add a number, subtract a number. -It is optimistic when adding or subtracting, whether it is drawn from the maxheap in the front or the minHeap behind. -Finish the balance

Note -When using maxHeap, minHeap, it is customary to choose one more number for maxHeap:

-The maxHeap on the left always has x + 1 or x numbers -The minHeap behind should always have x numbers

---

71. [Design Search Autocomplete System.java] (https://github.com/awangdev/LintCode/blob/master/Java/Design%20Search%20Autocomplete%20Syste Level: Hard Tags: [Design , Hash Table, MinHeap, PriorityQueue, Trie]

time: input: O (x), where x = possible words, constructor: O (mn) m = max length, n = # of words space: O (n ^ 2), n = # of possible words, n = # of trie levels; mainlay saving the Map <S, freq>

Description is long, but in short: do search auto complete.

Best problem to review Trie (prefix search), Top K frequent elements (Hash Map), and MinHeap (PriorityQueue)

Easier to revisit https://leetcode.com/problems/design-search-autocomplete-system/description/

Thinking direction

-do text search, undoubtedly use Prefix tree, trie.

Find all possible word / leaf, two solutions:

-? After Trie is completed, do a prefix search, and then DFS / BFS return all leaf items. [high runtime complexity] -Store all possible words in TrieNode. [high space usage] -Should n't it be in memory space? Big question, so we can choose store all? possible words

Given k words, find top k frequent items. MinHeap is definitely used, but there are two solutions:

-? Store MinHeap with TrieNode: Because it will continue to search for new articles, the same prefix (especially at the higher level) will be searched multiple times. -[? complexity: need to update heaps across all visited TrieNodes once new sentence is completed] -Compute MinHeap on the fly: Of course we can't come to a DFS every time? Otherwise it will be slow, so we must store list of possible candidates in TrieNode. we use Map <String, freq> in Top K Frequent Words, so O (m) constructs min-heap It's actually very convenient.

Train the system

-Each time a # is marked, an entry is added to the search history. Then insert it into trie. -This one meets # again at the end do it, very simple

Trie, PriorityQueue, HashMap

- Trie the Prefix Search Top k + Maintain Frequent items
-

. 72 [Integer to English Words.java] (https://github.com /awangdev/LintCode/blob/master/Java/Integer%20to%20English%20Words.java)### Level: Hard Tags: [Enumeration, Math, String]

Give a number less than Integer.MAX_VALUE (2 ^ 31-1) , Convert to English. (No need to add 'and')

String

-Basic implementation - Classification discussion : thounsand, million, billion. 3 digits and one division . Enumerate tokens with array -Use% and / to find English translation of each segment -3-digit part, you can use a helper funtion to find the result, the processing method of each paragraph is the same

Note

-StringBuffer is more efficient! Sb.insert (0, xxx) append in front of sb -When adding "", if necessary, try "trim ()" -Note, numbers less than 20 have their own special writing , Need extra handle -this topic is to be careful and patient, there is almost no algorithm, just to write efficiently and correctly, you need to be careful

---

73. [Alien Dictionary.java] (https: / /github.com/awangdev/LintCode/blob/master/Java/Alien%20Dictionary.java)### Level: Hard Tags: [BFS, Backtracking, DFS, Graph, Topological Sort]

Give an array of strings: if this array is According to a new alphabet dictionary, you need to find this alphabetical order.

There may be multiple sorting methods, and you can give one.

Graph

-Essence: two strings above and below, relative Corresponding to the same index, if the letters are different, the word on the first line is explained More ahead in the alphabet -Turn string array into topological sort graph: map <char, list <char >> result.- ex: cycle nodes from input, where inDegree of a one node would never reduce to 0, and will not be added to result -also List [26] edges (Course Schedule problem) -Build edges: find char diff between two rows, and store the order indication into graph -Note: indegree is always reversed (established in the opposite way to node to neighbors)

BFS

-topological sort itself is well written, But first understand the nature of alphabetic sorting in the title-in fact , the nature of the above sorting is very imaginary , but it will be a bit difficult to think of it as the code for constructing the graph -calculate indegree, and then use BFS to find those inDegree = = 0 node -the first node with inDegree == 0 is listed in front of the alphabet. -The following solution uses Graph: map <Character, List > instead of List [26], but it is actually more trial Dictionary with more than 26 letters.-If inDegree.size ()! = Result.length () , there is nodes that did not make it into result. -In this case, it will be treated as invalid input, and return ""

DFS

-The process of creating grpah is exactly the same as BFS -DFS differs in that it uses a visited map to mark the places you have passed -When you reach the leaf, add to result: but only add it because you have reached the end, the final order should be reversed (or, sb.insert (0, x) add directly in reverse order)

---

74. [Word Ladder II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Word%20Ladder%20II.java) Level: Hard Tags: [Array, BFS, Backtracking, DFS, Hash Table, String]

gives a string of string, start word, end word. Find the shortest path list from startWord-> endWord.

Variation: mutate 1 letter at a time.

BFS + Reverse Search

-Find the shortest with BFS Path. -Question: how to effectively store the path, if the number of paths are really large?
-If we store Queue <List >: all possibilities will very large and not maintainable -Make a reverse structure with BFS, and then reverse search

BFS Prep Step

-BFS find all start strings can Where you go, put it in an overall structure: Note, the way to save Map <s, list of sources> -BFS changes by 1 step each time, so recording a distance is actually the shortest path candidate (stop here)-1 . Reverse mutation map: `destination / end string-> all source candidates` using queue:`Mutation Map` -Mutation Map <s, List >: list possible source strings to mutate into target key string.-2. Inverse distance map: `destination / end string-> shortest distance to reach dest` -Distance Map <s , possible / shortest distance>: shortest distance from to mutate into target key string. -BFS prep step didn't solve the problem, not even the end string. We want to use the reverse mapping structure built by BFS, do search

## Search using DFS

-Scan from the end string, find all candidate dates && only visit candidate that is 1 step away -dfs until you find the start string.

Bi-directional BFS: Search using BFS

-overall for loop; clean up list: 1. over size; 2. last item -The reversed structure is ready, now you can search: you can also use bfs.- `Queue <List <String >>` to store candidates, searching from end-> start

---

75. [Text Justification. java] (https://github.com/awangdev/LintCode/blob/master/Java/Text%20Justification.java) Level: Hard Tags: [Enumeration, String]

Adjust text according to the rules. It is in Word: there is one line Too long, adjust the space in the middle of the word, and then ensure the top width of the total width of each line.

There are some details rules, see the original title

String

-Summing space = `width + (size-1)`. Maintain: 1. list of candidates, 2. width of actual words -calculate space in between: `remain / (size-1)` -not difficult at all, but be careful: deal with list of string, pay attention to clean sum size of list , just fine.- `Clear processing space`: only (n-1) items are processed, then the last one is taken out of the for loop, special processing.

Notes

- Clarification, observation:
- can start with greedy approach to stack as many words as possible

- once exceed the length, pop the top, and justify the added words (untouched words tracked by index)
- left justify: given list/stack of words with size t, overall remaining space length m,
- deal with last line with special care: just fill one space, and fill the rest of the row with space
- Does not seem very complicated, but need additional care of calculating the amount of space needed.
- Overall runtime: O(n) to go over all space
- Overall space O (maxWidth) for maxWidth amount of strings

---

76. [Read N Characters Given Read4 II-Call multiple times.java] (https://github.com/awangdev/LintCode/blob/master/Java/Read%20N%20Characters%20Given%20Read4 -% 20Call% 20multiple% 20times.java) Level: Hard Tags: [Enumeration, String]

Read N Character using `Read4 (char [] buf)` enhanced version: can read continuously read (buf, n)

## String

-Pay attention to the index handle of String, and slowly write the edge case.-Understand the meaning of the title: `read4 (char [] buf)` has a slightly smaller function of `populate input object`. -When you encounter it, understand the function carefully Usage, don't panic. In fact, the way of thinking is very simple, just carefully handle the string and edge case.

---

77. [Frog Jump.java] (https://github.com/awangdev/LintCode/blob/ master / Java / Frog% 20Jump.java) Level: Hard Tags: [DP, Hash Table]

The question of Frog jump needs a little understanding: each grid can jump k-1, k, k + 1 steps, and k depends on the number of steps jumped in the previous step. The default 0-> 1 must be a step.

Note: int [] stones is the unit in which the stone is located (not the number of steps that can be jumped, don't understand it wrong).

DP

-originally wanted to do it according to corrdiante dp, but found many problems, need to track different possible previous starting spot .

- according to jiuzhang answer: by definition, a use of the Map <stone, the Set <# Possible Steps to the REACH stone >>
- at a time when processing a stone, all according to his own set of , to go Next three steps: k-1, k, or k + 1 steps.
- Take one step each time to see if stone + step exists; if it exists, add it to the hash set of next position: `stone + step`

## Note init

- `dp.put (stone, new HashSet <> ())` mark The existence of each stone - `dp.get (0) .add (0)` init condition, used to do dp.put (1, 1)

Idea

-finally do the thinking mode, more like the BFS mode: starting from (0,0), add all possible ways -then again, try next stone with all possible future ways ... etc

---

78. [Longest Substring with At Most Two Distinct Characters.java] (https://github.com/awangdev/LintCode/blob/master/Java/Longest%20Substring%20with%20At%20Mos .java) Level: Hard Tags: [Hash Table, Sliding Window, String, Two Pointers]

as the title.

Two Pointer + HashMap

-originally wanted to use DP, but its practical sliding window idea -sliding window cutting : Use hashmap to store last occurrance of char index; -After map.remove (c), it is equivalent to cut out that section completely; then map.get (c) + 1 is the new left window border

---

79 [Shortest Distance from All Buildings.java] (https://github.com/awangdev/LintCode/blob/master/Java/Shortest%20Distance%20from%20All%20Buil Level: Hard Tags: [BFS]

It is very similar to Walls and Gates, except that this question needs to choose a coordinate, having shortest sum distance to all buildings (marked as 1).

BFS

-BFS can mark shortest distance from bulding-> any possible spot . -Try each building (marked as 1)-> BFS cover all 0. -time: O (n ^ 2) * # of building; use new visited [] [] to mark visited for each building. -O (n ^ 2) find the smallest point / aggregation value.-Note, this problem we update grid [] [] sum up with shortest path value from building. -Finally find a min value, even without return coordinate. -Analysis, Not written yet.

---

80. [Sliding Window Maximum.java] (https://github.com/awangdev/LintCode/blob/master/Java/Sliding%20Window%20Maximum.java) Level: Hard Tags: [Deque, Heap, Sliding Window]

Deque, Monotonous queue

-maintain monotonuous queue: one end is always at max and the other end is min. Always need to return the max end of queue. -When adding new elements x: start from small-end of the queue, drop all smaller elements and append to first element larger than x.-when sliding window: queue curr window max-end, remove it if needed. -wonderful: use deque data structure (actually using LinkedList's Form) to make a `decreasing queue`. -Each time smaller than the current node, all are eliminated, and the rest is naturally: the largest> the second largest> the third largest ... ETC. -We only It cares about the existence of the maximum value; any value that is less than the current value (which is added when it is about to be new) will not become the maximum value anyway, so throw it away!

---

81. [Median of Two Sorted Arrays.java] (https://github.com/awangdev/LintCode/blob/master/Java/Median%20of%20Two%20Sorted%20Arrays.ja Level: Hard Tags : [Array, Binary Search, DFS, Divide and Conquer]

Famously find the median of two sorted arrays. Definition of median: If the total length of the two arrays is even, take the average. The problem requires the log (m + n) Time to solve

-see log (m + n), I think of binary search, or recursive chop half each time -the two sorted arrays are uneven, and certainly can not be simple binary search

Divide and Conquer, recursive

-Here is a mathematical exclusion idea: consider the intermediate points of A and B. -If A [mid] <B [mid], then A [0 ~ mid-1] is not in the range of median and can be excluded. divide / conquer is like this.-For the specific logic, look at the code, which roughly means: compare the positions of A and B [x + k / 2-1] each time, and then do the range exclusion method -end cases: -1 . If We found that the start index of A or B in dfs () overflowed, then this is the simplest case: midian must be in another array -2. If k == 1: find 1st item in A / B, then make `Math.max (A [startA], B [startB])` -the total number length is (m + n) And every time the general content is deleted, then time is O (log (m + n))

---

82. [Bus Routes.java] (https://github.com/awangdev/LintCode/ blob / master / Java / Bus% 20Routes.java) Level: Hard Tags: [BFS]

---

83. [Sliding Puzzle.java] (https://github.com/awangdev/LintCode/blob/master/Java /Sliding%20Puzzle.java)### Level: Hard Tags: [BFS, Graph]

84. [Cracking the Safe.java] (https://github.com/awangdev/LintCode/blob/master/ Java / Cracking% 20the% 20Safe.java) Level: Hard Tags: [DFS, Greedy, Math]

Greedy, Iterative

-For 2 passwords, the shortest situation is both passwords overlap for n-1 chars.

