Kimi's LeetcodeGolang Notes

Leetcode in Golang

Kimi Tsai

Kimi Tsai

Table of Contents

1. Home 9
   - 1.1 Leetcode in Golang 9
   - 1.2 leetcode Content 9
   - 1.3 GeeksforGeeks Content 20
   - 1.4 Codility Content 21
   - 1.5 Reference 23
2. tags 24
   - 2.1 /Medium 24
   - 2.2 143. Reorder List 24
   - 2.3 33. Search in Rotated Sorted Array 24
   - 2.4 3Sum 24
   - 2.5 A1B2C3 24
   - 2.6 Add Two Numbers 24
   - 2.7 Adobe 24
   - 2.8 Algorithm 25
   - 2.9 Algorithms 25
   - 2.10 Amazon 25
   - 2.11 Apple 25
   - 2.12 Array 26
   - 2.13 Array & String 26
   - 2.14 BFS 26
   - 2.15 Backtracking 26
   - 2.16 Balanced Binary Tree 27
   - 2.17 Bank Requests 27
   - 2.18 Best Time to Buy and Sell Stock 27
   - 2.19 Binary Search 27
   - 2.20 Binary Tree 27
   - 2.21 Bit Manipulation 27
   - 2.22 Bitwise Manipulation 28
   - 2.23 Blind75 28
   - 2.24 Bloomberg 28
   - 2.25 Breadth First Search 28
   - 2.26 Climbing Stairs 28
   - 2.27 CodeSignal 28
   - 2.28 Codility 29

2.29 Coin Change 29
2.30 Concurrency 29
2.31 Construct Binary Tree from Preorder and Inorder Traversal 29
2.32 Container With Most Water 29
2.33 Contains Duplicate 29
2.34 Copy List with Random Pointer 29
2.35 Counting 29
2.36 DFS 30
2.37 DP 30
2.38 Design 30
2.39 Diameter of Binary Tree 30
2.40 Divide and Conquer 30
2.41 Dynamic Programming 30
2.42 Easy 31
2.43 Easy/Medium/Hard 31
2.44 Edit Distance 32
2.45 Facebook 32
2.46 Find First and Last Position of Element in Sorted Array 32
2.47 Find Target Last Index 32
2.48 Find the Index of the First Occurrence in a String 32
2.49 Fizz Buzz 32
2.50 Fizz Buzz Multithreaded 33
2.51 Flood Fill 33
2.52 Go 33
2.53 Golang 34
2.54 Goldman Sachs 35
2.55 Google 35
2.56 Greedy 35
2.57 Group Anagrams 35
2.58 Hard 35
2.59 Hash 35
2.60 Hash Table 36
2.61 Heap 36
2.62 Insert Delete GetRandom O(1) 36
2.63 Insert Delete GetRandom O(1) Duplicates allowed 36
2.64 Intersection 36
2.65 Interview 36
2.66 Invert Binary Tree 37

2.67 Iterations 37
2.68 Koko Eating Bananas 37
2.69 Kth Largest Element in a Stream 37
2.70 Kth Largest Element in an Array 37
2.71 LEETCODETITLE 37
2.72 Last Stone Weight 37
2.73 LeetCode 37
2.74 Left Bound 39
2.75 Linked List 39
2.76 LinkedIn 39
2.77 Longest Consecutive Sequence 39
2.78 Longest Substring Without Repeating Characters 39
2.79 Math 40
2.80 Matrix 40
2.81 Max Area of Island 40
2.82 Medium 24
2.83 Merge Two Sorted Lists 41
2.84 Microsoft 41
2.85 Min Cost Climbing Stairs 42
2.86 Move Zeroes 42
2.87 Multiple Pointers 42
2.88 Painless 42
2.89 Patience Sorting 42
2.90 Prefix Sum 42
2.91 Priority Queue 42
2.92 Product of Array Except Self 43
2.93 Recursion 43
2.94 Reverse Linked List 43
2.95 Right Bound 43
2.96 Same Tree 43
2.97 Search Graph 43
2.98 Search a 2D Matrix 43
2.99 Slide Windows 44
2.100 Sliding Window 44
2.101 Sort 44
2.102 Sorting 44
2.103 Spotify 44
2.104 Stack 44

2.105 String 44
2.106 String Matching 45
2.107 Subsets 45
2.108 Time Based Key-Value Store 45
2.109 Top K Frequent Elements 45
2.110 Tree 45
2.111 Two Pointers 45
2.112 Two Sum II Input Array Is Sorted 46
2.113 Union Find 46
2.114 Valid Anagram 46
2.115 Valid Palindrome 46
2.116 Valid Parentheses 46
2.117 WeightedEditDistance 46
2.118 amazon 25
2.119 array 26
2.120 bloomberg 28
2.121 container/heap 47
2.122 facebook 32
2.123 heap 36
2.124 math 40
2.125 microsoft 41
2.126 string 44
2.127 uber 48
3. Algorithms 49
3.1 A1B2C3 49
3.2 Find Target Last Index 51
3.3 Intersection of Two Sorted Arrays using In Place Approach 53
3.4 SearchGraph 54
3.5 WeightedEditDistance 56
4. CodeSignal 58
4.1 Code Signal 58
4.2 Bank Requests 59
5. Codility 63
5.1 Lesson 63
6. GeeksforGeeks 128
6.1 SortingAlgorithms 128
7. Leetcode 130
7.1 0000.xxxx 130

7.2 0001.Two Sum 131
7.3 0002.Add Two Numbers 136
7.4 0003.Longest Substring Without Repeating Characters 141
7.5 0005.Longest Palindromic Substring 144
7.6 0010.Regular Expression Matching 147
7.7 0011.Container With Most Water 148
7.8 0015.3Sum 150
7.9 0019.Remove Nth Node From End of List 153
7.10 0020.Valid Parentheses 155
7.11 0021.Merge Two Sorted Lists 157
7.12 0027.Remove Element 158
7.13 0028.Find the Index of the First Occurrence in a String 160
7.14 0033.Search in Rotated Sorted Array 162
7.15 0034.Find First and Last Position of Element in Sorted Array 165
7.16 0035.Search Insert Position 167
7.17 0046.Permutations 169
7.18 0049.Group Anagrams 171
7.19 0053.Maximum Subarray 174
7.20 0059.Spiral Matrix II 176
7.21 0070.Climbing Stairs 178
7.22 0072.Edit Distance 180
7.23 0074.Search a 2D Matrix 183
7.24 0075.Sort Colors 185
7.25 0078.Subsets 188
7.26 0088.Merge Sorted Array 190
7.27 0094.Binary Tree Inorder Traversal 192
7.28 0100.Same Tree 194
7.29 0104.Maximum Depth of Binary Tree 196
7.30 0105.Construct Binary Tree from Preorder and Inorder Traversal 198
7.31 0110.Balanced Binary Tree 200
7.32 0121.Best Time to Buy and Sell Stock 203
7.33 0125.Valid Palindrome 205
7.34 0128.Longest Consecutive Sequence 207
7.35 0138.Copy List with Random Pointer 208
7.36 0141.Linked List Cycle 210
7.37 0142.Linked List CycleII 211
7.38 0143.Reorder List 214
7.39 0153.Find Minimum in Rotated Sorted Array 216

7.40 0167.Two Sum II Input Array Is Sorted 217

7.41 0203.Remove Linked List Elements 219

7.42 0206.Reverse Linked List 221

7.43 0209.Minimum Size Subarray Sum 223

7.44 0215.Kth Largest Element in an Array 226

7.45 0217.Contains Duplicate 228

7.46 0226.Invert Binary Tree 229

7.47 0238.Product of Array Except Self 232

7.48 0242.Valid Anagram 234

7.49 0283.Move Zeroes 236

7.50 0300.Longest Increasing Subsequence 238

7.51 0310.Minimum Height Trees 241

7.52 0322.Coin Change 244

7.53 0344.Reverse String 247

7.54 0347.Top K Frequent Elements 248

7.55 0354.Russian Doll Envelopes 251

7.56 0380.Insert Delete GetRandom O1 254

7.57 0381.Insert Delete GetRandom O1 Duplicates allowed 256

7.58 0409.Longest Palindrome 258

7.59 0412.Fizz Buzz 260

7.60 0438.Find All Anagrams in a String 262

7.61 0509.Fibonacci Number 264

7.62 0516.Longest Palindromic Subsequence 267

7.63 0543.Diameter of Binary Tree 268

7.64 0567.Permutation in String 270

7.65 0693.Binary Number with Alternating Bits 272

7.66 0695.Max Area of Island 274

7.67 0703.Kth Largest Element in a Stream 277

7.68 0704.Binary Search 279

7.69 0721.Accounts Merge 282

7.70 0733.Flood Fill 285

7.71 0746.Min Cost Climbing Stairs 288

7.72 0752.Open the Lock 290

7.73 0875.Koko Eating Bananas 294

7.74 0876.Middle of the Linked List 296

7.75 0981.Time Based Key Value Store 298

7.76 1046.Last Stone Weight 300

7.77 1143.Longest Common Subsequence 302

7.78 1195.Fizz Buzz Multithreaded 305
7.79 3223.Minimum Length of String After Operations 308
7.80 3224.Minimum Array Changes to Make Differences Equal 309

1. Home

1.1 Leetcode in Golang

Release no status | Build and Deploy MkDocs failing | pages-build-deployment passing

Leetcode, Codility , GeekforGeeks algorithms exercises written in Golang.

./

1.2 Content

- Leetcode in Golang
- leetcode Content
- Data Structure
- Array \& String
- Matrix
- Linked List
- HashSet \& HashMap
- Stack \& Queue
- Heap \& Priority Queue
- Heap □□□□□□
- heap sort
- Disjoint Set Union
- Trie
- Binary Indexed Tree
- Design Data Structures
- Algorithm
- Greedy
- Sort
- Multiple Pointers
- Backtracking (□□□). DFS
- DFS \& BFS
- Dynamic Programming
- Sliding Window
- Sweep Line
- Rolling Sum
- Two Pointers
- Bit Manipulation∂
- Union Find
- Breadth First Search
- Binary Search
- Minimax

- Graph
- Graph
- Topological Sort
- Tree
- Tree Traversal
- Binary Search Tree
- Compputational Geometry
- Selected Topics
- Mathematics
- Random
- Bitwise Manipulation
- GeeksforGeeks Content
- Codility Content
- Reference

1.2.1 Data Structure

Array & String

No.	Title	Solution	Difficulty	Time	Space	Topic
0001	Two Sum	Go	Easy	O(n)	O(n)	Array
0003	Longest Substring Without Repeating Characters	Go	Medium	O(n)	O(1)	Array, Windo
0015	3 Sum	Go	Medium	O(n^2)	O(n)	Array
0027	Remove Element	Go	Easy	O(n)	O(1)	Array
0035	Search Insert Position	Go	Easy	O(n), O(logn)	O(1)	Array
0049	Search Insert Position	Go	Medium	O(kn)	O(kn)	Array
0059	Spiral Matrix II	Go	Medium	O(n)	O(n^2)	Array
0088	Merge Sorted Array	Go	Easy	O(n)	O(1)	Array
0217	0217.Contains Duplicate	Go	Easy	O(n)	O(n)	Array
0242	0242.Valid Anagram	Go	Easy	O(n)	O(n)	Array
0409	409. Longest Palindrome	Go	Easy	O(n)	O(1)	Array
0380	0380.Insert Delete GetRandom O(1)	Go	Medium	O(1)	O(n)	Array
0381	0381.Insert Delete GetRandom O(1) Duplicates allowed	Go	Medium	O(1)	O(n)	Array
0412	0412.Fizz Buzz	Go	Easy	O(n)	O(n)	Array,
1195	1195.Fizz Buzz Multithreaded	Go	Medium	O(n)		Array, string
0238	238. Product of Array Except Self	Go	Medium	O(n)		Array, Prefix
0128	128. Longest Consecutive Sequence	Go	Medium	O(n)	O(n)	Array, Union

Matrix

Linked List

No.	Title	Solution	Difficulty	Time	Space	Topic
0019	Remove Nth Node From End of List	Go	Medium	O(n)	O(1)	Linke Two P
0141	Linked List Cycle	Go	Easy	O(n)	O(1)	Linke Two P
0142	Linked List Cycle II	Go	Medium	O(n)	O(1)	Linke Two P
0203	Remove Linked List Elements	Go	Easy	O(n)	O(1)	Linke
0206	Reverse Linked List	Go	Easy	O(n)	O(1)	Linke
0876	Middle of the Linked List	Go	Easy			Linke Two P
0021	Merge Two Sorted Lists	Go	Easy	O(log n)	O(1)	Linke
0002	Add Two Number	Go	Medium	O(max(m,n))	O(1)	Linke

HashSet & HashMap

Stack & Queue

No.	Title	Solution	Difficulty	Time	Space	Topic
0020	Valid Parentheses	Go	Easy	O(n)	O(n)	Stack
0094	Binary Tree Inorder Traversal	Go	Medium	O(n)	O(1)	Stack

Heap & Priority Queue

Heap □□□□□□□□□□□□□□□□□□root□□□

HEAP □□□□□

- □□□□□□ binary tree □□
- □□ balanced
- max heap □ root □□□□□min heap □ root □□□□□
- □□□ tree□□□□□□□ array □□□

HEAP SORT

□□□□ heap □ root □□□□□(□□□ max heap)□□□□□□□□□□□□ heapify □□ n □ root deletion □□□□□□□□□□□□□□□□□ □□□□□ heap sort □ worst case □□□□□ O(nlogn) □□□□ □□ quick sort □ worst case □□□□□□ O(n^2)□□□ quick sort □□□□□□□□□□□□□□□□□□ google □□□□□ heap sort □□□□□ caching □□ spatial locality □□□□□□□□□□ * https://www.zhihu.com/question/23873747 * https://rust-algo.club/sorting/heapsort/index.html

No.	Title	Solution	Difficulty	Time	Space	Topic
0703	Kth Largest Element in a Stream	Go	Easy	O(K + (N-K)logK)	O(k)	Heap, Queue
1046	Last Stone Weight	Go	Easy	O(nlogn)	O(n)	Heap, Queue
0347	Top K Frequent Elements	Go	Medium	O(Nlogk)	O(n)	Heap, Queue Sort

Disjoint Set Union

Trie

Binary Indexed Tree

Design Data Structures

1.2.2 Algorithm

Greedy

Sort

No.	Title	Solution	Difficulty	Time	Space	Topic
0075	Sort Colors	Go	Medium	O(n)	O(1)	Sort
0215	Kth Largest Element in an Array	Go	Medium	O(n)	O(logn)	Sort

Multiple Pointers

Backtracking (□□□). DFS

DFS. □□□□□□□□, □□□□□□□□□□□□□□□. □□□□□□□□□□□□ 1. □□□□□□□□□□□□□□□□ 2. □□□□□□□□□□□□□□□□□□□□ 3. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ 4. https://www.bilibili.com/video/BV1P5411N7Xc

```
result = []
def backtrack(□□, □□□□):
    if □□□□□□:
        result.add(□□)
        return

    for □□ in □□□□:
        □□□(□□)
```

```
        backtrack(□□, □□□□)
        □□□□(□□)
```

No.	Title	Solution	Difficulty	Time	Space	Topic
0046	Permutations (□□□)	Go	Medium	O(n)	O(n)	Backt
0078	Subsets	Go	Medium	O(n^2)	O(n)	Backt

DFS & BFS

□□□□□□**BFS**, □□□□**DFS**□□□□□, □□□□□□□

□□□□□□□□□□,□□□□ N. □DFS□□□□□□□□□□□□, □□□□□□□□□□□□ O(log N) BFS□□, Queue□□□□□□□□□□□□□□, □□□□□□□□□□□□□□□□□□□□□□□□□□□□, □□□ N/2. □□□□□ O(N)

DFS□□□□□□□□□□□□□□□Stack□□□□□□DFS□□□□□□□□□□□□□□□□□□□

```
        for ...
            dp[□□1][□□2][...] = □□□(□□1□□□2...)
```

No.	Title	Solution	Difficulty	Time	Space	Topic
0053	Maximum Subarray	Go	Easy	O(n)	O(n)	Dynam Progra
0072	0072. Edit Distance	Go	Hard			Dynam Progra
0300	Longest-Increasing-Subsequence	Go	Medium	□□□:O(n^2) □□□:O(nlogn)	O(n)	Dynam Progra
0322	Coin Change	Go	Medium	O(nm)	O(n)	Dynam Progra
0354	Russian Doll Envelope	Go	Hard			Dynam Progra
0509	Fibonacci Number	Go	Easy	□□□□	□□□□	Dynam Progra
0070	0070.Climbing Stairs	Go	Easy	O(n)	O(n)	Dynam Progra
0746	0746.Min Cost Climbing Stairs	Go	Easy	O(n)	O(1)	Dynam Progra

---

Sliding Window

□□□□□□, □□□□

```
void slidingWindow(string s, string t){
    unordered map<char,int>need, window;
    for (char c:t) need[c++]
    int left = 0 , right = 0
    int valid = 0

    // □□□ right □□□ left. □□right□□ string□□□
    while(right < s.size()){
        // C□□□□□□□□□
        char c = s[right]
        // □□□□
        right++
        // □□□□□□□□□□□□□
        // ...

        /*** □□debug □□□□ ***/
        printf("window: [%d, %d)\n",left,right)
        /************************/

        // □□□□□□□□□□
        while(window needs shrink){
           // d□□□□□□□□□
           // □□□□
            left++
            // □□□□□□□□□□□□□
            // ...
        }
    }
}
```

No.	Title	Solution	Difficulty	Time	Space	Topic
0209	Minimum Size Subarray Sum	Go	Medium	O(n^2) / O(n) / O(nlog n)	O(1) / O(1) / O(n)	Sliding Window
0438	Find All Anagrams in a String	Go	Medium	O(n)	O(1)	Sliding Window
0567	Permutation in String	Go	Medium	O(n)	O(1)	Sliding Window

Sweep Line

Rolling Sum

Two Pointers

□□**array**□□, □□□□□□□□□□ □□□□ 1. "□,□□□" 2. "□,□□□" 1. □,□□□: □□□□ linkedlist □□, □□□□□ linkedlist □□□□□ 2. □,□□□: □□□□array(□ string)□□□□, □□□□□.

https://labuladong.gitee.io/algo/2/21/57/

No.	Title	Solution	Difficulty	Time	Space	Topic
0019	Remove Nth Node From End of List	Go	Medium	O(n)	O(1)	Linke Two P
0141	Linked List Cycle	Go	Easy	O(n)	O(1)	Linke Two P
0283	Move Zeroes	Go	Easy	O(n)	O(1)	Two P
0142	Linked List Cycle II	Go	Medium	O(n)	O(1)	Linke Two P
0344	Reverse String	Go	Easy	O(n)	O(1)	Two P
0876	Middle of the Linked List	Go	Easy			Linke Two P
0011	Container With Most Water	Go	Medium			Two P
0074	Search a 2D Matrix	Go	Medium			Binary Two P

Bit Manipulation∂

No.	Title	Solution	Difficulty	Time	Space	Topic
0693	Binary Number with Alternating Bits	Go	Easy	O(n)/ O(1)	O(1) / O(1)	Bit Manip

Union Find

No.	Title	Solution	Difficulty	Time	Space	Topic
0721	Accounts Merge	Go	Easy	O(n) / O(n log n)	O(n) / O(n)	Union

Breadth First Search

- DFS □□□□□□□□□□□□, BFS□□□□□**Queue**□□□□□□, □□□□□□□□□□□□□□□□Queue.
- BFS □□□□□□□□□□□□, □□□□□□□□□□□□DFS□. BFS vs DFS
- □□: □□ BFS □□, □ while □□□□□□□□. □□□□□□□□□□□□□□□□, □□□□□□□□□□□□□□□□, □□□□□□□□□□□□□□□ curDepth □ nextDepth □□□ □□□□□ BFS □□ □□BFS, □□□□BFS □□□□□□□□□□□

```
// □□□□□ start □ □□ target □□□□□
int BFS(Node start, Node targe){
    Queue<Node> q; // □□□□□□
    Set<Node> visited; // □□□□□□

    q.offer(start); // □□□□□ Queue
    visited.add(start);
    int step = 0; // □□□□□□□

    while(q not empty) {
        int sz = q.size();
        // □□ Queue □□□□□□□□□□□
        for (int i = 0 ; i < sz; i++) {
            Node cur = q.poll();
            // □□□□□□□□□□
            if (cur is target) {
                return step;
            }

            // □cur □□□□□□□ Queue
            for (Node x : cur.adj()) {
                if (x not in visited) {
                    q.offer(x);
                    visited.add(x);
                }
            }
        }
        // □□□□□□□
        step++
    }
}
```

No.	Title	Solution	Difficulty	Time	Space	Topic
0310	Minimum Height Trees	Go	Medium			Bread Searc
0752	752. Open the Lock	Go	Medium			Bread Searc

Binary Search

□□□□□□□□□: □□□□ **else**, □□□□□□□□ **else if** □□□. □□ mid □□□□□□□

```
int binarySearch(int[] nums, int target){
    int left = 0 , right = ...;
    while(...) {
        int mid = left + (right - left)/2
        if (nums[mid] == target){
            ...
        } else if (nums[mid] < target){
            left = ...
        } else if (nums[mid] > target){
            right = ...
        }
    }
    return ...;
}
```

□□□□□□□□□□□□□□□, □□□□ `[letf,right]`

```
func Search(nums []int, target int) int {
    lenght := len(nums)
    if lenght <= 0 {
        return -1
    }
    left, right := 0, lenght-1

    for left <= right {
        mid := (right-left)/2 + left
        if nums[mid] == target {
```

```
            return mid
        } else if nums[mid] < target {
            // □□□
            left = mid + 1
        } else if nums[mid] > target {
            // □□□
            right = mid - 1
        }
    }
    // □□□□
    return -1
}

// □□□□ nums □□ target□□□□□□
func LeftBound(nums []int, target int) (index int) {
    lenght := len(nums)
    if lenght <= 0 {
        return -1
    }
    left, right := 0, lenght-1

    for left <= right {
        // □□2
        // mid := left + (right-left)>>1
        mid := int(uint(right+left) >> 1)
        if nums[mid] == target {
            // □□□□□□, □□□□□□□
            right = mid - 1
        } else if nums[mid] < target {
            // □□□
            left = mid + 1
        } else if nums[mid] > target {
            // □□□
            right = mid - 1
        }
    }
    // □□□□ □□: left□□□□
    if left >= lenght || nums[left] != target {
        return -1
    }
    return left
}

// □□□□ nums □□ target□□□□□□
func RightBound(nums []int, target int) (index int) {
    lenght := len(nums)
    if lenght <= 0 {
        return -1
    }
    left, right := 0, lenght-1

    for left <= right {
        // □□2
        // mid := left + (right-left)>>1
        mid := int(uint(right+left) >> 1)
        if nums[mid] == target {
            // □□:□□□□□□, □□□□□□□=mid+1
            left = mid + 1
        } else if nums[mid] < target {
            // □□□
            left = mid + 1
        } else if nums[mid] > target {
            // □□□
            right = mid - 1
        }
    }
    // □□□□ □□:right□□□□
    if right < 0 || nums[right] != target {
        return -1
    }
    return right
}
```

`for left <= right {}` □

- □□□□□□□□□left □ right □□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□ left □ right □□□□□□□□□□□□□□□□□□□□

`for left < right {}` □

- □□□□□□□□□left □ right □□□□□□□□
- □□□□□□□□□

Tree

No.	Title	Solution	Difficulty	Time	Space	Topic
0226	Invert Binary Tree	Go	Easy	O(n)	O(1)	Tree
0104	Maximum Depth of Binary Tree	Go	Easy	O(n)	O(1)	Tree
0543	Diameter of Binary Tree	Go	Easy	O(n)	O(n), O(log(n))	Tree,
0110	Balanced Binary Tree	Go	Easy	O(n)	O(1)	Tree,
0100	Same Tree	Go	Easy	O(n)	O(1)	Tree
0105	Construct Binary Tree from Preorder and Inorder Traversal	Go	Medium	O(n)	O(n)	Array

Tree Traversal

Binary Search Tree

Compputational Geometry

1.2.4 Selected Topics

Mathematics

Random

Bitwise Manipulation

1.3 Content

Topic	Title	No.	Solution	Difficulty	TimeComplexity	Spa
Sorting	Find Minimum Difference Between Any Two Elements	0031	Go	Basic	O(n^2), O(n log n)	

1.4 Content

Topic		Title	Solution	Difficulty	TimeComp
Lesson 1	Iterations	Binary Gap	Go	Painless	O(log n)
Lesson 2	Array	Cyclic Rotation	Go	Painless	O(1)
		Odd Occurrences In Array	Go	Painless	O(n), O(n)
Lesson 3	Time Complexity	Frog Jmp	Go	Painless	O(1)
		Perm Missing Elem	Go	Painless	O(n)
		Tape Equilibrium	Go	Painless	O(n)
Lesson 4	Counting Elements	Frog River One	Go	Painless	O(n)
		Max Counters	Go	Respectable	O(n+m)
		Missing Integer	Go	Respectable	O(n)
		Perm Check	Go	Painless	O(n)
Lesson 5	Prefix Sums	Count Div	Go	Respectable	O(1)
		Genomic Range Query	Go	Respectable	O(n+m)
		MinAvg Two Slice	Go	Respectable	O(n)
		Passing Cars	Go	Painless	O(n)
Lesson 6	Sorting	Distinct	Go	Painless	O(nlogn)
		Max Product of Three	Go	Painless	O(nlogn)
		Number Of Disc Intersections	Go	Respectable	O(nlogn)
		Triangle	Go	Painless	O(nlogn)
Lesson 7	Stacks and Queues	Brackets	Go	Painless	O(n)
		Fish	Go	Painless	O(n)
		Nesting	Go	Painless	O(n)
		Stone Wall	Go	Painless	O(n)
Lesson 8	Leader	Dominator	Go	Painless	O(n)
		EquiLeader	Go	Painless	O(n)
Lesson 9	Maximum slice problem	Max Profit	Go	Painless	O(n)
		Max Slice Sum	Go	Painless	O(n)
		Max Double Slice Sum	Go	Respectable	O(n)
Lesson 10	Prime and composite numbers	Count Factors	Go	Painless	O(sqrt(n))
		Flags	Go	Respectable	O(n)
		MinPerimeterRectangle	Go	Painless	O(sqrt(n))
		Peaks	Go	Respectable	O(n*log(
Lesson 11	Sieve of Eratosthenes (□□□)	Count Non Divisible	Go	Respectable	O(N * log(
		Count Semiprimes	Go	Respectable	O(N*log(lo +M)
Lesson 12	Euclidean algorithm (□□□□□ or □□□□ □□)	Chocolates By Numbers	Go	Painless	O(log(N +
		Common Prime Divisors	Go	Respectable	O(Z * log(+ max(B))

Lesson 13	**Fibonacci numbers**	**FibFrog**	**Go**	**Respectable**	**O(N * log(**
		Ladder		**Respectable**	
Lesson 14	**Binary search algorithm**	**MinMaxDivision**		**Respectable**	
		NailingPlanks		**Respectable**	
Lesson 15	**Caterpillar method**	**AbsDistinct**		**Painless**	
		CountDistinctSlices		**Painless**	
		CountTriangles		**Painless**	
		MinAbsSumOfTwo		**Respectable**	
Lesson 16	**Greedy algorithms**	**MaxNonoverlappingSegments**		**Painless**	
		TieRopes		**Painless**	
Lesson 17	**Dynamic programming**	**MinAbsSum**		**Ambitious**	
		NumberSolitaire		**Respectable**	

1.5 Reference

- leetcode
- leetcode-cn
- halfrost
- anakin
- wufenggirl
- GeeksforGeeks
- Codility
- GitHub: labuladong/fucking-algorithm(labuladong □□□□)
- https://cses.fi/
- https://github.com/neetcode-gh/leetcode

2. tags

□□□□□□□□□□□□□□

2.1

/Medium

- 143. Reorder List

2.2

143. Reorder List

- 143. Reorder List

2.3

33. Search in Rotated Sorted Array

- 33. Search in Rotated Sorted Array

2.4

3Sum

- 0015. 3Sum

2.5

A1B2C3

- A1B2C3: Two Go Routine Print A1B2C3....Z26

2.6

Add Two Numbers

- 0002.Add Two Numbers
- UMPIRE 2.Add-Two-Numbers

2.7

Adobe

- 0005. Longest Palindromic Substring
- 0011.Container With Most Water

- 0153.Find-Minimum-in-Rotated-Sorted-Array
- 128. Longest Consecutive Sequence
- 143. Reorder List

2.8

Algorithm

- A1B2C3: Two Go Routine Print A1B2C3....Z26

2.9

Algorithms

- Find Target Last Index
- Intersection of Two Sorted Arrays using In Place Approach
- Search Graph in Golang

2.10

Amazon

- 0002.Add Two Numbers
- 0005. Longest Palindromic Substring
- 0011.Container With Most Water
- 0019. Remove Nth Node From End of List
- 0153.Find-Minimum-in-Rotated-Sorted-Array
- 128. Longest Consecutive Sequence
- 138. Copy List with Random Pointer
- 143. Reorder List
- 33. Search in Rotated Sorted Array
- 74. Search a 2D Matrix
- UMPIRE 2.Add-Two-Numbers

2.11

Apple

- 0002.Add Two Numbers
- 33. Search in Rotated Sorted Array
- 412. Fizz Buzz
- UMPIRE 2.Add-Two-Numbers

2.12

Array

- 0003.Longest Substring Without Repeating Characters
- 0011.Container With Most Water
- 0015. 3Sum
- 0027.Remove Element
- 0034. Find First and Last Position of Element in Sorted Array
- 0035.Search Insert Position
- 0059.Spiral Matrix II
- 0088.Merge Sorted Array
- 0167.Two Sum II Input Array Is Sorted
- 0217.Contains-Duplicate
- 105. Construct Binary Tree from Preorder and Inorder Traversal
- 128. Longest Consecutive Sequence
- 238. Product of Array Except Self
- 3224.Minimum-Array-Changes-to-Make-Differences-Equal
- 33. Search in Rotated Sorted Array
- 380. Insert Delete GetRandom O(1)
- 381.Insert Delete GetRandom O(1) Duplicates allowed
- 74. Search a 2D Matrix
- 875. Koko Eating Bananas
- Cyclic Rotation
- UMPIRE 1.Two-Sum

2.13

Array & String

- 0695.Max Area of Island

2.14

BFS

- 0695.Max Area of Island
- 0733.Flood Fill
- 226. Invert Binary Tree
- Search Graph in Golang

2.15

Backtracking

- 0046.Permutations

2.16

Balanced Binary Tree

- 110. Balanced Binary Tree

2.17

Bank Requests

- Bank Requests

2.18

Best Time to Buy and Sell Stock

- 0121.Best Time to Buy and Sell Stock

2.19

Binary Search

- 0034. Find First and Last Position of Element in Sorted Array
- 0167.Two Sum II Input Array Is Sorted
- 0300.Longest Increasing Subsequence
- 0354. Russian Doll Envelope
- 0704.Binary Search
- 0981.Time Based Key-Value Store
- 33. Search in Rotated Sorted Array
- 74. Search a 2D Matrix
- 875. Koko Eating Bananas

2.20

Binary Tree

- 105. Construct Binary Tree from Preorder and Inorder Traversal

2.21

Bit Manipulation

- 0693.Binary Number with Alternating Bits

2.22

Bitwise Manipulation

- Binary Gap

2.23

Blind75

- 0121.Best Time to Buy and Sell Stock
- 0217.Contains-Duplicate
- 238. Product of Array Except Self

2.24

Bloomberg

- 0002.Add Two Numbers
- 0019. Remove Nth Node From End of List
- 138. Copy List with Random Pointer
- 143. Reorder List
- 74. Search a 2D Matrix
- UMPIRE 2.Add-Two-Numbers

2.25

Breadth First Search

- 0310.Minimum Height Trees
- 0752.Open the Lock

2.26

Climbing Stairs

- 0070.Climbing Stairs

2.27

CodeSignal

- Bank Requests

2.28

Codility

- Binary Gap
- Cyclic Rotation

2.29

Coin Change

- 0322.Coin Change

2.30

Concurrency

- 1195. Fizz Buzz Multithreaded

2.31

Construct Binary Tree from Preorder and Inorder Traversal

- 105. Construct Binary Tree from Preorder and Inorder Traversal

2.32

Container With Most Water

- 0011.Container With Most Water

2.33

Contains Duplicate

- 0217.Contains-Duplicate

2.34

Copy List with Random Pointer

- 138. Copy List with Random Pointer

2.35

Counting

- 3223.Minimum-Length-of-String-After-Operations

2.36

DFS

- 0543. Diameter of Binary Tree
- 0695.Max Area of Island
- 0733.Flood Fill
- 110. Balanced Binary Tree

2.37

DP

- 0005. Longest Palindromic Substring
- 0070.Climbing Stairs
- 0121.Best Time to Buy and Sell Stock

2.38

Design

- 0981.Time Based Key-Value Store

2.39

Diameter of Binary Tree

- 0543. Diameter of Binary Tree

2.40

Divide and Conquer

- 105. Construct Binary Tree from Preorder and Inorder Traversal

2.41

Dynamic Programming

- 0010.Regular-Expression-Matching
- 0072. Edit Distance
- 0300.Longest Increasing Subsequence
- 0322.Coin Change
- 0354. Russian Doll Envelope
- 0509.Fibonacci Number
- 1143.Longest Common Subsequence
- 516. Longest Palindromic Subsequence
- WeightedEditDistance

2.42

Easy

- 0001.Two Sum
- 0020. Valid Parentheses
- 0027.Remove Element
- 0028. Find the Index of the First Occurrence in a String
- 0035.Search Insert Position
- 0070.Climbing Stairs
- 0088.Merge Sorted Array
- 0104.Maximum Depth of Binary Tree
- 0121.Best Time to Buy and Sell Stock
- 0125. Valid Palindrome
- 0141.Linked List Cycle
- 0203.Remove Linked List Elements
- 0217.Contains-Duplicate
- 0242.Valid-Anagram
- 0283. Move Zeroes
- 0344.Reverse String
- 0409. Longest Palindrome
- 0509.Fibonacci Number
- 0543. Diameter of Binary Tree
- 0703. Kth Largest Element in a Stream
- 0704.Binary Search
- 0721.Accounts Merge
- 0733.Flood Fill
- 0876.Middle of the Linked List
- 100. Same Tree
- 1046. Last Stone Weight
- 110. Balanced Binary Tree
- 206. Reverse Linked List
- 226. Invert Binary Tree
- 412. Fizz Buzz
- 746. Min Cost Climbing Stairs
- Find Target Last Index
- UMPIRE 1.Two-Sum

2.43

Easy/Medium/Hard

- 0021. Merge Two Sorted Lists
- NUM.LEETCODETITLE

2.44

Edit Distance

- 0072. Edit Distance

2.45

Facebook

- 0002.Add Two Numbers
- 0005. Longest Palindromic Substring
- 0011.Container With Most Water
- 0019. Remove Nth Node From End of List
- 0153.Find-Minimum-in-Rotated-Sorted-Array
- 138. Copy List with Random Pointer
- 143. Reorder List
- 33. Search in Rotated Sorted Array
- 412. Fizz Buzz
- 74. Search a 2D Matrix
- UMPIRE 2.Add-Two-Numbers

2.46

Find First and Last Position of Element in Sorted Array

- 0034. Find First and Last Position of Element in Sorted Array

2.47

Find Target Last Index

- Find Target Last Index

2.48

Find the Index of the First Occurrence in a String

- 0028. Find the Index of the First Occurrence in a String

2.49

Fizz Buzz

- 412. Fizz Buzz

2.50

Fizz Buzz Multithreaded

- 1195. Fizz Buzz Multithreaded

2.51

Flood Fill

- 0733.Flood Fill

2.52

Go

- 0002.Add Two Numbers
- 0003.Longest Substring Without Repeating Characters
- 0005. Longest Palindromic Substring
- 0010.Regular-Expression-Matching
- 0011.Container With Most Water
- 0015. 3Sum
- 0019. Remove Nth Node From End of List
- 0020. Valid Parentheses
- 0021. Merge Two Sorted Lists
- 0028. Find the Index of the First Occurrence in a String
- 0034. Find First and Last Position of Element in Sorted Array
- 0049.Group Anagrams
- 0070.Climbing Stairs
- 0078. Subsets
- 0104.Maximum Depth of Binary Tree
- 0121.Best Time to Buy and Sell Stock
- 0125. Valid Palindrome
- 0153.Find-Minimum-in-Rotated-Sorted-Array
- 0167.Two Sum II Input Array Is Sorted
- 0215. Kth Largest Element in an Array
- 0217.Contains-Duplicate
- 0242.Valid-Anagram
- 0283. Move Zeroes
- 0322.Coin Change
- 0543. Diameter of Binary Tree
- 0695.Max Area of Island
- 0703. Kth Largest Element in a Stream
- 0733.Flood Fill
- 0981.Time Based Key-Value Store

- 100. Same Tree
- 1046. Last Stone Weight
- 105. Construct Binary Tree from Preorder and Inorder Traversal
- 110. Balanced Binary Tree
- 1195. Fizz Buzz Multithreaded
- 128. Longest Consecutive Sequence
- 138. Copy List with Random Pointer
- 143. Reorder List
- 206. Reverse Linked List
- 226. Invert Binary Tree
- 238. Product of Array Except Self
- 3223.Minimum-Length-of-String-After-Operations
- 3224.Minimum-Array-Changes-to-Make-Differences-Equal
- 33. Search in Rotated Sorted Array
- 347. Top K Frequent Elements
- 380. Insert Delete GetRandom O(1)
- 381.Insert Delete GetRandom O(1) Duplicates allowed
- 412. Fizz Buzz
- 74. Search a 2D Matrix
- 746. Min Cost Climbing Stairs
- 875. Koko Eating Bananas
- Bank Requests
- Binary Gap
- Cyclic Rotation
- Find Target Last Index
- Golang: container/heap
- NUM.LEETCODETITLE
- UMPIRE 1.Two-Sum
- UMPIRE 2.Add-Two-Numbers

2.53 Golang

- 0072. Edit Distance
- A1B2C3: Two Go Routine Print A1B2C3....Z26
- Intersection of Two Sorted Arrays using In Place Approach
- Search Graph in Golang
- WeightedEditDistance

2.54

Goldman Sachs

- 0153.Find-Minimum-in-Rotated-Sorted-Array

2.55

Google

- 0005. Longest Palindromic Substring
- 0011.Container With Most Water
- 0019. Remove Nth Node From End of List
- 128. Longest Consecutive Sequence
- 138. Copy List with Random Pointer

2.56

Greedy

- 0011.Container With Most Water

2.57

Group Anagrams

- 0049.Group Anagrams

2.58

Hard

- 0010.Regular-Expression-Matching
- 0072. Edit Distance
- 0354. Russian Doll Envelope
- 381.Insert Delete GetRandom O(1) Duplicates allowed

2.59

Hash

- 380. Insert Delete GetRandom O(1)
- 381.Insert Delete GetRandom O(1) Duplicates allowed

2.60

Hash Table

- 0003.Longest Substring Without Repeating Characters
- 0981.Time Based Key-Value Store
- 105. Construct Binary Tree from Preorder and Inorder Traversal
- 128. Longest Consecutive Sequence
- 3223.Minimum-Length-of-String-After-Operations
- 3224.Minimum-Array-Changes-to-Make-Differences-Equal
- UMPIRE 1.Two-Sum

2.61

Heap

- 0215. Kth Largest Element in an Array
- 0703. Kth Largest Element in a Stream
- 1046. Last Stone Weight

2.62

Insert Delete GetRandom O(1)

- 380. Insert Delete GetRandom O(1)

2.63

Insert Delete GetRandom O(1) Duplicates allowed

- 381.Insert Delete GetRandom O(1) Duplicates allowed

2.64

Intersection

- Intersection of Two Sorted Arrays using In Place Approach

2.65

Interview

- Find Target Last Index

2.66

Invert Binary Tree

- 226. Invert Binary Tree

2.67

Iterations

- Binary Gap

2.68

Koko Eating Bananas

- 875. Koko Eating Bananas

2.69

Kth Largest Element in a Stream

- 0703. Kth Largest Element in a Stream

2.70

Kth Largest Element in an Array

- 0215. Kth Largest Element in an Array

2.71

LEETCODETITLE

- 0104.Maximum Depth of Binary Tree
- 0153.Find-Minimum-in-Rotated-Sorted-Array
- NUM.LEETCODETITLE

2.72

Last Stone Weight

- 1046. Last Stone Weight

2.73

LeetCode

- 0002.Add Two Numbers

- 0003.Longest Substring Without Repeating Characters
- 0005. Longest Palindromic Substring
- 0010.Regular-Expression-Matching
- 0011.Container With Most Water
- 0015. 3Sum
- 0019. Remove Nth Node From End of List
- 0020. Valid Parentheses
- 0021. Merge Two Sorted Lists
- 0028. Find the Index of the First Occurrence in a String
- 0034. Find First and Last Position of Element in Sorted Array
- 0049.Group Anagrams
- 0070.Climbing Stairs
- 0078. Subsets
- 0104.Maximum Depth of Binary Tree
- 0121.Best Time to Buy and Sell Stock
- 0125. Valid Palindrome
- 0153.Find-Minimum-in-Rotated-Sorted-Array
- 0167.Two Sum II Input Array Is Sorted
- 0215. Kth Largest Element in an Array
- 0217.Contains-Duplicate
- 0242.Valid-Anagram
- 0283. Move Zeroes
- 0322.Coin Change
- 0543. Diameter of Binary Tree
- 0695.Max Area of Island
- 0703. Kth Largest Element in a Stream
- 0733.Flood Fill
- 0981.Time Based Key-Value Store
- 100. Same Tree
- 1046. Last Stone Weight
- 105. Construct Binary Tree from Preorder and Inorder Traversal
- 110. Balanced Binary Tree
- 1195. Fizz Buzz Multithreaded
- 128. Longest Consecutive Sequence
- 138. Copy List with Random Pointer
- 143. Reorder List
- 206. Reverse Linked List
- 226. Invert Binary Tree
- 238. Product of Array Except Self
- 3223.Minimum-Length-of-String-After-Operations
- 3224.Minimum-Array-Changes-to-Make-Differences-Equal
- 33. Search in Rotated Sorted Array
- 347. Top K Frequent Elements

- 380. Insert Delete GetRandom O(1)
- 381.Insert Delete GetRandom O(1) Duplicates allowed
- 412. Fizz Buzz
- 74. Search a 2D Matrix
- 746. Min Cost Climbing Stairs
- 875. Koko Eating Bananas
- NUM.LEETCODETITLE
- UMPIRE 1.Two-Sum
- UMPIRE 2.Add-Two-Numbers

2.74
Left Bound

- Find Target Last Index

2.75
Linked List

- 0002.Add Two Numbers
- 0019. Remove Nth Node From End of List
- 0141.Linked List Cycle
- 0142.Linked List Cycle II
- 0203.Remove Linked List Elements
- 0876.Middle of the Linked List
- 143. Reorder List
- UMPIRE 2.Add-Two-Numbers

2.76
LinkedIn

- 33. Search in Rotated Sorted Array

2.77
Longest Consecutive Sequence

- 128. Longest Consecutive Sequence

2.78
Longest Substring Without Repeating Characters

- 0003.Longest Substring Without Repeating Characters

2.79

Math

- 0002.Add Two Numbers
- UMPIRE 2.Add-Two-Numbers

2.80

Matrix

- 0695.Max Area of Island
- 74. Search a 2D Matrix

2.81

Max Area of Island

- 0695.Max Area of Island

2.82

Medium

- 0002.Add Two Numbers
- 0003.Longest Substring Without Repeating Characters
- 0005. Longest Palindromic Substring
- 0011.Container With Most Water
- 0015. 3Sum
- 0019. Remove Nth Node From End of List
- 0034. Find First and Last Position of Element in Sorted Array
- 0046.Permutations
- 0049.Group Anagrams
- 0059.Spiral Matrix II
- 0075.Sort Colors
- 0078. Subsets
- 0094.Binary Tree Inorder Traversal
- 0142.Linked List Cycle II
- 0153.Find-Minimum-in-Rotated-Sorted-Array
- 0167.Two Sum II Input Array Is Sorted
- 0209. Minimum Size Subarray Sum
- 0215. Kth Largest Element in an Array
- 0300.Longest Increasing Subsequence
- 0310.Minimum Height Trees
- 0322.Coin Change
- 0438.Find All Anagrams in a String

- 0567.Permutation in String
- 0693.Binary Number with Alternating Bits
- 0695.Max Area of Island
- 0752.Open the Lock
- 0981.Time Based Key-Value Store
- 105. Construct Binary Tree from Preorder and Inorder Traversal
- 1143.Longest Common Subsequence
- 1195. Fizz Buzz Multithreaded
- 128. Longest Consecutive Sequence
- 138. Copy List with Random Pointer
- 238. Product of Array Except Self
- 3223.Minimum-Length-of-String-After-Operations
- 3224.Minimum-Array-Changes-to-Make-Differences-Equal
- 33. Search in Rotated Sorted Array
- 347. Top K Frequent Elements
- 380. Insert Delete GetRandom O(1)
- 516. Longest Palindromic Subsequence
- 74. Search a 2D Matrix
- 875. Koko Eating Bananas
- UMPIRE 2.Add-Two-Numbers

2.83

Merge Two Sorted Lists

- 0021. Merge Two Sorted Lists

2.84

Microsoft

- 0002.Add Two Numbers
- 0005. Longest Palindromic Substring
- 0011.Container With Most Water
- 0019. Remove Nth Node From End of List
- 0153.Find-Minimum-in-Rotated-Sorted-Array
- 128. Longest Consecutive Sequence
- 138. Copy List with Random Pointer
- 143. Reorder List
- 33. Search in Rotated Sorted Array
- 412. Fizz Buzz
- 74. Search a 2D Matrix
- UMPIRE 2.Add-Two-Numbers

2.85

Min Cost Climbing Stairs

- 746. Min Cost Climbing Stairs

2.86

Move Zeroes

- 0283. Move Zeroes

2.87

Multiple Pointers

- Cyclic Rotation

2.88

Painless

- Binary Gap
- Cyclic Rotation

2.89

Patience Sorting

- 0300.Longest Increasing Subsequence

2.90

Prefix Sum

- 238. Product of Array Except Self
- 3224.Minimum-Array-Changes-to-Make-Differences-Equal

2.91

Priority Queue

- 0215. Kth Largest Element in an Array
- 0703. Kth Largest Element in a Stream
- 1046. Last Stone Weight

2.92

Product of Array Except Self

- 238. Product of Array Except Self

2.93

Recursion

- 0002.Add Two Numbers
- 0010.Regular-Expression-Matching
- 143. Reorder List
- UMPIRE 2.Add-Two-Numbers

2.94

Reverse Linked List

- 206. Reverse Linked List

2.95

Right Bound

- Find Target Last Index

2.96

Same Tree

- 100. Same Tree

2.97

Search Graph

- Search Graph in Golang

2.98

Search a 2D Matrix

- 74. Search a 2D Matrix

2.99

Slide Windows

- 0121.Best Time to Buy and Sell Stock

2.100

Sliding Window

- 0003.Longest Substring Without Repeating Characters
- 0209. Minimum Size Subarray Sum
- 0438.Find All Anagrams in a String
- 0567.Permutation in String

