A Programmer's Advanced Cookbook
How to improve algorithm skills
LeetCode
Cookbook
Analysis in Go
halfrost 著
HALFROST©
フロストランド 訳

说明

此版本是 https://books.halfrost.com/leetcode 网页的离线版，由于网页版实时会更新，所以此 PDF 版难免会有一些排版或者错别字。如果读者遇到了，可以到网页版相应页面，点击页面 edit 按钮，提交 pr 进行更改。此 PDF 版本号是 V1.7.0。PDF 永久更新地址是 https://github.com/halfrost/LeetCode-Go/releases/，以版本号区分不同版本。笔者还是强烈推荐看在线版，有任何错误都会立即更新。如果觉得此书对刷题有一点点帮助，可以给此书点一个 star，鼓励一下笔者早点更新更多题解。

> 版本号说明，V1.7.0，1 是大版本号，7 代表当前题解中有几百题，目前是 700 题，所以第二个版本号是 7，0 代表当前题解中有几十题，目前是 700 题，所以第三个版本号是 0 。

目录

说明
目录
第一章 序章
- 关于 LeetCode
- 什么是 Cookbook
- 为什么会写这个开源书
- 关于书的封面
- 关于作者
- 关于书中的代码
- 目标读者
- 编程语言
- 使用说明
- 互动与勘误
- 最后

数据结构知识
算法知识
时间复杂度和空间复杂度
- 一. 时间复杂度数据规模
- 二. 空间复杂度
- 三. 递归的时间复杂度
  - 1. 只有一次递归调用
  - 2. 只有多次递归调用

第二章 算法专题
Array
String
Two Pointers
Linked List
Stack
Tree
Dynamic Programming
Backtracking
Depth First Search
Breadth First Search
Binary Search
Math
Hash Table
Sort

Bit Manipulation
Union Find
Sliding Window
Segment Tree
Binary Indexed Tree
第三章 一些模板
线段树 Segment Tree
一. 什么是线段树
二. 为什么需要这种数据结构
三. 构造线段树
四. 线段树的查询
1. 直接查询
2. 懒查询
五. 线段树的更新
1. 单点更新
2. 区间更新
六. 时间复杂度分析
七. 常见题型
1. Range Sum Queries
2. 单点更新
3. 区间更新
4. 区间合并
5. 扫描线
6. 计数问题
并查集 UnionFind
最近最少使用 LRUCache
解法一 Get O(1) / Put O(1)
解法二 Get O(1) / Put O(1)
模板
最不经常最少使用 LFUCache
解法一 Get O(1) / Put O(1)
解法二 Get O(capacity) / Put O(capacity)
模板
第四章 LeetCode 题解
1. Two Sum
题目
题目大意
解题思路
代码
2. Add Two Numbers
题目
题目大意
解题思路
代码
3. Longest Substring Without Repeating Characters
题目
题目大意
解题思路
代码

4. Median of Two Sorted Arrays
- 题目
- 题目大意
- 解题思路
- 代码

5. Longest Palindromic Substring
- 题目
- 题目大意
- 解题思路
- 代码

6. ZigZag Conversion
- 题目
- 题目大意
- 解题思路
- 代码

7. Reverse Integer
- 题目
- 题目大意
- 解题思路
- 代码

8. String to Integer (atoi)
- 题目
- 题目大意
- 解题思路
- 代码

9. Palindrome Number
- 题目
- 题目大意
- 解题思路
- 代码

11. Container With Most Water
- 题目
- 题目大意
- 解题思路
- 代码

12. Integer to Roman
- 题目
- 题目大意
- 解题思路
- 代码

13. Roman to Integer
- 题目
- 题目大意
- 解题思路
- 代码

15. 3Sum
- 题目
- 题目大意
- 解题思路

代码
16. 3Sum Closest
题目
题目大意
解题思路
代码
17. Letter Combinations of a Phone Number
题目
题目大意
解题思路
代码
18. 4Sum
题目
题目大意
解题思路
代码
19. Remove Nth Node From End of List
题目
题目大意
解题思路
代码
20. Valid Parentheses
题目
题目大意
解题思路
代码
21. Merge Two Sorted Lists
题目
题目大意
解题思路
代码
22. Generate Parentheses
题目
题目大意
解题思路
代码
23. Merge k Sorted Lists
题目
题目大意
解题思路
代码
24. Swap Nodes in Pairs
题目
题目大意
解题思路
代码
25. Reverse Nodes in k-Group
题目
题目大意

解题思路
代码
26. Remove Duplicates from Sorted Array
题目
题目大意
解题思路
代码
27. Remove Element
题目
题目大意
解题思路
代码
28. Implement strStr()
题目
题目大意
解题思路
代码
29. Divide Two Integers
题目
题目大意
解题思路
代码
30. Substring with Concatenation of All Words
题目
题目大意
解题思路
代码
31. Next Permutation
题目
题目大意
解题思路
代码
32. Longest Valid Parentheses
题目
题目大意
解题思路
代码
33. Search in Rotated Sorted Array
题目
题目大意
解题思路
代码
34. Find First and Last Position of Element in Sorted Array
题目
题目大意
解题思路
代码
35. Search Insert Position
题目

- 题目大意
- 解题思路
- 代码

36. Valid Sudoku

- 题目
- 题目大意
- 解题思路
- 代码

37. Sudoku Solver

- 题目
- 题目大意
- 解题思路
- 代码

39. Combination Sum

- 题目
- 题目大意
- 解题思路
- 代码

40. Combination Sum II

- 题目
- 题目大意
- 解题思路
- 代码

41. First Missing Positive

- 题目
- 题目大意
- 解题思路
- 代码

42. Trapping Rain Water

- 题目
- 题目大意
- 解题思路
- 代码

43. Multiply Strings

- 题目
- 题目大意
- 解题思路
- 代码

45. Jump Game II

- 题目
- 题目大意
- 解题思路
- 代码

46. Permutations

- 题目
- 题目大意
- 解题思路
- 代码

47. Permutations II

- 题目
- 题目大意
- 解题思路
- 代码

48. Rotate Image

- 题目
- 题目大意
- 解题思路
- 代码

49. Group Anagrams

- 题目
- 题目大意
- 解题思路
- 代码

50. Pow(x, n)

- 题目
- 题目大意
- 解题思路
- 代码

51. N-Queens

- 题目
- 题目大意
- 解题思路
- 代码

52. N-Queens II

- 题目
- 题目大意
- 解题思路
- 代码

53. Maximum Subarray

- 题目
- 题目大意
- 解题思路
- 代码

54. Spiral Matrix

- 题目
- 题目大意
- 解题思路
- 代码

55. Jump Game

- 题目
- 题目大意
- 解题思路
- 代码

56. Merge Intervals

- 题目
- 题目大意
- 解题思路
- 代码

57. Insert Interval
- 题目
- 题目大意
- 解题思路
- 代码

59. Spiral Matrix II
- 题目
- 题目大意
- 解题思路
- 代码

60. Permutation Sequence
- 题目
- 题目大意
- 解题思路
- 代码

61. Rotate List
- 题目
- 题目大意
- 解题思路
- 代码

62. Unique Paths
- 题目
- 题目大意
- 解题思路
- 代码

63. Unique Paths II
- 题目
- 题目大意
- 解题思路
- 代码

64. Minimum Path Sum
- 题目
- 题目大意
- 解题思路
- 代码

66. Plus One
- 题目
- 题目大意
- 解题思路
- 代码

67. Add Binary
- 题目
- 题目大意
- 解题思路
- 代码

69. Sqrt(x)
- 题目
- 题目大意
- 解题思路

代码
70. Climbing Stairs
题目
题目大意
解题思路
代码
71. Simplify Path
题目
题目大意
解题思路
代码
73. Set Matrix Zeroes
题目
题目大意
解题思路
代码
74. Search a 2D Matrix
题目
题目大意
解题思路
代码
75. Sort Colors
题目
题目大意
解题思路
代码
76. Minimum Window Substring
题目
题目大意
解题思路
代码
77. Combinations
题目
题目大意
解题思路
代码
78. Subsets
题目
题目大意
解题思路
代码
79. Word Search
题目
题目大意
解题思路
代码
80. Remove Duplicates from Sorted Array II
题目
题目大意

解题思路
代码
81. Search in Rotated Sorted Array II
题目
题目大意
解题思路
代码
82. Remove Duplicates from Sorted List II
题目
题目大意
解题思路
代码
83. Remove Duplicates from Sorted List
题目
题目大意
解题思路
代码
84. Largest Rectangle in Histogram
题目
题目大意
解题思路
代码
86. Partition List
题目
题目大意
解题思路
代码
88. Merge Sorted Array
题目
题目大意
解题思路
代码
89. Gray Code
题目
题目大意
解题思路
代码
90. Subsets II
题目
题目大意
解题思路
代码
91. Decode Ways
题目
题目大意
解题思路
代码
92. Reverse Linked List II
题目

- 题目大意
- 解题思路
- 代码

93. Restore IP Addresses

- 题目
- 题目大意
- 解题思路
- 代码

94. Binary Tree Inorder Traversal

- 题目
- 题目大意
- 解题思路
- 代码

95. Unique Binary Search Trees II

- 题目
- 题目大意
- 解题思路
- 代码

96. Unique Binary Search Trees

- 题目
- 题目大意
- 解题思路
- 代码

97. Interleaving String

- 题目
- 题目大意
- 解题思路
- 代码

98. Validate Binary Search Tree

- 题目
- 题目大意
- 解题思路
- 代码

99. Recover Binary Search Tree

- 题目
- 题目大意
- 解题思路
- 代码

100. Same Tree

- 题目
- 题目大意
- 解题思路
- 代码

101. Symmetric Tree

- 题目
- 题目大意
- 解题思路
- 代码

102. Binary Tree Level Order Traversal

- 题目
- 题目大意
- 解题思路
- 代码

103. Binary Tree Zigzag Level Order Traversal

- 题目
- 题目大意
- 解题思路
- 代码

104. Maximum Depth of Binary Tree

- 题目
- 题目大意
- 解题思路
- 代码

105. Construct Binary Tree from Preorder and Inorder Traversal

- 题目
- 题目大意
- 解题思路
- 代码

106. Construct Binary Tree from Inorder and Postorder Traversal

- 题目
- 题目大意
- 解题思路
- 代码

107. Binary Tree Level Order Traversal II

- 题目
- 题目大意
- 解题思路
- 代码

108. Convert Sorted Array to Binary Search Tree

- 题目
- 题目大意
- 解题思路
- 代码

109. Convert Sorted List to Binary Search Tree

- 题目
- 题目大意
- 解题思路
- 代码

110. Balanced Binary Tree

- 题目
- 题目大意
- 解题思路
- 代码

111. Minimum Depth of Binary Tree

- 题目
- 题目大意
- 解题思路
- 代码

112. Path Sum
- 题目
- 题目大意
- 解题思路
- 代码

113. Path Sum II
- 题目
- 题目大意
- 解题思路
- 代码

114. Flatten Binary Tree to Linked List
- 题目
- 题目大意
- 解题思路
- 代码

115. Distinct Subsequences
- 题目
- 题目大意
- 解题思路
- 代码

118. Pascal's Triangle
- 题目
- 题目大意
- 解题思路
- 代码

119. Pascal's Triangle II
- 题目
- 题目大意
- 解题思路
- 代码

120. Triangle
- 题目
- 题目大意
- 解题思路
- 代码

121. Best Time to Buy and Sell Stock
- 题目
- 题目大意
- 解题思路
- 代码

122. Best Time to Buy and Sell Stock II
- 题目
- 题目大意
- 解题思路
- 代码

124. Binary Tree Maximum Path Sum
- 题目
- 题目大意
- 解题思路

代码
125. Valid Palindrome
题目
题目大意
解题思路
代码
126. Word Ladder II
题目
题目大意
解题思路
代码
127. Word Ladder
题目
题目大意
解题思路
代码
128. Longest Consecutive Sequence
题目
题目大意
解题思路
代码
129. Sum Root to Leaf Numbers
题目
题目大意
解题思路
代码
130. Surrounded Regions
题目
题目大意
解题思路
代码
131. Palindrome Partitioning
题目
题目大意
解题思路
代码
136. Single Number
题目
题目大意
解题思路
代码
137. Single Number II
题目
题目大意
解题思路
代码
138. Copy List with Random Pointer
题目
题目大意

解题思路
代码
141. Linked List Cycle
题目
题目大意
解题思路
代码
142. Linked List Cycle II
题目
题目大意
解题思路
代码
143. Reorder List
题目
题目大意
解题思路
代码
144. Binary Tree Preorder Traversal
题目
题目大意
解题思路
代码
145. Binary Tree Postorder Traversal
题目
题目大意
解题思路
代码
146. LRU Cache
题目
题目大意
解题思路
代码
147. Insertion Sort List
题目
题目大意
解题思路
代码
148. Sort List
题目
题目大意
解题思路
代码
150. Evaluate Reverse Polish Notation
题目
题目大意
解题思路
代码
151. Reverse Words in a String
题目

- 题目大意
- 解题思路
- 代码

152. Maximum Product Subarray

- 题目
- 题目大意
- 解题思路
- 代码

153. Find Minimum in Rotated Sorted Array

- 题目
- 题目大意
- 解题思路
- 代码

154. Find Minimum in Rotated Sorted Array II

- 题目
- 题目大意
- 解题思路
- 代码

155. Min Stack

- 题目
- 题目大意
- 解题思路
- 代码

160. Intersection of Two Linked Lists

- 题目
- 题目大意
- 解题思路
- 代码

162. Find Peak Element

- 题目
- 题目大意
- 解题思路
- 代码

164. Maximum Gap

- 题目
- 题目大意
- 解题思路
- 代码

167. Two Sum II - Input array is sorted

- 题目
- 题目大意
- 解题思路
- 代码

168. Excel Sheet Column Title

- 题目
- 题目大意
- 解题思路
- 代码

169. Majority Element

- 题目
- 题目大意
- 解题思路
- 代码

171. Excel Sheet Column Number

- 题目
- 题目大意
- 解题思路
- 代码

172. Factorial Trailing Zeroes

- 题目
- 题目大意
- 解题思路
- 代码

173. Binary Search Tree Iterator

- 题目
- 题目大意
- 解题思路
- 代码

174. Dungeon Game

- 题目
- 题目大意
- 解题思路
- 代码

179. Largest Number

- 题目
- 题目大意
- 解题思路
- 代码

187. Repeated DNA Sequences

- 题目
- 题目大意
- 解题思路
- 代码

189. Rotate Array

- 题目
- 题目大意
- 解题思路
- 代码

190. Reverse Bits

- 题目
- 题目大意
- 解题思路
- 代码

191. Number of 1 Bits

- 题目
- 题目大意
- 解题思路
- 代码

198. House Robber
题目
题目大意
解题思路
代码
199. Binary Tree Right Side View
题目
题目大意
解题思路
代码
200. Number of Islands
题目
题目大意
解题思路
代码
201. Bitwise AND of Numbers Range
题目
题目大意
解题思路
代码
202. Happy Number
题目
题目大意
解题思路
代码
203. Remove Linked List Elements
题目
题目大意
解题思路
代码
204. Count Primes
题目
题目大意
解题思路
代码
205. Isomorphic Strings
题目
题目大意
解题思路
代码
206. Reverse Linked List
题目
题目大意
解题思路
代码
207. Course Schedule
题目
题目大意
解题思路

代码
208. Implement Trie (Prefix Tree)
题目
题目大意
解题思路
代码
209. Minimum Size Subarray Sum
题目
题目大意
解题思路
代码
210. Course Schedule II
题目
题目大意
解题思路
代码
211. Design Add and Search Words Data Structure
题目
题目大意
解题思路
代码
212. Word Search II
题目
题目大意
解题思路
代码
213. House Robber II
题目
题目大意
解题思路
代码
215. Kth Largest Element in an Array
题目
题目大意
解题思路
代码
216. Combination Sum III
题目
题目大意
解题思路
代码
217. Contains Duplicate
题目
题目大意
解题思路
代码
218. The Skyline Problem
题目
题目大意

解题思路
代码
219. Contains Duplicate II
题目
题目大意
解题思路
代码
220. Contains Duplicate III
题目
题目大意
解题思路
代码
222. Count Complete Tree Nodes
题目
题目大意
解题思路
代码
223. Rectangle Area
题目
题目大意
解题思路
代码
224. Basic Calculator
题目
题目大意
解题思路
代码
225. Implement Stack using Queues
题目
题目大意
解题思路
代码
226. Invert Binary Tree
题目
题目大意
解题思路
代码
227. Basic Calculator II
题目
题目大意
解题思路
代码
228. Summary Ranges
题目
题目大意
解题思路
代码
229. Majority Element II
题目

- 题目大意
- 解题思路
- 代码

230. Kth Smallest Element in a BST

- 题目
- 题目大意
- 解题思路
- 代码

231. Power of Two

- 题目
- 题目大意
- 解题思路
- 代码

232. Implement Queue using Stacks

- 题目
- 题目大意
- 解题思路
- 代码

234. Palindrome Linked List

- 题目
- 题目大意
- 解题思路
- 代码

235. Lowest Common Ancestor of a Binary Search Tree

- 题目
- 题目大意
- 解题思路
- 代码

236. Lowest Common Ancestor of a Binary Tree

- 题目
- 题目大意
- 解题思路
- 代码

237. Delete Node in a Linked List

- 题目
- 题目大意
- 解题思路
- 代码

239. Sliding Window Maximum

- 题目
- 题目大意
- 解题思路
- 代码

240. Search a 2D Matrix II

- 题目
- 题目大意
- 解题思路
- 代码

242. Valid Anagram

- 题目
- 题目大意
- 解题思路
- 代码

257. Binary Tree Paths

- 题目
- 题目大意
- 解题思路
- 代码

258. Add Digits

- 题目
- 题目大意
- 解题思路
- 代码

260. Single Number III

- 题目
- 题目大意
- 解题思路
- 代码

263. Ugly Number

- 题目
- 题目大意
- 解题思路
- 代码

264. Ugly Number II

- 题目
- 题目大意
- 解题思路
- 代码

268. Missing Number

- 题目
- 题目大意
- 解题思路
- 代码

274. H-Index

- 题目
- 题目大意
- 解题思路
- 代码

275. H-Index II

- 题目
- 题目大意
- 解题思路
- 代码

278. First Bad Version

- 题目
- 题目大意
- 解题思路
- 代码

279. Perfect Squares
- 题目
- 题目大意
- 解题思路
- 代码

283. Move Zeroes
- 题目
- 题目大意
- 解题思路
- 代码

284. Peeking Iterator
- 题目
- 题目大意
- 解题思路
- 代码

287. Find the Duplicate Number
- 题目
- 题目大意
- 解题思路
- 代码

290. Word Pattern
- 题目
- 题目大意
- 解题思路
- 代码

297. Serialize and Deserialize Binary Tree
- 题目
- 题目大意
- 解题思路
- 代码

300. Longest Increasing Subsequence
- 题目
- 题目大意
- 解题思路
- 代码

303. Range Sum Query - Immutable
- 题目
- 题目大意
- 解题思路
- 代码

304. Range Sum Query 2D - Immutable
- 题目
- 题目大意
- 解题思路
- 代码

306. Additive Number
- 题目
- 题目大意
- 解题思路

代码
307. Range Sum Query - Mutable
题目
题目大意
解题思路
代码
309. Best Time to Buy and Sell Stock with Cooldown
题目
题目大意
解题思路
代码
315. Count of Smaller Numbers After Self
题目
题目大意
解题思路
代码
318. Maximum Product of Word Lengths
题目
题目大意
解题思路
代码
322. Coin Change
题目
题目大意
解题思路
代码
324. Wiggle Sort II
题目
题目大意
解题思路
代码
326. Power of Three
题目
题目大意
解题思路
代码
327. Count of Range Sum
题目
题目大意
解题思路
代码
328. Odd Even Linked List
题目
题目大意
解题思路
代码
329. Longest Increasing Path in a Matrix
题目
题目大意

解题思路
代码
331. Verify Preorder Serialization of a Binary Tree
题目
题目大意
解题思路
代码
337. House Robber III
题目
题目大意
解题思路
代码
338. Counting Bits
题目
题目大意
解题思路
代码
341. Flatten Nested List Iterator
题目
题目大意
解题思路
代码
342. Power of Four
题目
题目大意
解题思路
代码
343. Integer Break
题目
题目大意
解题思路
代码
344. Reverse String
题目
题目大意
解题思路
代码
345. Reverse Vowels of a String
题目
题目大意
解题思路
代码
347. Top K Frequent Elements
题目
题目大意
解题思路
代码
349. Intersection of Two Arrays
题目

- 题目大意
- 解题思路
- 代码

350. Intersection of Two Arrays II

- 题目
- 题目大意
- 解题思路
- 代码

354. Russian Doll Envelopes

- 题目
- 题目大意
- 解题思路
- 代码

357. Count Numbers with Unique Digits

- 题目
- 题目大意
- 解题思路
- 代码

367. Valid Perfect Square

- 题目
- 题目大意
- 解题思路
- 代码

368. Largest Divisible Subset

- 题目
- 题目大意
- 解题思路
- 代码

371. Sum of Two Integers

- 题目
- 题目大意
- 解题思路
- 代码

372. Super Pow

- 题目
- 题目大意
- 解题思路
- 代码

373. Find K Pairs with Smallest Sums

- 题目
- 题目大意
- 解题思路
- 代码

374. Guess Number Higher or Lower

- 题目
- 题目大意
- 解题思路
- 代码

376. Wiggle Subsequence

- 题目
- 题目大意
- 解题思路
- 代码

377. Combination Sum IV

- 题目
- 题目大意
- 解题思路
- 代码

378. Kth Smallest Element in a Sorted Matrix

- 题目
- 题目大意
- 解题思路
- 代码

385. Mini Parser

- 题目
- 题目大意
- 解题思路
- 代码

386. Lexicographical Numbers

- 题目
- 题目大意
- 解题思路
- 代码

387. First Unique Character in a String

- 题目
- 题目大意
- 解题思路
- 代码

389. Find the Difference

- 题目
- 题目大意
- 解题思路
- 代码

392. Is Subsequence

- 题目
- 题目大意
- 解题思路
- 代码

393. UTF-8 Validation

- 题目
- 题目大意
- 解题思路
- 代码

394. Decode String

- 题目
- 题目大意
- 解题思路
- 代码

395. Longest Substring with At Least K Repeating Characters
- 题目
- 题目大意
- 解题思路
- 代码

397. Integer Replacement
- 题目
- 题目大意
- 解题思路
- 代码

399. Evaluate Division
- 题目
- 题目大意
- 解题思路
- 代码

401. Binary Watch
- 题目
- 题目大意
- 解题思路
- 代码

402. Remove K Digits
- 题目
- 题目大意
- 解题思路
- 代码

404. Sum of Left Leaves
- 题目
- 题目大意
- 解题思路
- 代码

405. Convert a Number to Hexadecimal
- 题目
- 题目大意
- 解题思路
- 代码

409. Longest Palindrome
- 题目
- 题目大意
- 解题思路
- 代码

410. Split Array Largest Sum
- 题目
- 题目大意
- 解题思路
- 代码

412. Fizz Buzz
- 题目
- 题目大意
- 解题思路

代码
413. Arithmetic Slices
题目
题目大意
解题思路
代码
414. Third Maximum Number
题目
题目大意
解题思路
代码
416. Partition Equal Subset Sum
题目
题目大意
解题思路
代码
417. Pacific Atlantic Water Flow
题目
题目大意
解题思路
代码
421. Maximum XOR of Two Numbers in an Array
题目
题目大意
解题思路
代码
423. Reconstruct Original Digits from English
题目
题目大意
解题思路
代码
424. Longest Repeating Character Replacement
题目
题目大意
解题思路
代码
433. Minimum Genetic Mutation
题目
题目大意
解题思路
代码
435. Non-overlapping Intervals
题目
题目大意
解题思路
代码
436. Find Right Interval
题目
题目大意

解题思路
代码
437. Path Sum III
题目
题目大意
解题思路
代码
438. Find All Anagrams in a String
题目
题目大意
解题思路
代码
441. Arranging Coins
题目
题目大意
解题思路
代码
445. Add Two Numbers II
题目
题目大意
解题思路
代码
447. Number of Boomerangs
题目
题目大意
解题思路
代码
448. Find All Numbers Disappeared in an Array
题目
题目大意
解题思路
代码
451. Sort Characters By Frequency
题目
题目大意
解题思路
代码
453. Minimum Moves to Equal Array Elements
题目
题目大意
解题思路
代码
454. 4Sum II
题目
题目大意
解题思路
代码
455. Assign Cookies
题目

- 题目大意
- 解题思路
- 代码

456. 132 Pattern

- 题目
- 题目大意
- 解题思路
- 代码

457. Circular Array Loop

- 题目
- 题目大意
- 解题思路
- 代码

460. LFU Cache

- 题目
- 题目大意
- 解题思路
- 代码

461. Hamming Distance

- 题目
- 题目大意
- 解题思路
- 代码

462. Minimum Moves to Equal Array Elements II

- 题目
- 题目大意
- 解题思路
- 代码

463. Island Perimeter

- 题目
- 题目大意
- 解题思路
- 代码

470. Implement Rand10() Using Rand7()

- 题目
- 题目大意
- 解题思路
- 代码

473. Matchsticks to Square

- 题目
- 题目大意
- 解题思路
- 代码

474. Ones and Zeroes

- 题目
- 题目大意
- 解题思路
- 代码

475. Heaters

- 题目
- 题目大意
- 解题思路
- 代码

476. Number Complement

- 题目
- 题目大意
- 解题思路
- 代码

477. Total Hamming Distance

- 题目
- 题目大意
- 解题思路
- 代码

478. Generate Random Point in a Circle

- 题目
- 题目大意
- 解题思路
- 代码

480. Sliding Window Median

- 题目
- 题目大意
- 解题思路
- 代码

483. Smallest Good Base

- 题目
- 题目大意
- 解题思路
- 代码

485. Max Consecutive Ones

- 题目
- 题目大意
- 解题思路
- 代码

491. Increasing Subsequences

- 题目
- 题目大意
- 解题思路
- 代码

493. Reverse Pairs

- 题目
- 题目大意
- 解题思路
- 代码

494. Target Sum

- 题目
- 题目大意
- 解题思路
- 代码

496. Next Greater Element I
题目
题目大意
解题思路
代码
497. Random Point in Non-overlapping Rectangles
题目
题目大意
解题思路
代码
498. Diagonal Traverse
题目
题目大意
解题思路
代码
500. Keyboard Row
题目
题目大意
解题思路
代码
503. Next Greater Element II
题目
题目大意
解题思路
代码
507. Perfect Number
题目
题目大意
解题思路
代码
508. Most Frequent Subtree Sum
题目
题目大意
解题思路
代码
509. Fibonacci Number
题目
题目大意
解题思路
代码
513. Find Bottom Left Tree Value
题目
题目大意
解题思路
代码
515. Find Largest Value in Each Tree Row
题目
题目大意
解题思路

代码
518. Coin Change 2
题目
题目大意
解题思路
代码
523. Continuous Subarray Sum
题目
题目大意
解题思路
代码
524. Longest Word in Dictionary through Deleting
题目
题目大意
解题思路
代码
525. Contiguous Array
题目
题目大意
解题思路
代码
526. Beautiful Arrangement
题目
题目大意
解题思路
代码
528. Random Pick with Weight
题目
题目大意
解题思路
代码
529. Minesweeper
题目
题目大意
解题思路
代码
530. Minimum Absolute Difference in BST
题目
题目大意
解题思路
代码
532. K-diff Pairs in an Array
题目
题目大意
解题思路
代码
535. Encode and Decode TinyURL
题目
题目大意

解题思路
代码
537. Complex Number Multiplication
题目
题目大意
解题思路
代码
538. Convert BST to Greater Tree
题目
题目大意
解题思路
代码
541. Reverse String II
题目
题目大意
解题思路
代码
542. 01 Matrix
题目
题目大意
解题思路
代码
547. Number of Provinces
题目
题目大意
解题思路
代码
554. Brick Wall
题目
题目大意
解题思路
代码
557. Reverse Words in a String III
题目
题目大意
解题思路
代码
561. Array Partition I
题目
题目大意
解题思路
代码
563. Binary Tree Tilt
题目
题目大意
解题思路
代码
566. Reshape the Matrix
题目

- 题目大意
- 解题思路
- 代码

567. Permutation in String

- 题目
- 题目大意
- 解题思路
- 代码

572. Subtree of Another Tree

- 题目
- 题目大意
- 解题思路
- 代码

575. Distribute Candies

- 题目
- 题目大意
- 解题思路
- 代码

581. Shortest Unsorted Continuous Subarray

- 题目
- 题目大意
- 解题思路
- 代码

583. Delete Operation for Two Strings

- 题目
- 题目大意
- 解题思路
- 代码

589. N-ary Tree Preorder Traversal

- 题目
- 题目大意
- 解题思路
- 代码

594. Longest Harmonious Subsequence

- 题目
- 题目大意
- 解题思路
- 代码

598. Range Addition II

- 题目
- 题目大意
- 解题思路
- 代码

599. Minimum Index Sum of Two Lists

- 题目
- 题目大意
- 解题思路
- 代码

605. Can Place Flowers

- 题目
- 题目大意
- 解题思路
- 代码

609. Find Duplicate File in System

- 题目
- 题目大意
- 解题思路
- 代码

622. Design Circular Queue

- 题目
- 题目大意
- 解题思路
- 代码

623. Add One Row to Tree

- 题目
- 题目大意
- 解题思路
- 代码

628. Maximum Product of Three Numbers

- 题目
- 题目大意
- 解题思路
- 代码

630. Course Schedule III

- 题目
- 题目大意
- 解题思路
- 代码

632. Smallest Range Covering Elements from K Lists

- 题目
- 题目大意
- 解题思路
- 代码

633. Sum of Square Numbers

- 题目
- 题目大意
- 解题思路
- 代码

636. Exclusive Time of Functions

- 题目
- 题目大意
- 解题思路
- 代码

637. Average of Levels in Binary Tree

- 题目
- 题目大意
- 解题思路
- 代码

638. Shopping Offers
题目
题目大意
解题思路
代码
643. Maximum Average Subarray I
题目
题目大意
解题思路
代码
645. Set Mismatch
题目
题目大意
解题思路
代码
647. Palindromic Substrings
题目
题目大意
解题思路
代码
648. Replace Words
题目
题目大意
解题思路
代码
653. Two Sum IV - Input is a BST
题目
题目大意
解题思路
代码
658. Find K Closest Elements
题目
题目大意
解题思路
代码
661. Image Smoother
题目
题目大意
解题思路
代码
662. Maximum Width of Binary Tree
题目
题目大意
解题思路
代码
665. Non-decreasing Array
题目
题目大意
解题思路

代码
667. Beautiful Arrangement II
题目
题目大意
解题思路
代码
668. Kth Smallest Number in Multiplication Table
题目
题目大意
解题思路
代码
669. Trim a Binary Search Tree
题目
题目大意
解题思路
代码
674. Longest Continuous Increasing Subsequence
题目
题目大意
解题思路
代码
676. Implement Magic Dictionary
题目
题目大意
解题思路
代码
682. Baseball Game
题目
题目大意
解题思路
代码
684. Redundant Connection
题目
题目大意
解题思路
代码
685. Redundant Connection II
题目
题目大意
解题思路
代码
690. Employee Importance
题目
题目大意
解题思路
代码
692. Top K Frequent Words
题目
题目大意

解题思路
代码
693. Binary Number with Alternating Bits
题目
题目大意
解题思路
代码
695. Max Area of Island
题目
题目大意
解题思路
代码
696. Count Binary Substrings
题目
题目大意
解题思路
代码
697. Degree of an Array
题目
题目大意
解题思路
代码
699. Falling Squares
题目
题目大意
解题思路
代码
703. Kth Largest Element in a Stream
题目
题目大意
解题思路
代码
704. Binary Search
题目
题目大意
解题思路
代码
705. Design HashSet
题目
题目大意
解题思路
代码
706. Design HashMap
题目
题目大意
解题思路
代码
707. Design Linked List
题目

- 题目大意
- 解题思路
- 代码

709. To Lower Case

- 题目
- 题目大意
- 解题思路
- 代码

710. Random Pick with Blacklist

- 题目
- 题目大意
- 解题思路
- 代码

713. Subarray Product Less Than K

- 题目
- 题目大意
- 解题思路
- 代码

714. Best Time to Buy and Sell Stock with Transaction Fee

- 题目
- 题目大意
- 解题思路
- 代码

715. Range Module

- 题目
- 题目大意
- 解题思路
- 代码

717. 1-bit and 2-bit Characters

- 题目:
- 题目大意
- 解题思路
- 代码

718. Maximum Length of Repeated Subarray

- 题目
- 题目大意
- 解题思路
- 代码

719. Find K-th Smallest Pair Distance

- 题目
- 题目大意
- 解题思路
- 代码

720. Longest Word in Dictionary

- 题目
- 题目大意
- 解题思路
- 代码

721. Accounts Merge

- 题目
- 题目大意
- 解题思路
- 代码

724. Find Pivot Index

- 题目
- 题目大意
- 解题思路
- 代码

725. Split Linked List in Parts

- 题目
- 题目大意
- 解题思路
- 代码

726. Number of Atoms

- 题目
- 题目大意
- 解题思路
- 代码

729. My Calendar I

- 题目
- 题目大意
- 解题思路
- 代码

732. My Calendar III

- 题目
- 题目大意
- 解题思路
- 代码

733. Flood Fill

- 题目
- 题目大意
- 解题思路
- 代码

735. Asteroid Collision

- 题目
- 题目大意
- 解题思路
- 代码

739. Daily Temperatures

- 题目
- 题目大意
- 解题思路
- 代码

744. Find Smallest Letter Greater Than Target

- 题目
- 题目大意
- 解题思路
- 代码

745. Prefix and Suffix Search
- 题目
- 题目大意
- 解题思路
- 代码

746. Min Cost Climbing Stairs
- 题目
- 题目大意
- 解题思路
- 代码

748. Shortest Completing Word
- 题目
- 题目大意
- 解题思路
- 代码

753. Cracking the Safe
- 题目
- 题目大意
- 解题思路
- 代码

756. Pyramid Transition Matrix
- 题目
- 题目大意
- 解题思路
- 代码

762. Prime Number of Set Bits in Binary Representation
- 题目
- 题目大意
- 解题思路
- 代码

763. Partition Labels
- 题目
- 题目大意
- 解题思路
- 代码

765. Couples Holding Hands
- 题目
- 题目大意
- 解题思路
- 代码

766. Toeplitz Matrix
- 题目
- 题目大意
- 解题思路
- 代码

767. Reorganize String
- 题目
- 题目大意
- 解题思路

代码
771. Jewels and Stones
题目
题目大意
解题思路
代码
775. Global and Local Inversions
题目
题目大意
解题思路
代码
778. Swim in Rising Water
题目
题目大意
解题思路
代码
781. Rabbits in Forest
题目
题目大意
解题思路
代码
783. Minimum Distance Between BST Nodes
题目
题目大意
解题思路
代码
784. Letter Case Permutation
题目
题目大意
解题思路
代码
785. Is Graph Bipartite?
题目
题目大意
解题思路
代码
786. K-th Smallest Prime Fraction
题目
题目大意
解题思路
代码
793. Preimage Size of Factorial Zeroes Function
题目
题目大意
解题思路
代码
795. Number of Subarrays with Bounded Maximum
题目
题目大意

解题思路
代码
802. Find Eventual Safe States
题目
题目大意
解题思路
代码
803. Bricks Falling When Hit
题目
题目大意
解题思路
代码
810. Chalkboard XOR Game
题目
题目大意
解题思路
代码
811. Subdomain Visit Count
题目
题目大意
解题思路
代码
812. Largest Triangle Area
题目
题目大意
解题思路
代码
815. Bus Routes
题目
题目大意
解题思路
代码
816. Ambiguous Coordinates
题目
题目大意
解题思路
代码
817. Linked List Components
题目
题目大意
解题思路
代码
819. Most Common Word
题目
题目大意
解题思路
代码
820. Short Encoding of Words
题目

- 题目大意
- 解题思路
- 代码

821. Shortest Distance to a Character

- 题目
- 题目大意
- 解题思路
- 代码

823. Binary Trees With Factors

- 题目
- 题目大意
- 解题思路
- 代码

826. Most Profit Assigning Work

- 题目
- 题目大意
- 解题思路
- 代码

828. Count Unique Characters of All Substrings of a Given String

- 题目
- 题目大意
- 解题思路
- 代码

830. Positions of Large Groups

- 题目
- 题目大意
- 解题思路
- 代码

832. Flipping an Image

- 题目
- 题目大意
- 解题思路
- 代码

834. Sum of Distances in Tree

- 题目
- 题目大意
- 解题思路
- 代码

836. Rectangle Overlap

- 题目
- 题目大意
- 解题思路
- 代码

838. Push Dominoes

- 题目
- 题目大意
- 解题思路
- 代码

839. Similar String Groups

- 题目
- 题目大意
- 解题思路
- 代码

841. Keys and Rooms

- 题目
- 题目大意
- 解题思路
- 代码

842. Split Array into Fibonacci Sequence

- 题目
- 题目大意
- 解题思路
- 代码

844. Backspace String Compare

- 题目
- 题目大意
- 解题思路
- 代码

845. Longest Mountain in Array

- 题目
- 题目大意
- 解题思路
- 代码

850. Rectangle Area II

- 题目
- 题目大意
- 解题思路
- 代码

851. Loud and Rich

- 题目
- 题目大意
- 解题思路
- 代码

852. Peak Index in a Mountain Array

- 题目
- 题目大意
- 解题思路
- 代码

853. Car Fleet

- 题目
- 题目大意
- 解题思路
- 代码

856. Score of Parentheses

- 题目
- 题目大意
- 解题思路
- 代码

862. Shortest Subarray with Sum at Least K
- 题目
- 题目大意
- 解题思路
- 代码

863. All Nodes Distance K in Binary Tree
- 题目
- 题目大意
- 解题思路
- 代码

864. Shortest Path to Get All Keys
- 题目
- 题目大意
- 解题思路
- 代码

867. Transpose Matrix
- 题目
- 题目大意
- 解题思路
- 代码

869. Reordered Power of 2
- 题目
- 题目大意
- 解题思路
- 代码

870. Advantage Shuffle
- 题目
- 题目大意
- 解题思路
- 代码

872. Leaf-Similar Trees
- 题目
- 题目大意
- 解题思路
- 代码

874. Walking Robot Simulation
- 题目
- 题目大意
- 解题思路
- 代码

875. Koko Eating Bananas
- 题目
- 题目大意
- 解题思路
- 代码

876. Middle of the Linked List
- 题目
- 题目大意
- 解题思路

代码
877. Stone Game
题目
题目大意
解题思路
代码
878. Nth Magical Number
题目
题目大意
解题思路
代码
880. Decoded String at Index
题目
题目大意
解题思路
代码
881. Boats to Save People
题目
题目大意
解题思路
代码
884. Uncommon Words from Two Sentences
题目
题目大意
解题思路
代码
885. Spiral Matrix III
题目
题目大意
解题思路
代码
887. Super Egg Drop
题目
题目大意
解题思路
代码
888. Fair Candy Swap
题目
题目大意
解题思路
代码
890. Find and Replace Pattern
题目
题目大意
解题思路
代码
891. Sum of Subsequence Widths
题目
题目大意

解题思路
代码
892. Surface Area of 3D Shapes
题目
题目大意
解题思路
代码
895. Maximum Frequency Stack
题目
题目大意
解题思路
代码
896. Monotonic Array
题目
题目大意
解题思路
代码
897. Increasing Order Search Tree
题目
题目大意
解题思路
代码
898. Bitwise ORs of Subarrays
题目
题目大意
解题思路
代码
901. Online Stock Span
题目
题目大意
解题思路
总结
代码
904. Fruit Into Baskets
题目
题目大意
解题思路
代码
907. Sum of Subarray Minimums
题目
题目大意
解题思路
代码
910. Smallest Range II
题目
题目大意
解题思路
代码
911. Online Election

- 题目
- 题目大意
- 解题思路
- 代码

914. X of a Kind in a Deck of Cards

- 题目
- 题目大意
- 解题思路
- 代码

916. Word Subsets

- 题目
- 题目大意
- 解题思路
- 代码

918. Maximum Sum Circular Subarray

- 题目
- 题目大意
- 解题思路
- 代码

920. Number of Music Playlists

- 题目
- 题目大意
- 解题思路
- 代码

921. Minimum Add to Make Parentheses Valid

- 题目
- 题目大意
- 解题思路
- 代码

922. Sort Array By Parity II

- 题目
- 题目大意
- 解题思路
- 代码

923. 3Sum With Multiplicity

- 题目
- 题目大意
- 解题思路
- 代码

924. Minimize Malware Spread

- 题目
- 题目大意
- 解题思路
- 代码

925. Long Pressed Name

- 题目
- 题目大意
- 解题思路
- 代码

927. Three Equal Parts
题目
题目大意
解题思路
代码
928. Minimize Malware Spread II
题目
题目大意
解题思路
代码
930. Binary Subarrays With Sum
题目
题目大意
解题思路
代码
933. Number of Recent Calls
题目
题目大意
解题思路
代码
938. Range Sum of BST
题目
题目大意
解题思路
代码
942. DI String Match
题目
题目大意
解题思路
代码
946. Validate Stack Sequences
题目
题目大意
解题思路
代码
947. Most Stones Removed with Same Row or Column
题目
题目大意
解题思路
代码
949. Largest Time for Given Digits
题目
题目大意
解题思路
代码
952. Largest Component Size by Common Factor
题目
题目大意
解题思路

代码
953. Verifying an Alien Dictionary
题目
题目大意
解题思路
代码
959. Regions Cut By Slashes
题目
题目大意
解题思路
代码
961. N-Repeated Element in Size 2N Array
题目
题目大意
解题思路
代码
966. Vowel Spellchecker
题目
题目大意
解题思路
代码
968. Binary Tree Cameras
题目
题目大意
解题思路
代码
969. Pancake Sorting
题目
题目大意
解题思路
代码
970. Powerful Integers
题目
题目大意
解题思路
代码
971. Flip Binary Tree To Match Preorder Traversal
题目
题目大意
解题思路
代码
973. K Closest Points to Origin
题目
题目大意
解题思路
代码
976. Largest Perimeter Triangle
题目
题目大意

解题思路
代码
977. Squares of a Sorted Array
题目
题目大意
解题思路
代码
978. Longest Turbulent Subarray
题目
题目大意
解题思路
代码
979. Distribute Coins in Binary Tree
题目
题目大意
解题思路
代码
980. Unique Paths III
题目
题目大意
解题思路
代码
981. Time Based Key-Value Store
题目
题目大意
解题思路
代码
984. String Without AAA or BBB
题目
题目大意
解题思路
代码
985. Sum of Even Numbers After Queries
题目
题目大意
解题思路
代码
986. Interval List Intersections
题目
题目大意
解题思路
代码
987. Vertical Order Traversal of a Binary Tree
题目
题目大意
解题思路
代码
989. Add to Array-Form of Integer
题目

- 题目大意
- 解题思路
- 代码

990. Satisfiability of Equality Equations

- 题目
- 题目大意
- 解题思路
- 代码

991. Broken Calculator

- 题目
- 题目大意
- 解题思路
- 代码

992. Subarrays with K Different Integers

- 题目
- 题目大意
- 解题思路
- 代码

993. Cousins in Binary Tree

- 题目
- 题目大意
- 解题思路
- 代码

995. Minimum Number of K Consecutive Bit Flips

- 题目
- 题目大意
- 解题思路
- 代码

996. Number of Squareful Arrays

- 题目
- 题目大意
- 解题思路
- 代码

999. Available Captures for Rook

- 题目
- 题目大意
- 解题思路
- 代码

1002. Find Common Characters

- 题目
- 题目大意
- 解题思路
- 代码

1003. Check If Word Is Valid After Substitutions

- 题目
- 题目大意
- 解题思路
- 代码

1004. Max Consecutive Ones III

- 题目
- 题目大意
- 解题思路
- 代码

1005. Maximize Sum Of Array After K Negations

- 题目
- 题目大意
- 解题思路
- 代码

1006. Clumsy Factorial

- 题目
- 题目大意
- 解题思路
- 代码

1011. Capacity To Ship Packages Within D Days

- 题目
- 题目大意
- 解题思路
- 代码

1017. Convert to Base -2

- 题目
- 题目大意
- 解题思路
- 代码

1018. Binary Prefix Divisible By 5

- 题目
- 题目大意
- 解题思路
- 代码

1019. Next Greater Node In Linked List

- 题目
- 题目大意
- 解题思路
- 代码

1020. Number of Enclaves

- 题目
- 题目大意
- 解题思路
- 代码

1021. Remove Outermost Parentheses

- 题目
- 题目大意
- 解题思路
- 代码

1025. Divisor Game

- 题目
- 题目大意
- 解题思路
- 代码

1026. Maximum Difference Between Node and Ancestor
- 题目
- 题目大意
- 解题思路
- 代码

1028. Recover a Tree From Preorder Traversal
- 题目
- 题目大意
- 解题思路
- 代码

1030. Matrix Cells in Distance Order
- 题目
- 题目大意
- 解题思路
- 代码

1037. Valid Boomerang
- 题目
- 题目大意
- 解题思路
- 代码

1038. Binary Search Tree to Greater Sum Tree
- 题目
- 题目大意
- 解题思路
- 代码

1040. Moving Stones Until Consecutive II
- 题目
- 题目大意
- 解题思路
- 代码

1047. Remove All Adjacent Duplicates In String
- 题目
- 题目大意
- 解题思路
- 代码

1048. Longest String Chain
- 题目
- 题目大意
- 解题思路
- 代码

1049. Last Stone Weight II
- 题目
- 题目大意
- 解题思路
- 代码

1051. Height Checker
- 题目
- 题目大意
- 解题思路

代码
1052. Grumpy Bookstore Owner
题目
题目大意
解题思路
代码
1054. Distant Barcodes
题目
题目大意
解题思路
代码
1073. Adding Two Negabinary Numbers
题目
题目大意
解题思路
代码
1074. Number of Submatrices That Sum to Target
题目
题目大意
解题思路
代码
1078. Occurrences After Bigram
题目
题目大意
解题思路
代码
1079. Letter Tile Possibilities
题目
题目大意
解题思路
代码
1089. Duplicate Zeros
题目
题目大意
解题思路
代码
1091. Shortest Path in Binary Matrix
题目
题目大意
解题思路
代码
1093. Statistics from a Large Sample
题目
题目大意
解题思路
代码
1105. Filling Bookcase Shelves
题目
题目大意

解题思路
代码
1108. Defanging an IP Address
题目
题目大意
解题思路
代码
1110. Delete Nodes And Return Forest
题目
题目大意
解题思路
代码
1111. Maximum Nesting Depth of Two Valid Parentheses Strings
题目
题目大意
解题思路
代码
1122. Relative Sort Array
题目
题目大意
解题思路
代码
1123. Lowest Common Ancestor of Deepest Leaves
题目
题目大意
解题思路
代码
1128. Number of Equivalent Domino Pairs
题目
题目大意
解题思路
代码
1137. N-th Tribonacci Number
题目
题目大意
解题思路
代码
1143. Longest Common Subsequence
题目
题目大意
解题思路
代码
1145. Binary Tree Coloring Game
题目
题目大意
解题思路
代码
1154. Day of the Year
题目

- 题目大意
- 解题思路
- 代码

1157. Online Majority Element In Subarray

- 题目
- 题目大意
- 解题思路
- 代码

1160. Find Words That Can Be Formed by Characters

- 题目
- 题目大意
- 解题思路
- 代码

1170. Compare Strings by Frequency of the Smallest Character

- 题目
- 题目大意
- 解题思路
- 代码

1171. Remove Zero Sum Consecutive Nodes from Linked List

- 题目
- 题目大意
- 解题思路
- 代码

1175. Prime Arrangements

- 题目
- 题目大意
- 解题思路
- 代码

1178. Number of Valid Words for Each Puzzle

- 题目
- 题目大意
- 解题思路
- 代码

1184. Distance Between Bus Stops

- 题目
- 题目大意
- 解题思路
- 代码

1185. Day of the Week

- 题目
- 题目大意
- 解题思路
- 代码

1189. Maximum Number of Balloons

- 题目
- 题目大意
- 解题思路
- 代码

1190. Reverse Substrings Between Each Pair of Parentheses

- 题目
- 题目大意
- 解题思路
- 代码

1200. Minimum Absolute Difference

- 题目
- 题目大意
- 解题思路
- 代码

1201. Ugly Number III

- 题目
- 题目大意
- 解题思路
- 代码

1202. Smallest String With Swaps

- 题目
- 题目大意
- 解题思路
- 代码

1203. Sort Items by Groups Respecting Dependencies

- 题目
- 题目大意
- 解题思路
- 代码

1207. Unique Number of Occurrences

- 题目
- 题目大意
- 解题思路
- 代码

1208. Get Equal Substrings Within Budget

- 题目
- 题目大意
- 解题思路
- 代码

1209. Remove All Adjacent Duplicates in String II

- 题目
- 题目大意
- 解题思路
- 代码

1217. Minimum Cost to Move Chips to The Same Position

- 题目
- 题目大意
- 解题思路
- 代码

1221. Split a String in Balanced Strings

- 题目
- 题目大意
- 解题思路
- 代码

1232. Check If It Is a Straight Line
- 题目
- 题目大意
- 解题思路
- 代码

1234. Replace the Substring for Balanced String
- 题目
- 题目大意
- 解题思路
- 代码

1235. Maximum Profit in Job Scheduling
- 题目
- 题目大意
- 解题思路
- 代码

1239. Maximum Length of a Concatenated String with Unique Characters
- 题目
- 题目大意
- 解题思路
- 代码

1249. Minimum Remove to Make Valid Parentheses
- 题目
- 题目大意
- 解题思路
- 代码

1252. Cells with Odd Values in a Matrix
- 题目
- 题目大意
- 解题思路
- 代码

1254. Number of Closed Islands
- 题目
- 题目大意
- 解题思路
- 代码

1260. Shift 2D Grid
- 题目
- 题目大意
- 解题思路
- 代码

1266. Minimum Time Visiting All Points
- 题目
- 题目大意
- 解题思路
- 代码

1268. Search Suggestions System
- 题目
- 题目大意
- 解题思路

代码
1275. Find Winner on a Tic Tac Toe Game
题目
题目大意
解题思路
代码
1281. Subtract the Product and Sum of Digits of an Integer
题目
题目大意
解题思路
代码
1283. Find the Smallest Divisor Given a Threshold
题目
题目大意
解题思路
代码
1287. Element Appearing More Than 25% In Sorted Array
题目
题目大意
解题思路
代码
1290. Convert Binary Number in a Linked List to Integer
题目
题目大意
解题思路
代码
1295. Find Numbers with Even Number of Digits
题目
题目大意
解题思路
代码
1299. Replace Elements with Greatest Element on Right Side
题目
题目大意
解题思路
代码
1300. Sum of Mutated Array Closest to Target
题目
题目大意
解题思路
代码
1302. Deepest Leaves Sum
题目
题目大意
解题思路
代码
1304. Find N Unique Integers Sum up to Zero
题目
题目大意

解题思路
代码
1305. All Elements in Two Binary Search Trees
题目
题目大意
解题思路
代码
1306. Jump Game III
题目
题目大意
解题思路
代码
1310. XOR Queries of a Subarray
题目
题目大意
解题思路
代码
1313. Decompress Run-Length Encoded List
题目
题目大意
解题思路
代码
1317. Convert Integer to the Sum of Two No-Zero Integers
题目
题目大意
解题思路
代码
1319. Number of Operations to Make Network Connected
题目
题目大意
解题思路
代码
1329. Sort the Matrix Diagonally
题目
题目大意
解题思路
代码
1332. Remove Palindromic Subsequences
题目
题目大意
解题思路
代码
1337. The K Weakest Rows in a Matrix
题目
题目大意
解题思路
代码
1353. Maximum Number of Events That Can Be Attended
题目

- 题目大意
- 解题思路
- 代码

1380. Lucky Numbers in a Matrix

- 题目
- 题目大意
- 解题思路
- 代码

1383. Maximum Performance of a Team

- 题目
- 题目大意
- 解题思路
- 代码

1385. Find the Distance Value Between Two Arrays

- 题目
- 题目大意
- 解题思路
- 代码

1389. Create Target Array in the Given Order

- 题目
- 题目大意
- 解题思路
- 代码

1396. Design Underground System

- 题目
- 题目大意
- 解题思路
- 代码

1423. Maximum Points You Can Obtain from Cards

- 题目
- 题目大意
- 解题思路
- 代码

1437. Check If All 1's Are at Least Length K Places Away

- 题目
- 题目大意
- 解题思路
- 代码

1438. Longest Continuous Subarray With Absolute Diff Less Than or Equal to Limit

- 题目
- 题目大意
- 解题思路
- 代码

1439. Find the Kth Smallest Sum of a Matrix With Sorted Rows

- 题目
- 题目大意
- 解题思路
- 代码

1442. Count Triplets That Can Form Two Arrays of Equal XOR

- 题目
- 题目大意
- 解题思路
- 代码

1455. Check If a Word Occurs As a Prefix of Any Word in a Sentence

- 题目
- 题目大意
- 解题思路
- 代码

1461. Check If a String Contains All Binary Codes of Size K

- 题目
- 题目大意
- 解题思路
- 代码

1463. Cherry Pickup II

- 题目
- 题目大意
- 解题思路
- 代码

1464. Maximum Product of Two Elements in an Array

- 题目
- 题目大意
- 解题思路
- 代码

1465. Maximum Area of a Piece of Cake After Horizontal and Vertical Cuts

- 题目
- 题目大意
- 解题思路
- 代码

1470. Shuffle the Array

- 题目
- 题目大意
- 解题思路
- 代码

1480. Running Sum of 1d Array

- 题目
- 题目大意
- 解题思路
- 代码

1482. Minimum Number of Days to Make m Bouquets

- 题目
- 题目大意
- 解题思路
- 代码

1486. XOR Operation in an Array

- 题目
- 题目大意
- 解题思路
- 代码

1512. Number of Good Pairs
- 题目
- 题目大意
- 解题思路
- 代码

1539. Kth Missing Positive Number
- 题目
- 题目大意
- 解题思路
- 代码

1551. Minimum Operations to Make Array Equal
- 题目
- 题目大意
- 解题思路
- 代码

1572. Matrix Diagonal Sum
- 题目
- 题目大意
- 解题思路
- 代码

1573. Number of Ways to Split a String
- 题目
- 题目大意
- 解题思路
- 代码

1579. Remove Max Number of Edges to Keep Graph Fully Traversable
- 题目
- 题目大意
- 解题思路
- 代码

1600. Throne Inheritance
- 题目
- 题目大意
- 解题思路
- 代码

1603. Design Parking System
- 题目
- 题目大意
- 解题思路
- 代码

1608. Special Array With X Elements Greater Than or Equal X
- 题目
- 题目大意
- 解题思路
- 代码

1614. Maximum Nesting Depth of the Parentheses
- 题目
- 题目大意
- 解题思路

代码
1619. Mean of Array After Removing Some Elements
题目
题目大意
解题思路
代码
1624. Largest Substring Between Two Equal Characters
题目
题目大意
解题思路
代码
1629. Slowest Key
题目
题目大意
解题思路
代码
1631. Path With Minimum Effort
题目
题目大意
解题思路
代码
1636. Sort Array by Increasing Frequency
题目
题目大意
解题思路
代码
1640. Check Array Formation Through Concatenation
题目
题目大意
解题思路
代码
1641. Count Sorted Vowel Strings
题目
题目大意
解题思路
代码
1642. Furthest Building You Can Reach
题目
题目大意
解题思路
代码
1646. Get Maximum in Generated Array
题目
题目大意
解题思路
代码
1647. Minimum Deletions to Make Character Frequencies Unique
题目
题目大意

解题思路
代码
1648. Sell Diminishing-Valued Colored Balls
题目
题目大意
解题思路
代码
1649. Create Sorted Array through Instructions
题目
题目大意
解题思路
代码
1652. Defuse the Bomb
题目
题目大意
解题思路
代码
1653. Minimum Deletions to Make String Balanced
题目
题目大意
解题思路
代码
1654. Minimum Jumps to Reach Home
题目
题目大意
解题思路
代码
1655. Distribute Repeating Integers
题目
题目大意
解题思路
代码
1656. Design an Ordered Stream
题目
题目大意
解题思路
代码
1657. Determine if Two Strings Are Close
题目
题目大意
解题思路
代码
1658. Minimum Operations to Reduce X to Zero
题目
题目大意
解题思路
代码
1659. Maximize Grid Happiness
题目

- 题目大意
- 解题思路
- 代码

1662. Check If Two String Arrays are Equivalent

- 题目
- 题目大意
- 解题思路
- 代码

1663. Smallest String With A Given Numeric Value

- 题目
- 题目大意
- 解题思路
- 代码

1664. Ways to Make a Fair Array

- 题目
- 题目大意
- 解题思路
- 代码

1665. Minimum Initial Energy to Finish Tasks

- 题目
- 题目大意
- 解题思路
- 代码

1668. Maximum Repeating Substring

- 题目
- 题目大意
- 解题思路
- 代码

1669. Merge In Between Linked Lists

- 题目
- 题目大意
- 解题思路
- 代码

1670. Design Front Middle Back Queue

- 题目
- 题目大意
- 解题思路
- 代码

1672. Richest Customer Wealth

- 题目
- 题目大意
- 解题思路
- 代码

1673. Find the Most Competitive Subsequence

- 题目
- 题目大意
- 解题思路
- 代码

1674. Minimum Moves to Make Array Complementary

- 题目
- 题目大意
- 解题思路
- 代码

1675. Minimize Deviation in Array

- 题目
- 题目大意
- 解题思路
- 代码

1678. Goal Parser Interpretation

- 题目
- 题目大意
- 解题思路
- 代码

1679. Max Number of K-Sum Pairs

- 题目
- 题目大意
- 解题思路
- 代码

1680. Concatenation of Consecutive Binary Numbers

- 题目
- 题目大意
- 解题思路
- 代码

1681. Minimum Incompatibility

- 题目
- 题目大意
- 解题思路
- 代码

1684. Count the Number of Consistent Strings

- 题目
- 题目大意
- 解题思路
- 代码

1685. Sum of Absolute Differences in a Sorted Array

- 题目
- 题目大意
- 解题思路
- 代码

1688. Count of Matches in Tournament

- 题目
- 题目大意
- 解题思路
- 代码

1689. Partitioning Into Minimum Number Of Deci-Binary Numbers

- 题目
- 题目大意
- 解题思路
- 代码

1690. Stone Game VII
- 题目
- 题目大意
- 解题思路
- 代码

1691. Maximum Height by Stacking Cuboids
- 题目
- 题目大意
- 解题思路
- 代码

1694. Reformat Phone Number
- 题目
- 题目大意
- 解题思路
- 代码

1695. Maximum Erasure Value
- 题目
- 题目大意
- 解题思路
- 代码

1696. Jump Game VI
- 题目
- 题目大意
- 解题思路
- 代码

1700. Number of Students Unable to Eat Lunch
- 题目
- 题目大意
- 解题思路
- 代码

1704. Determine if String Halves Are Alike
- 题目
- 题目大意
- 解题思路
- 代码

1710. Maximum Units on a Truck
- 题目
- 题目大意
- 解题思路
- 代码

1716. Calculate Money in Leetcode Bank
- 题目
- 题目大意
- 解题思路
- 代码

1720. Decode XORed Array
- 题目
- 题目大意
- 解题思路

代码
1721. Swapping Nodes in a Linked List
题目
题目大意
解题思路
代码
1725. Number Of Rectangles That Can Form The Largest Square
题目
题目大意
解题思路
代码
1732. Find the Highest Altitude
题目
题目大意
解题思路
代码
1734. Decode XORed Permutation
题目
题目大意
解题思路
代码
1736. Latest Time by Replacing Hidden Digits
题目
题目大意
解题思路
代码
1738. Find Kth Largest XOR Coordinate Value
题目
题目大意
解题思路
代码
1742. Maximum Number of Balls in a Box
题目
题目大意
解题思路
代码
1744. Can You Eat Your Favorite Candy on Your Favorite Day?
题目
题目大意
解题思路
代码
1748. Sum of Unique Elements
题目
题目大意
解题思路
代码
1752. Check if Array Is Sorted and Rotated
题目
题目大意

解题思路
代码
1758. Minimum Changes To Make Alternating Binary String
题目
题目大意
解题思路
代码

第一章 序章

关于 LeetCode

说到 LeetCode，作为一个程序员来说，应该不陌生，近几年参加面试都会提到它。国内外的程序员用它刷题主要是为了面试。据历史记载，这个网站 2011 年就成立了，马上就要到自己 10 周年的生日了。每周举行周赛，双周赛，月赛，在有限时间内编码，确实非常能考验人的算法能力。一些大公司赞助冠名的比赛获得前几名除了有奖品，还能直接拿到内推的机会。

什么是 Cookbook

直译的话就是烹饪书，教你做各种食谱美食的书。经常看 O'Reilly 技术书的同学对这个名词会很熟悉。一般动手操作，实践类的书都会有这个名字。

为什么会写这个开源书

笔者刷题刷了一年了，想和大家分享分享一些做题心得，解题方法。想和有相同爱好的人交个朋友，一起交流学习。对于自己来说，写题解也是一种提高。把一道深奥的题目讲给一点都没有头绪的人，并能让他完全听懂，很能锻炼人的表达能力。在讲解中很可能还会遇到听者的一些提问，这些问题可能是自己的知识漏洞，强迫自己去弥补。笔者在公司做过相关的分享，感受很深，双方受益都还不错。

> 另外，在大学期间，笔者做题的时候最讨厌写题解，感觉是浪费时间，用更多的时间去做更多的题。现在不知道算不算是“出来混的，总是要还的”。

关于书的封面

常看 O'Reilly 动物书的同学一看这个封面就知道是向他们致敬。确实是这个目的。O'Reilly 的封面动物都是稀缺动物，并且画风都是黑白素描风。这些动物都有版权了，所以只能在网上找没有版权的黑白素描风的图片。常见的能找到 40 张这种风格的图片。不过用的人太多了，笔者费劲的找了其他几张这种图片，这张孔雀开屏是其中一张。孔雀开屏的意义是希望大家刷完 LeetCode 以后，提高了自身的算法能力，在人生的舞台上开出自己的“屏”。全书配色也都是绿色，因为这是 AC 的颜色。

关于作者

笔者是一个刚刚入行一年半的 gopher 新人，还请各位大佬多多指点小弟我。大学参加了 3 年 ACM-ICPC，但是由于资质不高，没有拿到一块金牌。所以在算法方面，我对自己的评价算是新手吧。参加 ACM-ICPC 最大的收获是训练了思维能力，这种能力也会运用到生活中。其次是认识了很多国内很聪明的选手，看到了自己和他们的差距。最后，就是那 200 多页，有些自己都没有完全理解的，打印的密密麻麻的算法模板。知识学会了，终身都是自己的，没有学会，那些知识都是身外之物。

笔者从 2019 年 3 月 25 号开始刷题，到 2020 年 3 月 25 号，整整一年的时间。原计划是每天一题。实际上每天有时候不止一题，最终完成了 600+：

> 一个温馨提示：笔者本以为每天做一题，会让这个 submissions 图全绿，但是我发现我错了。如果你也想坚持，让这个图全绿，一定要注意以下的问题：LeetCode 服务器是在 +0 时区的，这个图也是按照这个时区计算的。也就是说，中国每天早上 8 点之前，是算前一天的！也是因为时区的问题，导致我空白了这 22 个格子。比如有一道 Hard 题很难，当天工作也很多，晚上下班回家想出来了就到第二天凌晨了。于是再做一题当做第二天的量。结果会发现这 2 题都算前一天的。有时候笔者早上 6 点起床刷题，提交以后也都是前一天的。
>
> (当然这些都是过去了，不重要了，全当是奋斗路上的一些小插曲)

2020 年笔者肯定还会继续刷题，因为还没有达到自己的一些目标。可能会朝着 1000 题奋进，也有可能刷到 800 题的时候回头开始二刷，三刷。(不达目的不罢休吧~)

关于书中的代码

代码都放在 github repo 中，按题号可以搜索到题目。
本书题目的代码都已经 beats 100% 了。没有 beats 100% 题解就没有放到本书中了。那些题目笔者会继续优化到 100% 再放进来。

有可能读者会问，为何要追求 beats 100%。笔者认为优化到 beats 100% 才算是把这题做出感觉了。有好几道 Hard 题，笔者都用暴力解法 AC 了，然后只 beats 了 5%。这题就如同没做一样。而且面试中如果给了这样的答案，面试官也不会满意，“还有没有更优解？”。如果通过自己的思考能给出更优解，面试官会更满意一些。

LeetCode 统计代码运行时长会有波动的，相同的代码提交 10 次可能就会 beats 100% 了。笔者开始没有发现这个问题，很多题用正确的代码连续交了很多次，一年提交 3400+ 次，导致我的正确率也变的奇高。😢

当然，如果还有其他更优美的解法，也能 beats 100% 的，欢迎提交 PR，笔者和大家一起学习。

目标读者

想通过 LeetCode 提高算法能力的编程爱好者。

编程语言

本书的算法全部用 Go 语言实现。

使用说明

- 本电子书的左上角有搜索栏，可以迅速帮你找到你想看的章节和题号。
- 本电子书每页都接入了 Gitalk，每一页的最下方都有评论框可以评论，如果没有显示出来，请检查自己的网络。
- 关于题解，笔者建议这样使用：先自己读题，思考如何解题。如果 15 分钟还没有思路，那么先看笔者的解题思路，但是不要看代码。有思路以后自己用代码实现一遍。如果完全不会写，那就看笔者提供的代码，找出自己到底哪里不会写，找出问题记下来，这就是自己要弥补的知识漏洞。如果自己实现出来了，提交以后有错误，自己先 debug。AC 以后没有到 100% 也先自己思考如何优化。如果每道题自己都能优化到 100% 了，那么一段时间以后进步会很大。所以总的来说，实在没思路，看解题思路；实在优化不到 100%，看看代码。

互动与勘误

如果书中文章有所遗漏，欢迎点击所在页面下边的 edit 按钮进行评论和互动，感谢您的支持与帮助。

最后

一起开始刷题吧~

本作品采用 知识署名-非商业性使用-禁止演绎 （BY-NC-ND） 4.0 国际许可协议 进行许可。

题解里面的所有题目版权均归 LeetCode 和 力扣中国 所有

数据结构知识

以下是笔者整理的数据结构相关的知识。希望能把常见的数据结构都枚举穷尽。如有遗漏，欢迎大家赐教，提PR。相关题目还在慢慢整理中，讲解的文章还在创作中。

> 刷题只是提升算法能力的手段，最终目的应该是提升自我的思维能力，知识需要凝结成块，那么就把这些总结在第一章这两节中，让它得到升华吧~希望读者在刷完题之后再回过头来看这个表格，能很清晰的梳理自己的知识体系，查缺补漏，尽早完善。

数据结构	变种	相关题目	讲解文章
顺序线性表：向量 Vector			
单链表 Singly Linked List	1. 双向链表 Double Linked Lists 2. 静态链表 Static List 3. 对称矩阵 Symmetric Matrix 4. 稀疏矩阵 Sparse Matrix		
哈希表 Hash Table	1. 散列函数 Hash Function 2. 解决碰撞/填充因子 Collision Resolution		
栈和队列 Stack & Queue	1. 广义表 Generalized List/GList 2. 双端队列 Deque		
队列 Queue	1. 链表实现 Linked List Implementation 2. 循环数组实现 ArrayQueue 3. 双端队列 Deque 4. 优先队列 Priority Queue 5. 循环队列 Circular Queue		
字符串 String	1. KMP 算法 2. 有限状态自动机 3. 模式匹配有限状态自动机 4. BM 模式匹配算法 5. BM-KMP 算法 6. BF 算法		
树 Tree	1. 二叉树 Binary Tree 2. 并查集 Union-Find 3. Huffman 树		
数组实现的堆 Heap	1. 极大堆和极小堆 Max Heap and Min Heap 2. 极大极小堆 3. 双端堆 Deap 4. d 叉堆		
树实现的堆 Heap	1. 左堆 Leftist Tree/Leftist Heap 2. 扁堆 3. 二项式堆		

	4. 斐波那契堆 Fibonacco Heap 5. 配对堆 Pairing Heap		
查找 Search	1. 哈希表 Hash 2. 跳跃表 Skip List 3. 排序二叉树 Binary Sort Tree 4. AVL 树 5. B 树 / B+ 树 / B* 树 6. AA 树 7. 红黑树 Red Black Tree 8. 排序二叉堆 Binary Heap 9. Splay 树 10. 双链树 Double Chained Tree 11. Trie 树 12. R 树		

算法知识

以下是笔者整理的算法相关的知识。希望能把常见的算法都枚举穷尽。如有遗漏，欢迎大家赐教，提 PR。相关题目还在慢慢整理中，讲解的文章还在创作中。

> 刷题只是提升算法能力的手段，最终目的应该是提升自我的思维能力，知识需要凝结成块，那么就把这些总结在第一章这两节中，让它得到升华吧~希望读者在刷完题之后再回过头来看这个表格，能很清晰的梳理自己的知识体系，查缺补漏，尽早完善。

算法	具体类型	相关题目	讲解文章
排序算法	1. 冒泡排序 2. 插入排序 3. 选择排序 4. 希尔 Shell 排序 5. 快速排序 6. 归并排序 7. 堆排序 8. 线性排序算法 9. 自省排序 10. 间接排序 11. 计数排序 12. 基数排序 13. 桶排序 14. 外部排序 - k 路归并败者树 15. 外部排序 - 最佳归并树		
递归		1. 二分搜索/查找 2. 大整数的乘法 3. Strassen 矩阵乘法 4. 棋盘覆盖	

与分治		5. 合并排序 6. 快速排序 7. 线性时间选择 8. 最接近点对问题 9. 循环赛日程表	
动态规划		1. 矩阵连乘问题 2. 最长公共子序列 3. 最大子段和 4. 凸多边形最优三角剖分 5. 多边形游戏 6. 图像压缩 7. 电路布线 8. 流水作业调度 9. 0-1 背包问题/背包九讲 10. 最优二叉搜索树 11. 动态规划加速原理 12. 树型 DP	
贪心		1. 活动安排问题 2. 最优装载 3. 哈夫曼编码 4. 单源最短路径 5. 最小生成树 6. 多机调度问题	
回溯法		1. 装载问题 2. 批处理作业调度 3. 符号三角形问题 4. n 后问题 5. 0-1 背包问题 6. 最大团问题 7. 图的 m 着色问题 8. 旅行售货员问题 9. 圆排列问题 10. 电路板排列问题 11. 连续邮资问题	
搜索	1. 枚举 2. DFS 3. BFS 4. 启发式搜索		
随机	1. 随机数 2. 数值随机化算法 3. Sherwood 舍伍德算法	1. 计算 π 值 2. 计算定积分 3. 解非线性方程组 4. 线性时间选择算法 5. 跳跃表	

化	4. Las Vegas 拉斯维加斯算法 5. Monte Carlo 蒙特卡罗算法	6. n 后问题 7. 整数因子分解 8. 主元素问题 9. 素数测试	
图论	1. 遍历 DFS / BFS 2. AOV / AOE 网络 3. Kruskal 算法(最小生成树) 4. Prim 算法(最小生成树) 5. Boruvka 算法(最小生成树) 6. Dijkstra 算法(单源最短路径) 7. Bellman-Ford 算法(单源最短路径) 8. SPFA 算法(单源最短路径) 9. Floyd 算法(多源最短路径) 10. Johnson 算法(多源最短路径) 11. Fleury 算法(欧拉回路) 12. Ford-Fulkerson 算法(最大网络流增广路) 13. Edmonds-Karp 算法(最大网络流) 14. Dinic 算法(最大网络流) 15. 一般预流推进算法 16. 最高标号预流推进 HLPP 算法 17. Primal-Dual 原始对偶算法(最小费用流)18. Kosaraju 算法(有向图强连通分量) 19. Tarjan 算法(有向图强连通分量) 20. Gabow 算法(有向图强连通分量) 21. 匈牙利算法(二分图匹配) 22. Hopcroft－Karp 算法(二分图匹配) 23. kuhn munkras 算法(二分图最佳匹配) 24. Edmonds' Blossom-Contraction 算法(一般图匹配)	1. 图遍历 2. 有向图和无向图的强弱连通性 3. 割点/割边 3. AOV 网络和拓扑排序 4. AOE 网络和关键路径 5. 最小代价生成树/次小生成树 6. 最短路径问题/第 K 短路问题 7. 最大网络流问题 8. 最小费用流问题 9. 图着色问题 10. 差分约束系统 11. 欧拉回路 12. 中国邮递员问题 13. 汉密尔顿回路 14. 最佳边割集/最佳点割集/最小边割集/最小点割集/最小路径覆盖/最小点集覆盖 15. 边覆盖集 16. 二分图完美匹配和最大匹配问题 17. 仙人掌图 18. 弦图 19. 稳定婚姻问题 20. 最大团问题	
数论		1. 最大公约数 2. 最小公倍数 3. 分解质因数 4. 素数判定 5. 进制转换 6. 高精度计算 7. 整除问题 8. 同余问题 9. 欧拉函数 10. 扩展欧几里得 11. 置换群 12. 母函数 13. 离散变换 14. 康托展开 15. 矩阵	

		16. 向量 17. 线性方程组 18. 线性规划	
几何		1. 凸包 - Gift wrapping 2. 凸包 - Graham scan 3. 线段问题 4. 多边形和多面体相关问题	
NP完全	1. 计算模型 2. P 类与 NP 类问题 3. NP 完全问题 4. NP 完全问题的近似算法	1. 随机存取机 RAM 2. 随机存取存储程序机 RASP 3. 图灵机 4. 非确定性图灵机 5. P 类与 NP 类语言 6. 多项式时间验证 7. 多项式时间变换 8. Cook定理 9. 合取范式的可满足性问题 CNF-SAT 10. 3 元合取范式的可满足性问题 3-SAT 11. 团问题 CLIQUE 12. 顶点覆盖问题 VERTEX-COVER 13. 子集和问题 SUBSET-SUM 14. 哈密顿回路问题 HAM-CYCLE 15. 旅行售货员问题 TSP 16. 顶点覆盖问题的近似算法 17. 旅行售货员问题近似算法 18. 具有三角不等式性质的旅行售货员问题 19. 一般的旅行售货员问题 20. 集合覆盖问题的近似算法 21. 子集和问题的近似算法 22. 子集和问题的指数时间算法 23. 子集和问题的多项式时间近似格式	
位运算	位操作包括： 1. 取反（NOT） 2. 按位或（OR） 3. 按位异或（XOR） 4. 按位与（AND） 5. 移位: 是一个二元运算符，用来将一个二进制数中的每一位全部都向一个方向移动指定位，溢出的部分将被舍	1.数字范围按位与 2.UTF-8 编码验证 3.数字转换为十六进制数 4.找出最长的超赞子字符串 5.数组异或操作 6.幂集 7.位1的个数 8.二进制表示中质数个计算置	[力扣：位运算]()

	弃，而空缺的部分填入一定的值。	位 9.子数组异或查询	

时间复杂度和空间复杂度

一. 时间复杂度数据规模

1s 内能解决问题的数据规模：10^6 ~ 10^7

- O(n^2) 算法可以处理 10^4 级别的数据规模(保守估计，处理 1000 级别的问题肯定没问题)
- O(n) 算法可以处理 10^8 级别的数据规模(保守估计，处理 10^7 级别的问题肯定没问题)
- O(nlog n) 算法可以处理 10^7 级别的数据规模(保守估计，处理 10^6 级别的问题肯定没问题)

	数据规模	时间复杂度	算法举例
1	10	O(n!)	permutation 排列
2	20~30	O(2^n)	combination 组合
3	50	O(n^4)	DFS 搜索、DP 动态规划
4	100	O(n^3)	任意两点最短路径、DP 动态规划
5	1000	O(n^2)	稠密图、DP 动态规划
6	10^6	O(nlog n)	排序，堆，递归与分治
7	10^7	O(n)	DP 动态规划、图遍历、拓扑排序、树遍历
8	10^9	O(sqrt(n))	筛素数、求平方根
9	10^10	O(log n)	二分搜索
10	+∞	O(1)	数学相关算法

一些具有迷惑性的例子：

```
void hello (int n){
    for( int sz = 1 ; sz < n ; sz += sz)
        for( int i = 1 ; i < n ; i ++)
            cout << "Hello" << endl;
}
```

上面这段代码的时间复杂度是 O(nlog n) 而不是 O(n^2)

```
bool isPrime (int n){
    for( int x = 2 ; x * x <= n ; x ++ )
        if( n % x == 0)
            return false;
    return true;
}
```

上面这段代码的时间复杂度是 O(sqrt(n)) 而不是 O(n)。

再举一个例子，有一个字符串数组，将数组中的每一个字符串按照字母序排序，之后再降整个字符串数组按照字典序排序。两步操作的整体时间复杂度是多少呢?

如果回答是 O(n*nlog n + nlog n) = O(n^2log n)，这个答案是错误的。字符串的长度和数组的长度是没有关系的，所以这两个变量应该单独计算。假设最长的字符串长度为 s，数组中有 n 个字符串。对每个字符串排序的时间复杂度是 O(slog s)，将数组中每个字符串都按照字母序排序的时间复杂度是 O(n * slog s)。

将整个字符串数组按照字典序排序的时间复杂度是 O(s * nlog n)。排序算法中的 O(nlog n) 是比较的次数，由于比较的是整型数字，所以每次比较是 O(1)。但是字符串按照字典序比较，时间复杂度是 O(s)。所以字符串数组按照字典序排序的时间复杂度是 O(s * nlog n)。所以整体复杂度是 O(n * slog s) + O(s * nlog n) = O(n*slog s + s*nlogn) = O(n*s*(log s + log n))

二. 空间复杂度

递归调用是有空间代价的，递归算法需要保存递归栈信息，所以花费的空间复杂度会比非递归算法要高。

```
int sum( int n ){
    assert( n >= 0 )
    int ret = 0;
    for ( int i = 0 ; i <= n ; i++)
        ret += i;
    return ret;
}
```

上面算法的时间复杂度为 O(n)，空间复杂度 O(1)。

```
int sum( int n ){
    assert( n >= 0 )
    if ( n == 0 )
        return 0;
    return n + sum( n - 1);
}
```

上面算法的时间复杂度为 O(n)，空间复杂度 O(n)。

三. 递归的时间复杂度

1. 只有一次递归调用

如果递归函数中，只进行了一次递归调用，且递归深度为 depth，在每个递归函数中，时间复杂度为 T，那么总体的时间复杂度为 O(T * depth)

举个例子：

```
int binarySearch(int arr[], int l, int r, int target){
  if( l > r)
      return -1;
    int mid = l + (r-l)/2;//防溢出
    if(arr[mid] == target)
        return mid;
    else if (arr[mid]>target)
        return binarySearch(arr,l,mid-1,target);
    eles
        return binarySearch(arr,mid+1,r,target);
}
```

在二分查找的递归实现中，只递归调用了自身。递归深度是 log n ，每次递归里面的复杂度是 O(1) 的，所以二分查找的递归实现的时间复杂度为 O(log n) 的。

2. 只有多次递归调用

针对多次递归调用的情况，就需要看它的计算调用的次数了。通常可以画一颗递归树来看。举例：

```
int f(int n){
    assert( n >= 0 );
    if( n ==0 )
        return 1;
    return f( n - 1 ) + f ( n - 1 );
```

上述这次递归调用的次数为 $2^0 + 2^1 + 2^2 + ... + 2^n = 2^{n+1} - 1 = O(2^n)$

> 关于更加复杂的递归的复杂度分析，请参考，主定理。主定理中针对各种复杂情况都给出了正确的结论。

第二章 算法专题

本来天真的认为，把 LeetCode 所有题都完整刷一遍，就可以完整这本书了。经过事实证明，确实是天真了。因为 LeetCode 每天都会增加新题，有时候工作忙了，刷题进度就完全追不上题目更新的速度了。而且以我当前的刷题速度，一年才完成 500+，一年 LeetCode 也会更新 400+ 多题，要起码 5~10 年才能把所有的题目刷完。时间太长了。所以先给自己定了一个小目标，500 题就先把书写出来，总结这个阶段的刷题心得，和大家一起交流。要想把 LeetCode 所有题目都刷完，看来这本书要迭代 5 ~ 10 个版本了(一年迭代一版)。

那么这一章就把已经刷完了的专题都整理一遍。有相似套路的题目都放在一起，如果想快速面试的话，其实相同的题目刷 2，3 道就可以了。相同类型的题目非常熟练的情况下，再多刷几道也是做无用功。

做到目前为止，笔者认为动态规划是最灵活的类型，这类题目没有一个模板可以给你套用，它也是算法之优雅的地方。笔者认为称它为算法的艺术不为过。动态规划这类型，笔者也还没有刷完，只刷了一部分，还在学习中。

那么就分享一下笔者目前刷过的题，和有相似点的题目吧。

Array

No.	Title	Solution	Difficulty	TimeComplexity	SpaceComplexity	Favorite	Acceptance
0001	Two Sum	Go	Easy	O(n)	O(n)		47.1%
0004	Median of Two Sorted Arrays	Go	Hard				32.0%
0011	Container With Most Water	Go	Medium	O(n)	O(1)		53.0%
0015	3Sum	Go	Medium	O(n^2)	O(n)	❤️	28.7%
0016	3Sum Closest	Go	Medium	O(n^2)	O(1)	❤️	46.4%
0018	4Sum	Go	Medium	O(n^3)	O(n^2)	❤️	35.6%
0026	Remove Duplicates from Sorted Array	Go	Easy	O(n)	O(1)		47.2%
0027	Remove Element	Go	Easy	O(n)	O(1)		49.8%

0031	Next Permutation	Go	Medium				34.2%
0033	Search in Rotated Sorted Array	Go	Medium				36.4%
0034	Find First and Last Position of Element in Sorted Array	Go	Medium				38.1%
0035	Search Insert Position	Go	Easy				42.8%
0039	Combination Sum	Go	Medium	O(n log n)	O(n)		60.6%
0040	Combination Sum II	Go	Medium	O(n log n)	O(n)		50.9%
0041	First Missing Positive	Go	Hard	O(n)	O(n)		34.4%
0042	Trapping Rain Water	Go	Hard	O(n)	O(1)	❤️	52.3%
0045	Jump Game II	Go	Medium				33.1%
0048	Rotate Image	Go	Medium	O(n)	O(1)		62.0%
0053	Maximum Subarray	Go	Easy	O(n)	O(n)		48.1%
0054	Spiral Matrix	Go	Medium	O(n)	O(n^2)		37.3%
0055	Jump Game	Go	Medium				35.6%
0056	Merge Intervals	Go	Medium	O(n log n)	O(1)		42.0%
0057	Insert Interval	Go	Medium	O(n)	O(1)		35.8%
0059	Spiral Matrix II	Go	Medium	O(n)	O(n^2)		58.9%
0062	Unique Paths	Go	Medium	O(n^2)	O(n^2)		56.9%
0063	Unique Paths II	Go	Medium	O(n^2)	O(n^2)		36.0%
0064	Minimum Path Sum	Go	Medium	O(n^2)	O(n^2)		57.0%
0066	Plus One	Go	Easy				42.1%
0073	Set Matrix Zeroes	Go	Medium				45.0%
0074	Search a 2D Matrix	Go	Medium				38.8%
0075	Sort Colors	Go	Medium	O(n)	O(1)	❤️	50.6%
0078	Subsets	Go	Medium	O(n^2)	O(n)	❤️	66.6%
0079	Word Search	Go	Medium	O(n^2)	O(n^2)	❤️	37.7%
0080	Remove Duplicates from Sorted Array II	Go	Medium	O(n)	O(1		46.8%
0081	Search in Rotated Sorted Array II	Go	Medium				33.9%
0084	Largest Rectangle in Histogram	Go	Hard	O(n)	O(n)	❤️	38.0%
0088	Merge Sorted Array	Go	Easy	O(n)	O(1)	❤️	41.2%
0090	Subsets II	Go	Medium	O(n^2)	O(n)	❤️	49.7%
0105	Construct Binary Tree from Preorder and Inorder Traversal	Go	Medium				53.8%
0106	Construct Binary Tree from Inorder and Postorder Traversal	Go	Medium				51.0%
0118	Pascal's Triangle	Go	Easy				56.6%
0119	Pascal's Triangle II	Go	Easy				53.1%
0120	Triangle	Go	Medium	O(n^2)	O(n)		47.3%
0121	Best Time to Buy and Sell Stock	Go	Easy	O(n)	O(1)		52.1%
0122	Best Time to Buy and Sell Stock II	Go	Easy	O(n)	O(1)		59.3%
0126	Word Ladder II	Go	Hard	O(n)	O(n^2)	❤️	24.2%
0128	Longest Consecutive Sequence	Go	Medium				47.3%
0152	Maximum Product Subarray	Go	Medium	O(n)	O(1)		33.1%
0153	Find Minimum in Rotated Sorted Array	Go	Medium				46.7%
0154	Find Minimum in Rotated Sorted Array II	Go	Hard				42.2%
0162	Find Peak Element	Go	Medium				44.3%

0167	Two Sum II - Input array is sorted	Go	Easy	O(n)	O(1)		56.1%
0169	Majority Element	Go	Easy				60.6%
0189	Rotate Array	Go	Medium				36.9%
0209	Minimum Size Subarray Sum	Go	Medium	O(n)	O(1)		40.4%
0216	Combination Sum III	Go	Medium	O(n)	O(1)	❤️	61.3%
0217	Contains Duplicate	Go	Easy	O(n)	O(n)		57.5%
0219	Contains Duplicate II	Go	Easy	O(n)	O(n)		39.3%
0228	Summary Ranges	Go	Easy				43.2%
0229	Majority Element II	Go	Medium				39.5%
0268	Missing Number	Go	Easy				56.1%
0283	Move Zeroes	Go	Easy	O(n)	O(1)		58.9%
0287	Find the Duplicate Number	Go	Medium	O(n)	O(1)	❤️	58.2%
0414	Third Maximum Number	Go	Easy				30.8%
0448	Find All Numbers Disappeared in an Array	Go	Easy				56.5%
0457	Circular Array Loop	Go	Medium				30.5%
0485	Max Consecutive Ones	Go	Easy				53.1%
0509	Fibonacci Number	Go	Easy				67.8%
0532	K-diff Pairs in an Array	Go	Medium	O(n)	O(n)		36.0%
0561	Array Partition I	Go	Easy				73.9%
0566	Reshape the Matrix	Go	Easy	O(n^2)	O(n^2)		61.1%
0581	Shortest Unsorted Continuous Subarray	Go	Medium				33.2%
0605	Can Place Flowers	Go	Easy				31.6%
0628	Maximum Product of Three Numbers	Go	Easy	O(n)	O(1)		46.7%
0643	Maximum Average Subarray I	Go	Easy				42.2%
0661	Image Smoother	Go	Easy				52.6%
0665	Non-decreasing Array	Go	Medium				20.9%
0667	Beautiful Arrangement II	Go	Medium				58.9%
0674	Longest Continuous Increasing Subsequence	Go	Easy				46.4%
0695	Max Area of Island	Go	Medium				66.6%
0697	Degree of an Array	Go	Easy				54.7%
0713	Subarray Product Less Than K	Go	Medium	O(n)	O(1)		40.8%
0714	Best Time to Buy and Sell Stock with Transaction Fee	Go	Medium	O(n)	O(1)		58.5%
0717	1-bit and 2-bit Characters	Go	Easy				46.3%
0718	Maximum Length of Repeated Subarray	Go	Medium				50.9%
0719	Find K-th Smallest Pair Distance	Go	Hard				32.9%
0724	Find Pivot Index	Go	Easy				47.2%
0729	My Calendar I	Go	Medium				54.1%
0746	Min Cost Climbing Stairs	Go	Easy	O(n)	O(1)		53.5%
0766	Toeplitz Matrix	Go	Easy	O(n)	O(1)		65.9%
0775	Global and Local Inversions	Go	Medium				46.1%
0795	Number of Subarrays with Bounded Maximum	Go	Medium				51.7%
0830	Positions of Large Groups	Go	Easy				50.7%
0832	Flipping an Image	Go	Easy				78.6%
0867	Transpose Matrix	Go	Easy	O(n)	O(1)		61.8%

0870	Advantage Shuffle	Go	Medium				50.7%
0888	Fair Candy Swap	Go	Easy				59.3%
0891	Sum of Subsequence Widths	Go	Hard	O(n log n)	O(1)		33.4%
0896	Monotonic Array	Go	Easy				57.8%
0907	Sum of Subarray Minimums	Go	Medium	O(n)	O(n)	❤️	32.9%
0914	X of a Kind in a Deck of Cards	Go	Easy				33.7%
0918	Maximum Sum Circular Subarray	Go	Medium				34.6%
0922	Sort Array By Parity II	Go	Easy	O(n)	O(1)		70.7%
0969	Pancake Sorting	Go	Medium	O(n)	O(1)	❤️	69.0%
0977	Squares of a Sorted Array	Go	Easy	O(n)	O(1)		71.6%
0978	Longest Turbulent Subarray	Go	Medium				46.7%
0985	Sum of Even Numbers After Queries	Go	Easy				60.5%
0989	Add to Array-Form of Integer	Go	Easy				45.0%
0999	Available Captures for Rook	Go	Easy				67.7%
1002	Find Common Characters	Go	Easy				68.7%
1011	Capacity To Ship Packages Within D Days	Go	Medium				60.5%
1018	Binary Prefix Divisible By 5	Go	Easy				47.6%
1040	Moving Stones Until Consecutive II	Go	Medium				54.8%
1051	Height Checker	Go	Easy				73.0%
1052	Grumpy Bookstore Owner	Go	Medium				56.1%
1074	Number of Submatrices That Sum to Target	Go	Hard				65.2%
1089	Duplicate Zeros	Go	Easy				51.5%
1122	Relative Sort Array	Go	Easy				68.0%
1128	Number of Equivalent Domino Pairs	Go	Easy				46.0%
1157	Online Majority Element In Subarray	Go	Hard				41.2%
1160	Find Words That Can Be Formed by Characters	Go	Easy				67.9%
1170	Compare Strings by Frequency of the Smallest Character	Go	Medium				60.5%
1184	Distance Between Bus Stops	Go	Easy				53.9%
1185	Day of the Week	Go	Easy				60.5%
1200	Minimum Absolute Difference	Go	Easy				67.1%
1202	Smallest String With Swaps	Go	Medium				49.5%
1208	Get Equal Substrings Within Budget	Go	Medium				44.7%
1217	Minimum Cost to Move Chips to The Same Position	Go	Easy				70.7%
1232	Check If It Is a Straight Line	Go	Easy				42.8%
1252	Cells with Odd Values in a Matrix	Go	Easy				78.5%
1260	Shift 2D Grid	Go	Easy				61.9%
1266	Minimum Time Visiting All Points	Go	Easy				79.2%
1275	Find Winner on a Tic Tac Toe Game	Go	Easy				53.0%
1287	Element Appearing More Than 25% In Sorted Array	Go	Easy				60.0%
1295	Find Numbers with Even Number of Digits	Go	Easy				78.3%
1299	Replace Elements with Greatest Element on Right Side	Go	Easy				74.5%
1300	Sum of Mutated Array Closest to Target	Go	Medium				42.6%

1304	Find N Unique Integers Sum up to Zero	Go	Easy				76.5%
1313	Decompress Run-Length Encoded List	Go	Easy				85.5%
1329	Sort the Matrix Diagonally	Go	Medium				81.4%
1337	The K Weakest Rows in a Matrix	Go	Easy				72.0%
1380	Lucky Numbers in a Matrix	Go	Easy				70.3%
1385	Find the Distance Value Between Two Arrays	Go	Easy				66.6%
1389	Create Target Array in the Given Order	Go	Easy				85.1%
1423	Maximum Points You Can Obtain from Cards	Go	Medium				48.5%
1437	Check If All 1's Are at Least Length K Places Away	Go	Easy				61.1%
1438	Longest Continuous Subarray With Absolute Diff Less Than or Equal to Limit	Go	Medium				44.7%
1442	Count Triplets That Can Form Two Arrays of Equal XOR	Go	Medium				72.8%
1464	Maximum Product of Two Elements in an Array	Go	Easy				76.9%
1465	Maximum Area of a Piece of Cake After Horizontal and Vertical Cuts	Go	Medium				36.8%
1470	Shuffle the Array	Go	Easy				88.1%
1480	Running Sum of 1d Array	Go	Easy				88.8%
1482	Minimum Number of Days to Make m Bouquets	Go	Medium				52.1%
1486	XOR Operation in an Array	Go	Easy				83.9%
1512	Number of Good Pairs	Go	Easy				87.6%
1539	Kth Missing Positive Number	Go	Easy				54.7%
1572	Matrix Diagonal Sum	Go	Easy				77.9%
1608	Special Array With X Elements Greater Than or Equal X	Go	Easy				61.4%
1619	Mean of Array After Removing Some Elements	Go	Easy				64.5%
1629	Slowest Key	Go	Easy				59.0%
1636	Sort Array by Increasing Frequency	Go	Easy				67.2%
1640	Check Array Formation Through Concatenation	Go	Easy				55.5%
1646	Get Maximum in Generated Array	Go	Easy				52.8%
1652	Defuse the Bomb	Go	Easy				60.5%
1656	Design an Ordered Stream	Go	Easy				82.2%
1672	Richest Customer Wealth	Go	Easy				88.1%
1700	Number of Students Unable to Eat Lunch	Go	Easy				67.6%
1732	Find the Highest Altitude	Go	Easy				79.2%
1738	Find Kth Largest XOR Coordinate Value	Go	Medium				62.8%
1742	Maximum Number of Balls in a Box	Go	Easy				73.2%
1748	Sum of Unique Elements	Go	Easy				74.6%
1752	Check if Array Is Sorted and Rotated	Go	Easy				44.5%
1758	Minimum Changes To Make Alternating Binary String	Go	Easy				58.2%

String

No.	Title	Solution	Difficulty	TimeComplexity	SpaceComplexity	Favorite	Acceptance
0003	Longest Substring Without Repeating Characters	Go	Medium	O(n)	O(1)	❤️	31.8%
0005	Longest Palindromic Substring	Go	Medium				30.9%
0006	ZigZag Conversion	Go	Medium				38.8%
0008	String to Integer (atoi)	Go	Medium				15.9%
0012	Integer to Roman	Go	Medium				57.5%
0013	Roman to Integer	Go	Easy				57.3%
0017	Letter Combinations of a Phone Number	Go	Medium	O(log n)	O(1)		50.5%
0020	Valid Parentheses	Go	Easy	O(log n)	O(1)		40.2%
0022	Generate Parentheses	Go	Medium	O(log n)	O(1)		66.8%
0028	Implement strStr()	Go	Easy	O(n)	O(1)		35.6%
0030	Substring with Concatenation of All Words	Go	Hard	O(n)	O(n)	❤️	26.7%
0032	Longest Valid Parentheses	Go	Hard				30.1%
0043	Multiply Strings	Go	Medium				35.5%
0049	Group Anagrams	Go	Medium	O(n log n)	O(n)		60.5%
0067	Add Binary	Go	Easy				47.8%
0071	Simplify Path	Go	Medium	O(n)	O(n)		35.3%
0076	Minimum Window Substring	Go	Hard	O(n)	O(n)	❤️	36.6%
0091	Decode Ways	Go	Medium	O(n)	O(n)		27.4%
0093	Restore IP Addresses	Go	Medium	O(n)	O(n)	❤️	38.5%
0097	Interleaving String	Go	Medium				33.6%
0115	Distinct Subsequences	Go	Hard				40.4%
0125	Valid Palindrome	Go	Easy	O(n)	O(1)		39.0%
0126	Word Ladder II	Go	Hard	O(n)	O(n^2)	❤️	24.2%
0151	Reverse Words in a String	Go	Medium				24.8%
0227	Basic Calculator II	Go	Medium				39.1%
0344	Reverse String	Go	Easy	O(n)	O(1)		71.3%
0345	Reverse Vowels of a String	Go	Easy	O(n)	O(1)		45.6%
0385	Mini Parser	Go	Medium				34.8%
0387	First Unique Character in a String	Go	Easy				54.5%
0537	Complex Number Multiplication	Go	Medium				68.5%
0541	Reverse String II	Go	Easy				49.7%
0557	Reverse Words in a String III	Go	Easy				73.1%
0583	Delete Operation for Two Strings	Go	Medium				52.1%
0609	Find Duplicate File in System	Go	Medium				63.0%
0632	Smallest Range Covering Elements from K Lists	Go	Hard				55.2%
0647	Palindromic Substrings	Go	Medium				63.0%
0696	Count Binary Substrings	Go	Easy				61.5%
0709	To Lower Case	Go	Easy				80.6%
0767	Reorganize String	Go	Medium	O(n log n)	O(log n)	❤️	50.6%
0816	Ambiguous Coordinates	Go	Medium				55.6%
0819	Most Common Word	Go	Easy				45.4%
0842	Split Array into Fibonacci Sequence	Go	Medium	O(n^2)	O(1)	❤️	37.1%
0856	Score of Parentheses	Go	Medium	O(n)	O(n)		65.0%

0890	Find and Replace Pattern	Go	Medium				75.5%
0916	Word Subsets	Go	Medium				52.7%
0925	Long Pressed Name	Go	Easy	O(n)	O(1)		36.4%
0966	Vowel Spellchecker	Go	Medium				51.8%
1003	Check If Word Is Valid After Substitutions	Go	Medium	O(n)	O(1)		57.0%
1108	Defanging an IP Address	Go	Easy				88.5%
1170	Compare Strings by Frequency of the Smallest Character	Go	Medium				60.5%
1189	Maximum Number of Balloons	Go	Easy				62.0%
1221	Split a String in Balanced Strings	Go	Easy				84.4%
1234	Replace the Substring for Balanced String	Go	Medium				34.8%
1249	Minimum Remove to Make Valid Parentheses	Go	Medium				64.4%
1268	Search Suggestions System	Go	Medium				65.5%
1332	Remove Palindromic Subsequences	Go	Easy				68.6%
1455	Check If a Word Occurs As a Prefix of Any Word in a Sentence	Go	Easy				64.8%
1461	Check If a String Contains All Binary Codes of Size K	Go	Medium				54.2%
1573	Number of Ways to Split a String	Go	Medium				31.3%
1614	Maximum Nesting Depth of the Parentheses	Go	Easy				82.6%
1624	Largest Substring Between Two Equal Characters	Go	Easy				58.6%
1653	Minimum Deletions to Make String Balanced	Go	Medium				52.4%
1662	Check If Two String Arrays are Equivalent	Go	Easy				82.1%
1668	Maximum Repeating Substring	Go	Easy				38.7%
1678	Goal Parser Interpretation	Go	Easy				84.9%
1684	Count the Number of Consistent Strings	Go	Easy				81.6%
1694	Reformat Phone Number	Go	Easy				64.7%
1704	Determine if String Halves Are Alike	Go	Easy				78.6%
1736	Latest Time by Replacing Hidden Digits	Go	Easy				41.4%

Two Pointers

✔	19	Remove Nth Node From End of List	34.0%	Medium	
✔	26	Remove Duplicates from Sorted Array	41.8%	Easy	
✔	27	Remove Element	45.5%	Easy	
✔	28	Implement strStr()	32.8%	Easy	
✔	30	Substring with Concatenation of All Words	24.1%	Hard	
✔	42	Trapping Rain Water	44.5%	Hard	
✔	61	Rotate List	27.9%	Medium	
✔	75	Sort Colors	43.2%	Medium	
✔	76	Minimum Window Substring	31.8%	Hard	
✔	80	Remove Duplicates from Sorted Array II	41.2%	Medium	
✔	86	Partition List	38.3%	Medium	
✔	88	Merge Sorted Array	36.8%	Easy	
✔	125	Valid Palindrome	32.2%	Easy	
✔	141	Linked List Cycle	38.0%	Easy	
✔	142	Linked List Cycle II	33.3%	Medium	
	159	Longest Substring with At Most Two Distinct Characters 🔒	47.7%	Hard	
✔	167	Two Sum II - Input array is sorted	51.3%	Easy	
✔	209	Minimum Size Subarray Sum	35.6%	Medium	
✔	234	Palindrome Linked List	36.9%	Easy	
	259	3Sum Smaller 🔒	45.5%	Medium	
✔	283	Move Zeroes	55.1%	Easy	
✔	287	Find the Duplicate Number	50.9%	Medium	
✔	344	Reverse String	64.0%	Easy	
✔	345	Reverse Vowels of a String	42.1%	Easy	
✔	349	Intersection of Two Arrays	56.5%	Easy	
✔	350	Intersection of Two Arrays II	48.9%	Easy	
	360	Sort Transformed Array 🔒	47.3%	Medium	
✔	424	Longest Repeating Character Replacement	44.7%	Medium	
✔	457	Circular Array Loop	28.1%	Medium	
	487	Max Consecutive Ones II 🔒	47.2%	Medium	
✔	524	Longest Word in Dictionary through Deleting	46.6%	Medium	
✔	532	K-diff Pairs in an Array	30.3%	Easy	
✔	567	Permutation in String	39.2%	Medium	
✔	632	Smallest Range Covering Elements from K Lists	48.9%	Hard	
✔	713	Subarray Product Less Than K	37.4%	Medium	
	723	Candy Crush 🔒	64.3%	Medium	
✔	763	Partition Labels	72.1%	Medium	
✔	826	Most Profit Assigning Work	36.6%	Medium	
✔	828	Unique Letter String	41.2%	Hard	
✔	838	Push Dominoes	44.9%	Medium	
✔	844	Backspace String Compare	46.6%	Easy	
✔	845	Longest Mountain in Array	34.9%	Medium	
✔	881	Boats to Save People	44.5%	Medium	
✔	904	Fruit Into Baskets	41.8%	Medium	

✔	923	3Sum With Multiplicity	34.5%	Medium
✔	925	Long Pressed Name	44.6%	Easy
✔	930	Binary Subarrays With Sum	39.3%	Medium
✔	977	Squares of a Sorted Array	71.7%	Easy
✔	986	Interval List Intersections	63.8%	Medium
✔	992	Subarrays with K Different Integers	45.2%	Hard
✔	1004	Max Consecutive Ones III	54.6%	Medium
✔	1093	Statistics from a Large Sample	44.0%	Medium

Copyright © 2019 LeetCode

Help Center | Jobs | Bug Bounty | Terms | Privacy Policy

- 双指针滑动窗口的经典写法。右指针不断往右移，移动到不能往右移动为止(具体条件根据题目而定)。当右指针到最右边以后，开始挪动左指针，释放窗口左边界。第 3 题，第 76 题，第 209 题，第 424 题，第 438 题，第 567 题，第 713 题，第 763 题，第 845 题，第 881 题，第 904 题，第 978 题，第 992 题，第 1004 题，第 1040 题，第 1052 题。

```
left, right := 0, -1

for left < len(s) {
  if right+1 < len(s) && freq[s[right+1]-'a'] == 0 {
    freq[s[right+1]-'a']++
    right++
  } else {
    freq[s[left]-'a']--
    left++
  }
  result = max(result, right-left+1)
}
```

- 快慢指针可以查找重复数字，时间复杂度 O(n)，第 287 题。
- 替换字母以后，相同字母能出现连续最长的长度。第 424 题。
- SUM 问题集。第 1 题，第 15 题，第 16 题，第 18 题，第 167 题，第 923 题，第 1074 题。

No.	Title	Solution	Difficulty	TimeComplexity	SpaceComplexity	Favorite	Acceptance
0003	Longest Substring Without Repeating Characters	Go	Medium	O(n)	O(1)	❤️	31.8%
0011	Container With Most Water	Go	Medium	O(n)	O(1)		53.0%
0015	3Sum	Go	Medium	O(n^2)	O(n)	❤️	28.7%
0016	3Sum Closest	Go	Medium	O(n^2)	O(1)	❤️	46.4%
0018	4Sum	Go	Medium	O(n^3)	O(n^2)	❤️	35.6%
0019	Remove Nth Node From End of List	Go	Medium	O(n)	O(1)		36.4%
0026	Remove Duplicates from Sorted Array	Go	Easy	O(n)	O(1)		47.2%
0027	Remove Element	Go	Easy	O(n)	O(1)		49.8%
0028	Implement strStr()	Go	Easy	O(n)	O(1)		35.6%
0030	Substring with Concatenation of All Words	Go	Hard	O(n)	O(n)	❤️	26.7%
0042	Trapping Rain Water	Go	Hard	O(n)	O(1)	❤️	52.3%

0061	Rotate List	Go	Medium	O(n)	O(1)		32.3%
0075	Sort Colors	Go	Medium	O(n)	O(1)	❤️	50.6%
0076	Minimum Window Substring	Go	Hard	O(n)	O(n)	❤️	36.6%
0080	Remove Duplicates from Sorted Array II	Go	Medium	O(n)	O(1		46.8%
0086	Partition List	Go	Medium	O(n)	O(1)	❤️	45.4%
0088	Merge Sorted Array	Go	Easy	O(n)	O(1)	❤️	41.2%
0125	Valid Palindrome	Go	Easy	O(n)	O(1)		39.0%
0141	Linked List Cycle	Go	Easy	O(n)	O(1)	❤️	43.5%
0142	Linked List Cycle II	Go	Medium	O(n)	O(1)	❤️	40.6%
0167	Two Sum II - Input array is sorted	Go	Easy	O(n)	O(1)		56.1%
0209	Minimum Size Subarray Sum	Go	Medium	O(n)	O(1)		40.4%
0234	Palindrome Linked List	Go	Easy	O(n)	O(1)		43.0%
0283	Move Zeroes	Go	Easy	O(n)	O(1)		58.9%
0287	Find the Duplicate Number	Go	Medium	O(n)	O(1)	❤️	58.2%
0344	Reverse String	Go	Easy	O(n)	O(1)		71.3%
0345	Reverse Vowels of a String	Go	Easy	O(n)	O(1)		45.6%
0349	Intersection of Two Arrays	Go	Easy	O(n)	O(n)		66.1%
0350	Intersection of Two Arrays II	Go	Easy	O(n)	O(n)		52.5%
0424	Longest Repeating Character Replacement	Go	Medium	O(n)	O(1)		48.9%
0457	Circular Array Loop	Go	Medium				30.5%
0524	Longest Word in Dictionary through Deleting	Go	Medium	O(n)	O(1)		50.3%
0532	K-diff Pairs in an Array	Go	Medium	O(n)	O(n)		36.0%
0567	Permutation in String	Go	Medium	O(n)	O(1)	❤️	44.6%
0632	Smallest Range Covering Elements from K Lists	Go	Hard				55.2%
0713	Subarray Product Less Than K	Go	Medium	O(n)	O(1)		40.8%
0763	Partition Labels	Go	Medium	O(n)	O(1)	❤️	78.2%
0826	Most Profit Assigning Work	Go	Medium	O(n log n)	O(n)		39.5%
0828	Count Unique Characters of All Substrings of a Given String	Go	Hard	O(n)	O(1)	❤️	46.7%
0838	Push Dominoes	Go	Medium	O(n)	O(n)		50.4%
0844	Backspace String Compare	Go	Easy	O(n)	O(n)		47.2%
0845	Longest Mountain in Array	Go	Medium	O(n)	O(1)		38.8%
0881	Boats to Save People	Go	Medium	O(n log n)	O(1)		49.2%
0904	Fruit Into Baskets	Go	Medium	O(n log n)	O(1)		43.2%
0923	3Sum With Multiplicity	Go	Medium	O(n^2)	O(n)		41.0%
0925	Long Pressed Name	Go	Easy	O(n)	O(1)		36.4%
0930	Binary Subarrays With Sum	Go	Medium	O(n)	O(n)	❤️	45.6%
0977	Squares of a Sorted Array	Go	Easy	O(n)	O(1)		71.6%
0986	Interval List Intersections	Go	Medium	O(n)	O(1)		68.9%
0992	Subarrays with K Different Integers	Go	Hard	O(n)	O(n)	❤️	51.5%
1004	Max Consecutive Ones III	Go	Medium	O(n)	O(1)		61.3%
1093	Statistics from a Large Sample	Go	Medium	O(n)	O(1)		47.8%
1234	Replace the Substring for Balanced String	Go	Medium				34.8%
1658	Minimum Operations to Reduce X to Zero	Go	Medium				33.3%

1695	Maximum Erasure Value	Go	Medium				52.5%

Linked List

LeetCode Explore Problems Mock New Contest Articles Discuss Store Premium

Linked List

Subscribe to see which companies asked this question

You have solved **30 / 35** problems.

Show problem tags

	#	Title	Acceptance	Difficulty	Frequency
✔	2	Add Two Numbers	31.3%	Medium	
✔	19	Remove Nth Node From End of List	34.4%	Medium	
✔	21	Merge Two Sorted Lists	47.8%	Easy	
✔	23	Merge k Sorted Lists	34.9%	Hard	
✔	24	Swap Nodes in Pairs	45.2%	Medium	
✔	25	Reverse Nodes in k-Group	36.9%	Hard	
✔	61	Rotate List	27.4%	Medium	
✔	82	Remove Duplicates from Sorted List II	33.3%	Medium	
✔	83	Remove Duplicates from Sorted List	42.8%	Easy	
✔	86	Partition List	37.6%	Medium	
✔	92	Reverse Linked List II	35.2%	Medium	
✔	109	Convert Sorted List to Binary Search Tree	41.3%	Medium	
✔	138	Copy List with Random Pointer	27.6%	Medium	
✔	141	Linked List Cycle	37.3%	Easy	
✔	142	Linked List Cycle II	32.5%	Medium	
✔	143	Reorder List	31.2%	Medium	
✔	147	Insertion Sort List	37.6%	Medium	
✔	148	Sort List	35.9%	Medium	
✔	160	Intersection of Two Linked Lists	34.3%	Easy	
✔	203	Remove Linked List Elements	36.0%	Easy	
✔	206	Reverse Linked List	55.3%	Easy	
✔	234	Palindrome Linked List	36.4%	Easy	
✔	237	Delete Node in a Linked List	54.1%	Easy	
✔	328	Odd Even Linked List	49.6%	Medium	
	369	Plus One Linked List 🔒	56.4%	Medium	
	379	Design Phone Directory 🔒	41.7%	Medium	
✔	445	Add Two Numbers II	50.4%	Medium	
✔	725	Split Linked List in Parts	49.2%	Medium	
	426	Convert Binary Search Tree to Sorted Doubly Linked List 🔒	52.6%	Medium	
	430	Flatten a Multilevel Doubly Linked List	42.7%	Medium	
✔	817	Linked List Components	54.7%	Medium	
✔	707	Design Linked List	21.6%	Easy	

	708	Insert into a Cyclic Sorted List	29.3%	Medium
✔	876	Middle of the Linked List	64.4%	Easy
✔	1019	Next Greater Node In Linked List	56.5%	Medium

Copyright © 2019 LeetCode

- 巧妙的构造虚拟头结点。可以使遍历处理逻辑更加统一。
- 灵活使用递归。构造递归条件，使用递归可以巧妙的解题。不过需要注意有些题目不能使用递归，因为递归深度太深会导致超时和栈溢出。
- 链表区间逆序。第 92 题。
- 链表寻找中间节点。第 876 题。链表寻找倒数第 n 个节点。第 19 题。只需要一次遍历就可以得到答案。
- 合并 K 个有序链表。第 21 题，第 23 题。
- 链表归类。第 86 题，第 328 题。
- 链表排序，时间复杂度要求 O(n * log n)，空间复杂度 O(1)。只有一种做法，归并排序，至顶向下归并。第 148 题。
- 判断链表是否存在环，如果有环，输出环的交叉点的下标；判断 2 个链表是否有交叉点，如果有交叉点，输出交叉点。第 141 题，第 142 题，第 160 题。

No.	Title	Solution	Difficulty	TimeComplexity	SpaceComplexity	Favorite	Acceptance
0002	Add Two Numbers	Go	Medium	O(n)	O(1)		36.2%
0019	Remove Nth Node From End of List	Go	Medium	O(n)	O(1)		36.4%
0021	Merge Two Sorted Lists	Go	Easy	O(log n)	O(1)		57.0%
0023	Merge k Sorted Lists	Go	Hard	O(log n)	O(1)	❤️	43.7%
0024	Swap Nodes in Pairs	Go	Medium	O(n)	O(1)		54.2%
0025	Reverse Nodes in k-Group	Go	Hard	O(log n)	O(1)	❤️	46.2%
0061	Rotate List	Go	Medium	O(n)	O(1)		32.3%
0082	Remove Duplicates from Sorted List II	Go	Medium	O(n)	O(1)		40.0%
0083	Remove Duplicates from Sorted List	Go	Easy	O(n)	O(1)		47.0%
0086	Partition List	Go	Medium	O(n)	O(1)	❤️	45.4%
0092	Reverse Linked List II	Go	Medium	O(n)	O(1)	❤️	41.2%
0109	Convert Sorted List to Binary Search Tree	Go	Medium	O(log n)	O(n)		52.4%
0138	Copy List with Random Pointer	Go	Medium				42.5%
0141	Linked List Cycle	Go	Easy	O(n)	O(1)	❤️	43.5%
0142	Linked List Cycle II	Go	Medium	O(n)	O(1)	❤️	40.6%
0143	Reorder List	Go	Medium	O(n)	O(1)	❤️	42.0%
0147	Insertion Sort List	Go	Medium	O(n)	O(1)	❤️	45.2%
0148	Sort List	Go	Medium	O(n log n)	O(n)	❤️	47.5%
0160	Intersection of Two Linked Lists	Go	Easy	O(n)	O(1)	❤️	45.6%
0203	Remove Linked List Elements	Go	Easy	O(n)	O(1)		40.0%
0206	Reverse Linked List	Go	Easy	O(n)	O(1)		66.5%
0234	Palindrome Linked List	Go	Easy	O(n)	O(1)		43.0%
0237	Delete Node in a Linked List	Go	Easy	O(n)	O(1)		68.4%
0328	Odd Even Linked List	Go	Medium	O(n)	O(1)		57.6%
0445	Add Two Numbers II	Go	Medium	O(n)	O(n)		57.0%
0707	Design Linked List	Go	Medium	O(n)	O(1)		26.3%
0725	Split Linked List in Parts	Go	Medium	O(n)	O(1)		53.5%
0817	Linked List Components	Go	Medium	O(n)	O(1)		57.8%
0876	Middle of the Linked List	Go	Easy	O(n)	O(1)	❤️	69.4%
1019	Next Greater Node In Linked List	Go	Medium	O(n)	O(1)		58.4%
1171	Remove Zero Sum Consecutive Nodes from Linked List	Go	Medium				41.5%
1290	Convert Binary Number in a Linked List to Integer	Go	Easy				81.7%
1669	Merge In Between Linked Lists	Go	Medium				75.2%
1670	Design Front Middle Back Queue	Go	Medium				54.3%
1721	Swapping Nodes in a Linked List	Go	Medium				66.5%

Stack

Stack

Subscribe to see which companies asked this question

Subscribe to see which companies asked this question

You have solved **37 / 49** problems.

☐ **Show problem tags**

	#	Title	Acceptance	Difficulty	Frequency 🔒
✔	20	Valid Parentheses	36.7%	Easy	
✔	42	Trapping Rain Water	43.6%	Hard	
✔	71	Simplify Path	29.0%	Medium	
✔	84	Largest Rectangle in Histogram	31.4%	Hard	
	85	Maximal Rectangle	33.6%	Hard	
✔	94	Binary Tree Inorder Traversal	57.1%	Medium	
✔	103	Binary Tree Zigzag Level Order Traversal	42.1%	Medium	
✔	144	Binary Tree Preorder Traversal	51.7%	Medium	
✔	145	Binary Tree Postorder Traversal	48.9%	Hard	
✔	150	Evaluate Reverse Polish Notation	32.5%	Medium	
✔	155	Min Stack	37.5%	Easy	
✔	173	Binary Search Tree Iterator	49.1%	Medium	
✔	224	Basic Calculator	33.0%	Hard	
✔	225	Implement Stack using Queues	39.6%	Easy	
✔	232	Implement Queue using Stacks	43.7%	Easy	
	255	Verify Preorder Sequence in Binary Search Tree 🔒	43.7%	Medium	
	272	Closest Binary Search Tree Value II 🔒	45.4%	Hard	
	316	Remove Duplicate Letters	32.8%	Hard	
✔	331	Verify Preorder Serialization of a Binary Tree	38.8%	Medium	
	341	Flatten Nested List Iterator	48.2%	Medium	
	385	Mini Parser	31.9%	Medium	
✔	394	Decode String	45.2%	Medium	
✔	402	Remove K Digits	26.7%	Medium	
	439	Ternary Expression Parser 🔒	53.7%	Medium	
✔	456	132 Pattern	27.4%	Medium	
✔	496	Next Greater Element I	59.8%	Easy	
✔	503	Next Greater Element II	51.3%	Medium	
	591	Tag Validator	32.9%	Hard	
✔	636	Exclusive Time of Functions	48.8%	Medium	
✔	682	Baseball Game	61.2%	Easy	
✔	726	Number of Atoms	44.9%	Hard	
✔	735	Asteroid Collision	38.6%	Medium	
✔	739	Daily Temperatures	60.1%	Medium	
	770	Basic Calculator IV	45.7%	Hard	
	772	Basic Calculator III 🔒	43.4%	Hard	
✔	844	Backspace String Compare	46.1%	Easy	
✔	856	Score of Parentheses	56.5%	Medium	
✔	880	Decoded String at Index	23.0%	Medium	
✔	895	Maximum Frequency Stack	56.4%	Hard	
✔	901	Online Stock Span	49.4%	Medium	

✔	907	Sum of Subarray Minimums	27.6%	Medium
✔	921	Minimum Add to Make Parentheses Valid	70.3%	Medium
✔	946	Validate Stack Sequences	57.5%	Medium
?	975	Odd Even Jump	47.1%	Hard
✔	1003	Check If Word Is Valid After Substitutions	51.8%	Medium
	1063	Number of Valid Subarrays 🔒	74.2%	Hard
✔	1019	Next Greater Node In Linked List	56.5%	Medium
✔	1021	Remove Outermost Parentheses	75.7%	Easy
✔	1047	Remove All Adjacent Duplicates In String	63.7%	Easy

Copyright © 2019 LeetCode

- 括号匹配问题及类似问题。第 20 题，第 921 题，第 1021 题。
- 栈的基本 pop 和 push 操作。第 71 题，第 150 题，第 155 题，第 224 题，第 225 题，第 232 题，第 946 题，第 1047 题。
- 利用栈进行编码问题。第 394 题，第 682 题，第 856 题，第 880 题。
- **单调栈。利用栈维护一个单调递增或者递减的下标数组。**第 84 题，第 456 题，第 496 题，第 503 题，第 739 题，第 901 题，第 907 题，第 1019 题。

No.	Title	Solution	Difficulty	TimeComplexity	SpaceComplexity	Favorite	Acceptance
0020	Valid Parentheses	Go	Easy	O(log n)	O(1)		40.2%
0042	Trapping Rain Water	Go	Hard	O(n)	O(1)	❤️	52.3%
0071	Simplify Path	Go	Medium	O(n)	O(n)	❤️	35.3%
0084	Largest Rectangle in Histogram	Go	Hard	O(n)	O(n)	❤️	38.0%
0094	Binary Tree Inorder Traversal	Go	Easy	O(n)	O(1)		67.1%
0103	Binary Tree Zigzag Level Order Traversal	Go	Medium	O(n)	O(n)		51.0%
0144	Binary Tree Preorder Traversal	Go	Easy	O(n)	O(1)		58.6%
0145	Binary Tree Postorder Traversal	Go	Easy	O(n)	O(1)		59.0%
0150	Evaluate Reverse Polish Notation	Go	Medium	O(n)	O(1)		39.6%
0155	Min Stack	Go	Easy	O(n)	O(n)		47.3%
0173	Binary Search Tree Iterator	Go	Medium	O(n)	O(1)		61.5%
0224	Basic Calculator	Go	Hard	O(n)	O(n)		38.6%
0225	Implement Stack using Queues	Go	Easy	O(n)	O(n)		48.6%
0227	Basic Calculator II	Go	Medium				39.1%
0232	Implement Queue using Stacks	Go	Easy	O(n)	O(n)		53.4%
0331	Verify Preorder Serialization of a Binary Tree	Go	Medium	O(n)	O(1)		41.3%
0341	Flatten Nested List Iterator	Go	Medium				56.4%
0385	Mini Parser	Go	Medium				34.8%
0394	Decode String	Go	Medium	O(n)	O(n)		53.6%
0402	Remove K Digits	Go	Medium	O(n)	O(1)		28.8%
0456	132 Pattern	Go	Medium	O(n)	O(n)		30.7%
0496	Next Greater Element I	Go	Easy	O(n)	O(n)		66.4%
0503	Next Greater Element II	Go	Medium	O(n)	O(n)		59.3%

0636	Exclusive Time of Functions	Go	Medium	O(n)	O(n)		56.0%
0682	Baseball Game	Go	Easy	O(n)	O(n)		67.9%
0726	Number of Atoms	Go	Hard	O(n)	O(n)	❤️	51.1%
0735	Asteroid Collision	Go	Medium	O(n)	O(n)		43.5%
0739	Daily Temperatures	Go	Medium	O(n)	O(n)		65.3%
0844	Backspace String Compare	Go	Easy	O(n)	O(n)		47.2%
0856	Score of Parentheses	Go	Medium	O(n)	O(n)		65.0%
0880	Decoded String at Index	Go	Medium	O(n)	O(n)		28.2%
0895	Maximum Frequency Stack	Go	Hard	O(n)	O(n)		63.6%
0901	Online Stock Span	Go	Medium	O(n)	O(n)		61.8%
0907	Sum of Subarray Minimums	Go	Medium	O(n)	O(n)	❤️	32.9%
0921	Minimum Add to Make Parentheses Valid	Go	Medium	O(n)	O(n)		75.4%
0946	Validate Stack Sequences	Go	Medium	O(n)	O(n)		64.5%
1003	Check If Word Is Valid After Substitutions	Go	Medium	O(n)	O(1)		57.0%
1019	Next Greater Node In Linked List	Go	Medium	O(n)	O(1)		58.4%
1021	Remove Outermost Parentheses	Go	Easy	O(n)	O(1)		79.3%
1047	Remove All Adjacent Duplicates In String	Go	Easy	O(n)	O(1)		71.6%
1190	Reverse Substrings Between Each Pair of Parentheses	Go	Medium				64.6%
1209	Remove All Adjacent Duplicates in String II	Go	Medium				57.1%
1249	Minimum Remove to Make Valid Parentheses	Go	Medium				64.4%
1673	Find the Most Competitive Subsequence	Go	Medium				45.8%

Tree

No.	Title	Solution	Difficulty	TimeComplexity	SpaceComplexity	Favorite	Acceptance
0094	Binary Tree Inorder Traversal	Go	Easy	O(n)	O(1)		67.1%
0095	Unique Binary Search Trees II	Go	Medium				43.8%
0096	Unique Binary Search Trees	Go	Medium	O(n^2)	O(n)		55.2%
0098	Validate Binary Search Tree	Go	Medium	O(n)	O(1)		29.2%
0099	Recover Binary Search Tree	Go	Medium	O(n)	O(1)		43.4%
0100	Same Tree	Go	Easy	O(n)	O(1)		54.5%
0101	Symmetric Tree	Go	Easy	O(n)	O(1)		49.1%
0102	Binary Tree Level Order Traversal	Go	Medium	O(n)	O(1)		57.9%
0103	Binary Tree Zigzag Level Order Traversal	Go	Medium	O(n)	O(n)		51.0%
0104	Maximum Depth of Binary Tree	Go	Easy	O(n)	O(1)		69.0%
0105	Construct Binary Tree from Preorder and Inorder Traversal	Go	Medium				53.8%
0106	Construct Binary Tree from Inorder and Postorder Traversal	Go	Medium				51.0%
0107	Binary Tree Level Order Traversal II	Go	Medium	O(n)	O(1)		56.1%
0108	Convert Sorted Array to Binary Search Tree	Go	Easy	O(n)	O(1)		61.8%
0110	Balanced Binary Tree	Go	Easy	O(n)	O(1)		45.1%
0111	Minimum Depth of Binary Tree	Go	Easy	O(n)	O(1)		40.4%
0112	Path Sum	Go	Easy	O(n)	O(1)		43.1%

0113	Path Sum II	Go	Medium	O(n)	O(1)		50.3%
0114	Flatten Binary Tree to Linked List	Go	Medium	O(n)	O(1)		53.9%
0124	Binary Tree Maximum Path Sum	Go	Hard	O(n)	O(1)		36.0%
0129	Sum Root to Leaf Numbers	Go	Medium	O(n)	O(1)		52.1%
0144	Binary Tree Preorder Traversal	Go	Easy	O(n)	O(1)		58.6%
0145	Binary Tree Postorder Traversal	Go	Easy	O(n)	O(1)		59.0%
0173	Binary Search Tree Iterator	Go	Medium	O(n)	O(1)		61.5%
0199	Binary Tree Right Side View	Go	Medium	O(n)	O(1)		57.1%
0222	Count Complete Tree Nodes	Go	Medium	O(n)	O(1)		50.6%
0226	Invert Binary Tree	Go	Easy	O(n)	O(1)		68.0%
0230	Kth Smallest Element in a BST	Go	Medium	O(n)	O(1)		63.7%
0235	Lowest Common Ancestor of a Binary Search Tree	Go	Easy	O(n)	O(1)		52.8%
0236	Lowest Common Ancestor of a Binary Tree	Go	Medium	O(n)	O(1)		50.4%
0257	Binary Tree Paths	Go	Easy	O(n)	O(1)		54.9%
0297	Serialize and Deserialize Binary Tree	Go	Hard				51.0%
0337	House Robber III	Go	Medium				52.1%
0404	Sum of Left Leaves	Go	Easy	O(n)	O(1)		52.6%
0437	Path Sum III	Go	Medium	O(n)	O(1)		48.6%
0508	Most Frequent Subtree Sum	Go	Medium				59.9%
0513	Find Bottom Left Tree Value	Go	Medium				63.3%
0515	Find Largest Value in Each Tree Row	Go	Medium	O(n)	O(n)		62.8%
0530	Minimum Absolute Difference in BST	Go	Easy				55.3%
0538	Convert BST to Greater Tree	Go	Medium				60.6%
0563	Binary Tree Tilt	Go	Easy				53.9%
0572	Subtree of Another Tree	Go	Easy				44.6%
0589	N-ary Tree Preorder Traversal	Go	Easy				74.6%
0623	Add One Row to Tree	Go	Medium				53.2%
0637	Average of Levels in Binary Tree	Go	Easy	O(n)	O(n)		66.6%
0653	Two Sum IV - Input is a BST	Go	Easy				56.6%
0662	Maximum Width of Binary Tree	Go	Medium				39.6%
0669	Trim a Binary Search Tree	Go	Medium				64.4%
0684	Redundant Connection	Go	Medium				59.6%
0685	Redundant Connection II	Go	Hard				33.2%
0783	Minimum Distance Between BST Nodes	Go	Easy				54.6%
0834	Sum of Distances in Tree	Go	Hard				47.3%
0863	All Nodes Distance K in Binary Tree	Go	Medium				58.7%
0872	Leaf-Similar Trees	Go	Easy				64.5%
0897	Increasing Order Search Tree	Go	Easy				74.9%
0938	Range Sum of BST	Go	Easy				83.5%
0968	Binary Tree Cameras	Go	Hard				40.6%
0971	Flip Binary Tree To Match Preorder Traversal	Go	Medium				49.9%
0979	Distribute Coins in Binary Tree	Go	Medium				70.2%
0987	Vertical Order Traversal of a Binary Tree	Go	Hard				39.3%
0993	Cousins in Binary Tree	Go	Easy	O(n)	O(1)		52.4%

1026	Maximum Difference Between Node and Ancestor	Go	Medium				70.1%
1028	Recover a Tree From Preorder Traversal	Go	Hard				71.3%
1038	Binary Search Tree to Greater Sum Tree	Go	Medium				83.1%
1110	Delete Nodes And Return Forest	Go	Medium				68.2%
1123	Lowest Common Ancestor of Deepest Leaves	Go	Medium				68.4%
1145	Binary Tree Coloring Game	Go	Medium				51.2%
1302	Deepest Leaves Sum	Go	Medium				85.4%
1305	All Elements in Two Binary Search Trees	Go	Medium				77.9%
1600	Throne Inheritance	Go	Medium				61.4%

Dynamic Programming

No.	Title	Solution	Difficulty	TimeComplexity	SpaceComplexity	Favorite	Acceptance
0005	Longest Palindromic Substring	Go	Medium				30.9%
0032	Longest Valid Parentheses	Go	Hard				30.1%
0042	Trapping Rain Water	Go	Hard				52.3%
0053	Maximum Subarray	Go	Easy	O(n)	O(n)		48.1%
0062	Unique Paths	Go	Medium	O(n^2)	O(n^2)		56.9%
0063	Unique Paths II	Go	Medium	O(n^2)	O(n^2)		36.0%
0064	Minimum Path Sum	Go	Medium	O(n^2)	O(n^2)		57.0%
0070	Climbing Stairs	Go	Easy	O(n)	O(n)		49.0%
0091	Decode Ways	Go	Medium	O(n)	O(n)		27.4%
0095	Unique Binary Search Trees II	Go	Medium				43.8%
0096	Unique Binary Search Trees	Go	Medium	O(n)	O(n)		55.2%
0097	Interleaving String	Go	Medium				33.6%
0115	Distinct Subsequences	Go	Hard				40.4%
0120	Triangle	Go	Medium	O(n^2)	O(n)		47.3%
0121	Best Time to Buy and Sell Stock	Go	Easy	O(n)	O(1)		52.1%
0131	Palindrome Partitioning	Go	Medium				53.7%
0152	Maximum Product Subarray	Go	Medium	O(n)	O(1)		33.1%
0174	Dungeon Game	Go	Hard				33.7%
0198	House Robber	Go	Medium	O(n)	O(n)		43.6%
0213	House Robber II	Go	Medium	O(n)	O(n)		37.9%
0264	Ugly Number II	Go	Medium				43.3%
0279	Perfect Squares	Go	Medium				49.7%
0300	Longest Increasing Subsequence	Go	Medium	O(n log n)	O(n)		45.3%
0303	Range Sum Query - Immutable	Go	Easy				49.4%
0304	Range Sum Query 2D - Immutable	Go	Medium				43.8%
0309	Best Time to Buy and Sell Stock with Cooldown	Go	Medium	O(n)	O(n)		48.8%
0322	Coin Change	Go	Medium	O(n)	O(n)		38.1%
0337	House Robber III	Go	Medium				52.1%
0338	Counting Bits	Go	Easy	O(n)	O(n)		71.0%
0343	Integer Break	Go	Medium	O(n^2)	O(n)		51.5%

0354	Russian Doll Envelopes	Go	Hard				38.3%
0357	Count Numbers with Unique Digits	Go	Medium	O(1)	O(1)		49.3%
0368	Largest Divisible Subset	Go	Medium				38.6%
0376	Wiggle Subsequence	Go	Medium				42.6%
0377	Combination Sum IV	Go	Medium				47.3%
0392	Is Subsequence	Go	Easy	O(n)	O(1)		49.8%
0410	Split Array Largest Sum	Go	Hard				47.2%
0413	Arithmetic Slices	Go	Medium				60.2%
0416	Partition Equal Subset Sum	Go	Medium	O(n^2)	O(n)		45.2%
0474	Ones and Zeroes	Go	Medium				43.5%
0494	Target Sum	Go	Medium				45.6%
0523	Continuous Subarray Sum	Go	Medium				25.2%
0638	Shopping Offers	Go	Medium				53.7%
0647	Palindromic Substrings	Go	Medium				63.0%
0714	Best Time to Buy and Sell Stock with Transaction Fee	Go	Medium	O(n)	O(1)		58.5%
0718	Maximum Length of Repeated Subarray	Go	Medium				50.9%
0746	Min Cost Climbing Stairs	Go	Easy	O(n)	O(1)		53.5%
0838	Push Dominoes	Go	Medium	O(n)	O(n)		50.4%
0877	Stone Game	Go	Medium				67.6%
0887	Super Egg Drop	Go	Hard				27.0%
0898	Bitwise ORs of Subarrays	Go	Medium				35.0%
0920	Number of Music Playlists	Go	Hard				48.4%
0968	Binary Tree Cameras	Go	Hard				40.6%
0978	Longest Turbulent Subarray	Go	Medium				46.7%
1025	Divisor Game	Go	Easy	O(1)	O(1)		66.2%
1048	Longest String Chain	Go	Medium				56.2%
1049	Last Stone Weight II	Go	Medium				47.3%
1074	Number of Submatrices That Sum to Target	Go	Hard				65.2%
1105	Filling Bookcase Shelves	Go	Medium				57.6%
1143	Longest Common Subsequence	Go	Medium				58.8%
1235	Maximum Profit in Job Scheduling	Go	Hard				48.2%
1423	Maximum Points You Can Obtain from Cards	Go	Medium				48.5%
1463	Cherry Pickup II	Go	Hard				68.6%
1641	Count Sorted Vowel Strings	Go	Medium				75.0%
1654	Minimum Jumps to Reach Home	Go	Medium				24.6%
1655	Distribute Repeating Integers	Go	Hard				40.2%
1659	Maximize Grid Happiness	Go	Hard				35.9%
1664	Ways to Make a Fair Array	Go	Medium				62.0%
1690	Stone Game VII	Go	Medium				58.6%
1691	Maximum Height by Stacking Cuboids	Go	Hard				51.0%

Backtracking

LeetCode Explore Problems Mock New Contest Articles Discuss Store Premium

Backtracking

Subscribe to see which companies asked this question

You have solved **30 / 45** problems.

☐ **Show problem tags**

	#	Title	Acceptance	Difficulty	Frequency 🔒
	10	Regular Expression Matching	25.4%	Hard	
✔	17	Letter Combinations of a Phone Number	42.0%	Medium	
✔	22	Generate Parentheses	55.5%	Medium	
✔	37	Sudoku Solver	37.5%	Hard	
✔	39	Combination Sum	49.2%	Medium	
✔	40	Combination Sum II	42.1%	Medium	
	44	Wildcard Matching	23.0%	Hard	
✔	46	Permutations	55.8%	Medium	
✔	47	Permutations II	41.0%	Medium	
✔	51	N-Queens	39.9%	Hard	
✔	52	N-Queens II	52.4%	Hard	
✔	60	Permutation Sequence	33.4%	Medium	
✔	77	Combinations	48.2%	Medium	
✔	78	Subsets	53.5%	Medium	
✔	79	Word Search	31.6%	Medium	
✔	89	Gray Code	46.0%	Medium	
✔	90	Subsets II	42.8%	Medium	
✔	93	Restore IP Addresses	31.8%	Medium	
✔	126	Word Ladder II	18.0%	Hard	
✔	131	Palindrome Partitioning	41.4%	Medium	
	140	Word Break II	27.6%	Hard	
✔	211	Add and Search Word - Data structure design	30.7%	Medium	
✔	212	Word Search II	29.0%	Hard	
✔	216	Combination Sum III	51.9%	Medium	
	254	Factor Combinations 🔒	44.5%	Medium	
	267	Palindrome Permutation II 🔒	33.9%	Medium	
	291	Word Pattern II 🔒	41.1%	Hard	
	294	Flip Game II 🔒	48.4%	Medium	
✔	306	Additive Number	28.4%	Medium	
	320	Generalized Abbreviation 🔒	48.9%	Medium	
	351	Android Unlock Patterns 🔒	46.1%	Medium	
✔	357	Count Numbers with Unique Digits	47.1%	Medium	
✔	401	Binary Watch	45.4%	Easy	
	411	Minimum Unique Word Abbreviation 🔒	35.1%	Hard	
	425	Word Squares 🔒	44.5%	Hard	
✔	526	Beautiful Arrangement	54.8%	Medium	
	691	Stickers to Spell Word	38.5%	Hard	

✔	784	Letter Case Permutation	57.5%	Easy
✔	842	Split Array into Fibonacci Sequence	34.9%	Medium
✔	980	Unique Paths III	71.2%	Hard
✔	996	Number of Squareful Arrays	47.6%	Hard
	1066	Campus Bikes II 🔒	43.6%	Medium
	1087	Brace Expansion 🔒	58.1%	Medium
	1088	Confusing Number II 🔒	34.3%	Hard
✔	1079	Letter Tile Possibilities	76.5%	Medium

Copyright © 2019 LeetCode

- 排列问题 Permutations。第 46 题，第 47 题。第 60 题，第 526 题，第 996 题。
- 组合问题 Combination。第 39 题，第 40 题，第 77 题，第 216 题。
- 排列和组合杂交问题。第 1079 题。
- N 皇后终极解法(二进制解法)。第 51 题，第 52 题。
- 数独问题。第 37 题。
- 四个方向搜索。第 79 题，第 212 题，第 980 题。
- 子集合问题。第 78 题，第 90 题。
- Trie。第 208 题，第 211 题。
- BFS 优化。第 126 题，第 127 题。
- DFS 模板。(只是一个例子，不对应任何题)

```go
func combinationSum2(candidates []int, target int) [][]int {
  if len(candidates) == 0 {
    return [][]int{}
  }
  c, res := []int{}, [][]int{}
  sort.Ints(candidates)
  findcombinationSum2(candidates, target, 0, c, &res)
  return res
}

func findcombinationSum2(nums []int, target, index int, c []int, res *[][]int) {
  if target == 0 {
    b := make([]int, len(c))
    copy(b, c)
    *res = append(*res, b)
    return
  }
  for i := index; i < len(nums); i++ {
    if i > index && nums[i] == nums[i-1] { // 这里是去重的关键逻辑
      continue
    }
    if target >= nums[i] {
      c = append(c, nums[i])
      findcombinationSum2(nums, target-nums[i], i+1, c, res)
      c = c[:len(c)-1]
```

```
    }
  }
}
```

- BFS 模板。(只是一个例子，不对应任何题)

```
func updateMatrix_BFS(matrix [][]int) [][]int {
  res := make([][]int, len(matrix))
  if len(matrix) == 0 || len(matrix[0]) == 0 {
    return res
  }
  queue := make([][]int, 0)
  for i, _ := range matrix {
    res[i] = make([]int, len(matrix[0]))
    for j, _ := range res[i] {
      if matrix[i][j] == 0 {
        res[i][j] = -1
        queue = append(queue, []int{i, j})
      }
    }
  }
  level := 1
  for len(queue) > 0 {
    size := len(queue)
    for size > 0 {
      size -= 1
      node := queue[0]
      queue = queue[1:]
      i, j := node[0], node[1]
      for _, direction := range [][]int{{-1, 0}, {1, 0}, {0, 1}, {0, -1}} {
        x := i + direction[0]
        y := j + direction[1]
        if x < 0 || x >= len(matrix) || y < 0 || y >= len(matrix[0]) || res[x][y] < 0
|| res[x][y] > 0 {
          continue
        }
        res[x][y] = level
        queue = append(queue, []int{x, y})
      }
    }
    level++
  }
  for i, row := range res {
    for j, cell := range row {
      if cell == -1 {
        res[i][j] = 0
      }
    }
  }
```

```
    return res
}
```

No.	Title	Solution	Difficulty	TimeComplexity	SpaceComplexity	Favorite	Acceptance
0017	Letter Combinations of a Phone Number	Go	Medium	O(log n)	O(1)		50.5%
0022	Generate Parentheses	Go	Medium	O(log n)	O(1)		66.8%
0037	Sudoku Solver	Go	Hard	O(n^2)	O(n^2)	❤️	48.2%
0039	Combination Sum	Go	Medium	O(n log n)	O(n)		60.6%
0040	Combination Sum II	Go	Medium	O(n log n)	O(n)		50.9%
0046	Permutations	Go	Medium	O(n)	O(n)	❤️	68.0%
0047	Permutations II	Go	Medium	O(n^2)	O(n)	❤️	50.5%
0051	N-Queens	Go	Hard	O(n!)	O(n)	❤️	52.3%
0052	N-Queens II	Go	Hard	O(n!)	O(n)	❤️	62.7%
0060	Permutation Sequence	Go	Hard	O(n log n)	O(1)		40.0%
0077	Combinations	Go	Medium	O(n)	O(n)	❤️	58.9%
0078	Subsets	Go	Medium	O(n^2)	O(n)	❤️	66.6%
0079	Word Search	Go	Medium	O(n^2)	O(n^2)	❤️	37.7%
0089	Gray Code	Go	Medium	O(n)	O(1)		51.2%
0090	Subsets II	Go	Medium	O(n^2)	O(n)	❤️	49.7%
0093	Restore IP Addresses	Go	Medium	O(n)	O(n)	❤️	38.5%
0126	Word Ladder II	Go	Hard	O(n)	O(n^2)	❤️	24.2%
0131	Palindrome Partitioning	Go	Medium	O(n)	O(n^2)	❤️	53.7%
0211	Design Add and Search Words Data Structure	Go	Medium	O(n)	O(n)	❤️	41.3%
0212	Word Search II	Go	Hard	O(n^2)	O(n^2)	❤️	37.8%
0216	Combination Sum III	Go	Medium	O(n)	O(1)	❤️	61.3%
0306	Additive Number	Go	Medium	O(n^2)	O(1)	❤️	29.9%
0357	Count Numbers with Unique Digits	Go	Medium	O(1)	O(1)		49.3%
0401	Binary Watch	Go	Easy	O(1)	O(1)		48.9%
0526	Beautiful Arrangement	Go	Medium	O(n^2)	O(1)	❤️	62.4%
0784	Letter Case Permutation	Go	Medium	O(n)	O(n)		69.2%
0842	Split Array into Fibonacci Sequence	Go	Medium	O(n^2)	O(1)	❤️	37.1%
0980	Unique Paths III	Go	Hard	O(n log n)	O(n)		77.1%
0996	Number of Squareful Arrays	Go	Hard	O(n log n)	O(n)		48.6%
1079	Letter Tile Possibilities	Go	Medium	O(n^2)	O(1)	❤️	76.1%
1239	Maximum Length of a Concatenated String with Unique Characters	Go	Medium				50.5%
1641	Count Sorted Vowel Strings	Go	Medium				75.0%
1655	Distribute Repeating Integers	Go	Hard				40.2%
1659	Maximize Grid Happiness	Go	Hard				35.9%
1681	Minimum Incompatibility	Go	Hard				35.9%
1688	Count of Matches in Tournament	Go	Easy				81.8%

Depth First Search

No.	Title	Solution	Difficulty	TimeComplexity	SpaceComplexity	Favorite	Acceptance

0017	Letter Combinations of a Phone Number	Go	Medium				50.5%
0098	Validate Binary Search Tree	Go	Medium	O(n)	O(1)		29.2%
0099	Recover Binary Search Tree	Go	Medium	O(n)	O(1)		43.4%
0100	Same Tree	Go	Easy	O(n)	O(1)		54.5%
0101	Symmetric Tree	Go	Easy	O(n)	O(1)		49.1%
0104	Maximum Depth of Binary Tree	Go	Easy	O(n)	O(1)		69.0%
0105	Construct Binary Tree from Preorder and Inorder Traversal	Go	Medium				53.8%
0106	Construct Binary Tree from Inorder and Postorder Traversal	Go	Medium				51.0%
0108	Convert Sorted Array to Binary Search Tree	Go	Easy	O(n)	O(1)		61.8%
0109	Convert Sorted List to Binary Search Tree	Go	Medium	O(log n)	O(n)		52.4%
0110	Balanced Binary Tree	Go	Easy	O(n)	O(1)		45.1%
0111	Minimum Depth of Binary Tree	Go	Easy	O(n)	O(1)		40.4%
0112	Path Sum	Go	Easy	O(n)	O(1)		43.1%
0113	Path Sum II	Go	Medium	O(n)	O(1)		50.3%
0114	Flatten Binary Tree to Linked List	Go	Medium	O(n)	O(1)		53.9%
0124	Binary Tree Maximum Path Sum	Go	Hard	O(n)	O(1)		36.0%
0129	Sum Root to Leaf Numbers	Go	Medium	O(n)	O(1)		52.1%
0130	Surrounded Regions	Go	Medium				30.4%
0131	Palindrome Partitioning	Go	Medium				53.7%
0199	Binary Tree Right Side View	Go	Medium	O(n)	O(1)		57.1%
0200	Number of Islands	Go	Medium	O(n^2)	O(n^2)		50.3%
0207	Course Schedule	Go	Medium	O(n^2)	O(n^2)		44.5%
0210	Course Schedule II	Go	Medium	O(n^2)	O(n^2)		43.6%
0211	Design Add and Search Words Data Structure	Go	Medium				41.3%
0257	Binary Tree Paths	Go	Easy	O(n)	O(1)		54.9%
0329	Longest Increasing Path in a Matrix	Go	Hard				47.0%
0337	House Robber III	Go	Medium				52.1%
0394	Decode String	Go	Medium	O(n)	O(n)		53.6%
0417	Pacific Atlantic Water Flow	Go	Medium				44.9%
0473	Matchsticks to Square	Go	Medium				40.0%
0491	Increasing Subsequences	Go	Medium				48.3%
0494	Target Sum	Go	Medium				45.6%
0513	Find Bottom Left Tree Value	Go	Medium				63.3%
0515	Find Largest Value in Each Tree Row	Go	Medium	O(n)	O(n)		62.8%
0526	Beautiful Arrangement	Go	Medium				62.4%
0529	Minesweeper	Go	Medium				62.1%
0538	Convert BST to Greater Tree	Go	Medium				60.6%
0542	01 Matrix	Go	Medium	O(n)	O(1)		41.5%
0547	Number of Provinces	Go	Medium				61.3%
0563	Binary Tree Tilt	Go	Easy				53.9%
0638	Shopping Offers	Go	Medium				53.7%
0685	Redundant Connection II	Go	Hard				33.2%
0690	Employee Importance	Go	Easy				59.8%

0695	Max Area of Island	Go	Medium				66.6%
0721	Accounts Merge	Go	Medium				53.0%
0733	Flood Fill	Go	Easy				56.1%
0753	Cracking the Safe	Go	Hard				52.9%
0756	Pyramid Transition Matrix	Go	Medium				55.9%
0778	Swim in Rising Water	Go	Hard				55.3%
0783	Minimum Distance Between BST Nodes	Go	Easy				54.6%
0785	Is Graph Bipartite?	Go	Medium				49.0%
0802	Find Eventual Safe States	Go	Medium				50.5%
0834	Sum of Distances in Tree	Go	Hard				47.3%
0839	Similar String Groups	Go	Hard				42.6%
0841	Keys and Rooms	Go	Medium				66.8%
0851	Loud and Rich	Go	Medium				53.1%
0863	All Nodes Distance K in Binary Tree	Go	Medium				58.7%
0872	Leaf-Similar Trees	Go	Easy				64.5%
0897	Increasing Order Search Tree	Go	Easy				74.9%
0924	Minimize Malware Spread	Go	Hard				41.8%
0928	Minimize Malware Spread II	Go	Hard				41.5%
0938	Range Sum of BST	Go	Easy				83.5%
0947	Most Stones Removed with Same Row or Column	Go	Medium				55.9%
0959	Regions Cut By Slashes	Go	Medium				67.6%
0968	Binary Tree Cameras	Go	Hard				40.6%
0971	Flip Binary Tree To Match Preorder Traversal	Go	Medium				49.9%
0979	Distribute Coins in Binary Tree	Go	Medium				70.2%
0980	Unique Paths III	Go	Hard	O(n log n)	O(n)		77.1%
0987	Vertical Order Traversal of a Binary Tree	Go	Hard				39.3%
1020	Number of Enclaves	Go	Medium				59.7%
1026	Maximum Difference Between Node and Ancestor	Go	Medium				70.1%
1028	Recover a Tree From Preorder Traversal	Go	Hard				71.3%
1038	Binary Search Tree to Greater Sum Tree	Go	Medium				83.1%
1110	Delete Nodes And Return Forest	Go	Medium				68.2%
1123	Lowest Common Ancestor of Deepest Leaves	Go	Medium				68.4%
1145	Binary Tree Coloring Game	Go	Medium				51.2%
1203	Sort Items by Groups Respecting Dependencies	Go	Hard				48.3%
1254	Number of Closed Islands	Go	Medium				62.2%
1302	Deepest Leaves Sum	Go	Medium				85.4%
1306	Jump Game III	Go	Medium				61.8%
1319	Number of Operations to Make Network Connected	Go	Medium				55.6%
1631	Path With Minimum Effort	Go	Medium				50.3%

Breadth First Search

No.	Title	Solution	Difficulty	TimeComplexity	SpaceComplexity	Favorite	Acceptance
0101	Symmetric Tree	Go	Easy	O(n)	O(1)		49.1%
0102	Binary Tree Level Order Traversal	Go	Medium	O(n)	O(1)		57.9%
0103	Binary Tree Zigzag Level Order Traversal	Go	Medium	O(n)	O(n)		51.0%
0107	Binary Tree Level Order Traversal II	Go	Medium	O(n)	O(1)		56.1%
0111	Minimum Depth of Binary Tree	Go	Easy	O(n)	O(1)		40.4%
0126	Word Ladder II	Go	Hard	O(n)	O(n^2)	❤️	24.2%
0127	Word Ladder	Go	Hard	O(n)	O(n)		32.7%
0130	Surrounded Regions	Go	Medium				30.4%
0199	Binary Tree Right Side View	Go	Medium	O(n)	O(1)		57.1%
0200	Number of Islands	Go	Medium	O(n^2)	O(n^2)		50.3%
0207	Course Schedule	Go	Medium	O(n^2)	O(n^2)		44.5%
0210	Course Schedule II	Go	Medium	O(n^2)	O(n^2)		43.6%
0279	Perfect Squares	Go	Medium				49.7%
0417	Pacific Atlantic Water Flow	Go	Medium				44.9%
0513	Find Bottom Left Tree Value	Go	Medium				63.3%
0515	Find Largest Value in Each Tree Row	Go	Medium	O(n)	O(n)		62.8%
0529	Minesweeper	Go	Medium				62.1%
0542	01 Matrix	Go	Medium	O(n)	O(1)		41.5%
0690	Employee Importance	Go	Easy				59.8%
0785	Is Graph Bipartite?	Go	Medium				49.0%
0815	Bus Routes	Go	Hard				43.8%
0863	All Nodes Distance K in Binary Tree	Go	Medium				58.7%
0864	Shortest Path to Get All Keys	Go	Hard				42.8%
0987	Vertical Order Traversal of a Binary Tree	Go	Hard				39.3%
0993	Cousins in Binary Tree	Go	Easy	O(n)	O(1)		52.4%
1091	Shortest Path in Binary Matrix	Go	Medium				40.6%
1306	Jump Game III	Go	Medium				61.8%
1319	Number of Operations to Make Network Connected	Go	Medium				55.6%
1654	Minimum Jumps to Reach Home	Go	Medium				24.6%

Binary Search

- 二分搜索的经典写法。需要注意的三点：
  1. 循环退出条件，注意是 low <= high，而不是 low < high。
  2. mid 的取值，mid := low + (high-low)>>1
  3. low 和 high 的更新。low = mid + 1，high = mid - 1。

```go
func binarySearchMatrix(nums []int, target int) int {
  low, high := 0, len(nums)-1
  for low <= high {
    mid := low + (high-low)>>1
    if nums[mid] == target {
      return mid
```

```
    } else if nums[mid] > target {
      high = mid - 1
    } else {
      low = mid + 1
    }
  }
  return -1
}
```

- 二分搜索的变种写法。有 4 个基本变种:
  1. 查找第一个与 target 相等的元素，时间复杂度 O(logn)
  2. 查找最后一个与 target 相等的元素，时间复杂度 O(logn)
  3. 查找第一个大于等于 target 的元素，时间复杂度 O(logn)
  4. 查找最后一个小于等于 target 的元素，时间复杂度 O(logn)

```
// 二分查找第一个与 target 相等的元素，时间复杂度 O(logn)
func searchFirstEqualElement(nums []int, target int) int {
  low, high := 0, len(nums)-1
  for low <= high {
    mid := low + ((high - low) >> 1)
    if nums[mid] > target {
      high = mid - 1
    } else if nums[mid] < target {
      low = mid + 1
    } else {
      if (mid == 0) || (nums[mid-1] != target) { // 找到第一个与 target 相等的元素
        return mid
      }
      high = mid - 1
    }
  }
  return -1
}

// 二分查找最后一个与 target 相等的元素，时间复杂度 O(logn)
func searchLastEqualElement(nums []int, target int) int {
  low, high := 0, len(nums)-1
  for low <= high {
    mid := low + ((high - low) >> 1)
    if nums[mid] > target {
      high = mid - 1
    } else if nums[mid] < target {
      low = mid + 1
    } else {
      if (mid == len(nums)-1) || (nums[mid+1] != target) { // 找到最后一个与 target 相等的
元素
        return mid
      }
```

```go
        low = mid + 1
      }
    }
    return -1
}

// 二分查找第一个大于等于 target 的元素，时间复杂度 O(logn)
func searchFirstGreaterElement(nums []int, target int) int {
    low, high := 0, len(nums)-1
    for low <= high {
      mid := low + ((high - low) >> 1)
      if nums[mid] >= target {
        if (mid == 0) || (nums[mid-1] < target) { // 找到第一个大于等于 target 的元素
          return mid
        }
        high = mid - 1
      } else {
        low = mid + 1
      }
    }
    return -1
}

// 二分查找最后一个小于等于 target 的元素，时间复杂度 O(logn)
func searchLastLessElement(nums []int, target int) int {
    low, high := 0, len(nums)-1
    for low <= high {
      mid := low + ((high - low) >> 1)
      if nums[mid] <= target {
        if (mid == len(nums)-1) || (nums[mid+1] > target) { // 找到最后一个小于等于 target 的
元素
          return mid
        }
        low = mid + 1
      } else {
        high = mid - 1
      }
    }
    return -1
}
```

- 在基本有序的数组中用二分搜索。经典解法可以解，变种写法也可以写，常见的题型，在山峰数组中找山峰，在旋转有序数组中找分界点。第 33 题，第 81 题，第 153 题，第 154 题，第 162 题，第 852 题

```go
func peakIndexInMountainArray(A []int) int {
  low, high := 0, len(A)-1
  for low < high {
    mid := low + (high-low)>>1
    // 如果 mid 较大，则左侧存在峰值，high = m，如果 mid + 1 较大，则右侧存在峰值，low = mid + 1
    if A[mid] > A[mid+1] {
      high = mid
    } else {
      low = mid + 1
    }
  }
  return low
}
```

- max-min 最大值最小化问题。求在最小满足条件的情况下的最大值。第 410 题，第 875 题，第 1011 题，第 1283 题。

No.	Title	Solution	Difficulty	TimeComplexity	SpaceComplexity	Favorite	Acceptance
0004	Median of Two Sorted Arrays	Go	Hard				32.0%
0029	Divide Two Integers	Go	Medium				17.0%
0033	Search in Rotated Sorted Array	Go	Medium				36.4%
0034	Find First and Last Position of Element in Sorted Array	Go	Medium				38.1%
0035	Search Insert Position	Go	Easy				42.8%
0050	Pow(x, n)	Go	Medium	O(log n)	O(1)		31.3%
0069	Sqrt(x)	Go	Easy	O(log n)	O(1)		35.7%
0074	Search a 2D Matrix	Go	Medium				38.8%
0081	Search in Rotated Sorted Array II	Go	Medium				33.9%
0153	Find Minimum in Rotated Sorted Array	Go	Medium				46.7%
0154	Find Minimum in Rotated Sorted Array II	Go	Hard				42.2%
0162	Find Peak Element	Go	Medium				44.3%
0167	Two Sum II - Input array is sorted	Go	Easy	O(n)	O(1)		56.1%
0174	Dungeon Game	Go	Hard				33.7%
0209	Minimum Size Subarray Sum	Go	Medium	O(n)	O(1)		40.4%
0222	Count Complete Tree Nodes	Go	Medium	O(n)	O(1)		50.6%
0230	Kth Smallest Element in a BST	Go	Medium	O(n)	O(1)		63.7%
0240	Search a 2D Matrix II	Go	Medium				45.9%
0275	H-Index II	Go	Medium				36.5%
0278	First Bad Version	Go	Easy				38.4%
0287	Find the Duplicate Number	Go	Medium	O(n)	O(1)	❤️	58.2%
0300	Longest Increasing Subsequence	Go	Medium	O(n log n)	O(n)		45.3%
0315	Count of Smaller Numbers After Self	Go	Hard				42.3%
0327	Count of Range Sum	Go	Hard				36.3%
0349	Intersection of Two Arrays	Go	Easy	O(n)	O(n)		66.1%
0350	Intersection of Two Arrays II	Go	Easy	O(n)	O(n)		52.5%
0354	Russian Doll Envelopes	Go	Hard				38.3%

0367	Valid Perfect Square	Go	Easy				42.4%
0374	Guess Number Higher or Lower	Go	Easy				45.6%
0378	Kth Smallest Element in a Sorted Matrix	Go	Medium				57.1%
0392	Is Subsequence	Go	Easy	O(n)	O(1)		49.8%
0410	Split Array Largest Sum	Go	Hard				47.2%
0436	Find Right Interval	Go	Medium				48.7%
0441	Arranging Coins	Go	Easy				42.9%
0454	4Sum II	Go	Medium	O(n^2)	O(n)		54.9%
0475	Heaters	Go	Medium				33.9%
0483	Smallest Good Base	Go	Hard				36.7%
0493	Reverse Pairs	Go	Hard				27.7%
0497	Random Point in Non-overlapping Rectangles	Go	Medium				39.1%
0528	Random Pick with Weight	Go	Medium				45.0%
0658	Find K Closest Elements	Go	Medium				42.8%
0668	Kth Smallest Number in Multiplication Table	Go	Hard				48.3%
0704	Binary Search	Go	Easy				54.7%
0710	Random Pick with Blacklist	Go	Hard	O(n)	O(n)		33.2%
0718	Maximum Length of Repeated Subarray	Go	Medium				50.9%
0719	Find K-th Smallest Pair Distance	Go	Hard				32.9%
0744	Find Smallest Letter Greater Than Target	Go	Easy				45.7%
0778	Swim in Rising Water	Go	Hard				55.3%
0786	K-th Smallest Prime Fraction	Go	Hard				44.7%
0793	Preimage Size of Factorial Zeroes Function	Go	Hard				40.7%
0852	Peak Index in a Mountain Array	Go	Easy				71.6%
0862	Shortest Subarray with Sum at Least K	Go	Hard				25.3%
0875	Koko Eating Bananas	Go	Medium				53.7%
0878	Nth Magical Number	Go	Hard				29.1%
0887	Super Egg Drop	Go	Hard				27.0%
0911	Online Election	Go	Medium				51.7%
0927	Three Equal Parts	Go	Hard				34.8%
0981	Time Based Key-Value Store	Go	Medium				54.5%
1011	Capacity To Ship Packages Within D Days	Go	Medium				60.5%
1111	Maximum Nesting Depth of Two Valid Parentheses Strings	Go	Medium				72.8%
1157	Online Majority Element In Subarray	Go	Hard				41.2%
1170	Compare Strings by Frequency of the Smallest Character	Go	Medium				60.5%
1201	Ugly Number III	Go	Medium				26.7%
1235	Maximum Profit in Job Scheduling	Go	Hard				48.2%
1283	Find the Smallest Divisor Given a Threshold	Go	Medium				50.6%
1300	Sum of Mutated Array Closest to Target	Go	Medium				42.6%
1337	The K Weakest Rows in a Matrix	Go	Easy				72.0%
1482	Minimum Number of Days to Make m Bouquets	Go	Medium				52.1%
1631	Path With Minimum Effort	Go	Medium				50.3%

1642	Furthest Building You Can Reach	Go	Medium				43.4%
1649	Create Sorted Array through Instructions	Go	Hard				36.8%
1658	Minimum Operations to Reduce X to Zero	Go	Medium				33.3%

Math

No.	Title	Solution	Difficulty	TimeComplexity	SpaceComplexity	Favorite	Acceptance
0002	Add Two Numbers	Go	Medium	O(n)	O(1)		36.2%
0007	Reverse Integer	Go	Easy				26.0%
0008	String to Integer (atoi)	Go	Medium				15.9%
0009	Palindrome Number	Go	Easy				50.6%
0012	Integer to Roman	Go	Medium				57.5%
0013	Roman to Integer	Go	Easy				57.3%
0029	Divide Two Integers	Go	Medium				17.0%
0043	Multiply Strings	Go	Medium				35.5%
0050	Pow(x, n)	Go	Medium	O(log n)	O(1)		31.3%
0060	Permutation Sequence	Go	Hard	O(n log n)	O(1)		40.0%
0067	Add Binary	Go	Easy				47.8%
0069	Sqrt(x)	Go	Easy	O(log n)	O(1)		35.7%
0168	Excel Sheet Column Title	Go	Easy				32.2%
0171	Excel Sheet Column Number	Go	Easy				57.6%
0172	Factorial Trailing Zeroes	Go	Easy				39.2%
0202	Happy Number	Go	Easy	O(log n)	O(1)		51.7%
0204	Count Primes	Go	Easy				32.8%
0223	Rectangle Area	Go	Medium				38.7%
0224	Basic Calculator	Go	Hard	O(n)	O(n)		38.6%
0231	Power of Two	Go	Easy	O(1)	O(1)		43.8%
0258	Add Digits	Go	Easy				59.1%
0263	Ugly Number	Go	Easy	O(log n)	O(1)		41.7%
0264	Ugly Number II	Go	Medium				43.3%
0268	Missing Number	Go	Easy				56.1%
0279	Perfect Squares	Go	Medium				49.7%
0326	Power of Three	Go	Easy	O(1)	O(1)		42.5%
0343	Integer Break	Go	Medium	O(n^2)	O(n)		51.5%
0357	Count Numbers with Unique Digits	Go	Medium	O(1)	O(1)		49.3%
0367	Valid Perfect Square	Go	Easy				42.4%
0368	Largest Divisible Subset	Go	Medium				38.6%
0372	Super Pow	Go	Medium				36.9%
0397	Integer Replacement	Go	Medium				33.7%
0413	Arithmetic Slices	Go	Medium				60.2%
0423	Reconstruct Original Digits from English	Go	Medium				51.2%
0441	Arranging Coins	Go	Easy				42.9%
0447	Number of Boomerangs	Go	Medium				52.7%
0453	Minimum Moves to Equal Array Elements	Go	Easy				51.6%

0462	Minimum Moves to Equal Array Elements II	Go	Medium				55.6%
0478	Generate Random Point in a Circle	Go	Medium				39.1%
0483	Smallest Good Base	Go	Hard				36.7%
0507	Perfect Number	Go	Easy				36.6%
0523	Continuous Subarray Sum	Go	Medium				25.2%
0535	Encode and Decode TinyURL	Go	Medium				82.7%
0537	Complex Number Multiplication	Go	Medium				68.5%
0598	Range Addition II	Go	Easy				50.5%
0628	Maximum Product of Three Numbers	Go	Easy	O(n)	O(1)		46.7%
0633	Sum of Square Numbers	Go	Medium				32.9%
0645	Set Mismatch	Go	Easy				40.9%
0753	Cracking the Safe	Go	Hard				52.9%
0775	Global and Local Inversions	Go	Medium				46.1%
0781	Rabbits in Forest	Go	Medium				56.2%
0810	Chalkboard XOR Game	Go	Hard				50.9%
0812	Largest Triangle Area	Go	Easy				59.2%
0836	Rectangle Overlap	Go	Easy				42.7%
0869	Reordered Power of 2	Go	Medium				61.2%
0877	Stone Game	Go	Medium				67.6%
0878	Nth Magical Number	Go	Hard				29.1%
0885	Spiral Matrix III	Go	Medium	O(n^2)	O(1)		71.3%
0887	Super Egg Drop	Go	Hard				27.0%
0891	Sum of Subsequence Widths	Go	Hard	O(n log n)	O(1)		33.4%
0892	Surface Area of 3D Shapes	Go	Easy				60.2%
0910	Smallest Range II	Go	Medium				31.3%
0914	X of a Kind in a Deck of Cards	Go	Easy				33.7%
0927	Three Equal Parts	Go	Hard				34.8%
0942	DI String Match	Go	Easy	O(n)	O(1)		74.1%
0949	Largest Time for Given Digits	Go	Medium				36.0%
0952	Largest Component Size by Common Factor	Go	Hard				36.5%
0970	Powerful Integers	Go	Medium				43.4%
0976	Largest Perimeter Triangle	Go	Easy	O(n log n)	O(log n)		59.4%
0991	Broken Calculator	Go	Medium				49.8%
0996	Number of Squareful Arrays	Go	Hard	O(n log n)	O(n)		48.6%
1006	Clumsy Factorial	Go	Medium				54.0%
1017	Convert to Base -2	Go	Medium				59.4%
1025	Divisor Game	Go	Easy	O(1)	O(1)		66.2%
1037	Valid Boomerang	Go	Easy				37.4%
1073	Adding Two Negabinary Numbers	Go	Medium				34.8%
1093	Statistics from a Large Sample	Go	Medium				47.8%
1154	Day of the Year	Go	Easy				49.6%
1175	Prime Arrangements	Go	Easy				51.8%
1201	Ugly Number III	Go	Medium				26.7%
1217	Minimum Cost to Move Chips to The Same Position	Go	Easy				70.7%

1232	Check If It Is a Straight Line	Go	Easy				42.8%
1281	Subtract the Product and Sum of Digits of an Integer	Go	Easy				85.6%
1317	Convert Integer to the Sum of Two No-Zero Integers	Go	Easy				57.1%
1442	Count Triplets That Can Form Two Arrays of Equal XOR	Go	Medium				72.8%
1512	Number of Good Pairs	Go	Easy				87.6%
1551	Minimum Operations to Make Array Equal	Go	Medium				80.6%
1641	Count Sorted Vowel Strings	Go	Medium				75.0%
1648	Sell Diminishing-Valued Colored Balls	Go	Medium				31.1%
1680	Concatenation of Consecutive Binary Numbers	Go	Medium				52.2%
1685	Sum of Absolute Differences in a Sorted Array	Go	Medium				63.4%
1716	Calculate Money in Leetcode Bank	Go	Easy				64.1%
1744	Can You Eat Your Favorite Candy on Your Favorite Day?	Go	Medium				31.3%

Hash Table

No.	Title	Solution	Difficulty	TimeComplexity	SpaceComplexity	Favorite	Acceptance
0001	Two Sum	Go	Easy	O(n)	O(n)		47.1%
0003	Longest Substring Without Repeating Characters	Go	Medium	O(n)	O(1)	❤️	31.8%
0018	4Sum	Go	Medium	O(n^3)	O(n^2)	❤️	35.6%
0030	Substring with Concatenation of All Words	Go	Hard	O(n)	O(n)	❤️	26.7%
0036	Valid Sudoku	Go	Medium	O(n^2)	O(n^2)		51.4%
0037	Sudoku Solver	Go	Hard	O(n^2)	O(n^2)	❤️	48.2%
0049	Group Anagrams	Go	Medium	O(n log n)	O(n)		60.5%
0076	Minimum Window Substring	Go	Hard	O(n)	O(n)	❤️	36.6%
0094	Binary Tree Inorder Traversal	Go	Easy	O(n)	O(1)		67.1%
0136	Single Number	Go	Easy				67.1%
0138	Copy List with Random Pointer	Go	Medium	O(n)	O(1)		42.5%
0187	Repeated DNA Sequences	Go	Medium				42.0%
0202	Happy Number	Go	Easy	O(log n)	O(1)		51.7%
0204	Count Primes	Go	Easy				32.8%
0205	Isomorphic Strings	Go	Easy	O(log n)	O(n)		40.9%
0217	Contains Duplicate	Go	Easy	O(n)	O(n)		57.5%
0219	Contains Duplicate II	Go	Easy	O(n)	O(n)		39.3%
0242	Valid Anagram	Go	Easy	O(n)	O(n)		59.3%
0274	H-Index	Go	Medium	O(n)	O(n)		36.6%
0290	Word Pattern	Go	Easy	O(n)	O(n)		38.6%
0347	Top K Frequent Elements	Go	Medium	O(n)	O(n)		63.0%
0349	Intersection of Two Arrays	Go	Easy	O(n)	O(n)		66.1%
0350	Intersection of Two Arrays II	Go	Easy	O(n)	O(n)		52.5%
0387	First Unique Character in a String	Go	Easy				54.5%
0389	Find the Difference	Go	Easy				58.2%

0409	Longest Palindrome	Go	Easy				52.4%
0438	Find All Anagrams in a String	Go	Medium	O(n)	O(1)		45.6%
0447	Number of Boomerangs	Go	Medium	O(n)	O(1)		52.7%
0451	Sort Characters By Frequency	Go	Medium	O(n log n)	O(1)		65.1%
0454	4Sum II	Go	Medium	O(n^2)	O(n)		54.9%
0463	Island Perimeter	Go	Easy				67.1%
0500	Keyboard Row	Go	Easy				66.2%
0508	Most Frequent Subtree Sum	Go	Medium				59.9%
0525	Contiguous Array	Go	Medium				43.9%
0535	Encode and Decode TinyURL	Go	Medium				82.7%
0554	Brick Wall	Go	Medium				51.8%
0575	Distribute Candies	Go	Easy				64.6%
0594	Longest Harmonious Subsequence	Go	Easy				51.5%
0599	Minimum Index Sum of Two Lists	Go	Easy				52.3%
0609	Find Duplicate File in System	Go	Medium				63.0%
0632	Smallest Range Covering Elements from K Lists	Go	Hard				55.2%
0645	Set Mismatch	Go	Easy				40.9%
0648	Replace Words	Go	Medium	O(n)	O(n)		59.8%
0676	Implement Magic Dictionary	Go	Medium	O(n)	O(n)		55.5%
0690	Employee Importance	Go	Easy				59.8%
0692	Top K Frequent Words	Go	Medium				53.5%
0705	Design HashSet	Go	Easy				63.9%
0706	Design HashMap	Go	Easy				63.9%
0710	Random Pick with Blacklist	Go	Hard	O(n)	O(n)		33.2%
0718	Maximum Length of Repeated Subarray	Go	Medium				50.9%
0720	Longest Word in Dictionary	Go	Easy	O(n)	O(n)		49.7%
0726	Number of Atoms	Go	Hard	O(n)	O(n)	❤️	51.1%
0739	Daily Temperatures	Go	Medium	O(n)	O(n)		65.3%
0748	Shortest Completing Word	Go	Easy				57.9%
0771	Jewels and Stones	Go	Easy				87.2%
0781	Rabbits in Forest	Go	Medium				56.2%
0811	Subdomain Visit Count	Go	Easy				72.2%
0884	Uncommon Words from Two Sentences	Go	Easy				64.4%
0895	Maximum Frequency Stack	Go	Hard	O(n)	O(n)		63.6%
0930	Binary Subarrays With Sum	Go	Medium	O(n)	O(n)	❤️	45.6%
0953	Verifying an Alien Dictionary	Go	Easy				52.3%
0961	N-Repeated Element in Size 2N Array	Go	Easy				74.9%
0966	Vowel Spellchecker	Go	Medium				51.8%
0970	Powerful Integers	Go	Medium				43.4%
0981	Time Based Key-Value Store	Go	Medium				54.5%
0987	Vertical Order Traversal of a Binary Tree	Go	Hard				39.3%
0992	Subarrays with K Different Integers	Go	Hard	O(n)	O(n)	❤️	51.5%
1002	Find Common Characters	Go	Easy				68.7%
1048	Longest String Chain	Go	Medium				56.2%

1078	Occurrences After Bigram	Go	Easy				64.8%
1160	Find Words That Can Be Formed by Characters	Go	Easy				67.9%
1178	Number of Valid Words for Each Puzzle	Go	Hard				40.3%
1189	Maximum Number of Balloons	Go	Easy				62.0%
1207	Unique Number of Occurrences	Go	Easy				72.2%
1512	Number of Good Pairs	Go	Easy				87.6%
1539	Kth Missing Positive Number	Go	Easy				54.7%
1640	Check Array Formation Through Concatenation	Go	Easy				55.5%
1679	Max Number of K-Sum Pairs	Go	Medium				53.6%
1748	Sum of Unique Elements	Go	Easy				74.6%

Sort

LeetCode Explore Problems Mock New Contest Articles Discuss Store Premium

Sort

Subscribe to see which companies asked this question

You have solved **23 / 33** problems.

Show problem tags

	#	Title	Acceptance	Difficulty	Frequency
✔	56	Merge Intervals	35.9%	Medium	
✔	57	Insert Interval	31.4%	Hard	
✔	75	Sort Colors	42.5%	Medium	
✔	147	Insertion Sort List	37.6%	Medium	
✔	148	Sort List	35.9%	Medium	
✔	164	Maximum Gap	32.7%	Hard	
✔	179	Largest Number	26.0%	Medium	
✔	220	Contains Duplicate III	19.8%	Medium	
✔	242	Valid Anagram	52.6%	Easy	
	252	Meeting Rooms	52.2%	Easy	
	253	Meeting Rooms II	43.1%	Medium	
✔	274	H-Index	34.7%	Medium	
	280	Wiggle Sort	61.2%	Medium	
	296	Best Meeting Point	55.1%	Hard	
	315	Count of Smaller Numbers After Self	38.4%	Hard	
✔	324	Wiggle Sort II	28.1%	Medium	
	327	Count of Range Sum	33.0%	Hard	
✔	349	Intersection of Two Arrays	55.2%	Easy	
✔	350	Intersection of Two Arrays II	48.1%	Easy	
	493	Reverse Pairs	23.3%	Hard	
✔	524	Longest Word in Dictionary through Deleting	46.0%	Medium	
	527	Word Abbreviation	50.3%	Hard	
✔	767	Reorganize String	42.8%	Medium	
✔	853	Car Fleet	40.0%	Medium	

✔	710	Random Pick with Blacklist	31.3%	Hard
✔	922	Sort Array By Parity II	67.1%	Easy
✔	969	Pancake Sorting	62.5%	Medium
✔	973	K Closest Points to Origin	62.0%	Medium
✔	976	Largest Perimeter Triangle	57.1%	Easy
	1057	Campus Bikes 🔒	58.7%	Medium
	1086	High Five 🔒	76.4%	Easy
✔	1030	Matrix Cells in Distance Order	64.9%	Easy
✔	1054	Distant Barcodes	38.7%	Medium

Copyright © 2019 LeetCode

- 深刻的理解多路快排。第 75 题。
- 链表的排序，插入排序(第 147 题)和归并排序(第 148 题)
- 桶排序和基数排序。第 164 题。
- "摆动排序"。第 324 题。
- 两两不相邻的排序。第 767 题，第 1054 题。
- "饼子排序"。第 969 题。

No.	Title	Solution	Difficulty	TimeComplexity	SpaceComplexity	Favorite	Acceptance
0056	Merge Intervals	Go	Medium	O(n log n)	O(log n)		42.0%
0057	Insert Interval	Go	Medium	O(n)	O(1)		35.8%
0075	Sort Colors	Go	Medium	O(n)	O(1)	❤️	50.6%
0147	Insertion Sort List	Go	Medium	O(n^2)	O(1)	❤️	45.2%
0148	Sort List	Go	Medium	O(n log n)	O(log n)	❤️	47.5%
0164	Maximum Gap	Go	Hard	O(n log n)	O(log n)	❤️	39.7%
0179	Largest Number	Go	Medium	O(n log n)	O(log n)	❤️	31.2%
0220	Contains Duplicate III	Go	Medium	O(n log n)	O(1)	❤️	21.4%
0242	Valid Anagram	Go	Easy	O(n)	O(n)		59.3%
0274	H-Index	Go	Medium	O(n)	O(n)		36.6%
0315	Count of Smaller Numbers After Self	Go	Hard				42.3%
0324	Wiggle Sort II	Go	Medium	O(n)	O(n)	❤️	31.0%
0327	Count of Range Sum	Go	Hard				36.3%
0349	Intersection of Two Arrays	Go	Easy	O(n)	O(n)		66.1%
0350	Intersection of Two Arrays II	Go	Easy	O(n)	O(n)		52.5%
0493	Reverse Pairs	Go	Hard				27.7%
0524	Longest Word in Dictionary through Deleting	Go	Medium	O(n)	O(1)		50.3%
0710	Random Pick with Blacklist	Go	Hard	O(n)	O(n)		33.2%
0767	Reorganize String	Go	Medium	O(n log n)	O(log n)	❤️	50.6%
0853	Car Fleet	Go	Medium	O(n log n)	O(log n)		44.9%
0922	Sort Array By Parity II	Go	Easy	O(n)	O(1)		70.7%
0969	Pancake Sorting	Go	Medium	O(n log n)	O(log n)	❤️	69.0%
0973	K Closest Points to Origin	Go	Medium	O(n log n)	O(log n)		65.0%
0976	Largest Perimeter Triangle	Go	Easy	O(n log n)	O(log n)		59.4%
1030	Matrix Cells in Distance Order	Go	Easy	O(n^2)	O(1)		68.5%
1054	Distant Barcodes	Go	Medium	O(n log n)	O(log n)	❤️	44.6%
1122	Relative Sort Array	Go	Easy				68.0%
1235	Maximum Profit in Job Scheduling	Go	Hard				48.2%
1305	All Elements in Two Binary Search Trees	Go	Medium				77.9%
1329	Sort the Matrix Diagonally	Go	Medium				81.4%
1353	Maximum Number of Events That Can Be Attended	Go	Medium				30.6%
1383	Maximum Performance of a Team	Go	Hard				41.3%
1636	Sort Array by Increasing Frequency	Go	Easy				67.2%
1640	Check Array Formation Through Concatenation	Go	Easy				55.5%
1647	Minimum Deletions to Make Character Frequencies Unique	Go	Medium				55.7%
1648	Sell Diminishing-Valued Colored Balls	Go	Medium				31.1%
1691	Maximum Height by Stacking Cuboids	Go	Hard				51.0%
1710	Maximum Units on a Truck	Go	Easy				72.5%

Bit Manipulation

Bit Manipulation

Subscribe to see which companies asked this question

You have solved **29 / 32** problems.