- We can use a window to cut out last (n-1) substring and append with new candidate char from [k-1 ~ 0]
- Track the newly formed string; if new, add the new char to overall result
- Note: this operation will run for k^n times: for all spots of [0 ~ n - 1] each spot tries all k values [k-1 ~ 0]
- Same concept as dfs

DFS

- Same concept: use window to cut out tail, and append with new candidate
- do this for k^n = Math.pow(k, n) times

---

85. Redundant Connection II.java### Level: Hard Tags: [DFS, Graph, Tree, Union Find]

Union Find

- 讨论3种情况
- http://www.cnblogs.com/grandyang/p/8445733.html

---

86. The Maze III.java### Level: Hard Tags: [BFS, DFS, PriorityQueue]

BFS

- 跟 Maze I, II 类似, 用一个 Node[][] 来存每一个(x,y)的state.
- Different from traditional BFS(shortest path): it terminates BFS when good solution exists (distance), but will finish all possible routes
-
    1. Termination condition : if we already have a good/better solution on nodeMap[x][y], no need to add a new one
-
    2. Always cache the node if passed the test in step1
-
    3. Always offer the moved position as a new node to queue (as long as it fits condition)
-
    4. Finally the item at nodeMap[target.x][target.y] will have the best solution.

---

87. [Regular Expression Matching.java] (https://github.com/awangdev/LintCode/blob/master/Java/Regular%20Expression%20Matching.java) Level: Hard Tags: [Backtracking, DP, Double Sequence DP, Sequence DP, String]

Like WildCard Matching, discuss the situation clearly. String p last char is '' *and not* ''

. The difference here is that '*' needs a precedent element, then:

- repeat 0 times
- repeat 1 times: need s [i-1] match with prior char p [i-2]

---

88. [Wildcard Matching.java] (https://github.com/awangdev/LintCode /blob/master/Java/Wildcard%20Matching.java)### Level: Hard Tags: [Backtracking, DP, Double Sequence DP, Greedy, Sequence DP, String]

Double sequence DP. Similar to regular expression.

Double Sequence DP

-Analyze the true meaning of the characters?, * And write out the expression. -Dp [i] [0] should always be false when initializing. When p is an empty string, it cannot match anyway (unless s = "" as well) -At the same time dp [0] [j] may not be false For example, s = "", p = "*" is a match.
-A. p [j]! = '' *1. last index match =>dp[i - 1][j - 1] 2. last index ==? => Dp [i-1] [j-1] -B. p [j] ==* "" 1. * is empty => dp [i] [j-1] 2. * match 1 or more chars => dp [i-1] [j]

---

89. [Robot Room Cleaner.java] (https: //github.com/awangdev/LintCode/blob/master/Java/Robot%20Room%20Cleaner.java)### Level: Hard Tags: [Backtracking, DFS]

DFS

-Different from regular dfs to visit all, the robot move () function need to be called, backtrack needs to move () manually and backtracking path shold not be blocked by visited positions

- IMPORTANT: Mark on the way in using set, but backtrack directly without re-check against set
- Mark coordinate 'x@y'
- Backtrack: turn 2 times to revert, move 1 step, and turn 2 times to revert back.
- Direction has to be up, right, down, left.
- int [] dx = {-1, 0, 1, 0};, int[] dy = {0, 1, 0, -1};

---

90. Maximum Vacation Days.java### Level: Hard Tags: [DP]

Review (5)

0. Maximum Subarray III.java### Level: Review Tags: []

---

1. [Valid Perfect Square.java] (https://github.com/awangdev/LintCode/blob/master/Java/Valid%20Perfect%20Square.java) Level: Review Tags: [Binary Search, Math]

Binary finds sqrt. Basic mid + 1, mid-1 template. Note: define index as long.

---

2. [Maximum Average Subarray II.java] (https://github.com/awangdev/LintCode/blob/master/Java/Maximum%20Average%20Subarray%20II.java Level: Review Tags: [Array , Binary Search, PreSum]

gives int [] nums and window min size k. The window size can be greater than K. Find the largest continuous number average

value.-The idea of Binary Search is used on the average sum you are looking for. The size is in [min, max] -When looking for k, it can be> = k, use the concept of min (preSum).-When looking for k, draw a picture, it can be seen that what is actually in the k window sum [x, i], so to use sum [0, i]-sum [0, x]

needs to read the following notes carefully.

---

3. [The Skyline Problem.java] (https://github.com/awangdev/LintCode/blob/master/Java/The%20Skyline%20Problem.java) Level:

Review Tags: [Binary Indexed Tree, Divide and Conquer, Heap, PriorityQueue, Segment Tree, Sweep Line] is

also called skyline. O (nLogN) made with Sweep Line, but it seems that there are many ways: segement tree, hashheap, treeSet?

Sweep Line, Time O (nLogN), Space O (n)-original

reference http://codechen.blogspot.com/2015/06/leetcode-skyline-problem.html?_sm_au_=isVmHvFmFs40TWRt-Drawing analysis: Need to find non-overlaping height point at current index; also height needs to be different than prev height peek to be visible. -Divide all points out, each point has index x, plus a height.
-Sort on this list, according to index and height. Note Mark building start point height with a negative number, so that start is guaranteed before end -Mark start with a negative height: When comparing startPoint.height, endPoint.height with the same x-pos in the priority queue, because the end height is an integer, the start point will be automatically placed in front of the end point during the compare when comparing-Of course, if the two start points are compared, the negative value of the second point is too large (that is, the height is high), it will be smooth. Manage compare to return positive numbers, forming inverses into chapters-use max-heap (reversed priorityqueue) in processes, then iterate heightPoints to store the maximum height. When peek is encountered, it is a reasonable solution
-Add a 0 to heightQueue to close it at the end

Segment Tree

-After reading some practices, the segment tree is complicated to write. It is estimated that it is difficult to write the segment tree in the interview: https: // www. cnblogs.com/tiezhibieek/p/5021202.html

HashHeap

-HashHeap template can be considered: https://www.jiuzhang.com/solution/building-outline/#tag-highlight-lang-java

Binary Indexed Tree?

---

4. [Remove Invalid Parentheses.java] (https://github.com/awangdev/LintCode/blob/master/Java/Remove%20Invalid%20Parentheses.java) Level: Review Tags: [BFS , DFS, DP]

Give a string with parentheses and other characters. Cut out the valid string with the least knife, find all such strings.

There are multiple solutions to this problem, the strongest is O (n) space and time

# DFS and reduce input string

-in dfs: remove the incorrect parentheses one at a time

- detect the incorrect parentheses by tracking/counting (similar to validation of the parentheses string): `if(count<0)`
- once detected, remove the char from middle of s, and dfs on the rest of the s that has not been tested yet.

Core concept: reverse test

- if a parenthese string is valid, the reverse of it should also be valid
- Test s with open='(', close=')' first; reverse s, and test it with open=')', close='('

Minor details

- only procceed to remove invalid parenthese when `count<0`, and also break && return dfs after the recursive calls.
- The above 2 facts eliminates all the redundant results.
- Reverse string before alternating open and close parentheses, so when returning final result, it will return the correct order.
- Open questions: how does it guarantee minimum removals?

Backtracking -If a stringbuffer is used, it will not create a new string every time, but if the string buffer is maintained, it will be backtracking

Complexity

-Seems to be O (n), but need to derive

BFS

TODO

DP

Hard (91)

0. Count of Smaller Number before itself.java### Level: Hard Tags: []

Very similar to Count of Smaller Number. The actual value is used to form the segment tree, and the leaf is stored (count of smaller number).

Trick: Query first, then modify.
Each time Query, A [i] has not been added to the Segment Tree, and A [i + 1, ... etc] has not been added yet.
Then naturally it is coutning smaller number before itself.
Tricky!

Also note:
In modify: Check root.start <= index and index <= root.end. It was ignored in the past. You can also write this later.
(In fact, it is Make sense, which is to check the index and root.left or root.right more strictly)

---

1. Kth Smallest Sum In Two Sorted Arrays.java### Level: Hard Tags: []

Use priority queue. Each time the smallest expansion, shift. X + 1, or y + 1:
Because x and y are the smallest in Min at the moment. So the next smallest is either (x + 1, y), or (x, y + 1).

Just poll () one every time, just put 2 new candidates in. Note that this approach will be repeated, for example the example (7,4) will appear twice. Block it with a HashSet.

Note that the uniqueness of HashSet can be solved by using an "x, y" string.

---

2. LFU Cache.java### Level: Hard Tags: [Design, Hash Table]

Hash Table

-See thoughts specifically, use map in several different ways

- `regular object map` : map of <key, item>, where `item` : `{int val; int count}`
- Use a Map<frequency count, doubly-linked node> to track the frequency
- Track constant capacity, and minimum frequency
- Every get(): update all frequency map as well
- Every put(): update all frequency map as well, with optional removal (if over capacity)
- Original post: http://www.cnblogs.com/grandyang/p/6258459.html
- TODO: one doubly linked list might be good enough to replace below:
- `frequency list map` : map of <frequency count, List>, where the list preserves `recency`
- `item location in frequency map` : map of <key, int location index in list>:
- index relative to the item in a particular list, not tracking which list here

---

3. Prefix and Suffix Search.java### Level: Hard Tags: [Trie]

4. Remove Node in Binary Search Tree.java### Level: Hard Tags: [BST]

Method 1: Brutle a little. Find the target and target's parent.
When removing the target, rearrange the child nodes of the target into a new BST: inorder traversal, build tree based on inorder traversal list.

Method 2: Analyze the rules, first find the target and parent, and then move the children nodes when the target is removed according to the nature, to ensure that it is still BST.

---

5. Subarray Sum II.java### Level: Hard Tags: [Array, Binary Search, Two Pointers]

---

6. k Sum.java### Level: Hard Tags: [DP]

DP. How the formula comes to mind needs to be re-understood.

dp[i][j][m]: # of possibilities such that from j elements, pick m elements and sum up to i. i: [0~target]

dp[i][j][m] = dp[i][j-1][m] + dp[i - A[j - 1]][j-1][m-1] (i not included) (i included)

---

7. Copy Books.java### Level: Hard Tags: [Binary Search, DP, Partition DP]

Give a list of books pages [i], k people, pages [i] represents the number of pages in each book. K people start to copy at the same time from different points.

Q. When can I copy it as soon as possible?

Partition DP

-The first step is to understand the problem required by the title: the first k individuals copy n books and find the least amount of time; it can also be translated into: `n books, let k individuals copy, that is, split into k segments`. -Finally, dp [n] [k] is required. On: int [n + 1] [k + 1]. -Principle: -1. Consider the last step: In [0 ~ n-1] book, the last person can choose to copy 1 book, 2 books ...... n books, the results of each cutting method are different -2. Discuss the situation of the kth person, looping at j = [0 ~ i]. And the slowest case when looping j determines the result of the kth person (barrel principle): `Math.max (dp [j] [ k-1], sum)`. -3. Among them: `dp [j] [k-1]` is the result of [k-1] individual reading the j book, which is also known as the `previous step`. Here the loop considers that the kth person is different j kind of previous step :) -4. The result of the loop is that dp [i] [k] = Math.min (Math.max (dp [j] [k-1], sum [j, i]), loop over i, k, j = [i ~ 0])

- Time: O (kn ^ 2), space O (nk)

Init

-Init: dp [0] [0] = 0, 0 people 0 books -Use of Integer.MAX_VALUE: -When i = 1, k = 1, Expression: dp [i] [k] = Math.min (dp [i] [k], Math.max (dp [j] [k-1], sum)) ; -There is only one case that works: i = 0, k = 0, exactly 0 people copy 0 book, dp [0] [0] = 0. -In other cases, i = 1, k = 0, 0 people read 1 book, it cannot happen: so use Integer.MAX_VALUE to break Math.max, and maintain ridiculous values. -When the case of i = 0, k = 0 is discussed, the above equation will calculate dp [i] [k] according to the actual situation -The init of this question is very important and tricky

Calculation order

-k people, need a for loop; -k people, starting from copy1 book, 2, 3, ... n-1, so i = [1, n], need a second for loop -On each i, the cutting method can be [0 ~ i], we have to calculate each worst time

Scrolling array

-[k] Only related to [k-1]

- Space: O(n)

Binary Search

-Based on: How much time does each person spend doing binary search: How long does it take each person to complete it in K people with the least amount of time? -The range of time variable is not index or page size. It is [minPage, pageSum] -Pay attention to 3 cases when validating: people are enough to use k> = 0, people are not enough so the ending is reduced to k <0, and there is a time (the time spent by each person) is less than the current page, return -1

- O(nLogM). n = pages.length; m = sum of pages.

---

8. Scramble String.java### Level: Hard Tags: [DP, Interval DP, String]

-Give two strings S, T. Check if they are scramble string. -scramble string definition: string can be split into binary tree form, that is, cut into substring; -After rotating a node that is not a leaf, a new substring is formed, which is the scramble of the original string.

Interval DP Interval

-Dimension reduction strike, split, dp by length. -dp [i] [j] [k]: array S starts at index i, T starts at index j, is a substring of length k, is it a scramble string

Break down

-After two halves of everything, look at two cases:, or not rotate the two halves. For these substrings, verify whether they are scrambled. -Two halves without rotation: S [part1] corresponds to T [part1] && S [part2] corresponds to T [part2]. -Rotate the two halves: S [part1] corresponds to T [part2] && S [part2] corresponds to T [part1].

Initialization

-When len == 1, it can't be rotated, that is, to see if the corresponding indexes of S and T are equal. -Initialization is very important. It 's amazing. This initaialization lays the foundation for DP, and the result is calculated with a mathematical expression. -input s1, s2 are hardly used in the main content of the entire problem, but only used in the initialization. -More details, see answer

---

9. Interleaving String.java### Level: Hard Tags: [DP, String]

Double-sequence DP, consider from the last point. At the end of the split problem, consider the association with s1, s2 subsequence.