2.101

Sort

- 0075.Sort Colors

2.102

Sorting

- 0015. 3Sum
- 0215. Kth Largest Element in an Array

2.103

Spotify

- 128. Longest Consecutive Sequence

2.104

Stack

- 0094.Binary Tree Inorder Traversal
- 143. Reorder List

2.105

String

- 0003.Longest Substring Without Repeating Characters
- 0010.Regular-Expression-Matching
- 0028. Find the Index of the First Occurrence in a String

- 0409. Longest Palindrome
- 0981.Time Based Key-Value Store
- 3223.Minimum-Length-of-String-After-Operations

2.106

String Matching

- 0028. Find the Index of the First Occurrence in a String

2.107

Subsets

- 0078. Subsets

2.108

Time Based Key-Value Store

- 0981.Time Based Key-Value Store

2.109

Top K Frequent Elements

- 347. Top K Frequent Elements

2.110

Tree

- 100. Same Tree
- 105. Construct Binary Tree from Preorder and Inorder Traversal

2.111

Two Pointers

- 0011.Container With Most Water
- 0015. 3Sum
- 0019. Remove Nth Node From End of List
- 0028. Find the Index of the First Occurrence in a String
- 0141.Linked List Cycle
- 0142.Linked List Cycle II
- 0167.Two Sum II Input Array Is Sorted
- 0344.Reverse String

- 0876.Middle of the Linked List
- 143. Reorder List

2.112

Two Sum II Input Array Is Sorted

- 0167.Two Sum II Input Array Is Sorted

2.113

Union Find

- 0721.Accounts Merge
- 128. Longest Consecutive Sequence

2.114

Valid Anagram

- 0242.Valid-Anagram

2.115

Valid Palindrome

- 0125. Valid Palindrome

2.116

Valid Parentheses

- 0020. Valid Parentheses

2.117

WeightedEditDistance

- WeightedEditDistance

2.118

amazon

- 0121.Best Time to Buy and Sell Stock

2.119

array

- 0121.Best Time to Buy and Sell Stock

2.120

bloomberg

- 0121.Best Time to Buy and Sell Stock

2.121

container/heap

- Golang: container/heap

2.122

facebook

- 0121.Best Time to Buy and Sell Stock

2.123

heap

- 347. Top K Frequent Elements

2.124

math

- 412. Fizz Buzz

2.125

microsoft

- 0121.Best Time to Buy and Sell Stock

2.126

string

- 412. Fizz Buzz

2.127

uber

- 0121.Best Time to Buy and Sell Stock

3. Algorithms

3.1 A1B2C3

A1B2C3 Algorithm Golang

3.1.1 A1B2C3

□□□ go routine □□ A1B2C3....Z26 Two Go Routine Print A1B2C3....Z26

Channle With Buffer

```go
func ChannelWBuffer() {
    abc := make(chan struct{}, 1)
    num := make(chan struct{}, 1)
    done := make(chan struct{})
    abc <- struct{}{}

    go func() {
        for i := 65; i <= 90; i++ {
            <-abc
            // fmt.Printf("%v", string(rune(i)))
            num <- struct{}{}
        }
    }()
    go func() {
        for i := 1; i <= 26; i++ {
            <-num
            // fmt.Printf("%v", i)
            abc <- struct{}{}
        }
        done <- struct{}{}
    }()
    // time.Sleep(1 * time.Second)
    <-done
}
```

Channel Without Buffer

```go
func ChannelWOBuffer() {
    abc := make(chan struct{})
    num := make(chan struct{})
    done := make(chan struct{})

    go func() {
        for i := 65; i <= 90; i++ {
            // fmt.Printf("%v", string(rune(i)))
            abc <- struct{}{}
            <-num
        }
    }()
    go func() {
        for i := 1; i <= 26; i++ {
            <-abc
            // fmt.Printf("%v", i)
            num <- struct{}{}
        }
        done <- struct{}{}
    }()
    // time.Sleep(1 * time.Second)
    <-done
}
```

Wait Group

```go
func WGLock() {
    abcMux := sync.Mutex{}
    numMux := sync.Mutex{}

    numMux.Lock()
    wg := sync.WaitGroup{}
    wg.Add(2)
    go func() {
```

```go
        defer wg.Done()
        for i := 65; i <= 90; i++ {
            abcMux.Lock()
            // fmt.Printf("%v", string(rune(i)))
            numMux.Unlock()
        }
    }()
    go func() {
        defer wg.Done()
        for i := 1; i <= 26; i++ {
            numMux.Lock()
            // fmt.Printf("%v", i)
            abcMux.Unlock()
        }
    }()
    wg.Wait()
}
```

Benchmark

```
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Algorithms/A1B2C3
cpu: Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz
BenchmarkChannelWBuffer-4          79159             14612 ns/op             344 B/op          5 allocs/op
BenchmarkChannelWOBuffer-4         83068             14451 ns/op             344 B/op          5 allocs/op
BenchmarkWGLock-4                  51303             23072 ns/op              96 B/op          5 allocs/op
PASS
ok      LeetcodeGolang/Algorithms/A1B2C3        4.092s
```

3.2 Find Target Last Index

Algorithms **Easy** **Find Target Last Index** **Go** **Interview** **Left Bound** **Right Bound**

3.2.1 Find Target Last Index

□□□□array□ □□target□array□□□□index□□□ □□□□□□□□□, RightBound

□□□ https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0704.Binary-Search/main.go

---

□□□

```go
func Solution(nums []int, target int) int {
    left, right := 0, len(nums)-1
    result := -1
    for left <= right {
        mid := int(uint(right+left) >> 1)
        if nums[mid] == target {
            // □□□, □□□□
            if target == nums[right] {
                result = right
                break
            } else {
                right--
            }
        } else if nums[mid] < target {
            // □□□
            left = mid + 1
        } else if nums[mid] > target {
            // □□□
            right = mid - 1
        }
    }
    return result
}
```

□□□□□

```go
func SolutionRecursive(nums []int, target int) int {
    left, right := 0, len(nums)-1
    return findTarget(nums, left, right, target)
}

func findTarget(nums []int, left, right, target int) int {
    if left > right {
        return -1
    }

    mid := int(uint(left+right) >> 1)
    if nums[mid] == target {
        if nums[right] == target {
            return right
        } else {
            return findTarget(nums, mid, right-1, target)
        }
    } else if nums[mid] < target {
        return findTarget(nums, mid+1, right, target)
    } else {
        return findTarget(nums, left, mid-1, target)
    }

}
```

```
go test -benchmem -run=none LeetcodeGolang/Algorithms/Find_Target_Last_Index -bench=.
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Algorithms/Find_Target_Last_Index
cpu: Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz
BenchmarkSolution-8             54216429                24.75 ns/op            0 B/op          0 allocs/op
BenchmarkSolutionRecursive-8    40756744                27.75 ns/op            0 B/op          0 allocs/op
PASS
ok      LeetcodeGolang/Algorithms/Find_Target_Last_Index        2.537s
```

RightBound

```go
// □□□□ nums □□ target□□□□□□
func RightBound(nums []int, target int) (index int) {
    lenght := len(nums)
    if lenght <= 0 {
        return -1
    }
    left, right := 0, lenght-1

    for left <= right {
        // □□2
        // mid := left + (right-left)>>1
        mid := int(uint(right+left) >> 1)
        if nums[mid] == target {
            // □□:□□□□□□, □□□□□□□=mid+1
            left = mid + 1
        } else if nums[mid] < target {
            // □□□
            left = mid + 1
        } else if nums[mid] > target {
            // □□□
            right = mid - 1
        }
    }
    // □□□□ □□:right□□□□
    if right < 0 || nums[right] != target {
        return -1
    }
    return right
}
```

LeftBound

```go
// □□□□ nums □□ target□□□□□□
func LeftBound(nums []int, target int) (index int) {
    lenght := len(nums)
    if lenght <= 0 {
        return -1
    }
    left, right := 0, lenght-1

    for left <= right {
        // □□2
        // mid := left + (right-left)>>1
        mid := int(uint(right+left) >> 1)
        if nums[mid] == target {
            // □□□□□□, □□□□□□□
            right = mid - 1
        } else if nums[mid] < target {
            // □□□
            left = mid + 1
        } else if nums[mid] > target {
            // □□□
            right = mid - 1
        }
    }
    // □□□□ □□: left□□□□
    if left >= lenght || nums[left] != target {
        return -1
    }
    return left
}
```

3.3 Intersection of Two Sorted Arrays using In Place Approach

Algorithms **Golang** **Intersection**

3.3.1 Intersection of Two Sorted Arrays using In Place Approach

Intersection of Two Sorted Arrays using In Place Approach

□□□□in-place□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□A□B□□□□□□□□□□□

□□□□□□□□□□□□□□i□j□□□□A□B□□□□□□□□□□□□□□□□□□□A[i]□□B[j]□□□□□□i□□□□□□□□□□A[i]□□B[j]□□□□□□j□□□□□□□□□□A[i]□□B[j]□□□□□□□□□□□□□□□□□□□

3.4 SearchGraph

Algorithms **BFS** **Golang** **Search Graph**

3.4.1 Search Graph

```
     2
   /   \
  /     \
 1 - 3 - 5 - 7
  \        /
   \      6
    \   /
      4

□□Graph,□□□□□□□□, □□print□□□□□

// 1 -> [2, 3, 4]
// 2 -> [1, 5]
// 3 -> [1, 5]
// 4 -> [1, 6]
// 5 -> [2, 3, 7]
// 6 -> [4, 7]
// 7 -> [5, 6]
// print : 1,2,3,4,5,6,7
// Also this is valid : 1,4,3,2,6,5,7

□□ □□□□1, print : 1,2,3,4,5,6,
```

□□

□BFS(Queue) □□□□□□O(V+E)□□□V□□□□□□□□□E□□□□□□□

□□

```go
package searchgraph

import (
    "LeetcodeGolang/Utility/crud"
    "LeetcodeGolang/structures"
    "fmt"
    "strconv"

    jsoniter "github.com/json-iterator/go"
)

func fetchNeighbours(node int) []int {
    crud := crud.NewCrud("https://hackbear.tv/graph/" + strconv.Itoa(node))
    var result = []int{}

    if got := crud.Get(); got.Error != nil {
        // fmt.Printf("got = %v", got)
    } else {
        var json = jsoniter.ConfigCompatibleWithStandardLibrary
        json.Unmarshal([]byte(got.Response), &result)
    }
    return result
}

/*
     2
   /   \
  /     \
 1 - 3 - 5 - 7
  \        /
   \      6
    \   /
      4
*/

// 1 -> [2, 3, 4]
// 2 -> [1, 5]
// 3 -> [1, 5]
// 4 -> [1, 6]
// 5 -> [2, 3, 7]
// 6 -> [4, 7]
// 7 -> [5, 6]
// print : 1,2,3,4,5,6,7
// Also this is valid : 1,4,3,2,6,5,7
```

```go
// You will be working on this part
/*
□□□□□□O(V+E)□□□V□□□□□□□□□E□□□□□□□
*/
func SearchGraph(start int) {
    queue := structures.NewQueue()
    queue.Push(start)
    visit := make(map[int][]int)
    for queue.Len() > 0 {
        node := queue.Pop()
        if _, ok := visit[node]; !ok {
            fmt.Printf("%d ", node)
            neighours := fetchNeighbours(node)
            visit[node] = neighours
            for _, neighour := range neighours {
                if _, ok := visit[neighour]; !ok {
                    queue.Push(neighour)
                }
            }
        }

    }
}

func SearchGraph2(start int) {
    queue := structures.NewQueue()
    queue.Push(start)
    visited := make(map[int]bool)
    for queue.Len() > 0 {
        node := queue.Pop()
        if !visited[node] {
            fmt.Printf("%d ", node)
            visited[node] = true
            neighbors := fetchNeighbours(node)
            for _, neighbor := range neighbors {
                if !visited[neighbor] {
                    queue.Push(neighbor)
                }
            }
        }
    }
}
```

3.4.2 Reference

- https://www.youtube.com/watch?v=BkszA-MvjXA
- https://web.ntnu.edu.tw/~algo/Graph.html

3.5 WeightedEditDistance

Dynamic Programming **Golang** **WeightedEditDistance**

3.5.1 WeightedEditDistance

WeightedEditDistance □□□□

```go
    // □□□□□□□□□□□□□□
    // dp□□□ dp[i][j] □□□ word1[:i] □□□ word2[:j] □□□□□□□
    dp := make([][]int, m+1)
    for i := 0; i <= m; i++ {
        dp[i] = make([]int, n+1)
    }

    // □□□□□□, base case
    for i := 1; i <= m; i++ {
        dp[i][0] = dp[i-1][0] + weights[rune(word1[i-1])]
    }

    // □□□□□□
    for j := 1; j <= n; j++ {
        dp[0][j] = dp[0][j-1] + weights[rune(word2[j-1])]

    }
    // fmt.Println(dp)
    // □□□□□□□□
    for i := 1; i <= m; i++ {
        for j := 1; j <= n; j++ {
            if word1[i-1] == word2[j-1] {
                dp[i][j] = dp[i-1][j-1]
            } else {
                // □□□□□□□□□□□□□□□
                // insert: □□□ word1[i]□□□□□□word2[j]□□□□□, □□word2[j]□□□□□,□□j, □□□i□□, □□□□+1
                insertCost := dp[i][j-1] + weights[rune(word2[j-1])]
                deleteCost := dp[i-1][j] + weights[rune(word1[i-1])]
                replaceCost := dp[i-1][j-1] + weights[rune(word1[i-1])] + weights[rune(word2[j-1])]
                // □□□□□□□□□□□□□□□□□
                dp[i][j] = min(insertCost, deleteCost, replaceCost)
            }
        }
    }
    fmt.Println(dp)
    return dp[m][n]
}

// □□□□□□□□□□□□□□□□
func min(a, b, c int) int {
    if a <= b && a <= c {
        return a
    } else if b <= a && b <= c {
        return b
    } else {
        return c
    }
}
```

0072.Edit-Distance

4. CodeSignal

4.1 Code Signal

4.1.1 Reference

- https://app.codesignal.com/arcade/intro/level-1
- https://github.com/abrar-abu/golanee/main
- https://github.com/crusty0gphr/bank-request-processor
- https://github.com/abrar-mahedavi/challenges/tree/main

4.2 Bank Requests

Bank Requests **CodeSignal** **Go**

4.2.1 Bank Requests

*A solution to one of my coding interview questions. Complete solution - written in **GoLang***

TASK:

You've been asked to program a bot for a popular bank that will automate the management of incoming requests. There are three types of requests the bank can receive:

- transfer i j sum: request to transfer sum amount of money from the `i-th` account to the `j-th` one
- deposit `i` sum: request to deposit sum amount of money in the `i-th` account
- withdraw `i` sum: request to withdraw sum amount of money from the `i-th` account.

Your bot should also be able to process invalid requests. There are two types of invalid requests: invalid account number in the requests; deposit / withdrawal of a larger amount of money than is currently available.

For the given list of accounts and requests, return the state of accounts after all requests have been processed, or an array of a single element `[- <request_id>]` (please note the minus sign), where is the **1-based** index of the first invalid request.

Example for `accounts = [10, 100, 20, 50, 30]` and `requests = ["withdraw 2 10", "transfer 5 1 20", "deposit 5 20", "transfer 3 4 15"]`, the output should be `bankRequests(accounts, requests) = [30, 90, 5, 65, 30]`.

`accounts = [10, 100, 20, 50, 30]` : □□ ID:1 □10□, ID:2 □100□ ... "withdraw 2 10" : □ID:2 □□ 10□, □□100-10 □□□□accounts= [10, 90, 20, 50, 30]

Here are the states of accounts after each request: * `"withdraw 2 10": [10, 90, 20, 50, 30]` * `"transfer 5 1 20": [30, 90, 20, 50, 10]` * `"deposit 5 20": [30, 90, 20, 50, 30]` * `"transfer 3 4 15": [30, 90, 5, 65, 30],` which is the answer

For `accounts = [20, 1000, 500, 40, 90]` and `requests = ["deposit 3 400", "transfer 1 2 30", "withdraw 4 50"]`, the output should be `bankRequests(accounts, requests) = [-2]`.

After the first request, accounts becomes equal to `[20, 1000, 900, 40, 90]`, but the second one turns it into `[-10, 1030, 900, 40, 90]`, which is invalid. Thus, the second request is invalid, and the answer is `[-2]`. Note that the last request is also invalid, but it shouldn't be included in the answer.

□□□□: □□□□□

□□□□□□□□□□□□□ * "withdraw 2 10": [10, 90, 20, 50, 30] * "transfer 5 1 20": [30, 90, 20, 50, 10] * "deposit 5 20": [30, 90, 20, 50, 30] * "transfer 3 4 15": [30, 90, 5, 65, 30] □□□□□

□□ accounts = [20, 1000, 500, 40, 90] □ requests = ["deposit 3 400", "transfer 1 2 30", "withdraw 4 50"] □□□□□ bankRequests(accounts, requests) = [-2] □

□□□□□□□□□□□ [20, 1000, 900, 40, 90] □□□□□□□□□□□ [-10, 1030, 900, 40, 90] □□□□□□□□□□□□□□□□□□□□□□ [-2] □□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□

```go
package bankrequests

import (
    "errors"
    "fmt"
    "regexp"
    "strconv"
    "strings"
    "sync"
)

var (
    errInvaldTransfer = errors.New("invalid transfer")
    errInvaldWithdraw = errors.New("invalid withdraw")
    errInvaldDeposit  = errors.New("invalid deposit")
    errUnknownAction  = errors.New("unknown action")
)

var requestScheme = [][]string{
    3: {"action", "from", "amount"},
    4: {"action", "from", "to", "amount"},
}

type BankService interface {
    getActionName() string
    transfer() error
    withdraw() error
    deposit() error
}

type Bank struct {
    BankService
    action        string
    requestId     int
    from          int
    to            int
    amount        int
    balances      []int
    requestFailed bool
    failedRequest []int

    mutex sync.RWMutex
}

func (bank *Bank) getActionName() string {
    return bank.action
}

func (bank *Bank) transfer() error {
    bank.mutex.Lock()
    defer bank.mutex.Unlock()

    invalidAmout := bank.balances[bank.from-1] < bank.amount
    invalidReceiver := len(bank.balances) < bank.to
    if invalidAmout || invalidReceiver {
        return errInvaldTransfer
    }
    bank.balances[bank.from-1] -= bank.amount
    bank.balances[bank.to-1] += bank.amount
    return nil
}

func (bank *Bank) withdraw() error {
    bank.mutex.Lock()
    defer bank.mutex.Unlock()

    if bank.balances[bank.from-1] < bank.amount {
        return errInvaldWithdraw
    }
    bank.balances[bank.from-1] -= bank.amount
    return nil
}

func (bank *Bank) deposit() error {
    bank.mutex.Lock()
    defer bank.mutex.Unlock()

    invalidAccount := len(bank.balances) < bank.from
```

```go
    if invalidAccount {
        return errInvaldDeposit
    }
    bank.balances[bank.from-1] += bank.amount
    return nil
}

func NewBank() *Bank {
    return &Bank{}
}

// □□□□: □□□□□
// □□□□□`[- <request_id>]`□□□□□□□□□□
func (bank *Bank) failedAction() {
    bank.mutex.Lock()
    defer bank.mutex.Unlock()
    failed := []int{-bank.requestId}

    bank.balances = []int{}
    bank.requestFailed = true
    bank.failedRequest = failed
}

func (bank *Bank) Action() {
    var requestErr error
    action := bank.getActionName()
    // fmt.Println("action:", action)

    switch action {
    case "transfer":
        requestErr = bank.transfer()
    case "withdraw":
        requestErr = bank.withdraw()
    case "deposit":
        requestErr = bank.deposit()
    default:
        requestErr = errUnknownAction
    }
    if requestErr != nil {
        fmt.Println("requestErr:", requestErr)
        bank.failedAction()
    }
}

func extractRequestParams(request string) map[string]interface{} {
    res := map[string]interface{}{}
    reqSlice := strToSlice(request)
    fmt.Println("reqSlice:", reqSlice)

    // len is 3: {"action", "from", "amount"},
    // len is 4: {"action", "from", "to", "amount"},
    scheme := requestScheme[len(reqSlice)]

    for i, v := range scheme {
        res[v] = reqSlice[i]
    }
    return res
}

func strToSlice(str string) []interface{} {
    var res []interface{}
    erp := strings.Fields(str)

    re, err := regexp.Compile("[0-9]+")
    if err != nil {
        fmt.Println("Error compiling regex:", err)
        return nil
    }

    for _, v := range erp {
        if isNum := re.MatchString(v); isNum {
            if n, _ := strconv.Atoi(v); n != 0 {
                res = append(res, n)
            }
        } else {
            res = append(res, v)
        }
    }

    return res
}

func bankRequests(requests []string, balances []int) []int {
    var res []int
    for index, request := range requests {
        reqParams := extractRequestParams(request)
        bank := NewBank()
        bank.requestId = index + 1
        bank.action = reqParams["action"].(string)
        bank.amount = reqParams["amount"].(int)
        bank.from = reqParams["from"].(int)
        if _, ok := reqParams["to"]; ok {
            bank.to = reqParams["to"].(int)
        }
```

```
        bank.balances = balances
        bank.Action()

        if bank.requestFailed {
            return bank.failedRequest
        }
        res = bank.balances
    }
    return res
}
```

unit test

```
package bankrequests

import (
    "reflect"
    "testing"
)

var tests = []struct {
    arg1 []string
    arg2 []int
    want []int
}{
    {
        []string{
            "transfer 1 4 10",
            "deposit 3 10",
            "withdraw 5 15",
        },
        []int{20, 30, 10, 90, 60},
        []int{10, 30, 20, 100, 45},
    },
    {
        []string{
            "transfer 1 4 40",
            "deposit 3 10",
            "withdraw 5 65",
        },
        []int{20, 30, 10, 90, 60},
        []int{-1},
    },
    {
        []string{
            "withdraw 2 10",
            "transfer 5 1 20",
            "deposit 5 20",
            "transfer 3 4 15",
        },
        []int{10, 100, 20, 50, 30},
        []int{30, 90, 5, 65, 30},
    },
    {
        []string{
            "deposit 3 400",
            "transfer 1 2 30",
            "withdraw 4 50",
        },
        []int{20, 1000, 500, 40, 90},
        []int{-2},
    },
}

func TestBankRequests(t *testing.T) {
    for _, tt := range tests {
        if got := bankRequests(tt.arg1, tt.arg2); !reflect.DeepEqual(got, tt.want) {
            t.Errorf("got = %v, want = %v", got, tt.want)
        }
    }
}
```

Reference

- https://github.com/crusty0gphr/bank-request-processor/tree/master

5. Codility

5.1 Lesson

5.1.1 0001.Iterations

Binary Gap

Bitwise Manipulation **Codility** **Go** **Iterations** **Painless**

BINARYGAP

START Find longest sequence of zeros in binary representation of an integer.

□□

A binary gap within a positive integer N is any maximal sequence of consecutive zeros that is surrounded by ones at both ends in the binary representation of N.

For example, number 9 has binary representation 1001 and contains a binary gap of length 2. The number 529 has binary representation 1000010001 and contains two binary gaps: one of length 4 and one of length 3. The number 20 has binary representation 10100 and contains one binary gap of length 1. The number 15 has binary representation 1111 and has no binary gaps. The number 32 has binary representation 100000 and has no binary gaps.

Write a function:

func Solution(N int) int

that, given a positive integer N, returns the length of its longest binary gap. The function should return 0 if N doesn't contain a binary gap.

For example, given N = 1041 the function should return 5, because N has binary representation 10000010001 and so its longest binary gap is of length 5. Given N = 32 the function should return 0, because N has binary representation '100000' and thus no binary gaps.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [1..2,147,483,647]. Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□□□, □□□□□□□□□,□□bit□□1□□□□□□□0□□□

□□□□

□□□bit 1□□□, □□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□N□□□□□□□□□□□□□□□□□□□□□□□□□□□1□□□□□□□□□□□□□□□□□□□□□□□□□□□1□

□□□□□: □□□□□□□□□□□□□□□N□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□N□□□□k□□□□□□□□□O(k)□

□□□□□: □□□□□□□□□□□□□□□□□□□□

```go
func Solution(N int) int {
    maxLen, curLen := 0, 0
    findOne := false
    for N > 0 {
        curBit := N & 1
        if curBit == 1 {
            curLen = 0
            findOne = true
        } else if curBit == 0 && findOne {
            curLen++
        }

        if curLen > maxLen {
            maxLen = curLen
        }
        N = N >> 1
    }
    return maxLen
}

// https://wandbox.org/permlink/totZwDAbL1wCgsqt
func evil(x int) int {
    if x&(x+1) > 0 {
        return evil(x|(x>>1)) + 1
    } else {
        return 0
    }
}

func SolutionRecur(N int) int {
    for (N & 1) == 0 {
        N = N >> 1
    }
    return evil(N)
}
```

5.1.2 0002.Array

CyclicRotation

Array **Codility** **Go** **Multiple Pointers** **Painless**

CYCLICROTATION

An array A consisting of N integers is given. Rotation of the array means that each element is shifted right by one index, and the last element of the array is moved to the first place. For example, the rotation of array A = [3, 8, 9, 7, 6] is [6, 3, 8, 9, 7] (elements are shifted right by one index and 6 is moved to the first place).

The goal is to rotate array A K times; that is, each element of A will be shifted to the right K times.

Write a function:

func Solution(A []int, K int) []int

that, given an array A consisting of N integers and an integer K, returns the array A rotated K times.

For example, given

```
A = [3, 8, 9, 7, 6]
K = 3
```

the function should return [9, 7, 6, 3, 8]. Three rotations were made:

```
[3, 8, 9, 7, 6] -> [6, 3, 8, 9, 7]
[6, 3, 8, 9, 7] -> [7, 6, 3, 8, 9]
[7, 6, 3, 8, 9] -> [9, 7, 6, 3, 8]
```

For another example, given

```
A = [0, 0, 0]
K = 1
```

the function should return [0, 0, 0]

Given

```
A = [1, 2, 3, 4]
K = 4
```

the function should return [1, 2, 3, 4]

Assume that:

N and K are integers within the range [0..100]; each element of array A is an integer within the range [−1,000..1,000]. In your solution, focus on correctness. The performance of your solution will not be the focus of the assessment.

Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

CyclicRotation□□□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0002.Array/CyclicRotation/CyclicRotation.go

```go
package cyclicrotation

func Solution(A []int, K int) []int {
    if K == 0 || len(A) <= 1 {
        return A
    }

    K = K % len(A)
    return append(A[len(A)-K:], A[:len(A)-K]...)
}

func Solution2(A []int, K int) []int {
    if K == 0 || len(A) <= 1 {
        return A
    }
    if K > len(A) {
        K = K % len(A)
    }
    return append(A[len(A)-K:], A[:len(A)-K]...)
}
```

OddOccurrencesInArray

ODDOCCURRENCESINARRAY

A non-empty array A consisting of N integers is given. The array contains an odd number of elements, and each element of the array can be paired with another element that has the same value, except for one element that is left unpaired.

For example, in array A such that:

A[0] = 9 A[1] = 3 A[2] = 9 A[3] = 3 A[4] = 9 A[5] = 7 A[6] = 9 the elements at indexes 0 and 2 have value 9, the elements at indexes 1 and 3 have value 3, the elements at indexes 4 and 6 have value 9, the element at index 5 has value 7 and is unpaired. Write a function:

func Solution(A []int) int

that, given an array A consisting of N integers fulfilling the above conditions, returns the value of the unpaired element.

For example, given array A such that:

A[0] = 9 A[1] = 3 A[2] = 9 A[3] = 3 A[4] = 9 A[5] = 7 A[6] = 9 the function should return 7, as explained in the example above.

Write an efficient algorithm for the following assumptions:

N is an odd integer within the range [1..1,000,000]; each element of array A is an integer within the range [1..1,000,000,000]; all but one of the values in A occur an even number of times. Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□□□□□array A□□□□N□□□□□□□□□□□□□□□□

□□□□

1. □□□: □□□Map□□□□□□□□□□□, □□□□□□□□□
2. □□□: □□□□□XOR□□, □□□□□□□□XOR□□0, □□□□□□□□□□□

□□

- https://app.codility.com/programmers/lessons/2-arrays/odd_occurrences_in_array/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0002.Array/OddOccurrencesInArray/OddOccurrencesInArray.go

```go
package oddoccurrencesinarray

func Solution(A []int) int {
    intMap := make(map[int]int)
    for i := 0; i < len(A); i++ {
        intMap[A[i]] += 1
    }

    for k, v := range intMap {
        if v%2 != 0 {
            return k
        }
    }
    return -1
}

// □□□□□XOR□□, □□□□□□□□XOR□□0, □□□□□□□□□□
// □□□□□□□□□□□
func Solution2(A []int) int {
    result := 0
    for i := 0; i < len(A); i++ {
        result ^= A[i]
    }
    return result
}
```

5.1.3 0003.Time Complexity

FrogJmp

FROGJMP

A small frog wants to get to the other side of the road. The frog is currently located at position X and wants to get to a position greater than or equal to Y. The small frog always jumps a fixed distance, D.

Count the minimal number of jumps that the small frog must perform to reach its target.

Write a function:

func Solution(X int, Y int, D int) int

that, given three integers X, Y and D, returns the minimal number of jumps from position X to a position equal to or greater than Y.

For example, given:

X = 10 Y = 85 D = 30 the function should return 3, because the frog will be positioned as follows:

after the first jump, at position 10 + 30 = 40 after the second jump, at position 10 + 30 + 30 = 70 after the third jump, at position 10 + 30 + 30 + 30 = 100 Write an efficient algorithm for the following assumptions:

X, Y and D are integers within the range [1..1,000,000,000]; X ≤ Y. Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□□□

□□

- https://app.codility.com/programmers/lessons/3-time_complexity/frog_jmp/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0003.Time-Complexity/FrogJmp/FrogJmp.go

```go
package frogjump

import (
    "math"
)

func Solution(X int, Y int, D int) int {
    if Y < X {
        return 0
    }
    remainDist := Y - X
    return int(math.Ceil(float64(remainDist) / float64(D)))
}
```

PermMissingElem

PERMMISSINGELEM

An array A consisting of N different integers is given. The array contains integers in the range [1..(N + 1)], which means that exactly one element is missing.

Your goal is to find that missing element.

Write a function:

func Solution(A []int) int

that, given an array A, returns the value of the missing element.

For example, given array A such that:

A[0] = 2 A[1] = 3 A[2] = 1 A[3] = 5 the function should return 4, as it is the missing element.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [0..100,000]; the elements of A are all distinct; each element of array A is an integer within the range [1..(N + 1)]. Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□□□

□□

- https://app.codility.com/programmers/lessons/3-time_complexity/perm_missing_elem/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0003.Time-Complexity/PermMissingElem/PermMissingElem.go

```go
package permmissingelem

func Solution(A []int) int {
    if len(A) < 1 {
        return 1
    }

    n := len(A) + 1
    predictSume := (n + 1) * n / 2

    var sum int
    for _, v := range A {
        sum += v
    }
    return predictSume - sum
}
```

TapeEquilibrium

TAPEEQUILIBRIUM

Minimize the value |(A[0] + ... + A[P-1]) - (A[P] + ... + A[N-1])|.

A non-empty array A consisting of N integers is given. Array A represents numbers on a tape.

Any integer P, such that 0 < P < N, splits this tape into two non-empty parts: A[0], A[1], ..., A[P − 1] and A[P], A[P + 1], ..., A[N − 1].

The difference between the two parts is the value of: |(A[0] + A[1] + ... + A[P − 1]) − (A[P] + A[P + 1] + ... + A[N − 1])|

In other words, it is the absolute difference between the sum of the first part and the sum of the second part.

For example, consider array A such that:

A[0] = 3 A[1] = 1 A[2] = 2 A[3] = 4 A[4] = 3 We can split this tape in four places:

P = 1, difference = |3 − 10| = 7 P = 2, difference = |4 − 9| = 5 P = 3, difference = |6 − 7| = 1 P = 4, difference = |10 − 3| = 7 Write a function:

func Solution(A []int) int

that, given a non-empty array A of N integers, returns the minimal difference that can be achieved.

For example, given:

A[0] = 3 A[1] = 1 A[2] = 2 A[3] = 4 A[4] = 3 the function should return 1, as explained above.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [2..100,000]; each element of array A is an integer within the range [−1,000..1,000].

Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□

□□□

□□

- https://app.codility.com/programmers/lessons/3-time_complexity/tape_equilibrium/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0003.Time-Complexity/TapeEquilibrium/TapeEquilibrium.go

```go
package TapeEquilibrium

import "math"

func Solution(A []int) int {

    totalSum := 0
    for _, v := range A {
        totalSum += v
    }

    leftSum := A[0]
    rightSum := totalSum - leftSum
    result := math.MaxInt32
    for i := 1; i < len(A); i++ {
        tmpDiff := int(math.Abs(float64(rightSum) - float64(leftSum)))
        if tmpDiff < result {
            result = tmpDiff
        }
        rightSum -= A[i]
        leftSum += A[i]
    }

    if result == math.MaxInt32 {
        result = 0
    }
```

```
    return result
}
```

5.1.4 0004.Counting Elements

FrogRiverOne

FROGRIVERONE

Find the earliest time when a frog can jump to the other side of a river.

A small frog wants to get to the other side of a river. The frog is initially located on one bank of the river (position 0) and wants to get to the opposite bank (position X+1). Leaves fall from a tree onto the surface of the river.

You are given an array A consisting of N integers representing the falling leaves. A[K] represents the position where one leaf falls at time K, measured in seconds.

The goal is to find the earliest time when the frog can jump to the other side of the river. The frog can cross only when leaves appear at every position across the river from 1 to X (that is, we want to find the earliest moment when all the positions from 1 to X are covered by leaves). You may assume that the speed of the current in the river is negligibly small, i.e. the leaves do not change their positions once they fall in the river.

For example, you are given integer X = 5 and array A such that:

A[0] = 1 A[1] = 3 A[2] = 1 A[3] = 4 A[4] = 2 A[5] = 3 A[6] = 5 A[7] = 4 In second 6, a leaf falls into position 5. This is the earliest time when leaves appear in every position across the river.

Write a function:

func Solution(X int, A []int) int

that, given a non-empty array A consisting of N integers and integer X, returns the earliest time when the frog can jump to the other side of the river.

If the frog is never able to jump to the other side of the river, the function should return −1.

For example, given X = 5 and array A such that:

A[0] = 1 A[1] = 3 A[2] = 1 A[3] = 4 A[4] = 2 A[5] = 3 A[6] = 5 A[7] = 4 the function should return 6, as explained above.

Write an efficient algorithm for the following assumptions:

N and X are integers within the range [1..100,000]; each element of array A is an integer within the range [1..X]. Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□X□□□□□□, □□□□□□□□□□□□□□□

□□□□

□□map□□□□□□□

□□

- https://app.codility.com/programmers/lessons/4-counting_elements/frog_river_one/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0004.Counting-Elements/FrogRiverOne/FrogRiverOne.go

```go
package FrogRiverOne

func Solution(X int, A []int) int {
    intMap := make(map[int]bool)
    for i := 0; i < len(A); i++ {
        intMap[A[i]] = true
        if len(intMap) == X {
            return i
        }
    }
```

```
    return -1
}
```

MaxCounters

MAXCOUNTERS

Calculate the values of counters after applying all alternating operations: increase counter by 1; set value of all counters to current maximum.

You are given N counters, initially set to 0, and you have two possible operations on them:

increase(X) − counter X is increased by 1, max counter − all counters are set to the maximum value of any counter. A non-empty array A of M integers is given. This array represents consecutive operations:

if A[K] = X, such that 1 ≤ X ≤ N, then operation K is increase(X), if A[K] = N + 1 then operation K is max counter. For example, given integer N = 5 and array A such that:

```
A[0] = 3
A[1] = 4
A[2] = 4
A[3] = 6
A[4] = 1
A[5] = 4
A[6] = 4
```

the values of the counters after each consecutive operation will be:

```
(0, 0, 1, 0, 0)
(0, 0, 1, 1, 0)
(0, 0, 1, 2, 0)
(2, 2, 2, 2, 2)
(3, 2, 2, 2, 2)
(3, 2, 2, 3, 2)
(3, 2, 2, 4, 2)
```

The goal is to calculate the value of every counter after all operations.

Write a function:

func Solution(N int, A []int) []int

that, given an integer N and a non-empty array A consisting of M integers, returns a sequence of integers representing the values of the counters.

Result array should be returned as an array of integers.

For example, given:

```
A[0] = 3
A[1] = 4
A[2] = 4
A[3] = 6
A[4] = 1
A[5] = 4
A[6] = 4
```

the function should return [3, 2, 2, 4, 2], as explained above.

Write an efficient algorithm for the following assumptions:

N and M are integers within the range [1..100,000]; each element of array A is an integer within the range [1..N + 1]. Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□A[i] □□ N □□□□□□□□□□□□+1, □□A[i] □□ N □□□□□□□□□□□□□□□□□□□□□□□□

□□□□

□□

https://app.codility.com/programmers/lessons/4-counting_elements/

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0004.Counting-Elements/MaxCounters/MaxCounters.go

```go
package MaxCounters

func Max(x, y int) int {
    if x > y {
        return x
    }
    return y
}

// □□ O(N+M) , □□ O(N)
func Solution(N int, A []int) []int {
    result := make([]int, N)
    maxNum := 0
    nowMaxNum := 0
    for i := 0; i < len(A); i++ {
        if A[i] > N {
            // □□A[i] □□ N □□□□□□□□□□□□□□□□□□□□□
            maxNum = nowMaxNum
        } else {
            // □□A[i] □□ N □□□□□□□□□□□□+1,
            if result[A[i]-1] < maxNum {
                result[A[i]-1] = maxNum
            }
            result[A[i]-1]++

            if nowMaxNum < result[A[i]-1] {
                nowMaxNum = result[A[i]-1]
            }
        }
    }
    for i := 0; i < N; i++ {
        if result[i] < maxNum {
            result[i] = maxNum
        }
    }
    return result
}
```

MissingInteger

MISSINGINTEGER

Find the smallest positive integer that does not occur in a given sequence.

This is a demo task.

Write a function:

func Solution(A []int) int

that, given an array A of N integers, returns the smallest positive integer (greater than 0) that does not occur in A.

For example, given A = [1, 3, 6, 4, 1, 2], the function should return 5.

Given A = [1, 2, 3], the function should return 4.

Given A = [−1, −3], the function should return 1.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [1..100,000]; each element of array A is an integer within the range [−1,000,000..1,000,000].
Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□□array□□□□□□□□

□□□□

□□□□□□□□□□□□, □□□□1□□□□□□□□□□□□□□□□□□

□□

https://app.codility.com/programmers/lessons/4-counting_elements/missing_integer/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0004.Counting-Elements/MissingInteger/MissingInteger.go

```go
package MissingInteger

func Solution(A []int) int {
    smallNum := 1
    intMap := make(map[int]bool)

    // □□□□□□□□map
    for _, v := range A {
        if v > 0 && !intMap[v] {
            intMap[v] = true
        }
    }

    for i := 1; i <= len(intMap); i++ {
        if !intMap[i] {
            // □□□□□□map□□
            return i
        }
        smallNum = i + 1
    }

    return smallNum
}
```

PermCheck

PERMCHECK

Check whether array A is a permutation.

A non-empty array A consisting of N integers is given.

A permutation(□□) is a sequence containing each element from 1 to N once, and only once.

For example, array A such that:

```
A[0] = 4
A[1] = 1
A[2] = 3
A[3] = 2
```

is a permutation, but array A such that:

```
A[0] = 4
A[1] = 1
A[2] = 3
```

is not a permutation, because value 2 is missing.

The goal is to check whether array A is a permutation.

Write a function:

func Solution(A []int) int

that, given an array A, returns 1 if array A is a permutation and 0 if it is not.

For example, given array A such that:

```
A[0] = 4
A[1] = 1
A[2] = 3
A[3] = 2
```

the function should return 1.

Given array A such that:

```
A[0] = 4
A[1] = 1
A[2] = 3
```

the function should return 0.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [1..100,000]; each element of array A is an integer within the range [1..1,000,000,000].
Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□□□□□□□array □□ 1 □□□□1

□□□□

□□lesson 4□MissingInteger. □□□□□□□□□map. □□□□□□□□□□□□□□□,□□□□□□□□□ □□□□□□□□□□

□□

https://app.codility.com/programmers/lessons/4-counting_elements/perm_check/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0004.Counting-Elements/PermCheck/PermCheck.go

```go
package PermCheck

func Solution(A []int) int {
    intMap := make(map[int]bool)

    for _, v := range A {
        if !intMap[v] {
            intMap[v] = true
        } else {
            // ▫▫▫▫
            return 0
        }
    }

    for i := 1; i <= len(A); i++ {
        if !intMap[i] {
            return 0
        }
    }
    return 1
}

func Solution2(A []int) int {
    intMap := make(map[int]bool)
    sum := 0
    for _, v := range A {
        if !intMap[v] {
            intMap[v] = true
            sum += v
        } else {
            // ▫▫▫▫
            return 0
        }
    }

    if sum == (len(A)+1)*len(A)/2 {
        return 1
    }
    return 0
}
```

5.1.5 0005.Prefix Sums

CountDiv

COUNTDIV

Compute number of integers divisible by k in range [a..b].

Write a function:

func Solution(A int, B int, K int) int

that, given three integers A, B and K, returns the number of integers within the range [A..B] that are divisible by K, i.e.:

{ i : A ≤ i ≤ B, i mod K = 0 }

For example, for A = 6, B = 11 and K = 2, your function should return 3, because there are three numbers divisible by 2 within the range [6..11], namely 6, 8 and 10.

Write an efficient algorithm for the following assumptions:

A and B are integers within the range [0..2,000,000,000]; K is an integer within the range [1..2,000,000,000]; A ≤ B. Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

A~B□□□□□ mod K □□0 □□□□□□

□□□□

B/K □□□□□□, A/K □□□□. □□□□□□□□□□□□□. □□A%K==0 □□□+1

□□

- https://app.codility.com/programmers/lessons/5-prefix_sums/count_div/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0005.Prefix-Sums/CountDiv/CountDiv.go

```go
package CountDiv

import "math"

// □□: O(n)
func SolutionBurst(A int, B int, K int) int {
    result := 0
    for i := A; i <= B; i++ {
        if i%K == 0 {
            result++
        }
    }

    return result
}

// □□:O(1) □□: O(1)
func Solution(A int, B int, K int) int {
    result := 0
    if A%2 == 0 {
        result = 1
    }
    return int(math.Floor(float64(B/K))) - int(math.Floor(float64(A/K))) + result
}
```

GenomicRangeQuery

GENOMICRANGEQUERY

Find the minimal nucleotide(□□□) from a range of sequence DNA.

A DNA sequence can be represented as a string consisting of the letters A, C, G and T, which correspond to the types of successive nucleotides in the sequence. Each nucleotide has an impact factor, which is an integer. Nucleotides of types A, C, G and T have impact factors of 1, 2, 3 and 4, respectively. You are going to answer several queries of the form: What is the minimal impact factor of nucleotides contained in a particular part of the given DNA sequence?

The DNA sequence is given as a non-empty string S = S[0]S[1]...S[N-1] consisting of N characters. There are M queries, which are given in non-empty arrays P and Q, each consisting of M integers. The K-th query (0 ≤ K < M) requires you to find the minimal impact factor of nucleotides contained in the DNA sequence between positions P[K] and Q[K] (inclusive).

For example, consider string S = CAGCCTA and arrays P, Q such that:

```
P[0] = 2    Q[0] = 4
P[1] = 5    Q[1] = 5
P[2] = 0    Q[2] = 6
```

The answers to these M = 3 queries are as follows:

The part of the DNA between positions 2 and 4 contains nucleotides G and C (twice), whose impact factors are 3 and 2 respectively, so the answer is 2. The part between positions 5 and 5 contains a single nucleotide T, whose impact factor is 4, so the answer is 4. The part between positions 0 and 6 (the whole string) contains all nucleotides, in particular nucleotide A whose impact factor is 1, so the answer is 1. Write a function:

func Solution(S string, P []int, Q []int) []int

that, given a non-empty string S consisting of N characters and two non-empty arrays P and Q consisting of M integers, returns an array consisting of M integers specifying the consecutive answers to all queries.

Result array should be returned as an array of integers.

For example, given the string S = CAGCCTA and arrays P, Q such that:

```
P[0] = 2    Q[0] = 4
P[1] = 5    Q[1] = 5
P[2] = 0    Q[2] = 6
```

the function should return the values [2, 4, 1], as explained above.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [1..100,000]; M is an integer within the range [1..50,000]; each element of arrays P, Q is an integer within the range [0..N − 1]; P[K] ≤ Q[K], where 0 ≤ K < M; string S consists only of upper-case English letters A, C, G, T.
Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□

CAGCCTA A □□1, C □□2, G □□3 ,T □□4 K=0: P[0]=2, Q[0]=4 □□□□□□ DNA(GCC)□□□□□3□2, □□□□□2. K=1: P[1]=5, Q[1]=5 DNA(T),□□□□4. K=2: P[2]=0, Q[2]=6 DNA(CAGCCTA),□□□□1.

□□□

□□

https://app.codility.com/programmers/lessons/5-prefix_sums/genomic_range_query/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0005.Prefix-Sums/GenomicRangeQuery/GenomicRangeQuery.go

```
package GenomicRangeQuery
```

```go
func Solution(S string, P []int, Q []int) []int {
    A, C, G, T := prefixSums(S)
    result := make([]int, len(P))
    /*
        // fmt.Println("A: ", A)
        // fmt.Println("C: ", C)
        // fmt.Println("G: ", G)
        // fmt.Println("T: ", T)
        idx  0 1 2 3 4 5 6 7
        S:  [C A G C C T A]
        A:  [0 0 1 1 1 1 1 2]
        C:  [0 1 1 1 2 3 3 3]
        G:  [0 0 0 1 1 1 1 1]
        T:  [0 0 0 0 0 0 1 1]
        P:  [2 5 0]
        Q:  [4 5 6]
    */

    for k, _ := range P {
        // □□ A[end of slice]-A[Begin of Slice]□□□□□ □□□□□□ A □□□
        if A[Q[k]+1]-A[P[k]] > 0 {
            result[k] = 1
        } else if C[Q[k]+1]-C[P[k]] > 0 {
            result[k] = 2
        } else if G[Q[k]+1]-G[P[k]] > 0 {
            result[k] = 3
        } else if T[Q[k]+1]-T[P[k]] > 0 {
            result[k] = 4
        }
    }

    return result
}

// □□□□□□□□□□□□□A,C,G,T□□. □□□□0
func prefixSums(S string) ([]int, []int, []int, []int) {
    n := len(S)
    A := make([]int, n+1)
    C := make([]int, n+1)
    G := make([]int, n+1)
    T := make([]int, n+1)

    for i := 1; i < n+1; i++ {
        s := string(S[i-1])
        A[i] = A[i-1]
        C[i] = C[i-1]
        G[i] = G[i-1]
        T[i] = T[i-1]

        switch s {
        case "A":
            A[i]++
        case "C":
            C[i]++
        case "G":
            G[i]++
        case "T":
            T[i]++
        }

    }

    return A, C, G, T
}

func inLoop(arr string, s string) bool {
    for _, v := range arr {
        if string(v) == s {
            return true
        }
    }
    return false
}

// O(N * M)
func SolutionBurst(S string, P []int, Q []int) []int {
    result := make([]int, len(P))
    for i := 0; i < len(P); i++ {
        tmp := S[P[i] : Q[i]+1]
        if inLoop(tmp, "A") {
            result[i] = 1
        } else if inLoop(tmp, "C") {
            result[i] = 2
        } else if inLoop(tmp, "G") {
            result[i] = 3
        } else if inLoop(tmp, "T") {
            result[i] = 4
        }
    }
    return result
}

/*
def solutionBySlice(S, P, Q):
```

```
  result = []
  length = len(P)
  for i in range(length):
    temp = (S[P[i]:Q[i]+1])
    if "A" in temp:
      result.append(1)
    elif "C" in temp:
      result.append(2)
    elif "G" in temp:
      result.append(3)
    elif "T" in temp:
      result.append(4)
  return result
*/
```

MinAvgTwoSlice

MINAVGTWOSLICE

Find the minimal average of any slice containing at least two elements.