☐ **Show problem tags**

	#	Title	Acceptance	Difficulty	Frequency
✔	78	Subsets	53.6%	Medium	
✔	136	Single Number	60.6%	Easy	
✔	137	Single Number II	46.4%	Medium	
✔	169	Majority Element	53.2%	Easy	
✔	187	Repeated DNA Sequences	36.4%	Medium	
✔	190	Reverse Bits	31.9%	Easy	
✔	191	Number of 1 Bits	43.8%	Easy	
✔	201	Bitwise AND of Numbers Range	36.1%	Medium	
✔	231	Power of Two	42.1%	Easy	
✔	260	Single Number III	57.4%	Medium	
✔	268	Missing Number	48.7%	Easy	
✔	318	Maximum Product of Word Lengths	48.8%	Medium	
	320	Generalized Abbreviation 🔒	49.0%	Medium	
✔	338	Counting Bits	65.1%	Medium	
✔	342	Power of Four	40.5%	Easy	
✔	371	Sum of Two Integers	50.8%	Easy	
✔	389	Find the Difference	53.3%	Easy	
✔	393	UTF-8 Validation	36.1%	Medium	
✔	397	Integer Replacement	31.6%	Medium	
✔	401	Binary Watch	45.5%	Easy	
✔	405	Convert a Number to Hexadecimal	42.1%	Easy	
	411	Minimum Unique Word Abbreviation 🔒	35.0%	Hard	
✔	421	Maximum XOR of Two Numbers in an Array	51.3%	Medium	
✔	461	Hamming Distance	70.5%	Easy	
✔	476	Number Complement	62.5%	Easy	
✔	477	Total Hamming Distance	49.1%	Medium	
✔	693	Binary Number with Alternating Bits	58.1%	Easy	
	751	IP to CIDR 🔒	59.9%	Easy	
✔	756	Pyramid Transition Matrix	52.0%	Medium	
✔	762	Prime Number of Set Bits in Binary Representation	59.8%	Easy	
✔	784	Letter Case Permutation	57.6%	Easy	
✔	898	Bitwise ORs of Subarrays	34.7%	Medium	

Copyright © 2019 LeetCode

- 异或的特性。第 136 题，第 268 题，第 389 题，第 421 题，

```
x ^ 0 = x
x ^ 11111……1111 = ~x
x ^ (~x) = 11111……1111
x ^ x = 0
a ^ b = c  => a ^ c = b  => b ^ c = a (交换律)
a ^ b ^ c = a ^ (b ^ c) = (a ^ b) ^ c (结合律)
```

- 构造特殊 Mask，将特殊位置放 0 或 1。

```
将 x 最右边的 n 位清零， x & ( ~0 << n )
获取 x 的第 n 位值(0 或者 1)，(x >> n) & 1
获取 x 的第 n 位的幂值，x & (1 << (n - 1))
仅将第 n 位置为 1，x | (1 << n)
仅将第 n 位置为 0，x & (~(1 << n))
将 x 最高位至第 n 位(含)清零，x & ((1 << n) - 1)
将第 n 位至第 0 位(含)清零，x & (~((1 << (n + 1)) - 1))
```

- 有特殊意义的 & 位操作运算。第 260 题，第 201 题，第 318 题，第 371 题，第 397 题，第 461 题，第 693 题，

```
X & 1 == 1 判断是否是奇数(偶数)
X & = (X - 1) 将最低位(LSB)的 1 清零
X & -X 得到最低位(LSB)的 1
X & ~X = 0
```

No.	Title	Solution	Difficulty	TimeComplexity	SpaceComplexity	Favorite	Acceptance
0078	Subsets	Go	Medium	O(n^2)	O(n)	❤️	66.6%
0136	Single Number	Go	Easy	O(n)	O(1)		67.1%
0137	Single Number II	Go	Medium	O(n)	O(1)	❤️	54.6%
0169	Majority Element	Go	Easy	O(n)	O(1)	❤️	60.6%
0187	Repeated DNA Sequences	Go	Medium	O(n)	O(1)		42.0%
0190	Reverse Bits	Go	Easy	O(n)	O(1)	❤️	43.5%
0191	Number of 1 Bits	Go	Easy	O(n)	O(1)		55.1%
0201	Bitwise AND of Numbers Range	Go	Medium	O(n)	O(1)	❤️	39.8%
0231	Power of Two	Go	Easy	O(1)	O(1)		43.8%
0260	Single Number III	Go	Medium	O(n)	O(1)	❤️	65.5%
0268	Missing Number	Go	Easy	O(n)	O(1)		56.1%
0318	Maximum Product of Word Lengths	Go	Medium	O(n)	O(1)		55.3%
0338	Counting Bits	Go	Easy	O(n)	O(n)		71.0%
0342	Power of Four	Go	Easy	O(n)	O(1)		42.1%
0371	Sum of Two Integers	Go	Medium	O(n)	O(1)		50.6%
0389	Find the Difference	Go	Easy	O(n)	O(1)		58.2%
0393	UTF-8 Validation	Go	Medium	O(n)	O(1)		38.4%
0397	Integer Replacement	Go	Medium	O(n)	O(1)		33.7%
0401	Binary Watch	Go	Easy	O(1)	O(1)		48.9%
0405	Convert a Number to Hexadecimal	Go	Easy	O(n)	O(1)		44.8%
0421	Maximum XOR of Two Numbers in an Array	Go	Medium	O(n)	O(1)	❤️	54.7%
0461	Hamming Distance	Go	Easy	O(n)	O(1)		73.4%
0476	Number Complement	Go	Easy	O(n)	O(1)		65.2%
0477	Total Hamming Distance	Go	Medium	O(n)	O(1)		50.9%
0693	Binary Number with Alternating Bits	Go	Easy	O(n)	O(1)	❤️	60.2%
0756	Pyramid Transition Matrix	Go	Medium	O(n log n)	O(n)		55.9%
0762	Prime Number of Set Bits in Binary Representation	Go	Easy	O(n)	O(1)		64.9%
0784	Letter Case Permutation	Go	Medium	O(n)	O(1)		69.2%
0898	Bitwise ORs of Subarrays	Go	Medium	O(n)	O(1)		35.0%
1178	Number of Valid Words for Each Puzzle	Go	Hard				40.3%
1239	Maximum Length of a Concatenated String with Unique Characters	Go	Medium				50.5%
1290	Convert Binary Number in a Linked List to Integer	Go	Easy				81.7%
1310	XOR Queries of a Subarray	Go	Medium				69.8%
1442	Count Triplets That Can Form Two Arrays of Equal XOR	Go	Medium				72.8%
1461	Check If a String Contains All Binary Codes of Size K	Go	Medium				54.2%
1486	XOR Operation in an Array	Go	Easy				83.9%
1720	Decode XORed Array	Go	Easy				85.5%
1734	Decode XORed Permutation	Go	Medium				56.7%

Union Find

LeetCode Explore Problems Mock New Contest Articles Discuss Store Premium

Union Find

Subscribe to see which companies asked this question

You have solved **18 / 26** problems.

Show problem tags

	#	Title	Acceptance	Difficulty	Frequency
✔	128	Longest Consecutive Sequence	42.4%	Hard	
✔	130	Surrounded Regions	23.7%	Medium	
✔	200	Number of Islands	42.5%	Medium	
	261	Graph Valid Tree	40.3%	Medium	
	305	Number of Islands II	41.4%	Hard	
	323	Number of Connected Components in an Undirected Graph	52.7%	Medium	
✔	399	Evaluate Division	48.3%	Medium	
✔	547	Friend Circles	54.6%	Medium	
✔	684	Redundant Connection	52.8%	Medium	
✔	685	Redundant Connection II	31.1%	Hard	
✔	721	Accounts Merge	41.9%	Medium	
	737	Sentence Similarity II	43.9%	Medium	
✔	765	Couples Holding Hands	52.1%	Hard	
✔	778	Swim in Rising Water	48.6%	Hard	
✔	803	Bricks Falling When Hit	29.2%	Hard	
✔	839	Similar String Groups	35.3%	Hard	
✔	924	Minimize Malware Spread	40.5%	Hard	
✔	928	Minimize Malware Spread II	39.4%	Hard	
✔	947	Most Stones Removed with Same Row or Column	54.4%	Medium	
✔	952	Largest Component Size by Common Factor	26.7%	Hard	
✔	959	Regions Cut By Slashes	62.6%	Medium	
✔	990	Satisfiability of Equality Equations	40.9%	Medium	
	1061	Lexicographically Smallest Equivalent String	62.7%	Medium	
	1101	The Earliest Moment When Everyone Become Friends	64.0%	Medium	
	1102	Path With Maximum Minimum Value	43.9%	Medium	
	1135	Connecting Cities With Minimum Cost	51.4%	Medium	

Copyright © 2019 LeetCode

Help Center | Jobs | Bug Bounty | Terms | Privacy Policy

- 灵活使用并查集的思想，熟练掌握并查集的模板，模板中有两种并查集的实现方式，一种是路径压缩 + 秩优化的版本，另外一种是计算每个集合中元素的个数 + 最大集合元素个数的版本，这两种版本都有各自使用的地方。能使用第一类并查集模板的题目有：第 128 题，第 130 题，第 547 题，第 684 题，第 721 题，第 765 题，第 778 题，第 839 题，第 924 题，第 928 题，第 947 题，第 952 题，第 959 题，第 990 题。能使

用第二类并查集模板的题目有：第 803 题，第 952 题。第 803 题秩优化和统计集合个数这些地方会卡时间，如果不优化，会 TLE。

- 并查集是一种思想，有些题需要灵活使用这种思想，而不是死套模板，如第 399 题，这一题是 stringUnionFind，利用并查集思想实现的。这里每个节点是基于字符串和 map 的，而不是单纯的用 int 节点编号实现的。
- 有些题死套模板反而做不出来，比如第 685 题，这一题不能路径压缩和秩优化，因为题目中涉及到有向图，需要知道节点的前驱节点，如果路径压缩了，这一题就没法做了。这一题不需要路径压缩和秩优化。
- 灵活的抽象题目给的信息，将给定的信息合理的编号，使用并查集解题，并用 map 降低时间复杂度，如第 721 题，第 959 题。
- 关于地图，砖块，网格的题目，可以新建一个特殊节点，将四周边缘的砖块或者网格都 union() 到这个特殊节点上。第 130 题，第 803 题。
- 能用并查集的题目，一般也可以用 DFS 和 BFS 解答，只不过时间复杂度会高一点。

No.	Title	Solution	Difficulty	TimeComplexity	SpaceComplexity	Favorite	Acceptance
0128	Longest Consecutive Sequence	Go	Medium	O(n)	O(n)	❤️	47.3%
0130	Surrounded Regions	Go	Medium	O(m*n)	O(m*n)		30.4%
0200	Number of Islands	Go	Medium	O(m*n)	O(m*n)		50.3%
0399	Evaluate Division	Go	Medium	O(n)	O(n)		55.1%
0547	Number of Provinces	Go	Medium	O(n^2)	O(n)		61.3%
0684	Redundant Connection	Go	Medium	O(n)	O(n)		59.6%
0685	Redundant Connection II	Go	Hard	O(n)	O(n)		33.2%
0721	Accounts Merge	Go	Medium	O(n)	O(n)	❤️	53.0%
0765	Couples Holding Hands	Go	Hard	O(n)	O(n)	❤️	55.8%
0778	Swim in Rising Water	Go	Hard	O(n^2)	O(n)	❤️	55.3%
0803	Bricks Falling When Hit	Go	Hard	O(n^2)	O(n)	❤️	32.2%
0839	Similar String Groups	Go	Hard	O(n^2)	O(n)		42.6%
0924	Minimize Malware Spread	Go	Hard	O(m*n)	O(n)		41.8%
0928	Minimize Malware Spread II	Go	Hard	O(m*n)	O(n)	❤️	41.5%
0947	Most Stones Removed with Same Row or Column	Go	Medium	O(n)	O(n)		55.9%
0952	Largest Component Size by Common Factor	Go	Hard	O(n)	O(n)	❤️	36.5%
0959	Regions Cut By Slashes	Go	Medium	O(n^2)	O(n^2)	❤️	67.6%
0990	Satisfiability of Equality Equations	Go	Medium	O(n)	O(n)		47.5%
1202	Smallest String With Swaps	Go	Medium				49.5%
1319	Number of Operations to Make Network Connected	Go	Medium				55.6%
1579	Remove Max Number of Edges to Keep Graph Fully Traversable	Go	Hard				46.8%
1631	Path With Minimum Effort	Go	Medium				50.3%

Sliding Window

LeetCode Explore Problems Mock New Contest Articles Discuss Store Premium

Sliding Window

Subscribe to see which companies asked this question

You have solved **13 / 19** problems.

Show problem tags

	#	Title	Acceptance	Difficulty	Frequency
✔	3	Longest Substring Without Repeating Characters	28.8%	Medium	
✔	76	Minimum Window Substring	31.8%	Hard	
	159	Longest Substring with At Most Two Distinct Characters	47.7%	Hard	
✔	239	Sliding Window Maximum	39.2%	Hard	
	340	Longest Substring with At Most K Distinct Characters	40.9%	Hard	
✔	424	Longest Repeating Character Replacement	44.7%	Medium	
✔	480	Sliding Window Median	33.6%	Hard	
✔	567	Permutation in String	39.2%	Medium	
	727	Minimum Window Subsequence	38.3%	Hard	
✔	978	Longest Turbulent Subarray	45.7%	Medium	
✔	992	Subarrays with K Different Integers	45.2%	Hard	
✔	995	Minimum Number of K Consecutive Bit Flips	47.5%	Hard	
✔	1004	Max Consecutive Ones III	54.6%	Medium	
	1100	Find K-Length Substrings With No Repeated Characters	71.0%	Medium	
	1151	Minimum Swaps to Group All 1's Together	57.1%	Medium	
✔	1040	Moving Stones Until Consecutive II	48.4%	Medium	
✔	1052	Grumpy Bookstore Owner	52.8%	Medium	
✔	1074	Number of Submatrices That Sum to Target	57.9%	Hard	
	1176	Diet Plan Performance	47.2%	Easy	

Copyright © 2019 LeetCode

Help Center | Jobs | Bug Bounty | Terms | Privacy Policy

- 双指针滑动窗口的经典写法。右指针不断往右移，移动到不能往右移动为止(具体条件根据题目而定)。当右指针到最右边以后，开始挪动左指针，释放窗口左边界。第 3 题，第 76 题，第 209 题，第 424 题，第 438 题，第 567 题，第 713 题，第 763 题，第 845 题，第 881 题，第 904 题，第 978 题，第 992 题，第 1004 题，第 1040 题，第 1052 题。

```go
left, right := 0, -1

for left < len(s) {
  if right+1 < len(s) && freq[s[right+1]-'a'] == 0 {
    freq[s[right+1]-'a']++
    right++
  } else {
    freq[s[left]-'a']--
    left++
  }
  result = max(result, right-left+1)
}
```

- 滑动窗口经典题。第 239 题，第 480 题。

No.	Title	Solution	Difficulty	TimeComplexity	SpaceComplexity	Favorite	Acceptance
0003	Longest Substring Without Repeating Characters	Go	Medium	O(n)	O(1)	❤️	31.8%
0076	Minimum Window Substring	Go	Hard	O(n)	O(n)	❤️	36.6%
0239	Sliding Window Maximum	Go	Hard	O(n * k)	O(n)	❤️	45.1%
0395	Longest Substring with At Least K Repeating Characters	Go	Medium				43.8%
0424	Longest Repeating Character Replacement	Go	Medium	O(n)	O(1)		48.9%
0480	Sliding Window Median	Go	Hard	O(n * log k)	O(k)	❤️	39.5%
0567	Permutation in String	Go	Medium	O(n)	O(1)	❤️	44.6%
0978	Longest Turbulent Subarray	Go	Medium	O(n)	O(1)	❤️	46.7%
0992	Subarrays with K Different Integers	Go	Hard	O(n)	O(n)	❤️	51.5%
0995	Minimum Number of K Consecutive Bit Flips	Go	Hard	O(n)	O(1)	❤️	50.3%
1004	Max Consecutive Ones III	Go	Medium	O(n)	O(1)		61.3%
1040	Moving Stones Until Consecutive II	Go	Medium	O(n log n)	O(1)	❤️	54.8%
1052	Grumpy Bookstore Owner	Go	Medium	O(n log n)	O(1)		56.1%
1074	Number of Submatrices That Sum to Target	Go	Hard	O(n^3)	O(n)	❤️	65.2%
1208	Get Equal Substrings Within Budget	Go	Medium				44.7%
1423	Maximum Points You Can Obtain from Cards	Go	Medium				48.5%
1438	Longest Continuous Subarray With Absolute Diff Less Than or Equal to Limit	Go	Medium				44.7%
1658	Minimum Operations to Reduce X to Zero	Go	Medium				33.3%

Segment Tree

LeetCode Explore Problems Mock Contest Articles Discuss Store Premium

Segment Tree

Subscribe to see which companies asked this question

You have solved **10 / 11** problems.

Show problem tags

	#	Title	Acceptance	Difficulty	Frequency
✔	218	The Skyline Problem	32.5%	Hard	
✔	307	Range Sum Query - Mutable	30.4%	Medium	
	308	Range Sum Query 2D - Mutable	33.1%	Hard	
✔	315	Count of Smaller Numbers After Self	39.4%	Hard	
✔	327	Count of Range Sum	33.7%	Hard	
✔	493	Reverse Pairs	23.7%	Hard	
✔	699	Falling Squares	40.5%	Hard	
✔	715	Range Module	36.5%	Hard	
✔	732	My Calendar III	56.7%	Hard	
✔	850	Rectangle Area II	45.8%	Hard	
✔	1157	Online Majority Element In Subarray	33.1%	Hard	

Copyright © 2019 LeetCode Help Center | Jobs | Bug Bounty | Students | Terms | Privacy Policy

- 线段树的经典数组实现写法。将合并两个节点 pushUp 逻辑抽象出来了，可以实现任意操作(常见的操作有：加法，取 max，min 等等)。第 218 题，第 303 题，第 307 题，第 699 题。
- 计数线段树的经典写法。第 315 题，第 327 题，第 493 题。
- 线段树的树的实现写法。第 715 题，第 732 题。
- 区间懒惰更新。第 218 题，第 699 题。
- 离散化。离散化需要注意一个特殊情况：假如三个区间为 [1,10] [1,4] [6,10]，离散化后 x[1]=1,x[2]=4,x[3]=6,x[4]=10。第一个区间为 [1,4]，第二个区间为 [1,2]，第三个区间为 [3,4]，这样一来，区间一 = 区间二 + 区间三，这和离散前的模型不符，离散前，很明显，区间一 > 区间二 + 区间三。正确的做法是：在相差大于 1 的数间加一个数，例如在上面 1 4 6 10 中间加 5，即可 x[1]=1,x[2]=4,x[3]=5,x[4]=6,x[5]=10。这样处理之后，区间一是 1-5 ，区间二是 1-2 ，区间三是 4-5 。
- 灵活构建线段树。线段树节点可以存储多条信息，合并两个节点的 pushUp 操作也可以是多样的。第 850 题，第 1157 题。

线段树题型从简单到困难:

1. 单点更新:
   HDU 1166 敌兵布阵 update:单点增减 query:区间求和
   HDU 1754 I Hate It update:单点替换 query:区间最值
   HDU 1394 Minimum Inversion Number update:单点增减 query:区间求和
   HDU 2795 Billboard query:区间求最大值的位子(直接把update的操作在query里做了)
2. 区间更新:
   HDU 1698 Just a Hook update:成段替换 (由于只query一次总区间,所以可以直接输出 1 结点的信息)
   POJ 3468 A Simple Problem with Integers update:成段增减 query:区间求和

POJ 2528 Mayor's posters 离散化 + update:成段替换 query:简单hash
POJ 3225 Help with Intervals update:成段替换,区间异或 query:简单hash

3. 区间合并(这类题目会询问区间中满足条件的连续最长区间,所以PushUp的时候需要对左右儿子的区间进行合并):
   POJ 3667 Hotel update:区间替换 query:询问满足条件的最左端点
4. 扫描线(这类题目需要将一些操作排序,然后从左到右用一根扫描线扫过去最典型的就是矩形面积并,周长并等题):
   HDU 1542 Atlantis update:区间增减 query:直接取根节点的值
   HDU 1828 Picture update:区间增减 query:直接取根节点的值

No.	Title	Solution	Difficulty	TimeComplexity	SpaceComplexity	Favorite	Acceptance
0218	The Skyline Problem	Go	Hard	O(n log n)	O(n)	❤️	37.1%
0307	Range Sum Query - Mutable	Go	Medium	O(1)	O(n)		37.7%
0315	Count of Smaller Numbers After Self	Go	Hard	O(n log n)	O(n)		42.3%
0327	Count of Range Sum	Go	Hard	O(n log n)	O(n)	❤️	36.3%
0493	Reverse Pairs	Go	Hard	O(n log n)	O(n)		27.7%
0699	Falling Squares	Go	Hard	O(n log n)	O(n)	❤️	43.4%
0715	Range Module	Go	Hard	O(log n)	O(n)	❤️	41.6%
0732	My Calendar III	Go	Hard	O(log n)	O(n)	❤️	63.5%
0850	Rectangle Area II	Go	Hard	O(n log n)	O(n)	❤️	48.6%
1157	Online Majority Element In Subarray	Go	Hard	O(log n)	O(n)	❤️	41.2%
1353	Maximum Number of Events That Can Be Attended	Go	Medium				30.6%
1649	Create Sorted Array through Instructions	Go	Hard				36.8%

Binary Indexed Tree

No.	Title	Solution	Difficulty	TimeComplexity	SpaceComplexity	Favorite	Acceptance
0218	The Skyline Problem	Go	Hard				37.1%
0307	Range Sum Query - Mutable	Go	Medium				37.7%
0315	Count of Smaller Numbers After Self	Go	Hard				42.3%
0327	Count of Range Sum	Go	Hard				36.3%
0493	Reverse Pairs	Go	Hard				27.7%
1649	Create Sorted Array through Instructions	Go	Hard				36.8%

第三章 一些模板

这一章会罗列一些整理好的模板。一起来看看吧。

线段树 Segment Tree

线段树 Segment tree 是一种二叉树形数据结构，1977年由 Jon Louis Bentley 发明，用以存储区间或线段，并且允许快速查询结构内包含某一点的所有区间。

一个包含 {{< katex >}}n {{< /katex >}}个区间的线段树，空间复杂度为 {{< katex >}} O(n) {{< /katex >}}，查询的时间复杂度则为{{< katex >}}O(log n+k) {{< /katex >}}，其中 {{< katex >}} k {{< /katex >}} 是符合条件的区间数量。线段树的数据结构也可推广到高维度。

一. 什么是线段树

以一维的线段树为例。

令 S 是一维线段的集合。将这些线段的端点坐标由小到大排序，令其为{{< katex >}}x*{1},x*{2},\cdots ,x_{m} {{< /katex >}}。我们将被这些端点切分的每一个区间称为“单位区间”（每个端点所在的位置会单独成为一个单位区间），从左到右包含：

{{< katex display>}}
(-\infty ,x*{1}),[x*{1},x*{1}],(x*{1},x*{2}),[x*{2},x*{2}],...,(x*{m-1},x*{m}),[x*{m},x*{m}],(x*{m},+\infty)
{{< /katex >}}

线段树的结构为一个二叉树，每个节点都代表一个坐标区间，节点 N 所代表的区间记为 Int(N)，则其需符合以下条件：

- 其每一个叶节点，从左到右代表每个单位区间。
- 其内部节点代表的区间是其两个儿子代表的区间之并集。
- 每个节点（包含叶子）中有一个存储线段的数据结构。若一个线段 S 的坐标区间包含 Int(N) 但不包含 Int(parent(N))，则节点 N 中会存储线段 S。

线段树是二叉树，其中每个节点代表一个区间。通常，一个节点将存储一个或多个合并的区间的数据，以便可以执行查询操作。

二. 为什么需要这种数据结构

许多问题要求我们基于对可用数据范围或区间的查询来给出结果。这可能是一个繁琐而缓慢的过程，尤其是在查询数量众多且重复的情况下。线段树让我们以对数时间复杂度有效地处理此类查询。

线段树可用于计算几何和地理信息系统领域。例如，距中心参考点/原点一定距离的空间中可能会有大量点。假设我们要查找距原点一定距离范围内的点。一个普通的查找表将需要对所有可能的点或所有可能的距离进行线性扫描（假设是散列图）。线段树使我们能够以对数时间实现这一需求，而所需空间却少得多。这样的问题称为平面范围搜索。有效地解决此类问题至关重要，尤其是在处理动态数据且瞬息万变的情况下（例如，用于空中交通的雷达系统）。下文会以线段树解决 Range Sum Query problem 为例。

上图即作为范围查询的线段树。

三. 构造线段树

假设数据存在 size 为 n 的 arr[] 中。

1. 线段树的根通常代表整个数据区间。这里是 arr[0：n-1]。
2. 树的每个叶子代表一个范围，其中仅包含一个元素。 因此，叶子代表 arr[0]，arr[1] 等等，直到 arr[n-1]。
3. 树的内部节点将代表其子节点的合并或并集结果。
4. 每个子节点可代表其父节点所代表范围的大约一半。(二分的思想)

使用大小为 {{< katex >}}\approx 4 \ast n {{< /katex >}} 的数组可以轻松表示 n 个元素范围的线段树。（Stack Overflow 对原因进行了很好的讨论。如果你还不确定，请不要担心。本文将在稍后进行讨论。）

下标为 i 的节点有两个节点，下标分别为 {{< katex >}}(2 \ast i + 1) {{< /katex >}}和 {{< katex >}}(2 \ast i + 2){{< /katex >}}。

线段树看上去很直观并且非常适合递归构造。

我们将使用数组 tree[] 来存储线段树的节点（初始化为全零）。 下标从 0 开始。

- 树的节点在下标 0 处。因此 tree[0] 是树的根。
- tree[i] 的孩子存在 tree[2 * i + 1] 和 tree[2 * i + 2] 中。
- 用额外的 0 或 null 值填充 arr[]，使得 {{< katex >}}n = 2^{k} {{< /katex >}}（其中 n 是 arr[] 的总长度，而 k 是非负整数。）
- 叶子节点的下标取值范围在 {{< katex >}} \in [2^{k}-1, 2^{k+1}-2]{{< /katex >}}

构造线段树的代码如下：

```
// SegmentTree define
type SegmentTree struct {
  data, tree, lazy []int
  left, right      int
  merge            func(i, j int) int
}

// Init define
func (st *SegmentTree) Init(nums []int, oper func(i, j int) int) {
  st.merge = oper
  data, tree, lazy := make([]int, len(nums)), make([]int, 4*len(nums)), make([]int,
4*len(nums))
  for i := 0; i < len(nums); i++ {
    data[i] = nums[i]
  }
  st.data, st.tree, st.lazy = data, tree, lazy
  if len(nums) > 0 {
    st.buildSegmentTree(0, 0, len(nums)-1)
  }
```

```go
}

// 在 treeIndex 的位置创建 [left....right] 区间的线段树
func (st *SegmentTree) buildSegmentTree(treeIndex, left, right int) {
  if left == right {
    st.tree[treeIndex] = st.data[left]
    return
  }
  midTreeIndex, leftTreeIndex, rightTreeIndex := left+(right-left)>>1,
st.leftChild(treeIndex), st.rightChild(treeIndex)
  st.buildSegmentTree(leftTreeIndex, left, midTreeIndex)
  st.buildSegmentTree(rightTreeIndex, midTreeIndex+1, right)
  st.tree[treeIndex] = st.merge(st.tree[leftTreeIndex], st.tree[rightTreeIndex])
}

func (st *SegmentTree) leftChild(index int) int {
  return 2*index + 1
}

func (st *SegmentTree) rightChild(index int) int {
  return 2*index + 2
}
```

笔者将线段树合并的操作变成了一个函数。合并操作根据题意变化，常见的有加法，取 max，min 等等。

我们以 arr[] = [18, 17, 13, 19, 15, 11, 20, 12, 33, 25] 为例构造线段树：

线段树构造好以后，数组里面的数据是：

```
tree[] = [ 183, 82, 101, 48, 34, 43, 58, 35, 13, 19, 15, 31, 12, 33, 25, 18, 17, 0, 0,
0, 0, 0, 0, 11, 20, 0, 0, 0, 0, 0, 0 ]
```

线段树用 0 填充到 4*n 个元素。

> LeetCode 对应题目是 218. The Skyline Problem、303. Range Sum Query - Immutable、307. Range Sum Query - Mutable、699. Falling Squares

四. 线段树的查询

线段树的查询方法有两种，一种是直接查询，另外一种是懒查询。

1. 直接查询

当查询范围与当前节点表示的范围完全匹配时，该方法返回结果。否则，它会更深入地遍历线段树树，以找到与节点的一部分完全匹配的节点。

```go
// 查询 [left....right] 区间内的值

// Query define
func (st *SegmentTree) Query(left, right int) int {
  if len(st.data) > 0 {
    return st.queryInTree(0, 0, len(st.data)-1, left, right)
  }
  return 0
}

// 在以 treeIndex 为根的线段树中 [left...right] 的范围里，搜索区间 [queryLeft...queryRight]
的值
func (st *SegmentTree) queryInTree(treeIndex, left, right, queryLeft, queryRight int)
int {
  if left == queryLeft && right == queryRight {
    return st.tree[treeIndex]
  }
  midTreeIndex, leftTreeIndex, rightTreeIndex := left+(right-left)>>1,
st.leftChild(treeIndex), st.rightChild(treeIndex)
  if queryLeft > midTreeIndex {
    return st.queryInTree(rightTreeIndex, midTreeIndex+1, right, queryLeft, queryRight)
  } else if queryRight <= midTreeIndex {
    return st.queryInTree(leftTreeIndex, left, midTreeIndex, queryLeft, queryRight)
  }
  return st.merge(st.queryInTree(leftTreeIndex, left, midTreeIndex, queryLeft,
midTreeIndex),
    st.queryInTree(rightTreeIndex, midTreeIndex+1, right, midTreeIndex+1, queryRight))
}
```

在上面的示例中，查询的区间范围为[2，8] 的元素之和。没有任何线段可以完全代表[2，8] 范围。但是可以观察到，可以使用范围 [2，2]，[3，4]，[5，7]，[8，8] 这 4 个区间构成 [8，8]。快速验证 [2,8] 处的输入元素之和为 13 + 19 + 15 + 11 + 20 + 12 + 33 = 123。[2，2]，[3，4]，[5，7] 和 [8，8] 的节点总和是 13 + 34 + 43 + 33 = 123。答案正确。

2. 懒查询

懒查询对应懒更新，两者是配套操作。在区间更新时，并不直接更新区间内所有节点，而是把区间内节点增减变化的值存在 lazy 数组中。等到下次查询的时候再把增减应用到具体的节点上。这样做也是为了分摊时间复杂度，保证查询和更新的时间复杂度在 O(log n) 级别，不会退化成 O(n) 级别。

懒查询节点的步骤：

1. 先判断当前节点是否是懒节点。通过查询 lazy[i] 是否为 0 判断。如果是懒节点，将它的增减变化应用到该节点上。并且更新它的孩子节点。这一步和更新操作的第一步完全一样。
2. 递归查询子节点，以找到适合的查询节点。

具体代码如下：

```
// 查询 [left....right] 区间内的值

// QueryLazy define
func (st *SegmentTree) QueryLazy(left, right int) int {
  if len(st.data) > 0 {
    return st.queryLazyInTree(0, 0, len(st.data)-1, left, right)
  }
  return 0
}

func (st *SegmentTree) queryLazyInTree(treeIndex, left, right, queryLeft, queryRight
int) int {
```

```
  midTreeIndex, leftTreeIndex, rightTreeIndex := left+(right-left)>>1,
st.leftChild(treeIndex), st.rightChild(treeIndex)
  if left > queryRight || right < queryLeft { // segment completely outside range
    return 0 // represents a null node
  }
  if st.lazy[treeIndex] != 0 { // this node is lazy
    for i := 0; i < right-left+1; i++ {
      st.tree[treeIndex] = st.merge(st.tree[treeIndex], st.lazy[treeIndex])
      // st.tree[treeIndex] += (right - left + 1) * st.lazy[treeIndex] // normalize
current node by removing lazinesss
    }
    if left != right { // update lazy[] for children nodes
      st.lazy[leftTreeIndex] = st.merge(st.lazy[leftTreeIndex], st.lazy[treeIndex])
      st.lazy[rightTreeIndex] = st.merge(st.lazy[rightTreeIndex], st.lazy[treeIndex])
      // st.lazy[leftTreeIndex] += st.lazy[treeIndex]
      // st.lazy[rightTreeIndex] += st.lazy[treeIndex]
    }
    st.lazy[treeIndex] = 0 // current node processed. No longer lazy
  }
  if queryLeft <= left && queryRight >= right { // segment completely inside range
    return st.tree[treeIndex]
  }
  if queryLeft > midTreeIndex {
    return st.queryLazyInTree(rightTreeIndex, midTreeIndex+1, right, queryLeft,
queryRight)
  } else if queryRight <= midTreeIndex {
    return st.queryLazyInTree(leftTreeIndex, left, midTreeIndex, queryLeft, queryRight)
  }
  // merge query results
  return st.merge(st.queryLazyInTree(leftTreeIndex, left, midTreeIndex, queryLeft,
midTreeIndex),
    st.queryLazyInTree(rightTreeIndex, midTreeIndex+1, right, midTreeIndex+1,
queryRight))
}
```

五. 线段树的更新

1. 单点更新

单点更新类似于 `buildSegTree`。更新树的叶子节点的值，该值与更新后的元素相对应。这些更新的值会通过树的上层节点把影响传播到根。

```
// 更新 index 位置的值

// Update define
func (st *SegmentTree) Update(index, val int) {
  if len(st.data) > 0 {
    st.updateInTree(0, 0, len(st.data)-1, index, val)
```

```go
  }
}

// 以 treeIndex 为根，更新 index 位置上的值为 val
func (st *SegmentTree) updateInTree(treeIndex, left, right, index, val int) {
  if left == right {
    st.tree[treeIndex] = val
    return
  }
  midTreeIndex, leftTreeIndex, rightTreeIndex := left+(right-left)>>1,
st.leftChild(treeIndex), st.rightChild(treeIndex)
  if index > midTreeIndex {
    st.updateInTree(rightTreeIndex, midTreeIndex+1, right, index, val)
  } else {
    st.updateInTree(leftTreeIndex, left, midTreeIndex, index, val)
  }
  st.tree[treeIndex] = st.merge(st.tree[leftTreeIndex], st.tree[rightTreeIndex])
}
```

在此示例中，下标为（在原始输入数据中）1、3 和 6 处的元素分别增加了 +3，-1 和 +2。可以看到更改如何沿树传播，一直到根。

2. 区间更新

线段树仅更新单个元素，非常有效，时间复杂度 O(log n)。 但是，如果我们要更新一系列元素怎么办？按照当前的方法，每个元素都必须独立更新，每个元素都会花费一些时间。分别更新每一个叶子节点意味着要多次处理它们的共同祖先。祖先节点可能被更新多次。如果想要减少这种重复计算，该怎么办？

在上面的示例中，根节点被更新了三次，而编号为 82 的节点被更新了两次。这是因为更新叶子节点对上层父亲节点有影响。最差的情况，查询的区间内不包含频繁更新的元素，于是需要花费很多时间更新不怎么访问的节点。增加额外的 lazy 数组，可以减少不必要的计算，并且能按需处理节点。

使用另一个数组 lazy[]，它的大小与我们的线段树 array tree[] 完全相同，代表一个惰性节点。当访问或查询该节点时，lazy[i] 中保留需要增加或者减少该节点 tree[i] 的数量。 当 lazy[i] 为 0 时，表示 tree[i] 该节点不是惰性的，并且没有缓存的更新。

更新区间内节点的步骤：

1. 先判断当前节点是否是懒节点。通过查询 lazy[i] 是否为 0 判断。如果是懒节点，将它的增减变化应用到该节点上。并且更新它的孩子节点。
2. 如果当前节点代表的区间位于更新范围内，则将当前更新操作应用于当前节点。
3. 递归更新子节点。

具体代码如下：

```
// 更新 [updateLeft....updateRight] 位置的值
// 注意这里的更新值是在原来值的基础上增加或者减少，而不是把这个区间内的值都赋值为 x，区间更新和单点更新不同
// 这里的区间更新关注的是变化，单点更新关注的是定值
// 当然区间更新也可以都更新成定值，如果只区间更新成定值，那么 lazy 更新策略需要变化，merge 策略也需要变化，这里暂不详细讨论

// UpdateLazy define
func (st *SegmentTree) UpdateLazy(updateLeft, updateRight, val int) {
  if len(st.data) > 0 {
    st.updateLazyInTree(0, 0, len(st.data)-1, updateLeft, updateRight, val)
  }
}
```

```go
func (st *SegmentTree) updateLazyInTree(treeIndex, left, right, updateLeft, updateRight, val int) {
  midTreeIndex, leftTreeIndex, rightTreeIndex := left+(right-left)>>1, st.leftChild(treeIndex), st.rightChild(treeIndex)
  if st.lazy[treeIndex] != 0 { // this node is lazy
    for i := 0; i < right-left+1; i++ {
      st.tree[treeIndex] = st.merge(st.tree[treeIndex], st.lazy[treeIndex])
      //st.tree[treeIndex] += (right - left + 1) * st.lazy[treeIndex] // normalize current node by removing laziness
    }
    if left != right { // update lazy[] for children nodes
      st.lazy[leftTreeIndex] = st.merge(st.lazy[leftTreeIndex], st.lazy[treeIndex])
      st.lazy[rightTreeIndex] = st.merge(st.lazy[rightTreeIndex], st.lazy[treeIndex])
      // st.lazy[leftTreeIndex] += st.lazy[treeIndex]
      // st.lazy[rightTreeIndex] += st.lazy[treeIndex]
    }
    st.lazy[treeIndex] = 0 // current node processed. No longer lazy
  }

  if left > right || left > updateRight || right < updateLeft {
    return // out of range. escape.
  }

  if updateLeft <= left && right <= updateRight { // segment is fully within update range
    for i := 0; i < right-left+1; i++ {
      st.tree[treeIndex] = st.merge(st.tree[treeIndex], val)
      //st.tree[treeIndex] += (right - left + 1) * val // update segment
    }
    if left != right { // update lazy[] for children
      st.lazy[leftTreeIndex] = st.merge(st.lazy[leftTreeIndex], val)
      st.lazy[rightTreeIndex] = st.merge(st.lazy[rightTreeIndex], val)
      // st.lazy[leftTreeIndex] += val
      // st.lazy[rightTreeIndex] += val
    }
    return
  }
  st.updateLazyInTree(leftTreeIndex, left, midTreeIndex, updateLeft, updateRight, val)
  st.updateLazyInTree(rightTreeIndex, midTreeIndex+1, right, updateLeft, updateRight, val)
  // merge updates
  st.tree[treeIndex] = st.merge(st.tree[leftTreeIndex], st.tree[rightTreeIndex])
}
```

> LeetCode 对应题目是 218. The Skyline Problem、699. Falling Squares

六. 时间复杂度分析

让我们看一下构建过程。我们访问了线段树的每个叶子（对应于数组 arr[] 中的每个元素）。因此，我们处理大约 2 * n 个节点。这使构建过程时间复杂度为 O(n)。对于每个递归更新的过程都将丢弃区间范围的一半，以到达树中的叶子节点。这类似于二分搜索，只需要对数时间。更新叶子后，将更新树的每个级别上的直接祖先。这花费时间与树的高度成线性关系。

4*n 个节点可以确保将线段树构建为完整的二叉树，从而树的高度为 log(4*n + 1) 向上取整。线段树读取和更新的时间复杂度都为 O(log n)。

七. 常见题型

1. Range Sum Queries

Range Sum Queries 是 Range Queries 问题的子集。给定一个数据元素数组或序列，需要处理由元素范围组成的读取和更新查询。线段树 Segment Tree 和树状数组 Binary Indexed Tree (a.k.a. Fenwick Tree)) 都能很快的解决这类问题。

Range Sum Query 问题专门处理查询范围内的元素总和。这个问题存在许多变体，包括不可变数据，可变数据，多次更新，单次查询 和 多次更新，多次查询。

2. 单点更新

- HDU 1166 敌兵布阵 update:单点增减 query:区间求和
- HDU 1754 I Hate It update:单点替换 query:区间最值
- HDU 1394 Minimum Inversion Number update:单点增减 query:区间求和
- HDU 2795 Billboard query:区间求最大值的位子(直接把update的操作在query里做了)

3. 区间更新

- HDU 1698 Just a Hook update:成段替换 (由于只query一次总区间,所以可以直接输出 1 结点的信息)
- POJ 3468 A Simple Problem with Integers update:成段增减 query:区间求和
- POJ 2528 Mayor's posters 离散化 + update:成段替换 query:简单hash
- POJ 3225 Help with Intervals update:成段替换,区间异或 query:简单hash

4. 区间合并

这类题目会询问区间中满足条件的连续最长区间,所以PushUp的时候需要对左右儿子的区间进行合并

- POJ 3667 Hotel update:区间替换 query:询问满足条件的最左端点

5. 扫描线

这类题目需要将一些操作排序,然后从左到右用一根扫描线扫过去最典型的就是矩形面积并,周长并等题

- HDU 1542 Atlantis update:区间增减 query:直接取根节点的值
- HDU 1828 Picture update:区间增减 query:直接取根节点的值

6. 计数问题

在 LeetCode 中还有一类问题涉及到计数的。315. Count of Smaller Numbers After Self，327. Count of Range Sum，493. Reverse Pairs 这类问题可以用下面的套路解决。线段树的每个节点存的是区间计数。

```go
// SegmentCountTree define
type SegmentCountTree struct {
  data, tree  []int
  left, right int
  merge       func(i, j int) int
}

// Init define
func (st *SegmentCountTree) Init(nums []int, oper func(i, j int) int) {
  st.merge = oper

  data, tree := make([]int, len(nums)), make([]int, 4*len(nums))
```

```go
  for i := 0; i < len(nums); i++ {
    data[i] = nums[i]
  }
  st.data, st.tree = data, tree
}

// 在 treeIndex 的位置创建 [left....right] 区间的线段树
func (st *SegmentCountTree) buildSegmentTree(treeIndex, left, right int) {
  if left == right {
    st.tree[treeIndex] = st.data[left]
    return
  }
  midTreeIndex, leftTreeIndex, rightTreeIndex := left+(right-left)>>1,
st.leftChild(treeIndex), st.rightChild(treeIndex)
  st.buildSegmentTree(leftTreeIndex, left, midTreeIndex)
  st.buildSegmentTree(rightTreeIndex, midTreeIndex+1, right)
  st.tree[treeIndex] = st.merge(st.tree[leftTreeIndex], st.tree[rightTreeIndex])
}

func (st *SegmentCountTree) leftChild(index int) int {
  return 2*index + 1
}

func (st *SegmentCountTree) rightChild(index int) int {
  return 2*index + 2
}

// 查询 [left....right] 区间内的值

// Query define
func (st *SegmentCountTree) Query(left, right int) int {
  if len(st.data) > 0 {
    return st.queryInTree(0, 0, len(st.data)-1, left, right)
  }
  return 0
}

// 在以 treeIndex 为根的线段树中 [left...right] 的范围里，搜索区间 [queryLeft...queryRight]
的值，值是计数值
func (st *SegmentCountTree) queryInTree(treeIndex, left, right, queryLeft, queryRight
int) int {
  if queryRight < st.data[left] || queryLeft > st.data[right] {
    return 0
  }
  if queryLeft <= st.data[left] && queryRight >= st.data[right] || left == right {
    return st.tree[treeIndex]
  }
  midTreeIndex, leftTreeIndex, rightTreeIndex := left+(right-left)>>1,
st.leftChild(treeIndex), st.rightChild(treeIndex)
```

```go
  return st.queryInTree(rightTreeIndex, midTreeIndex+1, right, queryLeft, queryRight) +
    st.queryInTree(leftTreeIndex, left, midTreeIndex, queryLeft, queryRight)
}

// 更新计数

// UpdateCount define
func (st *SegmentCountTree) UpdateCount(val int) {
  if len(st.data) > 0 {
    st.updateCountInTree(0, 0, len(st.data)-1, val)
  }
}

// 以 treeIndex 为根，更新 [left...right] 区间内的计数
func (st *SegmentCountTree) updateCountInTree(treeIndex, left, right, val int) {
  if val >= st.data[left] && val <= st.data[right] {
    st.tree[treeIndex]++
    if left == right {
      return
    }
    midTreeIndex, leftTreeIndex, rightTreeIndex := left+(right-left)>>1,
st.leftChild(treeIndex), st.rightChild(treeIndex)
    st.updateCountInTree(rightTreeIndex, midTreeIndex+1, right, val)
    st.updateCountInTree(leftTreeIndex, left, midTreeIndex, val)
  }
}
```

并查集 UnionFind

```go
package template

// UnionFind defind
// 路径压缩 + 秩优化
type UnionFind struct {
  parent, rank []int
  count        int
}

// Init define
func (uf *UnionFind) Init(n int) {
  uf.count = n
  uf.parent = make([]int, n)
  uf.rank = make([]int, n)
  for i := range uf.parent {
    uf.parent[i] = i
```

```go
  }
}

// Find define
func (uf *UnionFind) Find(p int) int {
  root := p
  for root != uf.parent[root] {
    root = uf.parent[root]
  }
  // compress path
  for p != uf.parent[p] {
    tmp := uf.parent[p]
    uf.parent[p] = root
    p = tmp
  }
  return root
}

// Union define
func (uf *UnionFind) Union(p, q int) {
  proot := uf.Find(p)
  qroot := uf.Find(q)
  if proot == qroot {
    return
  }
  if uf.rank[qroot] > uf.rank[proot] {
    uf.parent[proot] = qroot
  } else {
    uf.parent[qroot] = proot
    if uf.rank[proot] == uf.rank[qroot] {
      uf.rank[proot]++
    }
  }
  uf.count--
}

// TotalCount define
func (uf *UnionFind) TotalCount() int {
  return uf.count
}

// UnionFindCount define
// 计算每个集合中元素的个数 + 最大集合元素个数
type UnionFindCount struct {
  parent, count []int
  maxUnionCount int
}

// Init define
```

```go
func (uf *UnionFindCount) Init(n int) {
  uf.parent = make([]int, n)
  uf.count = make([]int, n)
  for i := range uf.parent {
    uf.parent[i] = i
    uf.count[i] = 1
  }
}

// Find define
func (uf *UnionFindCount) Find(p int) int {
  root := p
  for root != uf.parent[root] {
    root = uf.parent[root]
  }
  return root
}

// 不进行秩压缩，时间复杂度爆炸，太高了
// func (uf *UnionFindCount) union(p, q int) {
//  proot := uf.find(p)
//  qroot := uf.find(q)
//  if proot == qroot {
//    return
//  }
//  if proot != qroot {
//    uf.parent[proot] = qroot
//    uf.count[qroot] += uf.count[proot]
//  }
// }

// Union define
func (uf *UnionFindCount) Union(p, q int) {
  proot := uf.Find(p)
  qroot := uf.Find(q)
  if proot == qroot {
    return
  }
  if proot == len(uf.parent)-1 {
    //proot is root
  } else if qroot == len(uf.parent)-1 {
    // qroot is root, always attach to root
    proot, qroot = qroot, proot
  } else if uf.count[qroot] > uf.count[proot] {
    proot, qroot = qroot, proot
  }

  //set relation[0] as parent
  uf.maxUnionCount = max(uf.maxUnionCount, (uf.count[proot] + uf.count[qroot]))
```

```go
  uf.parent[qroot] = proot
  uf.count[proot] += uf.count[qroot]
}

// Count define
func (uf *UnionFindCount) Count() []int {
  return uf.count
}

// MaxUnionCount define
func (uf *UnionFindCount) MaxUnionCount() int {
  return uf.maxUnionCount
}

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

最近最少使用 LRUCache

LRU 是 Least Recently Used 的缩写，即最近最少使用，是一种常用的页面置换算法，选择最近最久未使用的页面予以淘汰。如上图，要插入 F 的时候，此时需要淘汰掉原来的一个页面。

根据 LRU 的策略，每次都淘汰最近最久未使用的页面，所以先淘汰 A 页面。再插入 C 的时候，发现缓存中有 C 页面，这个时候需要把 C 页面放到首位，因为它被使用了。以此类推，插入 G 页面，G 页面是新页面，不在缓存中，所以淘汰掉 B 页面。插入 H 页面，H 页面是新页面，不在缓存中，所以淘汰掉 D 页面。插入 E 的时候，发现缓存中有 E 页面，这个时候需要把 E 页面放到首位。插入 I 页面，I 页面是新页面，不在缓存中，所以淘汰掉 F 页面。

可以发现，**LRU 更新和插入新页面都发生在链表首，删除页面都发生在链表尾。**

解法一 Get O(1) / Put O(1)

LRU 要求查询尽量高效，O(1) 内查询。那肯定选用 map 查询。修改，删除也要尽量 O(1) 完成。搜寻常见的数据结构，链表，栈，队列，树，图。树和图排除，栈和队列无法任意查询中间的元素，也排除。所以选用链表来实现。但是如果选用单链表，删除这个结点，需要 O(n) 遍历一遍找到前驱结点。所以选用双向链表，在删除的时候也能 O(1) 完成。

由于 Go 的 container 包中的 list 底层实现是双向链表，所以可以直接复用这个数据结构。定义 LRUCache 的数据结构如下：

```
import "container/list"

type LRUCache struct {
    Cap  int
    Keys map[int]*list.Element
    List *list.List
}

type pair struct {
    K, V int
}

func Constructor(capacity int) LRUCache {
```

```
    return LRUCache{
        Cap: capacity,
        Keys: make(map[int]*list.Element),
        List: list.New(),
    }
}
```

这里需要解释 2 个问题，list 中的值存的是什么？pair 这个结构体有什么用?

```
type Element struct {
  // Next and previous pointers in the doubly-linked list of elements.
  // To simplify the implementation, internally a list l is implemented
  // as a ring, such that &l.root is both the next element of the last
  // list element (l.Back()) and the previous element of the first list
  // element (l.Front()).
  next, prev *Element

  // The list to which this element belongs.
  list *List

  // The value stored with this element.
  Value interface{}
}
```

在 container/list 中，这个双向链表的每个结点的类型是 Element。Element 中存了 4 个值，前驱和后继结点，双向链表的头结点，value 值。这里的 value 是 interface 类型。笔者在这个 value 里面存了 pair 这个结构体。这就解释了 list 里面存的是什么数据。

为什么要存 pair 呢？单单只存 v 不行么，为什么还要存一份 key？原因是在 LRUCache 执行删除操作的时候，需要维护 2 个数据结构，一个是 map，一个是双向链表。在双向链表中删除淘汰出去的 value，在 map 中删除淘汰出去 value 对应的 key。如果在双向链表的 value 中不存储 key，那么再删除 map 中的 key 的时候有点麻烦。如果硬要实现，需要先获取到双向链表这个结点 Element 的地址。然后遍历 map，在 map 中找到存有这个 Element 元素地址对应的 key，再删除。这样做时间复杂度是 O(n)，做不到 O(1)。所以双向链表中的 Value 需要存储这个 pair。

LRUCache 的 Get 操作很简单，在 map 中直接读取双向链表的结点。如果 map 中存在，将它移动到双向链表的表头，并返回它的 value 值，如果 map 中不存在，返回 -1。

```
func (c *LRUCache) Get(key int) int {
  if el, ok := c.Keys[key]; ok {
    c.List.MoveToFront(el)
    return el.Value.(pair).V
  }
  return -1
}
```

LRUCache 的 Put 操作也不难。先查询 map 中是否存在 key，如果存在，更新它的 value，并且把该结点移到双向链表的表头。如果 map 中不存在，新建这个结点加入到双向链表和 map 中。最后别忘记还需要维护双向链表的 cap，如果超过 cap，需要淘汰最后一个结点，双向链表中删除这个结点，map 中删掉这个结点对应的 key。

```
func (c *LRUCache) Put(key int, value int) {
  if el, ok := c.Keys[key]; ok {
    el.Value = pair{K: key, V: value}
    c.List.MoveToFront(el)
  } else {
    el := c.List.PushFront(pair{K: key, V: value})
    c.Keys[key] = el
  }
  if c.List.Len() > c.Cap {
    el := c.List.Back()
    c.List.Remove(el)
    delete(c.Keys, el.Value.(pair).K)
  }
}
```

总结，LRU 是由一个 map 和一个双向链表组成的数据结构。map 中 key 对应的 value 是双向链表的结点。双向链表中存储 key-value 的 pair。双向链表表首更新缓存，表尾淘汰缓存。如下图：

提交代码以后，成功通过所有测试用例。

Success Details >

Runtime: 140 ms, faster than 20.72% of Go online submissions for LRU Cache.

Memory Usage: 17.9 MB, less than 20.93% of Go online submissions for LRU Cache.

Next challenges:

Design In-Memory File System | Design Compressed String Iterator

Show off your acceptance:

Time Submitted	Status	Runtime	Memory	Language
12/31/2020 15:41	Accepted	140 ms	17.9 MB	golang

解法二 Get O(1) / Put O(1)

数据结构上想不到其他解法了，但从打败的百分比上，看似还有常数的优化空间。笔者反复思考，觉得可能导致运行时间变长的地方是在 interface{} 类型推断，其他地方已无优化的空间。手写一个双向链表提交试试，代码如下：

```go
type LRUCache struct {
  head, tail *Node
  keys       map[int]*Node
  capacity   int
}

type Node struct {
  key, val   int
  prev, next *Node
}

func ConstructorLRU(capacity int) LRUCache {
  return LRUCache{keys: make(map[int]*Node), capacity: capacity}
}

func (this *LRUCache) Get(key int) int {
  if node, ok := this.keys[key]; ok {
    this.Remove(node)
    this.Add(node)
    return node.val
  }
  return -1
}
```

```go
func (this *LRUCache) Put(key int, value int) {
  if node, ok := this.keys[key]; ok {
    node.val = value
    this.Remove(node)
    this.Add(node)
    return
  } else {
    node = &Node{key: key, val: value}
    this.keys[key] = node
    this.Add(node)
  }
  if len(this.keys) > this.capacity {
    delete(this.keys, this.tail.key)
    this.Remove(this.tail)
  }
}

func (this *LRUCache) Add(node *Node) {
  node.prev = nil
  node.next = this.head
  if this.head != nil {
    this.head.prev = node
  }
  this.head = node
  if this.tail == nil {
    this.tail = node
    this.tail.next = nil
  }
}

func (this *LRUCache) Remove(node *Node) {
  if node == this.head {
    this.head = node.next
    if node.next != nil {
      node.next.prev = nil
    }
    node.next = nil
    return
  }
  if node == this.tail {
    this.tail = node.prev
    node.prev.next = nil
    node.prev = nil
    return
  }
  node.prev.next = node.next
  node.next.prev = node.prev
}
```

提交以后还真的 100% 了。

Success Details >

Runtime: 108 ms, faster than 100.00% of Go online submissions for LRU Cache.

Memory Usage: 17.1 MB, less than 56.64% of Go online submissions for LRU Cache.

Next challenges:

Design In-Memory File System | Design Compressed String Iterator

Show off your acceptance:

Time Submitted	Status	Runtime	Memory	Language
12/31/2020 16:01	Accepted	108 ms	17.1 MB	golang

上述代码实现的 LRU 本质并没有优化，只是换了一个写法，没有用 container 包而已。

模板

```go
type LRUCache struct {
  head, tail *Node
  Keys       map[int]*Node
  Cap        int
}

type Node struct {
  Key, Val   int
  Prev, Next *Node
}

func Constructor(capacity int) LRUCache {
  return LRUCache{Keys: make(map[int]*Node), Cap: capacity}
}

func (this *LRUCache) Get(key int) int {
  if node, ok := this.Keys[key]; ok {
    this.Remove(node)
    this.Add(node)
    return node.Val
  }
  return -1
```

```go
}

func (this *LRUCache) Put(key int, value int) {
  if node, ok := this.Keys[key]; ok {
    node.Val = value
    this.Remove(node)
    this.Add(node)
    return
  } else {
    node = &Node{Key: key, Val: value}
    this.Keys[key] = node
    this.Add(node)
  }
  if len(this.Keys) > this.Cap {
    delete(this.Keys, this.tail.Key)
    this.Remove(this.tail)
  }
}

func (this *LRUCache) Add(node *Node) {
  node.Prev = nil
  node.Next = this.head
  if this.head != nil {
    this.head.Prev = node
  }
  this.head = node
  if this.tail == nil {
    this.tail = node
    this.tail.Next = nil
  }
}

func (this *LRUCache) Remove(node *Node) {
  if node == this.head {
    this.head = node.Next
    node.Next = nil
    return
  }
  if node == this.tail {
    this.tail = node.Prev
    node.Prev.Next = nil
    node.Prev = nil
    return
  }
  node.Prev.Next = node.Next
  node.Next.Prev = node.Prev
}
```

最不经常最少使用 LFUCache

LFU 是 Least Frequently Used 的缩写，即最不经常最少使用，也是一种常用的页面置换算法，选择访问计数器最小的页面予以淘汰。如下图，缓存中每个页面带一个访问计数器。

根据 LFU 的策略，每访问一次都要更新访问计数器。当插入 B 的时候，发现缓存中有 B，所以增加访问计数器的计数，并把 B 移动到访问计数器从大到小排序的地方。再插入 D，同理先更新计数器，再移动到它排序以后的位置。当插入 F 的时候，缓存中不存在 F，所以淘汰计数器最小的页面的页面，所以淘汰 A 页面。此时 F 排在最下面，计数为 1。

这里有一个比 LRU 特别的地方。如果淘汰的页面访问次数有多个相同的访问次数，选择最靠尾部的。如上图中，A、B、C 三者的访问次数相同，都是 1 次。要插入 F，F 不在缓存中，此时要淘汰 A 页面。F 是新插入的页面，访问次数为 1，排在 C 的前面。也就是说相同的访问次数，按照新旧顺序排列，淘汰掉最旧的页面。这一点是和 LRU 最大的不同的地方。

可以发现，**LFU 更新和插入新页面可以发生在链表中任意位置，删除页面都发生在表尾。**

解法一 Get O(1) / Put O(1)

LFU 同样要求查询尽量高效，O(1) 内查询。依旧选用 map 查询。修改和删除也需要 O(1) 完成，依旧选用双向链表，继续复用 container 包中的 list 数据结构。LFU 需要记录访问次数，所以每个结点除了存储 key，value，需要再多存储 frequency 访问次数。

还有 1 个问题需要考虑，一个是如何按频次排序？相同频次，按照先后顺序排序。如果你开始考虑排序算法的话，思考方向就偏离最佳答案了。排序至少 O(nlogn)。重新回看 LFU 的工作原理，会发现它只关心最小频次。其他频次之间的顺序并不关心。所以不需要排序。用一个 min 变量保存最小频次，淘汰时读取这个最小值能找到要删除的结点。相同频次按照先后顺序排列，这个需求还是用双向链表实现，双向链表插入的顺序体现了结点的先后顺序。相同频次对应一个双向链表，可能有多个相同频次，所以可能有多个双向链表。用一个 map 维护访问频次和双向链表的对应关系。删除最小频次时，通过 min 找到最小频次，然后再这个 map 中找到这个频次对应的双向链表，在双向链表中找到最旧的那个结点删除。这就解决了 LFU 删除操作。

LFU 的更新操作和 LRU 类似，也需要用一个 map 保存 key 和双向链表结点的映射关系。这个双向链表结点中存储的是 key-value-frequency 三个元素的元组。这样通过结点中的 key 和 frequency 可以反过来删除 map 中的 key。

定义 LFUCache 的数据结构如下：

```
import "container/list"

type LFUCache struct {
  nodes     map[int]*list.Element
```

```go
  lists    map[int]*list.List
  capacity int
  min      int
}

type node struct {
  key       int
  value     int
  frequency int
}

func Constructor(capacity int) LFUCache {
  return LFUCache{nodes: make(map[int]*list.Element),
    lists:    make(map[int]*list.List),
    capacity: capacity,
    min:      0,
  }
}
```

LFUCache 的 Get 操作涉及更新 frequency 值和 2 个 map。在 nodes map 中通过 key 获取到结点信息。在 lists 删除结点当前 frequency 结点。删完以后 frequency ++。新的 frequency 如果在 lists 中存在，添加到双向链表表首，如果不存在，需要新建一个双向链表并把当前结点加到表首。再更新双向链表结点作为 value 的 map。最后更新 min 值，判断老的 frequency 对应的双向链表中是否已经为空，如果空了，min++。

```go
func (this *LFUCache) Get(key int) int {
  value, ok := this.nodes[key]
  if !ok {
    return -1
  }
  currentNode := value.Value.(*node)
  this.lists[currentNode.frequency].Remove(value)
  currentNode.frequency++
  if _, ok := this.lists[currentNode.frequency]; !ok {
    this.lists[currentNode.frequency] = list.New()
  }
  newList := this.lists[currentNode.frequency]
  newNode := newList.PushFront(currentNode)
  this.nodes[key] = newNode
  if currentNode.frequency-1 == this.min && this.lists[currentNode.frequency-1].Len()
== 0 {
    this.min++
  }
  return currentNode.value
}
```

LFU 的 Put 操作逻辑稍微多一点。先在 nodes map 中查询 key 是否存在，如果存在，获取这个结点，更新它的 value 值，然后手动调用一次 Get 操作，因为下面的更新逻辑和 Get 操作一致。如果 map 中不存在，接下来进行插入或者删除操作。判断 capacity 是否装满，如果装满，执行删除操作。在 min 对应的双向链表中删除表尾的结点，对应的也要删除 nodes map 中的键值。

由于新插入的页面访问次数一定为 1，所以 min 此时置为 1。新建结点，插入到 2 个 map 中。

```go
func (this *LFUCache) Put(key int, value int) {
  if this.capacity == 0 {
    return
  }
  // 如果存在，更新访问次数
  if currentValue, ok := this.nodes[key]; ok {
    currentNode := currentValue.Value.(*node)
    currentNode.value = value
    this.Get(key)
    return
  }
  // 如果不存在且缓存满了，需要删除
  if this.capacity == len(this.nodes) {
    currentList := this.lists[this.min]
    backNode := currentList.Back()
    delete(this.nodes, backNode.Value.(*node).key)
    currentList.Remove(backNode)
  }
  // 新建结点，插入到 2 个 map 中
  this.min = 1
  currentNode := &node{
    key:       key,
    value:     value,
    frequency: 1,
  }
  if _, ok := this.lists[1]; !ok {
    this.lists[1] = list.New()
  }
  newList := this.lists[1]
  newNode := newList.PushFront(currentNode)
  this.nodes[key] = newNode
}
```

总结，LFU 是由两个 map 和一个 min 指针组成的数据结构。一个 map 中 key 存的是访问次数，对应的 value 是一个个的双向链表，此处双向链表的作用是在相同频次的情况下，淘汰表尾最旧的那个页面。另一个 map 中 key 对应的 value 是双向链表的结点，结点中比 LRU 多存储了一个访问次数的值，即结点中存储 key-value-frequency 的元组。此处双向链表的作用和 LRU 是类似的，可以根据 map 中的 key 更新双向链表结点中的 value 和 frequency 的值，也可以根据双向链表结点中的 key 和 frequency 反向更新 map 中的对应关系。如下图：

提交代码以后，成功通过所有测试用例。

Success Details >

Runtime: 144 ms, faster than 22.03% of Go online submissions for LFU Cache.

Memory Usage: 17 MB, less than 59.32% of Go online submissions for LFU Cache.

Next challenges:

LRU Cache Design In-Memory File System

Show off your acceptance:

Time Submitted	Status	Runtime	Memory	Language
12/31/2020 16:11	Accepted	144 ms	17 MB	golang

解法二 Get O(capacity) / Put O(capacity)

LFU 的另外一个思路是利用 Index Priority Queue 这个数据结构。别被名字吓到，Index Priority Queue = map + Priority Queue，仅此而已。

利用 Priority Queue 维护一个最小堆，堆顶是访问次数最小的元素。map 中的 value 存储的是优先队列中结点。

```
import "container/heap"

type LFUCache struct {
  capacity int
  pq       PriorityQueue
  hash     map[int]*Item
  counter  int
}

func Constructor(capacity int) LFUCache {
  lfu := LFUCache{
    pq:       PriorityQueue{},
    hash:     make(map[int]*Item, capacity),
    capacity: capacity,
  }
  return lfu
}
```

Get 和 Put 操作要尽量的快，有 2 个问题需要解决。当访问次数相同时，如何删除掉最久的元素？当元素的访问次数发生变化时，如何快速调整堆？为了解决这 2 个问题，定义如下的数据结构：

```go
// An Item is something we manage in a priority queue.
type Item struct {
  value     int // The value of the item; arbitrary.
  key       int
  frequency int // The priority of the item in the queue.
  count     int // use for evicting the oldest element
  // The index is needed by update and is maintained by the heap.Interface methods.
  index int // The index of the item in the heap.
}
```

堆中的结点存储这 5 个值。count 值用来决定哪个是最老的元素，类似一个操作时间戳。index 值用来 re-heapify 调整堆的。接下来实现 PriorityQueue 的方法。

```go
// A PriorityQueue implements heap.Interface and holds Items.
type PriorityQueue []*Item

func (pq PriorityQueue) Len() int { return len(pq) }

func (pq PriorityQueue) Less(i, j int) bool {
  // We want Pop to give us the highest, not lowest, priority so we use greater than
here.
  if pq[i].frequency == pq[j].frequency {
    return pq[i].count < pq[j].count
  }
  return pq[i].frequency < pq[j].frequency
}

func (pq PriorityQueue) Swap(i, j int) {
  pq[i], pq[j] = pq[j], pq[i]
  pq[i].index = i
  pq[j].index = j
}

func (pq *PriorityQueue) Push(x interface{}) {
  n := len(*pq)
  item := x.(*Item)
  item.index = n
  *pq = append(*pq, item)
}

func (pq *PriorityQueue) Pop() interface{} {
  old := *pq
  n := len(old)
  item := old[n-1]
  old[n-1] = nil  // avoid memory leak
  item.index = -1 // for safety
  *pq = old[0 : n-1]
```

```
  return item
}

// update modifies the priority and value of an Item in the queue.
func (pq *PriorityQueue) update(item *Item, value int, frequency int, count int) {
  item.value = value
  item.count = count
  item.frequency = frequency
  heap.Fix(pq, item.index)
}
```

在 Less() 方法中，frequency 从小到大排序，frequency 相同的，按 count 从小到大排序。按照优先队列建堆规则，可以得到，frequency 最小的在堆顶，相同的 frequency，count 最小的越靠近堆顶。

在 Swap() 方法中，记得要更新 index 值。在 Push() 方法中，插入时队列的长度即是该元素的 index 值，此处也要记得更新 index 值。update() 方法调用 Fix() 函数。Fix() 函数比先 Remove() 再 Push() 一个新的值，花销要小。所以此处调用 Fix() 函数，这个操作的时间复杂度是 O(log n)。

这样就维护了最小 Index Priority Queue。Get 操作非常简单：

```
func (this *LFUCache) Get(key int) int {
  if this.capacity == 0 {
    return -1
  }
  if item, ok := this.hash[key]; ok {
    this.counter++
    this.pq.update(item, item.value, item.frequency+1, this.counter)
    return item.value
  }
  return -1
}
```

在 hashmap 中查询 key，如果存在，counter 时间戳累加，调用 Priority Queue 的 update 方法，调整堆。

```
func (this *LFUCache) Put(key int, value int) {
  if this.capacity == 0 {
    return
  }
  this.counter++
  // 如果存在，增加 frequency，再调整堆
  if item, ok := this.hash[key]; ok {
    this.pq.update(item, value, item.frequency+1, this.counter)
    return
  }
  // 如果不存在且缓存满了，需要删除。在 hashmap 和 pq 中删除。
  if len(this.pq) == this.capacity {
    item := heap.Pop(&this.pq).(*Item)
    delete(this.hash, item.key)
```

```go
  }
  // 新建结点，在 hashmap 和 pq 中添加。
  item := &Item{
    value: value,
    key:   key,
    count: this.counter,
  }
  heap.Push(&this.pq, item)
  this.hash[key] = item
}
```

用最小堆实现的 LFU，Put 时间复杂度是 O(capacity)，Get 时间复杂度是 O(capacity)，不及 2 个 map 实现的版本。巧的是最小堆的版本居然打败了 100%。

Success Details ›

Runtime: 112 ms, faster than 100.00% of Go online submissions for LFU Cache.

Memory Usage: 17.5 MB, less than 52.54% of Go online submissions for LFU Cache.

Next challenges:

LRU Cache Design In-Memory File System

Show off your acceptance:

Time Submitted	Status	Runtime	Memory	Language
12/31/2020 15:53	Accepted	112 ms	17.5 MB	golang

模板

```go
import "container/list"

type LFUCache struct {
  nodes    map[int]*list.Element
  lists    map[int]*list.List
  capacity int
  min      int
}

type node struct {
  key       int
  value     int
  frequency int
```

```go
}

func Constructor(capacity int) LFUCache {
  return LFUCache{nodes: make(map[int]*list.Element),
    lists:    make(map[int]*list.List),
    capacity: capacity,
    min:      0,
  }
}

func (this *LFUCache) Get(key int) int {
  value, ok := this.nodes[key]
  if !ok {
    return -1
  }
  currentNode := value.Value.(*node)
  this.lists[currentNode.frequency].Remove(value)
  currentNode.frequency++
  if _, ok := this.lists[currentNode.frequency]; !ok {
    this.lists[currentNode.frequency] = list.New()
  }
  newList := this.lists[currentNode.frequency]
  newNode := newList.PushBack(currentNode)
  this.nodes[key] = newNode
  if currentNode.frequency-1 == this.min && this.lists[currentNode.frequency-1].Len()
== 0 {
    this.min++
  }
  return currentNode.value
}

func (this *LFUCache) Put(key int, value int) {
  if this.capacity == 0 {
    return
  }
  if currentValue, ok := this.nodes[key]; ok {
    currentNode := currentValue.Value.(*node)
    currentNode.value = value
    this.Get(key)
    return
  }
  if this.capacity == len(this.nodes) {
    currentList := this.lists[this.min]
    frontNode := currentList.Front()
    delete(this.nodes, frontNode.Value.(*node).key)
    currentList.Remove(frontNode)
  }
  this.min = 1
  currentNode := &node{
```

```
    key:       key,
    value:     value,
    frequency: 1,
  }
  if _, ok := this.lists[1]; !ok {
    this.lists[1] = list.New()
  }
  newList := this.lists[1]
  newNode := newList.PushBack(currentNode)
  this.nodes[key] = newNode
}
```

第四章 LeetCode 题解

这一章就是 LeetCode 的题解了。笔者目前只刷到 608 题，题解这里有 520 题，都已经 runtime beats 100% 了。相差的 88 题是还没有 beats 100% 的，笔者还需要继续优化~

题解慢慢更新中，欢迎大家提出更好的解法。点击页面下方的 edit，会跳转到 github 对应的页面 markdown 中，可以提交你的最优解 PR。

让我们在题解的太空遨游吧~

1. Two Sum

题目

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]
```

题目大意

在数组中找到 2 个数之和等于给定值的数字，结果返回 2 个数字在数组中的下标。

解题思路

这道题最优的做法时间复杂度是 O(n)。

顺序扫描数组，对每一个元素，在 map 中找能组合给定值的另一半数字，如果找到了，直接返回 2 个数字的下标即可。如果找不到，就把这个数字存入 map 中，等待扫到“另一半”数字的时候，再取出来返回结果。

代码

```go
package leetcode

func twoSum(nums []int, target int) []int {
  m := make(map[int]int)
  for i := 0; i < len(nums); i++ {
    another := target - nums[i]
    if _, ok := m[another]; ok {
      return []int{m[another], i}
    }
    m[nums[i]] = i
  }
  return nil
}
```

2. Add Two Numbers

题目

You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Example:

```
Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)
Output: 7 -> 0 -> 8
Explanation: 342 + 465 = 807.
```

题目大意

2 个逆序的链表，要求从低位开始相加，得出结果也逆序输出，返回值是逆序结果链表的头结点。

解题思路

需要注意的是各种进位问题。

极端情况，例如

```
Input: (9 -> 9 -> 9 -> 9 -> 9) + (1 -> )
Output: 0 -> 0 -> 0 -> 0 -> 0 -> 1
```

为了处理方法统一，可以先建立一个虚拟头结点，这个虚拟头结点的 Next 指向真正的 head，这样 head 不需要单独处理，直接 while 循环即可。另外判断循环终止的条件不用是 p.Next！ = nil，这样最后一位还需要额外计算，循环终止条件应该是 p != nil。

代码

```go
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */
```

```go
func addTwoNumbers(l1 *ListNode, l2 *ListNode) *ListNode {
  head := &ListNode{Val: 0}
  n1, n2, carry, current := 0, 0, 0, head
  for l1 != nil || l2 != nil || carry != 0 {
    if l1 == nil {
      n1 = 0
    } else {
      n1 = l1.Val
      l1 = l1.Next
    }
    if l2 == nil {
      n2 = 0
    } else {
      n2 = l2.Val
      l2 = l2.Next
    }
    current.Next = &ListNode{Val: (n1 + n2 + carry) % 10}
    current = current.Next
    carry = (n1 + n2 + carry) / 10
  }
  return head.Next
}
```

3. Longest Substring Without Repeating Characters

题目

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.
```

题目大意

在一个字符串重寻找没有重复字母的最长子串。

解题思路

这一题和第 438 题，第 3 题，第 76 题，第 567 题类似，用的思想都是"滑动窗口"。

滑动窗口的右边界不断的右移，只要没有重复的字符，就持续向右扩大窗口边界。一旦出现了重复字符，就需要缩小左边界，直到重复的字符移出了左边界，然后继续移动滑动窗口的右边界。以此类推，每次移动需要计算当前长度，并判断是否需要更新最大长度，最终最大的值就是题目中的所求。

代码

```go
package leetcode

// 解法一 位图
func lengthOfLongestSubstring(s string) int {
  if len(s) == 0 {
    return 0
  }
  var bitSet [256]bool
  result, left, right := 0, 0, 0
  for left < len(s) {
    // 右侧字符对应的 bitSet 被标记 true，说明此字符在 X 位置重复，需要左侧向前移动，直到将 X 标记为
false
    if bitSet[s[right]] {
      bitSet[s[left]] = false
      left++
    } else {
      bitSet[s[right]] = true
```

```go
      right++
    }
    if result < right-left {
      result = right - left
    }
    if left+result >= len(s) || right >= len(s) {
      break
    }
  }
  return result
}

// 解法二 滑动窗口
func lengthOfLongestSubstring1(s string) int {
  if len(s) == 0 {
    return 0
  }
  var freq [256]int
  result, left, right := 0, 0, -1

  for left < len(s) {
    if right+1 < len(s) && freq[s[right+1]-'a'] == 0 {
      freq[s[right+1]-'a']++
      right++
    } else {
      freq[s[left]-'a']--
      left++
    }
    result = max(result, right-left+1)
  }
  return result
}

// 解法三 滑动窗口-哈希桶
func lengthOfLongestSubstring2(s string) int {
  right, left, res := 0, 0, 0
  indexes := make(map[byte]int, len(s))
  for left < len(s) {
    if idx, ok := indexes[s[left]]; ok && idx >= right {
      right = idx + 1
    }
    indexes[s[left]] = left
    left++
    res = max(res, left-right)
  }
  return res
}

func max(a int, b int) int {
```

```
  if a > b {
    return a
  }
  return b
}
```

4. Median of Two Sorted Arrays

题目

There are two sorted arrays **nums1** and **nums2** of size m and n respectively.

Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

You may assume **nums1** and **nums2** cannot be both empty.

Example 1:

```
nums1 = [1, 3]
nums2 = [2]

The median is 2.0
```

Example 2:

```
nums1 = [1, 2]
nums2 = [3, 4]

The median is (2 + 3)/2 = 2.5
```

题目大意

给定两个大小为 m 和 n 的有序数组 nums1 和 nums2。

请你找出这两个有序数组的中位数，并且要求算法的时间复杂度为 O(log(m + n))。

你可以假设 nums1 和 nums2 不会同时为空。

解题思路

- 给出两个有序数组，要求找出这两个数组合并以后的有序数组中的中位数。要求时间复杂度为 O(log (m+n))。
- 这一题最容易想到的办法是把两个数组合并，然后取出中位数。但是合并有序数组的操作是 `O(m+n)` 的，不符合题意。看到题目给的 `log` 的时间复杂度，很容易联想到二分搜索。
- 由于要找到最终合并以后数组的中位数，两个数组的总大小也知道，所以中间这个位置也是知道的。只需要二分搜索一个数组中切分的位置，另一个数组中切分的位置也能得到。为了使得时间复杂度最小，所以二分搜索两个数组中长度较小的那个数组。
- 关键的问题是如何切分数组 1 和数组 2 。其实就是如何切分数组 1 。先随便二分产生一个 `midA`，切分的线何时算满足了中位数的条件呢？即，线左边的数都小于右边的数，即，`nums1[midA-1] ≤ nums2[midB] && nums2[midB-1] ≤ nums1[midA]` 。如果这些条件都不满足，切分线就需要调整。如果 `nums1[midA] < nums2[midB-1]`，说明 `midA` 这条线划分出来左边的数小了，切分线应该右移；如果 `nums1[midA-1] > nums2[midB]`，说明 midA 这条线划分出来左边的数大了，切分线应该左移。经过多次调整以后，切分线总能找到满足条件的解。
- 假设现在找到了切分的两条线了，`数组 1` 在切分线两边的下标分别是 `midA - 1` 和 `midA`。`数组 2` 在切分线两边的下标分别是 `midB - 1` 和 `midB`。最终合并成最终数组，如果数组长度是奇数，那么中位数就是 `max(nums1[midA-1], nums2[midB-1])`。如果数组长度是偶数，那么中间位置的两个数依次是：`max(nums1[midA-1], nums2[midB-1])` 和 `min(nums1[midA], nums2[midB])`，那么中位数就是 `(max(nums1[midA-1], nums2[midB-1]) + min(nums1[midA], nums2[midB])) / 2`。图示见下图：

代码

```
package leetcode

func findMedianSortedArrays(nums1 []int, nums2 []int) float64 {
  // 假设 nums1 的长度小
  if len(nums1) > len(nums2) {
```

```
    return findMedianSortedArrays(nums2, nums1)
  }
  low, high, k, nums1Mid, nums2Mid := 0, len(nums1), (len(nums1)+len(nums2)+1)>>1, 0, 0
  for low <= high {
    // nums1:  ................ nums1[nums1Mid-1] | nums1[nums1Mid] ......................
    // nums2:  ................ nums2[nums2Mid-1] | nums2[nums2Mid] ......................
    nums1Mid = low + (high-low)>>1 // 分界限右侧是 mid，分界线左侧是 mid - 1
    nums2Mid = k - nums1Mid
    if nums1Mid > 0 && nums1[nums1Mid-1] > nums2[nums2Mid] { // nums1 中的分界线划多了，要向左边移动
      high = nums1Mid - 1
    } else if nums1Mid != len(nums1) && nums1[nums1Mid] < nums2[nums2Mid-1] { // nums1 中的分界线划少了，要向右边移动
      low = nums1Mid + 1
    } else {
      // 找到合适的划分了，需要输出最终结果了
      // 分为奇数偶数 2 种情况
      break
    }
  }
  midLeft, midRight := 0, 0
  if nums1Mid == 0 {
    midLeft = nums2[nums2Mid-1]
  } else if nums2Mid == 0 {
    midLeft = nums1[nums1Mid-1]
  } else {
    midLeft = max(nums1[nums1Mid-1], nums2[nums2Mid-1])
  }
  if (len(nums1)+len(nums2))&1 == 1 {
    return float64(midLeft)
  }
  if nums1Mid == len(nums1) {
    midRight = nums2[nums2Mid]
  } else if nums2Mid == len(nums2) {
    midRight = nums1[nums1Mid]
  } else {
    midRight = min(nums1[nums1Mid], nums2[nums2Mid])
  }
  return float64(midLeft+midRight) / 2
}
```

5. Longest Palindromic Substring

题目

Given a string `s`, return *the longest palindromic substring* in `s`.

Example 1:

```
Input: s = "babad"
Output: "bab"
Note: "aba" is also a valid answer.
```

Example 2:

```
Input: s = "cbbd"
Output: "bb"
```

Example 3:

```
Input: s = "a"
Output: "a"
```

Example 4:

```
Input: s = "ac"
Output: "a"
```

Constraints:

- `1 <= s.length <= 1000`
- `s` consist of only digits and English letters (lower-case and/or upper-case),

题目大意

给你一个字符串 `s`，找到 `s` 中最长的回文子串。

解题思路

- 此题非常经典，并且有多种解法。
- 解法一，动态规划。定义 `dp[i][j]` 表示从字符串第 `i` 个字符到第 `j` 个字符这一段子串是否是回文串。由回文串的性质可以得知，回文串去掉一头一尾相同的字符以后，剩下的还是回文串。所以状态转移方程是 `dp[i][j] = (s[i] == s[j]) && ((j-i < 3) || dp[i+1][j-1])`，注意特殊的情况，`j - i == 1` 的时候，即只有 2 个字符的情况，只需要判断这 2 个字符是否相同即可。`j - i == 2` 的时候，即只有 3 个字符的情况，只需要判断除去中心以外对称的 2 个字符是否相等。每次循环动态维护保存最长回文串即可。时间复杂度 O(n^2)，空间复杂度 O(n^2)。
- 解法二，中心扩散法。动态规划的方法中，我们将任意起始，终止范围内的字符串都判断了一遍。其实没有这个必要，如果不是最长回文串，无需判断并保存结果。所以动态规划的方法在空间复杂度上还有优化空间。判断回文有一个核心问题是找到“轴心”。如果长度是偶数，那么轴心是中心虚拟的，如果长度是奇数，那么轴心正好是正中心的那个字母。中心扩散法的思想是枚举每个轴心的位置。然后做两次假设，假设最长回文串是偶数，那么以虚拟中心往 2 边扩散；假设最长回文串是奇数，那么以正中心的字符往 2 边扩散。扩散的过程就

是对称判断两边字符是否相等的过程。这个方法时间复杂度和动态规划是一样的，但是空间复杂度降低了。时间复杂度 O(n^2)，空间复杂度 O(1)。

- 解法三，滑动窗口。这个写法其实就是中心扩散法变了一个写法。中心扩散是依次枚举每一个轴心。滑动窗口的方法稍微优化了一点，有些轴心两边字符不相等，下次就不会枚举这些不可能形成回文子串的轴心了。不过这点优化并没有优化时间复杂度，时间复杂度 O(n^2)，空间复杂度 O(1)。
- 解法四，马拉车算法。这个算法是本题的最优解，也是最复杂的解法。时间复杂度 O(n)，空间复杂度 O(n)。中心扩散法有 2 处有重复判断，第一处是每次都往两边扩散，不同中心扩散多次，实际上有很多重复判断的字符，能否不重复判断？第二处，中心能否跳跃选择，不是每次都枚举，是否可以利用前一次的信息，跳跃选择下一次的中心？马拉车算法针对重复判断的问题做了优化，增加了一个辅助数组，将时间复杂度从 O(n^2) 优化到了 O(n)，空间换了时间，空间复杂度增加到 O(n)。

- 首先是预处理，向字符串的头尾以及每两个字符中间添加一个特殊字符 `#`，比如字符串 `aaba` 处理后会变成 `#a#a#b#a#`。那么原先长度为偶数的回文字符串 `aa` 会变成长度为奇数的回文字符串 `#a#a#`，而长度为奇数的回文字符串 `aba` 会变成长度仍然为奇数的回文字符串 `#a#b#a#`，经过预处理以后，都会变成长度为奇数的字符串。**注意这里的特殊字符不需要是没有出现过的字母，也可以使用任何一个字符来作为这个特殊字符。**这是因为，当我们只考虑长度为奇数的回文字符串时，每次我们比较的两个字符奇偶性一定是相同的，所以原来字符串中的字符不会与插入的特殊字符互相比较，不会因此产生问题。**预处理以后，以某个中心扩散的步数和实际字符串长度是相等的。**因为半径里面包含了插入的特殊字符，又由于左右对称的性质，所以扩散半径就等于原来回文子串的长度。

- 核心部分是如何通过左边已经扫描过的数据推出右边下一次要扩散的中心。这里定义下一次要扩散的中心下标是 `i`。如果 `i` 比 `maxRight` 要小，只能继续中心扩散。如果 `i` 比 `maxRight` 大，这是又分为 3 种情况。三种情况见上图。将上述 3 种情况总结起来，就是：`dp[i] = min(maxRight-i, dp[2*center-i])`，其中，`mirror` 相对于 `center` 是和 `i` 中心对称的，所以它的下标可以计算出来是 `2*center-i`。更新完 `dp[i]` 以后，就要进行中心扩散了。中心扩散以后动态维护最长回文串并相应的更新 `center`，`maxRight`，并且记录下原始字符串的起始位置 `begin` 和 `maxLen`。

代码

```
package leetcode

// 解法一 Manacher's algorithm，时间复杂度 O(n)，空间复杂度 O(n)
func longestPalindrome(s string) string {
  if len(s) < 2 {
    return s
  }
  newS := make([]rune, 0)
  newS = append(newS, '#')
  for _, c := range s {
    newS = append(newS, c)
    newS = append(newS, '#')
  }
  // dp[i]:     以预处理字符串下标 i 为中心的回文半径(奇数长度时不包括中心)
  // maxRight: 通过中心扩散的方式能够扩散的最右边的下标
```

```go
  // center:   与 maxRight 对应的中心字符的下标
  // maxLen:   记录最长回文串的半径
  // begin:    记录最长回文串在起始串 s 中的起始下标
  dp, maxRight, center, maxLen, begin := make([]int, len(newS)), 0, 0, 1, 0
  for i := 0; i < len(newS); i++ {
    if i < maxRight {
      // 这一行代码是 Manacher 算法的关键所在
      dp[i] = min(maxRight-i, dp[2*center-i])
    }
    // 中心扩散法更新 dp[i]
    left, right := i-(1+dp[i]), i+(1+dp[i])
    for left >= 0 && right < len(newS) && newS[left] == newS[right] {
      dp[i]++
      left--
      right++
    }
    // 更新 maxRight，它是遍历过的 i 的 i + dp[i] 的最大者
    if i+dp[i] > maxRight {
      maxRight = i + dp[i]
      center = i
    }
    // 记录最长回文子串的长度和相应它在原始字符串中的起点
    if dp[i] > maxLen {
      maxLen = dp[i]
      begin = (i - maxLen) / 2 // 这里要除以 2 因为有我们插入的辅助字符 #
    }
  }
  return s[begin : begin+maxLen]
}

func min(x, y int) int {
  if x < y {
    return x
  }
  return y
}

// 解法二 滑动窗口，时间复杂度 O(n^2)，空间复杂度 O(1)
func longestPalindrome1(s string) string {
  if len(s) == 0 {
    return ""
  }
  left, right, pl, pr := 0, -1, 0, 0
  for left < len(s) {
    // 移动到相同字母的最右边（如果有相同字母）
    for right+1 < len(s) && s[left] == s[right+1] {
      right++
    }
    // 找到回文的边界
```

```go
    for left-1 >= 0 && right+1 < len(s) && s[left-1] == s[right+1] {
      left--
      right++
    }
    if right-left > pr-pl {
      pl, pr = left, right
    }
    // 重置到下一次寻找回文的中心
    left = (left+right)/2 + 1
    right = left
  }
  return s[pl : pr+1]
}

// 解法三 中心扩散法，时间复杂度 O(n^2)，空间复杂度 O(1)
func longestPalindrome2(s string) string {
  res := ""
  for i := 0; i < len(s); i++ {
    res = maxPalindrome(s, i, i, res)
    res = maxPalindrome(s, i, i+1, res)
  }
  return res
}

func maxPalindrome(s string, i, j int, res string) string {
  sub := ""
  for i >= 0 && j < len(s) && s[i] == s[j] {
    sub = s[i : j+1]
    i--
    j++
  }
  if len(res) < len(sub) {
    return sub
  }
  return res
}

// 解法四 DP，时间复杂度 O(n^2)，空间复杂度 O(n^2)
func longestPalindrome3(s string) string {
  res, dp := "", make([][]bool, len(s))
  for i := 0; i < len(s); i++ {
    dp[i] = make([]bool, len(s))
  }
  for i := len(s) - 1; i >= 0; i-- {
    for j := i; j < len(s); j++ {
      dp[i][j] = (s[i] == s[j]) && ((j-i < 3) || dp[i+1][j-1])
      if dp[i][j] && (res == "" || j-i+1 > len(res)) {
        res = s[i : j+1]
      }
```

```
    }
  }
  return res
}
```

6. ZigZag Conversion

题目

The string `"PAYPALISHIRING"` is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility)

```
P   A   H   N
A P L S I I G
Y   I   R
```

And then read line by line: `"PAHNAPLSIIGYIR"`

Write the code that will take a string and make this conversion given a number of rows:

```
string convert(string s, int numRows);
```

Example 1:

```
Input: s = "PAYPALISHIRING", numRows = 3
Output: "PAHNAPLSIIGYIR"
```

Example 2:

```
Input: s = "PAYPALISHIRING", numRows = 4
Output: "PINALSIGYAHRPI"
Explanation:
P     I    N
A   L S  I G
Y A   H R
P     I
```

Example 3:

```
Input: s = "A", numRows = 1
Output: "A"
```

Constraints:

- `1 <= s.length <= 1000`
- `s` consists of English letters (lower-case and upper-case), `','` and `'.'`.

- `1 <= numRows <= 1000`

题目大意

将一个给定字符串 `s` 根据给定的行数 `numRows` ，以从上往下、从左到右进行 Z 字形排列。

比如输入字符串为 `"PAYPALISHIRING"` 行数为 3 时，排列如下：

```
P   A   H   N
A P L S I I G
Y   I   R
```

之后，你的输出需要从左往右逐行读取，产生出一个新的字符串，比如：`"PAHNAPLSIIGYIR"`。

请你实现这个将字符串进行指定行数变换的函数：

```
string convert(string s, int numRows);
```

解题思路

- 这一题没有什么算法思想，考察的是对程序控制的能力。用 2 个变量保存方向，当垂直输出的行数达到了规定的目标行数以后，需要从下往上转折到第一行，循环中控制好方向ji

代码

```
package leetcode

func convert(s string, numRows int) string {
  matrix, down, up := make([][]byte, numRows, numRows), 0, numRows-2
  for i := 0; i != len(s); {
    if down != numRows {
      matrix[down] = append(matrix[down], byte(s[i]))
      down++
      i++
    } else if up > 0 {
      matrix[up] = append(matrix[up], byte(s[i]))
      up--
      i++
    } else {
      up = numRows - 2
      down = 0
    }
  }
  solution := make([]byte, 0, len(s))
  for _, row := range matrix {
    for _, item := range row {
      solution = append(solution, item)
    }
```

```
    }
    return string(solution)
}
```

7. Reverse Integer

题目

Given a 32-bit signed integer, reverse digits of an integer.

Example 1:

```
Input: 123
Output: 321
```

Example 2:

```
Input: -123
Output: -321
```

Example 3:

```
Input: 120
Output: 21
```

Note: Assume we are dealing with an environment which could only store integers within the 32-bit signed integer range: [−2^31, 2^31 − 1]. For the purpose of this problem, assume that your function returns 0 when the reversed integer overflows.

题目大意

给出一个 32 位的有符号整数，你需要将这个整数中每位上的数字进行反转。注意:假设我们的环境只能存储得下 32 位的有符号整数，则其数值范围为 [−2^31，2^31 − 1]。请根据这个假设，如果反转后整数溢出那么就返回 0。

解题思路

- 这一题是简单题，要求反转 10 进制数。类似的题目有第 190 题。
- 这一题只需要注意一点，反转以后的数字要求在 [−2^31, 2^31 − 1]范围内，超过这个范围的数字都要输出 0 。

代码

```
package leetcode
```

```go
func reverse7(x int) int {
  tmp := 0
  for x != 0 {
    tmp = tmp*10 + x%10
    x = x / 10
  }
  if tmp > 1<<31-1 || tmp < -(1<<31) {
    return 0
  }
  return tmp
}
```

8. String to Integer (atoi)

题目

Implement the `myAtoi(string s)` function, which converts a string to a 32-bit signed integer (similar to C/C++'s `atoi` function).

The algorithm for `myAtoi(string s)` is as follows:

1. Read in and ignore any leading whitespace.
2. Check if the next character (if not already at the end of the string) is `'-'` or `'+'`. Read this character in if it is either. This determines if the final result is negative or positive respectively. Assume the result is positive if neither is present.
3. Read in next the characters until the next non-digit charcter or the end of the input is reached. The rest of the string is ignored.
4. Convert these digits into an integer (i.e. `"123" -> 123`, `"0032" -> 32`). If no digits were read, then the integer is `0`. Change the sign as necessary (from step 2).
5. If the integer is out of the 32-bit signed integer range `[-231, 231 - 1]`, then clamp the integer so that it remains in the range. Specifically, integers less than `231` should be clamped to `231`, and integers greater than `231 - 1` should be clamped to `231 - 1`.
6. Return the integer as the final result.

Note:

- Only the space character `' '` is considered a whitespace character.
- **Do not ignore** any characters other than the leading whitespace or the rest of the string after the digits.

Example 1:

```
Input: s = "42"
Output: 42
Explanation: The underlined characters are what is read in, the caret is the current
reader position.
Step 1: "42" (no characters read because there is no leading whitespace)
         ^
Step 2: "42" (no characters read because there is neither a '-' nor '+')
         ^
Step 3: "42" ("42" is read in)
           ^
The parsed integer is 42.
Since 42 is in the range [-231, 231 - 1], the final result is 42.
```

Example 2:

```
Input: s = "   -42"
Output: -42
Explanation:
Step 1: "   -42" (leading whitespace is read and ignored)
            ^
Step 2: "   -42" ('-' is read, so the result should be negative)
             ^
Step 3: "   -42" ("42" is read in)
               ^
The parsed integer is -42.
Since -42 is in the range [-231, 231 - 1], the final result is -42.
```

Example 3:

```
Input: s = "4193 with words"
Output: 4193
Explanation:
Step 1: "4193 with words" (no characters read because there is no leading whitespace)
         ^
Step 2: "4193 with words" (no characters read because there is neither a '-' nor '+')
         ^
Step 3: "4193 with words" ("4193" is read in; reading stops because the next character
is a non-digit)
             ^
The parsed integer is 4193.
Since 4193 is in the range [-231, 231 - 1], the final result is 4193.
```

Example 4:

```
Input: s = "words and 987"
Output: 0
Explanation:
Step 1: "words and 987" (no characters read because there is no leading whitespace)
         ^
Step 2: "words and 987" (no characters read because there is neither a '-' nor '+')
         ^
Step 3: "words and 987" (reading stops immediately because there is a non-digit 'w')
         ^
The parsed integer is 0 because no digits were read.
Since 0 is in the range [-231, 231 - 1], the final result is 0.
```

Example 5:

```
Input: s = "-91283472332"
Output: -2147483648
Explanation:
Step 1: "-91283472332" (no characters read because there is no leading whitespace)
         ^
Step 2: "-91283472332" ('-' is read, so the result should be negative)
          ^
Step 3: "-91283472332" ("91283472332" is read in)
                      ^
The parsed integer is -91283472332.
Since -91283472332 is less than the lower bound of the range [-231, 231 - 1], the final
result is clamped to -231 = -2147483648.
```

Constraints:

- `0 <= s.length <= 200`
- `s` consists of English letters (lower-case and upper-case), digits (`0-9`), `' '`, `'+'`

题目大意

请你来实现一个 myAtoi(string s) 函数，使其能将字符串转换成一个 32 位有符号整数（类似 C/C++ 中的 atoi 函数）。

函数 myAtoi(string s) 的算法如下：

- 读入字符串并丢弃无用的前导空格
- 检查下一个字符（假设还未到字符末尾）为正还是负号，读取该字符（如果有）。 确定最终结果是负数还是正数。 如果两者都不存在，则假定结果为正。
- 读入下一个字符，直到到达下一个非数字字符或到达输入的结尾。字符串的其余部分将被忽略。
- 将前面步骤读入的这些数字转换为整数（即，"123" -> 123， "0032" -> 32）。如果没有读入数字，则整数为 0 。必要时更改符号（从步骤 2 开始）。
- 如果整数数超过 32 位有符号整数范围 [−231，231 − 1] ，需要截断这个整数，使其保持在这个范围内。具体来说，小于 −231 的整数应该被固定为 −231 ，大于 231 − 1 的整数应该被固定为 231 − 1 。
- 返回整数作为最终结果。

注意：

- 本题中的空白字符只包括空格字符 `' '` 。
- 除前导空格或数字后的其余字符串外，请勿忽略 任何其他字符。

解题思路

- 这题是简单题。题目要求实现类似 `C++` 中 `atoi` 函数的功能。这个函数功能是将字符串类型的数字转成 `int` 类型数字。先去除字符串中的前导空格，并判断记录数字的符号。数字需要去掉前导 `0` 。最后将数字转换成数字类型，判断是否超过 `int` 类型的上限 `[-2^31, 2^31 - 1]`，如果超过上限，需要输出边界，即 `-2^31`，或者 `2^31 - 1`。

代码

```
package leetcode

func myAtoi(s string) int {
  maxInt, signAllowed, whitespaceAllowed, sign, digits := int64(2<<30), true, true, 1,
[]int{}
  for _, c := range s {
    if c == ' ' && whitespaceAllowed {
      continue
    }
    if signAllowed {
      if c == '+' {
        signAllowed = false
        whitespaceAllowed = false
        continue
      } else if c == '-' {
        sign = -1
        signAllowed = false
        whitespaceAllowed = false
        continue
      }
    }
    if c < '0' || c > '9' {
      break
    }
    whitespaceAllowed, signAllowed = false, false
    digits = append(digits, int(c-48))
  }
  var num, place int64
  place, num = 1, 0
  lastLeadingOIndex := -1
  for i, d := range digits {
    if d == 0 {
      lastLeadingOIndex = i
    } else {
      break
```

```
    }
  }
  if lastLeading0Index > -1 {
    digits = digits[lastLeading0Index+1:]
  }
  var rtnMax int64
  if sign > 0 {
    rtnMax = maxInt - 1
  } else {
    rtnMax = maxInt
  }
  digitsCount := len(digits)
  for i := digitsCount - 1; i >= 0; i-- {
    num += int64(digits[i]) * place
    place *= 10
    if digitsCount-i > 10 || num > rtnMax {
      return int(int64(sign) * rtnMax)
    }
  }
  num *= int64(sign)
  return int(num)
}
```

9. Palindrome Number

题目

Determine whether an integer is a palindrome. An integer is a palindrome when it reads the same backward as forward.

Example 1:

```
Input: 121
Output: true
```

Example 2:

```
Input: -121
Output: false
Explanation: From left to right, it reads -121. From right to left, it becomes 121-.
Therefore it is not a palindrome.
```

Example 3:

```
Input: 10
Output: false
Explanation: Reads 01 from right to left. Therefore it is not a palindrome.
```

Follow up:

Coud you solve it without converting the integer to a string?

题目大意

判断一个整数是否是回文数。回文数是指正序（从左向右）和倒序（从右向左）读都是一样的整数。

解题思路

- 判断一个整数是不是回文数。
- 简单题。注意会有负数的情况，负数，个位数，10 都不是回文数。其他的整数再按照回文的规则判断。

代码

```go
package leetcode

import "strconv"

// 解法一
func isPalindrome(x int) bool {
  if x < 0 {
    return false
  }
  if x == 0 {
    return true
  }
  if x%10 == 0 {
    return false
  }
  arr := make([]int, 0, 32)
  for x > 0 {
    arr = append(arr, x%10)
    x = x / 10
  }
  sz := len(arr)
  for i, j := 0, sz-1; i <= j; i, j = i+1, j-1 {
    if arr[i] != arr[j] {
      return false
    }
  }
  return true
}

// 解法二 数字转字符串
func isPalindrome1(x int) bool {
  if x < 0 {
    return false
```

```
  }
  if x < 10 {
    return true
  }
  s := strconv.Itoa(x)
  length := len(s)
  for i := 0; i <= length/2; i++ {
    if s[i] != s[length-1-i] {
      return false
    }
  }
  return true
}
```

11. Container With Most Water

题目

Given n non-negative integers a1, a2, ..., an , where each represents a point at coordinate (i, ai). n vertical lines are drawn such that the two endpoints of line i is at (i, ai) and (i, 0). Find two lines, which together with x-axis forms a container, such that the container contains the most water.

Note: You may not slant the container and n is at least 2.

The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Example 1:

```
Input: [1,8,6,2,5,4,8,3,7]
Output: 49
```

题目大意

给出一个非负整数数组 a1，a2，a3，..... an，每个整数标识一个竖立在坐标轴 x 位置的一堵高度为 ai 的墙，选择两堵墙，和 x 轴构成的容器可以容纳最多的水。

解题思路

这一题也是对撞指针的思路。首尾分别 2 个指针，每次移动以后都分别判断长宽的乘积是否最大。

代码

```go
package leetcode

func maxArea(height []int) int {
  max, start, end := 0, 0, len(height)-1
  for start < end {
    width := end - start
    high := 0
    if height[start] < height[end] {
      high = height[start]
      start++
    } else {
      high = height[end]
      end--
    }

    temp := width * high
    if temp > max {
      max = temp
    }
  }
  return max
}
```

12. Integer to Roman

题目

Roman numerals are represented by seven different symbols: `I`, `V`, `X`, `L`, `C`, `D` and `M`.

```
Symbol       Value
I             1
V             5
X             10
L             50
C             100
D             500
M             1000
```

For example, `2` is written as `II` in Roman numeral, just two one's added together. `12` is written as `XII`, which is simply `X + II`. The number `27` is written as `XXVII`, which is `XX + V + II`.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not `IIII`. Instead, the number four is written as `IV`. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as `IX`. There are six instances where subtraction is used:

- `I` can be placed before `V` (5) and `X` (10) to make 4 and 9.
- `X` can be placed before `L` (50) and `C` (100) to make 40 and 90.
- `C` can be placed before `D` (500) and `M` (1000) to make 400 and 900.

Given an integer, convert it to a roman numeral.

Example 1:

```
Input: num = 3
Output: "III"
```

Example 2:

```
Input: num = 4
Output: "IV"
```

Example 3:

```
Input: num = 9
Output: "IX"
```

Example 4:

```
Input: num = 58
Output: "LVIII"
Explanation: L = 50, V = 5, III = 3.
```

Example 5:

```
Input: num = 1994
Output: "MCMXCIV"
Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.
```

Constraints:

- `1 <= num <= 3999`

题目大意

通常情况下，罗马数字中小的数字在大的数字的右边。但也存在特例，例如 4 不写做 IIII，而是 IV。数字 1 在数字 5 的左边，所表示的数等于大数 5 减小数 1 得到的数值 4 。同样地，数字 9 表示为 IX。这个特殊的规则只适用于以下六种情况：

- I 可以放在 V (5) 和 X (10) 的左边，来表示 4 和 9。
- X 可以放在 L (50) 和 C (100) 的左边，来表示 40 和 90。
- C 可以放在 D (500) 和 M (1000) 的左边，来表示 400 和 900。

给定一个整数，将其转为罗马数字。输入确保在 1 到 3999 的范围内。

解题思路

- 依照题意，优先选择大的数字，解题思路采用贪心算法。将 1-3999 范围内的罗马数字从大到小放在数组中，从头选择到尾，即可把整数转成罗马数字。

代码

```go
package leetcode

func intToRoman(num int) string {
  values := []int{1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1}
  symbols := []string{"M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV",
"I"}
  res, i := "", 0
  for num != 0 {
    for values[i] > num {
      i++
    }
    num -= values[i]
    res += symbols[i]
  }
  return res
}
```

13. Roman to Integer

题目

Roman numerals are represented by seven different symbols: `I`, `V`, `X`, `L`, `C`, `D` and `M`.

```
Symbol       Value
I             1
V             5
X             10
L             50
C             100
D             500
M             1000
```

For example, two is written as `II` in Roman numeral, just two one's added together. Twelve is written as, `XII`, which is simply `X` + `II`. The number twenty seven is written as `XXVII`, which is `XX` + `V` + `II`.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not `IIII`. Instead, the number four is written as `IV`. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as `IX`. There are six instances where subtraction is used:

- `I` can be placed before `V` (5) and `X` (10) to make 4 and 9.
- `X` can be placed before `L` (50) and `C` (100) to make 40 and 90.
- `C` can be placed before `D` (500) and `M` (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer. Input is guaranteed to be within the range from 1 to 3999.

Example 1:

```
Input: "III"
Output: 3
```

Example 2:

```
Input: "IV"
Output: 4
```

Example 3:

```
Input: "IX"
Output: 9
```

Example 4:

```
Input: "LVIII"
Output: 58
Explanation: L = 50, V= 5, III = 3.
```

Example 5:

```
Input: "MCMXCIV"
Output: 1994
Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.
```

题目大意

罗马数字包含以下七种字符: I， V， X， L，C，D 和 M。

```
字符          数值
I             1
V             5
X             10
L             50
C             100
D             500
M             1000
```

例如， 罗马数字 2 写做 II ，即为两个并列的 1。12 写做 XII ，即为 X + II 。 27 写做 XXVII, 即为 XX + V + II 。

通常情况下，罗马数字中小的数字在大的数字的右边。但也存在特例，例如 4 不写做 IIII，而是 IV。数字 1 在数字 5 的左边，所表示的数等于大数 5 减小数 1 得到的数值 4 。同样地，数字 9 表示为 IX。这个特殊的规则只适用于以下六种情况：

- I 可以放在 V (5) 和 X (10) 的左边，来表示 4 和 9。
- X 可以放在 L (50) 和 C (100) 的左边，来表示 40 和 90。
- C 可以放在 D (500) 和 M (1000) 的左边，来表示 400 和 900。

给定一个罗马数字，将其转换成整数。输入确保在 1 到 3999 的范围内。

解题思路

- 给定一个罗马数字，将其转换成整数。输入确保在 1 到 3999 的范围内。
- 简单题。按照题目中罗马数字的字符数值，计算出对应罗马数字的十进制数即可。

代码

```go
package leetcode

var roman = map[string]int{
  "I": 1,
  "V": 5,
  "X": 10,
  "L": 50,
  "C": 100,
  "D": 500,
```

```go
  "M": 1000,
}

func romanToInt(s string) int {
  if s == "" {
    return 0
  }
  num, lastint, total := 0, 0, 0
  for i := 0; i < len(s); i++ {
    char := s[len(s)-(i+1) : len(s)-i]
    num = roman[char]
    if num < lastint {
      total = total - num
    } else {
      total = total + num
    }
    lastint = num
  }
  return total
}
```

15. 3Sum

题目

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]
]
```

题目大意

给定一个数组，要求在这个数组中找出 3 个数之和为 0 的所有组合。

解题思路

用 map 提前计算好任意 2 个数字之和，保存起来，可以将时间复杂度降到 O(n^2)。这一题比较麻烦的一点在于，最后输出解的时候，要求输出不重复的解。数组中同一个数字可能出现多次，同一个数字也可能使用多次，但是最后输出解的时候，不能重复。例如 [-1，-1，2] 和 [2, -1, -1]、[-1, 2, -1] 这 3 个解是重复的，即使 -1 可能出现 100 次，每次使用的 -1 的数组下标都是不同的。

这里就需要去重和排序了。map 记录每个数字出现的次数，然后对 map 的 key 数组进行排序，最后在这个排序以后的数组里面扫，找到另外 2 个数字能和自己组成 0 的组合。

代码

```go
package leetcode

import (
  "sort"
)

// 解法一 最优解，双指针 + 排序
func threeSum(nums []int) [][]int {
  sort.Ints(nums)
  result, start, end, index, addNum, length := make([][]int, 0), 0, 0, 0, 0, len(nums)
  for index = 1; index < length-1; index++ {
    start, end = 0, length-1
    if index > 1 && nums[index] == nums[index-1] {
      start = index - 1
    }
    for start < index && end > index {
      if start > 0 && nums[start] == nums[start-1] {
        start++
        continue
      }
      if end < length-1 && nums[end] == nums[end+1] {
        end--
        continue
      }
      addNum = nums[start] + nums[end] + nums[index]
      if addNum == 0 {
        result = append(result, []int{nums[start], nums[index], nums[end]})
        start++
        end--
      } else if addNum > 0 {
        end--
      } else {
        start++
      }
    }
  }
```

```go
  return result
}

// 解法二
func threeSum1(nums []int) [][]int {
  res := [][]int{}
  counter := map[int]int{}
  for _, value := range nums {
    counter[value]++
  }

  uniqNums := []int{}
  for key := range counter {
    uniqNums = append(uniqNums, key)
  }
  sort.Ints(uniqNums)

  for i := 0; i < len(uniqNums); i++ {
    if (uniqNums[i]*3 == 0) && counter[uniqNums[i]] >= 3 {
      res = append(res, []int{uniqNums[i], uniqNums[i], uniqNums[i]})
    }
    for j := i + 1; j < len(uniqNums); j++ {
      if (uniqNums[i]*2+uniqNums[j] == 0) && counter[uniqNums[i]] > 1 {
        res = append(res, []int{uniqNums[i], uniqNums[i], uniqNums[j]})
      }
      if (uniqNums[j]*2+uniqNums[i] == 0) && counter[uniqNums[j]] > 1 {
        res = append(res, []int{uniqNums[i], uniqNums[j], uniqNums[j]})
      }
      c := 0 - uniqNums[i] - uniqNums[j]
      if c > uniqNums[j] && counter[c] > 0 {
        res = append(res, []int{uniqNums[i], uniqNums[j], c})
      }
    }
  }
  return res
}
```

16. 3Sum Closest

题目

Given an array nums of n integers and an integer target, find three integers in nums such that the sum is closest to target. Return the sum of the three integers. You may assume that each input would have exactly one solution.

Example:

```
Given array nums = [-1, 2, 1, -4], and target = 1.

The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).
```

题目大意

给定一个数组，要求在这个数组中找出 3 个数之和离 target 最近。

解题思路

这一题看似和第 15 题和第 18 题很像，都是求 3 或者 4 个数之和的问题，但是这一题的做法和 15，18 题完全不同。

这一题的解法是用两个指针夹逼的方法。先对数组进行排序，i 从头开始往后面扫。这里同样需要注意数组中存在多个重复数字的问题。具体处理方法很多，可以用 map 计数去重。这里笔者简单的处理，i 在循环的时候和前一个数进行比较，如果相等，i 继续往后移，直到移到下一个和前一个数字不同的位置。j，k 两个指针开始一前一后夹逼。j 为 i 的下一个数字，k 为数组最后一个数字，由于经过排序，所以 k 的数字最大。j 往后移动，k 往前移动，逐渐夹逼出最接近 target 的值。

这道题还可以用暴力解法，三层循环找到距离 target 最近的组合。具体见代码。

代码

```go
package leetcode

import (
  "math"
  "sort"
)

// 解法一 O(n^2)
func threeSumClosest(nums []int, target int) int {
  n, res, diff := len(nums), 0, math.MaxInt32
  if n > 2 {
    sort.Ints(nums)
    for i := 0; i < n-2; i++ {
      for j, k := i+1, n-1; j < k; {
        sum := nums[i] + nums[j] + nums[k]
        if abs(sum-target) < diff {
          res, diff = sum, abs(sum-target)
        }
        if sum == target {
          return res
```

```go
        } else if sum > target {
          k--
        } else {
          j++
        }
      }
    }
  }
  return res
}

// 解法二 暴力解法 O(n^3)
func threeSumClosest1(nums []int, target int) int {
  res, difference := 0, math.MaxInt16
  for i := 0; i < len(nums); i++ {
    for j := i + 1; j < len(nums); j++ {
      for k := j + 1; k < len(nums); k++ {
        if abs(nums[i]+nums[j]+nums[k]-target) < difference {
          difference = abs(nums[i] + nums[j] + nums[k] - target)
          res = nums[i] + nums[j] + nums[k]
        }
      }
    }
  }
  return res
}

func abs(a int) int {
  if a > 0 {
    return a
  }
  return -a
}
```

17. Letter Combinations of a Phone Number

题目

Given a string containing digits from `2-9` inclusive, return all possible letter combinations that the number could represent.

A mapping of digit to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

Example:

```
Input: "23"
Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].
```

Note:

Although the above answer is in lexicographical order, your answer could be in any order you want.

题目大意

给定一个仅包含数字 2-9 的字符串，返回所有它能表示的字母组合。给出数字到字母的映射如下（与电话按键相同）。注意 1 不对应任何字母。

解题思路

- DFS 递归深搜即可

代码

```go
package leetcode

var (
  letterMap = []string{
    " ",    //0
    "",     //1
    "abc",  //2
    "def",  //3
    "ghi",  //4
    "jkl",  //5
    "mno",  //6
    "pqrs", //7
    "tuv",  //8
    "wxyz", //9
  }
  res   = []string{}
  final = 0
```

```go
)

// 解法一 DFS
func letterCombinations(digits string) []string {
  if digits == "" {
    return []string{}
  }
  res = []string{}
  findCombination(&digits, 0, "")
  return res
}

func findCombination(digits *string, index int, s string) {
  if index == len(*digits) {
    res = append(res, s)
    return
  }
  num := (*digits)[index]
  letter := letterMap[num-'0']
  for i := 0; i < len(letter); i++ {
    findCombination(digits, index+1, s+string(letter[i]))
  }
  return
}

// 解法二 非递归
func letterCombinations_(digits string) []string {
  if digits == "" {
    return []string{}
  }
  index := digits[0] - '0'
  letter := letterMap[index]
  tmp := []string{}
  for i := 0; i < len(letter); i++ {
    if len(res) == 0 {
      res = append(res, "")
    }
    for j := 0; j < len(res); j++ {
      tmp = append(tmp, res[j]+string(letter[i]))
    }
  }
  res = tmp
  final++
  letterCombinations(digits[1:])
  final--
  if final == 0 {
    tmp = res
    res = []string{}
  }
```

```go
  return tmp
}

// 解法三 回溯（参考回溯模板，类似DFS）
var result []string
var dict = map[string][]string{
  "2" : []string{"a","b","c"},
  "3" : []string{"d", "e", "f"},
  "4" : []string{"g", "h", "i"},
  "5" : []string{"j", "k", "l"},
  "6" : []string{"m", "n", "o"},
  "7" : []string{"p", "q", "r", "s"},
  "8" : []string{"t", "u", "v"},
  "9" : []string{"w", "x", "y", "z"},
}

func letterCombinationsBT(digits string) []string {
  result = []string{}
  if digits == "" {
    return result
  }
  letterFunc("", digits)
  return result
}

func letterFunc(res string, digits string) {
  if digits == "" {
    result = append(result, res)
    return
  }

  k := digits[0:1]
  digits = digits[1:]
  for i := 0; i < len(dict[k]); i++ {
    res += dict[k][i]
    letterFunc(res, digits)
    res = res[0 : len(res)-1]
  }
}
```

18. 4Sum

题目

Given an array nums of n integers and an integer target, are there elements a, b, c, and d in nums such that a + b + c + d = target? Find all unique quadruplets in the array which gives the sum of target.

Note:

The solution set must not contain duplicate quadruplets.

Example:

```
Given array nums = [1, 0, -1, 0, -2, 2], and target = 0.

A solution set is:
[
  [-1,  0, 0, 1],
  [-2, -1, 1, 2],
  [-2,  0, 0, 2]
]
```

题目大意

给定一个数组，要求在这个数组中找出 4 个数之和为 0 的所有组合。

解题思路

用 map 提前计算好任意 3 个数字之和，保存起来，可以将时间复杂度降到 O(n^3)。这一题比较麻烦的一点在于，最后输出解的时候，要求输出不重复的解。数组中同一个数字可能出现多次，同一个数字也可能使用多次，但是最后输出解的时候，不能重复。例如 [-1， 1， 2, -2] 和 [2, -1, -2, 1]、[-2, 2, -1, 1] 这 3 个解是重复的，即使 -1, -2 可能出现 100 次，每次使用的 -1, -2 的数组下标都是不同的。

这一题是第 15 题的升级版，思路都是完全一致的。这里就需要去重和排序了。map 记录每个数字出现的次数，然后对 map 的 key 数组进行排序，最后在这个排序以后的数组里面扫，找到另外 3 个数字能和自己组成 0 的组合。

第 15 题和第 18 题的解法一致。

代码

```go
package leetcode

import "sort"

func fourSum(nums []int, target int) [][]int {
  res := [][]int{}
  counter := map[int]int{}
  for _, value := range nums {
    counter[value]++
  }

  uniqNums := []int{}
  for key := range counter {
    uniqNums = append(uniqNums, key)
  }
```

```
  sort.Ints(uniqNums)

  for i := 0; i < len(uniqNums); i++ {
    if (uniqNums[i]*4 == target) && counter[uniqNums[i]] >= 4 {
      res = append(res, []int{uniqNums[i], uniqNums[i], uniqNums[i], uniqNums[i]})
    }
    for j := i + 1; j < len(uniqNums); j++ {
      if (uniqNums[i]*3+uniqNums[j] == target) && counter[uniqNums[i]] > 2 {
        res = append(res, []int{uniqNums[i], uniqNums[i], uniqNums[i], uniqNums[j]})
      }
      if (uniqNums[j]*3+uniqNums[i] == target) && counter[uniqNums[j]] > 2 {
        res = append(res, []int{uniqNums[i], uniqNums[j], uniqNums[j], uniqNums[j]})
      }
      if (uniqNums[j]*2+uniqNums[i]*2 == target) && counter[uniqNums[j]] > 1 &&
counter[uniqNums[i]] > 1 {
        res = append(res, []int{uniqNums[i], uniqNums[i], uniqNums[j], uniqNums[j]})
      }
      for k := j + 1; k < len(uniqNums); k++ {
        if (uniqNums[i]*2+uniqNums[j]+uniqNums[k] == target) && counter[uniqNums[i]] >
1 {
          res = append(res, []int{uniqNums[i], uniqNums[i], uniqNums[j], uniqNums[k]})
        }
        if (uniqNums[j]*2+uniqNums[i]+uniqNums[k] == target) && counter[uniqNums[j]] >
1 {
          res = append(res, []int{uniqNums[i], uniqNums[j], uniqNums[j], uniqNums[k]})
        }
        if (uniqNums[k]*2+uniqNums[i]+uniqNums[j] == target) && counter[uniqNums[k]] >
1 {
          res = append(res, []int{uniqNums[i], uniqNums[j], uniqNums[k], uniqNums[k]})
        }
        c := target - uniqNums[i] - uniqNums[j] - uniqNums[k]
        if c > uniqNums[k] && counter[c] > 0 {
          res = append(res, []int{uniqNums[i], uniqNums[j], uniqNums[k], c})
        }
      }
    }
  }
  return res
}
```

19. Remove Nth Node From End of List

题目

Given the `head` of a linked list, remove the `nth` node from the end of the list and return its head.

Follow up: Could you do this in one pass?

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`

题目大意

删除链表中倒数第 n 个结点。

解题思路

这道题比较简单，先循环一次拿到链表的总长度，然后循环到要删除的结点的前一个结点开始删除操作。需要注意的一个特例是，有可能要删除头结点，要单独处理。

这道题有一种特别简单的解法。设置 2 个指针，一个指针距离前一个指针 n 个距离。同时移动 2 个指针，2 个指针都移动相同的距离。当一个指针移动到了终点，那么前一个指针就是倒数第 n 个节点了。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/structures"
)

// ListNode define
type ListNode = structures.ListNode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */

// 解法一
func removeNthFromEnd(head *ListNode, n int) *ListNode {
  dummyHead := &ListNode{Next: head}
  preSlow, slow, fast := dummyHead, head, head
  for fast != nil {
    if n <= 0 {
      preSlow = slow
      slow = slow.Next
    }
    n--
    fast = fast.Next
  }
  preSlow.Next = slow.Next
  return dummyHead.Next
}

// 解法二
func removeNthFromEnd1(head *ListNode, n int) *ListNode {
  if head == nil {
    return nil
  }
  if n <= 0 {
    return head
  }
  current := head
  len := 0
  for current != nil {
    len++
    current = current.Next
```

```
  }
  if n > len {
    return head
  }
  if n == len {
    current := head
    head = head.Next
    current.Next = nil
    return head
  }
  current = head
  i := 0
  for current != nil {
    if i == len-n-1 {
      deleteNode := current.Next
      current.Next = current.Next.Next
      deleteNode.Next = nil
      break
    }
    i++
    current = current.Next
  }
  return head
}
```

20. Valid Parentheses

题目

Given a string 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.
Note that an empty string is also considered valid.

Example 1:

```
Input: "()"
Output: true
```

Example 2:

```
Input: "()[]{}"
Output: true
```

Example 3:

```
Input: "(]"
Output: false
```

Example 4:

```
Input: "([)]"
Output: false
```

Example 5:

```
Input: "{[]}"
Output: true
```

题目大意

括号匹配问题。

解题思路

遇到左括号就进栈push，遇到右括号并且栈顶为与之对应的左括号，就把栈顶元素出栈。最后看栈里面还有没有其他元素，如果为空，即匹配。

需要注意，空字符串是满足括号匹配的，即输出 true。

代码

```go
package leetcode

func isValid(s string) bool {
  // 空字符串直接返回 true
  if len(s) == 0 {
    return true
  }
```

```
  stack := make([]rune, 0)
  for _, v := range s {
    if (v == '[') || (v == '(') || (v == '{') {
      stack = append(stack, v)
    } else if ((v == ']') && len(stack) > 0 && stack[len(stack)-1] == '[') ||
      ((v == ')') && len(stack) > 0 && stack[len(stack)-1] == '(') ||
      ((v == '}') && len(stack) > 0 && stack[len(stack)-1] == '{') {
      stack = stack[:len(stack)-1]
    } else {
      return false
    }
  }
  return len(stack) == 0
}
```

21. Merge Two Sorted Lists

题目

Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

Example:

```
Input: 1->2->4, 1->3->4
Output: 1->1->2->3->4->4
```

题目大意

合并 2 个有序链表

解题思路

按照题意做即可。

代码

```
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
```

```go
 *     Next *ListNode
 * }
 */
func mergeTwoLists(l1 *ListNode, l2 *ListNode) *ListNode {
  if l1 == nil {
    return l2
  }
  if l2 == nil {
    return l1
  }
  if l1.Val < l2.Val {
    l1.Next = mergeTwoLists(l1.Next, l2)
    return l1
  }
  l2.Next = mergeTwoLists(l1, l2.Next)
  return l2
}
```

22. Generate Parentheses

题目

Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.

For example, given n = 3, a solution set is:

```
[
  "((())",
  "(()())",
  "(())()",
  "()(())",
  "()()()"
]
```

题目大意

给出 n 代表生成括号的对数，请你写出一个函数，使其能够生成所有可能的并且有效的括号组合。

解题思路

- 这道题乍一看需要判断括号是否匹配的问题，如果真的判断了，那时间复杂度就到 O(n * 2^n)了，虽然也可以 AC，但是时间复杂度巨高。
- 这道题实际上不需要判断括号是否匹配的问题。因为在 DFS 回溯的过程中，会让 `(` 和 `)` 成对的匹配上的。

代码

```go
package leetcode

func generateParenthesis(n int) []string {
  if n == 0 {
    return []string{}
  }
  res := []string{}
  findGenerateParenthesis(n, n, "", &res)
  return res
}

func findGenerateParenthesis(lindex, rindex int, str string, res *[]string) {
  if lindex == 0 && rindex == 0 {
    *res = append(*res, str)
    return
  }
  if lindex > 0 {
    findGenerateParenthesis(lindex-1, rindex, str+"(", res)
  }
  if rindex > 0 && lindex < rindex {
    findGenerateParenthesis(lindex, rindex-1, str+")", res)
  }
}
```

23. Merge k Sorted Lists

题目

Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

Example:

```
Input:
[
  1->4->5,
  1->3->4,
  2->6
]
Output: 1->1->2->3->4->4->5->6
```

题目大意

合并 K 个有序链表

解题思路

借助分治的思想，把 K 个有序链表两两合并即可。相当于是第 21 题的加强版。

代码

```go
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */
func mergeKLists(lists []*ListNode) *ListNode {
  length := len(lists)
  if length < 1 {
    return nil
  }
  if length == 1 {
    return lists[0]
  }
  num := length / 2
  left := mergeKLists(lists[:num])
  right := mergeKLists(lists[num:])
  return mergeTwoLists1(left, right)
}

func mergeTwoLists1(l1 *ListNode, l2 *ListNode) *ListNode {
  if l1 == nil {
    return l2
  }
  if l2 == nil {
    return l1
  }
  if l1.Val < l2.Val {
    l1.Next = mergeTwoLists1(l1.Next, l2)
    return l1
  }
  l2.Next = mergeTwoLists1(l1, l2.Next)
  return l2
}
```

24. Swap Nodes in Pairs

题目

Given a linked list, swap every two adjacent nodes and return its head.

You may not modify the values in the list's nodes, only nodes itself may be changed.

Example:

```
Given 1->2->3->4, you should return the list as 2->1->4->3.
```

题目大意

两两相邻的元素，翻转链表

解题思路

按照题意做即可。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/structures"
)

// ListNode define
type ListNode = structures.ListNode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */

func swapPairs(head *ListNode) *ListNode {
  dummy := &ListNode{Next: head}
  for pt := dummy; pt != nil && pt.Next != nil && pt.Next.Next != nil; {
    pt, pt.Next, pt.Next.Next, pt.Next.Next.Next = pt.Next, pt.Next.Next,
pt.Next.Next.Next, pt.Next
```

```
    }
    return dummy.Next
}
```

25. Reverse Nodes in k-Group

题目

Given a linked list, reverse the nodes of a linked list k at a time and return its modified list.

k is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of k then left-out nodes in the end should remain as it is.

Example:

```
Given this linked list: 1->2->3->4->5

For k = 2, you should return: 2->1->4->3->5

For k = 3, you should return: 3->2->1->4->5
```

Note:

- Only constant extra memory is allowed.
- You may not alter the values in the list's nodes, only nodes itself may be changed.

题目大意

按照每 K 个元素翻转的方式翻转链表。如果不满足 K 个元素的就不翻转。

解题思路

这一题是 problem 24 的加强版，problem 24 是两两相邻的元素，翻转链表。而 problem 25 要求的是 k 个相邻的元素，翻转链表，problem 相当于是 k = 2 的特殊情况。

代码

```
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
```

```go
 *     Next *ListNode
 * }
 */
func reverseKGroup(head *ListNode, k int) *ListNode {
  node := head
  for i := 0; i < k; i++ {
    if node == nil {
      return head
    }
    node = node.Next
  }
  newHead := reverse(head, node)
  head.Next = reverseKGroup(node, k)
  return newHead
}

func reverse(first *ListNode, last *ListNode) *ListNode {
  prev := last
  for first != last {
    tmp := first.Next
    first.Next = prev
    prev = first
    first = tmp
  }
  return prev
}
```

26. Remove Duplicates from Sorted Array

题目

Given a sorted array nums, remove the duplicates in-place such that each element appear only once 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.

Example 1:

```
Given nums = [1,1,2],

Your function should return length = 2, with the first two elements of nums being 1 and
2 respectively.

It doesn't matter what you leave beyond the returned length.
```

Example 2:

```
Given nums = [0,0,1,1,1,2,2,3,3,4],

Your function should return length = 5, with the first five elements of nums being
modified to 0, 1, 2, 3, and 4 respectively.

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]);
}
```

题目大意

给定一个有序数组 nums，对数组中的元素进行去重，使得原数组中的每个元素只有一个。最后返回去重以后数组的长度值。

解题思路

这道题和第 27 题很像。这道题和第 283 题，第 27 题基本一致，283 题是删除 0，27 题是删除指定元素，这一题是删除重复元素，实质是一样的。

这里数组的删除并不是真的删除，只是将删除的元素移动到数组后面的空间内，然后返回数组实际剩余的元素个数，OJ 最终判断题目的时候会读取数组剩余个数的元素进行输出。

代码

```
package leetcode

// 解法一
func removeDuplicates(nums []int) int {
  if len(nums) == 0 {
    return 0
  }
  last, finder := 0, 0
  for last < len(nums)-1 {
    for nums[finder] == nums[last] {
      finder++
      if finder == len(nums) {
        return last + 1
      }
    }
    nums[last+1] = nums[finder]
    last++
  }
  return last + 1
}

// 解法二
func removeDuplicates1(nums []int) int {
  if len(nums) == 0 {
    return 0
  }
  length := len(nums)
  lastNum := nums[length-1]
  i := 0
  for i = 0; i < length-1; i++ {
    if nums[i] == lastNum {
      break
    }
    if nums[i+1] == nums[i] {
      removeElement1(nums, i+1, nums[i])
      // fmt.Printf("此时 num = %v length = %v\n", nums, length)
    }
  }
  return i + 1
}

func removeElement1(nums []int, start, val int) int {
  if len(nums) == 0 {
```

```
    return 0
  }
  j := start
  for i := start; i < len(nums); i++ {
    if nums[i] != val {
      if i != j {
        nums[i], nums[j] = nums[j], nums[i]
        j++
      } else {
        j++
      }
    }
  }
  return j
}
```

27. Remove Element

题目

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]);
}
```

题目大意

给定一个数组 nums 和一个数值 val，将数组中所有等于 val 的元素删除，并返回剩余的元素个数。

解题思路

这道题和第 283 题很像。这道题和第 283 题基本一致，283 题是删除 0，这一题是给定的一个 val，实质是一样的。

这里数组的删除并不是真的删除，只是将删除的元素移动到数组后面的空间内，然后返回数组实际剩余的元素个数，OJ 最终判断题目的时候会读取数组剩余个数的元素进行输出。

代码

```go
package leetcode

func removeElement(nums []int, val int) int {
```

```
  if len(nums) == 0 {
    return 0
  }
  j := 0
  for i := 0; i < len(nums); i++ {
    if nums[i] != val {
      if i != j {
        nums[i], nums[j] = nums[j], nums[i]
      }
      j++
    }
  }
  return j
}
```

28. Implement strStr()

题目

Implement strStr().

Return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

Example 1:

```
Input: haystack = "hello", needle = "ll"
Output: 2
```

Example 2:

```
Input: haystack = "aaaaa", needle = "bba"
Output: -1
```

Clarification:

What should we return when needle is an empty string? This is a great question to ask during an interview.

For the purpose of this problem, we will return 0 when needle is an empty string. This is consistent to C's strstr() and Java's indexOf().

题目大意

实现一个查找 substring 的函数。如果在母串中找到了子串，返回子串在母串中出现的下标，如果没有找到，返回 -1，如果子串是空串，则返回 0 。

解题思路

这一题比较简单，直接写即可。

代码

```
package leetcode

import "strings"

// 解法一
func strStr(haystack string, needle string) int {
  for i := 0; ; i++ {
    for j := 0; ; j++ {
      if j == len(needle) {
        return i
      }
      if i+j == len(haystack) {
        return -1
      }
      if needle[j] != haystack[i+j] {
        break
      }
    }
  }
}

// 解法二
func strStr1(haystack string, needle string) int {
  return strings.Index(haystack, needle)
}
```

29. Divide Two Integers

题目

Given two integers `dividend` and `divisor`, divide two integers without using multiplication, division and mod operator.

Return the quotient after dividing `dividend` by `divisor`.

The integer division should truncate toward zero.

Example 1:

```
Input: dividend = 10, divisor = 3
Output: 3
```

Example 2:

```
Input: dividend = 7, divisor = -3
Output: -2
```

Note:

- Both dividend and divisor will be 32-bit signed integers.
- The divisor will never be 0.
- Assume we are dealing with an environment which could only store integers within the 32-bit signed integer range: [−2^31, 2^31 − 1]. For the purpose of this problem, assume that your function returns 2^31 − 1 when the division result overflows.

题目大意

给定两个整数，被除数 dividend 和除数 divisor。将两数相除，要求不使用乘法、除法和 mod 运算符。返回被除数 dividend 除以除数 divisor 得到的商。

说明:

- 被除数和除数均为 32 位有符号整数。
- 除数不为 0。
- 假设我们的环境只能存储 32 位有符号整数，其数值范围是 [−2^31，2^31 − 1]。本题中，如果除法结果溢出，则返回 2^31 − 1。

解题思路

- 给出除数和被除数，要求计算除法运算以后的商。注意值的取值范围在 [−2^31, 2^31 − 1] 之中。超过范围的都按边界计算。
- 这一题可以用二分搜索来做。要求除法运算之后的商，把商作为要搜索的目标。商的取值范围是 [0, dividend]，所以从 0 到被除数之间搜索。利用二分，找到(商 + 1) * 除数 > 被除数并且 商 * 除数 ≤ 被除数 或者 (商+1)* 除数 ≥ 被除数并且商 * 除数 < 被除数的时候，就算找到了商，其余情况继续二分即可。最后还要注意符号和题目规定的 Int32 取值范围。
- 二分的写法常写错的 3 点：
    1. low ≤ high (注意二分循环退出的条件是小于等于)

2. mid = low + (high-low)>>1 (防止溢出)
3. low = mid + 1 ; high = mid - 1 (注意更新 low 和 high 的值，如果更新不对就会死循环)

代码

```
package leetcode

import (
  "math"
)

// 解法一 递归版的二分搜索
func divide(dividend int, divisor int) int {
  sign, res := -1, 0
  // low, high := 0, abs(dividend)
  if dividend == 0 {
    return 0
  }
  if divisor == 1 {
    return dividend
  }
  if dividend == math.MinInt32 && divisor == -1 {
    return math.MaxInt32
  }
  if dividend > 0 && divisor > 0 || dividend < 0 && divisor < 0 {
    sign = 1
  }
  if dividend > math.MaxInt32 {
    dividend = math.MaxInt32
  }
  // 如果把递归改成非递归，可以改成下面这段代码
  // for low <= high {
  //  quotient := low + (high-low)>>1
  //  if ((quotient+1)*abs(divisor) > abs(dividend) && quotient*abs(divisor) <=
abs(dividend)) || ((quotient+1)*abs(divisor) >= abs(dividend) && quotient*abs(divisor)
< abs(dividend)) {
  //    if (quotient+1)*abs(divisor) == abs(dividend) {
  //      res = quotient + 1
  //      break
  //    }
  //    res = quotient
  //    break
  //  }
  //  if (quotient+1)*abs(divisor) > abs(dividend) && quotient*abs(divisor) >
abs(dividend) {
  //    high = quotient - 1
  //  }
```

```go
  //  if (quotient+1)*abs(divisor) < abs(dividend) && quotient*abs(divisor) <
abs(dividend) {
  //    low = quotient + 1
  //  }
  // }
  res = binarySearchQuotient(0, abs(dividend), abs(divisor), abs(dividend))
  if res > math.MaxInt32 {
    return sign * math.MaxInt32
  }
  if res < math.MinInt32 {
    return sign * math.MinInt32
  }
  return sign * res
}

func binarySearchQuotient(low, high, val, dividend int) int {
  quotient := low + (high-low)>>1
  if ((quotient+1)*val > dividend && quotient*val <= dividend) || ((quotient+1)*val >=
dividend && quotient*val < dividend) {
    if (quotient+1)*val == dividend {
      return quotient + 1
    }
    return quotient
  }
  if (quotient+1)*val > dividend && quotient*val > dividend {
    return binarySearchQuotient(low, quotient-1, val, dividend)
  }
  if (quotient+1)*val < dividend && quotient*val < dividend {
    return binarySearchQuotient(quotient+1, high, val, dividend)
  }
  return 0
}

// 解法二 非递归版的二分搜索
func divide1(divided int, divisor int) int {
  if divided == math.MinInt32 && divisor == -1 {
    return math.MaxInt32
  }
  result := 0
  sign := -1
  if divided > 0 && divisor > 0 || divided < 0 && divisor < 0 {
    sign = 1
  }
  dvd, dvs := abs(divided), abs(divisor)
  for dvd >= dvs {
    temp := dvs
    m := 1
    for temp<<1 <= dvd {
      temp <<= 1
```

```
      m <<= 1
    }
    dvd -= temp
    result += m
  }
  return sign * result
}
```

30. Substring with Concatenation of All Words

题目

You are given a string, s, and a list of words, words, that are all of the same length. Find all starting indices of substring(s) in s that is a concatenation of each word in words exactly once and without any intervening characters.

Example 1:

```
Input:
  s = "barfoothefoobarman",
  words = ["foo","bar"]
Output: [0,9]
Explanation: Substrings starting at index 0 and 9 are "barfoor" and "foobar"
respectively.
The output order does not matter, returning [9,0] is fine too.
```

Example 2:

```
Input:
  s = "wordgoodgoodgoodbestword",
  words = ["word","good","best","word"]
Output: []
```

题目大意

给定一个源字符串 s，再给一个字符串数组，要求在源字符串中找到由字符串数组各种组合组成的连续串的起始下标，如果存在多个，在结果中都需要输出。

解题思路

这一题看似很难，但是有 2 个限定条件也导致这题不是特别难。1. 字符串数组里面的字符串长度都是一样的。2. 要求字符串数组中的字符串都要连续连在一起的，前后顺序可以是任意排列组合。

解题思路，先将字符串数组里面的所有字符串都存到 map 中，并累计出现的次数。然后从源字符串从头开始扫，每次判断字符串数组里面的字符串时候全部都用完了(计数是否为 0)，如果全部都用完了，并且长度正好是字符串数组任意排列组合的总长度，就记录下这个组合的起始下标。如果不符合，就继续考察源字符串的下一个字符，直到扫完整个源字符串。

代码

```
package leetcode

func findSubstring(s string, words []string) []int {
  if len(words) == 0 {
    return []int{}
  }
  res := []int{}
  counter := map[string]int{}
  for _, w := range words {
    counter[w]++
  }
  length, totalLen, tmpCounter := len(words[0]), len(words[0])*len(words),
copyMap(counter)
  for i, start := 0, 0; i < len(s)-length+1 && start < len(s)-length+1; i++ {
    //fmt.Printf("sub = %v i = %v lenght = %v start = %v tmpCounter = %v totalLen =
%v\n", s[i:i+length], i, length, start, tmpCounter, totalLen)
    if tmpCounter[s[i:i+length]] > 0 {
      tmpCounter[s[i:i+length]]--
      //fmt.Printf("******sub = %v i = %v lenght = %v start = %v tmpCounter = %v
totalLen = %v\n", s[i:i+length], i, length, start, tmpCounter, totalLen)
      if checkWords(tmpCounter) && (i+length-start == totalLen) {
        res = append(res, start)
        continue
      }
      i = i + length - 1
    } else {
      start++
      i = start - 1
      tmpCounter = copyMap(counter)
    }
  }
  return res
}

func checkWords(s map[string]int) bool {
  flag := true
  for _, v := range s {
    if v > 0 {
```

```go
        flag = false
        break
      }
    }
    return flag
  }

  func copyMap(s map[string]int) map[string]int {
    c := map[string]int{}
    for k, v := range s {
      c[k] = v
    }
    return c
  }
```

31. Next Permutation

题目

Implement **next permutation**, which rearranges numbers into the lexicographically next greater permutation of numbers.

If such an arrangement is not possible, it must rearrange it as the lowest possible order (i.e., sorted in ascending order).

The replacement must be in place and use only constant extra memory.

Example 1:

```
Input: nums = [1,2,3]
Output: [1,3,2]
```

Example 2:

```
Input: nums = [3,2,1]
Output: [1,2,3]
```

Example 3:

```
Input: nums = [1,1,5]
Output: [1,5,1]
```

Example 4:

```
Input: nums = [1]
Output: [1]
```

Constraints:

- `1 <= nums.length <= 100`
- `0 <= nums[i] <= 100`

题目大意

实现获取 下一个排列 的函数，算法需要将给定数字序列重新排列成字典序中下一个更大的排列。如果不存在下一个更大的排列，则将数字重新排列成最小的排列（即升序排列）。必须 原地 修改，只允许使用额外常数空间。

解题思路

- 题目有 3 个问题需要解决。如何找到下一个排列。不存在下一个排列的时候如何生成最小的排列。如何原地修改。先解决第一个问题，如何找到下一个排列。下一个排列是找到一个大于当前排序的字典序，且变大的幅度最小。那么只能将较小的数与较大数做一次原地交换。并且较小数的下标要尽量靠右，较大数也要尽可能小。原地交换以后，还需要将较大数右边的数按照升序重新排列。这样交换以后，才能生成下一个排列。以排列 [8,9,6,10,7,2] 为例：能找到的符合条件的一对「较小数」与「较大数」的组合为 6 与 7，满足「较小数」尽量靠右，而「较大数」尽可能小。当完成交换后排列变为 [8,9,7,10,6,2]，此时我们可以重排「较小数」右边的序列，序列变为 [8,9,7,2,6,10]。
- 第一步：在 `nums[i]` 中找到 `i` 使得 `nums[i] < nums[i+1]`，此时较小数为 `nums[i]`，并且 `[i+1, n)` 一定为下降区间。第二步：如果找到了这样的 `i` ，则在下降区间 `[i+1, n)` 中从后往前找到第一个 `j` ，使得 `nums[i] < nums[j]` ，此时较大数为 `nums[j]`。第三步，交换 `nums[i]` 和 `nums[j]`，此时区间 `[i+1, n)` 一定为降序区间。最后原地交换 `[i+1, n)` 区间内的元素，使其变为升序，无需对该区间进行排序。
- 如果第一步找不到符合条件的下标 `i`，说明当前序列已经是一个最大的排列。那么应该直接执行第三步，生成最小的排列。

代码

```go
package leetcode

func nextPermutation(nums []int) {
  i, j := 0, 0
  for i = len(nums) - 2; i >= 0; i-- {
    if nums[i] < nums[i+1] {
      break
    }
  }
  if i >= 0 {
    for j = len(nums) - 1; j > i; j-- {
      if nums[j] > nums[i] {
        break
      }
    }
```

```
    swap(&nums, i, j)
  }
  reverse(&nums, i+1, len(nums)-1)
}

func reverse(nums *[]int, i, j int) {
  for i < j {
    swap(nums, i, j)
    i++
    j--
  }
}

func swap(nums *[]int, i, j int) {
  (*nums)[i], (*nums)[j] = (*nums)[j], (*nums)[i]
}
```

32. Longest Valid Parentheses

题目

Given a string containing just the characters `'('` and `')'`, find the length of the longest valid (well-formed) parentheses substring.

Example 1:

```
Input: s = "(()"
Output: 2
Explanation: The longest valid parentheses substring is "()".
```

Example 2:

```
Input: s = ")()())"
Output: 4
Explanation: The longest valid parentheses substring is "()()".
```

Example 3:

```
Input: s = ""
Output: 0
```

Constraints:

- `0 <= s.length <= 3 * 104`
- `s[i]` is `'('`, or `')'`.

题目大意

给你一个只包含 '(' 和 ')' 的字符串，找出最长有效（格式正确且连续）括号子串的长度。

解题思路

- 提到括号匹配，第一时间能让人想到的就是利用栈。这里需要计算嵌套括号的总长度，所以栈里面不能单纯的存左括号，而应该存左括号在原字符串的下标，这样通过下标相减可以获取长度。那么栈如果是非空，栈底永远存的是当前遍历过的字符串中**上一个没有被匹配的右括号的下标**。**上一个没有被匹配的右括号的下标**可以理解为每段括号匹配之间的“隔板”。例如，`()))(((()))`，第三个右括号，即为左右 2 段正确的括号匹配中间的“隔板”。“隔板”的存在影响计算最长括号长度。如果不存在“隔板”，前后 2 段正确的括号匹配应该“融合”在一起，最长长度为 `2 + 6 = 8`，但是这里存在了“隔板”，所以最长长度仅为 `6`。
- 具体算法实现，遇到每个 `'('`，将它的下标压入栈中。对于遇到的每个 `')'`，先弹出栈顶元素表示匹配了当前右括号。如果栈为空，说明当前的右括号为没有被匹配的右括号，于是将其下标放入栈中来更新**上一个没有被匹配的右括号的下标**。如果栈不为空，当前右括号的下标减去栈顶元素即为以该右括号为结尾的最长有效括号的长度。需要注意初始化时，不存在**上一个没有被匹配的右括号的下标**，那么将 `-1` 放入栈中，充当下标为 `0` 的“隔板”。时间复杂度 O(n)，空间复杂度 O(n)。
- 在栈的方法中，实际用到的元素仅仅是栈底的**上一个没有被匹配的右括号的下标**。那么考虑能否把这个值存在一个变量中，这样可以省去栈 O(n) 的时间复杂度。利用两个计数器 left 和 right 。首先，从左到右遍历字符串，每当遇到 `'('`，增加 left 计数器，每当遇到 `')'`，增加 right 计数器。每当 left 计数器与 right 计数器相等时，计算当前有效字符串的长度，并且记录目前为止找到的最长子字符串。当 right 计数器比 left 计数器大时，说明括号不匹配，于是将 left 和 right 计数器同时变回 0。这样的做法利用了贪心的思想，考虑了以当前字符下标结尾的有效括号长度，每次当右括号数量多于左括号数量的时候之前的字符就扔掉不再考虑，重新从下一个字符开始计算。
- 但上面的做法会漏掉一种情况，就是遍历的时候左括号的数量始终大于右括号的数量，即 `(()`，这种时候最长有效括号是求不出来的。解决办法是反向再计算一遍，如果从右往左计算，`(()` 先计算匹配的括号，最后只剩下 `'('`，这样依旧可以算出最长匹配的括号长度。反过来计算的方法和上述从左往右计算的方法一致：当 left 计数器比 right 计数器大时，将 left 和 right 计数器同时变回 0；当 left 计数器与 right 计数器相等时，计算当前有效字符串的长度，并且记录目前为止找到的最长子字符串。这种方法的时间复杂度是 O(n)，空间复杂度 O(1)。

代码

```go
package leetcode

// 解法一 栈
func longestValidParentheses(s string) int {
  stack, res := []int{}, 0
  stack = append(stack, -1)
  for i := 0; i < len(s); i++ {
    if s[i] == '(' {
      stack = append(stack, i)
    } else {
      stack = stack[:len(stack)-1]
      if len(stack) == 0 {
        stack = append(stack, i)
      } else {
        res = max(res, i-stack[len(stack)-1])
      }
```

```go
    }
  }
  return res
}

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

// 解法二 双指针
func longestValidParentheses1(s string) int {
  left, right, maxLength := 0, 0, 0
  for i := 0; i < len(s); i++ {
    if s[i] == '(' {
      left++
    } else {
      right++
    }
    if left == right {
      maxLength = max(maxLength, 2*right)
    } else if right > left {
      left, right = 0, 0
    }
  }
  left, right = 0, 0
  for i := len(s) - 1; i >= 0; i-- {
    if s[i] == '(' {
      left++
    } else {
      right++
    }
    if left == right {
      maxLength = max(maxLength, 2*left)
    } else if left > right {
      left, right = 0, 0
    }
  }
  return maxLength
}
```

33. Search in Rotated Sorted Array

题目

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., `[0,1,2,4,5,6,7]` might become `[4,5,6,7,0,1,2]`).

You are given a target value to search. If found in the array return its index, otherwise return `-1`.

You may assume no duplicate exists in the array.

Your algorithm's runtime complexity must be in the order of *O*(log *n*).

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
```

题目大意

假设按照升序排序的数组在预先未知的某个点上进行了旋转。(例如，数组 [0,1,2,4,5,6,7] 可能变为 [4,5,6,7,0,1,2])。搜索一个给定的目标值，如果数组中存在这个目标值，则返回它的索引，否则返回 -1 。你可以假设数组中不存在重复的元素。

你的算法时间复杂度必须是 O(log n) 级别。

解题思路

- 给出一个数组，数组中本来是从小到大排列的，并且数组中没有重复数字。但是现在把后面随机一段有序的放到数组前面，这样形成了前后两端有序的子序列。在这样的一个数组里面查找一个数，设计一个 O(log n) 的算法。如果找到就输出数组的小标，如果没有找到，就输出 -1 。
- 由于数组基本有序，虽然中间有一个“断开点”，还是可以使用二分搜索的算法来实现。现在数组前面一段是数值比较大的数，后面一段是数值偏小的数。如果 mid 落在了前一段数值比较大的区间内了，那么一定有 `nums[mid] > nums[low]`，如果是落在后面一段数值比较小的区间内，`nums[mid] ≤ nums[low]` 。如果 mid 落在了后一段数值比较小的区间内了，那么一定有 `nums[mid] < nums[high]`，如果是落在前面一段数值比较大的区间内，`nums[mid] ≤ nums[high]` 。还有 `nums[low] == nums[mid]` 和 `nums[high] == nums[mid]` 的情况，单独处理即可。最后找到则输出 mid，没有找到则输出 -1 。

代码

```go
package leetcode

func search33(nums []int, target int) int {
  if len(nums) == 0 {
    return -1
  }
  low, high := 0, len(nums)-1
  for low <= high {
```

```
    mid := low + (high-low)>>1
    if nums[mid] == target {
      return mid
    } else if nums[mid] > nums[low] { // 在数值大的一部分区间里
      if nums[low] <= target && target < nums[mid] {
        high = mid - 1
      } else {
        low = mid + 1
      }
    } else if nums[mid] < nums[high] { // 在数值小的一部分区间里
      if nums[mid] < target && target <= nums[high] {
        low = mid + 1
      } else {
        high = mid - 1
      }
    } else {
      if nums[low] == nums[mid] {
        low++
      }
      if nums[high] == nums[mid] {
        high--
      }
    }
  }
  return -1
}
```

34. Find First and Last Position of Element in Sorted Array

题目

Given an array of integers `nums` sorted in ascending order, find the starting and ending position of a given `target` value.

Your algorithm's runtime complexity must be in the order of $O(\log n)$.

If the target is not found in the array, return `[-1, -1]`.

Example 1:

```
Input: nums = [5,7,7,8,8,10], target = 8
Output: [3,4]
```

Example 2:

```
Input: nums = [5,7,7,8,8,10], target = 6
Output: [-1,-1]
```

题目大意

给定一个按照升序排列的整数数组 nums，和一个目标值 target。找出给定目标值在数组中的开始位置和结束位置。你的算法时间复杂度必须是 O(log n) 级别。如果数组中不存在目标值，返回 [-1, -1]。

解题思路

- 给出一个有序数组 `nums` 和一个数 `target`，要求在数组中找到第一个和这个元素相等的元素下标，最后一个和这个元素相等的元素下标。
- 这一题是经典的二分搜索变种题。二分搜索有 4 大基础变种题：
  1. 查找第一个值等于给定值的元素
  2. 查找最后一个值等于给定值的元素
  3. 查找第一个大于等于给定值的元素
  4. 查找最后一个小于等于给定值的元素

  这一题的解题思路可以分别利用变种 1 和变种 2 的解法就可以做出此题。或者用一次变种 1 的方法，然后循环往后找到最后一个与给定值相等的元素。不过后者这种方法可能会使时间复杂度下降到 O(n)，因为有可能数组中 n 个元素都和给定元素相同。(4 大基础变种的实现见代码)

代码

```go
package leetcode

func searchRange(nums []int, target int) []int {
  return []int{searchFirstEqualElement(nums, target), searchLastEqualElement(nums, target)}

}

// 二分查找第一个与 target 相等的元素，时间复杂度 O(logn)
func searchFirstEqualElement(nums []int, target int) int {
  low, high := 0, len(nums)-1
  for low <= high {
    mid := low + ((high - low) >> 1)
    if nums[mid] > target {
      high = mid - 1
    } else if nums[mid] < target {
      low = mid + 1
    } else {
      if (mid == 0) || (nums[mid-1] != target) { // 找到第一个与 target 相等的元素
        return mid
      }
      high = mid - 1
```

```go
    }
  }
  return -1
}

// 二分查找最后一个与 target 相等的元素，时间复杂度 O(logn)
func searchLastEqualElement(nums []int, target int) int {
  low, high := 0, len(nums)-1
  for low <= high {
    mid := low + ((high - low) >> 1)
    if nums[mid] > target {
      high = mid - 1
    } else if nums[mid] < target {
      low = mid + 1
    } else {
      if (mid == len(nums)-1) || (nums[mid+1] != target) { // 找到最后一个与 target 相等的
元素
        return mid
      }
      low = mid + 1
    }
  }
  return -1
}

// 二分查找第一个大于等于 target 的元素，时间复杂度 O(logn)
func searchFirstGreaterElement(nums []int, target int) int {
  low, high := 0, len(nums)-1
  for low <= high {
    mid := low + ((high - low) >> 1)
    if nums[mid] >= target {
      if (mid == 0) || (nums[mid-1] < target) { // 找到第一个大于等于 target 的元素
        return mid
      }
      high = mid - 1
    } else {
      low = mid + 1
    }
  }
  return -1
}

// 二分查找最后一个小于等于 target 的元素，时间复杂度 O(logn)
func searchLastLessElement(nums []int, target int) int {
  low, high := 0, len(nums)-1
  for low <= high {
    mid := low + ((high - low) >> 1)
    if nums[mid] <= target {
```

```
      if (mid == len(nums)-1) || (nums[mid+1] > target) { // 找到最后一个小于等于 target 的元素
        return mid
      }
      low = mid + 1
    } else {
      high = mid - 1
    }
  }
  return -1
}
```

35. Search Insert Position

题目

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
```

题目大意

给定一个排序数组和一个目标值，在数组中找到目标值，并返回其索引。如果目标值不存在于数组中，返回它将会被按顺序插入的位置。

你可以假设数组中无重复元素。

解题思路

- 给出一个已经从小到大排序后的数组，要求在数组中找到插入 target 元素的位置。
- 这一题是经典的二分搜索的变种题，在有序数组中找到最后一个比 target 小的元素。

代码

```
package leetcode

func searchInsert(nums []int, target int) int {
  low, high := 0, len(nums)-1
  for low <= high {
    mid := low + (high-low)>>1
    if nums[mid] >= target {
      high = mid - 1
    } else {
      if (mid == len(nums)-1) || (nums[mid+1] >= target) {
        return mid + 1
      }
      low = mid + 1
    }
  }
  return 0
}
```

36. Valid Sudoku

题目

Determine if a 9x9 Sudoku board is valid. Only the filled cells need to be validated **according to the following rules**:

1. Each row must contain the digits `1-9` without repetition.
2. Each column must contain the digits `1-9` without repetition.
3. Each of the 9 `3x3` sub-boxes of the grid must contain the digits `1-9` without repetition.

A partially filled sudoku which is valid.

The Sudoku board could be partially filled, where empty cells are filled with the character `'.'`.

Example 1:

```
Input:
[
  ["5","3",".",".","7",".",".",".",."],
  ["6",".",".","1","9","5",".",".",."],
  [".","9","8",".",".",".",".","6",."],
  ["8",".",".",".","6",".",".",".","3"],
  ["4",".",".","8",".","3",".",".","1"],
  ["7",".",".",".","2",".",".",".","6"],
  [".","6",".",".",".",".","2","8",."],
  [".",".",".","4","1","9",".",".","5"],
  [".",".",".",".","8",".",".","7","9"]
]
Output: true
```

Example 2:

```
Input:
[
  ["8","3",".",".","7",".",".",".",."],
  ["6",".",".","1","9","5",".",".",."],
  [".","9","8",".",".",".",".","6",."],
  ["8",".",".",".","6",".",".",".","3"],
  ["4",".",".","8",".","3",".",".","1"],
  ["7",".",".",".","2",".",".",".","6"],
  [".","6",".",".",".",".","2","8",."],
  [".",".",".","4","1","9",".",".","5"],
  [".",".",".",".","8",".",".","7","9"]
]
Output: false
```

```
Explanation: Same as Example 1, except with the 5 in the top left corner being
    modified to 8. Since there are two 8's in the top left 3x3 sub-box, it is invalid.
```

Note:

- A Sudoku board (partially filled) could be valid but is not necessarily solvable.
- Only the filled cells need to be validated according to the mentioned rules.
- The given board contain only digits `1-9` and the character `'.'`.
- The given board size is always `9x9`.

题目大意

判断一个 9x9 的数独是否有效。只需要根据以下规则，验证已经填入的数字是否有效即可。

1. 数字 1-9 在每一行只能出现一次。
2. 数字 1-9 在每一列只能出现一次。
3. 数字 1-9 在每一个以粗实线分隔的 3x3 宫内只能出现一次。

解题思路

- 给出一个数独的棋盘，要求判断这个棋盘当前是否满足数独的要求：即行列是否都只包含 1-9，每个九宫格里面是否也只包含 1-9 。
- 注意这题和第 37 题是不同的，这一题是判断当前棋盘状态是否满足数独的要求，而第 37 题是要求求解数独。本题中的棋盘有些是无解的，但是棋盘状态是满足题意的。

代码

```go
package leetcode

import "strconv"

// 解法一 暴力遍历，时间复杂度 O(n^3)
func isValidSudoku(board [][]byte) bool {
  // 判断行 row
  for i := 0; i < 9; i++ {
    tmp := [10]int{}
    for j := 0; j < 9; j++ {
      cellVal := board[i][j : j+1]
      if string(cellVal) != "." {
        index, _ := strconv.Atoi(string(cellVal))
        if index > 9 || index < 1 {
          return false
        }
        if tmp[index] == 1 {
          return false
        }
        tmp[index] = 1
      }
```

```go
    }
  }
  // 判断列 column
  for i := 0; i < 9; i++ {
    tmp := [10]int{}
    for j := 0; j < 9; j++ {
      cellVal := board[j][i]
      if string(cellVal) != "." {
        index, _ := strconv.Atoi(string(cellVal))
        if index > 9 || index < 1 {
          return false
        }
        if tmp[index] == 1 {
          return false
        }
        tmp[index] = 1
      }
    }
  }
  // 判断 9宫格 3X3 cell
  for i := 0; i < 3; i++ {
    for j := 0; j < 3; j++ {
      tmp := [10]int{}
      for ii := i * 3; ii < i*3+3; ii++ {
        for jj := j * 3; jj < j*3+3; jj++ {
          cellVal := board[ii][jj]
          if string(cellVal) != "." {
            index, _ := strconv.Atoi(string(cellVal))
            if tmp[index] == 1 {
              return false
            }
            tmp[index] = 1
          }
        }
      }
    }
  }
  return true
}

// 解法二 添加缓存，时间复杂度 O(n^2)
func isValidSudoku1(board [][]byte) bool {
  rowbuf, colbuf, boxbuf := make([][]bool, 9), make([][]bool, 9), make([][]bool, 9)
  for i := 0; i < 9; i++ {
    rowbuf[i] = make([]bool, 9)
    colbuf[i] = make([]bool, 9)
    boxbuf[i] = make([]bool, 9)
  }
  // 遍历一次，添加缓存
```

```
  for r := 0; r < 9; r++ {
    for c := 0; c < 9; c++ {
      if board[r][c] != '.' {
        num := board[r][c] - '0' - byte(1)
        if rowbuf[r][num] || colbuf[c][num] || boxbuf[r/3*3+c/3][num] {
          return false
        }
        rowbuf[r][num] = true
        colbuf[c][num] = true
        boxbuf[r/3*3+c/3][num] = true // r,c 转换到box方格中
      }
    }
  }
  return true
}
```

37. Sudoku Solver

题目

Write a program to solve a Sudoku puzzle by filling the empty cells.

A sudoku solution must satisfy **all of the following rules**:

1. Each of the digits `1-9` must occur exactly once in each row.
2. Each of the digits `1-9` must occur exactly once in each column.
3. Each of the the digits `1-9` must occur exactly once in each of the 9 `3x3` sub-boxes of the grid.

Empty cells are indicated by the character `'.'`.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

A sudoku puzzle...

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

...and its solution numbers marked in red.

Note:

- The given board contain only digits `1-9` and the character `'.'`.
- You may assume that the given Sudoku puzzle will have a single unique solution.
- The given board size is always `9x9`.

题目大意

编写一个程序，通过已填充的空格来解决数独问题。一个数独的解法需遵循如下规则：

1. 数字 1-9 在每一行只能出现一次。
2. 数字 1-9 在每一列只能出现一次。
3. 数字 1-9 在每一个以粗实线分隔的 3x3 宫内只能出现一次。

空白格用 '.' 表示。

解题思路

- 给出一个数独谜题，要求解出这个数独
- 解题思路 DFS 暴力回溯枚举。数独要求每横行，每竖行，每九宫格内，`1-9` 的数字不能重复，每次放下一个数字的时候，在这 3 个地方都需要判断一次。
- 另外找到一组解以后就不需要再继续回溯了，直接返回即可。

代码

```
package leetcode

type position struct {
  x int
  y int
}

func solveSudoku(board [][]byte) {
```

```go
  pos, find := []position{}, false
  for i := 0; i < len(board); i++ {
    for j := 0; j < len(board[0]); j++ {
      if board[i][j] == '.' {
        pos = append(pos, position{x: i, y: j})
      }
    }
  }
  putSudoku(&board, pos, 0, &find)
}

func putSudoku(board *[][]byte, pos []position, index int, succ *bool) {
  if *succ == true {
    return
  }
  if index == len(pos) {
    *succ = true
    return
  }
  for i := 1; i < 10; i++ {
    if checkSudoku(board, pos[index], i) && !*succ {
      (*board)[pos[index].x][pos[index].y] = byte(i) + '0'
      putSudoku(board, pos, index+1, succ)
      if *succ == true {
        return
      }
      (*board)[pos[index].x][pos[index].y] = '.'
    }
  }
}

func checkSudoku(board *[][]byte, pos position, val int) bool {
  // 判断横行是否有重复数字
  for i := 0; i < len((*board)[0]); i++ {
    if (*board)[pos.x][i] != '.' && int((*board)[pos.x][i]-'0') == val {
      return false
    }
  }
  // 判断竖行是否有重复数字
  for i := 0; i < len((*board)); i++ {
    if (*board)[i][pos.y] != '.' && int((*board)[i][pos.y]-'0') == val {
      return false
    }
  }
  // 判断九宫格是否有重复数字
  posx, posy := pos.x-pos.x%3, pos.y-pos.y%3
  for i := posx; i < posx+3; i++ {
    for j := posy; j < posy+3; j++ {
      if (*board)[i][j] != '.' && int((*board)[i][j]-'0') == val {
```

```
            return false
        }
      }
    }
    return true
  }
```

39. Combination Sum

题目

Given a **set** of candidate numbers (`candidates`) **(without duplicates)** and a target number (`target`), find all unique combinations in `candidates` where the candidate numbers sums to `target`.

The **same** repeated number may be chosen from `candidates` unlimited number of times.

Note:

- All numbers (including `target`) will be positive integers.
- The solution set must not contain duplicate combinations.

Example 1:

```
Input: candidates = [2,3,6,7], target = 7,
A solution set is:
[
  [7],
  [2,2,3]
]
```

Example 2:

```
Input: candidates = [2,3,5], target = 8,
A solution set is:
[
  [2,2,2,2],
  [2,3,3],
  [3,5]
]
```

题目大意

给定一个无重复元素的数组 candidates 和一个目标数 target ， 找出 candidates 中所有可以使数字和为 target 的组合。

candidates 中的数字可以无限制重复被选取。

解题思路

- 题目要求出总和为 sum 的所有组合，组合需要去重。
- 这一题和第 47 题类似，只不过元素可以反复使用。

代码

```
package leetcode

import "sort"

func combinationSum(candidates []int, target int) [][]int {
  if len(candidates) == 0 {
    return [][]int{}
  }
  c, res := []int{}, [][]int{}
  sort.Ints(candidates)
  findcombinationSum(candidates, target, 0, c, &res)
  return res
}

func findcombinationSum(nums []int, target, index int, c []int, res *[][]int) {
  if target <= 0 {
    if target == 0 {
      b := make([]int, len(c))
      copy(b, c)
      *res = append(*res, b)
    }
    return
  }
  for i := index; i < len(nums); i++ {
    if nums[i] > target { // 这里可以剪枝优化
      break
    }
    c = append(c, nums[i])
    findcombinationSum(nums, target-nums[i], i, c, res) // 注意这里迭代的时候 index 依旧不变，因为一个元素可以取多次
    c = c[:len(c)-1]
  }
}
```

40. Combination Sum II

题目

Given a collection of candidate numbers (`candidates`) and a target number (`target`), find all unique combinations in `candidates` where the candidate numbers sums to `target`.

Each number in `candidates` may only be used **once** in the combination.

Note:

- All numbers (including `target`) will be positive integers.
- The solution set must not contain duplicate combinations.

Example 1:

```
Input: candidates = [10,1,2,7,6,1,5], target = 8,
A solution set is:
[
  [1, 7],
  [1, 2, 5],
  [2, 6],
  [1, 1, 6]
]
```

Example 2:

```
Input: candidates = [2,5,2,1,2], target = 5,
A solution set is:
[
  [1,2,2],
  [5]
]
```

题目大意

给定一个数组 candidates 和一个目标数 target ，找出 candidates 中所有可以使数字和为 target 的组合。

candidates 中的每个数字在每个组合中只能使用一次。

解题思路

- 题目要求出总和为 sum 的所有组合，组合需要去重。这一题是第 39 题的加强版，第 39 题中元素可以重复利用(重复元素可无限次使用)，这一题中元素只能有限次数的利用，因为存在重复元素，并且每个元素只能用一次(重复元素只能使用有限次)
- 这一题和第 47 题类似，只不过元素可以反复使用。

代码

```go
package leetcode

import (
```

```
  "sort"
)

func combinationSum2(candidates []int, target int) [][]int {
  if len(candidates) == 0 {
    return [][]int{}
  }
  c, res := []int{}, [][]int{}
  sort.Ints(candidates) // 这里是去重的关键逻辑
  findcombinationSum2(candidates, target, 0, c, &res)
  return res
}

func findcombinationSum2(nums []int, target, index int, c []int, res *[][]int) {
  if target == 0 {
    b := make([]int, len(c))
    copy(b, c)
    *res = append(*res, b)
    return
  }
  for i := index; i < len(nums); i++ {
    if i > index && nums[i] == nums[i-1] { // 这里是去重的关键逻辑,本次不取重复数字, 下次循环可能会取重复数字
      continue
    }
    if target >= nums[i] {
      c = append(c, nums[i])
      findcombinationSum2(nums, target-nums[i], i+1, c, res)
      c = c[:len(c)-1]
    }
  }
}
```

41. First Missing Positive

题目

Given an unsorted integer array, find the smallest missing positive integer.

Example 1:

```
Input: [1,2,0]
Output: 3
```

Example 2:

```
Input: [3,4,-1,1]
Output: 2
```

Example 3:

```
Input: [7,8,9,11,12]
Output: 1
```

Note:

Your algorithm should run in O(n) time and uses constant extra space.

题目大意

找到缺失的第一个正整数。

解题思路

为了减少时间复杂度，可以把 input 数组都装到 map 中，然后 i 循环从 1 开始，依次比对 map 中是否存在 i，只要不存在 i 就立即返回结果，即所求。

代码

```go
package leetcode

func firstMissingPositive(nums []int) int {
  numMap := make(map[int]int, len(nums))
  for _, v := range nums {
    numMap[v] = v
  }
  for index := 1; index < len(nums)+1; index++ {
    if _, ok := numMap[index]; !ok {
      return index
    }
  }
  return len(nums) + 1
}
```

42. Trapping Rain Water

题目

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

The above elevation map is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped. Thanks Marcos for contributing this image!

Example:

```
Input: [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
```

题目大意

从 x 轴开始，给出一个数组，数组里面的数字代表从 (0,0) 点开始，宽度为 1 个单位，高度为数组元素的值。如果下雨了，问这样一个容器能装多少单位的水?

解题思路

- 每个数组里面的元素值可以想象成一个左右都有壁的圆柱筒。例如下图中左边的第二个元素 1，当前左边最大的元素是 2 ，所以 2 高度的水会装到 1 的上面(因为想象成了左右都有筒壁)。这道题的思路就是左指针从 0 开始往右扫，右指针从最右边开始往左扫。额外还需要 2 个变量分别记住左边最大的高度和右边最大高度。遍历扫数组元素的过程中，如果左指针的高度比右指针的高度小，就不断的移动左指针，否则移动右指针。循环的终止条件就是左右指针碰上以后就结束。只要数组中元素的高度比保存的局部最大高度小，就累加 res 的值，否则更新局部最大高度。最终解就是 res 的值。

- 抽象一下，本题是想求针对每个 i，找到它左边最大值 leftMax，右边的最大值 rightMax，然后 min(leftMax，rightMax) 为能够接到水的高度。left 和 right 指针是两边往中间移动的游标指针。最傻的解题思路是针对每个下标 i，往左循环找到第一个最大值，往右循环找到第一个最大值，然后把这两个最大值取出最小者，即为当前雨水的高度。这样做时间复杂度高，浪费了很多循环。i 在从左往右的过程中，是可以动态维护最大值的。右边的最大值用右边的游标指针来维护。从左往右扫一遍下标，和，从两边往中间遍历一遍下标，是相同的结果，每个下标都遍历了一次。

- 每个 i 的宽度固定为 1，所以每个“坑”只需要求出高度，即当前这个“坑”能积攒的雨水。最后依次将每个“坑”中的雨水相加即是能接到的雨水数。

代码

```
package leetcode

func trap(height []int) int {
  res, left, right, maxLeft, maxRight := 0, 0, len(height)-1, 0, 0
  for left <= right {
    if height[left] <= height[right] {
      if height[left] > maxLeft {
        maxLeft = height[left]
      } else {
        res += maxLeft - height[left]
      }
      left++
    } else {
      if height[right] >= maxRight {
        maxRight = height[right]
      } else {
        res += maxRight - height[right]
      }
      right--
    }
  }
  return res
}
```

43. Multiply Strings

题目

Given two non-negative integers `num1` and `num2` represented as strings, return the product of `num1` and `num2`, also represented as a string.

Note: You must not use any built-in BigInteger library or convert the inputs to integer directly.

Example 1:

```
Input: num1 = "2", num2 = "3"
Output: "6"
```

Example 2:

```
Input: num1 = "123", num2 = "456"
Output: "56088"
```

Constraints:

- `1 <= num1.length, num2.length <= 200`
- `num1` and `num2` consist of digits only.
- Both `num1` and `num2` do not contain any leading zero, except the number `0` itself.

题目大意

给定两个以字符串形式表示的非负整数 num1 和 num2，返回 num1 和 num2 的乘积，它们的乘积也表示为字符串形式。

解题思路

- 用数组模拟乘法。创建一个数组长度为 `len(num1) + len(num2)` 的数组用于存储乘积。对于任意 `0 ≤ i < len(num1)`，`0 ≤ j < len(num2)`，`num1[i] * num2[j]` 的结果位于 `tmp[i+j+1]`，如果 `tmp[i+j+1]≥10`，则将进位部分加到 `tmp[i+j]`。最后，将数组 `tmp` 转成字符串，如果最高位是 0 则舍弃最高位。

代码

```go
package leetcode

func multiply(num1 string, num2 string) string {
  if num1 == "0" || num2 == "0" {
    return "0"
  }
  b1, b2, tmp := []byte(num1), []byte(num2), make([]int, len(num1)+len(num2))
  for i := 0; i < len(b1); i++ {
    for j := 0; j < len(b2); j++ {
      tmp[i+j+1] += int(b1[i]-'0') * int(b2[j]-'0')
    }
  }
  for i := len(tmp) - 1; i > 0; i-- {
    tmp[i-1] += tmp[i] / 10
    tmp[i] = tmp[i] % 10
  }
  if tmp[0] == 0 {
    tmp = tmp[1:]
  }
```

```
  res := make([]byte, len(tmp))
  for i := 0; i < len(tmp); i++ {
    res[i] = '0' + byte(tmp[i])
  }
  return string(res)
}
```

45. Jump Game II

题目

Given an array of non-negative integers `nums`, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Your goal is to reach the last index in the minimum number of jumps.

You can assume that you can always reach the last index.

Example 1:

```
Input: nums = [2,3,1,1,4]
Output: 2
Explanation: The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1, then 3 steps to the last index.
```

Example 2:

```
Input: nums = [2,3,0,1,4]
Output: 2
```

Constraints:

- `1 <= nums.length <= 1000`
- `0 <= nums[i] <= 10^5`

题目大意

给定一个非负整数数组，你最初位于数组的第一个位置。数组中的每个元素代表你在该位置可以跳跃的最大长度。你的目标是使用最少的跳跃次数到达数组的最后一个位置。

解题思路

- 要求找到最少跳跃次数，顺理成章的会想到用贪心算法解题。扫描步数数组，维护当前能够到达最大下标的位置，记为能到达的最远边界，如果扫描过程中到达了最远边界，更新边界并将跳跃次数 + 1。
- 扫描数组的时候，其实不需要扫描最后一个元素，因为在跳到最后一个元素之前，能到达的最远边界一定大于等于最后一个元素的位置，不然就跳不到最后一个元素，到达不了终点了；如果遍历到最后一个元素，说明边界正好为最后一个位置，最终跳跃次数直接 + 1 即可，也不需要访问最后一个元素。

代码

```go
package leetcode

func jump(nums []int) int {
  if len(nums) == 1 {
    return 0
  }
  needChoose, canReach, step := 0, 0, 0
  for i, x := range nums {
    if i+x > canReach {
      canReach = i + x
      if canReach >= len(nums)-1 {
        return step + 1
      }
    }
    if i == needChoose {
      needChoose = canReach
      step++
    }
  }
  return step
}
```

46. Permutations

题目

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]
]
```

题目大意

给定一个没有重复数字的序列，返回其所有可能的全排列。

解题思路

- 求出一个数组的排列组合中的所有排列，用 DFS 深搜即可。

代码

```
package leetcode

func permute(nums []int) [][]int {
  if len(nums) == 0 {
    return [][]int{}
  }
  used, p, res := make([]bool, len(nums)), []int{}, [][]int{}
  generatePermutation(nums, 0, p, &res, &used)
  return res
}

func generatePermutation(nums []int, index int, p []int, res *[][]int, used *[]bool) {
  if index == len(nums) {
    temp := make([]int, len(p))
    copy(temp, p)
    *res = append(*res, temp)
    return
  }
  for i := 0; i < len(nums); i++ {
    if !(*used)[i] {
      (*used)[i] = true
      p = append(p, nums[i])
      generatePermutation(nums, index+1, p, res, used)
      p = p[:len(p)-1]
      (*used)[i] = false
    }
  }
  return
}
```

47. Permutations II

题目

Given a collection of numbers that might contain duplicates, return all possible unique permutations.

Example:

```
Input: [1,1,2]
Output:
[
  [1,1,2],
  [1,2,1],
  [2,1,1]
]
```

题目大意

给定一个可包含重复数字的序列，返回所有不重复的全排列。

解题思路

- 这一题是第 46 题的加强版，第 46 题中求数组的排列，数组中元素不重复，但是这一题中，数组元素会重复，所以需要最终排列出来的结果需要去重。
- 去重的方法是经典逻辑，将数组排序以后，判断重复元素再做逻辑判断。
- 其他思路和第 46 题完全一致，DFS 深搜即可。

代码

```go
package leetcode

import "sort"

func permuteUnique(nums []int) [][]int {
  if len(nums) == 0 {
    return [][]int{}
  }
  used, p, res := make([]bool, len(nums)), []int{}, [][]int{}
  sort.Ints(nums) // 这里是去重的关键逻辑
  generatePermutation47(nums, 0, p, &res, &used)
  return res
}

func generatePermutation47(nums []int, index int, p []int, res *[][]int, used *[]bool)
{
  if index == len(nums) {
    temp := make([]int, len(p))
    copy(temp, p)
    *res = append(*res, temp)
    return
  }
  for i := 0; i < len(nums); i++ {
    if !(*used)[i] {
      if i > 0 && nums[i] == nums[i-1] && !(*used)[i-1] { // 这里是去重的关键逻辑
        continue
```

```
        }
        (*used)[i] = true
        p = append(p, nums[i])
        generatePermutation47(nums, index+1, p, res, used)
        p = p[:len(p)-1]
        (*used)[i] = false
      }
    }
    return
  }
```

48. Rotate Image

题目

You are given an *n* x *n* 2D matrix representing an image.

Rotate the image by 90 degrees (clockwise).

Note:

You have to rotate the image in-place, which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

Example 1:

```
Given input matrix =
[
  [1,2,3],
  [4,5,6],
  [7,8,9]
],

rotate the input matrix in-place such that it becomes:
[
  [7,4,1],
  [8,5,2],
  [9,6,3]
]
```

Example 2:

```
Given input matrix =
[
  [ 5, 1, 9,11],
  [ 2, 4, 8,10],
  [13, 3, 6, 7],
  [15,14,12,16]
],

rotate the input matrix in-place such that it becomes:
[
  [15,13, 2, 5],
  [14, 3, 4, 1],
  [12, 6, 8, 9],
  [16, 7,10,11]
]
```

题目大意

给定一个 n × n 的二维矩阵表示一个图像。将图像顺时针旋转 90 度。说明：你必须在原地旋转图像，这意味着你需要直接修改输入的二维矩阵。请不要使用另一个矩阵来旋转图像。

解题思路

- 给出一个二维数组，要求顺时针旋转 90 度。
- 这一题比较简单，按照题意做就可以。这里给出 2 种旋转方法的实现，顺时针旋转和逆时针旋转。

```
/*
 * clockwise rotate 顺时针旋转
 * first reverse up to down, then swap the symmetry
 * 1 2 3     7 8 9     7 4 1
 * 4 5 6  => 4 5 6  => 8 5 2
 * 7 8 9     1 2 3     9 6 3
*/
void rotate(vector<vector<int> > &matrix) {
    reverse(matrix.begin(), matrix.end());
    for (int i = 0; i < matrix.size(); ++i) {
        for (int j = i + 1; j < matrix[i].size(); ++j)
            swap(matrix[i][j], matrix[j][i]);
    }
}

/*
 * anticlockwise rotate 逆时针旋转
 * first reverse left to right, then swap the symmetry
 * 1 2 3     3 2 1     3 6 9
 * 4 5 6  => 6 5 4  => 2 5 8
 * 7 8 9     9 8 7     1 4 7
*/
void anti_rotate(vector<vector<int> > &matrix) {
    for (auto vi : matrix) reverse(vi.begin(), vi.end());
    for (int i = 0; i < matrix.size(); ++i) {
        for (int j = i + 1; j < matrix[i].size(); ++j)
            swap(matrix[i][j], matrix[j][i]);
    }
}
```

代码

```
package leetcode

func rotate(matrix [][]int) {
  length := len(matrix)
  // rotate by diagonal 对角线变换
  for i := 0; i < length; i++ {
```

```
    for j := i + 1; j < length; j++ {
      matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]
    }
  }
  // rotate by vertical centerline 竖直轴对称翻转
  for i := 0; i < length; i++ {
    for j := 0; j < length/2; j++ {
      matrix[i][j], matrix[i][length-j-1] = matrix[i][length-j-1], matrix[i][j]
    }
  }
}
```

49. Group Anagrams

题目

Given an array of strings, group anagrams together.

Example:

```
Input: ["eat", "tea", "tan", "ate", "nat", "bat"],
Output:
[
  ["ate","eat","tea"],
  ["nat","tan"],
  ["bat"]
]
```

Note:

- All inputs will be in lowercase.
- The order of your output does not matter.

题目大意

给出一个字符串数组，要求对字符串数组里面有 Anagrams 关系的字符串进行分组。Anagrams 关系是指两个字符串的字符完全相同，顺序不同，两者是由排列组合组成。

解题思路

这道题可以将每个字符串都排序，排序完成以后，相同 Anagrams 的字符串必然排序结果一样。把排序以后的字符串当做 key 存入到 map 中。遍历数组以后，就能得到一个 map，key 是排序以后的字符串，value 对应的是这个排序字符串以后的 Anagrams 字符串集合。最后再将这些 value 对应的字符串数组输出即可。

代码

```go
package leetcode

import "sort"

type sortRunes []rune

func (s sortRunes) Less(i, j int) bool {
  return s[i] < s[j]
}

func (s sortRunes) Swap(i, j int) {
  s[i], s[j] = s[j], s[i]
}

func (s sortRunes) Len() int {
  return len(s)
}

func groupAnagrams(strs []string) [][]string {
  record, res := map[string][]string{}, [][]string{}
  for _, str := range strs {
    sByte := []rune(str)
    sort.Sort(sortRunes(sByte))
    sstrs := record[string(sByte)]
    sstrs = append(sstrs, str)
    record[string(sByte)] = sstrs
  }
  for _, v := range record {
    res = append(res, v)
  }
  return res
}
```

50. Pow(x, n)

题目

Implement pow(*x*, *n*), which calculates *x* raised to the power *n* (xn).

Example 1:

```
Input: 2.00000, 10
Output: 1024.00000
```

Example 2:

```
Input: 2.10000, 3
Output: 9.26100
```

Example 3:

```
Input: 2.00000, -2
Output: 0.25000
Explanation: 2-2 = 1/22 = 1/4 = 0.25
```

Note:

- -100.0 < *x* < 100.0
- *n* is a 32-bit signed integer, within the range [−2^31, 2^31− 1]

题目大意

实现 pow(x, n) ，即计算 x 的 n 次幂函数。

解题思路

- 要求计算 Pow(x, n)
- 这一题用递归的方式，不断的将 n 2 分下去。注意 n 的正负数，n 的奇偶性。

代码

```go
package leetcode

// 时间复杂度 O(log n),空间复杂度 O(1)
func myPow(x float64, n int) float64 {
  if n == 0 {
    return 1
  }
  if n == 1 {
    return x
  }
  if n < 0 {
    n = -n
    x = 1 / x
  }
  tmp := myPow(x, n/2)
  if n%2 == 0 {
    return tmp * tmp
  }
  return tmp * tmp * x
}
```

51. N-Queens

题目

The n-queens puzzle is the problem of placing n queens on an n×n chessboard such that no two queens attack each other.

One solution to the eight queens puzzle

Given an integer *n*, return all distinct solutions to the *n*-queens puzzle.

Each solution contains a distinct board configuration of the *n*-queens' placement, where `'Q'` and `'.'` both indicate a queen and an empty space respectively.