Seeking existence, boolean

---

10. Edit Distance.java### Level: Hard Tags: [DP, Double Sequence DP, Sequence DP, String]

time: O (MN) Space: O(N)

Two strings, A must be B, you can insert / delete / replace to find the smallest change in operation count

Double Sequence

-Consider the end of two strings, index? S [i], t [j]: If you need to make these two characters the same, you might use the three operations given in the title: insert / delete / replace? -Calculate worst case first, 3 operation count + 1; then compare match case. -Note that when i or j is 0, the steps that become another number can only be fully changed. -First step, space time is O (MN), O (MN) -Rolling array optimization, space O (N)

Detail analysis

- insert: assume insert on s, #ofOperation = (s[0 ~ i] to t[0 ~ j-1]) + 1;
- delete: assume delete on t, #ofOperatoin = (s[0 ~ i - 1] to t[0 ~ j]) + 1;
- replace: replace both s and t, #ofOperatoin = (s[0 ~ i - 1] to t[0 ~ j - 1]) + 1;
- dp [i] [j]? represents the nature of two sequences:? s [0 ~ i]? The minimum operation count required to convert to s [0 ~ j]
- init: When i == 0, dp [0] [j] = j;? + j characters each time; Similarly, when j == 0, dp [i] [0] = i;
- And dp [i] [j] has two cases: s [i] == t [j] or s [i]! = T [j]

When to initialize

-This judgment depends on experience: if you know that initialization can be done together in a double for loop, then you can keep doing that -This belongs to what is needed, initialize what -When doing space optimization afterwards, you can easily do rolling array on 1st dimension

Search

-It can be done, but it is not recommended: this question needs to find min count, not search / find all solutions, so search will be more complicated, killing chickens.

---

11. Distinct Subsequences.java### Level: Hard Tags: [DP, String]

Double Sequence DP: 0. DP size (n + 1): find the result of the previous nth, then the dp array needs to open n + 1, because the end needs to return dp [n] [m]

1. Initialize dp [0] [j] dp [i] [0] in the for loop

2. Rolling array is optimized to O (N): If dp [i] [j] is in the for loop, it is a good replacement for curr / prev

---

12. Ones and Zeroes.java### Level: Hard Tags: [DP]

Still Double Sequence, but consider the third state: the amount of string array given. So opened a 3-dimensional array.

If you use a scrolling array to optimize space, you need to put the for loop that you want to scroll to the outermost, not the innermost. Of course, this third bit of definition is not a big problem in dp [] [] [].

Also, pay attention to calcualte zeros and ones outside, saving time and complexity.

---

13. Word Break II.java### Level: Hard Tags: [Backtracking, DFS, DP, Hash Table, Memoization]

Find all word break variations, given dictionary

利用 memoization: Map<prefix, List<suffix variations>>

DFS + Memoization

- Realize the input s expands into a tree of possible prefixes.
- We can do top->bottom(add candidate+backtracking) OR bottom->top(find list of candidates from subproblem, and cross-match)
- DFS on string: find a valid word, dfs on the suffix. [NO backtraking in the solution]
- DFS returns List: every for loop takes a prefix substring, and append with all suffix (result of dfs)
- IMPORANT: Memoization: Map<prefix, List<suffix variations>>, which reduces repeated calculation if the substring has been tried.
- Time O(n!). Worst case, permutation of unique letters: s= 'abcdef....', and dict=[a,b,c,d,e,f...]

Regular DPs

-Two DPs are used together to solve the problem of timeout: when a invalid case 'aaaaaaaaa' occurs, isValid [] stops dfs from occuring -1. Isword [i] [j], subString (i, j) exist in dict? -2. Use isWord to speed up isValid [i]: Can [i ~ end] find a reasonable solution from dict?
-View i from the end: Because we need to test isWord [i] [j], j> i, and we observe the interval [i, j];
-The part of j> i also needs to be considered, we also need to know isValid [0 ~ j + 1]. So isValid [x] is a DP indicating whether [x, end] is valid this time.
-i is from the end to 0, probably because it is considered that isWord [i] [j] is within [0 ~ n], so the numbers are reversed and the coordinates are easier to figure out.
-(Looking back at Word Break I, there is also a practice of coordinate inversion) -3. dfs uses isValid and isWord for ordinary DFS.

Timeout Note

-Regarding regular solution: Without memoization or dp, 'aaaaa .... aaa' will repeatedly calculate the same substring -Regarding double DP solution: Set.contains (...) is used in Word Break, i is 0 in isValid. However, contains () itself is O (n); intead, use an isWord [i] [j], judge whether i ~ j exists in dictionary based on O (1)

---

14. Minimum Window Substring.java### Level: Hard Tags: [Hash Table, String, Two Pointers]

Basic idea: use a char [] to store the frequency of the string. Then 2pointer, end go to the end, and continue to validate. If it meets the process as result candidate.

HashMap is a bit more complicated to write than char [], but more generic

---

15. Longest Substring with At Most K Distinct Characters.java### Level: Hard Tags: [Hash Table, Sliding Window, String]

Large cleaning O (nk)
Once map.size> k, erase the char at the beginning of the longest string (marked by pointer: start)
Once a char is to be cleared, the char between 1st and last appearance of this char must be cleaned from map

---

16. Find Minimum in Rotated Sorted Array II.java### Level: Hard Tags: [Array, Binary Search]

A topic that requires rigorous thinking. Because duplicates cause constant translation, the time complexity is ultimately O (n) So it is better to scan it directly and give the answer.

But still write a Binary Search, but the worst result is O (n)

---

17. Number of Islands II.java### Level: Hard Tags: [Union Find]

给一个island grid[][], and list of operations to fill a particualr (x,y) position.

count # of remaining island after each operation.

Union Find, model with int[]

-After converting the board into a 1D array, you can use union-find to determine it. -Use int [] father's unionFind, need to convert 2D position into 1D index. This is relatively clean -When judging, one step is taken in each of the four directions to determine whether it is the same Land. -Each time the operator walks, it will count ++. If it is found to be the same island, count-- -The count addition and subtraction are all placed in UnionFind's own function, which is convenient for tracking. Just give a few helper functions.

- Time: O (k * log (mn))

Union Find, model with Hashmap

-Union-find with HashMap.

Note:

- Proof of UnionFind log(n) time: https://en.wikipedia.org/wiki/Proof_of_O(log*n)_time_complexity_of_union%E2%80%93find

---

18. Word Search II.java### Level: Hard Tags: [Backtracking, DFS, Trie]

Give a string of words, and a 2D character matrix. Find all the words that can be formed. Condition: 2D matrix can only be positioned next to each other.

Trie, DFS

-Compared with the previous implementation, there are some places that can be optimized: -1. During Backtracking, you can mark on board [] [] instead of opening a visited [] [] -2. You don't need to implement all the equations of the trie, you don't need it: only insert is needed here. -Common trie questions will let you search for a word, but here is a board, see if each letter of the board can come out of a word. -That is: the search here is written by hand, not the traditional trie search () funcombination -3. When there is end in TrieNode, the string word is stored, which means the end. When the word = null is used up, the problem of repeated search is just truncated.

About Trie

- Build Trie with target words: insert, search, startWith. Sometimes, just: `buildTree(words)` and return root.
- Still do DFS on the board matrix.
- no for loop on words. Directly to board DFS:
- Each layer will have an up-to-this-point string. Check if it exists in the Trie. Use this to judge.
- If it does not exist, you do not need to continue DFS.
- Trie solution time complexity, much better:
- build Trie: n * wordMaxLength
- search: boardWidth * boardHeight * (4^wordMaxLength + wordMaxLength[Trie Search])

Regular DFS

- for loop on words: inside, do board DFS based on each word.
- Time cpmplexity: word[].length * boardWidth * boardHeight * (4^wordMaxLength)

Previous Notes

- Big improvement: use boolean visited on TrieNode!
- Don't use rst.contains (...), because this is O (n) and timeout in leetcode (lintcode can pass)!
- In the Trie search () method, mark any visits.

---

19. Word Squares.java### Level: Hard Tags: [Backtracking, Trie]

Can open Trie class, which uses TrieNode. Open Trie (words) can be directly initalized with for loop A TrieNode can have a List startWith: record all strings that can reach this point: a bit like a tree, ancestor-shaped storage.

God operates: According to the nature of square, if list of words is selected, set int prefixIndex = list.size (). Take all the words [prefixedIndex] in the list and add them together to make the prefix of the next word candidate.

Image a bit: list = ["ball", "area"]; prefixIndex = list.size(); ball[prefixIndex] = 'l'; area[prefixIndex] = 'e'; //then candidatePrefix = ball[prefixIndex] + area[prefixIndex] = "le"; Here you can use the findByPrefix function of Trie. At each point, all the dates that can be generated by this point are stored. At this point, try all candidate: dfs

I can think of this inverted structure to store prefix candidates in Trie, this idea is very worth thinking about.

---

20. Trapping Rain Water.java### Level: Hard Tags: [Array, Stack, Two Pointers]

There are many ways to solve this problem.

method 1

Array, maintaining a left-hand highest wall array, and a right-hand highest strength array. For each index, the maximum water column that can be stored vertically is determined by the left and right highest walls: min(leftHighestWall, rightHighestWall) - currHeight.

Method 2

The optimization above method 1, two pointers, still find the highest left and highest right. O (1) space. The idea used in method 3 is the same: the entire structure is divided by the highest bar in the middle: The left is calculated as maxLeft, and the right is calculated as maxRight.

Method 3

2 Pointers, double-sided pinch:

1. Find the index of the highest bar in the middle
2. Swipe to the center on both sides: add (topBarIndex-currIndex) * (elevation from previous index) each time. That is, add one horizontal bar at a time.
3. Every time you want to subtract the height of the block itself

Method 4

The main idea is the same as Method 3: On the basis of the downhill slope, the bottom has been stacked with stacks. Before finally encountering the ascent, at this time the bottom can be used to compare with all the downhill indexes accumulated before the stack, which is the water that is different from their height. The idea of using a stack to record downhill, and then dig to the end with a while loop is great.

---

21. Largest Rectangle in Histogram.java### Level: Hard Tags: [Array, Monotonous Stack, Stack]

Give n bars to form a histogram. Find the rectangle with the largest area that can be found in this row of histograms.

Thinking: Finding the area of a rectangle is nothing more than finding two indices, then the length of the bottom edge * height.

Monotonous Stack

-The main point is to maintain a monotonically increasing stack according to the nature of the rectangle in the Histogram. -When loop over indexes: -If the height is> = previous peek (), then for that peek, it means, go down, keep going higher, the previous peek can always continue to bottom -When can't I buy a bottom? When is there a downward trend? -At this time, not all previous peeks are calculated, but all previous peeks larger than the current height are considered. -Find all the rectangles from the peek to the current height: stack.pop () -In the process of stack.pop (), the current height is not counted, because it needs to be retained in the next round, and the current index is added to the stack. -Why use stack? Because you need to know the continuously increasing peek, stack.peek () O (1), easy to use In fact, instead of stack, you can record all heights in other ways, but it is inconvenient to find peek by O (n)

Knowledge

-Understand how monotonous stack is maintained -Maintaining a monotonous stack is required for the problem, not the nature of the stack itself.It is a clever use of stack.peek () O (1).

---

22. Find Peak Element II.java### Level: Hard Tags: [Binary Search, DFS, Divide and Conquer]

2Dmatrix, the value inside has some increasing and decreasing characteristics (the details are longer, see the original question). The goal is to find the peak element

peak: greater than the point value in the surrounding 4 directions

DFS

Fundamental

-We can't locate (x, y) accurately at one go, but we can find the peak of 1D array in another row / col. -Based on this point, move in the remaining two directions -1. In the middle line i = midX, find y where peak is. -2. In the middle column j = midY, find the x where the peak is. (It is possible to find the y before the strong override, that is, give up the peak of that line and find the peak on midY) -3. According to the 4 neighbor check (x, y) of (x, y) whether peak (x, y), if not, move one block like a higher position -4. According to the previously calculated midX, midY divide the board into 4 quadrants, and continue to find in each -This question LintCode did not do it, so the idea is correct, but the answer has not been verified again.

Pruning / splitting quadrant

-Just find a peak in row / col every time! -Finding this point is equivalent to cutting the board in half. -Then, compared with the remaining two adjacent positions, I know where the bigger one is, and where to find the peak, that is, I cut the second knife again. -When cutting the second knife, also move (x, y) to the quadrant to be taken. DFS -Cut according to mid row:

- http://www.jiuzhang.com/solution/find-peak-element-ii/#tag-highlight-lang-java
- http://courses.csail.mit.edu/6.006/spring11/lectures/lec02.pdf

time complexity

-Each level is halved

- T(n) = n + T(n/2) = n + n/2 + n/4 + ... + 1 = n(1 + 1/2 + ... + 1/n) = 2n = O(n)

Binary Search

- EVERYTHING
- O (nLogN)

---

23. Palindrome Pairs.java### Level: Hard Tags: [Hash Table, String, Trie]

Obvious's method is to try it all, and judge, it becomes O (n ^ 2) * O (m) = O (mn ^ 2). O (m): isPalindrome () time.

Of course not, then it depends on O (nlogN), or O (n)?

Method 1: Properties of Hash Table + Palindrome. Compound.

O (mn)

Ideas

-Each word can be split into front + mid + end. If this word + other words can form palindrome, that is to say -Cut off (mid + end), front.reverse () should be in words []. -Cut off (front + mid), end.reverse () should be stored in words []. -We use HashMap to store all <word, index>, and then reverse, just find a match.

Corner case

-If there is an empty string "", then it can be paired with any palendrome word, and it can be converted back and forth according to the position to make 2 distinct indexes. -This has the logic of `if (reverseEnd.equals (" "))` {...}. -Note: Although the two for loops that deal with beheading / chopping are repeating records based on the empty string, But because "" itself cannot be used as a starting point, overall will only be recorded once when paired with other palendrome.