A non-empty array A consisting of N integers is given. A pair of integers (P, Q), such that 0 ≤ P < Q < N, is called a slice of array A (notice that the slice contains at least two elements). The average of a slice (P, Q) is the sum of A[P] + A[P + 1] + ... + A[Q] divided by the length of the slice. To be precise, the average equals (A[P] + A[P + 1] + ... + A[Q]) / (Q − P + 1).

For example, array A such that:

```
A[0] = 4
A[1] = 2
A[2] = 2
A[3] = 5
A[4] = 1
A[5] = 5
A[6] = 8
```

contains the following example slices:

slice (1, 2), whose average is (2 + 2) / 2 = 2; slice (3, 4), whose average is (5 + 1) / 2 = 3; slice (1, 4), whose average is (2 + 2 + 5 + 1) / 4 = 2.5. The goal is to find the starting position of a slice whose average is minimal.

Write a function:

func Solution(A []int) int

that, given a non-empty array A consisting of N integers, returns the starting position of the slice with the minimal average. If there is more than one slice with a minimal average, you should return the smallest starting position of such a slice.

For example, given array A such that:

```
A[0] = 4
A[1] = 2
A[2] = 2
A[3] = 5
A[4] = 1
A[5] = 5
A[6] = 8
```

the function should return 1, as explained above.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [2..100,000]; each element of array A is an integer within the range [−10,000..10,000].
Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□array□index□□□□□□ ,□□□start□ index

□□□□

□□avg□slice(n,m)□m-n+1□□□2□□3□□□□□□□□avg□slice□2□□□3□□□□□ □□□□□□□ `0<=n<m<N` □□□□□□2□□□3□□□□□□□□□□□□□length=3□□□□□□□□□□□□2□slice□length=2□□□□□□□□□□□□□□□□□□□□□□□“□□”□□□□avg□□□□□"□□"□□□□□□□□□□□□2□slice□□□□□□□□□□□N>=4□□□□□□□□□□□□□2□slice

□□

- https://app.codility.com/programmers/lessons/5-prefix_sums/min_avg_two_slice/
- https://blog.csdn.net/dear0607/article/details/42581149

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0005.Prefix-Sums/MinAvgTwoSlice/MinAvgTwoSlice.go

```go
package MinAvgTwoSlice

import "math"
```

```go
func Solution(A []int) int {

    min := math.MaxFloat64
    minIndex := -1
    for i := 0; i < len(A)-1; i++ {
        // 2□□□□
        if i+1 < len(A) {
            tmp := (float64(A[i]) + float64(A[i+1])) / float64(2)
            if tmp < min {
                min = tmp
                minIndex = i
            }
        }
        // 3□□□□
        if i+2 < len(A) {
            tmp := (float64(A[i]) + float64(A[i+1]) + float64(A[i+2])) / float64(3)
            if tmp < min {
                min = tmp
                minIndex = i
            }
        }
    }

    return minIndex
}
```

PassingCars

PASSINGCARS

A non-empty array A consisting of N integers is given. The consecutive(□□) elements of array A represent consecutive cars on a road.

Array A contains only 0s and/or 1s:

0 represents a car traveling east, 1 represents a car traveling west. The goal is to count passing cars. We say that a pair of cars (P, Q), where 0 ≤ P < Q < N, is passing when P is traveling to the east and Q is traveling to the west.

For example, consider array A such that:

A[0] = 0 // no.0 car trave to east A[1] = 1 // no.1 car trave to west A[2] = 0 // no.2 car trave to east A[3] = 1 // no.3 car trave to west A[4] = 1 // no.4 car trave to west We have five pairs of passing cars: (0, 1), (0, 3), (0, 4), (2, 3), (2, 4).

Write a function:

func Solution(A []int) int

that, given a non-empty array A of N integers, returns the number of pairs of passing cars.

The function should return −1 if the number of pairs of passing cars exceeds 1,000,000,000.

For example, given:

A[0] = 0 A[1] = 1 A[2] = 0 A[3] = 1 A[4] = 1 the function should return 5, as explained above.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [1..100,000]; each element of array A is an integer that can have one of the following values: 0, 1.

Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□□□□□□A[]□index, 0 □□□□, 1□□□□, □□□□□□□□□□□□□□. □□□□□□□□□□□

□□□□

□□□□□□□□□, □□□□□□□□□. □□□□□□□□□+1. □□□□0□□□□□□□□0□□□□□□. □□2□□□□□0□□□□□□ 1□□□□□□□□□0□□□. □□3□4□□□0□2□□□□□. □□□1+2*2=5

□□

https://app.codility.com/programmers/lessons/5-prefix_sums/passing_cars/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0005.Prefix-Sums/PassingCars/PassingCars.go

```go
package PassingCars

func Solution(A []int) int {
    addBase, result := 0, 0
    for i := 0; i < len(A); i++ {
        if A[i] == 0 {
            addBase++
        } else {
            result += addBase
        }
    }
    if result > 1000000000 {
        return -1
    }
    return result
}
```

5.1.6 0006.Sorting

Distinct

DISTINCT

Compute number of distinct values in an array.

Write a function

func Solution(A []int) int

that, given an array A consisting of N integers, returns the number of distinct values in array A.

For example, given array A consisting of six elements such that:

A[0] = 2 A[1] = 1 A[2] = 1 A[3] = 2 A[4] = 3 A[5] = 1 the function should return 3, because there are 3 distinct values appearing in array A, namely 1, 2 and 3.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [0..100,000]; each element of array A is an integer within the range [−1,000,000..1,000,000].
Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□□□□□□□□□□

□□□□

1. □□□: □□□, □□□□□□□□□□□□□□□□□□. □□□□□□+1
2. □□□: □□□□set. □□□□set□□□

□□

https://app.codility.com/programmers/lessons/6-sorting/distinct/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0006.Sorting/Distinct/Distinct.go

```go
package Distinct

import "sort"

func Solution(A []int) int {
    sort.Ints(A)
    if len(A) == 0 {
        return 0
    }
    result := 1
    for i := 1; i < len(A); i++ {
        if A[i] != A[i-1] {
            result++
        }
    }
    return result
}

func SolutionSet(A []int) int {
    set := make(map[int]struct{})
    var void struct{}

    for i := 0; i < len(A); i++ {
        if _, ok := set[A[i]]; !ok {
            set[A[i]] = void
        }
    }
    return len(set)
}
```

MaxProductOfThree

MAXPRODUCTOFTHREE

Maximize A[P] * A[Q] * A[R] for any triplet (P, Q, R).

A non-empty array A consisting of N integers is given. The product of triplet (P, Q, R) equates to A[P] * A[Q] * A[R] (0 ≤ P < Q < R < N).

For example, array A such that:

A[0] = -3 A[1] = 1 A[2] = 2 A[3] = -2 A[4] = 5 A[5] = 6 contains the following example triplets:

(0, 1, 2), product is −3 * 1 * 2 = −6 (1, 2, 4), product is 1 * 2 * 5 = 10 (2, 4, 5), product is 2 * 5 * 6 = 60 Your goal is to find the maximal product of any triplet.

Write a function:

func Solution(A []int) int

that, given a non-empty array A, returns the value of the maximal product of any triplet.

For example, given array A such that:

A[0] = -3 A[1] = 1 A[2] = 2 A[3] = -2 A[4] = 5 A[5] = 6 the function should return 60, as the product of triplet (2, 4, 5) is maximal.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [3..100,000]; each element of array A is an integer within the range [−1,000..1,000]. Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□□□□□A□□□□□□□□□□□□□□□□

□□□□

□□□.□□□□ □□□□□*□□□□□□□□□□ □ □□□□□□□□□□ □□□□

□□

https://app.codility.com/programmers/lessons/6-sorting/max_product_of_three/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0006.Sorting/MaxProductOfThree/MaxProductOfThree.go

```go
package MaxProductOfThree

import "sort"

func max(a, b int) int {
    if a > b {
        return a
    }
    return b
}

func Solution(A []int) int {
    sort.Ints(A)
    aLen := len(A)
    return max(A[0]*A[1]*A[aLen-1], A[aLen-1]*A[aLen-2]*A[aLen-3])
}
```

NumberOfDiscIntersections

NUMBEROFDISCINTERSECTIONS

Compute the number of intersections(□□) in a sequence of discs(□□).

We draw N discs on a plane. The discs are numbered from 0 to N − 1. An array A of N non-negative integers, specifying the radiuses(□□) of the discs, is given. The J-th disc is drawn with its center at (J, 0) and radius A[J].

We say that the J-th disc and K-th disc intersect if J ≠ K and the J-th and K-th discs have at least one common point (assuming that the discs contain their borders).

The figure below shows discs drawn for N = 6 and A as follows:

A[0] = 1 A[1] = 5 A[2] = 2 A[3] = 1 A[4] = 4 A[5] = 0

There are eleven (unordered) pairs of discs that intersect, namely:

- discs 1 and 4 intersect, and both intersect with all the other discs;
- disc 2 also intersects with discs 0 and 3. Write a function:

func Solution(A []int) int

that, given an array A describing N discs as explained above, returns the number of (unordered) pairs of intersecting discs. The function should return −1 if the number of intersecting pairs exceeds 10,000,000.

Given array A shown above, the function should return 11, as explained above.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [0..100,000]; each element of array A is an integer within the range [0..2,147,483,647].
Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

A[0] = 1, □□□(0,0)□□□□□□□□□□□1□□. □□□□□□□□□

□□□□

- □□□: □□□i□j□□□□□□□□□□□□□□□□ $A[i] + A[j] >= j - i, \quad i \neq j$

□□SolutionDirect. □□□□□□□O(n^2) * □□□ □□□□□□□□□□□□□□□□□2□□□□□□ □□□□□□□A□□A[i]+i□□□□□i_limit□□□j-A[j]□□□□□j_limit□□□□□□□□i_limit□□□□S□□ □□□□□□□□□□□□j_limit□□□□S□□□□□□□□□□□□□□□□□□O(N)□□□□□□□□□□□□□□□O(LogN)□□□□□□□□□□□□□□O(N*logN)□□□Solution□

$$A[i] + A[j] >= j - i, \quad i \neq j$$

$$\Rightarrow A[i] + i >= j - A[j], \quad i \neq j$$

□□

- https://app.codility.com/programmers/lessons/6-sorting/number_of_disc_intersections/
- https://github.com/Anfany/Codility-Lessons-By-Python3/blob/master/L6_Sorting/6.4%20NumberOfDiscIntersections.md
- https://rafal.io/posts/codility-intersecting-discs.html
- https://github.com/tmpapageorgiou/algorithm/blob/master/number_disc_intersections.py

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0006.Sorting/NumberOfDiscIntersections/NumberOfDiscIntersections.go

```go
package NumberOfDiscIntersections

import "sort"

func max(a, b int) int {
    if a > b {
        return a
    }
    return b
}

func min(a, b int) int {
    if a > b {
        return b
    }
    return a
}

// □□□□ O(n^2)
func SolutionDirect(A []int) int {
    count := 0
    for indexI, valueI := range A {
        tmpArr := A[indexI+1:]
        for indexJ, valueJ := range tmpArr {
            if valueI+valueJ >= indexJ+indexI+1-indexI {
                count++
            }
        }
    }

    return count
}

// □□□□ O(nlogn) or O(n)
// TODO: □□□
func Solution(A []int) int {
    iLimit := make([]int, len(A)) // □
    jLimit := make([]int, len(A)) // □
    result := 0

    for i := 0; i < len(A); i += 1 {
        iLimit[i] = i - A[i]
        jLimit[i] = i + A[i]
    }
    // □□iLimit□□□□□□□□□□□□□□□□□□□□□jLimit□□□□□□
    sort.Ints(iLimit)
    sort.Ints(jLimit)
    for idx, _ := range iLimit {
        end := jLimit[idx]

        // Binary search for index of element of the rightmost value less than to the interval-end
        count := sort.Search(len(iLimit), func(i int) bool {
            return iLimit[i] > end
        })

        // □□i=j□□A[i]+i □□□□□j-A[j],□□□□□□□□□□□□□□1□
        // □□idx□□□□□A□□□B□□□□□□□1□□□□B□□□A□□□□□□□1□□
        count = count - idx - 1
        result += count

        if result > 10000000 {
            return -1
        }

    }

    return result
}
```

Triangle

TRIANGLE

Determine whether a triangle can be built from a given set of edges.

An array A consisting of N integers is given. A triplet (P, Q, R) is triangular if 0 ≤ P < Q < R < N and:

A[P] + A[Q] > A[R], A[Q] + A[R] > A[P], A[R] + A[P] > A[Q]. For example, consider array A such that:

A[0] = 10 A[1] = 2 A[2] = 5 A[3] = 1 A[4] = 8 A[5] = 20 Triplet (0, 2, 4) is triangular.

Write a function:

func Solution(A []int) int

that, given an array A consisting of N integers, returns 1 if there exists a triangular triplet for this array and returns 0 otherwise.

For example, given array A such that:

A[0] = 10 A[1] = 2 A[2] = 5 A[3] = 1 A[4] = 8 A[5] = 20 the function should return 1, as explained above. Given array A such that:

A[0] = 10 A[1] = 50 A[2] = 5 A[3] = 1 the function should return 0.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [0..100,000]; each element of array A is an integer within the range [−2,147,483,648..2,147,483,647].

Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□□array□□□□□□□□□□□□,□□□1, □□□□0

□□□□

□□□□□□, □□□□□□□□□□□□□□, □□□□□□□□□. □□□□□□0□□□

□□

https://app.codility.com/programmers/lessons/6-sorting/triangle/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0006.Sorting/Triangle/Triangle.go

```go
package Triangle

import "sort"

func Solution(A []int) int {
    if len(A) < 3 {
        return 0
    }
    sort.Ints(A)
    for i := 0; i < len(A)-2; i++ {
        if A[i+2] < A[i+1]+A[i] {
            return 1
        }
    }
    return 0
}
```

5.1.7 0007.Stacks and Queues

Brackets

BRACKETS

Determine whether a given string of parentheses (multiple types) is properly nested.

A string S consisting of N characters is considered to be properly nested if any of the following conditions is true:

S is empty; S has the form "(U)" or "[U]" or "{U}" where U is a properly nested string; S has the form "VW" where V and W are properly nested strings. For example, the string "{[()()]}" is properly nested but "([)()]" is not.

Write a function:

func Solution(S string) int

that, given a string S consisting of N characters, returns 1 if S is properly nested and 0 otherwise.

For example, given S = "{[()()]}", the function should return 1 and given S = "([)()]", the function should return 0, as explained above.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [0..200,000]; string S consists only of the following characters: "(", "{", "[", "]", "}" and/or ")".
Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□□□, □□□□□1 □□□□0.

□□□□

□□□□□□□stack. □□□□□□□stack pop□□□□□pop□□□□□□□□□□□□□□.

□□

https://app.codility.com/programmers/lessons/7-stacks_and_queues/brackets/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0007.Stacks-and-Queues/Brackets/Brackets.go

```go
package Brackets

import (
    "LeetcodeGolang/Utility/structures"
)

func Solution(S string) int {
    if len(S) == 0 {
        return 1
    }
    if len(S)%2 != 0 {
        return 0
    }

    BracketMap := map[string]string{
        ")": "(",
        "]": "[",
        "}": "{",
    }

    stack := structures.NewArrayStack()
    for _, v := range S {
        val := string(v)
        if val == "(" || val == "[" || val == "{" {
            stack.Push(val)
        } else if val == ")" || val == "]" || val == "}" {
            if stack.IsEmpty() {
                return 0
            }

            topVal := stack.Top()
            if topVal == BracketMap[val] {
                stack.Pop()
```

```
            } else {
                // □□□□□□□□□
                return 0
            }
        }
    }
    if stack.IsEmpty() {
        return 1
    } else {
        return 0
    }
}
```

Fish

FISH

N voracious fish are moving along a river. Calculate how many fish are alive.

You are given two non-empty arrays A and B consisting of N integers. Arrays A and B represent N voracious fish in a river, ordered downstream along the flow of the river.

The fish are numbered from 0 to N − 1. If P and Q are two fish and P < Q, then fish P is initially upstream of fish Q. Initially, each fish has a unique position.

Fish number P is represented by A[P] and B[P]. Array A contains the sizes of the fish. All its elements are unique. Array B contains the directions of the fish. It contains only 0s and/or 1s, where:

0 represents a fish flowing upstream, 1 represents a fish flowing downstream. If two fish move in opposite directions and there are no other (living) fish between them, they will eventually meet each other. Then only one fish can stay alive − the larger fish eats the smaller one. More precisely, we say that two fish P and Q meet each other when P < Q, B[P] = 1 and B[Q] = 0, and there are no living fish between them. After they meet:

If A[P] > A[Q] then P eats Q, and P will still be flowing downstream, If A[Q] > A[P] then Q eats P, and Q will still be flowing upstream. We assume that all the fish are flowing at the same speed. That is, fish moving in the same direction never meet. The goal is to calculate the number of fish that will stay alive.

For example, consider arrays A and B such that:

A[0] = 4 B[0] = 0 A[1] = 3 B[1] = 1 A[2] = 2 B[2] = 0 A[3] = 1 B[3] = 0 A[4] = 5 B[4] = 0 Initially all the fish are alive and all except fish number 1 are moving upstream. Fish number 1 meets fish number 2 and eats it, then it meets fish number 3 and eats it too. Finally, it meets fish number 4 and is eaten by it. The remaining two fish, number 0 and 4, never meet and therefore stay alive.

Write a function:

func Solution(A []int, B []int) int

that, given two non-empty arrays A and B consisting of N integers, returns the number of fish that will stay alive.

For example, given the arrays shown above, the function should return 2, as explained above.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [1..100,000]; each element of array A is an integer within the range [0..1,000,000,000]; each element of array B is an integer that can have one of the following values: 0, 1; the elements of A are all distinct.

Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□□□□□□□□□□□□□□ A: □□□□□, B: □□□□□. □□□□□□□□□□□□□□□. □□□□□□□□□

□□□□

□ B □□□, □□□1 □□stack. □□□□□□□. □□□□□□□□. □□□□□□□□□□□□. □□□+1 □□□□0□stack□□, □□□□□□□□□□□□□□□++

□□

https://app.codility.com/programmers/lessons/7-stacks_and_queues/fish/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0007.Stacks-and-Queues/Fish/Fish.go

```go
package Fish

import "LeetcodeGolang/Utility/structures"

func Solution(A []int, B []int) int {
    stack := structures.NewArrayStack()
```

```
    aliveFish := 0
    for idx, val := range B {
        if val == 1 {
            stack.Push(A[idx])
        } else {
            // □□□□□
            for !stack.IsEmpty() {
                if stack.Top().(int) < A[idx] {
                    // stack□□□□□□□□□, stack□□□
                    stack.Pop()
                } else {
                    break
                }
            }
            if stack.IsEmpty() {
                aliveFish++
            }
        }
    }
    return aliveFish + stack.Size()
}
```

Nesting

NESTING

Determine whether a given string of parentheses (single type) is properly nested.

A string S consisting of N characters is called properly nested if:

S is empty; S has the form "(U)" where U is a properly nested string; S has the form "VW" where V and W are properly nested strings. For example, string "(()(())())" is properly nested but string "())" isn't.

Write a function:

func Solution(S string) int

that, given a string S consisting of N characters, returns 1 if string S is properly nested and 0 otherwise.

For example, given S = "(()(())())", the function should return 1 and given S = "())", the function should return 0, as explained above.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [0..1,000,000]; string S consists only of the characters "(" and/or ")". Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□□□

□□□□

□Bracket□□

□□

https://app.codility.com/programmers/lessons/7-stacks_and_queues/nesting/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0007.Stacks-and-Queues/Nesting/Nesting.go

```go
package Nesting

import "LeetcodeGolang/Utility/structures"

func Solution(S string) int {
    if len(S) == 0 {
        return 1
    }
    if len(S)%2 != 0 {
        return 0
    }

    stack := structures.NewArrayStack()
    for _, v := range S {
        val := string(v)
        if val == "(" {
            stack.Push(val)
        } else if val == ")" {
            stack.Pop()
        }
    }
    if stack.IsEmpty() {
        return 1
    } else {
        return 0
    }
}
```

StoneWall

STONEWALL

Cover "Manhattan skyline" using the minimum number of rectangles.

You are going to build a stone wall. The wall should be straight and N meters long, and its thickness should be constant; however, it should have different heights in different places. The height of the wall is specified by an array H of N positive integers. H[I] is the height of the wall from I to I+1 meters to the right of its left end. In particular, H[0] is the height of the wall's left end and H[N−1] is the height of the wall's right end.

The wall should be built of cuboid (□□□) stone blocks (that is, all sides of such blocks are rectangular). Your task is to compute the minimum number of blocks needed to build the wall.

Write a function:

func Solution(H []int) int

that, given an array H of N positive integers specifying the height of the wall, returns the minimum number of blocks needed to build it.

For example, given array H containing N = 9 integers:

H[0] = 8 H[1] = 8 H[2] = 5 H[3] = 7 H[4] = 9 H[5] = 8 H[6] = 7 H[7] = 4 H[8] = 8 the function should return 7. The figure shows one possible arrangement of seven blocks.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [1..100,000]; each element of array H is an integer within the range [1..1,000,000,000].
Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□

□□□□□□□□□□□□□ H □□□ □□□□□□N□□□. □□□□, □□□□□□□□□□. H[I]□□□□ I □ I+1 □□□□□. H[0]□□□□□□□1□□□□□ H[N-1]□□□N-1□□□□□□□□

□□□

□□□□array, □□□□□□□□□□□□□,□□stack□, □□□□□□□. □□□+1 □□□□□□□□□□,□□□□□pop□□.□□stack □□□□□□□□□□□□□□□□

□□

- https://app.codility.com/programmers/lessons/7-stacks_and_queues/stone_wall/
- https://github.com/Anfany/Codility-Lessons-By-Python3/blob/master/L7_Stacks%20and%20Queues/7.4%20StoneWall.md

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0007.Stacks-and-Queues/StoneWall/StoneWall.go

```go
package StoneWall

import "LeetcodeGolang/Utility/structures"

func Solution(H []int) int {
    stack := structures.NewArrayStack()
    result := 0
    for _, v := range H {
        for !stack.IsEmpty() && stack.Top().(int) > v {
            stack.Pop()
        }
        if stack.IsEmpty() || stack.Top().(int) < v {
            stack.Push(v)
            result++
```

```
        }
    }
    return result
}
```

5.1.8 0008.Leader

Dominator

DOMINATOR

Find an index of an array such that its value occurs at more than half of indices in the array.

An array A consisting of N integers is given. The dominator of array A is the value that occurs in more than half of the elements of A.

For example, consider array A such that

A[0] = 3 A[1] = 4 A[2] = 3 A[3] = 2 A[4] = 3 A[5] = -1 A[6] = 3 A[7] = 3 The dominator of A is 3 because it occurs in 5 out of 8 elements of A (namely in those with indices 0, 2, 4, 6 and 7) and 5 is more than a half of 8.

Write a function

func Solution(A []int) int

that, given an array A consisting of N integers, returns index of any element of array A in which the dominator of A occurs. The function should return −1 if array A does not have a dominator.

For example, given array A such that

A[0] = 3 A[1] = 4 A[2] = 3 A[3] = 2 A[4] = 3 A[5] = -1 A[6] = 3 A[7] = 3 the function may return 0, 2, 4, 6 or 7, as explained above.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [0..100,000]; each element of array A is an integer within the range [−2,147,483,648..2,147,483,647]. Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□Array□□□□□. A□□□□□3□□□□□□□□A□8□□□□□5□□□□(index□0□2□4□6□7). □5□8□□□□□ □□□□ 0,2,4,6,7□□□□□

□□□□

□map□□□□□□□□□. □□□□□□□□□□□□□□□□. □□□□□□□□□□□index, □□□□-1

□□

https://app.codility.com/programmers/lessons/8-leader/dominator/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0008.Leader/Dominator/Dominator.go

```go
package Dominator

import (
    "math"
)

func Solution(A []int) int {
    mapInt := make(map[int]int, len(A))

    for _, v := range A {
        if _, ok := mapInt[v]; !ok {
            mapInt[v] = 1
        } else {
            mapInt[v]++
        }
    }

    maxCount := 0
    maxVal := 0
    for k, v := range mapInt {
        if v > maxCount {
            maxCount = v
            maxVal = k
```

```
        }
    }
    minIndex := -1
    for k, v := range A {
        if v == maxVal {
            minIndex = k
            break
        }
    }

    if maxCount > int(math.Floor(float64(len(A))/2.0)) {
        return minIndex
    } else {
        return -1
    }
}
```

EquiLeader

EQUILEADER

Find the index S such that the leaders of the sequences A[0], A[1], ..., A[S] and A[S + 1], A[S + 2], ..., A[N - 1] are the same.

A non-empty array A consisting of N integers is given.

The leader of this array is the value that occurs in more than half of the elements of A.

An equi leader is an index S such that 0 ≤ S < N − 1 and two sequences A[0], A[1], ..., A[S] and A[S + 1], A[S + 2], ..., A[N − 1] have leaders of the same value.

For example, given array A such that:

```
A[0] = 4
A[1] = 3
A[2] = 4
A[3] = 4
A[4] = 4
A[5] = 2
```

we can find two equi leaders:

- 0, because sequences: (4) and (3, 4, 4, 4, 2) have the same leader, whose value is 4.
- 2, because sequences: (4, 3, 4) and (4, 4, 2) have the same leader, whose value is 4. The goal is to count the number of equi leaders.

Write a function:

func Solution(A []int) int

that, given a non-empty array A consisting of N integers, returns the number of equi leaders.

For example, given:

```
A[0] = 4
A[1] = 3
A[2] = 4
A[3] = 4
A[4] = 4
A[5] = 2
```

the function should return 2, as explained above.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [1..100,000]; each element of array A is an integer within the range [−1,000,000,000..1,000,000,000]. Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□

□□□□□, □□□□□□□□□□□□□□, □□□□□□□□□□□□□leader, □□□□□□EquiLeader, □□□□□□□□□□EquiLeader□□□n. □□□, □□□□□□□□□□□□□leader, □□□□□□□□□□□leader.

□□□

□□□□□□□□□leader, □□□□□□□□. □□□□□□□□□□□□□□□□□□□□□□□□□leader□□□□□s, □s□n-s□□□□□leader□□□□□□□□□□leader.

□□

- https://app.codility.com/programmers/lessons/8-leader/equi_leader/
- https://github.com/Anfany/Codility-Lessons-By-Python3/blob/master/L8_Leader/8.1%20EquiLeader.md

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0008.Leader/EquiLeader/EquiLeader.go

```go
package EquiLeader

func Solution(A []int) int {
    leaderDict := make(map[int]int)
    for i := 0; i < len(A); i++ {
        if _, ok := leaderDict[A[i]]; ok {
            leaderDict[A[i]]++
        } else {
            leaderDict[A[i]] = 1
        }
    }

    leader := 0
    times := 0
    for k, v := range leaderDict {
        if v > times {
            times = v
            leader = k
        }
    }

    equiCount := 0
    count := 0 // ▫▫▫▫▫▫▫▫▫

    for index, v := range A {
        if v == leader {
            count++
        }
        if count > (index+1)/2 && (times-count) > (len(A)-(index+1))/2 {
            equiCount++
        }

    }

    return equiCount
}
```

5.1.9 0009.Maximum Slice Problem

MaxDoubleSliceSum

MAXDOUBLESLICESUM

Find the maximal sum of any double slice.

A non-empty array A consisting of N integers is given.

A triplet (X, Y, Z), such that 0 ≤ X < Y < Z < N, is called a double slice.

The sum of double slice (X, Y, Z) is the total of A[X + 1] + A[X + 2] + ... + A[Y − 1] + A[Y + 1] + A[Y + 2] + ... + A[Z − 1].

For example, array A such that:

```
A[0] = 3
A[1] = 2
A[2] = 6
A[3] = -1
A[4] = 4
A[5] = 5
A[6] = -1
A[7] = 2
```

contains the following example double slices:

double slice (0, 3, 6), sum is 2 + 6 + 4 + 5 = 17, double slice (0, 3, 7), sum is 2 + 6 + 4 + 5 − 1 = 16, double slice (3, 4, 5), sum is 0. The goal is to find the maximal sum of any double slice.

Write a function:

func Solution(A []int) int

that, given a non-empty array A consisting of N integers, returns the maximal sum of any double slice.

For example, given:

```
A[0] = 3
A[1] = 2
A[2] = 6
A[3] = -1
A[4] = 4
A[5] = 5
A[6] = -1
A[7] = 2
```

the function should return 17, because no double slice of array A has a sum of greater than 17.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [3..100,000]; each element of array A is an integer within the range [−10,000..10,000].
Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□

A[X+1]□A[Y-1] + A[Y+1]□A[Z-1] □□□

□□□

□□□array, □□□□□index□□□□□□□□□□, □□□□□array□□□□□index□□□□□□□□□□□ □□□□□□□□□□□□□.□□□□□□□□□□ □□□□array□□□□□□□□

□□

- https://app.codility.com/programmers/lessons/9-maximum_slice_problem/max_double_slice_sum/
- https://github.com/Anfany/Codility-Lessons-By-Python3/blob/master/L9_Maximum%20Slice%20Problem/9.3%20%20MaxDoubleSliceSum.md

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0009.Maximum-Slice-Problem/MaxDoubleSliceSum/MaxDoubleSliceSum.go

```go
package MaxDoubleSliceSum

import (
    "math"
)

func Solution(A []int) int {
    if len(A) < 4 {
        return 0
    }
    N := len(A) - 2
    forwardSum := make([]int, N)
    reverseSum := make([]int, N)

    //  0 ≤ X < Y < Z < N,
    //  A[X + 1] + A[X + 2] + ... + A[Y − 1] + A[Y + 1] + A[Y + 2] + ... + A[Z − 1].
    //          A : [ 3,  2, 6, -1,  4,  5, -1, 2]
    // forwardSum : [ 0,  2, 8,  7, 11, 16]
    // reverseSum : [14,  8, 9,  5,  0,  0]
    for i := 0; i < N-1; i++ {
        forwardVal := A[i+1]
        reverseVal := A[N-i]

        forwardSum[i+1] = int(math.Max(0, float64(forwardVal)+float64(forwardSum[i])))
        reverseSum[N-i-2] = int(math.Max(0, float64(reverseVal)+float64(reverseSum[N-i-1])))
    }

    combineMax := math.MinInt64
    for i := 0; i < N; i++ {
        combineMax = int(math.Max(float64(combineMax), float64(forwardSum[i])+float64(reverseSum[i])))
    }

    return combineMax
}
```

MaxProfit

MAXPROFIT

Given a log of stock prices compute the maximum possible earning.

An array A consisting of N integers is given. It contains daily prices of a stock share for a period of N consecutive days. If a single share was bought on day P and sold on day Q, where 0 ≤ P ≤ Q < N, then the profit of such transaction is equal to A[Q] − A[P], provided that A[Q] ≥ A[P]. Otherwise, the transaction brings loss of A[P] − A[Q].

For example, consider the following array A consisting of six elements such that:

A[0] = 23171 A[1] = 21011 A[2] = 21123 A[3] = 21366 A[4] = 21013 A[5] = 21367 If a share was bought on day 0 and sold on day 2, a loss of 2048 would occur because A[2] − A[0] = 21123 − 23171 = −2048. If a share was bought on day 4 and sold on day 5, a profit of 354 would occur because A[5] − A[4] = 21367 − 21013 = 354. Maximum possible profit was 356. It would occur if a share was bought on day 1 and sold on day 5.

Write a function,

func Solution(A []int) int

that, given an array A consisting of N integers containing daily prices of a stock share for a period of N consecutive days, returns the maximum possible profit from one transaction during this period. The function should return 0 if it was impossible to gain any profit.

For example, given array A consisting of six elements such that:

A[0] = 23171 A[1] = 21011 A[2] = 21123 A[3] = 21366 A[4] = 21013 A[5] = 21367 the function should return 356, as explained above.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [0..400,000]; each element of array A is an integer within the range [0..200,000].

Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□

□□□□□□□□□□□□□□

□□□

□□□□array, □□□□□□□□□□, □□□□□□□□□□□□-□□□□□□, □□□□□□

□□

https://app.codility.com/programmers/lessons/9-maximum_slice_problem/max_profit/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0009.Maximum-Slice-Problem/MaxProfit/MaxProfit.go

```go
package MaxProfit

import (
    "math"
)

func Solution(A []int) int {
    minBuyPrice := math.MaxFloat64
    maxProfit := 0.0

    for _, v := range A {
        minBuyPrice = math.Min(minBuyPrice, float64(v))
        maxProfit = math.Max(maxProfit, float64(v)-minBuyPrice)
    }

    return int(maxProfit)
}
```

MaxSliceSum

MAXSLICESUM

Find a maximum sum of a compact subsequence of array elements.

A non-empty array A consisting of N integers is given. A pair of integers (P, Q), such that 0 ≤ P ≤ Q < N, is called a slice of array A. The sum of a slice (P, Q) is the total of A[P] + A[P+1] + ... + A[Q].

Write a function:

func Solution(A []int) int

that, given an array A consisting of N integers, returns the maximum sum of any slice of A.

For example, given array A such that:

A[0] = 3 A[1] = 2 A[2] = -6 A[3] = 4 A[4] = 0 the function should return 5 because:

(3, 4) is a slice of A that has sum 4, (2, 2) is a slice of A that has sum −6, (0, 1) is a slice of A that has sum 5, no other slice of A has sum greater than (0, 1). Write an efficient algorithm for the following assumptions:

N is an integer within the range [1..1,000,000]; each element of array A is an integer within the range [−1,000,000..1,000,000]; the result will be an integer within the range [−2,147,483,648..2,147,483,647].

Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□□□□□□□□□□□

□□□□

□□□□□1, □□□□□ □□□□□□□□□□□□□□□□, □□□□. □□□□□□□□□□□□□□□□□□□□□□□□,□□□□

□□

https://app.codility.com/programmers/lessons/9-maximum_slice_problem/max_slice_sum/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0009.Maximum-Slice-Problem/MaxSliceSum/MaxSliceSum.go

```go
package MaxSliceSum

import (
    "math"
)

func Solution(A []int) int {
    if len(A) == 1 {
        return A[0]
    }
    result := math.MinInt64
    sum := math.MinInt64
    for i := 0; i < len(A); i++ {
        sum = int(math.Max(float64(A[i]), float64(A[i])+float64(sum)))
        result = int(math.Max(float64(sum), float64(result)))
    }
    return result
}
```

5.1.10 0010.Prime And Composite Numbers

CountFactors

COUNTFACTORS

Count factors of given number n.

A positive integer D is a factor of a positive integer N if there exists an integer M such that N = D * M.

For example, 6 is a factor of 24, because M = 4 satisfies the above condition (24 = 6 * 4).

Write a function:

func Solution(N int) int

that, given a positive integer N, returns the number of its factors.

For example, given N = 24, the function should return 8, because 24 has 8 factors, namely 1, 2, 3, 4, 6, 8, 12, 24. There are no other factors of 24.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [1..2,147,483,647].

Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□□□□□□□□□

□□□□

□□□□□□□□□□□□, □□□□□□2□□□

□□

https://app.codility.com/programmers/lessons/10-prime_and_composite_numbers/count_factors/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0010.Prime-And-Composite-Numbers/CountFactors/CountFactors.go

```go
package CountFactors

import (
    "math"
)

func Solution(N int) int {
    result := 0
    for i := 1; i <= int(math.Pow(float64(N), 0.5)); i++ {
        if N%i == 0 {
            if i*i == N {
                // fmt.Println("+1 : ", i)
                result++
            } else {
                // fmt.Println("+2 : ", i)
                result += 2
            }
        }
    }
    return result
}
```

Flags

FLAGS

Find the maximum number of flags that can be set on mountain peaks.

A non-empty array A consisting of N integers is given.

A peak is an array element which is larger than its neighbours. More precisely, it is an index P such that $0 < P < N - 1$ and $A[P - 1] < A[P] > A[P + 1]$.

For example, the following array A:

```
A[0] = 1
A[1] = 5 //peaks
A[2] = 3
A[3] = 4 //peaks
A[4] = 3
A[5] = 4 //peaks
A[6] = 1
A[7] = 2
A[8] = 3
A[9] = 4
A[10] = 6 //peaks
A[11] = 2
```

has exactly four peaks: elements 1, 3, 5 and 10.

You are going on a trip to a range of mountains whose relative heights are represented by array A, as shown in a figure below. You have to choose how many flags you should take with you. The goal is to set the maximum number of flags on the peaks, according to certain rules.

Flags can only be set on peaks. What's more, if you take K flags, then the distance between any two flags should be greater than or equal to K. The distance between indices P and Q is the absolute value $|P - Q|$.

For example, given the mountain range represented by array A, above, with N = 12, if you take:

two flags, you can set them on peaks 1 and 5; three flags, you can set them on peaks 1, 5 and 10; four flags, you can set only three flags, on peaks 1, 5 and 10. You can therefore set a maximum of three flags in this case.

Write a function:

func Solution(A []int) int

that, given a non-empty array A of N integers, returns the maximum number of flags that can be set on the peaks of the array.

For example, the following array A:

```
A[0] = 1
A[1] = 5
A[2] = 3
A[3] = 4
A[4] = 3
A[5] = 4
A[6] = 1
A[7] = 2
A[8] = 3
A[9] = 4
```

```
A[10] = 6
A[11] = 2
```

the function should return 3, as explained above.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [1..400,000]; each element of array A is an integer within the range [0..1,000,000,000].

Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□, □□□□K□□□, □□□□□□□□□□□□□□□□□□K

□□□□

□□□peak□□□□□□□□array□. □□peak array□□□□□□□□□□□□□□□□□□□□□□□□□K: K*(K-1)<=P[-1] - P[0]=dis, □□K□□□□□ int(sqrt(P[-1] - P[0]) +1). □□□K□□□□□□□, □□peak array, □□□□□□□□□□□□□□□□□□□array, □□□□array□□□□K□□□□□

□□

- https://app.codility.com/programmers/lessons/10-prime_and_composite_numbers/
- https://github.com/Anfany/Codility-Lessons-By-Python3/blob/master/L10_Prime%20and%20composite%20numbers/10.4%20Flags.md

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0010.Prime-And-Composite-Numbers/Flags/Flags.go

```go
package Flags

import (
    "math"
)

func Solution(A []int) int {
    var peaksFlag []int

    for i := 1; i < len(A)-1; i++ {
        if A[i] > A[i-1] && A[i] > A[i+1] {
            peaksFlag = append(peaksFlag, i)
        }
    }

    if len(peaksFlag) == 0 {
        return 0
    }
    if len(peaksFlag) == 1 {
        return 1
    }

    maxFlag := int(math.Pow(float64(peaksFlag[len(peaksFlag)-1]-peaksFlag[0]), 0.5) + 1)

    for i := maxFlag; i > 1; i-- {
        addressFlag := []int{peaksFlag[0]}
        for _, val := range peaksFlag[1:] {
            if val-addressFlag[len(addressFlag)-1] >= i {
                addressFlag = append(addressFlag, val)
                if len(addressFlag) >= i {
                    return i
                }
            }
        }
    }

    return 1
}
```

MinPerimeterRectangle

MINPERIMETERRECTANGLE

Find the minimal perimeter of any rectangle whose area equals N.

An integer N is given, representing the area of some rectangle.

The area of a rectangle whose sides are of length A and B is A * B, and the perimeter is 2 * (A + B).

The goal is to find the minimal perimeter of any rectangle whose area equals N. The sides of this rectangle should be only integers.

For example, given integer N = 30, rectangles of area 30 are:

(1, 30), with a perimeter of 62, (2, 15), with a perimeter of 34, (3, 10), with a perimeter of 26, (5, 6), with a perimeter of 22. Write a function:

func Solution(N int) int

that, given an integer N, returns the minimal perimeter of any rectangle whose area is exactly equal to N.

For example, given an integer N = 30, the function should return 22, as explained above.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [1..1,000,000,000].

Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□□□□□□N, □□□□□□N□□□□□

□□□□

□□□□N□□□□□□□□□□□,□□□□N□□□ □□□□□□□□□□□□□.□□□□□□□□□□□□□□□□□□□□

□□

- https://app.codility.com/programmers/lessons/10-prime_and_composite_numbers/min_perimeter_rectangle/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0010.Prime-And-Composite-Numbers/MinPerimeterRectangle/MinPerimeterRectangle.go

```go
package minperimeterrectangle

import (
    "math"
)

// O(sqrt(N))
func Solution(N int) int {
    if N <= 0 {
        return 0
    }

    min := math.MaxInt32
    for i := 1; i*i <= N; i++ {
        if N%i == 0 {
            perimeter := 2 * (i + N/i)
            min = int(math.Min(float64(min), float64(perimeter)))
        }

    }

    if min == math.MaxInt32 {
        return 0
    }
    return min
}

/*
O(N)
```

```go
Task Score 60%
Correctness 100%
Performance 20%
*/
func Solution1(N int) int {
    if N <= 0 {
        return 0
    }

    min := math.MaxInt32
    for i := 1; i <= N; i++ {
        if N%i == 0 && i*i <= N {
            perimeter := 2 * (i + N/i)
            min = int(math.Min(float64(min), float64(perimeter)))
        }
    }
    if min == math.MaxInt32 {
        return 0
    }
    return min
}

// O(sqrt(N))
func Solution2(N int) int {
    if N <= 0 {
        return 0
    }
    pairs := make(map[int]int)
    i := 1
    for i*i <= N {
        if N%i == 0 {
            pairs[i] = N / i
        }
        i++
    }

    min := math.MaxInt32
    for i, v := range pairs {
        perimeter := 2 * (i + v)
        min = int(math.Min(float64(min), float64(perimeter)))

    }

    if min == math.MaxInt32 {
        return 0
    }
    return min
}
```

Peaks

PEAKS

Divide an array into the maximum number of same-sized blocks, each of which should contain an index P such that A[P - 1] < A[P] > A[P + 1].

A non-empty array A consisting of N integers is given.

A peak is an array element which is larger than its neighbors. More precisely, it is an index P such that 0 < P < N − 1, A[P − 1] < A[P] and A[P] > A[P + 1].

For example, the following array A:

```
A[0] = 1
A[1] = 2
A[2] = 3
A[3] = 4
A[4] = 3
A[5] = 4
A[6] = 1
A[7] = 2
A[8] = 3
A[9] = 4
A[10] = 6
A[11] = 2
```

has exactly three peaks: 3, 5, 10.

We want to divide this array into blocks containing the same number of elements. More precisely, we want to choose a number K that will yield the following blocks:

A[0], A[1], ..., A[K − 1], A[K], A[K + 1], ..., A[2K − 1], ... A[N − K], A[N − K + 1], ..., A[N − 1]. What's more, every block should contain at least one peak. Notice that extreme elements of the blocks (for example A[K − 1] or A[K]) can also be peaks, but only if they have both neighbors (including one in an adjacent blocks).

The goal is to find the maximum number of blocks into which the array A can be divided.

Array A can be divided into blocks as follows:

one block (1, 2, 3, 4, 3, 4, 1, 2, 3, 4, 6, 2). This block contains three peaks. two blocks (1, 2, 3, 4, 3, 4) and (1, 2, 3, 4, 6, 2). Every block has a peak. three blocks (1, 2, 3, 4), (3, 4, 1, 2), (3, 4, 6, 2). Every block has a peak. Notice in particular that the first block (1, 2, 3, 4) has a peak at A[3], because A[2] < A[3] > A[4], even though A[4] is in the adjacent block. However, array A cannot be divided into four blocks, (1, 2, 3), (4, 3, 4), (1, 2, 3) and (4, 6, 2), because the (1, 2, 3) blocks do not contain a peak. Notice in particular that the (4, 3, 4) block contains two peaks: A[3] and A[5].

The maximum number of blocks that array A can be divided into is three.

Write a function:

func Solution(A []int) int

that, given a non-empty array A consisting of N integers, returns the maximum number of blocks into which A can be divided.

If A cannot be divided into some number of blocks, the function should return 0.

For example, given:

```
A[0] = 1
A[1] = 2
A[2] = 3
A[3] = 4
A[4] = 3
A[5] = 4
A[6] = 1
A[7] = 2
A[8] = 3
A[9] = 4
A[10] = 6
A[11] = 2
```

the function should return 3, as explained above.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [1..100,000]; each element of array A is an integer within the range [0..1,000,000,000].

Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□ array □□□□□□□□□, □□□□□□□□□□□peak.

□□□□

□□□□□peak□index □□peaks array. □□peaks□□□□□□□□, □ A □□□□, □□□□□□□□□□□□□peak

□□

- https://app.codility.com/programmers/lessons/10-prime_and_composite_numbers/peaks/
- https://www.martinkysel.com/codility-peaks-solution/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0010.Prime-And-Composite-Numbers/Peaks/Peaks.go

```go
package peaks

/*
expected worst-case time complexity is O(N*log(log(N)));
expected worst-case space complexity is O(N)
*/
func Solution(A []int) int {
    // □□□peaks
    peaks := []int{}
    for i := 1; i < len(A)-1; i++ {
        if A[i-1] < A[i] && A[i] > A[i+1] {
            peaks = append(peaks, i)
        }
    }

    if len(peaks) < 0 {
        return 0
    } else if len(peaks) == 1 {
        return 1
    }

    for size := len(peaks); size > 0; size-- {
        if len(A)%size == 0 {
            // □□□□□size
            blockSize := len(A) / size
            found := make(map[int]bool, size)
            foundCnt := 0
            for _, peak := range peaks {
                // □□□□□□□□□□□ peak
                blockNr := peak / blockSize
                if ok := found[blockNr]; !ok {
                    found[blockNr] = true
                    foundCnt++
                }
            }
            if foundCnt == size {
                return size
            }
        }
    }
    return 0
}

/*
def solution(A):
    peaks = []

    for idx in range(1, len(A)-1):
        if A[idx-1] < A[idx] > A[idx+1]:
            peaks.append(idx)

    if len(peaks) == 0:
        return 0

    for size in range(len(peaks), 0, -1):
        if len(A) % size == 0:
            block_size = len(A) // size
            found = [False] * size
            found_cnt = 0
            for peak in peaks:
```

```
                block_nr = peak//block_size
                if found[block_nr] == False:
                    found[block_nr] = True
                    found_cnt += 1

            if found_cnt == size:
                return size

    return 0
*/
```

5.1.11 0011.Sieve of Eratosthenes

CountNonDivisible

COUNTNONDIVISIBLE

Calculate the number of elements of an array that are not divisors(□□) of each element.

You are given an array A consisting of N integers.

For each number A[i] such that 0 ≤ i < N, we want to count the number of elements of the array that are not the divisors of A[i]. We say that these elements are non-divisors.