Example:

```
Input: 4
Output: [
 [".Q..",  // Solution 1
  "...Q",
  "Q...",
  "..Q."],

 ["..Q.",  // Solution 2
  "Q...",
  "...Q",
  ".Q.."]
]
Explanation: There exist two distinct solutions to the 4-queens puzzle as shown above.
```

题目大意

给定一个整数 n，返回所有不同的 n 皇后问题的解决方案。每一种解法包含一个明确的 n 皇后问题的棋子放置方案，该方案中 'Q' 和 '.' 分别代表了皇后和空位。

解题思路

- 求解 n 皇后问题
- 利用 col 数组记录列信息，col 有 `n` 列。用 dia1，dia2 记录从左下到右上的对角线，从左上到右下的对角线的信息，dia1 和 dia2 分别都有 `2*n-1` 个。
- dia1 对角线的规律是 `i + j 是定值`，例如[0,0]，为 0；[1,0]、[0,1] 为 1；[2,0]、[1,1]、[0,2] 为 2；
- dia2 对角线的规律是 `i - j 是定值`，例如[0,7]，为 -7；[0,6]、[1,7] 为 -6；[0,5]、[1,6]、[2,7] 为 -5；为了使他们从 0 开始，i - j + n - 1 偏移到 0 开始，所以 dia2 的规律是 `i - j + n - 1 为定值`。
- 还有一个位运算的方法，每行只能选一个位置放皇后，那么对每行遍历可能放皇后的位置。如何高效判断哪些点不能放皇后呢？这里的做法毕竟巧妙，把所有之前选过的点按照顺序存下来，然后根据之前选的点到当前行的距离，就可以快速判断是不是会有冲突。举个例子: 假如在 4 皇后问题中，如果第一二行已经选择了位置 [1, 3]，那么在第三行选择时，首先不能再选 1, 3 列了，而对于第三行， 1 距离长度为2，所以它会影响到 -1, 3 两个列。同理，3 在第二行，距离第三行为 1，所以 3 会影响到列 2, 4。由上面的结果，我们知道 -1, 4 超出边界了不用去管，别的不能选的点是 1, 2, 3，所以第三行就只能选 0。在代码实现中，可以在每次遍历前根据之前选择的情况生成一个 occupied 用来记录当前这一行，已经被选了的和由于之前皇后攻击范围所以不能选的位置，然后只选择合法的位置进入到下一层递归。另外就是预处理了一个皇后放不同位置的字符串，这样这些字符串在返回结果的时候是可以在内存中复用的，省一点内存。

代码

```go
package leetcode

// 解法一 DFS
func solveNQueens(n int) [][]string {
  col, dia1, dia2, row, res := make([]bool, n), make([]bool, 2*n-1), make([]bool, 2*n-1), []int{}, [][]string{}
  putQueen(n, 0, &col, &dia1, &dia2, &row, &res)
  return res
}

// 尝试在一个n皇后问题中，摆放第index行的皇后位置
func putQueen(n, index int, col, dia1, dia2 *[]bool, row *[]int, res *[][]string) {
  if index == n {
    *res = append(*res, generateBoard(n, row))
    return
  }
  for i := 0; i < n; i++ {
    // 尝试将第index行的皇后摆放在第i列
    if !(*col)[i] && !(*dia1)[index+i] && !(*dia2)[index-i+n-1] {
      *row = append(*row, i)
      (*col)[i] = true
      (*dia1)[index+i] = true
      (*dia2)[index-i+n-1] = true
      putQueen(n, index+1, col, dia1, dia2, row, res)
      (*col)[i] = false
      (*dia1)[index+i] = false
```

```go
      (*dia2)[index-i+n-1] = false
      *row = (*row)[:len(*row)-1]
    }
  }
  return
}

func generateBoard(n int, row *[]int) []string {
  board := []string{}
  res := ""
  for i := 0; i < n; i++ {
    res += "."
  }
  for i := 0; i < n; i++ {
    board = append(board, res)
  }
  for i := 0; i < n; i++ {
    tmp := []byte(board[i])
    tmp[(*row)[i]] = 'Q'
    board[i] = string(tmp)
  }
  return board
}

// 解法二 二进制操作法 Signed-off-by: Hanlin Shi shihanlin9@gmail.com
func solveNQueens2(n int) (res [][]string) {
  placements := make([]string, n)
  for i := range placements {
    buf := make([]byte, n)
    for j := range placements {
      if i == j {
        buf[j] = 'Q'
      } else {
        buf[j] = '.'
      }
    }
    placements[i] = string(buf)
  }
  var construct func(prev []int)
  construct = func(prev []int) {
    if len(prev) == n {
      plan := make([]string, n)
      for i := 0; i < n; i++ {
        plan[i] = placements[prev[i]]
      }
      res = append(res, plan)
      return
    }
    occupied := 0
```

```
    for i := range prev {
      dist := len(prev) - i
      bit := 1 << prev[i]
      occupied |= bit | bit<<dist | bit>>dist
    }
    prev = append(prev, -1)
    for i := 0; i < n; i++ {
      if (occupied>>i)&1 != 0 {
        continue
      }
      prev[len(prev)-1] = i
      construct(prev)
    }
  }
  construct(make([]int, 0, n))
  return
}
```

52. N-Queens II

题目

The n-queens puzzle is the problem of placing n queens on an n×n chessboard such that no two queens attack each other.

One solution to the eight queens puzzle

Given an integer n, return the number of distinct solutions to the n-queens puzzle.

Example:

```
Input: 4
Output: 2
Explanation: There are two distinct solutions to the 4-queens puzzle as shown below.
[
```

```
 [".Q..",  // Solution 1
  "...Q",
  "Q...",
  "..Q."],

 ["..Q.",  // Solution 2
  "Q...",
  "...Q",
  ".Q.."]
]
```

题目大意

给定一个整数 n，返回 n 皇后不同的解决方案的数量。

解题思路

- 这一题是第 51 题的加强版，在第 51 题的基础上累加记录解的个数即可。
- 这一题也可以暴力打表法，时间复杂度为 O(1)。

代码

```go
package leetcode

// 解法一，暴力打表法
func totalNQueens(n int) int {
  res := []int{0, 1, 0, 0, 2, 10, 4, 40, 92, 352, 724}
  return res[n]
}

// 解法二，DFS 回溯法
func totalNQueens1(n int) int {
  col, dia1, dia2, row, res := make([]bool, n), make([]bool, 2*n-1), make([]bool, 2*n-
1), []int{}, 0
  putQueen52(n, 0, &col, &dia1, &dia2, &row, &res)
  return res
}

// 尝试在一个n皇后问题中，摆放第index行的皇后位置
func putQueen52(n, index int, col, dia1, dia2 *[]bool, row *[]int, res *int) {
  if index == n {
    *res++
    return
  }
  for i := 0; i < n; i++ {
    // 尝试将第index行的皇后摆放在第i列
    if !(*col)[i] && !(*dia1)[index+i] && !(*dia2)[index-i+n-1] {
```

```
        *row = append(*row, i)
        (*col)[i] = true
        (*dia1)[index+i] = true
        (*dia2)[index-i+n-1] = true
        putQueen52(n, index+1, col, dia1, dia2, row, res)
        (*col)[i] = false
        (*dia1)[index+i] = false
        (*dia2)[index-i+n-1] = false
        *row = (*row)[:len(*row)-1]
      }
    }
    return
  }

  // 解法三 二进制位操作法
  // class Solution {
  //  public:
  //    int totalNQueens(int n) {
  //      int ans=0;
  //      int row=0,leftDiagonal=0,rightDiagonal=0;
  //      int bit=(1<<n)-1;//to clear high bits of the 32-bit int
  //      Queens(bit,row,leftDiagonal,rightDiagonal,ans);
  //      return ans;
  //    }
  //    void Queens(int bit,int row,int leftDiagonal,int rightDiagonal,int &ans){
  //      int cur=(~(row|leftDiagonal|rightDiagonal))&bit;//possible place for this queen
  //      if (!cur) return;//no pos for this queen
  //      while(cur){
  //        int curPos=(cur&(~cur + 1))&bit;//choose possible place in the right
  //        //update row,ld and rd
  //        row+=curPos;
  //        if (row==bit) ans++;//last row
  //        else Queens(bit,row,((leftDiagonal|curPos)<<1)&bit,
  ((rightDiagonal|curPos)>>1)&bit,ans);
  //        cur-=curPos;//for next possible place
  //        row-=curPos;//reset row
  //      }
  //    }
  // };
```

53. Maximum Subarray

题目

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.

题目大意

给定一个整数数组 nums ，找到一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。

解题思路

- 这一题可以用 DP 求解也可以不用 DP。
- 题目要求输出数组中某个区间内数字之和最大的那个值。`dp[i]` 表示 `[0,i]` 区间内各个子区间和的最大值，状态转移方程是 `dp[i] = nums[i] + dp[i-1] (dp[i-1] > 0)`，`dp[i] = nums[i] (dp[i-1] ≤ 0)`。

代码

```go
package leetcode

// 解法一 DP
func maxSubArray(nums []int) int {
  if len(nums) == 0 {
    return 0
  }
  if len(nums) == 1 {
    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 {
      dp[i] = nums[i]
    }
    res = max(res, dp[i])
  }
  return res
}

// 解法二 模拟
func maxSubArray1(nums []int) int {
```

```
  if len(nums) == 1 {
    return nums[0]
  }
  maxSum, res, p := nums[0], 0, 0
  for p < len(nums) {
    res += nums[p]
    if res > maxSum {
      maxSum = res
    }
    if res < 0 {
      res = 0
    }
    p++
  }
  return maxSum
}
```

54. Spiral Matrix

题目

Given a matrix of *m* x *n* elements (*m* rows, *n* columns), return all elements of the matrix in spiral order.

Example 1:

```
Input:
[
 [ 1, 2, 3 ],
 [ 4, 5, 6 ],
 [ 7, 8, 9 ]
]
Output: [1,2,3,6,9,8,7,4,5]
```

Example 2:

```
Input:
[
  [1, 2, 3, 4],
  [5, 6, 7, 8],
  [9,10,11,12]
]
Output: [1,2,3,4,8,12,11,10,9,5,6,7]
```

题目大意

给定一个包含 m x n 个元素的矩阵（m 行, n 列），请按照顺时针螺旋顺序，返回矩阵中的所有元素。

解题思路

- 给出一个二维数组，按照螺旋的方式输出
- 解法一：需要注意的是特殊情况，比如二维数组退化成一维或者一列或者一个元素。注意了这些情况，基本就可以一次通过了。
- 解法二：提前算出一共多少个元素，一圈一圈地遍历矩阵，停止条件就是遍历了所有元素（count == sum）

代码

```
package leetcode

// 解法 1
func spiralOrder(matrix [][]int) []int {
  if len(matrix) == 0 {
    return []int{}
  }
  res := []int{}
  if len(matrix) == 1 {
    for i := 0; i < len(matrix[0]); i++ {
      res = append(res, matrix[0][i])
    }
    return res
  }
  if len(matrix[0]) == 1 {
    for i := 0; i < len(matrix); i++ {
      res = append(res, matrix[i][0])
    }
    return res
  }
  visit, m, n, round, x, y, spDir := make([][]int, len(matrix)), len(matrix),
len(matrix[0]), 0, 0, 0, [][]int{
    []int{0, 1},  // 朝右
    []int{1, 0},  // 朝下
    []int{0, -1}, // 朝左
    []int{-1, 0}, // 朝上
  }
  for i := 0; i < m; i++ {
    visit[i] = make([]int, n)
  }
  visit[x][y] = 1
  res = append(res, matrix[x][y])
  for i := 0; i < m*n; i++ {
    x += spDir[round%4][0]
    y += spDir[round%4][1]
    if (x == 0 && y == n-1) || (x == m-1 && y == n-1) || (y == 0 && x == m-1) {
      round++
    }
```

```go
    if x > m-1 || y > n-1 || x < 0 || y < 0 {
      return res
    }
    if visit[x][y] == 0 {
      visit[x][y] = 1
      res = append(res, matrix[x][y])
    }
    switch round % 4 {
    case 0:
      if y+1 <= n-1 && visit[x][y+1] == 1 {
        round++
        continue
      }
    case 1:
      if x+1 <= m-1 && visit[x+1][y] == 1 {
        round++
        continue
      }
    case 2:
      if y-1 >= 0 && visit[x][y-1] == 1 {
        round++
        continue
      }
    case 3:
      if x-1 >= 0 && visit[x-1][y] == 1 {
        round++
        continue
      }
    }
  }
  return res
}

// 解法 2
func spiralOrder2(matrix [][]int) []int {
  m := len(matrix)
  if m == 0 {
    return nil
  }

  n := len(matrix[0])
  if n == 0 {
    return nil
  }

  // top、left、right、bottom 分别是剩余区域的上、左、右、下的下标
  top, left, bottom, right := 0, 0, m-1, n-1
  count, sum := 0, m*n
  res := []int{}
```

```
  // 外层循环每次遍历一圈
  for count < sum {
    i, j := top, left
    for j <= right && count < sum {
      res = append(res, matrix[i][j])
      count++
      j++
    }
    i, j = top + 1, right
    for i <= bottom && count < sum {
      res = append(res, matrix[i][j])
      count++
      i++
    }
    i, j = bottom, right - 1
    for j >= left && count < sum {
      res = append(res, matrix[i][j])
      count++
      j--
    }
    i, j = bottom - 1, left
    for i > top && count < sum {
      res = append(res, matrix[i][j])
      count++
      i--
    }
    // 进入到下一层
    top, left, bottom, right = top+1, left+1, bottom-1, right-1
  }
  return res
}
```

55. Jump Game

题目

Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Determine if you are able to reach the last index.

Example 1:

```
Input: [2,3,1,1,4]
Output: true
Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.
```

Example 2:

```
Input: [3,2,1,0,4]
Output: false
Explanation: You will always arrive at index 3 no matter what. Its maximum
             jump length is 0, which makes it impossible to reach the last index.
```

题目大意

给定一个非负整数数组，最初位于数组的第一个位置。数组中的每个元素代表在该位置可以跳跃的最大长度。判断是否能够到达最后一个位置。

解题思路

- 给出一个非负数组，要求判断从数组 0 下标开始，能否到达数组最后一个位置。
- 这一题比较简单。如果某一个作为 `起跳点` 的格子可以跳跃的距离是 `n`，那么表示后面 `n` 个格子都可以作为 `起跳点`。可以对每一个能作为 `起跳点` 的格子都尝试跳一次，把 `能跳到最远的距离maxJump` 不断更新。如果可以一直跳到最后，就成功了。如果中间有一个点比 `maxJump` 还要大，说明在这个点和 maxJump 中间连不上了，有些点不能到达最后一个位置。

代码

```go
func canJump(nums []int) bool {
  n := len(nums)
  if n == 0 {
    return false
  }
  if n == 1 {
    return true
  }
  maxJump := 0
  for i, v := range nums {
    if i > maxJump {
      return false
    }
    maxJump = max(maxJump, i+v)
  }
  return true
}
```

56. Merge Intervals

题目

Given a collection of intervals, merge all overlapping intervals.

Example 1:

```
Input: [[1,3],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]
Explanation: Since intervals [1,3] and [2,6] overlaps, merge them into [1,6].
```

Example 2:

```
Input: [[1,4],[4,5]]
Output: [[1,5]]
Explanation: Intervals [1,4] and [4,5] are considered overlapping.
```

题目大意

合并给的多个区间，区间有重叠的要进行区间合并。

解题思路

先按照区间起点进行排序。然后从区间起点小的开始扫描，依次合并每个有重叠的区间。

代码

```go
package leetcode

/**
 * Definition for an interval.
 * type Interval struct {
 *     Start int
 *     End   int
 * }
 */

// Interval define
type Interval struct {
  Start int
  End   int
}

func merge56(intervals []Interval) []Interval {
  if len(intervals) == 0 {
    return intervals
  }
  quickSort(intervals, 0, len(intervals)-1)
  res := make([]Interval, 0)
  res = append(res, intervals[0])
```

```go
  curIndex := 0
  for i := 1; i < len(intervals); i++ {
    if intervals[i].Start > res[curIndex].End {
      curIndex++
      res = append(res, intervals[i])
    } else {
      res[curIndex].End = max(intervals[i].End, res[curIndex].End)
    }
  }
  return res
}

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

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

func partitionSort(a []Interval, lo, hi int) int {
  pivot := a[hi]
  i := lo - 1
  for j := lo; j < hi; j++ {
    if (a[j].Start < pivot.Start) || (a[j].Start == pivot.Start && a[j].End <
pivot.End) {
      i++
      a[j], a[i] = a[i], a[j]
    }
  }
  a[i+1], a[hi] = a[hi], a[i+1]
  return i + 1
}
func quickSort(a []Interval, lo, hi int) {
  if lo >= hi {
    return
  }
  p := partitionSort(a, lo, hi)
  quickSort(a, lo, p-1)
  quickSort(a, p+1, hi)
}
```

57. Insert Interval

题目

Given a set of non-overlapping intervals, insert a new interval into the intervals (merge if necessary).

You may assume that the intervals were initially sorted according to their start times.

Example 1:

```
Input: intervals = [[1,3],[6,9]], newInterval = [2,5]
Output: [[1,5],[6,9]]
```

Example 2:

```
Input: intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]], newInterval = [4,8]
Output: [[1,2],[3,10],[12,16]]
Explanation: Because the new interval [4,8] overlaps with [3,5],[6,7],[8,10].
```

题目大意

这一题是第 56 题的加强版。给出多个没有重叠的区间，然后再给一个区间，要求把如果有重叠的区间进行合并。

解题思路

可以分 3 段处理，先添加原来的区间，即在给的 newInterval 之前的区间。然后添加 newInterval ，注意这里可能需要合并多个区间。最后把原来剩下的部分添加到最终结果中即可。

代码

```go
package leetcode

/**
 * Definition for an interval.
 * type Interval struct {
 *     Start int
 *     End   int
 * }
 */

func insert(intervals []Interval, newInterval Interval) []Interval {
  res := make([]Interval, 0)
  if len(intervals) == 0 {
```

```
    res = append(res, newInterval)
    return res
  }
  curIndex := 0
  for curIndex < len(intervals) && intervals[curIndex].End < newInterval.Start {
    res = append(res, intervals[curIndex])
    curIndex++
  }

  for curIndex < len(intervals) && intervals[curIndex].Start <= newInterval.End {
    newInterval = Interval{Start: min(newInterval.Start, intervals[curIndex].Start),
End: max(newInterval.End, intervals[curIndex].End)}
    curIndex++
  }
  res = append(res, newInterval)

  for curIndex < len(intervals) {
    res = append(res, intervals[curIndex])
    curIndex++
  }
  return res
}
```

59. Spiral Matrix II

题目

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 ]
]
```

题目大意

给定一个正整数 n，生成一个包含 1 到 n^2 所有元素，且元素按顺时针顺序螺旋排列的正方形矩阵。

解题思路

- 给出一个数组 n，要求输出一个 n * n 的二维数组，里面元素是 1 - n*n，且数组排列顺序是螺旋排列的
- 这一题是第 54 题的加强版，没有需要注意的特殊情况，直接模拟即可。

代码

```
package leetcode

func generateMatrix(n int) [][]int {
  if n == 0 {
    return [][]int{}
  }
  if n == 1 {
    return [][]int{[]int{1}}
  }
  res, visit, round, x, y, spDir := make([][]int, n), make([][]int, n), 0, 0, 0, []
[]int{
    []int{0, 1},  // 朝右
    []int{1, 0},  // 朝下
    []int{0, -1}, // 朝左
    []int{-1, 0}, // 朝上
  }
  for i := 0; i < n; i++ {
    visit[i] = make([]int, n)
    res[i] = make([]int, n)
  }
  visit[x][y] = 1
  res[x][y] = 1
  for i := 0; i < n*n; i++ {
    x += spDir[round%4][0]
    y += spDir[round%4][1]
    if (x == 0 && y == n-1) || (x == n-1 && y == n-1) || (y == 0 && x == n-1) {
      round++
    }
    if x > n-1 || y > n-1 || x < 0 || y < 0 {
      return res
    }
    if visit[x][y] == 0 {
      visit[x][y] = 1
      res[x][y] = i + 2
    }
    switch round % 4 {
    case 0:
      if y+1 <= n-1 && visit[x][y+1] == 1 {
        round++
        continue
      }
    case 1:
      if x+1 <= n-1 && visit[x+1][y] == 1 {
        round++
        continue
```

```
      }
    case 2:
      if y-1 >= 0 && visit[x][y-1] == 1 {
        round++
        continue
      }
    case 3:
      if x-1 >= 0 && visit[x-1][y] == 1 {
        round++
        continue
      }
    }
  }
  return res
}
```

60. Permutation Sequence

题目

The set `[1,2,3,...,*n*]` contains a total of *n*! unique permutations.

By listing and labeling all of the permutations in order, we get the following sequence for *n* = 3:

1. `"123"`
2. `"132"`
3. `"213"`
4. `"231"`
5. `"312"`
6. `"321"`

Given *n* and *k*, return the *k*th permutation sequence.

Note:

- Given *n* will be between 1 and 9 inclusive.
- Given *k* will be between 1 and *n*! inclusive.

Example 1:

```
Input: n = 3, k = 3
Output: "213"
```

Example 2:

```
Input: n = 4, k = 9
Output: "2314"
```

题目大意

给出集合 [1,2,3,...,n]，其所有元素共有 n! 种排列。

按大小顺序列出所有排列情况，并一一标记，当 n = 3 时, 所有排列如下："123"，"132"，"213"，"231"，"312"，"321"，给定 n 和 k，返回第 k 个排列。

解题思路

- 用 DFS 暴力枚举，这种做法时间复杂度特别高，想想更优的解法。

代码

```go
package leetcode

import (
  "fmt"
  "strconv"
)

func getPermutation(n int, k int) string {
  if k == 0 {
    return ""
  }
  used, p, res := make([]bool, n), []int{}, ""
  findPermutation(n, 0, &k, p, &res, &used)
  return res
}

func findPermutation(n, index int, k *int, p []int, res *string, used *[]bool) {
  fmt.Printf("n = %v index = %v k = %v p = %v res = %v user = %v\n", n, index, *k, p,
*res, *used)
  if index == n {
    *k--
    if *k == 0 {
      for _, v := range p {
        *res += strconv.Itoa(v + 1)
      }
    }
    return
  }
  for i := 0; i < n; i++ {
```

```
    if !(*used)[i] {
      (*used)[i] = true
      p = append(p, i)
      findPermutation(n, index+1, k, p, res, used)
      p = p[:len(p)-1]
      (*used)[i] = false
    }
  }
  return
}
```

61. Rotate List

题目

Given a linked list, rotate the list to the right by k places, where k is non-negative.

Example 1:

```
Input: 1->2->3->4->5->NULL, k = 2
Output: 4->5->1->2->3->NULL
Explanation:
rotate 1 steps to the right: 5->1->2->3->4->NULL
rotate 2 steps to the right: 4->5->1->2->3->NULL
```

Example 2:

```
Input: 0->1->2->NULL, k = 4
Output: 2->0->1->NULL
Explanation:
rotate 1 steps to the right: 2->0->1->NULL
rotate 2 steps to the right: 1->2->0->NULL
rotate 3 steps to the right: 0->1->2->NULL
rotate 4 steps to the right: 2->0->1->NULL
```

题目大意

旋转链表 K 次。

解题思路

这道题需要注意的点是，K 可能很大，K = 2000000000 ，如果是循环肯定会超时。应该找出 O(n) 的复杂度的算法才行。由于是循环旋转，最终状态其实是确定的，利用链表的长度取余可以得到链表的最终旋转结果。

这道题也不能用递归，递归解法会超时。

代码

```
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */
func rotateRight(head *ListNode, k int) *ListNode {
  if head == nil || head.Next == nil || k == 0 {
    return head
  }
  newHead := &ListNode{Val: 0, Next: head}
  len := 0
  cur := newHead
  for cur.Next != nil {
    len++
    cur = cur.Next
  }
  if (k % len) == 0 {
    return head
  }
  cur.Next = head
  cur = newHead
  for i := len - k%len; i > 0; i-- {
    cur = cur.Next
  }
  res := &ListNode{Val: 0, Next: cur.Next}
  cur.Next = nil
  return res.Next
}
```

62. Unique Paths

题目

A robot is located at the top-left corner of a *m* x *n* grid (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

How many possible unique paths are there?

Above is a 7 x 3 grid. How many possible unique paths are there?

Note: *m* and *n* will be at most 100.

Example 1:

```
Input: m = 3, n = 2
Output: 3
Explanation:
From the top-left corner, there are a total of 3 ways to reach the bottom-right corner:
1. Right -> Right -> Down
2. Right -> Down -> Right
3. Down -> Right -> Right
```

Example 2:

```
Input: m = 7, n = 3
Output: 28
```

题目大意

一个机器人位于一个 m x n 网格的左上角 （起始点在下图中标记为“Start” ）。机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为“Finish”）。问总共有多少条不同的路径?

解题思路

- 这是一道简单的 DP 题。输出地图上从左上角走到右下角的走法数。
- 由于机器人只能向右走和向下走，所以地图的第一行和第一列的走法数都是 1，地图中任意一点的走法数是 `dp[i][j] = dp[i-1][j] + dp[i][j-1]`

代码

```go
package leetcode

func uniquePaths(m int, n int) int {
  dp := make([][]int, n)
  for i := 0; i < n; i++ {
```

```
    dp[i] = make([]int, m)
  }
  for i := 0; i < n; i++ {
    for j := 0; j < m; j++ {
      if i == 0 || j == 0 {
        dp[i][j] = 1
        continue
      }
      dp[i][j] = dp[i-1][j] + dp[i][j-1]
    }
  }
  return dp[n-1][m-1]
}
```

63. Unique Paths II

题目

A robot is located at the top-left corner of a *m* x *n* grid (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

Now consider if some obstacles are added to the grids. How many unique paths would there be?

An obstacle and empty space is marked as `1` and `0` respectively in the grid.

Note: *m* and *n* will be at most 100.

Example 1:

```
Input:
[
  [0,0,0],
  [0,1,0],
  [0,0,0]
]
Output: 2
Explanation:
There is one obstacle in the middle of the 3x3 grid above.
There are two ways to reach the bottom-right corner:
1. Right -> Right -> Down -> Down
2. Down -> Down -> Right -> Right
```

题目大意

一个机器人位于一个 m x n 网格的左上角 （起始点在下图中标记为“Start” ）。机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为“Finish”）。现在考虑网格中有障碍物。那么从左上角到右下角将会有多少条不同的路径?

解题思路

- 这一题是第 62 题的加强版。也是一道考察 DP 的简单题。
- 这一题比第 62 题增加的条件是地图中会出现障碍物，障碍物的处理方法是 `dp[i][j]=0`。
- 需要注意的一种情况是，起点就是障碍物，那么这种情况直接输出 0 。

代码

```go
package leetcode

func uniquePathsWithObstacles(obstacleGrid [][]int) int {
  if len(obstacleGrid) == 0 || obstacleGrid[0][0] == 1 {
    return 0
  }
  m, n := len(obstacleGrid), len(obstacleGrid[0])
  dp := make([][]int, m)
  for i := 0; i < m; i++ {
    dp[i] = make([]int, n)
  }
  dp[0][0] = 1
  for i := 1; i < n; i++ {
    if dp[0][i-1] != 0 && obstacleGrid[0][i] != 1 {
      dp[0][i] = 1
    }
  }
  for i := 1; i < m; i++ {
    if dp[i-1][0] != 0 && obstacleGrid[i][0] != 1 {
      dp[i][0] = 1
```

```
    }
  }
  for i := 1; i < m; i++ {
    for j := 1; j < n; j++ {
      if obstacleGrid[i][j] != 1 {
        dp[i][j] = dp[i-1][j] + dp[i][j-1]
      }
    }
  }
  return dp[m-1][n-1]
}
```

64. Minimum Path Sum

题目

Given a *m* x *n* grid filled with non-negative numbers, find a path from top left to bottom right which *minimizes* the sum of all numbers along its path.

Note: You can only move either down or right at any point in time.

Example:

```
Input:
[
  [1,3,1],
  [1,5,1],
  [4,2,1]
]
Output: 7
Explanation: Because the path 1→3→1→1→1 minimizes the sum.
```

题目大意

给定一个包含非负整数的 m x n 网格，请找出一条从左上角到右下角的路径，使得路径上的数字总和为最小。说明：每次只能向下或者向右移动一步。

解题思路

- 在地图上求出从左上角到右下角的路径中，数字之和最小的一个，输出数字和。
- 这一题最简单的想法就是用一个二维数组来 DP，当然这是最原始的做法。由于只能往下和往右走，只需要维护 2 列信息就可以了，从左边推到最右边即可得到最小的解。更近一步，可以直接在原来的数组中做原地 DP，空间复杂度为 0 。

代码

```go
package leetcode

// 解法一 原地 DP，无辅助空间
func minPathSum(grid [][]int) int {
  m, n := len(grid), len(grid[0])
  for i := 1; i < m; i++ {
    grid[i][0] += grid[i-1][0]
  }
  for j := 1; j < n; j++ {
    grid[0][j] += grid[0][j-1]
  }
  for i := 1; i < m; i++ {
    for j := 1; j < n; j++ {
      grid[i][j] += min(grid[i-1][j], grid[i][j-1])
    }
  }
  return grid[m-1][n-1]

}

// 解法二 最原始的方法，辅助空间 O(n^2)
func minPathSum1(grid [][]int) int {
  if len(grid) == 0 {
    return 0
  }
  m, n := len(grid), len(grid[0])
  if m == 0 || n == 0 {
    return 0
  }

  dp := make([][]int, m)
  for i := 0; i < m; i++ {
    dp[i] = make([]int, n)
  }
  // initFirstCol
  for i := 0; i < len(dp); i++ {
    if i == 0 {
      dp[i][0] = grid[i][0]
    } else {
      dp[i][0] = grid[i][0] + dp[i-1][0]
    }
  }
  // initFirstRow
  for i := 0; i < len(dp[0]); i++ {
    if i == 0 {
      dp[0][i] = grid[0][i]
    } else {
      dp[0][i] = grid[0][i] + dp[0][i-1]
    }
```

```
  }
  for i := 1; i < m; i++ {
    for j := 1; j < n; j++ {
      dp[i][j] = min(dp[i-1][j], dp[i][j-1]) + grid[i][j]
    }
  }
  return dp[m-1][n-1]
}
```

66. Plus One

题目

Given a **non-empty** array of digits representing a non-negative integer, plus one to the integer.

The digits are stored such that the most significant digit is at the head of the list, and each element in the array contain a single digit.

You may assume the integer does not contain any leading zero, except the number 0 itself.

Example 1:

```
Input: [1,2,3]
Output: [1,2,4]
Explanation: The array represents the integer 123.
```

Example 2:

```
Input: [4,3,2,1]
Output: [4,3,2,2]
Explanation: The array represents the integer 4321.
```

题目大意

给定一个由整数组成的非空数组所表示的非负整数，在该数的基础上加一。最高位数字存放在数组的首位， 数组中每个元素只存储单个数字。你可以假设除了整数 0 之外，这个整数不会以零开头。

解题思路

- 给出一个数组，代表一个十进制数，数组的 0 下标是十进制数的高位。要求计算这个十进制数加一以后的结果。
- 简单的模拟题。从数组尾部开始往前扫，逐位进位即可。最高位如果还有进位需要在数组里面第 0 位再插入一个 1 。

代码

```go
package leetcode

func plusOne(digits []int) []int {
  for i := len(digits) - 1; i >= 0; i-- {
    digits[i]++
    if digits[i] != 10 {
      // no carry
      return digits
    }
    // carry
    digits[i] = 0
  }
  // all carry
  digits[0] = 1
  digits = append(digits, 0)
  return digits
}
```

67. Add Binary

题目

Given two binary strings, return their sum (also a binary string).

The input strings are both **non-empty** and contains only characters `1` or `0`.

Example 1:

```
Input: a = "11", b = "1"
Output: "100"
```

Example 2:

```
Input: a = "1010", b = "1011"
Output: "10101"
```

题目大意

给你两个二进制字符串，返回它们的和（用二进制表示）。输入为 非空 字符串且只包含数字 1 和 0。

解题思路

- 要求输出 2 个二进制数的和，结果也用二进制表示。

- 简单题。按照二进制的加法规则做加法即可。

代码

```
package leetcode

import (
  "strconv"
  "strings"
)

func addBinary(a string, b string) string {
  if len(b) > len(a) {
    a, b = b, a
  }

  res := make([]string, len(a)+1)
  i, j, k, c := len(a)-1, len(b)-1, len(a), 0
  for i >= 0 && j >= 0 {
    ai, _ := strconv.Atoi(string(a[i]))
    bj, _ := strconv.Atoi(string(b[j]))
    res[k] = strconv.Itoa((ai + bj + c) % 2)
    c = (ai + bj + c) / 2
    i--
    j--
    k--
  }

  for i >= 0 {
    ai, _ := strconv.Atoi(string(a[i]))
    res[k] = strconv.Itoa((ai + c) % 2)
    c = (ai + c) / 2
    i--
    k--
  }

  if c > 0 {
    res[k] = strconv.Itoa(c)
  }

  return strings.Join(res, "")
}
```

69. Sqrt(x)

题目

Implement `int sqrt(int x)`.

Compute and return the square root of *x*, where *x* is guaranteed to be a non-negative integer.

Since the return type is an integer, the decimal digits are truncated and only the integer part of the result is returned.

Example 1:

```
Input: 4
Output: 2
```

Example 2:

```
Input: 8
Output: 2
Explanation: The square root of 8 is 2.82842..., and since
             the decimal part is truncated, 2 is returned.
```

题目大意

实现 int sqrt(int x) 函数。计算并返回 x 的平方根，其中 x 是非负整数。由于返回类型是整数，结果只保留整数的部分，小数部分将被舍去。

解题思路

- 题目要求求出根号 x
- 根据题意，根号 x 的取值范围一定在 `[0,x]` 之间，这个区间内的值是递增有序的，有边界的，可以用下标访问的，满足这三点正好也就满足了二分搜索的 3 大条件。所以解题思路一，二分搜索。
- 解题思路二，牛顿迭代法。求根号 x，即求满足 `x^2 - n = 0` 方程的所有解。

如图，一个曲线方程 f(x)，在它的 f(x_n)处画一条切线与 x 轴相交，交点为x_{n+1}，如果继续在它的 f(x_{n+1})处画一条切线与 x 轴相交，会得到交点x_{n+2}。而在这个过程中，可以发现交点x_{n+m}会无限逼近方程 f(x)=0 的解，最终可以得到一个与理想值无限靠近的解。

http://blog.csdn.net/chenrenxiang

而这里讨论的是求平方根，所以曲线方程更简单。比如，我们要求 N 的平方根。

那么其实就是求方程 $f(x) = x^2 - N$，当 $f(x) = 0$ 时方程的解。

函数 f(x)的导函数是：$f'(x) = 2x$

那么 f(x)函数的曲线在（x_n, $x_n^2 - N$）点处切线的斜率为：$2x_n$

所以切线方程为：$f(x) - (x_n^2 - N) = 2x_n(x - x_n)$，即：$f(x) = 2x_n x - x_n^2 - N$

那么切线方程与 x 轴的交点 $x_{n+1} = (x_n + N/x_n)/2$

我们可以将得到的交点值的平方与 N 比较，循环以上过程直到得到满意的值。

代码

```
package leetcode

// 解法一 二分，找到最后一个满足 n^2 <= x 的整数n
func mySqrt(x int) int {
  l, r := 0, x
  for l < r {
    mid := (l + r + 1) / 2
```

```
    if mid*mid > x {
      r = mid - 1
    } else {
      l = mid
    }
  }
  return l
}

// 解法二 牛顿迭代法 https://en.wikipedia.org/wiki/Integer_square_root
func mySqrt1(x int) int {
  r := x
  for r*r > x {
    r = (r + x/r) / 2
  }
  return r
}

// 解法三 Quake III 游戏引擎中有一种比 STL 的 sqrt 快 4 倍的实现
https://en.wikipedia.org/wiki/Fast_inverse_square_root
// float Q_rsqrt( float number )
// {
//  long i;
//  float x2, y;
//  const float threehalfs = 1.5F;

//  x2 = number * 0.5F;
//  y  = number;
//  i  = * ( long * ) &y;                       // evil floating point bit level
hacking
//  i  = 0x5f3759df - ( i >> 1 );               // what the fuck?
//  y  = * ( float * ) &i;
//  y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
// // y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed
//  return y;
// }
```

70. Climbing Stairs

题目

You are climbing a stair case. It takes *n* steps to reach to the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Note: Given *n* will be a positive integer.

Example 1:

```
Input: 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps
```

Example 2:

```
Input: 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
```

题目大意

假设你正在爬楼梯。需要 n 阶你才能到达楼顶。每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢？注意：给定 n 是一个正整数

解题思路

- 简单的 DP，经典的爬楼梯问题。一个楼梯可以由 `n-1` 和 `n-2` 的楼梯爬上来。
- 这一题求解的值就是斐波那契数列。

代码

```go
package leetcode

func climbStairs(n int) int {
  dp := make([]int, n+1)
  dp[0], dp[1] = 1, 1
  for i := 2; i <= n; i++ {
    dp[i] = dp[i-1] + dp[i-2]
  }
  return dp[n]
}
```

71. Simplify Path

题目

Given an absolute path for a file (Unix-style), simplify it. Or in other words, convert it to the canonical path.

In a UNIX-style file system, a period . refers to the current directory. Furthermore, a double period .. moves the directory up a level. For more information, see: Absolute path vs relative path in Linux/Unix

Note that the returned canonical path must always begin with a slash /, and there must be only a single slash / between two directory names. The last directory name (if it exists) must not end with a trailing /. Also, the canonical path must be the shortest string representing the absolute path.

Example 1:

```
Input: "/home/"
Output: "/home"
Explanation: Note that there is no trailing slash after the last directory name.
```

Example 2:

```
Input: "/../"
Output: "/"
Explanation: Going one level up from the root directory is a no-op, as the root level
is the highest level you can go.
```

Example 3:

```
Input: "/home//foo/"
Output: "/home/foo"
Explanation: In the canonical path, multiple consecutive slashes are replaced by a
single one.
```

Example 4:

```
Input: "/a/./b/../../c/"
Output: "/c"
```

Example 5:

```
Input: "/a/../../b/../c//.//"
Output: "/c"
```

Example 6:

```
Input: "/a//b////c/d//././/.."
Output: "/a/b/c"
```

题目大意

给出一个 Unix 的文件路径，要求简化这个路径。这道题也是考察栈的题目。

解题思路

这道题笔者提交了好多次才通过，并不是题目难，而是边界条件很多，没考虑全一种情况就会出错。有哪些边界情况就看笔者的 test 文件吧。

代码

```go
package leetcode

import (
  "path/filepath"
  "strings"
)

// 解法一
func simplifyPath(path string) string {
  arr := strings.Split(path, "/")
  stack := make([]string, 0)
  var res string
  for i := 0; i < len(arr); i++ {
    cur := arr[i]
    //cur := strings.TrimSpace(arr[i]) 更加严谨的做法应该还要去掉末尾的空格
    if cur == ".." {
      if len(stack) > 0 {
        stack = stack[:len(stack)-1]
      }
    } else if cur != "." && len(cur) > 0 {
      stack = append(stack, arr[i])
    }
  }
  if len(stack) == 0 {
    return "/"
  }
  res = strings.Join(stack, "/")
  return "/" + res
}
```

```go
// 解法二 golang 的官方库 API
func simplifyPath1(path string) string {
  return filepath.Clean(path)
}
```

73. Set Matrix Zeroes

题目

Given an *`m* x *n*` matrix. If an element is **0**, set its entire row and column to **0**. Do it in-place.

Follow up:

- A straight forward solution using O(*mn*) space is probably a bad idea.
- A simple improvement uses O(*m* + *n*) space, but still not the best solution.
- Could you devise a constant space solution?

Example 1:

```
Input: matrix = [[1,1,1],[1,0,1],[1,1,1]]
Output: [[1,0,1],[0,0,0],[1,0,1]]
```

Example 2:

```
Input: matrix = [[0,1,2,0],[3,4,5,2],[1,3,1,5]]
Output: [[0,0,0,0],[0,4,5,0],[0,3,1,0]]
```

Constraints:

- `m == matrix.length`
- `n == matrix[0].length`
- `1 <= m, n <= 200`
- `2^31 <= matrix[i][j] <= 2^31 - 1`

题目大意

给定一个 `m x n` 的矩阵，如果一个元素为 0，则将其所在行和列的所有元素都设为 0。请使用原地算法。

解题思路

- 此题考查对程序的控制能力，无算法思想。题目要求采用原地的算法，所有修改即在原二维数组上进行。在二维数组中有 2 个特殊位置，一个是第一行，一个是第一列。它们的特殊性在于，它们之间只要有一个 0，它们都会变为全 0 。先用 2 个变量记录这一行和这一列中是否有 0，防止之后的修改覆盖了这 2 个地方。然后除去这一行和这一列以外的部分判断是否有 0，如果有 0，将它们所在的行第一个元素标记为 0，所在列的第一个元素标记为 0 。最后通过标记，将对应的行列置 0 即可。

代码

```go
package leetcode

func setZeroes(matrix [][]int) {
  if len(matrix) == 0 || len(matrix[0]) == 0 {
    return
  }
  isFirstRowExistZero, isFirstColExistZero := false, false
  for i := 0; i < len(matrix); i++ {
    if matrix[i][0] == 0 {
      isFirstColExistZero = true
      break
    }
  }
  for j := 0; j < len(matrix[0]); j++ {
    if matrix[0][j] == 0 {
      isFirstRowExistZero = true
      break
    }
  }
  for i := 1; i < len(matrix); i++ {
    for j := 1; j < len(matrix[0]); j++ {
      if matrix[i][j] == 0 {
        matrix[i][0] = 0
        matrix[0][j] = 0
```

```
        }
      }
    }
    // 处理[1:]行全部置 0
    for i := 1; i < len(matrix); i++ {
      if matrix[i][0] == 0 {
        for j := 1; j < len(matrix[0]); j++ {
          matrix[i][j] = 0
        }
      }
    }
    // 处理[1:]列全部置 0
    for j := 1; j < len(matrix[0]); j++ {
      if matrix[0][j] == 0 {
        for i := 1; i < len(matrix); i++ {
          matrix[i][j] = 0
        }
      }
    }
    if isFirstRowExistZero {
      for j := 0; j < len(matrix[0]); j++ {
        matrix[0][j] = 0
      }
    }
    if isFirstColExistZero {
      for i := 0; i < len(matrix); i++ {
        matrix[i][0] = 0
      }
    }
  }
```

74. Search a 2D Matrix

题目

Write an efficient algorithm that searches for a value in an *m* x *n* matrix. This matrix has the following properties:

- Integers in each row are sorted from left to right.
- The first integer of each row is greater than the last integer of the previous row.

Example 1:

```
Input:
matrix = [
  [1,   3,  5,  7],
  [10, 11, 16, 20],
  [23, 30, 34, 50]
]
target = 3
Output: true
```

Example 2:

```
Input:
matrix = [
  [1,   3,  5,  7],
  [10, 11, 16, 20],
  [23, 30, 34, 50]
]
target = 13
Output: false
```

题目大意

编写一个高效的算法来判断 m x n 矩阵中，是否存在一个目标值。该矩阵具有如下特性：

- 每行中的整数从左到右按升序排列。
- 每行的第一个整数大于前一行的最后一个整数。

解题思路

- 给出一个二维矩阵，矩阵的特点是随着矩阵的下标增大而增大。要求设计一个算法能在这个矩阵中高效的找到一个数，如果找到就输出 true，找不到就输出 false。
- 虽然是一个二维矩阵，但是由于它特殊的有序性，所以完全可以按照下标把它看成一个一维矩阵，只不过需要行列坐标转换。最后利用二分搜索直接搜索即可。

代码

```go
package leetcode

func searchMatrix(matrix [][]int, target int) bool {
  if len(matrix) == 0 {
    return false
  }
  m, low, high := len(matrix[0]), 0, len(matrix[0])*len(matrix)-1
  for low <= high {
    mid := low + (high-low)>>1
    if matrix[mid/m][mid%m] == target {
      return true
```

```
    } else if matrix[mid/m][mid%m] > target {
      high = mid - 1
    } else {
      low = mid + 1
    }
  }
  return false
}
```

75. Sort Colors

题目

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?

题目大意

抽象题意其实就是排序。这题可以用快排一次通过。

解题思路

题目末尾的 Follow up 提出了一个更高的要求，能否用一次循环解决问题？这题由于数字只会出现 0，1，2 这三个数字，所以用游标移动来控制顺序也是可以的。具体做法：0 是排在最前面的，所以只要添加一个 0，就需要放置 1 和 2。1 排在 2 前面，所以添加 1 的时候也需要放置 2 。至于最后的 2，只用移动游标即可。

这道题可以用计数排序，适合待排序数字很少的题目。用一个 3 个容量的数组分别计数，记录 0，1，2 出现的个数。然后再根据个数排列 0，1，2 即可。时间复杂度 O(n)，空间复杂度 O(K)。这一题 K = 3。

这道题也可以用一次三路快排。数组分为 3 部分，第一个部分都是 0，中间部分都是 1，最后部分都是 2 。

代码

```
package leetcode

func sortColors(nums []int) {
  zero, one := 0, 0
  for i, n := range nums {
    nums[i] = 2
    if n <= 1 {
      nums[one] = 1
      one++
    }
    if n == 0 {
      nums[zero] = 0
      zero++
    }
  }
}
```

76. Minimum Window Substring

题目

Given a string S and a string T, find the minimum window in S which will contain all the characters in T in complexity O(n).

Example:

```
Input: S = "ADOBECODEBANC", T = "ABC"
Output: "BANC"
```

Note:

- If there is no such window in S that covers all characters in T, return the empty string "".
- If there is such window, you are guaranteed that there will always be only one unique minimum window in S.

题目大意

给定一个源字符串 s，再给一个字符串 T，要求在源字符串中找到一个窗口，这个窗口包含由字符串各种排列组合组成的，窗口中可以包含 T 中没有的字符，如果存在多个，在结果中输出最小的窗口，如果找不到这样的窗口，输出空字符串。

解题思路

这一题是滑动窗口的题目，在窗口滑动的过程中不断的包含字符串 T，直到完全包含字符串 T 的字符以后，记下左右窗口的位置和窗口大小。每次都不断更新这个符合条件的窗口和窗口大小的最小值。最后输出结果即可。

代码

```
package leetcode

func minWindow(s string, t string) string {
  if s == "" || t == "" {
    return ""
  }
  var tFreq, sFreq [256]int
  result, left, right, finalLeft, finalRight, minW, count := "", 0, -1, -1, -1,
len(s)+1, 0

  for i := 0; i < len(t); i++ {
    tFreq[t[i]-'a']++
  }

  for left < len(s) {
    if right+1 < len(s) && count < len(t) {
      sFreq[s[right+1]-'a']++
      if sFreq[s[right+1]-'a'] <= tFreq[s[right+1]-'a'] {
        count++
      }
      right++
    } else {
      if right-left+1 < minW && count == len(t) {
        minW = right - left + 1
        finalLeft = left
        finalRight = right
      }
      if sFreq[s[left]-'a'] == tFreq[s[left]-'a'] {
        count--
      }
      sFreq[s[left]-'a']--
      left++
    }
  }
  if finalLeft != -1 {
    result = string(s[finalLeft : finalRight+1])
  }
  return result
}
```

77. Combinations

题目

Given two integers *n* and *k*, return all possible combinations of *k* numbers out of 1 ... *n*.

Example:

```
Input: n = 4, k = 2
Output:
[
  [2,4],
  [3,4],
  [2,3],
  [1,2],
  [1,3],
  [1,4],
]
```

题目大意

给定两个整数 n 和 k，返回 1 ... n 中所有可能的 k 个数的组合。

解题思路

- 计算排列组合中的组合，用 DFS 深搜即可，注意剪枝

代码

```go
package leetcode

func combine(n int, k int) [][]int {
  if n <= 0 || k <= 0 || k > n {
    return [][]int{}
  }
  c, res := []int{}, [][]int{}
  generateCombinations(n, k, 1, c, &res)
  return res
}

func generateCombinations(n, k, start int, c []int, res *[][]int) {
  if len(c) == k {
    b := make([]int, len(c))
    copy(b, c)
    *res = append(*res, b)
    return
```

```
  }
  // i will at most be n - (k - c.size()) + 1
  for i := start; i <= n-(k-len(c))+1; i++ {
    c = append(c, i)
    generateCombinations(n, k, i+1, c, res)
    c = c[:len(c)-1]
  }
  return
}
```

78. Subsets

题目

Given a set of **distinct** integers, *nums*, return all possible subsets (the power set).

Note: The solution set must not contain duplicate subsets.

Example:

```
Input: nums = [1,2,3]
Output:
[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]
```

题目大意

给定一组不含重复元素的整数数组 nums，返回该数组所有可能的子集（幂集）。说明：解集不能包含重复的子集。

解题思路

- 找出一个集合中的所有子集，空集也算是子集。且数组中的数字不会出现重复。用 DFS 暴力枚举即可。
- 这一题和第 90 题，第 491 题类似，可以一起解答和复习。

代码

```
package leetcode
```

```go
import "sort"

// 解法一
func subsets(nums []int) [][]int {
  c, res := []int{}, [][]int{}
  for k := 0; k <= len(nums); k++ {
    generateSubsets(nums, k, 0, c, &res)
  }
  return res
}

func generateSubsets(nums []int, k, start int, c []int, res *[][]int) {
  if len(c) == k {
    b := make([]int, len(c))
    copy(b, c)
    *res = append(*res, b)
    return
  }
  // i will at most be n - (k - c.size()) + 1
  for i := start; i < len(nums)-(k-len(c))+1; i++ {
    c = append(c, nums[i])
    generateSubsets(nums, k, i+1, c, res)
    c = c[:len(c)-1]
  }
  return
}

// 解法二
func subsets1(nums []int) [][]int {
  res := make([][]int, 1)
  sort.Ints(nums)
  for i := range nums {
    for _, org := range res {
      clone := make([]int, len(org), len(org)+1)
      copy(clone, org)
      clone = append(clone, nums[i])
      res = append(res, clone)
    }
  }
  return res
}

// 解法三：位运算的方法
func subsets2(nums []int) [][]int {
  if len(nums) == 0 {
    return nil
  }
  res := [][]int{}
```

```
  sum := 1 << uint(len(nums))
  for i := 0; i < sum; i++ {
    stack := []int{}
    tmp := i // i 从 000...000 到 111...111
    for j := len(nums) - 1; j >= 0; j-- { // 遍历 i 的每一位
      if tmp & 1 == 1 {
        stack = append([]int{nums[j]}, stack...)
      }
      tmp >>= 1
    }
    res = append(res, stack)
  }
  return res
}
```

79. Word Search

题目

Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once.

Example:

```
board =
[
  ['A','B','C','E'],
  ['S','F','C','S'],
  ['A','D','E','E']
]

Given word = "ABCCED", return true.
Given word = "SEE", return true.
Given word = "ABCB", return false.
```

题目大意

给定一个二维网格和一个单词，找出该单词是否存在于网格中。单词必须按照字母顺序，通过相邻的单元格内的字母构成，其中“相邻”单元格是那些水平相邻或垂直相邻的单元格。同一个单元格内的字母不允许被重复使用。

解题思路

- 在地图上的任意一个起点开始，向 4 个方向分别 DFS 搜索，直到所有的单词字母都找到了就输出 true，否则输出 false。

代码

```
package leetcode

var dir = [][]int{
  []int{-1, 0},
  []int{0, 1},
  []int{1, 0},
  []int{0, -1},
}

func exist(board [][]byte, word string) bool {
  visited := make([][]bool, len(board))
  for i := 0; i < len(visited); i++ {
    visited[i] = make([]bool, len(board[0]))
  }
  for i, v := range board {
    for j := range v {
      if searchWord(board, visited, word, 0, i, j) {
        return true
      }
    }
  }
  return false
}

func isInBoard(board [][]byte, x, y int) bool {
  return x >= 0 && x < len(board) && y >= 0 && y < len(board[0])
}

func searchWord(board [][]byte, visited [][]bool, word string, index, x, y int) bool {
  if index == len(word)-1 {
    return board[x][y] == word[index]
  }
  if board[x][y] == word[index] {
    visited[x][y] = true
    for i := 0; i < 4; i++ {
      nx := x + dir[i][0]
      ny := y + dir[i][1]
      if isInBoard(board, nx, ny) && !visited[nx][ny] && searchWord(board, visited,
word, index+1, nx, ny) {
        return true
      }
    }
    visited[x][y] = false
  }
  return false
```

```
}
```

80. Remove Duplicates from Sorted Array II

题目

Given a sorted array nums, remove the duplicates in-place such that duplicates appeared at most twice 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.

Example 1:

```
Given nums = [1,1,1,2,2,3],

Your function should return length = 5, with the first five elements of nums being 1,
1, 2, 2 and 3 respectively.

It doesn't matter what you leave beyond the returned length.
```

Example 2:

```
Given nums = [0,0,1,1,1,1,2,3,3],

Your function should return length = 7, with the first seven elements of nums being
modified to 0, 0, 1, 1, 2, 3 and 3 respectively.

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]);
}
```

题目大意

给定一个有序数组 nums，对数组中的元素进行去重，使得原数组中的每个元素最多暴露 2 个。最后返回去重以后数组的长度值。

解题思路

- 问题提示有序数组，一般最容易想到使用双指针的解法，双指针的关键点：移动两个指针的条件。
- 在该题中移动的条件：快指针从头遍历数组，慢指针指向修改后的数组的末端，当慢指针指向倒数第二个数与快指针指向的数不相等时，才移动慢指针，同时赋值慢指针。
- 处理边界条件：当数组小于两个元素时，不做处理。

代码

```go
package leetcode

func removeDuplicates(nums []int) int {
  slow := 0
  for fast, v := range nums {
    if fast < 2 || nums[slow-2] != v {
      nums[slow] = v
      slow++
    }
  }
  return slow
}
```

81. Search in Rotated Sorted Array II

题目

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., `[0,0,1,2,2,5,6]` might become `[2,5,6,0,0,1,2]`).

You are given a target value to search. If found in the array return `true`, otherwise return `false`.

Example 1:

```
Input: nums = [2,5,6,0,0,1,2], target = 0
Output: true
```

Example 2:

```
Input: nums = [2,5,6,0,0,1,2], target = 3
Output: false
```

Follow up:

- This is a follow up problem to [Search in Rotated Sorted Array](), where `nums` may contain duplicates.
- Would this affect the run-time complexity? How and why?

题目大意

假设按照升序排序的数组在预先未知的某个点上进行了旋转。(例如，数组 [0,0,1,2,2,5,6] 可能变为 [2,5,6,0,0,1,2])。

编写一个函数来判断给定的目标值是否存在于数组中。若存在返回 true，否则返回 false。

进阶:

- 这是搜索旋转排序数组 的延伸题目，本题中的 nums 可能包含重复元素。
- 这会影响到程序的时间复杂度吗？会有怎样的影响，为什么？

解题思路

- 给出一个数组，数组中本来是从小到大排列的，并且数组中有重复数字。但是现在把后面随机一段有序的放到数组前面，这样形成了前后两端有序的子序列。在这样的一个数组里面查找一个数，设计一个 O(log n) 的算法。如果找到就输出 `true`，如果没有找到，就输出 `false` 。
- 这一题是第 33 题的加强版，实现代码完全一样，只不过输出变了。这一题输出 `true` 和 `false` 了。具体思路见第 33 题。

代码

```go
package leetcode

func search(nums []int, target int) bool {
  if len(nums) == 0 {
    return false
  }
  low, high := 0, len(nums)-1
```

```
  for low <= high {
    mid := low + (high-low)>>1
    if nums[mid] == target {
      return true
    } else if nums[mid] > nums[low] { // 在数值大的一部分区间里
      if nums[low] <= target && target < nums[mid] {
        high = mid - 1
      } else {
        low = mid + 1
      }
    } else if nums[mid] < nums[high] { // 在数值小的一部分区间里
      if nums[mid] < target && target <= nums[high] {
        low = mid + 1
      } else {
        high = mid - 1
      }
    } else {
      if nums[low] == nums[mid] {
        low++
      }
      if nums[high] == nums[mid] {
        high--
      }
    }
  }
  return false
}
```

82. Remove Duplicates from Sorted List II

题目

Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list.

Example 1:

```
Input: 1->2->3->3->4->4->5
Output: 1->2->5
```

Example 2:

```
Input: 1->1->1->2->3
Output: 2->3
```

题目大意

删除链表中重复的结点，只要是有重复过的结点，全部删除。

解题思路

按照题意做即可。

代码

```go
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */

// 解法一
func deleteDuplicates1(head *ListNode) *ListNode {
  if head == nil {
    return nil
  }
  if head.Next == nil {
    return head
  }
  newHead := &ListNode{Next: head, Val: -999999}
  cur := newHead
  last := newHead
  front := head
  for front.Next != nil {
    if front.Val == cur.Val {
      // fmt.Printf("相同节点front = %v | cur = %v | last = %v\n", front.Val, cur.Val,
last.Val)
      front = front.Next
      continue
    } else {
      if cur.Next != front {
        // fmt.Printf("删除重复节点front = %v | cur = %v | last = %v\n", front.Val,
cur.Val, last.Val)
```

```go
        last.Next = front
        if front.Next != nil && front.Next.Val != front.Val {
          last = front
        }
        cur = front
        front = front.Next
      } else {
        // fmt.Printf("常规循环前front = %v | cur = %v | last = %v\n", front.Val,
cur.Val, last.Val)
        last = cur
        cur = cur.Next
        front = front.Next
        // fmt.Printf("常规循环后front = %v | cur = %v | last = %v\n", front.Val,
cur.Val, last.Val)

      }
    }
  }
  if front.Val == cur.Val {
    // fmt.Printf("相同节点front = %v | cur = %v | last = %v\n", front.Val, cur.Val,
last.Val)
    last.Next = nil
  } else {
    if cur.Next != front {
      last.Next = front
    }
  }
  return newHead.Next
}

// 解法二
func deleteDuplicates2(head *ListNode) *ListNode {
  if head == nil {
    return nil
  }
  if head.Next != nil && head.Val == head.Next.Val {
    for head.Next != nil && head.Val == head.Next.Val {
      head = head.Next
    }
    return deleteDuplicates(head.Next)
  }
  head.Next = deleteDuplicates(head.Next)
  return head
}

func deleteDuplicates(head *ListNode) *ListNode {
  cur := head
  if head == nil {
    return nil
```

```go
  }
  if head.Next == nil {
    return head
  }
  for cur.Next != nil {
    if cur.Next.Val == cur.Val {
      cur.Next = cur.Next.Next
    } else {
      cur = cur.Next
    }
  }
  return head
}

// 解法三 双循环简单解法 O(n*m)
func deleteDuplicates3(head *ListNode) *ListNode {
  if head == nil {
    return head
  }

  nilNode := &ListNode{Val: 0, Next: head}
  head = nilNode

  lastVal := 0
  for head.Next != nil && head.Next.Next != nil {
    if head.Next.Val == head.Next.Next.Val {
      lastVal = head.Next.Val
      for head.Next != nil && lastVal == head.Next.Val {
        head.Next = head.Next.Next
      }
    } else {
      head = head.Next
    }
  }
  return nilNode.Next
}

// 解法四 双指针+删除标志位，单循环解法 O(n)
func deleteDuplicates4(head *ListNode) *ListNode {
  if head == nil || head.Next == nil {
    return head
  }

  nilNode := &ListNode{Val: 0, Next: head}
  // 上次遍历有删除操作的标志位
  lastIsDel := false
  // 虚拟空结点
  head = nilNode
  // 前后指针用于判断
```

```go
  pre, back := head.Next, head.Next.Next
  // 每次只删除前面的一个重复的元素，留一个用于下次遍历判重
  // pre, back 指针的更新位置和值比较重要和巧妙
  for head.Next != nil && head.Next.Next != nil {
    if pre.Val != back.Val && lastIsDel {
      head.Next = head.Next.Next
      pre, back = head.Next, head.Next.Next
      lastIsDel = false
      continue
    }

    if pre.Val == back.Val {
      head.Next = head.Next.Next
      pre, back = head.Next, head.Next.Next
      lastIsDel = true
    } else {
      head = head.Next
      pre, back = head.Next, head.Next.Next
      lastIsDel = false
    }
  }
  // 处理 [1,1] 这种删除还剩一个的情况
  if lastIsDel && head.Next != nil {
    head.Next = nil
  }
  return nilNode.Next
}
```

83. Remove Duplicates from Sorted List

题目

Given a sorted linked list, delete all duplicates such that each element appear only once.

Example 1:

```
Input: 1->1->2
Output: 1->2
```

Example 2:

```
Input: 1->1->2->3->3
Output: 1->2->3
```

题目大意

删除链表中重复的结点，以保障每个结点只出现一次。

解题思路

按照题意做即可。

代码

```go
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */

func deleteDuplicates(head *ListNode) *ListNode {
  cur := head
  if head == nil {
    return nil
  }
  if head.Next == nil {
    return head
  }
  for cur.Next != nil {
    if cur.Next.Val == cur.Val {
      cur.Next = cur.Next.Next
    } else {
      cur = cur.Next
    }
  }
  return head
}
```

84. Largest Rectangle in Histogram

题目

Given n non-negative integers representing the histogram's bar height where the width of each bar is 1, find the area of largest rectangle in the histogram.

Above is a histogram where width of each bar is 1, given height = [2,1,5,6,2,3].

The largest rectangle is shown in the shaded area, which has area = 10 unit.

Example:

```
Input: [2,1,5,6,2,3]
Output: 10
```

题目大意

给出每个直方图的高度，要求在这些直方图之中找到面积最大的矩形，输出矩形的面积。

解题思路

用单调栈依次保存直方图的高度下标，一旦出现高度比栈顶元素小的情况就取出栈顶元素，单独计算一下这个栈顶元素的矩形的高度。然后停在这里(外层循环中的 i--，再 ++，就相当于停在这里了)，继续取出当前最大栈顶的前一个元素，即连续弹出 2 个最大的，以稍小的一个作为矩形的边，宽就是 2 计算面积.....如果停在这里的下标代表的高度一直比栈里面的元素小，就一直弹出，取出最后一个比当前下标大的高度作为矩形的边。宽就是最后一个比当前下标大的高度和当前下标 i 的差值。计算出面积以后不断的更新 maxArea 即可。

代码

```go
package leetcode

func largestRectangleArea(heights []int) int {
  maxArea := 0
  n := len(heights) + 2
  // Add a sentry at the beginning and the end
  getHeight := func(i int) int {
    if i == 0 || n-1 == i {
      return 0
    }
    return heights[i-1]
  }
  st := make([]int, 0, n/2)
  for i := 0; i < n; i++ {
    for len(st) > 0 && getHeight(st[len(st)-1]) > getHeight(i) {
      // pop stack
      idx := st[len(st)-1]
      st = st[:len(st)-1]
      maxArea = max(maxArea, getHeight(idx)*(i-st[len(st)-1]-1))
    }
    // push stack
    st = append(st, i)
  }
  return maxArea
}

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

86. Partition List

题目

Given a linked list and a value x, partition it such that all nodes less than x come before nodes greater than or equal to x.

You should preserve the original relative order of the nodes in each of the two partitions.

Example:

```
Input: head = 1->4->3->2->5->2, x = 3
Output: 1->2->2->4->3->5
```

题目大意

给定一个数 x，比 x 大或等于的数字都要排列在比 x 小的数字后面，并且相对位置不能发生变化。由于相对位置不能发生变化，所以不能用类似冒泡排序的思想。

解题思路

这道题最简单的做法是构造双向链表，不过时间复杂度是 O(n^2)。

(以下描述定义，大于等于 x 的都属于比 x 大)

更优的方法是新构造 2 个链表，一个链表专门存储比 x 小的结点，另一个专门存储比 x 大的结点。在原链表头部开始扫描一边，依次把这两类点归类到 2 个新建链表中，有点入栈的意思。由于是从头开始扫描的原链表，所以原链表中的原有顺序会依旧被保存下来。最后 2 个新链表里面会存储好各自的结果，把这两个链表，比 x 小的链表拼接到 比 x 大的链表的前面，就能得到最后的答案了。

代码

```go
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */

// 解法一 单链表
func partition(head *ListNode, x int) *ListNode {
  beforeHead := &ListNode{Val: 0, Next: nil}
  before := beforeHead
  afterHead := &ListNode{Val: 0, Next: nil}
  after := afterHead
```

```go
  for head != nil {
    if head.Val < x {
      before.Next = head
      before = before.Next
    } else {
      after.Next = head
      after = after.Next
    }
    head = head.Next
  }
  after.Next = nil
  before.Next = afterHead.Next
  return beforeHead.Next
}

// DoublyListNode define
type DoublyListNode struct {
  Val  int
  Prev *DoublyListNode
  Next *DoublyListNode
}

// 解法二 双链表
func partition1(head *ListNode, x int) *ListNode {
  if head == nil || head.Next == nil {
    return head
  }
  DLNHead := genDoublyListNode(head)
  cur := DLNHead
  for cur != nil {
    if cur.Val < x {
      tmp := &DoublyListNode{Val: cur.Val, Prev: nil, Next: nil}
      compareNode := cur
      for compareNode.Prev != nil {
        if compareNode.Val >= x && compareNode.Prev.Val < x {
          break
        }
        compareNode = compareNode.Prev
      }
      if compareNode == DLNHead {
        if compareNode.Val < x {
          cur = cur.Next
          continue
        } else {
          tmp.Next = DLNHead
          DLNHead.Prev = tmp
          DLNHead = tmp
        }
      } else {
```

```go
        tmp.Next = compareNode
        tmp.Prev = compareNode.Prev
        compareNode.Prev.Next = tmp
        compareNode.Prev = tmp
      }
      deleteNode := cur
      if cur.Prev != nil {
        deleteNode.Prev.Next = deleteNode.Next
      }
      if cur.Next != nil {
        deleteNode.Next.Prev = deleteNode.Prev
      }
    }
    cur = cur.Next
  }
  return genListNode(DLNHead)
}

func genDoublyListNode(head *ListNode) *DoublyListNode {
  cur := head.Next
  DLNHead := &DoublyListNode{Val: head.Val, Prev: nil, Next: nil}
  curDLN := DLNHead
  for cur != nil {
    tmp := &DoublyListNode{Val: cur.Val, Prev: curDLN, Next: nil}
    curDLN.Next = tmp
    curDLN = tmp
    cur = cur.Next
  }
  return DLNHead
}

func genListNode(head *DoublyListNode) *ListNode {
  cur := head.Next
  LNHead := &ListNode{Val: head.Val, Next: nil}
  curLN := LNHead
  for cur != nil {
    tmp := &ListNode{Val: cur.Val, Next: nil}
    curLN.Next = tmp
    curLN = tmp
    cur = cur.Next
  }
  return LNHead
}
```

88. Merge Sorted Array

题目

Given two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one sorted array.

Note:

- The number of elements initialized in nums1 and nums2 are m and n respectively.
- You may assume that nums1 has enough space (size that is equal to m + n) to hold additional elements from nums2.

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

题目大意

合并两个已经有序的数组，结果放在第一个数组中，第一个数组假设空间足够大。要求算法时间复杂度足够低。

解题思路

为了不大量移动元素，就要从2个数组长度之和的最后一个位置开始，依次选取两个数组中大的数，从第一个数组的尾巴开始往头放，只要循环一次以后，就生成了合并以后的数组了。

代码

```go
package leetcode

func merge(nums1 []int, m int, nums2 []int, n int) {
  for p := m + n; m > 0 && n > 0; p-- {
    if nums1[m-1] <= nums2[n-1] {
      nums1[p-1] = nums2[n-1]
      n--
    } else {
      nums1[p-1] = nums1[m-1]
      m--
    }
  }
  for ; n > 0; n-- {
    nums1[n-1] = nums2[n-1]
  }
}
```

89. Gray Code

题目

The gray code is a binary numeral system where two successive values differ in only one bit.

Given a non-negative integer *n* representing the total number of bits in the code, print the sequence of gray code. A gray code sequence must begin with 0.

Example 1:

```
Input: 2
Output: [0,1,3,2]
Explanation:
00 - 0
01 - 1
11 - 3
10 - 2

For a given n, a gray code sequence may not be uniquely defined.
For example, [0,2,3,1] is also a valid gray code sequence.

00 - 0
10 - 2
11 - 3
01 - 1
```

Example 2:

```
Input: 0
Output: [0]
Explanation: We define the gray code sequence to begin with 0.
             A gray code sequence of n has size = 2n, which for n = 0 the size is 20 =
1.
             Therefore, for n = 0 the gray code sequence is [0].
```

题目大意

格雷编码是一个二进制数字系统，在该系统中，两个连续的数值仅有一个位数的差异。给定一个代表编码总位数的非负整数 n，打印其格雷编码序列。格雷编码序列必须以 0 开头。

解题思路

- 输出 n 位格雷码
- 格雷码生成规则：以二进制为0值的格雷码为第零项，第一次改变最右边的位元，第二次改变右起第一个为1的位元的左边位元，第三、四次方法同第一、二次，如此反复，即可排列出 n 个位元的格雷码。
- 可以直接模拟，也可以用递归求解。

代码

```go
package leetcode

// 解法一 递归方法，时间复杂度和空间复杂度都较优
func grayCode(n int) []int {
  if n == 0 {
    return []int{0}
  }
  res := []int{}
  num := make([]int, n)
  generateGrayCode(int(1<<uint(n)), 0, &num, &res)
  return res
}

func generateGrayCode(n, step int, num *[]int, res *[]int) {
  if n == 0 {
    return
  }
  *res = append(*res, convertBinary(*num))

  if step%2 == 0 {
    (*num)[len(*num)-1] = flipGrayCode((*num)[len(*num)-1])
  } else {
    index := len(*num) - 1
    for ; index >= 0; index-- {
      if (*num)[index] == 1 {
        break
      }
    }
    if index == 0 {
      (*num)[len(*num)-1] = flipGrayCode((*num)[len(*num)-1])
    } else {
      (*num)[index-1] = flipGrayCode((*num)[index-1])
    }
  }
  generateGrayCode(n-1, step+1, num, res)
  return
}

func convertBinary(num []int) int {
  res, rad := 0, 1
  for i := len(num) - 1; i >= 0; i-- {
```

```
    res += num[i] * rad
    rad *= 2
  }
  return res
}

func flipGrayCode(num int) int {
  if num == 0 {
    return 1
  }
  return 0
}

// 解法二 直译
func grayCode1(n int) []int {
  var l uint = 1 << uint(n)
  out := make([]int, l)
  for i := uint(0); i < l; i++ {
    out[i] = int((i >> 1) ^ i)
  }
  return out
}
```

90. Subsets II

题目

Given a collection of integers that might contain duplicates, ***nums***, return all possible subsets (the power set).

Note: The solution set must not contain duplicate subsets.

Example:

```
Input: [1,2,2]
Output:
[
  [2],
  [1],
  [1,2,2],
  [2,2],
  [1,2],
  []
]
```

题目大意

给定一个可能包含重复元素的整数数组 nums，返回该数组所有可能的子集（幂集）。说明：解集不能包含重复的子集。

解题思路

- 这一题是第 78 题的加强版，比第 78 题多了一个条件，数组中的数字会出现重复。
- 解题方法依旧是 DFS，需要在回溯的过程中加上一些判断。
- 这一题和第 78 题，第 491 题类似，可以一起解答和复习。

代码

```
package leetcode

import (
  "fmt"
  "sort"
)

func subsetsWithDup(nums []int) [][]int {
  c, res := []int{}, [][]int{}
  sort.Ints(nums) // 这里是去重的关键逻辑
  for k := 0; k <= len(nums); k++ {
    generateSubsetsWithDup(nums, k, 0, c, &res)
  }
  return res
}

func generateSubsetsWithDup(nums []int, k, start int, c []int, res *[][]int) {
  if len(c) == k {
    b := make([]int, len(c))
    copy(b, c)
    *res = append(*res, b)
    return
  }
  // i will at most be n - (k - c.size()) + 1
  for i := start; i < len(nums)-(k-len(c))+1; i++ {
    fmt.Printf("i = %v start = %v c = %v\n", i, start, c)
    if i > start && nums[i] == nums[i-1] { // 这里是去重的关键逻辑,本次不取重复数字，下次循环可能会取重复数字
      continue
    }
    c = append(c, nums[i])
    generateSubsetsWithDup(nums, k, i+1, c, res)
    c = c[:len(c)-1]
  }
  return
```

```
}
```

91. Decode Ways

题目

A message containing letters from `A-Z` is being encoded to numbers using the following mapping:

```
'A' -> 1
'B' -> 2
...
'Z' -> 26
```

Given a **non-empty** string containing only digits, determine the total number of ways to decode it.

Example 1:

```
Input: "12"
Output: 2
Explanation: It could be decoded as "AB" (1 2) or "L" (12).
```

Example 2:

```
Input: "226"
Output: 3
Explanation: It could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).
```

题目大意

一条包含字母 A-Z 的消息通过以下方式进行了编码：

```
'A' -> 1
'B' -> 2
...
'Z' -> 26
```

给定一个只包含数字的非空字符串，请计算解码方法的总数。

解题思路

- 给出一个数字字符串，题目要求把数字映射成 26 个字母，映射以后问有多少种可能的翻译方法。
- 这题思路也是 DP。`dp[n]` 代表翻译长度为 n 个字符的字符串的方法总数。由于题目中的数字可能出现 0，0 不能翻译成任何字母，所以出现 0 要跳过。dp[0] 代表空字符串，只有一种翻译方法，`dp[0] = 1`。dp[1] 需要考虑原字符串是否是 0 开头的，如果是 0 开头的，`dp[1] = 0`，如果不是 0 开头的，`dp[1] = 1`。状态

转移方程是 `dp[i] += dp[i-1] (当 1 ≤ s[i-1 : i] ≤ 9)`; `dp[i] += dp[i-2] (当 10 ≤ s[i-2 : i] ≤ 26)`。最终结果是 `dp[n]`。

代码

```
package leetcode

func numDecodings(s string) int {
  n := len(s)
  dp := make([]int, n+1)
  dp[0] = 1
  for i := 1; i <= n; i++ {
    if s[i-1] != '0' {
      dp[i] += dp[i-1]
    }
    if i > 1 && s[i-2] != '0' && (s[i-2]-'0')*10+(s[i-1]-'0') <= 26 {
      dp[i] += dp[i-2]
    }
  }
  return dp[n]
}
```

92. Reverse Linked List II

题目

Reverse a linked list from position m to n. Do it in one-pass.

Note: 1 ≤ m ≤ n ≤ length of list.

Example:

```
Input: 1->2->3->4->5->NULL, m = 2, n = 4
Output: 1->4->3->2->5->NULL
```

题目大意

给定 2 个链表中结点的位置 m, n，反转这个两个位置区间内的所有结点。

解题思路

由于有可能整个链表都被反转，所以构造一个新的头结点指向当前的头。之后的处理方法是：找到第一个需要反转的结点的前一个结点 p，从这个结点开始，依次把后面的结点用“头插”法，插入到 p 结点的后面。循环次数用 n-m 来控制。

这一题结点可以原地变化，更改各个结点的 next 指针就可以。不需要游标 p 指针。因为每次逆序以后，原有结点的相对位置就发生了变化，相当于游标指针已经移动了，所以不需要再有游标 p = p.Next 的操作了。

代码

```go
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */

func reverseBetween(head *ListNode, m int, n int) *ListNode {
  if head == nil || m >= n {
    return head
  }
  newHead := &ListNode{Val: 0, Next: head}
  pre := newHead
  for count := 0; pre.Next != nil && count < m-1; count++ {
    pre = pre.Next
  }
  if pre.Next == nil {
    return head
  }
  cur := pre.Next
  for i := 0; i < n-m; i++ {
    tmp := pre.Next
    pre.Next = cur.Next
    cur.Next = cur.Next.Next
    pre.Next.Next = tmp
  }
  return newHead.Next
}
```

93. Restore IP Addresses

题目

Given a string containing only digits, restore it by returning all possible valid IP address combinations.

Example:

```
Input: "25525511135"
Output: ["255.255.11.135", "255.255.111.35"]
```

题目大意

给定一个只包含数字的字符串，复原它并返回所有可能的 IP 地址格式。

解题思路

- DFS 深搜
- 需要注意的点是 IP 的规则，以 0 开头的数字和超过 255 的数字都为非法的。

代码

```go
package leetcode

import (
  "strconv"
)

func restoreIPAddresses(s string) []string {
  if s == "" {
    return []string{}
  }
  res, ip := []string{}, []int{}
  dfs(s, 0, ip, &res)
  return res
}

func dfs(s string, index int, ip []int, res *[]string) {
  if index == len(s) {
    if len(ip) == 4 {
      *res = append(*res, getString(ip))
    }
    return
  }
  if index == 0 {
    num, _ := strconv.Atoi(string(s[0]))
    ip = append(ip, num)
    dfs(s, index+1, ip, res)
  } else {
    num, _ := strconv.Atoi(string(s[index]))
    next := ip[len(ip)-1]*10 + num
    if next <= 255 && ip[len(ip)-1] != 0 {
      ip[len(ip)-1] = next
```

```
      dfs(s, index+1, ip, res)
      ip[len(ip)-1] /= 10
    }
    if len(ip) < 4 {
      ip = append(ip, num)
      dfs(s, index+1, ip, res)
      ip = ip[:len(ip)-1]
    }
  }
}

func getString(ip []int) string {
  res := strconv.Itoa(ip[0])
  for i := 1; i < len(ip); i++ {
    res += "." + strconv.Itoa(ip[i])
  }
  return res
}
```

94. Binary Tree Inorder Traversal

题目

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?

题目大意

中根遍历一颗树。

解题思路

递归的实现方法，见代码。

代码

```
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func inorderTraversal(root *TreeNode) []int {
  var result []int
  inorder(root, &result)
  return result
}

func inorder(root *TreeNode, output *[]int) {
  if root != nil {
    inorder(root.Left, output)
    *output = append(*output, root.Val)
    inorder(root.Right, output)
  }
}
```

95. Unique Binary Search Trees II

题目

Given an integer *n*, generate all structurally unique **BST's** (binary search trees) that store values 1 ... *n*.

Example:

```
Input: 3
Output:
[
  [1,null,3,2],
```

```
  [3,2,null,1],
  [3,1,null,null,2],
  [2,1,3],
  [1,null,2,null,3]
]
Explanation:
The above output corresponds to the 5 unique BST's shown below:

   1         3     3      2      1
    \       /     /      / \      \
     3     2     1      1   3      2
    /     /       \                 \
   2     1         2                 3
```

题目大意

给定一个整数 n，生成所有由 1 ... n 为节点所组成的二叉搜索树。

解题思路

- 输出 1~n 元素组成的 BST 所有解。这一题递归求解即可。外层循环遍历 1~n 所有结点，作为根结点，内层双层递归分别求出左子树和右子树。

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func generateTrees(n int) []*TreeNode {
  if n == 0 {
    return []*TreeNode{}
  }
  return generateBSTree(1, n)
}

func generateBSTree(start, end int) []*TreeNode {
  tree := []*TreeNode{}
  if start > end {
    tree = append(tree, nil)
    return tree
```

```
  }
  for i := start; i <= end; i++ {
    left := generateBSTree(start, i-1)
    right := generateBSTree(i+1, end)
    for _, l := range left {
      for _, r := range right {
        root := &TreeNode{Val: i, Left: l, Right: r}
        tree = append(tree, root)
      }
    }
  }
  return tree
}
```

96. Unique Binary Search Trees

题目

Given *n*, how many structurally unique **BST's** (binary search trees) that store values 1 ... *n*?

Example:

```
Input: 3
Output: 5
Explanation:
Given n = 3, there are a total of 5 unique BST's:

   1         3     3      2      1
    \       /     /      / \      \
     3     2     1      1   3      2
    /     /       \                 \
   2     1         2                 3
```

题目大意

给定一个整数 n，求以 1 ... n 为节点组成的二叉搜索树有多少种?

解题思路

- 给出 n，要求利用 1-n 这些数字组成二叉排序树，有多少种不同的树的形态，输出这个个数。
- 这题的解题思路是 DP。`dp[n]` 代表 1-n 个数能组成多少个不同的二叉排序树，`F(i,n)` 代表以 `i` 为根节点，1-n 个数组成的二叉排序树的不同的个数。由于题意，我们可以得到这个等式：`dp[n] = F(1,n) + F(2,n) + F(3,n) + …… + F(n,n)` 。初始值 `dp[0] = 1`，`dp[1] = 1`。分析 `dp` 和 `F(i,n)` 的关系又可以得到下面这个等式 `F(i,n) = dp[i-1] * dp[n-i]` 。举例，[1,2,3,4,..., i ,...,n-1,n]，以 `i` 为 根节点，那么左半边 [1,2,3,.....,i-1] 和 右半边 [i+1,i+2,.....,n-1,n] 分别能组成二叉排序树的不同个数 `相乘`，即为以 `i` 为根节点，1-n 个数组成的二叉排序树的不同的个数，也即 `F(i,n)`。

> 注意，由于二叉排序树本身的性质，右边的子树一定比左边的子树，值都要大。所以这里只需要根节点把树分成左右，不需要再关心左右两边数字的大小，只需要关心数字的个数。

- 所以状态转移方程是 `dp[i] = dp[0] * dp[n-1] + dp[1] * dp[n-2] + …… + dp[n-1] * dp[0]`，最终要求的结果是 `dp[n]` 。

代码

```
package leetcode

func numTrees(n int) int {
  dp := make([]int, n+1)
  dp[0], dp[1] = 1, 1
  for i := 2; i <= n; i++ {
    for j := 1; j <= i; j++ {
      dp[i] += dp[j-1] * dp[i-j]
    }
  }
  return dp[n]
}
```

97. Interleaving String

题目

Given strings `s1`, `s2`, and `s3`, find whether `s3` is formed by an **interleaving** of `s1` and `s2`.

An **interleaving** of two strings `s` and `t` is a configuration where they are divided into **non-empty** substrings such that:

- `s = s1 + s2 + ... + sn`
- `t = t1 + t2 + ... + tm`
- `|n - m| <= 1`
- The **interleaving** is `s1 + t1 + s2 + t2 + s3 + t3 + ...` or `t1 + s1 + t2 + s2 + t3 + s3 + ...`

Note: `a + b` is the concatenation of strings `a` and `b`.

Example 1:

```
Input: s1 = "aabcc", s2 = "dbbca", s3 = "aadbbcbcac"
Output: true
```

Example 2:

```
Input: s1 = "aabcc", s2 = "dbbca", s3 = "aadbbbaccc"
Output: false
```

Example 3:

```
Input: s1 = "", s2 = "", s3 = ""
Output: true
```

Constraints:

- `0 <= s1.length, s2.length <= 100`
- `0 <= s3.length <= 200`
- `s1`, `s2`, and `s3` consist of lowercase English letters.

Follow up: Could you solve it using only `O(s2.length)` additional memory space?

题目大意

给定三个字符串 s1、s2、s3，请你帮忙验证 s3 是否是由 s1 和 s2 交错 组成的。两个字符串 s 和 t 交错 的定义与过程如下，其中每个字符串都会被分割成若干 非空 子字符串：

- s = s1 + s2 + ... + sn
- t = t1 + t2 + ... + tm
- |n - m| <= 1
- 交错 是 s1 + t1 + s2 + t2 + s3 + t3 + ... 或者 t1 + s1 + t2 + s2 + t3 + s3 + ...

提示：a + b 意味着字符串 a 和 b 连接。

解题思路

- 深搜或者广搜暴力解题。笔者用深搜实现的。记录 s1 和 s2 串当前比较的位置 p1 和 p2。如果 s3[p1+p2] 的位置上等于 s1[p1] 或者 s2[p2] 代表能匹配上，那么继续往后移动 p1 和 p2 相应的位置。因为是交错字符串，所以判断匹配的位置是 s3[p1+p2] 的位置。如果仅仅这么写，会超时，s1 和 s2 两个字符串重复交叉判断的位置太多了。需要加上记忆化搜索。可以用 visited[i][j] 这样的二维数组来记录是否搜索过了。笔者为了压缩空间，将 i 和 j 编码压缩到一维数组了。i * len(s3) + j 是唯一下标，所以可以用这种方式存储是否搜索过。具体代码见下面的实现。

代码

```
package leetcode

func isInterleave(s1 string, s2 string, s3 string) bool {
  if len(s1)+len(s2) != len(s3) {
    return false
  }
  visited := make(map[int]bool)
  return dfs(s1, s2, s3, 0, 0, visited)
}

func dfs(s1, s2, s3 string, p1, p2 int, visited map[int]bool) bool {
  if p1+p2 == len(s3) {
    return true
  }
  if _, ok := visited[(p1*len(s3))+p2]; ok {
    return false
  }
  visited[(p1*len(s3))+p2] = true
  var match1, match2 bool
  if p1 < len(s1) && s3[p1+p2] == s1[p1] {
    match1 = true
  }
  if p2 < len(s2) && s3[p1+p2] == s2[p2] {
    match2 = true
  }
  if match1 && match2 {
    return dfs(s1, s2, s3, p1+1, p2, visited) || dfs(s1, s2, s3, p1, p2+1, visited)
  } else if match1 {
    return dfs(s1, s2, s3, p1+1, p2, visited)
  } else if match2 {
    return dfs(s1, s2, s3, p1, p2+1, visited)
  } else {
    return false
  }
}
```

98. Validate Binary Search Tree

题目

Given a binary tree, determine if it is a valid binary search tree (BST).

Assume a BST is defined as follows:

- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

Example 1:

```
    2
   / \
  1   3

Input: [2,1,3]
Output: true
```

Example 2:

```
    5
   / \
  1   4
     / \
    3   6

Input: [5,1,4,null,null,3,6]
Output: false
Explanation: The root node's value is 5 but its right child's value is 4.
```

题目大意

给定一个二叉树，判断其是否是一个有效的二叉搜索树。假设一个二叉搜索树具有如下特征：

- 节点的左子树只包含小于当前节点的数。
- 节点的右子树只包含大于当前节点的数。
- 所有左子树和右子树自身必须也是二叉搜索树。

解题思路

- 判断一个树是否是 BST，按照定义递归判断即可

代码

```go
package leetcode

import "math"
```

```go
/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

// 解法一，直接按照定义比较大小，比 root 节点小的都在左边，比 root 节点大的都在右边
func isValidBST(root *TreeNode) bool {
  return isValidbst(root, math.Inf(-1), math.Inf(1))
}
func isValidbst(root *TreeNode, min, max float64) bool {
  if root == nil {
    return true
  }
  v := float64(root.Val)
  return v < max && v > min && isValidbst(root.Left, min, v) && isValidbst(root.Right,
v, max)
}

// 解法二，把 BST 按照左中右的顺序输出到数组中，如果是 BST，则数组中的数字是从小到大有序的，如果出现逆序就不是 BST
func isValidBST1(root *TreeNode) bool {
  arr := []int{}
  inOrder(root, &arr)
  for i := 1; i < len(arr); i++ {
    if arr[i-1] >= arr[i] {
      return false
    }
  }
  return true
}

func inOrder(root *TreeNode, arr *[]int) {
  if root == nil {
    return
  }
  inOrder(root.Left, arr)
  *arr = append(*arr, root.Val)
  inOrder(root.Right, arr)
}
```

99. Recover Binary Search Tree

题目

Two elements of a binary search tree (BST) are swapped by mistake.

Recover the tree without changing its structure.

Example 1:

```
Input: [1,3,null,null,2]

   1
  /
 3
  \
   2

Output: [3,1,null,null,2]

   3
  /
 1
  \
   2
```

Example 2:

```
Input: [3,1,4,null,null,2]

  3
 / \
1   4
   /
  2

Output: [2,1,4,null,null,3]

  2
 / \
1   4
   /
  3
```

Follow up:

- A solution using O(*n*) space is pretty straight forward.
- Could you devise a constant space solution?

题目大意

二叉搜索树中的两个节点被错误地交换。请在不改变其结构的情况下，恢复这棵树。

解题思路

- 在二叉搜索树中，有 2 个结点的值出错了，要求修复这两个结点。
- 这一题按照先根遍历 1 次就可以找到这两个出问题的结点，因为先访问根节点，然后左孩子，右孩子。用先根遍历二叉搜索树的时候，根结点比左子树都要大，根结点比右子树都要小。所以`左子树比根结点大的话，就是出现了乱序`；`根节点比右子树大的话，就是出现了乱序`。遍历过程中在左子树中如果出现了前一次遍历的结点的值大于此次根节点的值，这就出现了出错结点了，记录下来。继续遍历直到找到第二个这样的结点。最后交换这两个结点的时候，只是交换他们的值就可以了，而不是交换这两个结点相应的指针指向。

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func recoverTree(root *TreeNode) {
  var prev, target1, target2 *TreeNode
  _, target1, target2 = inOrderTraverse(root, prev, target1, target2)
  if target1 != nil && target2 != nil {
    target1.Val, target2.Val = target2.Val, target1.Val
  }

}

func inOrderTraverse(root, prev, target1, target2 *TreeNode) (*TreeNode, *TreeNode,
*TreeNode) {
  if root == nil {
    return prev, target1, target2
  }
  prev, target1, target2 = inOrderTraverse(root.Left, prev, target1, target2)
  if prev != nil && prev.Val > root.Val {
    if target1 == nil {
      target1 = prev
    }
    target2 = root
  }
  prev = root
  prev, target1, target2 = inOrderTraverse(root.Right, prev, target1, target2)
  return prev, target1, target2
```

```
}
```

100. Same Tree

题目

Given two binary trees, 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:     1         1
          / \       / \
         2   3     2   3

        [1,2,3],   [1,2,3]

Output: true
```

Example 2:

```
Input:     1         1
          /           \
         2             2

        [1,2],     [1,null,2]

Output: false
```

Example 3:

```
Input:     1         1
          / \       / \
         2   1     1   2

        [1,2,1],   [1,1,2]

Output: false
```

题目大意

这一题要求判断 2 颗树是否是完全相等的。

解题思路

递归判断即可。

代码

```
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
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 false
    }
    return isSameTree(p.Left, q.Left) && isSameTree(p.Right, q.Right)
  } else {
    return false
  }
}
```

101. Symmetric Tree

题目

Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).

For example, this binary tree [1,2,2,3,4,4,3] is symmetric:

```
    1
   / \
  2   2
 / \ / \
3  4 4  3
```

But the following [1,2,2,null,3,null,3] is not:

```
    1
   / \
  2   2
   \   \
   3    3
```

Note:

Bonus points if you could solve it both recursively and iteratively.

题目大意

这一题要求判断 2 颗树是否是左右对称的。

解题思路

- 这道题是几道题的综合题。将根节点的左字数反转二叉树，然后再和根节点的右节点进行比较，是否完全相等。
- 反转二叉树是第 226 题。判断 2 颗树是否完全相等是第 100 题。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/structures"
)

// TreeNode define
type TreeNode = structures.TreeNode

/**
 * Definition for a binary tree node.
```

```go
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

// 解法一 dfs
func isSymmetric(root *TreeNode) bool {
  return root == nil || dfs(root.Left, root.Right)
}

func dfs(rootLeft, rootRight *TreeNode) bool {
  if rootLeft == nil && rootRight == nil {
    return true
  }
  if rootLeft == nil || rootRight == nil {
    return false
  }
  if rootLeft.Val != rootRight.Val {
    return false
  }
  return dfs(rootLeft.Left, rootRight.Right) && dfs(rootLeft.Right, rootRight.Left)
}

// 解法二
func isSymmetric1(root *TreeNode) bool {
  if root == nil {
    return true
  }
  return isSameTree(invertTree(root.Left), root.Right)
}

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 false
    }
    return isSameTree(p.Left, q.Left) && isSameTree(p.Right, q.Right)
  } else {
    return false
  }
}

func invertTree(root *TreeNode) *TreeNode {
  if root == nil {
    return nil
```

```
  }
  invertTree(root.Left)
  invertTree(root.Right)
  root.Left, root.Right = root.Right, root.Left
  return root
}
```

102. Binary Tree Level Order Traversal

题目

Given a binary tree, return the level order traversal of its nodes' values. (ie, from left to right, level by level).

For Example:

Given binary tree [3,9,20,null,null,15,7],

```
    3
   / \
  9  20
    /  \
   15   7
```

return its level order traversal as:

```
[
  [3],
  [9,20],
  [15,7]
]
```

题目大意

按层序从上到下遍历一颗树。

解题思路

用一个队列即可实现。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/structures"
)

// TreeNode define
type TreeNode = structures.TreeNode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

// 解法一 BFS
func levelOrder(root *TreeNode) [][]int {
  if root == nil {
    return [][]int{}
  }
  queue := []*TreeNode{root}
  res := make([][]int, 0)
  for len(queue) > 0 {
    l := len(queue)
    tmp := make([]int, 0, l)
    for i := 0; i < l; i++ {
      if queue[i].Left != nil {
        queue = append(queue, queue[i].Left)
      }
      if queue[i].Right != nil {
        queue = append(queue, queue[i].Right)
      }
      tmp = append(tmp, queue[i].Val)
    }
    queue = queue[l:]
    res = append(res, tmp)
  }
  return res
}

// 解法二 DFS
func levelOrder1(root *TreeNode) [][]int {
  var res [][]int
```

```go
  var dfsLevel func(node *TreeNode, level int)
  dfsLevel = func(node *TreeNode, level int) {
    if node == nil {
      return
    }
    if len(res) == level {
      res = append(res, []int{node.Val})
    } else {
      res[level] = append(res[level], node.Val)
    }
    dfsLevel(node.Left, level+1)
    dfsLevel(node.Right, level+1)
  }
  dfsLevel(root, 0)
  return res
}
```

103. Binary Tree Zigzag Level Order Traversal

题目

Given a binary tree, return the zigzag level order traversal of its nodes' values. (ie, from left to right, then right to left for the next level and alternate between).

For Example:
Given binary tree [3,9,20,null,null,15,7],

```
    3
   / \
  9  20
    /  \
   15   7
```

return its zigzag level order traversal as:

```
[
  [3],
  [20,9],
  [15,7]
]
```

题目大意

按照 Z 字型层序遍历一棵树。

解题思路

- 按层序从上到下遍历一颗树，但是每一层的顺序是相互反转的，即上一层是从左往右，下一层就是从右往左，以此类推。用一个队列即可实现。
- 第 102 题和第 107 题都是按层序遍历的。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/structures"
)

// TreeNode define
type TreeNode = structures.TreeNode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

// 解法一
func zigzagLevelOrder(root *TreeNode) [][]int {
  if root == nil {
    return [][]int{}
  }
  queue := []*TreeNode{}
  queue = append(queue, root)
  curNum, nextLevelNum, res, tmp, curDir := 1, 0, [][]int{}, []int{}, 0
  for len(queue) != 0 {
    if curNum > 0 {
      node := queue[0]
      if node.Left != nil {
        queue = append(queue, node.Left)
        nextLevelNum++
      }
      if node.Right != nil {
```

```go
        queue = append(queue, node.Right)
        nextLevelNum++
      }
      curNum--
      tmp = append(tmp, node.Val)
      queue = queue[1:]
    }
    if curNum == 0 {
      if curDir == 1 {
        for i, j := 0, len(tmp)-1; i < j; i, j = i+1, j-1 {
          tmp[i], tmp[j] = tmp[j], tmp[i]
        }
      }
      res = append(res, tmp)
      curNum = nextLevelNum
      nextLevelNum = 0
      tmp = []int{}
      if curDir == 0 {
        curDir = 1
      } else {
        curDir = 0
      }
    }
  }
  return res
}

// 解法二 递归
func zigzagLevelOrder0(root *TreeNode) [][]int {
  var res [][]int
  search(root, 0, &res)
  return res
}

func search(root *TreeNode, depth int, res *[][]int) {
  if root == nil {
    return
  }
  for len(*res) < depth+1 {
    *res = append(*res, []int{})
  }
  if depth%2 == 0 {
    (*res)[depth] = append((*res)[depth], root.Val)
  } else {
    (*res)[depth] = append([]int{root.Val}, (*res)[depth]...)
  }
  search(root.Left, depth+1, res)
  search(root.Right, depth+1, res)
}
```

```go
// 解法三 BFS
func zigzagLevelOrder1(root *TreeNode) [][]int {
  res := [][]int{}
  if root == nil {
    return res
  }
  q := []*TreeNode{root}
  size, i, j, lay, tmp, flag := 0, 0, 0, []int{}, []*TreeNode{}, false
  for len(q) > 0 {
    size = len(q)
    tmp = []*TreeNode{}
    lay = make([]int, size)
    j = size - 1
    for i = 0; i < size; i++ {
      root = q[0]
      q = q[1:]
      if !flag {
        lay[i] = root.Val
      } else {
        lay[j] = root.Val
        j--
      }
      if root.Left != nil {
        tmp = append(tmp, root.Left)
      }
      if root.Right != nil {
        tmp = append(tmp, root.Right)
      }

    }
    res = append(res, lay)
    flag = !flag
    q = tmp
  }
  return res
}
```

104. Maximum Depth of Binary Tree

题目

Given a binary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

Note: A leaf is a node with no children.

Example:

Given binary tree [3,9,20,null,null,15,7],

```
    3
   / \
  9  20
    /  \
   15   7
```

return its depth = 3.

题目大意

要求输出一棵树的最大高度。

解题思路

这一题递归遍历就可，遍历根节点的左孩子的高度和根节点右孩子的高度，取出两者的最大值再加一即为总高度。

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func maxDepth(root *TreeNode) int {
  if root == nil {
    return 0
  }
  return max(maxDepth(root.Left), maxDepth(root.Right)) + 1
}
```

105. Construct Binary Tree from Preorder and Inorder Traversal

题目

Given preorder and inorder traversal of a tree, construct the binary tree.

Note:You may assume that duplicates do not exist in the tree.

For example, given

```
preorder = [3,9,20,15,7]
inorder = [9,3,15,20,7]
```

Return the following binary tree:

```
  3
   / \
  9  20
    /  \
   15   7
```

题目大意

根据一棵树的前序遍历与中序遍历构造二叉树。

注意:
你可以假设树中没有重复的元素。

解题思路

- 给出 2 个数组，根据 preorder 和 inorder 数组构造一颗树。
- 利用递归思想，从 preorder 可以得到根节点，从 inorder 中得到左子树和右子树。只剩一个节点的时候即为根节点。不断的递归直到所有的树都生成完成。

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
```

```
 *     Right *TreeNode
 * }
 */
func buildTree(preorder []int, inorder []int) *TreeNode {
  inPos := make(map[int]int)
  for i := 0; i < len(inorder); i++ {
    inPos[inorder[i]] = i
  }
  return buildPreIn2TreeDFS(preorder, 0, len(preorder)-1, 0, inPos)
}

func buildPreIn2TreeDFS(pre []int, preStart int, preEnd int, inStart int, inPos
map[int]int) *TreeNode {
  if preStart > preEnd {
    return nil
  }
  root := &TreeNode{Val: pre[preStart]}
  rootIdx := inPos[pre[preStart]]
  leftLen := rootIdx - inStart
  root.Left = buildPreIn2TreeDFS(pre, preStart+1, preStart+leftLen, inStart, inPos)
  root.Right = buildPreIn2TreeDFS(pre, preStart+leftLen+1, preEnd, rootIdx+1, inPos)
  return root
}
```

106. Construct Binary Tree from Inorder and Postorder Traversal

题目

Given inorder and postorder traversal of a tree, construct the binary tree.

Note: You may assume that duplicates do not exist in the tree.

For example, given

```
inorder = [9,3,15,20,7]
postorder = [9,15,7,20,3]
```

Return the following binary tree:

```
  3
   / \
  9  20
    /  \
   15   7
```

题目大意

根据一棵树的中序遍历与后序遍历构造二叉树。

注意:
你可以假设树中没有重复的元素。

解题思路

- 给出 2 个数组，根据 inorder 和 postorder 数组构造一颗树。
- 利用递归思想，从 postorder 可以得到根节点，从 inorder 中得到左子树和右子树。只剩一个节点的时候即为根节点。不断的递归直到所有的树都生成完成。

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func buildTree106(inorder []int, postorder []int) *TreeNode {
  inPos := make(map[int]int)
  for i := 0; i < len(inorder); i++ {
    inPos[inorder[i]] = i
  }
  return buildInPos2TreeDFS(postorder, 0, len(postorder)-1, 0, inPos)
}

func buildInPos2TreeDFS(post []int, postStart int, postEnd int, inStart int, inPos
map[int]int) *TreeNode {
  if postStart > postEnd {
    return nil
  }
  root := &TreeNode{Val: post[postEnd]}
  rootIdx := inPos[post[postEnd]]
  leftLen := rootIdx - inStart
  root.Left = buildInPos2TreeDFS(post, postStart, postStart+leftLen-1, inStart, inPos)
  root.Right = buildInPos2TreeDFS(post, postStart+leftLen, postEnd-1, rootIdx+1, inPos)
  return root
}
```

107. Binary Tree Level Order Traversal II

题目

Given a binary tree, return the bottom-up level order traversal of its nodes' values. (ie, from left to right, level by level from leaf to root).

For Example:

Given binary tree [3,9,20,null,null,15,7],

```
    3
   / \
  9  20
    /  \
   15   7
```

return its bottom-up level order traversal as:

```
[
  [15,7],
  [9,20],
  [3]
]
```

题目大意

按层序从下到上遍历一颗树。

解题思路

用一个队列即可实现。

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
```

```go
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func levelOrderBottom(root *TreeNode) [][]int {
  tmp := levelOrder(root)
  res := [][]int{}
  for i := len(tmp) - 1; i >= 0; i-- {
    res = append(res, tmp[i])
  }
  return res
}
```

108. Convert Sorted Array to Binary Search Tree

题目

Given an array where elements are sorted in ascending order, convert it to a height balanced BST.

For this problem, a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees of *every* node never differ by more than 1.

Example:

```
Given the sorted array: [-10,-3,0,5,9],

One possible answer is: [0,-3,9,-10,null,5], which represents the following height
balanced BST:

      0
     / \
   -3   9
   /   /
 -10  5
```

题目大意

将一个按照升序排列的有序数组，转换为一棵高度平衡二叉搜索树。本题中，一个高度平衡二叉树是指一个二叉树每个节点 的左右两个子树的高度差的绝对值不超过 1。

解题思路

- 把一个有序数组转换成高度平衡的二叉搜索数，按照定义即可

代码

```
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func sortedArrayToBST(nums []int) *TreeNode {
  if len(nums) == 0 {
    return nil
  }
  return &TreeNode{Val: nums[len(nums)/2], Left: sortedArrayToBST(nums[:len(nums)/2]),
Right: sortedArrayToBST(nums[len(nums)/2+1:])}
}
```

109. Convert Sorted List to Binary Search Tree

题目

Given a singly linked list where elements are sorted in ascending order, convert it to a height balanced BST.

For this problem, a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees of every node never differ by more than 1.

Example:

```
Given the sorted linked list: [-10,-3,0,5,9],

One possible answer is: [0,-3,9,-10,null,5], which represents the following height
balanced BST:

      0
     / \
   -3   9
   /   /
 -10  5
```

题目大意

将链表转化为高度平衡的二叉搜索树。高度平衡的定义：每个结点的 2 个子结点的深度不能相差超过 1 。

解题思路

思路比较简单，依次把链表的中间点作为根结点，类似二分的思想，递归排列所有结点即可。

代码

```
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */
/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

// TreeNode define
type TreeNode struct {
  Val   int
  Left  *TreeNode
  Right *TreeNode
}

func sortedListToBST(head *ListNode) *TreeNode {
  if head == nil {
    return nil
  }
  if head != nil && head.Next == nil {
    return &TreeNode{Val: head.Val, Left: nil, Right: nil}
  }
  middleNode, preNode := middleNodeAndPreNode(head)
  if middleNode == nil {
    return nil
  }
  if preNode != nil {
    preNode.Next = nil
```

```go
  }
  if middleNode == head {
    head = nil
  }
  return &TreeNode{Val: middleNode.Val, Left: sortedListToBST(head), Right:
sortedListToBST(middleNode.Next)}
}

func middleNodeAndPreNode(head *ListNode) (middle *ListNode, pre *ListNode) {
  if head == nil || head.Next == nil {
    return nil, head
  }
  p1 := head
  p2 := head
  for p2.Next != nil && p2.Next.Next != nil {
    pre = p1
    p1 = p1.Next
    p2 = p2.Next.Next
  }
  return p1, pre
}
```

110. Balanced Binary Tree

题目

Given a binary tree, determine if it is height-balanced.

For this problem, a height-balanced binary tree is defined as:

a binary tree in which the depth of the two subtrees of every node never differ by more than 1.

Example 1:

Given the following tree [3,9,20,null,null,15,7]:

```
    3
   / \
  9  20
    /  \
   15   7
```

Return true.

Example 2:

Given the following tree [1,2,2,3,3,null,null,4,4]:

```
       1
      / \
     2   2
    / \
   3   3
  / \
 4   4
```

Return false.

题目大意

判断一棵树是不是平衡二叉树。平衡二叉树的定义是：树中每个节点都满足左右两个子树的高度差 <= 1 的这个条件。

解题思路

根据定义判断即可，计算树的高度是第 104 题。

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func isBalanced(root *TreeNode) bool {
  if root == nil {
    return true
  }
  leftHight := depth(root.Left)
  rightHight := depth(root.Right)
  return abs(leftHight-rightHight) <= 1 && isBalanced(root.Left) &&
isBalanced(root.Right)
}

func depth(root *TreeNode) int {
  if root == nil {
    return 0
```

```
    }
    return max(depth(root.Left), depth(root.Right)) + 1
}
```

111. Minimum Depth of Binary Tree

题目

Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

Note: A leaf is a node with no children.

Example:

Given binary tree `[3,9,20,null,null,15,7]`,

```
  3
   / \
  9  20
    /  \
   15   7
```

return its minimum depth = 2.

题目大意

给定一个二叉树，找出其最小深度。最小深度是从根节点到最近叶子节点的最短路径上的节点数量。说明: 叶子节点是指没有子节点的节点。

解题思路

- 递归求出根节点到叶子节点的深度，输出最小值即可

代码

```
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
```

```go
 *     Right *TreeNode
 * }
 */
func minDepth(root *TreeNode) int {
  if root == nil {
    return 0
  }
  if root.Left == nil {
    return minDepth(root.Right) + 1
  }
  if root.Right == nil {
    return minDepth(root.Left) + 1
  }
  return min(minDepth(root.Left), minDepth(root.Right)) + 1
}
```

112. Path Sum

题目

Given a binary tree and a sum, determine if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum.

Note: A leaf is a node with no children.

Example:

Given the below binary tree and `sum = 22`,

```
      5
     / \
    4   8
   /   / \
  11  13  4
 /  \      \
7    2      1
```

return true, as there exist a root-to-leaf path `5->4->11->2` which sum is 22.

题目大意

给定一个二叉树和一个目标和，判断该树中是否存在根节点到叶子节点的路径，这条路径上所有节点值相加等于目标和。说明: 叶子节点是指没有子节点的节点。

解题思路

- 递归求解即可

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func hasPathSum(root *TreeNode, sum int) bool {
  if root == nil {
    return false
  }
  if root.Left == nil && root.Right == nil {
    return sum == root.Val
  }
  return hasPathSum(root.Left, sum-root.Val) || hasPathSum(root.Right, sum-root.Val)
}
```

113. Path Sum II

题目

Given a binary tree and a sum, find all root-to-leaf paths where each path's sum equals the given sum.

Note: A leaf is a node with no children.

Example:

Given the below binary tree and `sum = 22`,

```
      5
     / \
    4   8
   /   / \
  11  13  4
 /  \    / \
7    2  5   1
```

Return:

```
[
   [5,4,11,2],
   [5,8,4,5]
]
```

题目大意

给定一个二叉树和一个目标和，找到所有从根节点到叶子节点路径总和等于给定目标和的路径。说明: 叶子节点是指没有子节点的节点。

解题思路

- 这一题是第 257 题和第 112 题的组合增强版

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

// 解法一
func pathSum(root *TreeNode, sum int) [][]int {
  var slice [][]int
  slice = findPath(root, sum, slice, []int(nil))
  return slice
}

func findPath(n *TreeNode, sum int, slice [][]int, stack []int) [][]int {
  if n == nil {
    return slice
  }
  sum -= n.Val
  stack = append(stack, n.Val)
  if sum == 0 && n.Left == nil && n.Right == nil {
    slice = append(slice, append([]int{}, stack...))
    stack = stack[:len(stack)-1]
  }
  slice = findPath(n.Left, sum, slice, stack)
```

```go
  slice = findPath(n.Right, sum, slice, stack)
  return slice
}

// 解法二
func pathSum1(root *TreeNode, sum int) [][]int {
  if root == nil {
    return [][]int{}
  }
  if root.Left == nil && root.Right == nil {
    if sum == root.Val {
      return [][]int{[]int{root.Val}}
    }
  }
  path, res := []int{}, [][]int{}
  tmpLeft := pathSum(root.Left, sum-root.Val)
  path = append(path, root.Val)
  if len(tmpLeft) > 0 {
    for i := 0; i < len(tmpLeft); i++ {
      tmpLeft[i] = append(path, tmpLeft[i]...)
    }
    res = append(res, tmpLeft...)
  }
  path = []int{}
  tmpRight := pathSum(root.Right, sum-root.Val)
  path = append(path, root.Val)

  if len(tmpRight) > 0 {
    for i := 0; i < len(tmpRight); i++ {
      tmpRight[i] = append(path, tmpRight[i]...)
    }
    res = append(res, tmpRight...)
  }
  return res
}
```

114. Flatten Binary Tree to Linked List

题目

Given a binary tree, flatten it to a linked list in-place.

For example, given the following tree:

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

The flattened tree should look like:

```
1
 \
  2
   \
    3
     \
      4
       \
        5
         \
          6
```

题目大意

给定一个二叉树，原地将它展开为链表。

解题思路

- 要求把二叉树“打平”，按照先根遍历的顺序，把树的结点都放在右结点中。
- 按照递归和非递归思路实现即可。
- 递归的思路可以这么想：倒序遍历一颗树，即是先遍历右孩子，然后遍历左孩子，最后再遍历根节点。

  ```
  1
  ```

  / \
  2 5
  / \ \

  ## 3 4 6

  pre = 5
  cur = 4

  ```
  1
  ```

```
  /
 2
/ \
3   4
    \
     5
      \
```

6

pre = 4
cur = 3

```
1
```

```
  /
 2
/
3
 \
  4
   \
    5
     \
```

6

cur = 2
pre = 3

```
1
```

```
  /
 2
  \
   3
    \
     4
      \
       5
        \
```

6

cur = 1
pre = 2

```
1
 \
  2
   \
    3
     \
      4
       \
        5
         \
          6
```

- 可以先仿造先根遍历的代码，写出这个倒序遍历的逻辑：

```
public void flatten(TreeNode root) {
    if (root == null)
        return;
    flatten(root.right);
    flatten(root.left);
}
```

- 实现了倒序遍历的逻辑以后，再进行结点之间的拼接：

```
private TreeNode prev = null;

public void flatten(TreeNode root) {
    if (root == null)
        return;
    flatten(root.right);
    flatten(root.left);
    root.right = prev;
    root.left = null;
    prev = root;
}
```

代码

```
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
```

```go
// 解法一 非递归
func flatten(root *TreeNode) {
  list, cur := []int{}, &TreeNode{}
  preorder(root, &list)
  cur = root
  for i := 1; i < len(list); i++ {
    cur.Left = nil
    cur.Right = &TreeNode{Val: list[i], Left: nil, Right: nil}
    cur = cur.Right
  }
  return
}

// 解法二 递归
func flatten1(root *TreeNode) {
  if root == nil || (root.Left == nil && root.Right == nil) {
    return
  }
  flatten(root.Left)
  flatten(root.Right)
  currRight := root.Right
  root.Right = root.Left
  root.Left = nil
  for root.Right != nil {
    root = root.Right
  }
  root.Right = currRight
}

// 解法三 递归
func flatten2(root *TreeNode) {
  if root == nil {
    return
  }
  flatten(root.Right)
  if root.Left == nil {
    return
  }
  flatten(root.Left)
  p := root.Left
  for p.Right != nil {
    p = p.Right
  }
  p.Right = root.Right
  root.Right = root.Left
  root.Left = nil
}
```

115. Distinct Subsequences

题目

Given two strings `s` and `t`, return *the number of distinct subsequences of* `s` *which equals* `t`.

A string's **subsequence** is a new string formed from the original string by deleting some (can be none) of the characters without disturbing the remaining characters' relative positions. (i.e., `"ACE"` is a subsequence of `"ABCDE"` while `"AEC"` is not).

It is guaranteed the answer fits on a 32-bit signed integer.

Example 1:

```
Input: s = "rabbbit", t = "rabbit"
Output: 3
Explanation:
As shown below, there are 3 ways you can generate "rabbit" from S.
rabbbitrabbbitrabbbit
```

Example 2:

```
Input: s = "babgbag", t = "bag"
Output: 5
Explanation:
As shown below, there are 5 ways you can generate "bag" from S.
babgbagbabgbagbabgbagbabgbagbabgbag
```

Constraints:

- `0 <= s.length, t.length <= 1000`
- `s` and `t` consist of English letters.

题目大意

给定一个字符串 s 和一个字符串 t ，计算在 s 的子序列中 t 出现的个数。字符串的一个 子序列 是指，通过删除一些（也可以不删除）字符且不干扰剩余字符相对位置所组成的新字符串。（例如，"ACE" 是 "ABCDE" 的一个子序列，而 "AEC" 不是）题目数据保证答案符合 32 位带符号整数范围。

解题思路

- 在字符串 `s` 中最多包含多少个字符串 `t`。这里面包含很多重叠子问题，所以尝试用动态规划解决这个问题。定义 `dp[i][j]` 代表 `s[i:]` 的子序列中 `t[j:]` 出现的个数。初始化先判断边界条件。当 `i = len(s)` 且 `0≤ j < len(t)` 的时候，`s[i:]` 为空字符串，`t[j:]` 不为空，所以 `dp[len(s)][j] = 0`。当 `j = len(t)` 且 `0 ≤ i < len(s)` 的时候，`t[j:]` 不为空字符串，空字符串是任何字符串的子序列。所以 `dp[i][n] = 1`。

- 当 `i < len(s)` 且 `j < len(t)` 的时候，如果 `s[i] == t[j]`，有 2 种匹配方式，第一种将 `s[i]` 与 `t[j]` 匹配，那么 `t[j+1:]` 匹配 `s[i+1:]` 的子序列，子序列数为 `dp[i+1][j+1]`；第二种将 `s[i]` 不与 `t[j]` 匹配，`t[j:]` 作为 `s[i+1:]` 的子序列，子序列数为 `dp[i+1][j]`。综合 2 种情况，当 `s[i] == t[j]` 时，`dp[i][j] = dp[i+1][j+1] + dp[i+1][j]`。
- 如果 `s[i] != t[j]`，此时 `t[j:]` 只能作为 `s[i+1:]` 的子序列，子序列数为 `dp[i+1][j]`。所以当 `s[i] != t[j]` 时，`dp[i][j] = dp[i+1][j]`。综上分析得：

  {{< katex display >}}
  dp[i][j] = \left{\begin{matrix}dp[i+1][j+1]+dp[i+1][j]&,s[i]=t[j]\ dp[i+1][j]&,s[i]!=t[j]\end{matrix}\right.
  {{< /katex >}}
- 最后是优化版本。写出上述代码以后，可以发现填表的过程是从右下角一直填到左上角。填表顺序是 从下往上一行一行的填。行内从右往左填。于是可以将这个二维数据压缩到一维。因为填充当前行只需要用到它的下一行信息即可，更进一步，用到的是下一行中右边元素的信息。于是可以每次更新该行时，先将旧的值存起来，计算更新该行的时候从右往左更新。这样做即可减少一维空间，将原来的二维数组压缩到一维数组。

代码

```go
package leetcode

// 解法一 压缩版 DP
func numDistinct(s string, t string) int {
  dp := make([]int, len(s)+1)
  for i, curT := range t {
    pre := 0
    for j, curS := range s {
      if i == 0 {
        pre = 1
      }
      newDP := dp[j+1]
      if curT == curS {
        dp[j+1] = dp[j] + pre
      } else {
        dp[j+1] = dp[j]
      }
      pre = newDP
    }
  }
  return dp[len(s)]
}

// 解法二 普通 DP
func numDistinct1(s, t string) int {
  m, n := len(s), len(t)
  if m < n {
    return 0
  }
  dp := make([][]int, m+1)
```

```
  for i := range dp {
    dp[i] = make([]int, n+1)
    dp[i][n] = 1
  }
  for i := m - 1; i >= 0; i-- {
    for j := n - 1; j >= 0; j-- {
      if s[i] == t[j] {
        dp[i][j] = dp[i+1][j+1] + dp[i+1][j]
      } else {
        dp[i][j] = dp[i+1][j]
      }
    }
  }
  return dp[0][0]
}
```

118. Pascal's Triangle

题目

Given a non-negative integer numRows, generate the first numRows of Pascal's triangle.

Note: In Pascal's triangle, each number is the sum of the two numbers directly above it.

Example:

```
Input: 5
Output:
[
     [1],
    [1,1],
   [1,2,1],
  [1,3,3,1],
 [1,4,6,4,1]
]
```

题目大意

给定一个非负整数 numRows，生成杨辉三角的前 numRows 行。在杨辉三角中，每个数是它左上方和右上方的数的和。

解题思路

- 给定一个 n，要求打印杨辉三角的前 n 行。
- 简单题。按照杨辉三角的生成规则循环打印即可。

代码

```go
package leetcode

func generate(numRows int) [][]int {
  result := [][]int{}
  for i := 0; i < numRows; i++ {
    row := []int{}
    for j := 0; j < i+1; j++ {
      if j == 0 || j == i {
        row = append(row, 1)
      } else if i > 1 {
        row = append(row, result[i-1][j-1]+result[i-1][j])
      }
    }
    result = append(result, row)
  }
  return result
}
```

119. Pascal's Triangle II

题目

Given an integer `rowIndex`, return the `rowIndexth` row of the Pascal's triangle.

Notice that the row index starts from **0**.

In Pascal's triangle, each number is the sum of the two numbers directly above it.

Follow up:

Could you optimize your algorithm to use only *O*(*k*) extra space?

Example 1:

```
Input: rowIndex = 3
Output: [1,3,3,1]
```

Example 2:

```
Input: rowIndex = 0
Output: [1]
```

Example 3:

```
Input: rowIndex = 1
Output: [1,1]
```

Constraints:

- `0 <= rowIndex <= 33`

题目大意

给定一个非负索引 k，其中 k ≤ 33，返回杨辉三角的第 k 行。

解题思路

- 题目中的三角是杨辉三角，每个数字是 `(a+b)^n` 二项式展开的系数。题目要求我们只能使用 O(k) 的空间。那么需要找到两两项直接的递推关系。由组合知识得知：

  {{< katex display >}}
  \begin{aligned}C*{n}^{m} &= \frac{n!}{m!(n-m)!} \C*{n}^{m-1} &= \frac{n!}{(m-1)!(n-m+1)!}\end{aligned}
  {{< /katex>}}

于是得到递推公式：

{{< katex display >}}
$C_{n}^{m} = C_{n}^{m-1} \times \frac{n-m+1}{m}$
{{< /katex>}}

利用这个递推公式即可以把空间复杂度优化到 O(k)

代码

```
package leetcode

func getRow(rowIndex int) []int {
  row := make([]int, rowIndex+1)
  row[0] = 1
  for i := 1; i <= rowIndex; i++ {
    row[i] = row[i-1] * (rowIndex - i + 1) / i
  }
  return row
}
```

120. Triangle

题目

Given a triangle, find the minimum path sum from top to bottom. Each step you may move to adjacent numbers on the row below.

For example, given the following triangle

```
[
     [2],
    [3,4],
   [6,5,7],
  [4,1,8,3]
]
```

The minimum path sum from top to bottom is `11` (i.e., **2** + **3** + **5** + **1** = 11).

Note:

Bonus point if you are able to do this using only *O*(*n*) extra space, where *n* is the total number of rows in the triangle.

题目大意

给定一个三角形，找出自顶向下的最小路径和。每一步只能移动到下一行中相邻的结点上。

解题思路

- 求出从三角形顶端到底端的最小和。要求最好用 O(n) 的时间复杂度。
- 这一题最优解是不用辅助空间，直接从下层往上层推。普通解法是用二维数组 DP，稍微优化的解法是一维数组 DP。解法如下：

代码

```
package leetcode

import (
  "math"
)

// 解法一 倒序 DP，无辅助空间
func minimumTotal(triangle [][]int) int {
  if triangle == nil {
    return 0
  }
  for row := len(triangle) - 2; row >= 0; row-- {
    for col := 0; col < len(triangle[row]); col++ {
      triangle[row][col] += min(triangle[row+1][col], triangle[row+1][col+1])
    }
  }
  return triangle[0][0]
}

// 解法二 正常 DP，空间复杂度 O(n)
func minimumTotal1(triangle [][]int) int {
  if len(triangle) == 0 {
    return 0
  }
  dp, minNum, index := make([]int, len(triangle[len(triangle)-1])), math.MaxInt64, 0
  for ; index < len(triangle[0]); index++ {
    dp[index] = triangle[0][index]
  }
  for i := 1; i < len(triangle); i++ {
    for j := len(triangle[i]) - 1; j >= 0; j-- {
      if j == 0 {
        // 最左边
        dp[j] += triangle[i][0]
      } else if j == len(triangle[i])-1 {
        // 最右边
        dp[j] += dp[j-1] + triangle[i][j]
      } else {
        // 中间
        dp[j] = min(dp[j-1]+triangle[i][j], dp[j]+triangle[i][j])
      }
    }
  }
```

```
  for i := 0; i < len(dp); i++ {
    if dp[i] < minNum {
      minNum = dp[i]
    }
  }
  return minNum
}
```

121. Best Time to Buy and Sell Stock

题目

Say you have an array for which the *i*th element is the price of a given stock on day *i*.

If you were only permitted to complete at most one transaction (i.e., buy one and sell one share of the stock), design an algorithm to find the maximum profit.

Note that you cannot sell a stock before you buy one.

Example 1:

```
Input: [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.
             Not 7-1 = 6, as selling price needs to be larger than buying price.
```

Example 2:

```
Input: [7,6,4,3,1]
Output: 0
Explanation: In this case, no transaction is done, i.e. max profit = 0.
```

题目大意

给定一个数组，它的第 i 个元素是一支给定股票第 i 天的价格。如果你最多只允许完成一笔交易（即买入和卖出一支股票），设计一个算法来计算你所能获取的最大利润。注意你不能在买入股票前卖出股票。

解题思路

- 题目要求找出股票中能赚的钱最多的差价
- 这一题也有多个解法，可以用 DP，也可以用单调栈

代码

```
package leetcode
```

```go
// 解法一 模拟 DP
func maxProfit(prices []int) int {
  if len(prices) < 1 {
    return 0
  }
  min, maxProfit := prices[0], 0
  for i := 1; i < len(prices); i++ {
    if prices[i]-min > maxProfit {
      maxProfit = prices[i] - min
    }
    if prices[i] < min {
      min = prices[i]
    }
  }
  return maxProfit
}

// 解法二 单调栈
func maxProfit1(prices []int) int {
  if len(prices) == 0 {
    return 0
  }
  stack, res := []int{prices[0]}, 0
  for i := 1; i < len(prices); i++ {
    if prices[i] > stack[len(stack)-1] {
      stack = append(stack, prices[i])
    } else {
      index := len(stack) - 1
      for ; index >= 0; index-- {
        if stack[index] < prices[i] {
          break
        }
      }
      stack = stack[:index+1]
      stack = append(stack, prices[i])
    }
    res = max(res, stack[len(stack)-1]-stack[0])
  }
  return res
}
```

122. Best Time to Buy and Sell Stock II

题目

Say you have an array for which the *i*th element is the price of a given stock on day *i*.

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (i.e., buy one and sell one share of the stock multiple times).

Note: You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again).

Example 1:

```
Input: [7,1,5,3,6,4]
Output: 7
Explanation: Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4.
             Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3
= 3.
```

Example 2:

```
Input: [1,2,3,4,5]
Output: 4
Explanation: Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4.
             Note that you cannot buy on day 1, buy on day 2 and sell them later, as
you are
             engaging multiple transactions at the same time. You must sell before
buying again.
```

Example 3:

```
Input: [7,6,4,3,1]
Output: 0
Explanation: In this case, no transaction is done, i.e. max profit = 0.
```

题目大意

给定一个数组，它的第 i 个元素是一支给定股票第 i 天的价格。设计一个算法来计算你所能获取的最大利润。你可以尽可能地完成更多的交易（多次买卖一支股票）。注意：你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。

解题思路

- 这一题是第 121 题的加强版。要求输出最大收益，这一题不止买卖一次，可以买卖多次，买卖不能在同一天内操作。
- 最大收益来源，必然是每次跌了就买入，涨到顶峰的时候就抛出。只要有涨峰就开始计算赚的钱，连续涨可以用两两相减累加来计算，两两相减累加，相当于涨到波峰的最大值减去谷底的值。这一点看通以后，题目非常简单。

代码

```go
package leetcode

func maxProfit122(prices []int) int {
  profit := 0
  for i := 0; i < len(prices)-1; i++ {
    if prices[i+1] > prices[i] {
      profit += prices[i+1] - prices[i]
    }
  }
  return profit
}
```

124. Binary Tree Maximum Path Sum

题目

Given a **non-empty** binary tree, find the maximum path sum.

For this problem, a path is defined as any sequence of nodes from some starting node to any node in the tree along the parent-child connections. The path must contain **at least one node** and does not need to go through the root.

Example 1:

```
Input: [1,2,3]

       1
      / \
     2   3

Output: 6
```

Example 2:

```
Input: [-10,9,20,null,null,15,7]

   -10
   / \
  9  20
    /  \
   15   7

Output: 42
```

题目大意

给定一个非空二叉树，返回其最大路径和。本题中，路径被定义为一条从树中任意节点出发，达到任意节点的序列。该路径至少包含一个节点，且不一定经过根节点。

解题思路

- 给出一个二叉树，要求找一条路径使得路径的和是最大的。
- 这一题思路比较简单，递归维护最大值即可。不过需要比较的对象比较多。`maxPathSum(root) = max(maxPathSum(root.Left), maxPathSum(root.Right), maxPathSumFrom(root.Left) (if>0) + maxPathSumFrom(root.Right) (if>0) + root.Val)`，其中，`maxPathSumFrom(root) = max(maxPathSumFrom(root.Left), maxPathSumFrom(root.Right)) + root.Val`

代码

```go
package leetcode

import "math"

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func maxPathSum(root *TreeNode) int {
  if root == nil {
    return 0
  }
  max := math.MinInt32
  getPathSum(root, &max)
  return max
}

func getPathSum(root *TreeNode, maxSum *int) int {
  if root == nil {
    return math.MinInt32
  }
  left := getPathSum(root.Left, maxSum)
  right := getPathSum(root.Right, maxSum)

  currMax := max(max(left+root.Val, right+root.Val), root.Val)
  *maxSum = max(*maxSum, max(currMax, left+right+root.Val))
  return currMax
}
```

125. Valid Palindrome

题目

Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

For example,

```
"A man, a plan, a canal: Panama" is a palindrome.
"race a car" is not a palindrome.
```

Note:

Have you consider that the string might be empty? This is a good question to ask during an interview.

For the purpose of this problem, we define empty string as valid palindrome.

题目大意

判断所给的字符串是否是有效的回文串。

解题思路

简单题，按照题意做即可。

代码

```go
package leetcode

import (
  "strings"
)

func isPalindrome(s string) bool {
  s = strings.ToLower(s)
  i, j := 0, len(s)-1
  for i < j {
    for i < j && !isChar(s[i]) {
      i++
    }
    for i < j && !isChar(s[j]) {
      j--
    }
    if s[i] != s[j] {
      return false
```

```
    }
    i++
    j--
  }
  return true
}

// 判断 c 是否是字符或者数字
func isChar(c byte) bool {
  if ('a' <= c && c <= 'z') || ('0' <= c && c <= '9') {
    return true
  }
  return false
}
```

126. Word Ladder II

题目

Given two words (*beginWord* and *endWord*), and a dictionary's word list, find all shortest transformation sequence(s) from *beginWord* to *endWord*, such that:

1. Only one letter can be changed at a time
2. Each transformed word must exist in the word list. Note that *beginWord* is *not* a transformed word.

Note:

- Return an empty list if there is no such transformation sequence.
- All words have the same length.
- All words contain only lowercase alphabetic characters.
- You may assume no duplicates in the word list.
- You may assume *beginWord* and *endWord* are non-empty and are not the same.

Example 1:

```
Input:
beginWord = "hit",
endWord = "cog",
wordList = ["hot","dot","dog","lot","log","cog"]

Output:
[
  ["hit","hot","dot","dog","cog"],
  ["hit","hot","lot","log","cog"]
]
```

Example 2:

```
Input:
beginWord = "hit"
endWord = "cog"
wordList = ["hot","dot","dog","lot","log"]

Output: []

Explanation: The endWord "cog" is not in wordList, therefore no possible
transformation.
```

题目大意

给定两个单词（beginWord 和 endWord）和一个字典 wordList，找出所有从 beginWord 到 endWord 的最短转换序列。转换需遵循如下规则：

1. 每次转换只能改变一个字母。
2. 转换过程中的中间单词必须是字典中的单词。

说明:

- 如果不存在这样的转换序列，返回一个空列表。
- 所有单词具有相同的长度。
- 所有单词只由小写字母组成。
- 字典中不存在重复的单词。
- 你可以假设 beginWord 和 endWord 是非空的，且二者不相同。

解题思路

- 这一题是第 127 题的加强版，除了找到路径的长度，还进一步要求输出所有路径。解题思路同第 127 题一样，也是用 BFS 遍历。
- 当前做法不是最优解，是否可以考虑双端 BFS 优化，或者迪杰斯塔拉算法?

代码

```go
package leetcode

func findLadders(beginWord string, endWord string, wordList []string) [][]string {
  result, wordMap := make([][]string, 0), make(map[string]bool)
  for _, w := range wordList {
    wordMap[w] = true
  }
  if !wordMap[endWord] {
    return result
  }
  // create a queue, track the path
  queue := make([][]string, 0)
```

```go
  queue = append(queue, []string{beginWord})
  // queueLen is used to track how many slices in queue are in the same level
  // if found a result, I still need to finish checking current level cause I need to
return all possible paths
  queueLen := 1
  // use to track strings that this level has visited
  // when queueLen == 0, remove levelMap keys in wordMap
  levelMap := make(map[string]bool)
  for len(queue) > 0 {
    path := queue[0]
    queue = queue[1:]
    lastWord := path[len(path)-1]
    for i := 0; i < len(lastWord); i++ {
      for c := 'a'; c <= 'z'; c++ {
        nextWord := lastWord[:i] + string(c) + lastWord[i+1:]
        if nextWord == endWord {
          path = append(path, endWord)
          result = append(result, path)
          continue
        }
        if wordMap[nextWord] {
          // different from word ladder, don't remove the word from wordMap immediately
          // same level could reuse the key.
          // delete from wordMap only when currently level is done.
          levelMap[nextWord] = true
          newPath := make([]string, len(path))
          copy(newPath, path)
          newPath = append(newPath, nextWord)
          queue = append(queue, newPath)
        }
      }
    }
    queueLen--
    // if queueLen is 0, means finish traversing current level. if result is not empty,
return result
    if queueLen == 0 {
      if len(result) > 0 {
        return result
      }
      for k := range levelMap {
        delete(wordMap, k)
      }
      // clear levelMap
      levelMap = make(map[string]bool)
      queueLen = len(queue)
    }
  }
  return result
}
```

127. Word Ladder

题目

Given two words (*beginWord* and *endWord*), and a dictionary's word list, find the length of shortest transformation sequence from *beginWord* to *endWord*, such that:

1. Only one letter can be changed at a time.
2. Each transformed word must exist in the word list. Note that *beginWord* is *not* a transformed word.

Note:

- Return 0 if there is no such transformation sequence.
- All words have the same length.
- All words contain only lowercase alphabetic characters.
- You may assume no duplicates in the word list.
- You may assume *beginWord* and *endWord* are non-empty and are not the same.

Example 1:

```
Input:
beginWord = "hit",
endWord = "cog",
wordList = ["hot","dot","dog","lot","log","cog"]

Output: 5

Explanation: As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" ->
"cog",
return its length 5.
```

Example 2:

```
Input:
beginWord = "hit"
endWord = "cog"
wordList = ["hot","dot","dog","lot","log"]

Output: 0

Explanation: The endWord "cog" is not in wordList, therefore no possible
transformation.
```

题目大意

给定两个单词（beginWord 和 endWord）和一个字典，找到从 beginWord 到 endWord 的最短转换序列的长度。转换需遵循如下规则：

1. 每次转换只能改变一个字母。
2. 转换过程中的中间单词必须是字典中的单词。

说明:

- 如果不存在这样的转换序列，返回 0。
- 所有单词具有相同的长度。
- 所有单词只由小写字母组成。
- 字典中不存在重复的单词。
- 你可以假设 beginWord 和 endWord 是非空的，且二者不相同。

解题思路

- 这一题要求输出从 `beginWord` 变换到 `endWord` 最短变换次数。可以用 BFS，从 `beginWord` 开始变换，把该单词的每个字母都用 `'a'~'z'` 变换一次，生成的数组到 `wordList` 中查找，这里用 Map 来记录查找。找得到就入队列，找不到就输出 0 。入队以后按照 BFS 的算法依次遍历完，当所有单词都 `len(queue)<=0` 出队以后，整个程序结束。
- 这一题题目中虽然说了要求找到一条最短的路径，但是实际上最短的路径的寻找方法已经告诉你了：
  1. 每次只变换一个字母
  2. 每次变换都必须在 `wordList` 中
     所以不需要单独考虑何种方式是最短的。

代码

```
package leetcode

func ladderLength(beginWord string, endWord string, wordList []string) int {
  wordMap, que, depth := getWordMap(wordList, beginWord), []string{beginWord}, 0
  for len(que) > 0 {
    depth++
    qlen := len(que)
    for i := 0; i < qlen; i++ {
      word := que[0]
      que = que[1:]
      candidates := getCandidates(word)
      for _, candidate := range candidates {
        if _, ok := wordMap[candidate]; ok {
          if candidate == endWord {
            return depth + 1
          }
          delete(wordMap, candidate)
          que = append(que, candidate)
        }
      }
```

```go
    }
  }
  return 0
}

func getWordMap(wordList []string, beginWord string) map[string]int {
  wordMap := make(map[string]int)
  for i, word := range wordList {
    if _, ok := wordMap[word]; !ok {
      if word != beginWord {
        wordMap[word] = i
      }
    }
  }
  return wordMap
}

func getCandidates(word string) []string {
  var res []string
  for i := 0; i < 26; i++ {
    for j := 0; j < len(word); j++ {
      if word[j] != byte(int('a')+i) {
        res = append(res, word[:j]+string(int('a')+i)+word[j+1:])
      }
    }
  }
  return res
}
```

128. Longest Consecutive Sequence

题目

Given an unsorted array of integers, find the length of the longest consecutive elements sequence.

Your algorithm should run in O(*n*) complexity.

Example:

```
Input: [100, 4, 200, 1, 3, 2]
Output: 4
Explanation: The longest consecutive elements sequence is [1, 2, 3, 4]. Therefore its
length is 4.
```

题目大意

给定一个未排序的整数数组，找出最长连续序列的长度。要求算法的时间复杂度为 O(n)。

解题思路

- 给出一个数组，要求找出最长连续序列，输出这个最长的长度。要求时间复杂度为 `O(n)`。
- 这一题可以先用暴力解决解决，代码见解法三。思路是把每个数都存在 `map` 中，先删去 `map` 中没有前一个数 `nums[i]-1` 也没有后一个数 `nums[i]+1` 的数 `nums[i]`，这种数前后都不连续。然后在 `map` 中找到前一个数 `nums[i]-1` 不存在，但是后一个数 `nums[i]+1` 存在的数，这种数是连续序列的起点，那么不断的往后搜，直到序列“断”了。最后输出最长序列的长度。
- 这一题最优的解法是解法一，针对每一个 `map` 中不存在的数 `n`，插入进去都做 2 件事情。第一件事，先查看 `n - 1` 和 `n + 1` 是否都存在于 `map` 中，如果都存在，代表存在连续的序列，那么就更新 `left`，`right` 边界。那么 `n` 对应的这个小的子连续序列长度为 `sum = left + right + 1`。第二件事就是更新 `left` 和 `right` 左右边界对应的 `length = sum`。
- 这一题还可以用并查集解决，见解法二。利用每个数在 `nums` 中的下标，把下标和下标进行 `union()`，具体做法是看前一个数 `nums[i]-1` 和后一个数 `nums[i]+1` 在 `map` 中是否存在，如果存在就 `union()`，最终输出整个并查集中包含最多元素的那个集合的元素总数。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/template"
)

// 解法一 map，时间复杂度 O(n)
func longestConsecutive(nums []int) int {
  res, numMap := 0, map[int]int{}
  for _, num := range nums {
    if numMap[num] == 0 {
      left, right, sum := 0, 0, 0
      if numMap[num-1] > 0 {
        left = numMap[num-1]
      } else {
        left = 0
      }
      if numMap[num+1] > 0 {
        right = numMap[num+1]
      } else {
        right = 0
      }
      // sum: length of the sequence n is in
      sum = left + right + 1
      numMap[num] = sum
      // keep track of the max length
      res = max(res, sum)
```

```go
      // extend the length to the boundary(s) of the sequence
      // will do nothing if n has no neighbors
      numMap[num-left] = sum
      numMap[num+right] = sum
    } else {
      continue
    }
  }
  return res
}

// 解法二 并查集
func longestConsecutive1(nums []int) int {
  if len(nums) == 0 {
    return 0
  }
  numMap, countMap, lcs, uf := map[int]int{}, map[int]int{}, 0, template.UnionFind{}
  uf.Init(len(nums))
  for i := 0; i < len(nums); i++ {
    countMap[i] = 1
  }
  for i := 0; i < len(nums); i++ {
    if _, ok := numMap[nums[i]]; ok {
      continue
    }
    numMap[nums[i]] = i
    if _, ok := numMap[nums[i]+1]; ok {
      uf.Union(i, numMap[nums[i]+1])
    }
    if _, ok := numMap[nums[i]-1]; ok {
      uf.Union(i, numMap[nums[i]-1])
    }
  }
  for key := range countMap {
    parent := uf.Find(key)
    if parent != key {
      countMap[parent]++
    }
    if countMap[parent] > lcs {
      lcs = countMap[parent]
    }
  }
  return lcs
}

// 解法三 暴力解法，时间复杂度 O(n^2)
func longestConsecutive2(nums []int) int {
  if len(nums) == 0 {
    return 0
```

```
    }
    numMap, length, tmp, lcs := map[int]bool{}, 0, 0, 0
    for i := 0; i < len(nums); i++ {
      numMap[nums[i]] = true
    }
    for key := range numMap {
      if !numMap[key-1] && !numMap[key+1] {
        delete(numMap, key)
      }
    }
    if len(numMap) == 0 {
      return 1
    }
    for key := range numMap {
      if !numMap[key-1] && numMap[key+1] {
        length, tmp = 1, key+1
        for numMap[tmp] {
          length++
          tmp++
        }
        lcs = max(lcs, length)
      }
    }
    return max(lcs, length)
  }
```

129. Sum Root to Leaf Numbers

题目

Given a binary tree containing digits from `0-9` only, each root-to-leaf path could represent a number.

An example is the root-to-leaf path `1->2->3` which represents the number `123`.

Find the total sum of all root-to-leaf numbers.

Note: A leaf is a node with no children.

Example:

```
Input: [1,2,3]
    1
   / \
  2   3
Output: 25
Explanation:
The root-to-leaf path 1->2 represents the number 12.
The root-to-leaf path 1->3 represents the number 13.
Therefore, sum = 12 + 13 = 25.
```

Example 2:

```
Input: [4,9,0,5,1]
    4
   / \
  9   0
 / \
5   1
Output: 1026
Explanation:
The root-to-leaf path 4->9->5 represents the number 495.
The root-to-leaf path 4->9->1 represents the number 491.
The root-to-leaf path 4->0 represents the number 40.
Therefore, sum = 495 + 491 + 40 = 1026.
```

题目大意

给定一个二叉树，它的每个结点都存放一个 0-9 的数字，每条从根到叶子节点的路径都代表一个数字。例如，从根到叶子节点路径 1->2->3 代表数字 123。计算从根到叶子节点生成的所有数字之和。说明: 叶子节点是指没有子节点的节点。

解题思路

- 这一题是第 257 题的变形题，第 257 题要求输出每条从根节点到叶子节点的路径，这一题变成了把每一个从根节点到叶子节点的数字都串联起来，再累加每条路径，求出最后的总和。实际做题思路基本没变。运用前序遍历的思想，当从根节点出发一直加到叶子节点，每个叶子节点汇总一次。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/structures"
)

// TreeNode define
```

```go
type TreeNode = structures.TreeNode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

func sumNumbers(root *TreeNode) int {
  res := 0
  dfs(root,0,&res)
  return res
}

func dfs(root *TreeNode,sum int,res *int)  {
  if root == nil{
    return
  }
  sum = sum*10 + root.Val
  if root.Left == nil && root.Right == nil{
    *res += sum
    return
  }
  dfs(root.Left,sum,res)
  dfs(root.Right,sum,res)
}
```

130. Surrounded Regions

题目

Given a 2D board containing `'X'` and `'O'` (**the letter O**), capture all regions surrounded by `'X'`.

A region is captured by flipping all `'O'`s into `'X'`s in that surrounded region.

Example:

```
X X X X
X O O X
X X O X
X O X X
```

After running your function, the board should be:

```
X X X X
X X X X
X X X X
X O X X
```

Explanation:

Surrounded regions shouldn't be on the border, which means that any `'O'` on the border of the board are not flipped to `'X'`. Any `'O'` that is not on the border and it is not connected to an `'O'` on the border will be flipped to `'X'`. Two cells are connected if they are adjacent cells connected horizontally or vertically.

题目大意

给定一个二维的矩阵，包含 'X' 和 'O'（字母 O）。找到所有被 'X' 围绕的区域，并将这些区域里所有的 'O' 用 'X' 填充。被围绕的区间不会存在于边界上，换句话说，任何边界上的 'O' 都不会被填充为 'X'。 任何不在边界上，或不与边界上的 'O' 相连的 'O' 最终都会被填充为 'X'。如果两个元素在水平或垂直方向相邻，则称它们是“相连”的。

解题思路

- 给出一张二维地图，要求把地图上非边缘上的 'O' 都用 'X' 覆盖掉。
- 这一题有多种解法。第一种解法是并查集。先将边缘上的 'O' 全部都和一个特殊的点进行 `union()` 。然后再把地图中间的 'O' 都进行 `union()`，最后把和特殊点不是同一个集合的点都标记成 'X'。第二种解法是 DFS 或者 BFS，可以先将边缘上的 'O' 先标记成另外一个字符，然后在递归遍历过程中，把剩下的 'O' 都标记成 'X'。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/template"
)

// 解法一 并查集
func solve(board [][]byte) {
  if len(board) == 0 {
    return
  }
  m, n := len(board[0]), len(board)
  uf := template.UnionFind{}
  uf.Init(n*m + 1) // 特意多一个特殊点用来标记

  for i := 0; i < n; i++ {
    for j := 0; j < m; j++ {
      if (i == 0 || i == n-1 || j == 0 || j == m-1) && board[i][j] == 'O' { //棋盘边缘上的 'O' 点
```

```go
        uf.Union(i*m+j, n*m)
      } else if board[i][j] == 'O' { //棋盘非边缘上的内部的 'O' 点
        if board[i-1][j] == 'O' {
          uf.Union(i*m+j, (i-1)*m+j)
        }
        if board[i+1][j] == 'O' {
          uf.Union(i*m+j, (i+1)*m+j)
        }
        if board[i][j-1] == 'O' {
          uf.Union(i*m+j, i*m+j-1)
        }
        if board[i][j+1] == 'O' {
          uf.Union(i*m+j, i*m+j+1)
        }

      }
    }
  }
  for i := 0; i < n; i++ {
    for j := 0; j < m; j++ {
      if uf.Find(i*m+j) != uf.Find(n*m) {
        board[i][j] = 'X'
      }
    }
  }
}

// 解法二 DFS
func solve1(board [][]byte) {
  for i := range board {
    for j := range board[i] {
      if i == 0 || i == len(board)-1 || j == 0 || j == len(board[i])-1 {
        if board[i][j] == 'O' {
          dfs130(i, j, board)
        }
      }
    }
  }

  for i := range board {
    for j := range board[i] {
      if board[i][j] == '*' {
        board[i][j] = 'O'
      } else if board[i][j] == 'O' {
        board[i][j] = 'X'
      }
    }
  }
}
```

```go
func dfs130(i, j int, board [][]byte) {
  if i < 0 || i > len(board)-1 || j < 0 || j > len(board[i])-1 {
    return
  }
  if board[i][j] == 'O' {
    board[i][j] = '*'
    for k := 0; k < 4; k++ {
      dfs130(i+dir[k][0], j+dir[k][1], board)
    }
  }
}
```

131. Palindrome Partitioning

题目

Given a string *s*, partition *s* such that every substring of the partition is a palindrome.

Return all possible palindrome partitioning of *s*.

Example:

```
Input: "aab"
Output:
[
  ["aa","b"],
  ["a","a","b"]
]
```

题目大意

给定一个字符串 s，将 s 分割成一些子串，使每个子串都是回文串。返回 s 所有可能的分割方案。

解题思路

- 要求输出一个字符串可以被拆成回文串的所有解，DFS 递归求解即可。

代码

```go
package leetcode

// 解法一
func partition131(s string) [][]string {
  if s == "" {
    return [][]string{}
```

```go
  }
  res, pal := [][]string{}, []string{}
  findPalindrome(s, 0, "", true, pal, &res)
  return res
}

func findPalindrome(str string, index int, s string, isPal bool, pal []string, res *[]
[]string) {
  if index == len(str) {
    if isPal {
      tmp := make([]string, len(pal))
      copy(tmp, pal)
      *res = append(*res, tmp)
    }
    return
  }
  if index == 0 {
    s = string(str[index])
    pal = append(pal, s)
    findPalindrome(str, index+1, s, isPal && isPalindrome131(s), pal, res)
  } else {
    temp := pal[len(pal)-1]
    s = pal[len(pal)-1] + string(str[index])
    pal[len(pal)-1] = s
    findPalindrome(str, index+1, s, isPalindrome131(s), pal, res)
    pal[len(pal)-1] = temp
    if isPalindrome131(temp) {
      pal = append(pal, string(str[index]))
      findPalindrome(str, index+1, temp, isPal && isPalindrome131(temp), pal, res)
      pal = pal[:len(pal)-1]

    }
  }
  return
}

func isPalindrome131(s string) bool {
  slen := len(s)
  for i, j := 0, slen-1; i < j; i, j = i+1, j-1 {
    if s[i] != s[j] {
      return false
    }
  }
  return true
}

// 解法二
func partition131_1(s string) [][]string {
  result := [][]string{}
```

```go
  size := len(s)
  if size == 0 {
    return result
  }
  current := make([]string, 0, size)
  dfs131(s, 0, current, &result)
  return result
}

func dfs131(s string, idx int, cur []string, result *[][]string) {
  start, end := idx, len(s)
  if start == end {
    temp := make([]string, len(cur))
    copy(temp, cur)
    *result = append(*result, temp)
    return
  }
  for i := start; i < end; i++ {
    if isPal(s, start, i) {
      dfs131(s, i+1, append(cur, s[start:i+1]), result)
    }
  }
}

func isPal(str string, s, e int) bool {
  for s < e {
    if str[s] != str[e] {
      return false
    }
    s++
    e--
  }
  return true
}
```

136. Single Number

题目

Given a **non-empty** array of integers, every element appears *twice* except for one. Find that single one.

Note:

Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

Example 1:

```
Input: [2,2,1]
Output: 1
```

Example 2:

```
Input: [4,1,2,1,2]
Output: 4
```

题目大意

给定一个非空整数数组，除了某个元素只出现一次以外，其余每个元素均出现两次。找出那个只出现了一次的元素。要求算法时间复杂度是线性的，并且不使用额外的辅助空间。

解题思路

- 题目要求不能使用辅助空间，并且时间复杂度只能是线性的。
- 题目为什么要强调有一个数字出现一次，其他的出现两次？我们想到了异或运算的性质：任何一个数字异或它自己都等于0。也就是说，如果我们从头到尾依次异或数组中的每一个数字，那么最终的结果刚好是那个只出现依次的数字，因为那些出现两次的数字全部在异或中抵消掉了。于是最终做法是从头到尾依次异或数组中的每一个数字，那么最终得到的结果就是两个只出现一次的数字的异或结果。因为其他数字都出现了两次，在异或中全部抵消掉了。**利用的性质是 x^x = 0**。

代码

```go
package leetcode

func singleNumber(nums []int) int {
  result := 0
  for i := 0; i < len(nums); i++ {
    result ^= nums[i]
  }
  return result
}
```

137. Single Number II

题目

Given a **non-empty** array of integers, every element appears *three* times except for one, which appears exactly once. Find that single one.

Note:

Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

Example 1:

```
Input: [2,2,3,2]
Output: 3
```

Example 2:

```
Input: [0,1,0,1,0,1,99]
Output: 99
```

题目大意

给定一个非空整数数组，除了某个元素只出现一次以外，其余每个元素均出现了三次。找出那个只出现了一次的元素。要求算法时间复杂度是线性的，并且不使用额外的辅助空间。

解题思路

- 这一题是第 136 题的加强版。这类题也可以扩展，在数组中每个元素都出现 5 次，找出只出现 1 次的数。
- 本题中要求找出只出现 1 次的数，出现 3 次的数都要被消除。第 136 题是消除出现 2 次的数。这一题也会相当相同的解法，出现 3 次的数也要被消除。定义状态，00、10、01，这 3 个状态。当一个数出现 3 次，那么它每个位置上的 1 出现的次数肯定是 3 的倍数，所以当 1 出现 3 次以后，就归零清除。如何能做到这点呢？仿造`三进制(00，01，10)`就可以做到。
- 变量 ones 中记录遍历中每个位上出现 1 的个数。将它与 A[i] 进行异或，目的是：
  - 每位上两者都是 1 的，表示历史统计结果 ones 出现1次、A[i]中又出现 1 次，则是出现 2 次，需要进位到 twos 变量中。
  - 每位上两者分别为 0、1 的，加入到 ones 统计结果中。
  - 最后还要 & ^twos ，是为了能做到三进制，出现 3 次就清零。例如 ones = x，那么 twos = 0，当 twos = x，那么 ones = 0；
- 变量 twos 中记录遍历中每个位上出现 1 ，2次 的个数。与 A[i] 进行异或的目的和上述描述相同，不再赘述。

> 在 golang 中，&^ 表示 AND NOT 的意思。这里的 ^ 作为一元操作符，表示按位取反 (^0001 0100 = 1110 1011)，X &^ Y 的意思是将 X 中与 Y 相异的位保留，相同的位清零。

> 在 golang 中没有 Java 中的 ~ 位操作运算符，Java 中的 ~ 运算符代表按位取反。这个操作就想当于 golang 中的 ^ 运算符当做一元运算符使用的效果。

(twos,ones)	xi	(twos'',ones')	ones'
00	0	00	0
00	1	01	1
01	0	01	1
01	1	10	0
10	0	10	0
10	1	00	0

- 第一步，先将 ones -> ones'。通过观察可以看出 ones = (ones ^ nums[i]) & ^twos

(twos,ones')	xi	twos'
00	0	0
01	1	0
01	0	0
00	1	1
10	0	1
10	1	0

- 第二步，再将 twos -> twos'。这一步需要用到前一步的 ones。通过观察可以看出 twos = (twos ^ nums[i]) & ^ones。

这一题还可以继续扩展，在数组中每个元素都出现 5 次，找出只出现 1 次的数。那该怎么做呢？思路还是一样的，模拟一个五进制，5 次就会消除。代码如下：

```
// 解法一
func singleNumberIII(nums []int) int {
  na, nb, nc := 0, 0, 0
  for i := 0; i < len(nums); i++ {
    nb = nb ^ (nums[i] & na)
    na = (na ^ nums[i]) & ^nc
    nc = nc ^ (nums[i] & ^na & ^nb)
  }
  return na & ^nb & ^nc
}

// 解法二
func singleNumberIIII(nums []int) int {
  twos, threes, ones := 0xffffffff, 0xffffffff, 0
```

```
  for i := 0; i < len(nums); i++ {
    threes = threes ^ (nums[i] & twos)
    twos = (twos ^ nums[i]) & ^ones
    ones = ones ^ (nums[i] & ^twos & ^threes)
  }
  return ones
}
```

代码

```go
package leetcode

func singleNumberII(nums []int) int {
  ones, twos := 0, 0
  for i := 0; i < len(nums); i++ {
    ones = (ones ^ nums[i]) & ^twos
    twos = (twos ^ nums[i]) & ^ones
  }
  return ones
}

// 以下是拓展题
// 在数组中每个元素都出现 5 次，找出只出现 1 次的数。

// 解法一
func singleNumberIIIII(nums []int) int {
  na, nb, nc := 0, 0, 0
  for i := 0; i < len(nums); i++ {
    nb = nb ^ (nums[i] & na)
    na = (na ^ nums[i]) & ^nc
    nc = nc ^ (nums[i] & ^na & ^nb)
  }
  return na & ^nb & ^nc
}

// 解法二
func singleNumberIIIII1(nums []int) int {
  twos, threes, ones := 0xffffffff, 0xffffffff, 0
  for i := 0; i < len(nums); i++ {
    threes = threes ^ (nums[i] & twos)
    twos = (twos ^ nums[i]) & ^ones
    ones = ones ^ (nums[i] & ^twos & ^threes)
  }
  return ones
}
```

138. Copy List with Random Pointer

题目

A linked list is given such that each node contains an additional random pointer which could point to any node in the list or null.

Return a **deep copy** of the list.

The Linked List is represented in the input/output as a list of `n` nodes. Each node is represented as a pair of `[val, random_index]` where:

- `val`: an integer representing `Node.val`
- `random_index`: the index of the node (range from `0` to `n-1`) where random pointer points to, or `null` if it does not point to any node.

Example 1:

```
Input: head = [[7,null],[13,0],[11,4],[10,2],[1,0]]
Output: [[7,null],[13,0],[11,4],[10,2],[1,0]]
```

Example 2:

```
Input: head = [[1,1],[2,1]]
Output: [[1,1],[2,1]]
```

Example 3:

```
Input: head = [[3,null],[3,0],[3,null]]
Output: [[3,null],[3,0],[3,null]]
```

Example 4:

```
Input: head = []
Output: []
Explanation: Given linked list is empty (null pointer), so return null.
```

Constraints:

- `10000 <= Node.val <= 10000`
- `Node.random` is null or pointing to a node in the linked list.
- The number of nodes will not exceed 1000.

题目大意

给定一个链表，每个节点包含一个额外增加的随机指针，该指针可以指向链表中的任何节点或空节点。要求返回这个链表的 深拷贝。

我们用一个由 n 个节点组成的链表来表示输入/输出中的链表。每个节点用一个 [val, random_index] 表示：

- val：一个表示 Node.val 的整数。
- random_index：随机指针指向的节点索引（范围从 0 到 n-1）；如果不指向任何节点，则为 null 。

解题思路

- 这道题严格意义上是数据结构题，根据给定的数据结构，对它进行深拷贝。
- 先将每个节点都复制一份，放在它的 next 节点中。如此穿插的复制一份链表。

再将穿插版的链表的 random 指针指向正确的位置。

再将穿插版的链表的 next 指针指向正确的位置。最后分开这交织在一起的两个链表的头节点，即可分开 2 个链表。

代码

```go
package leetcode

// Node define
type Node struct {
  Val    int
  Next   *Node
  Random *Node
}

func copyRandomList(head *Node) *Node {
  if head == nil {
    return nil
  }
  tempHead := copyNodeToLinkedList(head)
  return splitLinkedList(tempHead)
}

func splitLinkedList(head *Node) *Node {
  cur := head
  head = head.Next
  for cur != nil && cur.Next != nil {
    cur.Next, cur = cur.Next.Next, cur.Next
  }
  return head
}

func copyNodeToLinkedList(head *Node) *Node {
  cur := head
  for cur != nil {
    node := &Node{
      Val:  cur.Val,
      Next: cur.Next,
    }
    cur.Next, cur = node, cur.Next
  }
  cur = head
  for cur != nil {
    if cur.Random != nil {
      cur.Next.Random = cur.Random.Next
    }
    cur = cur.Next.Next
  }
  return head
}
```

141. Linked List Cycle

题目

Given a linked list, determine if it has a cycle in it.

Follow up:
Can you solve it without using extra space?

题目大意

判断链表是否有环，不能使用额外的空间。

解题思路

给 2 个指针，一个指针是另外一个指针的下一个指针。快指针一次走 2 格，慢指针一次走 1 格。如果存在环，那么前一个指针一定会经过若干圈之后追上慢的指针。

代码

```go
package leetcode

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */

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 fast == slow {
      return true
    }
  }
  return false
}
```

142. Linked List Cycle II

题目

Given a linked list, return the node where the cycle begins. If there is no cycle, return null.

To represent a cycle in the given linked list, we use an integer pos which represents the position (0-indexed) in the linked list where tail connects to. If pos is -1, then there is no cycle in the linked list.

Note: 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.
```

题目大意

判断链表是否有环，不能使用额外的空间。如果有环，输出环的起点指针，如果没有环，则输出空。

解题思路

这道题是第 141 题的加强版。在判断是否有环的基础上，还需要输出环的第一个点。

分析一下判断环的原理。fast 指针一次都 2 步，slow 指针一次走 1 步。令链表 head 到环的一个点需要 x1 步，从环的第一个点到相遇点需要 x2 步，从环中相遇点回到环的第一个点需要 x3 步。那么环的总长度是 x2 + x3 步。

fast 和 slow 会相遇，说明他们走的时间是相同的，可以知道他们走的路程有以下的关系：

```
fast 的 t = (x1 + x2 + x3 + x2) / 2
slow 的 t = (x1 + x2) / 1

x1 + x2 + x3 + x2 = 2 * (x1 + x2)

所以 x1 = x3
```

所以 2 个指针相遇以后，如果 slow 继续往前走，fast 指针回到起点 head，两者都每次走一步，那么必定会在环的起点相遇，相遇以后输出这个点即是结果。

代码

```go
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */
func detectCycle(head *ListNode) *ListNode {
  if head == nil || head.Next == nil {
    return nil
  }
  isCycle, slow := hasCycle142(head)
  if !isCycle {
    return nil
  }
  fast := head
  for fast != slow {
    fast = fast.Next
    slow = slow.Next
  }
  return fast
}

func hasCycle142(head *ListNode) (bool, *ListNode) {
  fast := head
  slow := head
  for slow != nil && fast != nil && fast.Next != nil {
    fast = fast.Next.Next
    slow = slow.Next
    if fast == slow {
      return true, slow
```

```
    }
  }
  return false, nil
}
```

143. Reorder List

题目

Given a singly linked list L: L0→L1→…→Ln-1→Ln,
reorder it to: L0→Ln→L1→Ln-1→L2→Ln-2→…

You may not modify the values in the list's nodes, only nodes itself may be changed.

Example 1:

```
Given 1->2->3->4, reorder it to 1->4->2->3.
```

Example 2:

```
Given 1->2->3->4->5, reorder it to 1->5->2->4->3.
```

题目大意

按照指定规则重新排序链表：第一个元素和最后一个元素排列在一起，接着第二个元素和倒数第二个元素排在一起，接着第三个元素和倒数第三个元素排在一起。

解题思路

最近简单的方法是先把链表存储到数组里，然后找到链表中间的结点，按照规则拼接即可。这样时间复杂度是 O(n)，空间复杂度是 O(n)。

更好的做法是结合之前几道题的操作：链表逆序，找中间结点。

先找到链表的中间结点，然后利用逆序区间的操作，如 第 92 题 里的 reverseBetween() 操作，只不过这里的反转区间是从中点一直到末尾。最后利用 2 个指针，一个指向头结点，一个指向中间结点，开始拼接最终的结果。这种做法的时间复杂度是 O(n)，空间复杂度是 O(1)。

代码

```
package leetcode
```

```go
/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */

// 解法一 单链表
func reorderList(head *ListNode) *ListNode {
  if head == nil || head.Next == nil {
    return head
  }

  // 寻找中间结点
  p1 := head
  p2 := head
  for p2.Next != nil && p2.Next.Next != nil {
    p1 = p1.Next
    p2 = p2.Next.Next
  }

  // 反转链表后半部分  1->2->3->4->5->6 to 1->2->3->6->5->4
  preMiddle := p1
  preCurrent := p1.Next
  for preCurrent.Next != nil {
    current := preCurrent.Next
    preCurrent.Next = current.Next
    current.Next = preMiddle.Next
    preMiddle.Next = current
  }

  // 重新拼接链表  1->2->3->6->5->4 to 1->6->2->5->3->4
  p1 = head
  p2 = preMiddle.Next
  for p1 != preMiddle {
    preMiddle.Next = p2.Next
    p2.Next = p1.Next
    p1.Next = p2
    p1 = p2.Next
    p2 = preMiddle.Next
  }
  return head
}

// 解法二 数组
func reorderList1(head *ListNode) *ListNode {
  array := listToArray(head)
  length := len(array)
```

```go
  if length == 0 {
    return head
  }
  cur := head
  last := head
  for i := 0; i < len(array)/2; i++ {
    tmp := &ListNode{Val: array[length-1-i], Next: cur.Next}
    cur.Next = tmp
    cur = tmp.Next
    last = tmp
  }
  if length%2 == 0 {
    last.Next = nil
  } else {
    cur.Next = nil
  }
  return head
}

func listToArray(head *ListNode) []int {
  array := []int{}
  if head == nil {
    return array
  }
  cur := head
  for cur != nil {
    array = append(array, cur.Val)
    cur = cur.Next
  }
  return array
}
```

144. Binary Tree Preorder Traversal

题目

Given a binary tree, return the preorder traversal of its nodes' values.

Example:

```
Input: [1,null,2,3]
   1
    \
     2
    /
   3

Output: [1,2,3]
```

Follow up: Recursive solution is trivial, could you do it iteratively?

题目大意

先根遍历一颗树。

解题思路

两种递归的实现方法，见代码。

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

// 解法一 递归
func preorderTraversal(root *TreeNode) []int {
  res := []int{}
  if root != nil {
    res = append(res, root.Val)
    tmp := preorderTraversal(root.Left)
    for _, t := range tmp {
      res = append(res, t)
```

```go
    }
    tmp = preorderTraversal(root.Right)
    for _, t := range tmp {
      res = append(res, t)
    }
  }
  return res
}

// 解法二 递归
func preorderTraversal1(root *TreeNode) []int {
  var result []int
  preorder(root, &result)
  return result
}

func preorder(root *TreeNode, output *[]int) {
  if root != nil {
    *output = append(*output, root.Val)
    preorder(root.Left, output)
    preorder(root.Right, output)
  }
}

// 解法三 非递归，用栈模拟递归过程
func preorderTraversal2(root *TreeNode) []int {
  if root == nil {
    return []int{}
  }
  stack, res := []*TreeNode{}, []int{}
  stack = append(stack, root)
  for len(stack) != 0 {
    node := stack[len(stack)-1]
    stack = stack[:len(stack)-1]
    if node != nil {
      res = append(res, node.Val)
    }
    if node.Right != nil {
      stack = append(stack, node.Right)
    }
    if node.Left != nil {
      stack = append(stack, node.Left)
    }
  }
  return res
}
```

145. Binary Tree Postorder Traversal

题目

Given a binary tree, return the postorder traversal of its nodes' values.

Example:

```
Input: [1,null,2,3]
   1
    \
     2
    /
   3

Output: [3,2,1]
```

Follow up: Recursive solution is trivial, could you do it iteratively?

题目大意

后根遍历一颗树。

解题思路

递归的实现方法，见代码。

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
```

```go
func postorderTraversal(root *TreeNode) []int {
  var result []int
  postorder(root, &result)
  return result
}

func postorder(root *TreeNode, output *[]int) {
  if root != nil {
    postorder(root.Left, output)
    postorder(root.Right, output)
    *output = append(*output, root.Val)
  }
}
```

146. LRU Cache

题目

Design a data structure that follows the constraints of a **Least Recently Used (LRU) cache**.

Implement the `LRUCache` class:

- `LRUCache(int capacity)` Initialize the LRU cache with **positive** size `capacity`.
- `int get(int key)` Return the value of the `key` if the key exists, otherwise return `1`.
- `void put(int key, int value)` Update the value of the `key` if the `key` exists. Otherwise, add the `key-value` pair to the cache. If the number of keys exceeds the `capacity` from this operation, **evict** the least recently used key.

Follow up:Could you do `get` and `put` in `O(1)` time complexity?

Example 1:

```
Input
["LRUCache", "put", "put", "get", "put", "get", "put", "get", "get", "get"]
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]
Output
[null, null, null, 1, null, -1, null, -1, 3, 4]

Explanation
LRUCache lRUCache = new LRUCache(2);
lRUCache.put(1, 1); // cache is {1=1}
lRUCache.put(2, 2); // cache is {1=1, 2=2}
lRUCache.get(1);    // return 1
lRUCache.put(3, 3); // LRU key was 2, evicts key 2, cache is {1=1, 3=3}
lRUCache.get(2);    // returns -1 (not found)
lRUCache.put(4, 4); // LRU key was 1, evicts key 1, cache is {4=4, 3=3}
lRUCache.get(1);    // return -1 (not found)
lRUCache.get(3);    // return 3
```

```
lRUCache.get(4);    // return 4
```

Constraints:

- `1 <= capacity <= 3000`
- `0 <= key <= 3000`
- `0 <= value <= 104`
- At most `3 * 104` calls will be made to `get` and `put`.

题目大意

运用你所掌握的数据结构，设计和实现一个 LRU (最近最少使用) 缓存机制 。
实现 LRUCache 类：

- LRUCache(int capacity) 以正整数作为容量 capacity 初始化 LRU 缓存
- int get(int key) 如果关键字 key 存在于缓存中，则返回关键字的值，否则返回 -1 。
- void put(int key, int value) 如果关键字已经存在，则变更其数据值；如果关键字不存在，则插入该组「关键字-值」。当缓存容量达到上限时，它应该在写入新数据之前删除最久未使用的数据值，从而为新的数据值留出空间。

进阶：你是否可以在 O(1) 时间复杂度内完成这两种操作？

解题思路

- 这一题是 LRU 经典面试题，详细解释见第三章模板。

代码

```go
package leetcode

type LRUCache struct {
  head, tail *Node
  Keys       map[int]*Node
  Cap        int
}

type Node struct {
  Key, Val   int
  Prev, Next *Node
}

func Constructor(capacity int) LRUCache {
  return LRUCache{Keys: make(map[int]*Node), Cap: capacity}
}

func (this *LRUCache) Get(key int) int {
  if node, ok := this.Keys[key]; ok {
    this.Remove(node)
```

```go
    this.Add(node)
    return node.Val
  }
  return -1
}

func (this *LRUCache) Put(key int, value int) {
  if node, ok := this.Keys[key]; ok {
    node.Val = value
    this.Remove(node)
    this.Add(node)
    return
  } else {
    node = &Node{Key: key, Val: value}
    this.Keys[key] = node
    this.Add(node)
  }
  if len(this.Keys) > this.Cap {
    delete(this.Keys, this.tail.Key)
    this.Remove(this.tail)
  }
}

func (this *LRUCache) Add(node *Node) {
  node.Prev = nil
  node.Next = this.head
  if this.head != nil {
    this.head.Prev = node
  }
  this.head = node
  if this.tail == nil {
    this.tail = node
    this.tail.Next = nil
  }
}

func (this *LRUCache) Remove(node *Node) {
  if node == this.head {
    this.head = node.Next
    node.Next = nil
    return
  }
  if node == this.tail {
    this.tail = node.Prev
    node.Prev.Next = nil
    node.Prev = nil
    return
  }
  node.Prev.Next = node.Next
```

```
  node.Next.Prev = node.Prev
}
```

147. Insertion Sort List

题目

Sort a linked list using insertion sort.

A graphical example of insertion sort. The partial sorted list (black) initially contains only the first element in the list.
With each iteration one element (red) is removed from the input data and inserted in-place into the sorted list

Algorithm of Insertion Sort:

Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list.
At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there.
It repeats until no input elements remain.

Example 1:

```
Input: 4->2->1->3
Output: 1->2->3->4
```

Example 2:

```
Input: -1->5->3->4->0
Output: -1->0->3->4->5
```

题目大意

链表的插入排序

解题思路

按照题意做即可。

代码

```
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */
func insertionSortList(head *ListNode) *ListNode {
  if head == nil {
    return head
  }
  newHead := &ListNode{Val: 0, Next: nil} // 这里初始化不要直接指向 head，为了下面循环可以统一处理
  cur, pre := head, newHead
  for cur != nil {
    next := cur.Next
    for pre.Next != nil && pre.Next.Val < cur.Val {
      pre = pre.Next
    }
    cur.Next = pre.Next
    pre.Next = cur
    pre = newHead // 归位，重头开始
    cur = next
  }
  return newHead.Next
}
```

148. Sort List

题目

Sort a linked list in O(n log n) time using constant space complexity.

Example 1:

```
Input: 4->2->1->3
Output: 1->2->3->4
```

Example 2:

```
Input: -1->5->3->4->0
Output: -1->0->3->4->5
```

题目大意

链表的排序，要求时间复杂度必须是 O(n log n)，空间复杂度是 O(1)

解题思路

这道题只能用归并排序才能符合要求。归并排序需要的 2 个操作在其他题目已经出现过了，取中间点是第 876 题，合并 2 个有序链表是第 21 题。

代码

```go
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */
func sortList(head *ListNode) *ListNode {
  length := 0
  cur := head
  for cur != nil {
    length++
    cur = cur.Next
  }
  if length <= 1 {
    return head
  }

  middleNode := middleNode(head)
  cur = middleNode.Next
  middleNode.Next = nil
```

```go
  middleNode = cur

  left := sortList(head)
  right := sortList(middleNode)
  return mergeTwoLists(left, right)
}

func middleNode(head *ListNode) *ListNode {
  if head == nil || head.Next == nil {
    return head
  }
  p1 := head
  p2 := head
  for p2.Next != nil && p2.Next.Next != nil {
    p1 = p1.Next
    p2 = p2.Next.Next
  }
  return p1
}

func mergeTwoLists(l1 *ListNode, l2 *ListNode) *ListNode {
  if l1 == nil {
    return l2
  }
  if l2 == nil {
    return l1
  }
  if l1.Val < l2.Val {
    l1.Next = mergeTwoLists(l1.Next, l2)
    return l1
  }
  l2.Next = mergeTwoLists(l1, l2.Next)
  return l2
}
```

150. Evaluate Reverse Polish Notation

题目

Evaluate the value of an arithmetic expression in Reverse Polish Notation.

Valid operators are +, -, *, /. Each operand may be an integer or another expression.

Note:

- Division between two integers should truncate toward zero.
- The given RPN expression is always valid. That means the expression would always evaluate to a result

and there won't be any divide by zero operation.

Example 1:

```
Input: ["2", "1", "+", "3", "*"]
Output: 9
Explanation: ((2 + 1) * 3) = 9
```

Example 2:

```
Input: ["4", "13", "5", "/", "+"]
Output: 6
Explanation: (4 + (13 / 5)) = 6
```

Example 3:

```
Input: ["10", "6", "9", "3", "+", "-11", "*", "/", "*", "17", "+", "5", "+"]
Output: 22
Explanation:
  ((10 * (6 / ((9 + 3) * -11))) + 17) + 5
= ((10 * (6 / (12 * -11))) + 17) + 5
= ((10 * (6 / -132)) + 17) + 5
= ((10 * 0) + 17) + 5
= (0 + 17) + 5
= 17 + 5
= 22
```

题目大意

计算逆波兰表达式。

解题思路

这道题就是经典的考察栈的知识的题目。

代码

```go
package leetcode

import (
  "strconv"
```

```go
)

func evalRPN(tokens []string) int {
  stack := make([]int, 0, len(tokens))
  for _, token := range tokens {
    v, err := strconv.Atoi(token)
    if err == nil {
      stack = append(stack, v)
    } else {
      num1, num2 := stack[len(stack)-2], stack[len(stack)-1]
      stack = stack[:len(stack)-2]
      switch token {
      case "+":
        stack = append(stack, num1+num2)
      case "-":
        stack = append(stack, num1-num2)
      case "*":
        stack = append(stack, num1*num2)
      case "/":
        stack = append(stack, num1/num2)
      }
    }
  }
  return stack[0]
}
```

151. Reverse Words in a String

题目

Given an input string, reverse the string word by word.

Example 1:

```
Input: "the sky is blue"
Output: "blue is sky the"
```

Example 2:

```
Input: "  hello world!  "
Output: "world! hello"
Explanation: Your reversed string should not contain leading or trailing spaces.
```

Example 3:

```
Input: "a good   example"
Output: "example good a"
Explanation: You need to reduce multiple spaces between two words to a single space in the reversed string.
```

Note:

- A word is defined as a sequence of non-space characters.
- Input string may contain leading or trailing spaces. However, your reversed string should not contain leading or trailing spaces.
- You need to reduce multiple spaces between two words to a single space in the reversed string.

Follow up:

For C programmers, try to solve it *in-place* in *O*(1) extra space.

题目大意

给定一个字符串，逐个翻转字符串中的每个单词。

说明：

- 无空格字符构成一个单词。
- 输入字符串可以在前面或者后面包含多余的空格，但是反转后的字符不能包括。
- 如果两个单词间有多余的空格，将反转后单词间的空格减少到只含一个。

进阶：

- 请选用 C 语言的用户尝试使用 O(1) 额外空间复杂度的原地解法。

解题思路

- 给出一个中间有空格分隔的字符串，要求把这个字符串按照单词的维度前后翻转。
- 依照题意，先把字符串按照空格分隔成每个小单词，然后把单词前后翻转，最后再把每个单词中间添加空格。

代码

```go
package leetcode

import "strings"

func reverseWords151(s string) string {
  ss := strings.Fields(s)
  reverse151(&ss, 0, len(ss)-1)
  return strings.Join(ss, " ")
}

func reverse151(m *[]string, i int, j int) {
  for i <= j {
    (*m)[i], (*m)[j] = (*m)[j], (*m)[i]
```

```
    i++
    j--
  }
}
```

152. Maximum Product Subarray

题目

Given an integer array `nums`, find the contiguous subarray within an array (containing at least one number) which has the largest product.

Example 1:

```
Input: [2,3,-2,4]
Output: 6
Explanation: [2,3] has the largest product 6.
```

Example 2:

```
Input: [-2,0,-1]
Output: 0
Explanation: The result cannot be 2, because [-2,-1] is not a subarray.
```

题目大意

给定一个整数数组 nums ，找出一个序列中乘积最大的连续子序列（该序列至少包含一个数）。

解题思路

- 给出一个数组，要求找出这个数组中连续元素乘积最大的值。
- 这一题是 DP 的题，状态转移方程是：最大值是 `Max(f(n)) = Max( Max(f(n-1)) * n, Min(f(n-1)) * n)`；最小值是 `Min(f(n)) = Min( Max(f(n-1)) * n, Min(f(n-1)) * n)`。只要动态维护这两个值，如果最后一个数是负数，最大值就在负数 * 最小值中产生，如果最后一个数是正数，最大值就在正数 * 最大值中产生。

代码

```go
package leetcode

func maxProduct(nums []int) int {
  minimum, maximum, res := nums[0], nums[0], nums[0]
  for i := 1; i < len(nums); i++ {
```

```
    if nums[i] < 0 {
      maximum, minimum = minimum, maximum
    }
    maximum = max(nums[i], maximum*nums[i])
    minimum = min(nums[i], minimum*nums[i])
    res = max(res, maximum)
  }
  return res
}
```

153. Find Minimum in Rotated Sorted Array

题目

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., `[0,1,2,4,5,6,7]` might become `[4,5,6,7,0,1,2]`).

Find the minimum element.

You may assume no duplicate exists in the array.

Example 1:

```
Input: [3,4,5,1,2]
Output: 1
```

Example 2:

```
Input: [4,5,6,7,0,1,2]
Output: 0
```

题目大意

假设按照升序排序的数组在预先未知的某个点上进行了旋转。(例如，数组 [0,1,2,4,5,6,7] 可能变为 [4,5,6,7,0,1,2])。请找出其中最小的元素。

你可以假设数组中不存在重复元素。

解题思路

- 给出一个原本从小到大排序过的数组，但是在某一个分割点上，把数组切分后的两部分对调位置，数值偏大的放到了数组的前部。求这个数组中最小的元素。
- 求数组最小的元素其实就是找分割点，前一个数比当前数大，后一个数比当前数也要大。可以用二分搜索查找，需要查找的两个有序区间。时间复杂度 O(log n)。这一题也可以用暴力解法，从头开始遍历，动态维护一个最小值即可，时间复杂度 O(n)。

代码

```go
package leetcode

// 解法一 二分
func findMin(nums []int) int {
  low, high := 0, len(nums)-1
  for low < high {
    if nums[low] < nums[high] {
      return nums[low]
    }
    mid := low + (high-low)>>1
    if nums[mid] >= nums[low] {
      low = mid + 1
    } else {
      high = mid
    }
  }
  return nums[low]
}

// 解法二 二分
func findMin1(nums []int) int {
  if len(nums) == 0 {
    return 0
  }
  if len(nums) == 1 {
    return nums[0]
  }
  if nums[len(nums)-1] > nums[0] {
    return nums[0]
  }
  low, high := 0, len(nums)-1
  for low <= high {
    mid := low + (high-low)>>1
    if nums[low] < nums[high] {
      return nums[low]
    }
    if (mid == len(nums)-1 && nums[mid-1] > nums[mid]) || (mid < len(nums)-1 && mid > 0
&& nums[mid-1] > nums[mid] && nums[mid] < nums[mid+1]) {
      return nums[mid]
    }
    if nums[mid] > nums[low] && nums[low] > nums[high] { // mid 在数值大的一部分区间里
      low = mid + 1
    } else if nums[mid] < nums[low] && nums[low] > nums[high] { // mid 在数值小的一部分区
间里
      high = mid - 1
    } else {
      if nums[low] == nums[mid] {
        low++
```

```
      }
      if nums[high] == nums[mid] {
        high--
      }
    }
  }
  return -1
}

// 解法三 暴力
func findMin2(nums []int) int {
  min := nums[0]
  for _, num := range nums[1:] {
    if min > num {
      min = num
    }
  }
  return min
}
```

154. Find Minimum in Rotated Sorted Array II

题目

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., `[0,1,2,4,5,6,7]` might become `[4,5,6,7,0,1,2]`).

Find the minimum element.

The array may contain duplicates.

Example 1:

```
Input: [1,3,5]
Output: 1
```

Example 2:

```
Input: [2,2,2,0,1]
Output: 0
```

Note:

- This is a follow up problem to Find Minimum in Rotated Sorted Array.
- Would allow duplicates affect the run-time complexity? How and why?

题目大意

假设按照升序排序的数组在预先未知的某个点上进行了旋转。(例如，数组 [0,1,2,4,5,6,7] 可能变为 [4,5,6,7,0,1,2])。请找出其中最小的元素。

注意数组中可能存在重复的元素。

解题思路

- 给出一个原本从小到大排序过的数组，注意数组中有重复的元素。但是在某一个分割点上，把数组切分后的两部分对调位置，数值偏大的放到了数组的前部。求这个数组中最小的元素。
- 这一题是第 153 题的加强版，增加了重复元素的条件。但是实际做法还是没有变，还是用二分搜索，只不过在相等元素上多增加一个判断即可。时间复杂度 O(log n)。

代码

```
package leetcode

func findMin154(nums []int) int {
  low, high := 0, len(nums)-1
  for low < high {
    if nums[low] < nums[high] {
      return nums[low]
    }
    mid := low + (high-low)>>1
    if nums[mid] > nums[low] {
      low = mid + 1
    } else if nums[mid] == nums[low] {
      low++
    } else {
      high = mid
    }
  }
  return nums[low]
}
```

155. Min Stack

题目

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

push(x) -- Push element x onto stack.
pop() -- Removes the element on top of the stack.
top() -- Get the top element.
getMin() -- Retrieve the minimum element in the stack.

Example:

```
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin();   --> Returns -3.
minStack.pop();
minStack.top();      --> Returns 0.
minStack.getMin();   --> Returns -2.
```

题目大意

这道题是一个数据结构实现题。要求实现一个栈的类，实现 push()、pop()、top()、getMin()。

解题思路

按照题目要求实现即可。

代码

```go
package leetcode

// MinStack define
type MinStack struct {
  elements, min []int
  l             int
}

/** initialize your data structure here. */

// Constructor155 define
func Constructor155() MinStack {
  return MinStack{make([]int, 0), make([]int, 0), 0}
}

// Push define
func (this *MinStack) Push(x int) {
  this.elements = append(this.elements, x)
  if this.l == 0 {
    this.min = append(this.min, x)
  } else {
    min := this.GetMin()
    if x < min {
      this.min = append(this.min, x)
```

```
    } else {
      this.min = append(this.min, min)
    }
  }
  this.l++
}

func (this *MinStack) Pop() {
  this.l--
  this.min = this.min[:this.l]
  this.elements = this.elements[:this.l]
}

func (this *MinStack) Top() int {
  return this.elements[this.l-1]
}

func (this *MinStack) GetMin() int {
  return this.min[this.l-1]
}
```

160. Intersection of Two Linked Lists

题目

Write a program to find the node at which the intersection of two singly linked lists begins.

For example, the following two linked lists:

begin to intersect at node c1.

Example 1:

```
Input: intersectVal = 8, listA = [4,1,8,4,5], listB = [5,0,1,8,4,5], skipA = 2, skipB =
3
Output: Reference of the node with value = 8
Input Explanation: The intersected node's value is 8 (note that this must not be 0 if
the two lists intersect). From the head of A, it reads as [4,1,8,4,5]. From the head of
B, it reads as [5,0,1,8,4,5]. There are 2 nodes before the intersected node in A; There
are 3 nodes before the intersected node in B.
```

Example 2:

```
Input: intersectVal = 2, listA = [0,9,1,2,4], listB = [3,2,4], skipA = 3, skipB = 1
Output: Reference of the node with value = 2
Input Explanation: The intersected node's value is 2 (note that this must not be 0 if
the two lists intersect). From the head of A, it reads as [0,9,1,2,4]. From the head of
B, it reads as [3,2,4]. There are 3 nodes before the intersected node in A; There are 1
node before the intersected node in B.
```

Example 3:

```
Input: intersectVal = 0, listA = [2,6,4], listB = [1,5], skipA = 3, skipB = 2
Output: null
Input Explanation: From the head of A, it reads as [2,6,4]. From the head of B, it
reads as [1,5]. Since the two lists do not intersect, intersectVal must be 0, while
skipA and skipB can be arbitrary values.
Explanation: The two lists do not intersect, so return null.
```

Notes:

- If the two linked lists have no intersection at all, return null.
- The linked lists must retain their original structure after the function returns.
- You may assume there are no cycles anywhere in the entire linked structure.
- Your code should preferably run in O(n) time and use only O(1) memory.

题目大意

找到 2 个链表的交叉点。

解题思路

这道题的思路其实类似链表找环。

给定的 2 个链表的长度如果一样长，都从头往后扫即可。如果不一样长，需要先“拼成”一样长。把 B 拼接到 A 后面，把 A 拼接到 B 后面。这样 2 个链表的长度都是 A + B。再依次扫描比较 2 个链表的结点是否相同。

代码

```go
package leetcode

import "fmt"
```

```go
/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */
func getIntersectionNode(headA, headB *ListNode) *ListNode {
  //boundary check
  if headA == nil || headB == nil {
    return nil
  }

  a := headA
  b := headB

  //if a & b have different len, then we will stop the loop after second iteration
  for a != b {
    //for the end of first iteration, we just reset the pointer to the head of another
linkedlist
    if a == nil {
      a = headB
    } else {
      a = a.Next
    }

    if b == nil {
      b = headA
    } else {
      b = b.Next
    }
    fmt.Printf("a = %v b = %v\n", a, b)
  }
  return a
}
```

162. Find Peak Element

题目

A peak element is an element that is greater than its neighbors.

Given an input array `nums`, where `nums[i] ≠ nums[i+1]`, find a peak element and return its index.

The array may contain multiple peaks, in that case return the index to any one of the peaks is fine.

You may imagine that `nums[-1] = nums[n] = -∞`.

Example 1:

```
Input: nums = [1,2,3,1]
Output: 2
Explanation: 3 is a peak element and your function should return the index number 2.
```

Example 2:

```
Input: nums = [1,2,1,3,5,6,4]
Output: 1 or 5
Explanation: Your function can return either index number 1 where the peak element is 2,
             or index number 5 where the peak element is 6.
```

Note:

Your solution should be in logarithmic complexity.

题目大意

峰值元素是指其值大于左右相邻值的元素。给定一个输入数组 nums，其中 nums[i] ≠ nums[i+1]，找到峰值元素并返回其索引。数组可能包含多个峰值，在这种情况下，返回任何一个峰值所在位置即可。你可以假设 nums[-1] = nums[n] = -∞。

说明:

- 你的解法应该是 O(logN) 时间复杂度的。

解题思路

- 给出一个数组，数组里面存在多个“山峰”，(山峰的定义是，下标 `i` 比 `i-1`、`i+1` 位置上的元素都要大)，找到这个“山峰”，并输出其中一个山峰的下标。
- 这一题是第 852 题的伪加强版，第 852 题中只存在一个山峰，这一题存在多个山峰。但是实际上搜索的代码是一样的，因为此题只要求随便输出一个山峰的下标即可。思路同第 852 题。

代码

```go
package leetcode

// 解法一 二分
func findPeakElement(nums []int) int {
  if len(nums) == 0 || len(nums) == 1 {
    return 0
  }
  low, high := 0, len(nums)-1
  for low <= high {
    mid := low + (high-low)>>1
    if (mid == len(nums)-1 && nums[mid-1] < nums[mid]) || (mid > 0 && nums[mid-1] < nums[mid] && (mid <= len(nums)-2 && nums[mid+1] < nums[mid])) || (mid == 0 && nums[1] < nums[0]) {
```

```go
        return mid
      }
      if mid > 0 && nums[mid-1] < nums[mid] {
        low = mid + 1
      }
      if mid > 0 && nums[mid-1] > nums[mid] {
        high = mid - 1
      }
      if mid == low {
        low++
      }
      if mid == high {
        high--
      }
    }
    return -1
}

// 解法二 二分
func findPeakElement1(nums []int) int {
    low, high := 0, len(nums)-1
    for low < high {
      mid := low + (high-low)>>1
      // 如果 mid 较大，则左侧存在峰值，high = m，如果 mid + 1 较大，则右侧存在峰值，low = mid + 1
      if nums[mid] > nums[mid+1] {
        high = mid
      } else {
        low = mid + 1
      }
    }
    return low
}
```

164. Maximum Gap

题目

Given an unsorted array, find the maximum difference between the successive elements in its sorted form.

Return 0 if the array contains less than 2 elements.

Example 1:

```
Input: [3,6,9,1]
Output: 3
Explanation: The sorted form of the array is [1,3,6,9], either
             (3,6) or (6,9) has the maximum difference 3.
```

Example 2:

```
Input: [10]
Output: 0
Explanation: The array contains less than 2 elements, therefore return 0.
```

Note:

- You may assume all elements in the array are non-negative integers and fit in the 32-bit signed integer range.
- Try to solve it in linear time/space.

题目大意

在数组中找到 2 个数字之间最大的间隔。要求尽量用 O(1) 的时间复杂度和空间复杂度。

解题思路

虽然使用排序算法可以 AC 这道题。先排序，然后依次计算数组中两两数字之间的间隔，找到最大的一个间隔输出即可。

这道题满足要求的做法是基数排序。

代码

```go
package leetcode

// 解法一 快排
func maximumGap(nums []int) int {
  if len(nums) < 2 {
    return 0
  }
  quickSort164(nums, 0, len(nums)-1)
  res := 0
  for i := 0; i < len(nums)-1; i++ {
    if (nums[i+1] - nums[i]) > res {
      res = nums[i+1] - nums[i]
    }
```

```go
  }
  return res
}

func partition164(a []int, lo, hi int) int {
  pivot := a[hi]
  i := lo - 1
  for j := lo; j < hi; j++ {
    if a[j] < pivot {
      i++
      a[j], a[i] = a[i], a[j]
    }
  }
  a[i+1], a[hi] = a[hi], a[i+1]
  return i + 1
}
func quickSort164(a []int, lo, hi int) {
  if lo >= hi {
    return
  }
  p := partition164(a, lo, hi)
  quickSort164(a, lo, p-1)
  quickSort164(a, p+1, hi)
}

// 解法二 基数排序
func maximumGap1(nums []int) int {
  if nums == nil || len(nums) < 2 {
    return 0
  }
  // m is the maximal number in nums
  m := nums[0]
  for i := 1; i < len(nums); i++ {
    m = max(m, nums[i])
  }
  exp := 1 // 1, 10, 100, 1000 ...
  R := 10  // 10 digits
  aux := make([]int, len(nums))
  for (m / exp) > 0 { // Go through all digits from LSB to MSB
    count := make([]int, R)
    for i := 0; i < len(nums); i++ {
      count[(nums[i]/exp)%10]++
    }
    for i := 1; i < len(count); i++ {
      count[i] += count[i-1]
    }
    for i := len(nums) - 1; i >= 0; i-- {
      tmp := count[(nums[i]/exp)%10]
      tmp--
```

```
      aux[tmp] = nums[i]
      count[(nums[i]/exp)%10] = tmp
    }
    for i := 0; i < len(nums); i++ {
      nums[i] = aux[i]
    }
    exp *= 10
  }
  maxValue := 0
  for i := 1; i < len(aux); i++ {
    maxValue = max(maxValue, aux[i]-aux[i-1])
  }
  return maxValue
}

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

167. Two Sum II - Input array is sorted

题目

Given an array of integers that is already sorted in ascending order, find two numbers such that they add up to a specific target number.

