Passed

Not passed

Not completed yet

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # | Difficulty | Question Title | Solving Strategy | Note | Runtime | Memory |
| 21 | Easy | Merge Two Sorted Lists | Iteration.  Use two pointers to track which elements we pick next from original list. | Will greatly speed up operation if we run out of one LinkedList by directly link result’s next empty node to the unprocessed original node. | ~~100.00%~~ | 70.56 % |
| 24 | Medium | Swap Nodes in Pairs | Iteration.  Manipulate pointers to achieve goal. |  | ~~100.00%~~ | 72.21 % |
| 53 | Easy | Maximum Subarray | Dynamic programming that stores max sum so far (by taking [this approach](https://www.bilibili.com/video/BV1Ez4y1S7WP))  Can’t use brute force (exceed time limit) | Note: there’s no way in Java to easily convert Array to List, you need to manually add all of them. | 34.33% | 99.74% |
| 50 |  | Pow(x, n) |  | Tried brute force ( O(N) ) and divide in halves (binary search, results in O(logN) ), but they both time out. |  |  |
| 98 | Medium | Validate Binary Search Tree |  | Overthink the problem being more difficult than I initially thought.  Need to use Long instead of Int as one edge case will overflow the Integer | ~~100%~~ | 89.54% |
| 100 | Easy | Same Tree | Base case: check whether tree 1 and tree 2’s node is null.  Recursive case: iterate call left children and right children. | Comparing the Node object itself is not comparing the Node’s value | ~~100%~~ | 90.60% |
| 113 | Medium | Path Sum II | Use LinkedList to store values, only copy the new list if the remaining sum is 0 and we’re at the leaf node. | There may be a simpler solution that do not need a lot of base cases. | 32.52% | 56.52% |
| 114 | Medium | Flatten Binary Tree to Linked List | Use Queue to store Pre-order traversal, and repeatedly poll Queue to flatten tree. | Should immediately return if the root is empty. | 32.46% | 57.29% |
| 143 | Medium | Reorder List | Use Dequeue to store nodes (hint: repeatedly use first and last of the “remaining” elements), and pop Stacks to return reversed Linked List.  Mistake: should not use Stack nor a regular Queue. | Forgot that size, isEmpty of Stack and Queue is a function call not a property. | 25.10% | 27.14% |
| 199 | Medium | Binary Tree Right Side View | Use two Queue method to find each layer in a tree.  Store the right-most side value while we proceed to the next layer. | I previously use a Map to store the right-most value with a layer number, but it’s inefficient. | 75.66% | 42.14% |
| 200 | Medium | Number of Islands | Same as Q695 | This question uses char[ ][ ] while Q695 uses int[ ][ ], so if I need to compare char[ ][ ] against a value (not the ASCII code), need to use char[r][c] == ‘1’ (note single quote mark) | 24.54% | 12.67% |
| 203 | Easy | Remove Linked List Elements | Use pointer fields to skip deleted value in original LinkedList.  Program will skip everything if the List only contains the value to be removed. | Need to use .next.next (jump one further) to achieve correct result.  Low memory usage compared to other Java implementations. | 74.26% | 98.24% |
| 206 | Easy | Reverse Linked List | Use Stack to store nodes (hint: reverse the order), and pop Stacks to return reversed Linked List. | This solution does not create new ListNode, just through manipulating nodes of existing pointers. | 7.35% | 86.68% |
| 224 | Hard | Basic Calculator | RPN calculator | I do remember Reverse Polish Operation beforehand. |  |  |
| 231 | Easy | Power of Two |  | WAIT until I’ve finished studying Bitwise operation |  |  |
| 234 | Easy | Palindrome Linked List | Use Dequeue (hint: repeatedly compare first and last elements) to store nodes, and repeatedly pull nodes off from both ends to see whether it’s palindrome. | O(N) runtime complexity O(N) space complexity. There’s an O(1) space complexity solution. | 34.72% | 29.47% |
| 255 | Medium | Verify Preorder Sequence in Binary Search Tree | Use Stack and compare whether the right subtree is smaller than the parent.  Use [this approach](https://www.youtube.com/watch?v=Psce8aMuX8s), but independently write code | Should not use recursion as it won’t have access to the parent node. | 74.57% | 97.22% |
| 300 | Medium | Longest Increasing Subsequence | Based on [this approach](https://www.bilibili.com/video/BV19b4y1R7K3) without looking at code, my code writeup is my own work. | CSE 417 problem | 34.49% | 57.78% |