For example, consider integer N = 5 and array A such that:

```
A[0] = 3
A[1] = 1
A[2] = 2
A[3] = 3
A[4] = 6
```

For the following elements:

A[0] = 3, the non-divisors are: 2, 6, A[1] = 1, the non-divisors are: 3, 2, 3, 6, A[2] = 2, the non-divisors are: 3, 3, 6, A[3] = 3, the non-divisors are: 2, 6, A[4] = 6, there aren't any non-divisors. Write a function:

func Solution(A []int) []int

that, given an array A consisting of N integers, returns a sequence of integers representing the amount of non-divisors.

Result array should be returned as an array of integers.

For example, given:

```
A[0] = 3
A[1] = 1
A[2] = 2
A[3] = 3
A[4] = 6
```

the function should return [2, 4, 3, 2, 0], as explained above.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [1..50,000]; each element of array A is an integer within the range [1..2 * N].

Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□array□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□□□□map □□A, □□□□□□□□1□sqrt(i)□□□□□,□□□□□,□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□, □□□□□□map, □□□□□□

factor <= int(math.Pow(float64(val), 0.5)) □□ factor*factor <= val □□□□□

□□

- https://app.codility.com/programmers/lessons/11-sieve_of_eratosthenes/count_non_divisible/
- https://github.com/Anfany/Codility-Lessons-By-Python3/blob/master/L11_Sieve%20of%20Eratosthenes/11.1%20CountNonDivisible.md

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0011.Sieve-of-Eratosthenes/CountNonDivisible/CountNonDivisible.go

```go
package countnondivisible

import (
    "math"
)

/*
Task Score 100%
Correctness 100%
Performance 100%
*/
func Solution(A []int) []int {
    // write your code in Go 1.4
    result := []int{}
    if len(A) < 0 {
        return result
    }

    elementDict := make(map[int]int)
    for _, i := range A {
        elementDict[i]++
    }

    // □ map□□□□□□□□□, □□□□□□
    nonDivisorsCountMap := make(map[int]int)
    for _, val := range A {
        if v, ok := nonDivisorsCountMap[val]; ok {
            result = append(result, v)
        } else {
            divisors := 0
            for factor := 1; factor*factor <= val; factor++ {
                if val%factor == 0 {
                    // □□□□□□□□□ array □, □□□□□□□
                    if v, ok := elementDict[factor]; ok {
                        divisors += v
                    }

                    // □□□□□□□□
                    otherFactor := int(val / factor)
                    if v, ok := elementDict[otherFactor]; ok && otherFactor != factor {
                        divisors += v
                    }
                }
            }
            // □□□□□□□
            nonDivisors := len(A) - divisors
            result = append(result, nonDivisors)
            nonDivisorsCountMap[val] = nonDivisors
        }
    }
    return result
}

/*
Task Score 88%
Correctness 100%
Performance 75%
*/
func Solution2(A []int) []int {
    result := []int{}
    if len(A) < 0 {
        return result
    }

    elementDict := make(map[int]int)
    for _, i := range A {
        elementDict[i]++
    }

    // □ map□□□□□□□□□, □□□□□□
    nonDivisorsCountMap := make(map[int]int)
    for _, val := range A {
        if v, ok := nonDivisorsCountMap[val]; ok {
            result = append(result, v)
        } else {
            divisors := 0
            for factor := 1; factor <= int(math.Pow(float64(val), 0.5)); factor++ {
                if val%factor == 0 {
                    // □□□□□□□□□ array □, □□□□□□□
                    if v, ok := elementDict[factor]; ok {
                        divisors += v
                    }

                    // □□□□□□□□
                    otherFactor := int(val / factor)
                    if v, ok := elementDict[otherFactor]; ok && otherFactor != factor {
                        divisors += v
```

```go
                    }
                }
            }
            // □□□□□□□
            nonDivisors := len(A) - divisors
            result = append(result, nonDivisors)
            nonDivisorsCountMap[val] = nonDivisors
        }
    }
    return result
}

/*
Task Score 77%
Correctness 100%
Performance 50%
*/
func Solution1(A []int) []int {
    result := []int{}
    if len(A) < 0 {
        return result
    }

    elementDict := make(map[int]int)
    for _, i := range A {
        elementDict[i]++
    }

    for _, val := range A {
        divisors := 0
        for factor := 1; factor <= int(math.Pow(float64(val), 0.5)); factor++ {
            if val%factor == 0 {
                // □□□□□□□□□ array □, □□□□□□□
                if v, ok := elementDict[factor]; ok {
                    divisors += v
                }

                // □□□□□□□□
                otherFactor := int(val / factor)
                if v, ok := elementDict[otherFactor]; ok && otherFactor != factor {
                    divisors += v
                }
            }
        }
        // □□□□□□□
        nonDivisors := len(A) - divisors
        result = append(result, nonDivisors)
    }
    return result
}
```

CountSemiprimes

COUNTSEMIPRIMES

Count the semiprime(□□□:□□□□□□□□□□□□□□□□□□□□□□□□) numbers in the given range [a..b]

A prime is a positive integer X that has exactly two distinct divisors: 1 and X. The first few prime integers are 2, 3, 5, 7, 11 and 13.

A semiprime is a natural number that is the product of two (not necessarily distinct) prime numbers. The first few semiprimes are 4, 6, 9, 10, 14, 15, 21, 22, 25, 26.

You are given two non-empty arrays P and Q, each consisting of M integers. These arrays represent queries about the number of semiprimes within specified ranges.

Query K requires you to find the number of semiprimes within the range (P[K], Q[K]), where 1 ≤ P[K] ≤ Q[K] ≤ N.

For example, consider an integer N = 26 and arrays P, Q such that:

```
P[0] = 1    Q[0] = 26
P[1] = 4    Q[1] = 10
P[2] = 16   Q[2] = 20
```

The number of semiprimes within each of these ranges is as follows:

(1, 26) is 10, (4, 10) is 4, (16, 20) is 0. Write a function:

func Solution(N int, P []int, Q []int) []int

that, given an integer N and two non-empty arrays P and Q consisting of M integers, returns an array consisting of M elements specifying the consecutive answers to all the queries.

For example, given an integer N = 26 and arrays P, Q such that:

```
P[0] = 1    Q[0] = 26
P[1] = 4    Q[1] = 10
P[2] = 16   Q[2] = 20
```

the function should return the values [10, 4, 0], as explained above.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [1..50,000]; M is an integer within the range [1..30,000]; each element of arrays P, Q is an integer within the range [1..N]; P[i] ≤ Q[i]. Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□[a,b]□□□□□□□□□

□□□□

□□□□□□□□□□. (□□□:□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□4, 6, 9, 10, 14, 15, 21, 22, 25, 26, ... □OEIS□□□□A001358□□□□□1□□□□□□□□□3□4□□□) □□□□array□□□□□□□□□□□ □□P arry □□ Q□P □□□□□□□□□, □□□□□

□□

- https://app.codility.com/programmers/lessons/11-sieve_of_eratosthenes/count_semiprimes/
- https://github.com/Luidy/codility-golang/blob/master/Lesson11/02_countSemiprimes.go
- https://github.com/Anfany/Codility-Lessons-By-Python3/blob/master/L11_Sieve%20of%20Eratosthenes/11.2%20CountSemiprimes.md

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0011.Sieve-of-Eratosthenes/CountSemiprimes/CountSemiprimes.go

```go
package countsemiprimes

func Solution(N int, P []int, Q []int) []int {
    semiPrime := []int{}

    // □□□:□□□□□□□□□□□□□□□□□□□□.
    // 4, 6, 9, 10, 14, 15, 21,22,25,26,33,34,35,38,39,46,49,51,55,57,58,62,65,69,74,77,82,85,86,87,91,93,94,95,106, ...
    // □□□□1□□□□□□□□3□4□□□
    for i := 1; i <= N; i++ {
        factorCount := 0
        sign := 0
        for j := 1; j*j <= i; j++ {
            if i%j == 0 {
                factorCount++
                f := i / j
                if f != j {
                    if f == j*j {
                        // 3□□□: ex i=27, j=3, f=9
                        sign = 1
                        semiPrime = append(semiPrime, 0)
                        break
                    } else {
                        factorCount++
                    }
                }
            }
            if factorCount > 4 {
                sign = 1
                semiPrime = append(semiPrime, 0)
                break
            }
        }
        if sign != 1 {
            if factorCount >= 3 {
                semiPrime = append(semiPrime, i)
            } else {
                semiPrime = append(semiPrime, 0)
            }
        }
    }
    // idx       0 1 2 3 4 5 6 7 8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
    // semiPrime 0 0 0 4 0 6 0 0 9 10  0  0  0 14 15  0  0  0  0  0 21 22  0  0 25 26

    // fmt.Println("semiPrime", semiPrime)

    // □□array□□□□□□□□□□□
    indexMap := make(map[int]int)
    // □□□□□□□□□ map
    semiMap := make(map[int]struct{})
    count := 0

    for i := 0; i < len(semiPrime); i++ {
        if semiPrime[i] != 0 {
            count++
            indexMap[semiPrime[i]] = count
            semiMap[semiPrime[i]] = struct{}{}
        } else {
            indexMap[i+1] = count
        }
    }
    //  indexMap :  map[1:0 2:0 3:0 4:1 5:1 6:2 7:2 8:2 9:3 10:4 11:4 12:4 13:4 14:5 15:6 16:6 17:6 18:6 19:6 20:6 21:7 22:8 23:8 24:8 25:9 26:10]
    // semiMap :  map[4:0 6:0 9:0 10:0 14:0 15:0 21:0 22:0 25:0 26:0]
    // fmt.Println("indexMap : ", indexMap)
    // fmt.Println("semiMap : ", semiMap)

    result := []int{}
    for i := 0; i < len(P); i++ {
        if _, ok := semiMap[P[i]]; ok {
            result = append(result, indexMap[Q[i]]-indexMap[P[i]]+1)
        } else {
            result = append(result, indexMap[Q[i]]-indexMap[P[i]])
        }
    }
    return result
}

// TODO:
func Solution2(N int, P []int, Q []int) []int {
    prime := make([]int, N+1)
    i := 2
    for i*i <= N {
        if prime[i] == 0 {
            k := i * i
            for k <= N {
                if prime[k] == 0 {
                    prime[k] = i
                }
                k = k + i
            }
        }
        i++
    }
    // fmt.Println("prime: ", prime)
```

```go
    // ▫▫▫▫▫▫▫▫▫▫
    semiprime := make([]int, N+1)
    for i := 1; i < len(prime); i++ {
        p := prime[i]
        if p == 0 {
            semiprime[i] = semiprime[i-1]
            continue
        }
        if prime[i/p] == 0 {
            semiprime[i] = semiprime[i-1] + 1
        } else {
            semiprime[i] = semiprime[i-1]
        }
    }
    // fmt.Println("semiprime: ", semiprime)

    result := make([]int, len(P))
    for i, p := range P {
        result[i] = semiprime[Q[i]] - semiprime[p-1]
    }
    return result
}
```

5.1.12 0012.Euclidean Algorithm

ChocolatesByNumbers

CHOCOLATESBYNUMBERS

There are N chocolates in a circle. Count the number of chocolates you will eat.

Two positive integers N and M are given. Integer N represents the number of chocolates arranged in a circle, numbered from 0 to N − 1.

You start to eat the chocolates. After eating a chocolate you leave only a wrapper.

You begin with eating chocolate number 0. Then you omit(□□) the next M − 1 chocolates or wrappers on the circle, and eat the following one.

More precisely(□□), if you ate chocolate number X, then you will next eat the chocolate with number (X + M) modulo N (remainder of division).

You stop eating when you encounter an empty wrapper.

For example, given integers N = 10 and M = 4. You will eat the following chocolates: 0, 4, 8, 2, 6.

The goal is to count the number of chocolates that you will eat, following the above rules.

Write a function:

func Solution(N int, M int) int

that, given two positive integers N and M, returns the number of chocolates that you will eat.

For example, given integers N = 10 and M = 4. the function should return 5, as explained above.

Write an efficient algorithm for the following assumptions:

N and M are integers within the range [1..1,000,000,000].

Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

N□□□□,□□□□□X□ □□□□□ (X + M) modulo N □ □□□□□□□.

□□□□

□□□: □0□□□□, □□□□□+M-1□. □□□□ □□□: □□□□□□□□□□□□□□□□□□□□□ N □□ N □ M □□□□□□. □□ N□M□□□□□□P, N□□P□□□□□□□

□□

https://app.codility.com/programmers/lessons/12-euclidean_algorithm/chocolates_by_numbers/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0012.Euclidean-Algorithm/ChocolatesByNumbers/ChocolatesByNumbers.go

```go
package chocolatesbynumbers

func gcd(N int, M int) int {
    if N%M == 0 {
        return M
    } else {
        return gcd(M, N%M)
    }
}

/*
□□□□□□□□□□□□□□□□□□□ N □□ N □ M □□□□□□
□□ N□M□□□□□□P, N□□P□□□□□□□
O(log(N + M))
*/
```

```go
func Solution(N int, M int) int {
    return N / gcd(N, M)
}

/*
Task Score 75%
Correctness 100%
Performance 50%
input (947853, 4453) the solution exceeded the time limit.
▫0▫▫▫▫, ▫▫▫▫▫+M-1▫
*/
func SolutionBurst(N int, M int) int {
    eaten := make(map[int]struct{})
    eatCount := 0

    if N == 1 || M == 1 {
        return N
    }

    for {
        sumNum := eatCount * M
        startNum := sumNum % N

        if _, ok := eaten[startNum]; !ok {
            eaten[startNum] = struct{}{}
            eatCount++
        } else {
            // ▫▫▫▫▫▫▫▫▫
            break
        }
    }
    return eatCount
}
```

CommonPrimeDivisors

COMMONPRIMEDIVISORS

Check whether two numbers have the same prime divisors.

A prime is a positive integer X that has exactly two distinct divisors: 1 and X. The first few prime integers are 2, 3, 5, 7, 11 and 13.

A prime D is called a prime divisor(□□□) of a positive integer P if there exists a positive integer K such that D * K = P. For example, 2 and 5 are prime divisors of 20.

You are given two positive integers N and M. The goal is to check whether the sets of prime divisors of integers N and M are exactly the same.

For example, given:

N = 15 and M = 75, the prime divisors are the same: {3, 5}; N = 10 and M = 30, the prime divisors aren't the same: {2, 5} is not equal to {2, 3, 5}; N = 9 and M = 5, the prime divisors aren't the same: {3} is not equal to {5}. Write a function:

func Solution(A []int, B []int) int

that, given two non-empty arrays A and B of Z integers, returns the number of positions K for which the prime divisors of A[K] and B[K] are exactly the same.

For example, given:

```
A[0] = 15   B[0] = 75
A[1] = 10   B[1] = 30
A[2] = 3    B[2] = 5
```

the function should return 1, because only one pair (15, 75) has the same set of prime divisors.

Write an efficient algorithm for the following assumptions:

Z is an integer within the range [1..6,000]; each element of arrays A, B is an integer within the range [1..2,147,483,647].
Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□□

□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□, □□□□□□□□□□□□□□□□□□□□□ 15 , 75 □□□□□□□ 3*5 15*= *3*5 75= 3*5*5

□□

- https://app.codility.com/programmers/lessons/12-euclidean_algorithm/common_prime_divisors/
- https://github.com/Anfany/Codility-Lessons-By-Python3/blob/master/L12_Euclidean%20algorithm/12.2%20CommonPrimeDivisors.md
- https://github.com/Luidy/codility-golang/blob/master/Lesson12/02_commonPrimeDivisors.go

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0012.Euclidean-Algorithm/CommonPrimeDivisors/CommonPrimeDivisors.go

```go
package commonprimedivisors

/*
func gcd(a, b int) int {
    for b != 0 {
        t := b
        b = a % b
        a = t
    }
    return a
}
```

```go
*/
func gcd(N int, M int) int {
    if N%M == 0 {
        return M
    } else {
        return gcd(M, N%M)
    }
}

func Solution(A []int, B []int) int {
    result := 0
    for i := 0; i < len(A); i++ {
        if A[i] == B[i] {
            result++
            continue
        }
        // □□□□□□□□□□□,
        abGcd := gcd(A[i], B[i])

        // □□□□□□□□□□□□□□□□□□
        a := A[i] / abGcd
        aGcd := gcd(a, abGcd)
        for aGcd != 1 {
            // □□□□□□
            a = a / aGcd
            aGcd = gcd(aGcd, a)
        }

        b := B[i] / abGcd
        bGcd := gcd(b, abGcd)
        for bGcd != 1 {
            b = b / bGcd
            bGcd = gcd(bGcd, b)
        }
        if a == b {
            result++
        }
    }
    return result
}
```

5.1.13 0013.Fibonacci Numbers

FibFrog

FIBFROG

The Fibonacci sequence is defined using the following recursive formula:

```
F(0) = 0
F(1) = 1
F(M) = F(M - 1) + F(M - 2) if M >= 2
```

A small frog wants to get to the other side of a river. The frog is initially located at one bank of the river (position −1) and wants to get to the other bank (position N). The frog can jump over any distance F(K), where F(K) is the K-th Fibonacci number. Luckily, there are many leaves on the river, and the frog can jump between the leaves, but only in the direction of the bank at position N.

The leaves on the river are represented in an array A consisting of N integers. Consecutive(□□□) elements of array A represent consecutive positions from 0 to N − 1 on the river. Array A contains only 0s and/or 1s:

0 represents a position without a leaf; 1 represents a position containing a leaf. The goal is to count the minimum number of jumps in which the frog can get to the other side of the river (from position −1 to position N). The frog can jump between positions −1 and N (the banks of the river) and every position containing a leaf.

For example, consider array A such that:

```
A[0] = 0
A[1] = 0
A[2] = 0
A[3] = 1
A[4] = 1
A[5] = 0
A[6] = 1
A[7] = 0
A[8] = 0
A[9] = 0
A[10] = 0
```

The frog can make three jumps of length F(5) = 5, F(3) = 2 and F(5) = 5.

Write a function:

func Solution(A []int) int

that, given an array A consisting of N integers, returns the minimum number of jumps by which the frog can get to the other side of the river. If the frog cannot reach the other side of the river, the function should return −1.

For example, given:

```
A[0] = 0
A[1] = 0
A[2] = 0
A[3] = 1
A[4] = 1
A[5] = 0
A[6] = 1
A[7] = 0
A[8] = 0
A[9] = 0
A[10] = 0
```

the function should return 3, as explained above.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [0..100,000]; each element of array A is an integer that can have one of the following values: 0, 1. Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

□□□

□□□□□□□□□□□□□□□□□□□□□□ □□-1, □□□□□□□□□ □□N . □□□□□□□□□□ F(K). □□F(K)□□K□□□□□□. □□□□□□□□□ A[0] = 0 □□□□ 0 □□□□, 1 □□□□□ □□□□□□□□□□□, □□□□□□ N □□□□ □□□□□□□□□

□□□□

□□□□□□ (Breadth-First Search, BFS) □□. □□□□□□□□□□index, □□□□index□□□□□□□f, □□ index - f □□□□□□□□, □index□□□□□□□□□□□□□□□f

□□

- https://app.codility.com/programmers/lessons/13-fibonacci_numbers/fib_frog/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Codility/Lesson/0013.Fibonacci-Numbers/FibFrog/FibFrog.go

```go
package fibfrog

/**
 * @description: □□□□□N□□□□□□□□□
 * @param {int} N
 * @return {*}
 */
func Fib(N int) (fibArr []int) {
    fibArr = append(fibArr, 0)
    fibArr = append(fibArr, 1)
    fibArr = append(fibArr, 1)
    i := 2
    for fibArr[i] < N {
        i = i + 1
        fibArr = append(fibArr, fibArr[i-1]+fibArr[i-2])
    }
    return fibArr
}

func Solution(A []int) int {
    // □□
    A = append(A, 1)
    N := len(A)
    fibArr := Fib(N)
    // □□□□□□ -1 □□ N
    if fibArr[len(fibArr)-1] == N {
        return 1
    }
    fibArr = fibArr[1 : len(fibArr)-1]
    // fmt.Println(fibArr)

    // get the leafs that can be reached from the starting shore
    reachable := make([]int, N)
    for _, v := range fibArr {
        if A[v-1] == 1 {
            // □□□□
            reachable[v-1] = 1
        }
    }
    // □□□□□□□ index: 4 , fib□ [1 1 2 3 5 8], □□□□ 5
    // fmt.Println("re", reachable) // [0 0 0 0 1 0 0 0 0 0 0 0]

    // iterate all the positions until you reach the other shore
    for i := 0; i < N; i++ {
        // □□□□□□□□□□□□path
        if A[i] != 1 || reachable[i] > 0 {
            continue
        }

        // get the optimal jump count to reach this leaf
        if A[i] == 1 {
            // □□□
            // □□□□□□□□, □□□□□□□□□
            minJump := i + 1
            canJump := false
            for _, f := range fibArr {
                previousIdx := i - f

                if previousIdx < 0 || reachable[previousIdx] == 0 {
                    // fmt.Printf("[No] %d :previousIdx = %d reachable = %v \n", i, previousIdx, reachable)
                    continue
                }

                if minJump > reachable[previousIdx] {
                    // □ previousIdx □□□□□□
                    // fmt.Printf("%d :previousIdx = %d reachable = %v \n", i, previousIdx, reachable)
                    minJump = reachable[previousIdx]
                    canJump = true
                }
            }
            if canJump {
                reachable[i] = minJump + 1
            }
        }
        // fmt.Printf("i=%d , reachable = %v \n", i, reachable)
    }

    if reachable[len(reachable)-1] == 0 {
```

```
        return -1
    }
    return reachable[len(reachable)-1]
}
```

5.1.14 0015.Caterpillar Method

AbsDistinct

ABSDISTINCT

A non-empty array A consisting of N numbers is given. The array is sorted in non-decreasing order. The absolute distinct count of this array is the number of distinct absolute values among the elements of the array.

For example, consider array A such that:

A[0] = -5 A[1] = -3 A[2] = -1 A[3] = 0 A[4] = 3 A[5] = 6 The absolute distinct count of this array is 5, because there are 5 distinct absolute values among the elements of this array, namely 0, 1, 3, 5 and 6.

Write a function:

func Solution(A []int) int

that, given a non-empty array A consisting of N numbers, returns absolute distinct count of array A.

For example, given array A such that:

A[0] = -5 A[1] = -3 A[2] = -1 A[3] = 0 A[4] = 3 A[5] = 6 the function should return 5, as explained above.

Write an efficient algorithm for the following assumptions:

N is an integer within the range [1..100,000]; each element of array A is an integer within the range [−2,147,483,648..2,147,483,647]; array A is sorted in non-decreasing order. Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

"□□□□□□□□□□□□□□□□"□□□□□□□□□□□□□□□□

6. GeeksforGeeks

6.1 SortingAlgorithms

6.1.1 0031.Find Minimum Difference Between Any Two Elements

0031. Find Minimum Difference Between Any Two Elements

□□

Given an unsorted array, find the minimum difference between any pair in given array. Examples :

Input : {1, 5, 3, 19, 18, 25}; Output : 1 Minimum difference is between 18 and 19

Input : {30, 5, 20, 9}; Output : 4 Minimum difference is between 5 and 9

Input : {1, 19, -4, 31, 38, 25, 100}; Output : 5 Minimum difference is between 1 and -4

□□

- https://www.geeksforgeeks.org/find-minimum-difference-pair/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/GeeksforGeeks/SortingAlgorithms/0031.Find-Minimum-Difference-Between-Any-Two-Elements/Find-Minimum-Difference-Between-Any-Two-Elements.go

```go
package findminimumdifferencebetweenanytwoelements

import (
    "math"
    "sort"
)

/*
https://yourbasic.org/golang/absolute-value-int-float/
http://cavaliercoder.com/blog/optimized-abs-for-int64-in-go.html
fmt.Println(abs(math.MinInt64)) // Output: -9223372036854775808

// □□□□□
func WithTwosComplement(n int64) int64 {
    y := n >> 63          // y ← x ⟫ 63
    return (n ^ y) - y    // (x ⊕ y) - y
}
*/
func abs(n int) int {
    if n < 0 {
        return -n
    }
    return n
}

func WithTwosComplement(n int64) int64 {
    y := n >> 63       // y ← x ⟫ 63
    return (n ^ y) - y // (x ⊕ y) - y
}

// O(n Log n)
func FindMinDiff(nums []int) int {
    if len(nums) <= 0 {
        return 0
    }
    sort.Ints(nums) // O(n Log n)

    minSize := math.MaxInt32

    // O(n)
    for i := 1; i < len(nums); i++ {
        tmp := int(math.Abs(float64(nums[i] - nums[i-1])))
        if minSize > tmp {
            minSize = tmp
        }
    }

    if minSize == math.MaxInt32 {
        minSize = 0
    }
    return minSize
```

```go
}

func FindMinDiff2(nums []int) int {
    if len(nums) <= 0 {
        return 0
    }
    sort.Ints(nums)

    minSize := math.MaxInt32
    for i := 1; i < len(nums); i++ {
        tmp := abs(nums[i] - nums[i-1])
        if minSize > tmp {
            minSize = tmp
        }
    }

    if minSize == math.MaxInt32 {
        minSize = 0
    }
    return minSize
}
```

7. Leetcode

7.1 0000.xxxx

Easy/Medium/Hard | Go | LEETCODETITLE | LeetCode

7.1.1 NUM.LEETCODETITLE

□□

□□□□

□□□□

Big O

- □□□□ : ``
- □□□□ : ``

□□

- LEETCODELINK
- https://leetcode.cn/problems/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/NUM.FOLDERPATH/main.go

Benchmark

7.2 0001.Two Sum

Easy

7.2.1 1. Two Sum

□□

Given an array of integers, return indices of the two numbers such that they add up to a specific target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

Example:

Given nums = [2, 7, 11, 15], target = 9,

Because nums[0] + nums[1] = 2 + 7 = 9, return [0, 1].

□□□□

□□□□□□ 2 □□□□□□□□□□□□, □□□□ 2 □□□□□□□□□□□.

□□□□

- □□□: □□□□ O(n^2), □□□□ O(1)
- □□□□□□□□□□□□□□ O(n) □□□□□□, □□□□□□□□ map □□□□□□□□□□□□□□□, □□□□□, □□□□ 2 □□□□□□□□. □□□□□, □□□□□□□□ map □, □□□□"□□□"□□□□□, □□□□□□□□.
- □□nums□□□ □□□□□□□□

□□

- https://leetcode-cn.com/problems/two-sum/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0001.Two-Sum/twosum.go

```go
package twosum

// □□□□ O(n^2), □□□□ O(1)
func Twosum(nums []int, target int) []int {
    for i, _ := range nums {
        for j := i + 1; j < len(nums); j++ {
            if target == nums[i]+nums[j] {
                return []int{i, j}
            }
        }
    }
    return []int{0, 0}
}

func Twosum2(nums []int, target int) []int {
    m := make(map[int]int)
    for i, v := range nums {
        if idx, ok := m[target-v]; ok {
            return []int{idx, i}
        }
        m[v] = i
    }
    return []int{0, 0}
}

// □□nums□□□ □□□□□□□□
// func Twosum3(nums []int, target int) []int {
//  left, right := 0, len(nums)-1
//  sort.Ints(nums)
```

```
//  for left < right {
//      sum := nums[left] + nums[right]
//      if sum == target {
//          return []int{left, right}
//      } else if sum < target {
//          left++
//      } else if sum > target {
//          right--
//      }
//  }
//  return []int{0, 0}
// }
```

Array Easy Go Hash Table LeetCode

7.2.2 Solution - UMPIRE

Output 1: UMPIRE □□□□□□□□□

U – UNDERSTAND□□□□□□

- □□□□□□□□□□□□□ `nums` □□□□□□□ `target` □□□□□□□□□□□□□□□□□□□ `target` □□□□□□□□□□□□□□□□
- □□□□□
- □□□□□□□□□□□□□
- □□□□□□□□□□□□□
- □□□□□□□□□□□
- □□□□□
- □□□□□□□ `target` □ `nums[i]` □□□□□□□
- □□□□□□□□ 2□
- □□□□□
- **Happy Path**: `nums = [2, 7, 11, 15], target = 9` -> `[0, 1]`
- **Edge Case 1 (□□□□)**: `nums = [3, 3], target = 6` -> `[0, 1]`
- **Edge Case 2 (□□)**: `nums = [-1, -8, 5], target = -9` -> `[0, 1]`

M – MATCH□□□□□□

- □□□□□/□□□□□**Hash Table (Map)**□
- □□□□□□□
- □□□ ($O(n^2)$) □□□□□□□□
- □□ Hash Table □□□□□□□□□ O(n) □□□ O(1)□□□□□□□□□□□ O(n)□
- □□□□□
- □□ + □□□: □□□□□□□□□□□ O(1)□□□□□□□□□□□□□□□□□□

R – REVIEW□□□□□□

- **Dry Run (** `nums = [2, 7, 11, 15], target = 9` **)**:
- `i=0, v=2` : complement=7□Map □□□□ `{2: 0}` □
- `i=1, v=7` : complement=2□Map □□ 2 (□□ 0)□□□ `[0, 1]` □□□□
- □□□□□Map □□□

E – EVALUATE□□□□□□

- **Time Complexity**: **O(n)**□□□□□□□□□□Map □□□□□□□□ O(1)□
- **Space Complexity**: **O(n)**□□□□□□□□□□□□□□□ Map□
- **Trade-offs**: □□□□□□O(n)□□□□□□□□□□□O(n)□□

Output 2: □□□□□□□□□□□□□□□□□□

1 □□□□□□

□□□□□□□□□□□□□□□□□□□□□□ `target` □□□□□□□□□□□□□□□□□□□□

7.2.3 Merging 2 Packages

Index

Given a package with a weight limit limit and an array arr of item weights, implement a function getIndicesOfItemWeights that finds two items whose sum of weights equals the weight limit. The function should return a pair [i, j] of the indices of the item weights, ordered such that i > j. If such a pair doesn't exist, return an empty array. input: arr = [4, 6, 10, 15, 16], lim = 21 output: [3, 1] # since these are the indices of the weights 6 and 15 whose sum equals to 21

7.3 0002.Add Two Numbers

Add Two Numbers **Amazon** **Apple** **Bloomberg** **Facebook** **Go** **LeetCode** **Linked List** **Math** **Medium** **Microsoft** **Recursion**

7.3.1 0002.Add Two Numbers

□□

□□□□

□□□□

□□□□□□□ □□□□□□□□□□□□□□□□ □□□□ carry □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

- □□ l1□ l2. □□□list□val□□, □□□□□□□□□next□□
- □□□□ carry □□□□, □□□□□□□□□□□

Big O

- □□□□ : `O(max(m,n)` □□□□□□ O(max(m,n)) □□□ m □ n □□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ O(1) □□□
- □□□□ : `O(1)` O(1) □ □□□□□□□□□□□□□□

□□

- https://leetcode.com/problems/add-two-numbers/description/
- https://leetcode.cn/problems/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0002.Add-Two-Numbers/main.go

```go
package addtwonumbers

// □□□□ O(max(m,n)), □□□□ O(1)
/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */

// □□ l1□ l2. □□□list□val□□, □□□□□□□□□next□□
// □□□□ carry □□□□, □□□□□□□□□□
func addTwoNumbers(l1 *ListNode, l2 *ListNode) *ListNode {
    var result, tail *ListNode
    carry := 0
    for l1 != nil || l2 != nil {
        n1, n2 := 0, 0
        if l1 != nil {
            n1 = l1.Val
            l1 = l1.Next
        }
        if l2 != nil {
            n2 = l2.Val
            l2 = l2.Next
        }
        sum := n1 + n2 + carry
        sum, carry = sum%10, sum/10

        if result == nil {
            result = &ListNode{Val: sum, Next: nil}
```

```
            tail = result
        } else {
            tail.Next = &ListNode{Val: sum, Next: nil}
            tail = tail.Next
        }
    }
    // □□□□ carry □□□□, □□□□□□□□□□
    if carry > 0 {
        tail.Next = &ListNode{Val: carry, Next: nil}
    }
    return result
}
```

Benchmark

```

```

Add Two Numbers **Amazon** **Apple** **Bloomberg** **Facebook** **Go** **LeetCode** **Linked List** **Math** **Medium** **Microsoft** **Recursion**

7.3.2 Solution - UMPIRE

Output 1: UMPIRE □□□□□□□□□

U – UNDERSTAND□□□□□□

- □□□□□□□□□

- □□□□□□□□□□
- `carry` □□□□□□□□□□□□□
- □□ `l1` □ `l2` □□□□□□□□□□□□□
- □□ **Bug**□
- □□ `while` □□□□□□□□ `carry` □□□ 0□□□□□□□□□□□□□
- □□□ `l1.Val` □□□□□□ `l1 != nil` □

I – IMPLEMENT□□□□□GOLANG□

```go
/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */
func addTwoNumbers(l1 *ListNode, l2 *ListNode) *ListNode {
    dummyHead := &ListNode{Val: 0}
    curr := dummyHead
    carry := 0

    for l1 != nil || l2 != nil || carry != 0 {
        val1, val2 := 0, 0
        if l1 != nil {
            val1 = l1.Val
            l1 = l1.Next
        }
        if l2 != nil {
            val2 = l2.Val
            l2 = l2.Next
        }

        sum := val1 + val2 + carry
        carry = sum / 10
        curr.Next = &ListNode{Val: sum % 10}
        curr = curr.Next
    }

    return dummyHead.Next
}
```

R – REVIEW□□□□□□

- **Dry Run**:
- □□□$L1: [2, 4], L2: [5, 6, 4]$
- 1st: $2+5+0 = 7, carry=0, Result: [7]$
- 2nd: $4+6+0 = 10, carry=1, Result: [7, 0]$
- 3rd: $0+4+1 = 5, carry=0, Result: [7, 0, 5]$
- □□ $[7, 0, 5]$□□□□$42 + 465 = 507$□□
- □□□□□□□□□□□□□□□□□□
- □□□□□□□ $5+5$ □□□□□□□□□□□ $[0, 1]$□□□□□□

E – EVALUATE□□□□□□

- □□□□□: $O(\max(M, N))$□□□ M, N □□□□□□□□□□□□□□□□□□□□□
- □□□□□: $O(\max(M, N))$□□□□□□□□□□□□□□□□□□□□ $O(1)$□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□In-place□□□□□□□□□□□□□□□□□□□

2 □□□□□

□□□□□□□□□□**Simultaneous Traversal**□□□□□□□□□□

7.4 0003.Longest Substring Without Repeating Characters

Array **Go** **Hash Table** **LeetCode** **Longest Substring Without Repeating Characters** **Medium** **Sliding Window** **String**

7.4.1 3. Longest Substring Without Repeating Characters

□□

Given a string, find the length of the longest substring without repeating characters.

Example 1:

```
Input: "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.
```

Example 2:

```
Input: "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
```

Example 3:

```
Input: "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
Note that the answer must be a substring, "pwke" is a subsequence and not a substring.
```

□□□□

□□□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□ 438 □□□ 3 □□□ 76 □□□ 567 □□□□□□□□□□"□□□□"□

□□□□□□□□□□□□□□□□□□□

```go
    slength := len(s)
    if slength == 0 || slength == 1 {
        return slength
    }

    tmpLen := 1
    var maxLen = 1

    for i := 1; i < slength; i++ {
        // □□□□□□□□
        j := i - tmpLen

        for ; j < i; j++ {
            if s[j] == s[i] { // □□□□□□□□S[J]□S[I-1]□□□□□□□□□□□□□□□□□□□
                tmpLen = i - j
                break
            }
        }

        if j == i { // □□□□
            tmpLen++
        }

        if tmpLen > maxLen {
            maxLen = tmpLen
        }
    }

    return maxLen
}

// LengthOfLongestSubstringMap □map □□□□□□.
func LengthOfLongestSubstringMap(s string) int {
    slength := len(s)
    if slength == 0 || slength == 1 {
        return slength
    }

    charMap := make(map[byte]bool)
    maxLen, left, right := 0, 0, 0

    for left < slength {
        if ok := charMap[s[right]]; ok {
            // □□□
            charMap[s[left]] = false
            left++
        } else {
            charMap[s[right]] = true
            right++
        }
        if maxLen < right-left {
            maxLen = right - left
        }
        if (left+maxLen) >= slength || right >= len(s) {
            break
        }
    }

    return maxLen
}

// LengthOfLongestSubstringBit □map□□□□□□□□□□□□
func LengthOfLongestSubstringBit(s string) int {
    slength := len(s)
    if slength == 0 || slength == 1 {
        return slength
    }

    // ASCII 0~255
    charMap := [256]bool{}
    maxLen, left, right := 0, 0, 0
    for left < slength {
        if ok := charMap[s[right]]; ok {
            // □□□
            charMap[s[left]] = false
            left++
        } else {
            charMap[s[right]] = true
            right++
        }

        if maxLen < right-left {
            maxLen = right - left
        }
        if left+maxLen >= slength || right >= len(s) {
            break
        }
    }
    return maxLen
}
```

Benchmark

```
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/0003.Longest-Substring-Without-Repeating-Characters
cpu: Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz
BenchmarkLengthOfLongestSubstring-8             66143602                19.08 ns/op            0 B/op          0 allocs/op
BenchmarkLengthOfLongestSubstringMap-8           2524627               397.8 ns/op             0 B/op          0 allocs/op
BenchmarkLengthOfLongestSubstringBit-8          65099846                21.37 ns/op            0 B/op          0 allocs/op
PASS
ok      LeetcodeGolang/Leetcode/0003.Longest-Substring-Without-Repeating-Characters     4.193s
```

7.5 0005.Longest Palindromic Substring

Adobe **Amazon** **DP** **Facebook** **Go** **Google** **LeetCode** **Medium** **Microsoft**

7.5.1 0005.Longest Palindromic Substring

□□

Given a string s, return the longest palindromic substring in s.

A string is called a palindrome string if the reverse of that string is the same as the original string.

Example 1:

```
Input: s = "babad"
Output: "bab"
Explanation: "aba" is also a valid answer.
```

Example 2:

```
Input: s = "cbbd"
Output: "bb"
```

Constraints: * 1 <= s.length <= 1000 * s consist of only digits and English letters.

□□□□

□□□□□□□ s□□□ s □□□□□□□□□

□□□□

- □□□□□□□□□□□
- □□□ 2, □□□□□□□□□□□
- □□>=3, □□□□□□, □□□□□□□□□□□□□. □□□□□□□□□□□

0005. Longest Palindromic string

最長回文子串 #DP

babad

end \ start	0	1	2	3	4
0	b ✓				
1	ba ✗	a ✓			
2	bab ✓	ab ✗	b ✓		
3	baba ✗	aba ✓	ba ✗	a ✓	
4	babad ✗	abad ✗	bad ✗	ad ✗	d ✓

每一個字符本身都是回文

長度為2，且首尾字符相同，則為回文

長度大於等於3
如果頭尾相同
則去掉頭尾後
看是否是回文
如果頭尾不同則不是回文

dp[start][end] = dp[start+1][end-1]

```
for length := 2 ; length <= len(s) ; length ++ {
    for start := 0 ; start < len(s) - length + 1 {
        :
}
```

□□

- https://leetcode.com/problems/longest-palindromic-substring/
- https://leetcode.cn/problems/longest-palindromic-substring/solutions/1348874/s-by-xext-5zk3/

□□

```go
package longestpalindromicsubstring

func longestPalindrome(s string) string {
    dp := make([][]bool, len(s))
    result := s[0:1]

    for i := 0; i < len(s); i++ {
        dp[i] = make([]bool, len(s))
        dp[i][i] = true // □□□□□□□□□□
    }
    for length := 2; length <= len(s); length++ {
        for start := 0; start < len(s)-length+1; start++ {
            end := start + length - 1
            if s[start] != s[end] {
                // □□□□□□, □□□□
                continue
            } else if length < 3 {
                // □□□2□□□□□□□, □□□□
                dp[start][end] = true
            } else {
                // □□□□ : □□□□□□, □□□□□□□□
                dp[start][end] = dp[start+1][end-1]
            }
            if dp[start][end] && (end-start+1) > len(result) {
                result = s[start : end+1]
            }
        }
    }
    return result
}
```

7.6 0010.Regular Expression Matching

Dynamic Programming **Go** **Hard** **LeetCode** **Recursion** **String**

7.6.1 0010.Regular-Expression-Matching

□□

Given an input string s and a pattern p, implement regular expression matching with support for '.' and '*' where:

```
&#39;.&#39; Matches any single character.
&#39;*&#39; Matches zero or more of the preceding element.
```

The matching should cover the entire input string (not partial).

Example 1:

Input: s = "aa", p = "a" Output: false Explanation: "a" does not match the entire string "aa".

Example 2:

Input: s = "aa", p = "a" *Output: true Explanation:* '' means zero or more of the preceding element, 'a'. Therefore, by repeating 'a' once, it becomes "aa".

Example 3:

Input: s = "ab", p = "." *Output: true Explanation:* "." means "zero or more (*) of any character (.)".

Constraints:

```
1 &lt;= s.length &lt;= 20
1 &lt;= p.length &lt;= 20
s contains only lowercase English letters.
p contains only lowercase English letters, &#39;.&#39;, and &#39;*&#39;.
It is guaranteed for each appearance of the character &#39;*&#39;, there will be a previous valid character to match.
```

□□□□

□□□□

□□

- https://leetcode.com/problems/regular-expression-matching/description/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0010.Regular-Expression-Matching/main.go

Benchmark

7.7 0011.Container With Most Water

Adobe Amazon Array Container With Most Water Facebook Go Google Greedy LeetCode Medium Microsoft Two Pointers

7.7.1 0011.Container With Most Water

□□

□□□□

□□□□

□□□□□, □□□□□□□□□□□□□□, □□□□ □□□□□□□□□□□□□□□, □□□□□□□, □□□□□□□□□.

Big O

- □□□□: `O(n)`
- □□□□: `O(1)`

□□

- https://leetcode.com/problems/container-with-most-water/description/
- https://leetcode.cn/problems/container-with-most-water/description/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0011.Container-With-Most-Water/main.go

```go
package containerwithmostwater

// □□□□ O(n), □□□□ O(1)
func maxArea(height []int) int {
    left, right := 0, len(height)-1
    result := 0
    for left < right {
        tmp := (right - left) * min(height[left], height[right])
        result = max(result, tmp)
        if height[left] > height[right] {
            right--
        } else {
            left++
        }
    }
    return result
}

func min(a, b int) int {
    if a < b {
        return a
    }
    return b
}

func max(a, b int) int {
    if a > b {
        return a
    }
    return b
}
```

Benchmark

7.8 0015.3Sum

3Sum Array Go LeetCode Medium Sorting Two Pointers

7.8.1 15. 3Sum

□□

Given an array nums of n integers, are there elements a, b, c in nums such that a + b + c = 0? Find all unique triplets in the array which gives the sum of zero.

Note:

The solution set must not contain duplicate triplets.

Example:

```
Given array nums = [-1, 0, 1, 2, -1, -4],

A solution set is:
[
  [-1, 0, 1],
  [-1, -1, 2]
]
```

□□□□

□□□□□□□□□□□□□□□□ 3 □□□□□ 0 □□□□□□

□□□□

□ map □□□□□□□ 2 □□□□□□□□□□□□□□□□□□□□□□ O(n^2)□□□□□□□□□□□□□□□□□□□

```go
                if nums[i]+nums[j]+nums[k] == 0 {
                    result = append(result, []int{nums[i], nums[j], nums[k]})
                }
            }
        }
    }
    return result
}

/*
ThreeSumDoublePoint : □□□, □□ + □□□□ (□□□□) O(n^2)
1. □□□□□□□□□ n□□□□□□ null □□□□□□□□ 3□□□ []□
2. □□□□□□□□
3. □□□□□□□□
  - □□□□□□□□□□□□□□□□□
  - □□□□ L=i+1□□□□ R=n−1□□ L<R □□□□□□□
  - □nums[i]+nums[L]+nums[R]==0□□□□□□□□□□□□□□□□□□□□□□□
    □□□□□□□□□□ L,R □□□□□□□□□□□□
  - □□□□ 0□□□ nums[R] □□□R □□
  - □□□□ 0□□□ nnums[L] □□□L □□
*/
func ThreeSumDoublePoint(nums []int) [][]int {
    result := [][]int{}
    if len(nums) < 3 {
        return result
    }
    sort.Ints(nums) // O(n log n)
    start, end, addNum := 0, 0, 0
    for i := 1; i < len(nums)-1; i++ {
        start, end = 0, len(nums)-1
        if i > 1 && nums[i] == nums[i-1] {
            // □□□□
            start = i - 1
        }
        for start < i && end > i {
            if start > 0 && nums[start] == nums[start-1] {
                // □□□□
                start++
                continue
            }
            if end < (len(nums)-1) && nums[end] == nums[end+1] {
                // □□□□
                end--
                continue
            }
            addNum = nums[start] + nums[end] + nums[i]
            if addNum == 0 {
                result = append(result, []int{nums[start], nums[i], nums[end]})
                start++
                end--
            } else if addNum > 0 {
                end--
            } else {
                start++
            }
        }
    }
    return result
}

func ThreeSumHashTable(nums []int) [][]int {
    result := [][]int{}
    if len(nums) < 3 {
        return result
    }
    sort.Ints(nums) // O(n log n)

    for i := 0; i < len(nums)-2; i++ {
        // □□□□□□□□□
        if i > 0 && nums[i] == nums[i-1] {
            continue
        }

        seen := make(map[int]bool)
        target := -nums[i] // □□□□□□□□□□□
        for j := i + 1; j < len(nums); j++ {
            complement := target - nums[j] // □□□□□□□□□□□□□
            if seen[complement] {
                result = append(result, []int{nums[i], complement, nums[j]})
                // □□□□□□□□□
                for j < len(nums)-1 && nums[j] == nums[j+1] {
                    j++
                }
            }
            seen[nums[j]] = true
        }
    }
    return result
}

func ThreeSumTwoPointer(nums []int) [][]int {
    result := [][]int{}
    sort.Ints(nums)
```

```go
    for i := 0; i < len(nums)-2; i++ {
        if i > 0 && nums[i] == nums[i-1] {
            continue
        }
        target, l, r := -nums[i], i+1, len(nums)-1
        for l < r {
            sum := nums[l] + nums[r]
            if sum == target {
                result = append(result, []int{nums[i], nums[l], nums[r]})
                l++
                r--
                for l < r && nums[l] == nums[l-1] {
                    l++
                }
                for l < r && nums[r] == nums[r+1] {
                    r--
                }
            } else if sum > target {
                r--
            } else if sum < target {
                l++
            }
        }
    }
    return result
}
```

```
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/0015.3Sum
cpu: Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz
BenchmarkThreeSumBurst-4                 9838000               121.4 ns/op            48 B/op          2 allocs/op
BenchmarkThreeSumDoublePoint-4           9069201               112.8 ns/op            48 B/op          2 allocs/op
BenchmarkThreeSumHashTable-4             7935907               147.1 ns/op            48 B/op          2 allocs/op
BenchmarkThreeSumTwoPointer-4           10888315               103.5 ns/op            48 B/op          2 allocs/op
PASS
ok      LeetcodeGolang/Leetcode/0015.3Sum       5.055s
```

7.9 0019.Remove Nth Node From End of List

Amazon Bloomberg Facebook Go Google LeetCode Linked List Medium Microsoft Two Pointers

7.9.1 19. Remove Nth Node From End of List

□□

Given the head of a linked list, remove the nth node from the end of the list and return its head.

