# data-structure-algorithm-practice

# 目录



## ▫ Single-Linked-List

```
使用dummy node指向head可以保留最原始的head reference
```

### 使用linked list的特性遍历

- 2.add-two-numbers

### 使用linked list的特性reference node

- 138.copy-list-with-random-pointer

### Reverse & Swap

- 206.reverse-linked-list
- 24.swap-nodes-in-pairs

### insert

- 708.insert-into-a-sorted-circular-linked-list

### Remove

- 203.remove-linked-list-elements
- 237.delete-node-in-a-linked-list

### Two Pointers/Multiple Pointers with Linked List

- 876.middle-of-the-linked-list
- 19.remove-nth-node-from-end-of-list
- 21.merge-two-sorted-lists
- 23.merge-k-sorted-lists [heap + k pointers]

### hash map/list 存储linked list记录

- 141.linked-list-cycle
- 83.remove-duplicates-from-sorted-list
- 148.sort-list
- 160.intersection-of-two-linked-lists

### 结合了多种基础操作

- 234.palindrome-linked-list
- 61.rotate-list
- 328.odd-even-linked-list [双指针以同样速度前进，最后拼接]
- 92.reverse-linked-list-ii [快慢指针 + reverse]

#### hard

```
for each group, disconnect, then reverse, and reconnect
```

- 25.reverse-nodes-in-k-group

## ▫ Doubly-Linked-List

- 146.lru-cache

- 460.lfu-cache
- 426.convert-binary-search-tree-to-sorted-doubly-linked-list

## String

```
String一般的题型有
   - 和math结合找规律
   - 和hash map结合
   - 和two pointers结合
   - 和其他概念结合，比如stack，merged intervals，etc
```

### Palindrome & Two Pointers

- 125.valid-palindrome
- 266.palindrome-permutation
- 680.valid-palindrome-ii(variation of palindrom string; two pointers)
- 408.valid-word-abbreviation (string & two pointers; take care of edge cases)

### Hash Table or Sort

- 249.group-shifted-strings
- 791.custom-sort-string(string & sorting & hash table)
- 616.add-bold-tag-in-string(string和merged interval结合)

### 和math结合找规律

- 415.add-strings
- 273.integer-to-english-words (hard; take care of edge cases)
- 65.valid-number (hard; string & math principle)

## Hash-Map

- 706.design-hashmap
- 1.two-sum (use hash map to record previous record)
- 350.intersection-of-two-arrays-ii
- 128.longest-consecutive-sequence
- 49.group-anagrams (use tuple as key)
- 380.insert-delete-getrandom-o1 (store list index as value so we could access the location in the list)
- 348.design-tic-tac-toe(use hash map to record a certain state)
- 36.valid-sudoku
- 1146.snapshot-array
- 146.lru-cache(OrderedDictionary)
- 460.lfu-cache (Hard)

## Stack

```
栈是一种后进先出（LIFO）的数据结构，只能在一端（栈顶）插入和删除元素，而python中的列表的append()方法对应的就是向栈顶添加元素，列表的pop()方法对
```

### 正常类型:利用Stack结构或特性

- 20.valid-parentheses
- 1047.remove-all-adjacent-duplicates-in-string
- 735.asteroid-collision
- 1190.reverse-substrings-between-each-pair-of-parentheses

### Stack进行operation

```
思路主要在于遇到 （ 与遇到 ） 分别该如何操作。一般遇到（ 前做一系列操作，遇到（ 时append to stack，遇到 ）时pop from stack
```

- 394.decode-string
- 227.basic-calculator-ii
- 224.basic-calculator

## Monotone-Stack:

```
基础知识：单调栈一般用于解决数组中找出每个数字的第一个大于 / 小于该数字的位置或者数字；
单调队列只见过一道题需要使用；
不论单调栈还是单调队列，单调的意思是保留在栈或者队列中的数字是单调递增或者单调递减的
increasing stack:
    - iterate elements
        - while stack and i <= stack[-1]
            - stack.pop(), do something
        - append elements to stack
decreasing stack:
    - iterate elements
        - while stack and i >= stack[-1]
            - stack.pop(), do something
        - append elements to stack
```

- 496. next-greater-element-i
- 739. daily-temperatures
- 402. remove-k-digits
- 456. 132-pattern
- 316. remove-duplicate-letters
- 1124. longest-well-performing-interval
- 1130. minimum-cost-tree-from-leaf-values(use m-stack to shrink element with multiplications; considering the min/max relation)