Method 2: Trie

Still have to do that.

---

24. Maximal Rectangle.java### Level: Hard Tags: [Array, DP, Hash Table, Stack]

Method 1: monotonous stack

Decomposed, it is actually 'Largest Rectangle in Histogram', but here you have to build your own model heights. The rectangle in a 2D array is also finally made with height * width. The clever thing is, treat each line as the bottom edge, and calculate the height of the bottom edge to the top: -If there is a value

== 0 on the bottom edge, then it is counted as no height (the rectangle is used as the bottom edge, and the position of value == 0 is the sky tower, it cannot be used) -If value == 1 on the bottom edge, then add the above height to make a histogram

If you look at specific examples, some rows seem to be calculated in vain, but there is no way. This is a search process, and the optimal solution will eventually be compared.

Method 2: DP

Coordinate DP?

---

25. Longest Increasing Path in a Matrix.java### Level: Hard Tags: [Coordinate DP, DFS, DP, Memoization, Topological Sort]

mxn's matrix, find the longest increasing sequence length. Here the default continuous sequence.

-It is not possible to make a circle, so visit (x, y) cannot go. -Cannot walk in oblique direction, only walk up, down, left and right -Can't do according to coordinate DP, because the calculation order can go in 4 directions. -In the end, all nodes must be visited, so DFS search is more appropriate.

DFS, Memoization

- 简单版: longest path, only allow right/down direction:
- dp[x][y] = Math.max(dp[prevUpX][prevUpY], or dp[prevUpX][prevUpY] + 1) ; and compare the other direction as well
- This problem, just compare the direction from dfs result
- DFS has too many double calculations; memoization (dp [] [], visited [] []) eliminates double calculations
- initialize dp [x] [y] = 1, (x, y) counts itself as a cell in path
- dfs (matrix, x, y): check 4 neighbors (nx, ny) of (x, y) each time, if they are increasing to (x, y), then consider and compare:
- Maht.max(dp[x][y], dp[nx][ny] + 1); where dp[n][ny] = dfs(matrix, nx, ny)
- top level: O (mn), try to start from each (x, y)
- O(m * n * k), where k is the longest path

Topological sort

Not done yet

---

26. Coins in a Line III.java### Level: Hard Tags: [Array, DP, Game Theory, Interval DP, Memoization]

LeetCode: Predict the Winner

Still 2 people take n coins, and coins can have different values.

But this time the player can take from any side, but not restricted from one side. Will the first mover win?

Memoization + Search

-Like Coins in a Line II, MaxiMin's idea: Find the maximum of my disadvantages - dp [i] [j] represents the sum of values that players can take in the [i, j] interval -Similarly, sum [i] [j] represents the sum of values between [i] and [j] -Worst case for opponent, best case for first mover:

- dp[i][j] = sum[i][j] - Math.min(dp[i][j - 1], dp[i + 1][j]);
- You need to search here, draw a tree to see how it is segmented according to before and after fetching.

Game + Interval DP, Interval DP

-Because it looks at the interval [i, j], it can be thought of as the interval DP -This method needs a review, and it is related to the inference of mathematical expressions: S (x) =-S (y) + m. Refer to the following formula for derivation. -dp [i] [j] indicates that from index (i) to index (j), the maximum value that the first player can get is different from the opponent's number. That is S (x). -One of S (x) = dp [i] [j] = a [i]-dp [i + 1] [j] -There are two cases where m is at the beginning and m is at the end:

- dp[i][j] = max{a[i] - dp[i + 1][j], a[j] - dp[i][j - 1]}
- len = 1, integral is values [i]
- Finally judge dp [0] [n]> = 0, the difference between the maximum number sum is greater than 0, and you win.
- Time / space O (n ^ 2)

Formula Derivation

-S (x) = X-Y, find the difference between the largest number and sum, where X and Y are the total score of player X and the total score of player Y. -For player X: If the maximum value of S (x) is greater than 0, you win; if the maximum value is less than 0, you must lose. -Player Y: S (y) to indicate the difference between the largest number and sum for Y. S (y) = Y-X -According to S (x), if you take out a number m from the number and X, that is X = m + Xwithout (m)

- S(x) = m + Xwithout(m) - Y = m + (Xwithout(m) - Y).
- If we simply remove m from the global, then S (y '') = Y-Xwithout (m)
- Then calculate it: S (x) = m + (Xwithout (m)-Y) = m-(Y-Xwithout (m)) = m-S (y '')
- In this question, when we model X and Y, they are actually dp [i] [j], and the difference is first-hand / last-hand.
- Apply the formula, a certain S (x) = a [i]-dp [i + 1] [j], which is m = a [i], and S (y '') = dp [i + 1] [j]

Note

-If you consider calculating the maximum value between the first hand [i, j], then two arrays may be needed, and finally used to compare the score size of the first hand and the opponent => then more dimensions are needed. -The number difference we consider here just happens to make people not need to calculate the total score of the first mover, very clever. -Trick: Using the difference formula, the derivation is a bit difficult to think of.

Interval Dynamic Programming

-Find the properties within the [i, j] interval: dp [i] [j] The subscript indicates the interval range [i, j] -Sub-question: behead, tail, behead -loop should be based on the length of the interval -template: consider len = 1, len = 2; when setting i must be i <= n-len; when setting j, j = len + i-1;

---

27. Burst Balloons.java### Level: Hard Tags: [DP, Divide and Conquer, Interval DP, Memoization]

A volleyball, each ball has a value, each time you break one, you will score: left * middle * right value. Find, how to tie, the maximum?

TODO: Need more thoughts on why using dp[n + 2][n + 2] for memoization, but dp[n][n] for interval DP.

Interval DP

-Because the regularity of the array changes, it is difficult to find the 'first burst'. On the contrary, which one is the last burst? -The last burst becomes a wall: separate the two sides, consider separately, the principle of addition; finally add the middle.

- dp[i][j] represent max value on range [i, j)
- Need to calculate dp[i][j] incrementally, starting from range size == 3 ---> n
- Use k to divide the range [i, j) and conquer each side.

Interval DP Three Axes:

-Split in the middle -Cut off head or tail -Range as the basis of iteration

Print the calculation process

- use pi[i][j] and print recursively.
- Print k, using pi[i][j]: max value taken at k

Memoization

-In fact, it will be a DP that I think about afterwards -dp [i] [j] = max between balloons i ~ j. -Then which point to start the burst? Set to x. -For loop all points are taken as x, go to burst. -Each burst is cut into three parts: the left side can be recusive to find the maximum value of the remaining part on the left side + the middle 3 terms are multiplied + the right side is recursive to find the maximum value. -Note: This is Memoization, not pure DP -Because it is recursive, it is still a search, but memorize the requested value, saving processing

---

28. K Edit Distance.java### Level: Hard Tags: [DP, Double Sequence DP, Sequence DP, Trie]

Give a string of String, target string, int k. Find all the dates in the string array: change K times, can become target.

Trie

EVERYTHING

Double Sequence DP

- Edit Distance的follow up.
- In fact, it is to change the function of minEditDistance and bring K for comparison.
- The main logic is exactly the same as Edit Distance.
- But LintCode 86% test case is timeout.
- Time O (mnh), where h = words.length, if n ~ m, Time is almost O (n ^ 2), which is too slow.

---

29. Paint House II.java### Level: Hard Tags: [DP, Sequence DP, Status DP]

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

A row of n houses, each house can be painted in k colors, the price of each house is different, expressed by costs [] [].

costs [0] [1] means that the house with index 0 is painted and color 1.

Rule: two adjacent houses cannot be the same color

Seek: the least cost

DP

-It's almost the same as Paint House I, but with more paint colors: k colors. -First consider simply using dp [i] to represent the minimum cost of the first i houses -But what colors dp [i] and dp [i-1] index will affect each other, it is difficult to discuss, so add state: the sequence DP is added to the state to 2D. -Consider the last bit, and the previous i-1 is limited by the color of the i bit, so when considering min dp [i], there is another layer of iteration. -Do dp [i] [j]: # cost for the first i houses, so you must first pick the cost of the (i-1) house, then find out the cost of the (i-2) house -K colors => O (NK ^ 2) -If not optimized, it is almost the same code as Paint House I

- Time O (NK ^ 2), space (NK)
- Rolling array: reduce space to O(K)

Note

-The sequence type dp [i] represents the result of 'top i-1'. So dp is best set to int [n + 1] size. -However, the color is the state here, so it remains in j: [0 ~ k) -[[8]] This edge case. Can't run into for loop, so special handle.

Optimization Solution

- Team: O (NK)
- If it is known that two different minimum costs must be selected from the cost each time, then the minimum two are selected first, and there is no need to have a third for loop to find min
- Find each time in the series: min value except yourself, use min / min value
- Maintain 2 minimum values: minimum / secondary value.
- When calculating, if the index of the minimum value is not removed, the minimum value is given; if the index of the minimum value is removed, the next smallest value is given.
- Every loop: 1. calculate the two min vlaues for each i; 2. calcualte dp[i][j]
- How to think of optimization: Write the expression and see where you can optimize
- In addition, you can still roll array, reduce space complexity to O (K)

---

30. Best Time to Buy and Sell Stock III.java### Level: Hard Tags: [Array, DP, Sequence DP]

One more restriction than stock II: only 2 sell opportunities.

DP plus status

-Sell only 2 times, split the sale into 5 status modules. -In state index 0, 2, 4: No stock is held. 1. Always in this state, max profit is unchanged; 2. Just sold, dp [i] [previous state] + profit -At state index 1, 3: Holding the stock. 1. Always in this state, daily profit. 2. Just bought, state changed, but no profit yet: dp [i] [previous state]

Partial profit

-Adding daily partial profit (diff) together, the final overall profit is the same. The only thing that is better is that you don't need to record the time of the middle buy. -When will profit accumulate? -1. The original stock is held. If there is no action, the status will not change and the profit diff will be accumulated. -2. The stock was sold, the status changed, and profit diff accumulated. -Note: Only when the state index: 0, 2, 4, that is, the stock is sold, can you accumulate profit

Rolling Array

-[i] only deals with [i-1], reduce space

- O (1) space, O (n) time

Looking for the peak

-Find the peak; then look for another peak.-How about Optimize twice? Start looking for Max from both sides at the same time! (Awesome idea) -leftProfit is the maximum Profit at each i point from left to right.-rightProfit is the maximum profit at each point starting from point i to the end.-At point i, it is the leftProfit, and the rightProfit split point. At point i, leftProfit + rightProfit is added to find the maximum value.-Three O (n), or O (n)

---

31. Best Time to Buy and Sell Stock IV.java### Level: Hard Tags: [DP, Sequence DP]

There are int [] price of stock, up to k transactions. Seeking maximum profit.

DP

-According to StockIII, it is not difficult to find that StockIV is to divide the state into 2k + 1 shares. Then the same code, transplant.

Note 1:

-If k is large and k> n / 2, then there can be at most n / 2 transactions in an array of length n -Then the problem is simplified to stockII, giving n arrays, unlimited transactions.-Note that the number of status is 2k + 1

- Time O(NK), Space O(2k+1) to store the status

Note 2:

-The final status is 'no stock' should be considered, make a for loop to compare max. -Of course, it is also possible to make a profit variable and keep comparing.

Method 2

-(previous notes, thinking about the first method is enough) -Remember to understand: Why did you sell and buy on day i-1, and you can make a transaction with the sale on day i?
-Because the price of daily transactions is fixed. So if you sell and buy, you are not selling! That's why you can merge. Be sensitive to prices.

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

Local optimal solution vs. global optimal solution:

- local[i][j] = max(global[i − 1][j − 1] + diff, local[i − 1][j] + diff)
- global[i][j] = max(global[i − 1][j], local[i][j])
- local [i] [j]: On day i, profit of j-th transaction must be made on that day
- global [i] [j]: On day i, a total of j transactions have been profitable.

-The difference between local [i] [j] and global [i] [j] is that local [i] [j] means that there must be a transaction (sell) on day i.
-When the price on day i is higher than day i-1 (that is, diff> 0), then this transaction (buy on day i-1 and sell on day i) can be traded with day i-1 (Sell) merge into one transaction, that is local [i] [j] = local [i-1] [j] + diff;
-When the price on day i is not higher than day i-1 (that is, diff <= 0), then local [i] [j] = global [i-1] [j-1] + diff, and because diff <= 0, so it can be written as local [i] [j] = global [i-1] [j-1].
-(Note: + diff is not omitted in this solution below)

-global [i] [j] is the maximum profit we can make for a maximum of k transactions in the first i days, which can be divided into two cases:
-If there is no transaction (sell) on day i, then global [i] [j] = global [i-1] [j];
-If there is a transaction (sell) on day i, then global [i] [j] = local [i] [j].

---

32. Russian Doll Envelopes.java### Level: Hard Tags: [Binary Search, Coordinate DP, DP]

Matryoshka, here is represented by envelope. For a string, each [x, y] is the length and width of envelope. [[5,4], [6,4], [6,7], [2,3]].

Look at these sets of dolls, you can set a few at most.

DP: 1D Coordinate

-Envelopes have no order, they are sorted first (mainly according to the first index) -Then observe: After sorting, it becomes 1D coordinate dynamic programming. -max number depends on the max value of the previous successful Russian doll + 1 -The previous index is unknown, so iterate to find the previous index. -The current state of index i depends on the state of previous index j, so iterate through two indexes.