| 339 | Medium | Nested List Weight Sum | Same as Q364 | More straightforward than Q364. | ~~100.00%~~ | 74.89% |
| 364 | Medium | Nested List Weight Sum II | First use recursion to find the depth, then use recursion to calculated weighted sum. |  | 11.29% | 22.58% |
| 366 | Medium | Find Leaves of Binary Tree | Perform DFS, increase layer only when we’re backtracking. | DFS operations can be improved by remove some auxiliary data structures. | 8.17% | 21.01% |
| 417 | Medium | Pacific Atlantic Water Flow | New after this [video](https://www.youtube.com/watch?v=krL3r7MY7Dc):  Still use bottom-up DP to determine whether we can go to the ocean, but use DFS in 4 directions to find inlands at which water can flow to ocean.  Original approach that does not pass all test cases:  Use DP concept (bottom-up) to fill out whether we can go to Pacific from 1 cell or go to Atlantic from 1 cell.  Cue to consider DP solution: if water can flow to Pacific at this cell, if the adjacent cell is higher/equal, then the adjacent cell can go to this case. | More edge cases to consider while scanning the entire 2-D array  Note: Inefficient DFS solution do exist(perform DFS on any cell; determine whether it can go to Pacific or Atlantic) | 82.55% | 65.31% |
| 472 | Hard | Concatenated Words | Use recursion (DFS, [similar to this approach](https://www.bilibili.com/video/BV1QJ411W7Bn)) to repeatedly add words to the temp string.  Use pruning to stop recursing impossible solutions.  Then I found that I need to manipulate the concatenated string itself rather than brute force using words to form a string. | Pruning was the most difficult part (need to know you don’t need to dive further as no solution would found).  Constraint is 0 ≤ words[i].length ≤ 1000  Hence we need to check empty string (str.length == 0) case and skip it.  A [dynamic programming solution](https://www.bilibili.com/video/BV1gf4y1p7PF) is more optimal but I my original (inefficient) DFS is my own idea. |  |  |
| 473 | Medium | Matchsticks to Square | Use [this approach](https://www.bilibili.com/video/BV1D5411j7UE), with my modified code | Did not initially realize that pre-calculate the length of square side (total matchstick length ÷ 4) is the starting point to solve this problem. Then this problem is a regular choose-test-unchoose recursion problem. | 42.16% | 96.11% |
| 509 | Easy | Fibonacci Number | Bottom-up approach, as it only needs two number before the n for each computation. | CSE 417 problem | ~~100%~~ | 82.92% |
| 513 | Medium | Find Bottom Left Tree Value | Use two Queue method to find layers, and store the first value in the layer right above the inner while loop. |  | 10.48% | 6.76% |
| 515 | Medium | Find Largest Value in Each Tree Row | Double Queue method from lecture. |  | 10.84% | 13.31% |
| 673 | Medium | Number of Longest Increasing Subsequence | Similar to Q300, and I’ve wrote the CSE 417 code to extract the sequence itself | Partially correct, it does not pass all test cases.  Did not realize that the number of unique increasing subsequence itself should also be stored as DP state. |  |  |
| 674 | Easy | Longest Continuous Increasing Subsequence | Use similar approach to Q53 | Need to assign return value to 1 at first instead of Integer.MIN\_VALUE, otherwise it will fail in case when array contains 1 number only. | 99.06% | 65.46% |
| 690 | Medium | Employee Importance | Use two HashMap to store every employee’s importance and subordinates, where two Map’s Key is employee’s ID.  Cue for using HashMap: employee ID are guaranteed to be unique but values are not unique. | Forgot that Map remove value by providing a Key, as I’m thinking whether there’s other Map’s remove methods. | 8.39% | 6.92% |
| 695 | Medium | Max Area of Island | Use regular 2-D array iteration from top left to bottom right (row by row). When we encounter 1, recursively scan top down right left (order matters!) until we run out of recursion. | Two edge cases:  Need to use visited[][] to track whether array has been visited.  Need to account when grid has no island or the largest island is ⅃ shaped | 31.36% | 72.81% |
| 1583 | Medium | Count Unhappy Friends |  | Similar to CSE 417 Homework 1 (Stable Matching problem using Gale Shapley algorithm) |  |  |