#### Hard

- 42. trapping-rain-water
- 84. largest-rectangle-in-histogram
- 85. maximal-rectangle
- 239. sliding-window-maximum

## Heap

```
Adding to/removing from the heap (or priority queue) only takes O(logk) time when the size of the heap is capped at k elements.

Heap主要的题型有
    - 找第k大或第k小的元素
    - 找前k个无序元素
```

### 找第k大或第k小的元素

- 215.kth-largest-element-in-an-array
- 378.kth-smallest-element-in-a-sorted-matrix

### 找前k个无序元素

- 629.top-k-frequent-words
- 973.k-closest-points-to-origin
- 347.top-k-frequent-elements

### 利用min heap/max heap特性

- 295.find-median-from-data-stream

### merge k sorted list

- 23.merge-k-sorted-lists (heap + k pointers)
- 632.smallest-range-covering-elements-from-k-lists ( hard; 结合了k sorted list的思想，同时需要更新interval，要找到维护interval的规律 )

## Tree

### Traverse

```
- 树一般有两种traverse方式，一种为DFS，另一种为BFS。一般需要level信息的时候可用BFS。
- 树也可以用iteration traverse
    - stack = [(root, False)]
    - while stack
        - node, is_visit = stack.pop()
        - continue if not node
        - if is_visit: do something
        - else: append in the reverse order
```

- 145. binary-tree-postorder-traversal
- 94. binary-tree-inorder-traversal (DFS/Iterative)
- 589. n-ary-tree-preorder-traversal (DFS)
- 144. binary-tree-preorder-traversal (DFS/Iterative)
- 102. binary-tree-level-order-traversal (BFS)
- 103. binary-tree-zigzag-level-order-traversal (BFS)
- 107. binary-tree-level-order-traversal-ii
- 314. binary-tree-vertical-order-traversal(BFS with col index hash map)

### Construct

```
树的构建一般需要在每层recursion创建新的node: node.val, node.left, node.right.
```

- 108. convert-sorted-array-to-binary-search-tree
- 105. construct-binary-tree-from-preorder-and-inorder-traversal
- 114. flatten-binary-tree-to-linked-list

### Path | Depth | Inverse | Others with Devide and Conquer

```
树的其他问题一般都由Devide and Conquer解决
正常divide and conquer思路：在递归的每一层，node需要做什么，左子树需要做什么，右子树需要做什么
```

#### Easy

- 104. maximum-depth-of-binary-tree
- 101. symmetric-tree
- 226. invert-binary-tree
- 543. diameter-of-binary-tree
- 257. binary-tree-paths
- 110. balanced-binary-tree
- 100. same-tree
- 112. path-sum

#### Medium

- 236. lowest-common-ancestor-of-a-binary-tree
- 222. count-complete-tree-nodes
- 113. path-sum-ii
- 129. sum-root-to-leaf-numbers
- 662. maximum-width-of-binary-tree
- 199. binary-tree-right-side-view
- 116. populating-next-right-pointers-in-each-node

#### Hard

- 124. binary-tree-maximum-path-sum
- 297. serialize-and-deserialize-binary-tree

## Binary-Search-Tree

```
BST特征：中序遍历为单调递增的二叉树，换句话说，根节点的值比左子树任意节点值都大，比右子树任意节点值都小，增删查改均为O(h)复杂度，h为树的高度；

BST的搜索：
    - while node
    - check larger or smaller, node.next
```

- 270. closest-binary-search-tree-value
- 98. validate-binary-search-tree
- 96. unique-binary-search-trees
- 173. binary-search-tree-iterator
- 230. kth-smallest-element-in-a-bst
- 99. recover-binary-search-tree

- 1008. construct-binary-search-tree-from-preorder-traversal
- 108. convert-sorted-array-to-binary-search-tree

## Trie

基础知识：（https://zh.wikipedia.org/wiki/Trie）；多数情况下可以通过用一个set来记录所有单词的prefix来替代，时间复杂度不变，但空间复杂度略高

创建trie逻辑：

```
1. trie as empty dictionary
2. for word in words; node = trie
3. for char in word; node = node.setdefault(char, {}); at the end set the final node['#'] as word
```