- O(n^2)的DP, n = envelopes.length;

DP: 2D Coordinate

-This method came up by myself, but the time complexity is too large, timeout -Mark the envelop on the 2D grid, and then like a robot, find the maximum count max in the bottom right corner. -count how many Russian dolls can be present -Two cases: current coordinate has no target, current coordinate has target -The current coordinate has no target: like robot moves, Math.max (dp [i-1] [j], dp [i] [j-1])

- 当下coordinate 有target: dp[i - 1][j - 1] + dp[i][j]
- timeout: O(n^2), n = largest coordinate.

---

33. Expression Tree Build.java### Level: Hard Tags: [Binary Tree, Expression Tree, Minimum Binary Tree, Stack]

Give a string of characters, which is the formula expression. Turn the formula into an expression tree

Monotonous Stack

-Like Max-tree, https://leetcode.com/problems/maximum-binary-tree -A bottom-> top incremental stack is used: the lowest root is maintained as the smallest element. -This topic is Min-tree, the smallest on the head, Logic is the same as max-tree

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

Features

-TreeNode: Use a TreeNode that is not the final result, store the weight, and use it for sorting -Weigh the symbols on the same level with the concept of base weight, numerical order -Each character is a node and has its own weight. Use a TreeNode to store the weight value, and use weight to determine: -1. (while loop) If node.val <= stack.peek (). NodeValue, change the current stack.peek () to left child. -2. (if condition) If the stack has residue, change the current node to stack.peek (). RightChild

---

34. Expression Evaluation.java### Level: Hard Tags: [Binary Tree, DFS, Expression Tree, Minimum Binary Tree, Stack]

Give a formula expression, array of strings, and evaluate the result.

DFS on Expression Tree

-Calculate the value of expression: 1. Construct expression tree. 2. DFS calculation result

- Expression Tree: Minimum Binary Tree (https://lintcode.com/en/problem/expression-tree-build/)
- After building Min Tree, do PostTraversal.
- Divde and Conquer: first recursively find the size of left and right, then evaluate the symbol in the middle
- Time, Space O(n), n = # expression nodes

Note

-1. When Handle number, if left && right Child is all Null, it must be our largest number node.
-2. If one child is null, then return another node.
-3. prevent Integer overflow during operation: Use a Long during the process, and the final result is cast back to int.

---

35. Convert Expression to Polish Notation.java### Level: Hard Tags: [Binary Tree, DFS, Expression Tree, Stack]

Give a string of characters to represent the formula expression. Convert this expression to Polish Notation (PN).

Expression Tree

- Expression Tree: Minimum Binary Tree (https://lintcode.com/en/problem/expression-tree-build/)
- After the Expression Tree is made according to the intent: After a Pre-order-traversal, you can record the Polish Notation
- This question is not given to 'ExpressionTreeNode', so TreeNode is regarded as the node we need, and it can be expanded to have left / right child.
- Note: label needs to be String. Although Operator is a char with length 1, the number can be multiple digits

---

36. Convert Expression to Reverse Polish Notation.java### Level: Hard Tags: [Binary Tree, DFS, Expression Tree, Stack]

Give a string of characters to represent the formula expression. Convert this expression to Reverse Polish Notation (RPN).

Expression Tree

- Expression Tree: Minimum Binary Tree (https://lintcode.com/en/problem/expression-tree-build/)
- After the Expression Tree is made according to the intent: Post-order-traversal can record Reverse Polish Notation
- This question is not given to 'ExpressionTreeNode', so TreeNode is regarded as the node we need, and it can be expanded to have left / right child.

---

37. Decode Ways II.java### Level: Hard Tags: [DP, Enumeration, Partition DP]

Given a string of numbers, you need to decode them into English letters. [1 ~ 26] Corresponding to the corresponding English letters. Find out how many ways you can decode.

The characters may be "*", which can represent [1-9]

DP

-Multiplication principle -Same as decode way I, the principle of addition, and the cutting point: whether 1 digit or 2 digits is used to decode -Define dp [i] = how many ways to decode the first i digits. New dp [n + 1]. -Different situations: In each partition, if there is "*", it will extend many different possibilities in itself.

- 那么: dp[i] = dp[i - 1] * (#variations of ss[i]) + dp[i - 2] * (#variations of ss[i,i+1])

Features

-Ability to enumerate: specifically analyze where the '*' appears, enumerate numbers, basic skills. -Note !! The title says * in [1, 9]. (It will be harder if 0 ~ 9) -Understand the reason for taking MOD: the number is too large, take the mod to give the final result: In fact, with a mod as large as 10 ^ 9 + 7, most examples can pass. -After enumerating, the writing and thinking process of this topic is not difficult.

---

38. Palindrome Partitioning II.java### Level: Hard Tags: [DP, Partition DP]

Give a String s, find out how many cuts to use, so that each substring that is cut out is palindrome

Partition DP

-Find minimum cut: Divide DP -dp [i]: How many knives to cut at least, so that the string of length i before the cut is all palindrome -You end up with dp [n], so int [n + 1] -Move the cutter, see where to cut, index j in [0 ~ i] -Consider whether [j, i-1] is a palindrome string, and if so, then: dp [i] = min (dp [i], d [j] + 1). -note: It is estimated that when traversing j, the reverse traversal is also possible.

Computing Palindrome Optimization

-Using the properties of palindrome, the situation of boolean palindrome [i, j] can be calculated. -Find an arbitrary mid point: -1. Assuming that the palindrome is an odd length, then mid is a separate character, and the characters [mid-1], [mid + 1] on both sides should be exactly equal. -2. Assuming that the palindrome is of even length, the characters at [mid] and [mid + 1] should be equal. -Do this palindrome [i, j]: whether the substring from character i to character j is palindrome -This will reasonably reduce the dimensionality of our problem, currently it is time: O (n ^ 2). -Otherwise, if we ask for palindrome once, that is n, it will become O (n ^ 3)

Previous Notes

-Double for loop checks each substring string (i ~ j). If i, j are adjacent or the same point, then isPal; otherwise, (i + 1, j-1) between i and j must be isPal. -It seems that when checking i, j, how can I know (i + 1, j-1) pressed in the middle first? Actually not .. When j grows up slowly, all 0 ~ j substrings are checked. So isPal [i + 1] [j-1] must already know the result. -okay. Then if any of the above is true, that is to say isPal [i] [j] == true. Then we have to judge how many ways to cut to the end point of the loop parameter j in the first layer? -The idea is smooth: we naturally think that it would be better to add the cut before i plus what happened between i ~ j. -Anyway, j is not changed now, let 's see where i is and whether cut [i-1] is smaller / minimum; then +1 on cut [i-1] is over. -Of course, if i == 0, and i ~ j is isPal, then there is nothing to talk about, don't cut, 0 knife. -In the end, brush to cut [s.length ()-1], which is the last point. return is right.

39. Backpack III.java### Level: Hard Tags: [Backpack DP, DP]

For n different items, int [] A weight, int [] V value, each item can be used unlimited times

Ask the maximum value can be packed into a package of size m?

DP

-Items can be used indefinitely, losing the meaning of last i, last unique item: because it can be reused. -So you can convert an angle: -1. Use i kinds of items, spell w, and satisfy the max value. Here, item i can be used unlimited times, so consider how many times K is used. -2. Although K can be infinite, it is also limited by k * A [i]: the maximum cannot exceed the size of the backpack. -dp [i] [w]: For the first i items, fill the w backpack, what is the maximum value?

- dp[i][w] = max {dp[i - 1][w - k*A[i-1]] + kV[i-1]}, k >= 0
- Time O (nmk)
- If k = 0 or 1, it is actually Backpack II: with or without

Optimization

-Optimize time complexity, draw and find: -Calculated (dp [i-1] [j-k * A [i-1]] + k * V [i-1]) -In fact, the grid of dp [i] [jA [i-1]] on the same line has V [i-1] -So there is no need to loop over k times every time -Simplify: dp [i] [j] One of the possibilities is: dp [i] [j-A [i-1]] + V [i-1]

- Time O (mn)

Space optimized to 1-dimensional array

-Draw a 2 rows grid according to the previous optimization -Found that dp [i] [j] depends on: 1. dp [i-1] [j], 2. dp [i] [j-A [i-1]] -Among them: dp [i-1] [j] is the settlement result of the previous round (i-1), it must be already calculated, ready to be used -However, when we have i ++, j ++, and before row = i-1, col <j, all of them are not needed. -Dimension reduction and simplification: We only need to keep the dimension of weigth, and the dimension of i can be omitted: -(i-1) row is just to use the old value calculated before: each round, j = [0 ~ m], then dp [j] itself has the function of recording the old value. -Becomes 1 bit array -Focus of dimensionality reduction optimization: look at the left and right calculation direction of the two lines

- Time(mn). Space(m)

---

40. First Missing Positive.java### Level: Hard Tags: [Array]

Give a string of unordered numbers with negative numbers: find the first missing positive integer in this array

The missing positive integer is actually compared with [1, n].

Array analysis, index tricks

-With a while loop, keep trying to get the number to the place -If index = nums [i] exceeds nums.length, of course, it will not move -Note: check val! = Nums [val], avoid infinitely loop -Test: Is nums [i] equal to i, if not, then find the result

Edge Case

-If nums == null, in fact missing positive integer is naturally 1 -During validation, there may be no disconnected integers in the string, but the largest integer is in the first place (because the index exceeds the standard, it cannot be placed in the correct place) -At this time, n is placed at index 0. In fact, the next integer should be n + 1 -In the end, if the array is completely sorted, it is not lacking, and it also meets the conditions of the superscript, then the only next one is the first positive number outside the range of the array: n

---

41. N-Queens.java### Level: Hard Tags: [Backtracking]

N-Queen question, give numbers n, and nxn board, find all answers for N-queens.

Backtracking

-Use dfs to find all situations, each iteration, pick the right point from the line, dfs -The selected points are added to the candidate list, remember to backtracking. -Each candidate needs validation, check if row, col, 2 diagnal is queen

validate n queue at certain (x, y)

-1. array cannot have target row # -2. diagnal. Remember the formula:

- row1 - row2 == col1 - col2. Diagnal elelment.fail
- row1 - row2 == - (col1 - col2). Diagnal element. fail
- Draw a 3x3 board to test the 2 scanarios:
- (0,0) and (3,3) are diagnal
- (0,2) and (2,0) are diagnal

---

42. N-Queens II.java### Level: Hard Tags: [Backtracking]

Like N-Queens, not all results, but how many results are counted.

Backtracking

-When list.size () == n, it means that a solution was found.

- 1. dfs function (List, n)
- 2. validate function

---

43. LRU Cache.java### Level: Hard Tags: [Design, Hash Table, Linked List]

Double Linked List

-Using a special bidirectional ListNode, with head and tail, this greatly speeds up the process.
-The main thing that speeds up is the process of updating the ranking. Finding the item hashmap O (1), and doing subtraction and transposition are O (1)

- Overall O(1)

Be smart

-1. Head and tail are particularly clever: removing heads and tails, and adding heads and tails, are particularly fast.
-2. Use two-way pointers: pre and next. When you need to remove any node, just know which one to remove.
-Just patiently connect node.pre and node.next, and the node will be disconnected naturally.
-Once you know how to solve it, it is not very special, and it is not difficult to write the algorithm:

- moveToHead()
- insertHead()
- remove()

O (n) check for duplicates

-timeout method, naive came an O (n) solution, and it turned out timeout.
-A map <key, value> stores the value. A queue to hold the rank.
-Every time there is an update, put the latest one at the end; every time you exceed the capaticity, kill the big head. Very simple, but it took too long to run and failed.

---

44. Binary Tree Maximum Path Sum.java### Level: Hard Tags: [DFS, DP, Tree, Tree DP]

Find max path sum, from any treeNode to any treeNode.

Kinda, Tree DP

-Two cases: 1. combo sum: left + right + root; 2. single path sum

- Note1: the path needs to be continuous, curr node cannot be skipped
- Note2: what about I want to skip curr node: handled by lower level of dfs(), where child branch max was compared.
- Note3: skip left / right child branch sum, by comparing with 0. Less than 0, no need to record

DP Thoughts

-tree gives us 2 branches, each branch is similar to dp [i-1], here is similar to dp [left], dp [right] -After finding dp [left], dp [right], combine with curr node. -Because it is looking for max sum, and can skip nodes, the global variable max is required -Each time dfs () returns must be a path that can continue continuously link ', so return one single path sum + curr value`.

DFS, PathSum object

- that just solves everything

---

45. Basic Calculator.java### Level: Hard Tags: [Binary Tree, Expression Tree, Math, Minimum Binary Tree, Stack]

Give an expression String to evaluate the value of the expression.

Expression strings include +,-, integers, opening and closing parentheses, and space.

Expression Tree

-Expression Tree is a weight-based min-tree -Tree based on operation symbol + number: number is always in leaf, then symbol is tree node, brackets do not appear in tree -Use monotonuous stack to build this tree

Thinking points

- Understand Expression Tree
- Use stack to build the expression tree + understand the weight system
- Use post-order traversal to evaluate the tree
- Note that the number in input will not be a single digit, so a buffer is needed to store the number string
- For the practice of the entire topic, please refer to Expression Evaluation

---

46. Longest Consecutive Sequence.java### Level: Hard Tags: [Array, Hash Table, Union Find]

Give a string of numbers, unsorted, find the length of the sequence of consecutive elements in the string of numbers

HashSet

-To see continuous elements, you must search for num ++, num-- -1. Need O (1) to find the element -2. Need to find num-1, num + 1.-If you open the array with min, max, it consumes space -Save with HashSet, use set.contains () to find num-1, num + 1 exists

- for loop. O(n)
- The while loop inside will generally not have O (n); once O (n), it also means that the set is cleared, and there will be no more inner while derivatives for the for loop.
- overall O (n) time complexity

Union Find

-In the end, we need to calculate the total length of the connected elements. In fact, the elements are grouped together, and the connected groups are together, so I think of UnionFind -An int [] size is used here to help deal with the question of parent when merging: always go to a union with a large group -In the main function, there is a map to track, and each element is processed only once.