Example 1:

```
Input: head = [1,2,3,4,5], n = 2
Output: [1,2,3,5]
```

Example 2:

```
Input: head = [1], n = 1
Output: []
```

Example 3:

```
Input: head = [1,2], n = 1
Output: [1]
```

Constraints: * The number of nodes in the list is sz. * 1 <= sz <= 30 * 0 <= Node.val <= 100 * 1 <= n <= sz

□□□□

□□□linked list□ □□□ n □□□□□□. □□□ linked list□□□□

□□□□

□□ fast□ `k` □, □□ `fast slow ▫▫▫▫` □□□fast□□nil□, slow□□□□□□□□□□ k □□□

□□

- https://leetcode.com/problems/remove-nth-node-from-end-of-list/

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0019.Remove-Nth-Node-From-End-of-List/main.go

```go
package removenthnodefromendoflist

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */

type ListNode struct {
    Val  int
    Next *ListNode
}

// □□ dummyHead,□ preslow
// □□□□□, □□ fast□ `k` □, □□ `fast slow □□□□`
// □□□fast□□nil□, slow□□□□□□□□□□ k □□□
// □ slow□□□□(preslow)□next □□ slow.Next
func RemoveNthFromEnd(head *ListNode, n int) *ListNode {
    dummyHead := &ListNode{Next: head}
    preSlow, slow, fast := dummyHead, head, head
    for fast != nil {
        if n <= 0 {
            // □□ fast□ `k` □, □□ `fast slow □□□□`
            // □□□fast□□nil□, slow□□□□□□□□□□ k □□□
            preSlow = slow
            slow = slow.Next
        }
        n--
        fast = fast.Next
    }
    preSlow.Next = slow.Next
    return dummyHead.Next
}
```

tags: `Medium` `Leetcode` `Two Pointers`

7.10 0020.Valid Parentheses

Easy **Go** **LeetCode** **Valid Parentheses**

7.10.1 0020. Valid Parentheses

□□

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

Open brackets must be closed by the same type of brackets. Open brackets must be closed in the correct order. Every close bracket has a corresponding open bracket of the same type.

Example 1:

Input: s = "()" Output: true Example 2:

Input: s = "()[]{}" Output: true Example 3:

Input: s = "(]" Output: false

Constraints:

1 <= s.length <= 104 s consists of parentheses only '()[]{}'.

□□□□

□□□□

Big O

□□□□: `□□□□ :`

□□

- https://leetcode.com/problems/valid-parentheses/
- https://leetcode.cn/problems/valid-parentheses/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0020.Valid-Parentheses/main.go

```go
package validparentheses

type Stack struct {
    runes []rune
}

func NewStack() *Stack {
    return &Stack{runes: []rune{}}
}

func (s *Stack) Push(str rune) {
    s.runes = append(s.runes, str)
}

func (s *Stack) Pop() rune {
    str := s.runes[len(s.runes)-1]
    s.runes = s.runes[:len(s.runes)-1]
    return str
}
```

```go
// □□□□ O(n), □□□□ O(n)
func IsValid(s string) bool {
    runeStack := NewStack()
    for _, v := range s {
        // fmt.Println(string(v))
        if v == '(' || v == '[' || v == '{' {
            runeStack.Push(v)
        } else if (v == ')' && len(runeStack.runes) > 0 && runeStack.runes[len(runeStack.runes)-1] == '(') || (v == ']' && len(runeStack.runes) > 0 && runeSta
ck.runes[len(runeStack.runes)-1] == '[') || (v == '}' && len(runeStack.runes) > 0 && runeStack.runes[len(runeStack.runes)-1] == '{') {
            runeStack.Pop()
        } else {
            return false
        }
    }
    return len(runeStack.runes) == 0
}
```

Benchmark

7.11 0021.Merge Two Sorted Lists

Easy/Medium/Hard **Go** **LeetCode** **Merge Two Sorted Lists**

7.11.1 0021. Merge Two Sorted Lists

□□

□□□□

□□□□

Big O

□□□□: `O( log n)` □□□□: `O(1)`

□□

- https://leetcode.com/problems/merge-two-sorted-lists/description/
- https://leetcode.cn/problems/merge-two-sorted-lists/description/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0021.Merge-Two-Sorted-Lists/main.go

```go
package mergetwosortedlists

type ListNode struct {
    Val  int
    Next *ListNode
}

// □□□□ O(log n), □□□□ O(1)
func MergeTwoLists(list1 *ListNode, list2 *ListNode) *ListNode {
    head := &ListNode{Next: nil}
    cur := head

    for list1 != nil && list2 != nil {
        if list1.Val < list2.Val {
            cur.Next = list1
            list1 = list1.Next
        } else {
            cur.Next = list2
            list2 = list2.Next
        }
        cur = cur.Next
    }

    if list1 != nil {
        cur.Next = list1
    }
    if list2 != nil {
        cur.Next = list2
    }
    return head.Next
}
```

Benchmark

7.12 0027.Remove Element

Array **Easy**

7.12.1 27. Remove Element

□□

Given an array nums and a value val, remove all instances of that value in-place and return the new length.

Do not allocate extra space for another array, you must do this by modifying the input array in-place with O(1) extra memory.

The order of elements can be changed. It doesn't matter what you leave beyond the new length.

Example 1:

```
Given nums = [3,2,2,3], val = 3,

Your function should return length = 2, with the first two elements of nums being 2.

It doesn't matter what you leave beyond the returned length.
```

Example 2:

```
Given nums = [0,1,2,2,3,0,4,2], val = 2,

Your function should return length = 5, with the first five elements of nums containing 0, 1, 3, 0, and 4.

Note that the order of those five elements can be arbitrary.

It doesn't matter what values are set beyond the returned length.
```

Clarification:

Confused why the returned value is an integer but your answer is an array?

Note that the input array is passed in by reference, which means modification to the input array will be known to the caller as well.

Internally you can think of this:

```
// nums is passed in by reference. (i.e., without making a copy)
int len = removeElement(nums, val);

// any modification to nums in your function would be known by the caller.
// using the length returned by your function, it prints the first len elements.
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
```

□□□□

□□□□□□ nums □□□□□ val□□□□□□□□□ val □□□□□□□□□□□□□□□□□

□□□□

□□□□□ 283 □□□□□□□□□ 283 □□□□□□283 □□□□ 0□□□□□□□□□□ val□□□□□□□□

□□□OJ □□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□

- https://books.halfrost.com/leetcode/ChapterFour/0001~0099/0027.Remove-Element/

- https://leetcode-cn.com/problems/remove-element/
- https://mp.weixin.qq.com/s/wj0T-Xs88_FHJFwayElQlA

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0027.Remove-Element/Remove-Element.go

```go
package removeelement

/*
□□□□
□□□□
```

7.13 0028.Find the Index of the First Occurrence in a String

Easy | Find the Index of the First Occurrence in a String | Go | LeetCode | String | String Matching | Two Pointers

7.13.1 0028. Find the Index of the First Occurrence in a String

□□

□□□□

□□□□

Big O

- □□□□ : ``
- □□□□ : ``

□□

- https://leetcode.com/problems/find-the-index-of-the-first-occurrence-in-a-string/description/
- https://leetcode.cn/problems/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0028.Find-the-Index-of-the-First-Occurrence-in-a-String/main.go

```go
package findtheindexofthefirstoccurrenceinastring

// □□□
// □□□□ O(M*N), □□□□ O()
func strStr(haystack string, needle string) int {
    haystackLen := len(haystack)
    needleLen := len(needle)
    index := 0
    for i := 0; i <= (haystackLen - needleLen); i++ {
        j := 0
        for j = 0; j < needleLen; j++ {
            if haystack[i+j] == needle[j] {
                index = i
            } else {
                break
            }
        }
        if j == needleLen {
            return index
        }
    }
    return -1
}

// Slice □□
func strStrSlice(haystack string, needle string) int {
    haystackLen := len(haystack)
    needleLen := len(needle)
    if haystackLen == 0 || haystackLen < needleLen {
        return -1
    }
    if needleLen == 0 {
        return 0
    }
    for i := 0; i <= (haystackLen - needleLen); i++ {
        if haystack[i:i+needleLen] == needle {
            return i
        }
```

```
    }
    return -1
}
```

Benchmark

```

```

7.14 0033.Search in Rotated Sorted Array

33. Search in Rotated Sorted Array **Amazon** **Apple** **Array** **Binary Search** **Facebook** **Go** **LeetCode** **LinkedIn** **Medium** **Microsoft**

7.14.1 33. Search in Rotated Sorted Array

□□

There is an integer array nums sorted in ascending order (with distinct values).

Prior to being passed to your function, nums is possibly rotated at an unknown pivot index k (1 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] (0-indexed). For example, [0,1,2,4,5,6,7] might be rotated at pivot index 3 and become [4,5,6,7,0,1,2].

Given the array nums after the possible rotation and an integer target, return the index of target if it is in nums, or -1 if it is not in nums.

You must write an algorithm with O(log n) runtime complexity.

Example 1:

Input: nums = [4,5,6,7,0,1,2], target = 0 Output: 4 Example 2:

Input: nums = [4,5,6,7,0,1,2], target = 3 Output: -1 Example 3:

Input: nums = [1], target = 0 Output: -1

Constraints:

1 <= nums.length <= 5000 -104 <= nums[i] <= 104 All values of nums are unique. nums is an ascending array that is possibly rotated. -104 <= target <= 104

□□□□

□□□□□□□□□□, □□□□ □□□□□□□□□□ nums □□□□□□□□□□□ k □ 0 <= k < nums.length □□□□□ □□□□□□□□□ [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] □□□ □ 0 □□ □□□□ □□□ [0,1,2,4,5,6,7] □□□ 3 □□□□□□□□□ [4,5,6,7,0,1,2] □

□□ □□□ □□□ nums □□□□□ target □□□ nums □□□□□□□□ target □□□□□□□□□□□□□□ -1 □

□□□□□□□□□□□□□ O(log n) □□□□□□□□□□□□

□□□□

□□□mid□□□□□□□□□□□, □□□target□□□□ □□□target□□□□□□□□□

33. Search in Rotated Sorted Array

Input: num[4, 5, 6, 7, 0, 1, 2] ; target = 0

Output: 4

如果找到target
回傳下標
如果沒找到
回傳 -1

if num[mid] >= num[left]

① mid 在上半區

① left target mid

righ = mid - 1

② target > mid

left = mid + 1

② mid 在下半區

① mid target right

left = mid + 1

② right = mid - 1

最後看結果有沒有出現在左、右邊

```
if num[left] == target {
    return left
}else if num[right] == target {
    return right
}else {
    return -1
}
```

Big O

- □□□□: `O(log n)`
- □□□□: `O(1)`

□□

- https://leetcode.com/problems/search-in-rotated-sorted-array/description/
- https://leetcode.cn/problems/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0033.Search-in-Rotated-Sorted-Array/main.go

```go
package searchinrotatedsortedarray

// □□□□ O(), □□□□ O()
func search(nums []int, target int) int {
    if nums == nil || len(nums) == 0 {
        return -1
    }
    left, right := 0, len(nums)-1
    for left < right {
        mid := int(uint(left+right) >> 1)

        if nums[mid] == target {
            return mid
        }

        if nums[mid] >= nums[left] {
            // mid□□□□

            if nums[left] <= target && target <= nums[mid] {
                // target □left□min□□
                right = mid - 1
            } else {
                left = mid + 1
            }
        } else {
            // mid□□□□
            if nums[mid] <= target && target <= nums[right] {
                // target □min□right□□
                left = mid + 1
            } else {
                right = mid - 1
            }
        }
    }
    if nums[left] == target {
        return left
    } else if nums[right] == target {
        return right
    } else {
        return -1
    }
}
```

Benchmark

```
```

7.15 0034.Find First and Last Position of Element in Sorted Array

Array **Binary Search** **Find First and Last Position of Element in Sorted Array** **Go** **LeetCode** **Medium**

7.15.1 0034. Find First and Last Position of Element in Sorted Array

□□

□□□□

□□□□

Big O

- □□□□: `O(log(n))`
- □□□□: `O(1)`

□□

- https://leetcode.com/problems/find-first-and-last-position-of-element-in-sorted-array/description/
- https://leetcode.cn/problems/find-first-and-last-position-of-element-in-sorted-array/description/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0034.Find-First-and-Last-Position-of-Element-in-Sorted-Array/main.go

```go
package findfirstandlastpositionofelementinsortedarray

// □□□□ O(log(n)), □□□□ O(1)
func searchRange(nums []int, target int) []int {
    result := []int{}

    // first index
    left, right := 0, len(nums)-1
    for left < right {
        mid := int(uint(left+right) >> 1) // [0,1] => 0
        if nums[mid] < target {
            left = mid + 1
        } else if nums[mid] > target {
            right = mid - 1
        } else {
            right = mid
        }
    }
    if left == right && nums[left] == target {
        result = append(result, left)
    } else {
        result = append(result, -1)
    }

    // last index
    left, right = 0, len(nums)-1
    for left < right {
        mid := int(uint(left+right+1) >> 1) // [0,1] => 1
        if nums[mid] < target {
            left = mid + 1
        } else if nums[mid] > target {
            right = mid - 1
        } else {
            left = mid
        }
    }
    if left == right && nums[left] == target {
        result = append(result, left)
```

```
    } else {
        result = append(result, -1)
    }
    return result
}
```

Benchmark

```

```

7.16 0035.Search Insert Position

Array **Easy**

7.16.1 35. Search Insert Position

□□

Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

Example 1:

```
Input: [1,3,5,6], 5
Output: 2
```

Example 2:

```
Input: [1,3,5,6], 2
Output: 1
```

Example 3:

```
Input: [1,3,5,6], 7
Output: 4
```

Example 4:

```
Input: [1,3,5,6], 0
Output: 0
```

□□□□

□□□□

```go
        if nums[i] >= target {
            return i
        }
    }
    return len(nums)
}

//□□□ □□□□ O(log n) □□□□ O(1)
func SearchInsertBisection(nums []int, target int) int {
    left, right := 0, len(nums)-1
    for left <= right {
        // / □□□□ □(left + right)/2
        mid := left + (right-left)>>1
        if nums[mid] >= target {
            right = mid - 1
        } else if nums[mid] < target {
            left = mid + 1
        } else {
            return mid
        }
    }
    // □□□□□□□□□□
    // targe□□□□□□□□ [0, -1]
    // targe□□□□□□□□□□ return middle;
    // targe□□□□□□□□ [left, right]□return right + 1
    // targe□□□□□□□□□□□□□ [left, right]□ return right + 1
    return right + 1
}

//□□□ □□□□ O(log n) □□□□ O(1)
func SearchInsertBisection2(nums []int, target int) int {
    left, right := 0, len(nums)-1
    for left <= right {
        // / □□□□ □(left + right)/2
        mid := left + (right-left)>>1
        if nums[mid] >= target {
            right = mid - 1
        } else {
            left = mid + 1
        }
    }
    return left
}
```

7.17 0046.Permutations

Backtracking **Medium**

7.17.1 46. Permutations

□□

Given a collection of **distinct** integers, return all possible permutations(□□). **Example:** Input: [1,2,3] Output: [[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], [3,2,1]]

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□ □□□ □□□□□ 1. □□: □□□□□□□□□ 2. □□□□: □□□□□□□□□□□□ 3. □□□□: □□□□□□□□□□□, □□□□□□□□□

```
result = []
def backtrack(□□, □□□□):
    if □□□□□□:
        result.add(□□)
        return

    for □□ in □□□□:
        □□□
        backtrack(□□, □□□□)
        □□□□
```

```
                    □□:[1,2,3]
                          []
          [1]/            |[2]           \[3]
        [2]/  \[3]      [1]/  \[3]      [1]/  \[2]
        |[3]   |[2]     |[3]   |[1]     |[2]    |[1]
□□  [1,2,3]  [1,3,2] [2,1,3] [2,3,1]  [3,1,2]  [3,2,1]
```

- □□□□□□□□□□□□□□□□□□□□ DFS □□□□□ □□□□□□□□□ ñ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ ñ□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□

□□

- https://books.halfrost.com/leetcode/ChapterFour/0001~0099/0046.Permutations/
- https://leetcode-cn.com/problems/permutations/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0046.Permutations/Permutations.go

□□□□ O(n)

```
package permutations

func Permute(nums []int) [][]int {
    numsLen := len(nums)
    if numsLen == 0 {
        return [][]int{}
    }
    used, path, res := make([]bool, numsLen), []int{}, [][]int{}
    dfs(nums, numsLen, 0, path, &used, &res)
    return res
}

/*
generatePermutation: (□□□□, □□□□, □□□□□□depth, path, □□□□, □□)
□□□□□□**BFS**, □□□□**DFS**□□□□□, □□□□□□□
□□□□□□□□□,□□□□ N. □DFS□□□□□□□□□□□, □□□□□□□□□□□ O(log N)
BFS□□, Queue□□□□□□□□□□□□□□, □□□□□□□□□□□□□□□□□□□□□□□□, □□□ N/2. □□□□□ O(N)
DFS□□□□□□□□□□□□□□□□Stack□□□□□□DFS□□□□□□□□□□□□□□□□□□□
```

7.18 0049.Group Anagrams

Go | Group Anagrams | LeetCode | Medium

7.18.1 0049.Grop Anagrams

□□

Given an array of strings strs, group the anagrams together. You can return the answer in any order.

An Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1:

Input: strs = ["eat","tea","tan","ate","nat","bat"] Output: [["bat"],["nat","tan"],["ate","eat","tea"]]

Example 2:

Input: strs = [""] Output: [[""]]

Example 3:

Input: strs = ["a"] Output: [["a"]]

Constraints:

- 1 <= strs.length <= 104
- 0 <= strs[i].length <= 100
- strs[i] consists of lowercase English letters.

□□□□

□□□□□□□□□□□□□□□□

□□□□

□□□: □□ □□□□□□□□□□□□□□□□□

```go
    m := make(map[[26]int][]string, len(strs))

    for _, str := range strs {
        // □□□□□string□□□`count`, □□□□□□□□□□□□
        count := [26]int{}
        for _, b := range str {
            count[b-'a']++
        }
        // □□□□□□□string□□□ map□
        m[count] = append(m[count], str)
    }

    // □map□□string□□□□□□ `[][]string` array □
    ans := [][]string{}
    for _, v := range m {
        ans = append(ans, v)
    }
    return ans
}

// □□: □□□□ O(nklog(k)), □□□□ O(klog(k))
// n □□□□□□ , k□□□□□□□□
// □□□□ O(n)
// □□□□ O(klog(k))
func GroupAnagramsBySort(strs []string) [][]string {
    m := make(map[string][]string, len(strs))

    for _, str := range strs {
        // □□□□□string□□□, □□□□□□
        s := []byte(str)
        sort.Slice(s, func(i, j int) bool { return s[i] < s[j] })
        sortedStr := string(s)

        // □□□□□□□string□□□ map□
        m[sortedStr] = append(m[sortedStr], str)
    }

    // □map□□string□□□□□□ `[][]string` array □
    ans := [][]string{}
    for _, v := range m {
        ans = append(ans, v)
    }
    return ans
}
```

Benchmark

```
go test -benchmem -run=none LeetcodeGolang/Leetcode/0049.Group-Anagrams -bench=.
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/0049.Group-Anagrams
cpu: Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz
BenchmarkGroupAnagrams-8                  899431              1615 ns/op             968 B/op         12 allocs/op
BenchmarkGroupAnagramsBySort-8            592148              3566 ns/op             776 B/op         33 allocs/op
PASS
ok      LeetcodeGolang/Leetcode/0049.Group-Anagrams     3.611s
```

```go
package groupanagrams

import (
    "reflect"
    "sort"
    "testing"
)

var tests = []struct {
    arg1 []string
    want [][]string
}{
    {
        []string{"eat", "tea", "tan", "ate", "nat", "bat"},
        [][]string{
            {"bat"}, {"nat", "tan"}, {"ate", "eat", "tea"}},
    },
    {
        []string{},
        [][]string{},
    }, {
        []string{"a"},
        [][]string{
            {"a"}},
    },
}

// □□□□□□□□□□□□□□□□□□□□
func sortSubStrings(groups [][]string) {
    for _, group := range groups {
        sort.Strings(group)
    }
    // □□□□□□
```

```go
    sort.Slice(groups, func(i, j int) bool {
        return groups[i][0] < groups[j][0]
    })
}

func TestGroupAnagrams(t *testing.T) {
    for _, tt := range tests {
        result := GroupAnagrams(tt.arg1)
        // □□□□□□□□□□□□
        sortSubStrings(result)

        // □□□□□□□□□□□□□□□
        sortSubStrings(tt.want)

        // □□□□□□□□□□□□□□□□
        if !reflect.DeepEqual(result, tt.want) {
            t.Errorf("got = %v, want = %v", result, tt.want)
        }
    }
}

func TestGroupAnagramsBySort(t *testing.T) {
    for _, tt := range tests {
        result := GroupAnagramsBySort(tt.arg1)
        // □□□□□□□□□□□□
        sortSubStrings(result)

        // □□□□□□□□□□□□□□□
        sortSubStrings(tt.want)

        // □□□□□□□□□□□□□□□□
        if !reflect.DeepEqual(result, tt.want) {
            t.Errorf("got = %v, want = %v", result, tt.want)
        }
    }
}

func BenchmarkGroupAnagrams(b *testing.B) {
    b.ResetTimer()
    for i := 0; i < b.N; i++ {
        GroupAnagrams(tests[0].arg1)
    }
}

func BenchmarkGroupAnagramsBySort(b *testing.B) {
    b.ResetTimer()
    for i := 0; i < b.N; i++ {
        GroupAnagramsBySort(tests[0].arg1)
    }
}
```

7.19 0053.Maximum Subarray

title: 0053.Maximum Subarray subtitle: "https://leetcode.com/problems/maximum-subarray" date: 2024-02-23T25:04:00+08:00 lastmod: 2024-02-23T25:04:00+08:00 draft: false author: "Kimi.Tsai" authorLink: "https://kimi0230.github.io/" description: "0053.Maximum-Subarray" license: "" images: []

tags: - LeetCode - Go - Medium - Array - Dynamic Programming - Blind75 categories: [LeetCode]

featuredImage: "" featuredImagePreview: ""

hiddenFromHomePage: false hiddenFromSearch: false twemoji: false lightgallery: true ruby: true fraction: true fontawesome: true linkToMarkdown: false rssFullText: false

toc: enable: true auto: true code: copy: true maxShownLines: 200 math: enable: false # ... mapbox: # ... share: enable: true # ... comment: enable: true # ... library: css: # someCSS = "some.css" # located in "assets/" # Or # someCSS = "https://cdn.example.com/some.css" js: # someJS = "some.js" # located in "assets/" # Or # someJS = "https://cdn.example.com/some.js" seo: images: [] # ...

7.19.1 53. Maximum Subarray

□□

Given an integer array `nums`, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

Example: Input: [-2,1,-3,4,-1,2,1,-5,4], Output: 6 Explanation: [4,-1,2,1] has the largest sum = 6.

Follow up: If you have figured out the O(*n*) solution, try coding another solution using the divide and conquer approach, which is more subtle.

□□□□

□□□□□□□□ nums □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

- □□□□□□ DP □□□□□□□ DP□
- □□□□□□□□□□□□□□□□□□□□□□□□□□ `dp[i]` □□ `[0,i]` □□□□□□□□□□□□□□□□□□□□□□□ `dp[i] = nums[i] + dp[i-1] (dp[i- 1] > 0)` □ `dp[i] = nums[i] (dp[i-1] ≤ 0)` □

□□

- https://books.halfrost.com/leetcode/ChapterFour/0001~0099/0053.Maximum-Subarray/
- https://leetcode-cn.com/problems/maximum-subarray/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0053.Maximum-Subarray/Maximum-Subarray.go

```go
package maximumsubarray

// MaxSubArrayDP : DP (dynamic programming)
func MaxSubArrayDP(nums []int) int {
    if len(nums) == 0 {
        return 0
    }

    if len(nums) == 1 {
```

```go
        return nums[0]
    }

    dp, res := make([]int, len(nums)), nums[0]
    dp[0] = nums[0]

    for i := 1; i < len(nums); i++ {
        if dp[i-1] > 0 {
            // □□□□□□□ □□□□□
            dp[i] = nums[i] + dp[i-1]
        } else {
            // □□□□□□□□□0 □□□□□□□□
            dp[i] = nums[i]
        }
        res = max(res, dp[i])
    }
    return res
}

// MaxSubArray1 : □□, □DP□
func MaxSubArray1(nums []int) int {
    if len(nums) == 1 {
        return nums[0]
    }

    maxSum := 0
    tmp := 0
    for i := 0; i < len(nums); i++ {
        tmp += nums[i]
        if tmp > maxSum {
            maxSum = tmp
        }
        if tmp < 0 {
            tmp = 0
        }
    }
    return maxSum
}

func max(a int, b int) int {
    if a > b {
        return a
    }
    return b
}

//TODO: □□□□, □□□□□□□□□□□□□□□□□□□□□□□□□□□pushUp□□
```

7.20 0059.Spiral Matrix II

Array **Medium**

7.20.1 59. Spiral Matrix II

□□

Given a positive integer *n*, generate a square matrix filled with elements from 1 to *n*2 in spiral order.

Example:

```
Input: 3
Output:
[
 [ 1, 2, 3 ],
 [ 8, 9, 4 ],
 [ 7, 6, 5 ]
]
```

□□□□

□□□□□□□ n□□□□□□□ 1 □ n^2 □□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

- □□□□□□ n□□□□□□□ n * n □□□□□□□□□□□ 1 - n*n□□□□□□□□□□□□□□□
- □□□□□ 54 □□□□□□□□□□□□□□□□□□□□□□□□□□□

□□

- https://books.halfrost.com/leetcode/ChapterFour/0001~0099/0059.Spiral-Matrix-II/
- https://leetcode-cn.com/problems/spiral-matrix-ii/
- □□□□□□□□□□□□□□□□

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0059.Spiral-Matrix-II/Spiral-Matrix-II.go

```go
package spiralmatrixii

// GenerateMatrix : □□□□, □□□□ O(n^2) □□□□ O(1)
func GenerateMatrix(n int) [][]int {
    result := make([][]int, n)
    for i := range result {
        result[i] = make([]int, n)
    }

    left, right, top, botton := 0, n-1, 0, n-1
    num := 1
    target := n * n

    for num <= target {
        // □□  left to right, □□□□++
        for i := left; i <= right; i++ {
            result[top][i] = num
            num++
        }
        top++

        // □□ top to botton , □□□□--
        for i := top; i <= botton; i++ {
            result[i][right] = num
            num++
        }
        right--
```

```go
        // □□ right to left , □□□□--
        for i := right; i >= left; i-- {
            result[botton][i] = num
            num++
        }
        botton--

        // □□  botton to top, □□□□++
        for i := botton; i >= top; i-- {
            result[i][left] = num
            num++
        }
        left++
    }

    return result
}

// □□ : O(n)
// https://books.halfrost.com/leetcode/ChapterFour/0001~0099/0059.Spiral-Matrix-II/
func GenerateMatrix2(n int) [][]int {
    if n == 0 {
        return [][]int{}
    }
    if n == 1 {
        return [][]int{{1}}
    }
    result, visit, round, x, y, spDir := make([][]int, n), make([][]int, n), 0, 0, 0, [][]int{
        {0, 1},  // □□
        {1, 0},  // □□
        {0, -1}, // □□
        {-1, 0}, // □□
    }
    for i := 0; i < n; i++ {
        visit[i] = make([]int, n)
        result[i] = make([]int, n)
    }
    visit[x][y] = 1
    result[x][y] = 1

    for i := 0; i < n*n; i++ {
        x += spDir[round%4][0]
        y += spDir[round%4][1]
    }

    return result
}
```

7.21 0070.Climbing Stairs

Climbing Stairs **DP** **Easy** **Go** **LeetCode**

7.21.1 0070.Climbing Stairs

□□

You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb `1` or `2` steps. In how many distinct ways can you climb to the top?

Example 1:

Input: n = 2 Output: 2 Explanation: There are two ways to climb to the top. 1. 1 step + 1 step 2. 2 steps Example 2:

Input: n = 3 Output: 3 Explanation: There are three ways to climb to the top. 1. 1 step + 1 step + 1 step 2. 1 step + 2 steps 3. 2 steps + 1 step

Constraints:

1 <= n <= 45

- Accepted: 2.8M
- Submissions: 5.4M
- Acceptance Rate: 52.3%

□□□□

□□ Fibonacci Number

□□□□

- □□□ DP□□□□□□□□□. □□□□□□□ n-1 □ n-2 □□□□□□□
- □□□□□□□□□□□□□□□□

Big O

□□□□ : `□□□□ :`

□□

- https://leetcode.com/problems/climbing-stairs/
- https://leetcode.cn/problems/climbing-stairs/
- https://books.halfrost.com/leetcode/ChapterFour/0001~0099/0070.Climbing-Stairs/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0070.Climbing-Stairs/main.go

```go
package climbingstairs

// □□□□ O(n), □□□□ O(n)
func ClimbStairs(n int) int {
    dp := make([]int, n+1)
    dp[0], dp[1] = 1, 1

    for i := 2; i <= n; i++ {
```

```go
        dp[i] = dp[i-1] + dp[i-2]
    }
    return dp[n]
}

func ClimbStairsCache(n int) int {
    dp := make([]int, n+1)
    dp[0], dp[1] = 1, 1

    for i := 2; i <= n; i++ {
        if val := dp[i]; val == 0 {
            dp[i] = dp[i-1] + dp[i-2]
        }
    }
    return dp[n]
}

func ClimbStairsRecursive(n int) int {
    dp := make([]int, n+1)
    // dp[0], dp[1] = 1, 1

    var climbClosure func(n int) int
    climbClosure = func(n int) int {
        if n <= 2 {
            return n
        }
        dp[n] = climbClosure(n-1) + climbClosure(n-2)
        return dp[n]
    }
    return climbClosure(n)
}
```

Benchmark

```
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/0070.Climbing-Stairs
cpu: Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz
BenchmarkClimbStairs-8                  10386211               112.1 ns/op           320 B/op          1 allocs/op
BenchmarkClimbStairsCache-8             10184984               118.8 ns/op           320 B/op          1 allocs/op
BenchmarkClimbStairsRecursive-8                4         281980486 ns/op             320 B/op          1 allocs/op
PASS
ok      LeetcodeGolang/Leetcode/0070.Climbing-Stairs    5.591s
```

7.22 0072.Edit Distance

Dynamic Programming **Edit Distance** **Golang** **Hard**

7.22.1 0072. Edit Distance

□□

Given two strings word1 and word2, return the minimum number of operations required to convert word1 to word2.

You have the following three operations permitted on a word:

- Insert a character
- Delete a character
- Replace a character

Example 1:

```
Input: word1 = "horse", word2 = "ros"
Output: 3
Explanation:
horse -> rorse (replace 'h' with 'r')
rorse -> rose (remove 'r')
rose -> ros (remove 'e')
```

Example 2:

```
Input: word1 = "intention", word2 = "execution"
Output: 5
Explanation:
intention -> inention (remove 't')
inention -> enention (replace 'i' with 'e')
enention -> exention (replace 'n' with 'x')
exention -> exection (replace 'n' with 'c')
exection -> execution (insert 'u')
```

Constraints:

- 0 <= word1.length, word2.length <= 500
- word1 and word2 consist of lowercase English letters.

□□□□

□□□□□□□□□□□□□□□, □□, □□, □□ □□□□□□□□□□word1,word2, □□□word1□□□word2□□□□□□□□□

□□□□

https://mp.weixin.qq.com/s/ShoZRjM8OyvDbZwoXh6ygg □□□□□□□□□□□□□□□, □□□□□□□□□ i, j □□□□□□□□□□□□□□, □□□□□□□□□, □□□□□□□□

```
if word1[i] == word2[j]:
    skip
    i,j□□□□
else:
    # insert
    # delete
    # replace
```

```
dp = [
    [0,1,2,3,4,5],
    [1,1,2,2,3,4],
    [2,2,1,2,3,4],
```

```
    [3,3,2,2,2,3]
]
```

word1 \ word2	""	h	o	r	s	e
""	0	1	2	3	4	5
r	1	1	2	2	3	4
o	2	2	1	2	3	4
s	3	3	2	2	2	3

dp(i,j) □□□, □□ word1[0..i] □ word2[0..j]□□□□□□□ dp(1,0) "ro" , "h" □□□□□□ 2

dp[i-1][j-1]	dp[i-1][j]
dp[i][j-1]	dp[i][j]

□□/□□	□□
□□	

□□

- https://leetcode.com/problems/edit-distance/
- https://labuladong.gitee.io/algo/3/25/77/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0072.Edit-Distance/main.go

```go
package editdistance

import "fmt"

// □□ (□□□)
func MinDistance(word1 string, word2 string) int {
    var dp func(int, int) int
    dp = func(i, j int) int {
        // base case
        if i == -1 {
            return j + 1
        }
        if j == -1 {
            return i + 1
        }
        if word1[i] == word2[j] {
            // word1[0..i] □ word2[0..j]□□□□□□□□□ word1[0..i-1] □ word2[0..j-1]
            // □□□□□□□□□□□□□□
            // □□□□ dp(i,j)□□ dp(i-1,j-1)
            return dp(i-1, j-1)
        } else {
            return min(
                dp(i, j-1)+1,   // insert: □□□ word1[i]□□□□□□word2[j]□□□□□, □□word2[j]□□□□□,□□j, □□□i□□, □□□□+1
                dp(i-1, j)+1,   // delete: □□□ word1[i] □□□□□□□, □□ i □□□ j □□, □□□□+1
                dp(i-1, j-1)+1, // replace: □□□ word1[i] □□□ word2[j], □□□□□□□□, □□□□ i, j □□□□, □□□□+1
            )
        }
    }

    return dp(len(word1)-1, len(word2)-1)
}

// Memo□□
func MinDistanceMemo(word1 string, word2 string) int {
    var dp func(int, int) int
    memo := map[string]int{}
    dp = func(i, j int) int {
        key := fmt.Sprintf("%d,%d", i, j)
        // □□□□□ □□□□
        if _, ok := memo[key]; ok == true {
            return memo[key]
        }
```

```go
        // base case
        if i == -1 {
            return j + 1
        }
        if j == -1 {
            return i + 1
        }
        if word1[i] == word2[j] {
            // word1[0..i] ▫ word2[0..j]▫▫▫▫▫▫▫▫▫ word1[0..i-1] ▫ word2[0..j-1]
            // ▫▫▫▫▫▫▫▫▫▫▫▫▫▫
            // ▫▫▫▫ dp(i,j)▫▫ dp(i-1,j-1)
            memo[key] = dp(i-1, j-1)
        } else {
            memo[key] = min(
                dp(i, j-1)+1,   // insert: ▫▫▫ word1[i]▫▫▫▫▫▫word2[j]▫▫▫▫▫, ▫▫word2[j]▫▫▫▫▫,▫▫j, ▫▫▫i▫▫, ▫▫▫▫+1
                dp(i-1, j)+1,   // delete: ▫▫▫ word1[i] ▫▫▫▫▫▫▫, ▫▫ i ▫▫▫ j ▫▫, ▫▫▫▫+1
                dp(i-1, j-1)+1, // replace: ▫▫▫ word1[i] ▫▫▫ word2[j], ▫▫▫▫▫▫▫▫, ▫▫▫▫ i, j ▫▫▫▫, ▫▫▫▫+1
            )
        }
        return memo[key]
    }

    return dp(len(word1)-1, len(word2)-1)
}

// DP table ▫▫, DP table ▫▫▫▫▫▫▫, ▫▫▫▫▫▫▫▫▫
func MinDistanceDP(word1 string, word2 string) int {
    m, n := len(word1), len(word2)
    // ▫▫▫  dp table : [][]int{}
    dp := make([][]int, m+1)
    for i := 0; i < m+1; i++ {
        dp[i] = make([]int, n+1)
    }

    // base case
    for i := 1; i <= m; i++ {
        dp[i][0] = i
    }
    for j := 1; j <= n; j++ {
        dp[0][j] = j
    }

    // ▫▫▫▫
    for i := 1; i <= m; i++ {
        for j := 1; j <= n; j++ {
            if word1[i-1] == word2[j-1] {
                dp[i][j] = dp[i-1][j-1]
            } else {
                dp[i][j] = min(
                    dp[i][j-1]+1,   // insert
                    dp[i-1][j]+1,   // delete
                    dp[i-1][j-1]+1, // replace
                )
            }
        }
    }
    return dp[m][n]
}

type Number interface {
    int | int64 | float64
}

func min[T Number](vars ...T) T {
    min := vars[0]

    for _, i := range vars {
        if min > i {
            min = i
        }
    }

    return min
}
```

```
go test -benchmem -run=none LeetcodeGolang/Leetcode/0072.Edit-Distance -bench=.
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/0072.Edit-Distance
cpu: Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz
BenchmarkMinDistance-8            398260              3748 ns/op               0 B/op          0 allocs/op
BenchmarkMinDistanceMemo-8        102272             10796 ns/op            2211 B/op         69 allocs/op
BenchmarkMinDistanceDP-8         1944886               794.2 ns/op           688 B/op          9 allocs/op
PASS
ok      LeetcodeGolang/Leetcode/0072.Edit-Distance      5.717s
```

7.23 0074.Search a 2D Matrix

Amazon Array Binary Search Bloomberg Facebook Go LeetCode Matrix Medium Microsoft Search a 2D Matrix

7.23.1 74. Search a 2D Matrix

□□

□□□□

□□□□□□□□□□□□□ m x n □□□□□□□□□□□□□□□ □□□□□□□□□□□ * □□□□□□□□□□□□□□□□□□ * □□□□□□□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□

Big O

- □□□□ : `O(logmn)` □□ m □ n □□□□□□□□□□□□
- □□□□ : `O(1)`

□□

- https://leetcode.com/problems/search-a-2d-matrix/description/
- https://leetcode.cn/problems/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0074.Search-a-2D-Matrix/main.go

```go
package searcha2dmatrix

// □□□□ O(logmn), □□□□ O(1)
func searchMatrix(matrix [][]int, target int) bool {
    m := len(matrix)
    if m <= 0 {
        return false
    }
    n := len(matrix[0])

    left, right := 0, m*n-1
    for left <= right {
        mid := int(uint(right+left) >> 1) //left + (right-left)/2
        val := getMatrix(matrix, mid)
        if val == target {
            return true
        } else if val < target {
            left = mid + 1
        } else {
            right = mid - 1
        }
    }
    return false
}

// □ mxn□□□□□ □□□□□□□
func getMatrix(matrix [][]int, index int) int {
    n := len(matrix[0])
    i, j := index/n, index%n
    return matrix[i][j]
}
```

Benchmark

7.24 0075.Sort Colors

Medium **Sort**

7.24.1 75. Sort Colors

□□

Given an array with n objects colored red, white or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.

Note: You are not suppose to use the library's sort function for this problem.

Example 1:

```
Input: [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]
```

Follow up:

- A rather straight forward solution is a two-pass algorithm using counting sort.
  First, iterate the array counting number of 0's, 1's, and 2's, then overwrite array with total number of 0's, then 1's and followed by 2's.
- Could you come up with a one-pass algorithm using only constant space?

□□□□□□□□□□□□□□□□ n □□□□□□□□□□□□□□□□□

□□□□

□□□□□ Follow up □□□□□□

```
            nums[i], nums[tail] = nums[tail], nums[i]
            tail--
        }
    }
    sortColorsQuickSort(nums[:equalHead])
    sortColorsQuickSort(nums[tail+1:])
}
```

7.25 0078.Subsets

Go LeetCode Medium Subsets

7.25.1 0078. Subsets

□□

Given an integer array nums of unique elements, return all possible subsets (the power set).

The solution set must not contain duplicate subsets. Return the solution in any order.

Example 1:

Input: nums = [1,2,3] Output: [[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]] Example 2:

Input: nums = [0] Output: [[],[0]]

Constraints:

1 <= nums.length <= 10 -10 <= nums[i] <= 10 All the numbers of nums are unique.

□□□□

□□□□□□□□□□□□□□□ nums□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□ □ DFS □□□□□□□□ □□□□□ 90 □□□ 491 □□□□□□□□□□□□□□

Big O

□□□□ : `O(n^2)` □□□□ : `O(n)`

□□

- https://leetcode.com/problems/subsets/description/
- https://leetcode.cn/problems/subsets/description/
- https://books.halfrost.com/leetcode/ChapterFour/0001~0099/0078.Subsets/
- https://github.com/suntong/lang/blob/master/lang/Go/src/ds/Recursion/LC-0B-Subsets-2.go

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0078.Subsets/main.go

```go
package subsets

// □□□□ O(n^2)  , □□□□ O(n)
func Subsets(nums []int) [][]int {
    path, result := []int{}, [][]int{}
    for i := 0; i <= len(nums); i++ {
        genSubsets(nums, i, 0, path, &result)
    }
    return result
}

func genSubsets(nums []int, elemSize, start int, path []int, result *[][]int) {
    if len(path) == elemSize {
        b := make([]int, len(path))
        copy(b, path)
        *result = append(*result, b)
```

```go
        return
    }

    for k := start; k < len(nums); k++ {
        path = append(path, nums[k])
        genSubsets(nums, elemSize, k+1, path, result)
        path = path[:len(path)-1]
    }
}

// □□□□ O(n^2)  , □□□□ O(n)
func SubsetsDFS(nums []int) [][]int {
    path, result := []int{}, [][]int{}
    genSubsetsDFS(nums, path, &result)
    return result
}

func genSubsetsDFS(nums []int, path []int, result *[][]int) {
    b := make([]int, len(path))
    copy(b, path)
    *result = append(*result, b)
    for i := 0; i < len(nums); i++ {
        // n□ element □slice□
        genSubsetsDFS(nums[i+1:], append(path, nums[i]), result)
    }
}
```

Benchmark

```
go test -benchmem -run=none LeetcodeGolang/Leetcode/0354.Russian-Doll-Envelopes -bench=. -cover
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/0354.Russian-Doll-Envelopes
cpu: Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz
BenchmarkMaxEnvelopes-8          7799131               160.3 ns/op            88 B/op          3 allocs/op
BenchmarkMaxEnvelopes2-8         5800399               195.6 ns/op            80 B/op          4 allocs/op
PASS
coverage: 96.0% of statements
ok      LeetcodeGolang/Leetcode/0354.Russian-Doll-Envelopes     3.726s
```

7.26 0088.Merge Sorted Array

Array **Easy**

7.26.1 88. Merge Sorted Array

□□

Given two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one sorted array.

Note:

You may assume that nums1 has enough space (size that is greater or equal to m + n) to hold additional elements from nums2. The number of elements initialized in nums1 and nums2 are m and n respectively.

Example:

Input: nums1 = [1,2,3,0,0,0], m = 3 nums2 = [2,5,6], n = 3

Output: [1,2,2,3,5,6] Constraints:

-10^9 <= nums1[i], nums2[i] <= 10^9 nums1.length == m + n nums2.length == n

□□□□

□□□□

```
    }
}
```

7.27 0094.Binary Tree Inorder Traversal

Medium **Stack**

7.27.1 94. Binary Tree Inorder Traversal

□□

Given a binary tree, return the inorder traversal of its nodes' values.

Example :

```
Input: [1,null,2,3]
   1
    \
     2
    /
   3

Output: [1,3,2]
```

Follow up: Recursive solution is trivial, could you do it iteratively?

□□□□

□□□□□□□

□□□□

□□□□ □□□□ □□□□□□□□□, □□□□□□□□, □□□□□□□□□□ -> □□□ -> □□□ □□□□, □□□□ □□□□□□□□□, □□□□□□-> □□□ -> □□□ □□□□, □□□□ □□□□□□□□□, □□□□□□□□□□□□□□□□ □□□□□□□□□, □□□□□□□□

□□

- https://books.halfrost.com/leetcode/ChapterFour/0001~0099/0094.Binary-Tree-Inorder-Traversal/
- https://leetcode-cn.com/problems/binary-tree-inorder-traversal/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0094.Binary-Tree-Inorder-Traversal/Binary-Tree-Inorder-Traversal.go

```go
package binarytreeinordertraversal

import (
    "LeetcodeGolang/Utility/structures"
)

type TreeNode = structures.TreeNode

func InorderTraversalStack(root *TreeNode) []int {
    var result []int
    inorderStack(root, &result)
    return result
}

func InorderTraversalRecursion(root *TreeNode) []int {
    var result []int
    inorderRecursion(root, &result)
    return result
}

func inorderRecursion(root *TreeNode, r *[]int) {
    if root != nil {
        inorderRecursion(root.Left, r)
        *r = append(*r, root.Val)
        inorderRecursion(root.Right, r)
```

```go
    }
}

func inorderStack(root *TreeNode, r *[]int) {
    s := structures.NewArrayStack()
    p := root

    for !s.IsEmpty() || p != nil {
        if p != nil {
            // □□□□□stack, □□□□□□□□□
            s.Push(p)
            p = p.Left
        } else {
            // □□□□,□□
            tmp := s.Pop().(*TreeNode)
            *r = append(*r, tmp.Val)
            p = tmp.Right
        }
    }
}
```

7.28 0100.Same Tree

Easy Go LeetCode Same Tree Tree

7.28.1 100. Same Tree

Given the roots of two binary trees p and q, write a function to check if they are the same or not.

Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

Example 1:

Input: p = [1,2,3], q = [1,2,3] Output: true

Example 2:

Input: p = [1,2], q = [1,null,2] Output: false

Example 3:

Input: p = [1,2,1], q = [1,1,2] Output: false

Constraints:

The number of nodes in both trees is in the range [0, 100]. -104 <= Node.val <= 104

- Accepted: 1.8M
- Submissions: 3.1M
- Acceptance Rate: 60.3%

□□□□

□□ 2 □□□□□□□□□□

□□□□

□□□□□□

Big O

□□□□: `O(n)` □□□□: `O(1)`

□□

- https://leetcode.com/problems/same-tree/
- https://leetcode.cn/problems/same-tree/
- https://books.halfrost.com/leetcode/ChapterFour/0100~0199/0100.Same-Tree/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0100.Same-Tree/main.go

```go
package sametree

import "LeetcodeGolang/structures"

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

type TreeNode = structures.TreeNode

// □□□□ O(n), □□□□ O(1)
func IsSameTree(p *TreeNode, q *TreeNode) bool {
    if p == nil && q == nil {
        return true
    } else if p != nil && q != nil {
        if p.Val == q.Val {
            return IsSameTree(p.Left, q.Left) && IsSameTree(p.Right, q.Right)
        }
        return false
    } else {
        return false
    }
}
```

Benchmark

7.29 0104.Maximum Depth of Binary Tree

Easy **Go** **LEETCODETITLE** **LeetCode**

7.29.1 0104.Maximum Depth of Binary Tree

□□

Given the root of a binary tree, return its maximum depth.