- 720. longest-word-in-dictionary
- 208. implement-trie-prefix-tree
- 692. top-k-frequent-words
- 421. maximum-xor-of-two-numbers-in-an-array

#### Hard

- 212. word-search-ii

## Data-Structure-Design

- 146. lru-cache (OrderedDict)
- 380. insert-delete-getrandom-O(1) (hash map + list)
- 359. logger-rate-limiter (hash map)
- 981. time-based-key-value-store (binary search + hash map, use sorted timestamp as search key; similar question: 1146. snapshot-array)

## Binary-Search

基础知识：二分法是用来解法基本模板，时间复杂度logN；常见的二分法题目可以分为两大类，显式与隐式，即是否能从字面上一眼看出二分法的特点：要查找的数

```
bisect.bisect_left returns the leftmost place in the sorted list to insert the given element.
bisect.bisect_right returns the rightmost place in the sorted list to insert the given element.
```

### 显式二分法

```
while start + 1 < end: mid = (start +_end) // 2
    - if looking for leftmost position: if array[mid] >= target: end = mid; else: start = mid
    - if looking for rightmost position:  if array[mid] <= target: start = mid; else: end = mid
```

- 278.first-bad-version
- 33.search-in-rotated-sorted-array
- 34.find-first-and-last-position-of-element-in-sorted-array
- 74.search-a-2d-matrix
- 153.find-minimum-in-rotated-sorted-array (find the fisrt rotated idx, then search either side)
- 162.find-peak-element
- 658.find-k-closest-elements (binary search + two pointers)
- 528.random-pick-with-weight (binary search + prefix sum)
- 1060.Missing Element in Sorted Array (need transformation into binary search)

#### Hard:

- 4.median-of-two-sorted-arrays (hard, advanced comparison and search)
- 302.smallest-rectangle-enclosing-black-pixels
- 154.find-minimum-in-rotated-sorted-array-ii (variant of 153) search space reduction: usually when row or column is sorted whereas another is not
- 1428.leftmost-column-with-at-least-a-one
- 240.search-a-2d-matrix-ii

### 隐式二分法

- [69. https://leetcode.com/problems/sqrtx/] (search space reduction)
- [540.https://leetcode.com/problems/single-element-in-a-sorted-array/] (search for an element has different pattern with others)
- 1062.longest-repeating-substring (search for a True/False boundary)

#### Find the value in a bounding range

```
Given the number of bags, return the minimum capacity of each bag, so that we can put items one by one into all bags.

We binary search the final result.
    - The left bound is max(A),
    - The right bound is sum(A).
```

- 1891.cutting-ribbons
- 410.split-array-largest-sum (hard)
- 1231.divide-chocolate (hard)
- 875.koko-eating-bananas
- 1011.capacity-to-ship-packages-within-d-days

## Two-Pointers

```
基础知识：常见双指针算法分为三类，同向（即两个指针都相同一个方向移动），背向（两个指针从相同或者相邻的位置出发，背向移动直到其中一根指针到达边界
```

### 基本同向双指针

- 88.merge-sorted-array
- 349.intersection-of-two-arrays
- 283.move-zeroes

### 相向双指针：(以two sum为基础的一系列题)

- 167.two-sum-ii-input-array-is-sorted
- 15.3sum
- 16.3sum-closest
- 75.sort-colors

### 同向双指针(Sliding Window)

```
Longest则尽可能move right pointer, until invalid
Shortest则尽可能move left pointer, while valid
```

- 3.longest-substring-without-repeating-characters
- 340.longest-substring-with-at-most-k-distinct-characters
- 424.longest-repeating-character-replacement
- 560.subarray-sum-equals-k (optimized with prefix-sum)

#### Hard:

- 76. minimum-window-substring
- 992. subarrays-with-k-different-integers

## Sorting

```
Time and Space complexity of all kinds of sort
```

### Quick Sort, Merge Sort, Bubble Sort, etc

- [https://leetcode.com/problems/sort-an-array/]

### Quick Select

- 215.kth-largest-element-in-an-array

### Sort & Intervals

- 56.merge-intervals
- 253.meeting-rooms-ii

### Indexing Sort

- 41.first-missing-positive(hard, need to take care of swap condition)

## Prefix-Sum

```
- 基础知识：前缀和本质上是在一个list当中，用O（N）的时间提前算好从第0个数字到第i个数字之和，在后续使用中可以在O（1）时间内计算出第i到第j个数字
- one pattern of prefix sum is that it utilizes subarray information
```