- union的内容: current number - 1, current number + 1
- https://www.jianshu.com/p/e6b955ca208f

Features

-Union Find seems easier to do on index

- 其他union find function: boolean connected(a,b){return find(a) == find(b)}

---

47. Serialize and Deserialize Binary Tree.java### Level: Hard Tags: [BFS, DFS, Deque, Design, Divide and Conquer, Tree]

Serialize and Deserialize Binary Tree

DFS, Divide and Conquer

Serilize

- Divide and conquer: Pre-order traversal to link all nodes together
- build the string data: use '#' to represent null child.
- the preorder string, can be parsed apart by `split(',')`

Deserialize

- Use a list (here we use `Deque` for the ease of get/remove in 1 function: remove())
- to take all parts of the parsed sring data: dfs on the Deque
- first node from the list is always the head
- '#' will be a null child: this should break dfs
- Deque is a global variable, so dfs(right child) will happen after dfs(left child) completes

DFS, Recursive [previous note]

- serilize: divide and conquer, pre-order traversal
- deserialize: slightly more complicated, use dfs. truncate input string each time:
- Keep dfs looking for left child, then right child until leaf is found.
- Use a StringBuffer to hold the string, because string is primitive, we need a pass reference here

BFS, Non-recursive

-using queue. The idea is intuitive. level-order traversal. Save to a string. -When encountering a null child, instead of ignoring it directly, you assign an Integer.MIN_VALUE, and then mark as '#' -BFS requires track queue size, each time only a specific number of nodes

---

48. Count of Smaller Numbers After Self.java### Level: Hard Tags: [BST, Binary Indexed Tree, Binary Search, Divide and Conquer, Segment Tree]

Give a string of numbers nums [], find a new array result, where result [i] = # of smaller items on right of nums [i]

Binary Search

-sort and insert into a new list, the new list is sorted -Traverse nums [] from the end i = n-1 -Each time insert nums [i] enters the list, it is # of smaller items on right side of nums [i] -Record result [i] every time -### Question : The binary search here is done with `end = list.size (); while (start <end) {...}`, can it be replaced with `end = list.size ()- 1`?

Segment Tree based on actual value

- Build segment tree based on min/max values of array: set each possible value into leaf
- query(min, target - 1): return count # of smaller items within range [min, target - 1]
- Very similar to `Count of Smaller Number`, where segment tree is built on actual value!!
- IMPORTANT: goal is to find elements on right -> elements processed from left-hand-side can be removed from segment tree
- Use `modify(root, target, -1)` to remove element count from segment tree. Reuse function
- time: `n * log(m)`, where m = Math.abs(max-min). log(m) is used to modify() the leaf element

Segment Tree solution - tricky part:

- negative nubmer works oddly with mid and generates endless loop in build(): `[-2, -1]` use case
- build entire segment tree based on [min, max], where min must be >= 0.
- we can do this by adding Math.abs(min) onto both min/max, as well as +diff during accessing nums[i]

Binary Indexed Tree

- TODO, have code

---

49. Remove Duplicate Letters.java### Level: Hard Tags: [Greedy, Hash Table, Stack]

Hash Table, Greedy

-count [] = int [256], no `c-'a'` required -boolean visited []: Once a letter has fixed its position, when it encounters it again, it skips the used character directly - If the tail letter can be made smaller, delete the tail and reconnect with the new letter (prerequisite: the removed letter will appear again after the set letter, set visited [tail] = false)

- Space: O(1) count[], visited[].
- Time: Go through all letters O(n)

Stack

- Use stack instead of stringBuffer: keep append/remove last added item
- However, stringBuffer appears to be faster than stack.

---

50. Expression Add Operators.java### Level: Hard Tags: [Backtracking, DFS, Divide and Conquer, String]

Give a number String, the numbers come from `0-9`, give 3 operators`+`,`-`, `*`, see how to piece together, you can make the result target.

output 所有 expression

string dfs, use list to track steps (backtracking)

-Related to string, it may be a little tedious to write -Numbers have dfs ([1,2,3 ...]) combination method -operator has [`+`, `-`,`*`] 3 combinations -Note 1: The multiplication sign must be treated specially, the numbers along the multiplication are passed, and when calculating the next product, sum-preProduct + product -Note 2: '01' is a skip number -Note 3: The first selected number does not need to be added, it is added directly

- Time: O(4^n), Space: O(4^n)
- T(n) = 3 * T(n-1) + 3 * T(n-2) + 3 * T(n-3) + ... + 3 *T(1);
- T(n-1) = 3 * T(n-2) + 3 * T(n-3) + ... 3 * T(1);
- Thus T(n) = 4T(n-1) = 4^2 * T(n - 1) = ... O(4^n)

String dfs, use string as buffer

-The logic is the same, the code is shorter, but instead of doing a list, pass `buffer +" + "+ curr` directly -It is slightly slower because new strings are created each time. Same as Time complexity

---

51. Insert Interval.java### Level: Hard Tags: [Array, PriorityQueue, Sort]

Sweep Line

-Interval teardown point, PriorityQueue rank point -Merge with count == 0 as the judgment point -Note, be sure to compare curr `px == queue.peek (). X` to ensure that all coincident points are processed by process:`count + = px` -PriorityQueue: O (logN). Scan n points, total: O (nLogn)

Basic Implementation

-### sorted intervals by start point have been given here. -Directly find the seat where insert newInterval can be inserted. Insert

- 然后loop to merge entire interval array
- Because it is a list, it is convenient for `intervals.remove (i)`
- pre.end will be reasserted before remove to ensure that the removed node.end is captured
- O (n)

Also

-Because interval has been sorted, I wanted to use Binary Search O (logn). -But to find the interval insert position, the final merge still uses O (n), so it is not necessary to use binary search

---

52. Shortest Palindrome.java### Level: Hard Tags: [KMP, String]

Divide by mid point, Brutle

- check (mid, mid+1), or (mid-1, mid+1).
- If the two position matches, that is a palindrome candidate
- Compare whether front string is a substring of end string

- O (n ^ 2)
- timeout on last case: ["aaaaaa....aaaacdaaa...aaaaaa"]

KMP

- EVERYTHING

---

53. K Empty Slots.java### Level: Hard Tags: [Array, BST, TreeSet]

题目解析后: find 2 number, that: 1. k slots between the 2 number, 2. no slots taken between the two number.

BST

- BST structure not given, use TreeSet to build BST with each node
- Every time find last/next inorder element
- `treeSet.lower(x)`, `treeSet.higher(x)`
- Once the positions are separated (k + 1), the subject conditions are met
- O(nlogn), good enough

Track slots of days

- Reverse the array, save days index into days[], where the new index is slot.
- days[i]: at slot i, which day a flower will be planted
- O (n)
- Needs to understand: http://www.cnblogs.com/grandyang/p/8415880.html

---

54. Count of Range Sum.java### Level: Hard Tags: [BST, Divide and Conquer, Merge Sort, PreSum]

TODO: Write the code + merge function

Divide and Conquer + PreSum + MergeSort

-The algorithm is very powerful: presum [], then sum range [i, j] is equal to preSum [j + 1]-preSum [i] -Divide and conquer: Consider the results in [start, mid] range, and then the results in [mid, end] range. (MergeSort separately) -Finally consider [low, high] overall results -Tip: PreSum is made (n + 1) length, then the range sum [i, j] can be reduced to preSum [j]-preSum [i]

- NOTE: should write merge() function, but that is minor, just use `Arrays.sort(nums, start, end)`, OJ passed
- Every mergeSort() has a for loop => O(n log n)

How to count range?

-A special method here: find a preSum after i and mid in [low, mid] for comparison (explained from: https://blog.csdn.net/qq508618087/article/details/51435944) -Find two boundaries in the right array, set to `m, n`, -Where m is the first in the array on the right such that `sum [m]-sum [i]> = lower`, -n is the first position that makes `sum [n]-sum [i]> upper`, -In this way, `nm` is the number of intervals in the `[lower, upper]` range formed by the element i on the left.

The magic point: why merge and sort

-The borders [lower, higher] are compared in the sorted array. Once the national borders, the calculation can be stopped and unnecessary calculations can be reduced. -The premise of n, m above is feasible: preSum [] has two ranges [low, mid], [mid, high] before and after sorted -In other words, when recursively mergeSort (), you really need to merge sorted 2 partitions -You may ask: Can you sort? Sort will disrupt the order soon? Yes, the order of preSum [] is disrupted. -But it doesn't matter: very clever, when dividing and conquering, the first half / the second half are separated and the original process is separated, and the merge is finally completed. -When doing the range of m, n, the principle is as follows, for example, preSum is divided into two sections: `[A, B, C]`, `[D, E, F]` -When comparing each preSum value `A` with preSum [i], `A-preSum <lower`, it is a single comparison, not involving B, C -Therefore, it does not matter whether `[A, B, C]` retains the order of the initial preSum -The most important thing at this time is that `[A, B, C]` and sorting are good, then when the `lower` boundary is compared, once the boundary is crossed, the calculation can be stopped (reduce unnecessary calculations)

BST

- EVERYTHING?

---

55. Max Sum of Rectangle No Larger Than K.java### Level: Hard Tags: [Array, BST, Binary Search, DP, Queue, TreeSet]

Given a non-empty two-dimensional matrix matrix and an integer k, find the largest rectangular sum of sums not greater than k within the matrix.

BST, Array, preSum

-Reduce the problem to: row of values, find 1st value> = target.

-
    1. loop over startingRow; 2. loop over [startingRow, m - 1]; 3. Use TreeSet to track areas and find boundary defined by k.
- When building more rows/cols the rectangle, total sum could be over k:
- when it happens, just need to find a new starting row or col,
- where the rectangle area can reduce/remain <= k
- Find the starting point of the extra area: extraArea = treeSet.ceiling (totalSum-k). That is, find the starting / left area after subtracting k.
- Remove these left starting areas, the rest is <= k. (Num-extraArea)
- Why use TreeSet: the size of the area is irregular, and find the first value of> = any value. Give a string of non-sorted numbers, find the number of> = target, if you do not write binary search, then BST is the most suitable
- O (m ^ 2 * nlogn)

Thought

-Considering the most basic O (m ^ 2 * n ^ 2): traversing startingRow / startingCol -rectangle? layer by layer? You can think of the idea of Presum. When it is larger than the required sum, subtract the extra part.-How to find extra areas? Then search: Save the content you need to search, you can think of using BST (TreeSet), or write Binary Search yourself.

---

56. Perfect Rectangle.java### Level: Hard Tags: [Design, Geometry, Hash Table]

See if the list of coordinates can form a perfect rectangle and does not allow overlap area.

Drawing features

-Feature 1: All the given points (find out the diagonal points without specifying), if the perfect rectangle is formed at the end, they should be eliminated from each other, and finally there are 4 corners left -Feature 2: Find the min / max (x, y) in all points, and the final maxArea should be equal to the area accumulate in the process -Feature 1 Make sure there is no hollow part in the middle, ensure that all coincident points will be eliminated from each other, and finally there are 4 vertices left -Feature 2 ensures no coincidence: coincident areas will eventually exceed maxArea

---

57. Max Points on a Line.java### Level: Hard Tags: [Array, Geometry, Hash Table, Math]

给list of (x,y) coordinates. Determine # of points on the same line

Observation

- If given n points, we can calculate all possible slopes. O(n^2) times
- For the two dots that generates the same slope, these dots could be on parallel### slopes
- figure out how to prune the parallel dots

Trick: prune parallel dots using greatest common divider

- GCD: greatest common divider
- Devide the x and y by their greatest common divider, such that x and y can be reduced to minimum value
- All other x and y can be reduced to such condition as well
- track the final reduced (x,y) in a map: they are the key to the count
- No need to use Map<Integer, Map<Integer, Integer>> to perform 2 level mapping; just `map<String, Integer>`, where the key is "x@y"

---

58. Number of Digit One.java### Level: Hard Tags: [Math]

Pure math problem, not quite representative

Explanation https://leetcode.com/problems/number-of-digit-one/discuss/64381/4+-lines-O(log-n)-C++JavaPython

---

59. Binary Representation.java### Level: Hard Tags: [Bit Manipulation, String]

String

-First we need to solve in two halves, the breakpoint is '.': Str.split ("\\."); -The half of Integer is easy to handle, in the while loop: num% 2, num / 2. Make a parseInteger () function -Decimal is more complicated. Make a parseDecimal () function: -bit == 1 Mathematical condition: Now num * 2> = 1. Update: num = num * 2-1; -Math condition for bit == 0: num * 2 <1. Update: num = num * 2

Note

-num is double, decimals may loop infinitely under the formula num = num * 2-1 -Therefore check: num repeatability, and binary code <32 bit. -So the title also has a 32BIT requirement!