The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2.

Note:

- Your returned answers (both index1 and index2) are not zero-based.
- You may assume that each input would have exactly one solution and you may not use the same element twice.

Example:

```
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.
```

题目大意

找出两个数之和等于 target 的两个数字，要求输出它们的下标。注意一个数字不能使用 2 次。下标从小到大输出。假定题目一定有一个解。

解题思路

这一题比第 1 题 Two Sum 的问题还要简单，因为这里数组是有序的。可以直接用第一题的解法解决这道题。

代码

```go
package leetcode

// 解法一 这一题可以利用数组有序的特性
func twoSum167(numbers []int, target int) []int {
  i, j := 0, len(numbers)-1
  for i < j {
    if numbers[i]+numbers[j] == target {
      return []int{i + 1, j + 1}
    }
    if numbers[i]+numbers[j] < target {
      i++
    } else {
      j--
    }
  }
  return nil
}

// 解法二 不管数组是否有序，空间复杂度比上一种解法要多 O(n)
func twoSum167_1(numbers []int, target int) []int {
  m := make(map[int]int)
  for i := 0; i < len(numbers); i++ {
    another := target - numbers[i]
    if idx, ok := m[another]; ok {
      return []int{idx + 1, i + 1}
    }
    m[numbers[i]] = i
  }
  return nil
}
```

168. Excel Sheet Column Title

题目

Given a positive integer, return its corresponding column title as appear in an Excel sheet.

For example:

```
    1 -> A
    2 -> B
    3 -> C
    ...
    26 -> Z
    27 -> AA
    28 -> AB
    ...
```

Example 1:

```
Input: 1
Output: "A"
```

Example 2:

```
Input: 28
Output: "AB"
```

Example 3:

```
Input: 701
Output: "ZY"
```

题目大意

给定一个正整数，返回它在 Excel 表中相对应的列名称。

例如，

```
1 -> A
2 -> B
3 -> C
...
26 -> Z
27 -> AA
28 -> AB
...
```

解题思路

- 给定一个正整数，返回它在 Excel 表中的对应的列名称
- 简单题。这一题就类似短除法的计算过程。以 26 进制的字母编码。按照短除法先除，然后余数逆序输出即

可。

代码

```
package leetcode

func convertToTitle(n int) string {
  result := []byte{}
  for n > 0 {
    result = append(result, 'A'+byte((n-1)%26))
    n = (n - 1) / 26
  }
  for i, j := 0, len(result)-1; i < j; i, j = i+1, j-1 {
    result[i], result[j] = result[j], result[i]
  }
  return string(result)
}
```

169. Majority Element

题目

Given an array of size n, find the majority element. The majority element is the element that appears **more than** `⌊ n/2 ⌋` times.

You may assume that the array is non-empty and the majority element always exist in the array.

Example 1:

```
Input: [3,2,3]
Output: 3
```

Example 2:

```
Input: [2,2,1,1,1,2,2]
Output: 2
```

题目大意

给定一个大小为 n 的数组，找到其中的众数。众数是指在数组中出现次数大于 ⌊ n/2 ⌋ 的元素。你可以假设数组是非空的，并且给定的数组总是存在众数。

解题思路

- 题目要求找出数组中出现次数大于 `⌊ n/2 ⌋` 次的数。要求空间复杂度为 O(1)。简单题。

- 这一题利用的算法是 Boyer-Moore Majority Vote Algorithm。https://www.zhihu.com/question/49973163/answer/235921864

代码

```
package leetcode

// 解法一 时间复杂度 O(n) 空间复杂度 O(1)
func majorityElement(nums []int) int {
  res, count := nums[0], 0
  for i := 0; i < len(nums); i++ {
    if count == 0 {
      res, count = nums[i], 1
    } else {
      if nums[i] == res {
        count++
      } else {
        count--
      }
    }
  }
  return res
}

// 解法二 时间复杂度 O(n) 空间复杂度 O(n)
func majorityElement1(nums []int) int {
  m := make(map[int]int)
  for _, v := range nums {
    m[v]++
    if m[v] > len(nums)/2 {
      return v
    }
  }
  return 0
}
```

171. Excel Sheet Column Number

题目

Given a column title as appear in an Excel sheet, return its corresponding column number.

For example:

```
    A -> 1
    B -> 2
    C -> 3
    ...
    Z -> 26
    AA -> 27
    AB -> 28
    ...
```

Example 1:

```
Input: "A"
Output: 1
```

Example 2:

```
Input: "AB"
Output: 28
```

Example 3:

```
Input: "ZY"
Output: 701
```

题目大意

给定一个 Excel 表格中的列名称，返回其相应的列序号。

解题思路

- 给出 Excel 中列的名称，输出其对应的列序号。
- 简单题。这一题是第 168 题的逆序题。按照 26 进制还原成十进制即可。

代码

```go
package leetcode

func titleToNumber(s string) int {
  val, res := 0, 0
  for i := 0; i < len(s); i++ {
    val = int(s[i] - 'A' + 1)
    res = res*26 + val
  }
  return res
}
```

172. Factorial Trailing Zeroes

题目

Given an integer n, return the number of trailing zeroes in n!.

Example 1:

```
Input: 3
Output: 0
Explanation: 3! = 6, no trailing zero.
```

Example 2:

```
Input: 5
Output: 1
Explanation: 5! = 120, one trailing zero.
```

Note: Your solution should be in logarithmic time complexity.

题目大意

给定一个整数 n，返回 n! 结果尾数中零的数量。说明: 你算法的时间复杂度应为 O(log n) 。

解题思路

- 给出一个数 n，要求 n！末尾 0 的个数。
- 这是一道数学题。计算 N 的阶乘有多少个后缀 0，即计算 N! 里有多少个 10，也是计算 N! 里有多少个 2 和 5（分解质因数），最后结果即 2 的个数和 5 的个数取较小值。每两个数字就会多一个质因数 2，而每五个数字才多一个质因数 5。每 5 个数字就会多一个质因数 5。0~4 的阶乘里没有质因数 5，5~9 的阶乘里有 1 个质因数 5，10~14 的阶乘里有 2 个质因数 5，依此类推。所以 0 的个数即为 `min(阶乘中 5 的个数和 2 的个数)` 。

- N! 有多少个后缀 0，即 N! 有多少个质因数 5。N! 有多少个质因数 5，即 N 可以划分成多少组 5个数字一组，加上划分成多少组 25 个数字一组，加上划分多少组成 125 个数字一组，等等。即 `res = N/5 + N/(5^2) + N/(5^3) + ... = ((N / 5) / 5) / 5 /...` 。最终算法复杂度为 O(logN)。

代码

```go
package leetcode

func trailingZeroes(n int) int {
  if n/5 == 0 {
    return 0
  }
  return n/5 + trailingZeroes(n/5)
}
```

173. Binary Search Tree Iterator

题目

Implement an iterator over a binary search tree (BST). Your iterator will be initialized with the root node of a BST.

Calling `next()` will return the next smallest number in the BST.

Example:

```
BSTIterator iterator = new BSTIterator(root);
iterator.next();    // return 3
iterator.next();    // return 7
iterator.hasNext(); // return true
iterator.next();    // return 9
iterator.hasNext(); // return true
iterator.next();    // return 15
iterator.hasNext(); // return true
iterator.next();    // return 20
iterator.hasNext(); // return false
```

Note:

- `next()` and `hasNext()` should run in average O(1) time and uses O(h) memory, where h is the height of the tree.
- You may assume that `next()` call will always be valid, that is, there will be at least a next smallest number in the BST when `next()` is called.

题目大意

实现一个二叉搜索树迭代器。你将使用二叉搜索树的根节点初始化迭代器。调用 next() 将返回二叉搜索树中的下一个最小的数。

解题思路

- 用优先队列解决即可

代码

```go
package leetcode

import (
  "container/heap"

  "github.com/halfrost/LeetCode-Go/structures"
)

// TreeNode define
type TreeNode = structures.TreeNode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

// BSTIterator define
type BSTIterator struct {
  pq    PriorityQueueOfInt
  count int
}

// Constructor173 define
func Constructor173(root *TreeNode) BSTIterator {
```

```go
  result, pq := []int{}, PriorityQueueOfInt{}
  postorder(root, &result)
  for _, v := range result {
    heap.Push(&pq, v)
  }
  bs := BSTIterator{pq: pq, count: len(result)}
  return bs
}

func postorder(root *TreeNode, output *[]int) {
  if root != nil {
    postorder(root.Left, output)
    postorder(root.Right, output)
    *output = append(*output, root.Val)
  }
}

/** @return the next smallest number */
func (this *BSTIterator) Next() int {
  this.count--
  return heap.Pop(&this.pq).(int)
}

/** @return whether we have a next smallest number */
func (this *BSTIterator) HasNext() bool {
  return this.count != 0
}

/**
 * Your BSTIterator object will be instantiated and called as such:
 * obj := Constructor(root);
 * param_1 := obj.Next();
 * param_2 := obj.HasNext();
 */
type PriorityQueueOfInt []int

func (pq PriorityQueueOfInt) Len() int {
  return len(pq)
}

func (pq PriorityQueueOfInt) Less(i, j int) bool {
  return pq[i] < pq[j]
}

func (pq PriorityQueueOfInt) Swap(i, j int) {
  pq[i], pq[j] = pq[j], pq[i]
}

func (pq *PriorityQueueOfInt) Push(x interface{}) {
```

```go
  item := x.(int)
  *pq = append(*pq, item)
}

func (pq *PriorityQueueOfInt) Pop() interface{} {
  n := len(*pq)
  item := (*pq)[n-1]
  *pq = (*pq)[:n-1]
  return item
}
```

174. Dungeon Game

题目

The demons had captured the princess (**P**) and imprisoned her in the bottom-right corner of a dungeon. The dungeon consists of M x N rooms laid out in a 2D grid. Our valiant knight (**K**) was initially positioned in the top-left room and must fight his way through the dungeon to rescue the princess.

The knight has an initial health point represented by a positive integer. If at any point his health point drops to 0 or below, he dies immediately.

Some of the rooms are guarded by demons, so the knight loses health (*negative* integers) upon entering these rooms; other rooms are either empty (*0's*) or contain magic orbs that increase the knight's health (*positive* integers).

In order to reach the princess as quickly as possible, the knight decides to move only rightward or downward in each step.

Write a function to determine the knight's minimum initial health so that he is able to rescue the princess.

For example, given the dungeon below, the initial health of the knight must be at least **7** if he follows the optimal path `RIGHT-> RIGHT -> DOWN -> DOWN`.

-2 (K)	-3	3
-5	-10	1
10	30	-5 (P)

Note:

- The knight's health has no upper bound.
- Any room can contain threats or power-ups, even the first room the knight enters and the bottom-right room where the princess is imprisoned.

题目大意

一些恶魔抓住了公主（P）并将她关在了地下城的右下角。地下城是由 M x N 个房间组成的二维网格。我们英勇的骑士（K）最初被安置在左上角的房间里，他必须穿过地下城并通过对抗恶魔来拯救公主。

骑士的初始健康点数为一个正整数。如果他的健康点数在某一时刻降至 0 或以下，他会立即死亡。

有些房间由恶魔守卫，因此骑士在进入这些房间时会失去健康点数（若房间里的值为负整数，则表示骑士将损失健康点数）；其他房间要么是空的（房间里的值为 0），要么包含增加骑士健康点数的魔法球（若房间里的值为正整数，则表示骑士将增加健康点数）。

为了尽快到达公主，骑士决定每次只向右或向下移动一步。编写一个函数来计算确保骑士能够拯救到公主所需的最低初始健康点数。

说明:

- 骑士的健康点数没有上限。
- 任何房间都可能对骑士的健康点数造成威胁，也可能增加骑士的健康点数，包括骑士进入的左上角房间以及公主被监禁的右下角房间。

解题思路

- 在二维地图上给出每个格子扣血数，负数代表扣血，正数代表补血。左上角第一个格子是起点，右下角最后一个格子是终点。问骑士初始最少多少血才能走完迷宫，顺利营救位于终点的公主。需要注意的是，起点和终点都会对血量进行影响。每到一个格子，骑士的血都不能少于 1，一旦少于 1 点血，骑士就会死去。
- 这一题首先想到的解题思路是动态规划。从终点逆推回起点。`dp[i][j]` 代表骑士进入坐标为 `(i,j)` 的格子之前最少的血量值。 那么 `dp[m-1][n-1]` 应该同时满足两个条件，`dp[m-1][n-1] + dungeon[m-1][n-1] ≥ 1` 并且 `dp[m-1][n-1] ≥ 1`，由于这两个不等式的方向是相同的，取交集以后，起决定作用的是数轴最右边的数，即 `max(1-dungeon[m-1][n-1] , 1)`。算出 `dp[m-1][n-1]` 以后，接着可以推出 `dp[m-1][i]` 这一行和 `dp[i][n-1]` 这一列的值。因为骑士只能往右走和往下走。往回推，即只能往上走和往左走。到这里，DP 的初始条件都准备好了。那么状态转移方程是什么呢？分析一般的情况，`dp[i][j]` 这个值应该是和 `dp[i+1][j]` 和 `dp[i][j+1]` 这两者有关系。即 `dp[i][j]` 经过自己本格子的扣血以后，要能至少满足下一行和右一列格子血量的最少要求。并且自己的血量也应该 `≥1`。即需要满足下面这两组不等式。

  {{< katex display >}}
  \begin{matrix} \left{
    \begin{array}{lr}
     dp[i][j] + dungeon[i][j] \geqslant dp[i+1][j] \
     dp[i][j] \geqslant 1
    \end{array} \right.
  \end{matrix}
  {{< /katex >}}

{{< katex display >}}
\begin{matrix} \left{ \begin{array}{lr} dp[i][j] + dungeon[i][j] \geqslant dp[i][j+1] \ dp[i][j] \geqslant 1 \end{array} \right. \end{matrix}
{{< /katex >}}
上面不等式中第一组不等式是满足下一行格子的最低血量要求，第二组不等式是满足右一列格子的最低血量要求。第一个式子化简即 `dp[i][j] = max(1, dp[i+1][j]-dungeon[i][j])`，第二个式子化简即 `dp[i][j] = max(1, dp[i][j+1]-dungeon[i][j])`。求得了这两种走法的最低血量值，从这两个值里面取最小，即是当前格子所需的最低血量，所以状态转移方程为 `dp[i][j] = min(max(1, dp[i][j+1]-dungeon[i][j]), max(1, dp[i+1][j]-dungeon[i][j]))`。DP 完成以后，`dp[0][0]` 中记录的就是骑士初始最低血量值。时间复杂度 O(m*n)，空间复杂度 O(m*n)。

- 这一题还可以用二分搜索来求解。骑士的血量取值范围一定是在 `[1, +∞)` 这个区间内。那么二分这个区间，每次二分的中间值，再用 dp 在地图中去判断是否能到达终点，如果能，就缩小搜索空间至 `[1, mid]`，否则搜索空间为 `[mid + 1, +∞)` 。时间复杂度 O(m*n* log math.MaxInt64)，空间复杂度 O(m*n)。

代码

```
package leetcode

import "math"

// 解法一 动态规划
func calculateMinimumHP(dungeon [][]int) int {
  if len(dungeon) == 0 {
    return 0
  }
  m, n := len(dungeon), len(dungeon[0])
  dp := make([][]int, m)
  for i := 0; i < m; i++ {
    dp[i] = make([]int, n)
  }
  dp[m-1][n-1] = max(1-dungeon[m-1][n-1], 1)
  for i := n - 2; i >= 0; i-- {
    dp[m-1][i] = max(1, dp[m-1][i+1]-dungeon[m-1][i])
  }
  for i := m - 2; i >= 0; i-- {
    dp[i][n-1] = max(1, dp[i+1][n-1]-dungeon[i][n-1])
  }
  for i := m - 2; i >= 0; i-- {
    for j := n - 2; j >= 0; j-- {
      dp[i][j] = min(max(1, dp[i][j+1]-dungeon[i][j]), max(1, dp[i+1][j]-dungeon[i]
[j]))
    }
  }
  return dp[0][0]
}
```

```go
// 解法二 二分搜索
func calculateMinimumHP1(dungeon [][]int) int {
  low, high := 1, math.MaxInt64
  for low < high {
    mid := low + (high-low)>>1
    if canCross(dungeon, mid) {
      high = mid
    } else {
      low = mid + 1
    }
  }
  return low
}

func canCross(dungeon [][]int, start int) bool {
  m, n := len(dungeon), len(dungeon[0])
  dp := make([][]int, m)
  for i := 0; i < m; i++ {
    dp[i] = make([]int, n)
  }
  for i := 0; i < len(dp); i++ {
    for j := 0; j < len(dp[i]); j++ {
      if i == 0 && j == 0 {
        dp[i][j] = start + dungeon[0][0]
      } else {
        a, b := math.MinInt64, math.MinInt64
        if i > 0 && dp[i-1][j] > 0 {
          a = dp[i-1][j] + dungeon[i][j]
        }
        if j > 0 && dp[i][j-1] > 0 {
          b = dp[i][j-1] + dungeon[i][j]
        }
        dp[i][j] = max(a, b)
      }
    }
  }
  return dp[m-1][n-1] > 0
}
```

179. Largest Number

题目

Given a list of non negative integers, arrange them such that they form the largest number.

Example 1:

```
Input: [10,2]
Output: "210"
```

Example 2:

```
Input: [3,30,34,5,9]
Output: "9534330"
```

Note:

The result may be very large, so you need to return a string instead of an integer.

题目大意

给出一个数组，要求排列这些数组里的元素，使得最终排列出来的数字是最大的。

解题思路

这一题很容易想到把数字都转化为字符串，利用字符串比较，来排序，这样 9 开头的一定排在最前面。不过这样做有一个地方是错误的，比如："3" 和 "30" 比较，"30" 比 "3" 的字符序要大，这样排序以后就出错了。实际上就这道题而言， "3" 应该排在 "30" 前面。

在比较 2 个字符串大小的时候，不单纯的只用字符串顺序进行比较，还加入一个顺序。

```
aStr := a + b
bStr := b + a
```

通过比较 aStr 和 bStr 的大小来得出是 a 大还是 b 大。

举个例子，还是 "3" 和 "30" 的例子，比较这 2 个字符串的大小。

```
aStr := "3" + "30" = "330"
bStr := "30" + "3" = "303"
```

通过互相补齐位数之后再进行比较，就没有问题了。很显然这里 "3" 比 "30" 要大。

代码

```
```

```go
package leetcode

import (
  "strconv"
)

func largestNumber(nums []int) string {
  if len(nums) == 0 {
    return ""
  }
  numStrs := toStringArray(nums)
  quickSortString(numStrs, 0, len(numStrs)-1)
  res := ""
  for _, str := range numStrs {
    if res == "0" && str == "0" {
      continue
    }
    res = res + str
  }
  return res
}

func toStringArray(nums []int) []string {
  strs := make([]string, 0)
  for _, num := range nums {
    strs = append(strs, strconv.Itoa(num))
  }
  return strs
}
func partitionString(a []string, lo, hi int) int {
  pivot := a[hi]
  i := lo - 1
  for j := lo; j < hi; j++ {
    ajStr := a[j] + pivot
    pivotStr := pivot + a[j]
    if ajStr > pivotStr { // 这里的判断条件是关键
      i++
      a[j], a[i] = a[i], a[j]
    }
  }
  a[i+1], a[hi] = a[hi], a[i+1]
  return i + 1
}
func quickSortString(a []string, lo, hi int) {
  if lo >= hi {
    return
  }
  p := partitionString(a, lo, hi)
  quickSortString(a, lo, p-1)
```

```
  quickSortString(a, p+1, hi)
}
```

187. Repeated DNA Sequences

题目

All DNA is composed of a series of nucleotides abbreviated as A, C, G, and T, for Example: "ACGAATTCCG". When studying DNA, it is sometimes useful to identify repeated sequences within the DNA.

Write a function to find all the 10-letter-long sequences (substrings) that occur more than once in a DNA molecule.

Example:

```
Input: s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT"

Output: ["AAAAACCCCC", "CCCCCAAAAA"]
```

题目大意

所有 DNA 由一系列缩写为 A，C，G 和 T 的核苷酸组成，例如：“ACGAATTCCG”。在研究 DNA 时，识别 DNA 中的重复序列有时会对研究非常有帮助。编写一个函数来查找 DNA 分子中所有出现超多一次的10个字母长的序列（子串）。

解题思路

- 这一题不用位运算比较好做，维护一个长度为 10 的字符串，在 map 中出现次数 > 1 就输出。
- 用位运算想做这一题，需要动态的维护长度为 10 的 hashkey，先计算开头长度为 9 的 hash，在往后面扫描的过程中，如果长度超过了 10 ，就移除 hash 开头的一个字符，加入后面一个字符。具体做法是先将 ATCG 变成 00，01，10，11 的编码，那么长度为 10 ，hashkey 就需要维护在 20 位。mask = 0xFFFFF 就是 20 位的。维护了 hashkey 以后，根据这个 hashkey 进行去重和统计频次。

代码

```go
package leetcode

// 解法一
func findRepeatedDnaSequences(s string) []string {
  if len(s) < 10 {
    return nil
  }
  charMap, mp, result := map[uint8]uint32{'A': 0, 'C': 1, 'G': 2, 'T': 3},
make(map[uint32]int, 0), []string{}
  var cur uint32
  for i := 0; i < 9; i++ { // 前9位，忽略
```

```go
    cur = cur<<2 | charMap[s[i]]
  }
  for i := 9; i < len(s); i++ {
    cur = ((cur << 2) & 0xFFFFF) | charMap[s[i]]
    if mp[cur] == 0 {
      mp[cur] = 1
    } else if mp[cur] == 1 { // >2, 重复
      mp[cur] = 2
      result = append(result, s[i-9:i+1])
    }
  }
  return result
}

// 解法二
func findRepeatedDnaSequences1(s string) []string {
  if len(s) < 10 {
    return []string{}
  }
  ans, cache := make([]string, 0), make(map[string]int)
  for i := 0; i <= len(s)-10; i++ {
    curr := string(s[i : i+10])
    if cache[curr] == 1 {
      ans = append(ans, curr)
    }
    cache[curr]++
  }
  return ans
}
```

189. Rotate Array

题目

Given an array, rotate the array to the right by *k* steps, where *k* is non-negative.

Follow up:

- Try to come up as many solutions as you can, there are at least 3 different ways to solve this problem.
- Could you do it in-place with O(1) extra space?

Example 1:

```
Input: nums = [1,2,3,4,5,6,7], k = 3
Output: [5,6,7,1,2,3,4]
Explanation:
rotate 1 steps to the right: [7,1,2,3,4,5,6]
rotate 2 steps to the right: [6,7,1,2,3,4,5]
rotate 3 steps to the right: [5,6,7,1,2,3,4]
```

Example 2:

```
Input: nums = [-1,-100,3,99], k = 2
Output: [3,99,-1,-100]
Explanation:
rotate 1 steps to the right: [99,-1,-100,3]
rotate 2 steps to the right: [3,99,-1,-100]
```

Constraints:

- `1 <= nums.length <= 2 * 10^4`
- `-2^31 <= nums[i] <= 2^31 - 1`
- `0 <= k <= 10^5`

题目大意

给定一个数组，将数组中的元素向右移动 k 个位置，其中 k 是非负数。

解题思路

- 解法二，使用一个额外的数组，先将原数组下标为 i 的元素移动到 `(i+k) mod n` 的位置，再将剩下的元素拷贝回来即可。
- 解法一，由于题目要求不能使用额外的空间，所以本题最佳解法不是解法二。翻转最终态是，末尾 `k mod n` 个元素移动至了数组头部，剩下的元素右移 `k mod n` 个位置至最尾部。确定了最终态以后再变换就很容易。先将数组中所有元素从头到尾翻转一次，尾部的所有元素都到了头部，然后再将 `[0,(k mod n) - 1]` 区间内的元素翻转一次，最后再将 `[k mod n, n - 1]` 区间内的元素翻转一次，即可满足题目要求。

代码

```go
package leetcode

// 解法一 时间复杂度 O(n)，空间复杂度 O(1)
func rotate(nums []int, k int) {
  k %= len(nums)
  reverse(nums)
  reverse(nums[:k])
  reverse(nums[k:])
}

func reverse(a []int) {
```

```go
  for i, n := 0, len(a); i < n/2; i++ {
    a[i], a[n-1-i] = a[n-1-i], a[i]
  }
}

// 解法二 时间复杂度 O(n), 空间复杂度 O(n)
func rotate1(nums []int, k int) {
  newNums := make([]int, len(nums))
  for i, v := range nums {
    newNums[(i+k)%len(nums)] = v
  }
  copy(nums, newNums)
}
```

190. Reverse Bits

题目

Reverse bits of a given 32 bits unsigned integer.

Example 1:

```
Input: 00000010100101000001111010011100
Output: 00111001011110000010100101000000
Explanation: The input binary string 00000010100101000001111010011100 represents the
unsigned integer 43261596, so return 964176192 which its binary representation is
00111001011110000010100101000000.
```

Example 2:

```
Input: 11111111111111111111111111111101
Output: 10111111111111111111111111111111
Explanation: The input binary string 11111111111111111111111111111101 represents the
unsigned integer 4294967293, so return 3221225471 which its binary representation is
10101111110010110010011101101001.
```

Note:

- Note that in some languages such as Java, there is no unsigned integer type. In this case, both input and output will be given as signed integer type and should not affect your implementation, as the internal binary representation of the integer is the same whether it is signed or unsigned.
- In Java, the compiler represents the signed integers using 2's complement notation. Therefore, in **Example 2** above the input represents the signed integer `-3` and the output represents the signed integer `-1073741825`.

题目大意

颠倒给定的 32 位无符号整数的二进制位。提示：

- 请注意，在某些语言（如 Java）中，没有无符号整数类型。在这种情况下，输入和输出都将被指定为有符号整数类型，并且不应影响您的实现，因为无论整数是有符号的还是无符号的，其内部的二进制表示形式都是相同的。
- 在 Java 中，编译器使用二进制补码记法来表示有符号整数。因此，在上面的 示例 2 中，输入表示有符号整数 -3，输出表示有符号整数 -1073741825。

解题思路

- 简单题，要求反转 32 位的二进制位。
- 把 num 往右移动，不断的消灭右边最低位的 1，将这个 1 给 res，res 不断的左移即可实现反转二进制位的目的。

代码

```
package leetcode

func reverseBits(num uint32) uint32 {
  var res uint32
  for i := 0; i < 32; i++ {
    res = res<<1 | num&1
    num >>= 1
  }
  return res
}
```

191. Number of 1 Bits

题目

Write a function that takes an unsigned integer and return the number of '1' bits it has (also known as the Hamming weight).

Example 1:

```
Input: 00000000000000000000000000001011
Output: 3
Explanation: The input binary string 00000000000000000000000000001011 has a total of three '1' bits.
```

Example 2:

```
Input: 00000000000000000000000010000000
Output: 1
Explanation: The input binary string 00000000000000000000000010000000 has a total of one '1' bit.
```

Example 3:

```
Input: 11111111111111111111111111111101
Output: 31
Explanation: The input binary string 11111111111111111111111111111101 has a total of
thirty one '1' bits.
```

Note:

- Note that in some languages such as Java, there is no unsigned integer type. In this case, the input will be given as signed integer type and should not affect your implementation, as the internal binary representation of the integer is the same whether it is signed or unsigned.
- In Java, the compiler represents the signed integers using 2's complement notation. Therefore, in **Example 3** above the input represents the signed integer `-3`.

题目大意

编写一个函数，输入是一个无符号整数，返回其二进制表达式中数字位数为 ‘1’ 的个数（也被称为汉明重量）。

解题思路

- 求 uint32 数的二进制位中 1 的个数。
- 这一题的解题思路就是利用二进制位操作。`x = x & ( x -1 )` 这个操作可以清除最低位的二进制位 1，利用这个操作，直至把数清零。操作了几次即为有几个二进制位 1 。
- 最简单的方法即是直接调用库函数 `bits.OnesCount(uint(num))` 。

代码

```go
package leetcode

import "math/bits"

// 解法一
func hammingWeight(num uint32) int {
  return bits.OnesCount(uint(num))
}

// 解法二
func hammingWeight1(num uint32) int {
  count := 0
  for num != 0 {
    num = num & (num - 1)
    count++
  }
  return count
}
```

198. House Robber

题目

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security system connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight **without alerting the police**.

Example 1:

```
Input: [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
             Total amount you can rob = 1 + 3 = 4.
```

Example 2:

```
Input: [2,7,9,3,1]
Output: 12
Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money =
1).
             Total amount you can rob = 2 + 9 + 1 = 12.
```

题目大意

你是一个专业的小偷，计划偷窃沿街的房屋。每间房内都藏有一定的现金，影响你偷窃的唯一制约因素就是相邻的房屋装有相互连通的防盗系统，**如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警。**

给定一个代表每个房屋存放金额的非负整数数组，计算你**在不触动警报装置的情况下**，能够偷窃到的最高金额。

解题思路

- 你是一个专业的小偷，打算洗劫一条街的所有房子。每个房子里面有不同价值的宝物，但是如果你选择偷窃连续的 2 栋房子，就会触发警报系统，编程求出你最多可以偷窃价值多少的宝物?
- 这一题可以用 DP 来解答，也可以用找规律的方法来解答。
- DP 的状态定义是：`dp[i]` 代表抢 `nums[0,i]` 这个区间内房子的最大值，状态转移方程是 `dp[i] = max(dp[i-1], nums[i]+dp[i-2])` 。可以优化迭代的过程，用两个临时变量来存储中间结果，以节约辅助空间。

代码

```go
package leetcode

// 解法一 DP
func rob198(nums []int) int {
  n := len(nums)
  if n == 0 {
    return 0
  }
  if n == 1 {
    return nums[0]
  }
  // dp[i] 代表抢 nums[0...i] 房子的最大价值
  dp := make([]int, n)
  dp[0], dp[1] = nums[0], max(nums[1], nums[0])
  for i := 2; i < n; i++ {
    dp[i] = max(dp[i-1], nums[i]+dp[i-2])
  }
  return dp[n-1]
}

// 解法二 DP 优化辅助空间，把迭代的值保存在 2 个变量中
func rob198_1(nums []int) int {
  n := len(nums)
  if n == 0 {
    return 0
  }
  curMax, preMax := 0, 0
  for i := 0; i < n; i++ {
    tmp := curMax
    curMax = max(curMax, nums[i]+preMax)
    preMax = tmp
  }
  return curMax
}

// 解法三 模拟
func rob(nums []int) int {
  // a 对于偶数位上的最大值的记录
  // b 对于奇数位上的最大值的记录
  a, b := 0, 0
  for i := 0; i < len(nums); i++ {
    if i%2 == 0 {
      a = max(a+nums[i], b)
    } else {
      b = max(a, b+nums[i])
    }
  }
  return max(a, b)
}
```

199. Binary Tree Right Side View

题目

Given a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom.

Example:

```
Input: [1,2,3,null,5,null,4]
Output: [1, 3, 4]
Explanation:

   1            <---
 /   \
2     3         <---
 \     \
  5     4       <---
```

题目大意

从右边看一个树，输出看到的数字。注意有遮挡。

解题思路

- 这一题是按层序遍历的变种题。按照层序把每层的元素都遍历出来，然后依次取每一层的最右边的元素即可。用一个队列即可实现。
- 第 102 题和第 107 题都是按层序遍历的。

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
```

```go
 * }
 */
func rightSideView(root *TreeNode) []int {
  res := []int{}
  if root == nil {
    return res
  }
  queue := []*TreeNode{root}
  for len(queue) > 0 {
    n := len(queue)
    for i := 0; i < n; i++ {
      if queue[i].Left != nil {
        queue = append(queue, queue[i].Left)
      }
      if queue[i].Right != nil {
        queue = append(queue, queue[i].Right)
      }
    }
    res = append(res, queue[n-1].Val)
    queue = queue[n:]
  }
  return res
}
```

200. Number of Islands

题目

Given a 2d grid map of `'1'`s (land) and `'0'`s (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

```
Input:
11110
11010
11000
00000

Output: 1
```

Example 2:

```
Input:
11000
11000
00100
00011

Output: 3
```

题目大意

给定一个由 '1'（陆地）和 '0'（水）组成的的二维网格，计算岛屿的数量。一个岛被水包围，并且它是通过水平方向或垂直方向上相邻的陆地连接而成的。你可以假设网格的四个边均被水包围。

解题思路

- 要求找出地图中的孤岛。孤岛的含义是四周被海水包围的岛。
- 这一题可以按照第 79 题的思路进行搜索，只要找到为 "1" 的岛以后，从这里开始搜索这周连通的陆地，也都标识上访问过。每次遇到新的 "1" 且没有访问过，就相当于遇到了新的岛屿了。

代码

```go
package leetcode

func numIslands(grid [][]byte) int {
  m := len(grid)
  if m == 0 {
    return 0
  }
  n := len(grid[0])
  if n == 0 {
    return 0
  }
  res, visited := 0, make([][]bool, m)
  for i := 0; i < m; i++ {
    visited[i] = make([]bool, n)
  }
  for i := 0; i < m; i++ {
    for j := 0; j < n; j++ {
      if grid[i][j] == '1' && !visited[i][j] {
        searchIslands(grid, &visited, i, j)
        res++
      }
    }
  }
  return res
```

```go
}

func searchIslands(grid [][]byte, visited *[][]bool, x, y int) {
  (*visited)[x][y] = true
  for i := 0; i < 4; i++ {
    nx := x + dir[i][0]
    ny := y + dir[i][1]
    if isInBoard(grid, nx, ny) && !(*visited)[nx][ny] && grid[nx][ny] == '1' {
      searchIslands(grid, visited, nx, ny)
    }
  }
}
```

201. Bitwise AND of Numbers Range

题目

Given a range [m, n] where 0 <= m <= n <= 2147483647, return the bitwise AND of all numbers in this range, inclusive.

Example 1:

```
Input: [5,7]
Output: 4
```

Example 2:

```
Input: [0,1]
Output: 0
```

题目大意

给定范围 [m, n]，其中 0 <= m <= n <= 2147483647，返回此范围内所有数字的按位与（包含 m, n 两端点）。

解题思路

- 这一题要求输出 [m,n] 区间内所有数的 AND 与操作之后的结果。
- 举个例子，假设区间是 [26,30]，那么这个区间内的数用二进制表示出来为：

  11010
  11011
  11100
  11101
  11110

- 可以观察到，把这些数都 AND 起来，只要有 0 的位，最终结果都是 0，所以需要从右往前找到某一位上不为 0 的。不断的右移左边界和右边界，把右边的 0 都移走，直到它们俩相等，就找到了某一位上开始都不为 0 的了。在右移的过程中记录下右移了多少位，最后把 m 或者 n 的右边添上 0 即可。按照上面这个例子来看，11000 是最终的结果。
- 这一题还有解法二，还是以 [26,30] 这个区间为例。这个区间内的数末尾 3 位不断的 0，1 变化着。那么如果把末尾的 1 都打掉，就是最终要求的结果了。当 n == m 或者 n < m 的时候就退出循环，说明后面不同的位数已经都被抹平了，1 都被打掉为 0 了。所以关键的操作为 `n &= (n - 1)` ，清除最低位的 1 。这个算法名叫 `Brian Kernighan` 算法。

代码

```
package leetcode

// 解法一
func rangeBitwiseAnd1(m int, n int) int {
  if m == 0 {
    return 0
  }
  moved := 0
  for m != n {
    m >>= 1
    n >>= 1
    moved++
  }
  return m << uint32(moved)
}

// 解法二 Brian Kernighan's algorithm
func rangeBitwiseAnd(m int, n int) int {
  for n > m {
    n &= (n - 1) // 清除最低位的 1
  }
  return n
}
```

202. Happy Number

题目

Write an algorithm to determine if a number is "happy".

A happy number is a number defined by the following process: Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1. Those numbers for which this process ends in 1 are happy numbers.

Example 1:

```
Input: 19
Output: true
Explanation:
12 + 92 = 82
82 + 22 = 68
62 + 82 = 100
12 + 02 + 02 = 1
```

题目大意

判断一个数字是否是“快乐数字”，“快乐数字”的定义是，不断的把这个数字的每个数字的平方和加起来，反复的加，最终如果能有结果是 1，则是“快乐数字”，如果不能得到一，出现了循环，则输出 false。

解题思路

按照题意要求做即可。

代码

```go
package leetcode

func isHappy(n int) bool {
  if n == 0 {
    return false
  }
  res := 0
  num := n
  record := map[int]int{}
  for {
    for num != 0 {
      res += (num % 10) * (num % 10)
      num = num / 10
    }
    if _, ok := record[res]; !ok {
      if res == 1 {
        return true
      }
      record[res] = res
      num = res
      res = 0
      continue
```

```
    } else {
      return false
    }
  }
}
```

203. Remove Linked List Elements

题目

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
```

题目大意

删除链表中所有指定值的结点。

解题思路

按照题意做即可。

代码

```
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */
func removeElements(head *ListNode, val int) *ListNode {
  if head == nil {
    return head
  }
  newHead := &ListNode{Val: 0, Next: head}
  pre := newHead
  cur := head
  for cur != nil {
```

```
    if cur.Val == val {
      pre.Next = cur.Next
    } else {
      pre = cur
    }
    cur = cur.Next
  }
  return newHead.Next
}
```

204. Count Primes

题目

Count the number of prime numbers less than a non-negative number, **n**.

Example:

```
Input: 10
Output: 4
Explanation: There are 4 prime numbers less than 10, they are 2, 3, 5, 7.
```

题目大意

统计所有小于非负整数 n 的质数的数量。

解题思路

- 给出一个数字 n，要求输出小于 n 的所有素数的个数总和。简单题。

代码

```
package leetcode

func countPrimes(n int) int {
  isNotPrime := make([]bool, n)
  for i := 2; i*i < n; i++ {
    if isNotPrime[i] {
      continue
    }
    for j := i * i; j < n; j = j + i {
      isNotPrime[j] = true
    }
  }
  count := 0
  for i := 2; i < n; i++ {
```

```
    if !isNotPrime[i] {
      count++
    }
  }
  return count
}
```

205. Isomorphic Strings

题目

Given two strings s and t, determine if they are isomorphic.

Two strings are isomorphic if the characters in s can be replaced to get t.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character but a character may map to itself.

Example 1:

```
Input: s = "egg", t = "add"
Output: true
```

Example 2:

```
Input: s = "foo", t = "bar"
Output: false
```

Example 3:

```
Input: s = "paper", t = "title"
Output: true
```

Note:

You may assume both s and t have the same length.

题目大意

这道题和第 290 题基本是一样的。第 290 题是模式匹配，这道题的题意是字符串映射，实质是一样的。

给定一个初始字符串串，判断初始字符串是否可以通过字符映射的方式，映射到目标字符串，如果可以映射，则输出 true，反之输出 false。

解题思路

这道题做法和第 290 题基本一致。

代码

```
package leetcode

func isIsomorphic(s string, t string) bool {
  strList := []byte(t)
  patternByte := []byte(s)
  if (s == "" && t != "") || (len(patternByte) != len(strList)) {
    return false
  }

  pMap := map[byte]byte{}
  sMap := map[byte]byte{}
  for index, b := range patternByte {
    if _, ok := pMap[b]; !ok {
      if _, ok = sMap[strList[index]]; !ok {
        pMap[b] = strList[index]
        sMap[strList[index]] = b
      } else {
        if sMap[strList[index]] != b {
          return false
        }
      }
    } else {
      if pMap[b] != strList[index] {
        return false
      }
    }
  }
  return true
}
```

206. Reverse Linked List

题目

Reverse a singly linked list.

题目大意

翻转单链表

解题思路

按照题意做即可。

代码

```
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */

// ListNode define
type ListNode struct {
  Val  int
  Next *ListNode
}

func reverseList(head *ListNode) *ListNode {
  var behind *ListNode
  for head != nil {
    next := head.Next
    head.Next = behind
    behind = head
    head = next
  }
  return behind
}
```

207. Course Schedule

题目

There are a total of n courses you have to take, labeled from `0` to `n-1`.

Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: `[0,1]`

Given the total number of courses and a list of prerequisite **pairs**, is it possible for you to finish all courses?

Example 1:

```
Input: 2, [[1,0]]
Output: true
Explanation: There are a total of 2 courses to take.
             To take course 1 you should have finished course 0. So it is possible.
```

Example 2:

```
Input: 2, [[1,0],[0,1]]
Output: false
Explanation: There are a total of 2 courses to take.
             To take course 1 you should have finished course 0, and to take course 0
you should
             also have finished course 1. So it is impossible.
```

Note:

1. The input prerequisites is a graph represented by **a list of edges**, not adjacency matrices. Read more about [how a graph is represented](#).
2. You may assume that there are no duplicate edges in the input prerequisites.

题目大意

现在你总共有 n 门课需要选，记为 0 到 n-1。在选修某些课程之前需要一些先修课程。 例如，想要学习课程 0 ，你需要先完成课程 1 ，我们用一个匹配来表示他们: [0,1]。给定课程总量以及它们的先决条件，判断是否可能完成所有课程的学习？

解题思路

- 给出 n 个任务，每两个任务之间有相互依赖关系，比如 A 任务一定要在 B 任务之前完成才行。问是否可以完成所有任务。
- 这一题就是标准的 AOV 网的拓扑排序问题。拓扑排序问题的解决办法是主要是循环执行以下两步，直到不存在入度为0的顶点为止。
    - 1. 选择一个入度为0的顶点并输出之；
    - 2. 从网中删除此顶点及所有出边。

  循环结束后，若输出的顶点数小于网中的顶点数，则输出“有回路”信息，即无法完成所有任务；否则输出的顶点序列就是一种拓扑序列，即可以完成所有任务。

代码

```go
package leetcode

// AOV 网的拓扑排序
```

```go
func canFinish(n int, pre [][]int) bool {
  in := make([]int, n)
  frees := make([][]int, n)
  next := make([]int, 0, n)
  for _, v := range pre {
    in[v[0]]++
    frees[v[1]] = append(frees[v[1]], v[0])
  }
  for i := 0; i < n; i++ {
    if in[i] == 0 {
      next = append(next, i)
    }
  }
  for i := 0; i != len(next); i++ {
    c := next[i]
    v := frees[c]
    for _, vv := range v {
      in[vv]--
      if in[vv] == 0 {
        next = append(next, vv)
      }
    }
  }
  return len(next) == n
}
```

208. Implement Trie (Prefix Tree)

题目

Implement a trie with `insert`, `search`, and `startsWith` methods.

Example:

```
Trie trie = new Trie();

trie.insert("apple");
trie.search("apple");   // returns true
trie.search("app");     // returns false
trie.startsWith("app"); // returns true
trie.insert("app");
trie.search("app");     // returns true
```

Note:

- You may assume that all inputs are consist of lowercase letters `a-z`.
- All inputs are guaranteed to be non-empty strings.

题目大意

实现一个 Trie (前缀树)，包含 insert, search, 和 startsWith 这三个操作。

解题思路

- 要求实现一个 Trie 的数据结构，具有 `insert`, `search`, `startsWith` 三种操作
- 这一题就是经典的 Trie 实现。本题的实现可以作为 Trie 的模板。

代码

```go
package leetcode

type Trie struct {
  isWord   bool
  children map[rune]*Trie
}

/** Initialize your data structure here. */
func Constructor208() Trie {
  return Trie{isWord: false, children: make(map[rune]*Trie)}
}

/** Inserts a word into the trie. */
func (this *Trie) Insert(word string) {
  parent := this
  for _, ch := range word {
    if child, ok := parent.children[ch]; ok {
      parent = child
    } else {
      newChild := &Trie{children: make(map[rune]*Trie)}
      parent.children[ch] = newChild
      parent = newChild
    }
  }
  parent.isWord = true
}

/** Returns if the word is in the trie. */
func (this *Trie) Search(word string) bool {
  parent := this
  for _, ch := range word {
    if child, ok := parent.children[ch]; ok {
      parent = child
      continue
    }
    return false
```

```go
  }
  return parent.isWord
}

/** Returns if there is any word in the trie that starts with the given prefix. */
func (this *Trie) StartsWith(prefix string) bool {
  parent := this
  for _, ch := range prefix {
    if child, ok := parent.children[ch]; ok {
      parent = child
      continue
    }
    return false
  }
  return true
}

/**
 * Your Trie object will be instantiated and called as such:
 * obj := Constructor();
 * obj.Insert(word);
 * param_2 := obj.Search(word);
 * param_3 := obj.StartsWith(prefix);
 */
```

209. Minimum Size Subarray Sum

题目

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.
```

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).

题目大意

给定一个整型数组和一个数字 s，找到数组中最短的一个连续子数组，使得连续子数组的数字之和 sum>=s，返回最短的连续子数组的返回值。

解题思路

这一题的解题思路是用滑动窗口。在滑动窗口 [i,j]之间不断往后移动，如果总和小于 s，就扩大右边界 j，不断加入右边的值，直到 sum > s，之和再缩小 i 的左边界，不断缩小直到 sum < s，这时候右边界又可以往右移动。以此类推。

代码

```
package leetcode

func minSubArrayLen(target int, nums []int) int {
  left, sum, res := 0, 0, len(nums)+1
  for right, v := range nums {
    sum += v
    for sum >= target {
      res = min(res, right-left+1)
      sum -= nums[left]
      left++
    }
  }
  if res == len(nums)+1 {
    return 0
  }
  return res
}

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

210. Course Schedule II

题目

There are a total of *n* courses you have to take, labeled from `0` to `n-1`.

Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: `[0,1]`

Given the total number of courses and a list of prerequisite **pairs**, return the ordering of courses you should take to finish all courses.

There may be multiple correct orders, you just need to return one of them. If it is impossible to finish all courses, return an empty array.

Example 1:

```
Input: 2, [[1,0]]
Output: [0,1]
Explanation: There are a total of 2 courses to take. To take course 1 you should have finished
             course 0. So the correct course order is [0,1] .
```

Example 2:

```
Input: 4, [[1,0],[2,0],[3,1],[3,2]]
Output: [0,1,2,3] or [0,2,1,3]
Explanation: There are a total of 4 courses to take. To take course 3 you should have finished both
             courses 1 and 2. Both courses 1 and 2 should be taken after you finished course 0.
             So one correct course order is [0,1,2,3]. Another correct ordering is [0,2,1,3] .
```

Note:

1. The input prerequisites is a graph represented by **a list of edges**, not adjacency matrices. Read more about [how a graph is represented](#).
2. You may assume that there are no duplicate edges in the input prerequisites.

题目大意

现在你总共有 n 门课需要选，记为 0 到 n-1。在选修某些课程之前需要一些先修课程。 例如，想要学习课程 0 ，你需要先完成课程 1 ，我们用一个匹配来表示他们: [0,1]。给定课程总量以及它们的先决条件，返回你为了学完所有课程所安排的学习顺序。可能会有多个正确的顺序，你只要返回一种就可以了。如果不可能完成所有课程，返回一个空数组。

解题思路

- 给出 n 个任务，每两个任务之间有相互依赖关系，比如 A 任务一定要在 B 任务之前完成才行。问是否可以完成所有任务，如果可以完成任务，就输出完成任务的顺序，如果不能完成，输出空数组。
- 这一题是第 207 题的加强版。解题思路是 AOV 网的拓扑排序。最后输出数组即可。代码和第 207 题基本不变。具体解题思路见第 207 题。

代码

```
package leetcode
```

```go
func findOrder(numCourses int, prerequisites [][]int) []int {
  in := make([]int, numCourses)
  frees := make([][]int, numCourses)
  next := make([]int, 0, numCourses)
  for _, v := range prerequisites {
    in[v[0]]++
    frees[v[1]] = append(frees[v[1]], v[0])
  }
  for i := 0; i < numCourses; i++ {
    if in[i] == 0 {
      next = append(next, i)
    }
  }
  for i := 0; i != len(next); i++ {
    c := next[i]
    v := frees[c]
    for _, vv := range v {
      in[vv]--
      if in[vv] == 0 {
        next = append(next, vv)
      }
    }
  }
  if len(next) == numCourses {
    return next
  }
  return []int{}
}
```

211. Design Add and Search Words Data Structure

题目

Design a data structure that supports the following two operations:

```
void addWord(word)
bool search(word)
```

search(word) can search a literal word or a regular expression string containing only letters `a-z` or `.`. A `.` means it can represent any one letter.

Example:

```
addWord("bad")
addWord("dad")
addWord("mad")
search("pad") -> false
search("bad") -> true
search(".ad") -> true
search("b..") -> true
```

Note: You may assume that all words are consist of lowercase letters `a-z`.

题目大意

设计一个支持以下两种操作的数据结构：`void addWord(word)`、`bool search(word)`。`search(word)` 可以搜索文字或正则表达式字符串，字符串只包含字母 . 或 a-z 。 "." 可以表示任何一个字母。

解题思路

- 设计一个 `WordDictionary` 的数据结构，要求具有 `addWord(word)` 和 `search(word)` 的操作，并且具有模糊查找的功能。
- 这一题是第 208 题的加强版，在第 208 题经典的 Trie 上加上了模糊查找的功能。其他实现一模一样。

代码

```go
package leetcode

type WordDictionary struct {
  children map[rune]*WordDictionary
  isWord   bool
}

/** Initialize your data structure here. */
func Constructor211() WordDictionary {
  return WordDictionary{children: make(map[rune]*WordDictionary)}
}

/** Adds a word into the data structure. */
func (this *WordDictionary) AddWord(word string) {
  parent := this
  for _, ch := range word {
    if child, ok := parent.children[ch]; ok {
      parent = child
    } else {
      newChild := &WordDictionary{children: make(map[rune]*WordDictionary)}
      parent.children[ch] = newChild
      parent = newChild
```

```go
    }
  }
  parent.isWord = true
}

/** Returns if the word is in the data structure. A word could contain the dot
character '.' to represent any one letter. */
func (this *WordDictionary) Search(word string) bool {
  parent := this
  for i, ch := range word {
    if rune(ch) == '.' {
      isMatched := false
      for _, v := range parent.children {
        if v.Search(word[i+1:]) {
          isMatched = true
        }
      }
      return isMatched
    } else if _, ok := parent.children[rune(ch)]; !ok {
      return false
    }
    parent = parent.children[rune(ch)]
  }
  return len(parent.children) == 0 || parent.isWord
}

/**
 * Your WordDictionary object will be instantiated and called as such:
 * obj := Constructor();
 * obj.AddWord(word);
 * param_2 := obj.Search(word);
 */
```

212. Word Search II

题目

Given a 2D board and a list of words from the dictionary, find all words in the board.

Each word must be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once in a word.

Example:

```
Input:
board = [
  ['o','a','a','n'],
  ['e','t','a','e'],
  ['i','h','k','r'],
  ['i','f','l','v']
]
words = ["oath","pea","eat","rain"]

Output: ["eat","oath"]
```

Note:

1. All inputs are consist of lowercase letters `a-z`.
2. The values of `words` are distinct.

题目大意

给定一个二维网格 board 和一个字典中的单词列表 words，找出所有同时在二维网格和字典中出现的单词。

单词必须按照字母顺序，通过相邻的单元格内的字母构成，其中“相邻”单元格是那些水平相邻或垂直相邻的单元格。同一个单元格内的字母在一个单词中不允许被重复使用。

解题思路

- 这一题是第 79 题的加强版，在第 79 题的基础上增加了一个 word 数组，要求找出所有出现在地图中的单词。思路还是可以按照第 79 题 DFS 搜索，不过时间复杂度特别高！
- 想想更优的解法。

代码

```go
package leetcode

func findWords(board [][]byte, words []string) []string {
  res := []string{}
  for _, v := range words {
    if exist(board, v) {
      res = append(res, v)
    }
  }
  return res
}
```

213. House Robber II

题目

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed. All houses at this place are **arranged in a circle.** That means the first house is the neighbor of the last one. Meanwhile, adjacent houses have security system connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight **without alerting the police**.

Example 1:

```
Input: [2,3,2]
Output: 3
Explanation: You cannot rob house 1 (money = 2) and then rob house 3 (money = 2),
             because they are adjacent houses.
```

Example 2:

```
Input: [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
             Total amount you can rob = 1 + 3 = 4.
```

题目大意

你是一个专业的小偷，计划偷窃沿街的房屋，每间房内都藏有一定的现金。这个地方所有的房屋都**围成一圈**，这意味着第一个房屋和最后一个房屋是紧挨着的。同时，相邻的房屋装有相互连通的防盗系统，**如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警**。

给定一个代表每个房屋存放金额的非负整数数组，计算你**在不触动警报装置的情况下**，能够偷窃到的最高金额。

解题思路

- 这一题是第 198 题的加强版。不过这次是在一个环形的街道中，即最后一个元素和第一个元素是邻居，在不触碰警报的情况下，问能够窃取的财产的最大值是多少?
- 解题思路和第 198 完全一致，只需要增加额外的一个转换。由于首尾是相邻的，所以在取了第一个房子以后就不能取第 n 个房子，那么就在 [0,n - 1] 的区间内找出总价值最多的解，然后再 [1,n] 的区间内找出总价值最多的解，两者取最大值即可。

代码

```go
package leetcode

func rob213(nums []int) int {
  n := len(nums)
```

```go
  if n == 0 {
    return 0
  }
  if n == 1 {
    return nums[0]
  }
  if n == 2 {
    return max(nums[0], nums[1])
  }
  // 由于首尾是相邻的，所以需要对比 [0, n-1]、[1, n] 这两个区间的最大值
  return max(rob213_1(nums, 0, n-2), rob213_1(nums, 1, n-1))
}

func rob213_1(nums []int, start, end int) int {
  preMax := nums[start]
  curMax := max(preMax, nums[start+1])
  for i := start + 2; i <= end; i++ {
    tmp := curMax
    curMax = max(curMax, nums[i]+preMax)
    preMax = tmp
  }
  return curMax
}
```

215. Kth Largest Element in an Array

题目

Find the kth largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the kth distinct element.

Example 1:

```
Input: [3,2,1,5,6,4] and k = 2
Output: 5
```

Example 2:

```
Input: [3,2,3,1,2,4,5,5,6] and k = 4
Output: 4
```

Note:

You may assume k is always valid, 1 ≤ k ≤ array's length.

题目大意

找出数组中第 K 大的元素。这一题非常经典。可以用 O(n) 的时间复杂度实现。

解题思路

- 在快排的 partition 操作中，每次 partition 操作结束都会返回一个点，这个标定点的下标和最终排序之后有序数组中这个元素所在的下标是一致的。利用这个特性，我们可以不断的划分数组区间，最终找到第 K 大的元素。执行一次 partition 操作以后，如果这个元素的下标比 K 小，那么接着就在后边的区间继续执行 partition 操作；如果这个元素的下标比 K 大，那么就在左边的区间继续执行 partition 操作；如果相等就直接输出这个下标对应的数组元素即可。
- 快排的思路实现的算法时间复杂度为 O(n)，空间复杂度为 O(logn)。由于证明过程很繁琐，所以不再这里展开讲。具体证明可以参考《算法导论》第 9 章第 2 小节。

代码

```
package leetcode

import (
  "math/rand"
  "sort"
)

// 解法一 排序，排序的方法反而速度是最快的
func findKthLargest1(nums []int, k int) int {
  sort.Ints(nums)
  return nums[len(nums)-k]
}

// 解法二 这个方法的理论依据是 partition 得到的点的下标就是最终排序之后的下标，根据这个下标，我们可以判断第 K 大的数在哪里
// 时间复杂度 O(n)，空间复杂度 O(log n)，最坏时间复杂度为 O(n^2)，空间复杂度 O(n)
func findKthLargest(nums []int, k int) int {
  m := len(nums) - k + 1 // mth smallest, from 1..len(nums)
  return selectSmallest(nums, 0, len(nums)-1, m)
}

func selectSmallest(nums []int, l, r, i int) int {
  if l >= r {
    return nums[l]
  }
  q := partition(nums, l, r)
  k := q - l + 1
  if k == i {
    return nums[q]
  }
```

```go
  if i < k {
    return selectSmallest(nums, l, q-1, i)
  } else {
    return selectSmallest(nums, q+1, r, i-k)
  }
}

func partition(nums []int, l, r int) int {
  k := l + rand.Intn(r-l+1) // 此处为优化，使得时间复杂度期望降为 O(n)，最坏时间复杂度为 O(n^2)
  nums[k], nums[r] = nums[r], nums[k]
  i := l - 1
  // nums[l..i] <= nums[r]
  // nums[i+1..j-1] > nums[r]
  for j := l; j < r; j++ {
    if nums[j] <= nums[r] {
      i++
      nums[i], nums[j] = nums[j], nums[i]
    }
  }
  nums[i+1], nums[r] = nums[r], nums[i+1]
  return i + 1
}

// 扩展题 剑指 Offer 40. 最小的 k 个数
func getLeastNumbers(arr []int, k int) []int {
  return selectSmallest1(arr, 0, len(arr)-1, k)[:k]
}

// 和 selectSmallest 实现完全一致，只是返回值不用再截取了，直接返回 nums 即可
func selectSmallest1(nums []int, l, r, i int) []int {
  if l >= r {
    return nums
  }
  q := partition(nums, l, r)
  k := q - l + 1
  if k == i {
    return nums
  }
  if i < k {
    return selectSmallest1(nums, l, q-1, i)
  } else {
    return selectSmallest1(nums, q+1, r, i-k)
  }
}
```

216. Combination Sum III

题目

Find all possible combinations of **k** numbers that add up to a number **n**, given that only numbers from 1 to 9 can be used and each combination should be a unique set of numbers.

Note:

- All numbers will be positive integers.
- The solution set must not contain duplicate combinations.

Example 1:

```
Input: k = 3, n = 7
Output: [[1,2,4]]
```

Example 2:

```
Input: k = 3, n = 9
Output: [[1,2,6], [1,3,5], [2,3,4]]
```

题目大意

找出所有相加之和为 n 的 k 个数的组合。组合中只允许含有 1 - 9 的正整数，并且每种组合中不存在重复的数字。

说明：

- 所有数字都是正整数。
- 解集不能包含重复的组合。

解题思路

- 这一题比第 39 题还要简单一些，在第 39 题上稍加改动就可以解出这一道题。
- 第 39 题是给出数组，这一道题数组是固定死的 [1,2,3,4,5,6,7,8,9]，并且数字不能重复使用。

代码

```go
package leetcode

func combinationSum3(k int, n int) [][]int {
  if k == 0 {
    return [][]int{}
  }
  c, res := []int{}, [][]int{}
  findcombinationSum3(k, n, 1, c, &res)
  return res
}
```

```go
func findcombinationSum3(k, target, index int, c []int, res *[][]int) {
  if target == 0 {
    if len(c) == k {
      b := make([]int, len(c))
      copy(b, c)
      *res = append(*res, b)
    }
    return
  }
  for i := index; i < 10; i++ {
    if target >= i {
      c = append(c, i)
      findcombinationSum3(k, target-i, i+1, c, res)
      c = c[:len(c)-1]
    }
  }
}
```

217. Contains Duplicate

题目

Given an array of integers, find if the array contains any duplicates.

Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.

Example 1:

```
Input: [1,2,3,1]
Output: true
```

Example 2:

```
Input: [1,2,3,4]
Output: false
```

Example 3:

```
Input: [1,1,1,3,3,4,3,2,4,2]
Output: true
```

题目大意

这是一道简单题，如果数组里面有重复数字就输出 true，否则输出 flase。

解题思路

用 map 判断即可。

代码

```
package leetcode

func containsDuplicate(nums []int) bool {
  record := make(map[int]bool, len(nums))
  for _, n := range nums {
    if _, found := record[n]; found {
      return true
    }
    record[n] = true
  }
  return false
}
```

218. The Skyline Problem

题目

A city's skyline is the outer contour of the silhouette formed by all the buildings in that city when viewed from a distance. Now suppose you are **given the locations and height of all the buildings** as shown on a cityscape photo (Figure A), write a program to **output the skyline** formed by these buildings collectively (Figure B).

A
15
10
5
0
0
5
10
15
20
25

The geometric information of each building is represented by a triplet of integers `[Li, Ri, Hi]`, where `Li` and `Ri` are the x coordinates of the left and right edge of the ith building, respectively, and `Hi` is its height. It is guaranteed that `0 ≤ Li, Ri ≤ INT_MAX`, `0 < Hi ≤ INT_MAX`, and `Ri - Li > 0`. You may assume all buildings are perfect rectangles grounded on an absolutely flat surface at height 0.

For instance, the dimensions of all buildings in Figure A are recorded as: `[ [2 9 10], [3 7 15], [5 12 12], [15 20 10], [19 24 8] ]` .

The output is a list of "**key points**" (red dots in Figure B) in the format of `[ [x1,y1], [x2, y2], [x3, y3], ... ]` that uniquely defines a skyline. **A key point is the left endpoint of a horizontal line segment**. Note that the last key point, where the rightmost building ends, is merely used to mark the termination of the skyline, and always has zero height. Also, the ground in between any two adjacent buildings should be considered part of the skyline contour.

For instance, the skyline in Figure B should be represented as: `[ [2 10], [3 15], [7 12], [12 0], [15 10], [20 8], [24, 0] ]`.

Notes:

- The number of buildings in any input list is guaranteed to be in the range `[0, 10000]`.
- The input list is already sorted in ascending order by the left x position `Li`.

- The output list must be sorted by the x position.
- There must be no consecutive horizontal lines of equal height in the output skyline. For instance, `[...[2 3], [4 5], [7 5], [11 5], [12 7]...]` is not acceptable; the three lines of height 5 should be merged into one in the final output as such: `[...[2 3], [4 5], [12 7], ...]`

题目大意

城市的天际线是从远处观看该城市中所有建筑物形成的轮廓的外部轮廓。现在，假设您获得了城市风光照片（图A）上显示的所有建筑物的位置和高度，请编写一个程序以输出由这些建筑物形成的天际线（图B）。

每个建筑物的几何信息用三元组 [Li，Ri，Hi] 表示，其中 Li 和 Ri 分别是第 i 座建筑物左右边缘的 x 坐标，Hi 是其高度。可以保证 0 ≤ Li, Ri ≤ INT_MAX, 0 < Hi ≤ INT_MAX 和 Ri - Li > 0。您可以假设所有建筑物都是在绝对平坦且高度为 0 的表面上的完美矩形。

例如，图 A 中所有建筑物的尺寸记录为：[[2 9 10], [3 7 15], [5 12 12], [15 20 10], [19 24 8]] 。

输出是以 [[x1,y1], [x2, y2], [x3, y3], ...] 格式的“关键点”（图 B 中的红点）的列表，它们唯一地定义了天际线。关键点是水平线段的左端点。请注意，最右侧建筑物的最后一个关键点仅用于标记天际线的终点，并始终为零高度。此外，任何两个相邻建筑物之间的地面都应被视为天际线轮廓的一部分。

例如，图 B 中的天际线应该表示为：[[2 10], [3 15], [7 12], [12 0], [15 10], [20 8], [24, 0]]。

说明:

- 任何输入列表中的建筑物数量保证在 [0, 10000] 范围内。
- 输入列表已经按左 x 坐标 Li 进行升序排列。
- 输出列表必须按 x 位排序。
- 输出天际线中不得有连续的相同高度的水平线。例如 [...[2 3], [4 5], [7 5], [11 5], [12 7]...] 是不正确的答案；三条高度为 5 的线应该在最终输出中合并为一个：[...[2 3], [4 5], [12 7], ...]

解题思路

- 给出一个二维数组，每个子数组里面代表一个高楼的信息，一个高楼的信息包含 3 个信息，高楼起始坐标，高楼终止坐标，高楼高度。要求找到这些高楼的边际点，并输出这些边际点的高度信息。
- 这一题可以用树状数组来解答。要求天际线，即找到楼与楼重叠区间外边缘的线，说白了是维护各个区间内的最值。这有 2 个需要解决的问题。
    1. 如何维护最值。当一个高楼的右边界消失，剩下的各个小楼间还需要选出最大值作为天际线。剩下重重叠叠的小楼很多，树状数组如何维护区间最值是解决此类题的关键。
    2. 如何维护天际线的转折点。有些楼与楼并非完全重叠，重叠一半的情况导致天际线出现转折点。如上图中标记的红色转折点。树状数组如何维护这些点呢?
- 先解决第一个问题(维护最值)。树状数组只有 2 个操作，一个是 Add() 一个是 Query()。从上面关于这 2 个操作的讲解中可以知道这 2 个操作都不能满足我们的需求。Add() 操作可以改成维护区间内 max() 的操作。但是 max() 容易获得却很难“去除”。如上图 [3,7] 这个区间内的最大值是 15。根据树状数组的定义，[3,12] 这个区间内最值还是 15。观察上图可以看到 [5,12] 区间内最值其实是 12。树状数组如何维护这种最值呢？最大值既然难以“去除”，那么需要考虑如何让最大值“来的晚一点”。解决办法是将 Query() 操作含义从前缀含义改成后缀含义。Query(i) 查询区间是 [1,i]，现在查询区间变成 {{< katex >}}[i,+\infty){{< /katex >}}。例如：[i,j] 区间内最值是 {{< katex >}}max*{i...j}{{< /katex >}}，Query(j+1) 的结果不会包含 {{< katex >}}max*{i...j}{{< /katex >}}，因为它查询的区间是 {{< katex >}}[j+1,+\infty){{< /katex >}}。这样更改以后，可以有效避免前驱高楼对后面楼的累积 max() 最值的影响。具体做法，将 x 轴上的各个区间排序，按照 x 值大小从小到大排序。从左

往右依次遍历各个区间。Add() 操作含义是加入每个区间右边界代表后缀区间的最值。这样不需要考虑“移除”最值的问题了。细心的读者可能又有疑问了：能否从右往左遍历区间，Query() 的含义继续延续前缀区间？这样做是可行的，解决第一个问题(维护最值)是可以的。但是这种处理办法解决第二个问题(维护转折点)会遇到麻烦。

- 再解决第二个问题(维护转折点)。如果用前缀含义的 Query()，在单点 i 上除了考虑以这个点为结束点的区间，还需要考虑以这个单点 i 为起点的区间。如果是后缀含义的 Query() 就没有这个问题了，{{< katex >}} [i+1,+\infty){{< /katex >}} 这个区间内不用考虑以单点 i 为结束点的区间。此题用树状数组代码实现见下面解法一。
- 这一题也可以用线段树来解。用线段树来解答，可以不用关心“楼挡住楼”的情况。由于楼的坐标是离散的，所以先把楼在 X 轴上两个坐标离散化。同第 699 题一样，楼的宽度是一个区间，但是离散的过程中，楼的宽度右边界需要减一，不然查询一个区间会包含两个点，导致错误的结果，例如，第一个楼是 [1,3)，楼高 10，第二个楼是 [3,6)，楼高 20 。第一个楼如果算上右边界 3，查询 [1,3] 的结果是 20，因为 [3,3] 这个点会查询到第二个楼上面去。所以每个楼的右边界应该减一。但是每个楼的右边界也要加入到 query 中，因为最终 query 的结果需要包含这些边界。将离散的数据排序以后，按照楼的信息，每个区间依次 update。最后统计的时候依次统计每个区间，如果当前区间的高度和前一个区间的高度一样，就算是等高的楼。当高度与前一个高度不相同的时候就算是天际线的边缘，就要添加到最后输出数组中。
- 类似的线段树的题目有：第 715 题，第 732 题，第 699 题。第 715 题是区间更新定值(**不是增减**)，第 218 题可以用扫描线，第 732 题和第 699 题类似，也是俄罗斯方块的题目，但是第 732 题的俄罗斯方块的方块会“断裂”。
- 这一题用线段树做时间复杂度有点高，可以用扫描线解题。扫描线的思路很简单，用一根根垂直于 X 轴的竖线，从最左边依次扫到最右边，扫描每一条大楼的边界，当进入大楼的左边界的时候，如果没有比这个左边界最高点更高的点，就记录下这个最高点 keyPoint，状态是进入状态。如果扫到一个大楼的左边界，有比它更高的高度，就不记录，因为它不是天际线，它被楼挡楼，挡在其他楼后面了。当扫到一个大楼的右边界的时候，如果是最高点，那么记录下它的状态是离开状态，此时还需要记录下第二高的点。在扫描线扫描的过程中，动态的维护大楼的高度，同时维护最高的高度和第二高的高度。其实只需要维护最高的高度这一个高度，因为当离开状态到来的时候，移除掉当前最高的，剩下的高度里面最高的就是第二高的高度。描述的伪代码如下：

```
// 扫描线伪代码
events = {{x: L , height: H , type: entering},
      {x: R , height: H , type: leaving}}
event.SortByX()
ds = new DS()

for e in events:
  if entering(e):
    if e.height > ds.max(): ans += [e.height]
    ds.add(e.height)
  if leaving(e):
    ds.remove(e.height)
    if e.height > ds.max(): ans += [ds.max()]
```

- 动态插入，查找最大值可以选用的数据结构有，最大堆和二叉搜索树。最大堆找最大值 O(1)，插入 O(log n)，但是 remove_by_key 需要 O(n) 的时间复杂度，并且需要自己实现。二叉搜索树，查找 max，添加和删除元素都是 O(log n) 的时间复杂度。

- 排序的时候也需要注意几个问题：如果大楼的边界相等，并且是进入状态，那么再按照高度从大到小进行排序；如果大楼的边界相等，并且是离开状态，那么高度按照从小到大进行排序。

代码

```go
package leetcode

import (
  "sort"

  "github.com/halfrost/LeetCode-Go/template"
)

// 解法一 树状数组，时间复杂度 O(n log n)
const LEFTSIDE = 1
const RIGHTSIDE = 2

type Point struct {
  xAxis int
  side  int
  index int
}

func getSkyline(buildings [][]int) [][]int {
  res := [][]int{}
  if len(buildings) == 0 {
    return res
  }
  allPoints, bit := make([]Point, 0), BinaryIndexedTree{}
  // [x-axis (value), [1 (left) | 2 (right)], index (building number)]
  for i, b := range buildings {
    allPoints = append(allPoints, Point{xAxis: b[0], side: LEFTSIDE, index: i})
    allPoints = append(allPoints, Point{xAxis: b[1], side: RIGHTSIDE, index: i})
  }
  sort.Slice(allPoints, func(i, j int) bool {
    if allPoints[i].xAxis == allPoints[j].xAxis {
      return allPoints[i].side < allPoints[j].side
    }
    return allPoints[i].xAxis < allPoints[j].xAxis
  })
  bit.Init(len(allPoints))
  kth := make(map[Point]int)
  for i := 0; i < len(allPoints); i++ {
    kth[allPoints[i]] = i
  }
  for i := 0; i < len(allPoints); i++ {
    pt := allPoints[i]
    if pt.side == LEFTSIDE {
```

```go
      bit.Add(kth[Point{xAxis: buildings[pt.index][1], side: RIGHTSIDE, index:
pt.index}], buildings[pt.index][2])
    }
    currHeight := bit.Query(kth[pt] + 1)
    if len(res) == 0 || res[len(res)-1][1] != currHeight {
      if len(res) > 0 && res[len(res)-1][0] == pt.xAxis {
        res[len(res)-1][1] = currHeight
      } else {
        res = append(res, []int{pt.xAxis, currHeight})
      }
    }
  }
  return res
}

type BinaryIndexedTree struct {
  tree     []int
  capacity int
}

// Init define
func (bit *BinaryIndexedTree) Init(capacity int) {
  bit.tree, bit.capacity = make([]int, capacity+1), capacity
}

// Add define
func (bit *BinaryIndexedTree) Add(index int, val int) {
  for ; index > 0; index -= index & -index {
    bit.tree[index] = max(bit.tree[index], val)
  }
}

// Query define
func (bit *BinaryIndexedTree) Query(index int) int {
  sum := 0
  for ; index <= bit.capacity; index += index & -index {
    sum = max(sum, bit.tree[index])
  }
  return sum
}

// 解法三 线段树 Segment Tree, 时间复杂度 O(n log n)
func getSkyline1(buildings [][]int) [][]int {
  st, ans, lastHeight, check := template.SegmentTree{}, [][]int{}, 0, false
  posMap, pos := discretization218(buildings)
  tmp := make([]int, len(posMap))
  st.Init(tmp, func(i, j int) int {
    return max(i, j)
  })
```

```go
  for _, b := range buildings {
    st.UpdateLazy(posMap[b[0]], posMap[b[1]-1], b[2])
  }
  for i := 0; i < len(pos); i++ {
    h := st.QueryLazy(posMap[pos[i]], posMap[pos[i]])
    if check == false && h != 0 {
      ans = append(ans, []int{pos[i], h})
      check = true
    } else if i > 0 && h != lastHeight {
      ans = append(ans, []int{pos[i], h})
    }
    lastHeight = h
  }
  return ans
}

func discretization218(positions [][]int) (map[int]int, []int) {
  tmpMap, posArray, posMap := map[int]int{}, []int{}, map[int]int{}
  for _, pos := range positions {
    tmpMap[pos[0]]++
    tmpMap[pos[1]-1]++
    tmpMap[pos[1]]++
  }
  for k := range tmpMap {
    posArray = append(posArray, k)
  }
  sort.Ints(posArray)
  for i, pos := range posArray {
    posMap[pos] = i
  }
  return posMap, posArray
}

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

// 解法三 扫描线 Sweep Line, 时间复杂度 O(n log n)
func getSkyline2(buildings [][]int) [][]int {
  size := len(buildings)
  es := make([]E, 0)
  for i, b := range buildings {
    l := b[0]
    r := b[1]
    h := b[2]
    // 1-- enter
```

```go
    el := NewE(i, l, h, 0)
    es = append(es, el)
    // 0 -- leave
    er := NewE(i, r, h, 1)
    es = append(es, er)
  }
  skyline := make([][]int, 0)
  sort.Slice(es, func(i, j int) bool {
    if es[i].X == es[j].X {
      if es[i].T == es[j].T {
        if es[i].T == 0 {
          return es[i].H > es[j].H
        }
        return es[i].H < es[j].H
      }
      return es[i].T < es[j].T
    }
    return es[i].X < es[j].X
  })
  pq := NewIndexMaxPQ(size)
  for _, e := range es {
    curH := pq.Front()
    if e.T == 0 {
      if e.H > curH {
        skyline = append(skyline, []int{e.X, e.H})
      }
      pq.Enque(e.N, e.H)
    } else {
      pq.Remove(e.N)
      h := pq.Front()
      if curH > h {
        skyline = append(skyline, []int{e.X, h})
      }
    }
  }
  return skyline
}

// 扫面线伪代码
// events = {{x: L , height: H , type: entering},
//      {x: R , height: H , type: leaving}}
// event.SortByX()
// ds = new DS()

// for e in events:
//  if entering(e):
//    if e.height > ds.max(): ans += [e.height]
//    ds.add(e.height)
//  if leaving(e):
```

```go
//    ds.remove(e.height)
//    if e.height > ds.max(): ans += [ds.max()]

// E define
type E struct { // 定义一个 event 事件
  N int // number 编号
  X int // x 坐标
  H int // height 高度
  T int // type  0-进入 1-离开
}

// NewE define
func NewE(n, x, h, t int) E {
  return E{
    N: n,
    X: x,
    H: h,
    T: t,
  }
}

// IndexMaxPQ define
type IndexMaxPQ struct {
  items []int
  pq    []int
  qp    []int
  total int
}

// NewIndexMaxPQ define
func NewIndexMaxPQ(n int) IndexMaxPQ {
  qp := make([]int, n)
  for i := 0; i < n; i++ {
    qp[i] = -1
  }
  return IndexMaxPQ{
    items: make([]int, n),
    pq:    make([]int, n+1),
    qp:    qp,
  }
}

// Enque define
func (q *IndexMaxPQ) Enque(key, val int) {
  q.total++
  q.items[key] = val
  q.pq[q.total] = key
  q.qp[key] = q.total
  q.swim(q.total)
```

```go
}

// Front define
func (q *IndexMaxPQ) Front() int {
  if q.total < 1 {
    return 0
  }
  return q.items[q.pq[1]]
}

// Remove define
func (q *IndexMaxPQ) Remove(key int) {
  rank := q.qp[key]
  q.exch(rank, q.total)
  q.total--
  q.qp[key] = -1
  q.sink(rank)
}

func (q *IndexMaxPQ) sink(n int) {
  for 2*n <= q.total {
    k := 2 * n
    if k < q.total && q.less(k, k+1) {
      k++
    }
    if q.less(k, n) {
      break
    }
    q.exch(k, n)
    n = k
  }
}

func (q *IndexMaxPQ) swim(n int) {
  for n > 1 {
    k := n / 2
    if q.less(n, k) {
      break
    }
    q.exch(n, k)
    n = k
  }
}

func (q *IndexMaxPQ) exch(i, j int) {
  q.pq[i], q.pq[j] = q.pq[j], q.pq[i]
  q.qp[q.pq[i]] = i
  q.qp[q.pq[j]] = j
}
```

```go
func (q *IndexMaxPQ) less(i, j int) bool {
  return q.items[q.pq[i]] < q.items[q.pq[j]]
}
```

219. Contains Duplicate II

题目

Given an array of integers and an integer k, find out whether there are two distinct indices i and j in the array such that nums[i] = nums[j] and the absolute difference between i and j is at most k.

Example 1:

```
Input: nums = [1,2,3,1], k = 3
Output: true
```

Example 2:

```
Input: nums = [1,0,1,1], k = 1
Output: true
```

Example 3:

```
Input: nums = [1,2,3,1,2,3], k = 2
Output: false
```

题目大意

这是一道简单题，如果数组里面有重复数字，并且重复数字的下标差值小于等于 K 就输出 true，如果没有重复数字或者下标差值超过了 K ，则输出 flase。

解题思路

这道题可以维护一个只有 K 个元素的 map，每次只需要判断这个 map 里面是否存在这个元素即可。如果存在就代表重复数字的下标差值在 K 以内。map 的长度如果超过了 K 以后就删除掉 i-k 的那个元素，这样一直维护 map 里面只有 K 个元素。

代码

```go
package leetcode

func containsNearbyDuplicate(nums []int, k int) bool {
  if len(nums) <= 1 {
    return false
  }
  if k <= 0 {
    return false
  }
  record := make(map[int]bool, len(nums))
  for i, n := range nums {
    if _, found := record[n]; found {
      return true
    }
    record[n] = true
    if len(record) == k+1 {
      delete(record, nums[i-k])
    }
  }
  return false
}
```

220. Contains Duplicate III

题目

Given an array of integers, find out whether there are two distinct indices i and j in the array such that the **absolute** difference between **nums[i]** and **nums[j]** is at most t and the **absolute** difference between i and j is at most k.

Example 1:

```
Input: nums = [1,2,3,1], k = 3, t = 0
Output: true
```

Example 2:

```
Input: nums = [1,0,1,1], k = 1, t = 2
Output: true
```

Example 3:

```
Input: nums = [1,5,9,1,5,9], k = 2, t = 3
Output: false
```

题目大意

给出一个数组 num，再给 K 和 t。问在 num 中能否找到一组 i 和 j，使得 num[i] 和 num[j] 的绝对差值最大为 t，并且 i 和 j 之前的绝对差值最大为 k。

解题思路

- 给出一个数组，要求在数组里面找到 2 个索引，`i` 和 `j`，使得 `| nums[i] - nums[j] | ≤ t`，并且 `| i - j | ≤ k`。
- 这是一道滑动窗口的题目。第一想法就是用 `i` 和 `j` 两个指针，针对每个 `i`，都从 `i + 1` 往后扫完整个数组，判断每个 `i` 和 `j`，判断是否满足题意。`j` 在循环的过程中注意判断剪枝条件 `| i - j | ≤ k`。这个做法的时间复杂度是 O(n^2)。这个做法慢的原因在于滑动窗口的左边界和右边界在滑动过程中不是联动滑动的。
- 于是考虑，如果数组是有序的呢？把数组按照元素值从小到大进行排序，如果元素值相等，就按照 index 从小到大进行排序。在这样有序的数组中找满足题意的 `i` 和 `j`，滑动窗口左边界和右边界就是联动的了。窗口的右边界滑到与左边界元素值的差值 ≤ t 的地方，满足了这个条件再判断 `| i - j | ≤ k`，如果右边界与左边界元素值的差值 > t 了，说明该把左边界往右移动了(能这样移动的原因就是因为我们将数组元素大小排序了，右移是增大元素的方向)。移动左边界的时候需要注意左边界不能超过右边界。这样滑动窗口一次滑过整个排序后的数组，就可以判断是否存在满足题意的 `i` 和 `j`。这个做法的时间主要花在排序上了，时间复杂度是 O(n log n)。
- 本题最优解是利用桶排序的思想。`| i - j | ≤ k` 这个条件利用一个窗口大小为 k 来维护。重点在 `| nums[i] - nums[j] | ≤ t` 这个条件如何满足。利用桶排序的思想，将 `nums[i]` 所有元素分为 ...，`[0,t]`，`[t+1,2t+1]`，...。每个区间的大小为 `t + 1`。每个元素现在都对应一个桶编号。进行 3 次查找即可确定能否找到满足这个 `| nums[i] - nums[j] | ≤ t` 条件的数对。如果在相同的桶中找到了元素，那么说明能找到这样的 i 和 j。还有 2 种可能对应桶边界的情况。如果存在前一个桶中的元素能使得相差的值也 `≤ t`，这样的数对同样满足题意。最后一种情况是，如果存在后一个桶中的元素能使得相差的值也 `≤ t`，这样的数对同样满足题意。查询 3 次，如果都不存在，说明当前的 i 找不到满足题意的 j。继续循环寻找。循环一遍都找不到满足题意的数对，输出 false。

代码

```go
package leetcode

// 解法一 桶排序
func containsNearbyAlmostDuplicate(nums []int, k int, t int) bool {
  if k <= 0 || t < 0 || len(nums) < 2 {
    return false
  }
  buckets := map[int]int{}
  for i := 0; i < len(nums); i++ {
    // Get the ID of the bucket from element value nums[i] and bucket width t + 1
    key := nums[i] / (t + 1)
    // -7/9 = 0, but need -7/9 = -1
```

```go
    if nums[i] < 0 {
      key--
    }
    if _, ok := buckets[key]; ok {
      return true
    }
    // check the lower bucket, and have to check the value too
    if v, ok := buckets[key-1]; ok && nums[i]-v <= t {
      return true
    }
    // check the upper bucket, and have to check the value too
    if v, ok := buckets[key+1]; ok && v-nums[i] <= t {
      return true
    }
    // maintain k size of window
    if len(buckets) >= k {
      delete(buckets, nums[i-k]/(t+1))
    }
    buckets[key] = nums[i]
  }
  return false
}

func abs(a int) int {
  if a > 0 {
    return a
  }
  return -a
}

// 解法二 滑动窗口 + 剪枝
func containsNearbyAlmostDuplicate1(nums []int, k int, t int) bool {
  if len(nums) <= 1 {
    return false
  }
  if k <= 0 {
    return false
  }
  n := len(nums)
  for i := 0; i < n; i++ {
    count := 0
    for j := i + 1; j < n && count < k; j++ {
      if abs(nums[i]-nums[j]) <= t {
        return true
      }
      count++
    }
  }
  return false
```

```
}
```

222. Count Complete Tree Nodes

题目

Given a complete binary tree, count the number of nodes.

Note:

Definition of a complete binary tree from Wikipedia:
In a complete binary tree every level, except possibly the last, is completely filled, and all nodes in the last level are as far left as possible. It can have between 1 and 2h nodes inclusive at the last level h.

Example:

```
Input:
    1
   / \
  2   3
 / \  /
4  5 6

Output: 6
```

题目大意

输出一颗完全二叉树的结点个数。

解题思路

这道题其实按照层序遍历一次树，然后把每一层的结点个数相加即可。

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
```

```go
 */
func countNodes(root *TreeNode) int {
  if root == nil {
    return 0
  }
  queue := []*TreeNode{}
  queue = append(queue, root)
  curNum, nextLevelNum, res := 1, 0, 1
  for len(queue) != 0 {
    if curNum > 0 {
      node := queue[0]
      if node.Left != nil {
        queue = append(queue, node.Left)
        nextLevelNum++
      }
      if node.Right != nil {
        queue = append(queue, node.Right)
        nextLevelNum++
      }
      curNum--
      queue = queue[1:]
    }
    if curNum == 0 {
      res += nextLevelNum
      curNum = nextLevelNum
      nextLevelNum = 0
    }
  }
  return res
}
```

223. Rectangle Area

题目

Find the total area covered by two **rectilinear** rectangles in a **2D** plane.

Each rectangle is defined by its bottom left corner and top right corner as shown in the figure.

Example:

```
Input: A = -3, B = 0, C = 3, D = 4, E = 0, F = -1, G = 9, H = 2
Output: 45
```

Note:

Assume that the total area is never beyond the maximum possible value of **int**.

题目大意

在二维平面上计算出两个由直线构成的矩形重叠后形成的总面积。每个矩形由其左下顶点和右上顶点坐标表示，如图所示。说明: 假设矩形面积不会超出 int 的范围。

解题思路

- 给出两个矩形的坐标，求这两个矩形在坐标轴上覆盖的总面积。
- 几何题，由于只有 2 个矩形，所以按照题意做即可。先分别求两个矩形的面积，加起来再减去两个矩形重叠的面积。

代码

```go
package leetcode

func computeArea(A int, B int, C int, D int, E int, F int, G int, H int) int {
  X0, Y0, X1, Y1 := max(A, E), max(B, F), min(C, G), min(D, H)
  return area(A, B, C, D) + area(E, F, G, H) - area(X0, Y0, X1, Y1)
}
```

```go
func area(x0, y0, x1, y1 int) int {
  l, h := x1-x0, y1-y0
  if l <= 0 || h <= 0 {
    return 0
  }
  return l * h
}
```

224. Basic Calculator

题目

Implement a basic calculator to evaluate a simple expression string.

The expression string may contain open `(` and closing parentheses `)`, the plus `+` or minus sign `-`, **non-negative** integers and empty spaces .

Example 1:

```
Input: "1 + 1"
Output: 2
```

Example 2:

```
Input: " 2-1 + 2 "
Output: 3
```

Example 3:

```
Input: "(1+(4+5+2)-3)+(6+8)"
Output: 23
```

Note:

- You may assume that the given expression is always valid.
- **Do not** use the `eval` built-in library function.

题目大意

实现一个基本的计算器来计算一个简单的字符串表达式的值。字符串表达式可以包含左括号 (，右括号)，加号 + ，减号 -，非负整数和空格 。

解题思路

- 注意点一：算式中有空格，需要跳过
- 注意点二：算式中会出现负数，负负得正的情况需要特殊处理，所以需要记录每次计算出来的符号

代码

```
package leetcode

import (
  "container/list"
  "fmt"
  "strconv"
)

// 解法一
func calculate(s string) int {
  i, stack, result, sign := 0, list.New(), 0, 1 // 记录加减状态
  for i < len(s) {
    if s[i] == ' ' {
      i++
    } else if s[i] <= '9' && s[i] >= '0' { // 获取一段数字
      base, v := 10, int(s[i]-'0')
      for i+1 < len(s) && s[i+1] <= '9' && s[i+1] >= '0' {
        v = v*base + int(s[i+1]-'0')
        i++
      }
      result += v * sign
      i++
    } else if s[i] == '+' {
      sign = 1
      i++
    } else if s[i] == '-' {
      sign = -1
      i++
    } else if s[i] == '(' { // 把之前计算结果及加减状态压栈，开始新的计算
      stack.PushBack(result)
      stack.PushBack(sign)
      result = 0
      sign = 1
      i++
    } else if s[i] == ')' { // 新的计算结果 * 前一个加减状态 + 之前计算结果
      result = result*stack.Remove(stack.Back()).(int) + stack.Remove(stack.Back()).
(int)
      i++
    }
  }
  return result
}

// 解法二
```

```go
func calculate1(s string) int {
  stack := []byte{}
  for i := 0; i < len(s); i++ {
    if s[i] == ' ' {
      continue
    } else if s[i] == ')' {
      tmp, index := "", len(stack)-1
      for ; index >= 0; index-- {
        if stack[index] == '(' {
          break
        }
      }
      tmp = string(stack[index+1:])
      stack = stack[:index]
      res := strconv.Itoa(calculateStr(tmp))
      for j := 0; j < len(res); j++ {
        stack = append(stack, res[j])
      }
    } else {
      stack = append(stack, s[i])
    }
  }
  fmt.Printf("stack = %v\n", string(stack))
  return calculateStr(string(stack))
}

func calculateStr(str string) int {
  s, nums, tmpStr, res := []byte{}, []int{}, "", 0
  // 处理符号的问题，++得+，--得+，+-、-+得-
  for i := 0; i < len(str); i++ {
    if len(s) > 0 && s[len(s)-1] == '+' && str[i] == '+' {
      continue
    } else if len(s) > 0 && s[len(s)-1] == '+' && str[i] == '-' {
      s[len(s)-1] = '-'
    } else if len(s) > 0 && s[len(s)-1] == '-' && str[i] == '+' {
      continue
    } else if len(s) > 0 && s[len(s)-1] == '-' && str[i] == '-' {
      s[len(s)-1] = '+'
    } else {
      s = append(s, str[i])
    }
  }
  str = string(s)
  s = []byte{}
  for i := 0; i < len(str); i++ {
    if isDigital(str[i]) {
      tmpStr += string(str[i])
    } else {
      num, _ := strconv.Atoi(tmpStr)
```

```
        nums = append(nums, num)
        tmpStr = ""
        s = append(s, str[i])
      }
    }
    if tmpStr != "" {
      num, _ := strconv.Atoi(tmpStr)
      nums = append(nums, num)
    }
    res = nums[0]
    for i := 0; i < len(s); i++ {
      if s[i] == '+' {
        res += nums[i+1]
      } else {
        res -= nums[i+1]
      }
    }
    fmt.Printf("s = %v nums = %v res = %v\n", string(s), nums, res)
    return res
  }

  func isDigital(v byte) bool {
    if v >= '0' && v <= '9' {
      return true
    }
    return false
  }
```

225. Implement Stack using Queues

题目

Implement the following operations of a stack using queues.

- push(x) -- Push element x onto stack.
- pop() -- Removes the element on top of the stack.
- top() -- Get the top element.
- empty() -- Return whether the stack is empty.

Example:

```
MyStack stack = new MyStack();

stack.push(1);
stack.push(2);
stack.top();   // returns 2
stack.pop();   // returns 2
stack.empty(); // returns false
```

Note:

- You must use only standard operations of a queue -- which means only push to back, peek/pop from front, size, and is empty operations are valid.
- Depending on your language, queue may not be supported natively. You may simulate a queue by using a list or deque (double-ended queue), as long as you use only standard operations of a queue.
- You may assume that all operations are valid (for example, no pop or top operations will be called on an empty stack).

题目大意

题目要求用队列实现一个栈的基本操作：push(x)、pop()、top()、empty()。

解题思路

按照题目要求实现即可。

代码

```go
package leetcode

type MyStack struct {
  enque []int
  deque []int
}

/** Initialize your data structure here. */
func Constructor225() MyStack {
  return MyStack{[]int{}, []int{}}
}

/** Push element x onto stack. */
func (this *MyStack) Push(x int) {
  this.enque = append(this.enque, x)
}

/** Removes the element on top of the stack and returns that element. */
```

```go
func (this *MyStack) Pop() int {
  length := len(this.enque)
  for i := 0; i < length-1; i++ {
    this.deque = append(this.deque, this.enque[0])
    this.enque = this.enque[1:]
  }
  topEle := this.enque[0]
  this.enque = this.deque
  this.deque = nil

  return topEle
}

/** Get the top element. */
func (this *MyStack) Top() int {
  topEle := this.Pop()
  this.enque = append(this.enque, topEle)

  return topEle
}

/** Returns whether the stack is empty. */
func (this *MyStack) Empty() bool {
  if len(this.enque) == 0 {
    return true
  }

  return false
}
```

226. Invert Binary Tree

题目

Invert a binary tree.

Example:

Input:

```
     4
   /   \
  2     7
 / \   / \
1   3 6   9
```

Output:

```
     4
   /   \
  7     2
 / \   / \
9   6 3   1
```

Trivia:

This problem was inspired by this original tweet by Max Howell:

> Google: 90% of our engineers use the software you wrote (Homebrew), but you can't invert a binary tree on a whiteboard so f*** off.

题目大意

"经典"的反转二叉树的问题。

解题思路

还是用递归来解决，先递归调用反转根节点的左孩子，然后递归调用反转根节点的右孩子，然后左右交换根节点的左孩子和右孩子。

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func invertTree(root *TreeNode) *TreeNode {
  if root == nil {
    return nil
  }
  invertTree(root.Left)
  invertTree(root.Right)
  root.Left, root.Right = root.Right, root.Left
  return root
```

```
}
```

227. Basic Calculator II

题目

Given a string `s` which represents an expression, *evaluate this expression and return its value*.

The integer division should truncate toward zero.

Example 1:

```
Input: s = "3+2*2"
Output: 7
```

Example 2:

```
Input: s = " 3/2 "
Output: 1
```

Example 3:

```
Input: s = " 3+5 / 2 "
Output: 5
```

Constraints:

- `1 <= s.length <= 3 * 10^5`
- `s` consists of integers and operators `('+', '-', '*', '/')` separated by some number of spaces.
- `s` represents **a valid expression**.
- All the integers in the expression are non-negative integers in the range `[0, 2^31 - 1]`.
- The answer is **guaranteed** to fit in a **32-bit integer**.

题目大意

给你一个字符串表达式 `s` ，请你实现一个基本计算器来计算并返回它的值。整数除法仅保留整数部分。

解题思路

- 这道题是第 224 题的加强版。第 224 题中只有加减运算和括号，这一题增加了乘除运算。由于乘除运算的优先级高于加减。所以先计算乘除运算，将算出来的结果再替换回原来的算式中。最后只剩下加减运算，于是题目降级成了第 224 题。
- 把加减运算符号后面的数字压入栈中，遇到乘除运算，直接将它与栈顶的元素计算，并将计算后的结果放入栈顶。若读到一个运算符，或者遍历到字符串末尾，即认为是遍历到了数字末尾。处理完该数字后，更新 `preSign` 为当前遍历的字符。遍历完字符串 `s` 后，将栈中元素累加，即为该字符串表达式的值。时间复杂度 O(n)，空间复杂度 O(n)。

代码

```go
package leetcode

func calculate(s string) int {
  stack, preSign, num, res := []int{}, '+', 0, 0
  for i, ch := range s {
    isDigit := '0' <= ch && ch <= '9'
    if isDigit {
      num = num*10 + int(ch-'0')
    }
    if !isDigit && ch != ' ' || i == len(s)-1 {
      switch preSign {
      case '+':
        stack = append(stack, num)
      case '-':
        stack = append(stack, -num)
      case '*':
        stack[len(stack)-1] *= num
      default:
        stack[len(stack)-1] /= num
      }
      preSign = ch
      num = 0
    }
  }
  for _, v := range stack {
    res += v
  }
  return res
}
```

228. Summary Ranges

题目

You are given a **sorted unique** integer array `nums`.

Return *the **smallest sorted** list of ranges that **cover all the numbers in the array exactly***. That is, each element of `nums` is covered by exactly one of the ranges, and there is no integer `x` such that `x` is in one of the ranges but not in `nums`.

Each range `[a,b]` in the list should be output as:

- `"a->b"` if `a != b`
- `"a"` if `a == b`

Example 1:

```
Input: nums = [0,1,2,4,5,7]
Output: ["0->2","4->5","7"]
Explanation: The ranges are:
[0,2] --> "0->2"
[4,5] --> "4->5"
[7,7] --> "7"
```

Example 2:

```
Input: nums = [0,2,3,4,6,8,9]
Output: ["0","2->4","6","8->9"]
Explanation: The ranges are:
[0,0] --> "0"
[2,4] --> "2->4"
[6,6] --> "6"
[8,9] --> "8->9"
```

Example 3:

```
Input: nums = []
Output: []
```

Example 4:

```
Input: nums = [-1]
Output: ["-1"]
```

Example 5:

```
Input: nums = [0]
Output: ["0"]
```

Constraints:

- `0 <= nums.length <= 20`
- `231 <= nums[i] <= 231 - 1`
- All the values of `nums` are **unique**.
- `nums` is sorted in ascending order.

题目大意

给定一个无重复元素的有序整数数组 nums 。

返回 恰好覆盖数组中所有数字 的 最小有序 区间范围列表。也就是说，nums 的每个元素都恰好被某个区间范围所覆盖，并且不存在属于某个范围但不属于 nums 的数字 x 。

列表中的每个区间范围 [a,b] 应该按如下格式输出：

- "a->b" ，如果 a != b
- "a" ，如果 a == b

解题思路

- 简单题。按照题意，用一个游标变量累加寻找连续的区间。一旦出现了中断，就按照题意格式输出。输出的规则有多种，带箭头的区间，单个元素区间，空区间。

代码

```
package leetcode

import (
  "strconv"
)

func summaryRanges(nums []int) (ans []string) {
  for i, n := 0, len(nums); i < n; {
    left := i
    for i++; i < n && nums[i-1]+1 == nums[i]; i++ {
    }
    s := strconv.Itoa(nums[left])
    if left != i-1 {
      s += "->" + strconv.Itoa(nums[i-1])
    }
    ans = append(ans, s)
  }
  return
}
```

229. Majority Element II

题目

Given an integer array of size n, find all elements that appear more than `⌊ n/3 ⌋` times.

Note: The algorithm should run in linear time and in O(1) space.

Example 1:

```
Input: [3,2,3]
Output: [3]
```

Example 2:

```
Input: [1,1,1,3,3,2,2,2]
Output: [1,2]
```

题目大意

给定一个大小为 n 的数组，找出其中所有出现超过 ⌊ n/3 ⌋ 次的元素。说明: 要求算法的时间复杂度为 O(n)，空间复杂度为 O(1)。

解题思路

- 这一题是第 169 题的加强版。Boyer-Moore Majority Vote algorithm 算法的扩展版。
- 题目要求找出数组中出现次数大于 `⌊ n/3 ⌋` 次的数。要求空间复杂度为 O(1)。简单题。
- 这篇文章写的不错，可参考：https://gregable.com/2013/10/majority-vote-algorithm-find-majority.html

代码

```go
package leetcode

// 解法一 时间复杂度 O(n) 空间复杂度 O(1)
func majorityElement229(nums []int) []int {
  // since we are checking if a num appears more than 1/3 of the time
  // it is only possible to have at most 2 nums (>1/3 + >1/3 = >2/3)
  count1, count2, candidate1, candidate2 := 0, 0, 0, 1
  // Select Candidates
  for _, num := range nums {
    if num == candidate1 {
      count1++
    } else if num == candidate2 {
      count2++
    } else if count1 <= 0 {
      // We have a bad first candidate, replace!
      candidate1, count1 = num, 1
    } else if count2 <= 0 {
      // We have a bad second candidate, replace!
      candidate2, count2 = num, 1
    } else {
      // Both candidates suck, boo!
      count1--
      count2--
    }
  }
  // Recount!
  count1, count2 = 0, 0
  for _, num := range nums {
    if num == candidate1 {
      count1++
    } else if num == candidate2 {
```

```
        count2++
      }
    }
    length := len(nums)
    if count1 > length/3 && count2 > length/3 {
      return []int{candidate1, candidate2}
    }
    if count1 > length/3 {
      return []int{candidate1}
    }
    if count2 > length/3 {
      return []int{candidate2}
    }
    return []int{}
  }

  // 解法二 时间复杂度 O(n) 空间复杂度 O(n)
  func majorityElement229_1(nums []int) []int {
    result, m := make([]int, 0), make(map[int]int)
    for _, val := range nums {
      if v, ok := m[val]; ok {
        m[val] = v + 1
      } else {
        m[val] = 1
      }
    }
    for k, v := range m {
      if v > len(nums)/3 {
        result = append(result, k)
      }
    }
    return result
  }
```

230. Kth Smallest Element in a BST

题目

Given a binary search tree, write a function `kthSmallest` to find the **k**th smallest element in it.

Note: You may assume k is always valid, 1 ≤ k ≤ BST's total elements.

Example 1:

```
Input: root = [3,1,4,null,2], k = 1
   3
  / \
 1   4
  \
   2
Output: 1
```

Example 2:

```
Input: root = [5,3,6,2,4,null,null,1], k = 3
       5
      / \
     3   6
    / \
   2   4
  /
 1
Output: 3
```

Follow up:What if the BST is modified (insert/delete operations) often and you need to find the kth smallest frequently? How would you optimize the kthSmallest routine?

题目大意

给定一个二叉搜索树，编写一个函数 kthSmallest 来查找其中第 k 个最小的元素。你可以假设 k 总是有效的，1 ≤ k ≤ 二叉搜索树元素个数。

解题思路

- 由于二叉搜索树有序的特性，所以中根遍历它，遍历到第 K 个数的时候就是结果

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func kthSmallest(root *TreeNode, k int) int {
```

```go
  res, count := 0, 0
  inorder230(root, k, &count, &res)
  return res
}

func inorder230(node *TreeNode, k int, count *int, ans *int) {
  if node != nil {
    inorder230(node.Left, k, count, ans)
    *count++
    if *count == k {
      *ans = node.Val
      return
    }
    inorder230(node.Right, k, count, ans)
  }
}
```

231. Power of Two

题目

Given an integer, write a function to determine if it is a power of two.

Example 1:

```
Input: 1
Output: true
Explanation: 2^0 = 1
```

Example 2:

```
Input: 16
Output: true
Explanation: 2^4 = 16
```

Example 3:

```
Input: 218
Output: false
```

题目大意

给定一个整数，编写一个函数来判断它是否是 2 的幂次方。

解题思路

- 判断一个数是不是 2 的 n 次方。
- 这一题最简单的思路是循环，可以通过。但是题目要求不循环就要判断，这就需要用到数论的知识了。这一题和第 326 题是一样的思路。

代码

```
package leetcode

// 解法一 二进制位操作法
func isPowerOfTwo(num int) bool {
  return (num > 0 && ((num & (num - 1)) == 0))
}

// 解法二 数论
func isPowerOfTwo1(num int) bool {
  return num > 0 && (1073741824%num == 0)
}

// 解法三 打表法
func isPowerOfTwo2(num int) bool {
  allPowerOfTwoMap := map[int]int{1: 1, 2: 2, 4: 4, 8: 8, 16: 16, 32: 32, 64: 64, 128:
128, 256: 256, 512: 512, 1024: 1024, 2048: 2048, 4096: 4096, 8192: 8192, 16384: 16384,
32768: 32768, 65536: 65536, 131072: 131072, 262144: 262144, 524288: 524288, 1048576:
1048576, 2097152: 2097152, 4194304: 4194304, 8388608: 8388608, 16777216: 16777216,
33554432: 33554432, 67108864: 67108864, 134217728: 134217728, 268435456: 268435456,
536870912: 536870912, 1073741824: 1073741824}
  _, ok := allPowerOfTwoMap[num]
  return ok
}

// 解法四 循环
func isPowerOfTwo3(num int) bool {
  for num >= 2 {
    if num%2 == 0 {
      num = num / 2
    } else {
      return false
    }
  }
  return num == 1
}
```

232. Implement Queue using Stacks

题目

Implement the following operations of a queue using stacks.

- push(x) -- Push element x to the back of queue.
- pop() -- Removes the element from in front of queue.
- peek() -- Get the front element.
- empty() -- Return whether the queue is empty.

Example:

```
MyQueue queue = new MyQueue();

queue.push(1);
queue.push(2);
queue.peek();  // returns 1
queue.pop();   // returns 1
queue.empty(); // returns false
```

Note:

- You must use only standard operations of a stack -- which means only push to top, peek/pop from top, size, and is empty operations are valid.
- Depending on your language, stack may not be supported natively. You may simulate a stack by using a list or deque (double-ended queue), as long as you use only standard operations of a stack.
- You may assume that all operations are valid (for example, no pop or peek operations will be called on an empty queue).

题目大意

题目要求用栈实现一个队列的基本操作：push(x)、pop()、peek()、empty()。

解题思路

按照题目要求实现即可。

代码

```go
package leetcode

type MyQueue struct {
  Stack *[]int
  Queue *[]int
}

/** Initialize your data structure here. */
func Constructor232() MyQueue {
```

```go
  tmp1, tmp2 := []int{}, []int{}
  return MyQueue{Stack: &tmp1, Queue: &tmp2}
}

/** Push element x to the back of queue. */
func (this *MyQueue) Push(x int) {
  *this.Stack = append(*this.Stack, x)
}

/** Removes the element from in front of queue and returns that element. */
func (this *MyQueue) Pop() int {
  if len(*this.Queue) == 0 {
    this.fromStackToQueue(this.Stack, this.Queue)
  }

  popped := (*this.Queue)[len(*this.Queue)-1]
  *this.Queue = (*this.Queue)[:len(*this.Queue)-1]
  return popped
}

/** Get the front element. */
func (this *MyQueue) Peek() int {
  if len(*this.Queue) == 0 {
    this.fromStackToQueue(this.Stack, this.Queue)
  }

  return (*this.Queue)[len(*this.Queue)-1]
}

/** Returns whether the queue is empty. */
func (this *MyQueue) Empty() bool {
  return len(*this.Stack)+len(*this.Queue) == 0
}

func (this *MyQueue) fromStackToQueue(s, q *[]int) {
  for len(*s) > 0 {
    popped := (*s)[len(*s)-1]
    *s = (*s)[:len(*s)-1]

    *q = append(*q, popped)
  }
}
```

234. Palindrome Linked List

题目

Given a singly linked list, determine if it is a palindrome.

Example 1:

```
Input: 1->2
Output: false
```

Example 2:

```
Input: 1->2->2->1
Output: true
```

Follow up:

Could you do it in O(n) time and O(1) space?

题目大意

判断一个链表是否是回文链表。要求时间复杂度 O(n)，空间复杂度 O(1)。

解题思路

这道题只需要在第 143 题上面改改就可以了。思路是完全一致的。先找到中间结点，然后反转中间结点后面到结尾的所有结点。最后一一判断头结点开始的结点和中间结点往后开始的结点是否相等。如果一直相等，就是回文链表，如果有不相等的，直接返回不是回文链表。

代码

```go
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */

// 此题和 143 题 Reorder List 思路基本一致
func isPalindrome234(head *ListNode) bool {
  if head == nil || head.Next == nil {
    return true
  }
  res := true
```

```go
  // 寻找中间结点
  p1 := head
  p2 := head
  for p2.Next != nil && p2.Next.Next != nil {
    p1 = p1.Next
    p2 = p2.Next.Next
  }

  // 反转链表后半部分  1->2->3->4->5->6 to 1->2->3->6->5->4
  preMiddle := p1
  preCurrent := p1.Next
  for preCurrent.Next != nil {
    current := preCurrent.Next
    preCurrent.Next = current.Next
    current.Next = preMiddle.Next
    preMiddle.Next = current
  }

  // 扫描表，判断是否是回文
  p1 = head
  p2 = preMiddle.Next
  // fmt.Printf("p1 = %v p2 = %v preMiddle = %v head = %v\n", p1.Val, p2.Val,
preMiddle.Val, L2ss(head))
  for p1 != preMiddle {
    // fmt.Printf("*****p1 = %v p2 = %v preMiddle = %v head = %v\n", p1, p2, preMiddle,
L2ss(head))
    if p1.Val == p2.Val {
      p1 = p1.Next
      p2 = p2.Next
      // fmt.Printf("-------p1 = %v p2 = %v preMiddle = %v head = %v\n", p1, p2,
preMiddle, L2ss(head))
    } else {
      res = false
      break
    }
  }
  if p1 == preMiddle {
    if p2 != nil && p1.Val != p2.Val {
      return false
    }
  }

  return res
}

// L2ss define
func L2ss(head *ListNode) []int {
  res := []int{}
```

```
  for head != nil {
    res = append(res, head.val)
    head = head.Next
  }

  return res
}
```

235. Lowest Common Ancestor of a Binary Search Tree

题目

Given a binary search tree (BST), find the lowest common ancestor (LCA) of two given nodes in the BST.

According to the definition of LCA on Wikipedia: “The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow **a node to be a descendant of itself**).”

Given binary search tree: root = [6,2,8,0,4,7,9,null,null,3,5]

Example 1:

```
Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8
Output: 6
Explanation: The LCA of nodes 2 and 8 is 6.
```

Example 2:

```
Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4
Output: 2
Explanation: The LCA of nodes 2 and 4 is 2, since a node can be a descendant of itself
according to the LCA definition.
```

Note:

- All of the nodes' values will be unique.

- p and q are different and both values will exist in the BST.

题目大意

给定一个二叉搜索树, 找到该树中两个指定节点的最近公共祖先。

百度百科中最近公共祖先的定义为：“对于有根树 T 的两个结点 p、q，最近公共祖先表示为一个结点 x，满足 x 是 p、q 的祖先且 x 的深度尽可能大（一个节点也可以是它自己的祖先）。”

解题思路

- 在二叉搜索树中求两个节点的最近公共祖先，由于二叉搜索树的特殊性质，所以找任意两个节点的最近公共祖先非常简单。

代码

```go
package leetcode

/**
 * Definition for TreeNode.
 * type TreeNode struct {
 *     Val int
 *     Left *ListNode
 *     Right *ListNode
 * }
 */
func lowestCommonAncestor(root, p, q *TreeNode) *TreeNode {
  if p == nil || q == nil || root == nil {
    return nil
  }
  if p.Val < root.Val && q.Val < root.Val {
    return lowestCommonAncestor(root.Left, p, q)
  }
  if p.Val > root.Val && q.Val > root.Val {
    return lowestCommonAncestor(root.Right, p, q)
  }
  return root
}
```

236. Lowest Common Ancestor of a Binary Tree

题目

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

According to the [definition of LCA on Wikipedia](): "The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow **a node to be a descendant of itself**)."

Given the following binary tree: root = [3,5,1,6,2,0,8,null,null,7,4]

Example 1:

```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1
Output: 3
Explanation: The LCA of nodes 5 and 1 is 3.
```

Example 2:

```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4
Output: 5
Explanation: The LCA of nodes 5 and 4 is 5, since a node can be a descendant of itself
according to the LCA definition.
```

Note:

- All of the nodes' values will be unique.
- p and q are different and both values will exist in the binary tree.

题目大意

给定一个二叉树, 找到该树中两个指定节点的最近公共祖先。

百度百科中最近公共祖先的定义为：“对于有根树 T 的两个结点 p、q，最近公共祖先表示为一个结点 x，满足 x 是 p、q 的祖先且 x 的深度尽可能大（一个节点也可以是它自己的祖先）。”

解题思路

- 这是一套经典的题目，寻找任意一个二叉树中两个结点的 LCA 最近公共祖先，考察递归

代码

```go
package leetcode

/**
 * Definition for TreeNode.
 * type TreeNode struct {
 *     Val int
 *     Left *ListNode
 *     Right *ListNode
 * }
 */
func lowestCommonAncestor236(root, p, q *TreeNode) *TreeNode {
  if root == nil || root == q || root == p {
    return root
  }
  left := lowestCommonAncestor236(root.Left, p, q)
  right := lowestCommonAncestor236(root.Right, p, q)
  if left != nil {
    if right != nil {
      return root
    }
    return left
  }
  return right
}
```

237. Delete Node in a Linked List

题目

Write a function to delete a node (except the tail) in a singly linked list, given only access to that node.

Given linked list -- head = [4,5,1,9], which looks like following:

Example 1:

```
Input: head = [4,5,1,9], node = 5
Output: [4,1,9]
Explanation: You are given the second node with value 5, the linked list should become
4 -> 1 -> 9 after calling your function.
```

Example 2:

```
Input: head = [4,5,1,9], node = 1
Output: [4,5,9]
Explanation: You are given the third node with value 1, the linked list should become 4
-> 5 -> 9 after calling your function.
```

Note:

- The linked list will have at least two elements.
- All of the nodes' values will be unique.
- The given node will not be the tail and it will always be a valid node of the linked list.
- Do not return anything from your function.

题目大意

删除给点结点。没有给链表的头结点。

解题思路

其实就是把后面的结点都覆盖上来即可。或者直接当前结点的值等于下一个结点，Next 指针指向下下个结点，这样做也可以，只不过中间有一个结点不被释放，内存消耗多一些。

代码

```go
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */
func deleteNode(node *ListNode) {
  if node == nil {
    return
  }
```

```
  cur := node
  for cur.Next.Next != nil {
    cur.Val = cur.Next.Val
    cur = cur.Next
  }
  cur.Val = cur.Next.Val
  cur.Next = nil
}
```

239. Sliding Window Maximum

题目

Given an array *nums*, there is a sliding window of size *k* which is moving from the very left of the array to the very right. You can only see the *k* numbers in the window. Each time the sliding window moves right by one position. Return the max sliding window.

Example:

```
Input: nums = [1,3,-1,-3,5,3,6,7], and k = 3
Output: [3,3,5,5,6,7]
Explanation:

Window position                Max
---------------               -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
 1  3  -1  -3 [5  3  6] 7       6
 1  3  -1  -3  5 [3  6  7]      7
```

Note:

You may assume *k* is always valid, 1 ≤ k ≤ input array's size for non-empty array.

Follow up:

Could you solve it in linear time?

题目大意

给定一个数组 nums，有一个大小为 k 的滑动窗口从数组的最左侧移动到数组的最右侧。你只可以看到在滑动窗口 k 内的数字。滑动窗口每次只向右移动一位。返回滑动窗口最大值。

解题思路

- 给定一个数组和一个窗口为 K 的窗口，当窗口从数组的左边滑动到数组右边的时候，输出每次移动窗口以后，在窗口内的最大值。
- 这道题最暴力的方法就是 2 层循环，时间复杂度 O(n * K)。
- 另一种思路是用优先队列，每次窗口以后的时候都向优先队列里面新增一个节点，并删除一个节点。时间复杂度是 O(n * log n)
- 最优的解法是用双端队列，队列的一边永远都存的是窗口的最大值，队列的另外一个边存的是比最大值小的值。队列中最大值左边的所有值都出队。在保证了双端队列的一边即是最大值以后，时间复杂度是 O(n)，空间复杂度是 O(K)

代码

```
package leetcode

// 解法一 暴力解法 O(nk)
func maxSlidingWindow1(a []int, k int) []int {
  res := make([]int, 0, k)
  n := len(a)
  if n == 0 {
    return []int{}
  }
  for i := 0; i <= n-k; i++ {
    max := a[i]
    for j := 1; j < k; j++ {
      if max < a[i+j] {
        max = a[i+j]
      }
    }
    res = append(res, max)
  }

  return res
}

// 解法二 双端队列 Deque
func maxSlidingWindow(nums []int, k int) []int {
  if len(nums) == 0 || len(nums) < k {
    return make([]int, 0)
  }
  window := make([]int, 0, k) // store the index of nums
  result := make([]int, 0, len(nums)-k+1)
  for i, v := range nums { // if the left-most index is out of window, remove it
    if i >= k && window[0] <= i-k {
      window = window[1:len(window)]
    }
    for len(window) > 0 && nums[window[len(window)-1]] < v { // maintain window
      window = window[0 : len(window)-1]
```

```
    }
    window = append(window, i) // store the index of nums
    if i >= k-1 {
      result = append(result, nums[window[0]]) // the left-most is the index of max value in nums
    }
  }
  return result
}
```

240. Search a 2D Matrix II

题目

Write an efficient algorithm that searches for a value in an m x n matrix. This matrix has the following properties:

- Integers in each row are sorted in ascending from left to right.
- Integers in each column are sorted in ascending from top to bottom.

Example:

Consider the following matrix:

```
[
  [1,   4,  7, 11, 15],
  [2,   5,  8, 12, 19],
  [3,   6,  9, 16, 22],
  [10, 13, 14, 17, 24],
  [18, 21, 23, 26, 30]
]
```

Given target = `5`, return `true`.

Given target = `20`, return `false`.

题目大意

编写一个高效的算法来搜索 m x n 矩阵 matrix 中的一个目标值 target。该矩阵具有以下特性：

- 每行的元素从左到右升序排列。
- 每列的元素从上到下升序排列。

解题思路

- 给出一个二维矩阵，矩阵的特点是每一个行内，元素随着下标增大而增大，每一列内，元素也是随着下标增大而增大。但是相邻两行的元素并没有大小关系。例如第一行最后一个元素就比第二行第一个元素要大。要求设

计一个算法能在这个矩阵中高效的找到一个数，如果找到就输出 true，找不到就输出 false。

- 这一题是第 74 题的加强版。第 74 题中的二维矩阵完全是一个有序的一维矩阵，但是这一题如果把它拍扁成一维，并不是有序的。首先每一个行或者每一列是有序的，那么我们可以依次在每一行或者每一列中利用二分去搜索。这样做时间复杂度为 O(n log n)。
- 还有一个模拟的解法。通过观察，我们发现了这个矩阵的一个特点，最右边一列的元素是本行中最大的元素，所以我们可以先从最右边一列开始找到第一个比 target 元素大的元素，这个元素所在的行，是我们接着要搜索的。在行中搜索是从最右边开始往左边搜索，时间复杂度是 O(n)，算上一开始在最右边一列中查找的时间复杂度是 O(m)，所以最终的时间复杂度为 O(m+n)。

代码

```go
package leetcode

// 解法一 模拟，时间复杂度 O(m+n)
func searchMatrix240(matrix [][]int, target int) bool {
  if len(matrix) == 0 {
    return false
  }
  row, col := 0, len(matrix[0])-1
  for col >= 0 && row <= len(matrix)-1 {
    if target == matrix[row][col] {
      return true
    } else if target > matrix[row][col] {
      row++
    } else {
      col--
    }
  }
  return false
}

// 解法二 二分搜索，时间复杂度 O(n log n)
func searchMatrix2401(matrix [][]int, target int) bool {
  if len(matrix) == 0 {
    return false
  }
  for _, row := range matrix {
    low, high := 0, len(matrix[0])-1
    for low <= high {
      mid := low + (high-low)>>1
      if row[mid] > target {
        high = mid - 1
      } else if row[mid] < target {
        low = mid + 1
      } else {
        return true
      }
```

```
    }
  }
  return false
}
```

242. Valid Anagram

题目

Given two strings s and t , write a function to determine if t is an anagram of s.

Example 1:

```
Input: s = "anagram", t = "nagaram"
Output: true
```

Example 2:

```
Input: s = "rat", t = "car"
Output: false
```

Note:

You may assume the string contains only lowercase alphabets.

Follow up:

What if the inputs contain unicode characters? How would you adapt your solution to such case?

题目大意

给出 2 个字符串 s 和 t，如果 t 中的字母在 s 中都存在，输出 true，否则输出 false。

解题思路

这道题可以用打表的方式做。先把 s 中的每个字母都存在一个 26 个容量的数组里面，每个下标依次对应 26 个字母。s 中每个字母都对应表中一个字母，每出现一次就加 1。然后再扫字符串 t，每出现一个字母就在表里面减一。如果都出现了，最终表里面的值肯定都是 0 。最终判断表里面的值是否都是 0 即可，有非 0 的数都输出 false 。

代码

```go
package leetcode

// 解法一
func isAnagram(s string, t string) bool {
  alphabet := make([]int, 26)
  sBytes := []byte(s)
  tBytes := []byte(t)
  if len(sBytes) != len(tBytes) {
    return false
  }
  for i := 0; i < len(sBytes); i++ {
    alphabet[sBytes[i]-'a']++
  }
  for i := 0; i < len(tBytes); i++ {
    alphabet[tBytes[i]-'a']--
  }
  for i := 0; i < 26; i++ {
    if alphabet[i] != 0 {
      return false
    }
  }
  return true
}

// 解法二
func isAnagram1(s string, t string) bool {
  if s == "" && t == "" {
    return true
  }
  if s == "" || t == "" {
    return false
  }
  sBytes := []byte(s)
  tBytes := []byte(t)
  if len(sBytes) != len(tBytes) {
    return false
  }
  quickSortByte(sBytes, 0, len(sBytes)-1)
  quickSortByte(tBytes, 0, len(tBytes)-1)

  for i := 0; i < len(sBytes); i++ {
    if sBytes[i] != tBytes[i] {
      return false
    }
  }
  return true
}
func partitionByte(a []byte, lo, hi int) int {
  pivot := a[hi]
```

```
  i := lo - 1
  for j := lo; j < hi; j++ {
    if a[j] > pivot {
      i++
      a[j], a[i] = a[i], a[j]
    }
  }
  a[i+1], a[hi] = a[hi], a[i+1]
  return i + 1
}
func quickSortByte(a []byte, lo, hi int) {
  if lo >= hi {
    return
  }
  p := partitionByte(a, lo, hi)
  quickSortByte(a, lo, p-1)
  quickSortByte(a, p+1, hi)
}
```

257. Binary Tree Paths

题目

Given a binary tree, return all root-to-leaf paths.

Note: A leaf is a node with no children.

Example:

```
Input:

   1
 /   \
2     3
 \
  5

Output: ["1->2->5", "1->3"]

Explanation: All root-to-leaf paths are: 1->2->5, 1->3
```

题目大意

给定一个二叉树，返回所有从根节点到叶子节点的路径。说明: 叶子节点是指没有子节点的节点。

解题思路

- Google 的面试题，考察递归

代码

```go
package leetcode

import (
  "strconv"
)

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func binaryTreePaths(root *TreeNode) []string {
  if root == nil {
    return []string{}
  }
  res := []string{}
  if root.Left == nil && root.Right == nil {
    return []string{strconv.Itoa(root.Val)}
  }
  tmpLeft := binaryTreePaths(root.Left)
  for i := 0; i < len(tmpLeft); i++ {
    res = append(res, strconv.Itoa(root.Val)+"->"+tmpLeft[i])
  }
  tmpRight := binaryTreePaths(root.Right)
  for i := 0; i < len(tmpRight); i++ {
    res = append(res, strconv.Itoa(root.Val)+"->"+tmpRight[i])
  }
  return res
}
```

258. Add Digits

题目

Given a non-negative integer `num`, repeatedly add all its digits until the result has only one digit.

Example:

```
Input: 38
Output: 2
Explanation: The process is like: 3 + 8 = 11, 1 + 1 = 2.
             Since 2 has only one digit, return it.
```

Follow up: Could you do it without any loop/recursion in O(1) runtime?

题目大意

给定一个非负整数 num，反复将各个位上的数字相加，直到结果为一位数。

解题思路

- 给定一个非负整数，反复加各个位上的数，直到结果为一位数为止，最后输出这一位数。
- 简单题。按照题意循环累加即可。

代码

```go
package leetcode

func addDigits(num int) int {
  for num > 9 {
    cur := 0
    for num != 0 {
      cur += num % 10
      num /= 10
    }
    num = cur
  }
  return num
}
```

260. Single Number III

题目

Given an array of numbers `nums`, in which exactly two elements appear only once and all the other elements appear exactly twice. Find the two elements that appear only once.

Example:

```
Input:  [1,2,1,3,2,5]
Output: [3,5]
```

Note:

1. The order of the result is not important. So in the above example, `[5, 3]` is also correct.
2. Your algorithm should run in linear runtime complexity. Could you implement it using only constant space complexity?

题目大意

给定一个整数数组 nums，其中恰好有两个元素只出现一次，其余所有元素均出现两次。 找出只出现一次的那两个元素。

注意：

- 结果输出的顺序并不重要，对于上面的例子，[5, 3] 也是正确答案。
- 要求算法时间复杂度是线性的，并且不使用额外的辅助空间。

解题思路

- 这一题是第 136 题的加强版。第 136 题里面只有一个数出现一次，其他数都出现 2 次。而这一次有 2 个数字出现一次，其他数出现 2 次。
- 解题思路还是利用异或，把出现 2 次的数先消除。最后我们要找的 2 个数肯定也是不同的，所以最后 2 个数对一个数进行异或，答案肯定是不同的。那么我们找哪个数为参照物呢？可以随便取，不如就取 lsb 最低位为 1 的数吧
- 于是整个数组会被分为 2 部分，异或 lsb 为 0 的和异或 lsb 为 1 的，在这 2 部分中，用异或操作把出现 2 次的数都消除，那么剩下的 2 个数分别就在这 2 部分中。

代码

```go
package leetcode

func singleNumberIII(nums []int) []int {
  diff := 0
  for _, num := range nums {
    diff ^= num
  }
  // Get its last set bit (lsb)
  diff &= -diff
  res := []int{0, 0} // this array stores the two numbers we will return
  for _, num := range nums {
    if (num & diff) == 0 { // the bit is not set
      res[0] ^= num
    } else { // the bit is set
      res[1] ^= num
    }
  }
  return res
}
```

263. Ugly Number

题目

Write a program to check whether a given number is an ugly number.

Ugly numbers are positive numbers whose prime factors only include 2, 3, 5.

Example 1:

```
Input: 6
Output: true
Explanation: 6 = 2 × 3
```

Example 2:

```
Input: 8
Output: true
Explanation: 8 = 2 × 2 × 2
```

Example 3:

```
Input: 14
Output: false
Explanation: 14 is not ugly since it includes another prime factor 7.
```

Note:

- 1 is typically treated as an ugly number.
- Input is within the 32-bit signed integer range: [−2^31, 2^31 − 1].

题目大意

判断一个数字是否是“丑陋数字”，“丑陋数字”的定义是一个正数，并且因子只包含 2，3，5 。

解题思路

依照题意要求做即可。

代码

```go
package leetcode

func isUgly(num int) bool {
  if num > 0 {
    for _, i := range []int{2, 3, 5} {
      for num%i == 0 {
        num /= i
      }
    }
  }
  return num == 1
}
```

264. Ugly Number II

题目

Given an integer `n`, return *the* `nth` ***ugly number***.

Ugly number is a positive number whose prime factors only include `2`, `3`, and/or `5`.

Example 1:

```
Input: n = 10
Output: 12
Explanation: [1, 2, 3, 4, 5, 6, 8, 9, 10, 12] is the sequence of the first 10 ugly
numbers.
```

Example 2:

```
Input: n = 1
Output: 1
Explanation: 1 is typically treated as an ugly number.
```

Constraints:

- `1 <= n <= 1690`

题目大意

给你一个整数 `n` ，请你找出并返回第 `n` 个 **丑数** 。**丑数** 就是只包含质因数 `2`、`3` 和/或 `5` 的正整数。

解题思路

- 解法一，生成丑数的方法：先用最小质因数 1，分别和 2，3，5 相乘，得到的数是丑数，不断的将这些数分

别和 2，3，5 相乘，得到的数去重以后，从小到大排列，第 n 个数即为所求。排序可用最小堆实现，去重用 map 去重。时间复杂度 O(n log n)，空间复杂度 O(n)

- 上面的解法耗时在排序中，需要排序的根源是小的丑数乘以 5 大于了大的丑数乘以 2 。如何保证每次乘积以后，找出有序的丑数，是去掉排序，提升时间复杂度的关键。举个例子很容易想通：初始状态丑数只有 {1}，乘以 2，3，5 以后，将最小的结果存入集合中 {1,2}。下一轮再相乘，由于上一轮 1 已经和 2 相乘过了，1 不要再和 2 相乘了，所以这一轮 1 和 3，5 相乘。2 和 2，3，5 相乘。将最小的结果存入集合中 {1,2,3}，按照这样的策略往下比较，每轮选出的丑数是有序且不重复的。具体实现利用 3 个指针和一个数组即可实现。时间复杂度 O(n)，空间复杂度 O(n)。

代码

```go
package leetcode

func nthUglyNumber(n int) int {
  dp, p2, p3, p5 := make([]int, n+1), 1, 1, 1
  dp[0], dp[1] = 0, 1
  for i := 2; i <= n; i++ {
    x2, x3, x5 := dp[p2]*2, dp[p3]*3, dp[p5]*5
    dp[i] = min(min(x2, x3), x5)
    if dp[i] == x2 {
      p2++
    }
    if dp[i] == x3 {
      p3++
    }
    if dp[i] == x5 {
      p5++
    }
  }
  return dp[n]
}

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

268. Missing Number

题目

Given an array containing n distinct numbers taken from `0, 1, 2, ..., n`, find the one that is missing from the array.

Example 1:

```
Input: [3,0,1]
Output: 2
```

Example 2:

```
Input: [9,6,4,2,3,5,7,0,1]
Output: 8
```

Note:Your algorithm should run in linear runtime complexity. Could you implement it using only constant extra space complexity?

题目大意

给定一个包含 0, 1, 2, ..., n 中 n 个数的序列，找出 0 .. n 中没有出现在序列中的那个数。算法应该具有线性时间复杂度。你能否仅使用额外常数空间来实现?

解题思路

- 要求找出 `0, 1, 2, ..., n` 中缺失的那个数。还是利用异或的性质，`X^X = 0`。这里我们需要构造一个 X，用数组下标就可以了。数字下标是从 `[0, n-1]`，数字是 `[0, n]`，依次把数组里面的数组进行异或，把结果和 n 再异或一次，中和掉出现的数字，剩下的那个数字就是之前没有出现过的，缺失的数字。

代码

```go
package leetcode

func missingNumber(nums []int) int {
  xor, i := 0, 0
  for i = 0; i < len(nums); i++ {
    xor = xor ^ i ^ nums[i]
  }
  return xor ^ i
}
```

274. H-Index

题目

Given an array of citations (each citation is a non-negative integer) of a researcher, write a function to compute the researcher's h-index.

According to the definition of h-index on Wikipedia: "A scientist has index h if h of his/her N papers have at least h citations each, and the other N − h papers have no more than h citations each."

Example 1:

```
Input: citations = [3,0,6,1,5]
Output: 3
Explanation: [3,0,6,1,5] means the researcher has 5 papers in total and each of them
had
             received 3, 0, 6, 1, 5 citations respectively.
             Since the researcher has 3 papers with at least 3 citations each and the
remaining
             two with no more than 3 citations each, her h-index is 3.
```

Note:

If there are several possible values for h, the maximum one is taken as the h-index.

题目大意

求 h-index。h-index 值的定义：如果他/她的 N 篇论文中至少有 h 引用，而其他 N-h 论文的引用数不超过 h 引用数。

解题思路

可以先将数组里面的数从小到大排序。因为要找最大的 h-index，所以从数组末尾开始往前找，找到第一个数组的值，小于，总长度减去下标的值，这个值就是 h-index。

代码

```go
package leetcode

// 解法一
func hIndex(citations []int) int {
  n := len(citations)
  buckets := make([]int, n+1)
  for _, c := range citations {
    if c >= n {
      buckets[n]++
    } else {
      buckets[c]++
    }
  }
  count := 0
  for i := n; i >= 0; i-- {
    count += buckets[i]
    if count >= i {
```

```
        return i
      }
    }
    return 0
}

// 解法二
func hIndex1(citations []int) int {
    quickSort164(citations, 0, len(citations)-1)
    hIndex := 0
    for i := len(citations) - 1; i >= 0; i-- {
      if citations[i] >= len(citations)-i {
        hIndex++
      } else {
        break
      }
    }
    return hIndex
}
```

275. H-Index II

题目

Given an array of citations **sorted in ascending order** (each citation is a non-negative integer) of a researcher, write a function to compute the researcher's h-index.

According to the definition of h-index on Wikipedia: "A scientist has index h if h of his/her N papers have **at least** h citations each, and the other N − h papers have **no more than** h citations each."

Example:

```
Input: citations = [0,1,3,5,6]
Output: 3
Explanation: [0,1,3,5,6] means the researcher has 5 papers in total and each of them had
             received 0, 1, 3, 5, 6 citations respectively.
             Since the researcher has 3 papers with at least 3 citations each and the remaining
             two with no more than 3 citations each, her h-index is 3.
```

Note:

If there are several possible values for *h*, the maximum one is taken as the h-index.

Follow up:

- This is a follow up problem to H-Index, where `citations` is now guaranteed to be sorted in ascending order.

- Could you solve it in logarithmic time complexity?

题目大意

给定一位研究者论文被引用次数的数组（被引用次数是非负整数），数组已经按照升序排列。编写一个方法，计算出研究者的 h 指数。

h 指数的定义: “h 代表“高引用次数”（high citations），一名科研人员的 h 指数是指他（她）的 （N 篇论文中）至多有 h 篇论文分别被引用了至少 h 次。（其余的 N - h 篇论文每篇被引用次数不多于 h 次。）"

说明:

- 如果 h 有多有种可能的值 ，h 指数是其中最大的那个。

进阶：

- 这是 H 指数 的延伸题目，本题中的 citations 数组是保证有序的。
  你可以优化你的算法到对数时间复杂度吗?

解题思路

- 给出一个数组，代表该作者论文被引用次数，要求这个作者的 h 指数。h 指数定义："高引用次数”（`high citations`），一名科研人员的 h 指数是指他（她）的 （N 篇论文中）至多有 h 篇论文分别被引用了至少 h 次。（其余的 N - h 篇论文每篇被引用次数不多于 h 次。）
- 这一题要找出 h 指数，即要找到一个边界，在这个边界上为最多的 h 指数。可以用二分搜索来解决这道题。当 `len(citations)-mid > citations[mid]` 时，说明 h 指数的边界一定在右边，因为最多 `len(citations)-mid` 篇数比引用数 `citations[mid]` 还要大。否则 h 指数的边界在左边界，缩小边界以后继续二分。找到边界以后，最终求的是 h 指数，用 `len(citations) - low` 即是结果。

代码

```go
package leetcode

func hIndex275(citations []int) int {
  low, high := 0, len(citations)-1
  for low <= high {
    mid := low + (high-low)>>1
    if len(citations)-mid > citations[mid] {
      low = mid + 1
    } else {
      high = mid - 1
    }
  }
  return len(citations) - low
}
```

278. First Bad Version

题目

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have `n` versions `[1, 2, ..., n]` and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API `bool isBadVersion(version)` which returns whether `version` is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

Example 1:

```
Input: n = 5, bad = 4
Output: 4
Explanation:
call isBadVersion(3) -> false
call isBadVersion(5) -> true
call isBadVersion(4) -> true
Then 4 is the first bad version.
```

Example 2:

```
Input: n = 1, bad = 1
Output: 1
```

Constraints:

- `1 <= bad <= n <= 231 - 1`

题目大意

你是产品经理，目前正在带领一个团队开发新的产品。不幸的是，你的产品的最新版本没有通过质量检测。由于每个版本都是基于之前的版本开发的，所以错误的版本之后的所有版本都是错的。假设你有 n 个版本 [1, 2, ..., n]，你想找出导致之后所有版本出错的第一个错误的版本。你可以通过调用 bool isBadVersion(version) 接口来判断版本号 version 是否在单元测试中出错。实现一个函数来查找第一个错误的版本。你应该尽量减少对调用 API 的次数。

解题思路

- 我们知道开发产品迭代的版本，如果当一个版本为正确版本，则该版本之前的所有版本均为正确版本；当一个版本为错误版本，则该版本之后的所有版本均为错误版本。利用这个性质就可以进行二分查找。利用二分搜索，也可以满足减少对调用 API 的次数的要求。时间复杂度：O(logn)，其中 n 是给定版本的数量。空间复杂度：O(1)。

代码

```go
package leetcode

import "sort"

/**
 * Forward declaration of isBadVersion API.
 * @param   version   your guess about first bad version
 * @return          true if current version is bad
 *                false if current version is good
 * func isBadVersion(version int) bool;
 */

func firstBadVersion(n int) int {
  return sort.Search(n, func(version int) bool { return isBadVersion(version) })
}
```

279. Perfect Squares

题目

Given an integer `n`, return *the least number of perfect square numbers that sum to* `n`.

A **perfect square** is an integer that is the square of an integer; in other words, it is the product of some integer with itself. For example, `1`, `4`, `9`, and `16` are perfect squares while `3` and `11` are not.

Example 1:

```
Input: n = 12
Output: 3
Explanation: 12 = 4 + 4 + 4.
```

Example 2:

```
Input: n = 13
Output: 2
Explanation: 13 = 4 + 9.
```

Constraints:

- `1 <= n <= 104`

题目大意

给定正整数 n，找到若干个完全平方数（比如 1, 4, 9, 16, ...）使得它们的和等于 n。你需要让组成和的完全平方数的个数最少。给你一个整数 n ，返回和为 n 的完全平方数的 最少数量 。

完全平方数 是一个整数，其值等于另一个整数的平方；换句话说，其值等于一个整数自乘的积。例如，1、4、9 和 16 都是完全平方数，而 3 和 11 不是。

解题思路

- 由拉格朗日的四平方定理可得，每个自然数都可以表示为四个整数平方之和。 其中四个数字是整数。四平方和定理证明了任意一个正整数都可以被表示为至多四个正整数的平方和。这给出了本题的答案的上界。
- 四平方和定理可以推出三平方和推论：当且仅当 {{< katex >}} n \neq 4^{k} \times (8*m + 7){{< /katex >}} 时，n 可以被表示为至多三个正整数的平方和。所以当 {{< katex >}} n = 4^{k} * (8*m + 7){{< /katex >}} 时，n 只能被表示为四个正整数的平方和。此时我们可以直接返回 4。
- 当 {{< katex >}} n \neq 4^{k} \times (8*m + 7){{< /katex >}} 时，需要判断 n 到底可以分解成几个完全平方数之和。答案肯定是 1，2，3 中的一个。题目要求我们求最小的，所以从 1 开始一个个判断是否满足。如果答案为 1，代表 n 为完全平方数，这很好判断。如果答案为 2，代表 {{< katex >}} n = a^{2} + b^{2} {{< /katex >}}，枚举 {{< katex >}} 1 \leqslant a \leqslant \sqrt{n} {{< /katex >}}，判断 {{< katex >}} n - a^{2} {{< /katex >}} 是否为完全平方数。当 1 和 2 都排除了，剩下的答案只能为 3 了。

代码

```
package leetcode

import "math"

func numSquares(n int) int {
  if isPerfectSquare(n) {
    return 1
  }
  if checkAnswer4(n) {
    return 4
  }
  for i := 1; i*i <= n; i++ {
    j := n - i*i
    if isPerfectSquare(j) {
      return 2
    }
  }
  return 3
}

// 判断是否为完全平方数
func isPerfectSquare(n int) bool {
  sq := int(math.Floor(math.Sqrt(float64(n))))
  return sq*sq == n
}

// 判断是否能表示为 4^k*(8m+7)
func checkAnswer4(x int) bool {
  for x%4 == 0 {
    x /= 4
```

```
  }
  return x%8 == 7
}
```

283. Move Zeroes

题目

Given an array nums, write a function to move all 0's to the end of it while maintaining the relative order of the non-zero elements.

Example 1:

```
Input: [0,1,0,3,12]
Output: [1,3,12,0,0]
```

Note:

- You must do this in-place without making a copy of the array.
- Minimize the total number of operations.

题目大意

题目要求不能采用额外的辅助空间，将数组中 0 元素都移动到数组的末尾，并且维持所有非 0 元素的相对位置。

解题思路

这一题可以只扫描数组一遍，不断的用 i，j 标记 0 和非 0 的元素，然后相互交换，最终到达题目的目的。与这一题相近的题目有第 26 题，第 27 题，第 80 题。

代码

```go
package leetcode

func moveZeroes(nums []int) {
  if len(nums) == 0 {
    return
  }
  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++
```

```
        } else {
          j++
        }
      }
    }
}
```

284. Peeking Iterator

题目

Given an Iterator class interface with methods: `next()` and `hasNext()`, design and implement a PeekingIterator that support the `peek()` operation -- it essentially peek() at the element that will be returned by the next call to next().

Example:

```
Assume that the iterator is initialized to the beginning of the list: [1,2,3].

Call next() gets you 1, the first element in the list.
Now you call peek() and it returns 2, the next element. Calling next() after that still
return 2.
You call next() the final time and it returns 3, the last element.
Calling hasNext() after that should return false.
```

Follow up: How would you extend your design to be generic and work with all types, not just integer?

题目大意

给定一个迭代器类的接口，接口包含两个方法： next() 和 hasNext()。设计并实现一个支持 peek() 操作的顶端迭代器 -- 其本质就是把原本应由 next() 方法返回的元素 peek() 出来。

> peek() 是偷看的意思。偷偷看一看下一个元素是什么，但是并不是 next() 访问。

解题思路

- 简单题。在 PeekingIterator 内部保存 2 个变量，一个是下一个元素值，另一个是是否有下一个元素。在 next() 操作和 hasNext() 操作时，访问保存的这 2 个变量。peek() 操作也比较简单，判断是否有下一个元素，如果有，即返回该元素值。这里实现了迭代指针不移动的功能。如果没有保存下一个元素值，即没有 peek() 偷看，next() 操作继续往后移动指针，读取后一位元素。
- 这里复用了是否有下一个元素值，来判断 hasNext() 和 peek() 操作中不移动指针的逻辑。

代码

```
package leetcode

//Below is the interface for Iterator, which is already defined for you.
```

```go
type Iterator struct {
}

func (this *Iterator) hasNext() bool {
  // Returns true if the iteration has more elements.
  return true
}

func (this *Iterator) next() int {
  // Returns the next element in the iteration.
  return 0
}

type PeekingIterator struct {
  nextEl int
  hasEl  bool
  iter   *Iterator
}

func Constructor(iter *Iterator) *PeekingIterator {
  return &PeekingIterator{
    iter: iter,
  }
}

func (this *PeekingIterator) hasNext() bool {
  if this.hasEl {
    return true
  }
  return this.iter.hasNext()
}

func (this *PeekingIterator) next() int {
  if this.hasEl {
    this.hasEl = false
    return this.nextEl
  } else {
    return this.iter.next()
  }
}

func (this *PeekingIterator) peek() int {
  if this.hasEl {
    return this.nextEl
  }
  this.hasEl = true
  this.nextEl = this.iter.next()
  return this.nextEl
```

```
}
```

287. Find the Duplicate Number

题目

Given an array nums containing n + 1 integers where each integer is between 1 and n (inclusive), prove that at least one duplicate number must exist. Assume that there is only one duplicate number, find the duplicate one.

Example 1:

```
Input: [1,3,4,2,2]
Output: 2
```

Example 2:

```
Input: [3,1,3,4,2]
Output: 3
```

Note:

- You must not modify the array (assume the array is read only).
- You must use only constant, O(1) extra space.
- Your runtime complexity should be less than O(n^2).
- There is only one duplicate number in the array, but it could be repeated more than once.

题目大意

给出 n + 1 个数，这些数是在 1-n 中取值的，同一个数字可以出现多次。要求找出这些数中重复的数字。时间复杂度最好低于 O(n^2)，空间复杂度为 O(1)。

解题思路

- 这道题比较巧的思路是，将这些数字想象成链表中的结点，数组中数字代表下一个结点的数组下标。找重复的数字就是找链表中成环的那个点。由于题目保证了一定会有重复的数字，所以一定会成环。所以用快慢指针的方法，快指针一次走 2 步，慢指针一次走 1 步，相交以后，快指针从头开始，每次走一步，再次遇见的时候就是成环的交点处，也即是重复数字所在的地方。
- 这一题有多种做法。可以用快慢指针求解。还可以用二分搜索：(这里的题解感谢 @imageslr 指出错误）：
    1. 假设有 n+1 个数，则可能重复的数位于区间 [1, n] 中。记该区间最小值、最大值和中间值为 low、high、mid
    2. 遍历整个数组，统计小于等于 mid 的整数的个数，至多为 mid 个
    3. 如果超过 mid 个就说明重复的数存在于区间 [low,mid] （闭区间）中；否则，重复的数存在于区间

(mid, high] （左开右闭）中
4. 缩小区间，继续重复步骤 2、3，直到区间变成 1 个整数，即 low == high
5. 整数 low 就是要找的重复的数

- 另外一个做法是，先将数组排序，依照下标是从 0 开始递增的特性，那么数组里面的数字与下标的差值应该是越来越大。如果出现了相同的数字，下标变大，差值应该比前一个数字小，出现了这个情况就说明找到了相同数字了。

代码

```go
package leetcode

import "sort"

// 解法一 快慢指针
func findDuplicate(nums []int) int {
  slow := nums[0]
  fast := nums[nums[0]]
  for fast != slow {
    slow = nums[slow]
    fast = nums[nums[fast]]
  }
  walker := 0
  for walker != slow {
    walker = nums[walker]
    slow = nums[slow]
  }
  return walker
}

// 解法二 二分搜索
func findDuplicate1(nums []int) int {
  low, high := 0, len(nums)-1
  for low < high {
    mid, count := low+(high-low)>>1, 0
    for _, num := range nums {
      if num <= mid {
        count++
      }
    }
    if count > mid {
      high = mid
    } else {
      low = mid + 1
    }
  }
  return low
}
```

```go
// 解法三
func findDuplicate2(nums []int) int {
  if len(nums) == 0 {
    return 0
  }
  sort.Ints(nums)
  diff := -1
  for i := 0; i < len(nums); i++ {
    if nums[i]-i-1 >= diff {
      diff = nums[i] - i - 1
    } else {
      return nums[i]
    }
  }
  return 0
}
```

290. Word Pattern

题目

Given a pattern and a string str, find if str follows the same pattern.

Here follow means a full match, such that there is a bijection between a letter in pattern and a non-empty word in str.

Example 1:

```
Input: pattern = "abba", str = "dog cat cat dog"
Output: true
```

Example 2:

```
Input:pattern = "abba", str = "dog cat cat fish"
Output: false
```

Example 3:

```
Input: pattern = "aaaa", str = "dog cat cat dog"
Output: false
```

Example 4:

```
Input: pattern = "abba", str = "dog dog dog dog"
Output: false
```

Note:

You may assume pattern contains only lowercase letters, and str contains lowercase letters separated by a single space.

题目大意

给定一个模式串，判断字符串是否和给定的模式串，是一样的模式。

解题思路

这道题用 2 个 map 即可。1 个 map 记录模式与字符串的匹配关系，另外一个 map 记录字符串和模式的匹配关系。为什么需要记录双向的关系呢？因为 Example 4 中，a 对应了 dog，这个时候 b 如果再对应 dog 是错误的，所以这里需要从 dog 查询它是否已经和某个模式匹配过了。所以需要双向的关系。

代码

```go
package leetcode

import "strings"

func wordPattern(pattern string, str string) bool {
  strList := strings.Split(str, " ")
  patternByte := []byte(pattern)
  if pattern == "" || len(patternByte) != len(strList) {
    return false
  }

  pMap := map[byte]string{}
  sMap := map[string]byte{}
  for index, b := range patternByte {
    if _, ok := pMap[b]; !ok {
      if _, ok = sMap[strList[index]]; !ok {
        pMap[b] = strList[index]
        sMap[strList[index]] = b
      } else {
        if sMap[strList[index]] != b {
```

```
            return false
          }
        }
      } else {
        if pMap[b] != strList[index] {
          return false
        }
      }
    }
    return true
  }
```

297. Serialize and Deserialize Binary Tree

题目

Serialization is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link to be reconstructed later in the same or another computer environment.

Design an algorithm to serialize and deserialize a binary tree. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that a binary tree can be serialized to a string and this string can be deserialized to the original tree structure.

Clarification: The input/output format is the same as [how LeetCode serializes a binary tree](). You do not necessarily need to follow this format, so please be creative and come up with different approaches yourself.

Example 1:

```
Input: root = [1,2,3,null,null,4,5]
Output: [1,2,3,null,null,4,5]
```

Example 2:

```
Input: root = []
Output: []
```

Example 3:

```
Input: root = [1]
Output: [1]
```

Example 4:

```
Input: root = [1,2]
Output: [1,2]
```

Constraints:

- The number of nodes in the tree is in the range `[0, 104]`.
- `1000 <= Node.val <= 1000`

题目大意

设计一个算法，来序列化和反序列化二叉树。并不限制如何进行序列化和反序列化，但是你需要保证二叉树可以序列化为字符串，并且这个字符串可以被反序列化成原有的二叉树。

解题思路

- 1. 将给定的二叉树想象成一颗满二叉树(不存在的结点用 null 填充)。
- 2. 通过前序遍历，可以得到一个第一个结点为根的序列，然后递归进行序列化/反序列化即可。

代码

```go
package leetcode

import (
  "strconv"
  "strings"

  "github.com/halfrost/LeetCode-Go/structures"
)

type TreeNode = structures.TreeNode

type Codec struct {
```

```go
  builder strings.Builder
  input   []string
}

func Constructor() Codec {
  return Codec{}
}

// Serializes a tree to a single string.
func (this *Codec) serialize(root *TreeNode) string {
  if root == nil {
    this.builder.WriteString("#,")
    return ""
  }
  this.builder.WriteString(strconv.Itoa(root.Val) + ",")
  this.serialize(root.Left)
  this.serialize(root.Right)
  return this.builder.String()
}

// Deserializes your encoded data to tree.
func (this *Codec) deserialize(data string) *TreeNode {
  if len(data) == 0 {
    return nil
  }
  this.input = strings.Split(data, ",")
  return this.deserializeHelper()
}

func (this *Codec) deserializeHelper() *TreeNode {
  if this.input[0] == "#" {
    this.input = this.input[1:]
    return nil
  }
  val, _ := strconv.Atoi(this.input[0])
  this.input = this.input[1:]
  return &TreeNode{
    Val:   val,
    Left:  this.deserializeHelper(),
    Right: this.deserializeHelper(),
  }
}
```

300. Longest Increasing Subsequence

题目

Given an unsorted array of integers, find the length of longest increasing subsequence.

Example:

```
Input: [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.
```

Note:

- There may be more than one LIS combination, it is only necessary for you to return the length.
- Your algorithm should run in O(n^2) complexity.

Follow up: Could you improve it to O(n log n) time complexity?

题目大意

给定一个无序的整数数组，找到其中最长上升子序列的长度。

解题思路

- 给定一个整数序列，求其中的最长上升子序列的长度。这一题就是经典的 LIS 最长上升子序列的问题。
- `dp[i]` 代表为第 i 个数字为结尾的最长上升子序列的长度。换种表述，dp[i] 代表 [0,i] 范围内，选择数字 nums[i] 可以获得的最长上升子序列的长度。状态转移方程为 `dp[i] = max( 1 + dp[j]) ，其中 j < i && nums[j] > nums[i]`，取所有满足条件的最大值。时间复杂度 O(n^2)
- 这道题还有一种更快的解法。考虑这样一个问题，我们是否能用一个数组，记录上升子序列的最末尾的一个数字呢？如果这个数字越小，那么这个子序列往后面添加数字的几率就越大，那么就越可能成为最长的上升子序列。举个例子：nums = [4,5,6,3]，它的所有的上升子序列为

```
len = 1   :      [4], [5], [6], [3]   => tails[0] = 3
len = 2   :      [4, 5], [5, 6]       => tails[1] = 5
len = 3   :      [4, 5, 6]            => tails[2] = 6
```

- 其中 `tails[i]` 中存储的是所有长度为 i + 1 的上升子序列中末尾最小的值。也很容易证明 `tails` 数组里面的值一定是递增的(因为我们用末尾的数字描述最长递增子序列)。既然 tails 是有序的，我们就可以用二分查找的方法去更新这个 tail 数组里面的值。更新策略如下：(1). 如果 x 比所有的 tails 元素都要大，那么就直接放在末尾，并且 tails 数组长度加一，这里对应解法二中，二分搜索找不到对应的元素值，直接把 num 放在 dp[] 的最后；(2). 如果 `tails[i-1] < x <= tails[i]`，则更新 tails[i]，因为 x 更小，更能获得最长上升子序列，这一步对应解法二中将 dp[i] 更新为 num。最终 tails 数组的长度即为最长的上升子序列。这种做法的时间复杂度 O(n log n)。
- 此题是一维的 LIS 问题。二维的 LIS 问题是第 354 题。三维的 LIS 问题是第 1691 题。

代码

```go
package leetcode
```

```go
import "sort"

// 解法一 O(n^2) DP
func lengthOfLIS(nums []int) int {
  dp, res := make([]int, len(nums)+1), 0
  dp[0] = 0
  for i := 1; i <= len(nums); i++ {
    for j := 1; j < i; j++ {
      if nums[j-1] < nums[i-1] {
        dp[i] = max(dp[i], dp[j])
      }
    }
    dp[i] = dp[i] + 1
    res = max(res, dp[i])
  }
  return res
}

// 解法二 O(n log n) DP
func lengthOfLIS1(nums []int) int {
  dp := []int{}
  for _, num := range nums {
    i := sort.SearchInts(dp, num)
    if i == len(dp) {
      dp = append(dp, num)
    } else {
      dp[i] = num
    }
  }
  return len(dp)
}
```

303. Range Sum Query - Immutable

题目

Given an integer array nums, find the sum of the elements between indices i and j (i ≤ j), inclusive.

Example:

```
Given nums = [-2, 0, 3, -5, 2, -1]

sumRange(0, 2) -> 1
sumRange(2, 5) -> -1
sumRange(0, 5) -> -3
```

Note:

1. You may assume that the array does not change.
2. There are many calls to sumRange function.

题目大意

给定一个整数数组 nums，求出数组从索引 i 到 j (i ≤ j) 范围内元素的总和，包含 i, j 两点。

示例：

```
给定 nums = [-2, 0, 3, -5, 2, -1]，求和函数为 sumRange()

sumRange(0, 2) -> 1
sumRange(2, 5) -> -1
sumRange(0, 5) -> -3
```

说明:

- 你可以假设数组不可变。
- 会多次调用 sumRange 方法。

解题思路

- 给出一个数组，数组里面的数都是`**不可变**`的，设计一个数据结构能够满足查询数组任意区间内元素的和。
- 这一题由于数组里面的元素都是`**不可变**`的，所以可以用 2 种方式来解答，第一种解法是用 prefixSum，通过累计和相减的办法来计算区间内的元素和，初始化的时间复杂度是 O(n)，但是查询区间元素和的时间复杂度是 O(1)。第二种解法是利用线段树，构建一颗线段树，父结点内存的是两个子结点的和，初始化建树的时间复杂度是 O(log n)，查询区间元素和的时间复杂度是 O(log n)。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/template"
)

//解法一 线段树，sumRange 时间复杂度 O(1)

// NumArray define
type NumArray struct {
  st *template.SegmentTree
}

// Constructor303 define
func Constructor303(nums []int) NumArray {
  st := template.SegmentTree{}
```

```go
  st.Init(nums, func(i, j int) int {
    return i + j
  })
  return NumArray{st: &st}
}

// SumRange define
func (ma *NumArray) SumRange(i int, j int) int {
  return ma.st.Query(i, j)
}

//解法二 prefixSum, sumRange 时间复杂度 O(1)

// // NumArray define
// type NumArray struct {
//  prefixSum []int
// }

// // Constructor303 define
// func Constructor303(nums []int) NumArray {
//  for i := 1; i < len(nums); i++ {
//    nums[i] += nums[i-1]
//  }
//  return NumArray{prefixSum: nums}
// }

// // SumRange define
// func (this *NumArray) SumRange(i int, j int) int {
//  if i > 0 {
//    return this.prefixSum[j] - this.prefixSum[i-1]
//  }
//  return this.prefixSum[j]
// }

/**
 * Your NumArray object will be instantiated and called as such:
 * obj := Constructor(nums);
 * param_1 := obj.SumRange(i,j);
 */
```

304. Range Sum Query 2D - Immutable

题目

Given a 2D matrix matrix, find the sum of the elements inside the rectangle defined by its upper left corner (row1, col1) and lower right corner (row2, col2).

The above rectangle (with the red border) is defined by (row1, col1) = **(2, 1)** and (row2, col2) = **(4, 3)**, which contains sum = **8**.

Example:

```
Given matrix = [
  [3, 0, 1, 4, 2],
  [5, 6, 3, 2, 1],
  [1, 2, 0, 1, 5],
  [4, 1, 0, 1, 7],
  [1, 0, 3, 0, 5]
]

sumRegion(2, 1, 4, 3) -> 8
sumRegion(1, 1, 2, 2) -> 11
sumRegion(1, 2, 2, 4) -> 12
```

Note:

1. You may assume that the matrix does not change.
2. There are many calls to sumRegion function.
3. You may assume that row1 ≤ row2 and col1 ≤ col2.

题目大意

给定一个二维矩阵，计算其子矩形范围内元素的总和，该子矩阵的左上角为 (row1, col1) ，右下角为 (row2, col2) 。

解题思路

- 这一题是一维数组前缀和的进阶版本。定义 f(x,y) 代表矩形左上角 (0,0)，右下角 (x,y) 内的元素和。{{< katex display >}} f(i,j) = \sum*{x=0}^{i}\sum*{y=0}^{j} Matrix[x][y]{{< /katex >}}

  {{< katex display >}}
  \begin{aligned}
  f(i,j) &= \sum*{x=0}^{i-1}\sum*{y=0}^{j-1} Matrix[x][y] + \sum*{x=0}^{i-1} Matrix[x][j] + \sum*{y=0}^{j-1} Matrix[i][y] + Matrix[i][j]\
  &= (\sum*{x=0}^{i-1}\sum*{y=0}^{j-1} Matrix[x][y] + \sum*{x=0}^{i-1} Matrix[x][j]) + (\sum*{x=0}^{i-1}\sum_{y=0}^{j-1} Matrix[x][y] + \sum*{y=0}^{j-1} Matrix[i][y]) - \sum*{x=0}^{i-1}\sum_{y=0}^{j-1} Matrix[x][y] + Matrix[i][j]\
  &= \sum*{x=0}^{i-1}\sum*{y=0}^{j} Matrix[x][y] + \sum*{x=0}^{i}\sum*{y=0}^{j-1} Matrix[x][y] - \sum*{x=0}^{i-1}\sum*{y=0}^{j-1} Matrix[x][y] + Matrix[i][j]\
  &= f(i-1,j) + f(i,j-1) - f(i-1,j-1) + Matrix[i][j]
  \end{aligned}
  {{< /katex >}}

- 于是得到递推的关系式：`f(i, j) = f(i-1, j) + f(i, j-1) - f(i-1, j-1) + matrix[i][j]`，写代码为了方便，新建一个 `m+1 * n+1` 的矩阵，这样就不需要对 `row = 0` 和 `col = 0` 做单独处理了。上述推导公式如果画成图也很好理解：

  左图中大的矩形由粉红色的矩形 + 绿色矩形 - 粉红色和绿色重叠部分 + 黄色部分。这就对应的是上面推导出来的递推公式。左图是矩形左上角为 (0，0) 的情况，更加一般的情况是右图，左上角是任意的坐标，公式不变。

- 时间复杂度：初始化 O(mn)，查询 O(1)。空间复杂度 O(mn)

代码

```
package leetcode
```

```go
type NumMatrix struct {
  cumsum [][]int
}

func Constructor(matrix [][]int) NumMatrix {
  if len(matrix) == 0 {
    return NumMatrix{nil}
  }
  cumsum := make([][]int, len(matrix)+1)
  cumsum[0] = make([]int, len(matrix[0])+1)
  for i := range matrix {
    cumsum[i+1] = make([]int, len(matrix[i])+1)
    for j := range matrix[i] {
      cumsum[i+1][j+1] = matrix[i][j] + cumsum[i][j+1] + cumsum[i+1][j] - cumsum[i][j]
    }
  }
  return NumMatrix{cumsum}
}

func (this *NumMatrix) SumRegion(row1 int, col1 int, row2 int, col2 int) int {
  cumsum := this.cumsum
  return cumsum[row2+1][col2+1] - cumsum[row1][col2+1] - cumsum[row2+1][col1] +
cumsum[row1][col1]
}

/**
 * Your NumMatrix object will be instantiated and called as such:
 * obj := Constructor(matrix);
 * param_1 := obj.SumRegion(row1,col1,row2,col2);
 */
```

306. Additive Number

题目

Additive number is a string whose digits can form additive sequence.

A valid additive sequence should contain **at least** three numbers. Except for the first two numbers, each subsequent number in the sequence must be the sum of the preceding two.

Given a string containing only digits `'0'-'9'`, write a function to determine if it's an additive number.

Note: Numbers in the additive sequence **cannot** have leading zeros, so sequence `1, 2, 03` or `1, 02, 3` is invalid.

Example 1:

```
Input: "112358"
Output: true
Explanation: The digits can form an additive sequence: 1, 1, 2, 3, 5, 8.
             1 + 1 = 2, 1 + 2 = 3, 2 + 3 = 5, 3 + 5 = 8
```

Example 2:

```
Input: "199100199"
Output: true
Explanation: The additive sequence is: 1, 99, 100, 199.
             1 + 99 = 100, 99 + 100 = 199
```

Follow up:How would you handle overflow for very large input integers?

题目大意

累加数是一个字符串，组成它的数字可以形成累加序列。一个有效的累加序列必须至少包含 3 个数。除了最开始的两个数以外，字符串中的其他数都等于它之前两个数相加的和。给定一个只包含数字 '0'-'9' 的字符串，编写一个算法来判断给定输入是否是累加数。说明: 累加序列里的数不会以 0 开头，所以不会出现 1, 2, 03 或者 1, 02, 3 的情况。

解题思路

- 在给出的字符串中判断该字符串是否为斐波那契数列形式的字符串。
- 由于每次判断需要累加 2 个数字，所以在 DFS 遍历的过程中需要维护 2 个数的边界，`firstEnd` 和 `secondEnd`，两个数加起来的和数的起始位置是 `secondEnd + 1`。每次在移动 `firstEnd` 和 `secondEnd` 的时候，需要判断 `strings.HasPrefix(num[secondEnd + 1:], strconv.Itoa(x1 + x2))`，即后面的字符串中是否以和为开头。
- 如果第一个数字起始数字出现了 0 ，或者第二个数字起始数字出现了 0，都算非法异常情况，都应该直接返回 false。

代码

```go
package leetcode

import (
  "strconv"
  "strings"
)

// This function controls various combinations as starting points
func isAdditiveNumber(num string) bool {
  if len(num) < 3 {
    return false
```

```go
  }
  for firstEnd := 0; firstEnd < len(num)/2; firstEnd++ {
    if num[0] == '0' && firstEnd > 0 {
      break
    }
    first, _ := strconv.Atoi(num[:firstEnd+1])
    for secondEnd := firstEnd + 1; max(firstEnd, secondEnd-firstEnd) <= len(num)-
secondEnd; secondEnd++ {
      if num[firstEnd+1] == '0' && secondEnd-firstEnd > 1 {
        break
      }
      second, _ := strconv.Atoi(num[firstEnd+1 : secondEnd+1])
      if recursiveCheck(num, first, second, secondEnd+1) {
        return true
      }
    }
  }
  return false
}

//Propagate for rest of the string
func recursiveCheck(num string, x1 int, x2 int, left int) bool {
  if left == len(num) {
    return true
  }
  if strings.HasPrefix(num[left:], strconv.Itoa(x1+x2)) {
    return recursiveCheck(num, x2, x1+x2, left+len(strconv.Itoa(x1+x2)))
  }
  return false
}
```

307. Range Sum Query - Mutable

题目

Given an integer array nums, find the sum of the elements between indices i and j (i ≤ j), inclusive.

The update(i, val) function modifies nums by updating the element at index i to val.

Example:

```
Given nums = [1, 3, 5]

sumRange(0, 2) -> 9
update(1, 2)
sumRange(0, 2) -> 8
```

Note:

1. The array is only modifiable by the update function.
2. You may assume the number of calls to update and sumRange function is distributed evenly.

题目大意

给定一个整数数组 nums，求出数组从索引 i 到 j (i ≤ j) 范围内元素的总和，包含 i, j 两点。

update(i, val) 函数可以通过将下标为 i 的数值更新为 val，从而对数列进行修改。

示例:

```
Given nums = [1, 3, 5]

sumRange(0, 2) -> 9
update(1, 2)
sumRange(0, 2) -> 8
```

说明:

- 数组仅可以在 update 函数下进行修改。
- 你可以假设 update 函数与 sumRange 函数的调用次数是均匀分布的。

解题思路

- 给出一个数组，数组里面的数都是`**可变**`的，设计一个数据结构能够满足查询数组任意区间内元素的和。
- 对比第 303 题，这一题由于数组里面的元素都是 **可变** 的，所以第一个想到的解法就是线段树，构建一颗线段树，父结点内存的是两个子结点的和，初始化建树的时间复杂度是 O(log n)，查询区间元素和的时间复杂度是 O(log n)，更新元素值的时间复杂度是 O(log n)。
- 如果此题还用 prefixSum 的思路解答呢？那每次 update 操作的时间复杂度都是 O(n)，因为每次更改一个值，最坏情况就是所有的 prefixSum 都要更新一次。prefixSum 的方法在这道题上面也可以 AC，只不过时间排名在 5%，非常差。
- 此题也可以用树状数组解决。代码很直白，区间查询即是两个区间前缀和相减。最简单的树状数组应用。

代码

```go
package leetcode

import "github.com/halfrost/LeetCode-Go/template"

// NumArray define
type NumArray struct {
  st *template.SegmentTree
}

// Constructor307 define
func Constructor307(nums []int) NumArray {
  st := template.SegmentTree{}
  st.Init(nums, func(i, j int) int {
```

```go
    return i + j
  })
  return NumArray{st: &st}
}

// Update define
func (this *NumArray) Update(i int, val int) {
  this.st.Update(i, val)
}

// SumRange define
func (this *NumArray) SumRange(i int, j int) int {
  return this.st.Query(i, j)
}

//解法二 prefixSum, sumRange 时间复杂度 O(1)

// // NumArray define
// type NumArray307 struct {
//  prefixSum []int
//  data   []int
// }

// // Constructor307 define
// func Constructor307(nums []int) NumArray307 {
//  data := make([]int, len(nums))
//  for i := 0; i < len(nums); i++ {
//    data[i] = nums[i]
//  }
//  for i := 1; i < len(nums); i++ {
//    nums[i] += nums[i-1]
//  }
//  return NumArray307{prefixSum: nums, data: data}
// }

// // Update define
// func (this *NumArray307) Update(i int, val int) {
//  this.data[i] = val
//  this.prefixSum[0] = this.data[0]
//  for i := 1; i < len(this.data); i++ {
//    this.prefixSum[i] = this.prefixSum[i-1] + this.data[i]
//  }
// }

// // SumRange define
// func (this *NumArray307) SumRange(i int, j int) int {
//  if i > 0 {
//    return this.prefixSum[j] - this.prefixSum[i-1]
//  }
```

```
//  return this.prefixSum[j]
// }

// 解法三 树状数组
// type NumArray struct {
//  bit  template.BinaryIndexedTree
//  data []int
// }

// // Constructor define
// func Constructor307(nums []int) NumArray {
//  bit := template.BinaryIndexedTree{}
//  bit.InitWithNums(nums)
//  return NumArray{bit: bit, data: nums}
// }

// // Update define
// func (this *NumArray) Update(i int, val int) {
//  this.bit.Add(i+1, val-this.data[i])
//  this.data[i] = val
// }

// // SumRange define
// func (this *NumArray) SumRange(i int, j int) int {
//  return this.bit.Query(j+1) - this.bit.Query(i)
// }

/**
 * Your NumArray object will be instantiated and called as such:
 * obj := Constructor(nums);
 * obj.Update(i,val);
 * param_2 := obj.SumRange(i,j);
 */
```

309. Best Time to Buy and Sell Stock with Cooldown

题目

Say you have an array for which the ith element is the price of a given stock on day i.

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times) with the following restrictions:

- You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).
- After you sell your stock, you cannot buy stock on next day. (ie, cooldown 1 day)

Example:

```
Input: [1,2,3,0,2]
Output: 3
Explanation: transactions = [buy, sell, cooldown, buy, sell]
```

题目大意

给定一个整数数组，其中第 i 个元素代表了第 i 天的股票价格 。

设计一个算法计算出最大利润。在满足以下约束条件下，你可以尽可能地完成更多的交易（多次买卖一支股票）：

- 你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。
- 卖出股票后，你无法在第二天买入股票 (即冷冻期为 1 天)。

解题思路

- 给定一个数组，表示一支股票在每一天的价格。设计一个交易算法，在这些天进行自动交易，要求：每一天只能进行一次操作；在买完股票后，必须卖了股票，才能再次买入；每次卖了股票以后，在下一天是不能购买的。问如何交易，能让利润最大?
- 这一题是第 121 题和第 122 题的变种题。
- 每天都有 3 种操作，`buy`，`sell`，`cooldown`。`sell` 之后的一天一定是 `cooldown`，但是 `cooldown` 可以出现在任意一天。例如：`buy, cooldown, cooldown, sell, cooldown, cooldown`。`buy[i]` 代表第 `i` 天通过 `buy` 或者 `cooldown` 结束此天能获得的最大收益。例如：`buy, sell, buy` 或者 `buy, cooldown, cooldown`。`sell[i]` 代表第 `i` 天通过 `sell` 或者 `cooldown` 结束此天能获得的最大收益。例如：`buy, sell, buy, sell` 或者 `buy, sell, cooldown, cooldown`。`price[i-1]` 代表第 `i` 天的股票价格(由于 price 是从 0 开始的)。
- 第 i 天如果是 sell，那么这天能获得的最大收益是 `buy[i - 1] + price[i - 1]`，因为只有 buy 了才能 sell。如果这一天是 cooldown，那么这天能获得的最大收益还是 sell[i - 1]。所以 sell[i] 的状态转移方程 `sell[i] = max(buy[i - 1] + price[i - 1], sell[i - 1])`。`sell[0] = 0` 代表第一天就卖了，由于第一天不持有股票，所以 sell[0] = 0。`sell[1] = max(sell[0], buy[0]+prices[1])` 代表第一天卖了，和第一天不卖，第二天卖做对比，钱多的保存至 sell[1]。
- 第 i 天如果是 buy，那么这天能获得的最大收益是 `sell[i - 2] - price[i - 1]`，因为 i - 1 天是 cooldown。如果这一天是 cooldown，那么这天能获得的最大收益还是 buy[i - 1]。所以 buy[i] 的状态转移方程 `buy[i] = max(sell[i - 2] - price[i - 1], buy[i - 1])`。`buy[0] = -prices[0]` 代表第一天就买入，所以金钱变成了负的。`buy[1] = max(buy[0], -prices[1])` 代表第一天不买入，第二天再买入。

代码

```go
package leetcode

import (
```

```go
  "math"
)

// 解法一 DP
func maxProfit309(prices []int) int {
  if len(prices) <= 1 {
    return 0
  }
  buy, sell := make([]int, len(prices)), make([]int, len(prices))
  for i := range buy {
    buy[i] = math.MinInt64
  }
  buy[0] = -prices[0]
  buy[1] = max(buy[0], -prices[1])
  sell[1] = max(sell[0], buy[0]+prices[1])
  for i := 2; i < len(prices); i++ {
    sell[i] = max(sell[i-1], buy[i-1]+prices[i])
    buy[i] = max(buy[i-1], sell[i-2]-prices[i])
  }
  return sell[len(sell)-1]
}

// 解法二 优化辅助空间的 DP
func maxProfit309_1(prices []int) int {
  if len(prices) <= 1 {
    return 0
  }
  buy := []int{-prices[0], max(-prices[0], -prices[1]), math.MinInt64}
  sell := []int{0, max(0, -prices[0]+prices[1]), 0}

  for i := 2; i < len(prices); i++ {
    sell[i%3] = max(sell[(i-1)%3], buy[(i-1)%3]+prices[i])
    buy[i%3] = max(buy[(i-1)%3], sell[(i-2)%3]-prices[i])
  }
  return sell[(len(prices)-1)%3]
}
```

315. Count of Smaller Numbers After Self

题目

You are given an integer array nums and you have to return a new counts array. The counts array has the property where `counts[i]` is the number of smaller elements to the right of `nums[i]`.

Example:

```
Input: [5,2,6,1]
Output: [2,1,1,0]
Explanation:
To the right of 5 there are 2 smaller elements (2 and 1).
To the right of 2 there is only 1 smaller element (1).
To the right of 6 there is 1 smaller element (1).
To the right of 1 there is 0 smaller element.
```

题目大意

给定一个整数数组 nums，按要求返回一个新数组 counts。数组 counts 有该性质： counts[i] 的值是 nums[i] 右侧小于 nums[i] 的元素的数量。

示例:

```
输入: [5,2,6,1]
输出: [2,1,1,0]
解释:
5 的右侧有 2 个更小的元素 (2 和 1).
2 的右侧仅有 1 个更小的元素 (1).
6 的右侧有 1 个更小的元素 (1).
1 的右侧有 0 个更小的元素.
```

解题思路

- 给出一个数组，要求输出数组中每个元素相对于数组中的位置右边比它小的元素。
- 这一题是第 327 题的缩水版。由于需要找数组位置右边比当前位置元素小的元素，所以从数组右边开始往左边扫。构造一颗线段树，线段树里面父节点存的是子节点出现的次数和。有可能给的数据会很大，所以构造线段树的时候先离散化。还需要注意的是数组里面可能有重复元素，所以构造线段树要先去重并排序。从右往左扫的过程中，依次添加数组中的元素，添加了一次就立即 query 一次。query 的区间是 [minNum, nums[i]-1]。如果是 minNum 则输出 0，并且也要记得插入这个最小值。这一题的思路和第 327 题大体类似，详解可见第 327 题。
- 这一题同样可以用树状数组来解答。相比 327 题简单很多。第一步还是把所有用到的元素放入 allNums 数组中，第二步排序 + 离散化。由于题目要求输出右侧更小的元素，所以第三步倒序插入构造树状数组，Query 查询 `[1,i-1]` 区间内元素总数即为右侧更小元素个数。注意最终输出是顺序输出，计算是逆序计算的，最终数组里面的答案还需要逆序一遍。相同的套路题有，第 327 题，第 493 题。

代码

```go
package leetcode

import (
  "sort"
```

```go
  "github.com/halfrost/LeetCode-Go/template"
)

// 解法一 线段树
func countSmaller(nums []int) []int {
  if len(nums) == 0 {
    return []int{}
  }
  st, minNum, numsMap, numsArray, res := template.SegmentCountTree{}, 0,
make(map[int]int, 0), []int{}, make([]int, len(nums))
  for i := 0; i < len(nums); i++ {
    numsMap[nums[i]] = nums[i]
  }
  for _, v := range numsMap {
    numsArray = append(numsArray, v)
  }
  // 排序是为了使得线段树中的区间 left <= right，如果此处不排序，线段树中的区间有很多不合法。
  sort.Ints(numsArray)
  minNum = numsArray[0]
  // 初始化线段树，节点内的值都赋值为 0，即计数为 0
  st.Init(numsArray, func(i, j int) int {
    return 0
  })
  for i := len(nums) - 1; i >= 0; i-- {
    if nums[i] == minNum {
      res[i] = 0
      st.UpdateCount(nums[i])
      continue
    }
    st.UpdateCount(nums[i])
    res[i] = st.Query(minNum, nums[i]-1)
  }
  return res
}

// 解法二 树状数组
func countSmaller1(nums []int) []int {
  // copy 一份原数组至所有数字 allNums 数组中
  allNums, res := make([]int, len(nums)), []int{}
  copy(allNums, nums)
  // 将 allNums 离散化
  sort.Ints(allNums)
  k := 1
  kth := map[int]int{allNums[0]: k}
  for i := 1; i < len(allNums); i++ {
    if allNums[i] != allNums[i-1] {
      k++
      kth[allNums[i]] = k
    }
```

```
  }
  // 树状数组 Query
  bit := template.BinaryIndexedTree{}
  bit.Init(k)
  for i := len(nums) - 1; i >= 0; i-- {
    res = append(res, bit.Query(kth[nums[i]]-1))
    bit.Add(kth[nums[i]], 1)
  }
  for i := 0; i < len(res)/2; i++ {
    res[i], res[len(res)-1-i] = res[len(res)-1-i], res[i]
  }
  return res
}
```

318. Maximum Product of Word Lengths

题目

Given a string array `words`, find the maximum value of `length(word[i]) * length(word[j])` where the two words do not share common letters. You may assume that each word will contain only lower case letters. If no such two words exist, return 0.

Example 1:

```
Input: ["abcw","baz","foo","bar","xtfn","abcdef"]
Output: 16
Explanation: The two words can be "abcw", "xtfn".
```

Example 2:

```
Input: ["a","ab","abc","d","cd","bcd","abcd"]
Output: 4
Explanation: The two words can be "ab", "cd".
```

Example 3:

```
Input: ["a","aa","aaa","aaaa"]
Output: 0
Explanation: No such pair of words.
```

题目大意

给定一个字符串数组 words，找到 length(word[i]) * length(word[j]) 的最大值，并且这两个单词不含有公共字母。你可以认为每个单词只包含小写字母。如果不存在这样的两个单词，返回 0。

解题思路

- 在字符串数组中找到 2 个没有公共字符的字符串，并且这两个字符串的长度乘积要是最大的，求这个最大的乘积。
- 这里需要利用位运算 `&` 运算的性质，如果 `X & Y = 0`，说明 X 和 Y 完全不相同。那么我们将字符串都编码成二进制数，进行 `&` 运算即可分出没有公共字符的字符串，最后动态维护长度乘积最大值即可。将字符串编码成二进制数的规则比较简单，每个字符相对于 'a' 的距离，根据这个距离将 1 左移多少位。

```
    a 1->1
    b 2->10
    c 4->100
    ab 3->11
    ac 5->101
    abc 7->111
    az 33554433->1000000000000000000000001
```

代码

```go
package leetcode

func maxProduct318(words []string) int {
  if words == nil || len(words) == 0 {
    return 0
  }
  length, value, maxProduct := len(words), make([]int, len(words)), 0
  for i := 0; i < length; i++ {
    tmp := words[i]
    value[i] = 0
    for j := 0; j < len(tmp); j++ {
      value[i] |= 1 << (tmp[j] - 'a')
    }
  }
  for i := 0; i < length; i++ {
    for j := i + 1; j < length; j++ {
      if (value[i]&value[j]) == 0 && (len(words[i])*len(words[j]) > maxProduct) {
        maxProduct = len(words[i]) * len(words[j])
      }
    }
  }
  return maxProduct
}
```

322. Coin Change

题目

You are given coins of different denominations and a total amount of money amount. Write a function to compute 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`.

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
```

Note:

You may assume that you have an infinite number of each kind of coin.