- 560.subarray-sum-equals-k(use hash map to store prefix sum)
- 523.continuous-subarray-sum (a variation of 560, using modulo math)
- 315.count-of-smaller-numbers-after-self(hard, use prefix sum to record count)
- 238.product-of-array-except-self(prefix product and postfix product)
- 1423.maximum-points-you-can-obtain-from-cards(prefix sum + two pointers)

## BFS

基础知识：
- 常见的BFS用来解决什么问题？(1) 简单图（有向无向皆可）的最短路径长度，注意是长度而不是具体的路径（2）拓扑排序 （3） 遍历一个图（或者树）
- BFS基本模板（需要记录层数或者不需要记录层数）
- 多数情况下时间复杂度空间复杂度都是O（N+M），N为节点个数，M为边的个数

### 基于图的BFS划分connected component：（一般需要一个set来记录访问过的节点）

- **690. employee-importance**
- **200. number-of-islands**
- **130. surrounded-regions**
- 1319. number-of-operations-to-make-network-connected
- **547. number-of-provinces**
- **785. is-graph-bipartite**
- 721.accounts-merge
- 133.clone-graph
- 827. making-a-large-island(hard; find all island areas first, then iterate each water cell)

### 基于BFS寻找最短路径

- **994. rotting-oranges**
- 752. open-the-lock
- **1197. minimum-knight-moves**
- 529. minesweeper
- **490. the-maze**(start and destination)
- 815. bus-routes(hard)
- 127. word-ladder(hard; the key is how to limit the search space)
- 1293. shortest-path-in-a-grid-with-obstacles-elimination

### Topological Sort

- **207. course-schedule**
- 210. course-schedule-ii
- **310.minimum-height-trees**
- 269. alien-dictionary(hard; use indegree to represent order, need to scan each word to define relative order)

## DFS-Backtracking

### 基于图的DFS/Backtracking:

- 和BFS一样一般需要一个set来记录访问过的节点，避免重复访问造成死循环；
- Word XXX 系列面试中非常常见，例如word break，word ladder，word pattern，word search。
- Backtrack基本逻辑：

```
Define base case
For each possible direction, check valid
append
next level traverse
pop
```

- 22. generate-parentheses
- 93. restore-ip-addresses
- 79. word-search

#### Hard

- **51. n-queens**
- **37. sudoku-solver**
- **word-pattern-ii**
- **remove-invalid-parentheses**
- **212. word-search-ii**
- 126. word-ladder-ii

- 1659. maximize-grid-happiness

### 基于排列组合的DFS

其实与图类DFS方法一致，但是排列组合的特征更明显
- 去重：sort，在每一层recursion检查当前num是否和之前一样，且至少为当前层第二位num

e.g. [1, 2, 2] -> [1] [1, 2] instead of [1] [1, 2] [1, 2, 2]

- **46. permutations**
- **78. subsets**
- **17. letter-combinations-of-a-phone-number**
- 90. subsets-ii
- **39. combination-sum**
- **77. combinations**
- **40. combination-sum-ii**
- 31. next-permutation
- 47. permutations-ii
- 842. split-array-into-fibonacci-sequence

### 记忆化搜索（DFS + Memoization Search)

算是动态规划的一种，递归每次返回时同时记录下已访问过的节点特征，避免重复访问同一个节点，可以有效的把指数级别的DFS时间复杂度降为多项式级别；
注意这一类的DFS必须在最后有返回值，不可以用排列组合类型的DFS方法写；for循环的dp题目都可以用记忆化搜索的方式写，但是不是所有的记忆化搜索题目都可以
- 当状态转移的拓扑顺序不明显或者边界情况比较难处理时，建议采用 记忆化搜索，也就是 DFS + Memo。
- 如果转移的拓扑顺序非常明显，建议采用 递推 的方式，因为这样可以加快运行速度，且不容易出现栈溢出等问题。

- 509.fibonacci-number - 139.word-break - 276.paint-fence - 1048.longest-string-chain DFS involving expression, such as 679.24 Game - 241.different-ways-to-add-parentheses

#### Hard

- 140.word-break-ii
- 329.longest-increasing-path-in-a-matrix
- 44.wildcard-matching
- 403.frog-jump(hard; not only need to record state at certain step, but also k at that step)