---

60. Recover Binary Search Tree.java### Level: Hard Tags: [BST, DFS, Tree]

There are 2 node misplaces in BST, which are classified as: Requirement: O (1) extra space

Observation

- BST inorder traversal should give small -> large sequence
- misplaced means: a large->small item would occur, and later a large>small### would occur.
- The first large && second small item are the 2 candidates. Example
- [1, 5, 7, 10, 12, 15, 18]
- [1, 5, 15, 10, 12, 7, 18]

dfs, O(1) extra space

- traverse, and take note of the candidate
- at the end, swap value of the 2 candidates

O(n) space

- inorder traversal the nodes and save in array, find the 2 items misplanced and swap them
- But O(n) space should not be allowed

---

61. Jump Game II.java### Level: Hard Tags: [Array, Coordinate DP, DP, Greedy]

Giving a string of numbers is the distance that can be jumped. Goal: the minimum number of jumps to the last index possible.

Greedy

- always aiming for the farest can go
- if the farest can go breaches the end, return steps
- otherwise, send start=end+1, end=farest and keep stepping from here
- though trying with 2 loops, worst case [1,1,1,...1,1] could have O(n^2)
- But on average should be jumping through the array without looking back
- time: average O(n)

Previous Notes, Greedy

-Maintain a range, the farthest we can go. -index / i is step by step, each time when i <= range, make a while loop, find the farthest reach in it maxRange -Then update range = maxRange -Where step is the same as index, step by step. -The final check condition is that the range you can walk the farthest is> = nums.length-1, indicating that the focus has been reached with the least steps. Good.

Even simpler Greedy

-Graphic http://www.cnblogs.com/lichen782/p/leetcode_Jump_Game_II.html

- track the farest point
- whenver curr index reachest the farest point, that means we are making a nother move, so count++
- there seems to have one assumption: must have a solution. Otherwise, count will be wrong number.
- In fact, it is exactly the same thinking pattern as the first greedy.

DP

-DP [i]: record at point i, the minimum number of jumps to point i

- dp[i] = Math.min(dp[i], dp[j] + 1);
- condition (j + nums[j] >= i)
- Note the use of dp [i] = Integer.MAX_VALUE as the starting value to find min
- time: O(n^2), slow, and timesout

---

62. Longest Valid Parentheses.java### Level: Hard Tags: [Coordinate DP, Stack, String]

Give a string with only `(`, `)` in it. Find the length of the longest valid parentheses.

1D Coordinate DP

- use dp[i] track local max, maintain global max
- int[] dp. dp[i]: longest valid string that ends on i.
- The ending is ')', 2 cases: 1. s [i-1] is just '('; 2. s [i] 's' is the beginning of a '('
- Note, if the ending is '(' is unreasonable, ignore it.
- init: dp [0] = 0, single char cannot be formed.
- Calculation order: from left to right, find local max, maintain global max
- O(n) space, O(n) runtime

Stack

-Store all open / close parentheses in the Stack. -If stack.top () happens to open and close, just stack.pop (). -The rest are unreasonable elements. -A bit like negatively looking for solution: `endIndex-The last failedIndex (stack.pop ())-1`, it should be the length of the last succeded string -Update endIndex to stack.top () each time, and then continue to find the next failedIndex from the stack -Compare all lengths to find the longest length -O (n) stack space, O (n) runtime. It should be slower than dp, because O (n) is done twice

---

63. Rearrange String k Distance Apart.java### Level: Hard Tags: [Greedy, Hash Table, Heap]

Give a string, all lowercase letters, and ask for rearrangement: then each unique character must have k distance apart.

It's similar to Task Scheduler, except that you can use other methods to find the count in Task. This problem requires the ranking results.

PriorityQueue + HashTable

-A classic usage of PriorityQueue sorting + distribution sorting.

- Count frequency and store in pq.
- Consume element of pq for k rounds, each time pick one element from queue.
- Exception: if k still has content but queue is consumed: cannot complete valid string, return "";
- space, O(n) extra space in sb, O(26) constant space with pq.
- time: O(n) to add all items

---

64. Valid Number.java### Level: Hard Tags: [Enumeration, Math, String]

team: O (n)

Analyze edge cases, and various situations, and then determine whether it is a valid number

Situation summary

-Several different cases of `.`,`e`, `+/-`, `int` -The results are different when the order of encounter is different. -This question is more about analyzing the situation, and then enumerating the edge case, the algorithm has less significance.

---

65. Bricks Falling When Hit.java### Level: Hard Tags: [Union Find]

Give a matrix of 1 and 0, `1` stands for brick. The brick connected to ceiling will not drop. Give a string of coordinate hits [] [], record how many drops each time you take down 1 brick.

UnionFind

-1. We know that most bricks may be connected to ceiling, so every forward check is traverse all and timeout -2. Can I use union to install the connectors together, and then drop the connected ones when I drop the brick? Difficult: Because I still have to check all the current status of the brick. -Inspired by other people's answers, since it is counting counts, we can `think backwards`: -Mark all hit-bricks = 2 (just ignore them), observe the last step of the whole situation, first calculate the total number of all bricks still connected to ceiling, and count all the counts in unionFind in count [0]. -The remaining ones that are not connected to ceiling are isolated islands -Method: Add the hit-bricks one by one, and then make a union again to see how many are eventually connected to ceiling. The increased count is the number of dropped bricks in forward thinking!

Union Find variants

-I still use the number index to do the union find, but I increase each index by +1, shift it to the right, and [0] is reserved for special purposes: -Use union at 0 to count the total remaining count of ceiling-connected bricks, where `x = 0`. -If you are on other topics, the condition may not be `x = 0`, but you can also use this union index = 0 to make a root statistic -The key: add the last hit brick back, and then re-union around this hit-brick: the increase in count is not the number of drops when the hit-brick is missing

DFS (timeout)

-Consider the surroundings of each hit, all traverse, all without the ceiling: -For example, a matrix of 200 x 200 all 1s, the traverse must reach the top every time; the calculation is repeated, so timeout -The algorithm is correct, but it is not efficient. -I want to reduce repeated calculations, but I can't calculate in advance: the grid is constantly changing. So can you group all connected ceilings, can O (1) quickly check?

---

66. Interval Sum II.java### Level: Hard Tags: [Binary Search, Lint, Segment Tree]

SegmentTree large collection. Methods: `build`, `query`, `modify`. Not difficult. Just remember to make no mistakes.

Segment Tree

- build: recursively build children based on index-mid and link to curr node
- query: recursively try to find `node.start == targetStart && node.end == targetEnd`. Compare with node.mid
- modify: recursively try to find `node.start == targetStart && node.end == targetEnd`; modify and recursively assign upper interval with updated interval property.

---

67. HashHeap.java### Level: Hard Tags: [HashHeap, Heap]

No. It is a HashHeap implementation found from Chapter 9.

---

68. Trapping Rain Water II.java### Level: Hard Tags: [BFS, Heap, MinHeap, PriorityQueue]

Give a 2Dmap, each position has height. Find Trapping water sum.

Min Heap

-Use PriorityQueue to sort the selected height for the position, create class Cell (x, y, height).

Note a few theories

-1. Start thinking around the matrix and find that the water that the matrix can hold depends on the low-height block -2. It must be considered from the outside, because the water is wrapped inside, and at least one layer must be existing outside -The above two points prompted us to use min-heap: the PriorityQueue of natural order.

Steps

-1. During the process, you can draw a picture to make it clear: that is to walk in all four directions, subtract the height of the surrounding cells from the height of the curr cell. -2. If it is greater than zero, then the surrounding cells will have standing water: because the cell is already the lowest on the periphery, so the inside is lower, there must be standing water. -3. Every visited cell must be marked, avoid revisit -4. Create a new cell and add it to the queue according to the movements of (mX, mY) `in 4 directions: cell (mX, mY) After the stagnant water has been calculated, the outer wall hours,? It becomes a wall. -5. Because what is done is to shrink a new fence, height = Math.max (cell.h, neighbor.h); -Same idea as trapping water I. Just from the periphery, it can only be added to the horizontal plane with the same height as the peripheral cell. Going inside, it is likely that the cell height changes.
-Attach the maximum height of curr cell and move-to cell here.

Why use Heap (min-heap-priorityQueue)

-To find the shortest board of a bucket -The shortest one needs to be processed first (on top)

Why traverse from outside to inside

-Wooden barrel theory -Peel onions and discard

---

69. Find Median from Data Stream.java### Level: Hard Tags: [Design, Heap, MaxHeap, MinHeap]

Principle

-Think of the input stream as an upward slope. The middle point of the slope is naturally median.-In the first half, as maxHeap, the focus is on the peak of PriorityQueue, which is actually median.
-The second half, as minHeap, normal PriorityQueue. The beginning is minimal.

Note

-First set here, which queue stores one more element. Here select maxHeap: maxHeap.size () == minHeap.size () + 1 || minHeap.size () -You must first maintain an element in maxHeap, otherwise it will cause problems when comparing sizes.

---

70. Sliding Window Median.java### Level: Hard Tags: [Design, Heap, MaxHeap, MinHeap, Sliding Window]

The same problem of Data Stream Median: not only increasing sequence, but also removing item (maintain a window size)

MaxHeap, MinHeap

-Median still uses min-heap and max-heap. Time (logN) -Add / subtract: prioirtyQueue, log (n) -findMedian: O (1) -Add a number, subtract a number.-It is optimistic when adding or subtracting, whether it is drawn from the maxheap in the front or the minHeap behind.-Finish the balance.

Note

-When using maxHeap, minHeap, it is customary to choose one more number for maxHeap:-The maxHeap on the left always has x + 1 or x numbers -MinHeap behind should always have x numbers

---

71. Design Search Autocomplete System.java### Level: Hard Tags: [Design, Hash Table, MinHeap, PriorityQueue, Trie]

time: input: O(x), where x = possible words, constructor: O(mn) m = max length, n = # of words space: O(n^2), n = # of possible words, n = # of trie levels; mainlay saving the `Map<S, freq>`

Description is long, but in short: 做 search auto complete.

Best problem to review Trie (prefix search), Top K frequent elements (Hash Map), and MinHeap (PriorityQueue)

Easier to revisit https://leetcode.com/problems/design-search-autocomplete-system/description/

Thinking direction

-For text search, there is no doubt to use Prefix tree, trie.

Find all possible word / leaf, two options:

-? After Trie is done, do prefix search, then DFS / BFS return all leaf items. [High runtime complexity] -Store all possible words in TrieNode. [High space usage] -Shouldn't it be in memory space? Big question, so we can choose store all? possible words

Given k words, find top k frequent items. MinHeap is definitely used, but there are two solutions:

-? Store MinHeap with TrieNode: Because it will continue to search for new articles, the same prefix (especially at higher level) will be searched multiple times.

- [complexity: need to update heaps across all visited TrieNodes once new sentence is completed]
- Compute MinHeap on the fly: Of course we can't come to DFS every time? Otherwise it will be slow, so we must store list of possible candidates in TrieNode.
- Here we use `Map <String, freq>` in `Top K Frequent Words`, so O (m) is actually very convenient to build min-heap.

Train the system

-Each time a `#` is marked, an entry is added to the search history. Then insert it into trie.-This one can be done at the end when `#` is met, very concise

Trie, PriorityQueue, HashMap

- Trie Prefix Search + maintain top k frequent items
-

---

72. Integer to English Words.java### Level: Hard Tags: [Enumeration, Math, String]

Give a number less than Integer.MAX_VALUE (2 ^ 31-1), convert to English. (No need to add 'and')

String

-Basic implementation -Classification Discussion: thounsand, million, billion.3 numbers per division.-Enumerate tokens with array -Use% and / to find English translation for each segment -3-digit part, you can use a helper funtion to find the result, the processing method of each segment is the same

Note

-StringBuffer is more efficient! `Sb.insert (0, xxx)` append before sb -Note that when adding "", if it is unnecessary, try "trim ()" -Note that numbers less than 20 have their own special writing and require additional handles -This question is to be careful and patient. There is almost no algorithm. It is to write efficiently and correctly. You need to be careful.

---

73. Alien Dictionary.java### Level: Hard Tags: [BFS, Backtracking, DFS, Graph, Topological Sort]

Give an array of strings: If the array is sorted according to a new alphabet dictionary, you need to find the alphabet.

It is possible to have multiple sorting methods, just give one.

Graph

-Essence: two strings above and below, corresponding to the same index, if the letters are different, the letter on the first line is more advanced in the alphabet -Turn string array into graph of topological sort: `map <char, list <char >>` -Also `List [26] edges` (Course Schedule problem)

- Build edges: find char diff between two row, and store the order indication into graph
- Note: indegree is always reversed (established in the opposite way to node to neighbors)

BFS

-topological sort itself is well written, but first understand the nature of alphabetical sorting in the title -In fact, the nature of the above sorting is very imaginary, but it will be a bit difficult to think of it as the code for constructing the graph. -Calculate indegree, then use BFS to find nodes with inDegree == 0 -The first node with inDegree == 0 is placed before the alphabet. -The following solution uses Graph: map <Character, List > instead of List [26], in fact, try a dictionary with more than 26 letters.

- 如果 `inDegree.size() != result.length()`, there is nodes that did not make it into result.
- ex: cycle nodes from input, where inDegree of a one node would never reduce to 0, and will not be added to result
- In this case, it will be treated as invalid input, and return ""

DFS

-It is exactly the same as the process of setting up grpah by BFS -The difference in DFS is: use visited map to mark the places you pass -When you get to leaf, add to result: but only add because you have reached the end, the final order should be reversed (or, sb.insert (0, x) directly add in reverse order)

---

74. Word Ladder II.java### Level: Hard Tags: [Array, BFS, Backtracking, DFS, Hash Table, String]

Give a string of string, start word, end word. Find all shortest path list from startWord-> endWord.