A binary tree's maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

Example 1:

Input: root = [3,9,20,null,null,15,7] Output: 3 Example 2:

Input: root = [1,null,2] Output: 2

Constraints:

The number of nodes in the tree is in the range [0, 104]. -100 <= Node.val <= 100

Accepted 2.7M Submissions 3.6M Acceptance Rate 74.7%

□□□□

□□□□

Big O

□□□□: □□□□ :

□□

- https://leetcode.com/problems/maximum-depth-of-binary-tree/description/
- https://leetcode.cn/problems/maximum-depth-of-binary-tree/description/

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0104.Maximum-Depth-of-Binary-Tree/main.go

Benchmark

7.30 0105.Construct Binary Tree from Preorder and Inorder Traversal

Array | Binary Tree | Construct Binary Tree from Preorder and Inorder Traversal | Divide and Conquer | Go | Hash Table | LeetCode | Medium | Tree

7.30.1 105. Construct Binary Tree from Preorder and Inorder Traversal

□□

□□□□

□□□□

Big O

- □□□□ : `O(n)` n□□□□
- □□□□ : `O(n)` □□□□□Stack□□□ O(h)□□□ h □□□□□□ □□□□□□□□□□□□□□□ TreeNode □□□□□□□□□□□□□ O(n)□□□ n □□□□□□□ h< n□□□□□□□ O(n)

□□

- https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/description/
- https://leetcode.cn/problems/construct-binary-tree-from-preorder-and-inorder-traversal/description/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0105.Construct-Binary-Tree-from-Preorder-and-Inorder-Traversal/main.go

```go
/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
// □□□□ O(), □□□□ O()
func buildTree(preorder []int, inorder []int) *TreeNode {
    if len(preorder) == 0 {
        return nil
    }
    result := &TreeNode{preorder[0], nil, nil}

    i := 0
    for ; i < len(inorder); i++ {
        if preorder[0] == inorder[i] {
            break
        }
    }
    result.Left = buildTree(preorder[1:i+1], inorder[:i])
    result.Right = buildTree(preorder[i+1:], inorder[i+1:])

    return result
}
```

$$x-(preLeft+1)=pIndex-1-inLeft$$

$$\Rightarrow x = pIndex - inLeft + preLeft$$

```
result.Left = buildTree(preorder[1 : len(inorder[:i])+1]
                        , inorder[:i])
result.Right = buildTree(preorder[len(inorder[:i])+1 :],
                         inorder[i+1:]
```

Benchmark

7.31 0110.Balanced Binary Tree

Balanced Binary Tree **DFS** **Easy** **Go** **LeetCode**

7.31.1 110. Balanced Binary Tree

Given a binary tree, determine if it is height-balanced .

Example 1:

Input: root = [3,9,20,null,null,15,7] Output: true

Example 2:

Input: root = [1,2,2,3,3,null,null,4,4] Output: false Example 3:

Input: root = [] Output: true

Constraints:

The number of nodes in the tree is in the range [0, 5000]. -104 <= Node.val <= 104

□□□□

□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ <= 1 □□□□□□

□□□□

□□□□□□□ 0104.Maximum-Depth-of-Binary-Tree

□□□□□□Balanced Binary Tree□□□□□□□□□□□□□□□□□□□

7.32 0121.Best Time to Buy and Sell Stock

Best Time to Buy and Sell Stock **Blind75** **DP** **Easy** **Go** **LeetCode** **Slide Windows**

amazon **array** **bloomberg** **facebook** **microsoft** **uber**

7.32.1 0121.Best Time to Buy and Sell Stock

□□

You are given an array prices where prices[i] is the price of a given stock on the ith day.

You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock.

Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.

Example 1:

Input: prices = [7,1,5,3,6,4] Output: 5 Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5. Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell. Example 2:

Input: prices = [7,6,4,3,1] Output: 0 Explanation: In this case, no transactions are done and the max profit = 0.

Constraints:

1 <= prices.length <= 105 0 <= prices[i] <= 104 Accepted 3,930,287 Submissions 7,349,138

□□□□

□□□□

Big O

□□□□: `O(n)` □□□□: `O(1)`

□□

- https://leetcode.com/problems/valid-parentheses/
- https://leetcode.cn/problems/valid-parentheses/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0121.Best-Time-to-Buy-and-Sell-Stock/main.go

```go
package besttimetobuyandsellstock

type numbers interface {
    int | int8 | int16 | int32 | int64 | float32 | float64
}

func max[T numbers](a T, b T) T {
    if a > b {
        return a
    }
    return b
}

// □□□□ O(n), □□□□ O(1)
// Slide windows
func MaxProfit(prices []int) int {
```

```go
    left, right := 0, 1
    maxProfit := 0
    for right < len(prices) {
        if prices[left] < prices[right] {
            profit := prices[right] - prices[left]
            maxProfit = max(maxProfit, profit)
        } else {
            left = right
        }
        right++
    }
    return maxProfit
}

// □□□□ O(n), □□□□ O(1)
// DP : □□□
func MaxProfitDP(prices []int) int {

    if len(prices) == 0 {
        return 0
    }

    minPrice := prices[0] // □□□□□□□□□□□□
    maxProfit := 0

    for i := 1; i < len(prices); i++ {
        // □□□□□□□□□□□□□□□□□□□
        if prices[i] < minPrice {
            minPrice = prices[i]
        } else {
            // □□□□□□□□□□□□□□□□□□□□□
            profit := prices[i] - minPrice
            if profit > maxProfit {
                maxProfit = profit
            }
        }
    }

    return maxProfit
}

// □□□□ O(), □□□□ O()
// DP
func MaxProfitDP2(prices []int) int {
    n := len(prices)
    dp := make([][]int, n)
    for i := 0; i < n; i++ {
        dp[i] = make([]int, 2)
        if i-1 == -1 {
            // base case
            dp[i][0] = 0
            dp[i][1] = -prices[i]
            continue
        }
        dp[i][0] = max(dp[i-1][0], dp[i-1][1]+prices[i])
        dp[i][1] = max(dp[i-1][1], -prices[i])
    }
    return dp[n-1][0]
}
```

Benchmark

```
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/0121.Best-Time-to-Buy-and-Sell-Stock
cpu: Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz
BenchmarkMaxProfit-8            208734571                8.156 ns/op           0 B/op          0 allocs/op
BenchmarkMaxProfitDP-8          240880467                5.720 ns/op           0 B/op          0 allocs/op
BenchmarkMaxProfitDP2-8          4095866               282.6 ns/op          240 B/op          7 allocs/op
PASS
ok      LeetcodeGolang/Leetcode/0121.Best-Time-to-Buy-and-Sell-Stock    6.732s
```

7.33 0125.Valid Palindrome

Easy **Go** **LeetCode** **Valid Palindrome**

7.33.1 0125. Valid Palindrome

□□

A phrase is a palindrome if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers.

Given a string s, return true if it is a palindrome, or false otherwise.

Example 1:

Input: s = "A man, a plan, a canal: Panama" Output: true Explanation: "amanaplanacanalpanama" is a palindrome. Example 2:

Input: s = "race a car" Output: false Explanation: "raceacar" is not a palindrome. Example 3:

Input: s = " " Output: true Explanation: s is an empty string "" after removing non-alphanumeric characters. Since an empty string reads the same forward and backward, it is a palindrome.

Constraints:

1 <= s.length <= 2 * 105 s consists only of printable ASCII characters.

□□□□

□□□□

□□□□

Big O

□□□□: `O(n)` □□□□: `O(1)`

□□

- https://leetcode.com/problems/valid-palindrome
- https://leetcode.cn/problems/valid-palindrome

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0125.Valid-Palindrome/main.go

```go
package validpalindrome

import (
    "strings"
    "unicode"
)

// □□□□ O(), □□□□ O()
func IsPalindrome(s string) bool {
    // □□□□□□□,
    s = strings.ToLower(s)

    // □□□□□, □□□□□□□□, □□□□
    left, right := 0, len(s)-1
    for left < right {
        for left < right && !isChar(s[left]) {
            left++
        }
```

```go
        for left < right && !isChar(s[right]) {
            right--
        }
        if s[left] != s[right] {
            return false
        }
        left++
        right--
    }
    return true
}

func isChar(c byte) bool {
    if unicode.IsLetter(rune(c)) || unicode.IsDigit(rune(c)) {
        return true
    }
    return false
}

func isCharAZ09(c byte) bool {
    if ('a' <= c && c <= 'z') || ('0' <= c && c <= '9') {
        return true
    }
    return false
}

func IsPalindromeStrBuilder(s string) bool {
    s = strings.ToLower(s)
    strLen := len(s)
    // □□□□□□□, □□□□□□□□□□□□□
    var sb strings.Builder
    for i := 0; i < strLen; i++ {
        c := rune(s[i])
        if unicode.IsLetter(c) || unicode.IsDigit(c) {
            sb.WriteRune(c)
        }
    }

    sbStr := sb.String()

    // □□□□□, □□□□□□□□, □□□□
    left, right := 0, len(sbStr)-1
    for left < right {
        if sbStr[left] != sbStr[right] {
            return false
        }
        left++
        right--
    }
    return true
}
```

Benchmark

```
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/0125.Valid-Palindrome
cpu: Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz
BenchmarkIsPalindrome-8                 3840048               552.2 ns/op            32 B/op          1 allocs/op
BenchmarkIsPalindromeStrBuilder-8       3164848               439.0 ns/op            88 B/op          4 allocs/op
PASS
ok      LeetcodeGolang/Leetcode/0125.Valid-Palindrome   5.242s
```

7.34 0128.Longest Consecutive Sequence

Adobe Amazon Array Go Google Hash Table LeetCode Longest Consecutive Sequence Medium Microsoft Spotify Union Find

7.34.1 128. Longest Consecutive Sequence

□□

□□□□□□□□□□□□□ nums □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□ O(n) □□□□□□□□□□□□

□□□□

□□□□

Big O

□□□□: `O(n)` □□□□: `O(n)`

□□

- https://leetcode.com/problems/longest-consecutive-sequence/description/
- https://leetcode.cn/problems/longest-consecutive-sequence/description/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0128.Longest-Consecutive-Sequence/main.go

```go
package longestconsecutivesequence

// □□□□ O(), □□□□ O()
func longestConsecutive(nums []int) int {
    m := make(map[int]struct{}, len(nums))
    for _, num := range nums {
        m[num] = struct{}{}
    }
    result := 0
    for v := range m {
        // □□□□□□□□□□□□□□□, □□□□□□□□□□□□□□□□□
        if _, ok := m[v-1]; !ok {
            length := 1
            for _, exit := m[v+length]; exit; _, exit = m[v+length] {
                length++
            }
            if result < length {
                result = length
            }
        }
    }
    return result
}
```

Benchmark

7.35 0138.Copy List with Random Pointer

Amazon **Bloomberg** **Copy List with Random Pointer** **Facebook** **Go** **Google** **LeetCode** **Medium** **Microsoft**

7.35.1 138. Copy List with Random Pointer

□□

□□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ + □□□□ * □□□□□□□clone□□□□□ Hash□□□□□□□clone□□□□□□□□□□; * □□□□□□□□□□□□□□□□□□□□□□□clone□□□□□□□□□□□

Big O

- □□□□ : `O(n)` □□□ n □□□□□□□ □□□□□□□□□□□□□□□□□□□□

```
    for cur != nil {
        newNode := &Node{Val: cur.Val}
        cMap[cur] = newNode
        cur = cur.Next
    }
    // □□□□□□□□□□□□□□□□□□□□□□clone□□□□□□□□□□
    newHead := cMap[head]
    for head != nil {
        node := cMap[head]
        node.Next = cMap[head.Next]
        node.Random = cMap[head.Random]
        head = head.Next
    }
    return newHead
}
```

Benchmark

```

```

7.36 0141.Linked List Cycle

Easy **Linked List** **Two Pointers**

7.36.1 141. Linked List Cycle

□□

Given a linked list, determine if it has a cycle in it.

Follow up: Can you solve it without using extra space?

□□□□

□□□□□□□□□□□□□□□□□□□□

□□□□

□ 2 □□□□□□□□□□□□□□□□□□□□□□□□□□□□ 2 □□□□□□□□ 1 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□

- https://books.halfrost.com/leetcode/ChapterFour/0100~0199/0141.Linked-List-Cycle/
- https://leetcode-cn.com/problems/linked-list-cycle/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0141.Linked-List-Cycle/Linked-List-Cycle.go

```go
package linkedlistcycle

type ListNode struct {
    Val  int
    Next *ListNode
}

func HasCycle(head *ListNode) bool {
    fast := head
    slow := head
    for slow != nil && fast != nil && fast.Next != nil {
        fast = fast.Next.Next
        slow = slow.Next
        if slow == fast {
            return true
        }
    }
    return false
}
```

7.37 0142.Linked List CycleII

Linked List **Medium** **Two Pointers**

7.37.1 142. Linked List Cycle II

Given the head of a linked list, return the node where the cycle begins. If there is no cycle, return null.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to (0-indexed). It is -1 if there is no cycle. Note that pos is not passed as a parameter.

Do not modify the linked list.

Example 1:

```
Input: head = [3,2,0,-4], pos = 1
Output: tail connects to node index 1
Explanation: There is a cycle in the linked list, where tail connects to the second node.
```

Example 2:

```
Input: head = [1,2], pos = 0
Output: tail connects to node index 0
Explanation: There is a cycle in the linked list, where tail connects to the first node.
```

Example 3:

```
Input: head = [1], pos = -1
Output: no cycle
Explanation: There is no cycle in the linked list.
```

Constraints:

- The number of the nodes in the list is in the range [0, 104].
- -105 <= Node.val <= 105
- pos is -1 or a valid index in the linked-list.

□□□□

□□linked list □□□□□, □□□□□□□□□□□

□□□□

□□□□□□□□, slow □□ `k` □, □□ `fast`□□□ `2k`，□□□□fast□slow□□□k□(□□□□□□□□) □□□□□□□□□□□□□□ `m`，□□□□□□□ `Head` □□□□ `k-m`

□□

- https://leetcode.com/problems/linked-list-cycle-ii/

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0142.Linked-List-CycleII/main.go

```go
package linkedlistcycleII

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */
type ListNode struct {
    Val  int
    Next *ListNode
}

func DetectCycle(head *ListNode) *ListNode {
    fast, slow := head, head
    for fast != nil && fast.Next != nil {
        fast = fast.Next.Next
        slow = slow.Next
        if slow == fast {
            // □□□□□
            break
        }
    }

    slow = head
    for slow != fast {
        fast = fast.Next
        slow = slow.Next
    }
    return slow
}
```

7.38 0143.Reorder List

/Medium | 143. Reorder List | Adobe | Amazon | Bloomberg | Facebook | Go | LeetCode | Linked List | Microsoft | Recursion | Stack | Two Pointers

7.38.1 143. Reorder List

□□

□□□□

□□□□

Big O

- □□□□ : ``
- □□□□ : ``

□□

- https://leetcode.com/problems/reorder-list/description/
- https://leetcode.cn/problems/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0143.Reorder-List/main.go

```go
package reorderlist

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */

// □□□□ O(), □□□□ O()
// □□□□□□□□Linked list□□□□
// □□□Linked list□□□□
// □□□□□Linked list□□
// □□□□□Linked list merge □□
func reorderList(head *ListNode) {
    mid := middleNode(head)

    // 2.□□□□□□□□□□□□
    right := resverseNode(mid.Next)
    mid.Next = nil
    left := head
    mergeNode(left, right)
}

// [876. Middle of the Linked List](https://leetcode.com/problems/middle-of-the-linked-list/)
func middleNode(head *ListNode) *ListNode {
    slow, fast := head, head
    for fast != nil && fast.Next != nil {
        slow = slow.Next
        fast = fast.Next.Next
    }
    return slow
}

// [206. Reverse Linked List](https://leetcode.com/problems/reverse-linked-list/)
func resverseNode(head *ListNode) *ListNode {
    var pre *ListNode
    for head != nil {
```

```go
        tmp := head.Next
        head.Next = pre
        pre = head
        head = tmp
    }
    return pre
}

func mergeNode(l1, l2 *ListNode) {
    lcur, rcur := l1, l2
    for lcur != nil && rcur != nil {
        lcur.Next, rcur.Next, lcur, rcur = rcur, lcur.Next, lcur.Next, rcur.Next
    }
}
```

Benchmark

```go

```

7.39 0153.Find Minimum in Rotated Sorted Array

Adobe Amazon Facebook Go Goldman Sachs LEETCODETITLE LeetCode Medium Microsoft

7.39.1 0153.Find-Minimum-in-Rotated-Sorted-Array

□□

□□□□

□□□□

Big O

- □□□□: `O(logN)`
- □□□□: `O(1)`

□□

- https://leetcode.com/problems/find-minimum-in-rotated-sorted-array/description/
- https://leetcode.cn/problems/find-minimum-in-rotated-sorted-array/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0153.Find-Minimum-in-Rotated-Sorted-Array/main.go

```go
package findminimuminrotatedsortedarray

// □□□□ O(logN), □□□□ O(1)
func findMin(nums []int) int {
    left, right := 0, len(nums)-1
    for left < right {
        if nums[left] < nums[right] {
            return nums[left]
        }
        mid := int(uint(left+right) >> 1)
        if nums[mid] >= nums[left] {
            // mid □□□□□□□□
            left = mid + 1
        } else {
            right = mid
        }
    }
    return nums[left]
}
```

Benchmark

```
```

7.40 0167.Two Sum II Input Array Is Sorted

Array | Binary Search | Go | LeetCode | Medium | Two Pointers | Two Sum II Input Array Is Sorted

7.40.1 0167.Two Sum II Input Array Is Sorted

□□

Given a 1-indexed array of integers numbers that is already sorted in non-decreasing order, find two numbers such that they add up to a specific target number. Let these two numbers be numbers[index1] and numbers[index2] where 1 <= index1 < index2 < numbers.length.

Return the indices of the two numbers, index1 and index2, added by one as an integer array [index1, index2] of length 2.

The tests are generated such that there is exactly one solution. You may not use the same element twice.

Your solution must use only constant extra space.

Example 1:

Input: numbers = [2,7,11,15], target = 9 Output: [1,2] Explanation: The sum of 2 and 7 is 9. Therefore, index1 = 1, index2 = 2. We return [1, 2]. Example 2:

Input: numbers = [2,3,4], target = 6 Output: [1,3] Explanation: The sum of 2 and 4 is 6. Therefore index1 = 1, index2 = 3. We return [1, 3]. Example 3:

Input: numbers = [-1,0], target = -1 Output: [1,2] Explanation: The sum of -1 and 0 is -1. Therefore index1 = 1, index2 = 2. We return [1, 2].

Constraints:

2 <= numbers.length <= 3 * 104 -1000 <= numbers[i] <= 1000 numbers is sorted in non-decreasing order. -1000 <= target <= 1000 The tests are generated such that there is exactly one solution.

□□□□

□□□□□□□□□ target □□□□□□□□□□□□□□□□ □□□□□□□□□□ 2 □□ □□□□□□□□□ □□□□□□□□□□□□

□□□□

Big O

1. □□1. Two Sun□□
2. O(n^2) □□□□□□□ O(1)□□□□□□□□
3. □□□□□□□ O(n) □□□□□□□ O(n) □□□□□□□□
4. □□□ □□□□□□ O(n)□□□ n □□□□□□□ □□□□□□□□□□□□□□ n □□ □□□□□□ O(1)

□□□□ : `□□□□ :`

□□

- https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/
- https://leetcode.cn/problems/two-sum-ii-input-array-is-sorted/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0167.Two-Sum-II-Input-Array-Is-Sorted/main.go

```go
package twosumiiinputarrayissorted

// □□□. □□□□□□ O(n)□□□ n □□□□□□□ □□□□□□□□□□□□□ n □□
// □□□□□□ O(1)

func TwoSum(numbers []int, target int) []int {
    left, right := 0, len(numbers)-1

    for left < right {
        if numbers[left]+numbers[right] == target {
            return []int{left + 1, right + 1}
        }
        if numbers[left]+numbers[right] < target {
            left++
        } else {
            right--
        }
    }

    return nil
}

// □□ O(n^2) □□□□□□□ O(1)□□□□□□□□
// □□□□□□□□□ O(n) □□□□□□□ O(n) □□□□□□□□

func TwoSum2(numbers []int, target int) []int {
    m := make(map[int]int, len(numbers))

    for i, v := range numbers {
        if j, ok := m[target-v]; ok {
            return []int{j + 1, i + 1}
        }
        m[v] = i
    }
    return nil
}
```

Benchmark

```
go test -benchmem -run=none LeetcodeGolang/Leetcode/0167.Two-Sum-II-Input-Array-Is-Sorted -bench=.
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/0167.Two-Sum-II-Input-Array-Is-Sorted
cpu: Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz
BenchmarkTwoSum-4       35161128                39.74 ns/op           16 B/op          1 allocs/op
BenchmarkTwoSum2-4      19309680                70.21 ns/op           16 B/op          1 allocs/op
PASS
ok      LeetcodeGolang/Leetcode/0167.Two-Sum-II-Input-Array-Is-Sorted   2.866s
```

7.41 0203.Remove Linked List Elements

Easy **Linked List**

7.41.1 203. Remove Linked List Elements

□□

Remove all elements from a linked list of integers that have value val.

Example :

```
Input: 1->2->6->3->4->5->6, val = 6
Output: 1->2->3->4->5
```

□□□□

□□□□□□□□□□□□□□

□□□□

□□□□□□□

□□

- https://books.halfrost.com/leetcode/ChapterFour/0200~0299/0203.Remove-Linked-List-Elements/
- https://leetcode-cn.com/problems/remove-linked-list-elements/
- https://mp.weixin.qq.com/s/slM1CH5Ew9XzK93YOQYSjA

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0203.Remove-Linked-List-Elements/Remove-Linked-List-Elements.go

```go
package removelinkedlistelements

import (
    "LeetcodeGolang/structures"
)

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */

type ListNode = structures.ListNode

// □□□: □□□□□□head□□
func RemoveElements(head *ListNode, val int) *ListNode {
    // □□□□□□head
    for head != nil && head.Val == val {
        head = head.Next
    }
    if head == nil {
        return head
    }
    tmpHead := head
    for tmpHead.Next != nil {
        if tmpHead.Next.Val == val {
            tmpHead.Next = tmpHead.Next.Next
        } else {
            tmpHead = tmpHead.Next
        }
    }
```

```go
    return head
}

/*
□□□ □□□□□□, □□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
return newHead.Next
*/
func RemoveElementsVirtualNode(head *ListNode, val int) *ListNode {
    if head == nil {
        return head
    }
    // □□□□□□ Head □□
    newHead := &ListNode{Val: 0, Next: head}
    preHead := newHead
    curHead := head
    for curHead != nil {
        if curHead.Val == val {
            preHead.Next = curHead.Next
        } else {
            preHead = curHead
        }
        curHead = curHead.Next
    }
    return newHead.Next
}

/*
□□□ □□
*/
func RemoveElementsRecurse(head *ListNode, val int) *ListNode {
    if head == nil {
        return head
    }
    head.Next = RemoveElementsRecurse(head.Next, val)
    if head.Val == val {
        return head.Next
    } else {
        return head
    }
}
```

7.42 0206.Reverse Linked List

Easy | Go | LeetCode | Reverse Linked List

7.42.1 206. Reverse Linked List

Given the head of a singly linked list, reverse the list, and return the reversed list.

Example 1:

```
Input: head = [1,2,3,4,5]
Output: [5,4,3,2,1]
```

Example 2:

```
Input: head = [1,2]
Output: [2,1]
```

Example 3:

```
Input: head = []
Output: []
```

Constraints:

- The number of nodes in the list is the range [0, 5000].
- -5000 <= Node.val <= 5000

Follow up: A linked list can be reversed either iteratively or recursively. Could you implement both?

□□□□

□ Linked List □□

□□□□

□□

- https://leetcode.com/problems/Reverse Linked List/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0000.kkkk/main.go

```go
package reverselinkedlist

import (
    . "LeetcodeGolang/structures"
)

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */

func ReverseList(head *ListNode) *ListNode {
    if head == nil || head.Next == nil {
        return head
    }

    var prev *ListNode

    for head != nil {
        next := head.Next
        head.Next = prev
        prev = head
        head = next
    }

    return prev
}

func ReverseListRecursively(head *ListNode) *ListNode {
    return ListRecursivelyChild(head, nil)
}

func ListRecursivelyChild(current *ListNode, prev *ListNode) *ListNode {
    if current == nil {
        return prev
    }
    next := current.Next
    current.Next = prev
    return ListRecursivelyChild(next, current)
}
```

Benchmark

```
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/0206.Reverse-Linked-List
cpu: Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz
BenchmarkReverseList-8                  1000000000               0.7960 ns/op          0 B/op          0 allocs/op
BenchmarkReverseListRecursively-8       276534334                4.374 ns/op           0 B/op          0 allocs/op
PASS
ok      LeetcodeGolang/Leetcode/0206.Reverse-Linked-List        2.597s
```

7.43 0209.Minimum Size Subarray Sum

Medium **Sliding Window**

7.43.1 209. Minimum Size Subarray Sum

□□

Given an array of n positive integers and a positive integer s, find the minimal length of a contiguous subarray of which the sum ≥ s. If there isn't one, return 0 instead.

Example 1:

```
Input: s = 7, nums = [2,3,1,2,4,3]
Output: 2
Explanation: the subarray [4,3] has the minimal length under the problem constraint.
```

```
Input: s = 4, nums = [1,4,4]
Output: 1
```

```
Input: s = 11, nums = [1,1,1,1,1,1,1,1]
Output: 0
```

Follow up:

If you have figured out the O(n) solution, try coding another solution of which the time complexity is O(n log n).

□□□□

□□□□□□□□□□□□□ s□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ sum>=s□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□[i,j]□□□□□□□□□□□□□□□□□s□□□□□□□j□□□□□□□□□□□□□□sum > s□□□□□□i □□□□□□□□□□□□sum < s□□□□□□□□□□□□□□□□□□□□□□

□□

□□□□□□□ O(n) □□□□□□□□□, □□□□□□□ O(n log(n)) □□□□□□□□□

□□

- https://books.halfrost.com/leetcode/ChapterFour/0200~0299/0209.Minimum-Size-Subarray-Sum/
- https://leetcode-cn.com/problems/minimum-size-subarray-sum/
- https://mp.weixin.qq.com/s/UrZynlqi4QpyLlLhBPglyg

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0209.Minimum-Size-Subarray-Sum/Minimum-Size-Subarray-Sum.go

```go
package minimumsizesubarraysum

import (
    "math"
    "sort"
)

// MinSubArrayLenBurst : □□□ □□□□ O(n^2), □□□□ O(1)
func MinSubArrayLenBurst(target int, nums []int) int {
    lens := len(nums)
```

```go
    if lens <= 0 {
        return 0
    }

    minSize := math.MaxInt32
    for i := 0; i < lens; i++ {
        sum := 0
        for j := i; j < lens; j++ {
            sum += nums[j]
            if sum >= target {
                minSize = min(minSize, j-i+1)
            }
        }
    }
    if minSize == math.MaxInt32 {
        minSize = 0
    }
    return minSize
}

// MinSubArrayLenSlidingWindow : □□□□ □□□□ O(n), □□□□ O(1)
// □□□□
```

```go
        // fmt.Println("i:", i, " tmpTarge", tmpTarge, "  bound:", bound, "  bound-(i-1)", bound-(i-1), " minSize:", minSize)
    }

    if minSize == math.MaxInt32 {
        minSize = 0
    }
    return minSize
}

func min(x, y int) int {
    if x < y {
        return x
    }
    return y
}
```

7.44 0215.Kth Largest Element in an Array

Go **Heap** **Kth Largest Element in an Array** **LeetCode** **Medium** **Priority Queue** **Sorting**

7.44.1 0215. Kth Largest Element in an Array

□□

Given an integer array nums and an integer k, return the kth largest element in the array.

Note that it is the kth largest element in the sorted order, not the kth distinct element.

Can you solve it without sorting?

Example 1:

```
Input: nums = [3,2,1,5,6,4], k = 2
Output: 5
```

Example 2:

```
Input: nums = [3,2,3,1,2,4,5,5,6], k = 4
Output: 4
```

Constraints:

- 1 <= k <= nums.length <= 105
- -104 <= nums[i] <= 104

□□□□

1. □□□□□K□□□□□ (Kth Largest Element in an Array)

   □□□□□□□□□□□□□□□□□□□K□□□□□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□K□□□□□□□□□□□□□□K□□□□□□□□□□□□□□□□□□□

```go
    "container/heap"
    "sort"
)

// □□□□□Heap□□□□
// container/heap□□□□□□□□□□□Queue□
// Heap(□□)□□□□□Complete Binary Tree(□□□□□).
// Go□Heap□□□ □□□□□□□□□□□□□, □□□□□□(index = 0 □□□□□□).
// □□□□□□□□□□□□□□□□□□
func FindKthLargestHeap(nums []int, k int) int {
    if k <= 0 || k > len(nums) {
        return 0
    }

    // □□□□□□
    h := &MinHeap{}
    heap.Init(h)

    // □□□□
    for _, num := range nums {
        if h.Len() < k {
            heap.Push(h, num)
        } else if num > (*h)[0] {
            heap.Pop(h)
            heap.Push(h, num)
        }
    }
    // fmt.Println(h)

    return (*h)[0]
}

// □□□□□ Heap □□□
type MinHeap []int

func (h MinHeap) Len() int {
    return len(h)
}
func (h MinHeap) Less(i, j int) bool {
    // □□□□□
    return h[i] < h[j]
}
func (h MinHeap) Swap(i, j int) {
    h[i], h[j] = h[j], h[i]
}

// Push □ Pop □□□□□□□□□□□□□□□ slice □□□□□□□□□□□
func (h *MinHeap) Push(x interface{}) {
    *h = append(*h, x.(int))
}
func (h *MinHeap) Pop() interface{} {
    old := *h
    n := len(old)
    x := old[n-1]
    *h = old[:n-1]
    return x
}

// □□□□□Sort□□□□k□, □□□
func FindKthLargestSort(nums []int, k int) int {
    sort.Ints(nums)
    return nums[len(nums)-k]
}
```

Benchmark

```
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/0215.Kth-Largest-Element-in-an-Array
cpu: Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz
BenchmarkFindKthLargestHeap-8            6112726               184.9 ns/op            48 B/op          3 allocs/op
BenchmarkFindKthLargestSort-8           18023403                60.38 ns/op           24 B/op          1 allocs/op
PASS
ok      LeetcodeGolang/Leetcode/0215.Kth-Largest-Element-in-an-Array    3.383s
```

7.45 0217.Contains Duplicate

Array Blind75 Contains Duplicate Easy Go LeetCode

7.45.1 0217.Contains-Duplicate

□□

Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.

Example 1:

```
Input: nums = [1,2,3,1]
Output: true
```

Example 2:

```
Input: nums = [1,2,3,4]
Output: false
```

Example 3:

```
Input: nums = [1,1,1,3,3,4,3,2,4,2]
Output: true
```

Constraints:

- 1 <= nums.length <= 105
- -109 <= nums[i] <= 109

□□□□

□□□□

□□

- https://leetcode.com/problems/contains-duplicate/description/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0217.Contains-Duplicate/main.go

```go
package containsduplicate

func ContainsDuplicate(nums []int) bool {
    numsMap := make(map[int]bool, len(nums))
    for _, v := range nums {
        if numsMap[v] {
            return true
        }
        numsMap[v] = true
    }
    return false
}
```

Benchmark

```
```

7.46 0226.Invert Binary Tree

BFS Easy Go Invert Binary Tree LeetCode

7.46.1 226. Invert Binary Tree

□□

Given the root of a binary tree, invert the tree, and return its root.

Example 1:

Input: root = [4,2,7,1,3,6,9] Output: [4,7,2,9,6,3,1]

Example 2:

Input: root = [2,1,3] Output: [2,3,1] Example 3:

Input: root = [] Output: []

Constraints:

- The number of nodes in the tree is in the range [0, 100].
- -100 <= Node.val <= 100

□□□□

□□□□□

□□□□

□□□□□□□□□□□□□□□□□□children□□□□□□□□□□□□□□children□□□□□□□□□□□□children□□children□ □□□□BFS

BFS□□□□□□□□□□□□□□Queue□□□□□□BFS□□□□□□□□□□□□□□□□□□

```go
}

func IntsToTree(nums []int) *TreeNode {
    return BuildTree(nums, 0)
}
```

Benchmark

```
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/0226.Invert-Binary-Tree
cpu: Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz
BenchmarkInvertTree-8            2533011               398.7 ns/op           168 B/op          7 allocs/op
BenchmarkInvertTree2-8           2667645               392.4 ns/op           168 B/op          7 allocs/op
BenchmarkInvertTree3-8           1403001               727.5 ns/op           296 B/op         13 allocs/op
PASS
ok      LeetcodeGolang/Leetcode/0226.Invert-Binary-Tree 4.889s
```

7.47 0238.Product of Array Except Self

Array Blind75 Go LeetCode Medium Prefix Sum Product of Array Except Self

7.47.1 238. Product of Array Except Self

□□

Given an integer array nums, return an array answer such that answer[i] is equal to the product of all the elements of nums except nums[i].

The product of any prefix or suffix of nums is guaranteed to fit in a 32-bit integer.

You must write an algorithm that runs in O(n) time and without using the division operation.

Example 1:

Input: nums = [1,2,3,4] Output: [24,12,8,6] Example 2:

Input: nums = [-1,1,0,-3,3] Output: [0,0,9,0,0]

Constraints:

2 <= nums.length <= 105 -30 <= nums[i] <= 30 The product of any prefix or suffix of nums is guaranteed to fit in a 32-bit integer.

Follow up: Can you solve the problem in O(1) extra space complexity? (The output array does not count as extra space for space complexity analysis.)

□□□□

□□□□

Big O

□□□□: `O(n)` □□□□:``

□□

- https://leetcode.com/problems/product-of-array-except-self/description/
- https://leetcode.cn/problems/product-of-array-except-self/description/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0238.Product-of-Array-Except-Self/main.go

```go
package productofarrayexceptself

// □□□□ O(), □□□□ O()
func productExceptSelf(nums []int) []int {
    result, left, right := make([]int, len(nums)), make([]int, len(nums)), make([]int, len(nums))

    // left □□□□□□□□
    // □□□'0' □□□, □□□□□□□□,□□left[0]=1
    left[0] = 1
    for i := 1; i < len(nums); i++ {
        left[i] = left[i-1] * nums[i-1]
    }

    right[len(nums)-1] = 1
    for i := len(nums) - 2; i >= 0; i-- {
        right[i] = right[i+1] * nums[i+1]
```

```go
    }

    for i := 0; i < len(nums); i++ {
        result[i] = left[i] * right[i]
    }
    return result
}

func productExceptSelf2(nums []int) []int {
    result := make([]int, len(nums))

    result[0] = 1
    for i := 1; i < len(nums); i++ {
        result[i] = result[i-1] * nums[i-1]
    }

    rightProduct := 1
    for i := len(nums) - 1; i >= 0; i-- {
        result[i] = result[i] * rightProduct
        rightProduct = rightProduct * nums[i]
    }

    return result
}
```

Benchmark

```
go test -benchmem -run=none LeetcodeGolang/Leetcode/0238.Product-of-Array-Except-Self -bench=.
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/0238.Product-of-Array-Except-Self
cpu: Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz
BenchmarkProductExceptSelf-8            12772174               158.4 ns/op            96 B/op          3 allocs/op
BenchmarkProductExceptSelf2-8           32292304                63.74 ns/op           32 B/op          1 allocs/op
PASS
ok      LeetcodeGolang/Leetcode/0238.Product-of-Array-Except-Self       4.228s
```

7.48 0242.Valid Anagram

Easy **Go** **LeetCode** **Valid Anagram**

7.48.1 0242.Valid-Anagram

□□□□□

□□

Given two strings s and t, return true if t is an anagram of s, and false otherwise.

An Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1:

Input: s = "anagram", t = "nagaram" Output: true Example 2:

Input: s = "rat", t = "car" Output: false

Constraints:

1 <= s.length, t.length <= 5 * 104 s and t consist of lowercase English letters.

Follow up: What if the inputs contain Unicode characters? How would you adapt your solution to such a case?

□□□□ rune □□□□□□□ Unicode □□□□□□ map[rune]int □□□□□□□□□□□□□□□□□□□□□□□□

□□□□

□□□□

□□□□□□□□□□□□□□□□□□□

```go
    }
    return true
}

func IsAnagram2(s string, t string) bool {
    if len(s) != len(t) {
        return false
    }

    record := make(map[rune]int, len(s))

    for _, v := range s {
        record[v]++
    }
    for _, v := range t {
        record[v]--
    }

    for _, v := range record {
        if v != 0 {
            return false
        }
    }
    return true
}
```

Benchmark

```
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/0242.Valid-Anagram
cpu: Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz
BenchmarkIsAnagram-8             6703497               276.3 ns/op             0 B/op          0 allocs/op
BenchmarkIsAnagram2-8            3660909               318.9 ns/op            48 B/op          2 allocs/op
PASS
ok      LeetcodeGolang/Leetcode/0242.Valid-Anagram      4.498s
```

7.49 0283.Move Zeroes

Easy **Go** **LeetCode** **Move Zeroes**

7.49.1 283. Move Zeroes

□□

Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the non-zero elements.

Note that you must do this in-place without making a copy of the array.

Example 1:

```
Input: nums = [0,1,0,3,12]
Output: [1,3,12,0,0]
```

Example 2:

```
Input: nums = [0]
Output: [0]
```

Constraints: * 1 <= nums.length <= 104 * -231 <= nums[i] <= 231 - 1

Follow up: Could you minimize the total number of operations done?

□□□□

□□□□□□□□□□□□□□□□□□□□ 0 □□□□□□□□□□□□□□□□□□□□ 0 □□□□□□□□

□□□□

1. □□□□□□, □□□□□□□□
2. □, □□□□□

□□

- https://leetcode.com/problems/move-zeroes/
- https://books.halfrost.com/leetcode/ChapterFour/0200~0299/0283.Move-Zeroes/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0283.Move-Zeroes/main.go

```go
package movezeroes

// □□O(n^2)
func MoveZeroes(nums []int) {
    for i := 0; i < len(nums); i++ {
        if nums[i] == 0 {
            for j := i + 1; j < len(nums); j++ {
                if nums[j] != 0 {
                    nums[i], nums[j] = nums[j], nums[i]
                    break
                }
            }
        }
    }
}

// □□O(n), □□O(1)
func MoveZeroes2point(nums []int) {
    j := 0
    for i := 0; i < len(nums); i++ {
```

```
        if nums[i] != 0 {
            if i != j {
                nums[i], nums[j] = nums[j], nums[i]
            }
            j++
        }
    }
}
```

Benchmark

```
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/0283.Move-Zeroes
cpu: Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz
BenchmarkMoveZeroes-4           145544966                9.164 ns/op           0 B/op          0 allocs/op
BenchmarkMoveZeroes2point-4     183269922                6.149 ns/op           0 B/op          0 allocs/op
PASS
ok      LeetcodeGolang/Leetcode/0283.Move-Zeroes        3.975s
```

7.50 0300.Longest Increasing Subsequence

Binary Search **Dynamic Programming** **Medium** **Patience Sorting**

7.50.1 300.Longest-Increasing-Subsequence

□□□□□□□(Longest Increasing Subsequence, LIS)

□□

Given an integer array nums, return the length of the longest strictly increasing subsequence.

A subsequence is a sequence that can be derived from an array by deleting some or no elements without changing the order of the remaining elements. For example, [3,6,2,7] is a subsequence of the array [0,3,1,6,2,2,7].

Example 1:

```
Input: nums = [10,9,2,5,3,7,101,18]
Output: 4
Explanation: The longest increasing subsequence is [2,3,7,101], therefore the length is 4.
```

Example 2:

```
Input: nums = [0,1,0,3,2,3]
Output: 4
```

Example 3:

```
Input: nums = [7,7,7,7,7,7,7]
Output: 1
 ```

Constraints:

* 1 <= nums.length <= 2500
* -104 <= nums[i] <= 104

Follow up: Can you come up with an algorithm that runs in O(n log(n)) time complexity?

□□□□
□□□□□□□□□ nums□□, □□□□□□□□□□□□□□
* □□□□□□□□□□
* □□□□□□□□

□□□□
□□□: DP
dp[i]□□nums[i]□□□□□□□□□□□□□□□□□
□□□□dp[5]□□, □□□□nums[5]□□□□□□□□□□□
□□□□□□□nums[5]□□□□□, □□□ nums[5]□□□□,
□□□□□□□□□□□□, □□□□□□□□□+1

□□□: DP + □□□□
patience sorting (□□□□)
□□□ [6, 3 ,5 ,10, 11, 2, 9, 14, 13, 7, 4, 8, 12]
□□□□□□□□□□□□□□□,
□□□□□□□□□□□□□□□□, □□□□□□,
□□□□□□□□□□□□□□, □□□□□□□□□□□□, □□□□□□□□□□□□□□□(2,4,7,8,12)
6 5 10 11 14
3 4 9 8 13
2 7 12

□□□□□□□□□□□□□□
[3,5,7,8,12]

□□
* https://leetcode.com/problems/longest-increasing-subsequence/

□□
https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0300.Longest-Increasing-Subsequence/main.go

```go
package longestincreasingsubsequence

import "fmt"
```

```go
func max(a, b int) int {
    if a > b {
        return a
    }
    return b
}

// DP □□ O(n^2)
func LengthOfLIS(nums []int) int {
    dp := make([]int, len(nums))
    for idx := 0; idx < len(dp); idx++ {
        dp[idx] = 1
    }

    res := 0
    for i := 0; i < len(nums); i++ {
        for j := 0; j < i; j++ {
            if nums[i] > nums[j] {
                // □□□□□□□□□□□□
                dp[i] = max(dp[i], dp[j]+1)
            }
        }
        res = max(res, dp[i])
    }
    // fmt.Println(dp)
    return res
}

func LengthOfLIS2(nums []int) int {
    dp := make([]int, len(nums)+1)
    dp[0] = 0
    res := 0

    for i := 1; i <= len(nums); i++ {
        for j := 1; j < i; j++ {
            if nums[i-1] > nums[j-1] {
                // □□□□□□□□□□□□□□□□
                dp[i] = max(dp[i], dp[j])
            }
        }
        dp[i] = dp[i] + 1
        res = max(res, dp[i])
    }
    // fmt.Println(dp)
    return res
}

// DP + □□□□:patience sorting. O(nlogn)
func LengthOfLISPatience(nums []int) int {
    top := make([]int, len(nums))

    // □□□□□□□0
    piles := 0

    for i := 0; i < len(nums); i++ {
        poker := nums[i]

        // □□□□□□□□□□□
        left, right := 0, piles
        // fmt.Printf("i:%d\tL:%d\tR:%d\n", i, left, right)
        for left < right {
            mid := left + (right-left)/2
            if top[mid] > poker {
                // □□□□□□□, □□□□
                right = mid
            } else if top[mid] < poker {
                // □□□□□□□
                left = mid + 1
            } else {
                right = mid
            }
        }

        // □□□□□, □□□□□□□
        if left == piles {
            piles++
        }
        // □□□□□□□□□□□
        top[left] = poker
    }
    fmt.Println(top)
    return piles
}
```

```
go test -benchmem -run=none LeetcodeGolang/Leetcode/300.Longest-Increasing-Subsequence -bench=.
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/300.Longest-Increasing-Subsequence
cpu: Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz
BenchmarkLengthOfLIS-8                  18300901                64.77 ns/op           64 B/op          1 allocs/op
BenchmarkLengthOfLIS2-8                 17410562                68.98 ns/op           80 B/op          1 allocs/op
BenchmarkLengthOfLISPatience-8          20768851                58.47 ns/op           64 B/op          1 allocs/op
```

```
PASS
ok      LeetcodeGolang/Leetcode/300.Longest-Increasing-Subsequence      3.812s
```

7.51 0310.Minimum Height Trees

Breadth First Search **Medium**

7.51.1 310. Minimum Height Trees

tags: Medium, BFS

A tree is an undirected graph in which any two vertices are connected by exactly one path. In other words, any connected graph without simple cycles is a tree.

Given a tree of n nodes labelled from 0 to n - 1, and an array of n - 1 edges where edges[i] = [ai, bi] indicates that there is an undirected edge between the two nodes ai and bi in the tree, you can choose any node of the tree as the root. When you select a node x as the root, the result tree has height h. Among all possible rooted trees, those with minimum height (i.e. min(h)) are called minimum height trees (MHTs).

Return a list of all MHTs' root labels. You can return the answer in any order.

The height of a rooted tree is the number of edges on the longest downward path between the root and a leaf.

Example 1:

```
Input: n = 4, edges = [[1,0],[1,2],[1,3]]
Output: [1]
Explanation: As shown, the height of the tree is 1 when the root is the node with label 1 which is the only MHT.
```

Example 2:

```
Input: n = 6, edges = [[3,0],[3,1],[3,2],[3,4],[5,4]]
Output: [3,4]
```

Constraints:

- 1 <= n <= 2 * 104
- edges.length == n - 1
- 0 <= ai, bi < n
- ai != bi
- All the pairs (ai, bi) are distinct.
- The given input is guaranteed to be a tree and there will be no repeated edges.

□□□□

□□□□n□□□□□□□, □□□□□□□□(root□left□□□□□)

□□□□

□BFS□□. Queue □□□□root, □□□□□□□□root□□□□□□(□□□□□□□□□□null)

□□

- https://leetcode.com/problems/minimum-height-trees/
- https://leetcode-cn.com/problems/minimum-height-trees/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0310.Minimum-Height-Trees/main.go

```go
package minimumheighttrees

import "fmt"

func FindMinHeightTrees(n int, edges [][]int) []int {
    if n == 1 {
        return []int{0}
    }

    graph := make(map[int][]int)
    degree := make(map[int]int, n) //□□: □□□□□□□□□□□□□□□□□□
    // □□□□□□
    for _, v := range edges {
```

```go
        if _, ok := graph[v[0]]; ok {
            graph[v[0]] = append(graph[v[0]], v[1])
        } else {
            // □□□
            graph[v[0]] = []int{v[1]}
        }

        if _, ok := graph[v[1]]; ok {
            graph[v[1]] = append(graph[v[1]], v[0])
        } else {
            // □□□
            graph[v[1]] = []int{v[0]}
        }
        degree[v[0]]++
        degree[v[1]]++
    }
    fmt.Printf("graph: %+v \n", graph)
    fmt.Printf("degree: %+v \n", degree)

    var queue []int
    for key, value := range degree { //□□□□□□1□□, □□Queue
        if value == 1 {
            queue = append(queue, key)
        }
    }
    fmt.Printf("queue: %+v \n", queue)

    leaves := []int{}
    for len(queue) != 0 {
        leaves = leaves[0:0]
        size := len(queue)

        for i := 0; i < size; i++ {
            leaves = append(leaves, queue[i])
            for _, neighbor := range graph[queue[i]] { // □□□□□□□-1, □□1□□□Queue
                degree[neighbor]--
                if degree[neighbor] == 1 {
                    queue = append(queue, neighbor)
                }
            }
        }
        queue = queue[size:]
    }
    return leaves
}

/*
n: 4,   edges: [][]int{{1, 0}, {1, 2}, {1, 3}},
graph: map[0:[1] 1:[0 2 3] 2:[1] 3:[1]]
           0 (degree: 1)
          /
         1 (3)
        / \
    2(1)   3(1)

degree: map[0:1 1:3 2:1 3:1]
queue: [2 3 0] (□degree=1)
*/
```

7.52 0322.Coin Change

Coin Change **Dynamic Programming** **Go** **LeetCode** **Medium**

7.52.1 322. Coin Change

□□

You are given an integer array coins representing coins of different denominations and an integer amount representing a total amount of money.

Return the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1.

You may assume that you have an infinite number of each kind of coin.