## Math & Greedy

- **204. count-primes**
- **628. maximum-product-of-three-numbers**
- 976. largest-perimeter-triangle
- **202. happy-number**
- **1232. check-if-it-is-a-straight-line**
- 29. divide-two-integers
- **343. integer-break**
- 166. fraction-to-recurring-decimal
- 31. next-permutation(find the math pattern of permutation)
-

#### Hard

- 149. max-points-on-a-line

### Randomized

- 528. random-pick-with-weight
- **470. implement-rand10-using-rand7**

### Geometry

- **1232. check-if-it-is-a-straight-line**
- 1266. minimum-time-visiting-all-points
- 892. surface-area-of-3d-shapes
- 1401. circle-and-rectangle-overlapping
- 963. minimum-area-rectangle-ii II

#### Hard

- 587. erect-the-fence
- 1515. best-position-for-a-service-centre

### Linear Algebra

- 311.sparse-matrix-multiplication

## Union-Find

基础知识：如果数据不是实时变化，本类问题可以用BFS或者DFS的方式遍历，
如果数据实时变化（data stream）则并查集每次的时间复杂度可以视为O（1）；需要牢记合并与查找两个操作的模板

- 200. number-of-islands
- 721. accounts-merge
- 547. number-of-provinces
- 1631. path-with-minimum-effort
- 399. evaluate-division

Hard

- 128. longest-consecutive-sequence
- 765. couples-holding-hands

## Dynamic-Programming

## 坐标 (Coordinate)

- **70.climbing-stairs**(state: total ways at each step)
- **53.maximum-subarray**(state: largest current subarray sum)
- **121. best-time-to-buy-and-sell-stock**(state1: minimum_cost state2: maximum_profit)
- **746.min-cost-climbing-stairs**
- **279. perfect-squares**(state: minimum ways to reach current num; transition: for each possible square number, calculate ways)
- **198. house-robber**
- **300. longest-increasing-subsequence**
- 368. largest-divisible-subset
- 152. maximum-product-subarray

Hard

- 354. russian-doll-envelopes
- 32. longest-valid-parentheses
- 123. best-time-to-buy-and-sell-stock-iii 2D board state transition
- 62.unique-paths
- **120. Triangle**(state: minimum cost at current step)
- 221.maximal-square(a transition of square matirx in a 2D board: dp[i][j] = min(dp[i][j - 1], dp[i - 1][j]), dp[i - 1][j - 1]) + 1)
- 1277.count-square-submatrices-with-all-ones
- 85.maximal-rectangle(hard; a transition of rectangle in a 2D board)

## 前缀 - 匹配 (Matching)

dp[i][j]表示第一个字符串的前i个字符与第二个字符串的前j个字符的状态

- 72. edit-distance(hard; need to take care of edge cases)
- **1143. longest-common-subsequence**
- 44. wildcard-matching (hard; need to take care of * situation)
- 10. regular-expression-matching (hard; need to take care of * situation)

## 前缀 - 划分 (partition)

指定划分部分：dp[i][j]表示前i个字符划分为j个部分的最优值
未指定划分部分： dp[i]表示前i个字符划分为若干个部分的最优值

- **139. word-break**
- **91. decode-ways**

## 区间 (devide-and-conquer)

- 大的subarray/substring依赖于小的subarray/substring
- dp[i][j] = max/min/sum/or(dp[i][j]之内更小的若干区间)
- iteration可以从possible length开始（for length in range(shortest_length, longest_length))

- 312. burst-ballons(hard; dp[i][j] represents maximum score players can get between i and j)
- **5. longest-palindromic-substring**
- 1000. minimum-cost-to-merge-stones

## 背包 (Knapsack)

```
- Unbounded Knapsack usually just need a 1-D array to track state because candadates could be used unlimited times
- 0-1 Knapsack usually need a 2-D array with dp[i][j] represents at item i, min/max/sum value you could get with j as the value
    - e.g. dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - A[i - 1]] + V[i - 1])
```

#### Unbounded

- rod-cutting-problem
- **322. coin-change**
- 518. coin-change2
- **983. minimum-cost-for-tickets**

#### 0-1

- **knapsack**
- kacpsack-with-value
- **494.target-sum**
- **416. partition-equal-subset-sum**
- 474. ones-and-zeroes
- 1049. last-stone-weight-ii