Variation: mutate 1 letter at a time.

BFS + Reverse Search

-Find the shortest path with BFS.

- 问题: how to effectively store the path, if the number of paths are really large?
- If we store Queue<List>: all possibilities will very large and not maintainable
- Make a reverse structure with BFS, and then reverse search

BFS Prep Step

-BFS finds all the places where the start string can go and put it in an overall structure: Note, the way to store Map <s, list of sources> -BFS changes by 1 step each time, so recording the distance is actually the shortest path candidate (stop here)

-
    1. 反向mutation map: `destination/end string -> all source candidates` using queue: `Mutation Map`
- Mutation Map<s, List>: list possible source strings to mutate into target key string.
-
    2. 反向distance map: `destination/end string -> shortest distance to reach dest`
- Distance Map<s, possible/shortest distance>: shortest distance from to mutate into target key string.
- BFS prep step didn't solve the problem, even did not use end string. We need to use the reverse mapping structure built by BFS for search

Search using DFS

-Scan from end string, find all candidate dates && only visit candidate that is 1 step away -dfs until start string is found.

Bi-directional BFS: Search using BFS

-The reversed structure is ready, now you can search: you can also use bfs.

- `Queue<List<String>>` to store candidates, searching from end-> start

---

75. Text Justification.java### Level: Hard Tags: [Enumeration, String]

Adjust the text according to the rules. It is in Word: there is a line too long, adjust the space in the middle of the word, and then ensure the total width of each line.

There are some detailed rules, see the original question

String

- Summing space = `width + (size-1)`. maintain: 1. list of candidates, 2. width of actual words
- calculate space in between: `remain/(size - 1)`
- overall for loop; clean up list: 1. over size; 2. last item
- It's not difficult at all, but be careful: when dealing with list of string, pay attention to the clean sum size of list .
- `Clean processing space`: only (n-1) items are processed, then the last one is taken out of the for loop, special processing.

Notes

- Clarification, observation:
- can start with greedy approach to stack as many words as possible
- once exceed the length, pop the top, and justify the added words (untouched words tracked by index)
- left justify: given list/stack of words with size t, overall remaining space length m,
- deal with last line with special care: just fill one space, and fill the rest of the row with space
- Does not seem very complicated, but need additional care of calculating the amount of space needed.
- Overall runtime: O(n) to go over all space
- Overall space O(maxWidth) for maxWidth amount of strings

---

76. Read N Characters Given Read4 II - Call multiple times.java### Level: Hard Tags: [Enumeration, String]

Read N Character using `Read4 (char [] buf)` enhanced version: can read continuously read (buf, n)

String

-Pay attention to the index handle of String, slowly write edge case -Understand the meaning of the title: `read4 (char [] buf)` like `populate input object` has slightly less functionality. -When you come across, carefully understand the function usage, don't panic. In fact, the way of thinking is very simple, just carefully handle the string and edge case.

---

77. Frog Jump.java### Level: Hard Tags: [DP, Hash Table]

The question of Frog jump needs a little understanding: each grid can jump k-1, k, k + 1 steps, and k depends on the number of steps jumped in the previous step. By default, 0-> 1 must be a step.

Note: int [] stones is the unit where stone is located (not the number of steps that can be skipped, don't understand it wrong).

DP

-I originally wanted to do it according to the corrdiante dp, but found many problems, and needed to track different possible previous starting spots. -According to jiuzhang answer: By definition, use a map of <stone, Set <possible # steps to reach stone >> -Each time a stone is processed, it takes three steps according to its own set of : k-1, k, or k + 1 steps. -Take a step each time to see if stone + step exists; if it exists, add it to the hash set of next position: stone + step

Note init

- dp.put (stone, new HashSet <> ()) mark the existence of each stone - dp.get (0) .add (0) init condition, used for dp.put (1, 1)

Thought

-In the end, think mode, more like BFS mode: starting from (0,0), add all possible ways

- 然后again, try next stone with all possible future ways ... etc

---

78. Longest Substring with At Most Two Distinct Characters.java### Level: Hard Tags: [Hash Table, Sliding Window, String, Two Pointers]

As the title.

Two Pointer + HashMap

-Originally wanted to use DP, but the idea of practical sliding window -Cutting of sliding window: use hashmap to store last occurrance of char index; -After map.remove (c), that section is completely cut off; then map.get (c) + 1 is the new left window border

---

79. Shortest Distance from All Buildings.java### Level: Hard Tags: [BFS]

It is very similar to Walls and Gates, except that this question is to choose a coordinate, having shortest sum distance to all buildings (marked as 1).

BFS

- BFS 可以 mark shortest distance from bulding -> any possible spot.
- Try each building (marked as 1) -> BFS cover all 0.
- time: O(n^2) * # of building; use new visited[][] to mark visited for each building.
- O(n^2) find smallest point/aggregation value.
- Note, this question we update grid [] [] sum up with shortest path value from building.
- Find a min value at the end, even without return coordinates.
- Analyzed, not written yet.

---

80. Sliding Window Maximum.java### Level: Hard Tags: [Deque, Heap, Sliding Window]

Deque, Monotonous tail

- 维持monotonuous queue: one end is always at max and the other end is min. Always need to return the max end of queue.
- when adding new elements x: start from small-end of the queue, drop all smaller elements and append to first element larger than x.
- when sliding window: queue curr window max-end, remove it if needed.
- Wonderful: Use a deque data structure (actually in the form of LinkedList) to make a decreasing queue.
- Every time the smaller than the current node is removed, the rest is naturally: the largest> the second largest> the third largest ... ETC.
- We only care about the existence of the maximum value; any value that is less than the current value (which is to be added when new) cannot reach the maximum value anyway, so throw it away!

---

81. Median of Two Sorted Arrays.java### Level: Hard Tags: [Array, Binary Search, DFS, Divide and Conquer]

Famously find the median of two sorted arrays. Definition of median: if the total length of two arrays is even, take the average. The problem requires to be solved in log (m + n) time

-When you see log (m + n), I think of binary search, or recursive. -The two sorted arrays are jagged, and certainly not a simple binary search

Divide and Conquer, recursive

-Here is a mathematical exclusion idea: consider the intermediate points of A and B respectively. -If A [mid] <B [mid], then A [0 ~ mid-1] is not in the range of median, which can be excluded. This is how divide / conquer comes. -See the code for the specific logic, which roughly means: compare the positions of A and B [x + k / 2-1] each time, and then do the range exclusion method

- end cases:
- 1. If we find that the start index of A or B in dfs () overflows, then this is the simplest case: midian must be in another array
- 2. If k == 1: find 1st item in A / B, then make Math.max (A [startA], B [startB])
- The total number length is (m + n) and every time the general content is deleted, then time is O (log (m + n))

---

82. Bus Routes.java### Level: Hard Tags: [BFS]

---

83. Sliding Puzzle.java### Level: Hard Tags: [BFS, Graph]

---

84. Cracking the Safe.java### Level: Hard Tags: [DFS, Greedy, Math]

Greedy, Iterative

- For 2 passwords, the shortest situation is both passwords overlap for n-1 chars.
- We can use a window to cut out last (n-1) substring and append with new candidate char from [k-1 ~ 0]
- Track the newly formed string; if new, add the new char to overall result
- Note: this operation will run for k^n times: for all spots of [0 ~ n - 1] each spot tries all k values [k-1 ~ 0]
- Same concept as dfs

DFS

- Same concept: use window to cut out tail, and append with new candidate
- do this for k^n = Math.pow(k, n) times

---

85. Redundant Connection II.java### Level: Hard Tags: [DFS, Graph, Tree, Union Find]

Union Find

-Discuss 3 situations

- http://www.cnblogs.com/grandyang/p/8445733.html

---

86. The Maze III.java### Level: Hard Tags: [BFS, DFS, PriorityQueue]

BFS

-Similar to Maze I, II, use a Node [] [] to store each (x, y) state.

- Different from traditional BFS(shortest path): it terminates BFS when good solution exists (distance), but will finish all possible routes
- 1. Termination condition : if we already have a good/better solution on nodeMap[x][y], no need to add a new one
- 2. Always cache the node if passed the test in step1
- 3. Always offer the moved position as a new node to queue (as long as it fits condition)
- 4. Finally the item at nodeMap[target.x][target.y] will have the best solution.

---

87. Regular Expression Matching.java### Level: Hard Tags: [Backtracking, DP, Double Sequence DP, Sequence DP, String]

As with WildCard Matching, discuss the situation clearly with string p last char is '' *and not* ''

The difference here is that '*' needs to have a preceding element, then:

- repeat 0 times
- repeat 1 times: need s[i-1] match with prior char p[i-2]

---

88. Wildcard Matching.java### Level: Hard Tags: [Backtracking, DP, Double Sequence DP, Greedy, Sequence DP, String]

Double sequence DP. Much like regular expression.

Double Sequence DP

-Analyze the true meaning of the characters?, * And write the expression. -Dp [i] [0] should always be false when initializing. When p is an empty string, it cannot be matched anyway (unless s = "" as well) -At the same time dp [0] [j] is not necessarily false. For example, s = "", p = "*" is a matching.

- A. p[j] != '*'
  1. last index match => dp[i - 1][j - 1]
  2. last index == ? => dp[i - 1][j - 1]
- B. p[j] == "*"
  1.
     - is empty => dp[i][j - 1]
  2.
     - match 1 or more chars => dp[i - 1][j]

---

89. Robot Room Cleaner.java### Level: Hard Tags: [Backtracking, DFS]

DFS

- Different from regular dfs to visit all, the robot move() function need to be called, backtrack needs to move() manually and backtracking path shold not be blocked by visited positions
- IMPORTANT: Mark on the way in using set, but backtrack directly without re-check against set
- Mark coordinate 'x@y'
- Backtrack: turn 2 times to revert, move 1 step, and turn 2 times to revert back.
- Direction has to be up, right, down, left.
- you [] dx = {-1, 0, 1, 0};, you [] dy = {0, 1, 0, -1};

---

90. Maximum Vacation Days.java### Level: Hard Tags: [DP]

Review (5)

0. Maximum Subarray III.java### Level: Review Tags: []

---

1. Valid Perfect Square.java### Level: Review Tags: [Binary Search, Math]

Binary looks for sqrt. Basic mid + 1, mid-1 template. 注意: define index as long.

---

2. Maximum Average Subarray II.java### Level: Review Tags: [Array, Binary Search, PreSum]

Give int [] nums and window min size k. The window size can be greater than K. Find the largest continuous series of average value.

-The idea of Binary Search, used on the average sum you are looking for. The size is in [min, max] -When looking for k, it can be> = k, use the concept of min (preSum). -When looking for k, draw a picture. It can be seen that what is actually required is the sum [x, i] in k window, so sum [0, i]-sum [0, x]

Need to read the notes below carefully.

---

3. The Skyline Problem.java### Level: Review Tags: [Binary Indexed Tree, Divide and Conquer, Heap, PriorityQueue, Segment Tree, Sweep Line]

Also called skyline. O (nLogN) with Sweep Line, but it seems that there are many ways: segement tree, hashheap, treeSet?

Sweep Line, Time O(nLogN), Space O(n)

- original reference http://codechen.blogspot.com/2015/06/leetcode-skyline-problem.html?_sm_au_=isVmHvFmFs40TWRt
- 画图分析: 需要找到 non-overlaping height point at current index; also height needs to be different than prev height peek to be visible.
- Divide all points, each point has index x, plus a height.
- Sort on this list, according to index and height. Note that building start point height is marked with a negative number, so that start is guaranteed to end
- Mark the start with a negative height: When comparing startPoint.height and endPoint.height with the same x-pos in the priority queue, because the end height is an integer, the start point is automatically placed before the end point when compare
- Of course, if the two start points are compared, if the negative value of the second point is too large (that is, the height is very high), it will naturally return a positive number by comparison, which will form an inverted position.
- Use max-heap (reversed priorityqueue) in processes, and then iterate heightPoints to save the maximum height. In the case of peek, it is a reasonable solution.
- Add 0 to heightQueue to close it at the end

Segment Tree

-After reading some practices, the segment tree is complicated to write. It is estimated that it is difficult to write segment tree in the interview: https://www.cnblogs.com/tiezhibieek/p/5021202.html

HashHeap

-HashHeap template can be considered: https://www.jiuzhang.com/solution/building-outline/#tag-highlight-lang-java

Binary Indexed Tree?

---

4. Remove Invalid Parentheses.java### Level: Review Tags: [BFS, DFS, DP]

Give a string with parentheses and other characters. Cut out the valid string with the least amount of knife, find all such strings.

There are multiple solutions to this problem, the strongest is O (n) space and time

DFS and reduce input string

- in dfs: remove the incorrect parentheses one at a time
- detect the incorrect parentheses by tracking/counting (similar to validation of the parentheses string): `if(count<0)`
- once detected, remove the char from middle of s, and dfs on the rest of the s that has not been tested yet.

Core concept: reverse test

- if a parenthese string is valid, the reverse of it should also be valid
- Test s with open='(', close=')' first; reverse s, and test it with open=')', close='('

Minor details

- only procceed to remove invalid parenthese when `count<0`, and also break && return dfs after the recursive calls.
- The above 2 facts eliminates all the redundant results.
- Reverse string before alternating open and close parentheses, so when returning final result, it will return the correct order.
- Open questions: how does it guarantee minimum removals?

Backtracking

-If you use a stringbuffer, you will not create a new string every time, but you need to maintain the string buffer, and you will backtracking

Complexity

- Seems to be O(n), but need to derive

BFS

EVERYTHING

DP