Example 1:

Input: coins = [1,2,5], amount = 11 Output: 3 Explanation: 11 = 5 + 5 + 1 Example 2:

Input: coins = [2], amount = 3 Output: -1 Example 3:

Input: coins = [1], amount = 0 Output: 0

Constraints:

1 <= coins.length <= 12 1 <= coins[i] <= 231 - 1 0 <= amount <= 104

□□□□

□□ k □□□□□□, □□□□□ c1,c2 ...,ck, □□□□□□□□□. □□□□□□□□. □□□□ □□□□□□□□□□□□□, □□□□□□ return -1.

□□□□

`dp(n)` □□□: □□□□□□□□□n, □□□□□□□□□n□□□□□□□

□□

- https://books.halfrost.com/leetcode/ChapterFour/0300~0399/0322.Coin-Change/
- https://leetcode.com/problems/coin-change/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0322.Coin-Change/Coin-Change.go

```go
package coinchange

import (
    "math"
)

func min(a, b int) int {
    if a > b {
        return b
    }
    return a
}

var memo = map[int]int{}

func dp(coins []int, n int) int {
    // □□□□□ □□□□
    if _, vok := memo[n]; vok {
        return memo[n]
```

```go
    }

    if n == 0 {
        return 0
    }

    if n < 0 {
        return -1
    }

    res := math.MaxInt
    for _, coin := range coins {
        subproblem := dp(coins, n-coin)
        if subproblem == -1 {
            continue
        }
        res = min(res, 1+subproblem)
    }
    if res != math.MaxInt {
        memo[n] = res
    } else {
        memo[n] = -1
    }
    return memo[n]
}

// □□□ □□ □□□□O(kn).
func CoinChangeMemoryTableRecursion(coins []int, amount int) int {
    return dp(coins, amount)
}

// dp array □□□□
func CoinChangeDP(coins []int, amount int) int {
    dp := make([]int, amount+1)
    dp[0] = 0

    // □□□, □□ amount □□□□□ □□□ amount □(□□□1□), □□ amount+1□□□□□□□
    // dp[]□□□: □□□□□□i□, □□□□dp[i]□□□□□
    for i := 1; i < len(dp); i++ {
        dp[i] = amount + 1
    }

    // □□ for □□□□□□□□□□□□□
    for i := 1; i <= amount; i++ {
        // □□ for □□□□□□□□□□□□
        for _, coin := range coins {
            if i-coin < 0 {
                continue
            }
            dp[i] = min(dp[i], 1+dp[i-coin])
        }
    }
    if dp[amount] > amount {
        return -1
    }

    return dp[amount]
}

func CoinChange(coins []int, n int) int {
    var dpClosure func(n int) int
    dpClosure = func(n int) int {
        if n == 0 {
            return 0
        }
        if n < 0 {
            return -1
        }
        res := math.MaxInt
        for _, coin := range coins {
            subproblem := dpClosure(n - coin)
            if subproblem == -1 {
                continue
            }
            res = min(res, 1+subproblem)
        }
        if res != math.MaxInt {
            return res
        } else {
            return -1
        }
    }
    return dpClosure(n)
}
```

```
go test -benchmem -run=none LeetcodeGolang/Leetcode/0322.Coin-Change -bench=.
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/0322.Coin-Change
cpu: Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz
BenchmarkCoinChange-4                          273376              4452 ns/op               0 B/op          0 allocs/op
BenchmarkCoinChangeDP-4                      11071686               128.1 ns/op            96 B/op          1 allocs/op
BenchmarkCoinChangeMemoryTableRecursion-4    57663068                23.69 ns/op            0 B/op          0 allocs/op
```

```
PASS
ok      LeetcodeGolang/Leetcode/0322.Coin-Change        4.194s
```

```
amount = 11. k=[1,2,5]

amount 0 1 2 3 4 5 ... 9 10 11
index  0 1 2 3 4 5 ... 9 10 11
dp     0 1 1 2 2 1 ... 3  2  3


dp[5] = 1+min(dp[5-1],dp[5-2],dp[5-5])
dp[11] = 1+min(dp[10],dp[9],dp[6])
```

7.53 0344.Reverse String

Easy **Two Pointers**

7.53.1 344. Reverse String

□□

Write a function that reverses a string. The input string is given as an array of characters char[].

Do not allocate extra space for another array, you must do this by modifying the input array in-place with O(1) extra memory.

You may assume all the characters consist of printable ascii characters.

Example 1:

```
Input: ["h","e","l","l","o"]
Output: ["o","l","l","e","h"]
```

Example 2:

```
Input: ["H","a","n","n","a","h"]
Output: ["h","a","n","n","a","H"]
```

□□□□

□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□ 2 □□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□

- https://books.halfrost.com/leetcode/ChapterFour/0300~0399/0344.Reverse-String/
- https://leetcode-cn.com/problems/reverse-string/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0344.Reverse-Stringm/Reverse-String.go

```go
package reversestring

func ReverseString(s []byte) {
    for i, j := 0, len(s)-1; i < j; i, j = i+1, j-1 {
        s[i], s[j] = s[j], s[i]
    }
}
```

7.54 0347.Top K Frequent Elements

Go LeetCode Medium Top K Frequent Elements heap

7.54.1 347. Top K Frequent Elements

□□

Given an integer array nums and an integer k, return the k most frequent elements. You may return the answer in any order.

Example 1:

Input: nums = [1,1,1,2,2,3], k = 2 Output: [1,2] Example 2:

Input: nums = [1], k = 1 Output: [1]

Constraints:

1 <= nums.length <= 105 -104 <= nums[i] <= 104 k is in the range [1, the number of unique elements in the array]. It is guaranteed that the answer is unique.

Follow up: Your algorithm's time complexity must be better than O(n log n), where n is the array's size.

Seen this question in a real interview before? 1/4

- Accepted 1.9M
- Submissions 3M
- Acceptance Rate 62.7%

□□□□

□□□□

□□□HEAP

□□□□□ □□□□□□

□□ □□□□□□□ k □□□□□□□□□□□□□□□□□□□□□□□ k □□□□

□□□□□□□□□□□□□□□□□□ □□□□□□O(N) □□□□□□□□□□□□N □□□□□□□□□□□□□□□□□□□□□□O(NlogN)□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□ k□□□□□□□□□□□□ □□□□□□□□□□□ k□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□ k □□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□ k □□□□

Big O

□□□HEAP

□□□□: `O(Nlogk)` □□□□: `O(N)`

□□□QUICK SORT

□□□□: `□□□□ :`

□□

- https://leetcode.com/problems/top-k-frequent-elements/description/
- https://leetcode.cn/problems/top-k-frequent-elements/description/
- https://books.halfrost.com/leetcode/ChapterFour/0300~0399/0347.Top-K-Frequent-Elements/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0347.Top-K-Frequent-Elements/main.go

```go
package topkfrequentelements

import (
    "container/heap"
    "sort"
)

// □□□: □□ PriorityQueue
// □□□□ O(Nlogk), □□□□ O(N)
// □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
// □□□□ PriortyQueue, □□□□□□□□□□□□
// □□ PriortyQueue pop□□□□ result
func TopKFrequent(nums []int, k int) []int {
    m := make(map[int]int)
    for i := 0; i < len(nums); i++ {
        m[nums[i]]++
    }

    q := PriorityQueue{}
    for key, count := range m {
        item := &Item{key: key, count: count}
        q.PushPQ(item)
    }

    var result []int
    for len(result) < k {
        item := q.PopPQ()
        result = append(result, item.key)
    }
    return result
}

// Item define
type Item struct {
    key   int
    count int
}

/*
// A PriorityQueue implements heap.Interface and holds Items.
package heap

import "sort"

type Interface interface {
    sort.Interface
    Push(x any) // add x as element Len()
    Pop() any   // remove and return element Len() - 1.
}
*/

type PriorityQueue []*Item

/*
// An implementation of Interface can be sorted by the routines in this package.
// The methods refer to elements of the underlying collection by integer index.

    type Interface interface {
        // Len is the number of elements in the collection.
```

```
        Len() int
        Less(i, j int) bool
        Swap(i, j int)
    }
*/
func (pq PriorityQueue) Len() int {
    return len(pq)
}

func (pq PriorityQueue) Less(i, j int) bool {
    // □□□□□
    return pq[i].count > pq[j].count
}

func (pq PriorityQueue) Swap(i, j int) {
    pq[i], pq[j] = pq[j], pq[i]
}

func (pg *PriorityQueue) Push(v interface{}) {
    item := v.(*Item)
    *pg = append(*pg, item)
}

func (pg *PriorityQueue) Pop() interface{} {
    n := len(*pg)
    item := (*pg)[n-1]
    *pg = (*pg)[:n-1]
    return item
}

func (pg *PriorityQueue) PushPQ(v *Item) {
    heap.Push(pg, v)
}

func (pg *PriorityQueue) PopPQ() *Item {
    return heap.Pop(pg).(*Item)
}

// □□□: □□ Quick Sort
// □□□□ O(), □□□□ O()
func TopKFrequentQuickSort(nums []int, k int) []int {
    m := make(map[int]int)
    for i := 0; i < len(nums); i++ {
        m[nums[i]]++
    }
    values := [][]int{}
    for key, count := range m {
        values = append(values, []int{key, count})
    }

    result := []int{}
    sort.Sort(sortValue(values))

    for i := 0; i < k; i++ {
        result = append(result, values[i][0])
    }
    return result
}

type sortValue [][]int

func (s sortValue) Len() int {
    return len(s)
}

func (s sortValue) Less(i, j int) bool {
    return s[i][1] > s[j][1]
}

func (s sortValue) Swap(i, j int) {
    s[i], s[j] = s[j], s[i]
}
```

Benchmark

```
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/0347.Top-K-Frequent-Elements
cpu: Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz
BenchmarkTopKFrequent-8                  1875834               648.6 ns/op           200 B/op         11 allocs/op
BenchmarkTopKFrequentQuickSort-8         1830498               704.2 ns/op           312 B/op         11 allocs/op
PASS
ok      LeetcodeGolang/Leetcode/0347.Top-K-Frequent-Elements    3.831s
```

7.55 0354.Russian Doll Envelopes

Binary Search **Dynamic Programming** **Hard**

7.55.1 354. Russian Doll Envelope

□□

You are given a 2D array of integers envelopes where envelopes[i] = [wi, hi] represents the width and the height of an envelope.

One envelope can fit into another if and only if both the width and height of one envelope are greater than the other envelope's width and height.

Return the maximum number of envelopes you can Russian doll (i.e., put one inside the other).

Note: You cannot rotate an envelope.

Example 1:

```
Input: envelopes = [[5,4],[6,4],[6,7],[2,3]]
Output: 3
Explanation: The maximum number of envelopes you can Russian doll is 3 ([2,3] => [5,4] => [6,7]).
```

Example 2:

```
Input: envelopes = [[1,1],[1,1],[1,1]]
Output: 1
```

Constraints:

- 1 <= envelopes.length <= 105
- envelopes[i].length == 2
- 1 <= wi, hi <= 105

□□□□

□□□□□, □□□□□□□□□□□□□□□□□(w,h)□□, □□□□□A□□□□□□□□□□□□□B□□□□, □B□□□□□A□. □□□□□□□□□□□□□□□□

□□□□

□□□□w□□□□□□, □□□□w□□□□□, □□□□□□□□□□□□. □□□□□□h□□□□array, □□array□ □□□□□□□□(Longest Increasing Subsequence, LIS)□□□□□□□

□□□□□□LIS□□, □□LIS□□ https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0300.Longest-Increasing-Subsequence/main.go □□ Leetcode No.1691

□□

- https://leetcode.com/problems/russian-doll-envelopes/
- https://leetcode-cn.com/problems/russian-doll-envelopes/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0354.Russian-Doll-Envelopes/main.go

```go
package russiandollenvelopes

import (
    longestincreasingsubsequence "LeetcodeGolang/Leetcode/0300.Longest-Increasing-Subsequence"
    "sort"
```

```
)

type sortEnvelopes [][]int

func (s sortEnvelopes) Len() int {
    return len(s)
}

func (s sortEnvelopes) Less(i, j int) bool {
    if s[i][0] == s[j][0] {
        // □□W□□□□□, □□□□□□□□□□□
        return s[i][1] > s[j][1]
    }
    // □□□W□□□□□□□
    return s[i][0] < s[j][0]
}

func (s sortEnvelopes) Swap(i, j int) {
    s[i], s[j] = s[j], s[i]
}

func MaxEnvelopes(envelopes [][]int) int {
    // □□□□W□□□□□□□, □□□□W□□□□□, □□□□□□□□□□□.
    sort.Sort(sortEnvelopes(envelopes))
    // fmt.Println(envelopes)

    // □□□h□□□□array, □□array□ □□□□□□□(Longest Increasing Subsequence, LIS)□□□□□□□
    height := make([]int, len(envelopes))
    for i := 0; i < len(envelopes); i++ {
        height[i] = envelopes[i][1]
    }

    //  DP + □□□□:patience sorting. O(nlogn)
    return longestincreasingsubsequence.LengthOfLISPatience(height)
}

func MaxEnvelopes2(envelopes [][]int) int {
    // □□□□W□□□□□□□, □□□□W□□□□□, □□□□□□□□□□□.
    sort.Sort(sortEnvelopes(envelopes))
    // fmt.Println(envelopes)

    // □□□□□□□(Longest Increasing Subsequence, LIS)□□□□□□□
    dp := []int{}
    for _, e := range envelopes {
        left, right := 0, len(dp)
        for left < right {
            mid := left + (right-left)/2
            if dp[mid] > e[1] {
                // □□□□□□□, □□□□
                right = mid
            } else if dp[mid] < e[1] {
                // □□□□□□□
                left = mid + 1
            } else {
                right = mid
            }
        }

        // □□□□□, □□□□□□□
        if left == len(dp) {
            dp = append(dp, e[1])
        }

        // □□□□□□□□□□□
        dp[left] = e[1]
    }

    return len(dp)
}

/*
func Sort(data Interface) {
    n := data.Len()
    quickSort(data, 0, n, maxDepth(n))
}
type Interface interface {
    // Len is the number of elements in the collection.
    Len() int

    // Less reports whether the element with index i
    // must sort before the element with index j.
    //
    // If both Less(i, j) and Less(j, i) are false,
    // then the elements at index i and j are considered equal.
    // Sort may place equal elements in any order in the final result,
    // while Stable preserves the original input order of equal elements.
    //
    // Less must describe a transitive ordering:
    //  - if both Less(i, j) and Less(j, k) are true, then Less(i, k) must be true as well.
    //  - if both Less(i, j) and Less(j, k) are false, then Less(i, k) must be false as well.
    //
    // Note that floating-point comparison (the < operator on float32 or float64 values)
    // is not a transitive ordering when not-a-number (NaN) values are involved.
    // See Float64Slice.Less for a correct implementation for floating-point values.
```

```
    Less(i, j int) bool

    // Swap swaps the elements with indexes i and j.
    Swap(i, j int)
}
*/
```

```
go test -benchmem -run=none LeetcodeGolang/Leetcode/0354.Russian-Doll-Envelopes -bench=.
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/0354.Russian-Doll-Envelopes
cpu: Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz
BenchmarkMaxEnvelopes-8          9067520               167.0 ns/op            88 B/op          3 allocs/op
BenchmarkMaxEnvelopes2-8         6726646               214.9 ns/op            80 B/op          4 allocs/op
PASS
ok      LeetcodeGolang/Leetcode/0354.Russian-Doll-Envelopes     6.237s
```

7.56 0380.Insert Delete GetRandom O1

Array Go Hash Insert Delete GetRandom O(1) LeetCode Medium

7.56.1 380. Insert Delete GetRandom O(1)

□□

Implement the RandomizedSet class:

RandomizedSet() Initializes the RandomizedSet object. bool insert(int val) Inserts an item val into the set if not present. Returns true if the item was not present, false otherwise. bool remove(int val) Removes an item val from the set if present. Returns true if the item was present, false otherwise. int getRandom() Returns a random element from the current set of elements (it's guaranteed that at least one element exists when this method is called). Each element must have the same probability of being returned. You must implement the functions of the class such that each function works in average O(1) time complexity.

Example 1:

Input ["RandomizedSet", "insert", "remove", "insert", "getRandom", "remove", "insert", "getRandom"] [[], [1], [2], [2], [], [1], [2], []] Output [null, true, false, true, 2, true, false, 2]

Explanation RandomizedSet randomizedSet = new RandomizedSet(); randomizedSet.insert(1); // Inserts 1 to the set. Returns true as 1 was inserted successfully. randomizedSet.remove(2); // Returns false as 2 does not exist in the set. randomizedSet.insert(2); // Inserts 2 to the set, returns true. Set now contains [1,2]. randomizedSet.getRandom(); // getRandom() should return either 1 or 2 randomly. randomizedSet.remove(1); // Removes 1 from the set, returns true. Set now contains [2]. randomizedSet.insert(2); // 2 was already in the set, so return false. randomizedSet.getRandom(); // Since 2 is the only number in the set, getRandom() will always return 2.

Constraints:

-231 <= val <= 231 - 1 At most 2 * 105 calls will be made to insert, remove, and getRandom. There will be at least one element in the data structure when getRandom is called.

□□□□

□□□□

□map□□□□□□□index

Big O

□□□□: `O(1)` □□□□: `O(n)`

□□

- https://leetcode.com/problems/insert-delete-getrandom-o1/description/
- https://leetcode.cn/problems/insert-delete-getrandom-o1/description/
- https://www.youtube.com/watch?v=46dZH7LDbf8

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0380.Insert-Delete-GetRandom-O1/main.go

```go
package insertdeletegetrandom

import "math/rand"
```

```go
// □□□□ O(1), □□□□ O(n)
type RandomizedSet struct {
    arr  []int
    set  map[int]int
    size int
}

func Constructor() RandomizedSet {
    arr := make([]int, 0)
    set := make(map[int]int)
    size := 0
    return RandomizedSet{arr, set, size}
}

func (this *RandomizedSet) Insert(val int) bool {
    if _, ok := this.set[val]; ok {
        return false
    }
    this.set[val] = this.size
    this.size++
    this.arr = append(this.arr, val)
    return true
}

func (this *RandomizedSet) Remove(val int) bool {
    index, ok := this.set[val]
    if !ok {
        return false
    }
    // swapping
    lastValue := this.arr[this.size-1]
    this.arr[index] = lastValue
    this.set[lastValue] = index

    // Remove last value
    this.arr = this.arr[:this.size-1]
    delete(this.set, val)
    this.size--
    return true
}

func (this *RandomizedSet) GetRandom() int {
    index := rand.Intn(this.size)
    return this.arr[index]
}

/**
 * Your RandomizedSet object will be instantiated and called as such:
 * obj := Constructor();
 * param_1 := obj.Insert(val);
 * param_2 := obj.Remove(val);
 * param_3 := obj.GetRandom();
 */
```

Benchmark

```go
```

Follow up

□□□□□□, □□□□1

0381.Insert-Delete-GetRandom-O1-Duplicates-allowed #Hard

7.57 0381.Insert Delete GetRandom O1 Duplicates allowed

Array Go Hard Hash Insert Delete GetRandom O(1) Duplicates allowed LeetCode

7.57.1 381.Insert Delete GetRandom O(1) Duplicates allowed

□□

□□□□

□□□□

RandomizedCollection is a data structure that contains a collection of numbers, possibly duplicates (i.e., a multiset). It should support inserting and removing specific elements and also reporting a random element.

Implement the RandomizedCollection class:

RandomizedCollection() Initializes the empty RandomizedCollection object. bool insert(int val) Inserts an item val into the multiset, even if the item is already present. Returns true if the item is not present, false otherwise. bool remove(int val) Removes an item val from the multiset if present. Returns true if the item is present, false otherwise. Note that if val has multiple occurrences in the multiset, we only remove one of them. int getRandom() Returns a random element from the current multiset of elements. The probability of each element being returned is linearly related to the number of the same values the multiset contains. You must implement the functions of the class such that each function works on average O(1) time complexity.

Note: The test cases are generated such that getRandom will only be called if there is at least one item in the RandomizedCollection.

Example 1:

Input ["RandomizedCollection", "insert", "insert", "insert", "getRandom", "remove", "getRandom"] [[], [1], [1], [2], [], [1], []] Output [null, true, false, true, 2, true, 1]

Explanation RandomizedCollection randomizedCollection = new RandomizedCollection(); randomizedCollection.insert(1); // return true since the collection does not contain 1. // Inserts 1 into the collection. randomizedCollection.insert(1); // return false since the collection contains 1. // Inserts another 1 into the collection. Collection now contains [1,1]. randomizedCollection.insert(2); // return true since the collection does not contain 2. // Inserts 2 into the collection. Collection now contains [1,1,2]. randomizedCollection.getRandom(); // getRandom should: // - return 1 with probability 2/3, or // - return 2 with probability 1/3. randomizedCollection.remove(1); // return true since the collection contains 1. // Removes 1 from the collection. Collection now contains [1,2]. randomizedCollection.getRandom(); // getRandom should return 1 or 2, both equally likely.

Constraints:

-231 <= val <= 231 - 1 At most 2 * 105 calls in total will be made to insert, remove, and getRandom. There will be at least one element in the data structure when getRandom is called.

Big O

□□□□: `O(1)` □□□□: `O(n)`

□□

- https://leetcode.com/problems/insert-delete-getrandom-o1-duplicates-allowed/description/
- https://leetcode.cn/problems/insert-delete-getrandom-o1-duplicates-allowed/description/
- https://www.youtube.com/watch?v=46dZH7LDbf8

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0381.Insert-Delete-GetRandom-O1-Duplicates-allowed/main.go

```go
package insertdeletegetrandomo1duplicatesallowed

import (
    "math/rand"
)

// □□□□ O(1), □□□□ O(n)
type RandomizedCollection struct {
    arr  []int
    set  map[int]map[int]struct{}
    size int
}

func Constructor() RandomizedCollection {
    arr := make([]int, 0)
    set := make(map[int]map[int]struct{})
    size := 0
    return RandomizedCollection{arr, set, size}
}

func (this *RandomizedCollection) Insert(val int) bool {
    ids, ok := this.set[val]
    if !ok {
        // □□□□□□, □□□□index□map
        ids = map[int]struct{}{}
        this.set[val] = ids
    }
    // index□map, key□□□□□□index, value□□□struct{}{}
    ids[this.size] = struct{}{}
    this.arr = append(this.arr, val)
    this.size++
    return !ok
}

func (this *RandomizedCollection) Remove(val int) bool {
    ids, ok := this.set[val]
    if !ok {
        return false
    }

    // □□□val□□index, □□□□□□□index□map□□, □□□□index□value□□□□struct{}{}
    var i int
    for id := range ids {
        i = id
        break
    }

    // □□□□□□□□□□□id
    this.arr[i] = this.arr[this.size-1]
    delete(ids, i)
    // □□□□□□□□□□□□□
    delete(this.set[this.arr[i]], this.size-1)

    if i < this.size-1 {
        // fmt.Printf("i =%d, this.size =%d\t", i, this.size)
        this.set[this.arr[i]][i] = struct{}{}
    }
    if len(ids) == 0 {
        // □□□□□□
        delete(this.set, val)
    }

    this.arr = this.arr[:this.size-1]
    this.size--
    return true
}

func (this *RandomizedCollection) GetRandom() int {
    index := rand.Intn(this.size)
    return this.arr[index]
}

/**
 * Your RandomizedCollection object will be instantiated and called as such:
 * obj := Constructor();
 * param_1 := obj.Insert(val);
 * param_2 := obj.Remove(val);
 * param_3 := obj.GetRandom();
 */
```

Benchmark

```
```

7.58 0409.Longest Palindrome

Easy **String**

7.58.1 409. Longest Palindrome

□□

Given a string s which consists of lowercase or uppercase letters, return the length of the longest palindrome that can be built with those letters.

Letters are case sensitive, for example, "Aa" is not considered a palindrome here.

Example 1:

```
Input: s = "abcccdd"
Output: 7
Explanation: One longest palindrome that can be built is "dccaccd", whose length is 7.
```

Example 2:

```
Input: s = "a"
Output: 1
Explanation: The longest palindrome that can be built is "a", whose length is 1.
```

Constraints:

- 1 <= s.length <= 2000
- s consists of lowercase and/or uppercase English letters only.

□□□□

□□□□

```
    return result
}
```

7.59 0412.Fizz Buzz

Apple | Easy | Facebook | Fizz Buzz | Go | LeetCode | Microsoft | math | string

7.59.1 412. Fizz Buzz

□□

`Facebook`, `Microsoft`, `Apple` Given an integer n, return a string array answer (1-indexed) where:

answer[i] == "FizzBuzz" if i is divisible by 3 and 5. answer[i] == "Fizz" if i is divisible by 3. answer[i] == "Buzz" if i is divisible by 5. answer[i] == i (as a string) if none of the above conditions are true.

Example 1:

Input: n = 3 Output: ["1","2","Fizz"] Example 2:

Input: n = 5 Output: ["1","2","Fizz","4","Buzz"] Example 3:

Input: n = 15 Output: ["1","2","Fizz","4","Buzz","Fizz","7","8","Fizz","Buzz","11","Fizz","13","14","FizzBuzz"]

Constraints:

1 <= n <= 104

□□□□

□□□□

Big O

□□□□: `O(n)` □□□□: `O(n)`

□□

- https://leetcode.com/problems/fizz-buzz/description/
- https://leetcode.cn/problems/fizz-buzz/description/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0412.Fizz-Buzz/main.go

```go
package fizzbuzz

import "strconv"

// □□□□ O(), □□□□ O()
func fizzBuzz(n int) []string {
    var result []string
    for i := 1; i <= n; i++ {
        if i%15 == 0 {
            result = append(result, "FizzBuzz")
        } else if i%3 == 0 {
            result = append(result, "Fizz")
        } else if i%5 == 0 {
            result = append(result, "Buzz")
        } else {
            result = append(result, strconv.Itoa(i))
        }
    }
    return result
}

func fizzBuzz2(n int) []string {
```

```go
    var result []string
    for i := 1; i <= n; i++ {
        var str string
        if i%3 == 0 {
            str += "Fizz"
        }
        if i%5 == 0 {
            str += "Buzz"
        }
        if len(str) <= 0 {
            str = strconv.Itoa(i)
        }
        result = append(result, str)
    }
    return result
}

// □□
func fizzBuzz3(n int) []string {
    var result []string
    fizz := 0
    buzz := 0
    for i := 1; i <= n; i++ {
        fizz++
        buzz++
        if fizz == 3 && buzz == 5 {
            result = append(result, "FizzBuzz")
            fizz = 0
            buzz = 0
        } else if fizz == 3 {
            result = append(result, "Fizz")
            fizz = 0
        } else if buzz == 5 {
            result = append(result, "Buzz")
            buzz = 0
        } else {
            result = append(result, strconv.Itoa(i))
        }
    }
    return result
}
```

Benchmark

```
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/0412.Fizz-Buzz
cpu: Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz
BenchmarkFizzBuzz-4      5918809               287.1 ns/op           112 B/op          3 allocs/op
BenchmarkFizzBuzz2-4     5024536               223.8 ns/op           112 B/op          3 allocs/op
BenchmarkFizzBuzz3-4     5406643               196.3 ns/op           112 B/op          3 allocs/op
PASS
ok      LeetcodeGolang/Leetcode/0412.Fizz-Buzz  5.507s
```

7.60 0438.Find All Anagrams in a String

Medium **Sliding Window**

7.60.1 438. Find All Anagrams in a String

□□

Given two strings s and p, return an array of all the start indices of p's anagrams in s. You may return the answer in any order.

An Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1:

```
Input: s = "cbaebabacd", p = "abc"
Output: [0,6]
Explanation:
The substring with start index = 0 is "cba", which is an anagram of "abc".
The substring with start index = 6 is "bac", which is an anagram of "abc".
```

Example 2:

```
Input: s = "abab", p = "ab"
Output: [0,1,2]
Explanation:
The substring with start index = 0 is "ab", which is an anagram of "ab".
The substring with start index = 1 is "ba", which is an anagram of "ab".
The substring with start index = 2 is "ab", which is an anagram of "ab".
```

Constraints:

1 <= s.length, p.length <= 3 * 104 s and p consist of lowercase English letters.

□□□□

□□□□□□□□, □□□□□ □□□□□□□ S □□□□□□□ P, □□ S □□□□ P □□□, □□□□□□□ index

□□□□

□ 0567.Permutation-in-String□□, □□□□□□□□□□□□

□□

- https://leetcode.com/problems/find-all-anagrams-in-a-string/
-

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0438.Find-All-Anagrams-in-a-String/main.go

```go
package findallanagramsinastring

func FindAnagrams(s string, p string) []int {
    need, window := make(map[rune]int), make(map[rune]int)
    for _, c := range p {
        need[c]++
    }

    left, right := 0, 0
    valid := 0
    res := []int{} //□□□□
    for right < len(s) {
        c := rune(s[right])
```

```go
        right++

        if need[c] > 0 {
            window[c]++
            if window[c] == need[c] {
                valid++
            }
        }
        // fmt.Printf("[%d,%d) \n", left, right)

        // □□□□□□□□□, □□□□□□□□□□□□□□□□□
        if (right - left) >= len(p) {
            if valid == len(need) {
                // □□□□□□□□□, □□index
                res = append(res, left)
            }
            d := rune(s[left])
            left++
            if need[d] > 0 {
                if window[d] == need[d] {
                    valid--
                }
                window[d]--
            }
        }
    }
    return res
}

// □ slice □□ map □□□
func FindAnagramsSlice(s string, p string) []int {
    need := [256]int{}
    for _, c := range p {
        need[c-'a']++
    }
    left, right := 0, 0
    count := len(p)
    res := []int{}

    for right < len(s) {
        c := s[right] - 'a'
        if need[c] > 0 {
            count--
        }
        need[c]--
        right++

        if count == 0 {
            res = append(res, left)
        }

        if (right - left) >= len(p) {
            d := s[left] - 'a'
            if need[d] >= 0 {
                count++
            }
            need[d]++
            left++
        }
    }
    return res
}
```

7.61 0509.Fibonacci Number

Dynamic Programming **Easy**

7.61.1 509. Fibonacci Number

□□

The Fibonacci numbers, commonly denoted F(n) form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

```
F(0) = 0, F(1) = 1
F(n) = F(n - 1) + F(n - 2), for n > 1.
```

Given n, calculate F(n).

Example 1:

```
Input: n = 2
Output: 1
Explanation: F(2) = F(1) + F(0) = 1 + 0 = 1.
```

Example 2:

```
Input: n = 3
Output: 2
Explanation: F(3) = F(2) + F(1) = 1 + 1 = 2.
```

Example 3:

```
Input: n = 4
Output: 3
Explanation: F(4) = F(3) + F(2) = 2 + 1 = 3.
```

Constraints: * 0 <= n <= 30

□□□□

□□□□□□, □□□ F(n) □□

F(0) = 0, F(1) = 1 F(N) = F(N - 1) + F(N - 2), □□ N > 1. □□ N□□□ F(N)□

□□□0 ≤ N ≤ 30 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377 ,610, 987.....

□□□□

□□□□□□□□□□□

```
            f(20)
           /     \
        f(19)   f(18)
       ...          ...
      /  \          /  \
    f(1) f(2)      f(1) f(2)
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□(dp)□□□□□□□□□□□□□□□□□□□□□□□□□□ 3 □□□□□□□□□□□□□□□□□□□□

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0509.Fibonacci-Number/Fibonacci-Number.go

```go
// --- Directions
// Print out the n-th entry in the fibonacci series.
// The fibonacci series is an ordering of numbers where
// each number is the sum of the preceeding two.
// For example, the sequence
//  [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
// forms the first ten entries of the fibonacci series.
// Example:
//   fib(4) === 3

package fibonaccinumber

import "math"

// Fib : iterative □□  O(n) . □□□□  O(n). □□□□□□□□□□
func FibIterative(n int) int {
    var result = []int{0, 1}

    for i := 2; i <= n; i++ {
        a := result[i-1]
        b := result[i-2]
        result = append(result, a+b)
    }
    return result[n]
}

// Fibrecur : recursive □□ O(2^n) . □□□□  O(n)
func Fibrecur(n int) int {
    if n < 2 {
        return n
    }

    return Fibrecur(n-1) + Fibrecur(n-2)
}

// the memo table
var memo = map[int]int{}

// FibDP : memoization □□□□□□
// □□□!
func FibDP(n int) int {
    // □□□cached □□□□
    if v, vok := memo[n]; vok == true {
        // cached
        return v
    }

    if _, vok := memo[0]; vok == false {
        // □cache
        memo[0] = 0
    }

    if _, vok := memo[1]; vok == false {
        // □cache
        memo[1] = 1
    }

    for i := 2; i <= n; i++ {
        if _, vok := memo[i]; vok == false {
            // □cache
            a := memo[i-1]
            b := memo[i-2]
            memo[i] = a + b
        }
        // fmt.Println(memo)
    }
    return memo[n]
    // return FibDP(n-1) + FibDP(n-2)
}

// □□□□
func FibDPStateCompression(n int) int {
    if n == 0 {
        return 0
    }
    if n == 2 || n == 1 {
        return 1
    }

    prev := 1
    curr := 1

    for i := 3; i <= n; i++ {
        sum := prev + curr
        prev = curr
        curr = sum
    }
    return curr
```

```go
}

var cache = map[int]int{}

func memoize(fn func(int) int) func(int) int {
    return func(args int) int {
        if _, vok := cache[args]; vok == true {
            return cache[args]
        }

        result := fn(args) // ▫ Fibrecur(args)
        cache[args] = result
        return result
    }
}

// Fibmem : memoization ▫▫▫▫function
func Fibmem(n int) int {
    result := memoize(Fibrecur)
    return result(n)
}

// ▫▫▫ ▫▫▫▫▫ ▫▫▫▫▫ O(log n)▫▫▫▫▫▫ O(log n)
// | 1 1 | ^ n   = | F(n+1) F(n)   |
// | 1 0 |         | F(n)   F(n-1) |
func Fib5(N int) int {
    if N <= 1 {
        return N
    }
    var A = [2][2]int{
        {1, 1},
        {1, 0},
    }
    A = matrixPower(A, N-1)
    return A[0][0]
}

func matrixPower(A [2][2]int, N int) [2][2]int {
    if N <= 1 {
        return A
    }
    A = matrixPower(A, N/2)
    A = multiply(A, A)

    var B = [2][2]int{
        {1, 1},
        {1, 0},
    }
    if N%2 != 0 {
        A = multiply(A, B)
    }

    return A
}

func multiply(A [2][2]int, B [2][2]int) [2][2]int {
    x := A[0][0]*B[0][0] + A[0][1]*B[1][0]
    y := A[0][0]*B[0][1] + A[0][1]*B[1][1]
    z := A[1][0]*B[0][0] + A[1][1]*B[1][0]
    w := A[1][0]*B[0][1] + A[1][1]*B[1][1]
    A[0][0] = x
    A[0][1] = y
    A[1][0] = z
    A[1][1] = w
    return A
}

// ▫▫▫ ▫▫▫ f(n)=(1/√5)*{[(1+√5)/2]^n -[(1-√5)/2]^n}▫▫ ▫▫▫▫▫▫ O(log n) ▫ O(n) ▫▫▫▫▫▫▫▫ O(1)
// ▫▫▫▫▫▫▫▫▫▫ pow() ▫▫▫▫▫▫▫▫▫▫▫▫ pow() ▫ O(log n) ▫
// ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫ n ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫ 0.618▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫ 0.618▫▫
// ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫ Round ▫▫▫▫
func Fib6(N int) int {
    var goldenRatio float64 = float64((1 + math.Sqrt(5)) / 2)
    return int(math.Round(math.Pow(goldenRatio, float64(N)) / math.Sqrt(5)))
}

// ▫▫▫ ▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫▫ LeetCode ▫▫▫▫▫▫▫▫▫ goroutine ▫▫▫
func Fib7(N int) int {
    return <-fibb(N)
}

func fibb(n int) <-chan int {
    result := make(chan int)
    go func() {
        defer close(result)

        if n <= 1 {
            result <- n
            return
        }
        result <- <-fibb(n-1) + <-fibb(n-2)
    }()
    return result
}
```

7.62 0516.Longest Palindromic Subsequence

Dynamic Programming **Medium**

7.62.1 516. Longest Palindromic Subsequence

□□

Given a string s, find the longest palindromic subsequence's length in s.

A subsequence is a sequence that can be derived from another sequence by deleting some or no elements without changing the order of the remaining elements.

Example 1:

```
Input: s = "bbbab"
Output: 4
Explanation: One possible longest palindromic subsequence is "bbbb".
```

Example 2:

```
Input: s = "cbbd"
Output: 2
Explanation: One possible longest palindromic subsequence is "bb".
```

Constraints:

- 1 <= s.length <= 1000
- s consists only of lowercase English letters.

□□□□

□□□□□□□ s□□□ s □□□□□□□□□□

□□□□

□□

- https://leetcode.com/problems/longest-palindromic-subsequence/

□□

7.63 0543.Diameter of Binary Tree

DFS **Diameter of Binary Tree** **Easy** **Go** **LeetCode**

7.63.1 543. Diameter of Binary Tree

□□

Given the root of a binary tree, return the length of the diameter of the tree.

The diameter of a binary tree is the length of the longest path between any two nodes in a tree. This path may or may not pass through the root.

The length of a path between two nodes is represented by the number of edges between them.

Example 1:

Input: root = [1,2,3,4,5] Output: 3 Explanation: 3 is the length of the path [4,2,1,3] or [5,2,1,3]. Example 2:

Input: root = [1,2] Output: 1

Constraints:

The number of nodes in the tree is in the range [1, 104]. -100 <= Node.val <= 100

□□□□

□□□□

□□□□□□□□□□□□□□□□□

Big O

□□□□ `O(n)`, □□□□: □□ `O(n)`, □□□ `O(log(n))`

□□

- https://leetcode.com/problems/diameter-of-binary-tree/
- https://leetcode.cn/problems/diameter-of-binary-tree/
-

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0543.Diameter-of-Binary-Tree/main.go

```go
package diameterofbinarytree

import "LeetcodeGolang/structures"

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

type TreeNode = structures.TreeNode

var maxDiameter int
```

```go
// □□□□ O(n), □□□□: □□ O(n), □□□ O(log(n))
func DiameterOfBinaryTree(root *TreeNode) int {
    if root == nil {
        return 0
    }
    maxDiameter = 0
    maxDepth(root)
    return maxDiameter
}

func maxDepth(node *TreeNode) int {
    if node == nil {
        return 0
    }
    left := maxDepth(node.Left)
    right := maxDepth(node.Right)

    maxDiameter = max(maxDiameter, left+right)
    return max(left, right) + 1
}

func max(a, b int) int {
    if a >= b {
        return a
    } else {
        return b
    }
}
```

Benchmark

```go

```

7.64 0567.Permutation in String

Medium **Sliding Window**

7.64.1 567. Permutation in String

tags: Medium, Sliding Window

□□

Given two strings s1 and s2, return true if s2 contains a permutation of s1, or false otherwise.

In other words, return true if one of s1's permutations is the substring of s2.

Example 1:

```
Input: s1 = "ab", s2 = "eidbaooo"
Output: true
Explanation: s2 contains one permutation of s1 ("ba").
```

Example 2:

Input: s1 = "ab", s2 = "eidboaoo" Output: false

Constraints:

- 1 <= s1.length, s2.length <= 104
- s1 and s2 consist of lowercase English letters.

□□□□

□□□□String S1 □ S2 , □□ S2 □□□□S1□□□, □□□□□□ S2 □□□□□□□□□□□□S1□□□□□□

□□□□

Sliding Window □□□ slice □□ map □□□

□□

- https://leetcode.com/problems/permutation-in-string/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0567.Permutation-in-String/main.go

```go
package permutationinstring

func CheckInclusion(s1 string, s2 string) bool {
    need, window := make(map[rune]int), make(map[rune]int)
    for _, c := range s1 {
        need[c]++
    }

    left, right := 0, 0
    valid := 0
    for right < len(s2) {
        c := rune(s2[right])
        right++

        // □□□□□□□□□□□□□
        if need[c] > 0 {
            window[c]++
            if window[c] == need[c] {
```

```go
                // □□□□□□□
                valid++
            }
        }

        // fmt.Printf("[%d,%d) \n", left, right)

        // □□□□□□□□□□
        // for (right - left) >= len(s1)
        if (right - left) >= len(s1) {

            if valid == len(need) {
                // □□□
                return true
            }
            d := rune(s2[left])
            left++
            if need[d] > 0 {
                if window[d] == need[d] {
                    valid--
                }
                window[d]--
            }
        }
    }
    return false
}

// □ slice □□ map □□□
func CheckInclusionSlice(s1 string, s2 string) bool {
    need := [256]int{}
    for _, c := range s1 {
        need[c-'a']++
    }

    left, right := 0, 0
    count := len(s1)
    for right < len(s2) {
        c := s2[right] - 'a'

        if need[c] > 0 {
            // □□□
            count--
        }
        need[c]--
        right++

        // fmt.Printf("[%d,%d)\n", left, right)
        if count == 0 {
            return true
        }

        // □□□□□□□□□□
        if (right - left) == len(s1) {
            d := s2[left] - 'a'
            if need[d] >= 0 {
                // □□□□□□□, □□□□□□□□□□□
                count++
            }
            need[d]++
            left++
        }
    }
    return false
}
```

tags: `Medium` `Leetcode` `Sliding Window`

7.65 0693.Binary Number with Alternating Bits

Bit Manipulation **Medium**

7.65.1 693. Binary Number with Alternating Bits

□□

Given a positive integer, check whether it has alternating bits: namely, if two adjacent bits will always have different values.

Example 1:

```
Input: 5
Output: True
Explanation:
The binary representation of 5 is: 101
```

Example 2:

```
Input: 7
Output: False
Explanation:
The binary representation of 7 is: 111.
```

Example 3:

```
Input: 11
Output: False
Explanation:
The binary representation of 11 is: 1011.
```

Example 4:

```
Input: 10
Output: True
Explanation:
The binary representation of 10 is: 1010.
```

□□□□

□□□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0693.Binary-Number-with-Alternating-Bits/Binary-Number-with-Alternating-Bits.go

```go
package binarynumberwithalternatingbits

// □□□ O(n)
func hasAlternatingBits(n int) bool {
    for n > 0 {
        preBit := n & 1
        n = n / 2
        curBit := n & 1
        if curBit == preBit {
            return false
        }
    }
    return true
}

// □□□
func hasAlternatingBits2(n int) bool {
    /*
        n=5
        n=              1 0 1
        n >> 1          0 1 0
        n^(n>>1)        1 1 1  (XOR □□□□1)
        n               1 1 1
        n+1           1 0 0 0
        n & (n+1)       0 0 0

        n=7
        n=              1 1 1
        n >> 1          0 1 1
        n^(n>>1)        1 0 0  (XOR □□□□1)
        n               1 0 0
        n+1             1 0 1
        n & (n+1)       1 0 0
    */
    n = n ^ (n >> 1)
    result := n & (n + 1)
    return result == 0
}
```

7.66 0695.Max Area of Island

Array & String | BFS | DFS | Go | LeetCode | Matrix | Max Area of Island | Medium

7.66.1 695. Max Area of Island

You are given an m x n binary matrix grid. An island is a group of 1's (representing land) connected 4-directionally (horizontal or vertical.) You may assume all four edges of the grid are surrounded by water.

The area of an island is the number of cells with a value 1 in the island.

Return the maximum area of an island in grid. If there is no island, return 0.

Example 1:

```
Input: grid = [[0,0,1,0,0,0,0,1,0,0,0,0],[0,0,0,0,0,0,1,1,1,0,0,0],[0,1,1,0,1,0,0,0,0,0,0,0],[0,1,0,0,1,1,0,0,1,0,1,0],[0,1,0,0,1,1,0,0,1,1,1,0],
[0,0,0,0,0,0,0,0,0,1,0,0],[0,0,0,0,0,0,1,1,1,0,0,0],[0,0,0,0,0,0,1,1,0,0,0,0]]
Output: 6
Explanation: The answer is not 11, because the island must be connected 4-directionally.
```

Example 2:

```
Input: grid = [[0,0,0,0,0,0,0,0]]
Output: 0
```

Constraints:

- m == grid.length
- n == grid[i].length
- 1 <= m, n <= 50
- grid[i][j] is either 0 or 1.

□□□□

□□□□□ '1'□□□□□ '0'□□□□□□□□□□□□□□□□□□□□□□□1□□□□□□□□□□□□□

□□□□

□□BFS□DFS□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□0□□□□□□□□□□□1□□□□□□□□□□□□□□□□□□□□ □□□□200□□□1254□□□□□□□□□□□□□□□□□□□

```
    grid[x][y] = 0
    total := 1

    // □□□□□□
    for i := 0; i < 4; i++ {
        nx := x + dir[i][0]
        ny := y + dir[i][1]
        total += areaOfIsland(grid, nx, ny)
    }
    return total
}

func max(x, y int) int {
    if x > y {
        return x
    }
    return y
}
```

Benchmark

7.67 0703.Kth Largest Element in a Stream

Easy | Go | Heap | Kth Largest Element in a Stream | LeetCode | Priority Queue

7.67.1 0703. Kth Largest Element in a Stream

□□

Design a class to find the kth largest element in a stream. Note that it is the kth largest element in the sorted order, not the kth distinct element.

Implement KthLargest class:

KthLargest(int k, int[] nums) Initializes the object with the integer k and the stream of integers nums. int add(int val) Appends the integer val to the stream and returns the element representing the kth largest element in the stream.

Example 1:

Input ["KthLargest", "add", "add", "add", "add", "add"] [[3, [4, 5, 8, 2]], [3], [5], [10], [9], [4]] Output [null, 4, 5, 5, 8, 8]

Explanation KthLargest kthLargest = new KthLargest(3, [4, 5, 8, 2]); kthLargest.add(3); // return 4 kthLargest.add(5); // return 5 kthLargest.add(10); // return 5 kthLargest.add(9); // return 8 kthLargest.add(4); // return 8

Constraints:

1 <= k <= 104 0 <= nums.length <= 104 -104 <= nums[i] <= 104 -104 <= val <= 104 At most 104 calls will be made to add. It is guaranteed that there will be at least k elements in the array when you search for the kth element. Accepted 479.1K Submissions 846.4K Acceptance Rate 56.6%

□□□□

□□□□□□□□□□□ k □□□□□□class□□ □□□□□□□□ k □□□□□□□ k □□□□□□□ □□□ KthLargest □□ * KthLargest□int k□ int[] nums□ □□□□ k □□□□ nums □□□□□□ * int add□int val□ □ val □□□□□ nums □□□□□□□□□□□ k □□□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□ 215.□□□□□ K □□□□□□

golang container/heap

Big O

□□□□ : □□□□□□□□□□ O(nlogk) □□□ n □□□□□ nums □□□; □□□□□□□□□□□ O(logk)

□□□□ : O(k)□ □□□□□□□□□□□ k □□□□

□□

- https://leetcode.com/problems/kth-largest-element-in-a-stream/
- https://leetcode.cn/problems/kth-largest-element-in-a-stream/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0703.Kth-Largest-Element-in-a-Stream/main.go

```
package kthlargestelementinastream

import (
```

```go
    "container/heap"
    "sort"
)

/**
 * Your KthLargest object will be instantiated and called as such:
 * obj := Constructor(k, nums);
 * param_1 := obj.Add(val);
 */

