info@ndcamps.com +1(289)925-2990

Day1

二分法：

有序的数据，每次通过判断逻辑排除掉一部分的答案，直到触发终止条件

(2)二分法实现模板（可以递归，可以迭代；一般以迭代为主）

(3)移动两个指针（start与end）的含义？移动条件是什么（筛选掉一部分数据的依据）？循环的截止条件？

(4)数据中是否有重复数字？对结果有什么影响？

(5)为什么你选择的模板中使用start < end 或者 start <= end 或者 start + 1 < end 作为终止条件？这样写是如何避免死循环的？不这么写在什么情况下会出现死循环？

(6)在处理逻辑中，当前结果>, <, = 目标值时分别如何处理？移动指针的依据是什么？

(7)循环退出后是否需要额外处理？

(8)如果遇到corner case根本没进主循环，你的代码是否能正常工作？

(9)为什么Java需要写 mid = start + (end - start) / 2 而 Python可以直接写 mid = (start + end) // 2 ？

(10)如何理解从基本的朴素二分，到相对复杂的条件二分，到更加抽象的答案二分？（在一个显性有序数组中一次砍掉一部分 --> 在一组有规律的数据上利用判断条件逐步缩小范围 --> 在一个有序的抽象模型里，利用不断的"猜测+检验"逐步逼近最终结果）

朴素二分法：Binary Search

https://leetcode.com/problems/binary-search/

条件二分法：Search in Rotated Sorted Array
https://leetcode.com/problems/search-in-rotated-sorted-array/

（条件二分法：Search in Rotated Sorted Array II，follow up）
https://leetcode.com/problems/search-in-rotated-sorted-array-ii/

info@ndcamps.com +1(289)925-2990

多指针：

多指针是一个非常广泛的概念，并不是一个固定的算法。但基本上是通过一些变量的控制与循环把问题的复杂度控制在一两层for循环之内。可以用在数组、链表、区间、滑动窗口、流、回文串、和差问题等多个场景。（前项和其实并不完全是指针问题，但也归并在这里）。

(2)Quick Sort和Merge Sort的基本原理与实现，排序的稳定性问题

(3)Quick Select的实现与复杂度

(4)同向指针与相向指针的使用场景

(5)不同场景下循环终止条件？

(6)两数之和，之差，特定条件下(大小于某值等）的计数问题

(7)三数或三数以上之和的通用写法(两数之和+搜索)

(8)数组有没有排序？是否需要排序？

(9)数组有没有去重？是否需要去重？

(10)离线数据（内存中，有限长）还是在线数据（无法放入内存，长度未知）？

(11)链表操作中dummy node与previous node的使用技巧

(12)链表的中点，判断是否有环，寻找环的交叉点

Day2

数组：

912. Sort an Array (Quick Sort and Merge Sort) https://leetcode.com/problems/sort-an-array/

75. Sort Colors https://leetcode.com/problems/sort-colors/

Day3

链表：21. Merge Two Sorted Listshttps://leetcode.com/problems/merge-two-sorted-lists/

区间：Lint-391. Number of Airplanes in the Skyhttps://www.lintcode.com/problem ... the-sky/description

Day4

滑动窗口：

3. Longest Substring Without Repeating Characters https://leetcode.com/problems/lo ... peating-characters/

前项和：

53. Maximum Subarray https://leetcode.com/problems/maximum-subarray/

Day5

和差问题：

Two Sum https://leetcode.com/problems/two-sum/

15. 3Sum https://leetcode.com/problems/3sum/

BFS

(2)BFS主要几种场景： 层级遍历，拓扑排序，图上搜索（包括二叉树，矩阵）

(3)Queue的使用技巧，BFS的终止条件？

(4)什么时候使用分层？什么时候不需要？实现的时候的区别在哪里？

(5)拓扑排序的概念？如何判断是否存在拓扑排序？是否存在唯一的拓扑排序？找到所有拓扑排序？

(6)什么时候需要使用set记录访问过的节点？（为什么二叉树上的BFS往往不需要set？）什么时候需要map记录到达过的节点距离？

(7)如何在矩阵中遍历下一步的所有节点？如果每次可能走不止一步怎么办（Maze II）？

(8)为什么BFS解决的基本都是简单图（边长为1）问题？如果边长不为1，该怎么办？

(9)BFS的时空复杂度估算？

(10)如何使用双向BFS进行优化？

Day6

二叉树:

297. Serialize and Deserialize Binary Tree https://leetcode.com/problems/se ... ialize-binary-tree/

拓扑排序:

Lint-127. Topological Sorting https://www.lintcode.com/problem/topological-sorting/description

Day7

矩阵：

200. Number of Islands https://leetcode.com/problems/number-of-islands/

图:

133. Clone Graph https://leetcode.com/problems/clone-graph/

二叉树与递归

(1)理解二叉树、平衡二叉树、二叉搜索树的关系和概念。

(2)理解递归的概念和方法，递归三要素。

(3)在解决递归问题的时候，有时可以返回多个值（Python），或者用一个额外的class包装多个值（Java）。

(4)熟练掌握用递归和非递归的方式分别前序、中序、后序遍历二叉树的方法。

(5)理解掌握分治和遍历的区别和联系。

(6)理解掌握top-down, buttom-up的思路。

(7)理解掌握二叉树上的Iterator。

Day8

二叉树前中后序遍历（需要熟练掌握非递归方式）：

94. Binary Tree Inorder Traversal
https://leetcode.com/problems/binary-tree-inorder-traversal/

95. Binary Tree Preorder Traversal
https://leetcode.com/problems/binary-tree-preorder-traversal/

Day9

96. Binary Tree Postorder Traversal
https://leetcode.com/problems/binary-tree-postorder-traversal/

反向复原二叉树：

97. Construct Binary Tree from Preorder and Inorder Traversal
https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/

Day10

Iterator相关：

173. Binary Search Tree Iterator https://leetcode.com/problems/binary-search-tree-iterator/

子树问题：

111. Minimum Depth of Binary Tree
https://leetcode.com/problems/minimum-depth-of-binary-tree/

info@ndcamps.com +1(289)925-2990

DFS

(1)DFS中递归的基本要素

(2)终止条件的选择；回溯；剪枝

(3)什么时候需要排序？

(4)如何去除重复元素？一个元素允许使用多次的情况？

(6)在图上进行DFS如何避免回到重复节点

(5)识别一个隐式图，并使用DFS

(6)在某些情况下，利用记忆化搜索进行优化

Day11

39. Combination Sum

https://leetcode.com/problems/combination-sum/

40. Combination Sum II

https://leetcode.com/problems/combination-sum-ii/

Day12

46. Permutations https://leetcode.com/problems/permutations/

47. Permutations II https://leetcode.com/problems/permutations-ii/

Day13

78. Subsets https://leetcode.com/problems/subsets/

90. Subsets II https://leetcode.com/problems/subsets-ii/

info@ndcamps.com +1(289)925-2990

数据结构

本章按照数据结构分类一些问题，和之前按算法分类的题目相比可能会有重复，因为一道题可能有多个标签。

(2) 对于每种数据结构，需要先学习掌握其基本原理，优缺点，复杂度，和对应语言中的API用法。对于其基本的实现方式也要了解。

(3) Array，Matrix，String，Hash都是一些常用的数据结构，一般在各种题里都会用到，这里主要列举一些没有涉及到其他算法的题目。

(4) Linked List往往自成一类，会涉及到一些pointer操作，需要细心。

(5) Queue一般用在BFS里面比较多，这里不单独列举了。

(6) Heap，Stack往往和其他知识点混用，但自己单独出题也可以。

(7) Trie，Union Find，Sweep Line的套路比较明显，需要记住模板。

(8) Binary Index Tree 和Segment Tree涉及到的题目有限，需要记住模板。Segment Tree解法一般来说可以覆盖BIT能解决的问题，但是BIT写起来短一些。

(9) 复合数据结构里面LRU和LFU相对比较重要。其他的在掌握基本数据结构即复杂度之后，可以随机应变。

Day14

Linked List:

2. Add Two Numbers https://leetcode.com/problems/add-two-numbers/

21. Merge Two Sorted Lists https://leetcode.com/problems/merge-two-sorted-lists/

Day15

Hash:

706. Design HashMap https://leetcode.com/problems/design-hashmap/

Heap:

info@ndcamps.com +1(289)925-2990

23. Merge k Sorted Lists https://leetcode.com/problems/merge-k-sorted-lists/

Day16

Stack:

155. Min Stack https://leetcode.com/problems/min-stack/

Monotonic MStack:

300. Longest Increasing Subsequence (Patience Sort)
https://leetcode.com/problems/longest-increasing-subsequence/

Day17

Trie:

208. Implement Trie (Prefix Tree) https://leetcode.com/problems/implement-trie-prefix-tree/

Union Find:

200. Number of Islands https://leetcode.com/problems/number-of-islands/

Day18

Sweep Line:

Lint-391. Number of Airplanes in the Sky https://www.lintcode.com/problem ...
the-sky/description

Binary Index Tree & Segment Tree:

307. Range Sum Query - Mutable https://leetcode.com/problems/range-sum-query-mutable/

Day19

Complex Data Structure:

146. LRU Cache https://leetcode.com/problems/lru-cache/

460. LFU Cache https://leetcode.com/problems/lfu-cache/

info@ndcamps.com +1(289)925-2990

动态规划

动态规划更准确的说是一种数学思想，而不是一种算法。学习曲线相对于前面的算法会比较陡峭，如果是有天赋的大佬，可能可以很快领悟。但是对于大部分平均水平的同学，可能需要前后间隔几个礼拜甚至几个月，反复思考两三遍才能顿悟并运用。所以作为初学者，一时半会想不明白没关系，隔几天回来再多看几次就能渐渐理解了。

(2) 不过针对目前的面试，除了少数那几家公司之外，动态规划的出现频率其实没有那么高，而且主要也都是中等难度的题目。所以如果准备时间有限，建议优先把时间放在前面的算法上，动态规划可以先看几道中等难度经典题，其他的题目后面有时间再看。

(3) 关于一道题是用动态规划还是用贪心法，一般来说时间复杂度类似的时候优先用动态规划，因为通用性、可解释性都比较强。而自己凭空想出来的贪心法，不但不容易解释，而且很容易是错的，面试风险相对比较高。不过有一些题目确实是贪心法最优，作者在后面也列出了几题，如果碰到原题或者类似题，可以参考。

(4) 对于新手而言，在学习动态规划的时候，看懂题目在问什么之后就可以在网上找答案了，别自己瞎折腾。网上各种大佬的博客有详细的图文解释，慢慢揣摩理解。

(5) 动态规划的一般思路是数学归纳法，就是用递推的方式把大问题（最终问题）分解为小问题（前置问题），然后一路倒推到边界；在边界附近计算出初始状态后，再原路反向往回计算，最后得到所求解。所以对于绝大部分题目，都需要遵循：分解子问题，写出转移方程，描述边界条件，计算出最终解这几个步骤。

(6) 有些动态规划问题，可以通过滚动数组的方式优化空间复杂度，一般可以降一个维度。但是要注意运算的方向，需要避免前序的结果在被用到之前就被覆盖掉的情况。

(7)大部分动态规划都是求解“可行性”，“最值”问题，如果有些题目要求输出结果，也可以考虑用“打印路径”的方式。

(8) 很多问题，通过细微的改一些条件，就会变成另外一道题，解法思路会产生明显差异，所以审题要小心。比如背包类问题，是否可以重复选同一个物品，是否有重复物品，求解最大重量还是最大价值， 背后的原理可能会产生变化。有时候是组合问题，有时候是排列问题，还叠加了是否可以重复的情况，需要透彻的理解。另外在解法上，比如说，正着走一遍循环和倒着走一遍循环可能代表的是两种不同的思考方式，这些往往需要反复细致的理解才能完善自己的思维体系。

(9) 有些问题需要求“所有可行解”，这时候往往会使用搜索（DFS，BFS）的方法。但为了进行时空优化，记忆化搜索也会常常被用到。其实DFS记忆化搜索和常规动态规划写法常常是一个思维的两种实现方式，在不同的题目中各有优劣。

(10) 在面试动态规划的时候，重点在于能够比较清晰地画图描述并解释清楚所写的动态方程，让面试官理解你的思路，注意初始化以及for循环的起始条件。至于代码本身，往往是for循环为主，一般也不长。

Day20

Backpack:

Lint-92. Backpack https://www.lintcode.com/problem/backpack/description

Matrix:

62. Unique Paths https://leetcode.com/problems/unique-paths/

Day21

刷题营总结

-
-