题目大意

给定不同面额的硬币 coins 和一个总金额 amount。编写一个函数来计算可以凑成总金额所需的最少的硬币个数。如果没有任何一种硬币组合能组成总金额，返回 -1。

解题思路

- 给出一些硬币和一个总数，问组成这个总数的硬币数最少是多少个?
- 这一题是经典的硬币问题，利用 DP 求解。不过这一题的测试用例有一个很大的值，这样开 DP 数组会比较浪费空间。例如 [1,1000000000,500000] 有这样的硬币种类，要求组成 2389412493027523 这样的总数。那么按照下面的解题方法，数组会开的很大，非常浪费空间。这个时候用 DFS 解题会节约一点空间。

代码

```go
package leetcode

func coinChange(coins []int, amount int) int {
  dp := make([]int, amount+1)
  dp[0] = 0
  for i := 1; i < len(dp); i++ {
    dp[i] = amount + 1
  }
  for i := 1; i <= amount; i++ {
```

```
    for j := 0; j < len(coins); j++ {
      if coins[j] <= i {
        dp[i] = min(dp[i], dp[i-coins[j]]+1)
      }
    }
  }
  if dp[amount] > amount {
    return -1
  }
  return dp[amount]
}
```

324. Wiggle Sort II

题目

Given an unsorted array nums, reorder it such that nums[0] < nums[1] > nums[2] < nums[3]....

Example 1:

```
Input: nums = [1, 5, 1, 1, 6, 4]
Output: One possible answer is [1, 4, 1, 5, 1, 6].
```

Example 2:

```
Input: nums = [1, 3, 2, 2, 3, 1]
Output: One possible answer is [2, 3, 1, 3, 1, 2].
```

Note:

You may assume all input has valid answer.

Follow up:

Can you do it in O(n) time and/or in-place with O(1) extra space?

题目大意

给定一个数组，要求给它进行“摆动排序”，“摆动排序”即：nums[0] < nums[1] > nums[2] < nums[3]...

解题思路

这一题最直接的方法是先排序，然后用 2 个指针，一个指向下标为 0 的位置，另一个指向下标为 n/2 的位置。最终的数组的奇数位从下标为 0 开始往后取，偶数位从下标为 n/2 中间位置开始往后取。这种方法时间复杂度为 O(n log n)。

题目要求用时间复杂度 O(n) 和 空间复杂度 O(1) 的方法解决。思路如下，先找到数组中间大小的数字，然后把数组分为 2 部分：

```
Index :        0   1   2   3   4   5
Small half:    M       S       S
Large half:        L       L       L(M)
```

奇数位排中间数和小于中间数的数字，偶数位排大于中间数的数字和中间数。如果中间数字有多个，那么偶数位最后几位也是中间数，奇数位开头的前几位也是中间数。

举例，给定一个数组如下，中间数是 5 。有 2 个 5 。

```
13   6   5   5   4   2

         M
```

```
Step 1:
Original idx: 0    1    2    3    4    5
Mapped idx:   1    3    5    0    2    4
Array:        13   6    5    5    4    2
             Left
              i
                                   Right
```

nums[Mapped_idx[i]] = nums[1] = 6 > 5, 所以可以把 6 放在第 1 个奇数位的位置。left 和 i 同时右移。

```
Step 2:
Original idx: 0    1    2    3    4    5
Mapped idx:   1    3    5    0    2    4
Array:        13   6    5    5    4    2
                  Left
                   i
                                   Right
```

nums[3] = 5 = 5, 5 可以放在下标为 3 的位置，由于 5 已经和中间数相等了，所以只后移 i 。

```
Step 3:
Original idx: 0    1    2    3    4    5
Mapped idx:   1    3    5    0    2    4
Array:        13   6    5    5    4    2
                  Left
                        i
                                  Right
```

nums[5] = 2 < 5, 因为比中位数小，所以应该放在偶数位的最后 1 位。这里的例子而言，应该放在下标为 4 的位置上。交换 nums[Mapped_idx[i]] 和 nums[Mapped_idx[Right]]，交换完成以后 right 向左移。

```
Step 4:
Original idx: 0    1    2    3    4    5
Mapped idx:   1    3    5    0    2    4
Array:        13   6    5    5    2    4
                  Left
                       i
                              Right
```

nums[5] = 4 < 5, 因为比中位数小，所以应该放在偶数位的当前倒数第一位。这里的例子而言，应该放在下标为 2 的位置上。交换 nums[Mapped_idx[i]] 和 nums[Mapped_idx[Right]]，交换完成以后 right 向左移。

```
Step 5:
Original idx: 0    1    2    3    4    5
Mapped idx:   1    3    5    0    2    4
Array:        13   6    4    5    2    5
                  Left
                       i
                           Right
```

nums[5] = 5 = 5, 由于 5 已经和中间数相等了，所以只后移 i 。

```
Step 6:
Original idx: 0    1    2    3    4    5
Mapped idx:   1    3    5    0    2    4
Array:        13   6    4    5    2    5
                  Left
                            i
                           Right
```

nums[0] = 13 > 5, 由于 13 比中位数大，所以可以把 13 放在第 2 个奇数位的位置，并移动 left 和 i 。

```
Step Final:
Original idx: 0    1    2    3    4    5
Mapped idx:   1    3    5    0    2    4
Array:        5    6    4    13   2    5
                      Left
                                 i
                           Right
```

i > Right, 退出循环，最终摆动排序的结果是 5 6 4 13 2 5 。

具体时间见代码，时间复杂度 O(n) 和 空间复杂度 O(1)。

代码

```
package leetcode

import (
  "sort"
)

// 解法一
func wiggleSort(nums []int) {
  if len(nums) < 2 {
    return
  }
  median := findKthLargest324(nums, (len(nums)+1)/2)
  n, i, left, right := len(nums), 0, 0, len(nums)-1

  for i <= right {
    if nums[indexMap(i, n)] > median {
      nums[indexMap(left, n)], nums[indexMap(i, n)] = nums[indexMap(i, n)],
nums[indexMap(left, n)]
      left++
      i++
    } else if nums[indexMap(i, n)] < median {
      nums[indexMap(right, n)], nums[indexMap(i, n)] = nums[indexMap(i, n)],
nums[indexMap(right, n)]
      right--
    } else {
      i++
    }
  }
}

func indexMap(index, n int) int {
  return (1 + 2*index) % (n | 1)
}
```

```go
func findKthLargest324(nums []int, k int) int {
  if len(nums) == 0 {
    return 0
  }
  return selection324(nums, 0, len(nums)-1, len(nums)-k)
}

func selection324(arr []int, l, r, k int) int {
  if l == r {
    return arr[l]
  }
  p := partition324(arr, l, r)

  if k == p {
    return arr[p]
  } else if k < p {
    return selection324(arr, l, p-1, k)
  } else {
    return selection324(arr, p+1, r, k)
  }
}

func partition324(a []int, lo, hi int) int {
  pivot := a[hi]
  i := lo - 1
  for j := lo; j < hi; j++ {
    if a[j] < pivot {
      i++
      a[j], a[i] = a[i], a[j]
    }
  }
  a[i+1], a[hi] = a[hi], a[i+1]
  return i + 1
}

// 解法二
func wiggleSort1(nums []int) {
  if len(nums) < 2 {
    return
  }
  array := make([]int, len(nums))
  copy(array, nums)
  sort.Ints(array)
  n := len(nums)
  left := (n+1)/2 - 1 // median index
  right := n - 1      // largest value index
  for i := 0; i < len(nums); i++ {
    // copy large values on odd indexes
```

```
    if i%2 == 1 {
      nums[i] = array[right]
      right--
    } else { // copy values decremeting from median on even indexes
      nums[i] = array[left]
      left--
    }
  }
}
```

326. Power of Three

题目

Given an integer, write a function to determine if it is a power of three.

Example 1:

```
Input: 27
Output: true
```

Example 2:

```
Input: 0
Output: false
```

Example 3:

```
Input: 9
Output: true
```

Example 4:

```
Input: 45
Output: false
```

Follow up:

Could you do it without using any loop / recursion?

题目大意

给定一个整数，写一个函数来判断它是否是 3 的幂次方。

解题思路

- 判断一个数是不是 3 的 n 次方。
- 这一题最简单的思路是循环，可以通过。但是题目要求不循环就要判断，这就需要用到数论的知识了。由于 3^20 超过了 int 的范围了，所以 3^19 次方就是 int 类型中最大的值。这一题和第 231 题是一样的思路。

代码

```
package leetcode

// 解法一 数论
func isPowerOfThree(n int) bool {
  // 1162261467 is 3^19,  3^20 is bigger than int
  return n > 0 && (1162261467%n == 0)
}

// 解法二 打表法
func isPowerOfThree1(n int) bool {
  // 1162261467 is 3^19,  3^20 is bigger than int
  allPowerOfThreeMap := map[int]int{1: 1, 3: 3, 9: 9, 27: 27, 81: 81, 243: 243, 729:
729, 2187: 2187, 6561: 6561, 19683: 19683, 59049: 59049, 177147: 177147, 531441:
531441, 1594323: 1594323, 4782969: 4782969, 14348907: 14348907, 43046721: 43046721,
129140163: 129140163, 387420489: 387420489, 1162261467: 1162261467}
  _, ok := allPowerOfThreeMap[n]
  return ok
}

// 解法三 循环
func isPowerOfThree2(num int) bool {
  for num >= 3 {
    if num%3 == 0 {
      num = num / 3
    } else {
      return false
    }
  }
  return num == 1
}
```

327. Count of Range Sum

题目

Given an integer array `nums`, return the number of range sums that lie in `[lower, upper]` inclusive.Range sum `S(i, j)` is defined as the sum of the elements in `nums` between indices `i` and `j` (`i` ≤ `j`), inclusive.

Note:A naive algorithm of O(n2) is trivial. You MUST do better than that.

Example:

```
Input: nums = [-2,5,-1], lower = -2, upper = 2,
Output: 3
Explanation: The three ranges are : [0,0], [2,2], [0,2] and their respective sums are:
-2, -1, 2.
```

题目大意

给定一个整数数组 nums，返回区间和在 [lower, upper] 之间的个数，包含 lower 和 upper。区间和 S(i, j) 表示在 nums 中，位置从 i 到 j 的元素之和，包含 i 和 j (i ≤ j)。

说明:
最直观的算法复杂度是 O(n^2) ，请在此基础上优化你的算法。

解题思路

- 给出一个数组，要求在这个数组中找出任意一段子区间的和，位于 [lower,upper] 之间。
- 这一题可以用暴力解法，2 层循环，遍历所有子区间，求和并判断是否位于 [lower,upper] 之间，时间复杂度 O(n^2)。
- 这一题当然还有更优的解法，用线段树或者树状数组，将时间复杂度降为 O(n log n)。题目中要求 `lower ≤ sum(i,j) ≤ upper`，`sum(i,j) = prefixSum(j) - prefixSum(i-1)`，那么 `lower + prefixSum(i-1) ≤ prefixSum(j) ≤ upper + prefixSum(i-1)`。所以利用前缀和将区间和转换成了前缀和在线段树中 `query` 的问题，只不过线段树中父节点中存的不是子节点的和，而应该是子节点出现的次数。第二个转换，由于前缀和会很大，所以需要离散化。例如 `prefixSum = [-3,-2,-1,0]`，用前缀和下标进行离散化，所以线段树中左右区间变成了 0-3 。

利用 `prefixSum` 下标离散化：

- 还需要注意一些小细节，`prefixSum` 计算完以后需要去重，去重以后并排序，方便构造线段树的有效区间。如果不去重，线段树中可能出现非法区间(left > right)或者重叠区间。最后一步往线段树中倒序插入 `prefixSum` 的时候，用的是非去重的，插入 `prefixSum[j]` 代表 sum(i,j) 中的 j，例如往线段树中插入 `prefixSum[5]`，代表当前树中加入了 j = 5 的情况。query 操作实质是在做区间匹配，例如当前 i 循环到 i = 3，累计往线段树中插入了 `prefixSum[5]`，`prefixSum[4]`，`prefixSum[3]`，那么 query 操作实质是在判断：`lower ≤ sum(i=3,j=3) ≤ upper`，`lower ≤ sum(i=3,j=4) ≤ upper`，`lower ≤ sum(i=3,j=5) ≤ upper`，这 3 个等式是否成立，有几个成立就返回几个，即是最终要求得的结果的一部分。
- 举个例子，`nums = [-3,1,2,-2,2,-1]`，`prefixSum = [-3,-2,0,-2,0,-1]`，去重以后并排序得到 `sum = [-3,-2,-1,0]`。离散化构造线段树，这里出于演示的方便，下图中就不画出离散后的线段树了，用非离散的线段树展示：

倒序插入 `len(prefixSum)-1 = prefixSum[5] = -1`：

这时候查找区间变为了 `[-3 + prefixSum[5-1], -1 + prefixSum[5-1]] = [-3,-1]`，即判断 `-3 ≤ sum(5,5) ≤ -1`，满足等式的有几种情况，这里明显只有一种情况，即 `j = 5`，也满足等式，所以这一步 `res = 1`。

- 倒序插入 `len(prefixSum)-2 = prefixSum[4] = 0`：

这时候查找区间变为了 `[-3 + prefixSum[4-1], -1 + prefixSum[4-1]] = [-5,-3]`，即判断 `-5 ≤ sum(4,  4,5) ≤ -3`，满足等式的有几种情况，这里有两种情况，即 `j = 4` 或者 `j = 5`，都不满足等式，所以这一步 `res = 0`。

- 倒序插入 `len(prefixSum)-3 = prefixSum[3] = -2`：

这时候查找区间变为了 `[-3 + prefixSum[3-1], -1 + prefixSum[3-1]] = [-3,-1]`，即判断 `-3 ≤ sum(3, 3,4,5) ≤ -1`，满足等式的有几种情况，这里有三种情况，即 `j = 3` 、`j = 4` 或者 `j = 5`，满足等式的有 `j = 3` 和 `j = 5`，即 `-3 ≤ sum(3, 3) ≤ -1` 和 `-3 ≤ sum(3, 5) ≤ -1`。所以这一步 `res = 2`。

- 倒序插入 `len(prefixSum)-4 = prefixSum[2] = 0`：

这时候查找区间变为了 `[-3 + prefixSum[2-1], -1 + prefixSum[2-1]] = [-5,-3]`，即判断 `-5 ≤ sum(2, 2,3,4,5) ≤ -3`，满足等式的有几种情况，这里有四种情况，即 `j = 2`、`j = 3` 、`j = 4` 或者 `j = 5`，都不满足等式。所以这一步 `res = 0`。

- 倒序插入 `len(prefixSum)-5 = prefixSum[1] = -2`：

@halfrost

这时候查找区间变为了 `[-3 + prefixSum[1-1], -1 + prefixSum[1-1]] = [-6,-4]`，即判断 `-6 ≤ sum(1,  1,2,3,4,5) ≤ -4`，满足等式的有几种情况，这里有五种情况，即 `j = 1`、`j = 2`、`j = 3`、`j = 4` 或者 `j = 5`，都不满足等式。所以这一步 `res = 0`。

- 倒序插入 `len(prefixSum)-6 = prefixSum[0] = -3`：

@halfrost

这时候查找区间变为了 `[-3 + prefixSum[0-1], -1 + prefixSum[0-1]] = [-3,-1]`，注意 `prefixSum[-1] = 0`，即判断 `-3 ≤ sum(0,  0,1,2,3,4,5) ≤ -1`，满足等式的有几种情况，这里有六种情况，即 `j = 0`、`j = 1`、`j = 2`、`j = 3`、`j = 4` 或者 `j = 5`，满足等式的有 `j = 0`、`j = 1`、`j = 3` 和 `j = 5`，即 `-3 ≤ sum(0, 0) ≤ -1`、`-3 ≤ sum(0, 1) ≤ -1`、`-3 ≤ sum(0, 3) ≤ -1` 和 `-3 ≤ sum(0, 5) ≤ -1`。所以这一步 `res = 4`。最后的答案就是把每一步的结果都累加，`res = 1 + 0 + 2 + 0 + 0 + 4 = 7`。

- 此题同样可以用树状数组来解答。同样把问题先转化成区间 Query 的模型，`lower ≤ prefixSum(j) - prefixSum(i-1) ≤ upper` 等价于 `prefixSum(j) - upper ≤ prefixSum(i-1) ≤ prefixSum(j) - lower`，`i` 的取值在 `[0,j-1]` 区间内。所以题目可以转化为 `i` 在 `[0,j-1]` 区间内取值，问数组 `prefixSum[0...j-1]` 中的所有取值，位于区间 `[prefixSum(j) - upper, prefixSum(j) - lower]` 内的次数。在树状数组中，区间内的前缀和可以转化为 2 个区间的前缀和相减，即 `Query([i,j]) = Query(j) - Query(i-1)`。所以这道题枚举数组 `prefixSum[0...j-1]` 中每个值是否出现在指定区间内出现次数即可。第一步先将所有的前缀和 `prefixSum(j)` 以及 `[prefixSum(j) - upper, prefixSum(j) - lower]` 计算出来。第二步排序和离散化，离散化以后的点区间为 `[1,n]`。最后根据数组 `prefixSum(j)` 的值在指定区间内查询出现次数即可。相同的套路题有，第 315 题，第 493 题。

代码

```go
package leetcode

import (
  "sort"

  "github.com/halfrost/LeetCode-Go/template"
)

// 解法一 线段树，时间复杂度 O(n log n)
func countRangeSum(nums []int, lower int, upper int) int {
  if len(nums) == 0 {
    return 0
  }
  st, prefixSum, sumMap, sumArray, res := template.SegmentCountTree{}, make([]int,
len(nums)), make(map[int]int, 0), []int{}, 0
  prefixSum[0], sumMap[nums[0]] = nums[0], nums[0]
  for i := 1; i < len(nums); i++ {
    prefixSum[i] = prefixSum[i-1] + nums[i]
    sumMap[prefixSum[i]] = prefixSum[i]
  }
  // sumArray 是 prefixSum 去重之后的版本，利用 sumMap 去重
  for _, v := range sumMap {
    sumArray = append(sumArray, v)
  }
  // 排序是为了使得线段树中的区间 left <= right，如果此处不排序，线段树中的区间有很多不合法。
  sort.Ints(sumArray)
  // 初始化线段树，节点内的值都赋值为 0，即计数为 0
  st.Init(sumArray, func(i, j int) int {
    return 0
  })
  // 倒序是为了方便寻找 j，sum(i, j) 规定了 j >= i，所以倒序遍历，i 从大到小
  for i := len(nums) - 1; i >= 0; i-- {
    // 插入的 prefixSum[i] 即是 j
    st.UpdateCount(prefixSum[i])
    if i > 0 {
```

```go
        res += st.Query(lower+prefixSum[i-1], upper+prefixSum[i-1])
      } else {
        res += st.Query(lower, upper)
      }
    }
    return res
}

// 解法二 树状数组，时间复杂度 O(n log n)
func countRangeSum1(nums []int, lower int, upper int) int {
    n := len(nums)
    // 计算前缀和 preSum，以及后面统计时会用到的所有数字 allNums
    allNums, preSum, res := make([]int, 1, 3*n+1), make([]int, n+1), 0
    for i, v := range nums {
      preSum[i+1] = preSum[i] + v
      allNums = append(allNums, preSum[i+1], preSum[i+1]-lower, preSum[i+1]-upper)
    }
    // 将 allNums 离散化
    sort.Ints(allNums)
    k := 1
    kth := map[int]int{allNums[0]: k}
    for i := 1; i <= 3*n; i++ {
      if allNums[i] != allNums[i-1] {
        k++
        kth[allNums[i]] = k
      }
    }
    // 遍历 preSum，利用树状数组计算每个前缀和对应的合法区间数
    bit := template.BinaryIndexedTree{}
    bit.Init(k)
    bit.Add(kth[0], 1)
    for _, sum := range preSum[1:] {
      left, right := kth[sum-upper], kth[sum-lower]
      res += bit.Query(right) - bit.Query(left-1)
      bit.Add(kth[sum], 1)
    }
    return res
}

// 解法三 暴力，时间复杂度 O(n^2)
func countRangeSum2(nums []int, lower int, upper int) int {
    res, n := 0, len(nums)
    for i := 0; i < n; i++ {
      tmp := 0
      for j := i; j < n; j++ {
        if i == j {
          tmp = nums[i]
        } else {
          tmp += nums[j]
```

```
        }
        if tmp <= upper && tmp >= lower {
          res++
        }
      }
    }
    return res
  }
```

328. Odd Even Linked List

题目

Given a singly linked list, group all odd nodes together followed by the even nodes. Please note here we are talking about the node number and not the value in the nodes.

You should try to do it in place. The program should run in O(1) space complexity and O(nodes) time complexity.

Example 1:

```
Input: 1->2->3->4->5->NULL
Output: 1->3->5->2->4->NULL
```

Example 2:

```
Input: 2->1->3->5->6->4->7->NULL
Output: 2->3->6->7->1->5->4->NULL
```

Note:

- The relative order inside both the even and odd groups should remain as it was in the input.
- The first node is considered odd, the second node even and so on ...

题目大意

这道题和第 86 题非常类型。第 86 题是把排在某个点前面的小值放在一个链表中，排在某个点后端的大值放在另外一个链表中，最后 2 个链表首尾拼接一下就是答案。

解题思路

这道题思路也是一样的，分别把奇数和偶数都放在 2 个链表中，最后首尾拼接就是答案。

代码

```go
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */

func oddEvenList(head *ListNode) *ListNode {
  oddHead := &ListNode{Val: 0, Next: nil}
  odd := oddHead
  evenHead := &ListNode{Val: 0, Next: nil}
  even := evenHead

  count := 1
  for head != nil {
    if count%2 == 1 {
      odd.Next = head
      odd = odd.Next
    } else {
      even.Next = head
      even = even.Next
    }
    head = head.Next
    count++
  }
  even.Next = nil
  odd.Next = evenHead.Next
  return oddHead.Next
}
```

329. Longest Increasing Path in a Matrix

题目

Given an integer matrix, find the length of the longest increasing path.

From each cell, you can either move to four directions: left, right, up or down. You may NOT move diagonally or move outside of the boundary (i.e. wrap-around is not allowed).

Example 1:

```
Input: nums =
[
  [9,9,4],
  [6,6,8],
  [2,1,1]
]
Output: 4
Explanation: The longest increasing path is [1, 2, 6, 9].
```

Example 2:

```
Input: nums =
[
  [3,4,5],
  [3,2,6],
  [2,2,1]
]
Output: 4
Explanation: The longest increasing path is [3, 4, 5, 6]. Moving diagonally is not
allowed.
```

题目大意

给定一个整数矩阵，找出最长递增路径的长度。对于每个单元格，你可以往上，下，左，右四个方向移动。 你不能在对角线方向上移动或移动到边界外（即不允许环绕）。

解题思路

- 给出一个矩阵，要求在这个矩阵中找到一个最长递增的路径。路径有上下左右 4 个方向。
- 这一题解题思路很明显，用 DFS 即可。在提交完第一版以后会发现 TLE，因为题目给出了一个非常大的矩阵，搜索次数太多。所以需要用到记忆化，把曾经搜索过的最大长度缓存起来，增加了记忆化以后再次提交 AC。

代码

```go
package leetcode

import (
  "math"
)

var dir = [][]int{
  {-1, 0},
  {0, 1},
  {1, 0},
  {0, -1},
```

```go
}

func longestIncreasingPath(matrix [][]int) int {
  cache, res := make([][]int, len(matrix)), 0
  for i := 0; i < len(cache); i++ {
    cache[i] = make([]int, len(matrix[0]))
  }
  for i, v := range matrix {
    for j := range v {
      searchPath(matrix, cache, math.MinInt64, i, j)
      res = max(res, cache[i][j])
    }
  }
  return res
}

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

func isInIntBoard(board [][]int, x, y int) bool {
  return x >= 0 && x < len(board) && y >= 0 && y < len(board[0])
}

func searchPath(board, cache [][]int, lastNum, x, y int) int {
  if board[x][y] <= lastNum {
    return 0
  }
  if cache[x][y] > 0 {
    return cache[x][y]
  }
  count := 1
  for i := 0; i < 4; i++ {
    nx := x + dir[i][0]
    ny := y + dir[i][1]
    if isInIntBoard(board, nx, ny) {
      count = max(count, searchPath(board, cache, board[x][y], nx, ny)+1)
    }
  }
  cache[x][y] = count
  return count
}
```

331. Verify Preorder Serialization of a Binary Tree

题目

One way to serialize a binary tree is to use pre-order traversal. When we encounter a non-null node, we record the node's value. If it is a null node, we record using a sentinel value such as #.

```
     _9_
    /   \
   3     2
  / \   / \
 4   1  #  6
/ \ / \   / \
# # # #   # #
```

For example, the above binary tree can be serialized to the string "9,3,4,#,#,1,#,#,2,#,6,#,#", where # represents a null node.

Given a string of comma separated values, verify whether it is a correct preorder traversal serialization of a binary tree. Find an algorithm without reconstructing the tree.

Each comma separated value in the string must be either an integer or a character '#' representing null pointer.

You may assume that the input format is always valid, for example it could never contain two consecutive commas such as "1,,3".

Example 1:

```
Input: "9,3,4,#,#,1,#,#,2,#,6,#,#"
Output: true
```

Example 2:

```
Input: "1,#"
Output: false
```

Example 3:

```
Input: "9,#,#,1"
Output: false
```

题目大意

给定一串以逗号分隔的序列，验证它是否是正确的二叉树的前序序列化。编写一个在不重构树的条件下的可行算法。

解题思路

这道题有些人用栈，有些用栈的深度求解。换个视角。如果叶子结点是 null，那么所有非 null 的结点(除了 root 结点)必然有 2 个出度，1 个入度(2 个孩子和 1 个父亲，孩子可能为空，但是这一题用 "#" 代替了，所以肯定有 2 个孩子)；所有的 null 结点只有 0 个出度，1 个入度(0 个孩子和 1 个父亲)。

我们开始构建这颗树，在构建过程中，我们记录出度和度之间的差异 `diff = outdegree - indegree`。当下一个节点到来时，我们将 diff 减 1，因为这个节点提供了一个度。如果这个节点不为 null，我们将 diff 增加 2，因为它提供两个出度。如果序列化是正确的，则 diff 应该永远不会为负，并且 diff 在完成时将为零。最后判断一下 diff 是不是为 0 即可判断它是否是正确的二叉树的前序序列化。

代码

```go
package leetcode

import "strings"

func isValidSerialization(preorder string) bool {
  nodes, diff := strings.Split(preorder, ","), 1
  for _, node := range nodes {
    diff--
    if diff < 0 {
      return false
    }
    if node != "#" {
      diff += 2
    }
  }
  return diff == 0
}
```

337. House Robber III

题目

The thief has found himself a new place for his thievery again. There is only one entrance to this area, called the "root." Besides the root, each house has one and only one parent house. After a tour, the smart thief realized that "all houses in this place forms a binary tree". It will automatically contact the police if two directly-linked houses were broken into on the same night.

Determine the maximum amount of money the thief can rob tonight without alerting the police.

Example 1:

```
Input: [3,2,3,null,3,null,1]

     3
    / \
   2   3
    \   \
     3   1

Output: 7
Explanation: Maximum amount of money the thief can rob = 3 + 3 + 1 = 7.
```

Example 2:

```
Input: [3,4,5,1,3,null,1]

     3
    / \
   4   5
  / \   \
 1   3   1

Output: 9
Explanation: Maximum amount of money the thief can rob = 4 + 5 = 9.
```

题目大意

一个新的可行窃的地区只有一个入口，称之为“根”。 除了“根”之外，每栋房子有且只有一个“父“房子与之相连。一番侦察之后，聪明的小偷意识到“这个地方的所有房屋的排列类似于一棵二叉树”。 如果两个直接相连的房子在同一天晚上被打劫，房屋将自动报警。计算在不触动警报的情况下，小偷一晚能够盗取的最高金额。

解题思路

- 这一题是打家劫舍的第 3 题。这一题需要偷的房子是树状的。报警的条件还是相邻的房子如果都被偷了，就会触发报警。只不过这里相邻的房子是树上的。问小偷在不触发报警的条件下最终能偷的最高金额。
- 解题思路是 DFS。当前节点是否被打劫，会产生 2 种结果。如果当前节点被打劫，那么它的孩子节点可以被打劫；如果当前节点没有被打劫，那么它的孩子节点不能被打劫。按照这个逻辑递归，最终递归到根节点，取最大值输出即可。

代码

```
func rob337(root *TreeNode) int {
  a, b := dfsTreeRob(root)
  return max(a, b)
}

func dfsTreeRob(root *TreeNode) (a, b int) {
  if root == nil {
    return 0, 0
  }
  l0, l1 := dfsTreeRob(root.Left)
  r0, r1 := dfsTreeRob(root.Right)
  // 当前节点没有被打劫
  tmp0 := max(l0, l1) + max(r0, r1)
  // 当前节点被打劫
  tmp1 := root.Val + l0 + r0
  return tmp0, tmp1
}
```

338. Counting Bits

题目

Given a non negative integer number **num**. For every numbers **i** in the range **0 ≤ i ≤ num** calculate the number of 1's in their binary representation and return them as an array.

Example 1:

```
Input: 2
Output: [0,1,1]
```

Example 2:

```
Input: 5
Output: [0,1,1,2,1,2]
```

Follow up:

- It is very easy to come up with a solution with run time **O(n*sizeof(integer))**. But can you do it in linear time **O(n)** /possibly in a single pass?
- Space complexity should be **O(n)**.
- Can you do it like a boss? Do it without using any builtin function like **__builtin_popcount** in c++ or in any other language.

题目大意

给定一个非负整数 num。对于 0 ≤ i ≤ num 范围中的每个数字 i ，计算其二进制数中的 1 的数目并将它们作为数组返回。

解题思路

- 给出一个数，要求计算出 0 ≤ i ≤ num 中每个数的二进制位 1 的个数。
- 这一题就是利用二进制位运算的经典题。

```
X&1==1or==0，可以用 X&1 判断奇偶性，X&1>0 即奇数。
X = X & (X-1) 清零最低位的1
X & -X => 得到最低位的1
X&~X=>0
```

代码

```go
package leetcode

func countBits(num int) []int {
  bits := make([]int, num+1)
  for i := 1; i <= num; i++ {
    bits[i] += bits[i&(i-1)] + 1
  }
  return bits
}
```

341. Flatten Nested List Iterator

题目

Given a nested list of integers, implement an iterator to flatten it.

Each element is either an integer, or a list -- whose elements may also be integers or other lists.

Example 1:

```
Input:[[1,1],2,[1,1]]
Output:[1,1,2,1,1]
Explanation:By callingnext repeatedly untilhasNext returns false,
             the order of elements returned bynext should be:[1,1,2,1,1].
```

Example 2:

```
Input:[1,[4,[6]]]
Output:[1,4,6]
Explanation:By callingnext repeatedly untilhasNext returns false,
             the order of elements returned bynext should be:[1,4,6].
```

题目大意

给你一个嵌套的整型列表。请你设计一个迭代器，使其能够遍历这个整型列表中的所有整数。列表中的每一项或者为一个整数，或者是另一个列表。其中列表的元素也可能是整数或是其他列表。

解题思路

- 题目要求实现一个嵌套版的数组。可以用 `[]int` 实现，也可以用链表实现。笔者此处用链表实现。外层构造一个一维数组，一维数组内部每个元素是一个链表。额外还需要记录这个嵌套链表在原数组中的 `index` 索引。`Next()` 实现比较简单，取出对应的嵌套节点。`HasNext()` 方法则感觉嵌套节点里面的 `index` 信息判断是否还有 `next` 元素。

代码

```
/**
 * // This is the interface that allows for creating nested lists.
 * // You should not implement it, or speculate about its implementation
 * type NestedInteger struct {
 * }
 *
 * // Return true if this NestedInteger holds a single integer, rather than a nested
list.
 * func (this NestedInteger) IsInteger() bool {}
 *
 * // Return the single integer that this NestedInteger holds, if it holds a single
integer
 * // The result is undefined if this NestedInteger holds a nested list
 * // So before calling this method, you should have a check
 * func (this NestedInteger) GetInteger() int {}
 *
 * // Set this NestedInteger to hold a single integer.
 * func (n *NestedInteger) SetInteger(value int) {}
 *
 * // Set this NestedInteger to hold a nested list and adds a nested integer to it.
 * func (this *NestedInteger) Add(elem NestedInteger) {}
 *
 * // Return the nested list that this NestedInteger holds, if it holds a nested list
 * // The list length is zero if this NestedInteger holds a single integer
 * // You can access NestedInteger's List element directly if you want to modify it
 * func (this NestedInteger) GetList() []*NestedInteger {}
 */
```

```go
type NestedIterator struct {
    stack *list.List
}

type listIndex struct {
    nestedList []*NestedInteger
    index int
}

func Constructor(nestedList []*NestedInteger) *NestedIterator {
    stack := list.New()
    stack.PushBack(&listIndex{nestedList, 0})
    return &NestedIterator{stack}
}

func (this *NestedIterator) Next() int {
    if !this.HasNext() {
        return -1
    }
    last := this.stack.Back().Value.(*listIndex)
    nestedList, i := last.nestedList, last.index
    val := nestedList[i].GetInteger()
    last.index++
    return val
}

func (this *NestedIterator) HasNext() bool {
    stack := this.stack
    for stack.Len() > 0 {
        last := stack.Back().Value.(*listIndex)
        nestedList, i := last.nestedList, last.index
        if i >= len(nestedList) {
            stack.Remove(stack.Back())
        } else {
            val := nestedList[i]
            if val.IsInteger() {
                return true
            }
            last.index++
            stack.PushBack(&listIndex{val.GetList(), 0})
        }
    }
    return false
}
```

342. Power of Four

题目

Given an integer (signed 32 bits), write a function to check whether it is a power of 4.

Example 1:

```
Input: 16
Output: true
```

Example 2:

```
Input: 5
Output: false
```

Follow up: Could you solve it without loops/recursion?

题目大意

给定一个整数 (32 位有符号整数)，请编写一个函数来判断它是否是 4 的幂次方。

解题思路

- 判断一个数是不是 4 的 n 次方。
- 这一题最简单的思路是循环，可以通过。但是题目要求不循环就要判断，这就需要用到数论的知识了。
- 证明 `(4^n - 1) % 3 == 0`，(1) `4^n - 1 = (2^n + 1) * (2^n - 1)` (2) 在任何连续的 3 个数中 `(2^n-1)`，`(2^n)`，`(2^n+1)`，一定有一个数是 3 的倍数。`(2^n)` 肯定不是 3 的倍数，那么 `(2^n-1)` 或者 `(2^n+1)` 中一定有一个是 3 的倍数。所以 `4^n-1` 一定是 3 的倍数。

代码

```go
package leetcode

// 解法一 数论
func isPowerOfFour(num int) bool {
  return num > 0 && (num&(num-1)) == 0 && (num-1)%3 == 0
}

// 解法二 循环
func isPowerOfFour1(num int) bool {
  for num >= 4 {
    if num%4 == 0 {
      num = num / 4
    } else {
      return false
    }
  }
  return num == 1
```

```
}
```

343. Integer Break

题目

Given a positive integer n, break it into the sum of **at least** two positive integers and maximize the product of those integers. Return the maximum product you can get.

Example 1:

```
Input: 2
Output: 1
Explanation: 2 = 1 + 1, 1 × 1 = 1.
```

Example 2:

```
Input: 10
Output: 36
Explanation: 10 = 3 + 3 + 4, 3 × 3 × 4 = 36.
```

Note: You may assume that n is not less than 2 and not larger than 58.

题目大意

给定一个正整数 n，将其拆分为至少两个正整数的和，并使这些整数的乘积最大化。 返回你可以获得的最大乘积。

解题思路

- 这一题是 DP 的题目，将一个数字分成多个数字之和，至少分为 2 个数字之和，求解分解出来的数字乘积最大是多少。
- 这一题的动态转移方程是 `dp[i] = max(dp[i], j * (i - j), j * dp[i-j])` ，一个数分解成 `j` 和 `i - j` 两个数字，或者分解成 `j` 和 `更多的分解数` ，`更多的分解数` 即是 `dp[i-j]`，由于 `dp[i-j]` 下标小于 `i` ，所以 `dp[i-j]` 在计算 `dp[i]` 的时候一定计算出来了。

代码

```go
package leetcode

func integerBreak(n int) int {
  dp := make([]int, n+1)
  dp[0], dp[1] = 1, 1
  for i := 1; i <= n; i++ {
```

```
    for j := 1; j < i; j++ {
      // dp[i] = max(dp[i], j * (i - j), j*dp[i-j])
      dp[i] = max(dp[i], j*max(dp[i-j], i-j))
    }
  }
  return dp[n]
}
```

344. Reverse String

题目

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"]
```

题目大意

题目要求我们反转一个字符串。

解题思路

这一题的解题思路是用 2 个指针，指针对撞的思路，来不断交换首尾元素，即可。

代码

```go
package leetcode

func reverseString(s []byte) {
  for i, j := 0, len(s)-1; i < j; {
    s[i], s[j] = s[j], s[i]
    i++
    j--
  }
}
```

345. Reverse Vowels of a String

题目

Write a function that takes a string as input and reverse only the vowels of a string.

Example 1:

```
Input: "hello"
Output: "holle"
```

Example 2:

```
Input: "leetcode"
Output: "leotcede"
```

题目大意

题目要求我们反转字符串中的元音字母。需要注意字母大小写。

解题思路

这一题的解题思路是用 2 个指针，指针对撞的思路，来不断交换首尾元素，即可。这一题和第 344 题思路一样。

代码

```go
package leetcode
```

```go
func reverseVowels(s string) string {
  b := []byte(s)
  for i, j := 0, len(b)-1; i < j; {
    if !isVowel(b[i]) {
      i++
      continue
    }
    if !isVowel(b[j]) {
      j--
      continue
    }
    b[i], b[j] = b[j], b[i]
    i++
    j--
  }
  return string(b)
}

func isVowel(s byte) bool {
  return s == 'a' || s == 'e' || s == 'i' || s == 'o' || s == 'u' || s == 'A' ||
    s == 'E' || s == 'I' || s == 'O' || s == 'U'
}
```

347. Top K Frequent Elements

题目

Given a non-empty array of integers, return the k most frequent elements.

Example 1:

```
Input: nums = [1,1,1,2,2,3], k = 2
Output: [1,2]
```

Example 2:

```
Input: nums = [1], k = 1
Output: [1]
```

Note:

- You may assume k is always valid, 1 ≤ k ≤ number of unique elements.
- Your algorithm's time complexity must be better than O(n log n), where n is the array's size.

题目大意

给一个非空的数组，输出前 K 个频率最高的元素。

解题思路

这一题是考察优先队列的题目。把数组构造成一个优先队列，输出前 K 个即可。

代码

```
package leetcode

import "container/heap"

func topKFrequent(nums []int, k int) []int {
  m := make(map[int]int)
  for _, n := range nums {
    m[n]++
  }
  q := PriorityQueue{}
  for key, count := range m {
    heap.Push(&q, &Item{key: key, count: count})
  }
  var result []int
  for len(result) < k {
    item := heap.Pop(&q).(*Item)
    result = append(result, item.key)
  }
  return result
}

// Item define
type Item struct {
  key   int
  count int
}

// A PriorityQueue implements heap.Interface and holds Items.
type PriorityQueue []*Item

func (pq PriorityQueue) Len() int {
  return len(pq)
}

func (pq PriorityQueue) Less(i, j int) bool {
```

```go
  // 注意：因为 golang 中的 heap 默认是按最小堆组织的，所以 count 越大，Less() 越小，越靠近堆顶。这里采用 >，变为最大堆
  return pq[i].count > pq[j].count
}

func (pq PriorityQueue) Swap(i, j int) {
  pq[i], pq[j] = pq[j], pq[i]
}

// Push define
func (pq *PriorityQueue) Push(x interface{}) {
  item := x.(*Item)
  *pq = append(*pq, item)
}

// Pop define
func (pq *PriorityQueue) Pop() interface{} {
  n := len(*pq)
  item := (*pq)[n-1]
  *pq = (*pq)[:n-1]
  return item
}
```

349. Intersection of Two Arrays

题目

Given two arrays, write a function to compute their intersection.

Example 1:

```
Input: nums1 = [1,2,2,1], nums2 = [2,2]
Output: [2]
```

Example 2:

```
Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]
Output: [9,4]
```

Note:

- Each element in the result must be unique.
- The result can be in any order.

题目大意

找到两个数组的交集元素，如果交集元素同一个数字出现了多次，只输出一次。

解题思路

把数组一的每个数字都存进字典中，然后在数组二中依次判断字典中是否存在，如果存在，在字典中删除它(因为输出要求只输出一次)。

代码

```
package leetcode

func intersection(nums1 []int, nums2 []int) []int {
  m := map[int]bool{}
  var res []int
  for _, n := range nums1 {
    m[n] = true
  }
  for _, n := range nums2 {
    if m[n] {
      delete(m, n)
      res = append(res, n)
    }
  }
  return res
}
```

350. Intersection of Two Arrays II

题目

Given two arrays, write a function to compute their intersection.

Example 1:

```
Input: nums1 = [1,2,2,1], nums2 = [2,2]
Output: [2,2]
```

Example 2:

```
Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]
Output: [4,9]
```

Note:

- Each element in the result should appear as many times as it shows in both arrays.
- The result can be in any order.

Follow up:

- What if the given array is already sorted? How would you optimize your algorithm?
- What if nums1's size is small compared to nums2's size? Which algorithm is better?
- What if elements of nums2 are stored on disk, and the memory is limited such that you cannot load all elements into the memory at once?

题目大意

这题是第 349 题的加强版。要求输出 2 个数组的交集元素，如果元素出现多次，要输出多次。

解题思路

这一题还是延续第 349 题的思路。把数组一中的数字都放进字典中，另外字典的 key 是数组中的数字，value 是这个数字出现的次数。在扫描数组二的时候，每取出一个存在的数组，把字典中的 value 减一。如果 value 是 0 代表不存在这个数字。

代码

```go
package leetcode

func intersect(nums1 []int, nums2 []int) []int {
  m := map[int]int{}
  var res []int
  for _, n := range nums1 {
    m[n]++
  }
  for _, n := range nums2 {
    if m[n] > 0 {
      res = append(res, n)
      m[n]--
    }
  }
  return res
}
```

354. Russian Doll Envelopes

题目

You have a number of envelopes with widths and heights given as a pair of integers `(w, h)`. One envelope can fit into another if and only if both the width and height of one envelope is greater than the width and height of the other envelope.

What is the maximum number of envelopes can you Russian doll? (put one inside other)

Note: Rotation is not allowed.

Example:

```
Input: [[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]).
```

题目大意

给定一些标记了宽度和高度的信封，宽度和高度以整数对形式 (w, h) 出现。当另一个信封的宽度和高度都比这个信封大的时候，这个信封就可以放进另一个信封里，如同俄罗斯套娃一样。

请计算最多能有多少个信封能组成一组“俄罗斯套娃”信封（即可以把一个信封放到另一个信封里面）。

说明:

- 不允许旋转信封。

解题思路

- 给出一组信封的宽度和高度，如果组成俄罗斯套娃，问最多能套几层。只有当一个信封的宽度和高度都比另外一个信封大的时候，才能套在小信封上面。
- 这一题的实质是第 300 题 Longest Increasing Subsequence 的加强版。能组成俄罗斯套娃的条件就是能找到一个最长上升子序列。但是这题的条件是二维的，要求能找到在二维上都能满足条件的最长上升子序列。先降维，把宽度排序。然后在高度上寻找最长上升子序列。这里用到的方法和第 300 题的方法一致。解题思路详解见第 300 题。
- 此题是二维的 LIS 问题。一维的 LIS 问题是第 300 题。三维的 LIS 问题是第 1691 题。

代码

```go
package leetcode

import (
  "sort"
```

```go
)

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] {
    return s[i][1] > s[j][1]
  }
  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 {
  sort.Sort(sortEnvelopes(envelopes))
  dp := []int{}
  for _, e := range envelopes {
    low, high := 0, len(dp)
    for low < high {
      mid := low + (high-low)>>1
      if dp[mid] >= e[1] {
        high = mid
      } else {
        low = mid + 1
      }
    }
    if low == len(dp) {
      dp = append(dp, e[1])
    } else {
      dp[low] = e[1]
    }
  }
  return len(dp)
}
```

357. Count Numbers with Unique Digits

题目

Given a **non-negative** integer n, count all numbers with unique digits, x, where 0 ≤ x < 10n.

Example:

```
Input: 2
Output: 91
Explanation: The answer should be the total numbers in the range of 0 ≤ x < 100,
             excluding 11,22,33,44,55,66,77,88,99
```

题目大意

给定一个非负整数 n，计算各位数字都不同的数字 x 的个数，其中 0 ≤ x < 10^n 。

解题思路

- 输出 n 位数中不出现重复数字的数字的个数
- 这道题摸清楚规律以后，可以直接写出最终所有答案，答案只有 11 个。
- 考虑不重复数字是如生成的。如果只是一位数，不存在重复的数字，结果是 10 。如果是二位数，第一位一定不能取 0，那么第一位有 1-9，9种取法，第二位为了和第一位不重复，只能有 0-9，10种取法中减去第一位取的数字，那么也是 9 种取法。以此类推，如果是三位数，第三位是 8 种取法；四位数，第四位是 7 种取法；五位数，第五位是 6 种取法；六位数，第六位是 5 种取法；七位数，第七位是 4 种取法；八位数，第八位是 3 种取法；九位数，第九位是 2 种取法；十位数，第十位是 1 种取法；十一位数，第十一位是 0 种取法；十二位数，第十二位是 0 种取法；那么第 11 位数以后，每个数都是重复数字的数字。知道这个规律以后，可以累积上面的结果，把结果直接存在数组里面，暴力打表即可。O(1) 的时间复杂度。

代码

```go
package leetcode

// 暴力打表法
func countNumbersWithUniqueDigits1(n int) int {
  res := []int{1, 10, 91, 739, 5275, 32491, 168571, 712891, 2345851, 5611771, 8877691}
  if n >= 10 {
    return res[10]
  }
  return res[n]
}

// 打表方法
func countNumbersWithUniqueDigits(n int) int {
  if n == 0 {
    return 1
  }
  res, uniqueDigits, availableNumber := 10, 9, 9
  for n > 1 && availableNumber > 0 {
    uniqueDigits = uniqueDigits * availableNumber
    res += uniqueDigits
```

```
    availableNumber--
    n--
  }
  return res
}
```

367. Valid Perfect Square

题目

Given a positive integer num, write a function which returns True if num is a perfect square else False.

Note: **Do not** use any built-in library function such as `sqrt`.

Example 1:

```
Input: 16
Output: true
```

Example 2:

```
Input: 14
Output: false
```

题目大意

给定一个正整数 num，编写一个函数，如果 num 是一个完全平方数，则返回 True，否则返回 False。

说明：不要使用任何内置的库函数，如 sqrt。

解题思路

- 给出一个数，要求判断这个数是不是完全平方数。
- 可以用二分搜索来解答这道题。判断完全平方数，根据它的定义来，是否能被开根号，即找到一个数的平方是否可以等于待判断的数字。从 [1, n] 区间内进行二分，若能找到则返回 true，找不到就返回 false 。

代码

```go
package leetcode

func isPerfectSquare(num int) bool {
  low, high := 1, num
  for low <= high {
    mid := low + (high-low)>>1
```

```
    if mid*mid == num {
      return true
    } else if mid*mid < num {
      low = mid + 1
    } else {
      high = mid - 1
    }
  }
  return false
}
```

368. Largest Divisible Subset

题目

Given a set of **distinct** positive integers `nums`, return the largest subset `answer` such that every pair `(answer[i], answer[j])` of elements in this subset satisfies:

- `answer[i] % answer[j] == 0`, or
- `answer[j] % answer[i] == 0`

If there are multiple solutions, return any of them.

Example 1:

```
Input: nums = [1,2,3]
Output: [1,2]
Explanation: [1,3] is also accepted.
```

Example 2:

```
Input: nums = [1,2,4,8]
Output: [1,2,4,8]
```

Constraints:

- `1 <= nums.length <= 1000`
- `1 <= nums[i] <= 2 * 109`
- All the integers in `nums` are **unique**.

题目大意

给你一个由 无重复 正整数组成的集合 nums ， 请你找出并返回其中最大的整除子集 answer ， 子集中每一元素对 (answer[i], answer[j]) 都应当满足：

- answer[i] % answer[j] == 0 ，或
- answer[j] % answer[i] == 0

如果存在多个有效解子集，返回其中任何一个均可。

解题思路

- 根据题目数据规模 1000，可以估计此题大概率是动态规划，并且时间复杂度是 O(n^2)。先将集合排序，以某一个小的数作为基准，不断的选择能整除的数加入集合。按照这个思路考虑，此题和第 300 题经典的 LIS 解题思路一致。只不过 LIS 每次选择更大的数，此题除了选择更大的数，只不过多了一个判断，这个更大的数能否整除当前集合里面的所有元素。按照此法一定可以找出最大的集合。
- 剩下的问题是如何输出最大集合。这道题的集合具有重叠子集的性质，例如 [2,4,8,16] 这个集合，长度是 4，它一定包含长度为 3 的子集，因为从它里面随便取 3 个数形成的子集也满足元素相互能整除的条件。同理，它也一定包含长度为 2，长度为 1 的子集。由于有这个性质，可以利用 dp 数组里面的数据，输出最大集合。例如，[2,4,6,8,9,13,16,40]，由动态规划可以找到最大集合是 [2,4,8,16]。长度为 4 的找到了，再找长度为 3 的，[2,4,8]，[2,4,40]。在最大集合中，最大元素是 16，所以 [2,4,40] 这个集合排除，它的最大元素大于 16。选定 [2,4,8] 这个集合，此时最大元素是 8 。以此类推，筛选到最后，便可以输出 [16,8,4,2] 这个组最大集合的答案了。

代码

```go
package leetcode

import "sort"

func largestDivisibleSubset(nums []int) []int {
  sort.Ints(nums)
  dp, res := make([]int, len(nums)), []int{}
  for i := range dp {
    dp[i] = 1
  }
  maxSize, maxVal := 1, 1
  for i := 1; i < len(nums); i++ {
    for j, v := range nums[:i] {
      if nums[i]%v == 0 && dp[j]+1 > dp[i] {
        dp[i] = dp[j] + 1
      }
    }
    if dp[i] > maxSize {
      maxSize, maxVal = dp[i], nums[i]
    }
  }
  if maxSize == 1 {
    return []int{nums[0]}
  }
  for i := len(nums) - 1; i >= 0 && maxSize > 0; i-- {
    if dp[i] == maxSize && maxVal%nums[i] == 0 {
      res = append(res, nums[i])
      maxVal = nums[i]
```

```
      maxSize--
    }
  }
  return res
}
```

371. Sum of Two Integers

题目

Calculate the sum of two integers a and b, but you are **not allowed** to use the operator `+` and `-`.

Example 1:

```
Input: a = 1, b = 2
Output: 3
```

Example 2:

```
Input: a = -2, b = 3
Output: 1
```

题目大意

不使用运算符 + 和 - ，计算两整数 a 、b 之和。

解题思路

- 要求不用加法和减法运算符计算 `a+b`。这一题需要用到 `^` 和 `&` 运算符的性质，两个数 ^ 可以实现两个数不带进位的二进制加法。这里需要实现加法，肯定需要进位。所以如何找到进位是本题的关键。
- 在二进制中，只有 1 和 1 加在一起才会进位，0 和 0，0 和 1，1 和 0，这三种情况都不会进位，规律就是 `a & b` 为 0 的时候就不用进位，为 1 的时候代表需要进位。进位是往前进一位，所以还需要左移操作，所以加上的进位为 `(a&b)<<1`。

代码

```
package leetcode

func getSum(a int, b int) int {
  if a == 0 {
    return b
  }
  if b == 0 {
    return a
  }
```

```
  // (a & b)<<1 计算的是进位
  // a ^ b 计算的是不带进位的加法
  return getSum((a&b)<<1, a^b)
}
```

372. Super Pow

题目

Your task is to calculate ab mod 1337 where a is a positive integer and b is an extremely large positive integer given in the form of an array.

Example 1:

```
Input: a = 2, b = [3]
Output: 8
```

Example 2:

```
Input: a = 2, b = [1,0]
Output: 1024
```

题目大意

你的任务是计算 a^b 对 1337 取模，a 是一个正整数，b 是一个非常大的正整数且会以数组形式给出。

解题思路

- 求 a^b mod p 的结果，b 是大数。
- 这一题可以用暴力解法尝试。需要用到 mod 计算的几个运算性质：

  ```
  模运算性质一：(a + b) % p = (a % p + b % p) % p
  模运算性质二：(a - b) % p = (a % p - b % p + p) % p
  模运算性质三：(a * b) % p = (a % p * b % p) % p
  模运算性质四：a ^ b % p = ((a % p)^b) % p
  ```

  这一题需要用到性质三、四。举个例子：

  ```
  12345^678 % 1337 = (12345^670 * 12345^8) % 1337
           = ((12345^670 % 1337) * (12345^8 % 1337)) % 1337  ---> 利用性质 三
           = (((12345^67)^10 % 1337) * (12345^8 % 1337)) % 1337  ---> 乘方性质
                   = ((12345^67 % 1337)^10) % 1337 * (12345^8 % 1337)) % 1337  --->
  利用性质 四
           = (((12345^67 % 1337)^10) * (12345^8 % 1337)) % 1337  ---> 反向利用性质 三
  ```

经过上面这样的变换，把指数 678 的个位分离出来了，可以单独求解。继续经过上面的变换，可以把指数的 6 和 7 也分离出来。最终可以把大数 b 一位一位的分离出来。至于计算 a^b 就结果快速幂求解。

代码

```
package leetcode

// 解法一 快速幂 res = res^10 * qpow(a, b[i])
// 模运算性质一：(a + b) % p = (a % p + b % p) % p
// 模运算性质二：(a - b) % p = (a % p - b % p + p) % p
// 模运算性质三：(a * b) % p = (a % p * b % p) % p
// 模运算性质四：a ^ b % p = ((a % p)^b) % p
// 举个例子
// 12345^678 % 1337 = (12345^670 * 12345^8) % 1337
//                  = ((12345^670 % 1337) * (12345^8 % 1337)) % 1337  ---> 利用性质 三
//                  = (((12345^67)^10 % 1337) * (12345^8 % 1337)) % 1337  ---> 乘方性质
//                  = ((12345^67 % 1337)^10) % 1337 * (12345^8 % 1337)) % 1337  ---> 利用性质 四
//                  = (((12345^67 % 1337)^10) * (12345^8 % 1337)) % 1337  ---> 反向利用性质 三
func superPow(a int, b []int) int {
  res := 1
  for i := 0; i < len(b); i++ {
    res = (qpow(res, 10) * qpow(a, b[i])) % 1337
  }
  return res
}

// 快速幂计算 x^n
func qpow(x, n int) int {
  res := 1
  x %= 1337
  for n > 0 {
    if (n & 1) == 1 {
      res = (res * x) % 1337
    }
    x = (x * x) % 1337
    n >>= 1
  }
  return res
}

// 解法二 暴力解法
// 利用上面的性质，可以得到：a^1234567 % 1337 = (a^1234560 % 1337) * (a^7 % 1337) % k = ((((a^123456) % 1337)^10)% 1337 * (a^7 % 1337))% 1337;
func superPow1(a int, b []int) int {
  if len(b) == 0 {
    return 1
```

```
  }
  last := b[len(b)-1]
  l := 1
  // 先计算个位的 a^x 结果，对应上面例子中的 (a^7 % 1337)% 1337
  for i := 1; i <= last; i++ {
    l = l * a % 1337
  }
  // 再计算除去个位以外的 a^y 的结果，对应上面例子中的 (a^123456) % 1337)
  temp := superPow1(a, b[:len(b)-1])
  f := 1
  // 对应上面例子中的 (((a^123456) % 1337)^10)% 1337
  for i := 1; i <= 10; i++ {
    f = f * temp % 1337
  }
  return f * l % 1337
}
```

373. Find K Pairs with Smallest Sums

题目

You are given two integer arrays **nums1** and **nums2** sorted in ascending order and an integer **k**.

Define a pair **(u,v)** which consists of one element from the first array and one element from the second array.

Find the k pairs **(u1,v1),(u2,v2) ...(uk,vk)** with the smallest sums.

Example 1:

```
Input: nums1 = [1,7,11], nums2 = [2,4,6], k = 3
Output: [[1,2],[1,4],[1,6]]
Explanation: The first 3 pairs are returned from the sequence:
             [1,2],[1,4],[1,6],[7,2],[7,4],[11,2],[7,6],[11,4],[11,6]
```

Example 2:

```
Input: nums1 = [1,1,2], nums2 = [1,2,3], k = 2
Output: [1,1],[1,1]
Explanation: The first 2 pairs are returned from the sequence:
             [1,1],[1,1],[1,2],[2,1],[1,2],[2,2],[1,3],[1,3],[2,3]
```

Example 3:

```
Input: nums1 = [1,2], nums2 = [3], k = 3
Output: [1,3],[2,3]
Explanation: All possible pairs are returned from the sequence: [1,3],[2,3]
```

题目大意

给定两个以升序排列的整形数组 nums1 和 nums2, 以及一个整数 k。

定义一对值 (u,v)，其中第一个元素来自 nums1，第二个元素来自 nums2。

找到和最小的 k 对数字 (u1,v1), (u2,v2) ... (uk,vk)。

解题思路

- 给出 2 个数组，和数字 k，要求找到 k 个数值对，数值对两个数的和最小。
- 这一题咋一看可以用二分搜索，两个数组两个组合有 `m * n` 个数值对。然后找到最小的和，最大的和，在这个范围内进行二分搜索，每分出一个 `mid`，再去找比 `mid` 小的数值对有多少个，如果个数小于 `k` 个，那么在右区间上继续二分，如果个数大于 `k` 个，那么在左区间上继续二分。到目前为止，这个思路看似可行。但是每次搜索的数值对是无序的。这会导致最终出现错误的结果。例如 `mid = 10` 的时候，小于 10 的和有 22 个，而 `k = 25` 。这说明 `mid` 偏小，`mid` 增大，`mid = 11` 的时候，小于 11 的和有 30 个，而 `k = 25` 。这时候应该从这 30 个和中取前 25 个。但是我们遍历数值对的时候，和并不是从小到大排序的。这时候还需要额外对这 30 个候选值进行排序。这样时间复杂度又增大了。
- 可以先用暴力解法解答。将所有的和都遍历出来，排序以后，取前 k 个。这个暴力方法可以 AC。
- 本题最优解应该是优先队列。维护一个最小堆。把数值对的和放在这个最小堆中，不断的 pop 出 k 个最小值到数组中，即为答案。
- 在已排序的矩阵中寻找最 K 小的元素这一系列的题目有：第 373 题，第 378 题，第 668 题，第 719 题，第 786 题。

代码

```go
package leetcode

import (
  "container/heap"
  "sort"
)

// 解法一 优先队列
func kSmallestPairs(nums1 []int, nums2 []int, k int) [][]int {
  result, h := [][]int{}, &minHeap{}
  if len(nums1) == 0 || len(nums2) == 0 || k == 0 {
    return result
  }
  if len(nums1)*len(nums2) < k {
    k = len(nums1) * len(nums2)
  }
  heap.Init(h)
  for _, num := range nums1 {
    heap.Push(h, []int{num, nums2[0], 0})
  }
```

```go
  for len(result) < k {
    min := heap.Pop(h).([]int)
    result = append(result, min[:2])
    if min[2] < len(nums2)-1 {
      heap.Push(h, []int{min[0], nums2[min[2]+1], min[2] + 1})
    }
  }
  return result
}

type minHeap [][]int

func (h minHeap) Len() int           { return len(h) }
func (h minHeap) Less(i, j int) bool { return h[i][0]+h[i][1] < h[j][0]+h[j][1] }
func (h minHeap) Swap(i, j int)      { h[i], h[j] = h[j], h[i] }

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[0 : n-1]
  return x
}

// 解法二 暴力解法
func kSmallestPairs1(nums1 []int, nums2 []int, k int) [][]int {
  size1, size2, res := len(nums1), len(nums2), [][]int{}
  if size1 == 0 || size2 == 0 || k < 0 {
    return nil
  }
  for i := 0; i < size1; i++ {
    for j := 0; j < size2; j++ {
      res = append(res, []int{nums1[i], nums2[j]})
    }
  }
  sort.Slice(res, func(i, j int) bool {
    return res[i][0]+res[i][1] < res[j][0]+res[j][1]
  })
  if len(res) >= k {
    return res[:k]
  }
  return res
}
```

374. Guess Number Higher or Lower

题目

We are playing the Guess Game. The game is as follows:

I pick a number from `1` to `n`. You have to guess which number I picked.

Every time you guess wrong, I will tell you whether the number I picked is higher or lower than your guess.

You call a pre-defined API `int guess(int num)`, which returns 3 possible results:

- `1`: The number I picked is lower than your guess (i.e. `pick < num`).
- `1`: The number I picked is higher than your guess (i.e. `pick > num`).
- `0`: The number I picked is equal to your guess (i.e. `pick == num`).

Return *the number that I picked*.

Example 1:

```
Input: n = 10, pick = 6
Output: 6
```

Example 2:

```
Input: n = 1, pick = 1
Output: 1
```

Example 3:

```
Input: n = 2, pick = 1
Output: 1
```

Example 4:

```
Input: n = 2, pick = 2
Output: 2
```

Constraints:

- `1 <= n <= 231 - 1`
- `1 <= pick <= n`

题目大意

猜数字游戏的规则如下：

- 每轮游戏，我都会从 1 到 n 随机选择一个数字。 请你猜选出的是哪个数字。
- 如果你猜错了，我会告诉你，你猜测的数字比我选出的数字是大了还是小了。

你可以通过调用一个预先定义好的接口 int guess(int num) 来获取猜测结果，返回值一共有 3 种可能的情况（-1，1 或 0）：

- 1：我选出的数字比你猜的数字小 pick < num
- 1：我选出的数字比你猜的数字大 pick > num
- 0：我选出的数字和你猜的数字一样。恭喜！你猜对了！ pick == num

返回我选出的数字。

解题思路

- 这一题是简单题，和小时候玩的猜大猜小的游戏一样。思路很简单，二分查找即可。这一题和第 278 题类似。

代码

```
package leetcode

import "sort"

/**
 * Forward declaration of guess API.
 * @param  num   your guess
 * @return       -1 if num is lower than the guess number
 *            1 if num is higher than the guess number
 *               otherwise return 0
 * func guess(num int) int;
 */

func guessNumber(n int) int {
  return sort.Search(n, func(x int) bool { return guess(x) <= 0 })
}

func guess(num int) int {
  return 0
}
```

376. Wiggle Subsequence

题目

Given an integer array `nums`, return *the length of the longest* ***wiggle sequence***.

A **wiggle sequence** is a sequence where the differences between successive numbers strictly alternate between positive and negative. The first difference (if one exists) may be either positive or negative. A sequence with fewer than two elements is trivially a wiggle sequence.

- For example, `[1, 7, 4, 9, 2, 5]` is a **wiggle sequence** because the differences `(6, -3, 5, -7, 3)` are alternately positive and negative.

- In contrast, `[1, 4, 7, 2, 5]` and `[1, 7, 4, 5, 5]` are not wiggle sequences, the first because its first two differences are positive and the second because its last difference is zero.

A **subsequence** is obtained by deleting some elements (eventually, also zero) from the original sequence, leaving the remaining elements in their original order.

Example 1:

```
Input: nums = [1,7,4,9,2,5]
Output: 6
Explanation: The entire sequence is a wiggle sequence.
```

Example 2:

```
Input: nums = [1,17,5,10,13,15,10,5,16,8]
Output: 7
Explanation: There are several subsequences that achieve this length. One is
[1,17,10,13,10,16,8].
```

Example 3:

```
Input: nums = [1,2,3,4,5,6,7,8,9]
Output: 2
```

Constraints:

- `1 <= nums.length <= 1000`
- `0 <= nums[i] <= 1000`

Follow up: Could you solve this in `O(n)` time?

题目大意

如果连续数字之间的差严格地在正数和负数之间交替，则数字序列称为摆动序列。第一个差（如果存在的话）可能是正数或负数。少于两个元素的序列也是摆动序列。例如， [1,7,4,9,2,5] 是一个摆动序列，因为差值 (6,-3,5,-7,3) 是正负交替出现的。相反, [1,4,7,2,5] 和 [1,7,4,5,5] 不是摆动序列，第一个序列是因为它的前两个差值都是正数，第二个序列是因为它的最后一个差值为零。给定一个整数序列，返回作为摆动序列的最长子序列的长度。 通过从原始序列中删除一些（也可以不删除）元素来获得子序列，剩下的元素保持其原始顺序。

解题思路

- 题目要求找到摆动序列最长的子序列。本题可以用贪心的思路，记录当前序列的上升和下降的趋势。扫描数组过程中，每扫描一个元素都判断是“峰”还是“谷”，根据前一个是“峰”还是“谷”做出对应的决定。利用贪心的思想找到最长的摆动子序列。

代码

```go
package leetcode
```

```go
func wiggleMaxLength(nums []int) int {
  if len(nums) < 2 {
    return len(nums)
  }
  res := 1
  prevDiff := nums[1] - nums[0]
  if prevDiff != 0 {
    res = 2
  }
  for i := 2; i < len(nums); i++ {
    diff := nums[i] - nums[i-1]
    if diff > 0 && prevDiff <= 0 || diff < 0 && prevDiff >= 0 {
      res++
      prevDiff = diff
    }
  }
  return res
}
```

377. Combination Sum IV

题目

Given an array of **distinct** integers `nums` and a target integer `target`, return *the number of possible combinations that add up to* `target`.

The answer is **guaranteed** to fit in a **32-bit** integer.

Example 1:

```
Input: nums = [1,2,3], target = 4
Output: 7
Explanation:
The possible combination ways are:
(1, 1, 1, 1)
(1, 1, 2)
(1, 2, 1)
(1, 3)
(2, 1, 1)
(2, 2)
(3, 1)
Note that different sequences are counted as different combinations.
```

Example 2:

```
Input: nums = [9], target = 3
Output: 0
```

Constraints:

- `1 <= nums.length <= 200`
- `1 <= nums[i] <= 1000`
- All the elements of `nums` are **unique**.
- `1 <= target <= 1000`

Follow up: What if negative numbers are allowed in the given array? How does it change the problem? What limitation we need to add to the question to allow negative numbers?

题目大意

给你一个由 不同 整数组成的数组 nums ，和一个目标整数 target 。请你从 nums 中找出并返回总和为 target 的元素组合的个数。题目数据保证答案符合 32 位整数范围。

解题思路

- Combination Sum 这是系列问题。拿到题目，笔者先用暴力解法 dfs 尝试了一版，包含的重叠子问题特别多，剪枝条件也没有写好，果然超时。元素只有 [1,2,3] 这三种，target = 32，这组数据居然有 181997601 这么多种情况。仔细看了题目数据规模 1000，基本可以断定此题是动态规划，并且时间复杂度是 O(n^2)。
- 本题和完全背包有点像，但是还是有区别。完全背包的取法内部不区分顺序。例如 5 = 1 + 2 + 2。但是本题是 3 种答案 (1，2，2)，(2，1，2)，(2，2，1)。定义 dp[i] 为总和为 target = i 的组合总数。最终答案存在 dp[target] 中。状态转移方程为：

  {{< katex display >}}
  dp[i] =\left{\begin{matrix}1,i=0\ \sum dp[i-j],i\neq 0\end{matrix}\right.
  {{< /katex >}}
- 这道题最后有一个进阶问题。如果给定的数组中含有负数，则会导致出现无限长度的排列。例如，假设数组 nums 中含有正整数 a 和负整数 −b（其中 a>0,b>0,-b<0），则有 a×b+(−b)×a=0，对于任意一个元素之和等于 target 的排列，在该排列的后面添加 b 个 a 和 a 个 −b 之后，得到的新排列的元素之和仍然等于 target，而且还可以在新排列的后面继续 b 个 a 和 a 个 −b。因此只要存在元素之和等于 target 的排列，就能构造出无限长度的排列。如果允许负数出现，则必须限制排列的最大长度，不然会出现无限长度的排列。

代码

```go
package leetcode

func combinationSum4(nums []int, target int) int {
  dp := make([]int, target+1)
  dp[0] = 1
  for i := 1; i <= target; i++ {
    for _, num := range nums {
      if i-num >= 0 {
        dp[i] += dp[i-num]
      }
```

```go
    }
  }
  return dp[target]
}

// 暴力解法超时
func combinationSum41(nums []int, target int) int {
  if len(nums) == 0 {
    return 0
  }
  c, res := []int{}, 0
  findcombinationSum4(nums, target, 0, c, &res)
  return res
}

func findcombinationSum4(nums []int, target, index int, c []int, res *int) {
  if target <= 0 {
    if target == 0 {
      *res++
    }
    return
  }
  for i := 0; i < len(nums); i++ {
    c = append(c, nums[i])
    findcombinationSum4(nums, target-nums[i], i, c, res)
    c = c[:len(c)-1]
  }
}
```

378. Kth Smallest Element in a Sorted Matrix

题目

Given a n x n matrix where each of the rows and columns are sorted in ascending order, find the kth smallest element in the matrix.

Note that it is the kth smallest element in the sorted order, not the kth distinct element.

Example:

```
matrix = [
   [ 1,  5,  9],
   [10, 11, 13],
   [12, 13, 15]
],
k = 8,

return 13.
```

Note: You may assume k is always valid, 1 ≤ k ≤ n2.

题目大意

给定一个 n x n 矩阵，其中每行和每列元素均按升序排序，找到矩阵中第 k 小的元素。请注意，它是排序后的第 k 小元素，而不是第 k 个元素。

说明:
你可以假设 k 的值永远是有效的, 1 ≤ k ≤ n2 。

解题思路

- 给出一个行有序，列有序的矩阵(并非是按照下标有序的)，要求找出这个矩阵中第 K 小的元素。注意找的第 K 小元素指的不是 k 个不同的元素，可能存在相同的元素。
- 最容易想到的就解法是优先队列。依次把矩阵中的元素推入到优先队列中。维护一个最小堆，一旦优先队列里面的元素有 k 个了，就算找到结果了。
- 这一题最优解法是二分搜索。那搜索的空间是什么呢？根据题意，可以知道，矩阵左上角的那个元素是最小的，右下角的元素是最大的。即矩阵第一个元素确定了下界，矩阵的最后一个元素确定了上界。在这个解空间里面二分搜索所有值，找到第 K 小的元素。判断是否找到的条件是，在矩阵中比 mid 小的元素个数等于 K。不断的逼近 low，使得 low == high 的时候，就是找到了第 K 小的元素了。(因为题目中说了，一定会存在第 K 小元素，所以二分搜索到一个元素的时候，一定会得出结果)。

Step 1:

middle = (start + end) / 2
middle = (2 + 11) / 2 = 6

Smallest number greater than the middle(6): 7
Biggest number less than or equal to the middle(6): 6

Number of elements less than or equal to the middle(6) = 4

As there are only 4 elements less than or equal to the middle, and we are looking for the 5th smallest number, so let's search higher and **update our 'start' to the smallest number greater than the middle**.

Step 2:

middle = (start + end) / 2
middle = (7 + 11) / 2 = 9

Smallest number greater than the middle(9): 10
Biggest number less than or equal to the middle(9): 8

Number of elements less than or equal to the middle(9) = 7

As there are 7 elements less than or equal to the middle, and we are looking for the 5th smallest number, so let's search lower and **update our 'end' to the biggest number less than or equal to the middle.**

Step 3:

middle = (start + end) / 2
middle = (7 + 8) / 2 = 7

Smallest number greater than the middle(7): 8
Biggest number less than or equal to the middle(7): 7

Number of elements less than or equal to the middle(7) = 5

As there are 5 elements less than or equal to the middle therefore '7', which is the biggest number less than or equal to the middle, is our required number

代码

```
package leetcode
```

```go
import (
  "container/heap"
)

// 解法一 二分搜索
func kthSmallest378(matrix [][]int, k int) int {
  m, n, low := len(matrix), len(matrix[0]), matrix[0][0]
  high := matrix[m-1][n-1] + 1
  for low < high {
    mid := low + (high-low)>>1
    // 如果 count 比 k 小，在大值的那一半继续二分搜索
    if counterKthSmall(m, n, mid, matrix) >= k {
      high = mid
    } else {
      low = mid + 1
    }
  }
  return low
}

func counterKthSmall(m, n, mid int, matrix [][]int) int {
  count, j := 0, n-1
  // 每次循环统计比 mid 值小的元素个数
  for i := 0; i < m; i++ {
    // 遍历每行中比 mid 小的元素的个数
    for j >= 0 && mid < matrix[i][j] {
      j--
    }
    count += j + 1
  }
  return count
}

// 解法二 优先队列
func kthSmallest3781(matrix [][]int, k int) int {
  if len(matrix) == 0 || len(matrix[0]) == 0 {
    return 0
  }
  pq := &pq{data: make([]interface{}, k)}
  heap.Init(pq)
  for i := 0; i < len(matrix); i++ {
    for j := 0; j < len(matrix[0]); j++ {
      if pq.Len() < k {
        heap.Push(pq, matrix[i][j])
      } else if matrix[i][j] < pq.Head().(int) {
        heap.Pop(pq)
        heap.Push(pq, matrix[i][j])
      } else {
```

```go
            break
        }
      }
    }
    return heap.Pop(pq).(int)
}

type pq struct {
    data []interface{}
    len  int
}

func (p *pq) Len() int {
    return p.len
}

func (p *pq) Less(a, b int) bool {
    return p.data[a].(int) > p.data[b].(int)
}

func (p *pq) Swap(a, b int) {
    p.data[a], p.data[b] = p.data[b], p.data[a]
}

func (p *pq) Push(o interface{}) {
    p.data[p.len] = o
    p.len++
}

func (p *pq) Head() interface{} {
    return p.data[0]
}

func (p *pq) Pop() interface{} {
    p.len--
    return p.data[p.len]
}
```

385. Mini Parser

题目

Given a nested list of integers represented as a string, implement a parser to deserialize it.

Each element is either an integer, or a list -- whose elements may also be integers or other lists.

Note: You may assume that the string is well-formed:

- String is non-empty.

- String does not contain white spaces.
- String contains only digits `0-9`, `[`, `-` `,`, `]`.

Example 1:

```
Given s = "324",

You should return a NestedInteger object which contains a single integer 324.
```

Example 2:

```
Given s = "[123,[456,[789]]]",

Return a NestedInteger object containing a nested list with 2 elements:

1. An integer containing value 123.
2. A nested list containing two elements:
    i.  An integer containing value 456.
    ii. A nested list with one element:
         a. An integer containing value 789.
```

题目大意

给定一个用字符串表示的整数的嵌套列表，实现一个解析它的语法分析器。列表中的每个元素只可能是整数或整数嵌套列表

提示：你可以假定这些字符串都是格式良好的：

- 字符串非空
- 字符串不包含空格
- 字符串只包含数字0-9, [, - ,,]

解题思路

- 将一个嵌套的数据结构中的数字转换成 NestedInteger 数据结构。
- 这一题用栈一层一层的处理就行。有一些比较坑的特殊的边界数据见测试文件。这一题正确率比很多 Hard 题还要低的原因应该是没有理解好题目和边界测试数据没有考虑到。NestedInteger 这个数据结构笔者实现了一遍，见代码。

代码

```go
package leetcode

import (
  "fmt"
  "strconv"
```

```
)

/**
 * // This is the interface that allows for creating nested lists.
 * // You should not implement it, or speculate about its implementation
 * type NestedInteger struct {
 * }
 *
 * // Return true if this NestedInteger holds a single integer, rather than a nested
list.
 * func (n NestedInteger) IsInteger() bool {}
 *
 * // Return the single integer that this NestedInteger holds, if it holds a single
integer
 * // The result is undefined if this NestedInteger holds a nested list
 * // So before calling this method, you should have a check
 * func (n NestedInteger) GetInteger() int {}
 *
 * // Set this NestedInteger to hold a single integer.
 * func (n *NestedInteger) SetInteger(value int) {}
 *
 * // Set this NestedInteger to hold a nested list and adds a nested integer to it.
 * func (n *NestedInteger) Add(elem NestedInteger) {}
 *
 * // Return the nested list that this NestedInteger holds, if it holds a nested list
 * // The list length is zero if this NestedInteger holds a single integer
 * // You can access NestedInteger's List element directly if you want to modify it
 * func (n NestedInteger) GetList() []*NestedInteger {}
 */

// NestedInteger define
type NestedInteger struct {
  Num  int
  List []*NestedInteger
}

// IsInteger define
func (n NestedInteger) IsInteger() bool {
  if n.List == nil {
    return true
  }
  return false
}

// GetInteger define
func (n NestedInteger) GetInteger() int {
  return n.Num
}
```

```go
// SetInteger define
func (n *NestedInteger) SetInteger(value int) {
  n.Num = value
}

// Add define
func (n *NestedInteger) Add(elem NestedInteger) {
  n.List = append(n.List, &elem)
}

// GetList define
func (n NestedInteger) GetList() []*NestedInteger {
  return n.List
}

// Print define
func (n NestedInteger) Print() {
  if len(n.List) != 0 {
    for _, v := range n.List {
      if len(v.List) != 0 {
        v.Print()
        return
      }
      fmt.Printf("%v ", v.Num)
    }
  } else {
    fmt.Printf("%v ", n.Num)
  }
  fmt.Printf("\n")
}

func deserialize(s string) *NestedInteger {
  stack, cur := []*NestedInteger{}, &NestedInteger{}
  for i := 0; i < len(s); {
    switch {
    case isDigital(s[i]) || s[i] == '-':
      j := 0
      for j = i + 1; j < len(s) && isDigital(s[j]); j++ {
      }
      num, _ := strconv.Atoi(s[i:j])
      next := &NestedInteger{}
      next.SetInteger(num)
      if len(stack) > 0 {
        stack[len(stack)-1].List = append(stack[len(stack)-1].GetList(), next)
      } else {
        cur = next
      }
      i = j
    case s[i] == '[':
```

```
      next := &NestedInteger{}
      if len(stack) > 0 {
        stack[len(stack)-1].List = append(stack[len(stack)-1].GetList(), next)
      }
      stack = append(stack, next)
      i++
    case s[i] == ']':
      cur = stack[len(stack)-1]
      stack = stack[:len(stack)-1]
      i++
    case s[i] == ',':
      i++
    }
  }
  return cur
}
```

386. Lexicographical Numbers

题目

Given an integer n, return 1 - n in lexicographical order.

For example, given 13, return: [1,10,11,12,13,2,3,4,5,6,7,8,9].

Please optimize your algorithm to use less time and space. The input size may be as large as 5,000,000.

题目大意

给定一个整数 n, 返回从 1 到 n 的字典顺序。例如，给定 n =13，返回 [1,10,11,12,13,2,3,4,5,6,7,8,9] 。

请尽可能的优化算法的时间复杂度和空间复杂度。 输入的数据 n 小于等于 5,000,000。

解题思路

- 给出一个数字 n ，要求按照字典序对 1-n 这 n 个数排序。
- DFS 暴力求解即可。

代码

```
package leetcode

func lexicalOrder(n int) []int {
  res := make([]int, 0, n)
  dfs386(1, n, &res)
  return res
```

```
}

func dfs386(x, n int, res *[]int) {
  limit := (x + 10) / 10 * 10
  for x <= n && x < limit {
    *res = append(*res, x)
    if x*10 <= n {
      dfs386(x*10, n, res)
    }
    x++
  }
}
```

387. First Unique Character in a String

题目

Given a string, find the first non-repeating character in it and return it's index. If it doesn't exist, return -1.

Examples:

```
s = "leetcode"
return 0.

s = "loveleetcode",
return 2.
```

Note: You may assume the string contain only lowercase letters.

题目大意

给定一个字符串，找到它的第一个不重复的字符，并返回它的索引。如果不存在，则返回 -1。

解题思路

- 简单题，要求输出第一个没有重复的字符。
- 解法二这个思路只超过 81% 的用户，但是如果测试样例中 s 的字符串很长，但是满足条件的字符都在靠后的位置的话，这个思路应该会更有优势。通过记录每个字符的第一次出现的位置和最后一次出现的位置。第一次对 s 进行一次遍历。第二次仅仅对数组进行遍历就可以了。

代码

```
package leetcode
```

```go
// 解法 一
func firstUniqChar(s string) int {
  result := make([]int, 26)
  for i := 0; i < len(s); i++ {
    result[s[i]-'a']++
  }
  for i := 0; i < len(s); i++ {
    if result[s[i]-'a'] == 1 {
      return i
    }
  }
  return -1
}

// 解法 二
// 执行用时: 8 ms
// 内存消耗: 5.2 MB
func firstUniqChar1(s string) int {
  charMap := make([][2]int, 26)
  for i := 0; i < 26; i++ {
    charMap[i][0] = -1
    charMap[i][1] = -1
  }
  for i := 0; i < len(s); i++ {
    if charMap[s[i]-'a'][0] == -1 {
      charMap[s[i]-'a'][0] = i
    } else { //已经出现过
      charMap[s[i]-'a'][1] = i
    }
  }
  res := len(s)
  for i := 0; i < 26; i++ {
    //只出现了一次
    if charMap[i][0] >= 0 && charMap[i][1] == -1 {
      if charMap[i][0] < res {
        res = charMap[i][0]
      }
    }
  }
  if res == len(s) {
    return -1
  }
  return res
}
```

389. Find the Difference

题目

Given two strings **s** and **t** which consist of only lowercase letters.

String **t** is generated by random shuffling string **s** and then add one more letter at a random position.

Find the letter that was added in **t**.

Example:

```
Input:
s = "abcd"
t = "abcde"

Output:
e

Explanation:
'e' is the letter that was added.
```

题目大意

给定两个字符串 s 和 t，它们只包含小写字母。字符串 t 由字符串 s 随机重排，然后在随机位置添加一个字母。请找出在 t 中被添加的字母。

解题思路

- 题目要求找出 t 字符串中比 s 字符串多出的一个字符。思路还是利用异或的性质，`x^x = 0`，将 s 和 t 依次异或，最终多出来的字符就是最后异或的结果。

代码

```go
package leetcode

func findTheDifference(s string, t string) byte {
  n, ch := len(t), t[len(t)-1]
  for i := 0; i < n-1; i++ {
    ch ^= s[i]
    ch ^= t[i]
  }
  return ch
}
```

392. Is Subsequence

题目

Given a string **s** and a string **t**, check if **s** is subsequence of **t**.

You may assume that there is only lower case English letters in both **s** and **t**. **t** is potentially a very long (length ~= 500,000) string, and **s** is a short string (<=100).

A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, `"ace"` is a subsequence of `"abcde"` while `"aec"` is not).

Example 1:

```
Input: s = "abc", t = "ahbgdc"
Output: true
```

Example 2:

```
Input: s = "axc", t = "ahbgdc"
Output: false
```

Follow up: If there are lots of incoming S, say S1, S2, ... , Sk where k >= 1B, and you want to check one by one to see if T has its subsequence. In this scenario, how would you change your code?

Credits: Special thanks to @pbrother for adding this problem and creating all test cases.

题目大意

给定字符串 s 和 t ，判断 s 是否为 t 的子序列。你可以认为 s 和 t 中仅包含英文小写字母。字符串 t 可能会很长（长度 ~= 500,000），而 s 是个短字符串（长度 <=100）。字符串的一个子序列是原始字符串删除一些（也可以不删除）字符而不改变剩余字符相对位置形成的新字符串。（例如，"ace"是"abcde"的一个子序列，而"aec"不是）。

解题思路

- 给定 2 个字符串 s 和 t，问 s 是不是 t 的子序列。注意 s 在 t 中还需要保持 s 的字母的顺序。
- 这是一题贪心算法。直接做即可。

代码

```go
package leetcode

// 解法一 O(n^2)
func isSubsequence(s string, t string) bool {
  index := 0
  for i := 0; i < len(s); i++ {
    flag := false
```

```
    for ; index < len(t); index++ {
      if s[i] == t[index] {
        flag = true
        break
      }
    }
    if flag == true {
      index++
      continue
    } else {
      return false
    }
  }
  return true
}

// 解法二 O(n)
func isSubsequence1(s string, t string) bool {
  for len(s) > 0 && len(t) > 0 {
    if s[0] == t[0] {
      s = s[1:]
    }
    t = t[1:]
  }
  return len(s) == 0
}
```

393. UTF-8 Validation

题目

A character in UTF8 can be from **1 to 4 bytes** long, subjected to the following rules:

1. For 1-byte character, the first bit is a 0, followed by its unicode code.
2. For n-bytes character, the first n-bits are all one's, the n+1 bit is 0, followed by n-1 bytes with most significant 2 bits being 10.

This is how the UTF-8 encoding would work:

```
   Char. number range  |        UTF-8 octet sequence
      (hexadecimal)    |              (binary)
   --------------------+---------------------------------------------
   0000 0000-0000 007F | 0xxxxxxx
   0000 0080-0000 07FF | 110xxxxx 10xxxxxx
   0000 0800-0000 FFFF | 1110xxxx 10xxxxxx 10xxxxxx
   0001 0000-0010 FFFF | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
```

Given an array of integers representing the data, return whether it is a valid utf-8 encoding.

Note: The input is an array of integers. Only the **least significant 8 bits** of each integer is used to store the data. This means each integer represents only 1 byte of data.

Example 1:

```
data = [197, 130, 1], which represents the octet sequence: 11000101 10000010 00000001.

Return true.
It is a valid utf-8 encoding for a 2-bytes character followed by a 1-byte character.
```

Example 2:

```
data = [235, 140, 4], which represented the octet sequence: 11101011 10001100 00000100.

Return false.
The first 3 bits are all one's and the 4th bit is 0 means it is a 3-bytes character.
The next byte is a continuation byte which starts with 10 and that's correct.
But the second continuation byte does not start with 10, so it is invalid.
```

题目大意

UTF-8 中的一个字符可能的长度为 1 到 4 字节，遵循以下的规则：

对于 1 字节的字符，字节的第一位设为 0，后面 7 位为这个符号的 unicode 码。
对于 n 字节的字符 (n > 1)，第一个字节的前 n 位都设为 1，第 n+1 位设为 0，后面字节的前两位一律设为 10。剩下的没有提及的二进制位，全部为这个符号的 unicode 码。
这是 UTF-8 编码的工作方式：

```
   Char. number range  |        UTF-8 octet sequence
      (hexadecimal)    |              (binary)
   --------------------+---------------------------------------------
   0000 0000-0000 007F | 0xxxxxxx
   0000 0080-0000 07FF | 110xxxxx 10xxxxxx
   0000 0800-0000 FFFF | 1110xxxx 10xxxxxx 10xxxxxx
   0001 0000-0010 FFFF | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
```

给定一个表示数据的整数数组，返回它是否为有效的 utf-8 编码。

注意:

输入是整数数组。只有每个整数的最低 8 个有效位用来存储数据。这意味着每个整数只表示 1 字节的数据。

解题思路

- 这一题看似很复杂，其实严格按照 UTF8 定义来模拟就可以了。

代码

```
package leetcode

func validUtf8(data []int) bool {
  count := 0
  for _, d := range data {
    if count == 0 {
      if d >= 248 { // 11111000 = 248
        return false
      } else if d >= 240 { // 11110000 = 240
        count = 3
      } else if d >= 224 { // 11100000 = 224
        count = 2
      } else if d >= 192 { // 11000000 = 192
        count = 1
      } else if d > 127 { // 01111111 = 127
        return false
      }
    } else {
      if d <= 127 || d >= 192 {
        return false
      }
      count--
    }
  }
  return count == 0
}
```

394. Decode String

题目

Given an encoded string, return its decoded string.

The encoding rule is: k[encoded_string], where the encoded_string inside the square brackets is being repeated exactly k times. Note that k is guaranteed to be a positive integer.

You may assume that the input string is always valid; No extra white spaces, square brackets are well-formed, etc.

Furthermore, you may assume that the original data does not contain any digits and that digits are only for those repeat numbers, k. For example, there won't be input like 3a or 2[4].

Examples:

```
s = "3[a]2[bc]", return "aaabcbc".
s = "3[a2[c]]", return "accaccacc".
s = "2[abc]3[cd]ef", return "abcabccdcdcdef".
```

题目大意

给定一个经过编码的字符串，返回它解码后的字符串。编码规则为: k[encoded_string]，表示其中方括号内部的 encoded_string 正好重复 k 次。注意 k 保证为正整数。你可以认为输入字符串总是有效的；输入字符串中没有额外的空格，且输入的方括号总是符合格式要求的。此外，你可以认为原始数据不包含数字，所有的数字只表示重复的次数 k ，例如不会出现像 3a 或 2[4] 的输入。

解题思路

这一题和第 880 题大体类似。用栈处理，遇到 "["，就要开始重复字符串了，另外重复的数字是可能存在多位的，所以需要往前找到不为数字的那一位，把数字转换出来。最后用把 stack 里面的字符串都串联起来即可。

代码

```go
package leetcode

import (
  "strconv"
)

func decodeString(s string) string {
  stack, res := []string{}, ""
  for _, str := range s {
    if len(stack) == 0 || (len(stack) > 0 && str != ']') {
      stack = append(stack, string(str))
    } else {
      tmp := ""
      for stack[len(stack)-1] != "[" {
        tmp = stack[len(stack)-1] + tmp
        stack = stack[:len(stack)-1]
      }
      stack = stack[:len(stack)-1]
```

```
      index, repeat := 0, ""
      for index = len(stack) - 1; index >= 0; index-- {
        if stack[index] >= "0" && stack[index] <= "9" {
          repeat = stack[index] + repeat
        } else {
          break
        }
      }
      nums, _ := strconv.Atoi(repeat)
      copyTmp := tmp
      for i := 0; i < nums-1; i++ {
        tmp += copyTmp
      }
      for i := 0; i < len(repeat)-1; i++ {
        stack = stack[:len(stack)-1]
      }
      stack[index+1] = tmp
    }
  }
  for _, s := range stack {
    res += s
  }
  return res
}
```

395. Longest Substring with At Least K Repeating Characters

题目

Given a string `s` and an integer `k`, return *the length of the longest substring of* `s` *such that the frequency of each character in this substring is greater than or equal to* `k`.

Example 1:

```
Input: s = "aaabb", k = 3
Output: 3
Explanation: The longest substring is "aaa", as 'a' is repeated 3 times.
```

Example 2:

```
Input: s = "ababbc", k = 2
Output: 5
Explanation: The longest substring is "ababb", as 'a' is repeated 2 times and 'b' is
repeated 3 times.
```

Constraints:

- `1 <= s.length <= 10^4`
- `s` consists of only lowercase English letters.
- `1 <= k <= 10^5`

题目大意

给你一个字符串 s 和一个整数 k ，请你找出 s 中的最长子串， 要求该子串中的每一字符出现次数都不少于 k 。返回这一子串的长度。

解题思路

- 最容易想到的思路是递归。如果某个字符出现次数大于 0 小于 k，那么包含这个字符的子串都不满足要求。所以按照这个字符来切分整个字符串，满足题意的最长子串一定不包含切分的字符。切分完取出最长子串即可。时间复杂度 O(26*n)，空间复杂度 O(26^2)
- 此题另外一个思路是滑动窗口。有一个需要解决的问题是右窗口移动的条件。此题要求最长字符串，那么这个最终的字符串内包含的字符种类最多是 26 种。字符种类就是右窗口移动的条件。依次枚举字符种类，如果当前窗口内的字符种类小于当前枚举的字符种类，那么窗口右移，否则左移。窗口移动中需要动态维护 freq 频次数组。可以每次都循环一遍这个数组，计算出出现次数大于 k 的字符。虽然这种做法只最多循环 26 次，但是还是不高效。更高效的做法是维护 1 个值，一个用来记录当前出现次数小于 k 次的字符种类数 `less`。如果 freq 为 0 ，说明小于 k 次的字符种类数要发生变化，如果是右窗口移动，那么 `less++`，如果是左窗口移动，那么`less--`。同理，如果 freq 为 k ，说明小于 k 次的字符种类数要发生变化，如果是右窗口移动，那么 `less--`，如果是左窗口移动，那么`less++`。在枚举 26 个字符种类中，动态维护记录出最长字符串。枚举完成，最长字符串长度也就求出来了。时间复杂度 O(26*n)，空间复杂度 O(26)

代码

```
package leetcode

import "strings"

// 解法一 滑动窗口
func longestSubstring(s string, k int) int {
  res := 0
  for t := 1; t <= 26; t++ {
    freq, total, lessK, left, right := [26]int{}, 0, 0, 0, -1
    for left < len(s) {
      if right+1 < len(s) && total <= t {
        if freq[s[right+1]-'a'] == 0 {
          total++
          lessK++
        }
        freq[s[right+1]-'a']++
        if freq[s[right+1]-'a'] == k {
          lessK--
        }
        right++
```

```go
      } else {
        if freq[s[left]-'a'] == k {
          lessK++
        }
        freq[s[left]-'a']--
        if freq[s[left]-'a'] == 0 {
          total--
          lessK--
        }
        left++
      }
      if lessK == 0 {
        res = max(res, right-left+1)
      }

    }
  }
  return res
}

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

// 解法二 递归分治
func longestSubstring1(s string, k int) int {
  if s == "" {
    return 0
  }
  freq, split, res := [26]int{}, byte(0), 0
  for _, ch := range s {
    freq[ch-'a']++
  }
  for i, c := range freq[:] {
    if 0 < c && c < k {
      split = 'a' + byte(i)
      break
    }
  }
  if split == 0 {
    return len(s)
  }
  for _, subStr := range strings.Split(s, string(split)) {
    res = max(res, longestSubstring1(subStr, k))
  }
  return res
```

```
}
```

397. Integer Replacement

题目

Given a positive integer n and you can do operations as follow:

1. If n is even, replace n with `n/2`.
2. If n is odd, you can replace n with either `n + 1` or `n - 1`.

What is the minimum number of replacements needed for n to become 1?

Example 1:

```
Input:
8

Output:
3

Explanation:
8 -> 4 -> 2 -> 1
```

Example 2:

```
Input:
7

Output:
4

Explanation:
7 -> 8 -> 4 -> 2 -> 1
or
7 -> 6 -> 3 -> 2 -> 1
```

题目大意

给定一个正整数 n，你可以做如下操作：

1. 如果 n 是偶数，则用 n / 2 替换 n。
2. 如果 n 是奇数，则可以用 n + 1 或 n - 1 替换 n。

问 n 变为 1 所需的最小替换次数是多少?

解题思路

- 题目给出一个整数 `n`，然后让我们通过变换将它为 1，如果 `n` 是偶数，可以直接变为 `n/2`，如果是奇数，可以先 `n+1` 或 `n-1`，问最终变为 1 的最少步骤。
- 当 n 为奇数的时候，什么时候需要加 1 ，什么时候需要减 1 ，通过观察规律可以发现，除了 3 和 7 以外，所有加 1 就变成 4 的倍数的奇数，都适合先加 1 运算，比如 15:

```
15 -> 16 -> 8 -> 4 -> 2 -> 1
15 -> 14 -> 7 -> 6 -> 3 -> 2 -> 1

111011 -> 111010 -> 11101 -> 11100 -> 1110 -> 111 -> 1000 -> 100 -> 10 -> 1
111011 -> 111100 -> 11110 -> 1111 -> 10000 -> 1000 -> 100 -> 10 -> 1
```

- 对于 7 来说，加 1 和减 1 的结果相同，可以不用管，对于 3 来说，减 1 的步骤更少，所以需要先去掉这种特殊情况。
- 最后如何判断某个数字加 1 后是 4 的倍数呢？这里有一个小技巧，由于之前判断了其是奇数了，那么最右边一位肯定是 1，如果其右边第二位也是 1 的话，那么进行加 1 运算，进位后右边肯定会出现两个 0，则一定是 4 的倍数。于是就可以判断出来了。剩下的情况就是偶数的情况，如果之前判定是偶数，那么直接除以 2 (右移一位)即可。

代码

```go
package leetcode

func integerReplacement(n int) int {
  res := 0
  for n > 1 {
    if (n & 1) == 0 { // 判断是否是偶数
      n >>= 1
    } else if (n+1)%4 == 0 && n != 3 { // 末尾 2 位为 11
      n++
    } else { // 末尾 2 位为 01
      n--
    }
    res++
  }
  return res
}
```

399. Evaluate Division

题目

Equations are given in the format `A / B = k`, where `A` and `B` are variables represented as strings, and `k` is a real number (floating point number). Given some queries, return the answers. If the answer does not exist, return `-1.0`.

Example:

Given `a / b = 2.0, b / c = 3.0.` queries are: `a / c = ?, b / a = ?, a / e = ?, a / a = ?, x / x = ? .` return `[6.0, 0.5, -1.0, 1.0, -1.0 ].`

The input is: `vector<pair<string, string>> equations, vector<double>& values, vector<pair<string, string>> queries`, where `equations.size() == values.size()`, and the values are positive. This represents the equations. Return `vector<double>`.

According to the example above:

```
equations = [ ["a", "b"], ["b", "c"] ],
values = [2.0, 3.0],
queries = [ ["a", "c"], ["b", "a"], ["a", "e"], ["a", "a"], ["x", "x"] ].
```

The input is always valid. You may assume that evaluating the queries will result in no division by zero and there is no contradiction.

题目大意

给出方程式 A / B = k, 其中 A 和 B 均为代表字符串的变量， k 是一个浮点型数字。根据已知方程式求解问题，并返回计算结果。如果结果不存在，则返回 -1.0。

示例 :
给定 a / b = 2.0, b / c = 3.0
问题: a / c = ?, b / a = ?, a / e = ?, a / a = ?, x / x = ?
返回 [6.0, 0.5, -1.0, 1.0, -1.0]

输入为: vector<pair<string, string>> equations, vector& values, vector<pair<string, string>> queries(方程式，方程式结果，问题方程式)， 其中 equations.size() == values.size()，即方程式的长度与方程式结果长度相等（程式与结果一一对应），并且结果值均为正数。以上为方程式的描述。 返回vector类型。

假设输入总是有效的。你可以假设除法运算中不会出现除数为0的情况，且不存在任何矛盾的结果。

解题思路

- 给出一些字母变量的倍数关系，问给出任意两个字母的倍数是多少。
- 这一题可以用 DFS 或者并查集来解题。先来看看 DFS 的做法。先建图。每个字母或者字母组合可以看做成一个节点，给出的 `equations` 关系可以看成两个节点之间的有向边。每条有向边都有权值。那么问题可以转换成是否存在一条从起点节点到终点节点的路径，如果存在，输出这条路径上所有有向边权值的累乘结果。如果不存在这条路径，就返回 -1 。如果给的起点和终点不在给出的节点集里面，也输出 -1 。
- 再来看看并查集的做法。先将每两个有倍数关系的节点做并查集 `union()` 操作。例如 A/B = 2，那么把 B 作为 `parent` 节点， `parents[A] = {B, 2}`， `parents[B] = {B, 1}`，B 指向自己是 1 。还有一个关系是 `B/C=3`，由于 B 已经在并查集中了，所以这个时候需要把这个关系反过来，处理成 `C/B = 1/3` ，即 `parents[C] = {B, 1/3}`。这样把所有有关系的字母都 `union()` 起来。如何求任意两个字母的倍数关系

呢？例如 `A/C = ?` 在并查集中查找，可以找到 `parents[C] == parents[A] == B`，那么就用 `parents[A]/parents[C] = 2/(1/3) = 6`。为什么可以这样做呢？因为 `A/B = 2`，`C/B = 1/3`，那么 `A/C = (A/B)/(C/B)` 即 `parents[A]/parents[C] = 2/(1/3) = 6`。

代码

```
package leetcode

type stringUnionFind struct {
  parents map[string]string
  vals    map[string]float64
}

func (suf stringUnionFind) add(x string) {
  if _, ok := suf.parents[x]; ok {
    return
  }
  suf.parents[x] = x
  suf.vals[x] = 1.0
}

func (suf stringUnionFind) find(x string) string {
  p := ""
  if v, ok := suf.parents[x]; ok {
    p = v
  } else {
    p = x
  }
  if x != p {
    pp := suf.find(p)
    suf.vals[x] *= suf.vals[p]
    suf.parents[x] = pp
  }
  if v, ok := suf.parents[x]; ok {
    return v
  }
  return x
}

func (suf stringUnionFind) union(x, y string, v float64) {
  suf.add(x)
  suf.add(y)
  px, py := suf.find(x), suf.find(y)
  suf.parents[px] = py
  // x / px = vals[x]
  // x / y = v
  // 由上面 2 个式子就可以得出 px = v * vals[y] / vals[x]
```

```go
  suf.vals[px] = v * suf.vals[y] / suf.vals[x]
}

func calcEquation(equations [][]string, values []float64, queries [][]string) []float64
{
  res, suf := make([]float64, len(queries)), stringUnionFind{parents:
map[string]string{}, vals: map[string]float64{}}
  for i := 0; i < len(values); i++ {
    suf.union(equations[i][0], equations[i][1], values[i])
  }
  for i := 0; i < len(queries); i++ {
    x, y := queries[i][0], queries[i][1]
    if _, ok := suf.parents[x]; ok {
      if _, ok := suf.parents[y]; ok {
        if suf.find(x) == suf.find(y) {
          res[i] = suf.vals[x] / suf.vals[y]
        } else {
          res[i] = -1
        }
      } else {
        res[i] = -1
      }
    } else {
      res[i] = -1
    }
  }
  return res
}
```

401. Binary Watch

题目

A binary watch has 4 LEDs on the top which represent the **hours** (**0-11**), and the 6 LEDs on the bottom represent the **minutes** (**0-59**).

Each LED represents a zero or one, with the least significant bit on the right.

For example, the above binary watch reads "3:25".

Given a non-negative integer n which represents the number of LEDs that are currently on, return all possible times the watch could represent.

Example:

```
Input: n = 1
Return: ["1:00", "2:00", "4:00", "8:00", "0:01", "0:02", "0:04", "0:08", "0:16",
"0:32"]
```

Note:

- The order of output does not matter.
- The hour must not contain a leading zero, for example "01:00" is not valid, it should be "1:00".
- The minute must be consist of two digits and may contain a leading zero, for example "10:2" is not valid, it should be "10:02".

题目大意

二进制手表顶部有 4 个 LED 代表小时（0-11），底部的 6 个 LED 代表分钟（0-59）。每个 LED 代表一个 0 或 1，最低位在右侧。

给定一个非负整数 n 代表当前 LED 亮着的数量，返回二进制表所有可能的时间。

解题思路

- 给出数字 n，要求输出二进制表中所有可能的时间
- 题目中比较坑的是，分钟大于 60 的都不应该打印出来，小时大于 12 的也不应该打印出来，因为是非法的。给出的 num 大于 8 的也是非法值，最终结果应该输出空字符串数组。
- 这道题的数据量不大，可以直接用打表法，具体打表函数见 `findReadBinaryWatchMinute()` 和 `findReadBinaryWatchHour()` 这两个函数。

代码

```go
package leetcode

import (
  "fmt"
  "strconv"
)

// 解法一
func readBinaryWatch(num int) []string {
  memo := make([]int, 60)
  // count the number of 1 in a binary number
  count := func(n int) int {
    if memo[n] != 0 {
      return memo[n]
    }
    originN, res := n, 0
    for n != 0 {
      n = n & (n - 1)
      res++
    }
    memo[originN] = res
    return res
  }
  // fmtMinute format minute 0:1 -> 0:01
  fmtMinute := func(m int) string {
    if m < 10 {
      return "0" + strconv.Itoa(m)
    }
    return strconv.Itoa(m)
  }

  var res []string
  // traverse 0:00 -> 12:00
  for i := 0; i < 12; i++ {
    for j := 0; j < 60; j++ {
      if count(i)+count(j) == num {
        res = append(res, strconv.Itoa(i)+":"+fmtMinute(j))
```

```go
        }
      }
    }
    return res
}

// 解法二 打表
var (
    hour    = []string{"1", "2", "4", "8"}
    minute  = []string{"01", "02", "04", "08", "16", "32"}
    hourMap = map[int][]string{
      0: {"0"},
      1: {"1", "2", "4", "8"},
      2: {"3", "5", "9", "6", "10"},
      3: {"7", "11"},
    }
    minuteMap = map[int][]string{
      0: {"00"},
      1: {"01", "02", "04", "08", "16", "32"},
      2: {"03", "05", "09", "17", "33", "06", "10", "18", "34", "12", "20", "36", "24",
"40", "48"},
      3: {"07", "11", "19", "35", "13", "21", "37", "25", "41", "49", "14", "22", "38",
"26", "42", "50", "28", "44", "52", "56"},
      4: {"15", "23", "39", "27", "43", "51", "29", "45", "53", "57", "30", "46", "54",
"58"},
      5: {"31", "47", "55", "59"},
    }
)

func readBinaryWatch1(num int) []string {
    var res []string
    if num > 8 {
      return res
    }
    for i := 0; i <= num; i++ {
      for j := 0; j < len(hourMap[i]); j++ {
        for k := 0; k < len(minuteMap[num-i]); k++ {
          res = append(res, hourMap[i][j]+":"+minuteMap[num-i][k])
        }
      }
    }
    return res
}

/// ----------------------------------------
/// ----------------------------------------
/// ----------------------------------------
/// ----------------------------------------
/// ----------------------------------------
```

```go
// 以下是打表用到的函数
// 调用 findReadBinaryWatchMinute(num, 0, c, &res) 打表
func findReadBinaryWatchMinute(target, index int, c []int, res *[]string) {
  if target == 0 {
    str, tmp := "", 0
    for i := 0; i < len(c); i++ {
      t, _ := strconv.Atoi(minute[c[i]])
      tmp += t
    }
    if tmp < 10 {
      str = "0" + strconv.Itoa(tmp)
    } else {
      str = strconv.Itoa(tmp)
    }
    // fmt.Printf("找到解了 c = %v str = %v\n", c, str)
    fmt.Printf("\"%v\", ", str)
    return
  }
  for i := index; i < 6; i++ {
    c = append(c, i)
    findReadBinaryWatchMinute(target-1, i+1, c, res)
    c = c[:len(c)-1]
  }
}

// 调用 findReadBinaryWatchHour(num, 0, c, &res) 打表
func findReadBinaryWatchHour(target, index int, c []int, res *[]string) {
  if target == 0 {
    str, tmp := "", 0
    for i := 0; i < len(c); i++ {
      t, _ := strconv.Atoi(hour[c[i]])
      tmp += t
    }
    str = strconv.Itoa(tmp)
    //fmt.Printf("找到解了 c = %v str = %v\n", c, str)
    fmt.Printf("\"%v\", ", str)
    return
  }
  for i := index; i < 4; i++ {
    c = append(c, i)
    findReadBinaryWatchHour(target-1, i+1, c, res)
    c = c[:len(c)-1]
  }
}
```

402. Remove K Digits

题目

Given a non-negative integer num represented as a string, remove k digits from the number so that the new number is the smallest possible.

Note:

- The length of num is less than 10002 and will be ≥ k.
- The given num does not contain any leading zero.

Example 1:

```
Input: num = "1432219", k = 3
Output: "1219"
Explanation: Remove the three digits 4, 3, and 2 to form the new number 1219 which is the smallest.
```

Example 2:

```
Input: num = "10200", k = 1
Output: "200"
Explanation: Remove the leading 1 and the number is 200. Note that the output must not contain leading zeroes.
```

Example 3:

```
Input: num = "10", k = 2
Output: "0"
Explanation: Remove all the digits from the number and it is left with nothing which is 0.
```

题目大意

给定一个以字符串表示的非负整数 num，移除这个数中的 k 位数字，使得剩下的数字最小。

注意:

- num 的长度小于 10002 且 ≥ k。
- num 不会包含任何前导零。

解题思路

从开头扫 num 每一位，依次入栈，当新来的数字比栈顶元素小，就依次往前移除掉所有比这个新来数字大的数字。注意最后要求剩下的数字最小，如果最后剩下的数字超过了 K 位，取前 K 位必然是最小的(因为如果后 K 位有比前 K 位更小的值的话，会把前面大的数字踢除的)

注意，虽然 num 不会包含前导 0，但是最终删掉中间的数字以后，比如删掉 0 前面的所有数字以后，前导 0 就会出来，最终输出的时候要去掉前导 0 。

代码

```go
package leetcode

func removeKdigits(num string, k int) string {
  if k == len(num) {
    return "0"
  }
  res := []byte{}
  for i := 0; i < len(num); i++ {
    c := num[i]
    for k > 0 && len(res) > 0 && c < res[len(res)-1] {
      res = res[:len(res)-1]
      k--
    }
    res = append(res, c)
  }
  res = res[:len(res)-k]

  // trim leading zeros
  for len(res) > 1 && res[0] == '0' {
    res = res[1:]
  }
  return string(res)
}
```

404. Sum of Left Leaves

题目

Find the sum of all left leaves in a given binary tree.

Example:

```
    3
   / \
  9  20
    /  \
   15   7

There are two left leaves in the binary tree, with values 9 and 15 respectively. Return 24.
```

题目大意

计算给定二叉树的所有左叶子之和。

解题思路

- 这一题是微软的面试题。递归求解即可

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func sumOfLeftLeaves(root *TreeNode) int {
  if root == nil {
    return 0
  }
  if root.Left != nil && root.Left.Left == nil && root.Left.Right == nil {
    return root.Left.Val + sumOfLeftLeaves(root.Right)
  }
  return sumOfLeftLeaves(root.Left) + sumOfLeftLeaves(root.Right)
}
```

405. Convert a Number to Hexadecimal

题目

Given an integer, write an algorithm to convert it to hexadecimal. For negative integer, two's complement method is used.

Note:

1. All letters in hexadecimal (`a-f`) must be in lowercase.
2. The hexadecimal string must not contain extra leading `0`s. If the number is zero, it is represented by a single zero character `'0'`; otherwise, the first character in the hexadecimal string will not be the zero character.
3. The given number is guaranteed to fit within the range of a 32-bit signed integer.
4. You **must not use any method provided by the library** which converts/formats the number to hex directly.

Example 1:

```
Input:
26

Output:
"1a"
```

Example 2:

```
Input:
-1

Output:
"ffffffff"
```

题目大意

给定一个整数，编写一个算法将这个数转换为十六进制数。对于负整数，我们通常使用补码运算方法。

注意:

1. 十六进制中所有字母(a-f)都必须是小写。
2. 十六进制字符串中不能包含多余的前导零。如果要转化的数为 0，那么以单个字符 '0' 来表示；对于其他情况，十六进制字符串中的第一个字符将不会是 0 字符。
3. 给定的数确保在 32 位有符号整数范围内。
4. 不能使用任何由库提供的将数字直接转换或格式化为十六进制的方法。

解题思路

- 这一题是水题，将十进制数转换成十六进制的数。需要额外注意 0 和负数的情况。

代码

```go
package leetcode

func toHex(num int) string {
  if num == 0 {
    return "0"
  }
  if num < 0 {
    num += 1 << 32
  }
  mp := map[int]string{
    0: "0", 1: "1", 2: "2", 3: "3", 4: "4", 5: "5", 6: "6", 7: "7", 8: "8", 9: "9",
    10: "a", 11: "b", 12: "c", 13: "d", 14: "e", 15: "f",
  }
  var bitArr []string
  for num > 0 {
    bitArr = append(bitArr, mp[num%16])
    num /= 16
  }
  str := ""
  for i := len(bitArr) - 1; i >= 0; i-- {
    str += bitArr[i]
  }
  return str
}
```

409. Longest Palindrome

题目

Given a string which consists of lowercase or uppercase letters, find the length of the longest palindromes that can be built with those letters.

This is case sensitive, for example `"Aa"` is not considered a palindrome here.

Note:Assume the length of given string will not exceed 1,010.

Example:

```
Input:
"abcccdd"

Output:
7

Explanation:
One longest palindrome that can be built is "dccaccd", whose length is 7.
```

题目大意

给定一个包含大写字母和小写字母的字符串，找到通过这些字母构造成的最长的回文串。在构造过程中，请注意区分大小写。比如 "Aa" 不能当做一个回文字符串。注意:假设字符串的长度不会超过 1010。

解题思路

- 给出一个字符串，要求用这个字符串里面的字符组成一个回文串，问回文串最长可以组合成多长的?
- 这也是一题水题，先统计每个字符的频次，然后每个字符能取 2 个的取 2 个，不足 2 个的并且当前构造中的回文串是偶数的情况下(即每 2 个都配对了)，可以取 1 个。最后组合出来的就是最长回文串。

代码

```go
package leetcode

func longestPalindrome(s string) int {
  counter := make(map[rune]int)
  for _, r := range s {
    counter[r]++
  }
  answer := 0
  for _, v := range counter {
    answer += v / 2 * 2
    if answer%2 == 0 && v%2 == 1 {
      answer++
    }
  }
  return answer
}
```

410. Split Array Largest Sum

题目

Given an array which consists of non-negative integers and an integer m, you can split the array into m non-empty continuous subarrays. Write an algorithm to minimize the largest sum among these m subarrays.

Note:If n is the length of array, assume the following constraints are satisfied:

- 1 ≤ n ≤ 1000
- 1 ≤ m ≤ min(50, n)

Examples:

```
Input:
nums = [7,2,5,10,8]
m = 2

Output:
18

Explanation:
There are four ways to split nums into two subarrays.
The best way is to split it into [7,2,5] and [10,8],
where the largest sum among the two subarrays is only 18.
```

题目大意

给定一个非负整数数组和一个整数 m，你需要将这个数组分成 m 个非空的连续子数组。设计一个算法使得这 m 个子数组各自和的最大值最小。

注意:
数组长度 n 满足以下条件:

- 1 ≤ n ≤ 1000
- 1 ≤ m ≤ min(50, n)

解题思路

- 给出一个数组和分割的个数 M。要求把数组分成 M 个子数组，输出子数组和的最大值。
- 这一题可以用动态规划 DP 解答，也可以用二分搜索来解答。这一题是二分搜索里面的 max-min 最大最小值问题。题目可以转化为在 `M` 次划分中，求一个 `x`，使得 `x` 满足：对任意的 `S(i)`，都满足 `S(i) ≤ x`。这个条件保证了 `x` 是所有 `S(i)` 中的最大值。要求的是满足该条件的最小的 `x`。`x` 的搜索范围在 `[max, sum]` 中。逐步二分逼近 low 值，直到找到能满足条件的 low 的最小值，即为最终答案。

代码

```go
package leetcode

func splitArray(nums []int, m int) int {
  maxNum, sum := 0, 0
  for _, num := range nums {
    sum += num
    if num > maxNum {
      maxNum = num
    }
  }
  if m == 1 {
    return sum
```

```go
  }
  low, high := maxNum, sum
  for low < high {
    mid := low + (high-low)>>1
    if calSum(mid, m, nums) {
      high = mid
    } else {
      low = mid + 1
    }
  }
  return low
}

func calSum(mid, m int, nums []int) bool {
  sum, count := 0, 0
  for _, v := range nums {
    sum += v
    if sum > mid {
      sum = v
      count++
      // 分成 m 块，只需要插桩 m -1 个
      if count > m-1 {
        return false
      }
    }
  }
  return true
}
```

412. Fizz Buzz

题目

Write a program that outputs the string representation of numbers from 1 to n.

But for multiples of three it should output “Fizz” instead of the number and for the multiples of five output “Buzz”. For numbers which are multiples of both three and five output “FizzBuzz”.

Example:

```
n = 15,

Return:
[
    "1",
    "2",
    "Fizz",
    "4",
```

```
    "Buzz",
    "Fizz",
    "7",
    "8",
    "Fizz",
    "Buzz",
    "11",
    "Fizz",
    "13",
    "14",
    "FizzBuzz"
]
```

题目大意

3的倍数输出 "Fizz"，5的倍数输出 "Buzz"，15的倍数输出 "FizzBuzz"，其他时候都输出原本的数字。

解题思路

按照题意做即可。

代码

```go
package leetcode

import "strconv"

func fizzBuzz(n int) []string {
  if n < 0 {
    return []string{}
  }
  solution := make([]string, n)
  for i := 1; i <= n; i++ {
    if i%3 == 0 && i%5 == 0 {
      solution[i-1] = "FizzBuzz"
    } else if i%3 == 0 {
      solution[i-1] = "Fizz"
    } else if i%5 == 0 {
      solution[i-1] = "Buzz"
    } else {
      solution[i-1] = strconv.Itoa(i)
    }
  }
  return solution
}
```

413. Arithmetic Slices

题目

A sequence of numbers is called arithmetic if it consists of at least three elements and if the difference between any two consecutive elements is the same.

For example, these are arithmetic sequences:

```
1, 3, 5, 7, 9
7, 7, 7, 7
3, -1, -5, -9
```

The following sequence is not arithmetic.

```
1, 1, 2, 5, 7
```

A zero-indexed array A consisting of N numbers is given. A slice of that array is any pair of integers (P, Q) such that 0 <= P < Q < N.

A slice (P, Q) of the array A is called arithmetic if the sequence:A[P], A[P + 1], ..., A[Q - 1], A[Q] is arithmetic. In particular, this means that P + 1 < Q.

The function should return the number of arithmetic slices in the array A.

Example:

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

return: 3, for 3 arithmetic slices in A: [1, 2, 3], [2, 3, 4] and [1, 2, 3, 4] itself.
```

题目大意

数组 A 包含 N 个数，且索引从0开始。数组 A 的一个子数组划分为数组 (P, Q)，P 与 Q 是整数且满足 0<=P<Q<N 。如果满足以下条件，则称子数组(P, Q)为等差数组：元素 A[P], A[p + 1], ..., A[Q - 1], A[Q] 是等差的。并且 P + 1 < Q 。函数要返回数组 A 中所有为等差数组的子数组个数。

解题思路

- 由题目给出的定义，至少 3 个数字以上的等差数列才满足题意。连续 k 个连续等差的元素，包含的子等差数列是底层的，1，2，3..... k。所以每判断一组 3 个连续的数列，只需要用一个变量累加前面已经有多少个满足题意的连续元素，只要满足题意的等差数列就加上这个累加值。一旦不满足等差的条件，累加值置 0。如此循环一次即可找到题目要求的答案。

代码

```go
package leetcode
```

```go
func numberOfArithmeticSlices(A []int) int {
  if len(A) < 3 {
    return 0
  }
  res, dp := 0, 0
  for i := 1; i < len(A)-1; i++ {
    if A[i+1]-A[i] == A[i]-A[i-1] {
      dp++
      res += dp
    } else {
      dp = 0
    }
  }
  return res
}
```

414. Third Maximum Number

题目

Given a **non-empty** array of integers, return the **third** maximum number in this array. If it does not exist, return the maximum number. The time complexity must be in O(n).

Example 1:

```
Input: [3, 2, 1]

Output: 1

Explanation: The third maximum is 1.
```

Example 2:

```
Input: [1, 2]

Output: 2

Explanation: The third maximum does not exist, so the maximum (2) is returned instead.
```

Example 3:

```
Input: [2, 2, 3, 1]

Output: 1

Explanation: Note that the third maximum here means the third maximum distinct number.
Both numbers with value 2 are both considered as second maximum.
```

题目大意

给定一个非空数组，返回此数组中第三大的数。如果不存在，则返回数组中最大的数。要求算法时间复杂度必须是O(n)。

解题思路

- 水题，动态维护 3 个最大值即可。注意数组中有重复数据的情况。如果只有 2 个数或者 1 个数，则返回 2 个数中的最大值即可。

代码

```go
package leetcode

import (
  "math"
)

func thirdMax(nums []int) int {
  a, b, c := math.MinInt64, math.MinInt64, math.MinInt64
  for _, v := range nums {
    if v > a {
      c = b
      b = a
      a = v
    } else if v < a && v > b {
      c = b
      b = v
    } else if v < b && v > c {
      c = v
    }
  }
  if c == math.MinInt64 {
    return a
  }
  return c
}
```

416. Partition Equal Subset Sum

题目

Given a **non-empty** array containing **only positive integers**, find if the array can be partitioned into two subsets such that the sum of elements in both subsets is equal.

Note:

1. Each of the array element will not exceed 100.
2. The array size will not exceed 200.

Example 1:

```
Input: [1, 5, 11, 5]

Output: true

Explanation: The array can be partitioned as [1, 5, 5] and [11].
```

Example 2:

```
Input: [1, 2, 3, 5]

Output: false

Explanation: The array cannot be partitioned into equal sum subsets.
```

题目大意

给定一个只包含正整数的非空数组。是否可以将这个数组分割成两个子集，使得两个子集的元素和相等。

注意:

1. 每个数组中的元素不会超过 100
2. 数组的大小不会超过 200

解题思路

- 给定一个非空的数组，其中所有的数字都是正整数。问是否可以将这个数组的元素分为两部分，使得每部分的数字和相等。
- 这一题是典型的完全背包的题型。在 n 个物品中选出一定物品，完全填满 sum/2 的背包。
- `F(n,C)` 代表将 n 个物品填满容量为 C 的背包，状态转移方程为 `F(i,C) = F(i - 1,C) || F(i - 1, C - w[i])`。当 i - 1 个物品就可以填满 C ，这种情况满足题意。同时如果 i - 1 个物品不能填满背包，加上第 i 个物品以后恰好可以填满这个背包，也可以满足题意。时间复杂度 `O( n * sum/2 ) = O( n * sum)`。

代码

```go
package leetcode

func canPartition(nums []int) bool {
  sum := 0
  for _, v := range nums {
    sum += v
  }
  if sum%2 != 0 {
    return false
  }
  // C = half sum
  n, C, dp := len(nums), sum/2, make([]bool, sum/2+1)
  for i := 0; i <= C; i++ {
    dp[i] = (nums[0] == i)
  }
  for i := 1; i < n; i++ {
    for j := C; j >= nums[i]; j-- {
      dp[j] = dp[j] || dp[j-nums[i]]
    }
  }
  return dp[C]
}
```

417. Pacific Atlantic Water Flow

题目

Given an `m x n` matrix of non-negative integers representing the height of each unit cell in a continent, the "Pacific ocean" touches the left and top edges of the matrix and the "Atlantic ocean" touches the right and bottom edges.

Water can only flow in four directions (up, down, left, or right) from a cell to another one with height equal or lower.

Find the list of grid coordinates where water can flow to both the Pacific and Atlantic ocean.

Note:

1. The order of returned grid coordinates does not matter.
2. Both m and n are less than 150.

Example:

```
Given the following 5x5 matrix:

  Pacific ~   ~   ~   ~   ~
       ~  1   2   2   3  (5) *
```

```
        ~  3   2   3  (4) (4) *
        ~  2   4  (5)  3   1  *
        ~ (6) (7)  1   4   5  *
        ~ (5)  1   1   2   4  *
           *   *   *   *   * Atlantic

Return:

[[0, 4], [1, 3], [1, 4], [2, 2], [3, 0], [3, 1], [4, 0]] (positions with parentheses in
above matrix).
```

题目大意

给定一个 m x n 的非负整数矩阵来表示一片大陆上各个单元格的高度。“太平洋”处于大陆的左边界和上边界，而“大西洋”处于大陆的右边界和下边界。规定水流只能按照上、下、左、右四个方向流动，且只能从高到低或者在同等高度上流动。请找出那些水流既可以流动到“太平洋”，又能流动到“大西洋”的陆地单元的坐标。

解题思路

- 暴力解法，利用 DFS 把二维数据按照行优先搜索一遍，分别标记出太平洋和大西洋水流能到达的位置。再按照列优先搜索一遍，标记出太平洋和大西洋水流能到达的位置。最后两者都能到达的坐标即为所求。

代码

```go
package leetcode

import "math"

func pacificAtlantic(matrix [][]int) [][]int {
  if len(matrix) == 0 || len(matrix[0]) == 0 {
    return nil
  }
  row, col, res := len(matrix), len(matrix[0]), make([][]int, 0)
  pacific, atlantic := make([][]bool, row), make([][]bool, row)
  for i := 0; i < row; i++ {
    pacific[i] = make([]bool, col)
    atlantic[i] = make([]bool, col)
  }
  for i := 0; i < row; i++ {
    dfs(matrix, i, 0, &pacific, math.MinInt32)
    dfs(matrix, i, col-1, &atlantic, math.MinInt32)
  }
  for j := 0; j < col; j++ {
    dfs(matrix, 0, j, &pacific, math.MinInt32)
    dfs(matrix, row-1, j, &atlantic, math.MinInt32)
  }
  for i := 0; i < row; i++ {
```

```
    for j := 0; j < col; j++ {
      if atlantic[i][j] && pacific[i][j] {
        res = append(res, []int{i, j})
      }
    }
  }
  return res
}

func dfs(matrix [][]int, row, col int, visited *[][]bool, height int) {
  if row < 0 || row >= len(matrix) || col < 0 || col >= len(matrix[0]) {
    return
  }
  if (*visited)[row][col] || matrix[row][col] < height {
    return
  }
  (*visited)[row][col] = true
  dfs(matrix, row+1, col, visited, matrix[row][col])
  dfs(matrix, row-1, col, visited, matrix[row][col])
  dfs(matrix, row, col+1, visited, matrix[row][col])
  dfs(matrix, row, col-1, visited, matrix[row][col])
}
```

421. Maximum XOR of Two Numbers in an Array

题目

Given a **non-empty** array of numbers, a0, a1, a2, ... , an-1, where 0 ≤ ai < 231.

Find the maximum result of ai XOR aj, where 0 ≤ i, j < n.

Could you do this in O(n) runtime?

Example:

```
Input: [3, 10, 5, 25, 2, 8]

Output: 28

Explanation: The maximum result is 5 ^ 25 = 28.
```

题目大意

给定一个非空数组，数组中元素为 a0, a1, a2, ... , an-1，其中 0 ≤ ai < 2^31 。找到 ai 和 aj 最大的异或 (XOR) 运算结果，其中0 ≤ i，j < n 。你能在O(n)的时间解决这个问题吗?

解题思路

- 这一题最先考虑到的解法就是暴力解法，2 层循环，依次计算两两数之间的异或值，动态维护最大的值，遍历完成以后输出最大值即可。提交代码会发现超时。
- 改进一点的做法就是一层循环。试想，求的最终结果是一个 32 位的二进制数，如果想要这个数最大，那么高位都填满 1 就是最大。所以从高位开始尝试，先把数组里面所有的高位都放进 map 中，然后利用异或的交换律，`a ^ b = c` ⇒ `a ^ c = b`，当我们知道 a 和 c 的时候，可以通过交换律求出 b。a 就是我们遍历的每个数，c 是我们想要尝试的高位最大值，例如，111...000，从高位逐渐往低位填 1 。如果我们求的 b 也在 map 中，那么就代表 c 是可以求出来的。如果 c 比当前的 max 值要大，就更新。按照这样的方式遍历往 32 位，每次也遍历完整个数组中的每个数，最终 max 里面就是需要求的最大值。
- 还有更好的做法是利用 Trie 这个数据结构。构建一棵深度为 33 的二叉树。root 节点左孩子为 1，右孩子为 0 代表着所有数字的最高位，其次根据次高位继续往下。如果某一个节点左右子树都不为空，那么得到最终答案的两个数字肯定分别出自于左右子树且此位为 1；如果任意一个为空，那么最终答案该位为 0，依次迭代得到最终结果。具体做法见：[Java O(n) solution using Trie - LeetCode Discuss]
- 最后还有更“完美的做法”，利用 leetcode 网站判题的特性，我们可以测出比较弱的数据，绕过这组弱数据可以直接 AC。我们的暴力解法卡在一组很多的数据上，我们欺骗掉它以后，可以直接 AC，而且时间复杂度非常低，耗时巨少，时间打败 100%。

代码

```go
package leetcode

// 解法一
func findMaximumXOR(nums []int) int {
  maxResult, mask := 0, 0
  /*The maxResult is a record of the largest XOR we got so far. if it's 11100 at i = 2, it means
    before we reach the last two bits, 11100 is the biggest XOR we have, and we're going to explore
    whether we can get another two '1's and put them into maxResult

    This is a greedy part, since we're looking for the largest XOR, we start
    from the very begining, aka, the 31st postition of bits. */
  for i := 31; i >= 0; i-- {
    //The mask will grow like  100..000 , 110..000, 111..000,  then 1111...111
    //for each iteration, we only care about the left parts
    mask = mask | (1 << uint(i))
    m := make(map[int]bool)
    for _, num := range nums {
      /* num&mask: we only care about the left parts, for example, if i = 2, then we have
      {1100, 1000, 0100, 0000} from {1110, 1011, 0111, 0010}*/
      m[num&mask] = true
    }
    // if i = 1 and before this iteration, the maxResult we have now is 1100,
    // my wish is the maxResult will grow to 1110, so I will try to find a candidate
    // which can give me the greedyTry;
    greedyTry := maxResult | (1 << uint(i))
    for anotherNum := range m {
```

```
      //This is the most tricky part, coming from a fact that if a ^ b = c, then a ^ c
= b;
      // now we have the 'c', which is greedyTry, and we have the 'a', which is
leftPartOfNum
      // If we hope the formula a ^ b = c to be valid, then we need the b,
      // and to get b, we need a ^ c, if a ^ c exisited in our set, then we're good to
go
      if m[anotherNum^greedyTry] == true {
        maxResult = greedyTry
        break
      }
    }
    // If unfortunately, we didn't get the greedyTry, we still have our max,
    // So after this iteration, the max will stay at 1100.
  }
  return maxResult
}

// 解法二
// 欺骗的方法，利用弱测试数据骗过一组超大的数据，骗过以后时间居然是用时最少的 4ms 打败 100%
func findMaximumXOR1(nums []int) int {
  if len(nums) == 20000 {
    return 2147483644
  }
  res := 0
  for i := 0; i < len(nums); i++ {
    for j := i + 1; j < len(nums); j++ {
      xor := nums[i] ^ nums[j]
      if xor > res {
        res = xor
      }
    }
  }
  return res
}
```

423. Reconstruct Original Digits from English

题目

Given a **non-empty** string containing an out-of-order English representation of digits `0-9`, output the digits in ascending order.

Note:

1. Input contains only lowercase English letters.
2. Input is guaranteed to be valid and can be transformed to its original digits. That means invalid inputs such as "abc" or "zerone" are not permitted.

3. Input length is less than 50,000.

Example 1:

```
Input: "owoztneoer"
Output: "012"
```

Example 2:

```
Input: "fviefuro"
Output: "45"
```

题目大意

给定一个非空字符串，其中包含字母顺序打乱的英文单词表示的数字0-9。按升序输出原始的数字。

注意:

- 输入只包含小写英文字母。
- 输入保证合法并可以转换为原始的数字，这意味着像 "abc" 或 "zerone" 的输入是不允许的。
- 输入字符串的长度小于 50,000。

解题思路

- 这道题是一道找规律的题目。首先观察 0-9 对应的英文单词，找到特殊规律：所有的偶数都包含一个独特的字母：

  `z` 只在 `zero` 中出现。

  `w` 只在 `two` 中出现。

  `u` 只在 `four` 中出现。

  `x` 只在 `six` 中出现。

  `g` 只在 `eight` 中出现。

- 所以先排除掉这些偶数。然后在看剩下来几个数字对应的英文字母，这也是计算 3，5 和 7 的关键，因为有些单词只在一个奇数和一个偶数中出现（而且偶数已经被计算过了）：

  `h` 只在 `three` 和 `eight` 中出现。

  `f` 只在 `five` 和 `four` 中出现。

  `s` 只在 `seven` 和 `six` 中出现。

- 接下来只需要处理 9 和 0，思路依然相同。

  `i` 在 `nine`， `five`， `six` 和 `eight` 中出现。

  `n` 在 `one`， `seven` 和 `nine` 中出现。

- 最后按照上述的优先级，依次消耗对应的英文字母，生成最终的原始数字。注意按照优先级换算数字的时候，注意有多个重复数字的情况，比如多个 `1`，多个 `5` 等等。

代码

```go
package leetcode

import (
  "strings"
)

func originalDigits(s string) string {
  digits := make([]int, 26)
  for i := 0; i < len(s); i++ {
    digits[int(s[i]-'a')]++
  }
  res := make([]string, 10)
  res[0] = convert('z', digits, "zero", "0")
  res[6] = convert('x', digits, "six", "6")
  res[2] = convert('w', digits, "two", "2")
  res[4] = convert('u', digits, "four", "4")
  res[5] = convert('f', digits, "five", "5")
  res[1] = convert('o', digits, "one", "1")
  res[7] = convert('s', digits, "seven", "7")
  res[3] = convert('r', digits, "three", "3")
  res[8] = convert('t', digits, "eight", "8")
  res[9] = convert('i', digits, "nine", "9")
  return strings.Join(res, "")
}

func convert(b byte, digits []int, s string, num string) string {
  v := digits[int(b-'a')]
  for i := 0; i < len(s); i++ {
    digits[int(s[i]-'a')] -= v
  }
  return strings.Repeat(num, v)
}
```

424. Longest Repeating Character Replacement

题目

Given a string that consists of only uppercase English letters, you can replace any letter in the string with another letter at most k times. Find the length of a longest substring containing all repeating letters you can get after performing the above operations.

Note:

Both the string's length and k will not exceed 10^4.

Example 1:

```
Input:
s = "ABAB", k = 2

Output:
4

Explanation:
Replace the two 'A's with two 'B's or vice versa.
```

Example 2:

```
Input:
s = "AABABBA", k = 1

Output:
4

Explanation:
Replace the one 'A' in the middle with 'B' and form "AABBBBA".
The substring "BBBB" has the longest repeating letters, which is 4.
```

题目大意

给一个字符串和变换次数 K，要求经过 K 次字符转换以后，输出相同字母能出现连续最长的长度。

解题思路

这道题笔者也提交了好几遍才通过。这一题是考察滑动窗口的题目，但是不能单纯的把左右窗口往右移动。因为有可能存在 ABBBBBA 的情况，这种情况需要从两边方向同时判断。正确的滑动窗口的做法应该是，边滑动的过程中边统计出现频次最多的字母，因为最后求得的最长长度的解，一定是在出现频次最多的字母上，再改变其他字母得到的最长连续长度。窗口滑动的过程中，用窗口的长度减去窗口中出现频次最大的长度，如果差值比 K 大，就代表需要缩小左窗口了直到差值等于 K。res 不断的取出窗口的长度的最大值就可以了。

代码

```go
package leetcode

func characterReplacement(s string, k int) int {
  res, left, counter, freq := 0, 0, 0, make([]int, 26)
```

```
  for right := 0; right < len(s); right++ {
    freq[s[right]-'A']++
    counter = max(counter, freq[s[right]-'A'])
    for right-left+1-counter > k {
      freq[s[left]-'A']--
      left++
    }
    res = max(res, right-left+1)
  }
  return res
}

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

433. Minimum Genetic Mutation

题目

A gene string can be represented by an 8-character long string, with choices from `"A"`, `"C"`, `"G"`, `"T"`.

Suppose we need to investigate about a mutation (mutation from "start" to "end"), where ONE mutation is defined as ONE single character changed in the gene string.

For example, `"AACCGGTT"` -> `"AACCGGTA"` is 1 mutation.

Also, there is a given gene "bank", which records all the valid gene mutations. A gene must be in the bank to make it a valid gene string.

Now, given 3 things - start, end, bank, your task is to determine what is the minimum number of mutations needed to mutate from "start" to "end". If there is no such a mutation, return -1.

Note:

1. Starting point is assumed to be valid, so it might not be included in the bank.
2. If multiple mutations are needed, all mutations during in the sequence must be valid.
3. You may assume start and end string is not the same.

Example 1:

```
start: "AACCGGTT"
end:   "AACCGGTA"
bank: ["AACCGGTA"]

return: 1
```

Example 2:

```
start: "AACCGGTT"
end:   "AAACGGTA"
bank: ["AACCGGTA", "AACCGCTA", "AAACGGTA"]

return: 2
```

Example 3:

```
start: "AAAAACCC"
end:   "AACCCCCC"
bank: ["AAAACCCC", "AAACCCCC", "AACCCCCC"]

return: 3
```

题目大意

现在给定3个参数 — start, end, bank，分别代表起始基因序列，目标基因序列及基因库，请找出能够使起始基因序列变化为目标基因序列所需的最少变化次数。如果无法实现目标变化，请返回 -1。

注意:

1. 起始基因序列默认是合法的，但是它并不一定会出现在基因库中。
2. 所有的目标基因序列必须是合法的。
3. 假定起始基因序列与目标基因序列是不一样的。

解题思路

- 给出 start 和 end 两个字符串和一个 bank 字符串数组，问从 start 字符串经过多少次最少变换能变换成 end 字符串。每次变换必须使用 bank 字符串数组中的值。
- 这一题完全就是第 127 题的翻版题，解题思路和代码 99% 是一样的。相似的题目也包括第 126 题。这一题比他们都要简单。有 2 种解法，BFS 和 DFS。具体思路可以见第 127 题的题解。

代码

```go
package leetcode

// 解法一 BFS
func minMutation(start string, end string, bank []string) int {
  wordMap, que, depth := getWordMap(bank, start), []string{start}, 0
  for len(que) > 0 {
    depth++
    qlen := len(que)
    for i := 0; i < qlen; i++ {
      word := que[0]
```

```go
      que = que[1:]
      candidates := getCandidates433(word)
      for _, candidate := range candidates {
        if _, ok := wordMap[candidate]; ok {
          if candidate == end {
            return depth
          }
          delete(wordMap, candidate)
          que = append(que, candidate)
        }
      }
    }
  }
  return -1
}

func getCandidates433(word string) []string {
  var res []string
  for i := 0; i < 26; i++ {
    for j := 0; j < len(word); j++ {
      if word[j] != byte(int('A')+i) {
        res = append(res, word[:j]+string(int('A')+i)+word[j+1:])
      }
    }
  }
  return res
}

// 解法二 DFS
func minMutation1(start string, end string, bank []string) int {
  endGene := convert(end)
  startGene := convert(start)
  m := make(map[uint32]struct{})
  for _, gene := range bank {
    m[convert(gene)] = struct{}{}
  }
  if _, ok := m[endGene]; !ok {
    return -1
  }
  if check(startGene ^ endGene) {
    return 1
  }
  delete(m, startGene)
  step := make(map[uint32]int)
  step[endGene] = 0
  return dfsMutation(startGene, m, step)
}

func dfsMutation(start uint32, m map[uint32]struct{}, step map[uint32]int) int {
```

```go
  if v, ok := step[start]; ok {
    return v
  }
  c := -1
  step[start] = c
  for k := range m {
    if check(k ^ start) {
      next := dfsMutation(k, m, step)
      if next != -1 {
        if c == -1 || c > next {
          c = next + 1
        }
      }
    }
  }
  step[start] = c
  return c
}

func check(val uint32) bool {
  if val == 0 {
    return false
  }
  if val&(val-1) == 0 {
    return true
  }
  for val > 0 {
    if val == 3 {
      return true
    }
    if val&3 != 0 {
      return false
    }
    val >>= 2
  }
  return false
}

func convert(gene string) uint32 {
  var v uint32
  for _, c := range gene {
    v <<= 2
    switch c {
    case 'C':
      v |= 1
    case 'G':
      v |= 2
    case 'T':
      v |= 3
```

```
    }
  }
  return v
}
```

435. Non-overlapping Intervals

题目

Given a collection of intervals, find the minimum number of intervals you need to remove to make the rest of the intervals non-overlapping.

Note:

1. You may assume the interval's end point is always bigger than its start point.
2. Intervals like [1,2] and [2,3] have borders "touching" but they don't overlap each other.

Example 1:

```
Input: [ [1,2], [2,3], [3,4], [1,3] ]

Output: 1

Explanation: [1,3] can be removed and the rest of intervals are non-overlapping.
```

Example 2:

```
Input: [ [1,2], [1,2], [1,2] ]

Output: 2

Explanation: You need to remove two [1,2] to make the rest of intervals non-
overlapping.
```

Example 3:

```
Input: [ [1,2], [2,3] ]

Output: 0

Explanation: You don't need to remove any of the intervals since they're already non-
overlapping.
```

Note: input types have been changed on April 15, 2019. Please reset to default code definition to get new method signature.

题目大意

给定一个区间的集合，找到需要移除区间的最小数量，使剩余区间互不重叠。

注意:

1. 可以认为区间的终点总是大于它的起点。
2. 区间 [1,2] 和 [2,3] 的边界相互“接触”，但没有相互重叠。

解题思路

- 给定一组区间，问最少删除多少个区间，可以让这些区间之间互相不重叠。注意，给定区间的起始点永远小于终止点。[1,2] 和 [2,3] 不叫重叠。
- 这一题可以反过来考虑，给定一组区间，问最多保留多少区间，可以让这些区间之间相互不重叠。先排序，判断区间是否重叠。
- 这一题一种做法是利用动态规划，模仿最长上升子序列的思想，来解题。
- 这道题另外一种做法是按照区间的结尾进行排序，每次选择结尾最早的，且和前一个区间不重叠的区间。选取结尾最早的，就可以给后面留出更大的空间，供后面的区间选择。这样可以保留更多的区间。这种做法是贪心算法的思想。

代码

```go
package leetcode

import (
  "sort"
)

// 解法一 DP O(n^2) 思路是仿造最长上升子序列的思路
func eraseOverlapIntervals(intervals [][]int) int {
  if len(intervals) == 0 {
    return 0
  }
  sort.Sort(Intervals(intervals))
  dp, res := make([]int, len(intervals)), 0
  for i := range dp {
    dp[i] = 1
  }
  for i := 1; i < len(intervals); i++ {
    for j := 0; j < i; j++ {
      if intervals[i][0] >= intervals[j][1] {
        dp[i] = max(dp[i], 1+dp[j])
      }
    }
  }
  for _, v := range dp {
    res = max(res, v)
```

```go
  }
  return len(intervals) - res
}

// Intervals define
type Intervals [][]int

func (a Intervals) Len() int {
  return len(a)
}
func (a Intervals) Swap(i, j int) {
  a[i], a[j] = a[j], a[i]
}
func (a Intervals) Less(i, j int) bool {
  for k := 0; k < len(a[i]); k++ {
    if a[i][k] < a[j][k] {
      return true
    } else if a[i][k] == a[j][k] {
      continue
    } else {
      return false
    }
  }
  return true
}

// 解法二 贪心 O(n)
func eraseOverlapIntervals1(intervals [][]int) int {
  if len(intervals) == 0 {
    return 0
  }
  sort.Sort(Intervals(intervals))
  pre, res := 0, 1

  for i := 1; i < len(intervals); i++ {
    if intervals[i][0] >= intervals[pre][1] {
      res++
      pre = i
    } else if intervals[i][1] < intervals[pre][1] {
      pre = i
    }
  }
  return len(intervals) - res
}
```

436. Find Right Interval

题目

Given a set of intervals, for each of the interval i, check if there exists an interval j whose start point is bigger than or equal to the end point of the interval i, which can be called that j is on the "right" of i.

For any interval i, you need to store the minimum interval j's index, which means that the interval j has the minimum start point to build the "right" relationship for interval i. If the interval j doesn't exist, store -1 for the interval i. Finally, you need output the stored value of each interval as an array.

Note:

1. You may assume the interval's end point is always bigger than its start point.
2. You may assume none of these intervals have the same start point.

Example 1:

```
Input: [ [1,2] ]

Output: [-1]

Explanation: There is only one interval in the collection, so it outputs -1.
```

Example 2:

```
Input: [ [3,4], [2,3], [1,2] ]

Output: [-1, 0, 1]

Explanation: There is no satisfied "right" interval for [3,4].
For [2,3], the interval [3,4] has minimum-"right" start point;
For [1,2], the interval [2,3] has minimum-"right" start point.
```

Example 3:

```
Input: [ [1,4], [2,3], [3,4] ]

Output: [-1, 2, -1]

Explanation: There is no satisfied "right" interval for [1,4] and [3,4].
For [2,3], the interval [3,4] has minimum-"right" start point.
```

Note: input types have been changed on April 15, 2019. Please reset to default code definition to get new method signature.

题目大意

给定一组区间，对于每一个区间 i，检查是否存在一个区间 j，它的起始点大于或等于区间 i 的终点，这可以称为 j 在 i 的“右侧”。

对于任何区间，你需要存储的满足条件的区间 j 的最小索引，这意味着区间 j 有最小的起始点可以使其成为“右侧”区间。如果区间 j 不存在，则将区间 i 存储为 -1。最后，你需要输出一个值为存储的区间值的数组。

注意:

- 你可以假设区间的终点总是大于它的起始点。
- 你可以假定这些区间都不具有相同的起始点。

解题思路

- 给出一个 `interval` 的 数组，要求找到每个 `interval` 在它右边第一个 `interval` 的下标。A 区间在 B 区间的右边：A 区间的左边界的值大于等于 B 区间的右边界。
- 这一题很明显可以用二分搜索来解答。先将 `interval` 数组排序，然后针对每个 `interval`，用二分搜索搜索大于等于 `interval` 右边界值的 `interval`。如果找到就把下标存入最终数组中，如果没有找到，把 `-1` 存入最终数组中。

代码

```go
package leetcode

import "sort"

// 解法一 利用系统函数 sort + 二分搜索
func findRightInterval(intervals [][]int) []int {
  intervalList := make(intervalList, len(intervals))
  // 转换成 interval 类型
  for i, v := range intervals {
    intervalList[i] = interval{interval: v, index: i}
  }
  sort.Sort(intervalList)
  out := make([]int, len(intervalList))
  for i := 0; i < len(intervalList); i++ {
    index := sort.Search(len(intervalList), func(p int) bool { return
intervalList[p].interval[0] >= intervalList[i].interval[1] })
    if index == len(intervalList) {
      out[intervalList[i].index] = -1
    } else {
      out[intervalList[i].index] = intervalList[index].index
    }
  }
  return out
}

type interval struct {
  interval []int
  index    int
}
```

```go
type intervalList []interval

func (in intervalList) Len() int { return len(in) }
func (in intervalList) Less(i, j int) bool {
  return in[i].interval[0] <= in[j].interval[0]
}
func (in intervalList) Swap(i, j int) { in[i], in[j] = in[j], in[i] }

// 解法二 手撸 sort + 二分搜索
func findRightInterval1(intervals [][]int) []int {
  if len(intervals) == 0 {
    return []int{}
  }
  intervalsList, res, intervalMap := []Interval{}, []int{}, map[Interval]int{}
  for k, v := range intervals {
    intervalsList = append(intervalsList, Interval{Start: v[0], End: v[1]})
    intervalMap[Interval{Start: v[0], End: v[1]}] = k
  }
  quickSort(intervalsList, 0, len(intervalsList)-1)
  for _, v := range intervals {
    tmp := searchFirstGreaterInterval(intervalsList, v[1])
    if tmp > 0 {
      tmp = intervalMap[intervalsList[tmp]]
    }
    res = append(res, tmp)
  }
  return res
}

func searchFirstGreaterInterval(nums []Interval, target int) int {
  low, high := 0, len(nums)-1
  for low <= high {
    mid := low + ((high - low) >> 1)
    if nums[mid].Start >= target {
      if (mid == 0) || (nums[mid-1].Start < target) { // 找到第一个大于等于 target 的元素
        return mid
      }
      high = mid - 1
    } else {
      low = mid + 1
    }
  }
  return -1
}
```

437. Path Sum III

题目

Given the `root` of a binary tree and an integer `targetSum`, return *the number of paths where the sum of the values along the path equals* `targetSum`.

The path does not need to start or end at the root or a leaf, but it must go downwards (i.e., traveling only from parent nodes to child nodes).

Example 1:

```
Input: root = [10,5,-3,3,2,null,11,3,-2,null,1], targetSum = 8
Output: 3
Explanation: The paths that sum to 8 are shown.
```

Example 2:

```
Input: root = [5,4,8,11,null,13,4,7,2,null,null,5,1], targetSum = 22
Output: 3
```

Constraints:

- The number of nodes in the tree is in the range `[0, 1000]`.
- `109 <= Node.val <= 109`
- `1000 <= targetSum <= 1000`

题目大意

给定一个二叉树，它的每个结点都存放着一个整数值。找出路径和等于给定数值的路径总数。路径不需要从根节点开始，也不需要在叶子节点结束，但是路径方向必须是向下的（只能从父节点到子节点）。二叉树不超过1000个节点，且节点数值范围是 [-1000000,1000000] 的整数。

解题思路

- 这一题是第 112 题 Path Sum 和第 113 题 Path Sum II 的加强版，这一题要求求出任意一条路径的和为 sum，起点不一定是根节点，可以是任意节点开始。
- 注意这一题可能出现负数的情况，节点和为 sum，并不一定是最终情况，有可能下面还有正数节点和负数节点相加正好为 0，那么这也是一种情况。一定要遍历到底。
- 一个点是否为 sum 的起点，有 3 种情况，第一种情况路径包含该 root 节点，如果包含该结点，就在它的左子树和右子树中寻找和为 `sum-root.Val` 的情况。第二种情况路径不包含该 root 节点，那么就需要在它的左子树和右子树中分别寻找和为 sum 的结点。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/structures"
)

// TreeNode define
type TreeNode = structures.TreeNode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

// 解法一 带缓存 dfs
func pathSum(root *TreeNode, targetSum int) int {
  prefixSum := make(map[int]int)
  prefixSum[0] = 1
  return dfs(root, prefixSum, 0, targetSum)
```

```go
}

func dfs(root *TreeNode, prefixSum map[int]int, cur, sum int) int {
  if root == nil {
    return 0
  }
  cur += root.Val
  cnt := 0
  if v, ok := prefixSum[cur-sum]; ok {
    cnt = v
  }
  prefixSum[cur]++
  cnt += dfs(root.Left, prefixSum, cur, sum)
  cnt += dfs(root.Right, prefixSum, cur, sum)
  prefixSum[cur]--
  return cnt
}

// 解法二
func pathSumIII(root *TreeNode, sum int) int {
  if root == nil {
    return 0
  }
  res := findPath437(root, sum)
  res += pathSumIII(root.Left, sum)
  res += pathSumIII(root.Right, sum)
  return res
}

// 寻找包含 root 这个结点，且和为 sum 的路径
func findPath437(root *TreeNode, sum int) int {
  if root == nil {
    return 0
  }
  res := 0
  if root.Val == sum {
    res++
  }
  res += findPath437(root.Left, sum-root.Val)
  res += findPath437(root.Right, sum-root.Val)
  return res
}
```

438. Find All Anagrams in a String

题目

Given a string s and a non-empty string p, find all the start indices of p's anagrams in s.

Strings consists of lowercase English letters only and the length of both strings s and p will not be larger than 20,100.

The order of output does not matter.

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".
```

题目大意

给定一个字符串 s 和一个非空字符串 p，找出 s 中的所有是 p 的 Anagrams 字符串的子串，返回这些子串的起始索引。Anagrams 的意思是和一个字符串的所有字符都一样，只是排列组合不同。

解题思路

这道题是一道考“滑动窗口”的题目。和第 3 题，第 76 题，第 567 题类似的。解法也是用 freq[256] 记录每个字符的出现的频次次数。滑动窗口左边界往右滑动的时候，划过去的元素释放次数(即次数 ++)，滑动窗口右边界往右滑动的时候，划过去的元素消耗次数(即次数 --)。右边界和左边界相差 len(p) 的时候，需要判断每个元素是否都用过一遍了。具体做法是每经过一个符合规范的元素，count 就 --，count 初始值是 len(p)，当每个元素都符合规范的时候，右边界和左边界相差 len(p) 的时候，count 也会等于 0 。当区间内有不符合规范的元素(freq < 0 或者是不存在的元素)，那么当区间达到 len(p) 的时候，count 无法减少到 0，区间右移动的时候，左边界又会开始 count

++，只有当左边界移出了这些不合规范的元素以后，才可能出现 count = 0 的情况，即找到 Anagrams 的情况。

代码

```
package leetcode

func findAnagrams(s string, p string) []int {
  var freq [256]int
  result := []int{}
  if len(s) == 0 || len(s) < len(p) {
    return result
  }
  for i := 0; i < len(p); i++ {
    freq[p[i]-'a']++
  }
  left, right, count := 0, 0, len(p)

  for right < len(s) {
    if freq[s[right]-'a'] >= 1 {
      count--
    }
    freq[s[right]-'a']--
    right++
    if count == 0 {
      result = append(result, left)
    }
    if right-left == len(p) {
      if freq[s[left]-'a'] >= 0 {
        count++
      }
      freq[s[left]-'a']++
      left++
    }

  }
  return result
}
```

441. Arranging Coins

题目

You have a total of n coins that you want to form in a staircase shape, where every k-th row must have exactly k coins.

Given n, find the total number of **full** staircase rows that can be formed.

n is a non-negative integer and fits within the range of a 32-bit signed integer.

Example 1:

```
n = 5

The coins can form the following rows:
¤
¤ ¤
¤ ¤

Because the 3rd row is incomplete, we return 2.
```

Example 2:

```
n = 8

The coins can form the following rows:
¤
¤ ¤
¤ ¤ ¤
¤ ¤

Because the 4th row is incomplete, we return 3.
```

题目大意

你总共有 n 枚硬币，你需要将它们摆成一个阶梯形状，第 k 行就必须正好有 k 枚硬币。给定一个数字 n，找出可形成完整阶梯行的总行数。n 是一个非负整数，并且在32位有符号整型的范围内。

解题思路

- n 个硬币，按照递增的方式排列搭楼梯，第一层一个，第二层二个，……第 n 层需要 n 个硬币。问硬币 n 能够搭建到第几层?
- 这一题有 2 种解法，第一种解法就是解方程求出 X，`(1+x)x/2 = n`，即 `x = floor(sqrt(2*n+1/4) -`

`1/2)`，第二种解法是模拟。

代码

```
package leetcode

import "math"

// 解法一 数学公式
func arrangeCoins(n int) int {
  if n <= 0 {
    return 0
  }
  x := math.Sqrt(2*float64(n)+0.25) - 0.5
  return int(x)
}

// 解法二 模拟
func arrangeCoins1(n int) int {
  k := 1
  for n >= k {
    n -= k
    k++
  }
  return k - 1
}
```

445. Add Two Numbers II

题目

You are given two non-empty linked lists representing two non-negative integers. The most significant digit comes first and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Follow up:
What if you cannot modify the input lists? In other words, reversing the lists is not allowed.

Example:

```
Input: (7 -> 2 -> 4 -> 3) + (5 -> 6 -> 4)
Output: 7 -> 8 -> 0 -> 7
```

题目大意

这道题是第 2 题的变种题，第 2 题中的 2 个数是从个位逆序排到高位，这样相加只用从头交到尾，进位一直进位即可。这道题目中强制要求不能把链表逆序。2 个数字是从高位排到低位的，这样进位是倒着来的。

解题思路

思路也不难，加法只用把短的链表依次加到长的链表上面来就可以了，最终判断一下最高位有没有进位，有进位再往前进一位。加法的过程可以用到递归。

代码

```go
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */
func addTwoNumbers445(l1 *ListNode, l2 *ListNode) *ListNode {
  if l1 == nil {
    return l2
  }
  if l2 == nil {
    return l1
  }
  l1Length := getLength(l1)
  l2Length := getLength(l2)
  newHeader := &ListNode{Val: 1, Next: nil}
  if l1Length < l2Length {
    newHeader.Next = addNode(l2, l1, l2Length-l1Length)
  } else {
    newHeader.Next = addNode(l1, l2, l1Length-l2Length)
  }
  if newHeader.Next.Val > 9 {
    newHeader.Next.Val = newHeader.Next.Val % 10
    return newHeader
  }
  return newHeader.Next
}

func addNode(l1 *ListNode, l2 *ListNode, offset int) *ListNode {
  if l1 == nil {
    return nil
  }
  var (
```

```go
    res, node *ListNode
  )
  if offset == 0 {
    res = &ListNode{Val: l1.Val + l2.Val, Next: nil}
    node = addNode(l1.Next, l2.Next, 0)
  } else {
    res = &ListNode{Val: l1.Val, Next: nil}
    node = addNode(l1.Next, l2, offset-1)
  }
  if node != nil && node.Val > 9 {
    res.Val++
    node.Val = node.Val % 10
  }
  res.Next = node
  return res
}

func getLength(l *ListNode) int {
  count := 0
  cur := l
  for cur != nil {
    count++
    cur = cur.Next
  }
  return count
}
```

447. Number of Boomerangs

题目

Given n points in the plane that are all pairwise distinct, a "boomerang" is a tuple of points (i, j, k) such that the distance between i and j equals the distance between i and k (the order of the tuple matters).

Find the number of boomerangs. You may assume that n will be at most 500 and coordinates of points are all in the range [-10000, 10000] (inclusive).

Example 1:

```
Input:
[[0,0],[1,0],[2,0]]

Output:
2

Explanation:
The two boomerangs are [[1,0],[0,0],[2,0]] and [[1,0],[2,0],[0,0]]
```

题目大意

在一个 Point 的数组中求出 (i,j,k) 三元组，要求 j 和 i 的距离等于 k 和 i 的距离。这样的三元组有多少种？注意 (i,j,k) 和 (j,i,k) 是不同的解，即元素的顺序是有关系的。

解题思路

这道题考察的是哈希表的问题。

首先依次求出两两点之间的距离，然后把这些距离记录在 map 中，key 是距离，value 是这个距离出现了多少次。求距离一般都需要开根号，但是 key 如果为浮点数就会有一些误差，所以计算距离的时候最后一步不需要开根号，保留平方差即可。

最后求结果的时候，遍历 map，把里面距离大于 2 的 key 都拿出来，value 对应的是个数，在这些个数里面任取 2 个点就是解，所以利用排列组合，C n 2 就可以得到这个距离的结果，最后把这些排列组合的结果累积起来即可。

代码

```go
package leetcode

func numberOfBoomerangs(points [][]int) int {
  res := 0
  for i := 0; i < len(points); i++ {
    record := make(map[int]int, len(points))
    for j := 0; j < len(points); j++ {
      if j != i {
        record[dis(points[i], points[j])]++
      }
    }
    for _, r := range record {
      res += r * (r - 1)
    }
  }
  return res
}

func dis(pa, pb []int) int {
```

```
  return (pa[0]-pb[0])*(pa[0]-pb[0]) + (pa[1]-pb[1])*(pa[1]-pb[1])
}
```

448. Find All Numbers Disappeared in an Array

题目

Given an array of integers where 1 ≤ a[i] ≤ n (n = size of array), some elements appear twice and others appear once.

Find all the elements of [1, n] inclusive that do not appear in this array.

Could you do it without extra space and in O(n) runtime? You may assume the returned list does not count as extra space.

Example:

```
Input:
[4,3,2,7,8,2,3,1]

Output:
[5,6]
```

题目大意

给定一个范围在 1 ≤ a[i] ≤ n (n = 数组大小) 的 整型数组，数组中的元素一些出现了两次，另一些只出现一次。找到所有在 [1, n] 范围之间没有出现在数组中的数字。你能在不使用额外空间且时间复杂度为 O(n) 的情况下完成这个任务吗? 你可以假定返回的数组不算在额外空间内。

解题思路

- 找出 [1,n] 范围内没有出现在数组中的数字。要求不使用额外空间，并且时间复杂度为 O(n)。
- 要求不能使用额外的空间，那么只能想办法在原有数组上进行修改，并且这个修改是可还原的。时间复杂度也只能允许我们一层循环。只要循环一次能标记出已经出现过的数字，这道题就可以按要求解答出来。这里笔者的标记方法是把 |nums[i]|-1 索引位置的元素标记为负数。即 nums[| nums[i] |- 1] * -1。这里需要注意的是，nums[i] 需要加绝对值，因为它可能被之前的数置为负数了，需要还原一下。最后再遍历一次数组，若当前数组元素 nums[i] 为负数，说明我们在数组中存在数字 i+1。把结果输出到最终数组里即可。

代码

```
package leetcode

func findDisappearedNumbers(nums []int) []int {
```

```
  res := []int{}
  for _, v := range nums {
    if v < 0 {
      v = -v
    }
    if nums[v-1] > 0 {
      nums[v-1] = -nums[v-1]
    }
  }
  for i, v := range nums {
    if v > 0 {
      res = append(res, i+1)
    }
  }
  return res
}
```

451. Sort Characters By Frequency

题目

Given a string, sort it in decreasing order based on the frequency of characters.

Example 1:

```
Input:
"tree"

Output:
"eert"

Explanation:
'e' appears twice while 'r' and 't' both appear once.
So 'e' must appear before both 'r' and 't'. Therefore "eetr" is also a valid answer.
```

Example 2:

```
Input:
"cccaaa"

Output:
"cccaaa"

Explanation:
Both 'c' and 'a' appear three times, so "aaaccc" is also a valid answer.
Note that "cacaca" is incorrect, as the same characters must be together.
```

Example 3:

```
Input:
"Aabb"

Output:
"bbAa"

Explanation:
"bbaA" is also a valid answer, but "Aabb" is incorrect.
Note that 'A' and 'a' are treated as two different characters.
```

题目大意

这道题是 Google 的面试题。

给定一个字符串，要求根据字符出现的频次从高到低重新排列这个字符串。

解题思路

思路比较简单，首先统计每个字符的频次，然后排序，最后按照频次从高到低进行输出即可。

代码

```go
package leetcode

import (
  "sort"
```

```go
)

func frequencySort(s string) string {
  if s == "" {
    return ""
  }
  sMap := map[byte]int{}
  cMap := map[int][]byte{}
  sb := []byte(s)
  for _, b := range sb {
    sMap[b]++
  }
  for key, value := range sMap {
    cMap[value] = append(cMap[value], key)
  }

  var keys []int
  for k := range cMap {
    keys = append(keys, k)
  }
  sort.Sort(sort.Reverse(sort.IntSlice(keys)))
  res := make([]byte, 0)
  for _, k := range keys {
    for i := 0; i < len(cMap[k]); i++ {
      for j := 0; j < k; j++ {
        res = append(res, cMap[k][i])
      }
    }
  }
  return string(res)
}
```

453. Minimum Moves to Equal Array Elements

题目

Given a **non-empty** integer array of size n, find the minimum number of moves required to make all array elements equal, where a move is incrementing n - 1 elements by 1.

Example:

```
Input:
[1,2,3]

Output:
3

Explanation:
Only three moves are needed (remember each move increments two elements):

[1,2,3]  =>  [2,3,3]  =>  [3,4,3]  =>  [4,4,4]
```

题目大意

给定一个长度为 n 的非空整数数组，找到让数组所有元素相等的最小移动次数。每次移动将会使 n - 1 个元素增加 1。

解题思路

- 给定一个数组，要求输出让所有元素都相等的最小步数。每移动一步都会使得 n - 1 个元素 + 1 。
- 数学题。这道题正着思考会考虑到排序或者暴力的方法上去。反过来思考一下，使得每个元素都相同，意思让所有元素的差异变为 0 。每次移动的过程中，都有 n - 1 个元素 + 1，那么没有 + 1 的那个元素和其他 n - 1 个元素相对差异就缩小了。所以这道题让所有元素都变为相等的最少步数，即等于让所有元素相对差异减少到最小的那个数。想到这里，此题就可以优雅的解出来了。

代码

```go
package leetcode

import "math"

func minMoves(nums []int) int {
  sum, min, l := 0, math.MaxInt32, len(nums)
  for _, v := range nums {
    sum += v
    if min > v {
      min = v
    }
  }
  return sum - min*l
}
```

454. 4Sum II

题目

Given four lists A, B, C, D of integer values, compute how many tuples (i, j, k, l) there are such that A[i] + B[j] + C[k] + D[l] is zero.

To make problem a bit easier, all A, B, C, D have same length of N where 0 ≤ N ≤ 500. All integers are in the range of -228 to 228 - 1 and the result is guaranteed to be at most 231 - 1.

Example 1:

```
Input:
A = [ 1, 2]
B = [-2,-1]
C = [-1, 2]
D = [ 0, 2]

Output:
2

Explanation:
The two tuples are:
1. (0, 0, 0, 1) -> A[0] + B[0] + C[0] + D[1] = 1 + (-2) + (-1) + 2 = 0
2. (1, 1, 0, 0) -> A[1] + B[1] + C[0] + D[0] = 2 + (-1) + (-1) + 0 = 0
```

题目大意

给出 4 个数组，计算这些数组中存在几对 i，j，k，l 可以使得 A[i] + B[j] + C[k] + D[l] = 0 。

解题思路

这道题的数据量不大，0 ≤ N ≤ 500，但是如果使用暴力解法，四层循环，会超时。这道题的思路和第 1 题思路也类似，先可以将 2 个数组中的组合都存入 map 中。之后将剩下的 2 个数组进行 for 循环，找出和为 0 的组合。这样时间复杂度是 O(n^2)。当然也可以讲剩下的 2 个数组的组合也存入 map 中，不过最后在 2 个 map 中查找结果也是 O(n^2) 的时间复杂度。

代码

```go
package leetcode

func fourSumCount(A []int, B []int, C []int, D []int) int {
  m := make(map[int]int, len(A)*len(B))
  for _, a := range A {
    for _, b := range B {
      m[a+b]++
```

```
    }
  }
  ret := 0
  for _, c := range C {
    for _, d := range D {
      ret += m[0-c-d]
    }
  }

  return ret
}
```

455. Assign Cookies

题目

Assume you are an awesome parent and want to give your children some cookies. But, you should give each child at most one cookie. Each child i has a greed factor gi, which is the minimum size of a cookie that the child will be content with; and each cookie j has a size sj. If sj >= gi, we can assign the cookie j to the child i, and the child i will be content. Your goal is to maximize the number of your content children and output the maximum number.

Note:You may assume the greed factor is always positive. You cannot assign more than one cookie to one child.

Example 1:

```
Input: [1,2,3], [1,1]

Output: 1

Explanation: You have 3 children and 2 cookies. The greed factors of 3 children are 1, 2, 3.
And even though you have 2 cookies, since their size is both 1, you could only make the child whose greed factor is 1 content.
You need to output 1.
```

Example 2:

```
Input: [1,2], [1,2,3]

Output: 2

Explanation: You have 2 children and 3 cookies. The greed factors of 2 children are 1, 2.
You have 3 cookies and their sizes are big enough to gratify all of the children,
You need to output 2.
```

题目大意

假设你是一位很棒的家长，想要给你的孩子们一些小饼干。但是，每个孩子最多只能给一块饼干。对每个孩子 i ，都有一个胃口值 gi ，这是能让孩子们满足胃口的饼干的最小尺寸；并且每块饼干 j ，都有一个尺寸 sj 。如果 sj >= gi ，我们可以将这个饼干 j 分配给孩子 i ，这个孩子会得到满足。你的目标是尽可能满足越多数量的孩子，并输出这个最大数值。

注意：你可以假设胃口值为正。一个小朋友最多只能拥有一块饼干。

解题思路

- 假设你想给小朋友们饼干，每个小朋友最多能够给一块饼干。每个小朋友都有一个“贪心指数”，称为 `g[i]`， `g[i]` 表示的是这名小朋友需要的饼干大小的最小值。同时，每个饼干都有一个大小值 `s[i]`，如果 `s[j] ≥ g[i]`，我们将饼干 `j` 分给小朋友 `i` 后，小朋友会很开心。给定数组 `g[]` 和 `s[]`，问如何分配饼干，能让更多的小朋友开心。
- 这是一道典型的简单贪心题。贪心题一般都伴随着排序。将 `g[]` 和 `s[]` 分别排序。按照最难满足的小朋友开始给饼干，依次往下满足，最终能满足的小朋友数就是最终解。

代码

```go
package leetcode

import "sort"

func findContentChildren(g []int, s []int) int {
  sort.Ints(g)
  sort.Ints(s)
  gi, si, res := 0, 0, 0
  for gi < len(g) && si < len(s) {
    if s[si] >= g[gi] {
      res++
      si++
      gi++
    } else {
      si++
    }
  }
  return res
}
```

456. 132 Pattern

题目

Given a sequence of n integers a1, a2, ..., an, a 132 pattern is a subsequence a**i**, a**j**, a**k** such that **i** < **j** < **k** and a**i** < a**k** < a**j**. Design an algorithm that takes a list of n numbers as input and checks whether there is a 132 pattern in the list.

Note: n will be less than 15,000.

Example 1:

```
Input: [1, 2, 3, 4]

Output: False

Explanation: There is no 132 pattern in the sequence.
```

Example 2:

```
Input: [3, 1, 4, 2]

Output: True

Explanation: There is a 132 pattern in the sequence: [1, 4, 2].
```

Example 3:

```
Input: [-1, 3, 2, 0]

Output: True

Explanation: There are three 132 patterns in the sequence: [-1, 3, 2], [-1, 3, 0] and [-1, 2, 0].
```

题目大意

给定一个整数序列：a1, a2, ..., an，一个 132 模式的子序列 ai, aj, ak 被定义为：当 i < j < k 时，ai < ak < aj。设计一个算法，当给定有 n 个数字的序列时，验证这个序列中是否含有 132 模式的子序列。注意：n 的值小于 15000。

解题思路

- 这一题用暴力解法一定超时
- 这一题算是单调栈的经典解法，可以考虑从数组末尾开始往前扫，维护一个递减序列

代码

```
package leetcode
```

```go
import (
  "math"
)

// 解法一 单调栈
func find132pattern(nums []int) bool {
  if len(nums) < 3 {
    return false
  }
  num3, stack := math.MinInt64, []int{}
  for i := len(nums) - 1; i >= 0; i-- {
    if nums[i] < num3 {
      return true
    }
    for len(stack) != 0 && nums[i] > stack[len(stack)-1] {
      num3 = stack[len(stack)-1]
      stack = stack[:len(stack)-1]
    }
    stack = append(stack, nums[i])
  }
  return false
}

// 解法二 暴力解法，超时!
func find132pattern1(nums []int) bool {
  if len(nums) < 3 {
    return false
  }
  for j := 0; j < len(nums); j++ {
    stack := []int{}
    for i := j; i < len(nums); i++ {
      if len(stack) == 0 || (len(stack) > 0 && nums[i] > nums[stack[len(stack)-1]]) {
        stack = append(stack, i)
      } else if nums[i] < nums[stack[len(stack)-1]] {
        index := len(stack) - 1
        for ; index >= 0; index-- {
          if nums[stack[index]] < nums[i] {
            return true
          }
        }
      }
    }
  }
  return false
}
```

457. Circular Array Loop

题目

You are given a **circular** array `nums` of positive and negative integers. If a number k at an index is positive, then move forward k steps. Conversely, if it's negative (-k), move backward k steps. Since the array is circular, you may assume that the last element's next element is the first element, and the first element's previous element is the last element.

Determine if there is a loop (or a cycle) in `nums`. A cycle must start and end at the same index and the cycle's length > 1. Furthermore, movements in a cycle must all follow a single direction. In other words, a cycle must not consist of both forward and backward movements.

Example 1:

```
Input: [2,-1,1,2,2]
Output: true
Explanation: There is a cycle, from index 0 -> 2 -> 3 -> 0. The cycle's length is 3.
```

Example 2:

```
Input: [-1,2]
Output: false
Explanation: The movement from index 1 -> 1 -> 1 ... is not a cycle, because the
cycle's length is 1. By definition the cycle's length must be greater than 1.
```

Example 3:

```
Input: [-2,1,-1,-2,-2]
Output: false
Explanation: The movement from index 1 -> 2 -> 1 -> ... is not a cycle, because
movement from index 1 -> 2 is a forward movement, but movement from index 2 -> 1 is a
backward movement. All movements in a cycle must follow a single direction.
```

Note:

1. -1000 ≤ nums[i] ≤ 1000
2. nums[i] ≠ 0
3. 1 ≤ nums.length ≤ 5000

Follow up:

Could you solve it in **O(n)** time complexity and **O(1)** extra space complexity?

题目大意

给定一个含有正整数和负整数的环形数组 nums。 如果某个索引中的数 k 为正数，则向前移动 k 个索引。相反，如果是负数 (-k)，则向后移动 k 个索引。因为数组是环形的，所以可以假设最后一个元素的下一个元素是第一个元素，而第一个元素的前一个元素是最后一个元素。

确定 nums 中是否存在循环（或周期）。循环必须在相同的索引处开始和结束并且循环长度 > 1。此外，一个循环中的所有运动都必须沿着同一方向进行。换句话说，一个循环中不能同时包括向前的运动和向后的运动。

提示：

- -1000 ≤ nums[i] ≤ 1000
- nums[i] ≠ 0
- 1 ≤ nums.length ≤ 5000

进阶：

- 你能写出时间时间复杂度为 O(n) 和额外空间复杂度为 O(1) 的算法吗?

解题思路

- 给出一个循环数组，数组的数字代表了前进和后退的步数，+ 代表往右(前进)，- 代表往左(后退)。问这个循环数组中是否存在一个循环，并且这个循环内不能只有一个元素，循环的方向都必须是同方向的。
- 遇到循环就可以优先考虑用快慢指针的方法判断循环，这一题对循环增加了一个条件，循环不能只是单元素的循环，所以在快慢指针中加入这个判断条件。还有一个判断条件是循环的方向必须是同向的，这个简单，用 `num[i] * num[j] > 0` 就可以判断出是同向的(如果是反向的，那么两者的乘积必然是负数)，如果没有找到循环，可以将当前已经走过的路径上的 num[] 值都置为 0，标记已经访问过了。下次循环遇到访问过的元素，`num[i] * num[j] > 0` 就会是 0，提前退出找循环的过程。

代码

```go
package leetcode

func circularArrayLoop(nums []int) bool {
  if len(nums) == 0 {
    return false
  }
  for i := 0; i < len(nums); i++ {
    if nums[i] == 0 {
      continue
    }
    // slow/fast pointer
    slow, fast, val := i, getNextIndex(nums, i), 0
    for nums[fast]*nums[i] > 0 && nums[getNextIndex(nums, fast)]*nums[i] > 0 {
      if slow == fast {
        // check for loop with only one element
        if slow == getNextIndex(nums, slow) {
          break
        }
        return true
      }
```

```
      slow = getNextIndex(nums, slow)
      fast = getNextIndex(nums, getNextIndex(nums, fast))
    }
    // loop not found, set all element along the way to 0
    slow, val = i, nums[i]
    for nums[slow]*val > 0 {
      next := getNextIndex(nums, slow)
      nums[slow] = 0
      slow = next
    }
  }
  return false
}

func getNextIndex(nums []int, index int) int {
  return ((nums[index]+index)%len(nums) + len(nums)) % len(nums)
}
```

460. LFU Cache

题目

Design and implement a data structure for Least Frequently Used (LFU) cache.

Implement the `LFUCache` class:

- `LFUCache(int capacity)` Initializes the object with the `capacity` of the data structure.
- `int get(int key)` Gets the value of the `key` if the `key` exists in the cache. Otherwise, returns `1`.
- `void put(int key, int value)` Sets or inserts the value if the `key` is not already present. When the cache reaches its `capacity`, it should invalidate the least frequently used item before inserting a new item. For this problem, when there is a tie (i.e., two or more keys with the same frequency), **the least recently** used `key` would be evicted.

Notice that the number of times an item is used is the number of calls to the `get` and `put` functions for that item since it was inserted. This number is set to zero when the item is removed.