// □□□□ O(), □□□□ O()
type KthLargest struct {
    sort.IntSlice
    k int
}

func Constructor(k int, nums []int) KthLargest {
    kl := KthLargest{k: k}
    for _, val := range nums {
        kl.Add(val)
    }
    return kl
}

func (kl *KthLargest) Add(val int) int {
    heap.Push(kl, val)
    if kl.Len() > kl.k {
        heap.Pop(kl)
    }
    return kl.IntSlice[0]
}

func (kl *KthLargest) Push(v interface{}) {
    kl.IntSlice = append(kl.IntSlice, v.(int))
}

func (kl *KthLargest) Pop() interface{} {
    a := kl.IntSlice
    v := a[len(a)-1]
    kl.IntSlice = a[:len(a)-1]
    return v
}

func KthLargestStream(k int, nums []int, elem []int) []int {
    obj := Constructor(k, nums)
    result := []int{0}
    for _, val := range elem {
        obj.Add(val)
        result = append(result, obj.IntSlice[0])
    }

    return result
}
```

Benchmark

7.68 0704.Binary Search

Binary Search **Easy**

7.68.1 704. Binary Search

tags: Easy, Binary Search

□□

Given an array of integers nums which is sorted in ascending order, and an integer target, write a function to search target in nums. If target exists, then return its index. Otherwise, return -1.

You must write an algorithm with O(log n) runtime complexity.

Example 1:

```
Input: nums = [-1,0,3,5,9,12], target = 9
Output: 4
Explanation: 9 exists in nums and its index is 4
```

Example 2:

```
Input: nums = [-1,0,3,5,9,12], target = 2
Output: -1
Explanation: 2 does not exist in nums so return -1
```

Constraints:

- 1 <= nums.length <= 104
- -104 < nums[i], target < 104
- All the integers in nums are unique.
- nums is sorted in ascending order.

□□□□

□□□□□□□□□array, □□target□□index □□□□□□ -1

□□□□

□□□

□□

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

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0704.Binary-Search/main.go

```go
package binarysearch

import "sort"

func Search(nums []int, target int) int {
    lenght := len(nums)
    if lenght <= 0 {
        return -1
    }
```

```go
    left, right := 0, lenght-1

    for left <= right {
        mid := (right-left)/2 + left
        if nums[mid] == target {
            return mid
        } else if nums[mid] < target {
            // □□□
            left = mid + 1
        } else if nums[mid] > target {
            // □□□
            right = mid - 1
        }
    }
    // □□□□
    return -1
}

func Search2(nums []int, target int) int {
    lenght := len(nums)
    if lenght <= 0 {
        return -1
    }
    left, right := 0, lenght-1

    for left <= right {
        // □□2
        // mid := left + (right-left)>>1
        mid := int(uint(right+left) >> 1)
        if nums[mid] == target {
            return mid
        } else if nums[mid] < target {
            // □□□
            left = mid + 1
        } else if nums[mid] > target {
            // □□□
            right = mid - 1
        }
    }
    // □□□□
    return -1
}

// □□sort
func BinarySearch(nums []int, target int) int {
    length := len(nums)

    index := sort.Search(length, func(i int) bool {
        return nums[i] >= target
    })

    if index < length && nums[index] == target {
        return index
    } else {
        return -1
    }
}

// □□□□
func BinarySearchRecur(nums []int, target int) (index int) {
    return BinarySearchRecursively(nums, target, 0, len(nums)-1)
}

func BinarySearchRecursively(nums []int, target int, start int, end int) int {
    if end < start {
        return -1
    }

    middle := int(uint(end+start) >> 1)

    if nums[middle] == target {
        return middle
    } else if target > nums[middle] {
        return BinarySearchRecursively(nums, target, middle+1, end)
    } else {
        return BinarySearchRecursively(nums, target, start, middle-1)
    }
}

// □□□□ nums □□ target□□□□□□
func LeftBound(nums []int, target int) (index int) {
    lenght := len(nums)
    if lenght <= 0 {
        return -1
    }
    left, right := 0, lenght-1

    for left <= right {
        // □□2
        // mid := left + (right-left)>>1
        mid := int(uint(right+left) >> 1)
        if nums[mid] == target {
            // □□□□□, □□□□□□□
            right = mid - 1
```

```go
        } else if nums[mid] < target {
            // □□□
            left = mid + 1
        } else if nums[mid] > target {
            // □□□
            right = mid - 1
        }
    }
    // □□□□ □□: left□□□□
    if left >= lenght || nums[left] != target {
        return -1
    }
    return left
}

func RightBound(nums []int, target int) (index int) {
    lenght := len(nums)
    if lenght <= 0 {
        return -1
    }
    left, right := 0, lenght-1

    for left <= right {
        // □□2
        // mid := left + (right-left)>>1
        mid := int(uint(right+left) >> 1)
        if nums[mid] == target {
            // □□:□□□□□□, □□□□□□□=mid+1
            left = mid + 1
        } else if nums[mid] < target {
            // □□□
            left = mid + 1
        } else if nums[mid] > target {
            // □□□
            right = mid - 1
        }
    }
    // □□□□ □□:right□□□□
    if right < 0 || nums[right] != target {
        return -1
    }
    return right
}
```

tags: `Easy` `Leetcode` `Binary Search`

7.69 0721.Accounts Merge

Easy **Union Find**

7.69.1 721. Accounts Merge

□□

Given a list `accounts`, each element `accounts[i]` is a list of strings, where the first element `accounts[i][0]` is a name, and the rest of the elements are emailsrepresenting emails of the account.

Now, we would like to merge these accounts. Two accounts definitely belong to the same person if there is some email that is common to both accounts. Note that even if two accounts have the same name, they may belong to different people as people could have the same name. A person can have any number of accounts initially, but all of their accounts definitely have the same name.

After merging the accounts, return the accounts in the following format: the first element of each account is the name, and the rest of the elements are emails **in sorted order**. The accounts themselves can be returned in any order.

Example 1:

```
Input:
accounts = [["John", "johnsmith@mail.com", "john00@mail.com"], ["John", "johnnybravo@mail.com"], ["John", "johnsmith@mail.com", "john_newyork@mail.com"],
["Mary", "mary@mail.com"]]
Output: [["John", 'john00@mail.com', 'john_newyork@mail.com', 'johnsmith@mail.com'],  ["John", "johnnybravo@mail.com"], ["Mary", "mary@mail.com"]]
Explanation:
The first and third John's are the same person as they have the common email "johnsmith@mail.com".
The second John and Mary are different people as none of their email addresses are used by other accounts.
We could return these lists in any order, for example the answer [['Mary', 'mary@mail.com'], ['John', 'johnnybravo@mail.com'],
['John', 'john00@mail.com', 'john_newyork@mail.com', 'johnsmith@mail.com']] would still be accepted.
```

Note:

- The length of `accounts` will be in the range `[1, 1000]`.
- The length of `accounts[i]` will be in the range `[1, 10]`.
- The length of `accounts[i][j]` will be in the range `[1, 30]`.

□□□□

□□□□□□ `accounts` □□□□□ `accounts[i]` □□□□□□□□□□□□□□□□ `accounts[i][0]` □ □□ (name)□□□□□□ emails □□□□□□□□□□□ □□□□□□□□□□□□□□□□

□□□□□ `O(n log n)`

□□ n □□□□□□□□□□□ □□□□□□□□□□□

```go
func AccountsMerge2(accounts [][]string) (r [][]string) {
    uf := template.UnionFind{}
    uf.Init(len(accounts))
    // emailToID □□□□ email □□□□□□□ id(□□□□) □□
    // idToName □ id(□□□□) □ name □□
    // idToEmails □ id(□□□□) □□□□□□□□□ email □□□
    emailToID, idToName, idToEmails, res := make(map[string]int), make(map[int]string), make(map[int][]string), [][]string{}
    for id, acc := range accounts {
        idToName[id] = acc[0]
        for i := 1; i < len(acc); i++ {
            pid, ok := emailToID[acc[i]]
            if ok {
                uf.Union(id, pid)
            }
            emailToID[acc[i]] = id
        }
    }
    for email, id := range emailToID {
        pid := uf.Find(id)
        idToEmails[pid] = append(idToEmails[pid], email)
    }
    for id, emails := range idToEmails {
        name := idToName[id]
        sort.Strings(emails)
        res = append(res, append([]string{name}, emails...))
    }
    return res
}

// □□□ : Union Find, □□□□□
func AccountsMergeBurst(accounts [][]string) [][]string {
    if len(accounts) == 0 {
        return [][]string{}
    }
    uf, result, visited := template.UnionFind{}, [][]string{}, map[int]bool{}
    uf.Init(len(accounts))
    for i := 0; i < len(accounts); i++ {
        for j := i + 1; j < len(accounts); j++ {
            if accounts[i][0] == accounts[j][0] {
                // A, B□□. □□□□
                tmpAmail, tmpBmail, flag := accounts[i][1:], accounts[j][1:], false
                for j := 0; j < len(tmpAmail); j++ {
                    for k := 0; k < len(tmpBmail); k++ {
                        if tmpAmail[j] == tmpBmail[k] {
                            // A,B □□, □□□□□email. □□□□□□
                            flag = true
                            break
                        }
                    }
                    if flag {
                        break
                    }
                }
                if flag {
                    uf.Union(i, j)
                }
            }
        }
    }
    for i := 0; i < len(accounts); i++ {
        if visited[i] {
            continue
        }
        emails, account, tmpMap := accounts[i][1:], []string{accounts[i][0]}, map[string]string{}
        for j := i + 1; j < len(accounts); j++ {
            if uf.Find(j) == uf.Find(i) {
                visited[j] = true
                for _, v := range accounts[j][1:] {
                    tmpMap[v] = v
                }
            }
        }
        // A □□ Email
        for _, v := range emails {
            tmpMap[v] = v
        }
        emails = []string{}
        for key := range tmpMap {
            emails = append(emails, key)
        }
        sort.Strings(emails)
        account = append(account, emails...)
        result = append(result, account)
    }
    return result
}
```

7.70 0733.Flood Fill

BFS DFS Easy Flood Fill Go LeetCode

7.70.1 0733.Flood Fill

An image is represented by an `m x n` integer grid image where `image[i][j]` represents the pixel value of the image.

You are also given three integers `sr`, `sc`, and `color`. You should perform a flood fill on the image starting from the pixel `image[sr][sc]`.

To perform a flood fill, consider the starting pixel, plus any pixels connected 4-directionally to the starting pixel of the same color as the starting pixel, plus any pixels connected 4-directionally to those pixels (also with the same color), and so on. Replace the color of all of the aforementioned pixels with `color`.

Return the modified image after performing the flood fill.

Example 1:

```
Input: image = [[1,1,1],[1,1,0],[1,0,1]], sr = 1, sc = 1, color = 2
Output: [[2,2,2],[2,2,0],[2,0,1]]
Explanation: From the center of the image with position (sr, sc) = (1, 1) (i.e., the red pixel), all pixels connected by a path of the same color as the start
ing pixel (i.e., the blue pixels) are colored with the new color.
Note the bottom corner is not colored 2, because it is not 4-directionally connected to the starting pixel.
```

Example 2:

```
Input: image = [[0,0,0],[0,0,0]], sr = 0, sc = 0, color = 0
Output: [[0,0,0],[0,0,0]]
Explanation: The starting pixel is already colored 0, so no changes are made to the image.
```

Constraints:

- m == image.length
- n == image[i].length
- 1 <= m, n <= 50
- 0 <= image[i][j], color < 216
- 0 <= sr < m
- 0 <= sc < n

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ 0 □ 65535 □□□ □□□□□□ (sr, sc) □□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□

```go
    dfsfill(image, sr, sc, oldColor, color)
    return image
}

func dfsfill(image [][]int, row, col, oldColor, newColor int) {
    // Check if the current pixel is out of bounds or does not have the old color
    if row < 0 || row >= len(image) || col < 0 || col >= len(image[0]) || image[row][col] != oldColor {
        return
    }

    // Update the current pixel with the new color
    image[row][col] = newColor

    // Recursively perform flood fill on the adjacent pixels
    dfsfill(image, row-1, col, oldColor, newColor) // Up
    dfsfill(image, row+1, col, oldColor, newColor) // Down
    dfsfill(image, row, col-1, oldColor, newColor) // Left
    dfsfill(image, row, col+1, oldColor, newColor) // Right
}

type Point struct {
    row, col int
}

func FloodFillBFS(image [][]int, sr int, sc int, newColor int) [][]int {
    // Check if the starting point is out of bounds or already has the new color
    if sr < 0 || sr >= len(image) || sc < 0 || sc >= len(image[0]) || image[sr][sc] == newColor {
        return image
    }

    // Get the old color at the starting point
    oldColor := image[sr][sc]

    // Create a queue and enqueue the starting point
    queue := []Point{{sr, sc}}

    // Define the directions (up, down, left, right)
    directions := [][]int{{-1, 0}, {1, 0}, {0, -1}, {0, 1}}

    // Perform BFS
    for len(queue) > 0 {
        // Dequeue a point from the queue
        point := queue[0]
        queue = queue[1:]

        // Update the point with the new color
        image[point.row][point.col] = newColor

        // Explore the neighboring pixels
        for _, dir := range directions {
            newRow, newCol := point.row+dir[0], point.col+dir[1]

            // Check if the neighboring pixel is within bounds and has the old color
            if newRow >= 0 && newRow < len(image) && newCol >= 0 && newCol < len(image[0]) && image[newRow][newCol] == oldColor {
                // Enqueue the neighboring pixel
                queue = append(queue, Point{newRow, newCol})
            }
        }
    }

    return image
}
```

Benchmark

```
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/0733.Flood-Fill
cpu: Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz
BenchmarkFloodFillDFS-8         384756555                4.486 ns/op           0 B/op          0 allocs/op
BenchmarkFloodFillBFS-8         309088303                3.642 ns/op           0 B/op          0 allocs/op
PASS
ok      LeetcodeGolang/Leetcode/0733.Flood-Fill 3.572s
```

7.71 0746.Min Cost Climbing Stairs

Easy **Go** **LeetCode** **Min Cost Climbing Stairs**

7.71.1 746. Min Cost Climbing Stairs

□□

You are given an integer array cost where cost[i] is the cost of ith step on a staircase. Once you pay the cost, you can either climb one or two steps.

You can either start from the step with index 0, or the step with index 1.

Return the minimum cost to reach the top of the floor.

Example 1:

Input: cost = [10,15,20] Output: 15 Explanation: You will start at index 1. - Pay 15 and climb two steps to reach the top. The total cost is 15. Example 2:

Input: cost = [1,100,1,1,1,100,1,1,100,1] Output: 6 Explanation: You will start at index 0. - Pay 1 and climb two steps to reach index 2. - Pay 1 and climb two steps to reach index 4. - Pay 1 and climb two steps to reach index 6. - Pay 1 and climb one step to reach index 7. - Pay 1 and climb two steps to reach index 9. - Pay 1 and climb one step to reach the top. The total cost is 6.

Constraints:

2 <= cost.length <= 1000 0 <= cost[i] <= 999

□□□□

□□□□□□□□□□□□□□□ i □□□□□□□□□□□□□□□□□□ cost[i] □□□□ 0 □□□□ □□□□□□□□□□□□□□□□□□

Big O

□□□□: `O(n)` □□□□: `O(1)`

□□

- https://leetcode.com/problems/min-cost-climbing-stairs/
- https://leetcode.cn/problems/min-cost-climbing-stairs/
- https://books.halfrost.com/leetcode/ChapterFour/0700~0799/0746.Min-Cost-Climbing-Stairs/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0746.Min-Cost-Climbing-Stairs/main.go

```go
package mincostclimbingstairs

// □□□□ O(n), □□□□ O(1)
func MinCostClimbingStairs(cost []int) int {
    dp := make([]int, len(cost))
    dp[0], dp[1] = cost[0], cost[1]
    for i := 2; i < len(cost); i++ {
        dp[i] = cost[i] + min(dp[i-1], dp[i-2])
    }
    return min(dp[len(cost)-2], dp[len(cost)-1])
}

func min(a, b int) int {
    if a > b {
        return b
    }
    return a
}

// □□□□ O(n), □□□□ O(1)
// □□
// □□□□□□□cur□last□
// cur □□□□□□ i-2 □□□□ i □□□□□□□
// last □□□□□□ i-1 □□□□ i □□□□□□□
// □□□□□□□□□□
// □□2□□□□i := 2□□□□□□□□□□□i < len(cost)+1□□
// □□□□□□□
// □□□□□□□□□□
//
//  □□□□□□□□□□□□□□last + cost[i-1]
//  □□□□□□□□□□□□□□cur + cost[i-2]
//
// □□□□□□□□□□
//
//  □□ last + cost[i-1] □□□□□ last □□□□□ cur + cost[i-2]□□□ cur □□□□□ last□□□□□□□□□□□□
//  □□□□ last □□□□□ last + cost[i-1]□□□ cur □□□□□ last□□□□□□□□□□□□
func MinCostClimbingStairsOptimize(cost []int) int {
    var cur, last int
    for i := 2; i < len(cost)+1; i++ {
        // □□□□□□□□□□□□□□last + cost[i-1]
        // □□□□□□□□□□□□□□cur + cost[i-2]
        if last+cost[i-1] > cur+cost[i-2] {
            cur, last = last, cur+cost[i-2]
        } else {
            cur, last = last, last+cost[i-1]
        }
        // fmt.Printf("%-d | %-d | %-d\n", i, cur, last)
    }
    return last
}
```

Benchmark

```
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/0746.Min-Cost-Climbing-Stairs
cpu: Intel(R) Core(TM) i5-6400 CPU @ 2.70GHz
BenchmarkMinCostClimbingStairs-4                36693742                30.17 ns/op           24 B/op          1 allocs/op
BenchmarkMinCostClimbingStairsOptimize-4        405489464                3.091 ns/op           0 B/op          0 allocs/op
PASS
ok      LeetcodeGolang/Leetcode/0746.Min-Cost-Climbing-Stairs   2.713s
```

7.72 0752.Open the Lock

Breadth First Search **Medium**

7.72.1 752. Open the Lock

tags: Medium, BFS

You have a lock in front of you with 4 circular wheels. Each wheel has 10 slots: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'. The wheels can rotate freely and wrap around: for example we can turn '9' to be '0', or '0' to be '9'. Each move consists of turning one wheel one slot.

The lock initially starts at '0000', a string representing the state of the 4 wheels.

You are given a list of deadends dead ends, meaning if the lock displays any of these codes, the wheels of the lock will stop turning and you will be unable to open it.

Given a target representing the value of the wheels that will unlock the lock, return the minimum total number of turns required to open the lock, or -1 if it is impossible.

Example 1:

```
Input: deadends = ["0201","0101","0102","1212","2002"], target = "0202"
Output: 6
Explanation:
A sequence of valid moves would be "0000" -> "1000" -> "1100" -> "1200" -> "1201" -> "1202" -> "0202".
Note that a sequence like "0000" -> "0001" -> "0002" -> "0102" -> "0202" would be invalid,
because the wheels of the lock become stuck after the display becomes the dead end "0102".
```

Example 2:

```
Input: deadends = ["8888"], target = "0009"
Output: 1
Explanation: We can turn the last wheel in reverse to move from "0000" -> "0009".
```

Example 3:

```
Input: deadends = ["8887","8889","8878","8898","8788","8988","7888","9888"], target = "8888"
Output: -1
Explanation: We cannot reach the target without getting stuck.
```

Constraints: * 1 <= deadends.length <= 500 * deadends[i].length == 4 * target.length == 4 * target will not be in the list deadends. * target and deadends[i] consist of digits only.

□□□□

□□□□□□□□□□□□□, □□□□□0~9□10□□□, □□□□□□□□□□□ 0□□9 □ 9□□0. □□□□□□□□□□ □□□□□0.□□□□ deadends □□. □□□□□□□□□□□□□. □□□□□□ target □□ -1

□□□□

□ 0000□□□, □□□□□□□ "1000", "9000", "0100", "0900", "0010", "0090", "0001", "0009". **8** □□□, □□□□□□□□□□□, □□□□□□□□□□, □□□□□□□ □□□□□□□□□, □□□□□8□□□□□□, □□□□□□□□□

□□□: □□□□□□BFS, □□□□□□□□□□□, □□□□□□□□□

□□□: □□ BFS □□, □ while □□□□□□□□. □□□□□□□□□□□□□□□□□, □□□□□□□□□□□□□□□□□□□, □□□□□□□□□□□□□□□ curDepth □ nextDepth □□□ □□□□□ BFS □□ □□BFS, □□ □□BFS □□□□□□□□□□□

□□

- https://leetcode.com/problems/open-the-lock/
- https://books.halfrost.com/leetcode/ChapterFour/0700~0799/0752.Open-the-Lock/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0752.Open-the-Lock/main.go

```
package openthelock

// □□□: □□BFS
func OpenLock(deadends []string, target string) int {
    if target == "0000" {
        return 0
    }
    targetNum := strToInt(target)

    // □□□□□□□□□, □□□□□□
    visited := make([]bool, 10000)
    visited[0] = true
    for _, deadend := range deadends {
        num := strToInt(deadend)
        if num == 0 {
            return -1
        }
        visited[num] = true
    }

    depth, curDepth, nextDepth := 0, []int16{0}, make([]int16, 0)
    var nextNum int16

    for len(curDepth) > 0 {
        nextDepth = nextDepth[0:0]
        // □□Queue□□□□□□□□□□
        for _, curNum := range curDepth {
            // □□□□□□
            for incrementer := int16(1000); incrementer > 0; incrementer /= 10 {
                nextNum = PlusOne(curNum, incrementer)
                if nextNum == targetNum {
                    return depth + 1
                }
                if !visited[nextNum] {
                    visited[nextNum] = true
                    nextDepth = append(nextDepth, nextNum)
                }

                nextNum = MinusOne(curNum, incrementer)
                if nextNum == targetNum {
                    return depth + 1
                }
                if !visited[nextNum] {
                    visited[nextNum] = true
                    nextDepth = append(nextDepth, nextNum)
                }
            }
        }
        curDepth, nextDepth = nextDepth, curDepth
        // □□□□
        depth++
    }

    return -1
}

/* Note: Golang set
type void struct{}
var member void

set := make(map[string]void) // New empty set
set["Foo"] = member          // Add
for k := range set {         // Loop
    fmt.Println(k)
}
delete(set, "Foo")      // Delete
size := len(set)        // Size
_, exists := set["Foo"] // Membership
*/

// □□□: □□BFS. □□□□ Queue□□□ hashset, □□□□□□□□□□
// □□□□□□□□□□, □□□□□□□□□
func OpenLockBiDirection(deadends []string, target string) int {
    if target == "0000" {
        return 0
    }
    targetNum := strToInt(target)

    // □□□□□□□□□, □□□□□□
```

```go
    visited := make([]bool, 10000)
    for _, deadend := range deadends {
        num := strToInt(deadend)
        if num == 0 {
            return -1
        }
        visited[num] = true
    }

    depth := 0

    // □□□□□□□□
    curDepth := make(map[int16]struct{})
    nextDepth := make(map[int16]struct{})
    curDepth[0] = struct{}{}
    nextDepth[targetNum] = struct{}{}

    var nextNum int16

    for len(curDepth) != 0 && len(nextDepth) != 0 {
        // □□ curDepth □□□□□
        tmp := make(map[int16]struct{})

        // curDepth□□□□□□□
        for curNum := range curDepth {
            // □□□□□□□□
            if visited[curNum] {
                continue
            }
            _, exists := nextDepth[curNum]
            if exists {
                return depth
            }
            visited[curNum] = true

            // □□□□□□
            for incrementer := int16(1000); incrementer > 0; incrementer /= 10 {
                nextNum = PlusOne(curNum, incrementer)
                if !visited[nextNum] {
                    tmp[nextNum] = struct{}{}
                }

                nextNum = MinusOne(curNum, incrementer)
                if !visited[nextNum] {
                    tmp[nextNum] = struct{}{}
                }
            }
        }

        // □□□, □□□□ curDepth, nextDepth .
        // □□□ whihe□□□ nextDepth.
        // □□□□□□□□curDepth, □□□□□□□□curDepth, nextDepth
        curDepth, nextDepth = nextDepth, tmp
        // □□□□
        depth++
    }

    return -1
}

// □□□ : □□ BFS □□, □ while □□□□□□□□. □□□□□□□□□□□□□□,
// □□□□□□□□□□□□□□□□, □□□□□□□□□□□□□ curDepth □ nextDepth □□□
// □□□□□ BFS □□ □□BFS, □□□□BFS □□□□□□□□□□
func OpenLockBiDirectionOptimization(deadends []string, target string) int {
    if target == "0000" {
        return 0
    }
    targetNum := strToInt(target)

    // □□□□□□□□□□, □□□□□□
    visited := make([]bool, 10000)
    for _, deadend := range deadends {
        num := strToInt(deadend)
        if num == 0 {
            return -1
        }
        visited[num] = true
    }

    depth := 0

    // □□□□□□□□
    curDepth := make(map[int16]struct{})
    nextDepth := make(map[int16]struct{})
    curDepth[0] = struct{}{}
    nextDepth[targetNum] = struct{}{}

    var nextNum int16

    for len(curDepth) != 0 && len(nextDepth) != 0 {
        if len(curDepth) > len(nextDepth) {
            curDepth, nextDepth = nextDepth, curDepth
        }
        // □□ curDepth □□□□□
```

```go
        tmp := make(map[int16]struct{})

        // curDepth□□□□□□□
        for curNum := range curDepth {
            // □□□□□□□□
            if visited[curNum] {
                continue
            }
            _, exists := nextDepth[curNum]
            if exists {
                return depth
            }
            visited[curNum] = true

            // □□□□□□
            for incrementer := int16(1000); incrementer > 0; incrementer /= 10 {
                nextNum = PlusOne(curNum, incrementer)
                if !visited[nextNum] {
                    tmp[nextNum] = struct{}{}
                }

                nextNum = MinusOne(curNum, incrementer)
                if !visited[nextNum] {
                    tmp[nextNum] = struct{}{}
                }
            }
        }

        // □□□, □□□□ curDepth, nextDepth .
        // □□□ whihe□□□ nextDepth.
        // □□□□□□□□curDepth, □□□□□□□□curDepth, nextDepth
        curDepth, nextDepth = nextDepth, tmp
        // □□□□
        depth++
    }

    return -1
}

func PlusOne(curNum int16, incrementer int16) (nextNum int16) {
    digit := (curNum / incrementer) % 10
    if digit == 9 {
        nextNum = curNum - 9*incrementer
    } else {
        nextNum = curNum + incrementer
    }
    return nextNum
}

func MinusOne(curNum int16, incrementer int16) (nextNum int16) {
    digit := (curNum / incrementer) % 10
    if digit == 0 {
        nextNum = curNum + 9*incrementer
    } else {
        nextNum = curNum - incrementer
    }
    return nextNum
}

func strToInt(str string) int16 {
    return int16(str[0]-'0')*1000 + int16(str[1]-'0')*100 + int16(str[2]-'0')*10 + int16(str[3]-'0')
}
```

tags: `Medium` `Leetcode` `BFS`

7.73 0875.Koko Eating Bananas

Array | Binary Search | Go | Koko Eating Bananas | LeetCode | Medium

7.73.1 875. Koko Eating Bananas

□□

□□□□

□□□□□□□□ □□□ N □□□□□ i □□□ piles[i] □□□□ □□□□□□□□□□ H □□□□□□ □□□□□□□□□□□ K □□□□□/□□□□ □□□□□□□□□□□□□□□□□□□ K □□ □□□□□□□□ K □□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□ H □□□□□□□□□□□□□□□ K□K □□□□□

□□□□

□□□□□□□□□□□□□□□ □ [0 , max(piles)] □□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□ k □□□□□□□□□□□□□□□□□□□□

Big O

- □□□□□: `O□n log m□` □□□ n □□□□□□□□□m □□□□□□□□□□□□□□□
- □□□□□: `O(1)`

□□

- https://leetcode.com/problems/koko-eating-bananas/description/
- https://leetcode.cn/problems/koko-eating-bananas/description/
- https://books.halfrost.com/leetcode/ChapterFour/0800~0899/0875.Koko-Eating-Bananas/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0875.Koko-Eating-Bananas/main.go

```go
package kokoeatingbananas

// □□□□ O□n log m□, □□□□ O(1)
func minEatingSpeed(piles []int, h int) int {
    // □□□□□□□□
    left, right := 1, maxPile(piles)
    for left < right {
        // mid □□□□□□□□□(k)
        mid := int(uint(left+right) >> 1)
        if executeTime(piles, mid) <= h {
            right = mid
        } else {
            left = mid + 1
        }
    }
    return left
}

// □□□□□□k, □□□□□□□□
func executeTime(piles []int, k int) int {
    time := 0
    for _, pile := range piles {
        time += pile / k
        if pile%k > 0 {
            time++
        }
        // time += int(math.Ceil(float64(pile) / float64(k)))
    }
    return time
}

func maxPile(piles []int) int {
    result := 0
    for _, pile := range piles {
```

```
        if result < pile {
            result = pile
        }
    }
    return result
}
```

Benchmark

```

```

7.74 0876.Middle of the Linked List

Easy **Linked List** **Two Pointers**

7.74.1 876. Middle of the Linked List

□□

Given the head of a singly linked list, return the middle node of the linked list.

If there are two middle nodes, return the second middle node.

Example 1:

```
Input: head = [1,2,3,4,5]
Output: [3,4,5]
Explanation: The middle node of the list is node 3.
```

Example 2:

```
Input: head = [1,2,3,4,5,6]
Output: [4,5,6]
Explanation: Since the list has two middle nodes with values 3 and 4, we return the second one.
```

Constraints:

The number of nodes in the list is in the range [1, 100]. 1 <= Node.val <= 100

□□□□

□□□□□ linked list □□□□□

□□□□

□□ two pointer. □□□□□□□□. □□□□□□□. □□□□□□□□□□, □□□□□□linked list□□□□□. □linked list□□□□□□□□, slow□□□□□□□□□; □□□□□□□□, slow□□□□□□

□□

- https://leetcode.com/problems/middle-of-the-linked-list/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0876.Middle-of-the-Linked-List/main.go

```go
package middleofthelinkedlist

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */

type ListNode struct {
    Val  int
    Next *ListNode
}

func MiddleNode(head *ListNode) *ListNode {
    if head == nil || head.Next == nil {
        return head
    }

    fast, slow := head, head
```

```go
    for fast.Next != nil && fast.Next.Next != nil {
        fast = fast.Next.Next
        slow = slow.Next
    }

    // ▫▫▫▫
    curr := head
    length := 0
    for curr != nil {
        length++
        curr = curr.Next
    }
    if length%2 == 0 {
        // ▫▫
        return slow.Next
    } else {
        // ▫▫
        return slow
    }
}
```

tags: `Medium` `Leetcode` `two pointers`

7.75 0981.Time Based Key Value Store

Binary Search **Design** **Go** **Hash Table** **LeetCode** **Medium** **String** **Time Based Key-Value Store**

7.75.1 0981.Time Based Key-Value Store

□□

□□□□

□□□□

Big O

- □□□□ : ``
- □□□□ : ``

□□

- https://leetcode.com/problems/time-based-key-value-store/description/
- https://leetcode.cn/problems/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/0981.Time-Based-Key-Value-Store/main.go

```go
package timebasedkeyvaluestore

import "sort"

// □□□□ O(), □□□□ O()
type element struct {
    key       string
    value     string
    timestamp int
}

type TimeMap struct {
    elements map[string][]element
}

func Constructor() TimeMap {
    return TimeMap{elements: make(map[string][]element)}
}

func (this *TimeMap) Set(key string, value string, timestamp int) {
    this.elements[key] = append(this.elements[key], element{
        key:       key,
        value:     value,
        timestamp: timestamp,
    })
}

// O(log n)
func (this *TimeMap) Get(key string, timestamp int) string {
    if _, ok := this.elements[key]; !ok {
        return ""
    }
    // □□□□□□□□□□□□□
    index := sort.Search(len(this.elements[key]), func(i int) bool {
        return this.elements[key][i].timestamp > timestamp
    })
    if index == 0 {
        return ""
    }
```

```
    index--
    return this.elements[key][index].value
}
```

Benchmark

```

```

7.76 1046.Last Stone Weight

Easy | Go | Heap | Last Stone Weight | LeetCode | Priority Queue

7.76.1 1046. Last Stone Weight

□□

You are given an array of integers stones where stones[i] is the weight of the ith stone.

We are playing a game with the stones. On each turn, we choose the heaviest two stones and smash them together. Suppose the heaviest two stones have weights x and y with x <= y. The result of this smash is:

If x == y, both stones are destroyed, and If x != y, the stone of weight x is destroyed, and the stone of weight y has new weight y - x. At the end of the game, there is at most one stone left.

Return the weight of the last remaining stone. If there are no stones left, return 0.

Example 1:

Input: stones = [2,7,4,1,8,1] Output: 1 Explanation: We combine 7 and 8 to get 1 so the array converts to [2,4,1,1,1] then, we combine 2 and 4 to get 2 so the array converts to [2,1,1,1] then, we combine 2 and 1 to get 1 so the array converts to [1,1,1] then, we combine 1 and 1 to get 0 so the array converts to [1] then that's the value of the last stone. Example 2:

Input: stones = [1] Output: 1

Constraints:

1 <= stones.length <= 30 1 <= stones[i] <= 1000

□□□□

□□□□□ stones□□□ stone □□□□□□□□□□ □□□□□□□□□□

`n` □□□□□. □□□□□□□□□□□□□ `O(logn)` □□□, □□□□□□□□□□ `n-1` □□□

□□

- https://leetcode.com/problems/last-stone-weight/
- https://leetcode.cn/problems/last-stone-weight/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/1046.Last-Stone-Weight/main.go

```go
package laststoneweight

import (
    "container/heap"
    "fmt"
    "sort"
)

/*
// IntSlice attaches the methods of Interface to []int, sorting in increasing order.
type IntSlice []int

func (x IntSlice) Len() int           { return len(x) }
func (x IntSlice) Less(i, j int) bool { return x[i] < x[j] }
func (x IntSlice) Swap(i, j int)      { x[i], x[j] = x[j], x[i] }

// Sort is a convenience method: x.Sort() calls Sort(x).
func (x IntSlice) Sort() { Sort(x) }
*/

type hp struct {
    sort.IntSlice
}

func (h hp) Less(i, j int) bool {
    // □□□□□
    return h.IntSlice[i] > h.IntSlice[j]
}

func (h *hp) Push(v interface{}) {
    h.IntSlice = append(h.IntSlice, v.(int))
}

func (h *hp) Pop() interface{} {
    old := h.IntSlice
    v := old[len(old)-1]
    h.IntSlice = old[:len(old)-1]
    return v
}

func (h *hp) PushInt(v int) {
    heap.Push(h, v)
}

func (h *hp) PopInt() int {
    return heap.Pop(h).(int)
}

// □□□□ O(nlogn), □□□□ O(n)
// n □□□□□. □□□□□□□□□□□□□O(logn) □□□, □□□□□□□□□ n-1 □□□
func LastStoneWeight(stones []int) int {
    q := &hp{stones}
    heap.Init(q)
    fmt.Println(q)
    for q.Len() > 1 {
        fmt.Println(q)
        x, y := q.PopInt(), q.PopInt()
        fmt.Printf("%d,%d\n", x, y)
        if x > y {
            q.PushInt(x - y)
        }
    }
    if q.Len() > 0 {
        return q.IntSlice[0]
    }
    return 0
}
```

Benchmark

7.77 1143.Longest Common Subsequence

Dynamic Programming **Medium**

7.77.1 1143. Longest Common Subsequence

□□

Given two strings text1 and text2, return the length of their longest common subsequence. If there is no common subsequence, return 0.

A subsequence of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

For example, "ace" is a subsequence of "abcde". A common subsequence of two strings is a subsequence that is common to both strings.

Example 1:

```
Input: text1 = "abcde", text2 = "ace"
Output: 3
Explanation: The longest common subsequence is "ace" and its length is 3.
```

Example 2:

```
Input: text1 = "abc", text2 = "abc"
Output: 3
Explanation: The longest common subsequence is "abc" and its length is 3.
```

Example 3:

```
Input: text1 = "abc", text2 = "def"
Output: 0
Explanation: There is no such common subsequence, so the result is 0.
```

Constraints:

- 1 <= text1.length, text2.length <= 1000
- text1 and text2 consist of only lowercase English characters.

□□□□

□□□string □□, □□□string □□□□□□□□□□□, □□□□□□□0. □□ str1="abcde", str2="aceb", □□□3, □□□□□□□□□□"ace"

□□□□

1. □□□□, □□□ dp(i,j) □□ s1[0..i]□s2[0..j]□□□□□□□□□□□, □□s1[i]==s2[j], □□□□□□□□□□□□lcs□, □□□□□s1[0..i-1]□s2[0..j-1]□□lcs□□, □□1□□s1[0..i]□s2[0..j]□lcs□□□

```
if (str[i] == str2[j]) {
    dp(i,j) = dp(i-1,j-1)+1
}
```

□□s1[i]!=s2[j], □□□□□□□□□□□□□□lcs□,

```
if (str[i] != str2[j]){
    dp(i,j) = max( dp(i-1,j) , dp(i,j-1))
}
```

```
def longestCommonSubsequence(str1,str2) ->int:
    def dp(i,j):
        # □□base code
```

```
        if i == -1 or j == -1:
            return 0
        if str[i] == str2[j]:
            # □□□□lcs□□□□
            return dp(i-1, j-1)+1
        if str[i] != str2[j]:
            # □□□□□□□□□lcs□, □□□□,□□□□lcs□□
            return max( dp(i-1,j) , dp(i,j-1))
    return dp(len(str1)-1,len(str2)-1)
```

2. DP□□

```
int longestCommonSubsequence(string str1, string str2) {
    int m = str1.size(), n = str2.size();
    // □□□s1[0..i-1] □ s2[0..j-1], □□□lcs□□□dp[i][j]
    vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));
    // base case: dp[0][...] = dp[..][0] = 0, □□□□

    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            // □□□□□□
            if (str1[i - 1] == str2[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1] + 1;
            } else {
                // □□□□ dp[i - 1][j-1] , □□□□□□□□□□□, max□□□□□□□□
                dp[i][j] = max(dp[i][j - 1], dp[i - 1][j])
            }
        }
    }
    return dp[m][n];
}
```

□□

- https://leetcode.com/problems/longest-common-subsequence/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/1143.Longest-Common-Subsequence/main.go

```go
package longestcommonsubsequence

func LongestCommonSubsequence(text1 string, text2 string) int {
    var dp func(int, int) int
    dp = func(i, j int) int {
        if i == -1 || j == -1 {
            return 0
        }
        if text1[i] == text2[j] {
            return dp(i-1, j-1) + 1
        }
        if text1[i] != text2[j] {
            return max(dp(i-1, j), dp(i, j-1))
        }
        return 0
    }
    return dp(len(text1)-1, len(text2)-1)
}

func LongestCommonSubsequenceDP(text1 string, text2 string) int {
    m, n := len(text1), len(text2)
    if m == 0 || n == 0 {
        return 0
    }

    dp := make([][]int, m+1)
    for i := range dp {
        dp[i] = make([]int, n+1)
    }
    for i := 1; i <= m; i++ {
        for j := 1; j <= n; j++ {
            if text1[i-1] == text2[j-1] {
                dp[i][j] = dp[i-1][j-1] + 1
            } else {
                dp[i][j] = max(dp[i][j-1], dp[i-1][j])
            }
        }
    }
    return dp[m][n]
}

func max(a, b int) int {
    if a > b {
        return a
    }
```

```
    return b
}
```

```
goos: darwin
goarch: amd64
pkg: LeetcodeGolang/Leetcode/1143.Longest-Common-Subsequence
cpu: Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz
BenchmarkLongestCommonSubsequence-8                   100         737158262 ns/op               0 B/op          0 allocs/op
BenchmarkLongestCommonSubsequenceDP-8             2355297               491.3 ns/op           912 B/op          8 allocs/op
PASS
ok      LeetcodeGolang/Leetcode/1143.Longest-Common-Subsequence 75.400s
```

7.78 1195.Fizz Buzz Multithreaded

Concurrency **Fizz Buzz Multithreaded** **Go** **LeetCode** **Medium**

7.78.1 1195. Fizz Buzz Multithreaded

□□

You have the four functions:

printFizz that prints the word "fizz" to the console, printBuzz that prints the word "buzz" to the console, printFizzBuzz that prints the word "fizzbuzz" to the console, and printNumber that prints a given integer to the console. You are given an instance of the class FizzBuzz that has four functions: fizz, buzz, fizzbuzz and number. The same instance of FizzBuzz will be passed to four different threads:

Thread A: calls fizz() that should output the word "fizz". Thread B: calls buzz() that should output the word "buzz". Thread C: calls fizzbuzz() that should output the word "fizzbuzz". Thread D: calls number() that should only output the integers. Modify the given class to output the series [1, 2, "fizz", 4, "buzz", ...] where the ith token (1-indexed) of the series is:

"fizzbuzz" if i is divisible by 3 and 5, "fizz" if i is divisible by 3 and not 5, "buzz" if i is divisible by 5 and not 3, or i if i is not divisible by 3 or 5. Implement the FizzBuzz class:

FizzBuzz(int n) Initializes the object with the number n that represents the length of the sequence that should be printed. void fizz(printFizz) Calls printFizz to output "fizz". void buzz(printBuzz) Calls printBuzz to output "buzz". void fizzbuzz(printFizzBuzz) Calls printFizzBuzz to output "fizzbuzz". void number(printNumber) Calls printnumber to output the numbers.

Example 1:

Input: n = 15 Output: [1,2,"fizz",4,"buzz","fizz",7,8,"fizz","buzz",11,"fizz",13,14,"fizzbuzz"] Example 2:

Input: n = 5 Output: [1,2,"fizz",4,"buzz"]

Constraints:

1 <= n <= 50

□□□□

□□□□

Big O

□□□□: □□□□ :

□□

- https://leetcode.com/problems/fizz-buzz-multithreaded/description/
- https://leetcode.cn/problems/fizz-buzz-multithreaded/description/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/1195.Fizz-Buzz-Multithreaded/main.go

```go
package fizzbuzzmultithreaded

import (
    "fmt"
    "sync"
```

```go
)

var (
    wg = sync.WaitGroup{}
)

// □□□□ O(), □□□□ O()
func FizzBuzz(n int) {
    fb := NewFizzBuzz()

    wg.Add(4)

    go fb.fizz()
    go fb.buzz()
    go fb.fizzbuzz()
    go fb.number()

    for i := 1; i <= n; i++ {
        if i%3 == 0 && i%5 == 0 {
            fb.fizzbuzzCh <- struct{}{}
        } else if i%3 == 0 {
            fb.fizzCh <- struct{}{}
        } else if i%5 == 0 {
            fb.buzzCh <- struct{}{}
        } else {
            fb.numberCh <- i
        }
        <-fb.orderCh
    }
    fb.done <- struct{}{}
    fb.done <- struct{}{}
    fb.done <- struct{}{}
    fb.done <- struct{}{}
    wg.Wait()
}

type FizzBuzzStruct struct {
    numberCh   chan int
    fizzCh     chan struct{}
    buzzCh     chan struct{}
    fizzbuzzCh chan struct{}
    orderCh    chan struct{}
    done       chan struct{}
}

func NewFizzBuzz() *FizzBuzzStruct {
    return &FizzBuzzStruct{
        numberCh:   make(chan int),
        fizzCh:     make(chan struct{}),
        buzzCh:     make(chan struct{}),
        fizzbuzzCh: make(chan struct{}),
        orderCh:    make(chan struct{}),
        done:       make(chan struct{}, 4),
    }
}

func (fb *FizzBuzzStruct) fizz() {
    defer wg.Done()
    for {
        select {
        case <-fb.fizzCh:
            fmt.Print("fizz")
            fb.orderCh <- struct{}{}
        case <-fb.done:
            return
        }
    }
}

func (fb *FizzBuzzStruct) buzz() {
    defer wg.Done()
    for {
        select {
        case <-fb.buzzCh:
            fmt.Print("buzz")
            fb.orderCh <- struct{}{}
        case <-fb.done:
            return
        }
    }
}

func (fb *FizzBuzzStruct) fizzbuzz() {
    defer wg.Done()
    for {
        select {
        case <-fb.fizzbuzzCh:
            fmt.Print("fizzbuzz")
            fb.orderCh <- struct{}{}
        case <-fb.done:
            return
        }
    }
}
```

```go
func (fb *FizzBuzzStruct) number() {
    defer wg.Done()
    for {
        select {
        case v := <-fb.numberCh:
            fmt.Print(v)
            fb.orderCh <- struct{}{}
        case <-fb.done:
            return
        }
    }
}
```

Benchmark

```
```

7.79 3223.Minimum Length of String After Operations

Counting **Go** **Hash Table** **LeetCode** **Medium** **String**

7.79.1 3223.Minimum-Length-of-String-After-Operations

□□

You are given a string s.

You can perform the following process on s any number of times:

```
Choose an index i in the string such that there is at least one character to the left of index i that is equal to s[i], and at least one character to the
right that is also equal to s[i].
Delete the closest occurrence of s[i] located to the left of i.
Delete the closest occurrence of s[i] located to the right of i.
```

Return the minimum length of the final string s that you can achieve.

Example 1:

Input: s = "abaacbcbb"

Output: 5

Explanation: We do the following operations:

```
Choose index 2, then remove the characters at indices 0 and 3. The resulting string is s = &quot;bacbcbb&quot;.
Choose index 3, then remove the characters at indices 0 and 5. The resulting string is s = &quot;acbcb&quot;.
```

Example 2:

Input: s = "aa"

Output: 2

Explanation: We cannot perform any operations, so we return the length of the original string.

Constraints:

```
1 &lt;= s.length &lt;= 2 * 105
s consists only of lowercase English letters.
```

□□□□

□□□□

□□

- https://leetcode.com/problems/minimum-length-of-string-after-operations/description/

□□

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/3223.Minimum-Length-of-String-After-Operations/main.go

Benchmark

7.80 3224.Minimum Array Changes to Make Differences Equal

Array Go Hash Table LeetCode Medium Prefix Sum

7.80.1 3224.Minimum-Array-Changes-to-Make-Differences-Equal

□□

You are given an integer array nums of size n where n is even, and an integer k.

You can perform some changes on the array, where in one change you can replace any element in the array with any integer in the range from 0 to k.

You need to perform some changes (possibly none) such that the final array satisfies the following condition:

```
There exists an integer X such that abs(a[i] - a[n - i - 1]) = X for all (0 &lt;= i &lt; n).
```

Return the minimum number of changes required to satisfy the above condition.

Example 1:

Input: nums = [1,0,1,2,4,3], k = 4

Output: 2

Explanation: We can perform the following changes:

```
Replace nums[1] by 2. The resulting array is nums = [1,2,1,2,4,3].
Replace nums[3] by 3. The resulting array is nums = [1,2,1,3,4,3].
```

The integer X will be 2.

Example 2:

Input: nums = [0,1,2,3,3,6,5,4], k = 6

Output: 2

Explanation: We can perform the following operations:

```
Replace nums[3] by 0. The resulting array is nums = [0,1,2,0,3,6,5,4].
Replace nums[4] by 4. The resulting array is nums = [0,1,2,0,4,6,5,4].
```

The integer X will be 4.

Constraints:

```
2 &lt;= n == nums.length &lt;= 105
n is even.
0 &lt;= nums[i] &lt;= k &lt;= 105
```

□□□□

□□□□

□□

- https://leetcode.com/problems/minimum-array-changes-to-make-differences-equal/description/

https://github.com/kimi0230/LeetcodeGolang/blob/master/Leetcode/3224.Minimum-Array-Changes-to-Make-Differences-Equal/main.go

Benchmark