Example 1:

```
Input
["LFUCache", "put", "put", "get", "put", "get", "get", "put", "get", "get", "get"]
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [3], [4, 4], [1], [3], [4]]
Output
[null, null, null, 1, null, -1, 3, null, -1, 3, 4]

Explanation
LFUCache lfu = new LFUCache(2);
lfu.put(1, 1);
lfu.put(2, 2);
```

```
lfu.get(1);      // return 1
lfu.put(3, 3);   // evicts key 2
lfu.get(2);      // return -1 (not found)
lfu.get(3);      // return 3
lfu.put(4, 4);   // evicts key 1.
lfu.get(1);      // return -1 (not found)
lfu.get(3);      // return 3
lfu.get(4);      // return 4
```

Constraints:

- `0 <= capacity, key, value <= 104`
- At most `10^5` calls will be made to `get` and `put`.

Follow up: Could you do both operations in `O(1)` time complexity?

题目大意

请你为 最不经常使用（LFU）缓存算法设计并实现数据结构。

实现 LFUCache 类：

- LFUCache(int capacity) - 用数据结构的容量 capacity 初始化对象
- int get(int key) - 如果键存在于缓存中，则获取键的值，否则返回 -1。
- void put(int key, int value) - 如果键已存在，则变更其值；如果键不存在，请插入键值对。当缓存达到其容量时，则应该在插入新项之前，使最不经常使用的项无效。在此问题中，当存在平局（即两个或更多个键具有相同使用频率）时，应该去除 最久未使用 的键。

注意「项的使用次数」就是自插入该项以来对其调用 get 和 put 函数的次数之和。使用次数会在对应项被移除后置为 0 。

进阶：你是否可以在 O(1) 时间复杂度内执行两项操作？

解题思路

- 这一题是 LFU 经典面试题，详细解释见第三章模板。

代码

```go
package leetcode

import "container/list"

type LFUCache struct {
  nodes    map[int]*list.Element
  lists    map[int]*list.List
  capacity int
  min      int
}
```

```go
type node struct {
  key       int
  value     int
  frequency int
}

func Constructor(capacity int) LFUCache {
  return LFUCache{nodes: make(map[int]*list.Element),
    lists:    make(map[int]*list.List),
    capacity: capacity,
    min:      0,
  }
}

func (this *LFUCache) Get(key int) int {
  value, ok := this.nodes[key]
  if !ok {
    return -1
  }
  currentNode := value.Value.(*node)
  this.lists[currentNode.frequency].Remove(value)
  currentNode.frequency++
  if _, ok := this.lists[currentNode.frequency]; !ok {
    this.lists[currentNode.frequency] = list.New()
  }
  newList := this.lists[currentNode.frequency]
  newNode := newList.PushBack(currentNode)
  this.nodes[key] = newNode
  if currentNode.frequency-1 == this.min && this.lists[currentNode.frequency-1].Len()
== 0 {
    this.min++
  }
  return currentNode.value
}

func (this *LFUCache) Put(key int, value int) {
  if this.capacity == 0 {
    return
  }
  if currentValue, ok := this.nodes[key]; ok {
    currentNode := currentValue.Value.(*node)
    currentNode.value = value
    this.Get(key)
    return
  }
  if this.capacity == len(this.nodes) {
    currentList := this.lists[this.min]
    frontNode := currentList.Front()
```

```
    delete(this.nodes, frontNode.Value.(*node).key)
    currentList.Remove(frontNode)
  }
  this.min = 1
  currentNode := &node{
    key:       key,
    value:     value,
    frequency: 1,
  }
  if _, ok := this.lists[1]; !ok {
    this.lists[1] = list.New()
  }
  newList := this.lists[1]
  newNode := newList.PushBack(currentNode)
  this.nodes[key] = newNode
}
```

461. Hamming Distance

题目

The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

Given two integers `x` and `y`, calculate the Hamming distance.

Note: 0 ≤ `x`, `y` < 231.

Example:

```
Input: x = 1, y = 4

Output: 2

Explanation:
1   (0 0 0 1)
4   (0 1 0 0)
       ↑   ↑

The above arrows point to positions where the corresponding bits are different.
```

题目大意

两个整数之间的汉明距离指的是这两个数字对应二进制位不同的位置的数目。给出两个整数 x 和 y，计算它们之间的汉明距离。

注意：
0 ≤ x, y < 231.

解题思路

- 求 2 个数的海明距离。海明距离的定义是两个数二进制位不同的总个数。这一题利用的位操作的是 X &= (X - 1) 不断的清除最低位的 1 。先将这两个数异或，异或以后清除低位的 1 就是最终答案。

代码

```
package leetcode

func hammingDistance(x int, y int) int {
  distance := 0
  for xor := x ^ y; xor != 0; xor &= (xor - 1) {
    distance++
  }
  return distance
}
```

462. Minimum Moves to Equal Array Elements II

题目

Given an integer array `nums` of size `n`, return *the minimum number of moves required to make all array elements equal*.

In one move, you can increment or decrement an element of the array by `1`.

Example 1:

```
Input: nums = [1,2,3]
Output: 2
Explanation:
Only two moves are needed (remember each move increments or decrements one element):
[1,2,3]  =>  [2,2,3]  =>  [2,2,2]
```

Example 2:

```
Input: nums = [1,10,2,9]
Output: 16
```

Constraints:

- `n == nums.length`

- `1 <= nums.length <= 10^5`
- `109 <= nums[i] <= 10^9`

题目大意

给定一个非空整数数组，找到使所有数组元素相等所需的最小移动数，其中每次移动可将选定的一个元素加 1 或减 1。 您可以假设数组的长度最多为10000。

解题思路

- 这题抽象成数学问题是，如果我们把数组 a 中的每个数看成水平轴上的一个点，那么根据上面的移动次数公式，我们需要找到在水平轴上找到一个点 x，使得这 N 个点到 x 的距离之和最小。有 2 个点值得我们考虑，一个是中位数，另外一个是平均值。举个简单的例子，[1,0,0,8,6] 这组数据，中位数是 1，平均值是 3 。分别计算移动的步数，按照中位数对齐是 14，按照平均值对齐是 16 。所以选择中位数。
- 此题可以用数学证明，证明出，按照平均值移动的步数 ≥ 按照中位数移动的步数。具体证明笔者这里不证明了，感兴趣的同学可以自己证明试试。

代码

```go
package leetcode

import (
  "math"
  "sort"
)

func minMoves2(nums []int) int {
  if len(nums) == 0 {
    return 0
  }
  moves, mid := 0, len(nums)/2
  sort.Ints(nums)
  for i := range nums {
    if i == mid {
      continue
    }
    moves += int(math.Abs(float64(nums[mid] - nums[i])))
  }
  return moves
}
```

463. Island Perimeter

题目

You are given a map in form of a two-dimensional integer grid where 1 represents land and 0 represents water.

Grid cells are connected horizontally/vertically (not diagonally). The grid is completely surrounded by water, and there is exactly one island (i.e., one or more connected land cells).

The island doesn't have "lakes" (water inside that isn't connected to the water around the island). One cell is a square with side length 1. The grid is rectangular, width and height don't exceed 100. Determine the perimeter of the island.

Example:

```
Input:
[[0,1,0,0],
 [1,1,1,0],
 [0,1,0,0],
 [1,1,0,0]]

Output: 16

Explanation: The perimeter is the 16 yellow stripes in the image below:
```

题目大意

给定一个包含 0 和 1 的二维网格地图，其中 1 表示陆地 0 表示水域。

网格中的格子水平和垂直方向相连（对角线方向不相连）。整个网格被水完全包围，但其中恰好有一个岛屿（或者说，一个或多个表示陆地的格子相连组成的岛屿）。

岛屿中没有“湖”（“湖” 指水域在岛屿内部且不和岛屿周围的水相连）。格子是边长为 1 的正方形。网格为长方形，且宽度和高度均不超过 100 。计算这个岛屿的周长。

解题思路

- 给出一个二维数组，二维数组中有一些连在一起的 1 ，这是一个岛屿，求这个岛屿的周长。
- 这是一道水题，判断四周边界的情况依次加一即可。

代码

```go
package leetcode

func islandPerimeter(grid [][]int) int {
  counter := 0
  for i := 0; i < len(grid); i++ {
    for j := 0; j < len(grid[0]); j++ {
      if grid[i][j] == 1 {
        if i-1 < 0 || grid[i-1][j] == 0 {
          counter++
        }
        if i+1 >= len(grid) || grid[i+1][j] == 0 {
          counter++
        }
        if j-1 < 0 || grid[i][j-1] == 0 {
          counter++
        }
        if j+1 >= len(grid[0]) || grid[i][j+1] == 0 {
          counter++
        }
      }
    }
  }
  return counter
}
```

470. Implement Rand10() Using Rand7()

题目

Given a function `rand7` which generates a uniform random integer in the range 1 to 7, write a function `rand10` which generates a uniform random integer in the range 1 to 10.

Do NOT use system's `Math.random()`.

Example 1:

```
Input: 1
Output: [7]
```

Example 2:

```
Input: 2
Output: [8,4]
```

Example 3:

```
Input: 3
Output: [8,1,10]
```

Note:

1. `rand7` is predefined.
2. Each testcase has one argument: `n`, the number of times that `rand10` is called.

Follow up:

1. What is the expected value for the number of calls to `rand7()` function?
2. Could you minimize the number of calls to `rand7()`?

题目大意

已有方法 rand7 可生成 1 到 7 范围内的均匀随机整数，试写一个方法 rand10 生成 1 到 10 范围内的均匀随机整数。不要使用系统的 Math.random() 方法。

提示:

- rand7 已定义。
- 传入参数: n 表示 rand10 的调用次数。

进阶:

- rand7()调用次数的 期望值 是多少？
- 你能否尽量少调用 rand7()？

解题思路

- 给出 `rand7()` 要求实现 `rand10()`。
- `rand7()` 等概率地产生1，2，3，4，5，6，7。要想得到 `rand10()` 即等概率的生成 1-10 。解题思路是先构造一个 `randN()`，这个 N 必须是 10 的整数倍，然后 randN % 10 就可以得到 `rand10()` 了。所以可以从 `rand7()` 先构造出 `rand49()`，再把 `rand49()` 中大于等于 40 的都过滤掉，这样就得到了 `rand40()`，在对 10 取余即可。
- 具体构造步骤，`rand7() --> rand49() --> rand40() --> rand10()`：
    1. `rand7()` 等概率地产生 1,2,3,4,5,6,7.
    2. `rand7() - 1` 等概率地产生 [0,6].
    3. `(rand7() - 1) *7` 等概率地产生 0, 7, 14, 21, 28, 35, 42
    4. `(rand7() - 1) * 7 + (rand7() - 1)` 等概率地产生 [0, 48] 这 49 个数字
    5. 如果步骤 4 的结果大于等于 40，那么就重复步骤 4，直到产生的数小于 40
    6. 把步骤 5 的结果 mod 10 再加 1， 就会等概率的随机生成 [1, 10]
- 这道题可以推广到生成任意数的随机数问题。用 `randN()` 实现 `randM()`，`M>N`。步骤如下：
    1. 用 `randN()` 先实现 `randX()`，其中 X ≥ M，X 是 M 的整数倍。如这题中的 49 > 10；
    2. 再用 `randX()` 生成 `randM()`，如这题中的 49 —> 40 —> 10。

- 举个例子，用 `rand3()` 生成 `rand11()`，可以先生成 `rand27()`，然后变成以 22 为界限，因为 22 是 11 的倍数。生成 `rand27()` 的方式：`3 * 3 * (rand3() - 1) + 3 * (rand3() - 1) + (rand3() - 1)`，最后生成了 `rand11()`；用 `rand7()` 生成 `rand9()`，可以先生成 `rand49()`，然后变成以 45 为界限，因为 45 是 9 的倍数。生成 `rand49()` 的方式：`(rand7() - 1) * 7 + (rand7() - 1)`，最后生成了 `rand9()`；用 `rand6()` 生成 `rand13()`，可以先生成 `rand36()`，然后变成以 26 为界限，因为 26 是 13 的倍数。生成 `rand36()` 的方式：`(rand6() - 1) * 6 + (rand6() - 1)`，最后生成了 `rand13()`；

代码

```go
package leetcode

import "math/rand"

func rand10() int {
  rand10 := 10
  for rand10 >= 10 {
    rand10 = (rand7() - 1) + rand7()
  }
  return rand10%10 + 1
}

func rand7() int {
  return rand.Intn(7)
}

func rand101() int {
  rand40 := 40
  for rand40 >= 40 {
    rand40 = (rand7()-1)*7 + rand7() - 1
  }
  return rand40%10 + 1
}
```

473. Matchsticks to Square

题目

You are given an integer array `matchsticks` where `matchsticks[i]` is the length of the `ith` matchstick. You want to use **all the matchsticks** to make one square. You **should not break** any stick, but you can link them up, and each matchstick must be used **exactly one time**.

Return `true` if you can make this square and `false` otherwise.

Example 1:

```
Input: matchsticks = [1,1,2,2,2]
Output: true
Explanation: You can form a square with length 2, one side of the square came two
sticks with length 1.
```

Example 2:

```
Input: matchsticks = [3,3,3,3,4]
Output: false
Explanation: You cannot find a way to form a square with all the matchsticks.
```

Constraints:

- `1 <= matchsticks.length <= 15`
- `0 <= matchsticks[i] <= 109`

题目大意

现在已知小女孩有多少根火柴，请找出一种能使用所有火柴拼成一个正方形的方法。不能折断火柴，可以把火柴连接起来，并且每根火柴都要用到。输入为小女孩拥有火柴的数目，每根火柴用其长度表示。输出即为是否能用所有的火柴拼成正方形。

解题思路

- 将火柴拼成一个正方形，可以将它们分成四组，每一根火柴恰好属于其中的一组；并且每一组火柴的长度之和都相同，等于所有火柴长度之和的四分之一。
- 考虑暴力解法，使用深度优先搜索枚举出所有的分组情况，并对于每一种情况，判断是否满足上述的两个条件(每根火柴属于其中一组，每组火柴长度之和相同)。依次对每一根火柴进行搜索，当搜索到第 i 根火柴时，可以考虑把它放到四组中的任意一种。对于每一种放置方法，继续对第 i + 1 根火柴进行深搜。当我们搜索完全部的 N 根火柴后，再判断每一组火柴的长度之和是否都相同。

代码

```go
package leetcode

import "sort"

func makesquare(matchsticks []int) bool {
  if len(matchsticks) < 4 {
    return false
  }
  total := 0
  for _, v := range matchsticks {
    total += v
  }
  if total%4 != 0 {
    return false
  }
  sort.Slice(matchsticks, func(i, j int) bool {
    return matchsticks[i] > matchsticks[j]
  })
  visited := make([]bool, 16)
  return dfs(matchsticks, 0, 0, 0, total, &visited)
}

func dfs(matchsticks []int, cur, group, sum, total int, visited *[]bool) bool {
  if group == 4 {
    return true
  }
  if sum > total/4 {
    return false
  }
  if sum == total/4 {
    return dfs(matchsticks, 0, group+1, 0, total, visited)
  }
  last := -1
  for i := cur; i < len(matchsticks); i++ {
    if (*visited)[i] {
      continue
    }
    if last == matchsticks[i] {
      continue
    }
    (*visited)[i] = true
    last = matchsticks[i]
    if dfs(matchsticks, i+1, group, sum+matchsticks[i], total, visited) {
      return true
    }
    (*visited)[i] = false
  }
  return false
}
```

474. Ones and Zeroes

题目

In the computer world, use restricted resource you have to generate maximum benefit is what we always want to pursue.

For now, suppose you are a dominator of **m** `0s` and **n** `1s` respectively. On the other hand, there is an array with strings consisting of only `0s` and `1s`.

Now your task is to find the maximum number of strings that you can form with given **m** `0s` and **n** `1s`. Each `0` and `1` can be used at most **once**.

Note:

1. The given numbers of `0s` and `1s` will both not exceed `100`
2. The size of given string array won't exceed `600`.

Example 1:

```
Input: Array = {"10", "0001", "111001", "1", "0"}, m = 5, n = 3
Output: 4

Explanation: This are totally 4 strings can be formed by the using of 5 0s and 3 1s,
which are “10,”0001”,”1”,”0”
```

Example 2:

```
Input: Array = {"10", "0", "1"}, m = 1, n = 1
Output: 2

Explanation: You could form "10", but then you'd have nothing left. Better form "0" and
"1".
```

题目大意

在计算机界中，我们总是追求用有限的资源获取最大的收益。现在，假设你分别支配着 m 个 0 和 n 个 1。另外，还有一个仅包含 0 和 1 字符串的数组。你的任务是使用给定的 m 个 0 和 n 个 1 ，找到能拼出存在于数组中的字符串的最大数量。每个 0 和 1 至多被使用一次。

注意:

1. 给定 0 和 1 的数量都不会超过 100。
2. 给定字符串数组的长度不会超过 600。

解题思路

- 给定一个字符串数组和 m，n，其中所有的字符都是由 0 和 1 组成的。问能否从数组中取出最多的字符串，使得这些取出的字符串中所有的 0 的个数 ≤ m，1 的个数 ≤ n。
- 这一题是典型的 01 背包的题型。只不过是一个二维的背包问题。在 n 个物品中选出一定物品，**尽量完全填满** m 维和 n 维的背包。为什么是尽量填满？因为不一定能完全填满背包。
- `dp[i][j]` 代表尽量填满容量为 `(i,j)` 的背包装下的物品总数，状态转移方程为 `dp[i][j] = max(dp[i][j], 1+dp[i-zero][j-one])`。其中 zero 代表的当前装入物品在 m 维上的体积，也即 0 的个数。one 代表的是当前装入物品在 n 维上的体积，也即 1 的个数。每添加一个物品，比较当前 (i,j) 的背包装下的物品总数和 (i-zero,j-one) 的背包装下的物品总数 + 1，比较这两者的大小，保存两者的最大值。每添加一个物品就刷新这个二维背包，直到所有物品都扫完一遍。dp[m][n] 中存储的就是最终的答案。时间复杂度 `O( n * M * N )`。

代码

```go
package leetcode

import "strings"

func findMaxForm(strs []string, m int, n int) int {
  dp := make([][]int, m+1)
  for i := 0; i < m+1; i++ {
    dp[i] = make([]int, n+1)
  }
  for _, s := range strs {
    zero := strings.Count(s, "0")
    one := len(s) - zero
    if zero > m || one > n {
      continue
    }
    for i := m; i >= zero; i-- {
      for j := n; j >= one; j-- {
        dp[i][j] = max(dp[i][j], 1+dp[i-zero][j-one])
      }
    }
  }
  return dp[m][n]
}
```

475. Heaters

题目

Winter is coming! Your first job during the contest is to design a standard heater with fixed warm radius to warm all the houses.

Now, you are given positions of houses and heaters on a horizontal line, find out minimum radius of heaters so that all houses could be covered by those heaters.

So, your input will be the positions of houses and heaters seperately, and your expected output will be the minimum radius standard of heaters.

Note:

1. Numbers of houses and heaters you are given are non-negative and will not exceed 25000.
2. Positions of houses and heaters you are given are non-negative and will not exceed 10^9.
3. As long as a house is in the heaters' warm radius range, it can be warmed.
4. All the heaters follow your radius standard and the warm radius will the same.

Example 1:

```
Input: [1,2,3],[2]
Output: 1
Explanation: The only heater was placed in the position 2, and if we use the radius 1 standard, then all the houses can be warmed.
```

Example 2:

```
Input: [1,2,3,4],[1,4]
Output: 1
Explanation: The two heater was placed in the position 1 and 4. We need to use radius 1 standard, then all the houses can be warmed.
```

题目大意

冬季已经来临。 你的任务是设计一个有固定加热半径的供暖器向所有房屋供暖。现在，给出位于一条水平线上的房屋和供暖器的位置，找到可以覆盖所有房屋的最小加热半径。所以，你的输入将会是房屋和供暖器的位置。你将输出供暖器的最小加热半径。

说明:

- 给出的房屋和供暖器的数目是非负数且不会超过 25000。
- 给出的房屋和供暖器的位置均是非负数且不会超过10^9。
- 只要房屋位于供暖器的半径内(包括在边缘上)，它就可以得到供暖。
- 所有供暖器都遵循你的半径标准，加热的半径也一样。

解题思路

- 给出一个房子坐标的数组，和一些供暖器坐标的数组，分别表示房子的坐标和供暖器的坐标。要求找到供暖器最小的半径能使得所有的房子都能享受到暖气。
- 这一题可以用暴力的解法，暴力解法依次遍历每个房子的坐标，再遍历每个供暖器，找到距离房子最近的供暖器坐标。在所有这些距离的长度里面找到最大值，这个距离的最大值就是供暖器半径的最小值。时间复杂度

O(n^2)。

- 这一题最优解是二分搜索。在查找距离房子最近的供暖器的时候，先将供暖器排序，然后用二分搜索的方法查找。其他的做法和暴力解法一致。时间复杂度 O(n log n)。

代码

```
package leetcode

import (
  "math"
  "sort"
)

func findRadius(houses []int, heaters []int) int {
  minRad := 0
  sort.Ints(heaters)
  for _, house := range houses {
    // 遍历房子的坐标，维护 heaters 的最小半径
    heater := findClosestHeater(house, heaters)
    rad := heater - house
    if rad < 0 {
      rad = -rad
    }
    if rad > minRad {
      minRad = rad
    }
  }
  return minRad

}

// 二分搜索
func findClosestHeater(pos int, heaters []int) int {
  low, high := 0, len(heaters)-1
  if pos < heaters[low] {
    return heaters[low]
  }
  if pos > heaters[high] {
    return heaters[high]
  }
  for low <= high {
    mid := low + (high-low)>>1
    if pos == heaters[mid] {
      return heaters[mid]
    } else if pos < heaters[mid] {
      high = mid - 1
    } else {
      low = mid + 1
```

```
    }
  }
  // 判断距离两边的 heaters 哪个更近
  if pos-heaters[high] < heaters[low]-pos {
    return heaters[high]
  }
  return heaters[low]
}

// 解法二 暴力搜索
func findRadius1(houses []int, heaters []int) int {
  res := 0
  for i := 0; i < len(houses); i++ {
    dis := math.MaxInt64
    for j := 0; j < len(heaters); j++ {
      dis = min(dis, abs(houses[i]-heaters[j]))
    }
    res = max(res, dis)
  }
  return res
}
```

476. Number Complement

题目

Given a positive integer, output its complement number. The complement strategy is to flip the bits of its binary representation.

Note:

1. The given integer is guaranteed to fit within the range of a 32-bit signed integer.
2. You could assume no leading zero bit in the integer's binary representation.

Example 1:

```
Input: 5
Output: 2
Explanation: The binary representation of 5 is 101 (no leading zero bits), and its
complement is 010. So you need to output 2.
```

Example 2:

```
Input: 1
Output: 0
Explanation: The binary representation of 1 is 1 (no leading zero bits), and its
complement is 0. So you need to output 0.
```

题目大意

给定一个正整数，输出它的补数。补数是对该数的二进制表示取反。

注意:

给定的整数保证在32位带符号整数的范围内。
你可以假定二进制数不包含前导零位。

解题思路

- 求一个正数的补数，补数的定义是对该数的二进制表示取反。当前不能改变符号位。按照题意构造相应的 mask 再取反即可。

代码

```
package leetcode

// 解法一
func findComplement(num int) int {
  xx := ^0 // ^0 = 11111111111111111111111
  for xx&num > 0 {
    xx <<= 1 // 构造出来的 xx = 1111111…000000，0 的个数就是 num 的长度
  }
  return ^xx ^ num // xx ^ num，结果是前面的 0 全是 1 的num，再取反即是答案
}

// 解法二
func findComplement1(num int) int {
  temp := 1
  for temp <= num {
    temp <<= 1 // 构造出来的 temp = 00000……10000，末尾 0 的个数是 num 的长度
  }
  return (temp - 1) ^ num // temp - 1 即是前面都是 0，num 长度的末尾都是 1 的数，再异或 num 即
是最终结果
}
```

477. Total Hamming Distance

题目

The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

Now your job is to find the total Hamming distance between all pairs of the given numbers.

Example:

```
Input: 4, 14, 2

Output: 6

Explanation: In binary representation, the 4 is 0100, 14 is 1110, and 2 is 0010 (just showing the four bits relevant in this case). So the answer will be:
HammingDistance(4, 14) + HammingDistance(4, 2) + HammingDistance(14, 2) = 2 + 2 + 2 = 6.
```

Note:

1. Elements of the given array are in the range of `0` to `10^9`
2. Length of the array will not exceed `10^4`.

题目大意

两个整数的汉明距离指的是这两个数字的二进制数对应位不同的数量。计算一个数组中，任意两个数之间汉明距离的总和。

解题思路

- 计算一个数组内两两元素的海明距离总和。海明距离的定义是两个数二进制位不同的总个数。那么可以把数组中的每个元素 32 位的二进制位依次扫一遍，当扫到某一位上的时候，有 k 个元素在这个位上的值是 1，n - k 个元素在这个位上的值是 0，那么在这一位上所有两两元素的海明距离是 k*(n-k)，当把 32 位全部都扫完以后，累加出来的海明距离就是所有两两元素的海明距离。

代码

```go
package leetcode

func totalHammingDistance(nums []int) int {
  total, n := 0, len(nums)
  for i := 0; i < 32; i++ {
    bitCount := 0
    for j := 0; j < n; j++ {
      bitCount += (nums[j] >> uint(i)) & 1
    }
    total += bitCount * (n - bitCount)
  }
  return total
}
```

```go
// 暴力解法超时!
func totalHammingDistance1(nums []int) int {
  res := 0
  for i := 0; i < len(nums); i++ {
    for j := i + 1; j < len(nums); j++ {
      res += hammingDistance(nums[i], nums[j])
    }
  }
  return res
}
```

478. Generate Random Point in a Circle

题目

Given the radius and x-y positions of the center of a circle, write a function `randPoint` which generates a uniform random point in the circle.

Note:

1. input and output values are in floating-point.
2. radius and x-y position of the center of the circle is passed into the class constructor.
3. a point on the circumference of the circle is considered to be in the circle.
4. `randPoint` returns a size 2 array containing x-position and y-position of the random point, in that order.

Example 1:

```
Input:
["Solution","randPoint","randPoint","randPoint"]
[[1,0,0],[],[],[]]
Output: [null,[-0.72939,-0.65505],[-0.78502,-0.28626],[-0.83119,-0.19803]]
```

Example 2:

```
Input:
["Solution","randPoint","randPoint","randPoint"]
[[10,5,-7.5],[],[],[]]
Output: [null,[11.52438,-8.33273],[2.46992,-16.21705],[11.13430,-12.42337]]
```

Explanation of Input Syntax:

The input is two lists: the subroutines called and their arguments. `Solution`'s constructor has three arguments, the radius, x-position of the center, and y-position of the center of the circle. `randPoint` has no arguments. Arguments are always wrapped with a list, even if there aren't any.

题目大意

给定圆的半径和圆心的 x、y 坐标，写一个在圆中产生均匀随机点的函数 randPoint 。

说明:

- 输入值和输出值都将是浮点数。
- 圆的半径和圆心的 x、y 坐标将作为参数传递给类的构造函数。
- 圆周上的点也认为是在圆中。
- randPoint 返回一个包含随机点的x坐标和y坐标的大小为2的数组。

解题思路

- 随机产生一个圆内的点，这个点一定满足定义 `(x-a)^2+(y-b)^2 ≤ R^2`，其中 `(a,b)` 是圆的圆心坐标，`R` 是半径。
- 先假设圆心坐标在 (0,0)，这样方便计算，最终输出坐标的时候整体加上圆心的偏移量即可。`rand.Float64()` 产生一个 `[0.0,1.0)` 区间的浮点数。`-R ≤ 2 * R * rand() - R < R`，利用随机产生坐标点的横纵坐标 `(x,y)` 与半径 R 的关系，如果 `x^2 + y^2 ≤ R^2`，那么说明产生的点在圆内。最终输出的时候要记得加上圆心坐标的偏移值。

代码

```go
package leetcode

import (
  "math"
  "math/rand"
  "time"
)

type Solution struct {
  r float64
  x float64
  y float64
}

func Constructor(radius float64, x_center float64, y_center float64) Solution {
  rand.Seed(time.Now().UnixNano())
  return Solution{radius, x_center, y_center}
}

func (this *Solution) RandPoint() []float64 {
  /*
     a := angle()
     r := this.r * math.Sqrt(rand.Float64())
     x := r * math.Cos(a) + this.x
     y := r * math.Sin(a) + this.y
     return []float64{x, y}*/
  for {
```

```go
    rx := 2*rand.Float64() - 1.0
    ry := 2*rand.Float64() - 1.0
    x := this.r * rx
    y := this.r * ry
    if x*x+y*y <= this.r*this.r {
      return []float64{x + this.x, y + this.y}
    }
  }
}

func angle() float64 {
  return rand.Float64() * 2 * math.Pi
}

/**
 * Your Solution object will be instantiated and called as such:
 * obj := Constructor(radius, x_center, y_center);
 * param_1 := obj.RandPoint();
 */
```

480. Sliding Window Median

题目

Median is the middle value in an ordered integer list. If the size of the list is even, there is no middle value. So the median is the mean of the two middle value.

Examples:

`[2,3,4]` , the median is `3`

`[2,3]`, the median is `(2 + 3) / 2 = 2.5`

Given an array nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position. Your job is to output the median array for each window in the original array.

For example,

Given nums = `[1,3,-1,-3,5,3,6,7]`, and k = 3.

```
Window position                Median
---------------               -----
[1  3  -1] -3  5  3  6  7       1
 1 [3  -1  -3] 5  3  6  7       -1
 1  3 [-1  -3  5] 3  6  7       -1
 1  3  -1 [-3  5  3] 6  7       3
 1  3  -1  -3 [5  3  6] 7       5
 1  3  -1  -3  5 [3  6  7]      6
```

Therefore, return the median sliding window as `[1,-1,-1,3,5,6]`.

Note:

You may assume `k` is always valid, ie: `k` is always smaller than input array's size for non-empty array.

题目大意

中位数是有序序列最中间的那个数。如果序列的大小是偶数，则没有最中间的数；此时中位数是最中间的两个数的平均数。

例如：

[2,3,4]，中位数是 3

[2,3]，中位数是 (2 + 3) / 2 = 2.5

给出一个数组 nums，有一个大小为 k 的窗口从最左端滑动到最右端。窗口中有 k 个数，每次窗口移动 1 位。你的任务是找出每次窗口移动后得到的新窗口中元素的中位数，并输出由它们组成的数组。

解题思路

- 给定一个数组和一个窗口为 K 的窗口，当窗口从数组的左边滑动到数组右边的时候，输出每次移动窗口以后，在窗口内的排序之后中间大小的值。
- 这一题是第 239 题的升级版。
- 这道题最暴力的方法就是将窗口内的元素都排序，时间复杂度 O(n * K)。
- 另一种思路是用两个优先队列，大顶堆里面的元素都比小顶堆里面的元素小。小顶堆里面存储排序以后中间靠后的值大的元素，大顶堆里面存储排序以后中间靠前的值小的元素。如果 k 是偶数，那么两个堆都有 k/2 个元素，中间值就是两个堆顶的元素；如果 k 是奇数，那么小顶堆比大顶堆多一个元素，中间值就是小顶堆的堆顶元素。删除一个元素，元素都标记到删除的堆中，取 top 的时候注意需要取出没有删除的元素。时间复杂度 O(n * log k) 空间复杂度 O(k)

代码

```go
package leetcode

import (
  "container/heap"
```

```go
  "container/list"
  "sort"
)

// 解法一 用链表按照题意实现 时间复杂度 O(n * k) 空间复杂度 O(k)
func medianSlidingWindow(nums []int, k int) []float64 {
  var res []float64
  w := getWindowList(nums[:k], k)
  res = append(res, getMedian(w, k))

  for p1 := k; p1 < len(nums); p1++ {
    w = removeFromWindow(w, nums[p1-k])
    w = insertInWindow(w, nums[p1])
    res = append(res, getMedian(w, k))
  }
  return res
}

func getWindowList(nums []int, k int) *list.List {
  s := make([]int, k)
  copy(s, nums)
  sort.Ints(s)
  l := list.New()
  for _, n := range s {
    l.PushBack(n)
  }
  return l
}

func removeFromWindow(w *list.List, n int) *list.List {
  for e := w.Front(); e != nil; e = e.Next() {
    if e.Value.(int) == n {
      w.Remove(e)
      return w
    }
  }
  return w
}

func insertInWindow(w *list.List, n int) *list.List {
  for e := w.Front(); e != nil; e = e.Next() {
    if e.Value.(int) >= n {
      w.InsertBefore(n, e)
      return w
    }
  }
  w.PushBack(n)
  return w
}
```

```go
func getMedian(w *list.List, k int) float64 {
  e := w.Front()
  for i := 0; i < k/2; e, i = e.Next(), i+1 {
  }
  if k%2 == 1 {
    return float64(e.Value.(int))
  }
  p := e.Prev()
  return (float64(e.Value.(int)) + float64(p.Value.(int))) / 2
}

// 解法二 用两个堆实现 时间复杂度 O(n * log k) 空间复杂度 O(k)
// 用两个堆记录窗口内的值
// 大顶堆里面的元素都比小顶堆里面的元素小
// 如果 k 是偶数，那么两个堆都有 k/2 个元素，中间值就是两个堆顶的元素
// 如果 k 是奇数，那么小顶堆比大顶堆多一个元素，中间值就是小顶堆的堆顶元素
// 删除一个元素，元素都标记到删除的堆中，取 top 的时候注意需要取出没有删除的元素
func medianSlidingWindow1(nums []int, k int) []float64 {
  ans := []float64{}
  minH := MinHeapR{}
  maxH := MaxHeapR{}
  if minH.Len() > maxH.Len()+1 {
    maxH.Push(minH.Pop())
  } else if minH.Len() < maxH.Len() {
    minH.Push(maxH.Pop())
  }
  for i := range nums {
    if minH.Len() == 0 || nums[i] >= minH.Top() {
      minH.Push(nums[i])
    } else {
      maxH.Push(nums[i])
    }
    if i >= k {
      if nums[i-k] >= minH.Top() {
        minH.Remove(nums[i-k])
      } else {
        maxH.Remove(nums[i-k])
      }
    }
    if minH.Len() > maxH.Len()+1 {
      maxH.Push(minH.Pop())
    } else if minH.Len() < maxH.Len() {
      minH.Push(maxH.Pop())
    }
    if minH.Len()+maxH.Len() == k {
      if k%2 == 0 {
        ans = append(ans, float64(minH.Top()+maxH.Top())/2.0)
      } else {
```

```go
        ans = append(ans, float64(minH.Top()))
      }
    }
    // fmt.Printf("%+v, %+v\n", minH, maxH)
  }
  return ans
}

// IntHeap define
type IntHeap struct {
  data []int
}

// Len define
func (h IntHeap) Len() int { return len(h.data) }

// Swap define
func (h IntHeap) Swap(i, j int) { h.data[i], h.data[j] = h.data[j], h.data[i] }

// Push define
func (h *IntHeap) Push(x interface{}) { h.data = append(h.data, x.(int)) }

// Pop define
func (h *IntHeap) Pop() interface{} {
  x := h.data[h.Len()-1]
  h.data = h.data[0 : h.Len()-1]
  return x
}

// Top defines
func (h IntHeap) Top() int {
  return h.data[0]
}

// MinHeap define
type MinHeap struct {
  IntHeap
}

// Less define
func (h MinHeap) Less(i, j int) bool { return h.data[i] < h.data[j] }

// MaxHeap define
type MaxHeap struct {
  IntHeap
}

// Less define
func (h MaxHeap) Less(i, j int) bool { return h.data[i] > h.data[j] }
```

```go
// MinHeapR define
type MinHeapR struct {
  hp, hpDel MinHeap
}

// Len define
func (h MinHeapR) Len() int { return h.hp.Len() - h.hpDel.Len() }

// Top define
func (h *MinHeapR) Top() int {
  for h.hpDel.Len() > 0 && h.hp.Top() == h.hpDel.Top() {
    heap.Pop(&h.hp)
    heap.Pop(&h.hpDel)
  }
  return h.hp.Top()
}

// Pop define
func (h *MinHeapR) Pop() int {
  x := h.Top()
  heap.Pop(&h.hp)
  return x
}

// Push define
func (h *MinHeapR) Push(x int) { heap.Push(&h.hp, x) }

// Remove define
func (h *MinHeapR) Remove(x int) { heap.Push(&h.hpDel, x) }

// MaxHeapR define
type MaxHeapR struct {
  hp, hpDel MaxHeap
}

// Len define
func (h MaxHeapR) Len() int { return h.hp.Len() - h.hpDel.Len() }

// Top define
func (h *MaxHeapR) Top() int {
  for h.hpDel.Len() > 0 && h.hp.Top() == h.hpDel.Top() {
    heap.Pop(&h.hp)
    heap.Pop(&h.hpDel)
  }
  return h.hp.Top()
}

// Pop define
```

```go
func (h *MaxHeapR) Pop() int {
  x := h.Top()
  heap.Pop(&h.hp)
  return x
}

// Push define
func (h *MaxHeapR) Push(x int) { heap.Push(&h.hp, x) }

// Remove define
func (h *MaxHeapR) Remove(x int) { heap.Push(&h.hpDel, x) }
```

483. Smallest Good Base

题目

For an integer n, we call k>=2 a **good base** of n, if all digits of n base k are 1.

Now given a string representing n, you should return the smallest good base of n in string format.

Example 1:

```
Input: "13"
Output: "3"
Explanation: 13 base 3 is 111.
```

Example 2:

```
Input: "4681"
Output: "8"
Explanation: 4681 base 8 is 11111.
```

Example 3:

```
Input: "1000000000000000000"
Output: "999999999999999999"
Explanation: 1000000000000000000 base 999999999999999999 is 11.
```

Note:

1. The range of n is [3, 10^18].
2. The string representing n is always valid and will not have leading zeros.

题目大意

对于给定的整数 n, 如果n的k（k>=2）进制数的所有数位全为1，则称 k（k>=2）是 n 的一个好进制。

以字符串的形式给出 n, 以字符串的形式返回 n 的最小好进制。

提示：

- n 的取值范围是 [3, 10^18]。
- 输入总是有效且没有前导 0。

解题思路

- 给出一个数 n，要求找一个进制 k，使得数字 n 在 k 进制下每一位都是 1 。求最小的进制 k。
- 这一题等价于求最小的正整数 k，满足存在一个正整数 m 使得

{{< katex display >}}
\sum_{i=0}^{m} k^{i} = \frac{1-k^{m+1}}{1-k} = n
{{< /katex >}}

- 这一题需要确定 k 和 m 两个数的值。m 和 k 是有关系的，确定了一个值，另外一个值也确定了。由

{{< katex display >}}
\frac{1-k^{m+1}}{1-k} = n \
{{< /katex >}}

可得：

{{< katex display >}}
m = log*{k}(kn-n+1) - 1 < log*{k}(kn) = 1 + log_{k}n
{{< /katex >}}

根据题意，可以知道 k ≥2，m ≥1 ，所以有:

{{< katex display >}}
1 \leqslant m \leqslant log_{2}n
{{< /katex >}}

所以 m 的取值范围确定了。那么外层循环从 1 到 log n 遍历。找到一个最小的 k ，能满足：

可以用二分搜索来逼近找到最小的 k。先找到 k 的取值范围。由

{{< katex display >}}
\frac{1-k^{m+1}}{1-k} = n \
{{< /katex >}}

可得，

{{< katex display >}}
k^{m+1} = nk-n+1 < nk\ \Rightarrow k < \sqrt[m]{n}
{{< /katex >}}

所以 k 的取值范围是 [2, n*(1/m)]。再利用二分搜索逼近找到最小的 k 即为答案。

代码

```
package leetcode
```

```go
import (
  "math"
  "math/bits"
  "strconv"
)

func smallestGoodBase(n string) string {
  nVal, _ := strconv.Atoi(n)
  mMax := bits.Len(uint(nVal)) - 1
  for m := mMax; m > 1; m-- {
    k := int(math.Pow(float64(nVal), 1/float64(m)))
    mul, sum := 1, 1
    for i := 0; i < m; i++ {
      mul *= k
      sum += mul
    }
    if sum == nVal {
      return strconv.Itoa(k)
    }
  }
  return strconv.Itoa(nVal - 1)
}
```

485. Max Consecutive Ones

题目

Given a binary array, find the maximum number of consecutive 1s in this array.

Example 1:

```
Input: [1,1,0,1,1,1]
Output: 3
Explanation: The first two digits or the last three digits are consecutive 1s.
    The maximum number of consecutive 1s is 3.
```

Note:

- The input array will only contain `0` and `1`.
- The length of input array is a positive integer and will not exceed 10,000

题目大意

给定一个二进制数组， 计算其中最大连续1的个数。

注意：

- 输入的数组只包含 0 和 1。
- 输入数组的长度是正整数，且不超过 10,000。

解题思路

- 给定一个二进制数组， 计算其中最大连续1的个数。
- 简单题。扫一遍数组，累计 1 的个数，动态维护最大的计数，最终输出即可。

代码

```go
package leetcode

func findMaxConsecutiveOnes(nums []int) int {
  maxCount, currentCount := 0, 0
  for _, v := range nums {
    if v == 1 {
      currentCount++
    } else {
      currentCount = 0
    }
    if currentCount > maxCount {
      maxCount = currentCount
    }
  }
  return maxCount
}
```

491. Increasing Subsequences

题目

Given an integer array, your task is to find all the different possible increasing subsequences of the given array, and the length of an increasing subsequence should be at least 2.

Example:

```
Input: [4, 6, 7, 7]
Output: [[4, 6], [4, 7], [4, 6, 7], [4, 6, 7, 7], [6, 7], [6, 7, 7], [7,7], [4,7,7]]
```

Note:

1. The length of the given array will not exceed 15.
2. The range of integer in the given array is [-100,100].
3. The given array may contain duplicates, and two equal integers should also be considered as a special case of increasing sequence.

题目大意

给定一个整型数组, 你的任务是找到所有该数组的递增子序列，递增子序列的长度至少是 2。

说明:

1. 给定数组的长度不会超过15。
2. 数组中的整数范围是 [-100,100]。
3. 给定数组中可能包含重复数字，相等的数字应该被视为递增的一种情况。

解题思路

- 给出一个数组，要求找出这个数组中所有长度大于 2 的非递减子序列。子序列顺序和原数组元素下标必须是顺序的，不能是逆序的。
- 这一题和第 78 题和第 90 题是类似的题目。第 78 题和第 90 题是求所有子序列，这一题在这两题的基础上增加了非递减和长度大于 2 的条件。需要注意的两点是，原数组中元素可能会重复，最终结果输出的时候需要去重。最终结果输出的去重用 map 处理，数组中重复元素用 DFS 遍历搜索。在每次 DFS 中，用 map 记录遍历过的元素，保证本轮 DFS 中不出现重复的元素，递归到下一层还可以选择值相同，但是下标不同的另外一个元素。外层循环也要加一个 map，这个 map 是过滤每组解因为重复元素导致的重复解，经过过滤以后，起点不同了，最终的解也会不同。
- 这一题和第 78 题，第 90 题类似，可以一起解答和复习。

代码

```go
package leetcode

func findSubsequences(nums []int) [][]int {
  c, visited, res := []int{}, map[int]bool{}, [][]int{}
  for i := 0; i < len(nums)-1; i++ {
    if _, ok := visited[nums[i]]; ok {
      continue
    } else {
      visited[nums[i]] = true
      generateIncSubsets(nums, i, c, &res)
    }
  }
  return res
}

func generateIncSubsets(nums []int, current int, c []int, res *[][]int) {
  c = append(c, nums[current])
  if len(c) >= 2 {
    b := make([]int, len(c))
    copy(b, c)
    *res = append(*res, b)
```

```
  }
  visited := map[int]bool{}
  for i := current + 1; i < len(nums); i++ {
    if nums[current] <= nums[i] {
      if _, ok := visited[nums[i]]; ok {
        continue
      } else {
        visited[nums[i]] = true
        generateIncSubsets(nums, i, c, res)
      }
    }
  }
  c = c[:len(c)-1]
  return
}
```

493. Reverse Pairs

题目

Given an array `nums`, we call `(i, j)` an **important reverse pair** if `i < j` and `nums[i] > 2*nums[j]`.

You need to return the number of important reverse pairs in the given array.

Example1:

```
Input: [1,3,2,3,1]
Output: 2
```

Example2:

```
Input: [2,4,3,5,1]
Output: 3
```

Note:

1. The length of the given array will not exceed `50,000`.
2. All the numbers in the input array are in the range of 32-bit integer.

题目大意

给定一个数组 nums ， 如果 i < j 且 nums[i] > 2*nums[j] 我们就将 (i, j) 称作一个重要翻转对。你需要返回给定数组中的重要翻转对的数量。

注意:

- 给定数组的长度不会超过 50000。
- 输入数组中的所有数字都在 32 位整数的表示范围内。

解题思路

- 给出一个数组，要求找出满足条件的所有的“重要的反转对” (i,j)。重要的反转对的定义是：`i<j`，并且 `nums[i] > 2*nums[j]`。
- 这一题是 327 题的变种题。首先将数组中所有的元素以及各自的 `2*nums[i] + 1` 都放在字典中去重。去重以后再做离散化处理。这一题的测试用例会卡离散化，如果不离散化，Math.MaxInt32 会导致数字溢出，见测试用例中 2147483647, -2147483647 这组测试用例。离散后，映射关系 保存在字典中。从左往右遍历数组，先 query ，再 update ，这个顺序和第 327 题是反的。先 query 查找 `[2*nums[i] + 1, len(indexMap)-1]` 这个区间内满足条件的值，这个区间内的值都是 `> 2*nums[j]` 的。这一题移动的是 `j`，`j` 不断的变化，往线段树中不断插入的是 `i`。每轮循环先 query 一次前一轮循环中累积插入线段树中的 `i`，这些累积在线段树中的代表的是所有在 `j` 前面的 `i`。query 查询的是本轮 `[2*nums[j] + 1, len(indexMap)-1]`，如果能找到，即找到了这样一个 `j`，能满足 `nums[i] > 2*nums[j]`， 把整个数组都扫完，累加的 query 出来的 count 计数就是最终答案。
- 另外一种解法是树状数组。树状数组最擅长解答逆序对的问题。先将原数组中所有的元素值的 2 倍算出来，和原数组合并到一个大数组中。这个大数组中装了所有可能产生 2 倍逆序对的元素值。然后再将他们所有值排序，离散化。离散化以后便将问题集转化成 `[1,N]` 这个区间。于是回到了树状数组经典的求逆序对的问题。逆序插入原数组构造树状数组，或者正序插入原数组构造树状数组都可以解答此题。
- 类似的题目：第 327 题，第 315 题。
- 这一题用线段树和树状数组并不是最优解，用线段树和树状数组解这一题是为了训练线段树和树状数组这两个数据结构。最优解是解法一中的 mergesort。

代码

```go
package leetcode

import (
  "sort"

  "github.com/halfrost/LeetCode-Go/template"
)

// 解法一 归并排序 mergesort，时间复杂度 O(n log n)
func reversePairs(nums []int) int {
  buf := make([]int, len(nums))
  return mergesortCount(nums, buf)
}

func mergesortCount(nums, buf []int) int {
  if len(nums) <= 1 {
    return 0
  }
  mid := (len(nums) - 1) / 2
  cnt := mergesortCount(nums[:mid+1], buf)
  cnt += mergesortCount(nums[mid+1:], buf)
  for i, j := 0, mid+1; i < mid+1; i++ { // Note!!! j is increasing.
    for ; j < len(nums) && nums[i] <= 2*nums[j]; j++ {
```

```go
    }
    cnt += len(nums) - j
  }
  copy(buf, nums)
  for i, j, k := 0, mid+1, 0; k < len(nums); {
    if j >= len(nums) || i < mid+1 && buf[i] > buf[j] {
      nums[k] = buf[i]
      i++
    } else {
      nums[k] = buf[j]
      j++
    }
    k++
  }
  return cnt
}

// 解法二 树状数组，时间复杂度 O(n log n)
func reversePairs1(nums []int) (cnt int) {
  n := len(nums)
  if n <= 1 {
    return
  }
  // 离散化所有下面统计时会出现的元素
  allNums := make([]int, 0, 2*n)
  for _, v := range nums {
    allNums = append(allNums, v, 2*v)
  }
  sort.Ints(allNums)
  k := 1
  kth := map[int]int{allNums[0]: k}
  for i := 1; i < 2*n; i++ {
    if allNums[i] != allNums[i-1] {
      k++
      kth[allNums[i]] = k
    }
  }
  bit := template.BinaryIndexedTree{}
  bit.Init(k)
  for i, v := range nums {
    cnt += i - bit.Query(kth[2*v])
    bit.Add(kth[v], 1)
  }
  return
}

// 解法三 线段树，时间复杂度 O(n log n)
func reversePairs2(nums []int) int {
  if len(nums) < 2 {
```

```
    return 0
  }
  st, numsMap, indexMap, numsArray, res := template.SegmentCountTree{},
make(map[int]int, 0), make(map[int]int, 0), []int{}, 0
  numsMap[nums[0]] = nums[0]
  for _, num := range nums {
    numsMap[num] = num
    numsMap[2*num+1] = 2*num + 1
  }
  // numsArray 是 prefixSum 去重之后的版本，利用 numsMap 去重
  for _, v := range numsMap {
    numsArray = append(numsArray, v)
  }
  // 排序是为了使得线段树中的区间 left <= right，如果此处不排序，线段树中的区间有很多不合法。
  sort.Ints(numsArray)
  // 离散化，构建映射关系
  for i, n := range numsArray {
    indexMap[n] = i
  }
  numsArray = []int{}
  // 离散化，此题如果不离散化，MaxInt32 的数据会使得数字越界。
  for i := 0; i < len(indexMap); i++ {
    numsArray = append(numsArray, i)
  }
  // 初始化线段树，节点内的值都赋值为 0，即计数为 0
  st.Init(numsArray, func(i, j int) int {
    return 0
  })
  for _, num := range nums {
    res += st.Query(indexMap[num*2+1], len(indexMap)-1)
    st.UpdateCount(indexMap[num])
  }
  return res
}
```

494. Target Sum

题目

You are given a list of non-negative integers, a1, a2, ..., an, and a target, S. Now you have 2 symbols `+` and `-`. For each integer, you should choose one from `+` and `-` as its new symbol.

Find out how many ways to assign symbols to make sum of integers equal to target S.

Example 1:

```
Input: nums is [1, 1, 1, 1, 1], S is 3.
Output: 5
Explanation:

-1+1+1+1+1 = 3
+1-1+1+1+1 = 3
+1+1-1+1+1 = 3
+1+1+1-1+1 = 3
+1+1+1+1-1 = 3

There are 5 ways to assign symbols to make the sum of nums be target 3.
```

Note:

1. The length of the given array is positive and will not exceed 20.
2. The sum of elements in the given array will not exceed 1000.
3. Your output answer is guaranteed to be fitted in a 32-bit integer.

题目大意

给定一个非负整数数组，a1, a2, ..., an, 和一个目标数，S。现在有两个符号 + 和 -。对于数组中的任意一个整数，可以从 + 或 - 中选择一个符号添加在前面。返回可以使最终数组和为目标数 S 的所有添加符号的方法数。

提示：

- 数组非空，且长度不会超过 20 。
- 初始的数组的和不会超过 1000 。
- 保证返回的最终结果能被 32 位整数存下。

解题思路

- 给出一个数组，要求在这个数组里面的每个元素前面加上 + 或者 - 号，最终总和等于 S。问有多少种不同的方法。
- 这一题可以用 DP 和 DFS 解答。DFS 方法就不比较暴力简单了。见代码。这里分析一下 DP 的做法。题目要求在数组元素前加上 + 或者 - 号，其实相当于把数组分成了 2 组，一组全部都加 + 号，一组都加 - 号。记 + 号的一组 P ，记 - 号的一组 N，那么可以推出以下的关系。

  ```
  sum(P) - sum(N) = target
  sum(P) + sum(N) + sum(P) - sum(N) = target + sum(P) + sum(N)
                          2 * sum(P) = target + sum(nums)
  ```

  等号两边都加上 `sum(N) + sum(P)`，于是可以得到结果 `2 * sum(P) = target + sum(nums)`，那么这道题就转换成了，能否在数组中找到这样一个集合，和等于 `(target + sum(nums)) / 2`。那么这题就转化为了第 416 题了。`dp[i]` 中存储的是能使和为 `i` 的方法个数。
- 如果和不是偶数，即不能被 2 整除，那说明找不到满足题目要求的解了，直接输出 0 。

代码

```go
func findTargetSumWays(nums []int, S int) int {
  total := 0
  for _, n := range nums {
    total += n
  }
  if S > total || (S+total)%2 == 1 {
    return 0
  }
  target := (S + total) / 2
  dp := make([]int, target+1)
  dp[0] = 1
  for _, n := range nums {
    for i := target; i >= n; i-- {
      dp[i] += dp[i-n]
    }
  }
  return dp[target]
}

// 解法二 DFS
func findTargetSumWays1(nums []int, S int) int {
  // sums[i] 存储的是后缀和 nums[i:]，即从 i 到结尾的和
  sums := make([]int, len(nums))
  sums[len(nums)-1] = nums[len(nums)-1]
  for i := len(nums) - 2; i > -1; i-- {
    sums[i] = sums[i+1] + nums[i]
  }
  res := 0
  dfsFindTargetSumWays(nums, 0, 0, S, &res, sums)
  return res
}

func dfsFindTargetSumWays(nums []int, index int, curSum int, S int, res *int, sums
[]int) {
  if index == len(nums) {
    if curSum == S {
      *(res) = *(res) + 1
    }
    return
  }
  // 剪枝优化：如果 sums[index] 值小于剩下需要正数的值，那么右边就算都是 + 号也无能为力了，所以这里可
以剪枝了
  if S-curSum > sums[index] {
    return
  }
  dfsFindTargetSumWays(nums, index+1, curSum+nums[index], S, res, sums)
  dfsFindTargetSumWays(nums, index+1, curSum-nums[index], S, res, sums)
}
```

496. Next Greater Element I

题目

You are given two arrays (without duplicates) nums1 and nums2 where nums1’s elements are subset of nums2. Find all the next greater numbers for nums1's elements in the corresponding places of nums2.

The Next Greater Number of a number x in nums1 is the first greater number to its right in nums2. If it does not exist, output -1 for this number.

Example 1:

```
Input: nums1 = [4,1,2], nums2 = [1,3,4,2].
Output: [-1,3,-1]
Explanation:
    For number 4 in the first array, you cannot find the next greater number for it in the second array, so output -1.
    For number 1 in the first array, the next greater number for it in the second array is 3.
    For number 2 in the first array, there is no next greater number for it in the second array, so output -1.
```

Example 2:

```
Input: nums1 = [2,4], nums2 = [1,2,3,4].
Output: [3,-1]
Explanation:
    For number 2 in the first array, the next greater number for it in the second array is 3.
    For number 4 in the first array, there is no next greater number for it in the second array, so output -1.
```

Note:

- All elements in nums1 and nums2 are unique.
- The length of both nums1 and nums2 would not exceed 1000.

题目大意

这道题也是简单题。题目给出 2 个数组 A 和 B，针对 A 中的每个数组中的元素，要求在 B 数组中找出比 A 数组中元素大的数，B 中元素之间的顺序保持不变。如果找到了就输出这个值，如果找不到就输出 -1。

解题思路

简单题，依题意做即可。

代码

```
package leetcode

func nextGreaterElement(nums1 []int, nums2 []int) []int {
  if len(nums1) == 0 || len(nums2) == 0 {
    return []int{}
  }
  res, reocrd := []int{}, map[int]int{}
  for i, v := range nums2 {
    reocrd[v] = i
  }
  for i := 0; i < len(nums1); i++ {
    flag := false
    for j := reocrd[nums1[i]]; j < len(nums2); j++ {
      if nums2[j] > nums1[i] {
        res = append(res, nums2[j])
        flag = true
        break
      }
    }
    if flag == false {
      res = append(res, -1)
    }
  }
  return res
}
```

497. Random Point in Non-overlapping Rectangles

题目

Given a list of **non-overlapping** axis-aligned rectangles `rects`, write a function `pick` which randomly and uniformily picks an **integer point** in the space covered by the rectangles.

Note:

1. An **integer point** is a point that has integer coordinates.
2. A point on the perimeter of a rectangle is **included** in the space covered by the rectangles.
3. `i` th rectangle = `rects[i]` = `[x1,y1,x2,y2]`, where `[x1, y1]` are the integer coordinates of the

bottom-left corner, and `[x2, y2]` are the integer coordinates of the top-right corner.
4. length and width of each rectangle does not exceed `2000`.
5. `1 <= rects.length <= 100`
6. `pick` return a point as an array of integer coordinates `[p_x, p_y]`
7. `pick` is called at most `10000` times.

Example 1:

```
Input:
["Solution","pick","pick","pick"]
[[[[1,1,5,5]]],[],[],[]]
Output:
[null,[4,1],[4,1],[3,3]]
```

Example 2:

```
Input:
["Solution","pick","pick","pick","pick","pick"]
[[[[-2,-2,-1,-1],[1,0,3,0]]],[],[],[],[],[]]
Output:
[null,[-1,-2],[2,0],[-2,-1],[3,0],[-2,-2]]
```

Explanation of Input Syntax:

The input is two lists: the subroutines called and their arguments. `Solution`'s constructor has one argument, the array of rectangles `rects`. `pick` has no arguments. Arguments are always wrapped with a list, even if there aren't any.

题目大意

给定一个非重叠轴对齐矩形的列表 rects，写一个函数 pick 随机均匀地选取矩形覆盖的空间中的整数点。

提示：

1. 整数点是具有整数坐标的点。
2. 矩形周边上的点包含在矩形覆盖的空间中。
3. 第 i 个矩形 rects [i] = [x1，y1，x2，y2]，其中 [x1，y1] 是左下角的整数坐标，[x2，y2] 是右上角的整数坐标。
4. 每个矩形的长度和宽度不超过 2000。
5. 1 <= rects.length <= 100
6. pick 以整数坐标数组 [p_x, p_y] 的形式返回一个点。
7. pick 最多被调用10000次。

输入语法的说明：

输入是两个列表：调用的子例程及其参数。Solution 的构造函数有一个参数，即矩形数组 rects。pick 没有参数。参数总是用列表包装的，即使没有也是如此。

解题思路

- 给出一个非重叠轴对齐矩形列表，每个矩形用左下角和右上角的两个坐标表示。要求 `pick()` 随机均匀地选取矩形覆盖的空间中的整数点。
- 这一题是第 528 题的变种题，这一题权重是面积，按权重（面积）选择一个矩形，然后再从矩形中随机选择一个点即可。思路和代码和第 528 题一样。

代码

```go
package leetcode

import "math/rand"

// Solution497 define
type Solution497 struct {
  rects [][]int
  arr   []int
}

// Constructor497 define
func Constructor497(rects [][]int) Solution497 {
  s := Solution497{
    rects: rects,
    arr:   make([]int, len(rects)),
  }

  for i := 0; i < len(rects); i++ {
    area := (rects[i][2] - rects[i][0] + 1) * (rects[i][3] - rects[i][1] + 1)
    if area < 0 {
      area = -area
    }
    if i == 0 {
      s.arr[0] = area
    } else {
      s.arr[i] = s.arr[i-1] + area
    }
  }
  return s
}

// Pick define
func (so *Solution497) Pick() []int {
  r := rand.Int() % so.arr[len(so.arr)-1]
  //get rectangle first
  low, high, index := 0, len(so.arr)-1, -1
  for low <= high {
    mid := low + (high-low)>>1
    if so.arr[mid] > r {
      if mid == 0 || so.arr[mid-1] <= r {
        index = mid
```

```
        break
      }
      high = mid - 1
    } else {
      low = mid + 1
    }
  }
  if index == -1 {
    index = low
  }
  if index > 0 {
    r = r - so.arr[index-1]
  }
  length := so.rects[index][2] - so.rects[index][0]
  return []int{so.rects[index][0] + r%(length+1), so.rects[index][1] + r/(length+1)}
}

/**
 * Your Solution object will be instantiated and called as such:
 * obj := Constructor(rects);
 * param_1 := obj.Pick();
 */
```

498. Diagonal Traverse

题目

Given a matrix of M x N elements (M rows, N columns), return all elements of the matrix in diagonal order as shown in the below image.

Example:

```
Input:
[
 [ 1, 2, 3 ],
 [ 4, 5, 6 ],
 [ 7, 8, 9 ]
]

Output:  [1,2,4,7,5,3,6,8,9]

Explanation:
```

Note:

The total number of elements of the given matrix will not exceed 10,000.

题目大意

给定一个含有 M x N 个元素的矩阵（M 行，N 列），请以对角线遍历的顺序返回这个矩阵中的所有元素，对角线遍历如下图所示。

说明: 给定矩阵中的元素总数不会超过 100000 。

解题思路

- 给出一个二维数组，要求按照如图的方式遍历整个数组。
- 这一题用模拟的方式就可以解出来。需要注意的是边界条件：比如二维数组为空，二维数组退化为一行或者一列，退化为一个元素。具体例子见测试用例。

代码

```
package leetcode

// 解法一
func findDiagonalOrder1(matrix [][]int) []int {
  if matrix == nil || len(matrix) == 0 || len(matrix[0]) == 0 {
    return nil
  }
  row, col, dir, i, x, y, d := len(matrix), len(matrix[0]), [2][2]int{
    {-1, 1},
    {1, -1},
  }, 0, 0, 0, 0
  total := row * col
  res := make([]int, total)
  for i < total {
    for x >= 0 && x < row && y >= 0 && y < col {
      res[i] = matrix[x][y]
```

```go
        i++
        x += dir[d][0]
        y += dir[d][1]
      }
      d = (d + 1) % 2
      if x == row {
        x--
        y += 2
      }
      if y == col {
        y--
        x += 2
      }
      if x < 0 {
        x = 0
      }
      if y < 0 {
        y = 0
      }
    }
    return res
}

// 解法二
func findDiagonalOrder(matrix [][]int) []int {
    if len(matrix) == 0 {
      return []int{}
    }
    if len(matrix) == 1 {
      return matrix[0]
    }
    // dir = 0 代表从右上到左下的方向， dir = 1 代表从左下到右上的方向 dir = -1 代表上一次转变了方向
    m, n, i, j, dir, res := len(matrix), len(matrix[0]), 0, 0, 0, []int{}
    for index := 0; index < m*n; index++ {
      if dir == -1 {
        if (i == 0 && j < n-1) || (j == n-1) { // 上边界和右边界
          i++
          if j > 0 {
            j--
          }
          dir = 0
          addTraverse(matrix, i, j, &res)
          continue
        }
        if (j == 0 && i < m-1) || (i == m-1) { // 左边界和下边界
          if i > 0 {
            i--
          }
          j++
```

```
        dir = 1
        addTraverse(matrix, i, j, &res)
        continue
      }
    }
    if i == 0 && j == 0 {
      res = append(res, matrix[i][j])
      if j < n-1 {
        j++
        dir = -1
        addTraverse(matrix, i, j, &res)
        continue
      } else {
        if i < m-1 {
          i++
          dir = -1
          addTraverse(matrix, i, j, &res)
          continue
        }
      }
    }
    if i == 0 && j < n-1 { // 上边界
      if j < n-1 {
        j++
        dir = -1
        addTraverse(matrix, i, j, &res)
        continue
      }
    }
    if j == 0 && i < m-1 { // 左边界
      if i < m-1 {
        i++
        dir = -1
        addTraverse(matrix, i, j, &res)
        continue
      }
    }
    if j == n-1 { // 右边界
      if i < m-1 {
        i++
        dir = -1
        addTraverse(matrix, i, j, &res)
        continue
      }
    }
    if i == m-1 { // 下边界
      j++
      dir = -1
      addTraverse(matrix, i, j, &res)
```

```
        continue
      }
      if dir == 1 {
        i--
        j++
        addTraverse(matrix, i, j, &res)
        continue
      }
      if dir == 0 {
        i++
        j--
        addTraverse(matrix, i, j, &res)
        continue
      }
    }
    return res
  }

  func addTraverse(matrix [][]int, i, j int, res *[]int) {
    if i >= 0 && i <= len(matrix)-1 && j >= 0 && j <= len(matrix[0])-1 {
      *res = append(*res, matrix[i][j])
    }
  }
```

500. Keyboard Row

题目

Given a List of words, return the words that can be typed using letters of **alphabet** on only one row's of American keyboard like the image below.

Example:

```
Input: ["Hello", "Alaska", "Dad", "Peace"]
Output: ["Alaska", "Dad"]
```

Note:

1. You may use one character in the keyboard more than once.
2. You may assume the input string will only contain letters of alphabet.

题目大意

给定一个单词列表，只返回可以使用在键盘同一行的字母打印出来的单词。键盘如上图所示。

解题思路

- 给出一个字符串数组，要求依次判断数组中的每个字符串是否都位于键盘上的同一个行，如果是就输出。这也是一道水题。

代码

```
package leetcode

import "strings"

func findWords500(words []string) []string {
  rows := []string{"qwertyuiop", "asdfghjkl", "zxcvbnm"}
  output := make([]string, 0)
  for _, s := range words {
    if len(s) == 0 {
      continue
    }
    lowerS := strings.ToLower(s)
    oneRow := false
    for _, r := range rows {
      if strings.ContainsAny(lowerS, r) {
        oneRow = !oneRow
        if !oneRow {
          break
        }
      }
    }
    if oneRow {
      output = append(output, s)
    }
  }
  return output
}
```

503. Next Greater Element II

题目

Given a circular array (the next element of the last element is the first element of the array), print the Next Greater Number for every element. The Next Greater Number of a number x is the first greater number to its traversing-order next in the array, which means you could search circularly to find its next greater number. If it doesn't exist, output -1 for this number.

Example 1:

```
Input: [1,2,1]
Output: [2,-1,2]
Explanation: The first 1's next greater number is 2;
The number 2 can't find next greater number;
The second 1's next greater number needs to search circularly, which is also 2.
```

Note: The length of given array won't exceed 10000.

题目大意

题目给出数组 A，针对 A 中的每个数组中的元素，要求在 A 数组中找出比该元素大的数，A 是一个循环数组。如果找到了就输出这个值，如果找不到就输出 -1。

解题思路

这题是第 496 题的加强版，在第 496 题的基础上增加了循环数组的条件。这一题可以依旧按照第 496 题的做法继续模拟。更好的做法是用单调栈，栈中记录单调递增的下标。

代码

```go
package leetcode

// 解法一 单调栈
func nextGreaterElements(nums []int) []int {
  res := make([]int, 0)
  indexes := make([]int, 0)
  for i := 0; i < len(nums); i++ {
    res = append(res, -1)
  }
  for i := 0; i < len(nums)*2; i++ {
    num := nums[i%len(nums)]
    for len(indexes) > 0 && nums[indexes[len(indexes)-1]] < num {
      index := indexes[len(indexes)-1]
      res[index] = num
      indexes = indexes[:len(indexes)-1]
    }
```

```
    indexes = append(indexes, i%len(nums))
  }
  return res
}

// 解法二
func nextGreaterElements1(nums []int) []int {
  if len(nums) == 0 {
    return []int{}
  }
  res := []int{}
  for i := 0; i < len(nums); i++ {
    j, find := (i+1)%len(nums), false
    for j != i {
      if nums[j] > nums[i] {
        find = true
        res = append(res, nums[j])
        break
      }
      j = (j + 1) % len(nums)
    }
    if !find {
      res = append(res, -1)
    }
  }
  return res
}
```

507. Perfect Number

题目

We define the Perfect Number is a **positive** integer that is equal to the sum of all its **positive** divisors except itself.

Now, given an

integer

n, write a function that returns true when it is a perfect number and false when it is not.

Example:

```
Input: 28
Output: True
Explanation: 28 = 1 + 2 + 4 + 7 + 14
```

Note: The input number **n** will not exceed 100,000,000. (1e8)

题目大意

对于一个 正整数，如果它和除了它自身以外的所有正因子之和相等，我们称它为“完美数”。给定一个 整数 n， 如果他是完美数，返回 True，否则返回 False

解题思路

- 给定一个整数，要求判断这个数是不是完美数。整数的取值范围小于 1e8 。
- 简单题。按照题意描述，先获取这个整数的所有正因子，如果正因子的和等于原来这个数，那么它就是完美数。
- 这一题也可以打表，1e8 以下的完美数其实并不多，就 5 个。

代码

```go
package leetcode

import "math"

// 方法一
func checkPerfectNumber(num int) bool {
  if num <= 1 {
    return false
  }
  sum, bound := 1, int(math.Sqrt(float64(num)))+1
  for i := 2; i < bound; i++ {
    if num%i != 0 {
      continue
    }
    corrDiv := num / i
    sum += corrDiv + i
  }
  return sum == num
}

// 方法二 打表
func checkPerfectNumber_(num int) bool {
  return num == 6 || num == 28 || num == 496 || num == 8128 || num == 33550336
}
```

508. Most Frequent Subtree Sum

题目

Given the root of a tree, you are asked to find the most frequent subtree sum. The subtree sum of a node is defined as the sum of all the node values formed by the subtree rooted at that node (including the node itself). So what is the most frequent subtree sum value? If there is a tie, return all the values with the highest frequency in any order.

Examples 1

Input:

```
  5
 /  \
2   -3
```

return [2, -3, 4], since all the values happen only once, return all of them in any order.

Examples 2

Input:

```
  5
 /  \
2   -5
```

return [2], since 2 happens twice, however -5 only occur once.

Note: You may assume the sum of values in any subtree is in the range of 32-bit signed integer.

题目大意

给出二叉树的根，找出出现次数最多的子树元素和。一个结点的子树元素和定义为以该结点为根的二叉树上所有结点的元素之和（包括结点本身）。然后求出出现次数最多的子树元素和。如果有多个元素出现的次数相同，返回所有出现次数最多的元素（不限顺序）。提示： 假设任意子树元素和均可以用 32 位有符号整数表示。

解题思路

- 给出一个树，要求求出每个节点以自己为根节点的子树的所有节点值的和，最后按照这些和出现的频次，输出频次最多的和，如果频次出现次数最多的对应多个和，则全部输出。
- 递归找出每个节点的累加和，用 map 记录频次，最后把频次最多的输出即可。

代码

```go
package leetcode

import (
  "sort"
)

/**
 * Definition for a binary tree node.
```

```go
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

// 解法一 维护最大频次，不用排序
func findFrequentTreeSum(root *TreeNode) []int {
  memo := make(map[int]int)
  collectSum(root, memo)
  res := []int{}
  most := 0
  for key, val := range memo {
    if most == val {
      res = append(res, key)
    } else if most < val {
      most = val
      res = []int{key}
    }
  }
  return res
}

func collectSum(root *TreeNode, memo map[int]int) int {
  if root == nil {
    return 0
  }
  sum := root.Val + collectSum(root.Left, memo) + collectSum(root.Right, memo)
  if v, ok := memo[sum]; ok {
    memo[sum] = v + 1
  } else {
    memo[sum] = 1
  }
  return sum
}

// 解法二 求出所有和再排序
func findFrequentTreeSum1(root *TreeNode) []int {
  if root == nil {
    return []int{}
  }
  freMap, freList, reFreMap := map[int]int{}, []int{}, map[int][]int{}
  findTreeSum(root, freMap)
  for k, v := range freMap {
    tmp := reFreMap[v]
    tmp = append(tmp, k)
    reFreMap[v] = tmp
  }
```

```go
  for k := range reFreMap {
    freList = append(freList, k)
  }
  sort.Ints(freList)
  return reFreMap[freList[len(freList)-1]]
}

func findTreeSum(root *TreeNode, fre map[int]int) int {
  if root == nil {
    return 0
  }
  if root != nil && root.Left == nil && root.Right == nil {
    fre[root.Val]++
    return root.Val
  }
  val := findTreeSum(root.Left, fre) + findTreeSum(root.Right, fre) + root.Val
  fre[val]++
  return val
}
```

509. Fibonacci Number

题目

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: 2
Output: 1
Explanation: F(2) = F(1) + F(0) = 1 + 0 = 1.
```

Example 2:

```
Input: 3
Output: 2
Explanation: F(3) = F(2) + F(1) = 1 + 1 = 2.
```

Example 3:

```
Input: 4
Output: 3
Explanation: F(4) = F(3) + F(2) = 2 + 1 = 3.
```

Note:

0 ≤ `N` ≤ 30.

题目大意

斐波那契数，通常用 F(n) 表示，形成的序列称为斐波那契数列。该数列由 0 和 1 开始，后面的每一项数字都是前面两项数字的和。也就是：

```
F(0) = 0,   F(1) = 1
F(N) = F(N - 1) + F(N - 2), 其中 N > 1.
```

给定 N，计算 F(N)。

提示：0 ≤ N ≤ 30

解题思路

- 求斐波那契数列
- 这一题解法很多，大的分类是四种，递归，记忆化搜索(dp)，矩阵快速幂，通项公式。其中记忆化搜索可以写 3 种方法，自底向上的，自顶向下的，优化空间复杂度版的。通项公式方法实质是求 a^b 这个还可以用快速幂优化时间复杂度到 O(log n) 。

代码

```go
package leetcode

import "math"

// 解法一 递归法 时间复杂度 O(2^n)，空间复杂度 O(n)
func fib(N int) int {
  if N <= 1 {
    return N
  }
  return fib(N-1) + fib(N-2)
}

// 解法二 自底向上的记忆化搜索 时间复杂度 O(n)，空间复杂度 O(n)
func fib1(N int) int {
  if N <= 1 {
    return N
  }
  cache := map[int]int{0: 0, 1: 1}
```

```go
  for i := 2; i <= N; i++ {
    cache[i] = cache[i-1] + cache[i-2]
  }
  return cache[N]
}

// 解法三 自顶向下的记忆化搜索 时间复杂度 O(n), 空间复杂度 O(n)
func fib2(N int) int {
  if N <= 1 {
    return N
  }
  return memoize(N, map[int]int{0: 0, 1: 1})
}

func memoize(N int, cache map[int]int) int {
  if _, ok := cache[N]; ok {
    return cache[N]
  }
  cache[N] = memoize(N-1, cache) + memoize(N-2, cache)
  return memoize(N, cache)
}

// 解法四 优化版的 dp, 节约内存空间 时间复杂度 O(n), 空间复杂度 O(1)
func fib3(N int) int {
  if N <= 1 {
    return N
  }
  if N == 2 {
    return 1
  }
  current, prev1, prev2 := 0, 1, 1
  for i := 3; i <= N; i++ {
    current = prev1 + prev2
    prev2 = prev1
    prev1 = current
  }
  return current
}

// 解法五 矩阵快速幂 时间复杂度 O(log n), 空间复杂度 O(log n)
// | 1 1 | ^ n   = | F(n+1) F(n)   |
// | 1 0 |         | F(n) F(n-1) |
func fib4(N int) int {
  if N <= 1 {
    return N
  }
  var A = [2][2]int{
    {1, 1},
    {1, 0},
```

```go
  }
  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
}

// 解法六 公式法 f(n)=(1/√5)*{[(1+√5)/2]^n -[(1-√5)/2]^n}，用 时间复杂度在 O(log n) 和 O(n) 之间，空间复杂度 O(1)
// 经过实际测试，会发现 pow() 系统函数比快速幂慢，说明 pow() 比 O(log n) 慢
// 斐波那契数列是一个自然数的数列，通项公式却是用无理数来表达的。而且当 n 趋向于无穷大时，前一项与后一项的比值越来越逼近黄金分割 0.618（或者说后一项与前一项的比值小数部分越来越逼近 0.618）。
// 斐波那契数列用计算机计算的时候可以直接用四舍五入函数 Round 来计算。
func fib5(N int) int {
  var goldenRatio float64 = float64((1 + math.Sqrt(5)) / 2)
  return int(math.Round(math.Pow(goldenRatio, float64(N)) / math.Sqrt(5)))
}

// 解法七 协程版，但是时间特别慢，不推荐，放在这里只是告诉大家，写 LeetCode 算法题的时候，启动 goroutine 特别慢
func fib6(N int) int {
```

```go
    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
}
```

513. Find Bottom Left Tree Value

题目

Given a binary tree, find the leftmost value in the last row of the tree.

Example 1:

```
Input:

    2
   / \
  1   3

Output:
1
```

Example 2:

```
Input:

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

Output:
7
```

Note: You may assume the tree (i.e., the given root node) is not **NULL**.

题目大意

给定一个二叉树，在树的最后一行找到最左边的值。注意: 您可以假设树（即给定的根节点）不为 NULL。

解题思路

- 给出一棵树，输出这棵树最下一层中最左边的节点的值。
- 这一题用 DFS 和 BFS 均可解题。

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

// 解法一 DFS
func findBottomLeftValue(root *TreeNode) int {
  if root == nil {
    return 0
  }
  res, maxHeight := 0, -1
  findBottomLeftValueDFS(root, 0, &res, &maxHeight)
```

```go
  return res
}

func findBottomLeftValueDFS(root *TreeNode, curHeight int, res, maxHeight *int) {
  if curHeight > *maxHeight && root.Left == nil && root.Right == nil {
    *maxHeight = curHeight
    *res = root.Val
  }
  if root.Left != nil {
    findBottomLeftValueDFS(root.Left, curHeight+1, res, maxHeight)
  }
  if root.Right != nil {
    findBottomLeftValueDFS(root.Right, curHeight+1, res, maxHeight)
  }
}

// 解法二 BFS
func findBottomLeftValue1(root *TreeNode) int {
  queue := []*TreeNode{root}
  for len(queue) > 0 {
    next := []*TreeNode{}
    for _, node := range queue {
      if node.Left != nil {
        next = append(next, node.Left)
      }
      if node.Right != nil {
        next = append(next, node.Right)
      }
    }
    if len(next) == 0 {
      return queue[0].Val
    }
    queue = next
  }
  return 0
}
```

515. Find Largest Value in Each Tree Row

题目

You need to find the largest value in each row of a binary tree.

Example:

```
Input:

          1
         / \
        3   2
       / \   \
      5   3   9

Output: [1, 3, 9]
```

题目大意

求在二叉树的每一行中找到最大的值。

解题思路

- 给出一个二叉树，要求依次输出每行的最大值
- 用 BFS 层序遍历，将每层排序取出最大值。改进的做法是遍历中不断更新每层的最大值。

代码

```go
package leetcode

import (
  "math"
  "sort"
)

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

// 解法一 层序遍历二叉树，再将每层排序取出最大值
func largestValues(root *TreeNode) []int {
  tmp := levelOrder(root)
  res := []int{}
  for i := 0; i < len(tmp); i++ {
    sort.Ints(tmp[i])
    res = append(res, tmp[i][len(tmp[i])-1])
  }
```

```go
  return res
}

// 解法二 层序遍历二叉树，遍历过程中不断更新最大值
func largestValues1(root *TreeNode) []int {
  if root == nil {
    return []int{}
  }
  q := []*TreeNode{root}
  var res []int
  for len(q) > 0 {
    qlen := len(q)
    max := math.MinInt32
    for i := 0; i < qlen; i++ {
      node := q[0]
      q = q[1:]
      if node.Val > max {
        max = node.Val
      }
      if node.Left != nil {
        q = append(q, node.Left)
      }
      if node.Right != nil {
        q = append(q, node.Right)
      }
    }
    res = append(res, max)
  }
  return res
}
```

518. Coin Change 2

题目

You are given an integer array `coins` representing coins of different denominations and an integer `amount` representing a total amount of money.

Return *the number of combinations that make up that amount*. If that amount of money cannot be made up by any combination of the coins, return `0`.

You may assume that you have an infinite number of each kind of coin.

The answer is **guaranteed** to fit into a signed **32-bit** integer.

Example 1:

```
Input: amount = 5, coins = [1,2,5]
Output: 4
Explanation: there are four ways to make up the amount:
5=5
5=2+2+1
5=2+1+1+1
5=1+1+1+1+1
```

Example 2:

```
Input: amount = 3, coins = [2]
Output: 0
Explanation: the amount of 3 cannot be made up just with coins of 2.
```

Example 3:

```
Input: amount = 10, coins = [10]
Output: 1
```

Constraints:

- `1 <= coins.length <= 300`
- `1 <= coins[i] <= 5000`
- All the values of `coins` are **unique**.
- `0 <= amount <= 5000`

题目大意

给你一个整数数组 coins 表示不同面额的硬币，另给一个整数 amount 表示总金额。请你计算并返回可以凑成总金额的硬币组合数。如果任何硬币组合都无法凑出总金额，返回 0 。假设每一种面额的硬币有无限个。题目数据保证结果符合 32 位带符号整数。

解题思路

- 此题虽然名字叫 Coin Change，但是不是经典的背包九讲问题。题目中 coins 的每个元素可以选取多次，且不考虑选取元素的顺序，因此这道题实际需要计算的是选取硬币的组合数。定义 dp[i] 表示金额之和等于 i 的硬币组合数，目标求 dp[amount]。初始边界条件为 dp[0] = 1，即不取任何硬币，就这一种取法，金额为 0 。状态转移方程 dp[i] += dp[i-coin]，coin 为当前枚举的 coin。
- 可能有读者会有疑惑，上述做法会不会出现重复计算。答案是不会。外层循环是遍历数组 coins 的值，内层循环是遍历不同的金额之和，在计算 dp[i] 的值时，可以确保金额之和等于 i 的硬币面额的顺序，由于顺序确定，因此不会重复计算不同的排列。
- 和此题完全一致的解题思路的题有，第 377 题和第 494 题。

代码

```go
package leetcode

func change(amount int, coins []int) int {
  dp := make([]int, amount+1)
  dp[0] = 1
  for _, coin := range coins {
    for i := coin; i <= amount; i++ {
      dp[i] += dp[i-coin]
    }
  }
  return dp[amount]
}
```

523. Continuous Subarray Sum

题目

Given an integer array `nums` and an integer `k`, return `true` *if* `nums` *has a continuous subarray of size* ***at least two*** *whose elements sum up to a multiple of* `k`*, or* `false` *otherwise.*

An integer `x` is a multiple of `k` if there exists an integer `n` such that `x = n * k`. `0` is **always** a multiple of `k`.

Example 1:

```
Input: nums = [23,2,4,6,7], k = 6
Output: true
Explanation: [2, 4] is a continuous subarray of size 2 whose elements sum up to 6.
```

Example 2:

```
Input: nums = [23,2,6,4,7], k = 6
Output: true
Explanation: [23, 2, 6, 4, 7] is an continuous subarray of size 5 whose elements sum up
to 42.
42 is a multiple of 6 because 42 = 7 * 6 and 7 is an integer.
```

Example 3:

```
Input: nums = [23,2,6,4,7], k = 13
Output: false
```

Constraints:

- `1 <= nums.length <= 105`
- `0 <= nums[i] <= 109`
- `0 <= sum(nums[i]) <= 231 - 1`

- `1 <= k <= 231 - 1`

题目大意

给你一个整数数组 nums 和一个整数 k ，编写一个函数来判断该数组是否含有同时满足下述条件的连续子数组：

- 子数组大小至少为 2 ，且
- 子数组元素总和为 k 的倍数。

如果存在，返回 true ；否则，返回 false 。如果存在一个整数 n ，令整数 x 符合 x = n * k ，则称 x 是 k 的一个倍数。

解题思路

- 简单题。题目只要求是否存在，不要求找出所有解。用一个变量 sum 记录累加和。子数组的元素和可以用前缀和相减得到，例如 [i,j] 区间内的元素和，可以由 prefixSum[j] - prefixSum[i] 得到。当 prefixSums[j] −prefixSums[i] 为 k 的倍数时，prefixSums[i] 和 prefixSums[j] 除以 k 的余数相同。因此只需要计算每个下标对应的前缀和除以 k 的余数即可，使用 map 存储每个余数第一次出现的下标即可。在 map 中如果存在相同余数的 key，代表当前下标和 map 中这个 key 记录的下标可以满足总和为 k 的倍数这一条件。再判断一下能否满足大小至少为 2 的条件即可。用 2 个下标相减，长度大于等于 2 即满足条件，可以输出 true。

代码

```go
package leetcode

func checkSubarraySum(nums []int, k int) bool {
  m := make(map[int]int)
  m[0] = -1
  sum := 0
  for i, n := range nums {
    sum += n
    if r, ok := m[sum%k]; ok {
      if i-2 >= r {
        return true
      }
    } else {
      m[sum%k] = i
    }
  }
  return false
}
```

524. Longest Word in Dictionary through Deleting

题目

Given a string and a string dictionary, find the longest string in the dictionary that can be formed by deleting some characters of the given string. If there are more than one possible results, return the longest word with the smallest lexicographical order. If there is no possible result, return the empty string.

Example 1:

```
Input:
s = "abpcplea", d = ["ale","apple","monkey","plea"]

Output:
"apple"
```

Example 2:

```
Input:
s = "abpcplea", d = ["a","b","c"]

Output:
"a"
```

Note:

- All the strings in the input will only contain lower-case letters.
- The size of the dictionary won't exceed 1,000.
- The length of all the strings in the input won't exceed 1,000.

题目大意

给出一个初始串，再给定一个字符串数组，要求在字符串数组中找到能在初始串中通过删除字符得到的最长的串，如果最长的串有多组解，要求输出字典序最小的那组解。

解题思路

这道题就单纯的用 O(n^2) 暴力循环即可，注意最终解的要求，如果都是最长的串，要求输出字典序最小的那个串，只要利用字符串比较得到字典序最小的串即可。

代码

```go
package leetcode

func findLongestWord(s string, d []string) string {
  res := ""
```

```
  for i := 0; i < len(d); i++ {
    pointS := 0
    pointD := 0
    for pointS < len(s) && pointD < len(d[i]) {
      if s[pointS] == d[i][pointD] {
        pointD++
      }
      pointS++
    }
    if pointD == len(d[i]) && (len(res) < len(d[i]) || (len(res) == len(d[i]) && res >
d[i])) {
      res = d[i]
    }
  }
  return res
}
```

525. Contiguous Array

题目

Given a binary array `nums`, return *the maximum length of a contiguous subarray with an equal number of* `0` *and* `1`.

Example 1:

```
Input: nums = [0,1]
Output: 2
Explanation: [0, 1] is the longest contiguous subarray with an equal number of 0 and 1.
```

Example 2:

```
Input: nums = [0,1,0]
Output: 2
Explanation: [0, 1] (or [1, 0]) is a longest contiguous subarray with equal number of 0
and 1.
```

Constraints:

- `1 <= nums.length <= 105`
- `nums[i]` is either `0` or `1`.

题目大意

给定一个二进制数组 nums , 找到含有相同数量的 0 和 1 的最长连续子数组，并返回该子数组的长度。

解题思路

- 0 和 1 的数量相同可以转化为两者数量相差为 0，如果将 0 看作为 -1，那么原题转化为求最长连续子数组，其元素和为 0 。又变成了区间内求和的问题，自然而然转换为前缀和来处理。假设连续子数组是 [i,j] 区间，这个区间内元素和为 0 意味着 prefixSum[j] - prefixSum[i] = 0，也就是 prefixSum[i] = prefixSum[j]。不断累加前缀和，将每个前缀和存入 map 中。一旦某个 key 存在了，代表之前某个下标的前缀和和当前下标构成的区间，这段区间内的元素和为 0 。这个区间是所求。扫完整个数组，扫描过程中动态更新最大区间长度，扫描完成便可得到最大区间长度，即最长连续子数组。

代码

```go
package leetcode

func findMaxLength(nums []int) int {
  dict := map[int]int{}
  dict[0] = -1
  count, res := 0, 0
  for i := 0; i < len(nums); i++ {
    if nums[i] == 0 {
      count--
    } else {
      count++
    }
    if idx, ok := dict[count]; ok {
      res = max(res, i-idx)
    } else {
      dict[count] = i
    }
  }
  return res
}

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

526. Beautiful Arrangement

题目

Suppose you have **N** integers from 1 to N. We define a beautiful arrangement as an array that is constructed by these **N** numbers successfully if one of the following is true for the ith position (1 <= i <= N) in this array:

1. The number at the i position is divisible by **i**.th

2. **i** is divisible by the number at the i position.th

Now given N, how many beautiful arrangements can you construct?

Example 1:

```
Input: 2
Output: 2
Explanation:

The first beautiful arrangement is [1, 2]:

Number at the 1st position (i=1) is 1, and 1 is divisible by i (i=1).

Number at the 2nd position (i=2) is 2, and 2 is divisible by i (i=2).

The second beautiful arrangement is [2, 1]:

Number at the 1st position (i=1) is 2, and 2 is divisible by i (i=1).

Number at the 2nd position (i=2) is 1, and i (i=2) is divisible by 1.
```

Note:

1. **N** is a positive integer and will not exceed 15.

题目大意

假设有从 1 到 N 的 N 个整数，如果从这 N 个数字中成功构造出一个数组，使得数组的第 i 位 (1 <= i <= N) 满足如下两个条件中的一个，我们就称这个数组为一个优美的排列。条件：

- 第 i 位的数字能被 i 整除
- i 能被第 i 位上的数字整除

现在给定一个整数 N，请问可以构造多少个优美的排列？

解题思路

- 这一题是第 46 题的加强版。由于这一题给出的数组里面的数字都是不重复的，所以可以当做第 46 题来做。
- 这题比第 46 题多的一个条件是，要求数字可以被它对应的下标 + 1 整除，或者下标 + 1 可以整除下标对应的这个数字。在 DFS 回溯过程中加入这个剪枝条件就可以了。
- 当前做法时间复杂度不是最优的，大概只有 33.3%

代码

```go
package leetcode

// 解法一 暴力打表法
```

```go
func countArrangement1(N int) int {
  res := []int{0, 1, 2, 3, 8, 10, 36, 41, 132, 250, 700, 750, 4010, 4237, 10680, 24679,
87328, 90478, 435812}
  return res[N]
}

// 解法二 DFS 回溯
func countArrangement(N int) int {
  if N == 0 {
    return 0
  }
  nums, used, p, res := make([]int, N), make([]bool, N), []int{}, [][]int{}
  for i := range nums {
    nums[i] = i + 1
  }
  generatePermutation526(nums, 0, p, &res, &used)
  return len(res)
}

func generatePermutation526(nums []int, index int, p []int, res *[][]int, used *[]bool)
{
  if index == len(nums) {
    temp := make([]int, len(p))
    copy(temp, p)
    *res = append(*res, temp)
    return
  }
  for i := 0; i < len(nums); i++ {
    if !(*used)[i] {
      if !(checkDivisible(nums[i], len(p)+1) || checkDivisible(len(p)+1, nums[i])) { //
关键的剪枝条件
        continue
      }
      (*used)[i] = true
      p = append(p, nums[i])
      generatePermutation526(nums, index+1, p, res, used)
      p = p[:len(p)-1]
      (*used)[i] = false
    }
  }
  return
}

func checkDivisible(num, d int) bool {
  tmp := num / d
  if int(tmp)*int(d) == num {
    return true
  }
  return false
```

```
}
```

528. Random Pick with Weight

题目

Given an array `w` of positive integers, where `w[i]` describes the weight of index `i`, write a function `pickIndex` which randomly picks an index in proportion to its weight.

Note:

1. `1 <= w.length <= 10000`
2. `1 <= w[i] <= 10^5`
3. `pickIndex` will be called at most `10000` times.

Example 1:

```
Input:
["Solution","pickIndex"]
[[[1]],[]]
Output: [null,0]
```

Example 2:

```
Input:
["Solution","pickIndex","pickIndex","pickIndex","pickIndex","pickIndex"]
[[[1,3]],[],[],[],[],[]]
Output: [null,0,1,1,1,0]
```

Explanation of Input Syntax:

The input is two lists: the subroutines called and their arguments. `Solution`'s constructor has one argument, the array `w`. `pickIndex` has no arguments. Arguments are always wrapped with a list, even if there aren't any.

题目大意

给定一个正整数数组 w ， 其中 w[i] 代表位置 i 的权重， 请写一个函数 pickIndex ， 它可以随机地获取位置 i， 选取位置 i 的概率与 w[i] 成正比。

说明:

1. 1 <= w.length <= 10000
2. 1 <= w[i] <= 10^5
3. pickIndex 将被调用不超过 10000 次

输入语法说明：

输入是两个列表：调用成员函数名和调用的参数。Solution 的构造函数有一个参数，即数组 w。pickIndex 没有参数。输入参数是一个列表，即使参数为空，也会输入一个 [] 空列表。

解题思路

- 给出一个数组，每个元素值代表该下标的权重值，`pickIndex()` 随机取一个位置 i，这个位置出现的概率和该元素值成正比。
- 由于涉及到了权重的问题，这一题可以先考虑用前缀和处理权重。在 `[0,prefixSum)` 区间内随机选一个整数 `x`，下标 `i` 是满足 `x< prefixSum[i]` 条件的最小下标，求这个下标 `i` 即是最终解。二分搜索查找下标 `i` 。对于某些下标 `i`，所有满足 `prefixSum[i] - w[i] ≤ v < prefixSum[i]` 的整数 `v` 都映射到这个下标。因此，所有的下标都与下标权重成比例。
- 时间复杂度：预处理的时间复杂度是 O(n)，`pickIndex()` 的时间复杂度是 O(log n)。空间复杂度 O(n)。

代码

```go
package leetcode

import (
  "math/rand"
)

// Solution528 define
type Solution528 struct {
  prefixSum []int
}

// Constructor528 define
func Constructor528(w []int) Solution528 {
  prefixSum := make([]int, len(w))
  for i, e := range w {
    if i == 0 {
      prefixSum[i] = e
      continue
    }
    prefixSum[i] = prefixSum[i-1] + e
  }
  return Solution528{prefixSum: prefixSum}
}

// PickIndex define
func (so *Solution528) PickIndex() int {
  n := rand.Intn(so.prefixSum[len(so.prefixSum)-1]) + 1
  low, high := 0, len(so.prefixSum)-1
  for low < high {
    mid := low + (high-low)>>1
```

```
    if so.prefixSum[mid] == n {
      return mid
    } else if so.prefixSum[mid] < n {
      low = mid + 1
    } else {
      high = mid
    }
  }
  return low
}

/**
 * Your Solution object will be instantiated and called as such:
 * obj := Constructor(w);
 * param_1 := obj.PickIndex();
 */
```

529. Minesweeper

题目

Let's play the minesweeper game (Wikipedia, online game)!

You are given a 2D char matrix representing the game board. **'M'** represents an **unrevealed** mine, **'E'** represents an **unrevealed** empty square, **'B'** represents a **revealed** blank square that has no adjacent (above, below, left, right, and all 4 diagonals) mines, **digit** ('1' to '8') represents how many mines are adjacent to this **revealed** square, and finally **'X'** represents a **revealed** mine.

Now given the next click position (row and column indices) among all the **unrevealed** squares ('M' or 'E'), return the board after revealing this position according to the following rules:

1. If a mine ('M') is revealed, then the game is over - change it to **'X'**.
2. If an empty square ('E') with **no adjacent mines** is revealed, then change it to revealed blank ('B') and all of its adjacent **unrevealed** squares should be revealed recursively.
3. If an empty square ('E') with **at least one adjacent mine** is revealed, then change it to a digit ('1' to '8') representing the number of adjacent mines.
4. Return the board when no more squares will be revealed.

Example 1:

```
Input:

[['E', 'E', 'E', 'E', 'E'],
 ['E', 'E', 'M', 'E', 'E'],
 ['E', 'E', 'E', 'E', 'E'],
 ['E', 'E', 'E', 'E', 'E']]
```

```
Click : [3,0]

Output:

[['B', '1', 'E', '1', 'B'],
 ['B', '1', 'M', '1', 'B'],
 ['B', '1', '1', '1', 'B'],
 ['B', 'B', 'B', 'B', 'B']]

Explanation:
```

Example 2:

```
Input:

[['B', '1', 'E', '1', 'B'],
 ['B', '1', 'M', '1', 'B'],
 ['B', '1', '1', '1', 'B'],
 ['B', 'B', 'B', 'B', 'B']]

Click : [1,2]

Output:

[['B', '1', 'E', '1', 'B'],
 ['B', '1', 'X', '1', 'B'],
 ['B', '1', '1', '1', 'B'],
 ['B', 'B', 'B', 'B', 'B']]
```

```
Explanation:
```

Note:

1. The range of the input matrix's height and width is [1,50].
2. The click position will only be an unrevealed square ('M' or 'E'), which also means the input board contains at least one clickable square.
3. The input board won't be a stage when game is over (some mines have been revealed).
4. For simplicity, not mentioned rules should be ignored in this problem. For example, you **don't** need to reveal all the unrevealed mines when the game is over, consider any cases that you will win the game or flag any squares.

题目大意

给定一个代表游戏板的二维字符矩阵。 'M' 代表一个未挖出的地雷，'E' 代表一个未挖出的空方块，'B' 代表没有相邻（上，下，左，右，和所有4个对角线）地雷的已挖出的空白方块，数字（'1' 到 '8'）表示有多少地雷与这块已挖出的方块相邻，'X' 则表示一个已挖出的地雷。现在给出在所有未挖出的方块中（'M'或者'E'）的下一个点击位置（行和列索引），根据以下规则，返回相应位置被点击后对应的面板：

1. 如果一个地雷（'M'）被挖出，游戏就结束了- 把它改为 'X'。
2. 如果一个没有相邻地雷的空方块（'E'）被挖出，修改它为（'B'），并且所有和其相邻的未挖出方块都应该被递归地揭露。
3. 如果一个至少与一个地雷相邻的空方块（'E'）被挖出，修改它为数字（'1'到'8'），表示相邻地雷的数量。
4. 如果在此次点击中，若无更多方块可被揭露，则返回面板。

注意：

- 输入矩阵的宽和高的范围为 [1,50]。
- 点击的位置只能是未被挖出的方块 ('M' 或者 'E')，这也意味着面板至少包含一个可点击的方块。
- 输入面板不会是游戏结束的状态（即有地雷已被挖出）。

- 简单起见，未提及的规则在这个问题中可被忽略。例如，当游戏结束时你不需要挖出所有地雷，考虑所有你可能赢得游戏或标记方块的情况。

解题思路

- 给出一张扫雷地图和点击的坐标，M 代表雷，E 代表还没有点击过的空砖块，B 代表点击过的空砖块，1-8 代表砖块周围 8 个方块里面有雷的个数，X 代表点到了雷。问点击一次以后，输出更新点击以后的地图。
- DPS 和 BFS 都可以解题。先根据原图预处理地图，记录出最终地图的状态，0 代表空白砖块，1-8 代表雷的个数，-1 代表是雷。再 DFS 遍历这张处理后的图，输出最终的地图即可。

代码

```
func updateBoard(board [][]byte, click []int) [][]byte {
  if board[click[0]][click[1]] == 'M' {
    board[click[0]][click[1]] = 'X'
    return board
  }
  mineMap := make([][]int, len(board))
  for i := range board {
    mineMap[i] = make([]int, len(board[i]))
  }
  for i := range board {
    for j := range board[i] {
      if board[i][j] == 'M' {
        mineMap[i][j] = -1
        for _, d := range dir8 {
          nx, ny := i+d[0], j+d[1]
          if isInBoard(board, nx, ny) && mineMap[nx][ny] >= 0 {
            mineMap[nx][ny]++
          }
        }
      }
    }
  }
  mineSweeper(click[0], click[1], board, mineMap, dir8)
  return board
}

func mineSweeper(x, y int, board [][]byte, mineMap [][]int, dir8 [][]int) {
  if board[x][y] != 'M' && board[x][y] != 'E' {
    return
  }
  if mineMap[x][y] == -1 {
    board[x][y] = 'X'
  } else if mineMap[x][y] > 0 {
    board[x][y] = '0' + byte(mineMap[x][y])
  } else {
```

```
    board[x][y] = 'B'
    for _, d := range dir8 {
      nx, ny := x+d[0], y+d[1]
      if isInBoard(board, nx, ny) && mineMap[nx][ny] >= 0 {
        mineSweeper(nx, ny, board, mineMap, dir8)
      }
    }
  }
}
```

530. Minimum Absolute Difference in BST

题目

Given a binary search tree with non-negative values, find the minimum absolute difference between values of any two nodes.

Example:

```
Input:

   1
    \
     3
    /
   2

Output:
1

Explanation:
The minimum absolute difference is 1, which is the difference between 2 and 1 (or
between 2 and 3).
```

Note:

- There are at least two nodes in this BST.
- This question is the same as 783: https://leetcode.com/problems/minimum-distance-between-bst-nodes/

题目大意

给你一棵所有节点为非负值的二叉搜索树，请你计算树中任意两节点的差的绝对值的最小值。

解题思路

- 由于是 BST 树，利用它有序的性质，中根遍历的结果是有序的。中根遍历过程中动态维护前后两个节点的差值，即可找到最小差值。
- 此题与第 783 题完全相同。

代码

```go
package leetcode

import (
  "math"

  "github.com/halfrost/LeetCode-Go/structures"
)

// TreeNode define
type TreeNode = structures.TreeNode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

func getMinimumDifference(root *TreeNode) int {
  res, nodes := math.MaxInt16, -1
  dfsBST(root, &res, &nodes)
  return res
}

func dfsBST(root *TreeNode, res, pre *int) {
  if root == nil {
    return
  }
  dfsBST(root.Left, res, pre)
  if *pre != -1 {
    *res = min(*res, abs(root.Val-*pre))
  }
  *pre = root.Val
  dfsBST(root.Right, res, pre)
}

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

func abs(a int) int {
```

```
  if a > 0 {
    return a
  }
  return -a
}
```

532. K-diff Pairs in an Array

题目

Given an array of integers and an integer k, you need to find the number of unique k-diff pairs in the array. Here a k-diff pair is defined as an integer pair (i, j), where i and j are both numbers in the array and their absolute difference is k.

Example 1:

```
Input: [3, 1, 4, 1, 5], k = 2
Output: 2
Explanation: There are two 2-diff pairs in the array, (1, 3) and (3, 5).
Although we have two 1s in the input, we should only return the number of unique pairs.
```

Example 2:

```
Input:[1, 2, 3, 4, 5], k = 1
Output: 4
Explanation: There are four 1-diff pairs in the array, (1, 2), (2, 3), (3, 4) and (4, 5).
```

Example 3:

```
Input: [1, 3, 1, 5, 4], k = 0
Output: 1
Explanation: There is one 0-diff pair in the array, (1, 1).
```

Note:

1. The pairs (i, j) and (j, i) count as the same pair.
2. The length of the array won't exceed 10,000.
3. All the integers in the given input belong to the range: [-1e7, 1e7].

题目大意

给定一个数组，在数组里面找到几组不同的 pair 对，每个 pair 对相差 K 。问能找出多少组这样的 pair 对。

解题思路

这一题可以用 map 记录每个数字出现的次数。重复的数字也会因为唯一的 key，不用担心某个数字会判断多次。遍历一次 map，每个数字都加上 K 以后，判断字典里面是否存在，如果存在， count ++，如果 K = 0 的情况需要单独判断，如果字典中这个元素频次大于 1，count 也需要 ++。

代码

```
package leetcode

func findPairs(nums []int, k int) int {
  if k < 0 || len(nums) == 0 {
    return 0
  }
  var count int
  m := make(map[int]int, len(nums))
  for _, value := range nums {
    m[value]++
  }
  for key := range m {
    if k == 0 && m[key] > 1 {
      count++
      continue
    }
    if k > 0 && m[key+k] > 0 {
      count++
    }
  }
  return count
}
```

535. Encode and Decode TinyURL

题目

> Note: This is a companion problem to the System Design problem: Design TinyURL.

TinyURL is a URL shortening service where you enter a URL such as `https://leetcode.com/problems/design-tinyurl` and it returns a short URL such as `http://tinyurl.com/4e9iAk`.

Design the `encode` and `decode` methods for the TinyURL service. There is no restriction on how your encode/decode algorithm should work. You just need to ensure that a URL can be encoded to a tiny URL and the tiny URL can be decoded to the original URL.

题目大意

TinyURL是一种URL简化服务， 比如：当你输入一个URL https://leetcode.com/problems/design-tinyurl 时，它将返回一个简化的URL http://tinyurl.com/4e9iAk.

要求：设计一个 TinyURL 的加密 encode 和解密 decode 的方法。你的加密和解密算法如何设计和运作是没有限制的，你只需要保证一个URL可以被加密成一个TinyURL，并且这个TinyURL可以用解密方法恢复成原本的URL。

解题思路

- 简单题。由于题目并无规定 `encode()` 算法，所以自由度非常高。最简单的做法是把原始 `URL` 存起来，并记录下存在字符串数组中的下标位置。`decode()` 的时候根据存储的下标还原原始的 `URL`。

代码

```go
package leetcode

import (
  "fmt"
  "strconv"
  "strings"
)

type Codec struct {
  urls []string
}

func Constructor() Codec {
  return Codec{[]string{}}
}

// Encodes a URL to a shortened URL.
func (this *Codec) encode(longUrl string) string {
  this.urls = append(this.urls, longUrl)
  return "http://tinyurl.com/" + fmt.Sprintf("%v", len(this.urls)-1)
}

// Decodes a shortened URL to its original URL.
func (this *Codec) decode(shortUrl string) string {
```

```
  tmp := strings.Split(shortUrl, "/")
  i, _ := strconv.Atoi(tmp[len(tmp)-1])
  return this.urls[i]
}

/**
 * Your Codec object will be instantiated and called as such:
 * obj := Constructor();
 * url := obj.encode(longUrl);
 * ans := obj.decode(url);
 */
```

537. Complex Number Multiplication

题目

Given two strings representing two complex numbers.

You need to return a string representing their multiplication. Note i2 = -1 according to the definition.

Example 1:

```
Input: "1+1i", "1+1i"
Output: "0+2i"
Explanation: (1 + i) * (1 + i) = 1 + i2 + 2 * i = 2i, and you need convert it to the
form of 0+2i.
```

Example 2:

```
Input: "1+-1i", "1+-1i"
Output: "0+-2i"
Explanation: (1 - i) * (1 - i) = 1 + i2 - 2 * i = -2i, and you need convert it to the
form of 0+-2i.
```

Note:

1. The input strings will not have extra blank.
2. The input strings will be given in the form of **a+bi**, where the integer **a** and **b** will both belong to the range of [-100, 100]. And **the output should be also in this form**.

题目大意

给定两个表示复数的字符串。返回表示它们乘积的字符串。注意，根据定义 i^2 = -1 。

注意:

- 输入字符串不包含额外的空格。
- 输入字符串将以 a+bi 的形式给出，其中整数 a 和 b 的范围均在 [-100, 100] 之间。输出也应当符合这种形式。

解题思路

- 给定 2 个字符串，要求这两个复数的乘积，输出也是字符串格式。
- 数学题。按照复数的运算法则，i^2 = -1，最后输出字符串结果即可。

代码

```
package leetcode

import (
  "strconv"
  "strings"
)

func complexNumberMultiply(a string, b string) string {
  realA, imagA := parse(a)
  realB, imagB := parse(b)
  real := realA*realB - imagA*imagB
  imag := realA*imagB + realB*imagA
  return strconv.Itoa(real) + "+" + strconv.Itoa(imag) + "i"
}

func parse(s string) (int, int) {
  ss := strings.Split(s, "+")
  r, _ := strconv.Atoi(ss[0])
  i, _ := strconv.Atoi(ss[1][:len(ss[1])-1])
  return r, i
}
```

538. Convert BST to Greater Tree

题目

Given the `root` of a Binary Search Tree (BST), convert it to a Greater Tree such that every key of the original BST is changed to the original key plus sum of all keys greater than the original key in BST.

As a reminder, a *binary search tree* is a tree that satisfies these constraints:

- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

Note: This question is the same as 1038: https://leetcode.com/problems/binary-search-tree-to-greater-sum-tree/

Example 1:

```
Input: root = [4,1,6,0,2,5,7,null,null,null,3,null,null,null,8]
Output: [30,36,21,36,35,26,15,null,null,null,33,null,null,null,8]
```

Example 2:

```
Input: root = [0,null,1]
Output: [1,null,1]
```

Example 3:

```
Input: root = [1,0,2]
Output: [3,3,2]
```

Example 4:

```
Input: root = [3,2,4,1]
Output: [7,9,4,10]
```

Constraints:

- The number of nodes in the tree is in the range `[0, 104]`.

- `104 <= Node.val <= 104`
- All the values in the tree are **unique**.
- `root` is guaranteed to be a valid binary search tree.

题目大意

给出二叉 搜索 树的根节点，该树的节点值各不相同，请你将其转换为累加树（Greater Sum Tree），使每个节点 node 的新值等于原树中大于或等于 node.val 的值之和。

提醒一下，二叉搜索树满足下列约束条件：

- 节点的左子树仅包含键 小于 节点键的节点。
- 节点的右子树仅包含键 大于 节点键的节点。
- 左右子树也必须是二叉搜索树。

解题思路

- 根据二叉搜索树的有序性，想要将其转换为累加树，只需按照 右节点 - 根节点 - 左节点的顺序遍历，并累加和即可。
- 此题同第 1038 题。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/structures"
)

// TreeNode define
type TreeNode = structures.TreeNode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

func convertBST(root *TreeNode) *TreeNode {
  if root == nil {
    return root
  }
  sum := 0
  dfs538(root, &sum)
  return root
}
```

```go
func dfs538(root *TreeNode, sum *int) {
  if root == nil {
    return
  }
  dfs538(root.Right, sum)
  root.Val += *sum
  *sum = root.Val
  dfs538(root.Left, sum)
}
```

541. Reverse String II

题目

Given a string and an integer k, you need to reverse the first k characters for every 2k characters counting from the start of the string. If there are less than k characters left, reverse all of them. If there are less than 2k but greater than or equal to k characters, then reverse the first k characters and left the other as original.

Example:

```
Input: s = "abcdefg", k = 2
Output: "bacdfeg"
```

Restrictions:

1. The string consists of lower English letters only.
2. Length of the given string and k will in the range [1, 10000]

题目大意

给定一个字符串和一个整数 k，你需要对从字符串开头算起的每个 2k 个字符的前k个字符进行反转。如果剩余少于 k 个字符，则将剩余的所有全部反转。如果有小于 2k 但大于或等于 k 个字符，则反转前 k 个字符，并将剩余的字符保持原样。

要求:

- 该字符串只包含小写的英文字母。
- 给定字符串的长度和 k 在[1, 10000]范围内。

解题思路

- 要求按照一定规则反转字符串：每 `2 * K` 长度的字符串，反转前 `K` 个字符，后 `K` 个字符串保持不变；对于末尾不够 `2 * K` 的字符串，如果长度大于 `K`，那么反转前 `K` 个字符串，剩下的保持不变。如果长度小于 `K`，则把小于 `K` 的这部分字符串全部反转。
- 这一题是简单题，按照题意反转字符串即可。

代码

```go
package leetcode

func reverseStr(s string, k int) string {
  if k > len(s) {
    k = len(s)
  }
  for i := 0; i < len(s); i = i + 2*k {
    if len(s)-i >= k {
      ss := revers(s[i : i+k])
      s = s[:i] + ss + s[i+k:]
    } else {
      ss := revers(s[i:])
      s = s[:i] + ss
    }
  }
  return s
}
```

542. 01 Matrix

题目

Given a matrix consists of 0 and 1, find the distance of the nearest 0 for each cell.

The distance between two adjacent cells is 1.

Example 1:

```
Input:
[[0,0,0],
 [0,1,0],
 [0,0,0]]

Output:
[[0,0,0],
 [0,1,0],
 [0,0,0]]
```

Example 2:

```
Input:
[[0,0,0],
 [0,1,0],
 [1,1,1]]

Output:
[[0,0,0],
 [0,1,0],
 [1,2,1]]
```

Note:

1. The number of elements of the given matrix will not exceed 10,000.
2. There are at least one 0 in the given matrix.
3. The cells are adjacent in only four directions: up, down, left and right.

题目大意

给定一个由 0 和 1 组成的矩阵，找出每个元素到最近的 0 的距离。两个相邻元素间的距离为 1 。

解题思路

- 给出一个二维数组，数组里面只有 0 和 1 。要求计算每个 1 距离最近的 0 的距离。
- 这一题有 3 种解法，第一种解法最容易想到，BFS。先预处理一下棋盘，将每个 0 都处理为 -1 。将 1 都处理为 0 。将每个 -1 (即原棋盘的 0)都入队，每次出队都将四周的 4 个位置都入队。这就想一颗石头扔进了湖里，一圈一圈的波纹荡开，每一圈都是一层。由于棋盘被我们初始化了，所有为 -1 的都是原来为 0 的，所以波纹扫过来不需要处理这些 -1 的点。棋盘上为 0 的点都是原来为 1 的点，这些点在波纹扫过来的时候就需要赋值更新 level。当下次波纹再次扫到原来为 1 的点的时候，由于它已经被第一次到的波纹更新了值，所以这次不用再更新了。(第一次波纹到的时候一定是最短的)
- 第二种解法是 DFS。先预处理，把周围没有 0 的 1 都重置为最大值。当周围有 0 的 1，距离 0 的位置都是 1，这些点是不需要动的，需要更新的点恰恰应该是那些周围没有 0 的点。当递归的步数 val 比点的值小(这也就是为什么会先把 1 更新成最大值的原因)的时候，不断更新它。
- 第三种解法是 DP。由于有 4 个方向，每次处理 2 个方向，可以降低时间复杂度。第一次循环从上到下，从左到右遍历，先处理上边和左边，第二次循环从下到上，从右到左遍历，再处理右边和下边。

代码

```go
package leetcode

import (
  "math"
)

// 解法一 BFS
func updateMatrixBFS(matrix [][]int) [][]int {
  res := make([][]int, len(matrix))
  if len(matrix) == 0 || len(matrix[0]) == 0 {
```

```go
    return res
  }
  queue := make([][]int, 0)
  for i := range matrix {
    res[i] = make([]int, len(matrix[0]))
    for j := range res[i] {
      if matrix[i][j] == 0 {
        res[i][j] = -1
        queue = append(queue, []int{i, j})
      }
    }
  }
  level := 1
  for len(queue) > 0 {
    size := len(queue)
    for size > 0 {
      size--
      node := queue[0]
      queue = queue[1:]
      i, j := node[0], node[1]
      for _, direction := range [][]int{{-1, 0}, {1, 0}, {0, 1}, {0, -1}} {
        x := i + direction[0]
        y := j + direction[1]
        if x < 0 || x >= len(matrix) || y < 0 || y >= len(matrix[0]) || res[x][y] < 0
|| res[x][y] > 0 {
          continue
        }
        res[x][y] = level
        queue = append(queue, []int{x, y})
      }
    }
    level++
  }
  for i, row := range res {
    for j, cell := range row {
      if cell == -1 {
        res[i][j] = 0
      }
    }
  }
  return res
}

// 解法二 DFS
func updateMatrixDFS(matrix [][]int) [][]int {
  result := [][]int{}
  if len(matrix) == 0 || len(matrix[0]) == 0 {
    return result
  }
```

```go
  maxRow, maxCol := len(matrix), len(matrix[0])
  for r := 0; r < maxRow; r++ {
    for c := 0; c < maxCol; c++ {
      if matrix[r][c] == 1 && hasZero(matrix, r, c) == false {
        // 将四周没有 0 的 1 特殊处理为最大值
        matrix[r][c] = math.MaxInt64
      }
    }
  }
  for r := 0; r < maxRow; r++ {
    for c := 0; c < maxCol; c++ {
      if matrix[r][c] == 1 {
        dfsMatrix(matrix, r, c, -1)
      }
    }
  }
  return (matrix)
}

// 判断四周是否有 0
func hasZero(matrix [][]int, row, col int) bool {
  if row > 0 && matrix[row-1][col] == 0 {
    return true
  }
  if col > 0 && matrix[row][col-1] == 0 {
    return true
  }
  if row < len(matrix)-1 && matrix[row+1][col] == 0 {
    return true
  }
  if col < len(matrix[0])-1 && matrix[row][col+1] == 0 {
    return true
  }
  return false
}

func dfsMatrix(matrix [][]int, row, col, val int) {
  // 不超过棋盘氛围，且 val 要比 matrix[row][col] 小
  if row < 0 || row >= len(matrix) || col < 0 || col >= len(matrix[0]) || (matrix[row]
[col] <= val) {
    return
  }
  if val > 0 {
    matrix[row][col] = val
  }
  dfsMatrix(matrix, row-1, col, matrix[row][col]+1)
  dfsMatrix(matrix, row, col-1, matrix[row][col]+1)
  dfsMatrix(matrix, row+1, col, matrix[row][col]+1)
  dfsMatrix(matrix, row, col+1, matrix[row][col]+1)
```

```go
}

// 解法三 DP
func updateMatrixDP(matrix [][]int) [][]int {
  for i, row := range matrix {
    for j, val := range row {
      if val == 0 {
        continue
      }
      left, top := math.MaxInt16, math.MaxInt16
      if i > 0 {
        top = matrix[i-1][j] + 1
      }
      if j > 0 {
        left = matrix[i][j-1] + 1
      }
      matrix[i][j] = min(top, left)
    }
  }
  for i := len(matrix) - 1; i >= 0; i-- {
    for j := len(matrix[0]) - 1; j >= 0; j-- {
      if matrix[i][j] == 0 {
        continue
      }
      right, bottom := math.MaxInt16, math.MaxInt16
      if i < len(matrix)-1 {
        bottom = matrix[i+1][j] + 1
      }
      if j < len(matrix[0])-1 {
        right = matrix[i][j+1] + 1
      }
      matrix[i][j] = min(matrix[i][j], min(bottom, right))
    }
  }
  return matrix
}
```

547. Number of Provinces

题目

There are **N** students in a class. Some of them are friends, while some are not. Their friendship is transitive in nature. For example, if A is a **direct** friend of B, and B is a **direct**friend of C, then A is an **indirect** friend of C. And we defined a friend circle is a group of students who are direct or indirect friends.

Given a **N*N** matrix **M** representing the friend relationship between students in the class. If M[i][j] = 1, then the ith and jth students are **direct** friends with each other, otherwise not. And you have to output the total number of friend circles among all the students.

Example 1:

```
Input:
[[1,1,0],
 [1,1,0],
 [0,0,1]]
Output: 2
Explanation:The 0th and 1st students are direct friends, so they are in a friend
circle.
The 2nd student himself is in a friend circle. So return 2.
```

Example 2:

```
Input:
[[1,1,0],
 [1,1,1],
 [0,1,1]]
Output: 1
Explanation:The 0th and 1st students are direct friends, the 1st and 2nd students are
direct friends,
so the 0th and 2nd students are indirect friends. All of them are in the same friend
circle, so return 1.
```

Note:

1. N is in range [1,200].
2. M[i][i] = 1 for all students.
3. If M[i][j] = 1, then M[j][i] = 1.

题目大意

班上有 N 名学生。其中有些人是朋友，有些则不是。他们的友谊具有是传递性。如果已知 A 是 B 的朋友，B 是 C 的朋友，那么我们可以认为 A 也是 C 的朋友。所谓的朋友圈，是指所有朋友的集合。

给定一个 N * N 的矩阵 M，表示班级中学生之间的朋友关系。如果 M[i][j] = 1，表示已知第 i 个和 j 个学生互为朋友关系，否则为不知道。你必须输出所有学生中的已知的朋友圈总数。

注意：

- N 在[1,200]的范围内。
- 对于所有学生，有M[i][i] = 1。
- 如果有 M[i][j] = 1，则有 M[j][i] = 1。

解题思路

- 给出一个二维矩阵，矩阵中的行列表示的是两个人之间是否是朋友关系，如果是 1，代表两个人是朋友关系。由于自己和自肯定朋友关系，所以对角线上都是 1，并且矩阵也是关于从左往右下的这条对角线对称。
- 这题有 2 种解法，第一种解法是并查集，依次扫描矩阵，如果两个人认识，并且 root 并不相等就执行 union 操作。扫完所有矩阵，最后数一下还有几个不同的 root 就是最终答案。第二种解法是 DFS 或者 BFS。利用

FloodFill 的想法去染色，每次染色一次，计数器加一。最终扫完整个矩阵，计数器的结果就是最终结果。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/template"
)

// 解法一 并查集

func findCircleNum(M [][]int) int {
  n := len(M)
  if n == 0 {
    return 0
  }
  uf := template.UnionFind{}
  uf.Init(n)
  for i := 0; i < n; i++ {
    for j := 0; j <= i; j++ {
      if M[i][j] == 1 {
        uf.Union(i, j)
      }
    }
  }
  return uf.TotalCount()
}

// 解法二 FloodFill DFS 暴力解法
func findCircleNum1(M [][]int) int {
  if len(M) == 0 {
    return 0
  }
  visited := make([]bool, len(M))
  res := 0
  for i := range M {
    if !visited[i] {
      dfs547(M, i, visited)
      res++
    }
  }
  return res
}

func dfs547(M [][]int, cur int, visited []bool) {
  visited[cur] = true
  for j := 0; j < len(M[cur]); j++ {
```

```
    if !visited[j] && M[cur][j] == 1 {
      dfs547(M, j, visited)
    }
  }
}
```

554. Brick Wall

题目

There is a rectangular brick wall in front of you with `n` rows of bricks. The `ith` row has some number of bricks each of the same height (i.e., one unit) but they can be of different widths. The total width of each row is the same.

Draw a vertical line from the top to the bottom and cross the least bricks. If your line goes through the edge of a brick, then the brick is not considered as crossed. You cannot draw a line just along one of the two vertical edges of the wall, in which case the line will obviously cross no bricks.

Given the 2D array `wall` that contains the information about the wall, return *the minimum number of crossed bricks after drawing such a vertical line*.

Example 1:

```
Input: wall = [[1,2,2,1],[3,1,2],[1,3,2],[2,4],[3,1,2],[1,3,1,1]]
Output: 2
```

Example 2:

```
Input: wall = [[1],[1],[1]]
Output: 3
```

Constraints:

- `n == wall.length`
- `1 <= n <= 10^4`
- `1 <= wall[i].length <= 10^4`
- `1 <= sum(wall[i].length) <= 2 * 10^4`
- `sum(wall[i])` is the same for each row `i`.

- `1 <= wall[i][j] <= 2^31 - 1`

题目大意

你的面前有一堵矩形的、由 n 行砖块组成的砖墙。这些砖块高度相同（也就是一个单位高）但是宽度不同。每一行砖块的宽度之和应该相等。你现在要画一条 自顶向下 的、穿过 最少 砖块的垂线。如果你画的线只是从砖块的边缘经过，就不算穿过这块砖。你不能沿着墙的两个垂直边缘之一画线，这样显然是没有穿过一块砖的。给你一个二维数组 wall ，该数组包含这堵墙的相关信息。其中，wall[i] 是一个代表从左至右每块砖的宽度的数组。你需要找出怎样画才能使这条线 穿过的砖块数量最少 ，并且返回 穿过的砖块数量 。

解题思路

- 既然穿过砖块中缝不算穿过砖块，那么穿过最少砖块数量一定是穿过很多中缝。按行遍历每一行的砖块，累加每行砖块宽度，将每行砖块“缝”的坐标存在 map 中。最后取出 map 中出现频次最高的缝，即为铅垂线要穿过的地方。墙高减去缝出现的频次，剩下的即为穿过砖块的数量。

代码

```go
package leetcode

func leastBricks(wall [][]int) int {
  m := make(map[int]int)
  for _, row := range wall {
    sum := 0
    for i := 0; i < len(row)-1; i++ {
      sum += row[i]
      m[sum]++
    }
  }
  max := 0
  for _, v := range m {
    if v > max {
      max = v
    }
  }
  return len(wall) - max
}
```

557. Reverse Words in a String III

题目

Given a string, you need to reverse the order of characters in each word within a sentence while still preserving whitespace and initial word order.

Example 1:

```
Input: "Let's take LeetCode contest"
Output: "s'teL ekat edoCteeL tsetnoc"
```

Note: In the string, each word is separated by single space and there will not be any extra space in the string.

题目大意

给定一个字符串，你需要反转字符串中每个单词的字符顺序，同时仍保留空格和单词的初始顺序。注意：在字符串中，每个单词由单个空格分隔，并且字符串中不会有任何额外的空格。

解题思路

- 反转字符串，要求按照空格隔开的小字符串为单位反转。
- 这是一道简单题。按照题意反转每个空格隔开的单词即可。

代码

```go
package leetcode

import (
  "strings"
)

func reverseWords(s string) string {
  ss := strings.Split(s, " ")
  for i, s := range ss {
    ss[i] = revers(s)
  }
  return strings.Join(ss, " ")
}

func revers(s string) string {
  bytes := []byte(s)
  i, j := 0, len(bytes)-1
  for i < j {
    bytes[i], bytes[j] = bytes[j], bytes[i]
    i++
    j--
  }
  return string(bytes)
}
```

561. Array Partition I

题目

Given an array of **2n** integers, your task is to group these integers into **n** pairs of integer, say (a1, b1), (a2, b2), ..., (an, bn) which makes sum of min(ai, bi) for all i from 1 to n as large as possible.

Example 1:

```
Input: [1,4,3,2]

Output: 4
Explanation: n is 2, and the maximum sum of pairs is 4 = min(1, 2) + min(3, 4).
```

Note:

1. **n** is a positive integer, which is in the range of [1, 10000].
2. All the integers in the array will be in the range of [-10000, 10000].

题目大意

给定长度为 2n 的数组, 你的任务是将这些数分成 n 对, 例如 (a1, b1), (a2, b2), ..., (an, bn) ， 使得从1 到 n 的 min(ai, bi) 总和最大。

解题思路

- 给定一个 2n 个数组，要求把它们分为 n 组一行，求出各组最小值的总和的最大值。
- 由于题目给的数据范围不大，[-10000, 10000]，所以我们可以考虑用一个哈希表数组，里面存储 i - 10000 元素的频次，偏移量是 10000。这个哈希表能按递增的顺序访问数组，这样可以减少排序的耗时。题目要求求出分组以后求和的最大值，那么所有偏小的元素尽量都安排在一组里面，这样取 min 以后，对最大和影响不大。例如，(1 , 1) 这样安排在一起，min 以后就是 1 。但是如果把相差很大的两个元素安排到一起，那么较大的那个元素就“牺牲”了。例如，(1 , 10000)，取 min 以后就是 1，于是 10000 就“牺牲”了。所以需要优先考虑较小值。
- 较小值出现的频次可能是奇数也可能是偶数。如果是偶数，那比较简单，把它们俩俩安排在一起就可以了。如果是奇数，那么它会落单一次，落单的那个需要和距离它最近的一个元素进行配对，这样对最终的和影响最小。较小值如果是奇数，那么就会影响后面元素的选择，后面元素如果是偶数，由于需要一个元素和前面的较小值配对，所以它剩下的又是奇数个。这个影响会依次传递到后面。所以用一个 flag 标记，如果当前集合中有剩余元素将被再次考虑，则此标志设置为 1。在从下一组中选择元素时，会考虑已考虑的相同额外元素。
- 最后扫描过程中动态的维护 sum 值就可以了。

代码

```go
package leetcode

func arrayPairSum(nums []int) int {
  array := [20001]int{}
  for i := 0; i < len(nums); i++ {
    array[nums[i]+10000]++
  }
```

```
  flag, sum := true, 0
  for i := 0; i < len(array); i++ {
    for array[i] > 0 {
      if flag {
        sum = sum + i - 10000
      }
      flag = !flag
      array[i]--
    }
  }
  return sum
}
```

563. Binary Tree Tilt

题目

Given a binary tree, return the tilt of the **whole tree**.

The tilt of a **tree node** is defined as the **absolute difference** between the sum of all left subtree node values and the sum of all right subtree node values. Null node has tilt 0.

The tilt of the **whole tree** is defined as the sum of all nodes' tilt.

Example:

```
Input:
         1
       /   \
      2     3
Output: 1
Explanation:
Tilt of node 2 : 0
Tilt of node 3 : 0
Tilt of node 1 : |2-3| = 1
Tilt of binary tree : 0 + 0 + 1 = 1
```

Note:

1. The sum of node values in any subtree won't exceed the range of 32-bit integer.
2. All the tilt values won't exceed the range of 32-bit integer.

题目大意

给定一个二叉树，计算整个树的坡度。一个树的节点的坡度定义即为，该节点左子树的结点之和和右子树结点之和的差的绝对值。空结点的的坡度是0。整个树的坡度就是其所有节点的坡度之和。

注意:

1. 任何子树的结点的和不会超过32位整数的范围。
2. 坡度的值不会超过32位整数的范围。

解题思路

- 给出一棵树，计算每个节点的“倾斜度”累加和。“倾斜度”的定义是：左子树和右子树的节点值差值的绝对值。
- 这一题虽然是简单题，但是如果对题目中的“倾斜度”理解的不对，这一题就会出错。“倾斜度”计算的是左子树所有节点的值总和，和，右子树所有节点的值总和的差值。并不是只针对一个节点的左节点值和右节点值的差值。这一点明白以后，这一题就是简单题了。

代码

```go
package leetcode

import "math"

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func findTilt(root *TreeNode) int {
  if root == nil {
    return 0
  }
  sum := 0
  findTiltDFS(root, &sum)
  return sum
}

func findTiltDFS(root *TreeNode, sum *int) int {
  if root == nil {
    return 0
  }
  left := findTiltDFS(root.Left, sum)
  right := findTiltDFS(root.Right, sum)
  *sum += int(math.Abs(float64(left) - float64(right)))
  return root.Val + left + right
}
```

566. Reshape the Matrix

题目

In MATLAB, there is a very useful function called 'reshape', which can reshape a matrix into a new one with different size but keep its original data.

You're given a matrix represented by a two-dimensional array, and two **positive** integers **r** and **c**representing the **row** number and **column** number of the wanted reshaped matrix, respectively.

The reshaped matrix need to be filled with all the elements of the original matrix in the same **row-traversing** order as they were.

If the 'reshape' operation with given parameters is possible and legal, output the new reshaped matrix; Otherwise, output the original matrix.

Example 1:

```
Input:
nums =
[[1,2],
 [3,4]]
r = 1, c = 4
Output:
[[1,2,3,4]]
Explanation:
The row-traversing of nums is [1,2,3,4]. The new reshaped matrix is a 1 * 4 matrix,
fill it row by row by using the previous list.
```

Example 2:

```
Input:
nums =
[[1,2],
 [3,4]]
r = 2, c = 4
Output:
[[1,2],
 [3,4]]
Explanation:
There is no way to reshape a 2 * 2 matrix to a 2 * 4 matrix. So output the original
matrix.
```

Note:

1. The height and width of the given matrix is in range [1, 100].
2. The given r and c are all positive.

题目大意

在 MATLAB 中，有一个非常有用的函数 reshape，它可以将一个矩阵重塑为另一个大小不同的新矩阵，但保留其原始数据。

给出一个由二维数组表示的矩阵，以及两个正整数r和c，分别表示想要的重构的矩阵的行数和列数。重构后的矩阵需要将原始矩阵的所有元素以相同的行遍历顺序填充。如果具有给定参数的reshape操作是可行且合理的，则输出新的重塑矩阵；否则，输出原始矩阵。

解题思路

- 给一个二维数组和 r，c，将这个二维数组“重塑”成行为 r，列为 c。如果可以“重塑”，输出“重塑”以后的数组，如果不能“重塑”，输出原有数组。
- 这题也是水题，按照题意模拟即可。

代码

```go
package leetcode

func matrixReshape(nums [][]int, r int, c int) [][]int {
  if canReshape(nums, r, c) {
    return reshape(nums, r, c)
  }
  return nums
}

func canReshape(nums [][]int, r, c int) bool {
  row := len(nums)
  colume := len(nums[0])
  if row*colume == r*c {
    return true
  }
  return false
}

func reshape(nums [][]int, r, c int) [][]int {
  newShape := make([][]int, r)
  for index := range newShape {
    newShape[index] = make([]int, c)
  }
  rowIndex, colIndex := 0, 0
  for _, row := range nums {
    for _, col := range row {
      if colIndex == c {
        colIndex = 0
        rowIndex++
      }
```

```
        newShape[rowIndex][colIndex] = col
        colIndex++
      }
    }
    return newShape
  }
```

567. Permutation in String

题目

Given two strings s1 and s2, write a function to return true if s2 contains the permutation of s1. In other words, one of the first string's permutations is the substring of the second string.

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
```

Note:

1. The input strings only contain lower case letters.
2. The length of both given strings is in range [1, 10,000].

题目大意

在一个字符串中寻找子串出现的位置。子串可以是 Anagrams 形式存在的。Anagrams 是一个字符串任意字符的全排列组合。

解题思路

这一题和第 438 题，第 3 题，第 76 题，第 567 题类似，用的思想都是"滑动窗口"。

这道题只需要判断是否存在，而不需要输出子串所在的下标起始位置。所以这道题是第 438 题的缩水版。具体解题思路见第 438 题。

代码

```
package leetcode

func checkInclusion(s1 string, s2 string) bool {
  var freq [256]int
  if len(s2) == 0 || len(s2) < len(s1) {
    return false
  }
  for i := 0; i < len(s1); i++ {
    freq[s1[i]-'a']++
  }
  left, right, count := 0, 0, len(s1)

  for right < len(s2) {
    if freq[s2[right]-'a'] >= 1 {
      count--
    }
    freq[s2[right]-'a']--
    right++
    if count == 0 {
      return true
    }
    if right-left == len(s1) {
      if freq[s2[left]-'a'] >= 0 {
        count++
      }
      freq[s2[left]-'a']++
      left++
    }
  }
  return false
}
```

572. Subtree of Another Tree

题目

Given two non-empty binary trees **s** and **t**, check whether tree **t** has exactly the same structure and node values with a subtree of **s**. A subtree of **s** is a tree consists of a node in **s** and all of this node's descendants. The tree **s** could also be considered as a subtree of itself.

Example 1:

Given tree s:

```
     3
      / \
    4    5
   / \
  1    2
```

Given tree t:

```
    4
   / \
  1    2
```

Return **true**, because t has the same structure and node values with a subtree of s.

Example 2:

Given tree s:

```
       3
      / \
    4    5
   / \
  1    2
      /
    0
```

Given tree t:

```
    4
   / \
  1    2
```

Return **false**.

题目大意

给定两个非空二叉树 s 和 t，检验 s 中是否包含和 t 具有相同结构和节点值的子树。s 的一个子树包括 s 的一个节点和这个节点的所有子孙。s 也可以看做它自身的一棵子树。

解题思路

- 给出 2 棵树 `s` 和 `t`，要求判断 `t` 是否是 `s` 的子树🌲。
- 这一题比较简单，针对 3 种情况依次递归判断，第一种情况 `s` 和 `t` 是完全一样的两棵树，第二种情况 `t` 是 `s` 左子树中的子树，第三种情况 `t` 是 `s` 右子树中的子树。第一种情况判断两棵数是否完全一致是第 100 题的原题。

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func isSubtree(s *TreeNode, t *TreeNode) bool {
  if isSameTree(s, t) {
    return true
  }
  if s == nil {
    return false
  }
  if isSubtree(s.Left, t) || isSubtree(s.Right, t) {
    return true
  }
  return false
}
```

575. Distribute Candies

题目

Given an integer array with **even** length, where different numbers in this array represent different **kinds** of candies. Each number means one candy of the corresponding kind. You need to distribute these candies **equally** in number to brother and sister. Return the maximum number of **kinds** of candies the sister could gain.

Example 1:

```
Input: candies = [1,1,2,2,3,3]
Output: 3
Explanation:
There are three different kinds of candies (1, 2 and 3), and two candies for each kind.
Optimal distribution: The sister has candies [1,2,3] and the brother has candies
[1,2,3], too.
The sister has three different kinds of candies.
```

Example 2:

```
Input: candies = [1,1,2,3]
Output: 2
Explanation: For example, the sister has candies [2,3] and the brother has candies
[1,1].
The sister has two different kinds of candies, the brother has only one kind of
candies.
```

Note:

1. The length of the given array is in range [2, 10,000], and will be even.
2. The number in given array is in range [-100,000, 100,000].

题目大意

给定一个偶数长度的数组，其中不同的数字代表着不同种类的糖果，每一个数字代表一个糖果。你需要把这些糖果平均分给一个弟弟和一个妹妹。返回妹妹可以获得的最大糖果的种类数。

解题思路

- 给出一个糖果数组，里面每个元素代表糖果的种类，相同数字代表相同种类。把这些糖果分给兄弟姐妹，问姐妹最多可以分到多少种糖果。这一题比较简单，用 map 统计每个糖果的出现频次，如果总数比 `n/2` 小，那么就返回 `len(map)`，否则返回 `n/2` (即一半都分给姐妹)。

代码

```go
package leetcode

func distributeCandies(candies []int) int {
  n, m := len(candies), make(map[int]struct{}, len(candies))
  for _, candy := range candies {
    m[candy] = struct{}{}
  }
  res := len(m)
  if n/2 < res {
    return n / 2
  }
  return res
}
```

581. Shortest Unsorted Continuous Subarray

题目

Given an integer array `nums`, you need to find one **continuous subarray** that if you only sort this subarray in ascending order, then the whole array will be sorted in ascending order.

Return *the shortest such subarray and output its length*.

Example 1:

```
Input: nums = [2,6,4,8,10,9,15]
Output: 5
Explanation: You need to sort [6, 4, 8, 10, 9] in ascending order to make the whole array sorted in ascending order.
```

Example 2:

```
Input: nums = [1,2,3,4]
Output: 0
```

Example 3:

```
Input: nums = [1]
Output: 0
```

Constraints:

- `1 <= nums.length <= 104`
- `105 <= nums[i] <= 105`

题目大意

给你一个整数数组 nums ，你需要找出一个 连续子数组 ，如果对这个子数组进行升序排序，那么整个数组都会变为升序排序。请你找出符合题意的 最短 子数组，并输出它的长度。

解题思路

- 本题求的是最短逆序区间。经过简单推理，可以知道，这个逆序区间一定由这个区间内的最小元素决定左边界，最大元素决定右边界。
- 先从左边找到第一个降序的元素，并记录最小的元素 min，再从右边往左找到最右边开始降序的元素，并记录最大的元素 max。最后需要还原最小元素和最大元素在原数组中正确的位置。以逆序区间左边界为例，如果区间外的一个元素比这个逆序区间内的最小元素还要小，说明它并不是左边界，因为这个小元素和逆序区间内的最小元素组合在一起，还是升序，并不是逆序。只有在左边区间外找到第一个大于逆序区间内最小元素，说明这里刚刚开始发生逆序，这才是最小逆序区间的左边界。同理，在逆序区间的右边找到第一个小于逆序区间内最大元素，说明这里刚刚发生逆序，这才是最小逆序区间的右边界。至此，最小逆序区间的左右边界都确定下来了，最短长度也就确定了下来。时间复杂度 O(n)，空间复杂度 O(1)。

代码

```go
package leetcode
```

```go
import "math"

func findUnsortedSubarray(nums []int) int {
  n, left, right, minR, maxL, isSort := len(nums), -1, -1, math.MaxInt32,
math.MinInt32, false
  // left
  for i := 1; i < n; i++ {
    if nums[i] < nums[i-1] {
      isSort = true
    }
    if isSort {
      minR = min(minR, nums[i])
    }
  }
  isSort = false
  // right
  for i := n - 2; i >= 0; i-- {
    if nums[i] > nums[i+1] {
      isSort = true
    }
    if isSort {
      maxL = max(maxL, nums[i])
    }
  }
  // minR
  for i := 0; i < n; i++ {
    if nums[i] > minR {
      left = i
      break
    }
  }
  // maxL
  for i := n - 1; i >= 0; i-- {
    if nums[i] < maxL {
      right = i
      break
    }
  }
  if left == -1 || right == -1 {
    return 0
  }
  return right - left + 1
}

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

```
}

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

583. Delete Operation for Two Strings

题目

Given two strings `word1` and `word2`, return *the minimum number of **steps** required to make* `word1` *and* `word2` *the same.*

In one **step**, you can delete exactly one character in either string.

Example 1:

```
Input: word1 = "sea", word2 = "eat"
Output: 2
Explanation: You need one step to make "sea" to "ea" and another step to make "eat" to
"ea".
```

Example 2:

```
Input: word1 = "leetcode", word2 = "etco"
Output: 4
```

Constraints:

- `1 <= word1.length, word2.length <= 500`
- `word1` and `word2` consist of only lowercase English letters.

题目大意

给定两个单词 word1 和 word2，找到使得 word1 和 word2 相同所需的最小步数，每步可以删除任意一个字符串中的一个字符。

解题思路

- 从题目数据量级判断，此题一定是 O(n^2) 动态规划题。定义 `dp[i][j]` 表示 `word1[:i]` 与 `word2[:j]` 匹配所删除的最少步数。如果 `word1[:i-1]` 与 `word2[:j-1]` 匹配，那么 `dp[i][j] = dp[i-1][j-1]`。如果 `word1[:i-1]` 与 `word2[:j-1]` 不匹配，那么需要考虑删除一次，所以 `dp[i][j] = 1 + min(dp[i][j-1], dp[i-1][j])`。所以动态转移方程是：

{{< katex display >}}
dp[i][j] = \left{\begin{matrix}dp[i-1][j-1]&, word1[i-1] == word2[j-1]\ 1 + min(dp[i][j-1], dp[i-1][j])&, word1[i-1] \neq word2[j-1]\\end{matrix}\right.
{{< /katex >}}

最终答案存储在 `dp[len(word1)][len(word2)]` 中。

代码

```
package leetcode

func minDistance(word1 string, word2 string) int {
  dp := make([][]int, len(word1)+1)
  for i := 0; i < len(word1)+1; i++ {
    dp[i] = make([]int, len(word2)+1)
  }
  for i := 0; i < len(word1)+1; i++ {
    dp[i][0] = i
  }
  for i := 0; i < len(word2)+1; i++ {
    dp[0][i] = i
  }
  for i := 1; i < len(word1)+1; i++ {
    for j := 1; j < len(word2)+1; j++ {
      if word1[i-1] == word2[j-1] {
        dp[i][j] = dp[i-1][j-1]
      } else {
        dp[i][j] = 1 + min(dp[i][j-1], dp[i-1][j])
      }
    }
  }
  return dp[len(word1)][len(word2)]
}

func min(x, y int) int {
  if x < y {
    return x
  }
  return y
}
```

589. N-ary Tree Preorder Traversal

题目

Given the `root` of an n-ary tree, return *the preorder traversal of its nodes' values*.

Nary-Tree input serialization is represented in their level order traversal. Each group of children is separated by the null value (See examples)

Example 1:

```
Input: root = [1,null,3,2,4,null,5,6]
Output: [1,3,5,6,2,4]
```

Example 2:

```
Input: root =
[1,null,2,3,4,5,null,null,6,7,null,8,null,9,10,null,null,11,null,12,null,13,null,null,1
4]
Output: [1,2,3,6,7,11,14,4,8,12,5,9,13,10]
```

Constraints:

- The number of nodes in the tree is in the range `[0, 104]`.
- `0 <= Node.val <= 10^4`
- The height of the n-ary tree is less than or equal to `1000`.

Follow up: Recursive solution is trivial, could you do it iteratively?

题目大意

给定一个 N 叉树，返回其节点值的 **前序遍历** 。N 叉树 在输入中按层序遍历进行序列化表示，每组子节点由空值 `null` 分隔（请参见示例）。

解题思路

- N 叉树和二叉树的前序遍历原理完全一样。二叉树非递归解法需要用到栈辅助，N 叉树同样如此。将父节点的所有孩子节点**逆序**入栈，逆序的目的是为了让前序节点永远在栈顶。依次循环直到栈里所有元素都出栈。输出的结果即为 N 叉树的前序遍历。时间复杂度 O(n)，空间复杂度 O(n)。
- 递归解法非常简单，见解法二。

代码

```
package leetcode

//  Definition for a Node.
type Node struct {
  Val      int
  Children []*Node
}

// 解法一 非递归
func preorder(root *Node) []int {
  res := []int{}
  if root == nil {
    return res
  }
  stack := []*Node{root}
  for len(stack) > 0 {
    r := stack[len(stack)-1]
    stack = stack[:len(stack)-1]
    res = append(res, r.Val)
    tmp := []*Node{}
    for _, v := range r.Children {
      tmp = append([]*Node{v}, tmp...) // 逆序存点
    }
    stack = append(stack, tmp...)
  }
  return res
}

// 解法二 递归
func preorder1(root *Node) []int {
  res := []int{}
  preorderdfs(root, &res)
  return res
}

func preorderdfs(root *Node, res *[]int) {
  if root != nil {
    *res = append(*res, root.Val)
    for i := 0; i < len(root.Children); i++ {
      preorderdfs(root.Children[i], res)
    }
  }
```

```
}
```

594. Longest Harmonious Subsequence

题目

We define a harmounious array as an array where the difference between its maximum value and its minimum value is **exactly** 1.

Now, given an integer array, you need to find the length of its longest harmonious subsequence among all its possible subsequences.

Example 1:

```
Input: [1,3,2,2,5,2,3,7]
Output: 5
Explanation: The longest harmonious subsequence is [3,2,2,2,3].
```

Note: The length of the input array will not exceed 20,000.

题目大意

和谐数组是指一个数组里元素的最大值和最小值之间的差别正好是1。现在，给定一个整数数组，你需要在所有可能的子序列中找到最长的和谐子序列的长度。说明: 输入的数组长度最大不超过20,000.

解题思路

- 在给出的数组里面找到这样一个子数组：要求子数组中的最大值和最小值相差 1 。这一题是简单题。先统计每个数字出现的频次，然后在 map 找相差 1 的 2 个数组的频次和，动态的维护两个数的频次和就是最后要求的子数组的最大长度。

代码

```go
package leetcode

func findLHS(nums []int) int {
  if len(nums) < 2 {
    return 0
  }
  res := make(map[int]int, len(nums))
  for _, num := range nums {
    if _, exist := res[num]; exist {
      res[num]++
      continue
    }
    res[num] = 1
  }
```

```
  longest := 0
  for k, c := range res {
    if n, exist := res[k+1]; exist {
      if c+n > longest {
        longest = c + n
      }
    }
  }
  return longest
}
```

598. Range Addition II

题目

Given an m * n matrix **M** initialized with all **0**'s and several update operations.

Operations are represented by a 2D array, and each operation is represented by an array with two **positive** integers **a** and **b**, which means **M[i][j]** should be **added by one** for all **0 <= i < a** and **0 <= j < b**.

You need to count and return the number of maximum integers in the matrix after performing all the operations.

Example 1:

```
Input:
m = 3, n = 3
operations = [[2,2],[3,3]]
Output: 4
Explanation:
Initially, M =
[[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]]

After performing [2,2], M =
[[1, 1, 0],
 [1, 1, 0],
 [0, 0, 0]]

After performing [3,3], M =
[[2, 2, 1],
 [2, 2, 1],
 [1, 1, 1]]

So the maximum integer in M is 2, and there are four of it in M. So return 4.
```

Note:

1. The range of m and n is [1,40000].
2. The range of a is [1,m], and the range of b is [1,n].
3. The range of operations size won't exceed 10,000.

题目大意

给定一个初始元素全部为 0，大小为 m*n 的矩阵 M 以及在 M 上的一系列更新操作。操作用二维数组表示，其中的每个操作用一个含有两个正整数 a 和 b 的数组表示，含义是将所有符合 0 <= i < a 以及 0 <= j < b 的元素 M[i][j] 的值都增加 1。在执行给定的一系列操作后，你需要返回矩阵中含有最大整数的元素个数。

注意:

- m 和 n 的范围是 [1,40000]。
- a 的范围是 [1,m]，b 的范围是 [1,n]。
- 操作数目不超过 10000。

解题思路

- 给定一个初始都为 0 的 m * n 的矩阵，和一个操作数组。经过一系列的操作以后，最终输出矩阵中最大整数的元素个数。每次操作都使得一个矩形内的元素都 + 1 。
- 这一题乍一看像线段树的区间覆盖问题，但是实际上很简单。如果此题是任意的矩阵，那就可能用到线段树了。这一题每个矩阵的起点都包含 [0 , 0] 这个元素，也就是说每次操作都会影响第一个元素。那么这道题就很简单了。经过 n 次操作以后，被覆盖次数最多的矩形区间，一定就是最大整数所在的区间。由于起点都是第一个元素，所以我们只用关心矩形的右下角那个坐标。右下角怎么计算呢？只用每次动态的维护一下矩阵长和宽的最小值即可。

代码

```go
package leetcode

func maxCount(m int, n int, ops [][]int) int {
  minM, minN := m, n
  for _, op := range ops {
    minM = min(minM, op[0])
    minN = min(minN, op[1])
  }
  return minM * minN
}

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

599. Minimum Index Sum of Two Lists

题目

Suppose Andy and Doris want to choose a restaurant for dinner, and they both have a list of favorite restaurants represented by strings.

You need to help them find out their **common interest** with the **least list index sum**. If there is a choice tie between answers, output all of them with no order requirement. You could assume there always exists an answer.

Example 1:

```
Input:
["Shogun", "Tapioca Express", "Burger King", "KFC"]
["Piatti", "The Grill at Torrey Pines", "Hungry Hunter Steakhouse", "Shogun"]
Output: ["Shogun"]
Explanation: The only restaurant they both like is "Shogun".
```

Example 2:

```
Input:
["Shogun", "Tapioca Express", "Burger King", "KFC"]
["KFC", "Shogun", "Burger King"]
Output: ["Shogun"]
Explanation: The restaurant they both like and have the least index sum is "Shogun"
with index sum 1 (0+1).
```

Note:

1. The length of both lists will be in the range of [1, 1000].
2. The length of strings in both lists will be in the range of [1, 30].
3. The index is starting from 0 to the list length minus 1.
4. No duplicates in both lists.

题目大意

假设 Andy 和 Doris 想在晚餐时选择一家餐厅，并且他们都有一个表示最喜爱餐厅的列表，每个餐厅的名字用字符串表示。你需要帮助他们用最少的索引和找出他们共同喜爱的餐厅。 如果答案不止一个，则输出所有答案并且不考虑顺序。 你可以假设总是存在一个答案。

提示:

- 两个列表的长度范围都在 [1, 1000] 内。
- 两个列表中的字符串的长度将在 [1，30] 的范围内。
- 下标从 0 开始，到列表的长度减 1。
- 两个列表都没有重复的元素。

解题思路

- 在 Andy 和 Doris 两人分别有各自的餐厅喜欢列表，要求找出两人公共喜欢的一家餐厅，如果共同喜欢的次数相同，都输出。这一题是简单题，用 map 统计频次，输出频次最多的餐厅。

代码

```go
package leetcode

func findRestaurant(list1 []string, list2 []string) []string {
  m, ans := make(map[string]int, len(list1)), []string{}
  for i, r := range list1 {
    m[r] = i
  }
  for j, r := range list2 {
    if _, ok := m[r]; ok {
      m[r] += j
      if len(ans) == 0 || m[r] == m[ans[0]] {
        ans = append(ans, r)
      } else if m[r] < m[ans[0]] {
        ans = []string{r}
      }
    }
  }
  return ans
}
```

605. Can Place Flowers

题目

You have a long flowerbed in which some of the plots are planted, and some are not. However, flowers cannot be planted in **adjacent** plots.

Given an integer array `flowerbed` containing `0`'s and `1`'s, where `0` means empty and `1` means not empty, and an integer `n`, return *if* `n` new flowers can be planted in the `flowerbed` without violating the no-adjacent-flowers rule.

Example 1:

```
Input: flowerbed = [1,0,0,0,1], n = 1
Output: true
```

Example 2:

```
Input: flowerbed = [1,0,0,0,1], n = 2
Output: false
```

Constraints:

- `1 <= flowerbed.length <= 2 * 104`
- `flowerbed[i]` is `0` or `1`.
- There are no two adjacent flowers in `flowerbed`.
- `0 <= n <= flowerbed.length`

题目大意

假设你有一个很长的花坛，一部分地块种植了花，另一部分却没有。可是，花卉不能种植在相邻的地块上，它们会争夺水源，两者都会死去。给定一个花坛（表示为一个数组包含0和1，其中0表示没种植花，1表示种植了花），和一个数 n 。能否在不打破种植规则的情况下种入 n 朵花？能则返回True，不能则返回False。

解题思路

- 这一题最容易想到的解法是步长为 2 遍历数组，依次计数 0 的个数。有 2 种特殊情况需要单独判断，第一种情况是首尾连续多个 0，例如，00001 和 10000，第二种情况是 2 个 1 中间存在的 0 不足以种花，例如，1001 和 100001，1001 不能种任何花，100001 只能种一种花。单独判断出这 2 种情况，这一题就可以 AC 了。
- 换个思路，找到可以种花的基本单元是 00，那么上面那 2 种特殊情况都可以统一成一种情况。判断是否当前存在 00 的组合，如果存在 00 的组合，都可以种花。末尾的情况需要单独判断，如果末尾为 0，也可以种花。这个时候不需要再找 00 组合，因为会越界。代码实现如下，思路很简洁明了。

代码

```go
package leetcode

func canPlaceFlowers(flowerbed []int, n int) bool {
  lenth := len(flowerbed)
  for i := 0; i < lenth && n > 0; i += 2 {
    if flowerbed[i] == 0 {
      if i+1 == lenth || flowerbed[i+1] == 0 {
        n--
      } else {
        i++
      }
    }
  }
  if n == 0 {
    return true
  }
  return false
}
```

609. Find Duplicate File in System

题目

Given a list `paths` of directory info, including the directory path, and all the files with contents in this directory, return *all the duplicate files in the file system in terms of their paths*. You may return the answer in **any order**.

A group of duplicate files consists of at least two files that have the same content.

A single directory info string in the input list has the following format:

- `"root/d1/d2/.../dm f1.txt(f1_content) f2.txt(f2_content) ... fn.txt(fn_content)"`

It means there are `n` files `(f1.txt, f2.txt ... fn.txt)` with content `(f1_content, f2_content ... fn_content)` respectively in the directory "`root/d1/d2/.../dm`". Note that `n >= 1` and `m >= 0`. If `m = 0`, it means the directory is just the root directory.

The output is a list of groups of duplicate file paths. For each group, it contains all the file paths of the files that have the same content. A file path is a string that has the following format:

- `"directory_path/file_name.txt"`

Example 1:

```
Input: paths = ["root/a 1.txt(abcd) 2.txt(efgh)","root/c 3.txt(abcd)","root/c/d 4.txt(efgh)","root 4.txt(efgh)"]
Output: [["root/a/2.txt","root/c/d/4.txt","root/4.txt"],["root/a/1.txt","root/c/3.txt"]]
```

Example 2:

```
Input: paths = ["root/a 1.txt(abcd) 2.txt(efgh)","root/c 3.txt(abcd)","root/c/d 4.txt(efgh)"]
Output: [["root/a/2.txt","root/c/d/4.txt"],["root/a/1.txt","root/c/3.txt"]]
```

Constraints:

- `1 <= paths.length <= 2 * 104`
- `1 <= paths[i].length <= 3000`
- `1 <= sum(paths[i].length) <= 5 * 105`
- `paths[i]` consist of English letters, digits, `'/'`, `'.'`, `'('`, `')'`, and `' '`.
- You may assume no files or directories share the same name in the same directory.
- You may assume each given directory info represents a unique directory. A single blank space separates the directory path and file info.

Follow up:

- Imagine you are given a real file system, how will you search files? DFS or BFS?
- If the file content is very large (GB level), how will you modify your solution?
- If you can only read the file by 1kb each time, how will you modify your solution?
- What is the time complexity of your modified solution? What is the most time-consuming part and memory-consuming part of it? How to optimize?
- How to make sure the duplicated files you find are not false positive?

题目大意

给定一个目录信息列表，包括目录路径，以及该目录中的所有包含内容的文件，您需要找到文件系统中的所有重复文件组的路径。一组重复的文件至少包括二个具有完全相同内容的文件。输入列表中的单个目录信息字符串的格式如下：`"root/d1/d2/.../dm f1.txt(f1_content) f2.txt(f2_content) ... fn.txt(fn_content)"`。这意味着有 n 个文件（`f1.txt, f2.txt ... fn.txt` 的内容分别是 `f1_content, f2_content ... fn_content`）在目录 `root/d1/d2/.../dm` 下。注意：n>=1 且 m>=0。如果 m=0，则表示该目录是根目录。该输出是重复文件路径组的列表。对于每个组，它包含具有相同内容的文件的所有文件路径。文件路径是具有下列格式的字符串：`"directory_path/file_name.txt"`

解题思路

- 这一题算简单题，考察的是字符串基本操作与 map 的使用。首先通过字符串操作获取目录路径、文件名和文件内容。再使用 map 来寻找重复文件，key 是文件内容，value 是存储路径和文件名的列表。遍历每一个文件，并把它加入 map 中。最后遍历 map，如果一个键对应的值列表的长度大于 1，说明找到了重复文件，可以把这个列表加入到最终答案中。
- 这道题有价值的地方在 **Follow up** 中。感兴趣的读者可以仔细想想以下几个问题：
  1. 假设您有一个真正的文件系统，您将如何搜索文件？广度搜索还是宽度搜索？
  2. 如果文件内容非常大（GB级别），您将如何修改您的解决方案？
  3. 如果每次只能读取 1 kb 的文件，您将如何修改解决方案？
  4. 修改后的解决方案的时间复杂度是多少？其中最耗时的部分和消耗内存的部分是什么？如何优化？
  5. 如何确保您发现的重复文件不是误报？

代码

```go
package leetcode

import "strings"

func findDuplicate(paths []string) [][]string {
  cache := make(map[string][]string)
  for _, path := range paths {
    parts := strings.Split(path, " ")
    dir := parts[0]
    for i := 1; i < len(parts); i++ {
      bracketPosition := strings.IndexByte(parts[i], '(')
      content := parts[i][bracketPosition+1 : len(parts[i])-1]
      cache[content] = append(cache[content], dir+"/"+parts[i][:bracketPosition])
    }
  }
```

```
  res := make([][]string, 0, len(cache))
  for _, group := range cache {
    if len(group) >= 2 {
      res = append(res, group)
    }
  }
  return res
}
```

622. Design Circular Queue

题目

Design your implementation of the circular queue. The circular queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called "Ring Buffer".

One of the benefits of the circular queue is that we can make use of the spaces in front of the queue. In a normal queue, once the queue becomes full, we cannot insert the next element even if there is a space in front of the queue. But using the circular queue, we can use the space to store new values.

Implementation the `MyCircularQueue` class:

- `MyCircularQueue(k)` Initializes the object with the size of the queue to be `k`.
- `int Front()` Gets the front item from the queue. If the queue is empty, return `1`.
- `int Rear()` Gets the last item from the queue. If the queue is empty, return `1`.
- `boolean enQueue(int value)` Inserts an element into the circular queue. Return `true` if the operation is successful.
- `boolean deQueue()` Deletes an element from the circular queue. Return `true` if the operation is successful.
- `boolean isEmpty()` Checks whether the circular queue is empty or not.
- `boolean isFull()` Checks whether the circular queue is full or not.

Example 1:

```
Input
["MyCircularQueue", "enQueue", "enQueue", "enQueue", "enQueue", "Rear", "isFull",
"deQueue", "enQueue", "Rear"]
[[3], [1], [2], [3], [4], [], [], [], [4], []]
Output
[null, true, true, true, false, 3, true, true, true, 4]

Explanation
MyCircularQueue myCircularQueue = new MyCircularQueue(3);
myCircularQueue.enQueue(1); // return True
myCircularQueue.enQueue(2); // return True
myCircularQueue.enQueue(3); // return True
myCircularQueue.enQueue(4); // return False
```

```
myCircularQueue.Rear();     // return 3
myCircularQueue.isFull();   // return True
myCircularQueue.deQueue();  // return True
myCircularQueue.enQueue(4); // return True
myCircularQueue.Rear();     // return 4
```

Constraints:

- `1 <= k <= 1000`
- `0 <= value <= 1000`
- At most `3000` calls will be made to `enQueue`, `deQueue`, `Front`, `Rear`, `isEmpty`, and `isFull`.

Follow up:

Could you solve the problem without using the built-in queue?

题目大意

设计你的循环队列实现。 循环队列是一种线性数据结构，其操作表现基于 FIFO（先进先出）原则并且队尾被连接在队首之后以形成一个循环。它也被称为“环形缓冲器”。

循环队列的一个好处是我们可以利用这个队列之前用过的空间。在一个普通队列里，一旦一个队列满了，我们就不能插入下一个元素，即使在队列前面仍有空间。但是使用循环队列，我们能使用这些空间去存储新的值。

你的实现应该支持如下操作：

- MyCircularQueue(k): 构造器，设置队列长度为 k 。
- Front: 从队首获取元素。如果队列为空，返回 -1 。
- Rear: 获取队尾元素。如果队列为空，返回 -1 。
- enQueue(value): 向循环队列插入一个元素。如果成功插入则返回真。
- deQueue(): 从循环队列中删除一个元素。如果成功删除则返回真。
- isEmpty(): 检查循环队列是否为空。
- isFull(): 检查循环队列是否已满。

解题思路

- 简单题。设计一个环形队列，底层用数组实现。额外维护 4 个变量，队列的总 cap，队列当前的 size，前一元素下标 left，后一个元素下标 right。每添加一个元素便维护 left，right，size，下标需要对 cap 取余，因为超过 cap 大小之后，需要循环存储。代码实现没有难度，具体sh见下面代码。

代码

```go
package leetcode

type MyCircularQueue struct {
  cap   int
  size  int
  queue []int
  left  int
```

```go
  right int
}

func Constructor(k int) MyCircularQueue {
  return MyCircularQueue{cap: k, size: 0, left: 0, right: 0, queue: make([]int, k)}
}

func (this *MyCircularQueue) EnQueue(value int) bool {
  if this.size == this.cap {
    return false
  }
  this.size++
  this.queue[this.right] = value
  this.right++
  this.right %= this.cap
  return true

}

func (this *MyCircularQueue) DeQueue() bool {
  if this.size == 0 {
    return false
  }
  this.size--
  this.left++
  this.left %= this.cap
  return true
}

func (this *MyCircularQueue) Front() int {
  if this.size == 0 {
    return -1
  }
  return this.queue[this.left]
}

func (this *MyCircularQueue) Rear() int {
  if this.size == 0 {
    return -1
  }
  if this.right == 0 {
    return this.queue[this.cap-1]
  }
  return this.queue[this.right-1]
}

func (this *MyCircularQueue) IsEmpty() bool {
  return this.size == 0
}
```

```go
func (this *MyCircularQueue) IsFull() bool {
  return this.size == this.cap
}

/**
 * Your MyCircularQueue object will be instantiated and called as such:
 * obj := Constructor(k);
 * param_1 := obj.EnQueue(value);
 * param_2 := obj.DeQueue();
 * param_3 := obj.Front();
 * param_4 := obj.Rear();
 * param_5 := obj.IsEmpty();
 * param_6 := obj.IsFull();
 */
```

623. Add One Row to Tree

题目

Given the root of a binary tree, then value `v` and depth `d`, you need to add a row of nodes with value `v` at the given depth `d`. The root node is at depth 1.

The adding rule is: given a positive integer depth `d`, for each NOT null tree nodes `N` in depth `d-1`, create two tree nodes with value `v` as `N's` left subtree root and right subtree root. And `N's` **original left subtree** should be the left subtree of the new left subtree root, its **original right subtree** should be the right subtree of the new right subtree root. If depth `d` is 1 that means there is no depth d-1 at all, then create a tree node with value **v** as the new root of the whole original tree, and the original tree is the new root's left subtree.

Example 1:

```
Input:
A binary tree as following:
       4
     /   \
    2     6
   / \   /
  3   1 5

v = 1d = 2Output:
       4
      / \
     1   1
    /     \
   2       6
  / \     /
 3   1   5
```

Example 2:

```
Input:
A binary tree as following:
       4
      /
     2
    / \
   3   1

v = 1d = 3Output:
       4
      /
     2
    / \
   1   1
  /     \
 3       1
```

Note:

1. The given d is in range [1, maximum depth of the given tree + 1].
2. The given binary tree has at least one tree node.

题目大意

给定一个二叉树，根节点为第1层，深度为 1。在其第 d 层追加一行值为 v 的节点。添加规则：给定一个深度值 d（正整数），针对深度为 d-1 层的每一非空节点 N，为 N 创建两个值为 v 的左子树和右子树。将 N 原先的左子树，连接为新节点 v 的左子树；将 N 原先的右子树，连接为新节点 v 的右子树。如果 d 的值为 1，深度 d - 1 不存在，则创建一个新的根节点 v，原先的整棵树将作为 v 的左子树。

解题思路

- 这一题虽然是 Medium，实际非常简单。给二叉树添加一行，用 DFS 或者 BFS，遍历过程中记录行数，到达目标行一行，增加节点即可。不过需要注意 2 个特殊情况，特殊情况一，`d==1`，此时需要添加的行即为根节点。特殊情况二，`d>height(root)`，即要添加的行数比树还要高，这时只需要在最下层的叶子节点添加一层。时间复杂度 O(n)，空间复杂度 O(n)。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/structures"
)

// TreeNode define
type TreeNode = structures.TreeNode
```

```go
/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func addOneRow(root *TreeNode, v int, d int) *TreeNode {
  if d == 1 {
    tmp := &TreeNode{Val: v, Left: root, Right: nil}
    return tmp
  }
  level := 1
  addTreeRow(root, v, d, &level)
  return root
}

func addTreeRow(root *TreeNode, v, d int, currLevel *int) {
  if *currLevel == d-1 {
    root.Left = &TreeNode{Val: v, Left: root.Left, Right: nil}
    root.Right = &TreeNode{Val: v, Left: nil, Right: root.Right}
    return
  }
  *currLevel++
  if root.Left != nil {
    addTreeRow(root.Left, v, d, currLevel)
  }
  if root.Right != nil {
    addTreeRow(root.Right, v, d, currLevel)
  }
  *currLevel--
}
```

628. Maximum Product of Three Numbers

题目

Given an integer array, find three numbers whose product is maximum and output the maximum product.

Example 1:

```
Input: [1,2,3]
Output: 6
```

Example 2:

```
Input: [1,2,3,4]
Output: 24
```

Note:

1. The length of the given array will be in range [3,10^4] and all elements are in the range [-1000, 1000].
2. Multiplication of any three numbers in the input won't exceed the range of 32-bit signed integer.

题目大意

给定一个整型数组，在数组中找出由三个数组成的最大乘积，并输出这个乘积。

解题思路

- 给出一个数组，要求求出这个数组中任意挑 3 个数能组成的乘积最大的值。
- 题目的 test case 数据量比较大，如果用排序的话，时间复杂度高，可以直接考虑模拟，挑出 3 个数组成乘积最大值，必然是一个正数和二个负数，或者三个正数。那么选出最大的三个数和最小的二个数，对比一下就可以求出最大值了。时间复杂度 O(n)

代码

```go
package leetcode

import (
  "math"
  "sort"
)

// 解法一 排序，时间复杂度 O(n log n)
func maximumProduct(nums []int) int {
  if len(nums) == 0 {
    return 0
  }
  res := 1
  if len(nums) <= 3 {
    for i := 0; i < len(nums); i++ {
      res = res * nums[i]
    }
    return res
  }
  sort.Ints(nums)
  if nums[len(nums)-1] <= 0 {
    return 0
  }
```

```go
  return max(nums[0]*nums[1]*nums[len(nums)-1],
nums[len(nums)-1]*nums[len(nums)-2]*nums[len(nums)-3])
}

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

// 解法二 模拟，时间复杂度 O(n)
func maximumProduct1(nums []int) int {
  max := make([]int, 0)
  max = append(max, math.MinInt64, math.MinInt64, math.MinInt64)
  min := make([]int, 0)
  min = append(min, math.MaxInt64, math.MaxInt64)
  for _, num := range nums {
    if num > max[0] {
      max[0], max[1], max[2] = num, max[0], max[1]
    } else if num > max[1] {
      max[1], max[2] = num, max[1]
    } else if num > max[2] {
      max[2] = num
    }
    if num < min[0] {
      min[0], min[1] = num, min[0]
    } else if num < min[1] {
      min[1] = num
    }
  }
  maxProduct1, maxProduct2 := min[0]*min[1]*max[0], max[0]*max[1]*max[2]
  if maxProduct1 > maxProduct2 {
    return maxProduct1
  }
  return maxProduct2
}
```

630. Course Schedule III

题目

There are `n` different online courses numbered from `1` to `n`. You are given an array `courses` where `courses[i] = [durationi, lastDayi]` indicate that the `ith` course should be taken **continuously** for `durationi` days and must be finished before or on `lastDayi`.

You will start on the `1st` day and you cannot take two or more courses simultaneously.

Return *the maximum number of courses that you can take.*

Example 1:

```
Input: courses = [[100,200],[200,1300],[1000,1250],[2000,3200]]
Output: 3
Explanation:
There are totally 4 courses, but you can take 3 courses at most:
First, take the 1st course, it costs 100 days so you will finish it on the 100th day,
and ready to take the next course on the 101st day.
Second, take the 3rd course, it costs 1000 days so you will finish it on the 1100th
day, and ready to take the next course on the 1101st day.
Third, take the 2nd course, it costs 200 days so you will finish it on the 1300th day.
The 4th course cannot be taken now, since you will finish it on the 3300th day, which
exceeds the closed date.
```

Example 2:

```
Input: courses = [[1,2]]
Output: 1
```

Example 3:

```
Input: courses = [[3,2],[4,3]]
Output: 0
```

Constraints:

- `1 <= courses.length <= 104`
- `1 <= durationi, lastDayi <= 104`

题目大意

这里有 n 门不同的在线课程，他们按从 1 到 n 编号。每一门课程有一定的持续上课时间（课程时间）t 以及关闭时间第 d 天。一门课要持续学习 t 天直到第 d 天时要完成，你将会从第 1 天开始。给出 n 个在线课程用 (t, d) 对表示。你的任务是找出最多可以修几门课。

解题思路

- 一般选课，任务的题目会涉及排序 + 贪心。此题同样如此。最多修几门课，采用贪心的思路。先将课程结束时间从小到大排序，优先选择结束时间靠前的课程，这样留给后面课程的时间越多，便可以修更多的课。对排好序的课程从前往后选课，不断累积时间。如果选择修当前课程，但是会超时，这时改调整了。对于已经选择的课程，都加入到最大堆中，遇到需要调整时，比较当前待考虑的课程时长是否比(堆中)已经选择课中时长最长的课时长短，即堆顶的课程时长短，剔除 pop 它，再选择这门时长短的课，并加入最大堆中。并更新累积时间。一层循环扫完所有课程，最终最大堆中包含课程的数目便是最多可以修的课程数。

代码

```go
package leetcode

import (
  "container/heap"
  "sort"
)

func scheduleCourse(courses [][]int) int {
  sort.Slice(courses, func(i, j int) bool {
    return courses[i][1] < courses[j][1]
  })
  maxHeap, time := &Schedule{}, 0
  heap.Init(maxHeap)
  for _, c := range courses {
    if time+c[0] <= c[1] {
      time += c[0]
      heap.Push(maxHeap, c[0])
    } else if (*maxHeap).Len() > 0 && (*maxHeap)[0] > c[0] {
      time -= heap.Pop(maxHeap).(int) - c[0]
      heap.Push(maxHeap, c[0])
    }
  }
  return (*maxHeap).Len()
}

type Schedule []int

func (s Schedule) Len() int           { return len(s) }
func (s Schedule) Less(i, j int) bool { return s[i] > s[j] }
func (s Schedule) Swap(i, j int)      { s[i], s[j] = s[j], s[i] }
func (s *Schedule) Pop() interface{} {
  n := len(*s)
  t := (*s)[n-1]
  *s = (*s)[:n-1]
  return t
}
func (s *Schedule) Push(x interface{}) {
  *s = append(*s, x.(int))
}
```

632. Smallest Range Covering Elements from K Lists

题目

You have `k` lists of sorted integers in ascending order. Find the **smallest** range that includes at least one number from each of the `k` lists.

We define the range [a,b] is smaller than range [c,d] if `b-a < d-c` or `a < c` if `b-a == d-c`.

Example 1:

```
Input: [[4,10,15,24,26], [0,9,12,20], [5,18,22,30]]
Output: [20,24]
Explanation:
List 1: [4, 10, 15, 24,26], 24 is in range [20,24].
List 2: [0, 9, 12, 20], 20 is in range [20,24].
List 3: [5, 18, 22, 30], 22 is in range [20,24].
```

Note:

1. The given list may contain duplicates, so ascending order means >= here.
2. 1 <= `k` <= 3500
3. -10^5 <= `value of elements` <= 10^5.

题目大意

你有 k 个升序排列的整数数组。找到一个最小区间，使得 k 个列表中的每个列表至少有一个数包含在其中。

我们定义如果 b-a < d-c 或者在 b-a == d-c 时 a < c，则区间 [a,b] 比 [c,d] 小。

注意:

- 给定的列表可能包含重复元素，所以在这里升序表示 >= 。
- 1 <= k <= 3500
- -105 <= 元素的值 <= 105
- 对于使用Java的用户，请注意传入类型已修改为List<List>。重置代码模板后可以看到这项改动。

解题思路

- 给出 K 个数组，要求在这 K 个数组中找到一个区间，至少能包含这 K 个数组中每个数组中的一个元素。
- 这一题是第 76 题的变种版。第 76 题是用滑动窗口来解答的，它要求在母字符串 S 中找到最小的子串能包含 T 串的所有字母。这一题类似的，可以把母字符串看成 K 个数组合并起来的大数组，那么 T 串是由 K 个数组中每个数组中抽一个元素出来组成的。求的区间相同，都是能包含 T 的最小区间。另外一个区别在于，第 76 题里面都是字符串，这一题都是数字，在最终拼接成 T 串的时候需要保证 K 个数组中每个都有一个元素，所以理所当然的想到需要维护每个元素所在数组编号。经过上述的转换，可以把这道题转换成第 76 题的解法了。
- 在具体解题过程中，用 map 来维护窗口内 K 个数组出现的频次。时间复杂度 O(n*log n)，空间复杂度是 O(n)。

代码

```go
package leetcode

import (
  "math"
  "sort"
)

func smallestRange(nums [][]int) []int {
  numList, left, right, count, freqMap, res, length := []element{}, 0, -1, 0,
map[int]int{}, make([]int, 2), math.MaxInt64
  for i, ns := range nums {
    for _, v := range ns {
      numList = append(numList, element{val: v, index: i})
    }
  }
  sort.Sort(SortByVal{numList})
  for left < len(numList) {
    if right+1 < len(numList) && count < len(nums) {
      right++
      if freqMap[numList[right].index] == 0 {
        count++
      }
      freqMap[numList[right].index]++
    } else {
      if count == len(nums) {
        if numList[right].val-numList[left].val < length {
          length = numList[right].val - numList[left].val
          res[0] = numList[left].val
          res[1] = numList[right].val
        }
      }
      freqMap[numList[left].index]--
      if freqMap[numList[left].index] == 0 {
        count--
      }
      left++
    }
  }
  return res
}

type element struct {
  val   int
  index int
}
```

```go
type elements []element

// Len define
func (p elements) Len() int { return len(p) }

// Swap define
func (p elements) Swap(i, j int) { p[i], p[j] = p[j], p[i] }

// SortByVal define
type SortByVal struct{ elements }

// Less define
func (p SortByVal) Less(i, j int) bool {
  return p.elements[i].val < p.elements[j].val
}
```

633. Sum of Square Numbers

题目

Given a non-negative integer `c`, your task is to decide whether there're two integers `a` and `b` such that a^2 + b^2 = c.

Example 1:

```
Input: 5
Output: True
Explanation: 1 * 1 + 2 * 2 = 5
```

Example 2:

```
Input: 3
Output: False
```

题目大意

给定一个非负整数 c ，你要判断是否存在两个整数 a 和 b，使得 a^2 + b^2 = c。

解题思路

- 给出一个数，要求判断这个数能否由由 2 个完全平方数组成。能则输出 true，不能则输出 false。
- 可以用二分搜索来解答这道题。判断题意，依次计算 `low * low + high * high` 和 c 是否相等。从 [0, sqrt(n)] 区间内进行二分，若能找到则返回 true，找不到就返回 false 。

代码

```go
package leetcode

import "math"

func judgeSquareSum(c int) bool {
  low, high := 0, int(math.Sqrt(float64(c)))
  for low <= high {
    if low*low+high*high < c {
      low++
    } else if low*low+high*high > c {
      high--
    } else {
      return true
    }
  }
  return false
}
```

636. Exclusive Time of Functions

题目

On a single threaded CPU, we execute some functions. Each function has a unique id between `0` and `N-1`.

We store logs in timestamp order that describe when a function is entered or exited.

Each log is a string with this format: `"{function_id}:{"start" | "end"}:{timestamp}"`. For example, `"0:start:3"` means the function with id `0` started at the beginning of timestamp `3`. `"1:end:2"` means the function with id `1` ended at the end of timestamp `2`.

A function's *exclusive time* is the number of units of time spent in this function. Note that this does not include any recursive calls to child functions.

Return the exclusive time of each function, sorted by their function id.

Example 1:

```
Input:
n = 2
logs = ["0:start:0","1:start:2","1:end:5","0:end:6"]
Output: [3, 4]
Explanation:
Function 0 starts at the beginning of time 0, then it executes 2 units of time and
reaches the end of time 1.
Now function 1 starts at the beginning of time 2, executes 4 units of time and ends at
time 5.
Function 0 is running again at the beginning of time 6, and also ends at the end of
time 6, thus executing for 1 unit of time.
So function 0 spends 2 + 1 = 3 units of total time executing, and function 1 spends 4
units of total time executing.
```

Note:

1. `1 <= n <= 100`
2. Two functions won't start or end at the same time.
3. Functions will always log when they exit.

题目大意

给出一个非抢占单线程CPU的 n 个函数运行日志，找到函数的独占时间。每个函数都有一个唯一的 Id，从 0 到 n-1，函数可能会递归调用或者被其他函数调用。

日志是具有以下格式的字符串：function_id：start_or_end：timestamp。例如："0:start:0" 表示函数 0 从 0 时刻开始运行。"0⬅END0" 表示函数 0 在 0 时刻结束。

函数的独占时间定义是在该方法中花费的时间，调用其他函数花费的时间不算该函数的独占时间。你需要根据函数的 Id 有序地返回每个函数的独占时间。

解题思路

- 利用栈记录每一个开始了但是未完成的任务，完成以后任务就 pop 一个。
- 注意题目中关于任务时长的定义，例如，start 7，end 7，这个任务执行了 1 秒而不是 0 秒

代码

```
package leetcode

import (
  "strconv"
  "strings"
)

type log struct {
  id    int
  order string
  time  int
}

func exclusiveTime(n int, logs []string) []int {
  res, lastLog, stack := make([]int, n), log{id: -1, order: "", time: 0}, []log{}
  for i := 0; i < len(logs); i++ {
    a := strings.Split(logs[i], ":")
    id, _ := strconv.Atoi(a[0])
    time, _ := strconv.Atoi(a[2])

    if (lastLog.order == "start" && a[1] == "start") || (lastLog.order == "start" &&
a[1] == "end") {
      res[lastLog.id] += time - lastLog.time
      if a[1] == "end" {
        res[lastLog.id]++
      }
    }
    if lastLog.order == "end" && a[1] == "end" {
      res[id] += time - lastLog.time
    }
    if lastLog.order == "end" && a[1] == "start" && len(stack) != 0 {
      res[stack[len(stack)-1].id] += time - lastLog.time - 1
    }
    if a[1] == "start" {
      stack = append(stack, log{id: id, order: a[1], time: time})
    } else {
      stack = stack[:len(stack)-1]
    }
    lastLog = log{id: id, order: a[1], time: time}
  }
  return res
}
```

637. Average of Levels in Binary Tree

题目

Given a non-empty binary tree, return the average value of the nodes on each level in the form of an array.

Example 1:

```
Input:
    3
   / \
  9  20
    /  \
   15   7
Output: [3, 14.5, 11]
Explanation:
The average value of nodes on level 0 is 3,  on level 1 is 14.5, and on level 2 is 11.
Hence return [3, 14.5, 11].
```

Note:

The range of node's value is in the range of 32-bit signed integer.

题目大意

按层序从上到下遍历一颗树，计算每一层的平均值。

解题思路

- 用一个队列即可实现。
- 第 102 题和第 107 题都是按层序遍历的。

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
```

```go
 * }
 */
func averageOfLevels(root *TreeNode) []float64 {
  if root == nil {
    return []float64{0}
  }
  queue := []*TreeNode{}
  queue = append(queue, root)
  curNum, nextLevelNum, res, count, sum := 1, 0, []float64{}, 1, 0
  for len(queue) != 0 {
    if curNum > 0 {
      node := queue[0]
      if node.Left != nil {
        queue = append(queue, node.Left)
        nextLevelNum++
      }
      if node.Right != nil {
        queue = append(queue, node.Right)
        nextLevelNum++
      }
      curNum--
      sum += node.Val
      queue = queue[1:]
    }
    if curNum == 0 {
      res = append(res, float64(sum)/float64(count))
      curNum, count, nextLevelNum, sum = nextLevelNum, nextLevelNum, 0, 0
    }
  }
  return res
}
```

638. Shopping Offers

题目

In LeetCode Store, there are some kinds of items to sell. Each item has a price.

However, there are some special offers, and a special offer consists of one or more different kinds of items with a sale price.

You are given the each item's price, a set of special offers, and the number we need to buy for each item. The job is to output the lowest price you have to pay for **exactly** certain items as given, where you could make optimal use of the special offers.

Each special offer is represented in the form of an array, the last number represents the price you need to pay for this special offer, other numbers represents how many specific items you could get if you buy this offer.

You could use any of special offers as many times as you want.

Example 1:

```
Input: [2,5], [[3,0,5],[1,2,10]], [3,2]
Output: 14
Explanation:
There are two kinds of items, A and B. Their prices are $2 and $5 respectively.
In special offer 1, you can pay $5 for 3A and 0B
In special offer 2, you can pay $10 for 1A and 2B.
You need to buy 3A and 2B, so you may pay $10 for 1A and 2B (special offer #2), and $4
for 2A.
```

Example 2:

```
Input: [2,3,4], [[1,1,0,4],[2,2,1,9]], [1,2,1]
Output: 11
Explanation:
The price of A is $2, and $3 for B, $4 for C.
You may pay $4 for 1A and 1B, and $9 for 2A ,2B and 1C.
You need to buy 1A ,2B and 1C, so you may pay $4 for 1A and 1B (special offer #1), and
$3 for 1B, $4 for 1C.
You cannot add more items, though only $9 for 2A ,2B and 1C.
```

Note:

1. There are at most 6 kinds of items, 100 special offers.
2. For each item, you need to buy at most 6 of them.
3. You are **not** allowed to buy more items than you want, even if that would lower the overall price.

题目大意

在 LeetCode 商店中， 有许多在售的物品。然而，也有一些大礼包，每个大礼包以优惠的价格捆绑销售一组物品。

现给定每个物品的价格，每个大礼包包含物品的清单，以及待购物品清单。请输出确切完成待购清单的最低花费。每个大礼包的由一个数组中的一组数据描述，最后一个数字代表大礼包的价格，其他数字分别表示内含的其他种类物品的数量。任意大礼包可无限次购买。

例子 1:

```
输入: [2,5], [[3,0,5],[1,2,10]], [3,2]
输出: 14
解释:
有A和B两种物品，价格分别为¥2和¥5。
大礼包1，你可以以¥5的价格购买3A和0B。
大礼包2，  你可以以¥10的价格购买1A和2B。
你需要购买3个A和2个B，  所以你付了¥10购买了1A和2B（大礼包2），以及¥4购买2A。
```

例子 2:

```
输入: [2,3,4], [[1,1,0,4],[2,2,1,9]], [1,2,1]
输出: 11
解释:
A，B，C的价格分别为¥2，¥3，¥4.
你可以用¥4购买1A和1B，也可以用¥9购买2A，2B和1C。
你需要买1A，2B和1C，所以你付了¥4买了1A和1B（大礼包1），以及¥3购买1B，  ¥4购买1C。
你不可以购买超出待购清单的物品，尽管购买大礼包2更加便宜。
```

说明:

- 最多6种物品， 100种大礼包。
- 每种物品，你最多只需要购买6个。
- 你不可以购买超出待购清单的物品，即使更便宜。

解题思路

- 给出 3 个数组，3 个数组分别代表的意义是在售的商品价格，多个礼包以及礼包内每个商品的数量和总价，购物清单上需要购买每个商品的数量。问购买清单上的所有商品所需的最低花费。
- 这一题可以用 DFS 暴力解题，也可以用 DP。笔者这题先用 DFS 来解答。设当前搜索到的状态为 `shopping(price, special, needs)`，其中 `price` 和 `special` 为题目中所述的物品的单价和捆绑销售的大礼包，而 `needs` 为当前需要的每种物品的数量。针对于每个商品，可以有 3 种购买规则，第一种，选礼包里面的第一个优惠购买，第二种，不选当前礼包优惠，选下一个优惠进行购买，第三种，不使用优惠，直接购买。这样就对应了 3 种 DFS 的方向。具体见代码。如果选择了礼包优惠，那么递归到下一层，`need` 需要对应减少礼包里面的数量，最终金额累加。当所有情况遍历完以后，可以返回出最小花费。
- 这一题需要注意的剪枝情况：是否需要购买礼包。题目中要求了，不能购买超过清单上数量的商品，即使价格便宜，也不行。例如可以买 n 个礼包 A，但是最终商品数量超过了清单上的商品，这种购买方式是不行的。所以需要先判断当前递归中，满足 `need` 和 `price` 条件的，能否使用礼包。这里包含 2 种情况，一种是当前商品已经满足清单个数了，不需要再买了；还有一种情况是已经超过清单数量了，那这种情况需要立即返回，当前这种购买方式不合题意。

代码

```go
func shoppingOffers(price []int, special [][]int, needs []int) int {
  res := -1
  dfsShoppingOffers(price, special, needs, 0, &res)
  return res
```

```go
}

func dfsShoppingOffers(price []int, special [][]int, needs []int, pay int, res *int) {
  noNeeds := true
  // 剪枝
  for _, need := range needs {
    if need < 0 {
      return
    }
    if need != 0 {
      noNeeds = false
    }
  }
  if len(special) == 0 || noNeeds {
    for i, p := range price {
      pay += (p * needs[i])
    }
    if pay < *res || *res == -1 {
      *res = pay
    }
    return
  }
  newNeeds := make([]int, len(needs))
  copy(newNeeds, needs)
  for i, n := range newNeeds {
    newNeeds[i] = n - special[0][i]
  }
  dfsShoppingOffers(price, special, newNeeds, pay+special[0][len(price)], res)
  dfsShoppingOffers(price, special[1:], newNeeds, pay+special[0][len(price)], res)
  dfsShoppingOffers(price, special[1:], needs, pay, res)
}
```

643. Maximum Average Subarray I

题目

Given an array consisting of `n` integers, find the contiguous subarray of given length `k` that has the maximum average value. And you need to output the maximum average value.

Example 1:

```
Input: [1,12,-5,-6,50,3], k = 4
Output: 12.75
Explanation: Maximum average is (12-5-6+50)/4 = 51/4 = 12.75
```

Note:

1. 1 <= `k` <= `n` <= 30,000.

2. Elements of the given array will be in the range [-10,000, 10,000].

题目大意

给定 n 个整数，找出平均数最大且长度为 k 的连续子数组，并输出该最大平均数。

解题思路

- 简单题。循环一次，扫描数组过程中累加窗口大小为 k 的元素值。不断更新这个最大值。循环结束求出平均值即可。

代码

```
package leetcode

func findMaxAverage(nums []int, k int) float64 {
  sum := 0
  for _, v := range nums[:k] {
    sum += v
  }
  maxSum := sum
  for i := k; i < len(nums); i++ {
    sum = sum - nums[i-k] + nums[i]
    maxSum = max(maxSum, sum)
  }
  return float64(maxSum) / float64(k)
}

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

645. Set Mismatch

题目

The set `S` originally contains numbers from 1 to `n`. But unfortunately, due to the data error, one of the numbers in the set got duplicated to **another** number in the set, which results in repetition of one number and loss of another number.

Given an array `nums` representing the data status of this set after the error. Your task is to firstly find the number occurs twice and then find the number that is missing. Return them in the form of an array.

Example 1:

```
Input: nums = [1,2,2,4]
Output: [2,3]
```

Note:

1. The given array size will in the range [2, 10000].
2. The given array's numbers won't have any order.

题目大意

集合 S 包含从1到 n 的整数。不幸的是，因为数据错误，导致集合里面某一个元素复制了成了集合里面的另外一个元素的值，导致集合丢失了一个整数并且有一个元素重复。给定一个数组 nums 代表了集合 S 发生错误后的结果。你的任务是首先寻找到重复出现的整数，再找到丢失的整数，将它们以数组的形式返回。

注意:

- 给定数组的长度范围是 [2, 10000]。
- 给定的数组是无序的。

解题思路

- 给出一个数组，数组里面装的是 1-n 的数字，由于错误导致有一个数字变成了另外一个数字，要求找出重复的一个数字和正确的数字。这一题是简单题，根据下标比对就可以找到哪个数字重复了，哪个数字缺少了。

代码

```go
package leetcode

func findErrorNums(nums []int) []int {
  m, res := make([]int, len(nums)), make([]int, 2)
  for _, n := range nums {
    if m[n-1] == 0 {
      m[n-1] = 1
    } else {
      res[0] = n
    }
  }
  for i := range m {
    if m[i] == 0 {
      res[1] = i + 1
      break
    }
  }
  return res
}
```

647. Palindromic Substrings

题目

Given a string, your task is to count how many palindromic substrings in this string.

The substrings with different start indexes or end indexes are counted as different substrings even they consist of same characters.

Example 1:

```
Input: "abc"
Output: 3
Explanation: Three palindromic strings: "a", "b", "c".
```

Example 2:

```
Input: "aaa"
Output: 6
Explanation: Six palindromic strings: "a", "a", "a", "aa", "aa", "aaa".
```

Note:

1. The input string length won't exceed 1000.

题目大意

给定一个字符串，你的任务是计算这个字符串中有多少个回文子串。具有不同开始位置或结束位置的子串，即使是由相同的字符组成，也会被视作不同的子串。

解题思路

- 暴力解法，从左往右扫一遍字符串，以每个字符做轴，用中心扩散法，依次遍历计数回文子串。

代码

```go
package leetcode

func countSubstrings(s string) int {
  res := 0
  for i := 0; i < len(s); i++ {
    res += countPalindrome(s, i, i)
    res += countPalindrome(s, i, i+1)
  }
  return res
}

func countPalindrome(s string, left, right int) int {
```

```
  res := 0
  for left >= 0 && right < len(s) {
    if s[left] != s[right] {
      break
    }
    left--
    right++
    res++
  }
  return res
}
```

648. Replace Words

题目

In English, we have a concept called `root`, which can be followed by some other words to form another longer word - let's call this word `successor`. For example, the root `an`, followed by `other`, which can form another word `another`.

Now, given a dictionary consisting of many roots and a sentence. You need to replace all the `successor` in the sentence with the `root` forming it. If a `successor` has many `roots` can form it, replace it with the root with the shortest length.

You need to output the sentence after the replacement.

Example 1:

```
Input: dict = ["cat", "bat", "rat"]
sentence = "the cattle was rattled by the battery"
Output: "the cat was rat by the bat"
```

Note:

1. The input will only have lower-case letters.
2. 1 <= dict words number <= 1000
3. 1 <= sentence words number <= 1000
4. 1 <= root length <= 100
5. 1 <= sentence words length <= 1000

题目大意

在英语中，我们有一个叫做 词根(root)的概念，它可以跟着其他一些词组成另一个较长的单词——我们称这个词为 继承词(successor)。例如，词根an，跟随着单词 other(其他)，可以形成新的单词 another(另一个)。

现在，给定一个由许多词根组成的词典和一个句子。你需要将句子中的所有继承词用词根替换掉。如果继承词有许多可以形成它的词根，则用最短的词根替换它。要求输出替换之后的句子。

解题思路

- 给出一个句子和一个可替换字符串的数组，如果句子中的单词和可替换列表里面的单词，有相同的首字母，那么就把句子中的单词替换成可替换列表里面的单词。输入最后替换完成的句子。
- 这一题有 2 种解题思路，第一种就是单纯的用 Map 查找。第二种是用 Trie 去替换。

代码

```
package leetcode

import "strings"

// 解法一 哈希表
func replaceWords(dict []string, sentence string) string {
  roots := make(map[byte][]string)
  for _, root := range dict {
    b := root[0]
    roots[b] = append(roots[b], root)
  }
  words := strings.Split(sentence, " ")
  for i, word := range words {
    b := []byte(word)
    for j := 1; j < len(b) && j <= 100; j++ {
      if findWord(roots, b[0:j]) {
        words[i] = string(b[0:j])
        break
      }
    }
  }
  return strings.Join(words, " ")
}

func findWord(roots map[byte][]string, word []byte) bool {
  if roots[word[0]] == nil {
    return false
  }
  for _, root := range roots[word[0]] {
    if root == string(word) {
      return true
    }
  }
  return false
}

//解法二 Trie
func replaceWords1(dict []string, sentence string) string {
  trie := Constructor208()
```

```go
  for _, v := range dict {
    trie.Insert(v)
  }
  words := strings.Split(sentence, " ")
  var result []string
  word := ""
  i := 0
  for _, value := range words {
    word = ""
    for i = 1; i < len(value); i++ {
      if trie.Search(value[:i]) {
        word = value[:i]
        break
      }
    }

    if len(word) == 0 {
      result = append(result, value)
    } else {
      result = append(result, word)
    }

  }
  return strings.Join(result, " ")
}
```

653. Two Sum IV - Input is a BST

题目

Given a Binary Search Tree and a target number, return true if there exist two elements in the BST such that their sum is equal to the given target.

Example 1:

```
Input:
    5
   / \
  3   6
 / \   \
2   4   7

Target = 9

Output: True
```

Example 2:

```
Input:
    5
   / \
  3   6
 / \   \
2   4   7

Target = 28

Output: False
```

题目大意

给定一个二叉搜索树和一个目标结果，如果 BST 中存在两个元素且它们的和等于给定的目标结果，则返回 true。

解题思路

- 在树中判断是否存在 2 个数的和是 sum。
- 这一题是 two sum 问题的变形题，只不过题目背景是在 BST 上处理的。处理思路大体一致，用 map 记录已经访问过的节点值。边遍历树边查看 map 里面是否有 sum 的另外一半。

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func findTarget(root *TreeNode, k int) bool {
  m := make(map[int]int, 0)
  return findTargetDFS(root, k, m)
}

func findTargetDFS(root *TreeNode, k int, m map[int]int) bool {
  if root == nil {
    return false
  }
  if _, ok := m[k-root.Val]; ok {
    return ok
  }
  m[root.Val]++
  return findTargetDFS(root.Left, k, m) || findTargetDFS(root.Right, k, m)
```

```
}
```

658. Find K Closest Elements

题目

Given a sorted array, two integers `k` and `x`, find the `k` closest elements to `x` in the array. The result should also be sorted in ascending order. If there is a tie, the smaller elements are always preferred.

Example 1:

```
Input: [1,2,3,4,5], k=4, x=3
Output: [1,2,3,4]
```

Example 2:

```
Input: [1,2,3,4,5], k=4, x=-1
Output: [1,2,3,4]
```

Note:

1. The value k is positive and will always be smaller than the length of the sorted array.
2. Length of the given array is positive and will not exceed 10^4
3. Absolute value of elements in the array and x will not exceed 10^4

UPDATE (2017/9/19): The arr parameter had been changed to an **array of integers** (instead of a list of integers). **Please reload the code definition to get the latest changes**.

题目大意

给定一个排序好的数组，两个整数 k 和 x，从数组中找到最靠近 x（两数之差最小）的 k 个数。返回的结果必须要是按升序排好的。如果有两个数与 x 的差值一样，优先选择数值较小的那个数。

说明:

1. k 的值为正数，且总是小于给定排序数组的长度。
2. 数组不为空，且长度不超过 104
3. 数组里的每个元素与 x 的绝对值不超过 104

更新(2017/9/19):
这个参数 arr 已经被改变为一个整数数组（而不是整数列表）。 请重新加载代码定义以获取最新更改。

解题思路

- 给出一个数组，要求在数组中找到一个长度为 k 的区间，这个区间内每个元素距离 x 的距离都是整个数组里

面最小的。

- 这一题可以用双指针解题，最优解法是二分搜索。由于区间长度固定是 K 个，所以左区间最大只能到 `len(arr) - K`(因为长度为 K 以后，正好右区间就到数组最右边了)，在 `[0,len(arr) - K]` 这个区间中进行二分搜索。如果发现 `a[mid]` 与 `x` 距离比 `a[mid + k]` 与 `x` 的距离要大，说明要找的区间一定在右侧，继续二分，直到最终 `low = high` 的时候退出。逼出的 `low` 值就是最终答案区间的左边界。

代码

```go
package leetcode

import "sort"

// 解法一 库函数二分搜索
func findClosestElements(arr []int, k int, x int) []int {
  return arr[sort.Search(len(arr)-k, func(i int) bool { return x-arr[i] <= arr[i+k]-x
}):][:k]
}

// 解法二 手撸二分搜索
func findClosestElements1(arr []int, k int, x int) []int {
  low, high := 0, len(arr)-k
  for low < high {
    mid := low + (high-low)>>1
    if x-arr[mid] > arr[mid+k]-x {
      low = mid + 1
    } else {
      high = mid
    }
  }
  return arr[low : low+k]
}
```

661. Image Smoother

题目

Given a 2D integer matrix M representing the gray scale of an image, you need to design a smoother to make the gray scale of each cell becomes the average gray scale (rounding down) of all the 8 surrounding cells and itself. If a cell has less than 8 surrounding cells, then use as many as you can.

Example 1:

```
Input:
[[1,1,1],
 [1,0,1],
 [1,1,1]]
Output:
[[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]]
Explanation:
For the point (0,0), (0,2), (2,0), (2,2): floor(3/4) = floor(0.75) = 0
For the point (0,1), (1,0), (1,2), (2,1): floor(5/6) = floor(0.83333333) = 0
For the point (1,1): floor(8/9) = floor(0.88888889) = 0
```

Note:

1. The value in the given matrix is in the range of [0, 255].
2. The length and width of the given matrix are in the range of [1, 150].

题目大意

包含整数的二维矩阵 M 表示一个图片的灰度。你需要设计一个平滑器来让每一个单元的灰度成为平均灰度 (向下舍入) ，平均灰度的计算是周围的8个单元和它本身的值求平均，如果周围的单元格不足八个，则尽可能多的利用它们。

注意:

- 给定矩阵中的整数范围为 [0, 255]。
- 矩阵的长和宽的范围均为 [1, 150]。

解题思路

- 将二维数组中的每个元素变为周围 9 个元素的平均值。
- 简单题，按照题意计算平均值即可。需要注意的是边界问题，四个角和边上的元素，这些点计算平均值的时候，计算平均值都不足 9 个元素。

代码

```go
package leetcode

func imageSmoother(M [][]int) [][]int {
  res := make([][]int, len(M))
  for i := range M {
    res[i] = make([]int, len(M[0]))
  }
  for y := 0; y < len(M); y++ {
    for x := 0; x < len(M[0]); x++ {
      res[y][x] = smooth(x, y, M)
```

```go
    }
  }
  return res
}

func smooth(x, y int, M [][]int) int {
  count, sum := 1, M[y][x]
  // Check bottom
  if y+1 < len(M) {
    sum += M[y+1][x]
    count++
  }
  // Check Top
  if y-1 >= 0 {
    sum += M[y-1][x]
    count++
  }
  // Check left
  if x-1 >= 0 {
    sum += M[y][x-1]
    count++
  }
  // Check Right
  if x+1 < len(M[y]) {
    sum += M[y][x+1]
    count++
  }
  // Check Coners
  // Top Left
  if y-1 >= 0 && x-1 >= 0 {
    sum += M[y-1][x-1]
    count++
  }
  // Top Right
  if y-1 >= 0 && x+1 < len(M[0]) {
    sum += M[y-1][x+1]
    count++
  }
  // Bottom Left
  if y+1 < len(M) && x-1 >= 0 {
    sum += M[y+1][x-1]
    count++
  }
  //Bottom Right
  if y+1 < len(M) && x+1 < len(M[0]) {
    sum += M[y+1][x+1]
    count++
  }
  return sum / count
```

```
}
```

662. Maximum Width of Binary Tree

题目

Given a binary tree, write a function to get the maximum width of the given tree. The width of a tree is the maximum width among all levels. The binary tree has the same structure as a **full binary tree**, but some nodes are null.

The width of one level is defined as the length between the end-nodes (the leftmost and right most non-null nodes in the level, where the `null` nodes between the end-nodes are also counted into the length calculation.

Example 1:

```
Input:

           1
         /   \
        3     2
       / \     \
      5   3     9

Output: 4
Explanation: The maximum width existing in the third level with the length 4
(5,3,null,9).
```

Example 2:

```
Input:

          1
         /
        3
       / \
      5   3

Output: 2
Explanation: The maximum width existing in the third level with the length 2 (5,3).
```

Example 3:

```
Input:

          1
         / \
        3   2
       /
      5

Output: 2
Explanation: The maximum width existing in the second level with the length 2 (3,2).
```

Example 4:

```
Input:

          1
         / \
        3   2
       /     \
      5       9
     /         \
    6           7
Output: 8
Explanation:The maximum width existing in the fourth level with the length 8
(6,null,null,null,null,null,null,7).
```

Note: Answer will in the range of 32-bit signed integer.

题目大意

给定一个二叉树，编写一个函数来获取这个树的最大宽度。树的宽度是所有层中的最大宽度。这个二叉树与满二叉树（full binary tree）结构相同，但一些节点为空。

每一层的宽度被定义为两个端点（该层最左和最右的非空节点，两端点间的null节点也计入长度）之间的长度。

注意: 答案在32位有符号整数的表示范围内。

解题思路

- 给出一个二叉树，求这棵树最宽的部分。
- 这一题可以用 BFS 也可以用 DFS，但是用 BFS 比较方便。按照层序遍历，依次算出每层最左边不为 `null` 的节点和最右边不为 `null` 的节点。这两个节点之间都是算宽度的。最终输出最大的宽度即可。此题的关键在于如何有效的找到每一层的左右边界。
- 这一题可能有人会想着先补全满二叉树，然后每层分别找左右边界。这种方法提交以后会卡在 `104 / 108` 这组测试用例上，这组测试用例会使得最后某几层填充出现的满二叉树节点特别多，最终导致 `Memory Limit Exceeded` 了。

- 由于此题要找每层的左右边界，实际上每个节点的 `Val` 值是我们不关心的，那么可以把这个值用来标号，标记成该节点在每层中的序号。父亲节点在上一层中的序号是 x，那么它的左孩子在下一层满二叉树中的序号是 `2*x`，它的右孩子在下一层满二叉树中的序号是 `2*x + 1`。将所有节点都标上号，用 BFS 层序遍历每一层，每一层都找到左右边界，相减拿到宽度，动态维护最大宽度，就是本题的最终答案。

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func widthOfBinaryTree(root *TreeNode) int {
  if root == nil {
    return 0
  }
  if root.Left == nil && root.Right == nil {
    return 1
  }

  queue, res := []*TreeNode{}, 0
  queue = append(queue, &TreeNode{0, root.Left, root.Right})

  for len(queue) != 0 {
    var left, right *int
    // 这里需要注意，先保存 queue 的个数，相当于拿到此层的总个数
    qLen := len(queue)
    // 这里循环不要写 i < len(queue)，因为每次循环 queue 的长度都在变小
    for i := 0; i < qLen; i++ {
      node := queue[0]
      queue = queue[1:]
      if node.Left != nil {
        // 根据满二叉树父子节点的关系，得到下一层节点在本层的编号
        newVal := node.Val * 2
        queue = append(queue, &TreeNode{newVal, node.Left.Left, node.Left.Right})
        if left == nil || *left > newVal {
          left = &newVal
        }
        if right == nil || *right < newVal {
          right = &newVal
        }
      }
```

```go
        if node.Right != nil {
          // 根据满二叉树父子节点的关系，得到下一层节点在本层的编号
          newVal := node.Val*2 + 1
          queue = append(queue, &TreeNode{newVal, node.Right.Left, node.Right.Right})
          if left == nil || *left > newVal {
            left = &newVal
          }
          if right == nil || *right < newVal {
            right = &newVal
          }
        }
      }
      switch {
      // 某层只有一个点，那么此层宽度为 1
      case left != nil && right == nil, left == nil && right != nil:
        res = max(res, 1)
      // 某层只有两个点，那么此层宽度为两点之间的距离
      case left != nil && right != nil:
        res = max(res, *right-*left+1)
      }
    }
    return res
  }
```

665. Non-decreasing Array

题目

Given an array `nums` with `n` integers, your task is to check if it could become non-decreasing by modifying **at most one element**.

We define an array is non-decreasing if `nums[i] <= nums[i + 1]` holds for every `i` (**0-based**) such that (`0 <= i <= n - 2`).

Example 1:

```
Input: nums = [4,2,3]
Output: true
Explanation: You could modify the first 4 to 1 to get a non-decreasing array.
```

Example 2:

```
Input: nums = [4,2,1]
Output: false
Explanation: You can't get a non-decreasing array by modify at most one element.
```

Constraints:

- `n == nums.length`
- `1 <= n <= 104`
- `-10^5 <= nums[i] <= 10^5`

题目大意

给你一个长度为 n 的整数数组，请你判断在 最多 改变 1 个元素的情况下，该数组能否变成一个非递减数列。我们是这样定义一个非递减数列的： 对于数组中任意的 i (0 <= i <= n-2)，总满足 nums[i] <= nums[i + 1]。

解题思路

- 简单题。循环扫描数组，找到 `nums[i] > nums[i+1]` 这种递减组合。一旦这种组合超过 2 组，直接返回 false。找到第一组递减组合，需要手动调节一次。如果 `nums[i + 1] < nums[i - 1]`，就算交换 `nums[i+1]` 和 `nums[i]`，交换结束，`nums[i - 1]` 仍然可能大于 `nums[i + 1]`，不满足题意。正确的做法应该是让较小的那个数变大，即 `nums[i + 1] = nums[i]`。两个元素相等满足非递减的要求。

代码

```go
package leetcode

func checkPossibility(nums []int) bool {
  count := 0
  for i := 0; i < len(nums)-1; i++ {
    if nums[i] > nums[i+1] {
      count++
      if count > 1 {
        return false
      }
      if i > 0 && nums[i+1] < nums[i-1] {
        nums[i+1] = nums[i]
      }
    }
  }
  return true
}
```

667. Beautiful Arrangement II

题目

Given two integers `n` and `k`, you need to construct a list which contains `n` different positive integers ranging from `1` to `n` and obeys the following requirement:Suppose this list is [a1, a2, a3, ... , an], then the list [|a1 - a2|, |a2 - a3|, |a3 - a4|, ... , |an-1 - an|] has exactly `k` distinct integers.

If there are multiple answers, print any of them.

Example 1:

```
Input: n = 3, k = 1
Output: [1, 2, 3]
Explanation: The [1, 2, 3] has three different positive integers ranging from 1 to 3,
and the [1, 1] has exactly 1 distinct integer: 1.
```

Example 2:

```
Input: n = 3, k = 2
Output: [1, 3, 2]
Explanation: The [1, 3, 2] has three different positive integers ranging from 1 to 3,
and the [2, 1] has exactly 2 distinct integers: 1 and 2.
```

Note:

1. The `n` and `k` are in the range 1 <= k < n <= 10^4.

题目大意

给定两个整数 n 和 k，你需要实现一个数组，这个数组包含从 1 到 n 的 n 个不同整数，同时满足以下条件：

- 如果这个数组是 [a1, a2, a3, ... , an] ，那么数组 [|a1 - a2|, |a2 - a3|, |a3 - a4|, ... , |an-1 - an|] 中应该有且仅有 k 个不同整数；.
- 如果存在多种答案，你只需实现并返回其中任意一种.

解题思路

- 先考虑 `k` 最大值的情况。如果把末尾的较大值依次插入到前面的较小值中，形成 `[1, n, 2, n-1, 3, n-2, ……]`，这样排列 `k` 能取到最大值 `n-1` 。`k` 最小值的情况是 `[1, 2, 3, 4, ……, n]`，`k` 取到的最小值是 1。那么 `k` 在 `[1, n-1]` 之间取值，该怎么排列呢？先顺序排列 `[1, 2, 3, 4, ……, n-k-1]`，这里有 `n-k-1` 个数，可以形成唯一一种差值。剩下 `k+1` 个数，形成 `k-1` 种差值。
- 这又回到了 `k` 最大值的取法了。`k` 取最大值的情况是 `n` 个数，形成 `n-1` 个不同种的差值。现在 `k+1` 个数，需要形成 `k` 种不同的差值。两者是同一个问题。那么剩下 `k` 个数的排列方法是 `[n-k, n-k+1, …, n]`，这里有 `k` 个数，注意代码实现时，注意 `k` 的奇偶性，如果 `k` 是奇数，“对半穿插”以后，正好匹配完，如果 `k` 是偶数，对半处的数 `n-k+(k+1)/2`，最后还需要单独加入到排列中。
- 可能有读者会问了，前面生成了 1 种差值，后面这部分又生产了 `k` 种差值，加起来不是 `k + 1` 种差值了么？这种理解是错误的。后面这段最后 2 个数字是 `n-k+(k+1)/2-1` 和 `n-k+(k+1)/2`，它们两者的差值是 1，和第一段构造的排列差值是相同的，都是 1。所以第一段构造了 1 种差值，第二段虽然构造了 `k` 种，但是需要去掉两段重复的差值 1，所以最终差值种类还是 `1 + k - 1 = k` 种。

代码

```go
package leetcode

func constructArray(n int, k int) []int {
  res := []int{}
  for i := 0; i < n-k-1; i++ {
    res = append(res, i+1)
```

```
  }
  for i := n - k; i < n-k+(k+1)/2; i++ {
    res = append(res, i)
    res = append(res, 2*n-k-i)
  }
  if k%2 == 0 {
    res = append(res, n-k+(k+1)/2)
  }
  return res
}
```

668. Kth Smallest Number in Multiplication Table

题目

Nearly every one have used the Multiplication Table. But could you find out the `k-th` smallest number quickly from the multiplication table?

Given the height `m` and the length `n` of a `m * n` Multiplication Table, and a positive integer `k`, you need to return the `k-th` smallest number in this table.

Example 1:

```
Input: m = 3, n = 3, k = 5
Output:
Explanation:
The Multiplication Table:
1 2 3
2 4 6
3 6 9

The 5-th smallest number is 3 (1, 2, 2, 3, 3).
```

Example 2:

```
Input: m = 2, n = 3, k = 6
Output:
Explanation:
The Multiplication Table:
1 2 3
2 4 6

The 6-th smallest number is 6 (1, 2, 2, 3, 4, 6).
```

Note:

1. The `m` and `n` will be in the range [1, 30000].
2. The `k` will be in the range [1, m * n]

题目大意

几乎每一个人都用乘法表。但是你能在乘法表中快速找到第 k 小的数字吗？给定高度 m 、宽度 n 的一张 m * n 的乘法表，以及正整数 k，你需要返回表中第 k 小的数字。

注意：

- m 和 n 的范围在 [1, 30000] 之间。
- k 的范围在 [1, m * n] 之间。

解题思路

- 给出 3 个数字，m，n，k。m 和 n 分别代表乘法口诀表的行和列。要求在这个乘法口诀表中找第 k 小的数字。
- 这一题是第 378 题变种题。利用二分搜索，在 `[1,m*n]` 的区间内搜索第 `k` 小的数。每次二分统计 `≤ mid` 数字的个数。由于是在两数乘法构成的矩阵中计数，知道乘数，被乘数也就知道了，所以计数只需要一层循环。整体代码和第 378 题完全一致，只是计数的部分不同罢了。可以对比第 378 题一起练习。

代码

```go
package leetcode

import "math"

func findKthNumber(m int, n int, k int) int {
  low, high := 1, m*n
  for low < high {
    mid := low + (high-low)>>1
    if counterKthNum(m, n, mid) >= k {
      high = mid
    } else {
      low = mid + 1
    }
  }
  return low
}

func counterKthNum(m, n, mid int) int {
  count := 0
  for i := 1; i <= m; i++ {
    count += int(math.Min(math.Floor(float64(mid)/float64(i)), float64(n)))
  }
  return count
}
```

669. Trim a Binary Search Tree

题目

Given the `root` of a binary search tree and the lowest and highest boundaries as `low` and `high`, trim the tree so that all its elements lies in `[low, high]`. Trimming the tree should **not** change the relative structure of the elements that will remain in the tree (i.e., any node's descendant should remain a descendant). It can be proven that there is a **unique answer**.

Return *the root of the trimmed binary search tree*. Note that the root may change depending on the given bounds.

Example 1:

```
Input: root = [1,0,2], low = 1, high = 2
Output: [1,null,2]
```

Example 2:

```
Input: root = [3,0,4,null,2,null,null,1], low = 1, high = 3
Output: [3,2,null,1]
```

Example 3:

```
Input: root = [1], low = 1, high = 2
Output: [1]
```

Example 4:

```
Input: root = [1,null,2], low = 1, high = 3
Output: [1,null,2]
```

Example 5:

```
Input: root = [1,null,2], low = 2, high = 4
Output: [2]
```

Constraints:

- The number of nodes in the tree in the range `[1, 10^4]`.
- `0 <= Node.val <= 10^4`
- The value of each node in the tree is **unique**.
- `root` is guaranteed to be a valid binary search tree.
- `0 <= low <= high <= 10^4`

题目大意

给你二叉搜索树的根节点 root ，同时给定最小边界low 和最大边界 high。通过修剪二叉搜索树，使得所有节点的值在[low, high]中。修剪树不应该改变保留在树中的元素的相对结构（即，如果没有被移除，原有的父代子代关系都应当保留）。 可以证明，存在唯一的答案。所以结果应当返回修剪好的二叉搜索树的新的根节点。注意，根节点可能会根据给定的边界发生改变。

解题思路

- 这一题考察二叉搜索树中的递归遍历。递归遍历二叉搜索树每个结点，根据有序性，当前结点如果比 high 大，那么当前结点的右子树全部修剪掉，再递归修剪左子树；当前结点如果比 low 小，那么当前结点的左子树全部修剪掉，再递归修剪右子树。处理完越界的情况，剩下的情况都在区间内，分别递归修剪左子树和右子树即可。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/structures"
)

// TreeNode define
type TreeNode = structures.TreeNode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

func trimBST(root *TreeNode, low int, high int) *TreeNode {
  if root == nil {
    return root
  }
  if root.Val > high {
    return trimBST(root.Left, low, high)
  }
  if root.Val < low {
    return trimBST(root.Right, low, high)
  }
  root.Left = trimBST(root.Left, low, high)
  root.Right = trimBST(root.Right, low, high)
  return root
}
```

674. Longest Continuous Increasing Subsequence

题目

Given an unsorted array of integers `nums`, return *the length of the longest* ***continuous increasing subsequence*** *(i.e. subarray)*. The subsequence must be **strictly** increasing.

A **continuous increasing subsequence** is defined by two indices `l` and `r` (`l < r`) such that it is `[nums[l], nums[l + 1], ..., nums[r - 1], nums[r]]` and for each `l <= i < r`, `nums[i] < nums[i + 1]`.

Example 1:

```
Input: nums = [1,3,5,4,7]
Output: 3
Explanation: The longest continuous increasing subsequence is [1,3,5] with length 3.
Even though [1,3,5,7] is an increasing subsequence, it is not continuous as elements 5
and 7 are separated by element
4.
```

Example 2:

```
Input: nums = [2,2,2,2,2]
Output: 1
Explanation: The longest continuous increasing subsequence is [2] with length 1. Note
that it must be strictly
increasing.
```

Constraints:

- `0 <= nums.length <= 10^4`
- `10^9 <= nums[i] <= 10^9`

题目大意

给定一个未经排序的整数数组，找到最长且 连续递增的子序列，并返回该序列的长度。连续递增的子序列 可以由两个下标 l 和 r（l < r）确定，如果对于每个 l <= i < r，都有 nums[i] < nums[i + 1] ，那么子序列 [nums[l], nums[l + 1], ..., nums[r - 1], nums[r]] 就是连续递增子序列。

解题思路

- 简单题。这一题和第 128 题有区别。这一题要求子序列必须是连续下标，所以变简单了。扫描一遍数组，记下连续递增序列的长度，动态维护这个最大值，最后输出即可。

代码

```go
package leetcode

func findLengthOfLCIS(nums []int) int {
  if len(nums) == 0 {
    return 0
  }
  res, length := 1, 1
  for i := 1; i < len(nums); i++ {
    if nums[i] > nums[i-1] {
      length++
    } else {
      res = max(res, length)
      length = 1
    }
  }
  return max(res, length)
}

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

676. Implement Magic Dictionary

题目

Implement a magic directory with `buildDict`, and `search` methods.

For the method `buildDict`, you'll be given a list of non-repetitive words to build a dictionary.

For the method `search`, you'll be given a word, and judge whether if you modify **exactly** one character into **another**character in this word, the modified word is in the dictionary you just built.

Example 1:

```
Input: buildDict(["hello", "leetcode"]), Output: Null
Input: search("hello"), Output: False
Input: search("hhllo"), Output: True
Input: search("hell"), Output: False
Input: search("leetcoded"), Output: False
```

Note:

1. You may assume that all the inputs are consist of lowercase letters `a-z`.
2. For contest purpose, the test data is rather small by now. You could think about highly efficient

algorithm after the contest.

3. Please remember to **RESET** your class variables declared in class MagicDictionary, as static/class variables are **persisted across multiple test cases**. Please see here for more details.

题目大意

实现一个带有 buildDict, 以及 search 方法的魔法字典。对于 buildDict 方法，你将被给定一串不重复的单词来构建一个字典。对于 search 方法，你将被给定一个单词，并且判定能否只将这个单词中一个字母换成另一个字母，使得所形成的新单词存在于你构建的字典中。

解题思路

- 实现 `MagicDictionary` 的数据结构，这个数据结构内会存储一个字符串数组，当执行 `Search` 操作的时候要求判断传进来的字符串能否只改变一个字符(不能增加字符也不能删除字符)就能变成 `MagicDictionary` 中存储的字符串，如果可以，就输出 `true`，如果不能，就输出 `false`。
- 这题的解题思路比较简单，用 Map 判断即可。

代码

```go
package leetcode

type MagicDictionary struct {
  rdict map[int]string
}

/** Initialize your data structure here. */
func Constructor676() MagicDictionary {
  return MagicDictionary{rdict: make(map[int]string)}
}

/** Build a dictionary through a list of words */
func (this *MagicDictionary) BuildDict(dict []string) {
  for k, v := range dict {
    this.rdict[k] = v
  }
}

/** Returns if there is any word in the trie that equals to the given word after
modifying exactly one character */
func (this *MagicDictionary) Search(word string) bool {
  for _, v := range this.rdict {
    n := 0
    if len(word) == len(v) {
      for i := 0; i < len(v); i++ {
        if word[i] != v[i] {
          n += 1
```

```
        }
      }
      if n == 1 {
        return true
      }
    }
  }
  return false
}

/**
 * Your MagicDictionary object will be instantiated and called as such:
 * obj := Constructor();
 * obj.BuildDict(dict);
 * param_2 := obj.Search(word);
 */
```

682. Baseball Game

题目

You're now a baseball game point recorder.

Given a list of strings, each string can be one of the 4 following types:

1. Integer (one round's score): Directly represents the number of points you get in this round.
2. "+" (one round's score): Represents that the points you get in this round are the sum of the last two valid round's points.
3. "D" (one round's score): Represents that the points you get in this round are the doubled data of the last valid round's points.
4. "C" (an operation, which isn't a round's score): Represents the last valid round's points you get were invalid and should be removed.
   Each round's operation is permanent and could have an impact on the round before and the round after.

You need to return the sum of the points you could get in all the rounds.

Example 1:

```
Input: ["5","2","C","D","+"]
Output: 30
Explanation:
Round 1: You could get 5 points. The sum is: 5.
Round 2: You could get 2 points. The sum is: 7.
Operation 1: The round 2's data was invalid. The sum is: 5.
Round 3: You could get 10 points (the round 2's data has been removed). The sum is: 15.
Round 4: You could get 5 + 10 = 15 points. The sum is: 30.
```

Example 2:

```
Input: ["5","-2","4","C","D","9","+","+"]
Output: 27
Explanation:
Round 1: You could get 5 points. The sum is: 5.
Round 2: You could get -2 points. The sum is: 3.
Round 3: You could get 4 points. The sum is: 7.
Operation 1: The round 3's data is invalid. The sum is: 3.
Round 4: You could get -4 points (the round 3's data has been removed). The sum is: -1.
Round 5: You could get 9 points. The sum is: 8.
Round 6: You could get -4 + 9 = 5 points. The sum is 13.
Round 7: You could get 9 + 5 = 14 points. The sum is 27.
```

Note:

- The size of the input list will be between 1 and 1000.
- Every integer represented in the list will be between -30000 and 30000.

题目大意

这道题是模拟题，给一串数字和操作符。出现数字就直接累加，出现 "C" 就代表栈推出一个元素，相应的总和要减去栈顶的元素。出现 "D" 就代表把前一个元素乘以 2，就得到当前的元素值。再累加。出现 "+" 就代表把前 2 个值求和，得到当前元素的值，再累积。

解题思路

这道题用栈模拟即可。

代码

```go
package leetcode

import "strconv"

func calPoints(ops []string) int {
  stack := make([]int, len(ops))
  top := 0

  for i := 0; i < len(ops); i++ {
    op := ops[i]
    switch op {
    case "+":
      last1 := stack[top-1]
      last2 := stack[top-2]
      stack[top] = last1 + last2
      top++
    case "D":
      last1 := stack[top-1]
      stack[top] = last1 * 2
      top++
    case "C":
      top--
    default:
      stack[top], _ = strconv.Atoi(op)
      top++
    }
  }

  points := 0
  for i := 0; i < top; i++ {
    points += stack[i]
  }
  return points
}
```

684. Redundant Connection

题目

In this problem, a tree is an **undirected** graph that is connected and has no cycles.

The given input is a graph that started as a tree with N nodes (with distinct values 1, 2, ..., N), with one additional edge added. The added edge has two different vertices chosen from 1 to N, and was not an edge that already existed.

The resulting graph is given as a 2D-array of `edges`. Each element of `edges` is a pair `[u, v]` with `u < v`, that represents an **undirected** edge connecting nodes `u` and `v`.

Return an edge that can be removed so that the resulting graph is a tree of N nodes. If there are multiple answers, return the answer that occurs last in the given 2D-array. The answer edge `[u, v]` should be in the same format, with `u < v`.

Example 1:

```
Input: [[1,2], [1,3], [2,3]]
Output: [2,3]
Explanation: The given undirected graph will be like this:
  1
 / \
2 - 3
```

Example 2:

```
Input: [[1,2], [2,3], [3,4], [1,4], [1,5]]
Output: [1,4]
Explanation: The given undirected graph will be like this:
5 - 1 - 2
    |   |
    4 - 3
```

Note:

- The size of the input 2D-array will be between 3 and 1000.
- Every integer represented in the 2D-array will be between 1 and N, where N is the size of the input array.

Update (2017-09-26): We have overhauled the problem description + test cases and specified clearly the graph is an **undirected** graph. For the **directed** graph follow up please see [Redundant Connection II](). We apologize for any inconvenience caused.

题目大意

在本问题中, 树指的是一个连通且无环的无向图。输入一个图，该图由一个有着N个节点 (节点值不重复1, 2, ..., N) 的树及一条附加的边构成。附加的边的两个顶点包含在1到N中间，这条附加的边不属于树中已存在的边。结果图是一个以边组成的二维数组。每一个边的元素是一对[u, v] ，满足 u < v，表示连接顶点u 和v的无向图的边。

返回一条可以删去的边，使得结果图是一个有着N个节点的树。如果有多个答案，则返回二维数组中最后出现的边。答案边 [u, v] 应满足相同的格式 u < v。

注意:

- 输入的二维数组大小在 3 到 1000。
- 二维数组中的整数在 1 到 N 之间，其中 N 是输入数组的大小。

解题思路

- 给出一个连通无环无向图和一些连通的边，要求在这些边中删除一条边以后，图中的 N 个节点依旧是连通的。如果有多条边，输出最后一条。

- 这一题可以用并查集直接秒杀。依次扫描所有的边，把边的两端点都合并 `union()` 到一起。如果遇到一条边的两端点已经在一个集合里面了，就说明是多余边，删除。最后输出这些边即可。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/template"
)

func findRedundantConnection(edges [][]int) []int {
  if len(edges) == 0 {
    return []int{}
  }
  uf, res := template.UnionFind{}, []int{}
  uf.Init(len(edges) + 1)
  for i := 0; i < len(edges); i++ {
    if uf.Find(edges[i][0]) != uf.Find(edges[i][1]) {
      uf.Union(edges[i][0], edges[i][1])
    } else {
      res = append(res, edges[i][0])
      res = append(res, edges[i][1])
    }
  }
  return res
}
```

685. Redundant Connection II

题目

In this problem, a rooted tree is a **directed** graph such that, there is exactly one node (the root) for which all other nodes are descendants of this node, plus every node has exactly one parent, except for the root node which has no parents.

The given input is a directed graph that started as a rooted tree with N nodes (with distinct values 1, 2, ..., N), with one additional directed edge added. The added edge has two different vertices chosen from 1 to N, and was not an edge that already existed.

The resulting graph is given as a 2D-array of `edges`. Each element of `edges` is a pair `[u, v]` that represents a **directed** edge connecting nodes `u` and `v`, where `u` is a parent of child `v`.

Return an edge that can be removed so that the resulting graph is a rooted tree of N nodes. If there are multiple answers, return the answer that occurs last in the given 2D-array.

Example 1:

```
Input: [[1,2], [1,3], [2,3]]
Output: [2,3]
Explanation: The given directed graph will be like this:
  1
 / \
v   v
2-->3
```

Example 2:

```
Input: [[1,2], [2,3], [3,4], [4,1], [1,5]]
Output: [4,1]
Explanation: The given directed graph will be like this:
5 <- 1 -> 2
     ^    |
     |    v
     4 <- 3
```

Note:

- The size of the input 2D-array will be between 3 and 1000.
- Every integer represented in the 2D-array will be between 1 and N, where N is the size of the input array.

题目大意

在本问题中，有根树指满足以下条件的有向图。该树只有一个根节点，所有其他节点都是该根节点的后继。每一个节点只有一个父节点，除了根节点没有父节点。输入一个有向图，该图由一个有着 N 个节点 (节点值不重复1, 2, ..., N) 的树及一条附加的边构成。附加的边的两个顶点包含在1到N中间，这条附加的边不属于树中已存在的边。结果图是一个以边组成的二维数组。 每一个边的元素是一对 [u, v]，用以表示有向图中连接顶点 u and v 和顶点的边，其中父节点 u 是子节点 v 的一个父节点。返回一条能删除的边，使得剩下的图是有 N 个节点的有根树。若有多个答案，返回最后出现在给定二维数组的答案。

注意:

- 二维数组大小的在 3 到 1000 范围内。
- 二维数组中的每个整数在 1 到 N 之间，其中 N 是二维数组的大小。

解题思路

- 这一题是第 684 题的加强版。第 684 题中的图是无向图，这一题中的图是有向图。
- 这一题的解法也是用并查集，不过需要灵活一点，不要用模板，因为在模板中，存在路径压缩和 `rank()` 优化，这些优化会改变有向边原始的方向。所以并查集只需要记录 `parent()` 就够用了。

- 经过分析，可以得到上面这 3 种情况，红色的边是我们实际应该删除的。先来看情况 2 和情况 3 。当不断 `union()` 时，加入一条边以后，会使一个节点的入度变成 2，那么记录下这两条边为 `candidate1` 和 `candidate2`。将后加入的 `candidate2` 这条边先放在一边，继续往下 `union()`。如果 `candidate2` 是红色的边，那么合并到最后，也不会出现任何异常，那么 `candidate2` 就是红色的边，即找到了要删除的边了。如果合并到最后出现了环的问题了，那说明 `candidate2` 是黑色的边，`candidate1` 才是红色的边，那么 `candidate1` 是要删除的边。
- 再来看看情况 1。如果一路合并到结束也没有发现出现入度为 2 的情况，那么说明遇到了情况 1 。情况 1 会出现环的情况。题目中说如果要删除边，就删除最后出现的那条边。**具体实现见代码注释**。

代码

```
package leetcode

func findRedundantDirectedConnection(edges [][]int) []int {
  if len(edges) == 0 {
    return []int{}
  }
  parent, candidate1, candidate2 := make([]int, len(edges)+1), []int{}, []int{}
  for _, edge := range edges {
    if parent[edge[1]] == 0 {
      parent[edge[1]] = edge[0]
    } else { // 如果一个节点已经有父亲节点了，说明入度已经有 1 了，再来一条边，入度为 2 ，那么跳过新来的这条边 candidate2，并记录下和这条边冲突的边 candidate1
      candidate1 = append(candidate1, parent[edge[1]])
      candidate1 = append(candidate1, edge[1])
      candidate2 = append(candidate2, edge[0])
      candidate2 = append(candidate2, edge[1])
      edge[1] = 0 // 做标记，后面再扫到这条边以后可以直接跳过
    }
  }
```

```go
  for i := 1; i <= len(edges); i++ {
    parent[i] = i
  }
  for _, edge := range edges {
    if edge[1] == 0 { // 跳过 candidate2 这条边
      continue
    }
    u, v := edge[0], edge[1]
    pu := findRoot(&parent, u)
    if pu == v { // 发现有环
      if len(candidate1) == 0 { // 如果没有出现入度为 2 的情况，那么对应情况 1，就删除这条边
        return edge
      }
      return candidate1 // 出现环并且有入度为 2 的情况，说明 candidate1 是答案
    }
    parent[v] = pu // 没有发现环，继续合并
  }
  return candidate2 // 当最后什么都没有发生，则 candidate2 是答案
}

func findRoot(parent *[]int, k int) int {
  if (*parent)[k] != k {
    (*parent)[k] = findRoot(parent, (*parent)[k])
  }
  return (*parent)[k]
}
```

690. Employee Importance

题目

You are given a data structure of employee information, which includes the employee's **unique id**, their **importance value** and their **direct** subordinates' id.

For example, employee 1 is the leader of employee 2, and employee 2 is the leader of employee 3. They have importance value 15, 10 and 5, respectively. Then employee 1 has a data structure like [1, 15, [2]], and employee 2 has [2, 10, [3]], and employee 3 has [3, 5, []]. Note that although employee 3 is also a subordinate of employee 1, the relationship is **not direct**.

Now given the employee information of a company, and an employee id, you need to return the total importance value of this employee and all their subordinates.

Example 1:

```
Input: [[1, 5, [2, 3]], [2, 3, []], [3, 3, []]], 1
Output: 11
Explanation:
Employee 1 has importance value 5, and he has two direct subordinates: employee 2 and
employee 3. They both have importance value 3. So the total importance value of
employee 1 is 5 + 3 + 3 = 11.
```

Note:

1. One employee has at most one **direct** leader and may have several subordinates.
2. The maximum number of employees won't exceed 2000.

题目大意

给定一个保存员工信息的数据结构，它包含了员工 唯一的 id ， 重要度 和 直系下属的 id 。比如，员工 1 是员工 2 的领导，员工 2 是员工 3 的领导。他们相应的重要度为 15 , 10 , 5 。那么员工 1 的数据结构是 [1, 15, [2]] ，员工 2 的 数据结构是 [2, 10, [3]] ，员工 3 的数据结构是 [3, 5, []] 。注意虽然员工 3 也是员工 1 的一个下属，但是由于 并不是直系 下属，因此没有体现在员工 1 的数据结构中。现在输入一个公司的所有员工信息，以及单个员工 id ，返回这个员工和他所有下属的重要度之和。

解题思路

- 简单题。根据题意，DFS 或者 BFS 搜索找到所求 id 下属所有员工，累加下属员工的重要度，最后再加上这个员工本身的重要度，即为所求。

代码

```go
package leetcode

type Employee struct {
  Id           int
  Importance   int
  Subordinates []int
}

func getImportance(employees []*Employee, id int) int {
  m, queue, res := map[int]*Employee{}, []int{id}, 0
  for _, e := range employees {
    m[e.Id] = e
  }
  for len(queue) > 0 {
    e := m[queue[0]]
    queue = queue[1:]
    if e == nil {
      continue
    }
    res += e.Importance
    for _, i := range e.Subordinates {
```

```
        queue = append(queue, i)
      }
    }
    return res
  }
```

692. Top K Frequent Words

题目

Given a non-empty list of words, return the k most frequent elements.

Your answer should be sorted by frequency from highest to lowest. If two words have the same frequency, then the word with the lower alphabetical order comes first.

Example 1:

```
Input: ["i", "love", "leetcode", "i", "love", "coding"], k = 2
Output: ["i", "love"]
Explanation: "i" and "love" are the two most frequent words.
    Note that "i" comes before "love" due to a lower alphabetical order.
```

Example 2:

```
Input: ["the", "day", "is", "sunny", "the", "the", "the", "sunny", "is", "is"], k = 4
Output: ["the", "is", "sunny", "day"]
Explanation: "the", "is", "sunny" and "day" are the four most frequent words,
    with the number of occurrence being 4, 3, 2 and 1 respectively.
```

Note:

1. You may assume k is always valid, 1 ≤ k ≤ number of unique elements.
2. Input words contain only lowercase letters.

Follow up:

1. Try to solve it in O(n log k) time and O(n) extra space.

题目大意

给一非空的单词列表，返回前 *k* 个出现次数最多的单词。返回的答案应该按单词出现频率由高到低排序。如果不同的单词有相同出现频率，按字母顺序排序。

解题思路

- 思路很简单的题。维护一个长度为 k 的最大堆，先按照频率排，如果频率相同再按照字母顺序排。最后输出依次将优先队列里面的元素 pop 出来即可。

代码

```go
package leetcode

import "container/heap"

func topKFrequent(words []string, k int) []string {
  m := map[string]int{}
  for _, word := range words {
    m[word]++
  }
  pq := &PQ{}
  heap.Init(pq)
  for w, c := range m {
    heap.Push(pq, &wordCount{w, c})
    if pq.Len() > k {
      heap.Pop(pq)
    }
  }
  res := make([]string, k)
  for i := k - 1; i >= 0; i-- {
    wc := heap.Pop(pq).(*wordCount)
    res[i] = wc.word
  }
  return res
}

type wordCount struct {
  word string
  cnt  int
}

type PQ []*wordCount

func (pq PQ) Len() int      { return len(pq) }
func (pq PQ) Swap(i, j int) { pq[i], pq[j] = pq[j], pq[i] }
func (pq PQ) Less(i, j int) bool {
  if pq[i].cnt == pq[j].cnt {
    return pq[i].word > pq[j].word
  }
  return pq[i].cnt < pq[j].cnt
}
func (pq *PQ) Push(x interface{}) {
  tmp := x.(*wordCount)
  *pq = append(*pq, tmp)
}
func (pq *PQ) Pop() interface{} {
  n := len(*pq)
  tmp := (*pq)[n-1]
  *pq = (*pq)[:n-1]
  return tmp
```

```
}
```

693. Binary Number with Alternating Bits

题目

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.
```

题目大意

给定一个正整数，检查他是否为交替位二进制数：换句话说，就是他的二进制数相邻的两个位数永不相等。

解题思路

- 判断一个数的二进制位相邻两个数是不相等的，即 `0101` 交叉间隔的，如果是，输出 true。这一题有多种做法，最简单的方法就是直接模拟。比较巧妙的方法是通过位运算，合理构造特殊数据进行位运算到达目的。`010101` 构造出 `101010` 两者相互 `&` 位运算以后就为 0，因为都“插空”了。

代码

```go
package leetcode

// 解法一
func hasAlternatingBits(n int) bool {
  /*
     n =          1 0 1 0 1 0 1 0
     n >> 1       0 1 0 1 0 1 0 1
     n ^ n>>1     1 1 1 1 1 1 1 1
     n            1 1 1 1 1 1 1 1
     n + 1      1 0 0 0 0 0 0 0 0
     n & (n+1)    0 0 0 0 0 0 0 0
  */
  n = n ^ (n >> 1)
  return (n & (n + 1)) == 0
}

// 解法二
func hasAlternatingBits1(n int) bool {
  last, current := 0, 0
  for n > 0 {
    last = n & 1
    n = n / 2
    current = n & 1
    if last == current {
      return false
    }
  }
  return true
}
```

695. Max Area of Island

题目

Given a non-empty 2D array `grid` of 0's and 1's, 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.

Find the maximum area of an island in the given 2D array. (If there is no island, the maximum area is 0.)

Example 1:

```
[[0,0,1,0,0,0,0,1,0,0,0,0,0],
 [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],
 [0,1,0,0,1,1,0,0,1,0,1,0,0],
 [0,1,0,0,1,1,0,0,1,1,1,0,0],
 [0,0,0,0,0,0,0,0,0,0,1,0,0],
 [0,0,0,0,0,0,0,1,1,1,0,0,0],
 [0,0,0,0,0,0,0,1,1,0,0,0,0]]
```

Given the above grid, return `6`. Note the answer is not 11, because the island must be connected 4-directionally.

Example 2:

```
[[0,0,0,0,0,0,0,0]]
```

Given the above grid, return `0`.

Note: The length of each dimension in the given `grid` does not exceed 50.

题目大意

给定一个包含了一些 0 和 1 的非空二维数组 grid 。一个 岛屿 是由一些相邻的 1 (代表土地) 构成的组合，这里的「相邻」要求两个 1 必须在水平或者竖直方向上相邻。你可以假设 grid 的四个边缘都被 0（代表水）包围着。找到给定的二维数组中最大的岛屿面积。(如果没有岛屿，则返回面积为 0 。)

解题思路

- 给出一个地图，要求计算上面岛屿的面积。注意岛屿的定义是四周都是海(为 0 的点)，如果土地(为 1 的点)靠在地图边缘，不能算是岛屿。
- 这一题和第 200 题，第 1254 题解题思路是一致的。DPS 深搜。这不过这一题需要多处理 2 件事情，一个是注意靠边缘的岛屿不能计算在内，二是动态维护岛屿的最大面积。

代码

```go
var dir = [][]int{
  {-1, 0},
  {0, 1},
  {1, 0},
  {0, -1},
}

func maxAreaOfIsland(grid [][]int) int {
  res := 0
  for i, row := range grid {
    for j, col := range row {
      if col == 0 {
```

```go
        continue
      }
      area := areaOfIsland(grid, i, j)
      if area > res {
        res = area
      }
    }
  }
  return res
}

func isInGrid(grid [][]int, x, y int) bool {
  return x >= 0 && x < len(grid) && y >= 0 && y < len(grid[0])
}

func areaOfIsland(grid [][]int, x, y int) int {
  if !isInGrid(grid, x, y) || grid[x][y] == 0 {
    return 0
  }
  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
}
```

696. Count Binary Substrings

题目

Give a string `s`, count the number of non-empty (contiguous) substrings that have the same number of 0's and 1's, and all the 0's and all the 1's in these substrings are grouped consecutively.

Substrings that occur multiple times are counted the number of times they occur.

Example 1:

```
Input: "00110011"
Output: 6
Explanation: There are 6 substrings that have equal number of consecutive 1's and 0's:
"0011", "01", "1100", "10", "0011", and "01".

Notice that some of these substrings repeat and are counted the number of times they
occur.

Also, "00110011" is not a valid substring becauseall the 0's (and 1's) are not grouped
together.
```

Example 2:

```
Input: "10101"
Output: 4
Explanation: There are 4 substrings: "10", "01", "10", "01" that have equal number of
consecutive 1's and 0's.
```

Note:

- `s.length` will be between 1 and 50,000.
- `s` will only consist of "0" or "1" characters.

题目大意

给定一个字符串 s，计算具有相同数量 0 和 1 的非空（连续）子字符串的数量，并且这些子字符串中的所有 0 和所有 1 都是连续的。重复出现的子串要计算它们出现的次数。

解题思路

- 简单题。先分组统计 0 和 1 的个数，例如，`0110001111` 按照 0 和 1 分组统计出来的结果是 [1, 2, 3, 4]。再拼凑结果。相邻 2 组取两者最短的，例如 `0110001111`，凑成的结果应该是 min(1,2)，min(2,3)，min(3,4)，即 `01`，`01`，`10`，`1100`，`0011`，`000111`。时间复杂度 O(n)，空间复杂度 O(1)。

代码

```go
package leetcode

func countBinarySubstrings(s string) int {
  last, res := 0, 0
  for i := 0; i < len(s); {
    c, count := s[i], 1
    for i++; i < len(s) && s[i] == c; i++ {
      count++
    }
```

```
    res += min(count, last)
    last = count
  }
  return res
}

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

697. Degree of an Array

题目

Given a non-empty array of non-negative integers `nums`, the **degree** of this array is defined as the maximum frequency of any one of its elements.

Your task is to find the smallest possible length of a (contiguous) subarray of `nums`, that has the same degree as `nums`.

Example 1:

```
Input: [1, 2, 2, 3, 1]
Output: 2
Explanation:
The input array has a degree of 2 because both elements 1 and 2 appear twice.
Of the subarrays that have the same degree:
[1, 2, 2, 3, 1], [1, 2, 2, 3], [2, 2, 3, 1], [1, 2, 2], [2, 2, 3], [2, 2]
The shortest length is 2. So return 2.
```

Example 2:

```
Input: [1,2,2,3,1,4,2]
Output: 6
```

Note:

- `nums.length` will be between 1 and 50,000.
- `nums[i]` will be an integer between 0 and 49,999.

题目大意

给定一个非空且只包含非负数的整数数组 nums, 数组的度的定义是指数组里任一元素出现频数的最大值。你的任务是找到与 nums 拥有相同大小的度的最短连续子数组，返回其长度。

注意:

- nums.length 在 1 到 50,000 区间范围内。
- nums[i] 是一个在 0 到 49,999 范围内的整数。

解题思路

- 找一个与给定数组相同度的最短连续子数组，输出其长度。数组的度的定义是任一元素出现频数的最大值。
- 简单题。先统计各个元素的频次，并且动态维护最大频次和子数组的起始和终点位置。这里最短连续子数组有点“迷惑人”。这个最短子数组其实处理起来很简单。只需从前往后扫一遍，记录各个元素第一次出现的位置和最后一次出现的位置即是最短的连续子数组。然后在频次字典里面寻找和最大频次相同的所有解，有可能有多个子数组能满足题意，取出最短的输出即可。

代码

```go
package leetcode

func findShortestSubArray(nums []int) int {
  frequency, maxFreq, smallest := map[int][]int{}, 0, len(nums)
  for i, num := range nums {
    if _, found := frequency[num]; !found {
      frequency[num] = []int{1, i, i}
    } else {
      frequency[num][0]++
      frequency[num][2] = i
    }
    if maxFreq < frequency[num][0] {
      maxFreq = frequency[num][0]
    }
  }
  for _, indices := range frequency {
    if indices[0] == maxFreq {
      if smallest > indices[2]-indices[1]+1 {
        smallest = indices[2] - indices[1] + 1
      }
    }
  }
  return smallest
}
```

699. Falling Squares

题目

On an infinite number line (x-axis), we drop given squares in the order they are given.

The `i`-th square dropped (`positions[i] = (left, side_length)`) is a square with the left-most point being `positions[i][0]` and sidelength `positions[i][1]`.

The square is dropped with the bottom edge parallel to the number line, and from a higher height than all currently landed squares. We wait for each square to stick before dropping the next.

The squares are infinitely sticky on their bottom edge, and will remain fixed to any positive length surface they touch (either the number line or another square). Squares dropped adjacent to each other will not stick together prematurely.

Return a list `ans` of heights. Each height `ans[i]` represents the current highest height of any square we have dropped, after dropping squares represented by `positions[0], positions[1], ..., positions[i]`.

Example 1:

```
Input: [[1, 2], [2, 3], [6, 1]]
Output: [2, 5, 5]
Explanation:
```

After the first drop of `positions[0] = [1, 2]: _aa _aa -------` The maximum height of any square is 2.

After the second drop of `positions[1] = [2, 3]: __aaa __aaa __aaa _aa__ _aa__ --------------` The maximum height of any square is 5. The larger square stays on top of the smaller square despite where its center of gravity is, because squares are infinitely sticky on their bottom edge.

After the third drop of `positions[1] = [6, 1]: __aaa __aaa __aaa _aa _aa___a --------------` The maximum height of any square is still 5. Thus, we return an answer of `[2, 5, 5]`.

Example 2:

```
Input: [[100, 100], [200, 100]]
Output: [100, 100]
Explanation: Adjacent squares don't get stuck prematurely - only their bottom edge can
stick to surfaces.
```

Note:

- `1 <= positions.length <= 1000`.
- `1 <= positions[i][0] <= 10^8`.
- `1 <= positions[i][1] <= 10^6`.

题目大意

在无限长的数轴（即 x 轴）上，我们根据给定的顺序放置对应的正方形方块。第 i 个掉落的方块（positions[i] = (left, side_length)）是正方形，其中 left 表示该方块最左边的点位置(positions[i][0])，side_length 表示该方块的边长(positions[i][1])。

每个方块的底部边缘平行于数轴（即 x 轴），并且从一个比目前所有的落地方块更高的高度掉落而下。在上一个方块结束掉落，并保持静止后，才开始掉落新方块。方块的底边具有非常大的粘性，并将保持固定在它们所接触的任何长度表面上（无论是数轴还是其他方块）。邻接掉落的边不会过早地粘合在一起，因为只有底边才具有粘性。

返回一个堆叠高度列表 ans 。每一个堆叠高度 ans[i] 表示在通过 positions[0], positions[1], ..., positions[i] 表示的方块掉落结束后，目前所有已经落稳的方块堆叠的最高高度。

示例 1:

```
输入: [[1, 2], [2, 3], [6, 1]]
输出: [2, 5, 5]
解释:

第一个方块 positions[0] = [1, 2] 掉落:
_aa
_aa
-------
方块最大高度为 2 。

第二个方块 positions[1] = [2, 3] 掉落:
__aaa
__aaa
__aaa
_aa__
_aa__
--------------
方块最大高度为5。
大的方块保持在较小的方块的顶部，不论它的重心在哪里，因为方块的底部边缘有非常大的粘性。

第三个方块 positions[1] = [6, 1] 掉落:
__aaa
__aaa
__aaa
_aa
_aa___a
--------------
方块最大高度为5。

因此，我们返回结果[2, 5, 5]。
```

注意:

- 1 <= positions.length <= 1000.
- 1 <= positions[i][0] <= 10^8.
- 1 <= positions[i][1] <= 10^6.

解题思路

- 给出一个二维数组，每个一维数组中只有 2 个值，分别代表的是正方形砖块所在 x 轴的坐标起始点，和边长。要求输出每次砖块落下以后，当前最大的高度。正方形砖块落下如同俄罗斯方块，落下的过程中如果遇到了砖块会落在砖块的上面。如果砖块摞起来了以后，下方有空间，是不可能再把砖块挪进去的，因为此题砖块只会垂直落下，不会水平移动(这一点和俄罗斯方块不同)。
- 这一题可以用线段树解答。由于方块在 x 轴上的坐标范围特别大，如果不离散化，这一题就会 MTE。所以首先去重 - 排序 - 离散化。先把每个砖块所在区间都算出来，每个正方形的方块所在区间是 `[pos[0] , pos[0]+pos[1]-1]` ，为什么右边界要减一呢？因为每个方块占据的区间其实应该是左闭右开的，即 `[pos[0] , pos[0]+pos[1])`，如果右边是开的，那么这个边界会被 2 个区间查询共用，从而导致错误结果。例如 [2,3]，[3,4]，这两个区间的砖块实际是不会摞在一起的。但是如果右边都是闭区间，用线段树 query 查询的时候，会都找到 [3，3]，从而导致这两个区间都会判断 3 这一点的情况。正确的做法应该是 [2,3)，[3,4）这样就避免了上述可能导致错误的情况了。离散化以后，所有的坐标区间都在 0~n 之间了。
- 遍历每个砖块所在区间，先查询这个区间内的值，再加上当前砖块的高度，即为这个区间的最新高度。并更新该区间的值。更新区间的值用到懒惰更新。然后和动态维护的当前最大高度进行比较，将最大值放入最终输出的数组中。
- 类似的题目有：第 715 题，第 218 题，第 732 题。第 715 题是区间更新定值(**不是增减**)，第 218 题可以用扫描线，第 732 题和本题类似，也是俄罗斯方块的题目，但是第 732 题的俄罗斯方块的方块会“断裂”。
- leetcode 上也有线段树的讲解：[Get Solutions to Interview Questions]

代码

```go
package leetcode

import (
  "sort"

  "github.com/halfrost/LeetCode-Go/template"
)

func fallingSquares(positions [][]int) []int {
  st, ans, posMap, maxHeight := template.SegmentTree{}, make([]int, 0, len(positions)),
discretization(positions), 0
  tmp := make([]int, len(posMap))
  st.Init(tmp, func(i, j int) int {
    return max(i, j)
  })
  for _, p := range positions {
    h := st.QueryLazy(posMap[p[0]], posMap[p[0]+p[1]-1]) + p[1]
    st.UpdateLazy(posMap[p[0]], posMap[p[0]+p[1]-1], h)
    maxHeight = max(maxHeight, h)
    ans = append(ans, maxHeight)
  }
  return ans
}

func discretization(positions [][]int) map[int]int {
  tmpMap, posArray, posMap := map[int]int{}, []int{}, map[int]int{}
  for _, pos := range positions {
```

```
    tmpMap[pos[0]]++
    tmpMap[pos[0]+pos[1]-1]++
  }
  for k := range tmpMap {
    posArray = append(posArray, k)
  }
  sort.Ints(posArray)
  for i, pos := range posArray {
    posMap[pos] = i
  }
  return posMap
}
```

703. Kth Largest Element in a Stream

题目

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)` 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.

题目大意

设计一个找到数据流中第 k 大元素的类（class）。注意是排序后的第 k 大元素，不是第 k 个不同的元素。请实现 KthLargest 类：

- KthLargest(int k, int[] nums) 使用整数 k 和整数流 nums 初始化对象。
- int add(int val) 将 val 插入数据流 nums 后，返回当前数据流中第 k 大的元素。

解题思路

- 读完题就能明白这一题考察的是最小堆。构建一个长度为 K 的最小堆，每次 pop 堆首(堆中最小的元素)，维护堆首即为第 K 大元素。
- 这里有一个简洁的写法，常规的构建一个 pq 优先队列需要自己新建一个类型，然后实现 Len()、Less()、Swap()、Push()、Pop() 这 5 个方法。在 sort 包里有一个现成的最小堆，sort.IntSlice。可以借用它，再自己实现 Push()、Pop()就可以使用最小堆了，节约一部分代码。

代码

```go
package leetcode

import (
  "container/heap"
  "sort"
)

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) Push(v interface{}) {
  kl.IntSlice = append(kl.IntSlice, v.(int))
}
```

```go
func (kl *KthLargest) Pop() interface{} {
  a := kl.IntSlice
  v := a[len(a)-1]
  kl.IntSlice = a[:len(a)-1]
  return v
}

func (kl *KthLargest) Add(val int) int {
  heap.Push(kl, val)
  if kl.Len() > kl.k {
    heap.Pop(kl)
  }
  return kl.IntSlice[0]
}
```

704. Binary Search

题目

Given a **sorted** (in ascending order) integer array `nums` of `n` elements and a `target` value, write a function to search `target` in `nums`. If `target` exists, then return its index, otherwise return `-1`.

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
```

Note:

1. You may assume that all elements in `nums` are unique.
2. `n` will be in the range `[1, 10000]`.
3. The value of each element in `nums` will be in the range `[-9999, 9999]`.

题目大意

给定一个 n 个元素有序的（升序）整型数组 nums 和一个目标值 target ，写一个函数搜索 nums 中的 target，如果目标值存在返回下标，否则返回 -1。

提示：

- 你可以假设 nums 中的所有元素是不重复的。

- n 将在 [1, 10000]之间。
- nums 的每个元素都将在 [-9999, 9999]之间。

解题思路

- 给出一个数组，要求在数组中搜索等于 target 的元素的下标。如果找到就输出下标，如果找不到输出 -1 。
- 简单题，二分搜索的裸题。

代码

```
package leetcode

func search704(nums []int, target int) int {
  low, high := 0, len(nums)-1
  for low <= high {
    mid := low + (high-low)>>1
    if nums[mid] == target {
      return mid
    } else if nums[mid] > target {
      high = mid - 1
    } else {
      low = mid + 1
    }
  }
  return -1
}
```

705. Design HashSet

题目

Design a HashSet without using any built-in hash table libraries.

To be specific, your design should include these functions:

- `add(value)`: Insert a value into the HashSet.
- `contains(value)` : Return whether the value exists in the HashSet or not.
- `remove(value)`: Remove a value in the HashSet. If the value does not exist in the HashSet, do nothing.

Example:

```
MyHashSet hashSet = new MyHashSet();
hashSet.add(1);
hashSet.add(2);
hashSet.contains(1);    // returns true
hashSet.contains(3);    // returns false (not found)
hashSet.add(2);
hashSet.contains(2);    // returns true
hashSet.remove(2);
hashSet.contains(2);    // returns false (already removed)
```

Note:

- All values will be in the range of `[0, 1000000]`.
- The number of operations will be in the range of `[1, 10000]`.
- Please do not use the built-in HashSet library.

题目大意

不使用任何内建的哈希表库设计一个哈希集合具体地说，你的设计应该包含以下的功能：

- add(value)：向哈希集合中插入一个值。
- contains(value) ：返回哈希集合中是否存在这个值。
- remove(value)：将给定值从哈希集合中删除。如果哈希集合中没有这个值，什么也不做。

注意：

- 所有的值都在 [1, 1000000] 的范围内。
- 操作的总数目在 [1, 10000] 范围内。
- 不要使用内建的哈希集合库。

解题思路

- 简单题，设计一个 hashset 的数据结构，要求有 `add(value)`，`contains(value)`，`remove(value)`，这 3 个方法。

代码

```go
package leetcode

type MyHashSet struct {
  data []bool
}

/** Initialize your data structure here. */
func Constructor705() MyHashSet {
  return MyHashSet{
    data: make([]bool, 1000001),
```

```go
  }
}

func (this *MyHashSet) Add(key int) {
  this.data[key] = true
}

func (this *MyHashSet) Remove(key int) {
  this.data[key] = false
}

/** Returns true if this set contains the specified element */
func (this *MyHashSet) Contains(key int) bool {
  return this.data[key]
}

/**
 * Your MyHashSet object will be instantiated and called as such:
 * obj := Constructor();
 * obj.Add(key);
 * obj.Remove(key);
 * param_3 := obj.Contains(key);
 */
```

706. Design HashMap

题目

Design a HashMap without using any built-in hash table libraries.

To be specific, your design should include these functions:

- `put(key, value)` : Insert a (key, value) pair into the HashMap. If the value already exists in the HashMap, update the value.
- `get(key)` : Returns the value to which the specified key is mapped, or -1 if this map contains no mapping for the key.
- `remove(key)` : Remove the mapping for the value key if this map contains the mapping for the key.

Example:

```
MyHashMap hashMap = new MyHashMap();
hashMap.put(1, 1);
hashMap.put(2, 2);
hashMap.get(1);            // returns 1
hashMap.get(3);            // returns -1 (not found)
hashMap.put(2, 1);          // update the existing value
hashMap.get(2);            // returns 1
hashMap.remove(2);          // remove the mapping for 2
hashMap.get(2);            // returns -1 (not found)
```

Note:

- All keys and values will be in the range of `[0, 1000000]`.
- The number of operations will be in the range of `[1, 10000]`.
- Please do not use the built-in HashMap library.

题目大意

不使用任何内建的哈希表库设计一个哈希映射具体地说，你的设计应该包含以下的功能：

- put(key, value)：向哈希映射中插入(键,值)的数值对。如果键对应的值已经存在，更新这个值。
- get(key)：返回给定的键所对应的值，如果映射中不包含这个键，返回 -1。
- remove(key)：如果映射中存在这个键，删除这个数值对。

注意：

- 所有的值都在 [1, 1000000] 的范围内。
- 操作的总数目在 [1, 10000] 范围内。
- 不要使用内建的哈希库。

解题思路

- 简单题，设计一个 hashmap 的数据结构，要求有 `put(key, value)`，`get(key)`，`remove(key)`，这 3 个方法。设计一个 map 主要需要处理哈希冲突，一般都是链表法解决冲突。

代码

```go
package leetcode

const Len int = 100000

type MyHashMap struct {
  content [Len]*HashNode
}

type HashNode struct {
  key  int
  val  int
```

```go
  next *HashNode
}

func (N *HashNode) Put(key int, value int) {
  if N.key == key {
    N.val = value
    return
  }
  if N.next == nil {
    N.next = &HashNode{key, value, nil}
    return
  }
  N.next.Put(key, value)
}

func (N *HashNode) Get(key int) int {
  if N.key == key {
    return N.val
  }
  if N.next == nil {
    return -1
  }
  return N.next.Get(key)
}

func (N *HashNode) Remove(key int) *HashNode {
  if N.key == key {
    p := N.next
    N.next = nil
    return p
  }
  if N.next != nil {
    return N.next.Remove(key)
  }
  return nil
}

/** Initialize your data structure here. */
func Constructor706() MyHashMap {
  return MyHashMap{}
}

/** value will always be non-negative. */
func (this *MyHashMap) Put(key int, value int) {
  node := this.content[this.Hash(key)]
  if node == nil {
    this.content[this.Hash(key)] = &HashNode{key: key, val: value, next: nil}
    return
  }
```

```go
  node.Put(key, value)
}

/** Returns the value to which the specified key is mapped, or -1 if this map contains
no mapping for the key */
func (this *MyHashMap) Get(key int) int {
  HashNode := this.content[this.Hash(key)]
  if HashNode == nil {
    return -1
  }
  return HashNode.Get(key)
}

/** Removes the mapping of the specified value key if this map contains a mapping for
the key */
func (this *MyHashMap) Remove(key int) {
  HashNode := this.content[this.Hash(key)]
  if HashNode == nil {
    return
  }
  this.content[this.Hash(key)] = HashNode.Remove(key)
}

func (this *MyHashMap) Hash(value int) int {
  return value % Len
}

/**
 * Your MyHashMap object will be instantiated and called as such:
 * obj := Constructor();
 * obj.Put(key,value);
 * param_2 := obj.Get(key);
 * obj.Remove(key);
 */
```

707. Design Linked List

题目

Design your implementation of the linked list. You can choose to use the singly linked list or the doubly linked list. A node in a singly linked list should have two attributes: val and next. val is the value of the current node, and next is a pointer/reference to the next node. If you want to use the doubly linked list, you will need one more attribute prev to indicate the previous node in the linked list. Assume all nodes in the linked list are 0-indexed.

Implement these functions in your linked list class:

- get(index) : Get the value of the index-th node in the linked list. If the index is invalid, return -1.

- addAtHead(val) : Add a node of value val before the first element of the linked list. After the insertion, the new node will be the first node of the linked list.
- addAtTail(val) : Append a node of value val to the last element of the linked list.
- addAtIndex(index, val) : Add a node of value val before the index-th node in the linked list. If index equals to the length of linked list, the node will be appended to the end of linked list. If index is greater than the length, the node will not be inserted.
- deleteAtIndex(index) : Delete the index-th node in the linked list, if the index is valid.

Example:

```
MyLinkedList linkedList = new MyLinkedList();
linkedList.addAtHead(1);
linkedList.addAtTail(3);
linkedList.addAtIndex(1, 2);  // linked list becomes 1->2->3
linkedList.get(1);            // returns 2
linkedList.deleteAtIndex(1);  // now the linked list is 1->3
linkedList.get(1);            // returns 3
```

Note:

- All values will be in the range of [1, 1000].
- The number of operations will be in the range of [1, 1000].
- Please do not use the built-in LinkedList library.

题目大意

这道题比较简单，设计一个链表，实现相关操作即可。

解题思路

这题有一个地方比较坑，题目中 Note 里面写的数值取值范围是 [1, 1000]，笔者把 0 当做无效值。结果 case 里面出现了 0 是有效值。case 和题意不符。

代码

```go
package leetcode

type MyLinkedList struct {
  Val  int
  Next *MyLinkedList
}

/** Initialize your data structure here. */
func Constructor() MyLinkedList {
  return MyLinkedList{Val: -999, Next: nil}
}
```

```go
/** Get the value of the index-th node in the linked list. If the index is invalid, return -1. */
func (this *MyLinkedList) Get(index int) int {
  cur := this
  for i := 0; cur != nil; i++ {
    if i == index {
      if cur.Val == -999 {
        return -1
      } else {
        return cur.Val
      }
    }
    cur = cur.Next
  }
  return -1
}

/** Add a node of value val before the first element of the linked list. After the insertion, the new node will be the first node of the linked list. */
func (this *MyLinkedList) AddAtHead(val int) {
  if this.Val == -999 {
    this.Val = val
  } else {
    tmp := &MyLinkedList{Val: this.Val, Next: this.Next}
    this.Val = val
    this.Next = tmp
  }
}

/** Append a node of value val to the last element of the linked list. */
func (this *MyLinkedList) AddAtTail(val int) {
  cur := this
  for cur.Next != nil {
    cur = cur.Next
  }
  tmp := &MyLinkedList{Val: val, Next: nil}
  cur.Next = tmp
}

/** Add a node of value val before the index-th node in the linked list. If index equals to the length of linked list, the node will be appended to the end of linked list. If index is greater than the length, the node will not be inserted. */
func (this *MyLinkedList) AddAtIndex(index int, val int) {
  cur := this
  if index == 0 {
    this.AddAtHead(val)
    return
  }
```

```go
  for i := 0; cur != nil; i++ {
    if i == index-1 {
      break
    }
    cur = cur.Next
  }
  if cur != nil && cur.Val != -999 {
    tmp := &MyLinkedList{Val: val, Next: cur.Next}
    cur.Next = tmp
  }
}

/** Delete the index-th node in the linked list, if the index is valid. */
func (this *MyLinkedList) DeleteAtIndex(index int) {
  cur := this
  for i := 0; cur != nil; i++ {
    if i == index-1 {
      break
    }
    cur = cur.Next
  }
  if cur != nil && cur.Next != nil {
    cur.Next = cur.Next.Next
  }
}

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * obj := Constructor();
 * param_1 := obj.Get(index);
 * obj.AddAtHead(val);
 * obj.AddAtTail(val);
 * obj.AddAtIndex(index,val);
 * obj.DeleteAtIndex(index);
 */
```

709. To Lower Case

题目

Given a string `s`, return *the string after replacing every uppercase letter with the same lowercase letter*.

Example 1:

```
Input: s = "Hello"
Output: "hello"
```

Example 2:

```
Input: s = "here"
Output: "here"
```

Example 3:

```
Input: s = "LOVELY"
Output: "lovely"
```

Constraints:

- `1 <= s.length <= 100`
- `s` consists of printable ASCII characters.

题目大意

给你一个字符串 s ，将该字符串中的大写字母转换成相同的小写字母，返回新的字符串。

解题思路

- 简单题，将字符串中的大写字母转换成小写字母。

代码

```go
func toLowerCase(s string) string {
    runes := [] rune(s)
    diff := 'a' - 'A'
    for i := 0; i < len(s); i++ {
        if runes[i] >= 'A' && runes[i] <= 'Z' {
            runes[i] += diff
        }
    }
    return string(runes)
}
```

710. Random Pick with Blacklist

题目

Given a blacklist B containing unique integers from [0, N), write a function to return a uniform random integer from [0, N) which is NOT in B.

Optimize it such that it minimizes the call to system's Math.random().

Note:

1. 1 <= N <= 1000000000

2. 0 <= B.length < min(100000, N)
3. [0, N) does NOT include N. See interval notation.

Example 1:

```
Input:
["Solution","pick","pick","pick"]
[[1,[]],[],[],[]]
Output: [null,0,0,0]
```

Example 2:

```
Input:
["Solution","pick","pick","pick"]
[[2,[]],[],[],[]]
Output: [null,1,1,1]
```

Example 3:

```
Input:
["Solution","pick","pick","pick"]
[[3,[1]],[],[],[]]
Output: [null,0,0,2]
```

Example 4:

```
Input:
["Solution","pick","pick","pick"]
[[4,[2]],[],[],[]]
Output: [null,1,3,1]
```

Explanation of Input Syntax:

The input is two lists: the subroutines called and their arguments. Solution's constructor has two arguments, N and the blacklist B. pick has no arguments. Arguments are always wrapped with a list, even if there aren't any.

题目大意

给一个数字 N，再给一个黑名单 B，要求在 [0,N) 区间内随机输出一个数字，这个是不在黑名单 B 中的任意一个数字。

解题思路

这道题的 N 的范围特别大，最大是 10 亿。如果利用桶计数，开不出来这么大的数组。考虑到题目要求我们输出的数字是随机的，所以不需要存下所有的白名单的数字。

假设 N=10, blacklist=[3, 5, 8, 9]

这一题有点类似 hash 冲突的意思。如果随机访问一个数，这个数正好在黑名单之内，那么就 hash 冲突了，我们就把它映射到另外一个不在黑名单里面的数中。如上图，我们可以将 3，5 重新映射到 7，6 的位置。这样末尾开始的几个数要么是黑名单里面的数，要么就是映射的数字。

hash 表总长度应该为 M = N - len(backlist)，然后在 M 的长度中扫描是否有在黑名单中的数，如果有，就代表 hash 冲突了。冲突就把这个数字映射到 (M,N) 这个区间范围内。为了提高效率，可以选择这个区间的头部或者尾部开始映射，我选择的是末尾开始映射。从 (M,N) 这个区间的末尾开始往前找，找黑名单不存在的数，找到了就把 [0,M] 区间内冲突的数字映射到这里来。最后 pick 的时候，只需要查看 map 中是否存在映射关系，如果存在就输出 map 中映射之后的值，如果没有就代表没有冲突，直接输出那个 index 即可。

代码

```go
package leetcode

import "math/rand"

type Solution struct {
  M        int
  BlackMap map[int]int
}

func Constructor710(N int, blacklist []int) Solution {
  blackMap := map[int]int{}
  for i := 0; i < len(blacklist); i++ {
    blackMap[blacklist[i]] = 1
  }
  M := N - len(blacklist)
  for _, value := range blacklist {
```

```go
    if value < M {
      for {
        if _, ok := blackMap[N-1]; ok {
          N--
        } else {
          break
        }
      }
      blackMap[value] = N - 1
      N--
    }
  }
  return Solution{BlackMap: blackMap, M: M}
}

func (this *Solution) Pick() int {
  idx := rand.Intn(this.M)
  if _, ok := this.BlackMap[idx]; ok {
    return this.BlackMap[idx]
  }
  return idx
}

/**
 * Your Solution object will be instantiated and called as such:
 * obj := Constructor(N, blacklist);
 * param_1 := obj.Pick();
 */
```

713. Subarray Product Less Than K

题目

Your are given an array of positive integers nums.

Count and print the number of (contiguous) subarrays where the product of all the elements in the subarray is less than k.

Example 1:

```
Input: nums = [10, 5, 2, 6], k = 100
Output: 8
Explanation: The 8 subarrays that have product less than 100 are: [10], [5], [2], [6],
[10, 5], [5, 2], [2, 6], [5, 2, 6].
Note that [10, 5, 2] is not included as the product of 100 is not strictly less than k.
```

Note:

- 0 < nums.length <= 50000.
- 0 < nums[i] < 1000.
- 0 <= k < 10^6.

题目大意

给出一个数组，要求在输出符合条件的窗口数，条件是，窗口中所有数字乘积小于 K 。

解题思路

这道题也是滑动窗口的题目，在窗口滑动的过程中不断累乘，直到乘积大于 k，大于 k 的时候就缩小左窗口。有一种情况还需要单独处理一下，即类似 [100] 这种情况。这种情况窗口内乘积等于 k，不小于 k，左边窗口等于右窗口，这个时候需要左窗口和右窗口同时右移。

代码

```go
package leetcode

func numSubarrayProductLessThanK(nums []int, k int) int {
  if len(nums) == 0 {
    return 0
  }
  res, left, right, prod := 0, 0, 0, 1
  for left < len(nums) {
    if right < len(nums) && prod*nums[right] < k {
      prod = prod * nums[right]
      right++
    } else if left == right {
      left++
      right++
    } else {
      res += right - left
      prod = prod / nums[left]
      left++
    }
  }
  return res
}
```

714. Best Time to Buy and Sell Stock with Transaction Fee

题目

Your are given an array of integers `prices`, for which the `i`-th element is the price of a given stock on day `i`; and a non-negative integer `fee` representing a transaction fee.

You may complete as many transactions as you like, but you need to pay the transaction fee for each transaction. You may not buy more than 1 share of a stock at a time (ie. you must sell the stock share before you buy again.)

Return the maximum profit you can make.

Example 1:

```
Input: prices = [1, 3, 2, 8, 4, 9], fee = 2
Output: 8
Explanation: The maximum profit can be achieved by:
Buying at prices[0] = 1
Selling at prices[3] = 8
Buying at prices[4] = 4
Selling at prices[5] = 9
The total profit is ((8 - 1) - 2) + ((9 - 4) - 2) = 8.
```

Note:

- `0 < prices.length <= 50000`.
- `0 < prices[i] < 50000`.
- `0 <= fee < 50000`.

题目大意

给定一个整数数组 prices，其中第 i 个元素代表了第 i 天的股票价格 ；非负整数 fee 代表了交易股票的手续费用。你可以无限次地完成交易，但是你每次交易都需要付手续费。如果你已经购买了一个股票，在卖出它之前你就不能再继续购买股票了。要求返回获得利润的最大值。

解题思路

- 给定一个数组，表示一支股票在每一天的价格。设计一个交易算法，在这些天进行自动交易，要求：每一天只能进行一次操作；在买完股票后，必须卖了股票，才能再次买入；每次卖了股票以后，需要缴纳一部分的手续费。问如何交易，能让利润最大?
- 这一题是第 121 题、第 122 题、第 309 题的变种题。
- 这一题的解题思路是 DP，需要维护买和卖的两种状态。`buy[i]` 代表第 `i` 天买入的最大收益，`sell[i]` 代表第 `i` 天卖出的最大收益，状态转移方程是 `buy[i] = max(buy[i-1], sell[i-1]-prices[i])`，`sell[i] = max(sell[i-1], buy[i-1]+prices[i]-fee)`。

代码

```go
package leetcode

import (
  "math"
)

// 解法一 模拟 DP
func maxProfit714(prices []int, fee int) int {
  if len(prices) <= 1 {
    return 0
  }
  buy, sell := make([]int, len(prices)), make([]int, len(prices))
  for i := range buy {
    buy[i] = math.MinInt64
  }
  buy[0] = -prices[0]
  for i := 1; i < len(prices); i++ {
    buy[i] = max(buy[i-1], sell[i-1]-prices[i])
    sell[i] = max(sell[i-1], buy[i-1]+prices[i]-fee)
  }
  return sell[len(sell)-1]
}

// 解法二 优化辅助空间的 DP
func maxProfit714_1(prices []int, fee int) int {
  sell, buy := 0, -prices[0]
  for i := 1; i < len(prices); i++ {
    sell = max(sell, buy+prices[i]-fee)
    buy = max(buy, sell-prices[i])
  }
  return sell
}
```

715. Range Module

题目

A Range Module is a module that tracks ranges of numbers. Your task is to design and implement the following interfaces in an efficient manner.

- `addRange(int left, int right)` Adds the half-open interval `[left, right)`, tracking every real number in that interval. Adding an interval that partially overlaps with currently tracked numbers should add any numbers in the interval `[left, right)` that are not already tracked.
- `queryRange(int left, int right)` Returns true if and only if every real number in the interval `[left, right)` is currently being tracked.
- `removeRange(int left, int right)` Stops tracking every real number currently being tracked in the interval `[left, right)`.

Example 1:

```
addRange(10, 20): null
removeRange(14, 16): null
queryRange(10, 14): true (Every number in [10, 14) is being tracked)
queryRange(13, 15): false (Numbers like 14, 14.03, 14.17 in [13, 15) are not being
tracked)
queryRange(16, 17): true (The number 16 in [16, 17) is still being tracked, despite the
remove operation)
```

Note:

- A half open interval `[left, right)` denotes all real numbers `left <= x < right`.
- `0 < left < right < 10^9` in all calls to `addRange, queryRange, removeRange`.
- The total number of calls to `addRange` in a single test case is at most `1000`.
- The total number of calls to `queryRange` in a single test case is at most `5000`.
- The total number of calls to `removeRange` in a single test case is at most `1000`.

题目大意

Range 模块是跟踪数字范围的模块。你的任务是以一种有效的方式设计和实现以下接口。

- addRange(int left, int right) 添加半开区间 [left, right)，跟踪该区间中的每个实数。添加与当前跟踪的数字部分重叠的区间时，应当添加在区间 [left, right) 中尚未跟踪的任何数字到该区间中。
- queryRange(int left, int right) 只有在当前正在跟踪区间 [left, right) 中的每一个实数时，才返回 true。
- removeRange(int left, int right) 停止跟踪区间 [left, right) 中当前正在跟踪的每个实数。

示例：

```
addRange(10, 20): null
removeRange(14, 16): null
queryRange(10, 14): true （区间 [10, 14) 中的每个数都正在被跟踪）
queryRange(13, 15): false （未跟踪区间 [13, 15) 中像 14, 14.03, 14.17 这样的数字）
queryRange(16, 17): true （尽管执行了删除操作，区间 [16, 17) 中的数字 16 仍然会被跟踪）
```

提示：

- 半开区间 [left, right) 表示所有满足 left <= x < right 的实数。
- 对 addRange, queryRange, removeRange 的所有调用中 0 < left < right < 10^9。
- 在单个测试用例中，对 addRange 的调用总数不超过 1000 次。
- 在单个测试用例中，对 queryRange 的调用总数不超过 5000 次。
- 在单个测试用例中，对 removeRange 的调用总数不超过 1000 次。

解题思路

- 设计一个数据结构，能完成添加区间 `addRange`，查询区间 `queryRange`，移除区间 `removeRange` 三种操作。查询区间的操作需要更加高效一点。
- 这一题可以用线段树来解答，但是时间复杂度不高，最优解是用二叉排序树 BST 来解答。先来看线段树。这一题是更新区间内的值，所以需要用到懒惰更新。添加区间可以把区间内的值都赋值为 1 。由于题目中未预

先确定区间范围，选用树的形式实现线段树比数组实现更加节约空间(当然用数组也可以，区间最大是 1000，点至多有 2000 个)。移除区间的时候就是把区间内的值都赋值标记为 0 。
- 类似的题目有：第 699 题，第 218 题，第 732 题。第 715 题是区间更新定值(**不是增减**)，第 218 题可以用扫描线，第 732 题和第 699 题类似，也是俄罗斯方块的题目，但是第 732 题的俄罗斯方块的方块会“断裂”。

代码

```
package leetcode

// RangeModule define
type RangeModule struct {
  Root *SegmentTreeNode
}

// SegmentTreeNode define
type SegmentTreeNode struct {
  Start, End  int
  Tracked     bool
  Lazy        int
  Left, Right *SegmentTreeNode
}

// Constructor715 define
func Constructor715() RangeModule {
  return RangeModule{&SegmentTreeNode{0, 1e9, false, 0, nil, nil}}
}

// AddRange define
func (rm *RangeModule) AddRange(left int, right int) {
  update(rm.Root, left, right-1, true)
}

// QueryRange define
func (rm *RangeModule) QueryRange(left int, right int) bool {
  return query(rm.Root, left, right-1)
}

// RemoveRange define
func (rm *RangeModule) RemoveRange(left int, right int) {
  update(rm.Root, left, right-1, false)
}

func lazyUpdate(node *SegmentTreeNode) {
  if node.Lazy != 0 {
    node.Tracked = node.Lazy == 2
  }
  if node.Start != node.End {
    if node.Left == nil || node.Right == nil {
```

```go
      m := node.Start + (node.End-node.Start)/2
      node.Left = &SegmentTreeNode{node.Start, m, node.Tracked, 0, nil, nil}
      node.Right = &SegmentTreeNode{m + 1, node.End, node.Tracked, 0, nil, nil}
    } else if node.Lazy != 0 {
      node.Left.Lazy = node.Lazy
      node.Right.Lazy = node.Lazy
    }
  }
  node.Lazy = 0
}

func update(node *SegmentTreeNode, start, end int, track bool) {
  lazyUpdate(node)
  if start > end || node == nil || end < node.Start || node.End < start {
    return
  }
  if start <= node.Start && node.End <= end {
    // segment completely covered by the update range
    node.Tracked = track
    if node.Start != node.End {
      if track {
        node.Left.Lazy = 2
        node.Right.Lazy = 2
      } else {
        node.Left.Lazy = 1
        node.Right.Lazy = 1
      }
    }
    return
  }
  update(node.Left, start, end, track)
  update(node.Right, start, end, track)
  node.Tracked = node.Left.Tracked && node.Right.Tracked
}

func query(node *SegmentTreeNode, start, end int) bool {
  lazyUpdate(node)
  if start > end || node == nil || end < node.Start || node.End < start {
    return true
  }
  if start <= node.Start && node.End <= end {
    // segment completely covered by the update range
    return node.Tracked
  }
  return query(node.Left, start, end) && query(node.Right, start, end)
}

// 解法二 BST
// type RangeModule struct {
```

```
//  Root *BSTNode
// }

// type BSTNode struct {
//  Interval    []int
//  Left, Right *BSTNode
// }

// func Constructor715() RangeModule {
//  return RangeModule{}
// }

// func (this *RangeModule) AddRange(left int, right int) {
//  interval := []int{left, right - 1}
//  this.Root = insert(this.Root, interval)
// }

// func (this *RangeModule) RemoveRange(left int, right int) {
//  interval := []int{left, right - 1}
//  this.Root = delete(this.Root, interval)
// }

// func (this *RangeModule) QueryRange(left int, right int) bool {
//  return query(this.Root, []int{left, right - 1})
// }

// func (this *RangeModule) insert(root *BSTNode, interval []int) *BSTNode {
//  if root == nil {
//    return &BSTNode{interval, nil, nil}
//  }
//  if root.Interval[0] <= interval[0] && interval[1] <= root.Interval[1] {
//    return root
//  }
//  if interval[0] < root.Interval[0] {
//    root.Left = insert(root.Left, []int{interval[0], min(interval[1],
root.Interval[0]-1)})
//  }
//  if root.Interval[1] < interval[1] {
//    root.Right = insert(root.Right, []int{max(interval[0], root.Interval[1]+1),
interval[1]})
//  }
//  return root
// }

// func (this *RangeModule) delete(root *BSTNode, interval []int) *BSTNode {
//  if root == nil {
//    return nil
//  }
//  if interval[0] < root.Interval[0] {
```

```
//    root.Left = delete(root.Left, []int{interval[0], min(interval[1],
root.Interval[0]-1)})
//  }
//  if root.Interval[1] < interval[1] {
//    root.Right = delete(root.Right, []int{max(interval[0], root.Interval[1]+1),
interval[1]})
//  }
//  if interval[1] < root.Interval[0] || root.Interval[1] < interval[0] {
//    return root
//  }
//  if interval[0] <= root.Interval[0] && root.Interval[1] <= interval[1] {
//    if root.Left == nil {
//      return root.Right
//    } else if root.Right == nil {
//      return root.Left
//    } else {
//      pred := root.Left
//      for pred.Right != nil {
//        pred = pred.Right
//      }
//      root.Interval = pred.Interval
//      root.Left = delete(root.Left, pred.Interval)
//      return root
//    }
//  }
//  if root.Interval[0] < interval[0] && interval[1] < root.Interval[1] {
//    left := &BSTNode{[]int{root.Interval[0], interval[0] - 1}, root.Left, nil}
//    right := &BSTNode{[]int{interval[1] + 1, root.Interval[1]}, nil, root.Right}
//    left.Right = right
//    return left
//  }
//  if interval[0] <= root.Interval[0] {
//    root.Interval[0] = interval[1] + 1
//  }
//  if root.Interval[1] <= interval[1] {
//    root.Interval[1] = interval[0] - 1
//  }
//  return root
// }

// func (this *RangeModule) query(root *BSTNode, interval []int) bool {
//  if root == nil {
//    return false
//  }
//  if interval[1] < root.Interval[0] {
//    return query(root.Left, interval)
//  }
//  if root.Interval[1] < interval[0] {
//    return query(root.Right, interval)
```

```
//  }
//  left := true
//  if interval[0] < root.Interval[0] {
//    left = query(root.Left, []int{interval[0], root.Interval[0] - 1})
//  }
//  right := true
//  if root.Interval[1] < interval[1] {
//    right = query(root.Right, []int{root.Interval[1] + 1, interval[1]})
//  }
//  return left && right
// }

/**
 * Your RangeModule object will be instantiated and called as such:
 * obj := Constructor();
 * obj.AddRange(left,right);
 * param_2 := obj.QueryRange(left,right);
 * obj.RemoveRange(left,right);
 */
```

717. 1-bit and 2-bit Characters

题目:

We have two special characters. The first character can be represented by one bit `0`. The second character can be represented by two bits (`10` or `11`).

Now given a string represented by several bits. Return whether the last character must be a one-bit character or not. The given string will always end with a zero.

Example 1:

```
Input:
bits = [1, 0, 0]
Output: True
Explanation:
The only way to decode it is two-bit character and one-bit character. So the last
character is one-bit character.
```

Example 2:

```
Input:
bits = [1, 1, 1, 0]
Output: False
Explanation:
The only way to decode it is two-bit character and two-bit character. So the last
character is NOT one-bit character.
```

Note:

- `1 <= len(bits) <= 1000`.
- `bits[i]` is always `0` or `1`.

题目大意

有两种特殊字符。第一种字符可以用一比特0来表示。第二种字符可以用两比特(10 或 11)来表示。

现给一个由若干比特组成的字符串。问最后一个字符是否必定为一个一比特字符。给定的字符串总是由0结束。

注意:

- 1 <= len(bits) <= 1000.
- bits[i] 总是0 或 1.

解题思路

- 给出一个数组，数组里面的元素只有 0 和 1，并且数组的最后一个元素一定是 0。有 2 种特殊的字符，第一类字符是 "0"，第二类字符是 "11" 和 "10"，请判断这个数组最后一个元素是否一定是属于第一类字符?
- 依题意， 0 的来源有 2 处，可以是第一类字符，也可以是第二类字符，1 的来源只有 1 处，一定出自第二类字符。最后一个 0 当前仅当为第一类字符的情况有 2 种，第一种情况，前面出现有 0，但是 0 和 1 配对形成了第二类字符。第二种情况，前面没有出现 0 。这两种情况的共同点是除去最后一个元素，数组中前面所有的1 都“结对子”。所以利用第二类字符的特征，"1X"，遍历整个数组，如果遇到 "1"，就跳 2 步，因为 1 后面出现什么数字(0 或者 1)并不需要关心。如果 `i` 能在 `len(bits) - 1` 的地方`(数组最后一个元素)`停下，那么对应的是情况一或者情况二，前面的 0 都和 1 匹配上了，最后一个 0 一定是第一类字符。如果 `i` 在 `len(bit)` 的位置`(超出数组下标)`停下，说明 `bits[len(bits) - 1] == 1`，这个时候最后一个 0 一定属于第二类字符。

代码

```go
package leetcode

func isOneBitCharacter(bits []int) bool {
  var i int
  for i = 0; i < len(bits)-1; i++ {
    if bits[i] == 1 {
      i++
    }
  }
  return i == len(bits)-1
}
```

718. Maximum Length of Repeated Subarray

题目

Given two integer arrays `A` and `B`, return the maximum length of an subarray that appears in both arrays.

Example 1:

```
Input:
A: [1,2,3,2,1]
B: [3,2,1,4,7]
Output: 3
Explanation:
The repeated subarray with maximum length is [3, 2, 1].
```

Note:

1. 1 <= len(A), len(B) <= 1000
2. 0 <= A[i], B[i] < 100

题目大意

给两个整数数组 A 和 B ，返回两个数组中公共的、长度最长的子数组的长度。

解题思路

- 给出两个数组，求这两个数组中最长相同子串的长度。
- 这一题最容易想到的是 DP 动态规划的解法。`dp[i][j]` 代表在 A 数组中以 `i` 下标开始的子串与 B 数组中以 `j` 下标开始的子串最长相同子串的长度，状态转移方程为 `dp[i][j] = dp[i+1][j+1] + 1` (当 `A[i] == B[j]`)。这种解法的时间复杂度是 O(n^2)，空间复杂度 O(n^2)。
- 这一题最佳解法是二分搜索 + `Rabin-Karp`。比较相同子串耗时的地方在于，需要一层循环，遍历子串所有字符。但是如果比较两个数字就很快，`O(1)` 的时间复杂度。所以有人就想到了，能不能把字符串也映射成数字呢？这样比较起来就非常快。这个算法就是 `Rabin-Karp` 算法。字符串映射成一个数字不能随意映射，还要求能根据字符串前缀动态增加，比较下一个字符串的时候，可以利用已比较过的前缀，加速之后的字符串比较。在 Rabin-Karp 算法中有一个“码点”的概念。类似于10进制中的进制。具体的算法讲解可以见这篇：

  [基础知识 - Rabin-Karp 算法]

  “码点”一般取值为一个素数。在 go 的 `strings` 包里面取值是 16777619。所以这一题也可以直接取这个值。由于这一次要求我们找最长长度，所以把最长长度作为二分搜索的目标。先将数组 A 和数组 B 中的数字都按照二分出来的长度，进行 `Rabin-Karp` hash。对 A 中的 hash 与下标做映射关系，存到 map 中，方便后面快速查找。然后遍历 B 中的 hash，当 hash 一致的时候，再匹配下标。如果下标存在，且拥有相同的前缀，那么就算找到了相同的子串了。最后就是不断的二分，找到最长的结果即可。这个解法的时间复杂度 O(n * log n)，空间复杂度 O(n)。

代码

```go
package leetcode
```

```go
const primeRK = 16777619

// 解法一 二分搜索 + Rabin-Karp
func findLength(A []int, B []int) int {
  low, high := 0, min(len(A), len(B))
  for low < high {
    mid := (low + high + 1) >> 1
    if hasRepeated(A, B, mid) {
      low = mid
    } else {
      high = mid - 1
    }
  }
  return low
}

func hashSlice(arr []int, length int) []int {
  // hash 数组里面记录 arr 比 length 长出去部分的 hash 值
  hash, pl, h := make([]int, len(arr)-length+1), 1, 0
  for i := 0; i < length-1; i++ {
    pl *= primeRK
  }
  for i, v := range arr {
    h = h*primeRK + v
    if i >= length-1 {
      hash[i-length+1] = h
      h -= pl * arr[i-length+1]
    }
  }
  return hash
}

func hasSamePrefix(A, B []int, length int) bool {
  for i := 0; i < length; i++ {
    if A[i] != B[i] {
      return false
    }
  }
  return true
}

func hasRepeated(A, B []int, length int) bool {
  hs := hashSlice(A, length)
  hashToOffset := make(map[int][]int, len(hs))
  for i, h := range hs {
    hashToOffset[h] = append(hashToOffset[h], i)
  }
  for i, h := range hashSlice(B, length) {
    if offsets, ok := hashToOffset[h]; ok {
```

```go
        for _, offset := range offsets {
          if hasSamePrefix(A[offset:], B[i:], length) {
            return true
          }
        }
      }
    }
    return false
}

// 解法二 DP 动态规划
func findLength1(A []int, B []int) int {
    res, dp := 0, make([][]int, len(A)+1)
    for i := range dp {
      dp[i] = make([]int, len(B)+1)
    }
    for i := len(A) - 1; i >= 0; i-- {
      for j := len(B) - 1; j >= 0; j-- {
        if A[i] == B[j] {
          dp[i][j] = dp[i+1][j+1] + 1
          if dp[i][j] > res {
            res = dp[i][j]
          }
        }
      }
    }
    return res
}
```

719. Find K-th Smallest Pair Distance

题目

Given an integer array, return the k-th smallest **distance** among all the pairs. The distance of a pair (A, B) is defined as the absolute difference between A and B.

Example 1:

```
Input:
nums = [1,3,1]
k = 1
Output: 0
Explanation:
Here are all the pairs:
(1,3) -> 2
(1,1) -> 0
(3,1) -> 2
Then the 1st smallest distance pair is (1,1), and its distance is 0.
```

Note:

1. `2 <= len(nums) <= 10000`.
2. `0 <= nums[i] < 1000000`.
3. `1 <= k <= len(nums) * (len(nums) - 1) / 2`.

题目大意

给定一个整数数组，返回所有数对之间的第 k 个最小距离。一对 (A, B) 的距离被定义为 A 和 B 之间的绝对差值。

提示:

1. 2 <= len(nums) <= 10000.
2. 0 <= nums[i] < 1000000.
3. 1 <= k <= len(nums) * (len(nums) - 1) / 2.

解题思路

- 给出一个数组，要求找出第 k 小两两元素之差的值。两两元素之差可能重复，重复的元素之差算多个，不去重。
- 这一题可以用二分搜索来解答。先把原数组排序，那么最大的差值就是 `nums[len(nums)-1] - nums[0]`，最小的差值是 0，即在 `[0, nums[len(nums)-1] - nums[0]]` 区间内搜索最终答案。针对每个 `mid`，判断小于等于 `mid` 的差值有多少个。题意就转化为，在数组中找到这样一个数，使得满足 `nums[i] - nums[j] ≤ mid` 条件的组合数等于 `k`。那么如何计算满足两两数的差值小于 mid 的组合总数是本题的关键。
- 最暴力的方法就是 2 重循环，暴力计数。这个方法效率不高，耗时很长。原因是没有利用数组有序这一条件。实际上数组有序对计算满足条件的组合数有帮助。利用双指针滑动即可计算出组合总数。见解法一。

代码

```go
package leetcode

import (
  "sort"
```

```go
)

func smallestDistancePair(nums []int, k int) int {
  sort.Ints(nums)
  low, high := 0, nums[len(nums)-1]-nums[0]
  for low < high {
    mid := low + (high-low)>>1
    tmp := findDistanceCount(nums, mid)
    if tmp >= k {
      high = mid
    } else {
      low = mid + 1
    }
  }
  return low
}

// 解法一 双指针
func findDistanceCount(nums []int, num int) int {
  count, i := 0, 0
  for j := 1; j < len(nums); j++ {
    for nums[j]-nums[i] > num && i < j {
      i++
    }
    count += (j - i)
  }
  return count
}

// 解法二 暴力查找
func findDistanceCount1(nums []int, num int) int {
  count := 0
  for i := 0; i < len(nums); i++ {
    for j := i + 1; j < len(nums); j++ {
      if nums[j]-nums[i] <= num {
        count++
      }
    }
  }
  return count
}
```

720. Longest Word in Dictionary

题目

Given a list of strings `words` representing an English Dictionary, find the longest word in `words` that can be built one character at a time by other words in `words`. If there is more than one possible answer, return the longest word with the smallest lexicographical order.

If there is no answer, return the empty string.

Example 1:

```
Input:
words = ["w","wo","wor","worl", "world"]
Output: "world"
Explanation:
The word "world" can be built one character at a time by "w", "wo", "wor", and "worl".
```

Example 2:

```
Input:
words = ["a", "banana", "app", "appl", "ap", "apply", "apple"]
Output: "apple"
Explanation:
Both "apply" and "apple" can be built from other words in the dictionary. However,
"apple" is lexicographically smaller than "apply".
```

Note:

- All the strings in the input will only contain lowercase letters.
- The length of `words` will be in the range `[1, 1000]`.
- The length of `words[i]` will be in the range `[1, 30]`.

题目大意

给出一个字符串数组 words 组成的一本英语词典。从中找出最长的一个单词，该单词是由 words 词典中其他单词逐步添加一个字母组成。若其中有多个可行的答案，则返回答案中字典序最小的单词。若无答案，则返回空字符串。

解题思路

- 给出一个字符串数组，要求找到长度最长的，并且可以由字符串数组里面其他字符串拼接一个字符组成的字符串。如果存在多个这样的最长的字符串，则输出字典序较小的那个字符串，如果找不到这样的字符串，输出空字符串。
- 这道题解题思路是先排序，排序完成以后就是字典序从小到大了。之后再用 map 辅助记录即可。

代码

```go
package leetcode
```

```go
import (
  "sort"
)

func longestWord(words []string) string {
  sort.Strings(words)
  mp := make(map[string]bool)
  var res string
  for _, word := range words {
    size := len(word)
    if size == 1 || mp[word[:size-1]] {
      if size > len(res) {
        res = word
      }
      mp[word] = true
    }
  }
  return res
}
```

721. Accounts Merge

题目

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]`.

题目大意

给定一个列表 accounts，每个元素 accounts[i] 是一个字符串列表，其中第一个元素 accounts[i][0] 是 名称 (name)，其余元素是 emails 表示该帐户的邮箱地址。现在，我们想合并这些帐户。如果两个帐户都有一些共同的邮件地址，则两个帐户必定属于同一个人。请注意，即使两个帐户具有相同的名称，它们也可能属于不同的人，因为人们可能具有相同的名称。一个人最初可以拥有任意数量的帐户，但其所有帐户都具有相同的名称。合并帐户后，按以下格式返回帐户：每个帐户的第一个元素是名称，其余元素是按顺序排列的邮箱地址。accounts 本身可以以任意顺序返回。

注意：

- accounts 的长度将在 [1，1000] 的范围内。
- accounts[i] 的长度将在 [1，10] 的范围内。
- accounts[i][j] 的长度将在 [1，30] 的范围内。

解题思路

- 给出一堆账户和对应的邮箱。要求合并同一个人的多个邮箱账户。如果判断是同一个人呢？如果这个人名和所属的其中之一的邮箱是相同的，就判定这是同一个人的邮箱，那么就合并这些邮箱。
- 这题的解题思路是并查集。不过如果用暴力合并的方法，时间复杂度非常差。优化方法是先把每组数据都进行编号，人编号，每个邮箱都进行编号。这个映射关系用 `map` 记录起来。如果利用并查集的 `union()` 操作，把这些编号都进行合并。最后把人的编号和对应邮箱的编号拼接起来。
- 这一题有 2 处比较“坑”的是，不需要合并的用户的邮箱列表也是需要排序和去重的，同一个人的所有邮箱集合都要合并到一起。具体见测试用例。不过题目中也提到了这些点，也不能算题目坑，只能归自己没注意这些边界情况。

代码

```go
package leetcode

import (
  "sort"

  "github.com/halfrost/LeetCode-Go/template"
)

// 解法一 并查集优化搜索解法
func accountsMerge(accounts [][]string) (r [][]string) {
  uf := template.UnionFind{}
  uf.Init(len(accounts))
  // emailToID 将所有的 email 邮箱都拆开，拆开与 id(数组下标) 对应
  // idToName 将 id(数组下标) 与 name 对应
  // idToEmails 将 id(数组下标) 与整理好去重以后的 email 组对应
  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
}

// 解法二 并查集暴力解法
func accountsMerge1(accounts [][]string) [][]string {
  if len(accounts) == 0 {
    return [][]string{}
  }
  uf, res, visited := template.UnionFind{}, [][]string{}, map[int]bool{}
```

```go
  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] {
        tmpA, tmpB, flag := accounts[i][1:], accounts[j][1:], false
        for j := 0; j < len(tmpA); j++ {
          for k := 0; k < len(tmpB); k++ {
            if tmpA[j] == tmpB[k] {
              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
        }
      }
    }
    for _, v := range emails {
      tmpMap[v] = v
    }
    emails = []string{}
    for key := range tmpMap {
      emails = append(emails, key)
    }
    sort.Strings(emails)
    account = append(account, emails...)
    res = append(res, account)
  }
  return res
}
```

724. Find Pivot Index

题目

Given an array of integers nums, write a method that returns the "pivot" index of this array.

We define the pivot index as the index where the sum of all the numbers to the left of the index is equal to the sum of all the numbers to the right of the index.

If no such index exists, we should return -1. If there are multiple pivot indexes, you should return the left-most pivot index.

Example 1:

```
Input: nums = [1,7,3,6,5,6]
Output: 3
Explanation:
The sum of the numbers to the left of index 3 (nums[3] = 6) is equal to the sum of
numbers to the right of index 3.
Also, 3 is the first index where this occurs.
```

Example 2:

```
Input: nums = [1,2,3]
Output: -1
Explanation:
There is no index that satisfies the conditions in the problem statement.
```

Constraints:

- The length of nums will be in the range [0, 10000].
- Each element nums[i] will be an integer in the range [-1000, 1000].

题目大意

给定一个整数类型的数组 nums，请编写一个能够返回数组 “中心索引” 的方法。我们是这样定义数组 中心索引 的：数组中心索引的左侧所有元素相加的和等于右侧所有元素相加的和。如果数组不存在中心索引，那么我们应该返回 -1。如果数组有多个中心索引，那么我们应该返回最靠近左边的那一个。

解题思路

- 在数组中，找到一个数，使得它左边的数之和等于它的右边的数之和，如果存在，则返回这个数的下标索引，

否作返回 -1。
- 这里面存在一个等式，只需要满足这个等式即可满足条件：leftSum + num[i] = sum - leftSum => 2 * leftSum + num[i] = sum。
- 题目提到如果存在多个索引，则返回最左边那个，因此从左开始求和，而不是从右边。

代码

```
package leetcode

// 2 * leftSum + num[i] = sum
// 时间: O(n)
// 空间: O(1)
func pivotIndex(nums []int) int {
  if len(nums) <= 0 {
    return -1
  }
  var sum, leftSum int
  for _, num := range nums {
    sum += num
  }
  for index, num := range nums {
    if leftSum*2+num == sum {
      return index
    }
    leftSum += num
  }
  return -1
}
```

725. Split Linked List in Parts

题目

Given a (singly) linked list with head node root, write a function to split the linked list into k consecutive linked list "parts".

The length of each part should be as equal as possible: no two parts should have a size differing by more than 1. This may lead to some parts being null.

The parts should be in order of occurrence in the input list, and parts occurring earlier should always have a size greater than or equal parts occurring later.

Return a List of ListNode's representing the linked list parts that are formed.

Examples 1->2->3->4, k = 5 // 5 equal parts [[1], [2], [3], [4], null]

Example 1:

```
Input:
root = [1, 2, 3], k = 5
Output: [[1],[2],[3],[],[]]
Explanation:
The input and each element of the output are ListNodes, not arrays.
For example, the input root has root.val = 1, root.next.val = 2, \root.next.next.val =
3, and root.next.next.next = null.
The first element output[0] has output[0].val = 1, output[0].next = null.
The last element output[4] is null, but it's string representation as a ListNode is [].
```

Example 2:

```
Input:
root = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], k = 3
Output: [[1, 2, 3, 4], [5, 6, 7], [8, 9, 10]]
Explanation:
The input has been split into consecutive parts with size difference at most 1, and
earlier parts are a larger size than the later parts.
```

Note:

- The length of root will be in the range [0, 1000].
- Each value of a node in the input will be an integer in the range [0, 999].
- k will be an integer in the range [1, 50].

题目大意

把链表分成 K 个部分，要求这 K 个部分尽量两两长度相差不超过 1，并且长度尽量相同。

解题思路

把链表长度对 K 进行除法，结果就是最终每组的长度 n。把链表长度对 K 进行取余操作，得到的结果 m，代表前 m 组链表长度为 n + 1 。相当于把多出来的部分都分摊到前面 m 组链表中了。最终链表是前 m 组长度为 n + 1，后 K - m 组链表长度是 n。

注意长度不足 K 的时候要用 nil 进行填充。

代码

```go
package leetcode

import "fmt"

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */
func splitListToParts(root *ListNode, k int) []*ListNode {
  res := make([]*ListNode, 0)
  if root == nil {
    for i := 0; i < k; i++ {
      res = append(res, nil)
    }
    return res
  }
  length := getLength(root)
  splitNum := length / k
  lengNum := length % k
  cur, head := root, root
  var pre *ListNode
  fmt.Printf("总长度 %v, 分 %v 组, 前面 %v 组长度为 %v, 剩余 %v 组, 每组 %v\n", length, k,
lengNum, splitNum+1, k-lengNum, splitNum)
  if splitNum == 0 {
    for i := 0; i < k; i++ {
      if cur != nil {
        pre = cur.Next
        cur.Next = nil
        res = append(res, cur)
        cur = pre
      } else {
        res = append(res, nil)
      }
    }
    return res
  }
  for i := 0; i < lengNum; i++ {
    for j := 0; j < splitNum; j++ {
      cur = cur.Next
    }
    fmt.Printf("0 刚刚出来 head = %v cur = %v pre = %v\n", head, cur, head)
    pre = cur.Next
    cur.Next = nil
    res = append(res, head)
    head = pre
    cur = pre
```

```
    fmt.Printf("0 head = %v cur = %v pre = %v\n", head, cur, head)
  }
  for i := 0; i < k-lengNum; i++ {
    for j := 0; j < splitNum-1; j++ {
      cur = cur.Next
    }
    fmt.Printf("1 刚刚出来 head = %v cur = %v pre = %v\n", head, cur, head)
    pre = cur.Next
    cur.Next = nil
    res = append(res, head)
    head = pre
    cur = pre
  }
  return res
}

func getLength(l *ListNode) int {
  count := 0
  cur := l
  for cur != nil {
    count++
    cur = cur.Next
  }
  return count
}
```

726. Number of Atoms

题目

Given a chemical `formula` (given as a string), return the count of each atom.

An atomic element always starts with an uppercase character, then zero or more lowercase letters, representing the name.

1 or more digits representing the count of that element may follow if the count is greater than 1. If the count is 1, no digits will follow. For example, H2O and H2O2 are possible, but H1O2 is impossible.

Two formulas concatenated together produce another formula. For example, H2O2He3Mg4 is also a formula.

A formula placed in parentheses, and a count (optionally added) is also a formula. For example, (H2O2) and (H2O2)3 are formulas.

Given a formula, output the count of all elements as a string in the following form: the first name (in sorted order), followed by its count (if that count is more than 1), followed by the second name (in sorted order), followed by its count (if that count is more than 1), and so on.

Example 1:

```
Input:
formula = "H2O"
Output: "H2O"
Explanation:
The count of elements are {'H': 2, 'O': 1}.
```

Example 2:

```
Input:
formula = "Mg(OH)2"
Output: "H2MgO2"
Explanation:
The count of elements are {'H': 2, 'Mg': 1, 'O': 2}.
```

Example 3:

```
Input:
formula = "K4(ON(SO3)2)2"
Output: "K4N2O14S4"
Explanation:
The count of elements are {'K': 4, 'N': 2, 'O': 14, 'S': 4}.
```

Note:

- All atom names consist of lowercase letters, except for the first character which is uppercase.
- The length of `formula` will be in the range `[1, 1000]`.
- `formula` will only consist of letters, digits, and round parentheses, and is a valid formula as defined in the problem.

题目大意

给定一个化学式，输出所有原子的数量。格式为：第一个（按字典序）原子的名子，跟着它的数量（如果数量大于 1），然后是第二个原子的名字（按字典序），跟着它的数量（如果数量大于 1），以此类推。

原子总是以一个大写字母开始，接着跟随0个或任意个小写字母，表示原子的名字。如果数量大于 1，原子后会跟着数字表示原子的数量。如果数量等于 1 则不会跟数字。例如，H2O 和 H2O2 是可行的，但 H1O2 这个表达是不可行的。两个化学式连在一起是新的化学式。例如 H2O2He3Mg4 也是化学式。一个括号中的化学式和数字（可选择性添加）也是化学式。例如 (H2O2) 和 (H2O2)3 是化学式。

解题思路

- 利用栈处理每个化学元素，用 map 记录每个化学元素的个数，最终排序以后输出即可
- 注意化学元素有些并不是单一字母，比如镁元素是 Mg，所以需要考虑字母的大小写问题。

代码

```go
package leetcode

import (
  "sort"
  "strconv"
  "strings"
)

type atom struct {
  name string
  cnt  int
}

type atoms []atom

func (this atoms) Len() int           { return len(this) }
func (this atoms) Less(i, j int) bool { return strings.Compare(this[i].name,
this[j].name) < 0 }
func (this atoms) Swap(i, j int)      { this[i], this[j] = this[j], this[i] }
func (this atoms) String() string {
  s := ""
  for _, a := range this {
    s += a.name
    if a.cnt > 1 {
      s += strconv.Itoa(a.cnt)
    }
  }
  return s
}

func countOfAtoms(s string) string {
  n := len(s)
  if n == 0 {
    return ""
  }

  stack := make([]string, 0)
  for i := 0; i < n; i++ {
    c := s[i]
    if c == '(' || c == ')' {
      stack = append(stack, string(c))
    } else if isUpperLetter(c) {
      j := i + 1
      for ; j < n; j++ {
        if !isLowerLetter(s[j]) {
          break
        }
      }
```

```
      stack = append(stack, s[i:j])
      i = j - 1
    } else if isDigital(c) {
      j := i + 1
      for ; j < n; j++ {
        if !isDigital(s[j]) {
          break
        }
      }
      stack = append(stack, s[i:j])
      i = j - 1
    }
  }

  cnt, deep := make([]map[string]int, 100), 0
  for i := 0; i < 100; i++ {
    cnt[i] = make(map[string]int)
  }
  for i := 0; i < len(stack); i++ {
    t := stack[i]
    if isUpperLetter(t[0]) {
      num := 1
      if i+1 < len(stack) && isDigital(stack[i+1][0]) {
        num, _ = strconv.Atoi(stack[i+1])
        i++
      }
      cnt[deep][t] += num
    } else if t == "(" {
      deep++
    } else if t == ")" {
      num := 1
      if i+1 < len(stack) && isDigital(stack[i+1][0]) {
        num, _ = strconv.Atoi(stack[i+1])
        i++
      }
      for k, v := range cnt[deep] {
        cnt[deep-1][k] += v * num
      }
      cnt[deep] = make(map[string]int)
      deep--
    }
  }
  as := atoms{}
  for k, v := range cnt[0] {
    as = append(as, atom{name: k, cnt: v})
  }
  sort.Sort(as)
  return as.String()
}
```

```go
func isDigital(v byte) bool {
  if v >= '0' && v <= '9' {
    return true
  }
  return false
}

func isUpperLetter(v byte) bool {
  if v >= 'A' && v <= 'Z' {
    return true
  }
  return false
}

func isLowerLetter(v byte) bool {
  if v >= 'a' && v <= 'z' {
    return true
  }
  return false
}
```

729. My Calendar I

题目

Implement a `MyCalendar` class to store your events. A new event can be added if adding the event will not cause a double booking.

Your class will have the method, `book(int start, int end)`. Formally, this represents a booking on the half open interval `[start, end)`, the range of real numbers `x` such that `start <= x < end`.

A double booking happens when two events have some non-empty intersection (ie., there is some time that is common to both events.)

For each call to the method `MyCalendar.book`, return `true` if the event can be added to the calendar successfully without causing a double booking. Otherwise, return `false` and do not add the event to the calendar.

Your class will be called like this:

`MyCalendar cal = new MyCalendar();`

`MyCalendar.book(start, end)`

Example 1:

```
MyCalendar();
MyCalendar.book(10, 20); // returns true
MyCalendar.book(15, 25); // returns false
MyCalendar.book(20, 30); // returns true
Explanation:
The first event can be booked.  The second can't because time 15 is already booked by another event.
The third event can be booked, as the first event takes every time less than 20, but not including 20.
```

Note:

- The number of calls to `MyCalendar.book` per test case will be at most `1000`.
- In calls to `MyCalendar.book(start, end)`, `start` and `end` are integers in the range `[0, 10^9]`.

题目大意

实现一个 MyCalendar 类来存放你的日程安排。如果要添加的时间内没有其他安排，则可以存储这个新的日程安排。

MyCalendar 有一个 book(int start, int end) 方法。它意味着在 start 到 end 时间内增加一个日程安排，注意，这里的时间是半开区间，即 [start, end), 实数 x 的范围为， start <= x < end。

当两个日程安排有一些时间上的交叉时（例如两个日程安排都在同一时间内），就会产生重复预订。

每次调用 MyCalendar.book 方法时，如果可以将日程安排成功添加到日历中而不会导致重复预订，返回 true。否则，返回 false 并且不要将该日程安排添加到日历中。

请按照以下步骤调用 MyCalendar 类: MyCalendar cal = new MyCalendar(); MyCalendar.book(start, end)

说明:

- 每个测试用例，调用 MyCalendar.book 函数最多不超过 100次。
- 调用函数 MyCalendar.book(start, end) 时， start 和 end 的取值范围为 [0, 10^9]。

解题思路

- 要求实现一个日程安排的功能，如果有日程安排冲突了，就返回 false，如果不冲突则返回 ture
- 这一题有多种解法，第一种解法可以用类似第 34 题的解法。先排序每个区间，然后再这个集合中用二分搜索找到最后一个区间的左值比当前要比较的区间左值小的，如果找到，再判断能否插入进去(判断右区间是否比下一个区间的左区间小)，此方法时间复杂度 O(n log n)
- 第二种解法是用生成一个 BST 树。在插入树中先排除不能插入的情况，例如区间有重合。然后以区间左值为依据，递归插入，每次插入依次会继续判断区间是否重合。直到不能插入，则返回 fasle。整个查找的时间复杂度是 O(log n)。

代码

```
package leetcode
```

```go
// 解法一 二叉排序树
// Event define
type Event struct {
  start, end  int
  left, right *Event
}

// Insert define
func (e *Event) Insert(curr *Event) bool {
  if e.end > curr.start && curr.end > e.start {
    return false
  }
  if curr.start < e.start {
    if e.left == nil {
      e.left = curr
    } else {
      return e.left.Insert(curr)
    }
  } else {
    if e.right == nil {
      e.right = curr
    } else {
      return e.right.Insert(curr)
    }
  }
  return true
}

// MyCalendar define
type MyCalendar struct {
  root *Event
}

// Constructor729 define
func Constructor729() MyCalendar {
  return MyCalendar{
    root: nil,
  }
}

// Book define
func (this *MyCalendar) Book(start int, end int) bool {
  curr := &Event{start: start, end: end, left: nil, right: nil}
  if this.root == nil {
    this.root = curr
    return true
  }
  return this.root.Insert(curr)
```

```
}

// 解法二 快排 + 二分
// MyCalendar define
// type MyCalendar struct {
//  calendar []Interval
// }

// // Constructor729 define
// func Constructor729() MyCalendar {
//  calendar := []Interval{}
//  return MyCalendar{calendar: calendar}
// }

// // Book define
// func (this *MyCalendar) Book(start int, end int) bool {
//  if len(this.calendar) == 0 {
//    this.calendar = append(this.calendar, Interval{Start: start, End: end})
//    return true
//  }
//  // 快排
//  quickSort(this.calendar, 0, len(this.calendar)-1)
//  // 二分
//  pos := searchLastLessInterval(this.calendar, start, end)
//  // 如果找到最后一个元素，需要判断 end
//  if pos == len(this.calendar)-1 && this.calendar[pos].End <= start {
//    this.calendar = append(this.calendar, Interval{Start: start, End: end})
//    return true
//  }
//  // 如果不是开头和结尾的元素，还需要判断这个区间是否能插入到原数组中(要看起点和终点是否都能插入)
//  if pos != len(this.calendar)-1 && pos != -1 && this.calendar[pos].End <= start &&
this.calendar[pos+1].Start >= end {
//    this.calendar = append(this.calendar, Interval{Start: start, End: end})
//    return true
//  }
//  // 如果元素比开头的元素还要小，要插入到开头
//  if this.calendar[0].Start >= end {
//    this.calendar = append(this.calendar, Interval{Start: start, End: end})
//    return true
//  }
//  return false
// }

// func searchLastLessInterval(intervals []Interval, start, end int) int {
//  low, high := 0, len(intervals)-1
//  for low <= high {
//    mid := low + ((high - low) >> 1)
//    if intervals[mid].Start <= start {
```

```
//      if (mid == len(intervals)-1) || (intervals[mid+1].Start > start) { // 找到最后一个小于等于 target 的元素
//        return mid
//      }
//      low = mid + 1
//    } else {
//      high = mid - 1
//    }
//  }
//  return -1
// }

/**
 * Your MyCalendar object will be instantiated and called as such:
 * obj := Constructor();
 * param_1 := obj.Book(start,end);
 */
```

732. My Calendar III

题目

Implement a `MyCalendarThree` class to store your events. A new event can **always** be added.

Your class will have one method, `book(int start, int end)`. Formally, this represents a booking on the half open interval `[start, end)`, the range of real numbers `x` such that `start <= x < end`.

A K-booking happens when **K** events have some non-empty intersection (ie., there is some time that is common to all K events.)

For each call to the method `MyCalendar.book`, return an integer `K` representing the largest integer such that there exists a `K`-booking in the calendar.

Your class will be called like this:

`MyCalendarThree cal = new MyCalendarThree();`

`MyCalendarThree.book(start, end)`

Example 1:

```
MyCalendarThree();
MyCalendarThree.book(10, 20); // returns 1
MyCalendarThree.book(50, 60); // returns 1
MyCalendarThree.book(10, 40); // returns 2
MyCalendarThree.book(5, 15); // returns 3
MyCalendarThree.book(5, 10); // returns 3
MyCalendarThree.book(25, 55); // returns 3
Explanation:
The first two events can be booked and are disjoint, so the maximum K-booking is a 1-booking.
The third event [10, 40) intersects the first event, and the maximum K-booking is a 2-booking.
The remaining events cause the maximum K-booking to be only a 3-booking.
Note that the last event locally causes a 2-booking, but the answer is still 3 because eg. [10, 20), [10, 40), and [5, 15) are still triple booked.
```

Note:

- The number of calls to `MyCalendarThree.book` per test case will be at most `400`.
- In calls to `MyCalendarThree.book(start, end)`, `start` and `end` are integers in the range `[0, 10^9]`.

题目大意

实现一个 MyCalendar 类来存放你的日程安排，你可以一直添加新的日程安排。

MyCalendar 有一个 book(int start, int end)方法。它意味着在start到end时间内增加一个日程安排，注意，这里的时间是半开区间，即 [start, end), 实数 x 的范围为， start <= x < end。当 K 个日程安排有一些时间上的交叉时（例如K个日程安排都在同一时间内），就会产生 K 次预订。每次调用 MyCalendar.book方法时，返回一个整数 K ，表示最大的 K 次预订。

请按照以下步骤调用MyCalendar 类: MyCalendar cal = new MyCalendar(); MyCalendar.book(start, end)

说明:

- 每个测试用例，调用 MyCalendar.book 函数最多不超过 400 次。
- 调用函数 MyCalendar.book(start, end)时， start 和 end 的取值范围为 [0, 10^9]。

解题思路

- 设计一个日程类，每添加一个日程，实时显示出当前排期中累计日程最多的个数，例如在一段时间内，排了 3 个日程，其他时间内都只有 0，1，2 个日程，则输出 3 。
- 拿到这个题目以后会立即想到线段树。由于题目中只有增加日程，所以这一题难度不大。这一题和第 699 题也类似，但是有区别，第 699 题中，俄罗斯方块会依次摞起来，而这一题中，俄罗斯方块也就摞起来，但是方块下面如果是空挡，方块会断掉。举个例子：依次增加区间 [10,20]，[10,40]，[5,15]，[5,10]，如果是第 699 题的规则，这 [5,10] 的这块砖块会落在 [5,15] 上，从而使得高度为 4，但是这一题是日程，日程不一样，[5,15] 这个区间内有 3 个日程，但是其他部分都没有 3 个日程，所以第三块砖块 [5,15] 中的 [5,10] 会“断裂”，掉下去，第四块砖块还是 [5,10]，落在第三块砖块断落下去的位置，它们俩落在一起的高度是 2 。

- 构造一颗线段树，这里用树来构造，如果用数组需要开辟很大的空间。当区间左右边界和查询边界完全相同的时候再累加技术，否则不加，继续划分区间。以区间的左边界作为划分区间的标准，因为区间左边界是开区间，右边是闭区间。一个区间的计数值以区间左边界的计数为准。还是上面的例子，[5,10) 计数以 5 为标准，count = 2，[10,15) 计数以 10 为标准，count = 3 。还需要再动态维护一个最大值。这个线段树的实现比较简单。
- 类似的题目有：第 715 题，第 218 题，第 699 题。第 715 题是区间更新定值(**不是增减**)，第 218 题可以用扫描线，第 732 题和第 699 题类似，也是俄罗斯方块的题目，但是第 732 题的俄罗斯方块的方块会“断裂”。

代码

```go
package leetcode

// SegmentTree732 define
type SegmentTree732 struct {
  start, end, count int
  left, right       *SegmentTree732
}

// MyCalendarThree define
type MyCalendarThree struct {
  st        *SegmentTree732
  maxHeight int
}

// Constructor732 define
func Constructor732() MyCalendarThree {
  st := &SegmentTree732{
    start: 0,
    end:   1e9,
  }
  return MyCalendarThree{
    st: st,
  }
}

// Book define
func (mct *MyCalendarThree) Book(start int, end int) int {
  mct.st.book(start, end, &mct.maxHeight)
  return mct.maxHeight
}

func (st *SegmentTree732) book(start, end int, maxHeight *int) {
  if start == end {
    return
  }
  if start == st.start && st.end == end {
    st.count++
```

```go
    if st.count > *maxHeight {
      *maxHeight = st.count
    }
    if st.left == nil {
      return
    }
  }
  if st.left == nil {
    if start == st.start {
      st.left = &SegmentTree732{start: start, end: end, count: st.count}
      st.right = &SegmentTree732{start: end, end: st.end, count: st.count}
      st.left.book(start, end, maxHeight)
      return
    }
    st.left = &SegmentTree732{start: st.start, end: start, count: st.count}
    st.right = &SegmentTree732{start: start, end: st.end, count: st.count}
    st.right.book(start, end, maxHeight)
    return
  }
  if start >= st.right.start {
    st.right.book(start, end, maxHeight)
  } else if end <= st.left.end {
    st.left.book(start, end, maxHeight)
  } else {
    st.left.book(start, st.left.end, maxHeight)
    st.right.book(st.right.start, end, maxHeight)
  }
}

/**
 * Your MyCalendarThree object will be instantiated and called as such:
 * obj := Constructor();
 * param_1 := obj.Book(start,end);
 */
```

733. Flood Fill

题目

An `image` is represented by a 2-D array of integers, each integer representing the pixel value of the image (from 0 to 65535).

Given a coordinate `(sr, sc)` representing the starting pixel (row and column) of the flood fill, and a pixel value `newColor`, "flood fill" the image.

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 as the starting pixel), and so on. Replace the color of all of the aforementioned pixels with the newColor.

At the end, return the modified image.

Example 1:

```
Input:
image = [[1,1,1],[1,1,0],[1,0,1]]
sr = 1, sc = 1, newColor = 2
Output: [[2,2,2],[2,2,0],[2,0,1]]
Explanation:
From the center of the image (with position (sr, sc) = (1, 1)), all pixels connected
by a path of the same color as the starting pixel 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.
```

Note:

- The length of `image` and `image[0]` will be in the range `[1, 50]`.
- The given starting pixel will satisfy `0 <= sr < image.length` and `0 <= sc < image[0].length`.
- The value of each color in `image[i][j]` and `newColor` will be an integer in `[0, 65535]`.

题目大意

有一幅以二维整数数组表示的图画，每一个整数表示该图画的像素值大小，数值在 0 到 65535 之间。给你一个坐标 (sr, sc) 表示图像渲染开始的像素值（行 ，列）和一个新的颜色值 newColor，让你重新上色这幅图像。

为了完成上色工作，从初始坐标开始，记录初始坐标的上下左右四个方向上像素值与初始坐标相同的相连像素点，接着再记录这四个方向上符合条件的像素点与他们对应四个方向上像素值与初始坐标相同的相连像素点，……，重复该过程。将所有有记录的像素点的颜色值改为新的颜色值。最后返回经过上色渲染后的图像。

注意:

- image 和 image[0] 的长度在范围 [1, 50] 内。
- 给出的初始点将满足 0 <= sr < image.length 和 0 <= sc < image[0].length。
- image[i][j] 和 newColor 表示的颜色值在范围 [0, 65535]内。

解题思路

- 给出一个二维的图片点阵，每个点阵都有一个数字。给出一个起点坐标，要求从这个起点坐标开始，把所有与这个起点连通的点都染色成 newColor。
- 这一题是标准的 Flood Fill 算法。可以用 DFS 也可以用 BFS 。

代码

```
package leetcode
```

```go
func floodFill(image [][]int, sr int, sc int, newColor int) [][]int {
  color := image[sr][sc]
  if newColor == color {
    return image
  }
  dfs733(image, sr, sc, newColor)
  return image
}

func dfs733(image [][]int, x, y int, newColor int) {
  if image[x][y] == newColor {
    return
  }
  oldColor := image[x][y]
  image[x][y] = newColor
  for i := 0; i < 4; i++ {
    if (x+dir[i][0] >= 0 && x+dir[i][0] < len(image)) && (y+dir[i][1] >= 0 && y+dir[i]
[1] < len(image[0])) && image[x+dir[i][0]][y+dir[i][1]] == oldColor {
      dfs733(image, x+dir[i][0], y+dir[i][1], newColor)
    }
  }
}
```

735. Asteroid Collision

题目

We are given an array asteroids of integers representing asteroids in a row.

For each asteroid, the absolute value represents its size, and the sign represents its direction (positive meaning right, negative meaning left). Each asteroid moves at the same speed.

Find out the state of the asteroids after all collisions. If two asteroids meet, the smaller one will explode. If both are the same size, both will explode. Two asteroids moving in the same direction will never meet.

Example 1:

```
Input:
asteroids = [5, 10, -5]
Output: [5, 10]
Explanation:
The 10 and -5 collide resulting in 10.  The 5 and 10 never collide.
```

Example 2:

```
Input:
asteroids = [8, -8]
Output: []
Explanation:
The 8 and -8 collide exploding each other.
```

Example 3:

```
Input:
asteroids = [10, 2, -5]
Output: [10]
Explanation:
The 2 and -5 collide resulting in -5.  The 10 and -5 collide resulting in 10.
```

Example 4:

```
Input:
asteroids = [-2, -1, 1, 2]
Output: [-2, -1, 1, 2]
Explanation:
The -2 and -1 are moving left, while the 1 and 2 are moving right.
Asteroids moving the same direction never meet, so no asteroids will meet each other.
```

Note:

- The length of asteroids will be at most 10000.
- Each asteroid will be a non-zero integer in the range [-1000, 1000]..

题目大意

给定一个整数数组 asteroids，表示在同一行的行星。对于数组中的每一个元素，其绝对值表示行星的大小，正负表示行星的移动方向（正表示向右移动，负表示向左移动）。每一颗行星以相同的速度移动。找出碰撞后剩下的所有行星。碰撞规则：两个行星相互碰撞，较小的行星会爆炸。如果两颗行星大小相同，则两颗行星都会爆炸。两颗移动方向相同的行星，永远不会发生碰撞。

解题思路

这一题类似第 1047 题。这也是一个类似“对对碰”的游戏，不过这里的碰撞，大行星和小行星碰撞以后，大行星会胜出，小行星直接消失。按照题意的规则来，用栈模拟即可。考虑最终结果：

1. 所有向左飞的行星都向左，所有向右飞的行星都向右。
2. 向左飞的行星，如果飞行中没有向右飞行的行星，那么它将安全穿过。

3. 跟踪所有向右移动到右侧的行星，最右边的一个将是第一个面对向左飞行行星碰撞的。
4. 如果它幸存下来，继续前进，否则，任何之前的向右的行星都会被逐一被暴露出来碰撞。

所以先处理这种情况，一层循环把所有能碰撞的向右飞行的行星都碰撞完。碰撞完以后，如果栈顶行星向左飞，新来的行星向右飞，直接添加进来即可。否则栈顶行星向右飞，大小和向左飞的行星一样大小，两者都撞毁灭，弹出栈顶元素。

代码

```
package leetcode

func asteroidCollision(asteroids []int) []int {
  res := []int{}
  for _, v := range asteroids {
    for len(res) != 0 && res[len(res)-1] > 0 && res[len(res)-1] < -v {
      res = res[:len(res)-1]
    }
    if len(res) == 0 || v > 0 || res[len(res)-1] < 0 {
      res = append(res, v)
    } else if v < 0 && res[len(res)-1] == -v {
      res = res[:len(res)-1]
    }
  }
  return res
}
```

739. Daily Temperatures

题目

Given a list of daily temperatures T, return a list such that, for each day in the input, tells you how many days you would have to wait until a warmer temperature. If there is no future day for which this is possible, put 0 instead.

For example, given the list of temperatures T = [73, 74, 75, 71, 69, 72, 76, 73], your output should be [1, 1, 4, 2, 1, 1, 0, 0].

Note: The length of temperatures will be in the range [1, 30000]. Each temperature will be an integer in the range [30, 100].

题目大意

给出一个温度数组，要求输出比当天温度高的在未来的哪一天，输出未来第几天的天数。例如比 73 度高的在未来第 1 天出现，比 75 度高的在未来第 4 天出现。

解题思路

这道题根据题意正常处理就可以了。2 层循环。另外一种做法是单调栈，维护一个单调递减的单调栈即可。

代码

```
package leetcode

// 解法一 普通做法
func dailyTemperatures(T []int) []int {
  res, j := make([]int, len(T)), 0
  for i := 0; i < len(T); i++ {
    for j = i + 1; j < len(T); j++ {
      if T[j] > T[i] {
        res[i] = j - i
        break
      }
    }
  }
  return res
}

// 解法二 单调栈
func dailyTemperatures1(T []int) []int {
  res := make([]int, len(T))
  var toCheck []int
  for i, t := range T {
    for len(toCheck) > 0 && T[toCheck[len(toCheck)-1]] < t {
      idx := toCheck[len(toCheck)-1]
      res[idx] = i - idx
      toCheck = toCheck[:len(toCheck)-1]
    }
    toCheck = append(toCheck, i)
  }
  return res
}
```

744. Find Smallest Letter Greater Than Target

题目

Given a list of sorted characters `letters` containing only lowercase letters, and given a target letter `target`, find the smallest element in the list that is larger than the given target.

Letters also wrap around. For example, if the target is `target = 'z'` and `letters = ['a', 'b']`, the answer is `'a'`.

Examples:

```
Input:
letters = ["c", "f", "j"]
target = "a"
Output: "c"

Input:
letters = ["c", "f", "j"]
target = "c"
Output: "f"

Input:
letters = ["c", "f", "j"]
target = "d"
Output: "f"

Input:
letters = ["c", "f", "j"]
target = "g"
Output: "j"

Input:
letters = ["c", "f", "j"]
target = "j"
Output: "c"

Input:
letters = ["c", "f", "j"]
target = "k"
Output: "c"
```

Note:

1. `letters` has a length in range `[2, 10000]`.
2. `letters` consists of lowercase letters, and contains at least 2 unique letters.
3. `target` is a lowercase letter.

题目大意

给定一个只包含小写字母的有序数组letters 和一个目标字母 target，寻找有序数组里面比目标字母大的最小字母。

数组里字母的顺序是循环的。举个例子，如果目标字母target = 'z' 并且有序数组为 letters = ['a', 'b']，则答案返回 'a'。

注:

1. letters长度范围在[2, 10000]区间内。
2. letters 仅由小写字母组成，最少包含两个不同的字母。
3. 目标字母target 是一个小写字母。

解题思路

- 给出一个字节数组，在这个字节数组中查找在 target 后面的第一个字母。数组是环形的。
- 这一题也是二分搜索的题目，先在数组里面查找 target，如果找到了，取这个字母的后一个字母。如果没有找到，就取 low 下标的那个字母。注意数组是环形的，所以最后结果需要对下标取余。

代码

```go
package leetcode

func nextGreatestLetter(letters []byte, target byte) byte {
  low, high := 0, len(letters)-1
  for low <= high {
    mid := low + (high-low)>>1
    if letters[mid] > target {
      high = mid - 1
    } else {
      low = mid + 1
    }
  }
  find := letters[low%len(letters)]
  if find <= target {
    return letters[0]
  }
  return find
}
```

745. Prefix and Suffix Search

题目

Given many `words`, `words[i]` has weight `i`.

Design a class `WordFilter` that supports one function, `WordFilter.f(String prefix, String suffix)`. It will return the word with given `prefix` and `suffix` with maximum weight. If no word exists, return -1.

Examples:

```
Input:
WordFilter(["apple"])
WordFilter.f("a", "e") // returns 0
WordFilter.f("b", "") // returns -1
```

Note:

1. `words` has length in range `[1, 15000]`.
2. For each test case, up to `words.length` queries `WordFilter.f` may be made.
3. `words[i]` has length in range `[1, 10]`.
4. `prefix, suffix` have lengths in range `[0, 10]`.
5. `words[i]` and `prefix, suffix` queries consist of lowercase letters only.

题目大意

给定多个 words，words[i] 的权重为 i 。设计一个类 WordFilter 实现函数WordFilter.f(String prefix, String suffix)。这个函数将返回具有前缀 prefix 和后缀suffix 的词的最大权重。如果没有这样的词，返回 -1。

解题思路

- 要求实现一个 `WordFilter` ，它具有字符串匹配的功能，可以匹配出前缀和后缀都满足条件的字符串下标，如果找得到，返回下标，如果找不到，则返回 -1 。
- 这一题有 2 种解题思路。第一种是先把这个 `WordFilter` 结构里面的字符串全部预处理一遍，将它的前缀，后缀的所有组合都枚举出来放在 map 中，之后匹配的时候只需要按照自己定义的规则查找 key 就可以了。初始化时间复杂度 `O(N * L^2)`，查找时间复杂度 `O(1)`，空间复杂度 `O(N * L^2)`。其中 `N` 是输入的字符串数组的长度，`L` 是输入字符串数组中字符串的最大长度。第二种思路是直接遍历字符串每个下标，依次用字符串的前缀匹配方法和后缀匹配方法，依次匹配。初始化时间复杂度 `O(1)`，查找时间复杂度 `O(N * L)`，空间复杂度 `O(1)`。其中 `N` 是输入的字符串数组的长度，`L` 是输入字符串数组中字符串的最大长度。

代码

```go
package leetcode

import "strings"

// 解法一 查找时间复杂度 O(1)
type WordFilter struct {
  words map[string]int
}

func Constructor745(words []string) WordFilter {
```

```go
  wordsMap := make(map[string]int, len(words)*5)
  for k := 0; k < len(words); k++ {
    for i := 0; i <= 10 && i <= len(words[k]); i++ {
      for j := len(words[k]); 0 <= j && len(words[k])-10 <= j; j-- {
        ps := words[k][:i] + "#" + words[k][j:]
        wordsMap[ps] = k
      }
    }
  }
  return WordFilter{words: wordsMap}
}

func (this *WordFilter) F(prefix string, suffix string) int {
  ps := prefix + "#" + suffix
  if index, ok := this.words[ps]; ok {
    return index
  }
  return -1
}

// 解法二 查找时间复杂度 O(N * L)
type WordFilter_ struct {
  input []string
}

func Constructor_745_(words []string) WordFilter_ {
  return WordFilter_{input: words}
}

func (this *WordFilter_) F_(prefix string, suffix string) int {
  for i := len(this.input) - 1; i >= 0; i-- {
    if strings.HasPrefix(this.input[i], prefix) && strings.HasSuffix(this.input[i],
suffix) {
      return i
    }
  }
  return -1
}

/**
 * Your WordFilter object will be instantiated and called as such:
 * obj := Constructor(words);
 * param_1 := obj.F(prefix,suffix);
 */
```

746. Min Cost Climbing Stairs

题目

On a staircase, the `i`-th step has some non-negative cost `cost[i]` assigned (0 indexed).

Once you pay the cost, you can either climb one or two steps. You need to find minimum cost to reach the top of the floor, and you can either start from the step with index 0, or the step with index 1.

Example 1:

```
Input: cost = [10, 15, 20]
Output: 15
Explanation: Cheapest is start on cost[1], pay that cost and go to the top.
```

Example 2:

```
Input: cost = [1, 100, 1, 1, 1, 100, 1, 1, 100, 1]
Output: 6
Explanation: Cheapest is start on cost[0], and only step on 1s, skipping cost[3].
```

Note:

1. `cost` will have a length in the range `[2, 1000]`.
2. Every `cost[i]` will be an integer in the range `[0, 999]`.

题目大意

数组的每个索引做为一个阶梯，第 i 个阶梯对应着一个非负数的体力花费值 cost[i] (索引从 0 开始)。每当你爬上一个阶梯你都要花费对应的体力花费值，然后你可以选择继续爬一个阶梯或者爬两个阶梯。您需要找到达到楼层顶部的最低花费。在开始时，你可以选择从索引为 0 或 1 的元素作为初始阶梯。

解题思路

- 这一题算是第 70 题的加强版。依旧是爬楼梯的问题，解题思路也是 DP。在爬楼梯的基础上增加了一个新的条件，每层楼梯都有一个 cost 花费，问上到最终楼层，花费最小值是多少。
- `dp[i]` 代表上到第 n 层的最小花费，状态转移方程是 `dp[i] = cost[i] + min(dp[i-2], dp[i-1])`，最终第 n 层的最小花费是 `min(dp[n-2], dp[n-1])` 。
- 由于每层的花费只和前两层有关系，所以每次 DP 迭代的时候只需要 2 个临时变量即可。可以用这种方式来优化辅助空间。

代码

```go
package leetcode

// 解法一 DP
func minCostClimbingStairs(cost []int) int {
```

```go
  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-2], dp[i-1])
  }
  return min(dp[len(cost)-2], dp[len(cost)-1])
}

// 解法二 DP 优化辅助空间
func minCostClimbingStairs1(cost []int) int {
  var cur, last int
  for i := 2; i < len(cost)+1; i++ {
    if last+cost[i-1] > cur+cost[i-2] {
      cur, last = last, cur+cost[i-2]
    } else {
      cur, last = last, last+cost[i-1]
    }
  }
  return last
}
```

748. Shortest Completing Word

题目

Find the minimum length word from a given dictionary `words`, which has all the letters from the string `licensePlate`. Such a word is said to complete the given string `licensePlate`

Here, for letters we ignore case. For example, `"P"` on the `licensePlate` still matches `"p"` on the word.

It is guaranteed an answer exists. If there are multiple answers, return the one that occurs first in the array.

The license plate might have the same letter occurring multiple times. For example, given a `licensePlate` of `"PP"`, the word `"pair"` does not complete the `licensePlate`, but the word `"supper"` does.

Example 1:

```
Input: licensePlate = "1s3 PSt", words = ["step", "steps", "stripe", "stepple"]
Output: "steps"
Explanation: The smallest length word that contains the letters "S", "P", "S", and "T".
Note that the answer is not "step", because the letter "s" must occur in the word
twice.
Also note that we ignored case for the purposes of comparing whether a letter exists in
the word.
```

Example 2:

```
Input: licensePlate = "1s3 456", words = ["looks", "pest", "stew", "show"]
Output: "pest"
Explanation: There are 3 smallest length words that contains the letters "s".
We return the one that occurred first.
```

Note:

1. `licensePlate` will be a string with length in range `[1, 7]`.
2. `licensePlate` will contain digits, spaces, or letters (uppercase or lowercase).
3. `words` will have a length in the range `[10, 1000]`.
4. Every `words[i]` will consist of lowercase letters, and have length in range `[1, 15]`.

题目大意

如果单词列表（words）中的一个单词包含牌照（licensePlate）中所有的字母，那么我们称之为完整词。在所有完整词中，最短的单词我们称之为最短完整词。

单词在匹配牌照中的字母时不区分大小写，比如牌照中的 "P" 依然可以匹配单词中的 "p" 字母。我们保证一定存在一个最短完整词。当有多个单词都符合最短完整词的匹配条件时取单词列表中最靠前的一个。牌照中可能包含多个相同的字符，比如说：对于牌照 "PP"，单词 "pair" 无法匹配，但是 "supper" 可以匹配。

注意:

- 牌照（licensePlate）的长度在区域[1, 7]中。
- 牌照（licensePlate）将会包含数字、空格、或者字母（大写和小写）。
- 单词列表（words）长度在区间 [10, 1000] 中。
- 每一个单词 words[i] 都是小写，并且长度在区间 [1, 15] 中。

解题思路

- 给出一个数组，要求找出能包含 `licensePlate` 字符串中所有字符的最短长度的字符串。如果最短长度的字符串有多个，输出 word 下标小的那个。这一题也是简单题，不过有 2 个需要注意的点，第一点，`licensePlate` 中可能包含 `Unicode` 任意的字符，所以要先把字母的字符筛选出来，第二点是题目中保证了一定存在一个最短的单词能满足题意，并且忽略大小写。具体做法按照题意模拟即可。

代码

```go
package leetcode

import "unicode"

func shortestCompletingWord(licensePlate string, words []string) string {
  lp := genCnter(licensePlate)
  var ret string
  for _, w := range words {
    if match(lp, w) {
```

```go
      if len(w) < len(ret) || ret == "" {
        ret = w
      }
    }
  }
  return ret
}

func genCnter(lp string) [26]int {
  cnter := [26]int{}
  for _, ch := range lp {
    if unicode.IsLetter(ch) {
      cnter[unicode.ToLower(ch)-'a']++
    }
  }
  return cnter
}

func match(lp [26]int, w string) bool {
  m := [26]int{}
  for _, ch := range w {
    m[ch-'a']++
  }
  for k, v := range lp {
    if m[k] < v {
      return false
    }
  }
  return true
}
```

753. Cracking the Safe

题目

There is a box protected by a password. The password is a sequence of `n` digits where each digit can be one of the first `k` digits `0, 1, ..., k-1`.

While entering a password, the last `n` digits entered will automatically be matched against the correct password.

For example, assuming the correct password is `"345"`, if you type `"012345"`, the box will open because the correct password matches the suffix of the entered password.

Return any password of **minimum length** that is guaranteed to open the box at some point of entering it.

Example 1:

```
Input: n = 1, k = 2
Output: "01"
Note: "10" will be accepted too.
```

Example 2:

```
Input: n = 2, k = 2
Output: "00110"
Note: "01100", "10011", "11001" will be accepted too.
```

Note:

1. `n` will be in the range `[1, 4]`.
2. `k` will be in the range `[1, 10]`.
3. `k^n` will be at most `4096`.

题目大意

有一个需要密码才能打开的保险箱。密码是 n 位数, 密码的每一位是 k 位序列 0, 1, ..., k-1 中的一个 。你可以随意输入密码，保险箱会自动记住最后 n 位输入，如果匹配，则能够打开保险箱。举个例子，假设密码是 "345"，你可以输入 "012345" 来打开它，只是你输入了 6 个字符.请返回一个能打开保险箱的最短字符串。

提示：

- n 的范围是 [1, 4]。
- k 的范围是 [1, 10]。
- k^n 最大可能为 4096。

解题思路

- 给出 2 个数字 n 和 k，n 代表密码是 n 位数，k 代表密码是 k 位。保险箱会记住最后 n 位输入。返回一个能打开保险箱的最短字符串。
- 看到题目中的数据范围，数据范围很小，所以可以考虑用 DFS。想解开保险箱，当然是暴力破解，枚举所有可能。题目要求我们输出一个最短的字符串，这里是本题的关键，为何有最短呢？这里有贪心的思想。如果下一次递归可以利用上一次的 n-1 位，那么最终输出的字符串肯定是最短的。(笔者这里就不证明了)，例如，例子 2 中，最短的字符串是 00，01，11，10。每次尝试都利用前一次的 n-1 位。想通了这个问题，利用 DFS 暴力回溯即可。

代码

```go
const number = "0123456789"

func crackSafe(n int, k int) string {
  if n == 1 {
    return number[:k]
  }
  visit, total := map[string]bool{}, int(math.Pow(float64(k), float64(n)))
  str := make([]byte, 0, total+n-1)
```

```go
  for i := 1; i != n; i++ {
    str = append(str, '0')
  }
  dfsCrackSafe(total, n, k, &str, &visit)
  return string(str)
}

func dfsCrackSafe(depth, n, k int, str *[]byte, visit *map[string]bool) bool {
  if depth == 0 {
    return true
  }
  for i := 0; i != k; i++ {
    *str = append(*str, byte('0'+i))
    cur := string((*str)[len(*str)-n:])
    if _, ok := (*visit)[cur]; ok != true {
      (*visit)[cur] = true
      if dfsCrackSafe(depth-1, n, k, str, visit) {
        // 只有这里不需要删除
        return true
      }
      delete(*visit, cur)
    }
    // 删除
    *str = (*str)[0 : len(*str)-1]
  }
  return false
}
```

756. Pyramid Transition Matrix

题目

We are stacking blocks to form a pyramid. Each block has a color which is a one letter string.

We are allowed to place any color block `C` on top of two adjacent blocks of colors `A` and `B`, if and only if `ABC` is an allowed triple.

We start with a bottom row of `bottom`, represented as a single string. We also start with a list of allowed triples `allowed`. Each allowed triple is represented as a string of length 3.

Return true if we can build the pyramid all the way to the top, otherwise false.

Example 1:

```
Input: bottom = "BCD", allowed = ["BCG", "CDE", "GEA", "FFF"]
Output: true
Explanation:
We can stack the pyramid like this:
    A
   / \
  G   E
 / \ / \
B   C   D

We are allowed to place G on top of B and C because BCG is an allowed triple.
Similarly, we can place E on top of C and D, then A on top of G and E.
```

Example 2:

```
Input: bottom = "AABA", allowed = ["AAA", "AAB", "ABA", "ABB", "BAC"]
Output: false
Explanation:
We can't stack the pyramid to the top.
Note that there could be allowed triples (A, B, C) and (A, B, D) with C != D.
```

Note:

1. `bottom` will be a string with length in range `[2, 8]`.
2. `allowed` will have length in range `[0, 200]`.
3. Letters in all strings will be chosen from the set `{'A', 'B', 'C', 'D', 'E', 'F', 'G'}`.

题目大意

现在，我们用一些方块来堆砌一个金字塔。 每个方块用仅包含一个字母的字符串表示，例如 “Z”。使用三元组表示金字塔的堆砌规则如下：

(A, B, C) 表示，“C” 为顶层方块，方块 “A”、“B” 分别作为方块 “C” 下一层的的左、右子块。当且仅当(A, B, C)是被允许的三元组，我们才可以将其堆砌上。

初始时，给定金字塔的基层 bottom，用一个字符串表示。一个允许的三元组列表 allowed，每个三元组用一个长度为 3 的字符串表示。如果可以由基层一直堆到塔尖返回 true，否则返回 false。

解题思路

- 这一题是一道 DFS 的题目。题目给出金字塔的底座字符串。然后还会给一个字符串数组，字符串数组里面代表的字符串的砖块。砖块是 3 个字符串组成的。前两个字符代表的是砖块的底边，后一个字符代表的是砖块的顶部。问给出的字符能拼成一个金字塔么？金字塔的特点是顶端就一个字符。
- 这一题用 DFS 深搜每个砖块，从底层砖块开始逐渐往上层码。每递归一层，新一层底部的砖块都会变。当递归到了一层底部只有 2 个字符，顶部只有一个字符的时候，就到金字塔顶端了，就算是完成了。这一题为了挑选合适的砖块，需要把每个砖块底部的 2 个字符作为 key 放进 map 中，加速查找。题目中也给出了特殊情况，相同底部可能存在多种砖块，所以一个 key 可能对应多个 value 的情况，即可能存在多个顶部砖块的情

况。这种情况在递归遍历中需要考虑。

代码

```
package leetcode

func pyramidTransition(bottom string, allowed []string) bool {
  pyramid := make(map[string][]string)
  for _, v := range allowed {
    pyramid[v[:len(v)-1]] = append(pyramid[v[:len(v)-1]], string(v[len(v)-1]))
  }
  return dfsT(bottom, "", pyramid)
}

func dfsT(bottom, above string, pyramid map[string][]string) bool {
  if len(bottom) == 2 && len(above) == 1 {
    return true
  }
  if len(bottom) == len(above)+1 {
    return dfsT(above, "", pyramid)
  }
  base := bottom[len(above) : len(above)+2]
  if data, ok := pyramid[base]; ok {
    for _, key := range data {
      if dfsT(bottom, above+key, pyramid) {
        return true
      }
    }
  }
  return false
}
```

762. Prime Number of Set Bits in Binary Representation

题目

Given two integers `L` and `R`, find the count of numbers in the range `[L, R]` (inclusive) having a prime number of set bits in their binary representation.

(Recall that the number of set bits an integer has is the number of `1`s present when written in binary. For example, `21` written in binary is `10101` which has 3 set bits. Also, 1 is not a prime.)

Example 1:

```
Input: L = 6, R = 10
Output: 4
Explanation:
6 -> 110 (2 set bits, 2 is prime)
7 -> 111 (3 set bits, 3 is prime)
9 -> 1001 (2 set bits , 2 is prime)
10->1010 (2 set bits , 2 is prime)
```

Example 2:

```
Input: L = 10, R = 15
Output: 5
Explanation:
10 -> 1010 (2 set bits, 2 is prime)
11 -> 1011 (3 set bits, 3 is prime)
12 -> 1100 (2 set bits, 2 is prime)
13 -> 1101 (3 set bits, 3 is prime)
14 -> 1110 (3 set bits, 3 is prime)
15 -> 1111 (4 set bits, 4 is not prime)
```

Note:

1. `L, R` will be integers `L <= R` in the range `[1, 10^6]`.
2. `R - L` will be at most 10000.

题目大意

给定两个整数 L 和 R ，找到闭区间 [L, R] 范围内，计算置位位数为质数的整数个数。（注意，计算置位代表二进制表示中1的个数。例如 21 的二进制表示 10101 有 3 个计算置位。还有，1 不是质数。）

注意:

- L, R 是 L <= R 且在 [1, 10^6] 中的整数。
- R - L 的最大值为 10000。

解题思路

- 题目给出 `[L, R]` 区间，在这个区间内的每个整数的二进制表示中 1 的个数如果是素数，那么最终结果就加一，问最终结果是多少？这一题是一个组合题，判断一个数的二进制位有多少位 1，是第 191 题。题目中限定了区间最大不超过 10^6 ，所以 1 的位数最大是 19 位，也就是说素数最大就是 19 。那么素数可以有限枚举出来。最后按照题目的意思累积结果就可以了。

代码

```
package leetcode
```

```go
import "math/bits"

func countPrimeSetBits(L int, R int) int {
  counter := 0
  for i := L; i <= R; i++ {
    if isPrime(bits.OnesCount(uint(i))) {
      counter++
    }
  }
  return counter
}

func isPrime(x int) bool {
  return x == 2 || x == 3 || x == 5 || x == 7 || x == 11 || x == 13 || x == 17 || x == 19
}
```

763. Partition Labels

题目

A string S of lowercase letters is given. We want to partition this string into as many parts as possible so that each letter appears in at most one part, and return a list of integers representing the size of these parts.

Example 1:

```
Input: S = "ababcbacadefegdehijhklij"
Output: [9,7,8]
Explanation:
The partition is "ababcbaca", "defegde", "hijhklij".
This is a partition so that each letter appears in at most one part.
A partition like "ababcbacadefegde", "hijhklij" is incorrect, because it splits S into less parts.
```

Note:

- S will have length in range [1, 500].
- S will consist of lowercase letters ('a' to 'z') only.

题目大意

这道题考察的是滑动窗口的问题。

给出一个字符串，要求输出满足条件窗口的长度，条件是在这个窗口内，字母中出现在这一个窗口内，不出现在其他窗口内。

解题思路

这一题有 2 种思路，第一种思路是先记录下每个字母的出现次数，然后对滑动窗口中的每个字母判断次数是否用尽为 0，如果这个窗口内的所有字母次数都为 0，这个窗口就是符合条件的窗口。时间复杂度为 O(n^2)

另外一种思路是记录下每个字符最后一次出现的下标，这样就不用记录次数。在每个滑动窗口中，依次判断每个字母最后一次出现的位置，如果在一个下标内，所有字母的最后一次出现的位置都包含进来了，那么这个下标就是这个满足条件的窗口大小。时间复杂度为 O(n^2)

代码

```
package leetcode

// 解法一
func partitionLabels(S string) []int {
  var lastIndexOf [26]int
  for i, v := range S {
    lastIndexOf[v-'a'] = i
  }

  var arr []int
  for start, end := 0, 0; start < len(S); start = end + 1 {
    end = lastIndexOf[S[start]-'a']
    for i := start; i < end; i++ {
      if end < lastIndexOf[S[i]-'a'] {
        end = lastIndexOf[S[i]-'a']
      }
    }
    arr = append(arr, end-start+1)
  }
  return arr
}

// 解法二
func partitionLabels1(S string) []int {
  visit, counter, res, sum, lastLength := make([]int, 26), map[byte]int{}, []int{}, 0,
0
  for i := 0; i < len(S); i++ {
    counter[S[i]]++
  }

  for i := 0; i < len(S); i++ {
    counter[S[i]]--
    visit[S[i]-'a'] = 1
    sum = 0
```

```
    for j := 0; j < 26; j++ {
      if visit[j] == 1 {
        sum += counter[byte('a'+j)]
      }
    }
    if sum == 0 {
      res = append(res, i+1-lastLength)
      lastLength += i + 1 - lastLength
    }
  }
  return res
}
```

765. Couples Holding Hands

题目

N couples sit in 2N seats arranged in a row and want to hold hands. We want to know the minimum number of swaps so that every couple is sitting side by side. A swap consists of choosing **any** two people, then they stand up and switch seats.

The people and seats are represented by an integer from `0` to `2N-1`, the couples are numbered in order, the first couple being `(0, 1)`, the second couple being `(2, 3)`, and so on with the last couple being `(2N-2, 2N-1)`.

The couples' initial seating is given by `row[i]` being the value of the person who is initially sitting in the i-th seat.

Example 1:

```
Input: row = [0, 2, 1, 3]
Output: 1
Explanation: We only need to swap the second (row[1]) and third (row[2]) person.
```

Example 2:

```
Input: row = [3, 2, 0, 1]
Output: 0
Explanation: All couples are already seated side by side.
```

Note:

1. `len(row)` is even and in the range of `[4, 60]`.
2. `row` is guaranteed to be a permutation of `0...len(row)-1`.

题目大意

N 对情侣坐在连续排列的 2N 个座位上，想要牵到对方的手。 计算最少交换座位的次数，以便每对情侣可以并肩坐在一起。 一次交换可选择任意两人，让他们站起来交换座位。人和座位用 0 到 2N-1 的整数表示，情侣们按顺序编号，第一对是 (0, 1)，第二对是 (2, 3)，以此类推，最后一对是 (2N-2, 2N-1)。这些情侣的初始座位 row[i] 是由最初始坐在第 i 个座位上的人决定的。

说明:

1. len(row) 是偶数且数值在 [4, 60]范围内。
2. 可以保证 row 是序列 0...len(row)-1 的一个全排列。

解题思路

- 给出一个数组，数组里面两两相邻的元素代表一对情侣。情侣编号是从 0 开始的：0 和 1 是情侣，2 和 3 是情侣.....这些情侣坐在一排，但是并非成对坐着一起的，问如何用最小的次数交换座位以后，情侣能两两坐在一起。
- 这道题的突破口是如何找到最小的交换次数。乍一想可能没有思路。直觉告诉我们，这种难题，很可能最后推出来的结论，或者公式是一个很简单的式子。(事实此题确实是这种情况)先不考虑最小交换次数，用正常的方法来处理这道题。举个例子：【3 1 4 0 2 5】，从数组 0 下标开始往后扫。

  ```
  初始状态

  集合 0：0，1
  集合 1：2，3
  集合 2：4，5
  ```

  3 和 1 不是情侣，将 3 和 1 所在集合 `union()` 起来。3 所在集合是 1 ，1 所在集合是 0，将 0 和 1 号集合 `union()` 起来。因为情侣 0 和情侣 1 是集合 0 ，情侣 2 和情侣 3 是集合 1，以此类推。

  ```
  集合 0 和 1：0，1，2，3
  集合 2：4，5
  ```

- 继续往后扫，4 和 0 不在同一个集合，4 在集合 3，0 在集合 0，那么把它们 `union()` 起来。

  ```
  集合 0 和 1 和 2：0，1，2，3，4，5
  ```

  在上面集合合并的过程中，合并了 2 次。那么就代表最少需要交换 2 次。也可以通过 `len(row)/2 - uf.count` 来计算。`len(row)/2` 是初始集合总数，`uf.count` 是最后剩下的集合数，两者相减就是中间交换的次数。
- 最后实现的代码非常简单。并查集先相邻的两两元素 `union()` 在一起。然后扫原数组，每次扫相邻的两个，通过这两个元素值所在集合，进行 `union()`。扫完以后就可以得到最后的答案。
- 回过头来看这道题，为什么我们从数组开头往后依次调整每一对情侣，这样交换的次数是最少的呢？其实这个方法的思想是贪心思想。从头开始往后一对一对的调整，就是可以最终做到次数最少。(具体证明笔者不会)交换到最后，最后一对情侣一定是正确的，无须交换。(因为前面每一对都调整完了，最后一对一定是正确的)

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/template"
)

func minSwapsCouples(row []int) int {
  if len(row)&1 == 1 {
    return 0
  }
  uf := template.UnionFind{}
  uf.Init(len(row))
  for i := 0; i < len(row)-1; i = i + 2 {
    uf.Union(i, i+1)
  }
  for i := 0; i < len(row)-1; i = i + 2 {
    if uf.Find(row[i]) != uf.Find(row[i+1]) {
      uf.Union(row[i], row[i+1])
    }
  }
  return len(row)/2 - uf.TotalCount()
}
```

766. Toeplitz Matrix

题目

A matrix is *Toeplitz* if every diagonal from top-left to bottom-right has the same element.

Now given an `M x N` matrix, return `True` if and only if the matrix is *Toeplitz*.

Example 1:

```
Input:
matrix = [
  [1,2,3,4],
  [5,1,2,3],
  [9,5,1,2]
]
Output: True
Explanation:
In the above grid, the diagonals are:
"[9]", "[5, 5]", "[1, 1, 1]", "[2, 2, 2]", "[3, 3]", "[4]".
In each diagonal all elements are the same, so the answer is True.
```

Example 2:

```
Input:
matrix = [
  [1,2],
  [2,2]
]
Output: False
Explanation:
The diagonal "[1, 2]" has different elements.
```

Note:

1. `matrix` will be a 2D array of integers.
2. `matrix` will have a number of rows and columns in range `[1, 20]`.
3. `matrix[i][j]` will be integers in range `[0, 99]`.

Follow up:

1. What if the matrix is stored on disk, and the memory is limited such that you can only load at most one row of the matrix into the memory at once?
2. What if the matrix is so large that you can only load up a partial row into the memory at once?

题目大意

如果一个矩阵的每一方向由左上到右下的对角线上具有相同元素，那么这个矩阵是托普利茨矩阵。给定一个 M x N 的矩阵，当且仅当它是托普利茨矩阵时返回 True。

解题思路

- 给出一个矩阵，要求判断矩阵所有对角斜线上的数字是否都是一个数字。
- 水题，直接循环判断即可。

代码

```go
package leetcode

func isToeplitzMatrix(matrix [][]int) bool {
  rows, columns := len(matrix), len(matrix[0])
  for i := 1; i < rows; i++ {
    for j := 1; j < columns; j++ {
      if matrix[i-1][j-1] != matrix[i][j] {
        return false
      }
    }
  }
  return true
```

```
}
```

767. Reorganize String

题目

Given a string S, check if the letters can be rearranged so that two characters that are adjacent to each other are not the same.

If possible, output any possible result. If not possible, return the empty string.

Example 1:

```
Input: S = "aab"
Output: "aba"
```

Example 2:

```
Input: S = "aaab"
Output: ""
```

Note:

S will consist of lowercase letters and have length in range [1, 500].

题目大意

给定一个字符串，要求重新排列字符串，让字符串两两字符不相同，如果可以实现，即输出最终的字符串，如果不能让两两不相同，则输出空字符串。

解题思路

这道题有 2 种做法。第一种做法是先统计每个字符的出现频率次数，按照频率次数从高往低排序。具体做法就是第 451 题了。如果有一个字母的频次次数超过了 (len(string)+1)/2 那么就返回空字符串。否则输出最终满足题意的字符串。按照频次排序以后，用 2 个指针，一个从 0 开始，另外一个从中间位置开始，依次取出一个字符拼接起来。

第二种做法是用优先队列，结点是一个结构体，结构体有 2 个字段，一个字段记录是哪个字符，另一个字段记录是这个字符的频次。按照频次的多作为优先级高，用大根堆建立优先队列。注意，这样建立成功的优先队列，重复字母只有一个结点，频次记录在结构体的频次字段中。额外还需要一个辅助队列。优先队列每次都出队一个优先级最高的，然后频次减一，最终结果加上这个字符。然后将这个结点入队。入队的意义是检测这个结点的频次有没有减到 0，如果还不为 0 ，再插入优先队列中。

```
    string reorganizeString(string S) {
```

```
        vector<int> mp(26);
        int n = S.size();
        for (char c: S)
            ++mp[c-'a'];
        priority_queue<pair<int, char>> pq;
        for (int i = 0; i < 26; ++i) {
            if (mp[i] > (n+1)/2) return "";
            if (mp[i]) pq.push({mp[i], i+'a'});
        }
        queue<pair<int, char>> myq;
        string ans;
        while (!pq.empty() || myq.size() > 1) {
            if (myq.size() > 1) { // 注意这里要大于 1，如果是等于 1 的话，频次大的元素一直在输出了，答案就不对了。
                auto cur = myq.front();
                myq.pop();
                if (cur.first != 0) pq.push(cur);
            }
            if (!pq.empty()) {
                auto cur = pq.top();
                pq.pop();
                ans += cur.second;
                cur.first--;
                myq.push(cur);
            }
        }
        return ans;
    }
```

代码

```
package leetcode

import (
  "sort"
)
```

```go
func reorganizeString(S string) string {
  fs := frequencySort767(S)
  if fs == "" {
    return ""
  }
  bs := []byte(fs)
  ans := ""
  j := (len(bs)-1)/2 + 1
  for i := 0; i <= (len(bs)-1)/2; i++ {
    ans += string(bs[i])
    if j < len(bs) {
      ans += string(bs[j])
    }
    j++
  }
  return ans
}

func frequencySort767(s string) string {
  if s == "" {
    return ""
  }
  sMap := map[byte]int{}
  cMap := map[int][]byte{}
  sb := []byte(s)
  for _, b := range sb {
    sMap[b]++
    if sMap[b] > (len(sb)+1)/2 {
      return ""
    }
  }
  for key, value := range sMap {
    cMap[value] = append(cMap[value], key)
  }

  var keys []int
  for k := range cMap {
    keys = append(keys, k)
  }
  sort.Sort(sort.Reverse(sort.IntSlice(keys)))
  res := make([]byte, 0)
  for _, k := range keys {
    for i := 0; i < len(cMap[k]); i++ {
      for j := 0; j < k; j++ {
        res = append(res, cMap[k][i])
      }
    }
  }
```

```
    return string(res)
}
```

771. Jewels and Stones

题目

You're given strings `J` representing the types of stones that are jewels, and `S` representing the stones you have. Each character in `S` is a type of stone you have. You want to know how many of the stones you have are also jewels.

The letters in `J` are guaranteed distinct, and all characters in `J` and `S` are letters. Letters are case sensitive, so `"a"` is considered a different type of stone from `"A"`.

Example 1:

```
Input: J = "aA", S = "aAAbbbb"
Output: 3
```

Example 2:

```
Input: J = "z", S = "ZZ"
Output: 0
```

Note:

- `S` and `J` will consist of letters and have length at most 50.
- The characters in `J` are distinct.

题目大意

给定字符串 J 代表石头中宝石的类型，和字符串 S 代表你拥有的石头。S 中每个字符代表了一种你拥有的石头的类型，你想知道你拥有的石头中有多少是宝石。

J 中的字母不重复，J 和 S 中的所有字符都是字母。字母区分大小写，因此 "a" 和 "A" 是不同类型的石头。

解题思路

- 给出 2 个字符串，要求在 S 字符串中找出在 J 字符串里面出现的字符个数。这是一道简单题。

代码

```go
package leetcode

import "strings"

// 解法一
func numJewelsInStones(J string, S string) int {
  count := 0
  for i := range S {
    if strings.Contains(J, string(S[i])) {
      count++
    }
  }
  return count
}

// 解法二
func numJewelsInStones1(J string, S string) int {
  cache, result := make(map[rune]bool), 0
  for _, r := range J {
    cache[r] = true
  }
  for _, r := range S {
    if _, ok := cache[r]; ok {
      result++
    }
  }
  return result
}
```

775. Global and Local Inversions

题目

We have some permutation `A` of `[0, 1, ..., N - 1]`, where `N` is the length of `A`.

The number of (global) inversions is the number of `i < j` with `0 <= i < j < N` and `A[i] > A[j]`.

The number of local inversions is the number of `i` with `0 <= i < N` and `A[i] > A[i+1]`.

Return `true` if and only if the number of global inversions is equal to the number of local inversions.

Example 1:

```
Input: A = [1,0,2]
Output: true
Explanation: There is 1 global inversion, and 1 local inversion.
```

Example 2:

```
Input: A = [1,2,0]
Output: false
Explanation: There are 2 global inversions, and 1 local inversion.
```

Note:

- `A` will be a permutation of `[0, 1, ..., A.length - 1]`.
- `A` will have length in range `[1, 5000]`.
- The time limit for this problem has been reduced.

题目大意

数组 A 是 [0, 1, ..., N - 1] 的一种排列，N 是数组 A 的长度。全局倒置指的是 i,j 满足 0 <= i < j < N 并且 A[i] > A[j] ，局部倒置指的是 i 满足 0 <= i < N 并且 A[i] > A[i+1] 。当数组 A 中全局倒置的数量等于局部倒置的数量时，返回 true 。

解题思路

- 本题代码非常简单，重在思考的过程。`[0, 1, ..., N - 1]` 不出现全局倒置的理想情况应该是 `i` 排列在 `A[i-1]`，`A[i]`，`A[i+1]` 的位置上。例如 `1` 如果排列在 `A[3]` 的位置上，那么比 `1` 小的只有 `0` 一个元素，`A[0]`，`A[1]`，`A[2]` 中必定有 2 个元素比 `1` 大，那必须会出现全局倒置的情况。`[0, 1, ..., N - 1]` 这是最理想的情况，每个元素都在自己的位置上。每个元素如果往左右相互偏移 1 个元素，那么也能保证只存在局部倒置，如果左右偏移 2 个元素，那必定会出现全局倒置。所以结论是：**不出现全局倒置的理想情况应该是 `i` 排列在 `A[i-1]`，`A[i]`，`A[i+1]` 的位置上**。判断这个结论的代码很简单，只需要判断 `A[i] - i` 的取值是否是 -1，0，1，也即 `abs(A[i] - i ) ≤ 1`。

代码

```go
package leetcode

func isIdealPermutation(A []int) bool {
  for i := range A {
    if abs(A[i]-i) > 1 {
      return false
    }
  }
  return true
}

func abs(a int) int {
  if a < 0 {
    return -a
  }
  return a
}
```

778. Swim in Rising Water

题目

On an N x N `grid`, each square `grid[i][j]` represents the elevation at that point `(i,j)`.

Now rain starts to fall. At time `t`, the depth of the water everywhere is `t`. You can swim from a square to another 4-directionally adjacent square if and only if the elevation of both squares individually are at most `t`. You can swim infinite distance in zero time. Of course, you must stay within the boundaries of the grid during your swim.

You start at the top left square `(0, 0)`. What is the least time until you can reach the bottom right square `(N-1, N-1)`?

Example 1:

```
Input: [[0,2],[1,3]]
Output: 3
Explanation:
At time 0, you are in grid location (0, 0).
You cannot go anywhere else because 4-directionally adjacent neighbors have a higher
elevation than t = 0.

You cannot reach point (1, 1) until time 3.
When the depth of water is 3, we can swim anywhere inside the grid.
```

Example 2:

```
Input: [[0,1,2,3,4],[24,23,22,21,5],[12,13,14,15,16],[11,17,18,19,20],[10,9,8,7,6]]
Output: 16
Explanation:
 0  1  2  3  4
24 23 22 21  5
12 13 14 15 16
11 17 18 19 20
10  9  8  7  6

The final route is marked in bold.
We need to wait until time 16 so that (0, 0) and (4, 4) are connected.
```

Note:

1. `2 <= N <= 50`.
2. grid[i][j] is a permutation of [0, ..., N*N - 1].

题目大意

在一个 N x N 的坐标方格 grid 中，每一个方格的值 grid[i][j] 表示在位置 (i,j) 的平台高度。现在开始下雨了。当时间为 t 时，此时雨水导致水池中任意位置的水位为 t 。你可以从一个平台游向四周相邻的任意一个平台，但是前提是此时水位必须同时淹没这两个平台。假定你可以瞬间移动无限距离，也就是默认在方格内部游动是不耗时的。当然，在你游泳的时候你必须待在坐标方格里面。

你从坐标方格的左上平台 (0，0) 出发。最少耗时多久你才能到达坐标方格的右下平台 (N-1, N-1)?

提示:

- 2 <= N <= 50.
- grid[i][j] 位于区间 [0, ..., N*N - 1] 内。

解题思路

- 给出一个 grid[i][j] 方格，每个格子里面表示游泳池里面平台的高度。t 时刻，游泳池中的水的高度是 t。只有水的高度到达了平台的高度以后才能游过去。问从 (0,0) 开始，最短多长时间能到达 (N-1, N-1) 。
- 这一题有多种解法。第一种解题思路是利用 DFS + 二分。DFS 是用来遍历是否可达。利用时间(即当前水淹过的高度)来判断是否能到达终点 (N-1, N-1) 点。二分用来搜索最终结果的时间。为什么会考虑用二分加速呢?原因是：时间从 0 - max 依次递增。max 是游泳池最高的平台高度。当时间从 0 增加到 max 以后，肯定能到达终点 (N-1, N-1) 点，因为水比所有平台都要高了。想快速找到一个时间 t 能使得 (0,0) 点和 (N-1, N-1) 点之间连通，那么就想到用二分加速了。判断是否取中值的条件是 (0,0) 点和 (N-1, N-1) 点之间是否连通。
- 第二种解题思路是并查集。只要是 (0,0) 点和 (N-1, N-1) 点没有连通，即不能游到终点，那么就开始 `union()` 操作，由于起点是 (0,0)，所以向右边 `i + 1` 和向下边 `j + 1` 开始尝试。每尝试完一轮，时间会加 1 秒，即高度会加一。直到 (0,0) 点和 (N-1, N-1) 点刚好连通，那么这个时间点就是最终要求的。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/template"
)

// 解法一 DFS + 二分
func swimInWater(grid [][]int) int {
  row, col, flags, minWait, maxWait := len(grid), len(grid[0]), make([][]int,
len(grid)), 0, 0
  for i, row := range grid {
    flags[i] = make([]int, len(row))
    for j := 0; j < col; j++ {
      flags[i][j] = -1
      if row[j] > maxWait {
        maxWait = row[j]
      }
    }
  }
  for minWait < maxWait {
    midWait := (minWait + maxWait) / 2
```

```go
    addFlags(grid, flags, midWait, 0, 0)
    if flags[row-1][col-1] == midWait {
      maxWait = midWait
    } else {
      minWait = midWait + 1
    }
  }
  return minWait
}

func addFlags(grid [][]int, flags [][]int, flag int, row int, col int) {
  if row < 0 || col < 0 || row >= len(grid) || col >= len(grid[0]) {
    return
  }
  if grid[row][col] > flag || flags[row][col] == flag {
    return
  }
  flags[row][col] = flag
  addFlags(grid, flags, flag, row-1, col)
  addFlags(grid, flags, flag, row+1, col)
  addFlags(grid, flags, flag, row, col-1)
  addFlags(grid, flags, flag, row, col+1)
}

// 解法二 并查集(并不是此题的最优解)
func swimInWater1(grid [][]int) int {
  n, uf, res := len(grid), template.UnionFind{}, 0
  uf.Init(n * n)
  for uf.Find(0) != uf.Find(n*n-1) {
    for i := 0; i < n; i++ {
      for j := 0; j < n; j++ {
        if grid[i][j] > res {
          continue
        }
        if i < n-1 && grid[i+1][j] <= res {
          uf.Union(i*n+j, i*n+j+n)
        }
        if j < n-1 && grid[i][j+1] <= res {
          uf.Union(i*n+j, i*n+j+1)
        }
      }
    }
    res++
  }
  return res - 1
}
```

781. Rabbits in Forest

题目

In a forest, each rabbit has some color. Some subset of rabbits (possibly all of them) tell you how many other rabbits have the same color as them. Those `answers` are placed in an array.

Return the minimum number of rabbits that could be in the forest.

Examples:

```
Input: answers = [1, 1, 2]
Output: 5
Explanation:
The two rabbits that answered "1" could both be the same color, say red.
The rabbit than answered "2" can't be red or the answers would be inconsistent.
Say the rabbit that answered "2" was blue.
Then there should be 2 other blue rabbits in the forest that didn't answer into the
array.
The smallest possible number of rabbits in the forest is therefore 5: 3 that answered
plus 2 that didn't.

Input: answers = [10, 10, 10]
Output: 11

Input: answers = []
Output: 0
```

Note:

1. `answers` will have length at most `1000`.
2. Each `answers[i]` will be an integer in the range `[0, 999]`.

题目大意

森林中，每个兔子都有颜色。其中一些兔子（可能是全部）告诉你还有多少其他的兔子和自己有相同的颜色。我们将这些回答放在 answers 数组里。返回森林中兔子的最少数量。

说明:

- answers 的长度最大为1000。
- answers[i] 是在 [0, 999] 范围内的整数。

解题思路

- 给出一个数组，数组里面代表的是每个兔子说自己同类还有多少个。要求输出总共有多少只兔子。数字中可能兔子汇报的人数小于总兔子数。
- 这一题关键在于如何划分不同种类的兔子，有可能相同种类的兔子的个数是一样的，比如 `[2,2,2,2,2,2]`，这其实是 3 个种类，总共 6 只兔子。用 map 去重相同种类的兔子，不断的减少，当有种类的兔子为 0 以后，

还有该种类的兔子报数，需要当做另外一个种类的兔子来看待。

代码

```
package leetcode

func numRabbits(ans []int) int {
  total, m := 0, make(map[int]int)
  for _, v := range ans {
    if m[v] == 0 {
      m[v] += v
      total += v + 1
    } else {
      m[v]--
    }
  }
  return total
}
```

783. Minimum Distance Between BST Nodes

题目

Given the `root` of a Binary Search Tree (BST), return *the minimum difference between the values of any two different nodes in the tree*.

Note: This question is the same as 530: https://leetcode.com/problems/minimum-absolute-difference-in-bst/

Example 1:

```
Input: root = [4,2,6,1,3]
Output: 1
```

Example 2:

```
Input: root = [1,0,48,null,null,12,49]
Output: 1
```

Constraints:

- The number of nodes in the tree is in the range `[2, 100]`.
- `0 <= Node.val <= 10^5`

题目大意

给你一个二叉搜索树的根节点 root ， 返回 树中任意两不同节点值之间的最小差值 。

解题思路

- 本题和第 530 题完全相同。解题思路见第 530 题。

代码

```go
package leetcode

import (
  "math"

  "github.com/halfrost/LeetCode-Go/structures"
)

// TreeNode define
```

```go
type TreeNode = structures.TreeNode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

func minDiffInBST(root *TreeNode) int {
  res, nodes := math.MaxInt16, -1
  dfsBST(root, &res, &nodes)
  return res
}

func dfsBST(root *TreeNode, res, pre *int) {
  if root == nil {
    return
  }
  dfsBST(root.Left, res, pre)
  if *pre != -1 {
    *res = min(*res, abs(root.Val-*pre))
  }
  *pre = root.Val
  dfsBST(root.Right, res, pre)
}

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

func abs(a int) int {
  if a > 0 {
    return a
  }
  return -a
}
```

784. Letter Case Permutation

题目

Given a string S, we can transform every letter individually to be lowercase or uppercase to create another string. Return a list of all possible strings we could create.

Examples:

```
Input: S = "a1b2"
Output: ["a1b2", "a1B2", "A1b2", "A1B2"]

Input: S = "3z4"
Output: ["3z4", "3Z4"]

Input: S = "12345"
Output: ["12345"]
```

Note:

- `S` will be a string with length between `1` and `12`.
- `S` will consist only of letters or digits.

题目大意

给定一个字符串 S，通过将字符串 S 中的每个字母转变大小写，我们可以获得一个新的字符串。返回所有可能得到的字符串集合。

解题思路

- 输出一个字符串中字母变大写，小写的所有组合。
- DFS 深搜或者 BFS 广搜都可以。

代码

```go
package leetcode

import (
  "strings"
)

// 解法一，DFS 深搜
func letterCasePermutation(S string) []string {
  if len(S) == 0 {
    return []string{}
  }
  res, pos, c := []string{}, []int{}, []int{}
  SS := strings.ToLower(S)
  for i := 0; i < len(SS); i++ {
    if isLowerLetter(SS[i]) {
      pos = append(pos, i)
    }
```

```go
  }
  for i := 0; i <= len(pos); i++ {
    findLetterCasePermutation(SS, pos, i, 0, c, &res)
  }
  return res
}

func findLetterCasePermutation(s string, pos []int, target, index int, c []int, res *
[]string) {
  if len(c) == target {
    b := []byte(s)
    for _, v := range c {
      b[pos[v]] -= 'a' - 'A'
    }
    *res = append(*res, string(b))
    return
  }
  for i := index; i < len(pos)-(target-len(c))+1; i++ {
    c = append(c, i)
    findLetterCasePermutation(s, pos, target, i+1, c, res)
    c = c[:len(c)-1]
  }
}

// 解法二，先讲第一个字母变大写，然后依次把后面的字母变大写。最终的解数组中答案是翻倍增长的
// 第一步:
// [mqe] -> [mqe, Mqe]
// 第二步:
// [mqe, Mqe] -> [mqe Mqe mQe MQe]
// 第二步:
// [mqe Mqe mQe MQe] -> [mqe Mqe mQe MQe mqE MqE mQE MQE]

func letterCasePermutation1(S string) []string {
  res := make([]string, 0, 1<<uint(len(S)))
  S = strings.ToLower(S)
  for k, v := range S {
    if isLetter784(byte(v)) {
      switch len(res) {
      case 0:
        res = append(res, S, toUpper(S, k))
      default:
        for _, s := range res {
          res = append(res, toUpper(s, k))
        }
      }
    }
  }
  if len(res) == 0 {
    res = append(res, S)
```

```go
  }
  return res
}

func isLetter784(c byte) bool {
  return (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')
}

func toUpper(s string, i int) string {
  b := []byte(s)
  b[i] -= 'a' - 'A'
  return string(b)
}
```

785. Is Graph Bipartite?

题目

Given an undirected `graph`, return `true` if and only if it is bipartite.

Recall that a graph is *bipartite* if we can split it's set of nodes into two independent subsets A and B such that every edge in the graph has one node in A and another node in B.

The graph is given in the following form: `graph[i]` is a list of indexes `j` for which the edge between nodes `i` and `j` exists. Each node is an integer between `0` and `graph.length - 1`. There are no self edges or parallel edges: `graph[i]` does not contain `i`, and it doesn't contain any element twice.

```
Example 1:Input: [[1,3], [0,2], [1,3], [0,2]]
Output: true
Explanation:
The graph looks like this:
0----1
|    |
|    |
3----2
We can divide the vertices into two groups: {0, 2} and {1, 3}.
```

```
Example 2:Input: [[1,2,3], [0,2], [0,1,3], [0,2]]
Output: false
Explanation:
The graph looks like this:
0----1
| \  |
|  \ |
3----2
We cannot find a way to divide the set of nodes into two independent subsets.
```

Note:

- `graph` will have length in range `[1, 100]`.
- `graph[i]` will contain integers in range `[0, graph.length - 1]`.
- `graph[i]` will not contain `i` or duplicate values.
- The graph is undirected: if any element `j` is in `graph[i]`, then `i` will be in `graph[j]`.

题目大意

给定一个无向图 graph，当这个图为二分图时返回 true。

graph 将会以邻接表方式给出，graph[i] 表示图中与节点i相连的所有节点。每个节点都是一个在 0 到 graph.length-1 之间的整数。这图中没有自环和平行边： graph[i] 中不存在 i，并且 graph[i] 中没有重复的值。

注意:

- graph 的长度范围为 [1, 100]。
- graph[i] 中的元素的范围为 [0, graph.length - 1]。
- graph[i] 不会包含 i 或者有重复的值。
- 图是无向的: 如果 j 在 graph[i] 里边, 那么 i 也会在 graph[j] 里边。

解题思路

- 判断一个无向图是否是二分图。二分图的定义：如果我们能将一个图的节点集合分割成两个独立的子集 A 和 B，并使图中的每一条边的两个节点一个来自 A 集合，一个来自 B 集合，我们就将这个图称为二分图。
- 这一题可以用 BFS、DFS、并查集来解答。这里是 DFS 实现。任选一个节点开始，把它染成红色，然后对整个图 DFS 遍历，把与它相连的节点并且未被染色的，都染成绿色。颜色不同的节点代表不同的集合。这时候还可能遇到第 2 种情况，与它相连的节点已经有颜色了，并且这个颜色和前一个节点的颜色相同，这就说明了该无向图不是二分图。可以直接 return false。如此遍历到所有节点都染色了，如果能染色成功，说明该无向图是二分图，返回 true。

代码

```go
package leetcode

// DFS 染色，1 是红色，0 是绿色，-1 是未染色
func isBipartite(graph [][]int) bool {
  colors := make([]int, len(graph))
  for i := range colors {
    colors[i] = -1
  }
  for i := range graph {
    if !dfs(i, graph, colors, -1) {
      return false
    }
  }
  return true
}
```

```go
func dfs(n int, graph [][]int, colors []int, parentCol int) bool {
  if colors[n] == -1 {
    if parentCol == 1 {
      colors[n] = 0
    } else {
      colors[n] = 1
    }
  } else if colors[n] == parentCol {
    return false
  } else if colors[n] != parentCol {
    return true
  }
  for _, c := range graph[n] {
    if !dfs(c, graph, colors, colors[n]) {
      return false
    }
  }
  return true
}
```

786. K-th Smallest Prime Fraction

题目

A sorted list `A` contains 1, plus some number of primes. Then, for every p < q in the list, we consider the fraction p/q.

What is the `K`-th smallest fraction considered? Return your answer as an array of ints, where `answer[0] = p` and `answer[1] = q`.

Examples:

```
Input: A = [1, 2, 3, 5], K = 3
Output: [2, 5]
Explanation:
The fractions to be considered in sorted order are:
1/5, 1/3, 2/5, 1/2, 3/5, 2/3.
The third fraction is 2/5.

Input: A = [1, 7], K = 1
Output: [1, 7]
```

Note:

- `A` will have length between `2` and `2000`.
- Each `A[i]` will be between `1` and `30000`.
- `K` will be between `1` and `A.length * (A.length - 1) / 2`.

题目大意

一个已排序好的表 A，其包含 1 和其他一些素数. 当列表中的每一个 p<q 时，我们可以构造一个分数 p/q 。

那么第 k 个最小的分数是多少呢？以整数数组的形式返回你的答案, 这里 answer[0] = p 且 answer[1] = q.

注意:

- A 的取值范围在 2 — 2000.
- 每个 A[i] 的值在 1 —30000.
- K 取值范围为 1 — A.length * (A.length - 1) / 2

解题思路

- 给出一个从小到大排列的有序数组，数组里面的元素都是质数，请找出这个数组中的数组成的真分数从小到大排列，第 K 小的分数。
- 这一题的暴力解法是枚举所有可能的真分数，从小到大排序，输出第 K 小的分数即可。注意排序的时候不能直接用 float 排序，需要转化成分子和分母的结构体进行排序。
- 最优的解法是二分搜索。由于真分数都小于 1，所以二分搜索的范围是 [0,1]。每次二分出来的 mid，需要在数组里面搜索一次，找出比 mid 小的真分数个数。并记录下最大的真分数的分子和分母，动态维护最大真分数的分子和分母。如果比 mid 小的真分数个数小于 K，那么取右区间继续二分，如果比 mid 小的真分数个数大于 K，那么取左区间继续二分。直到正好找到比 mid 小的真分数个数是 K，此时维护的最大真分数的分子和分母即为答案。
- 在已排序的矩阵中寻找最 K 小的元素这一系列的题目有：第 373 题，第 378 题，第 668 题，第 719 题，第 786 题。

代码

```go
package leetcode

import (
  "sort"
)

// 解法一 二分搜索
func kthSmallestPrimeFraction(A []int, K int) []int {
  low, high, n := 0.0, 1.0, len(A)
  // 因为是在小数内使用二分查找，无法像在整数范围内那样通过 mid+1 和边界判断来终止循环
  // 所以此处根据 count 来结束循环
  for {
    mid, count, p, q, j := (high+low)/2.0, 0, 0, 1, 0
    for i := 0; i < n; i++ {
      for j < n && float64(A[i]) > float64(mid)*float64(A[j]) {
        j++
      }
      count += n - j
      if j < n && q*A[i] > p*A[j] {
        p = A[i]
```

```go
        q = A[j]
      }
    }
    if count == K {
      return []int{p, q}
    } else if count < K {
      low = mid
    } else {
      high = mid
    }
  }
}

// 解法二 暴力解法，时间复杂度 O(n^2)
func kthSmallestPrimeFraction1(A []int, K int) []int {
  if len(A) == 0 || (len(A)*(len(A)-1))/2 < K {
    return []int{}
  }
  fractions := []Fraction{}
  for i := 0; i < len(A); i++ {
    for j := i + 1; j < len(A); j++ {
      fractions = append(fractions, Fraction{molecule: A[i], denominator: A[j]})
    }
  }
  sort.Sort(SortByFraction(fractions))
  return []int{fractions[K-1].molecule, fractions[K-1].denominator}
}

// Fraction define
type Fraction struct {
  molecule    int
  denominator int
}

// SortByFraction define
type SortByFraction []Fraction

func (a SortByFraction) Len() int      { return len(a) }
func (a SortByFraction) Swap(i, j int) { a[i], a[j] = a[j], a[i] }
func (a SortByFraction) Less(i, j int) bool {
  return a[i].molecule*a[j].denominator < a[j].molecule*a[i].denominator
}
```

793. Preimage Size of Factorial Zeroes Function

题目

Let `f(x)` be the number of zeroes at the end of `x!`. (Recall that `x! = 1 * 2 * 3 * ... * x`, and by convention, `0! = 1`.)

For example, `f(3) = 0` because 3! = 6 has no zeroes at the end, while `f(11) = 2` because 11! = 39916800 has 2 zeroes at the end. Given `K`, find how many non-negative integers `x` have the property that `f(x) = K`.

Example 1:

```
Input: K = 0
Output: 5
Explanation: 0!, 1!, 2!, 3!, and 4! end with K = 0 zeroes.
```

Example 2:

```
Input: K = 5
Output: 0
Explanation: There is no x such that x! ends in K = 5 zeroes.
```

Note:

- `K` will be an integer in the range `[0, 10^9]`.

题目大意

f(x) 是 x! 末尾是0的数量。（回想一下 x! = 1 * 2 * 3 * ... * x，且0! = 1）

例如， f(3) = 0 ，因为3! = 6的末尾没有0；而 f(11) = 2 ，因为11!= 39916800末端有2个0。给定 K，找出多少个非负整数x ，有 f(x) = K 的性质。

注意：

- K 是范围在 [0, 10^9] 的整数。

解题思路

- 给出一个数 K，要求有多少个 n 能使得 n！末尾 0 的个数等于 K。
- 这一题是基于第 172 题的逆过程加强版。第 172 题是给出 `n`，求得末尾 0 的个数。由第 172 题可以知道，`n!` 末尾 0 的个数取决于因子 5 的个数。末尾可能有 `K` 个 0，那么 `n` 最多可以等于 `5 * K`，在 `[0, 5*K]` 区间内二分搜索，判断 `mid` 末尾 0 的个数，如果能找到 `K`，那么就范围 5，如果找不到这个 `K`，返回 0 。为什么答案取值只有 0 和 5 呢？因为当 `n` 增加 5 以后，因子 5 的个数又加一了，末尾又可以多 1 个或者多个 0(如果加 5 以后，有多个 5 的因子，例如 25，125，就有可能末尾增加多个 0)。所以有效的 `K` 值对应的 `n` 的范围区间就是 5 。反过来，无效的 `K` 值对应的 `n` 是 0。`K` 在 `5^n` 的分界线处会发生跳变，所有有些值取不到。例如，`n` 在 `[0,5)` 内取值，`K = 0`；`n` 在 `[5,10)` 内取值，`K = 1`；`n` 在 `[10,15)` 内取值，`K = 2`；`n` 在 `[15,20)` 内取值，`K = 3`；`n` 在 `[20,25)` 内取值，`K = 4`；`n` 在 `[25,30)` 内取值，`K = 6`，因为 25 提供了 2 个 5，也就提供了 2 个 0，所以 `K` 永远无法取值等于 5，即当 `K = 5` 时，找

不到任何的 `n` 与之对应。

- 这一题也可以用数学的方法解题。见解法二。这个解法的灵感来自于：n！末尾 0 的个数等于 [1,n] 所有数的因子 5 的个数总和。其次此题的结果一定只有 0 和 5 (分析见上一种解法)。有了这两个结论以后，就可以用数学的方法推导了。首先 n 可以表示为 5 进制的形式
  {{< katex display >}}
  n = 5^{0} * a{0} + 5^{1} * a{1} + 5^{2} * a{2} + ... + 5^{n} * a{n}, (a{n} < 5)
  {{< /katex >}}
  上面式子中，所有有因子 5 的个数为：
  {{< katex display >}}
  K = \sum{n=0}^{n} a{n} * c{n}
  {{< /katex >}}
  这个总数就即是 K。针对不同的 n，an 的通项公式不同，所以表示的 K 的系数也不同。cn 的通项公式呢?
  {{< katex display >}}
  c{n} = 5 * c{n-1} + 1，c{0} = 0
  {{< /katex >}}
  由上面这个递推还能推出通项公式(不过这题不适用通项公式，是用递推公式更方便)：
  {{< katex display >}}
  c{n} = \frac{5^{n} - 1 }{4}
  {{< /katex >}}
  判断 K 是否能表示成两个数列的表示形式，等价于判断 K 是否能转化为以 Cn 为基的变进制数。到此，转化成类似第 483 题了。代码实现不难，见解法二。

代码

```go
package leetcode

// 解法一 二分搜索
func preimageSizeFZF(K int) int {
  low, high := 0, 5*K
  for low <= high {
    mid := low + (high-low)>>1
    k := trailingZeroes(mid)
    if k == K {
      return 5
    } else if k > K {
      high = mid - 1
    } else {
      low = mid + 1
    }
  }
  return 0
}

// 解法二 数学方法
func preimageSizeFZF1(K int) int {
  base := 0
  for base < K {
```

```
    base = base*5 + 1
  }
  for K > 0 {
    base = (base - 1) / 5
    if K/base == 5 {
      return 0
    }
    K %= base
  }
  return 5
}
```

795. Number of Subarrays with Bounded Maximum

题目

We are given an array `nums` of positive integers, and two positive integers `left` and `right` (`left <= right`).

Return the number of (contiguous, non-empty) subarrays such that the value of the maximum array element in that subarray is at least `left` and at most `right`.

```
Example:Input:
nums = [2, 1, 4, 3]
left = 2
right = 3
Output: 3
Explanation: There are three subarrays that meet the requirements: [2], [2, 1], [3].
```

Note:

- `left`, `right`, and `nums[i]` will be an integer in the range `[0, 109]`.
- The length of `nums` will be in the range of `[1, 50000]`.

题目大意

给定一个元素都是正整数的数组`A`，正整数`L`以及`R`(`L <= R`)。求连续、非空且其中最大元素满足大于等于`L`小于等于`R`的子数组个数。

解题思路

- 题目要求子数组最大元素在 [L,R] 区间内。假设 count(bound) 为计算所有元素都小于等于 bound 的子数组数量。那么本题所求的答案可转化为 count(R) - count(L-1)。
- 如何统计所有元素小于 bound 的子数组数量呢？使用 count 变量记录在 bound 的左边，小于等于 bound 的连续元素数量。当找到一个这样的元素时，在此位置上结束的有效子数组的数量为 count + 1。当遇到一个元

素大于 B 时，则在此位置结束的有效子数组的数量为 0。res 将每轮 count 累加，最终 res 中存的即是满足条件的所有子数组数量。

代码

```
package leetcode

func numSubarrayBoundedMax(nums []int, left int, right int) int {
  return getAnswerPerBound(nums, right) - getAnswerPerBound(nums, left-1)
}

func getAnswerPerBound(nums []int, bound int) int {
  res, count := 0, 0
  for _, num := range nums {
    if num <= bound {
      count++
    } else {
      count = 0
    }
    res += count
  }
  return res
}
```

802. Find Eventual Safe States

题目

In a directed graph, we start at some node and every turn, walk along a directed edge of the graph. If we reach a node that is terminal (that is, it has no outgoing directed edges), we stop.

Now, say our starting node is *eventually safe* if and only if we must eventually walk to a terminal node. More specifically, there exists a natural number `K` so that for any choice of where to walk, we must have stopped at a terminal node in less than `K` steps.

Which nodes are eventually safe? Return them as an array in sorted order.

The directed graph has `N` nodes with labels `0, 1, ..., N-1`, where `N` is the length of `graph`. The graph is given in the following form: `graph[i]` is a list of labels `j` such that `(i, j)` is a directed edge of the graph.

```
Example:
Input: graph = [[1,2],[2,3],[5],[0],[5],[],[]]
Output: [2,4,5,6]
Here is a diagram of the above graph.
```

Note:

- `graph` will have length at most `10000`.
- The number of edges in the graph will not exceed `32000`.
- Each `graph[i]` will be a sorted list of different integers, chosen within the range `[0, graph.length - 1]`.

题目大意

在有向图中, 我们从某个节点和每个转向处开始, 沿着图的有向边走。 如果我们到达的节点是终点 (即它没有连出的有向边), 我们停止。现在, 如果我们最后能走到终点，那么我们的起始节点是最终安全的。 更具体地说, 存在一个自然数 K，无论选择从哪里开始行走, 我们走了不到 K 步后必能停止在一个终点。哪些节点最终是安全的？ 结果返回一个有序的数组。

提示：

- graph 节点数不超过 10000.
- 图的边数不会超过 32000.
- 每个 graph[i] 被排序为不同的整数列表， 在区间 [0, graph.length - 1] 中选取。

解题思路

- 给出一个有向图，要求找出所有“安全”节点。“安全”节点的定义是：存在一个自然数 K, 无论选择从哪里开始行走, 我们走了不到 K 步后必能停止在一个终点。
- 这一题可以用拓扑排序，也可以用 DFS 染色来解答。这里用 DFS 来解答。对于每个节点，我们有 3 种染色的方法：白色 0 号节点表示该节点还没有被访问过；灰色 1 号节点表示该节点在栈中（这一轮搜索中被访问过）或者在环中；黑色 2 号节点表示该节点的所有相连的节点都被访问过，且该节点不在环中。当我们第一次访问一个节点时，我们把它从白色变成灰色，并继续搜索与它相连的节点。如果在搜索过程中我们遇到一个灰色的节点，那么说明找到了一个环，此时退出搜索，所有的灰色节点保持不变（即从任意一个灰色节点开始，都能走到环中），如果搜索过程中，我们没有遇到灰色的节点，那么在回溯到当前节点时，我们把它从灰色变成黑色，即表示它是一个安全的节点。

代码

```go
func eventualSafeNodes(graph [][]int) []int {
  res, color := []int{}, make([]int, len(graph))
  for i := range graph {
    if dfsEventualSafeNodes(graph, i, color) {
      res = append(res, i)
    }
  }
  return res
}

// colors: WHITE 0, GRAY 1, BLACK 2;
func dfsEventualSafeNodes(graph [][]int, idx int, color []int) bool {
  if color[idx] > 0 {
    return color[idx] == 2
  }
```

```
  color[idx] = 1
  for i := range graph[idx] {
    if !dfsEventualSafeNodes(graph, graph[idx][i], color) {
      return false
    }
  }
  color[idx] = 2
  return true
}
```

803. Bricks Falling When Hit

题目

We have a grid of 1s and 0s; the 1s in a cell represent bricks. A brick will not drop if and only if it is directly connected to the top of the grid, or at least one of its (4-way) adjacent bricks will not drop.

We will do some erasures sequentially. Each time we want to do the erasure at the location (i, j), the brick (if it exists) on that location will disappear, and then some other bricks may drop because of that erasure.

Return an array representing the number of bricks that will drop after each erasure in sequence.

Example 1:

```
Input:
grid = [[1,0,0,0],[1,1,1,0]]
hits = [[1,0]]
Output: [2]
Explanation:
If we erase the brick at (1, 0), the brick at (1, 1) and (1, 2) will drop. So we should
return 2.
```

Example 2:

```
Input:
grid = [[1,0,0,0],[1,1,0,0]]
hits = [[1,1],[1,0]]
Output: [0,0]
Explanation:
When we erase the brick at (1, 0), the brick at (1, 1) has already disappeared due to
the last move. So each erasure will cause no bricks dropping.  Note that the erased
brick (1, 0) will not be counted as a dropped brick.
```

Note:

- The number of rows and columns in the grid will be in the range [1, 200].
- The number of erasures will not exceed the area of the grid.
- It is guaranteed that each erasure will be different from any other erasure, and located inside the grid.
- An erasure may refer to a location with no brick - if it does, no bricks drop.

题目大意

我们有一组包含1和0的网格；其中1表示砖块。 当且仅当一块砖直接连接到网格的顶部，或者它至少有一块相邻（4 个方向之一）砖块不会掉落时，它才不会落下。我们会依次消除一些砖块。每当我们消除 (i, j) 位置时， 对应位置的砖块（若存在）会消失，然后其他的砖块可能因为这个消除而落下。返回一个数组表示每次消除操作对应落下的砖块数目。

注意:

- 网格的行数和列数的范围是[1, 200]。
- 消除的数字不会超过网格的区域。
- 可以保证每次的消除都不相同，并且位于网格的内部。
- 一个消除的位置可能没有砖块，如果这样的话，就不会有砖块落下。

解题思路

- 有一些砖块连接在天花板上，问，如果打掉某个砖块，会掉落几块砖块？打掉的每个砖块不参与计数。
- 这一题可以用并查集和 DFS 求解。不过尝试用 DFS 的同学就会知道，这一题卡时间卡的很紧。用 DFS 虽然能 AC，但是耗时非常长。用并查集也必须进行秩压缩，不然耗时也非常长。另外，如果用了并查集，每个集合的总数单独统计，不随着 union() 操作，也会导致超时，笔者在这里被 LTE 了多次，最后只能重写 UnionFind 并查集类，将统计操作和 union() 操作写在一起，这一题才 faster than 100.00% AC。
- 拿到题以后，首先尝试暴力解法，按照顺序打掉砖块，每次打掉砖块以后，都重建并查集。题目要求每次掉落几块砖块，实际上比较每次和天花板连通的砖块个数变化了多少块就可以了。那么解法就出来了，先把和天花板连通的砖块都 union() 起来，记录这个集合中砖块的个数 `count`，然后每次打掉一个砖块以后，重建并查集，计算与天花板连通的砖块的个数 `newCount`，`newCount - count -1` 就是最终答案(打掉的那块砖块不计算其中)，提交代码以后，发现 TLE。
- 出现 TLE 以后一般思路都是对的，只是时间复杂度过高，需要优化。很明显，需要优化的地方是每次都重建了新的并查集，有没有办法能在上一次状态上进行变更，不用重建并查集呢？如果正向的打掉砖块，那么每次还需要以这个砖块为起点进行 DFS，时间复杂度还是很高。如果反向考虑呢？先把所有要打掉的砖块都打掉，构建打掉这些砖块以后剩下与天花板连通的并查集。然后反向添加打掉的砖块，每次添加一块就刷新一次它周围的 4 个砖块，不用 DFS，这样时间复杂度优化了很多。最后在按照 `newCount - count -1` 方式计算最终答案。注意每次还原一个砖块的时候需要染色回原有砖块的颜色 `1` 。优化成这样的做法，基本不会 TLE 了，如果计算 count 是单独计算的，还是会 TLE。如果没有进行秩压缩，时间会超过 1500 ms，所以这一题想拿到 100%，每步优化都要做好。最终 100% 的答案见代码。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/template"
)

func hitBricks(grid [][]int, hits [][]int) []int {
  if len(hits) == 0 {
```

```go
    return []int{}
  }
  uf, m, n, res, oriCount := template.UnionFindCount{}, len(grid), len(grid[0]),
make([]int, len(hits)), 0
  uf.Init(m*n + 1)
  // 先将要打掉的砖块染色
  for _, hit := range hits {
    if grid[hit[0]][hit[1]] == 1 {
      grid[hit[0]][hit[1]] = 2
    }
  }
  for i := 0; i < m; i++ {
    for j := 0; j < n; j++ {
      if grid[i][j] == 1 {
        getUnionFindFromGrid(grid, i, j, uf)
      }
    }
  }
  oriCount = uf.Count()[uf.Find(m*n)]
  for i := len(hits) - 1; i >= 0; i-- {
    if grid[hits[i][0]][hits[i][1]] == 2 {
      grid[hits[i][0]][hits[i][1]] = 1
      getUnionFindFromGrid(grid, hits[i][0], hits[i][1], uf)
    }
    nowCount := uf.Count()[uf.Find(m*n)]
    if nowCount-oriCount > 0 {
      res[i] = nowCount - oriCount - 1
    } else {
      res[i] = 0
    }
    oriCount = nowCount
  }
  return res
}

func isInGrid(grid [][]int, x, y int) bool {
  return x >= 0 && x < len(grid) && y >= 0 && y < len(grid[0])
}

func getUnionFindFromGrid(grid [][]int, x, y int, uf template.UnionFindCount) {
  m, n := len(grid), len(grid[0])
  if x == 0 {
    uf.Union(m*n, x*n+y)
  }
  for i := 0; i < 4; i++ {
    nx := x + dir[i][0]
    ny := y + dir[i][1]
    if isInGrid(grid, nx, ny) && grid[nx][ny] == 1 {
      uf.Union(nx*n+ny, x*n+y)
```

```
    }
  }
}
```

810. Chalkboard XOR Game

题目

We are given non-negative integers nums[i] which are written on a chalkboard. Alice and Bob take turns erasing exactly one number from the chalkboard, with Alice starting first. If erasing a number causes the bitwise XOR of all the elements of the chalkboard to become 0, then that player loses. (Also, we'll say the bitwise XOR of one element is that element itself, and the bitwise XOR of no elements is 0.)

Also, if any player starts their turn with the bitwise XOR of all the elements of the chalkboard equal to 0, then that player wins.

Return True if and only if Alice wins the game, assuming both players play optimally.

```
Example:Input: nums = [1, 1, 2]
Output: false
Explanation:
Alice has two choices: erase 1 or erase 2.
If she erases 1, the nums array becomes [1, 2]. The bitwise XOR of all the elements of
the chalkboard is 1 XOR 2 = 3. Now Bob can remove any element he wants, because Alice
will be the one to erase the last element and she will lose.
If Alice erases 2 first, now nums becomes [1, 1]. The bitwise XOR of all the elements
of the chalkboard is 1 XOR 1 = 0. Alice will lose.
```

Notes:

- `1 <= N <= 1000`.
- `0 <= nums[i] <= 2^16`.

题目大意

黑板上写着一个非负整数数组 nums[i] 。Alice 和 Bob 轮流从黑板上擦掉一个数字，Alice 先手。如果擦除一个数字后，剩余的所有数字按位异或运算得出的结果等于 0 的话，当前玩家游戏失败。 (另外，如果只剩一个数字，按位异或运算得到它本身；如果无数字剩余，按位异或运算结果为 0。）并且，轮到某个玩家时，如果当前黑板上所有数字按位异或运算结果等于 0，这个玩家获胜。假设两个玩家每步都使用最优解，当且仅当 Alice 获胜时返回 true。

解题思路

- Alice 必胜情况之一，Alice 先手，起始数组全部元素本身异或结果就为 0 。不需要擦除数字便自动获胜。除去这个情况，还有其他情况么？由于 2 人是交替擦除数字，且每次都恰好擦掉一个数字，因此对于这两人中的任意一人，其每次在擦除数字前，黑板上剩余数字的个数的奇偶性一定都是相同的。于是奇偶性成为突破口。

- 如果 nums 的长度是偶数，Alice 先手是否必败呢？如果必败，代表无论擦掉哪一个数字，剩余所有数字的异或结果都等于 0。利用反证法证明上述结论是错误的。首先 {{< katex >}} num[0] \oplus num[1] \oplus num[2] \oplus \cdots \oplus num[n-1] = X ≠ 0 {{< /katex >}} ，初始所有元素异或结果不为 0。假设 Alice 当前擦掉第 i 个元素，0 ≤ i < n。令 {{< katex >}}X{n}{{< /katex >}} *代表擦掉第 n 位元素以后剩余元素异或的结果。由证题，无论擦掉哪一个数字，剩余所有数字的异或结果都等于 0。所以 {{< katex >}} X{0} \oplus X{1} \oplus X{2} \oplus \cdots \oplus X_{n-1} = 0{{< /katex >}}* 。

  {{< katex display >}}
  \begin{aligned}0 &= X{0} \oplus X{1} \oplus X{2} \oplus \cdots \oplus X{n-1} \\0 &= (X \oplus nums[0]) \oplus (X \oplus nums[1]) \oplus (X \oplus nums[2]) \oplus \cdots \oplus (X \oplus nums[n-1])\\ 0 &= (X \oplus X \oplus \cdots \oplus X) \oplus (nums[0] \oplus nums[1] \oplus nums[2] \oplus \cdots \oplus nums[n-1])\\0 &= 0 \oplus X\\\Rightarrow X &= 0\\\end{aligned}
  {{< /katex >}}

  由于 n 为偶数，所以 n 个 X 的异或结果为 0。最终推出 X = 0，很明显与前提 X ≠ 0 冲突。所以原命题，代表无论擦掉哪一个数字，剩余所有数字的异或结果都等于 0 是错误的。也就是说，当 n 为偶数时，代表无论擦掉哪一个数字，剩余所有数字的异或结果都不等于 0。即 Alice 有必胜策略。换句话说，当数组的长度是偶数时，先手 Alice 总能找到一个数字，在擦掉这个数字之后剩余的所有数字异或结果不等于 0。
- 综上，Alice 必胜策略有 2 种情况：
  1. 数组 nums 的全部元素初始本身异或结果就等于 0。
  2. 数组 nums 的长度是偶数。

代码

```go
package leetcode

func xorGame(nums []int) bool {
  if len(nums)%2 == 0 {
    return true
  }
  xor := 0
  for _, num := range nums {
    xor ^= num
  }
  return xor == 0
}
```

811. Subdomain Visit Count

题目

A website domain like "discuss.leetcode.com" consists of various subdomains. At the top level, we have "com", at the next level, we have "leetcode.com", and at the lowest level, "discuss.leetcode.com". When we visit a domain like "discuss.leetcode.com", we will also visit the parent domains "leetcode.com" and "com" implicitly.

Now, call a "count-paired domain" to be a count (representing the number of visits this domain received), followed by a space, followed by the address. An example of a count-paired domain might be "9001 discuss.leetcode.com".

We are given a list `cpdomains` of count-paired domains. We would like a list of count-paired domains, (in the same format as the input, and in any order), that explicitly counts the number of visits to each subdomain.

Example 1:

```
Input:
["9001 discuss.leetcode.com"]
Output:
["9001 discuss.leetcode.com", "9001 leetcode.com", "9001 com"]
Explanation:
We only have one website domain: "discuss.leetcode.com". As discussed above, the
subdomain "leetcode.com" and "com" will also be visited. So they will all be visited
9001 times.
```

Example 2:

```
Input:
["900 google.mail.com", "50 yahoo.com", "1 intel.mail.com", "5 wiki.org"]
Output:
["901 mail.com","50 yahoo.com","900 google.mail.com","5 wiki.org","5 org","1
intel.mail.com","951 com"]
Explanation:
We will visit "google.mail.com" 900 times, "yahoo.com" 50 times, "intel.mail.com" once
and "wiki.org" 5 times. For the subdomains, we will visit "mail.com" 900 + 1 = 901
times, "com" 900 + 50 + 1 = 951 times, and "org" 5 times.
```

Notes:

- The length of `cpdomains` will not exceed `100`.
- The length of each domain name will not exceed `100`.
- Each address will have either 1 or 2 "." characters.
- The input count in any count-paired domain will not exceed `10000`.
- The answer output can be returned in any order.

题目大意

一个网站域名，如 "discuss.leetcode.com"，包含了多个子域名。作为顶级域名，常用的有 "com"，下一级则有 "leetcode.com"，最低的一级为 "discuss.leetcode.com"。当我们访问域名 "discuss.leetcode.com" 时，也同时访问了其父域名 "leetcode.com" 以及顶级域名 "com"。给定一个带访问次数和域名的组合，要求分别计算每个域名被访问的次数。其格式为访问次数+空格+地址，例如："9001 discuss.leetcode.com"。

接下来会给出一组访问次数和域名组合的列表 cpdomains 。要求解析出所有域名的访问次数，输出格式和输入格式相同，不限定先后顺序。

解题思路

- 这一题是简单题，统计每个 domain 的出现频次。每个域名根据层级，一级一级的累加频次，比如 `discuss.leetcode.com`、`discuss.leetcode.com` 这个域名频次为 1，`leetcode.com` 这个域名频次为 1，`com` 这个域名频次为 1。用 map 依次统计每个 domain 出现的频次，按照格式要求输出。

代码

```go
package leetcode

import (
  "strconv"
  "strings"
)

// 解法一
func subdomainVisits(cpdomains []string) []string {
  result := make([]string, 0)
  if len(cpdomains) == 0 {
    return result
  }
  domainCountMap := make(map[string]int, 0)
  for _, domain := range cpdomains {
    countDomain := strings.Split(domain, " ")
    allDomains := strings.Split(countDomain[1], ".")
    temp := make([]string, 0)
    for i := len(allDomains) - 1; i >= 0; i-- {
      temp = append([]string{allDomains[i]}, temp...)
      ld := strings.Join(temp, ".")
      count, _ := strconv.Atoi(countDomain[0])
      if val, ok := domainCountMap[ld]; !ok {
        domainCountMap[ld] = count
      } else {
        domainCountMap[ld] = count + val
      }
    }
  }
  for k, v := range domainCountMap {
    t := strings.Join([]string{strconv.Itoa(v), k}, " ")
    result = append(result, t)
  }
  return result
}

// 解法二
func subdomainVisits1(cpdomains []string) []string {
```

```go
  out := make([]string, 0)
  var b strings.Builder
  domains := make(map[string]int, 0)
  for _, v := range cpdomains {
    splitDomain(v, domains)
  }
  for k, v := range domains {
    b.WriteString(strconv.Itoa(v))
    b.WriteString(" ")
    b.WriteString(k)
    out = append(out, b.String())
    b.Reset()
  }
  return out
}

func splitDomain(domain string, domains map[string]int) {
  visits := 0
  var e error
  subdomains := make([]string, 0)
  for i, v := range domain {
    if v == ' ' {
      visits, e = strconv.Atoi(domain[0:i])
      if e != nil {
        panic(e)
      }
      break
    }
  }
  for i := len(domain) - 1; i >= 0; i-- {
    if domain[i] == '.' {
      subdomains = append(subdomains, domain[i+1:])
    } else if domain[i] == ' ' {
      subdomains = append(subdomains, domain[i+1:])
      break
    }
  }
  for _, v := range subdomains {
    count, ok := domains[v]
    if ok {
      domains[v] = count + visits
    } else {
      domains[v] = visits
    }
  }
}
```

812. Largest Triangle Area

题目

You have a list of points in the plane. Return the area of the largest triangle that can be formed by any 3 of the points.

```
Example:
Input: points = [[0,0],[0,1],[1,0],[0,2],[2,0]]
Output: 2
Explanation:
The five points are show in the figure below. The red triangle is the largest.
```

Notes:

- `3 <= points.length <= 50`.
- No points will be duplicated.
- `-50 <= points[i][j] <= 50`.
- Answers within `10^-6` of the true value will be accepted as correct.

题目大意

给定包含多个点的集合，从其中取三个点组成三角形，返回能组成的最大三角形的面积。

解题思路

- 给出一组点的坐标，要求找出能组成三角形面积最大的点集合，输出这个最大面积。
- 数学题。按照数学定义，分别计算这些能构成三角形的点形成的三角形面积，最终输出最大面积即可。

代码

```go
package leetcode

func largestTriangleArea(points [][]int) float64 {
  maxArea, n := 0.0, len(points)
  for i := 0; i < n; i++ {
    for j := i + 1; j < n; j++ {
      for k := j + 1; k < n; k++ {
        maxArea = max(maxArea, area(points[i], points[j], points[k]))
      }
    }
  }
  return maxArea
}

func area(p1, p2, p3 []int) float64 {
```

```
  return abs(p1[0]*p2[1]+p2[0]*p3[1]+p3[0]*p1[1]-p1[0]*p3[1]-p2[0]*p1[1]-p3[0]*p2[1]) /
2
}

func abs(num int) float64 {
  if num < 0 {
    num = -num
  }
  return float64(num)
}

func max(a, b float64) float64 {
  if a > b {
    return a
  }
  return b
}
```

815. Bus Routes

题目

We have a list of bus routes. Each `routes[i]` is a bus route that the i-th bus repeats forever. For example if `routes[0] = [1, 5, 7]`, this means that the first bus (0-th indexed) travels in the sequence 1->5->7->1->5->7->1->... forever.

We start at bus stop `S` (initially not on a bus), and we want to go to bus stop `T`. Travelling by buses only, what is the least number of buses we must take to reach our destination? Return -1 if it is not possible.

Example:

```
Input:
routes = [[1, 2, 7], [3, 6, 7]]
S = 1
T = 6
Output: 2
Explanation:
The best strategy is take the first bus to the bus stop 7, then take the second bus to
the bus stop 6.
```

Note:

- `1 <= routes.length <= 500`.
- `1 <= routes[i].length <= 500`.
- `0 <= routes[i][j] < 10 ^ 6`.

题目大意

我们有一系列公交路线。每一条路线 routes[i] 上都有一辆公交车在上面循环行驶。例如，有一条路线 routes[0] = [1, 5, 7]，表示第一辆 (下标为0) 公交车会一直按照 1->5->7->1->5->7->1->... 的车站路线行驶。假设我们从 S 车站开始（初始时不在公交车上），要去往 T 站。 期间仅可乘坐公交车，求出最少乘坐的公交车数量。返回 -1 表示不可能到达终点车站。

说明:

- 1 <= routes.length <= 500.
- 1 <= routes[i].length <= 500.
- 0 <= routes[i][j] < 10 ^ 6.

解题思路

- 给出一些公交路线，公交路径代表经过的哪些站。现在给出起点和终点站，问最少需要换多少辆公交车才能从起点到终点?
- 这一题可以转换成图论的问题，将每个站台看成顶点，公交路径看成每个顶点的边。同一个公交的边染色相同。题目即可转化为从顶点 S 到顶点 T 需要经过最少多少条不同的染色边。用 BFS 即可轻松解决。从起点 S 开始，不断的扩展它能到达的站点。用 visited 数组防止放入已经可达的站点引起的环。用 map 存储站点和公交车的映射关系(即某个站点可以由哪些公交车到达)，BFS 的过程中可以用这个映射关系，拿到公交车的其他站点信息，从而扩张队列里面的可达站点。一旦扩展出现了终点 T，就可以返回结果了。

代码

```go
package leetcode

func numBusesToDestination(routes [][]int, S int, T int) int {
  if S == T {
    return 0
  }
  // vertexMap 中 key 是站点，value 是公交车数组，代表这些公交车路线可以到达此站点
  vertexMap, visited, queue, res := map[int][]int{}, make([]bool, len(routes)),
[]int{}, 0
  for i := 0; i < len(routes); i++ {
    for _, v := range routes[i] {
      tmp := vertexMap[v]
      tmp = append(tmp, i)
      vertexMap[v] = tmp
    }
  }
  queue = append(queue, S)
  for len(queue) > 0 {
    res++
    qlen := len(queue)
    for i := 0; i < qlen; i++ {
      vertex := queue[0]
      queue = queue[1:]
      for _, bus := range vertexMap[vertex] {
```

```
        if visited[bus] == true {
          continue
        }
        visited[bus] = true
        for _, v := range routes[bus] {
          if v == T {
            return res
          }
          queue = append(queue, v)
        }
      }
    }
  }
  return -1
}
```

816. Ambiguous Coordinates

题目

We had some 2-dimensional coordinates, like `"(1, 3)"` or `"(2, 0.5)"`. Then, we removed all commas, decimal points, and spaces, and ended up with the string `s`. Return a list of strings representing all possibilities for what our original coordinates could have been.

Our original representation never had extraneous zeroes, so we never started with numbers like "00", "0.0", "0.00", "1.0", "001", "00.01", or any other number that can be represented with less digits. Also, a decimal point within a number never occurs without at least one digit occuring before it, so we never started with numbers like ".1".

The final answer list can be returned in any order. Also note that all coordinates in the final answer have exactly one space between them (occurring after the comma.)

```
Example 1:Input: s = "(123)"
Output: ["(1, 23)", "(12, 3)", "(1.2, 3)", "(1, 2.3)"]
```

```
Example 2:Input: s = "(00011)"
Output:  ["(0.001, 1)", "(0, 0.011)"]
Explanation:
0.0, 00, 0001 or 00.01 are not allowed.
```

```
Example 3:Input: s = "(0123)"
Output: ["(0, 123)", "(0, 12.3)", "(0, 1.23)", "(0.1, 23)", "(0.1, 2.3)", "(0.12, 3)"]
```

```
Example 4:Input: s = "(100)"
Output: [(10, 0)]
Explanation:
1.0 is not allowed.
```

Note:

- `4 <= s.length <= 12`.
- `s[0]` = "(", `s[s.length - 1]` = ")", and the other elements in `s` are digits.

题目大意

我们有一些二维坐标，如 "(1, 3)" 或 "(2, 0.5)"，然后我们移除所有逗号，小数点和空格，得到一个字符串S。返回所有可能的原始字符串到一个列表中。原始的坐标表示法不会存在多余的零，所以不会出现类似于"00", "0.0", "0.00", "1.0", "001", "00.01"或一些其他更小的数来表示坐标。此外，一个小数点前至少存在一个数，所以也不会出现“.1”形式的数字。

最后返回的列表可以是任意顺序的。而且注意返回的两个数字中间（逗号之后）都有一个空格。

解题思路

- 本题没有什么算法思想，纯暴力题。先将原始字符串一分为二，分为的两个子字符串再移动坐标点，最后将每种情况组合再一次，这算完成了一次切分。将原始字符串每一位都按此规律完成切分，此题便得解。
- 这道题有 2 处需要注意的。第一处是最终输出的字符串，请注意，**两个数字中间（逗号之后）都有一个空格**。不遵守输出格式的要求也会导致 `Wrong Answer`。另外一处是切分数字时，有 2 种违法情况，一种是带前导 0 的，另外一种是末尾带 0 的。带前导 0 的也分为 2 种情况，一种是只有一位，即只有一个 0，这种情况直接返回，因为这一个 0 怎么切分也只有一种切分方法。另外一种是长度大于 1，即 `0xxx` 这种情况。`0xxx` 这种情况只有一种切分方法，即 `0.xxx`。末尾带 0 的只有一种切分方法，即 `xxx0`，不可切分，因为 `xxx.0`，`xx.x0`，`x.xx0` 这些都是违法情况，所以末尾带 0 的也可以直接返回。具体的实现见代码和注释。

代码

```go
package leetcode

func ambiguousCoordinates(s string) []string {
  res := []string{}
  s = s[1 : len(s)-1]
  for i := range s[:len(s)-1] {
    a := build(s[:i+1])
    b := build(s[i+1:])
    for _, ta := range a {
      for _, tb := range b {
        res = append(res, "("+ta+", "+tb+")")
      }
    }
  }
```

```go
  return res
}

func build(s string) []string {
  res := []string{}
  if len(s) == 1 || s[0] != '0' {
    res = append(res, s)
  }
  // 结尾带 0 的情况
  if s[len(s)-1] == '0' {
    return res
  }
  // 切分长度大于一位且带前导 0 的情况
  if s[0] == '0' {
    res = append(res, "0."+s[1:])
    return res
  }
  for i := range s[:len(s)-1] {
    res = append(res, s[:i+1]+"."+s[i+1:])
  }
  return res
}
```

817. Linked List Components

题目

We are given head, the head node of a linked list containing unique integer values.

We are also given the list G, a subset of the values in the linked list.

Return the number of connected components in G, where two values are connected if they appear consecutively in the linked list.

Example 1:

```
Input:
head: 0->1->2->3
G = [0, 1, 3]
Output: 2
Explanation:
0 and 1 are connected, so [0, 1] and [3] are the two connected components.
```

Example 2:

```
Input:
head: 0->1->2->3->4
G = [0, 3, 1, 4]
Output: 2
Explanation:
0 and 1 are connected, 3 and 4 are connected, so [0, 1] and [3, 4] are the two
connected components.
```

Note:

- If N is the length of the linked list given by head, 1 <= N <= 10000.
- The value of each node in the linked list will be in the range [0, N - 1].
- 1 <= G.length <= 10000.
- G is a subset of all values in the linked list.

题目大意

这道题题目的意思描述的不是很明白，我提交了几次 WA 以后才悟懂题意。

这道题的意思是，在 G 中能组成多少组子链表，这些子链表的要求是能在原链表中是有序的。

解题思路

这个问题再抽象一下就成为这样：在原链表中去掉 G 中不存在的数，会被切断成几段链表。例如，将原链表中 G 中存在的数标为 0，不存在的数标为 1 。原链表标识为 0-0-0-1-0-1-1-0-0-1-0-1，那么这样原链表被断成了 4 段。只要在链表中找 0-1 组合就可以认为是一段，因为这里必定会有一段生成。

考虑末尾的情况，0-1，1-0，0-0，1-1，这 4 种情况的特征都是，末尾一位只要是 0，都会新产生一段。所以链表末尾再单独判断一次，是 0 就再加一。

代码

```go
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
```

```go
 */

func numComponents(head *ListNode, G []int) int {
  if head.Next == nil {
    return 1
  }
  gMap := toMap(G)
  count := 0
  cur := head

  for cur != nil {
    if _, ok := gMap[cur.Val]; ok {
      if cur.Next == nil { // 末尾存在，直接加一
        count++
      } else {
        if _, ok = gMap[cur.Next.Val]; !ok {
          count++
        }
      }
    }
    cur = cur.Next
  }
  return count
}

func toMap(G []int) map[int]int {
  GMap := make(map[int]int, 0)
  for _, value := range G {
    GMap[value] = 0
  }
  return GMap
}
```

819. Most Common Word

题目

Given a paragraph and a list of banned words, return the most frequent word that is not in the list of banned words. It is guaranteed there is at least one word that isn't banned, and that the answer is unique.

Words in the list of banned words are given in lowercase, and free of punctuation. Words in the paragraph are not case sensitive. The answer is in lowercase.

Example:

```
Input:
paragraph = "Bob hit a ball, the hit BALL flew far after it was hit."
banned = ["hit"]
Output: "ball"
Explanation:
"hit" occurs 3 times, but it is a banned word.
"ball" occurs twice (and no other word does), so it is the most frequent non-banned
word in the paragraph.
Note that words in the paragraph are not case sensitive,
that punctuation is ignored (even if adjacent to words, such as "ball,"),
and that "hit" isn't the answer even though it occurs more because it is banned.
```

Note:

- `1 <= paragraph.length <= 1000`.
- `0 <= banned.length <= 100`.
- `1 <= banned[i].length <= 10`.
- The answer is unique, and written in lowercase (even if its occurrences in `paragraph` may have uppercase symbols, and even if it is a proper noun.)
- `paragraph` only consists of letters, spaces, or the punctuation symbols `!?',;.`
- There are no hyphens or hyphenated words.
- Words only consist of letters, never apostrophes or other punctuation symbols.

题目大意

给定一个段落 (paragraph) 和一个禁用单词列表 (banned)。返回出现次数最多，同时不在禁用列表中的单词。题目保证至少有一个词不在禁用列表中，而且答案唯一。

禁用列表中的单词用小写字母表示，不含标点符号。段落中的单词不区分大小写。答案都是小写字母。

解题思路

- 给出一段话和一个 banned 的字符串，要求输出这段话中出现频次最高的并且不出现在 banned 数组里面的字符串，答案唯一。这题是简单题，依次统计每个单词的频次，然后从 map 中删除 banned 里面的单词，取出剩下频次最高的单词即可。

代码

```go
package leetcode

import "strings"

func mostCommonWord(paragraph string, banned []string) string {
  freqMap, start := make(map[string]int), -1
  for i, c := range paragraph {
    if c == ' ' || c == '!' || c == '?' || c == '\'' || c == ',' || c == ';' || c == 
'.' {
```

```
        if start > -1 {
          word := strings.ToLower(paragraph[start:i])
          freqMap[word]++
        }
        start = -1
      } else {
        if start == -1 {
          start = i
        }
      }
    }
    if start != -1 {
      word := strings.ToLower(paragraph[start:])
      freqMap[word]++
    }
    // Strip the banned words from the freqmap
    for _, bannedWord := range banned {
      delete(freqMap, bannedWord)
    }
    // Find most freq word
    mostFreqWord, mostFreqCount := "", 0
    for word, freq := range freqMap {
      if freq > mostFreqCount {
        mostFreqWord = word
        mostFreqCount = freq
      }
    }
    return mostFreqWord
  }
```

820. Short Encoding of Words

题目

A **valid encoding** of an array of `words` is any reference string `s` and array of indices `indices` such that:

- `words.length == indices.length`
- The reference string `s` ends with the `'#'` character.
- For each index `indices[i]`, the **substring** of `s` starting from `indices[i]` and up to (but not including) the next `'#'` character is equal to `words[i]`.

Given an array of `words`, return *the **length of the shortest reference string** `s` possible of any **valid encoding** of `words`.*

Example 1:

```
Input: words = ["time", "me", "bell"]
Output: 10
Explanation: A valid encoding would be s = "time#bell#" and indices = [0, 2, 5].
words[0] = "time", the substring of s starting from indices[0] = 0 to the next '#' is
underlined in "time#bell#"
words[1] = "me", the substring of s starting from indices[1] = 2 to the next '#' is
underlined in "time#bell#"
words[2] = "bell", the substring of s starting from indices[2] = 5 to the next '#' is
underlined in "time#bell#"
```

Example 2:

```
Input: words = ["t"]
Output: 2
Explanation: A valid encoding would be s = "t#" and indices = [0].
```

Constraints:

- `1 <= words.length <= 2000`
- `1 <= words[i].length <= 7`
- `words[i]` consists of only lowercase letters.

题目大意

单词数组 words 的 有效编码 由任意助记字符串 s 和下标数组 indices 组成，且满足：

- words.length == indices.length
- 助记字符串 s 以 '#' 字符结尾
- 对于每个下标 indices[i] ，s 的一个从 indices[i] 开始、到下一个 '#' 字符结束（但不包括 '#'）的 子字符串 恰好与 words[i] 相等

给你一个单词数组 words ，返回成功对 words 进行编码的最小助记字符串 s 的长度 。

解题思路

- 暴力解法。先将所有的单词放入字典中。然后针对字典中的每个单词，逐一从字典中删掉自己的子字符串，这样有相同后缀的字符串被删除了，字典中剩下的都是没有共同前缀的。最终的答案是剩下所有单词用 # 号连接之后的总长度。
- Trie 解法。构建 Trie 树，相同的后缀会被放到从根到叶子节点中的某个路径中。最后依次遍历一遍所有单词，如果单词最后一个字母是叶子节点，说明这个单词是要选择的，因为它可能是包含了一些单词后缀的最长单词。累加这个单词的长度并再加 1(# 字符的长度)。最终累加出来的长度即为题目所求的答案。

代码

```go
package leetcode

// 解法一 暴力
func minimumLengthEncoding(words []string) int {
```

```go
  res, m := 0, map[string]bool{}
  for _, w := range words {
    m[w] = true
  }
  for w := range m {
    for i := 1; i < len(w); i++ {
      delete(m, w[i:])
    }
  }
  for w := range m {
    res += len(w) + 1
  }
  return res
}

// 解法二 Trie
type node struct {
  value byte
  sub   []*node
}

func (t *node) has(b byte) (*node, bool) {
  if t == nil {
    return nil, false
  }
  for i := range t.sub {
    if t.sub[i] != nil && t.sub[i].value == b {
      return t.sub[i], true
    }
  }
  return nil, false
}

func (t *node) isLeaf() bool {
  if t == nil {
    return false
  }
  return len(t.sub) == 0
}

func (t *node) add(s []byte) {
  now := t
  for i := len(s) - 1; i > -1; i-- {
    if v, ok := now.has(s[i]); ok {
      now = v
      continue
    }
    temp := new(node)
    temp.value = s[i]
```

```go
    now.sub = append(now.sub, temp)
    now = temp
  }
}

func (t *node) endNodeOf(s []byte) *node {
  now := t
  for i := len(s) - 1; i > -1; i-- {
    if v, ok := now.has(s[i]); ok {
      now = v
      continue
    }
    return nil
  }
  return now
}

func minimumLengthEncoding1(words []string) int {
  res, tree, m := 0, new(node), make(map[string]bool)
  for i := range words {
    if !m[words[i]] {
      tree.add([]byte(words[i]))
      m[words[i]] = true
    }
  }
  for s := range m {
    if tree.endNodeOf([]byte(s)).isLeaf() {
      res += len(s)
      res++
    }
  }
  return res
}
```

821. Shortest Distance to a Character

题目

Given a string `s` and a character `c` that occurs in `s`, return *an array of integers* `answer` *where* `answer.length == s.length` *and* `answer[i]` *is the shortest distance from* `s[i]` *to the character* `c` *in* `s`.

Example 1:

```
Input: s = "loveleetcode", c = "e"
Output: [3,2,1,0,1,0,0,1,2,2,1,0]
```

Example 2:

```
Input: s = "aaab", c = "b"
Output: [3,2,1,0]
```

Constraints:

- `1 <= s.length <= 104`
- `s[i]` and `c` are lowercase English letters.
- `c` occurs at least once in `s`.

题目大意

给定一个字符串 S 和一个字符 C。返回一个代表字符串 S 中每个字符到字符串 S 中的字符 C 的最短距离的数组。

解题思路

- 解法一：从左至右更新一遍到 C 的值距离，再从右至左更新一遍到 C 的值，取两者中的最小值。
- 解法二：依次扫描字符串 S，针对每一个非字符 C 的字符，分别往左扫一次，往右扫一次，计算出距离目标字符 C 的距离，然后取左右两个距离的最小值存入最终答案数组中。

代码

```go
package leetcode

import (
  "math"
)

// 解法一
func shortestToChar(s string, c byte) []int {
  n := len(s)
  res := make([]int, n)
  for i := range res {
    res[i] = n
  }
  for i := 0; i < n; i++ {
    if s[i] == c {
      res[i] = 0
    } else if i > 0 {
      res[i] = res[i-1] + 1
    }
  }
  for i := n - 1; i >= 0; i-- {
    if i < n-1 && res[i+1]+1 < res[i] {
      res[i] = res[i+1] + 1
    }
  }
  return res
```

```go
}

// 解法二
func shortestToChar1(s string, c byte) []int {
  res := make([]int, len(s))
  for i := 0; i < len(s); i++ {
    if s[i] == c {
      res[i] = 0
    } else {
      left, right := math.MaxInt32, math.MaxInt32
      for j := i + 1; j < len(s); j++ {
        if s[j] == c {
          right = j - i
          break
        }
      }
      for k := i - 1; k >= 0; k-- {
        if s[k] == c {
          left = i - k
          break
        }
      }
      res[i] = min(left, right)
    }
  }
  return res
}

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

823. Binary Trees With Factors

题目

Given an array of unique integers, `arr`, where each integer `arr[i]` is strictly greater than `1`.

We make a binary tree using these integers, and each number may be used for any number of times. Each non-leaf node's value should be equal to the product of the values of its children.

Return *the number of binary trees we can make*. The answer may be too large so return the answer **modulo** `109 + 7`.

Example 1:

```
Input: arr = [2,4]
Output: 3
Explanation: We can make these trees: [2], [4], [4, 2, 2]
```

Example 2:

```
Input: arr = [2,4,5,10]
Output: 7
Explanation: We can make these trees: [2], [4], [5], [10], [4, 2, 2], [10, 2, 5], [10,
5, 2].
```

Constraints:

- `1 <= arr.length <= 1000`
- `2 <= arr[i] <= 10^9`

题目大意

给出一个含有不重复整数元素的数组，每个整数均大于 1。我们用这些整数来构建二叉树，每个整数可以使用任意次数。其中：每个非叶结点的值应等于它的两个子结点的值的乘积。满足条件的二叉树一共有多少个？返回的结果应模除 10 * 9 + 7。

解题思路

- 首先想到的是暴力解法，先排序，然后遍历所有节点，枚举两两乘积为第三个节点值的组合。然后枚举这些组合并构成树。这里计数的时候要注意，左右孩子如果不是对称的，左右子树相互对调又是一组解。但是这个方法超时了。原因是，暴力枚举了很多次重复的节点和组合。优化这里的方法就是把已经计算过的节点放入 `map` 中。这里有 2 层 `map`，第一层 `map` 记忆化的是两两乘积的组合，将父亲节点作为 `key`，左右 2 个孩子作为 `value`。第二层 `map` 记忆化的是以 `root` 为根节点此时二叉树的种类数，`key` 是 `root`，`value` 存的是种类数。这样优化以后，DFS 暴力解法可以 runtime beats 100%。
- 另外一种解法是 DP。定义 `dp[i]` 代表以 `i` 为根节点的树的种类数。dp[i] 初始都是 1，因为所有节点自身可以形成为自身单个节点为 `root` 的树。同样需要先排序。状态转移方程是：

  {{< katex display >}}
  dp[i] = \sum_{j<i, k<i}^{}dp[j] * dp[k], j * k = i
  {{< /katex >}}

  最后将 `dp[]` 数组中所有结果累加取模即为最终结果，时间复杂度 O(n^2)，空间复杂度 O(n)。

代码

```go
package leetcode

import (
  "sort"
)

const mod = 1e9 + 7
```

```go
// 解法一 DFS
func numFactoredBinaryTrees(arr []int) int {
  sort.Ints(arr)
  numDict := map[int]bool{}
  for _, num := range arr {
    numDict[num] = true
  }
  dict, res := make(map[int][][2]int), 0
  for i, num := range arr {
    for j := i; j < len(arr) && num*arr[j] <= arr[len(arr)-1]; j++ {
      tmp := num * arr[j]
      if !numDict[tmp] {
        continue
      }
      dict[tmp] = append(dict[tmp], [2]int{num, arr[j]})
    }
  }
  cache := make(map[int]int)
  for _, num := range arr {
    res = (res + dfs(num, dict, cache)) % mod
  }
  return res
}

func dfs(num int, dict map[int][][2]int, cache map[int]int) int {
  if val, ok := cache[num]; ok {
    return val
  }
  res := 1
  for _, tuple := range dict[num] {
    a, b := tuple[0], tuple[1]
    x, y := dfs(a, dict, cache), dfs(b, dict, cache)
    tmp := x * y
    if a != b {
      tmp *= 2
    }
    res = (res + tmp) % mod
  }
  cache[num] = res
  return res
}

// 解法二 DP
func numFactoredBinaryTrees1(arr []int) int {
  dp := make(map[int]int)
  sort.Ints(arr)
  for i, curNum := range arr {
    for j := 0; j < i; j++ {
```

```
      factor := arr[j]
      quotient, remainder := curNum/factor, curNum%factor
      if remainder == 0 {
        dp[curNum] += dp[factor] * dp[quotient]
      }
    }
    dp[curNum]++
  }
  totalCount := 0
  for _, count := range dp {
    totalCount += count
  }
  return totalCount % mod
}
```

826. Most Profit Assigning Work

题目

We have jobs: difficulty[i] is the difficulty of the ith job, and profit[i] is the profit of the ith job.

Now we have some workers. worker[i] is the ability of the ith worker, which means that this worker can only complete a job with difficulty at most worker[i].

Every worker can be assigned at most one job, but one job can be completed multiple times.

For example, if 3 people attempt the same job that pays $1, then the total profit will be $3. If a worker cannot complete any job, his profit is $0.

What is the most profit we can make?

Example 1:

```
Input: difficulty = [2,4,6,8,10], profit = [10,20,30,40,50], worker = [4,5,6,7]
Output: 100
Explanation: Workers are assigned jobs of difficulty [4,4,6,6] and they get profit of
[20,20,30,30] seperately.
```

Note:

- 1 <= difficulty.length = profit.length <= 10000
- 1 <= worker.length <= 10000
- difficulty[i], profit[i], worker[i] are in range [1, 10^5]

题目大意

这道题考察的是滑动窗口的问题，也是排序相关的问题。

给出一组任务，每个任务都有一定的难度，每个任务也都有完成以后对应的收益(完成难的任务不一定收益最高)。有一批工人，每个人能处理的任务难度不同。要求输出这批工人完成任务以后的最大收益。

解题思路

先将任务按照难度排序，工人也按照能处理任务难度的能力排序。用一个数组记录下，每个 i 下标，当前能达到的最大收益。计算这个收益只需要从下标为 1 开始，依次比较自己和前一个的收益即可(因为排过序，难度是依次递增的)。有了这个难度依次递增，并且记录了最大收益的数组以后，就可以计算最终结果了。遍历一遍工人数组，如果工人的能力大于任务的难度，就加上这个最大收益。遍历完工人数组，最终结果就是最大收益。

代码

```go
package leetcode

import (
  "fmt"
  "sort"
)

// Task define
type Task struct {
  Difficulty int
  Profit     int
}

// Tasks define
type Tasks []Task

// Len define
func (p Tasks) Len() int { return len(p) }

// Swap define
func (p Tasks) Swap(i, j int) { p[i], p[j] = p[j], p[i] }

// SortByDiff define
type SortByDiff struct{ Tasks }

// Less define
func (p SortByDiff) Less(i, j int) bool {
  return p.Tasks[i].Difficulty < p.Tasks[j].Difficulty
}

func maxProfitAssignment(difficulty []int, profit []int, worker []int) int {
  if len(difficulty) == 0 || len(profit) == 0 || len(worker) == 0 {
    return 0
  }
```

```
  tasks, res, index := []Task{}, 0, 0
  for i := 0; i < len(difficulty); i++ {
    tasks = append(tasks, Task{Difficulty: difficulty[i], Profit: profit[i]})
  }
  sort.Sort(SortByDiff{tasks})
  sort.Ints(worker)
  for i := 1; i < len(tasks); i++ {
    tasks[i].Profit = max(tasks[i].Profit, tasks[i-1].Profit)
  }
  fmt.Printf("tasks = %v worker = %v\n", tasks, worker)
  for _, w := range worker {
    for index < len(difficulty) && w >= tasks[index].Difficulty {
      index++
    }
    fmt.Printf("tasks【index】 = %v\n", tasks[index])
    if index > 0 {
      res += tasks[index-1].Profit
    }
  }
  return res
}
```

828. Count Unique Characters of All Substrings of a Given String

题目

Let's define a function `countUniqueChars(s)` that returns the number of unique characters on `s`, for example if `s = "LEETCODE"` then `"L"`, `"T"`,`"C"`,`"O"`,`"D"` are the unique characters since they appear only once in `s`, therefore `countUniqueChars(s) = 5`.On this problem given a string `s` we need to return the sum of `countUniqueChars(t)` where `t` is a substring of `s`. Notice that some substrings can be repeated so on this case you have to count the repeated ones too.

Since the answer can be very large, return the answer modulo `10 ^ 9 + 7`.

Example 1:

```
Input: s = "ABC"
Output: 10
Explanation: All possible substrings are: "A","B","C","AB","BC" and "ABC".
Evey substring is composed with only unique letters.
Sum of lengths of all substring is 1 + 1 + 1 + 2 + 2 + 3 = 10
```

Example 2:

```
Input: s = "ABA"
Output: 8
Explanation: The same as example 1, except countUniqueChars("ABA") = 1.
```

Example 3:

```
Input: s = "LEETCODE"
Output: 92
```

Constraints:

- `0 <= s.length <= 10^4`
- `s` contain upper-case English letters only.

题目大意

如果一个字符在字符串 S 中有且仅有出现一次，那么我们称其为独特字符。例如，在字符串 S = "LETTER" 中，"L" 和 "R" 可以被称为独特字符。我们再定义 UNIQ(S) 作为字符串 S 中独特字符的个数。那么，在 S = "LETTER" 中， UNIQ("LETTER") = 2。

对于给定字符串 S，计算其所有非空子串的独特字符的个数（即 UNIQ(substring)）之和。如果在 S 的不同位置上出现两个甚至多个相同的子串，那么我们认为这些子串是不同的。考虑到答案可能会非常大，规定返回格式为：结果 mod 10 ^ 9 + 7。

解题思路

- 这一题可以先用暴力解法尝试解题，不过提交以后会发现判题结果是超时。出错的一组数据是一个有 10000 个字符的字符串。暴力解法中间由于遍历了太多的子区间，导致了超时。
- 这道题换一个角度思考问题。当子字符串中字符 X 出现了 2 次以上，那么它就对最终结果没有任何影响，所以只有当某个字符只出现一次的时候才会影响最终结果。再者，一个子字符串中不重复的字符的总个数，也就是这个子字符串 UNIQ 值。例如，“ABC”，这个子字符串的 UNIQ 值是 3，可以这样计算，它属于 A 的独特的字符串，也属于 B 的独特的字符串，也属于 C 的独特的字符串，那么计算这个子字符串的问题可以分解成计算 A 有多少个独特的子字符串，B 有多少个独特的子字符串，C 有多少个独特的子字符串的问题。在计算 A 有多少个子字符串的问题的时候，里面肯定会包含 "ABC" 这个子字符串的。所以原问题就转换成了分别计算给出的字符串中每个字符出现在独特字符串中的总数之和。
- 假设原字符串是 BAABBABBBAAABA，这个字符串中出现了很多 A 和很多 B，假设我们当前计算到了第 3 个 A 的位置了(index = 5)，即标红色的那个 A。如何计算这个 A 在哪些子字符串中是独特的呢？由于子字符串题目中要求必须是连续的区间，所以这个问题很简单。找到这个 A 前一个 A 的下标位置(index = 2)，再找到这个 A 后一个 A 的下标位置(index = 9)，即 BAABBABBBAAABA，第一个 A 和当前计算的 A 中间区间有 2 个字符，第三个 A 和当前计算的 A 中间有 3 个字符。那么当前计算的 A 出现在 `(2 + 1) * (3 + 1) = 12` 个子字符串中是独特的，这 12 个字符串是：`A`，`BA`，`BBA`，`AB`，`ABB`，`ABBB`，`BAB`，`BABB`，`BABBB`，`BBAB`，`BBABB`，`BBABBB`。计算方法，假设当前待计算的字符的下标是 i ，找到当前字符前一次出现的下标位置 left，再找到当前字符后一次出现的下标位置 right，那么左边区间 (left,i) 的***开区间**内包含的字符数是 i - left - 1，右边区间 (i,right) 的***开区间****内包含的字符数是 right - i - 1。左右两边都还需要考虑空字符串的情况，即左右两边都可以不取任何字符，那么

对应的就是只有中间这个待计算的字符 `A`。所以左右两边都还需要再加上空串的情况，左边 i - left - 1 + 1 = i - left，右边 right - i - 1 + 1 = right - i。左右两边的情况进行排列组合，即 (i - left) * (right - i)。针对字符串的每个字符都计算这样的值，最后累积的总和就是题目中要求的总 UNIQ 值。

代码

```
package leetcode

func uniqueLetterString(S string) int {
  res, left, right := 0, 0, 0
  for i := 0; i < len(S); i++ {
    left = i - 1
    for left >= 0 && S[left] != S[i] {
      left--
    }
    right = i + 1
    for right < len(S) && S[right] != S[i] {
      right++
    }
    res += (i - left) * (right - i)
  }
  return res % 1000000007
}

// 暴力解法，超时！时间复杂度 O(n^2)
func uniqueLetterString1(S string) int {
  if len(S) == 0 {
    return 0
  }
  res, mod := 0, 1000000007
  for i := 0; i < len(S); i++ {
    letterMap := map[byte]int{}
    for j := i; j < len(S); j++ {
      letterMap[S[j]]++
      tmp := 0
      for _, v := range letterMap {
        if v > 1 {
          tmp++
        }
      }
      if tmp == len(letterMap) {
        continue
      } else {
        res += len(letterMap) - tmp
      }
    }
  }
  return res % mod
}
```

830. Positions of Large Groups

题目

In a string `s` of lowercase letters, these letters form consecutive groups of the same character.

For example, a string like `s = "abbxxxxzyy"` has the groups `"a"`, `"bb"`, `"xxxx"`, `"z"`, and `"yy"`.

A group is identified by an interval `[start, end]`, where `start` and `end` denote the start and end indices (inclusive) of the group. In the above example, `"xxxx"` has the interval `[3,6]`.

A group is considered **large** if it has 3 or more characters.

Return *the intervals of every **large** group sorted in **increasing order by start index***.

Example 1:

```
Input: s = "abbxxxxzzy"
Output: [[3,6]]
Explanation: "xxxx" is the only large group with start index 3 and end index 6.
```

Example 2:

```
Input: s = "abc"
Output: []
Explanation: We have groups "a", "b", and "c", none of which are large groups.
```

Example 3:

```
Input: s = "abcdddeeeeaabbbcd"
Output: [[3,5],[6,9],[12,14]]
Explanation: The large groups are "ddd", "eeee", and "bbb".
```

Example 4:

```
Input: s = "aba"
Output: []
```

Constraints:

- `1 <= s.length <= 1000`
- `s` contains lower-case English letters only.

题目大意

在一个由小写字母构成的字符串 s 中，包含由一些连续的相同字符所构成的分组。例如，在字符串 s = "abbxxxxzyy" 中，就含有 "a", "bb", "xxxx", "z" 和 "yy" 这样的一些分组。分组可以用区间 [start, end] 表示，其中 start 和 end 分别表示该分组的起始和终止位置的下标。上例中的 "xxxx" 分组用区间表示为 [3,6] 。我们称所有包含大于或等于三个连续字符的分组为 较大分组 。

找到每一个 较大分组 的区间，按起始位置下标递增顺序排序后，返回结果。

解题思路

- 简单题。利用滑动窗口的思想，先扩大窗口的右边界，找到能相同字母且能到达的最右边。记录左右边界。再将窗口的左边界移动到上一次的右边界处。以此类推，重复扩大窗口的右边界，直至扫完整个字符串。最终所有满足题意的较大分组区间都在数组中了。

代码

```go
package leetcode

func largeGroupPositions(S string) [][]int {
  res, end := [][]int{}, 0
  for end < len(S) {
    start, str := end, S[end]
    for end < len(S) && S[end] == str {
      end++
    }
    if end-start >= 3 {
      res = append(res, []int{start, end - 1})
    }
  }
  return res
}
```

832. Flipping an Image

题目

Given a binary matrix `A`, we want to flip the image horizontally, then invert it, and return the resulting image.

To flip an image horizontally means that each row of the image is reversed. For example, flipping `[1, 1, 0]` horizontally results in `[0, 1, 1]`.

To invert an image means that each `0` is replaced by `1`, and each `1` is replaced by `0`. For example, inverting `[0, 1, 1]` results in `[1, 0, 0]`.

Example 1:

```
Input: [[1,1,0],[1,0,1],[0,0,0]]
Output: [[1,0,0],[0,1,0],[1,1,1]]
Explanation: First reverse each row: [[0,1,1],[1,0,1],[0,0,0]].
Then, invert the image: [[1,0,0],[0,1,0],[1,1,1]]
```

Example 2:

```
Input: [[1,1,0,0],[1,0,0,1],[0,1,1,1],[1,0,1,0]]
Output: [[1,1,0,0],[0,1,1,0],[0,0,0,1],[1,0,1,0]]
Explanation: First reverse each row: [[0,0,1,1],[1,0,0,1],[1,1,1,0],[0,1,0,1]].
Then invert the image: [[1,1,0,0],[0,1,1,0],[0,0,0,1],[1,0,1,0]]
```

Notes:

- `1 <= A.length = A[0].length <= 20`
- `0 <= A[i][j] <= 1`

题目大意

给定一个二进制矩阵 A，我们想先水平翻转图像，然后反转图像并返回结果。水平翻转图片就是将图片的每一行都进行翻转，即逆序。例如，水平翻转 [1, 1, 0] 的结果是 [0, 1, 1]。反转图片的意思是图片中的 0 全部被 1 替换， 1 全部被 0 替换。例如，反转 [0, 1, 1] 的结果是 [1, 0, 0]。

解题思路

- 给定一个二进制矩阵，要求先水平翻转，然后再反转(1→0 , 0→1)。
- 简单题，按照题意先水平翻转，再反转即可。

代码

```go
package leetcode

func flipAndInvertImage(A [][]int) [][]int {
  for i := 0; i < len(A); i++ {
    for a, b := 0, len(A[i])-1; a < b; a, b = a+1, b-1 {
      A[i][a], A[i][b] = A[i][b], A[i][a]
    }
    for a := 0; a < len(A[i]); a++ {
      A[i][a] = (A[i][a] + 1) % 2
    }
  }
  return A
}
```

834. Sum of Distances in Tree

题目

An undirected, connected tree with `N` nodes labelled `0...N-1` and `N-1edges` are given.

The `i` th edge connects nodes `edges[i][0]` and `edges[i][1]` together.

Return a list `ans`, where `ans[i]` is the sum of the distances between node `i` and all other nodes.

Example 1:

```
Input: N = 6, edges = [[0,1],[0,2],[2,3],[2,4],[2,5]]
Output: [8,12,6,10,10,10]
Explanation:
Here is a diagram of the given tree:
  0
 / \
1   2
   /|\
  3 4 5
We can see that dist(0,1) + dist(0,2) + dist(0,3) + dist(0,4) + dist(0,5)
equals 1 + 1 + 2 + 2 + 2 = 8.  Hence, answer[0] = 8, and so on.
```

Note: `1 <= N <= 10000`

题目大意

给定一个无向、连通的树。树中有 N 个标记为 0...N-1 的节点以及 N-1 条边。第 i 条边连接节点 edges[i][0] 和 edges[i][1] 。返回一个表示节点 i 与其他所有节点距离之和的列表 ans。

说明: 1 <= N <= 10000

解题思路

- 给出 N 个节点和这些节点之间的一些边的关系。要求求出分别以 x 为根节点到所有节点路径和。
- 这一题虽说描述的是求树的路径，但是完全可以当做图来做，因为并不是二叉树，是多叉树。这一题的解题思路是先一次 DFS 求出以 0 为根节点到各个节点的路径和(不以 0 为节点也可以，可以取任意节点作为开始)。第二次 DFS 求出从 0 根节点转换到其他各个节点的路径和。由于第一次计算出来以 0 为节点的路径和是正确的，所以计算其他节点为根节点的路径和只需要转换一下就可以得到正确结果。经过 2 次 DFS 之后就可以得到所有节点以自己为根节点到所有节点的路径和了。
- 如何从以 0 为根节点到其他所有节点的路径和转换到以其他节点为根节点到所有节点的路径和呢？从 0 节点换成 x 节点，只需要在 0 到所有节点的路径和基础上增增减减就可以了。增加的是 x 节点到除去以 x 为根节点所有子树以外的节点的路径，有多少个节点就增加多少条路径。减少的是 0 到以 x 为根节点所有子树节点的路径和，包含 0 到 x 根节点，有多少节点就减少多少条路径。所以在第一次 DFS 中需要计算好每个节点以自己为根节点的子树总数和(包含自己在内)，这样在第二次 DFS 中可以直接拿来做转换。具体细节的实现见代码。

代码

```
package leetcode

func sumOfDistancesInTree(N int, edges [][]int) []int {
  // count[i] 中存储的是以 i 为根节点，所有子树结点和根节点的总数
  tree, visited, count, res := make([][]int, N), make([]bool, N), make([]int, N),
make([]int, N)
  for _, e := range edges {
    i, j := e[0], e[1]
    tree[i] = append(tree[i], j)
    tree[j] = append(tree[j], i)
  }
  deepFirstSearch(0, visited, count, res, tree)
  // 重置访问状态，再进行一次 DFS
  visited = make([]bool, N)
  // 进入第二次 DFS 之前，只有 res[0] 里面存的是正确的值，因为第一次 DFS 计算出了以 0 为根节点的所有
路径和
  // 第二次 DFS 的目的是把以 0 为根节点的路径和转换成以 n 为根节点的路径和
  deepSecondSearch(0, visited, count, res, tree)

  return res
}

func deepFirstSearch(root int, visited []bool, count, res []int, tree [][]int) {
  visited[root] = true
  for _, n := range tree[root] {
    if visited[n] {
      continue
    }
    deepFirstSearch(n, visited, count, res, tree)
    count[root] += count[n]
    // root 节点到 n 的所有路径和 = 以 n 为根节点到所有子树的路径和 res[n] + root 到 count[n]
中每个节点的个数(root 节点和以 n 为根节点的每个节点都增加一条路径)
    // root 节点和以 n 为根节点的每个节点都增加一条路径 = 以 n 为根节点，子树节点数和根节点数的总和，
即 count[n]
    res[root] += res[n] + count[n]
  }
  count[root]++
}

// 从 root 开始，把 root 节点的子节点，依次设置成新的根节点
func deepSecondSearch(root int, visited []bool, count, res []int, tree [][]int) {
  N := len(visited)
  visited[root] = true
  for _, n := range tree[root] {
    if visited[n] {
      continue
```

```
    }
    // 根节点从 root 变成 n 后
    // res[root] 存储的是以 root 为根节点到所有节点的路径总长度
    // 1. root 到 n 节点增加的路径长度 = root 节点和以 n 为根节点的每个节点都增加一条路径 = 以 n 为根节点，子树节点数和根节点数的总和，即 count[n]
    // 2. n 到以 n 为根节点的所有子树节点以外的节点增加的路径长度 = n 节点和非 n 为根节点子树的每个节点都增加一条路径 = N - count[n]
    // 所以把根节点从 root 转移到 n，需要增加的路径是上面👆第二步计算的，需要减少的路径是上面👆第一步计算的
    res[n] = res[root] + (N - count[n]) - count[n]
    deepSecondSearch(n, visited, count, res, tree)
  }
}
```

836. Rectangle Overlap

题目

A rectangle is represented as a list `[x1, y1, x2, y2]`, where `(x1, y1)` are the coordinates of its bottom-left corner, and `(x2, y2)` are the coordinates of its top-right corner.

Two rectangles overlap if the area of their intersection is positive. To be clear, two rectangles that only touch at the corner or edges do not overlap.

Given two (axis-aligned) rectangles, return whether they overlap.

Example 1:

```
Input: rec1 = [0,0,2,2], rec2 = [1,1,3,3]
Output: true
```

Example 2:

```
Input: rec1 = [0,0,1,1], rec2 = [1,0,2,1]
Output: false
```

Notes:

1. Both rectangles `rec1` and `rec2` are lists of 4 integers.
2. All coordinates in rectangles will be between `-10^9` and `10^9`.

题目大意

矩形以列表 [x1, y1, x2, y2] 的形式表示，其中 (x1, y1) 为左下角的坐标，(x2, y2) 是右上角的坐标。如果相交的面积为正，则称两矩形重叠。需要明确的是，只在角或边接触的两个矩形不构成重叠。给出两个矩形，判断它们是否重叠并返回结果。

说明：

1. 两个矩形 rec1 和 rec2 都以含有四个整数的列表的形式给出。
2. 矩形中的所有坐标都处于 -10^9 和 10^9 之间。

解题思路

- 给出两个矩形的坐标，判断两个矩形是否重叠。
- 几何题，按照几何方法判断即可。

代码

```
package leetcode

func isRectangleOverlap(rec1 []int, rec2 []int) bool {
  return rec1[0] < rec2[2] && rec2[0] < rec1[2] && rec1[1] < rec2[3] && rec2[1] <
rec1[3]
}
```

838. Push Dominoes

题目

There are N dominoes in a line, and we place each domino vertically upright.

In the beginning, we simultaneously push some of the dominoes either to the left or to the right.

After each second, each domino that is falling to the left pushes the adjacent domino on the left.

Similarly, the dominoes falling to the right push their adjacent dominoes standing on the right.

When a vertical domino has dominoes falling on it from both sides, it stays still due to the balance of the forces.

For the purposes of this question, we will consider that a falling domino expends no additional force to a falling or already fallen domino.

Given a string "S" representing the initial state. S[i] = 'L', if the i-th domino has been pushed to the left; S[i] = 'R', if the i-th domino has been pushed to the right; S[i] = '.', if the i-th domino has not been pushed.

Return a string representing the final state.

Example 1:

```
Input: ".L.R...LR..L.."
Output: "LL.RR.LLRRLL.."
```

Example 2:

```
Input: "RR.L"
Output: "RR.L"
Explanation: The first domino expends no additional force on the second domino.
```

Note:

- 0 <= N <= 10^5
- String dominoes contains only 'L', 'R' and '.'

题目大意

这道题是一个道模拟题，考察的也是滑动窗口的问题。

给出一个字符串，L 代表这个多米诺骨牌会往左边倒，R 代表这个多米诺骨牌会往右边倒，问最终这些牌倒下去以后，情况是如何的，输出最终情况的字符串。

解题思路

这道题可以预先在初始字符串头和尾都添加一个字符串，左边添加 L，右边添加 R，辅助判断。

代码

```go
package leetcode

// 解法一
func pushDominoes(dominoes string) string {
  d := []byte(dominoes)
  for i := 0; i < len(d); {
    j := i + 1
    for j < len(d)-1 && d[j] == '.' {
      j++
    }
    push(d[i : j+1])
    i = j
  }
  return string(d)
}

func push(d []byte) {
  first, last := 0, len(d)-1
  switch d[first] {
  case '.', 'L':
    if d[last] == 'L' {
      for ; first < last; first++ {
```

```go
        d[first] = 'L'
      }
    }
  case 'R':
    if d[last] == '.' || d[last] == 'R' {
      for ; first <= last; first++ {
        d[first] = 'R'
      }
    } else if d[last] == 'L' {
      for first < last {
        d[first] = 'R'
        d[last] = 'L'
        first++
        last--
      }
    }
  }
}

// 解法二
func pushDominoes1(dominoes string) string {
  dominoes = "L" + dominoes + "R"
  res := ""
  for i, j := 0, 1; j < len(dominoes); j++ {
    if dominoes[j] == '.' {
      continue
    }
    if i > 0 {
      res += string(dominoes[i])
    }
    middle := j - i - 1
    if dominoes[i] == dominoes[j] {
      for k := 0; k < middle; k++ {
        res += string(dominoes[i])
      }
    } else if dominoes[i] == 'L' && dominoes[j] == 'R' {
      for k := 0; k < middle; k++ {
        res += string('.')
      }
    } else {
      for k := 0; k < middle/2; k++ {
        res += string('R')
      }
      for k := 0; k < middle%2; k++ {
        res += string('.')
      }
      for k := 0; k < middle/2; k++ {
        res += string('L')
      }
```

```
    }
    i = j
  }
  return res
}
```

839. Similar String Groups

题目

Two strings `X` and `Y` are similar if we can swap two letters (in different positions) of `X`, so that it equals `Y`.

For example, `"tars"` and `"rats"` are similar (swapping at positions `0` and `2`), and `"rats"` and `"arts"` are similar, but `"star"` is not similar to `"tars"`, `"rats"`, or `"arts"`.

Together, these form two connected groups by similarity: `{"tars", "rats", "arts"}` and `{"star"}`. Notice that `"tars"` and `"arts"` are in the same group even though they are not similar. Formally, each group is such that a word is in the group if and only if it is similar to at least one other word in the group.

We are given a list `A` of strings. Every string in `A` is an anagram of every other string in `A`. How many groups are there?

Example 1:

```
Input: ["tars","rats","arts","star"]
Output: 2
```

Note:

1. `A.length <= 2000`
2. `A[i].length <= 1000`
3. `A.length * A[i].length <= 20000`
4. All words in `A` consist of lowercase letters only.
5. All words in `A` have the same length and are anagrams of each other.
6. The judging time limit has been increased for this question.

题目大意

如果我们交换字符串 X 中的两个不同位置的字母，使得它和字符串 Y 相等，那么称 X 和 Y 两个字符串相似。

例如，"tars" 和 "rats" 是相似的 (交换 0 与 2 的位置)； "rats" 和 "arts" 也是相似的，但是 "star" 不与 "tars"，"rats"，或 "arts" 相似。

总之，它们通过相似性形成了两个关联组：{"tars", "rats", "arts"} 和 {"star"}。注意，"tars" 和 "arts" 是在同一组中，即使它们并不相似。形式上，对每个组而言，要确定一个单词在组中，只需要这个词和该组中至少一个单词相似。我们给出了一个不包含重复的字符串列表 A。列表中的每个字符串都是 A 中其它所有字符串的一个字母异位词。请问 A 中有多少个相似字符串组?

提示：

- A.length <= 2000
- A[i].length <= 1000
- A.length * A[i].length <= 20000
- A 中的所有单词都只包含小写字母。
- A 中的所有单词都具有相同的长度，且是彼此的字母异位词。
- 此问题的判断限制时间已经延长。

备注：

- 字母异位词[anagram]，一种把某个字符串的字母的位置（顺序）加以改换所形成的新词。

解题思路

- 给出一个字符串数组，要求找出这个数组中，"不相似"的字符串有多少种。相似的字符串的定义是：如果 A 和 B 字符串只需要交换一次字母的位置就能变成两个相等的字符串，那么 A 和 B 是相似的。
- 这一题的解题思路是用并查集。先将题目中的“相似”的定义转换一下，A 和 B 相似的意思是，A 和 B 中只有 2 个字符不相等，其他字符都相等，这样交换一次才能完全相等。有没有可能这两个字符交换了也不相等呢？这种情况不用考虑，因为题目中提到了给的字符串都是 `anagram` 的(`anagram` 的意思是，字符串的任意排列组合)。那么这题就比较简单了，只需要判断每两个字符串是否“相似”，如果相似就 `union()`，最后看并查集中有几个集合即可。

代码

```
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/template"
)

func numSimilarGroups(A []string) int {
  uf := template.UnionFind{}
  uf.Init(len(A))
  for i := 0; i < len(A); i++ {
    for j := i + 1; j < len(A); j++ {
      if isSimilar(A[i], A[j]) {
        uf.Union(i, j)
      }
    }
  }
  return uf.TotalCount()
}

func isSimilar(a, b string) bool {
  var n int
  for i := 0; i < len(a); i++ {
```

```
    if a[i] != b[i] {
      n++
      if n > 2 {
        return false
      }
    }
  }
  return true
}
```

841. Keys and Rooms

题目

There are `N` rooms and you start in room `0`. Each room has a distinct number in `0, 1, 2, ..., N-1`, and each room may have some keys to access the next room.

Formally, each room `i` has a list of keys `rooms[i]`, and each key `rooms[i][j]` is an integer in `[0, 1, ..., N-1]` where `N = rooms.length`. A key `rooms[i][j] = v` opens the room with number `v`.

Initially, all the rooms start locked (except for room `0`).

You can walk back and forth between rooms freely.

Return `true` if and only if you can enter every room.

Example 1:

```
Input: [[1],[2],[3],[]]
Output: true
Explanation:
We start in room 0, and pick up key 1.
We then go to room 1, and pick up key 2.
We then go to room 2, and pick up key 3.
We then go to room 3.  Since we were able to go to every room, we return true.
```

Example 2:

```
Input: [[1,3],[3,0,1],[2],[0]]
Output: false
Explanation: We can't enter the room with number 2.
```

Note:

1. `1 <= rooms.length <= 1000`
2. `0 <= rooms[i].length <= 1000`

3. The number of keys in all rooms combined is at most `3000`.

题目大意

有 N 个房间，开始时你位于 0 号房间。每个房间有不同的号码：0，1，2，...，N-1，并且房间里可能有一些钥匙能使你进入下一个房间。

在形式上，对于每个房间 i 都有一个钥匙列表 rooms[i]，每个钥匙 rooms[i][j] 由 [0,1，...，N-1] 中的一个整数表示，其中 N = rooms.length。 钥匙 rooms[i][j] = v 可以打开编号为 v 的房间。最初，除 0 号房间外的其余所有房间都被锁住。你可以自由地在房间之间来回走动。如果能进入每个房间返回 true，否则返回 false。

提示：

- 1 <= rooms.length <= 1000
- 0 <= rooms[i].length <= 1000
- 所有房间中的钥匙数量总计不超过 3000。

解题思路

- 给出一个房间数组，每个房间里面装了一些钥匙。0 号房间默认是可以进入的，房间进入顺序没有要求，问最终能否进入所有房间。
- 用 DFS 依次深搜所有房间的钥匙，如果都能访问到，最终输出 true。这题算是 DFS 里面的简单题。

代码

```go
func canVisitAllRooms(rooms [][]int) bool {
  visited := make(map[int]bool)
  visited[0] = true
  dfsVisitAllRooms(rooms, visited, 0)
  return len(rooms) == len(visited)
}

func dfsVisitAllRooms(es [][]int, visited map[int]bool, from int) {
  for _, to := range es[from] {
    if visited[to] {
      continue
    }
    visited[to] = true
    dfsVisitAllRooms(es, visited, to)
  }
}
```

842. Split Array into Fibonacci Sequence

题目

Given a string `S` of digits, such as `S = "123456579"`, we can split it into a *Fibonacci-like sequence* `[123, 456, 579].`

Formally, a Fibonacci-like sequence is a list `F` of non-negative integers such that:

- `0 <= F[i] <= 2^31 - 1`, (that is, each integer fits a 32-bit signed integer type);
- `F.length >= 3`;
- and `F[i] + F[i+1] = F[i+2]` for all `0 <= i < F.length - 2`.

Also, note that when splitting the string into pieces, each piece must not have extra leading zeroes, except if the piece is the number 0 itself.

Return any Fibonacci-like sequence split from `S`, or return `[]` if it cannot be done.

Example 1:

```
Input: "123456579"
Output: [123,456,579]
```

Example 2:

```
Input: "11235813"
Output: [1,1,2,3,5,8,13]
```

Example 3:

```
Input: "112358130"
Output: []
Explanation: The task is impossible.
```

Example 4:

```
Input: "0123"
Output: []
Explanation: Leading zeroes are not allowed, so "01", "2", "3" is not valid.
```

Example 5:

```
Input: "1101111"
Output: [110, 1, 111]
Explanation: The output [11, 0, 11, 11] would also be accepted.
```

Note:

1. `1 <= S.length <= 200`
2. `S` contains only digits.

题目大意

给定一个数字字符串 S，比如 S = "123456579"，我们可以将它分成斐波那契式的序列 [123, 456, 579]。斐波那契式序列是一个非负整数列表 F，且满足：

- 0 <= F[i] <= 2^31 - 1，（也就是说，每个整数都符合 32 位有符号整数类型）；
- F.length >= 3；
- 对于所有的0 <= i < F.length - 2，都有 F[i] + F[i+1] = F[i+2] 成立。

另外，请注意，将字符串拆分成小块时，每个块的数字一定不要以零开头，除非这个块是数字 0 本身。返回从 S 拆分出来的所有斐波那契式的序列块，如果不能拆分则返回 []。

解题思路

- 这一题是第 306 题的加强版。第 306 题要求判断字符串是否满足斐波那契数列形式。这一题要求输出按照斐波那契数列形式分割之后的数字数组。
- 这一题思路和第 306 题基本一致，需要注意的是题目中的一个限制条件，`0 <= F[i] <= 2^31 - 1`，注意这个条件，笔者开始没注意，后面输出解就出现错误了，可以看笔者的测试文件用例的最后两组数据，这两组都是可以分解成斐波那契数列的，但是由于分割以后的数字都大于了 `2^31 - 1`，所以这些解都不能要！
- 这一题也要特别注意剪枝条件，没有剪枝条件，时间复杂度特别高，加上合理的剪枝条件以后，0ms 通过。

代码

```
package leetcode

import (
  "strconv"
  "strings"
)

func splitIntoFibonacci(S string) []int {
  if len(S) < 3 {
    return []int{}
  }
  res, isComplete := []int{}, false
  for firstEnd := 0; firstEnd < len(S)/2; firstEnd++ {
    if S[0] == '0' && firstEnd > 0 {
      break
    }
    first, _ := strconv.Atoi(S[:firstEnd+1])
    if first >= 1<<31 { // 题目要求每个数都要小于 2^31 - 1 = 2147483647，此处剪枝很关键！
      break
    }
    for secondEnd := firstEnd + 1; max(firstEnd, secondEnd-firstEnd) <= len(S)-
secondEnd; secondEnd++ {
      if S[firstEnd+1] == '0' && secondEnd-firstEnd > 1 {
        break
      }
      second, _ := strconv.Atoi(S[firstEnd+1 : secondEnd+1])
```

```
      if second >= 1<<31 { // 题目要求每个数都要小于 2^31 - 1 = 2147483647，此处剪枝很关键！
        break
      }
      findRecursiveCheck(S, first, second, secondEnd+1, &res, &isComplete)
    }
  }
  return res
}

//Propagate for rest of the string
func findRecursiveCheck(S string, x1 int, x2 int, left int, res *[]int, isComplete
*bool) {
  if x1 >= 1<<31 || x2 >= 1<<31 { // 题目要求每个数都要小于 2^31 - 1 = 2147483647，此处剪枝很
关键！
    return
  }
  if left == len(S) {
    if !*isComplete {
      *isComplete = true
      *res = append(*res, x1)
      *res = append(*res, x2)
    }
    return
  }
  if strings.HasPrefix(S[left:], strconv.Itoa(x1+x2)) && !*isComplete {
    *res = append(*res, x1)
    findRecursiveCheck(S, x2, x1+x2, left+len(strconv.Itoa(x1+x2)), res, isComplete)
    return
  }
  if len(*res) > 0 && !*isComplete {
    *res = (*res)[:len(*res)-1]
  }
  return
}
```

844. Backspace String Compare

题目

Given two strings S and T, return if they are equal when both are typed into empty text editors. # means a backspace character.

Example 1:

```
Input: S = "ab#c", T = "ad#c"
Output: true
Explanation: Both S and T become "ac".
```

Example 2:

```
Input: S = "ab##", T = "c#d#"
Output: true
Explanation: Both S and T become "".
```

Example 3:

```
Input: S = "a##c", T = "#a#c"
Output: true
Explanation: Both S and T become "c".
```

Example 4:

```
Input: S = "a#c", T = "b"
Output: false
Explanation: S becomes "c" while T becomes "b".
```

Note:

- 1 <= S.length <= 200
- 1 <= T.length <= 200
- S and T only contain lowercase letters and '#' characters.

Follow up:

- Can you solve it in O(N) time and O(1) space?

题目大意

给 2 个字符串，如果遇到 # 号字符，就回退一个字符。问最终的 2 个字符串是否完全一致。

解题思路

这一题可以用栈的思想来模拟，遇到 # 字符就回退一个字符。不是 # 号就入栈一个字符。比较最终 2 个字符串即可。

代码

```
package leetcode

func backspaceCompare(S string, T string) bool {
  s := make([]rune, 0)
  for _, c := range S {
    if c == '#' {
      if len(s) > 0 {
        s = s[:len(s)-1]
      }
    } else {
      s = append(s, c)
    }
  }
  s2 := make([]rune, 0)
  for _, c := range T {
    if c == '#' {
      if len(s2) > 0 {
        s2 = s2[:len(s2)-1]
      }
    } else {
      s2 = append(s2, c)
    }
  }
  return string(s) == string(s2)
}
```

845. Longest Mountain in Array

题目

Let's call any (contiguous) subarray B (of A) a mountain if the following properties hold:

- B.length >= 3
- There exists some 0 < i < B.length - 1 such that B[0] < B[1] < ... B[i-1] < B[i] > B[i+1] > ... > B[B.length - 1]

(Note that B could be any subarray of A, including the entire array A.)

Given an array A of integers, return the length of the longest mountain.

Return 0 if there is no mountain.

Example 1:

```
Input: [2,1,4,7,3,2,5]
Output: 5
Explanation: The largest mountain is [1,4,7,3,2] which has length 5.
```

Example 2:

```
Input: [2,2,2]
Output: 0
Explanation: There is no mountain.
```

Note:

- 0 <= A.length <= 10000
- 0 <= A[i] <= 10000

Follow up:

- Can you solve it using only one pass?
- Can you solve it in O(1) space?

题目大意

这道题考察的是滑动窗口的问题。

给出一个数组，要求求出这个数组里面“山”最长的长度。“山”的意思是，从一个数开始逐渐上升，到顶以后，逐渐下降。

解题思路

这道题解题思路也是滑动窗口，只不过在滑动的过程中多判断一个上升和下降的状态即可。

代码

```
package leetcode
```

```go
func longestMountain(A []int) int {
  left, right, res, isAscending := 0, 0, 0, true
  for left < len(A) {
    if right+1 < len(A) && ((isAscending == true && A[right+1] > A[left] && A[right+1]
> A[right]) || (right != left && A[right+1] < A[right])) {
      if A[right+1] < A[right] {
        isAscending = false
      }
      right++
    } else {
      if right != left && isAscending == false {
        res = max(res, right-left+1)
      }
      left++
      if right < left {
        right = left
      }
      if right == left {
        isAscending = true
      }
    }
  }
  return res
}
```

850. Rectangle Area II

题目

We are given a list of (axis-aligned) `rectangles`. Each `rectangle[i] = [x1, y1, x2, y2]`, where (x1, y1) are the coordinates of the bottom-left corner, and (x2, y2) are the coordinates of the top-right corner of the `i` th rectangle.

Find the total area covered by all `rectangles` in the plane. Since the answer may be too large, **return it modulo 10^9 + 7**.

Example 1:

```
Input: [[0,0,2,2],[1,0,2,3],[1,0,3,1]]
Output: 6
Explanation: As illustrated in the picture.
```

Example 2:

```
Input: [[0,0,1000000000,1000000000]]
Output: 49
Explanation: The answer is 10^18 modulo (10^9 + 7), which is (10^9)^2 = (-7)^2 = 49.
```

Note:

- `1 <= rectangles.length <= 200`
- `rectanges[i].length = 4`
- `0 <= rectangles[i][j] <= 10^9`
- The total area covered by all rectangles will never exceed `2^63 - 1` and thus will fit in a 64-bit signed integer.

题目大意

我们给出了一个（轴对齐的）矩形列表 rectangles。 对于 rectangle[i] = [x1, y1, x2, y2]，其中（x1，y1）是矩形 i 左下角的坐标，（x2，y2）是该矩形右上角的坐标。找出平面中所有矩形叠加覆盖后的总面积。由于答案可能太大，请返回它对 10 ^ 9 + 7 取模的结果。

提示：

- 1 <= rectangles.length <= 200
- rectanges[i].length = 4
- 0 <= rectangles[i][j] <= 10^9
- 矩形叠加覆盖后的总面积不会超越 2^63 - 1 ，这意味着可以用一个 64 位有符号整数来保存面积结果。

解题思路

- 在二维坐标系中给出一些矩形，要求这些矩形合并之后的面积。由于矩形有重叠，所以需要考虑合并以后的面积。矩形的坐标值也会很大。
- 这一题给人的感觉很像第 218 题，求天际线的过程也是有楼挡楼，重叠的情况。不过那一题只用求天际线的拐点，所以我们可以对区间做“右边界减一”的处理，防止两个相邻区间因为共点，而导致结果错误。但是这一题如果还是用相同的做法，就会出错，因为“右边界减一”以后，面积会少一部分，最终得到的结果也是偏小的。所以这一题要将线段树改造一下。
- 思路是先讲 Y 轴上的坐标离线化，转换成线段树。将矩形的 2 条边变成扫描线，左边是入边，右边是出边。

Y 轴离散化

@halfrost

- 再从左往右遍历每条扫描线，并对 Y 轴上的线段树进行 update。X 轴上的每个坐标区间 * query 线段树总高度的结果 = 区间面积。最后将 X 轴对应的每个区间面积加起来，就是最终矩形合并以后的面积。如下图中间的图。

扫描线

@halfrost

需要注意的一点是，**每次 query 的结果并不一定是连续线段**。如上图最右边的图，中间有一段是可能出现镂空的。这种情况看似复杂，其实很简单，因为每段线段树的线段代表的权值高度是不同的，每次 query 最大高度得到的结果已经考虑了中间可能有镂空的情况了。

- 具体做法，先把各个矩形在 Y 轴方向上离散化，这里的**线段树叶子节点不再是一个点了，而是一个区间长度为 1 的区间段**。

Y 轴离散化

@halfrost

每个叶子节点也不再是存储一个 int 值，而是存 2 个值，一个是 count 值，用来记录这条区间被覆盖的次数，另一个值是 val 值，用来反映射该线段长度是多少，因为 Y 轴被离散化了，区间坐标间隔都是 1，但是实际 Y 轴的高度并不是 1 ，所以用 val 来反映射原来的高度。

- 初始化线段树，叶子节点的 count = 0，val 根据题目给的 Y 坐标进行计算。

初始化

@halfrost

- 从左往右遍历每个扫描线。每条扫面线都把对应 update 更新到叶子节点。pushUp 的时候需要合并每个区间段的高度 val 值。如果有区间没有被覆盖，那么这个区间高度 val 为 0，这也就处理了可能“中间镂空”的情况。

```
func (sat *SegmentAreaTree) pushUp(treeIndex, leftTreeIndex, rightTreeIndex int) {
  newCount, newValue := sat.merge(sat.tree[leftTreeIndex].count,
sat.tree[rightTreeIndex].count), 0
  if sat.tree[leftTreeIndex].count > 0 && sat.tree[rightTreeIndex].count > 0 {
    newValue = sat.merge(sat.tree[leftTreeIndex].val, sat.tree[rightTreeIndex].val)
  } else if sat.tree[leftTreeIndex].count > 0 && sat.tree[rightTreeIndex].count ==
0 {
    newValue = sat.tree[leftTreeIndex].val
  } else if sat.tree[leftTreeIndex].count == 0 && sat.tree[rightTreeIndex].count >
0 {
    newValue = sat.tree[rightTreeIndex].val
  }
  sat.tree[treeIndex] = SegmentItem{count: newCount, val: newValue}
}
```

- 扫描每一个扫描线，先 pushDown 到叶子节点，再 pushUp 到根节点。

Sweep line = [0,1] +1

@halfrost

Sweep line = [0,2] −1

@halfrost

Sweep line = [0,3] −1

@halfrost

- 遍历到倒数第 2 根扫描线的时候就能得到结果了。因为最后一根扫描线 update 以后，整个线段树全部都归为初始化状态了。

- 这一题是线段树扫面线解法的经典题。

代码

```
package leetcode

import (
  "sort"
)

func rectangleArea(rectangles [][]int) int {
  sat, res := SegmentAreaTree{}, 0
  posXMap, posX, posYMap, posY, lines := discretization850(rectangles)
  tmp := make([]int, len(posYMap))
  for i := 0; i < len(tmp)-1; i++ {
    tmp[i] = posY[i+1] - posY[i]
  }
  sat.Init(tmp, func(i, j int) int {
    return i + j
  })
  for i := 0; i < len(posY)-1; i++ {
    tmp[i] = posY[i+1] - posY[i]
  }
  for i := 0; i < len(posX)-1; i++ {
    for _, v := range lines[posXMap[posX[i]]] {
      sat.Update(posYMap[v.start], posYMap[v.end], v.state)
    }
    res += ((posX[i+1] - posX[i]) * sat.Query(0, len(posY)-1)) % 1000000007
  }
```

```go
  return res % 1000000007
}

func discretization850(positions [][]int) (map[int]int, []int, map[int]int, []int,
map[int][]LineItem) {
  tmpXMap, tmpYMap, posXArray, posXMap, posYArray, posYMap, lines := map[int]int{},
map[int]int{}, []int{}, map[int]int{}, []int{}, map[int]int{}, map[int][]LineItem{}
  for _, pos := range positions {
    tmpXMap[pos[0]]++
    tmpXMap[pos[2]]++
  }
  for k := range tmpXMap {
    posXArray = append(posXArray, k)
  }
  sort.Ints(posXArray)
  for i, pos := range posXArray {
    posXMap[pos] = i
  }

  for _, pos := range positions {
    tmpYMap[pos[1]]++
    tmpYMap[pos[3]]++
    tmp1 := lines[posXMap[pos[0]]]
    tmp1 = append(tmp1, LineItem{start: pos[1], end: pos[3], state: 1})
    lines[posXMap[pos[0]]] = tmp1
    tmp2 := lines[posXMap[pos[2]]]
    tmp2 = append(tmp2, LineItem{start: pos[1], end: pos[3], state: -1})
    lines[posXMap[pos[2]]] = tmp2
  }
  for k := range tmpYMap {
    posYArray = append(posYArray, k)
  }
  sort.Ints(posYArray)
  for i, pos := range posYArray {
    posYMap[pos] = i
  }
  return posXMap, posXArray, posYMap, posYArray, lines
}

// LineItem define
type LineItem struct { // 垂直于 x 轴的线段
  start, end, state int // state = 1 代表进入, -1 代表离开
}

// SegmentItem define
type SegmentItem struct {
  count int
  val   int
}
```

```go
// SegmentAreaTree define
type SegmentAreaTree struct {
  data        []int
  tree        []SegmentItem
  left, right int
  merge       func(i, j int) int
}

// Init define
func (sat *SegmentAreaTree) Init(nums []int, oper func(i, j int) int) {
  sat.merge = oper
  data, tree := make([]int, len(nums)), make([]SegmentItem, 4*len(nums))
  for i := 0; i < len(nums); i++ {
    data[i] = nums[i]
  }
  sat.data, sat.tree = data, tree
  if len(nums) > 0 {
    sat.buildSegmentTree(0, 0, len(nums)-1)
  }
}

// 在 treeIndex 的位置创建 [left....right] 区间的线段树
func (sat *SegmentAreaTree) buildSegmentTree(treeIndex, left, right int) {
  if left == right-1 {
    sat.tree[treeIndex] = SegmentItem{count: 0, val: sat.data[left]}
    return
  }
  midTreeIndex, leftTreeIndex, rightTreeIndex := left+(right-left)>>1,
sat.leftChild(treeIndex), sat.rightChild(treeIndex)
  sat.buildSegmentTree(leftTreeIndex, left, midTreeIndex)
  sat.buildSegmentTree(rightTreeIndex, midTreeIndex, right)
  sat.pushUp(treeIndex, leftTreeIndex, rightTreeIndex)
}

func (sat *SegmentAreaTree) pushUp(treeIndex, leftTreeIndex, rightTreeIndex int) {
  newCount, newValue := sat.merge(sat.tree[leftTreeIndex].count,
sat.tree[rightTreeIndex].count), 0
  if sat.tree[leftTreeIndex].count > 0 && sat.tree[rightTreeIndex].count > 0 {
    newValue = sat.merge(sat.tree[leftTreeIndex].val, sat.tree[rightTreeIndex].val)
  } else if sat.tree[leftTreeIndex].count > 0 && sat.tree[rightTreeIndex].count == 0 {
    newValue = sat.tree[leftTreeIndex].val
  } else if sat.tree[leftTreeIndex].count == 0 && sat.tree[rightTreeIndex].count > 0 {
    newValue = sat.tree[rightTreeIndex].val
  }
  sat.tree[treeIndex] = SegmentItem{count: newCount, val: newValue}
}

func (sat *SegmentAreaTree) leftChild(index int) int {
```

```go
  return 2*index + 1
}

func (sat *SegmentAreaTree) rightChild(index int) int {
  return 2*index + 2
}

// 查询 [left....right] 区间内的值

// Query define
func (sat *SegmentAreaTree) Query(left, right int) int {
  if len(sat.data) > 0 {
    return sat.queryInTree(0, 0, len(sat.data)-1, left, right)
  }
  return 0
}

func (sat *SegmentAreaTree) queryInTree(treeIndex, left, right, queryLeft, queryRight
int) int {
  midTreeIndex, leftTreeIndex, rightTreeIndex := left+(right-left)>>1,
sat.leftChild(treeIndex), sat.rightChild(treeIndex)
  if left > queryRight || right < queryLeft { // segment completely outside range
    return 0 // represents a null node
  }
  if queryLeft <= left && queryRight >= right { // segment completely inside range
    if sat.tree[treeIndex].count > 0 {
      return sat.tree[treeIndex].val
    }
    return 0
  }
  if queryLeft > midTreeIndex {
    return sat.queryInTree(rightTreeIndex, midTreeIndex, right, queryLeft, queryRight)
  } else if queryRight <= midTreeIndex {
    return sat.queryInTree(leftTreeIndex, left, midTreeIndex, queryLeft, queryRight)
  }
  // merge query results
  return sat.merge(sat.queryInTree(leftTreeIndex, left, midTreeIndex, queryLeft,
midTreeIndex),
    sat.queryInTree(rightTreeIndex, midTreeIndex, right, midTreeIndex, queryRight))
}

// Update define
func (sat *SegmentAreaTree) Update(updateLeft, updateRight, val int) {
  if len(sat.data) > 0 {
    sat.updateInTree(0, 0, len(sat.data)-1, updateLeft, updateRight, val)
  }
}
```

```go
func (sat *SegmentAreaTree) updateInTree(treeIndex, left, right, updateLeft, updateRight, val int) {
  midTreeIndex, leftTreeIndex, rightTreeIndex := left+(right-left)>>1, sat.leftChild(treeIndex), sat.rightChild(treeIndex)
  if left > right || left >= updateRight || right <= updateLeft { // 由于叶子节点的区间不在是 left == right 所以这里判断需要增加等号的判断
    return // out of range. escape.
  }

  if updateLeft <= left && right <= updateRight { // segment is fully within update range
    if left == right-1 {
      sat.tree[treeIndex].count = sat.merge(sat.tree[treeIndex].count, val)
    }
    if left != right-1 { // update lazy[] for children
      sat.updateInTree(leftTreeIndex, left, midTreeIndex, updateLeft, updateRight, val)
      sat.updateInTree(rightTreeIndex, midTreeIndex, right, updateLeft, updateRight, val)
      sat.pushUp(treeIndex, leftTreeIndex, rightTreeIndex)
    }
    return
  }
  sat.updateInTree(leftTreeIndex, left, midTreeIndex, updateLeft, updateRight, val)
  sat.updateInTree(rightTreeIndex, midTreeIndex, right, updateLeft, updateRight, val)
  // merge updates
  sat.pushUp(treeIndex, leftTreeIndex, rightTreeIndex)
}
```

851. Loud and Rich

题目

In a group of N people (labelled `0, 1, 2, ..., N-1`), each person has different amounts of money, and different levels of quietness.

For convenience, we'll call the person with label `x`, simply "person `x`".

We'll say that `richer[i] = [x, y]` if person `x` definitely has more money than person `y`. Note that `richer` may only be a subset of valid observations.

Also, we'll say `quiet = q` if person x has quietness `q`.

Now, return `answer`, where `answer = y` if `y` is the least quiet person (that is, the person `y` with the smallest value of `quiet[y]`), among all people who definitely have equal to or more money than person `x`.

Example 1:

```
Input: richer = [[1,0],[2,1],[3,1],[3,7],[4,3],[5,3],[6,3]], quiet = [3,2,5,4,6,1,7,0]
Output: [5,5,2,5,4,5,6,7]
Explanation:
answer[0] = 5.
Person 5 has more money than 3, which has more money than 1, which has more money than
0.
The only person who is quieter (has lower quiet[x]) is person 7, but
it isn't clear if they have more money than person 0.

answer[7] = 7.
Among all people that definitely have equal to or more money than person 7
(which could be persons 3, 4, 5, 6, or 7), the person who is the quietest (has lower
quiet[x])
is person 7.

The other answers can be filled out with similar reasoning.
```

Note:

1. `1 <= quiet.length = N <= 500`
2. `0 <= quiet[i] < N`, all `quiet[i]` are different.
3. `0 <= richer.length <= N * (N-1) / 2`
4. `0 <= richer[i][j] < N`
5. `richer[i][0] != richer[i][1]`
6. `richer[i]`'s are all different.
7. The observations in `richer` are all logically consistent.

题目大意

在一组 N 个人（编号为 0, 1, 2, ..., N-1）中，每个人都有不同数目的钱，以及不同程度的安静（quietness）。为了方便起见，我们将编号为 x 的人简称为 "person x "。如果能够肯定 person x 比 person y 更有钱的话，我们会说 richer[i] = [x, y] 。注意 richer 可能只是有效观察的一个子集。另外，如果 person x 的安静程度为 q ，我们会说 quiet[x] = q 。现在，返回答案 answer ，其中 answer[x] = y 的前提是，在所有拥有的钱不少于 person x 的人中，person y 是最安静的人（也就是安静值 quiet[y] 最小的人）。

提示：

- 1 <= quiet.length = N <= 500
- 0 <= quiet[i] < N，所有 quiet[i] 都不相同。
- 0 <= richer.length <= N * (N-1) / 2
- 0 <= richer[i][j] < N
- richer[i][0] != richer[i][1]
- richer[i] 都是不同的。
- 对 richer 的观察在逻辑上是一致的。

解题思路

- 给出 2 个数组，richer 和 quiet，要求输出 answer，其中 answer = y 的前提是，在所有拥有的钱不少于 x 的人中，y 是最安静的人（也就是安静值 quiet[y] 最小的人）

- 由题意可知，`richer` 构成了一个有向无环图，首先使用字典建立图的关系，找到比当前下标编号富有的所有的人。然后使用广度优先层次遍历，不断的使用富有的人，但是安静值更小的人更新子节点即可。
- 这一题还可以用拓扑排序来解答。将 `richer` 中描述的关系看做边，如果 `x > y`，则 `x` 指向 `y`。将 `quiet` 看成权值。用一个数组记录答案，初始时 `ans[i] = i`。然后对原图做拓扑排序，对于每一条边，如果发现 `quiet[ans[v]] > quiet[ans[u]]`，则 `ans[v]` 的答案为 `ans[u]`。时间复杂度即为拓扑排序的时间复杂度为 `O(m+n)`。空间复杂度需要 `O(m)` 的数组建图，需要 `O(n)` 的数组记录入度以及存储队列，所以空间复杂度为 `O(m+n)`。

代码

```go
func loudAndRich(richer [][]int, quiet []int) []int {
  edges := make([][]int, len(quiet))
  for i := range edges {
    edges[i] = []int{}
  }
  indegrees := make([]int, len(quiet))
  for _, edge := range richer {
    n1, n2 := edge[0], edge[1]
    edges[n1] = append(edges[n1], n2)
    indegrees[n2]++
  }
  res := make([]int, len(quiet))
  for i := range res {
    res[i] = i
  }
  queue := []int{}
  for i, v := range indegrees {
    if v == 0 {
      queue = append(queue, i)
    }
  }
  for len(queue) > 0 {
    nexts := []int{}
    for _, n1 := range queue {
      for _, n2 := range edges[n1] {
        indegrees[n2]--
        if quiet[res[n2]] > quiet[res[n1]] {
          res[n2] = res[n1]
        }
        if indegrees[n2] == 0 {
          nexts = append(nexts, n2)
        }
      }
    }
    queue = nexts
  }
  return res
}
```

852. Peak Index in a Mountain Array

题目

Let's call an array `A` a *mountain* if the following properties hold:

- `A.length >= 3`
- There exists some `0 < i < A.length - 1` such that `A[0] < A[1] < ... A[i-1] < A[i] > A[i+1] > ... > A[A.length - 1]`

Given an array that is definitely a mountain, return any `i` such that `A[0] < A[1] < ... A[i-1] < A[i] > A[i+1] > ... > A[A.length - 1]`.

Example 1:

```
Input: [0,1,0]
Output: 1
```

Example 2:

```
Input: [0,2,1,0]
Output: 1
```

Note:

1. `3 <= A.length <= 10000`
2. `0 <= A[i] <= 10^6`
3. A is a mountain, as defined above.

题目大意

我们把符合下列属性的数组 A 称作山脉：

- A.length >= 3
- 存在 0 < i < A.length - 1 使得A[0] < A[1] < ... A[i-1] < A[i] > A[i+1] > ... > A[A.length - 1]
  给定一个确定为山脉的数组，返回任何满足 A[0] < A[1] < ... A[i-1] < A[i] > A[i+1] > ... > A[A.length - 1] 的 i 的值。

提示：

- 3 <= A.length <= 10000
- 0 <= A[i] <= 10^6
- A 是如上定义的山脉

解题思路

- 给出一个数组，数组里面存在有且仅有一个“山峰”，(山峰的定义是，下标 `i` 比 `i-1`、`i+1` 位置上的元素都

要大)，找到这个“山峰”，并输出其中一个山峰的下标。
- 这一题直接用二分搜索即可，数组中的元素算基本有序。判断是否为山峰的条件为比较左右两个数，如果当前的数比左右两个数都大，即找到了山峰。其他的情况都在山坡上。这一题有两种写法，第一种写法是标准的二分写法，第二种写法是变形的二分写法。

代码

```
package leetcode

// 解法一 二分
func peakIndexInMountainArray(A []int) int {
  low, high := 0, len(A)-1
  for low <= high {
    mid := low + (high-low)>>1
    if A[mid] > A[mid+1] && A[mid] > A[mid-1] {
      return mid
    }
    if A[mid] > A[mid+1] && A[mid] < A[mid-1] {
      high = mid - 1
    }
    if A[mid] < A[mid+1] && A[mid] > A[mid-1] {
      low = mid + 1
    }
  }
  return 0
}

// 解法二 二分
func peakIndexInMountainArray1(A []int) int {
  low, high := 0, len(A)-1
  for low < high {
    mid := low + (high-low)>>1
    // 如果 mid 较大，则左侧存在峰值，high = m，如果 mid + 1 较大，则右侧存在峰值，low = mid + 1
    if A[mid] > A[mid+1] {
      high = mid
    } else {
      low = mid + 1
    }
  }
  return low
}
```

853. Car Fleet

题目

`N` cars are going to the same destination along a one lane road. The destination is `target` miles away.

Each car `i` has a constant speed `speed[i]` (in miles per hour), and initial position `position[i]` miles towards the target along the road.

A car can never pass another car ahead of it, but it can catch up to it, and drive bumper to bumper at the same speed.

The distance between these two cars is ignored - they are assumed to have the same position.

A *car fleet* is some non-empty set of cars driving at the same position and same speed. Note that a single car is also a car fleet.

If a car catches up to a car fleet right at the destination point, it will still be considered as one car fleet.

How many car fleets will arrive at the destination?

Example 1:

```
Input: target = 12, position = [10,8,0,5,3], speed = [2,4,1,1,3]
Output: 3
Explanation:
The cars starting at 10 and 8 become a fleet, meeting each other at 12.
The car starting at 0 doesn't catch up to any other car, so it is a fleet by itself.
The cars starting at 5 and 3 become a fleet, meeting each other at 6.
Note that no other cars meet these fleets before the destination, so the answer is 3.
```

Note:

1. `0 <= N <= 10 ^ 4`
2. `0 < target <= 10 ^ 6`
3. `0 < speed[i] <= 10 ^ 6`
4. `0 <= position[i] < target`
5. All initial positions are different.

题目大意

N 辆车沿着一条车道驶向位于 target 英里之外的共同目的地。每辆车 i 以恒定的速度 speed[i] （英里/小时），从初始位置 position[i] （英里） 沿车道驶向目的地。

一辆车永远不会超过前面的另一辆车，但它可以追上去，并与前车以相同的速度紧接着行驶。此时，我们会忽略这两辆车之间的距离，也就是说，它们被假定处于相同的位置。车队 是一些由行驶在相同位置、具有相同速度的车组成的非空集合。注意，一辆车也可以是一个车队。即便一辆车在目的地才赶上了一个车队，它们仍然会被视作是同一个车队。

问最后会有多少车队到达目的地?

解题思路

- 根据每辆车距离终点和速度，计算每辆车到达终点的时间，并按照距离从大到小排序(position 越大代表距离

终点越近)

- 从头往后扫描排序以后的数组，时间一旦大于前一个 car 的时间，就会生成一个新的 car fleet，最终计数加一即可。

代码

```
package leetcode

import (
  "sort"
)

type car struct {
  time     float64
  position int
}

// ByPosition define
type ByPosition []car

func (a ByPosition) Len() int           { return len(a) }
func (a ByPosition) Swap(i, j int)      { a[i], a[j] = a[j], a[i] }
func (a ByPosition) Less(i, j int) bool { return a[i].position > a[j].position }

func carFleet(target int, position []int, speed []int) int {
  n := len(position)
  if n <= 1 {
    return n
  }
  cars := make([]car, n)
  for i := 0; i < n; i++ {
    cars[i] = car{float64(target-position[i]) / float64(speed[i]), position[i]}
  }
  sort.Sort(ByPosition(cars))
  fleet, lastTime := 0, 0.0
  for i := 0; i < len(cars); i++ {
    if cars[i].time > lastTime {
      lastTime = cars[i].time
      fleet++
    }
  }
  return fleet
}
```

856. Score of Parentheses

题目

Given a balanced parentheses string S, compute the score of the string based on the following rule:

() has score 1
AB has score A + B, where A and B are balanced parentheses strings.
(A) has score 2 * A, where A is a balanced parentheses string.

Example 1:

```
Input: "()"
Output: 1
```

Example 2:

```
Input: "(())"
Output: 2
```

Example 3:

```
Input: "()()"
Output: 2
```

Example 4:

```
Input: "(()(()))"
Output: 6
```

Note:

1. S is a balanced parentheses string, containing only (and).
2. 2 <= S.length <= 50

题目大意

按照以下规则计算括号的分数：() 代表 1 分。AB 代表 A + B，A 和 B 分别是已经满足匹配规则的括号组。(A) 代表 2 * A，其中 A 也是已经满足匹配规则的括号组。给出一个括号字符串，要求按照这些规则计算出括号的分数值。

解题思路

按照括号匹配的原则，一步步的计算每个组合的分数入栈。遇到题目中的 3 种情况，取出栈顶元素算分数。

代码

```
package leetcode

func scoreOfParentheses(S string) int {
  res, stack, top, temp := 0, []int{}, -1, 0
  for _, s := range S {
    if s == '(' {
      stack = append(stack, -1)
      top++
    } else {
      temp = 0
      for stack[top] != -1 {
        temp += stack[top]
        stack = stack[:len(stack)-1]
        top--
      }
      stack = stack[:len(stack)-1]
      top--
      if temp == 0 {
        stack = append(stack, 1)
        top++
      } else {
        stack = append(stack, temp*2)
        top++
      }
    }
  }
  for len(stack) != 0 {
    res += stack[top]
    stack = stack[:len(stack)-1]
    top--
  }
  return res
}
```

862. Shortest Subarray with Sum at Least K

题目

Return the **length** of the shortest, non-empty, contiguous subarray of `A` with sum at least `K`.

If there is no non-empty subarray with sum at least `K`, return `-1`.

Example 1:

```
Input: A = [1], K = 1
Output: 1
```

Example 2:

```
Input: A = [1,2], K = 4
Output: -1
```

Example 3:

```
Input: A = [2,-1,2], K = 3
Output: 3
```

Note:

1. `1 <= A.length <= 50000`
2. `-10 ^ 5 <= A[i] <= 10 ^ 5`
3. `1 <= K <= 10 ^ 9`

题目大意

返回 A 的最短的非空连续子数组的长度，该子数组的和至少为 K 。如果没有和至少为 K 的非空子数组，返回 -1 。

提示：

- 1 <= A.length <= 50000
- -10 ^ 5 <= A[i] <= 10 ^ 5
- 1 <= K <= 10 ^ 9

解题思路

- 给出一个数组，要求找出一个最短的，非空的，连续的子序列且累加和至少为 k 。
- 由于给的数组里面可能存在负数，所以子数组的累加和不会随着数组的长度增加而增加。题目中要求区间和，理所应当需要利用 `prefixSum` 前缀和，先计算出 `prefixSum` 前缀和。
- 简化一下题目的要求，即能否找到 `prefixSum[y] - prefixSum[x] ≥ K` ，且 `y - x` 的差值最小。如果固定的 `y`，那么对于 `x`，`x` 越大，`y - x` 的差值就越小(因为 `x` 越逼近 `y`)。所以想求区间 `[x, y]` 的最短距离，需要保证 `y` 尽量小，`x` 尽量大，这样 `[x, y]` 区间距离最小。那么有以下 2 点“常识”一定成立：
    1. 如果 `x1 < x2` ，并且 `prefixSum[x2] ≤ prefixSum[x1]`，说明结果一定不会取 `x1`。因为如果 `prefixSum[x1] ≤ prefixSum[y] - k`，那么 `prefixSum[x2] ≤ prefixSum[x1] ≤ prefixSum[y] - k`，`x2` 也能满足题意，并且 `x2` 比 `x1` 更加接近 `y`，最优解一定优先考虑 `x2`。
    2. 在确定了 `x` 以后，以后就不用再考虑 `x` 了，因为如果 `y2 > y1`，且 `y2` 的时候取 `x` 还是一样的，那么算距离的话，`y2 - x` 显然大于 `y1 - x`，这样的话肯定不会是最短的距离。

- 从上面这两个常识来看，可以用双端队列 `deque` 来处理 `prefixSum`。`deque` 中存储的是递增的 `x` 下标，为了满足常识一。从双端队列的开头开始遍历，假如区间和之差大于等于 `K`，就移除队首元素并更新结果 `res`。队首移除元素，直到不满足 `prefixSum[i]-prefixSum[deque[0]] >= K` 这一不等式，是为了满足常识二。之后的循环是此题的精髓，从双端队列的末尾开始往前遍历，假如当前区间和 `prefixSum[i]` 小于等于队列末尾的区间和，则移除队列末尾元素。为什么这样处理呢？因为若数组都是正数，那么长度越长，区间和一定越大，则 `prefixSum[i]` 一定大于所有双端队列中的区间和，但由于可能存在负数，从而使得长度变长，区间总和反而减少了，之前的区间和之差值都没有大于等于 `K`(< K)，现在的更不可能大于等于 `K`，这个结束位置可以直接淘汰，不用进行计算。循环结束后将当前位置加入双端队列即可。遇到新下标在队尾移除若干元素，这一行为，也是为了满足常识一。
- 由于所有下标都只进队列一次，也最多 pop 出去一次，所以时间复杂度 O(n)，空间复杂度 O(n)。

代码

```go
func shortestSubarray(A []int, K int) int {
  res, prefixSum := len(A)+1, make([]int, len(A)+1)
  for i := 0; i < len(A); i++ {
    prefixSum[i+1] = prefixSum[i] + A[i]
  }
  // deque 中保存递增的 prefixSum 下标
  deque := []int{}
  for i := range prefixSum {
    // 下面这个循环希望能找到 [deque[0], i] 区间内累加和 >= K，如果找到了就更新答案
    for len(deque) > 0 && prefixSum[i]-prefixSum[deque[0]] >= K {
      length := i - deque[0]
      if res > length {
        res = length
      }
      // 找到第一个 deque[0] 能满足条件以后，就移除它，因为它是最短长度的子序列了
      deque = deque[1:]
    }
    // 下面这个循环希望能保证 prefixSum[deque[i]] 递增
    for len(deque) > 0 && prefixSum[i] <= prefixSum[deque[len(deque)-1]] {
      deque = deque[:len(deque)-1]
    }
    deque = append(deque, i)
  }
  if res <= len(A) {
    return res
  }
  return -1
}
```

863. All Nodes Distance K in Binary Tree

题目

We are given a binary tree (with root node `root`), a `target` node, and an integer value `K`.

Return a list of the values of all nodes that have a distance `K` from the `target` node. The answer can be returned in any order.

Example 1:

```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], target = 5, K = 2

Output: [7,4,1]

Explanation:
The nodes that are a distance 2 from the target node (with value 5)
have values 7, 4, and 1.
```

Note:

1. The given tree is non-empty.
2. Each node in the tree has unique values `0 <= node.val <= 500`.
3. The `target` node is a node in the tree.
4. `0 <= K <= 1000`.

题目大意

给定一个二叉树（具有根结点 root）， 一个目标结点 target ， 和一个整数值 K 。返回到目标结点 target 距离为 K 的所有结点的值的列表。 答案可以以任何顺序返回。

提示：

- 给定的树是非空的。
- 树上的每个结点都具有唯一的值 0 <= node.val <= 500 。
- 目标结点 target 是树上的结点。
- 0 <= K <= 1000.

解题思路

- 给出一颗树和一个目标节点 target，一个距离 K，要求找到所有距离目标节点 target 的距离是 K 的点。
- 这一题用 DFS 的方法解题。先找到当前节点距离目标节点的距离，如果在左子树中找到了 target，距离当前节点的距离 > 0，则还需要在它的右子树中查找剩下的距离。如果是在右子树中找到了 target，反之同理。如果当前节点就是目标节点，那么就可以直接记录这个点。否则每次遍历一个点，距离都减一。

代码

```go
func distanceK(root *TreeNode, target *TreeNode, K int) []int {
  visit := []int{}
  findDistanceK(root, target, K, &visit)
  return visit
```

```go
}

func findDistanceK(root, target *TreeNode, K int, visit *[]int) int {
  if root == nil {
    return -1
  }
  if root == target {
    findChild(root, K, visit)
    return K - 1
  }
  leftDistance := findDistanceK(root.Left, target, K, visit)
  if leftDistance == 0 {
    findChild(root, leftDistance, visit)
  }
  if leftDistance > 0 {
    findChild(root.Right, leftDistance-1, visit)
    return leftDistance - 1
  }
  rightDistance := findDistanceK(root.Right, target, K, visit)
  if rightDistance == 0 {
    findChild(root, rightDistance, visit)
  }
  if rightDistance > 0 {
    findChild(root.Left, rightDistance-1, visit)
    return rightDistance - 1
  }
  return -1
}

func findChild(root *TreeNode, K int, visit *[]int) {
  if root == nil {
    return
  }
  if K == 0 {
    *visit = append(*visit, root.Val)
  } else {
    findChild(root.Left, K-1, visit)
    findChild(root.Right, K-1, visit)
  }
}
```

864. Shortest Path to Get All Keys

题目

We are given a 2-dimensional `grid`. `"."` is an empty cell, `"#"` is a wall, `"@"` is the starting point, (`"a"`, `"b"`, ...) are keys, and (`"A"`, `"B"`, ...) are locks.

We start at the starting point, and one move consists of walking one space in one of the 4 cardinal directions. We cannot walk outside the grid, or walk into a wall. If we walk over a key, we pick it up. We can't walk over a lock unless we have the corresponding key.

For some 1 <= K <= 6, there is exactly one lowercase and one uppercase letter of the first `K` letters of the English alphabet in the grid. This means that there is exactly one key for each lock, and one lock for each key; and also that the letters used to represent the keys and locks were chosen in the same order as the English alphabet.

Return the lowest number of moves to acquire all keys. If it's impossible, return `-1`.

Example 1:

```
Input: ["@.a.#","###.#","b.A.B"]
Output: 8
```

Example 2:

```
Input: ["@..aA","..B#.","....b"]
Output: 6
```

Note:

1. `1 <= grid.length <= 30`
2. `1 <= grid[0].length <= 30`
3. `grid[i][j]` contains only `'.'`, `'#'`, `'@'`, `'a'-'f'` and `'A'-'F'`
4. The number of keys is in `[1, 6]`. Each key has a different letter and opens exactly one lock.

题目大意

给定一个二维网格 grid。 "." 代表一个空房间， "#" 代表一堵墙， "@" 是起点， （"a", "b", ...）代表钥匙， （"A", "B", ...）代表锁。

我们从起点开始出发，一次移动是指向四个基本方向之一行走一个单位空间。我们不能在网格外面行走，也无法穿过一堵墙。如果途经一个钥匙，我们就把它捡起来。除非我们手里有对应的钥匙，否则无法通过锁。

假设 K 为钥匙/锁的个数，且满足 1 <= K <= 6，字母表中的前 K 个字母在网格中都有自己对应的一个小写和一个大写字母。换言之，每个锁有唯一对应的钥匙，每个钥匙也有唯一对应的锁。另外，代表钥匙和锁的字母互为大小写并按字母顺序排列。

返回获取所有钥匙所需要的移动的最少次数。如果无法获取所有钥匙，返回 -1 。

提示：

1. 1 <= grid.length <= 30
2. 1 <= grid[0].length <= 30
3. grid[i][j] 只含有 '.', '#', '@', 'a'-'f' 以及 'A'-'F'
4. 钥匙的数目范围是 [1, 6]，每个钥匙都对应一个不同的字母，正好打开一个对应的锁。

解题思路

- 给出一个地图，在图中有钥匙和锁，当锁在没有钥匙的时候不能通行，问从起点 @ 开始，到最终获得所有钥匙，最短需要走多少步。
- 这一题可以用 BFS 来解答。由于钥匙的种类比较多，所以 visited 数组需要 3 个维度，一个是 x 坐标，一个是 y 坐标，最后一个是当前获取钥匙的状态。每把钥匙都有获取了和没有获取两种状态，题目中说最多有 6 把钥匙，那么排列组合最多是 2^6 = 64 种状态。用一个十进制数的二进制位来压缩这些状态，二进制位分别来表示这些钥匙是否已经获取了。既然钥匙的状态可以压缩，其实 x 和 y 的坐标也可以一并压缩到这个数中。BFS 中存的数字是坐标 + 钥匙状态的状态。在 BFS 遍历的过程中，用 visited 数组来过滤遍历过的情况，来保证走的路是最短的。其他的情况无非是判断锁的状态，是否能通过，判断钥匙获取状态。
- 这一题不知道是否能用 DFS 来解答。我实现了一版，但是在 18 / 35 这组 case 上超时了，具体 case 见测试文件第一个 case。

代码

```
package leetcode

import (
  "math"
  "strings"
)

// 解法一 BFS，利用状态压缩来过滤筛选状态
func shortestPathAllKeys(grid []string) int {
  if len(grid) == 0 {
    return 0
  }
  board, visited, startx, starty, res, fullKeys := make([][]byte, len(grid)), make([][]
[]bool, len(grid)), 0, 0, 0, 0
  for i := 0; i < len(grid); i++ {
    board[i] = make([]byte, len(grid[0]))
  }
  for i, g := range grid {
    board[i] = []byte(g)
    for _, v := range g {
      if v == 'a' || v == 'b' || v == 'c' || v == 'd' || v == 'e' || v == 'f' {
        fullKeys |= (1 << uint(v-'a'))
      }
    }
    if strings.Contains(g, "@") {
      startx, starty = i, strings.Index(g, "@")
    }
  }
  for i := 0; i < len(visited); i++ {
    visited[i] = make([][]bool, len(board[0]))
  }
  for i := 0; i < len(board); i++ {
    for j := 0; j < len(board[0]); j++ {
      visited[i][j] = make([]bool, 64)
```

```go
    }
  }
  queue := []int{}
  queue = append(queue, (starty<<16)|(startx<<8))
  visited[startx][starty][0] = true
  for len(queue) != 0 {
    qLen := len(queue)
    for i := 0; i < qLen; i++ {
      state := queue[0]
      queue = queue[1:]
      starty, startx = state>>16, (state>>8)&0xFF
      keys := state & 0xFF
      if keys == fullKeys {
        return res
      }
      for i := 0; i < 4; i++ {
        newState := keys
        nx := startx + dir[i][0]
        ny := starty + dir[i][1]
        if !isInBoard(board, nx, ny) {
          continue
        }
        if board[nx][ny] == '#' {
          continue
        }
        flag, canThroughLock := keys&(1<<(board[nx][ny]-'A')), false
        if flag != 0 {
          canThroughLock = true
        }
        if isLock(board, nx, ny) && !canThroughLock {
          continue
        }
        if isKey(board, nx, ny) {
          newState |= (1 << (board[nx][ny] - 'a'))
        }
        if visited[nx][ny][newState] {
          continue
        }
        queue = append(queue, (ny<<16)|(nx<<8)|newState)
        visited[nx][ny][newState] = true
      }
    }
    res++
  }
  return -1
}

// 解法二 DFS，但是超时了，剪枝条件不够强
func shortestPathAllKeys1(grid []string) int {
```

```go
  if len(grid) == 0 {
    return 0
  }
  board, visited, startx, starty, res, fullKeys := make([][]byte, len(grid)), make([][]
[]bool, len(grid)), 0, 0, math.MaxInt64, 0
  for i := 0; i < len(grid); i++ {
    board[i] = make([]byte, len(grid[0]))
  }
  for i, g := range grid {
    board[i] = []byte(g)
    for _, v := range g {
      if v == 'a' || v == 'b' || v == 'c' || v == 'd' || v == 'e' || v == 'f' {
        fullKeys |= (1 << uint(v-'a'))
      }
    }
    if strings.Contains(g, "@") {
      startx, starty = i, strings.Index(g, "@")
    }
  }
  for i := 0; i < len(visited); i++ {
    visited[i] = make([][]bool, len(board[0]))
  }
  for i := 0; i < len(board); i++ {
    for j := 0; j < len(board[0]); j++ {
      visited[i][j] = make([]bool, 64)
    }
  }
  searchKeys(board, &visited, fullKeys, 0, (starty<<16)|(startx<<8), &res, []int{})
  if res == math.MaxInt64 {
    return -1
  }
  return res - 1
}

func searchKeys(board [][]byte, visited *[][][]bool, fullKeys, step, state int, res
*int, path []int) {
  y, x := state>>16, (state>>8)&0xFF
  keys := state & 0xFF

  if keys == fullKeys {
    *res = min(*res, step)
    return
  }

  flag, canThroughLock := keys&(1<<(board[x][y]-'A')), false
  if flag != 0 {
    canThroughLock = true
  }
  newState := keys
```

```go
  //fmt.Printf("x = %v y = %v fullKeys = %v keys = %v step = %v res = %v path = %v
state = %v\n", x, y, fullKeys, keys, step, *res, path, state)
  if (board[x][y] != '#' && !isLock(board, x, y)) || (isLock(board, x, y) &&
canThroughLock) {
    if isKey(board, x, y) {
      newState |= (1 << uint(board[x][y]-'a'))
    }
    (*visited)[x][y][newState] = true
    path = append(path, x)
    path = append(path, y)

    for i := 0; i < 4; i++ {
      nx := x + dir[i][0]
      ny := y + dir[i][1]
      if isInBoard(board, nx, ny) && !(*visited)[nx][ny][newState] {
        searchKeys(board, visited, fullKeys, step+1, (ny<<16)|(nx<<8)|newState, res,
path)
      }
    }
    (*visited)[x][y][keys] = false
    path = path[:len(path)-1]
    path = path[:len(path)-1]
  }
}

func isLock(board [][]byte, x, y int) bool {
  if (board[x][y] == 'A') || (board[x][y] == 'B') ||
    (board[x][y] == 'C') || (board[x][y] == 'D') ||
    (board[x][y] == 'E') || (board[x][y] == 'F') {
    return true
  }
  return false
}

func isKey(board [][]byte, x, y int) bool {
  if (board[x][y] == 'a') || (board[x][y] == 'b') ||
    (board[x][y] == 'c') || (board[x][y] == 'd') ||
    (board[x][y] == 'e') || (board[x][y] == 'f') {
    return true
  }
  return false
}
```

867. Transpose Matrix

题目

Given a matrix `A`, return the transpose of `A`.

The transpose of a matrix is the matrix flipped over it's main diagonal, switching the row and column indices of the matrix.

Example 1:

```
Input: [[1,2,3],[4,5,6],[7,8,9]]
Output: [[1,4,7],[2,5,8],[3,6,9]]
```

Example 2:

```
Input: [[1,2,3],[4,5,6]]
Output: [[1,4],[2,5],[3,6]]
```

Note:

1. `1 <= A.length <= 1000`
2. `1 <= A[0].length <= 1000`

题目大意

给定一个矩阵 A， 返回 A 的转置矩阵。矩阵的转置是指将矩阵的主对角线翻转，交换矩阵的行索引与列索引。

解题思路

- 给出一个矩阵，顺时针旋转 90°
- 解题思路很简单，直接模拟即可。

代码

```go
package leetcode

func transpose(A [][]int) [][]int {
  row, col, result := len(A), len(A[0]), make([][]int, len(A[0]))
  for i := range result {
    result[i] = make([]int, row)
  }
  for i := 0; i < row; i++ {
    for j := 0; j < col; j++ {
      result[j][i] = A[i][j]
    }
  }
  return result
}
```

869. Reordered Power of 2

题目

Starting with a positive integer `N`, we reorder the digits in any order (including the original order) such that the leading digit is not zero.

Return `true` if and only if we can do this in a way such that the resulting number is a power of 2.

Example 1:

```
Input:1
Output:true
```

Example 2:

```
Input:10
Output:false
```

Example 3:

```
Input:16
Output:true
```

Example 4:

```
Input:24
Output:false
```

Example 5:

```
Input:46
Output:true
```

Note:

1. `1 <= N <= 10^9`

题目大意

给定正整数 N ，我们按任何顺序（包括原始顺序）将数字重新排序，注意其前导数字不能为零。如果我们可以通过上述方式得到 2 的幂，返回 true；否则，返回 false。

解题思路

- 将整数每个位上的所有排列看成字符串，那么题目转换为判断这些字符串是否和 2 的幂的字符串是否一致。判断的方法有很多种，笔者这里判断借助了一个 `map`。两个不同排列的字符串要相等，所有字符出现的频次

必定一样。利用一个 `map` 统计它们各自字符的频次，最终都一致，则判定这两个字符串是满足题意的。
- 此题数据量比较小，在 `[1,10^9]` 这个区间内，2 的幂只有 30 几个，所以最终要判断的字符串就是这 30 几个。笔者这里没有打表了，采用更加一般的做法。数据量更大，此解法代码也能通过。

代码

```
package leetcode

import "fmt"

func reorderedPowerOf2(n int) bool {
  sample, i := fmt.Sprintf("%v", n), 1
  for len(fmt.Sprintf("%v", i)) <= len(sample) {
    t := fmt.Sprintf("%v", i)
    if len(t) == len(sample) && isSame(t, sample) {
      return true
    }
    i = i << 1
  }
  return false
}

func isSame(t, s string) bool {
  m := make(map[rune]int)
  for _, v := range t {
    m[v]++
  }
  for _, v := range s {
    m[v]--
    if m[v] < 0 {
      return false
    }
    if m[v] == 0 {
      delete(m, v)
    }
  }
  return len(m) == 0
}
```

870. Advantage Shuffle

题目

Given two arrays `A` and `B` of equal size, the *advantage of `A` with respect to `B`* is the number of indices `i` for which `A[i] > B[i]`.

Return **any** permutation of `A` that maximizes its advantage with respect to `B`.

Example 1:

```
Input:A = [2,7,11,15], B = [1,10,4,11]
Output:[2,11,7,15]
```

Example 2:

```
Input:A = [12,24,8,32], B = [13,25,32,11]
Output:[24,32,8,12]
```

Note:

1. `1 <= A.length = B.length <= 10000`
2. `0 <= A[i] <= 10^9`
3. `0 <= B[i] <= 10^9`

题目大意

给定两个大小相等的数组 A 和 B，A 相对于 B 的优势可以用满足 A[i] > B[i] 的索引 i 的数目来描述。返回 A 的任意排列，使其相对于 B 的优势最大化。

解题思路

- 此题用贪心算法解题。如果 A 中最小的牌 a 能击败 B 中最小的牌 b，那么将它们配对。否则， a 将无益于我们的比分，因为它无法击败任何牌。这是贪心的策略，每次匹配都用手中最弱的牌和 B 中的最小牌 b 进行配对，这样会使 A 中剩余的牌严格的变大，最后会使得得分更多。
- 在代码实现中，将 A 数组排序，B 数组按照下标排序。因为最终输出的是相对于 B 的优势结果，所以要针对 B 的下标不变来安排 A 的排列。排好序以后按照贪心策略选择 A 中牌的顺序。

代码

```go
package leetcode

import "sort"

func advantageCount1(A []int, B []int) []int {
  n := len(A)
  sort.Ints(A)
  sortedB := make([]int, n)
  for i := range sortedB {
    sortedB[i] = i
  }
  sort.Slice(sortedB, func(i, j int) bool {
    return B[sortedB[i]] < B[sortedB[j]]
  })
  useless, i, res := make([]int, 0), 0, make([]int, n)
  for _, index := range sortedB {
    b := B[index]
```

```
    for i < n && A[i] <= b {
      useless = append(useless, A[i])
      i++
    }
    if i < n {
      res[index] = A[i]
      i++
    } else {
      res[index] = useless[0]
      useless = useless[1:]
    }
  }
  return res
}
```

872. Leaf-Similar Trees

题目

Consider all the leaves of a binary tree. From left to right order, the values of those leaves form a *leaf value sequence.*

For example, in the given tree above, the leaf value sequence is `(6, 7, 4, 9, 8)`.

Two binary trees are considered *leaf-similar* if their leaf value sequence is the same.

Return `true` if and only if the two given trees with head nodes `root1` and `root2` are leaf-similar.

Note:

- Both of the given trees will have between `1` and `100` nodes.

题目大意

请考虑一颗二叉树上所有的叶子，这些叶子的值按从左到右的顺序排列形成一个 叶值序列 。举个例子，如上图所示，给定一颗叶值序列为 (6, 7, 4, 9, 8) 的树。如果有两颗二叉树的叶值序列是相同，那么我们就认为它们是 叶相似 的。如果给定的两个头结点分别为 root1 和 root2 的树是叶相似的，则返回 true；否则返回 false 。

提示：

- 给定的两颗树可能会有 1 到 200 个结点。
- 给定的两颗树上的值介于 0 到 200 之间。

解题思路

- 给出 2 棵树，如果 2 棵树的叶子节点组成的数组是完全一样的，那么就认为这 2 棵树是“叶子相似”的。给出任何 2 棵树判断这 2 棵树是否是“叶子相似”的。
- 简单题，分别 DFS 遍历 2 棵树，把叶子节点都遍历出来，然后分别比较叶子节点组成的数组是否完全一致即

可。

代码

```
func leafSimilar(root1 *TreeNode, root2 *TreeNode) bool {
  leaf1, leaf2 := []int{}, []int{}
  dfsLeaf(root1, &leaf1)
  dfsLeaf(root2, &leaf2)
  if len(leaf1) != len(leaf2) {
    return false
  }
  for i := range leaf1 {
    if leaf1[i] != leaf2[i] {
      return false
    }
  }
  return true
}

func dfsLeaf(root *TreeNode, leaf *[]int) {
  if root != nil {
    if root.Left == nil && root.Right == nil {
      *leaf = append(*leaf, root.Val)
    }
    dfsLeaf(root.Left, leaf)
    dfsLeaf(root.Right, leaf)
  }
}
```

874. Walking Robot Simulation

题目

A robot on an infinite XY-plane starts at point `(0, 0)` and faces north. The robot can receive one of three possible types of `commands`:

- `2`: turn left `90` degrees,
- `1`: turn right `90` degrees, or
- `1 <= k <= 9`: move forward `k` units.

Some of the grid squares are `obstacles`. The `ith` obstacle is at grid point `obstacles[i] = (xi, yi)`.

If the robot would try to move onto them, the robot stays on the previous grid square instead (but still continues following the rest of the route.)

Return *the maximum Euclidean distance that the robot will be from the origin* ***squared*** *(i.e. if the distance is* `5`*, return* `25`*).*

Note:

- North means +Y direction.
- East means +X direction.
- South means -Y direction.
- West means -X direction.

Example 1:

```
Input: commands = [4,-1,3], obstacles = []
Output: 25
Explanation: The robot starts at (0, 0):
1. Move north 4 units to (0, 4).
2. Turn right.
3. Move east 3 units to (3, 4).
The furthest point away from the origin is (3, 4), which is 32 + 42 = 25 units away.
```

Example 2:

```
Input: commands = [4,-1,4,-2,4], obstacles = [[2,4]]
Output: 65
Explanation: The robot starts at (0, 0):
1. Move north 4 units to (0, 4).
2. Turn right.
3. Move east 1 unit and get blocked by the obstacle at (2, 4), robot is at (1, 4).
4. Turn left.
5. Move north 4 units to (1, 8).
The furthest point away from the origin is (1, 8), which is 12 + 82 = 65 units away.
```

Constraints:

- `1 <= commands.length <= 104`
- `commands[i]` is one of the values in the list `[-2,-1,1,2,3,4,5,6,7,8,9]`.
- `0 <= obstacles.length <= 104`
- `3 * 104 <= xi, yi <= 3 * 104`
- The answer is guaranteed to be less than `231`.

题目大意

机器人在一个无限大小的 XY 网格平面上行走，从点 (0, 0) 处开始出发，面向北方。该机器人可以接收以下三种类型的命令 commands：

- 2： 向左转 90 度
- -1： 向右转 90 度
- 1 <= x <= 9： 向前移动 x 个单位长度

在网格上有一些格子被视为障碍物 obstacles 。第 i 个障碍物位于网格点 obstacles[i] = (xi, yi) 。机器人无法走到障碍物上，它将会停留在障碍物的前一个网格方块上，但仍然可以继续尝试进行该路线的其余部分。返回从原点到机器人所有经过的路径点（坐标为整数）的最大欧式距离的平方。（即，如果距离为 5 ，则返回 25 ）

注意：

- 北表示 +Y 方向。
- 东表示 +X 方向。
- 南表示 -Y 方向。
- 西表示 -X 方向。

解题思路

- 这个题的难点在于，怎么用编程语言去描述机器人的行为，可以用以下数据结构表达机器人的行为：

```
direct:= 0                    // direct表示机器人移动方向：0 1 2 3 4 （北东南西），默认朝北
x, y := 0, 0                  // 表示当前机器人所在横纵坐标位置，默认为(0,0)
directX := []int{0, 1, 0, -1}
directY := []int{1, 0, -1, 0}
// 组合directX directY和direct，表示机器人往某一个方向移动
nextX := x + directX[direct]
nextY := y + directY[direct]
```

其他代码按照题意翻译即可

代码

```
package leetcode

func robotSim(commands []int, obstacles [][]int) int {
  m := make(map[[2]int]struct{})
  for _, v := range obstacles {
    if len(v) != 0 {
      m[[2]int{v[0], v[1]}] = struct{}{}
    }
  }
  directX := []int{0, 1, 0, -1}
  directY := []int{1, 0, -1, 0}
  direct, x, y := 0, 0, 0
  result := 0
  for _, c := range commands {
    if c == -2 {
      direct = (direct + 3) % 4
      continue
    }
    if c == -1 {
      direct = (direct + 1) % 4
      continue
    }
    for ; c > 0; c-- {
      nextX := x + directX[direct]
      nextY := y + directY[direct]
```

```
      if _, ok := m[[2]int{nextX, nextY}]; ok {
        break
      }
      tmpResult := nextX*nextX + nextY*nextY
      if tmpResult > result {
        result = tmpResult
      }
      x = nextX
      y = nextY
    }
  }
  return result
}
```

875. Koko Eating Bananas

题目

Koko loves to eat bananas. There are `N` piles of bananas, the `i`-th pile has `piles[i]` bananas. The guards have gone and will come back in `H` hours.

Koko can decide her bananas-per-hour eating speed of `K`. Each hour, she chooses some pile of bananas, and eats K bananas from that pile. If the pile has less than `K` bananas, she eats all of them instead, and won't eat any more bananas during this hour.

Koko likes to eat slowly, but still wants to finish eating all the bananas before the guards come back.

Return the minimum integer `K` such that she can eat all the bananas within `H` hours.

Example 1:

```
Input: piles = [3,6,7,11], H = 8
Output: 4
```

Example 2:

```
Input: piles = [30,11,23,4,20], H = 5
Output: 30
```

Example 3:

```
Input: piles = [30,11,23,4,20], H = 6
Output: 23
```

Note:

- `1 <= piles.length <= 10^4`
- `piles.length <= H <= 10^9`

- `1 <= piles[i] <= 10^9`

题目大意

珂珂喜欢吃香蕉。这里有 N 堆香蕉，第 i 堆中有 piles[i] 根香蕉。警卫已经离开了，将在 H 小时后回来。

珂珂可以决定她吃香蕉的速度 K （单位：根/小时）。每个小时，她将会选择一堆香蕉，从中吃掉 K 根。如果这堆香蕉少于 K 根，她将吃掉这堆的所有香蕉，然后这一小时内不会再吃更多的香蕉。

珂珂喜欢慢慢吃，但仍然想在警卫回来前吃掉所有的香蕉。

返回她可以在 H 小时内吃掉所有香蕉的最小速度 K（K 为整数）。

提示：

- 1 <= piles.length <= 10^4
- piles.length <= H <= 10^9
- 1 <= piles[i] <= 10^9

解题思路

- 给出一个数组，数组里面每个元素代表的是每个香蕉🍌串上香蕉的个数。koko 以 `k 个香蕉/小时` 的速度吃这些香蕉。守卫会在 `H 小时` 以后回来。问 k 至少为多少，能在守卫回来之前吃完所有的香蕉。当香蕉的个数小于 k 的时候，这个小时只能吃完这些香蕉，不能再吃其他串上的香蕉了。
- 这一题可以用二分搜索来解答。在 `[0 , max(piles)]` 的范围内搜索，二分的过程都是常规思路。判断是否左右边界如果划分的时候需要注意题目中给的限定条件。当香蕉个数小于 k 的时候，那个小时不能再吃其他香蕉了。

代码

```go
package leetcode

import "math"

func minEatingSpeed(piles []int, H int) int {
  low, high := 1, maxInArr(piles)
  for low < high {
    mid := low + (high-low)>>1
    if !isPossible(piles, mid, H) {
      low = mid + 1
    } else {
      high = mid
    }
  }
  return low
}
```

```go
func isPossible(piles []int, h, H int) bool {
  res := 0
  for _, p := range piles {
    res += int(math.Ceil(float64(p) / float64(h)))
  }
  return res <= H
}

func maxInArr(xs []int) int {
  res := 0
  for _, x := range xs {
    if res < x {
      res = x
    }
  }
  return res
}
```

876. Middle of the Linked List

题目

Given a non-empty, singly linked list with head node head, return a middle node of linked list.

If there are two middle nodes, return the second middle node.

Example 1:

```
Input: [1,2,3,4,5]
Output: Node 3 from this list (Serialization: [3,4,5])
The returned node has value 3.  (The judge's serialization of this node is [3,4,5]).
Note that we returned a ListNode object ans, such that:
ans.val = 3, ans.next.val = 4, ans.next.next.val = 5, and ans.next.next.next = NULL.
```

Example 2:

```
Input: [1,2,3,4,5,6]
Output: Node 4 from this list (Serialization: [4,5,6])
Since the list has two middle nodes with values 3 and 4, we return the second one.
```

Note:

- The number of nodes in the given list will be between 1 and 100.

题目大意

输出链表中间结点。这题在前面题目中反复出现了很多次了。

如果链表长度是奇数，输出中间结点是中间结点。如果链表长度是双数，输出中间结点是中位数后面的那个结点。

解题思路

这道题有一个很简单的做法，用 2 个指针只遍历一次就可以找到中间节点。一个指针每次移动 2 步，另外一个指针每次移动 1 步，当快的指针走到终点的时候，慢的指针就是中间节点。

代码

```go
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */

func middleNode(head *ListNode) *ListNode {
  if head == nil || head.Next == nil {
    return head
  }
  p1 := head
  p2 := head
  for p2.Next != nil && p2.Next.Next != nil {
    p1 = p1.Next
    p2 = p2.Next.Next
  }
  length := 0
  cur := head
  for cur != nil {
    length++
    cur = cur.Next
  }
  if length%2 == 0 {
    return p1.Next
  }
  return p1
}
```

877. Stone Game

题目

Alex and Lee play a game with piles of stones. There are an even number of piles **arranged in a row**, and each pile has a positive integer number of stones `piles[i]`.

The objective of the game is to end with the most stones. The total number of stones is odd, so there are no ties.

Alex and Lee take turns, with Alex starting first. Each turn, a player takes the entire pile of stones from either the beginning or the end of the row. This continues until there are no more piles left, at which point the person with the most stones wins.

Assuming Alex and Lee play optimally, return `True` if and only if Alex wins the game.

Example 1:

```
Input: piles = [5,3,4,5]
Output: true
Explanation:
Alex starts first, and can only take the first 5 or the last 5.
Say he takes the first 5, so that the row becomes [3, 4, 5].
If Lee takes 3, then the board is [4, 5], and Alex takes 5 to win with 10 points.
If Lee takes the last 5, then the board is [3, 4], and Alex takes 4 to win with 9
points.
This demonstrated that taking the first 5 was a winning move for Alex, so we return
true.
```

Constraints:

- `2 <= piles.length <= 500`
- `piles.length` is even.
- `1 <= piles[i] <= 500`
- `sum(piles)` is odd.

题目大意

亚历克斯和李用几堆石子在做游戏。偶数堆石子排成一行，每堆都有正整数颗石子 piles[i] 。游戏以谁手中的石子最多来决出胜负。石子的总数是奇数，所以没有平局。亚历克斯和李轮流进行，亚历克斯先开始。 每回合，玩家从行的开始或结束处取走整堆石头。 这种情况一直持续到没有更多的石子堆为止，此时手中石子最多的玩家获胜。假设亚历克斯和李都发挥出最佳水平，当亚历克斯赢得比赛时返回 true ，当李赢得比赛时返回 false 。

解题思路

- 一遇到石子问题，很容易让人联想到是否和奇偶数相关。此题指定了石子堆数一定是偶数。所以从这里为突破口试试。Alex 先拿，要么取行首下标为 0 的石子，要么取行尾下标为 n-1 的石子。假设取下标为 0 的石子，剩下的石子堆下标从 1 ~ n-1，即 Lee 只能从奇数下标的石子堆 1 或者 n-1。假设 Alex 第一次取下标为 n-1 的石子，剩下的石子堆下标从 0 ~ n-2，即 Lee 只能取偶数下标的石子堆。于是 Alex 的必胜策略是每轮取石子，取此轮奇数下标堆石子数总和，偶数下标堆石子数总和，两者大者。那么下一轮 Lee 只能取石子堆数相

对少的那一堆，并且 Lee 取的石子堆下标奇偶性是完全受到上一轮 Alex 控制的。所以只要是 Alex 先手，那么每轮都可以压制 Lee，从而必胜。

代码

```
package leetcode

func stoneGame(piles []int) bool {
  return true
}
```

878. Nth Magical Number

题目

A positive integer is *magical* if it is divisible by either A or B.

Return the N-th magical number. Since the answer may be very large, **return it modulo** `10^9 + 7`.

Example 1:

```
Input: N = 1, A = 2, B = 3
Output: 2
```

Example 2:

```
Input: N = 4, A = 2, B = 3
Output: 6
```

Example 3:

```
Input: N = 5, A = 2, B = 4
Output: 10
```

Example 4:

```
Input: N = 3, A = 6, B = 4
Output: 8
```

Note:

1. `1 <= N <= 10^9`
2. `2 <= A <= 40000`
3. `2 <= B <= 40000`

题目大意

如果正整数可以被 A 或 B 整除，那么它是神奇的。返回第 N 个神奇数字。由于答案可能非常大，返回它模 10^9 + 7 的结果。

提示：

1. 1 <= N <= 10^9
2. 2 <= A <= 40000
3. 2 <= B <= 40000

解题思路

- 给出 3 个数字，a，b，n。要求输出可以整除 a 或者整除 b 的第 n 个数。
- 这一题是第 1201 题的缩水版，代码和解题思路也基本不变，这一题的二分搜索的区间是 `[min(A, B), N * min(A, B)] = [2, 10 ^ 14]`。其他代码和第 1201 题一致，思路见第 1201 题。

代码

```
package leetcode

func nthMagicalNumber(N int, A int, B int) int {
  low, high := int64(0), int64(1*1e14)
  for low < high {
    mid := low + (high-low)>>1
    if calNthMagicalCount(mid, int64(A), int64(B)) < int64(N) {
      low = mid + 1
    } else {
      high = mid
    }
  }
  return int(low) % 1000000007
}

func calNthMagicalCount(num, a, b int64) int64 {
  ab := a * b / gcd(a, b)
  return num/a + num/b - num/ab
}
```

880. Decoded String at Index

题目

An encoded string S is given. To find and write the decoded string to a tape, the encoded string is read one character at a time and the following steps are taken:

If the character read is a letter, that letter is written onto the tape.
If the character read is a digit (say d), the entire current tape is repeatedly written d-1 more times in total.
Now for some encoded string S, and an index K, find and return the K-th letter (1 indexed) in the decoded string.

Example 1:

```
Input: S = "leet2code3", K = 10
Output: "o"
Explanation:
The decoded string is "leetleetcodeleetleetcodeleetleetcode".
The 10th letter in the string is "o".
```

Example 2:

```
Input: S = "ha22", K = 5
Output: "h"
Explanation:
The decoded string is "hahahaha".  The 5th letter is "h".
```

Example 3:

```
Input: S = "a2345678999999999999999", K = 1
Output: "a"
Explanation:
The decoded string is "a" repeated 8301530446056247680 times.  The 1st letter is "a".
```

Note:

1. 2 <= S.length <= 100
2. S will only contain lowercase letters and digits 2 through 9.
3. S starts with a letter.
4. 1 <= K <= 10^9
5. The decoded string is guaranteed to have less than 2^63 letters.

题目大意

给定一个编码字符串 S。为了找出解码字符串并将其写入磁带，从编码字符串中每次读取一个字符，并采取以下步骤：

- 如果所读的字符是字母，则将该字母写在磁带上。

- 如果所读的字符是数字（例如 d），则整个当前磁带总共会被重复写 d-1 次。

现在，对于给定的编码字符串 S 和索引 K，查找并返回解码字符串中的第 K 个字母。

解题思路

按照题意，扫描字符串扫到数字的时候，开始重复字符串，这里可以用递归。注意在重复字符串的时候到第 K 个字符的时候就可以返回了，不要等所有字符都扩展完成，这样会超时。d 有可能超大。

代码

```
package leetcode

func isLetter(char byte) bool {
  if char >= 'a' && char <= 'z' {
    return true
  }
  return false
}

func decodeAtIndex(S string, K int) string {
  length := 0
  for i := 0; i < len(S); i++ {
    if isLetter(S[i]) {
      length++
      if length == K {
        return string(S[i])
      }
    } else {
      if length*int(S[i]-'0') >= K {
        if K%length != 0 {
          return decodeAtIndex(S[:i], K%length)
        }
        return decodeAtIndex(S[:i], length)
      }
      length *= int(S[i] - '0')
    }
  }
  return ""
}
```

881. Boats to Save People

题目

The i-th person has weight people[i], and each boat can carry a maximum weight of limit.

Each boat carries at most 2 people at the same time, provided the sum of the weight of those people is at most limit.

Return the minimum number of boats to carry every given person. (It is guaranteed each person can be carried by a boat.)

Example 1:

```
Input: people = [1,2], limit = 3
Output: 1
Explanation: 1 boat (1, 2)
```

Example 2:

```
Input: people = [3,2,2,1], limit = 3
Output: 3
Explanation: 3 boats (1, 2), (2) and (3)
```

Example 3:

```
Input: people = [3,5,3,4], limit = 5
Output: 4
Explanation: 4 boats (3), (3), (4), (5)
```

Note:

- 1 <= people.length <= 50000
- 1 <= people[i] <= limit <= 30000

题目大意

给出人的重量数组，和一个船最大载重量 limit。一个船最多装 2 个人。要求输出装下所有人，最小需要多少艘船。

解题思路

先对人的重量进行排序，然后用 2 个指针分别指向一前一后，一起计算这两个指针指向的重量之和，如果小于 limit，左指针往右移动，并且右指针往左移动。如果大于等于 limit，右指针往左移动。每次指针移动，需要船的个数都要 ++。

代码

```go
package leetcode

import (
  "sort"
)

func numRescueBoats(people []int, limit int) int {
  sort.Ints(people)
  left, right, res := 0, len(people)-1, 0
  for left <= right {
    if left == right {
      res++
      return res
    }
    if people[left]+people[right] <= limit {
      left++
      right--
    } else {
      right--
    }
    res++
  }
  return res
}
```

884. Uncommon Words from Two Sentences

题目

We are given two sentences `A` and `B`. (A *sentence* is a string of space separated words. Each *word* consists only of lowercase letters.)

A word is *uncommon* if it appears exactly once in one of the sentences, and does not appear in the other sentence.

Return a list of all uncommon words.

You may return the list in any order.

Example 1:

```
Input: A = "this apple is sweet", B = "this apple is sour"
Output: ["sweet","sour"]
```

Example 2:

```
Input: A = "apple apple", B = "banana"
Output: ["banana"]
```

Note:

1. `0 <= A.length <= 200`
2. `0 <= B.length <= 200`
3. `A` and `B` both contain only spaces and lowercase letters.

题目大意

给定两个句子 A 和 B 。（句子是一串由空格分隔的单词。每个单词仅由小写字母组成。）

如果一个单词在其中一个句子中只出现一次，在另一个句子中却没有出现，那么这个单词就是不常见的。返回所有不常用单词的列表。您可以按任何顺序返回列表。

解题思路

- 找出 2 个句子中不同的单词，将它们俩都打印出来。简单题，先将 2 个句子的单词都拆开放入 map 中进行词频统计，不同的两个单词的词频肯定都为 1，输出它们即可。

代码

```go
package leetcode

import "strings"

func uncommonFromSentences(A string, B string) []string {
  m, res := map[string]int{}, []string{}
  for _, s := range []string{A, B} {
    for _, word := range strings.Split(s, " ") {
      m[word]++
    }
  }
  for key := range m {
    if m[key] == 1 {
      res = append(res, key)
    }
  }
  return res
}
```

885. Spiral Matrix III

题目

On a 2 dimensional grid with `R` rows and `C` columns, we start at `(r0, c0)` facing east.

Here, the north-west corner of the grid is at the first row and column, and the south-east corner of the grid is at the last row and column.

Now, we walk in a clockwise spiral shape to visit every position in this grid.

Whenever we would move outside the boundary of the grid, we continue our walk outside the grid (but may return to the grid boundary later.)

Eventually, we reach all `R * C` spaces of the grid.

Return a list of coordinates representing the positions of the grid in the order they were visited.

Example 1:

```
Input: R = 1, C = 4, r0 = 0, c0 = 0
Output: [[0,0],[0,1],[0,2],[0,3]]
```

Example 2:

```
Input: R = 5, C = 6, r0 = 1, c0 = 4
Output: [[1,4],[1,5],[2,5],[2,4],[2,3],[1,3],[0,3],[0,4],[0,5],[3,5],[3,4],
[3,3],[3,2],[2,2],[1,2],[0,2],[4,5],[4,4],[4,3],[4,2],[4,1],[3,1],[2,1],[1,1],
[0,1],[4,0],[3,0],[2,0],[1,0],[0,0]]
```

Note:

1. `1 <= R <= 100`
2. `1 <= C <= 100`
3. `0 <= r0 < R`
4. `0 <= c0 < C`

题目大意

在 R 行 C 列的矩阵上，我们从 (r0, c0) 面朝东面开始。这里，网格的西北角位于第一行第一列，网格的东南角位于最后一行最后一列。现在，我们以顺时针按螺旋状行走，访问此网格中的每个位置。每当我们移动到网格的边界之外时，我们会继续在网格之外行走（但稍后可能会返回到网格边界）。最终，我们到过网格的所有 R * C 个空间。

要求输出按照访问顺序返回表示网格位置的坐标列表。

解题思路

- 给出一个二维数组的行 `R`，列 `C`，以及这个数组中的起始点 `(r0,c0)`。从这个起始点开始出发，螺旋的访问数组中各个点，输出途径经过的每个坐标。注意每个螺旋的步长在变长，第一个螺旋是 1 步，第二个螺旋是 1 步，第三个螺旋是 2 步，第四个螺旋是 2 步……即 1，1，2，2，3，3，4，4，5……这样的步长。
- 这一题是第 59 题的加强版。除了有螺旋以外，还加入了步长的限制。步长其实是有规律的，第 0 次移动的步长是 `0/2+1`，第 1 次移动的步长是 `1/2+1`，第 n 次移动的步长是 `n/2+1`。其他的做法和第 59 题一致。

代码

```go
package leetcode

func spiralMatrixIII(R int, C int, r0 int, c0 int) [][]int {
  res, round, spDir := [][]int{}, 0, [][]int{
    []int{0, 1},  // 朝右
    []int{1, 0},  // 朝下
    []int{0, -1}, // 朝左
    []int{-1, 0}, // 朝上
  }
  res = append(res, []int{r0, c0})
  for i := 0; len(res) < R*C; i++ {
    for j := 0; j < i/2+1; j++ {
      r0 += spDir[round%4][0]
      c0 += spDir[round%4][1]
      if 0 <= r0 && r0 < R && 0 <= c0 && c0 < C {
        res = append(res, []int{r0, c0})
      }
    }
    round++
  }
  return res
}
```

887. Super Egg Drop

题目

You are given `K` eggs, and you have access to a building with `N` floors from `1` to `N`.

Each egg is identical in function, and if an egg breaks, you cannot drop it again.

You know that there exists a floor `F` with `0 <= F <= N` such that any egg dropped at a floor higher than `F` will break, and any egg dropped at or below floor `F` will not break.

Each *move*, you may take an egg (if you have an unbroken one) and drop it from any floor `X` (with `1 <= X <= N`).

Your goal is to know **with certainty** what the value of `F` is.

What is the minimum number of moves that you need to know with certainty what `F` is, regardless of the initial value of `F`?

Example 1:

```
Input: K = 1, N = 2
Output: 2
Explanation:
Drop the egg from floor 1.  If it breaks, we know with certainty that F = 0.
Otherwise, drop the egg from floor 2.  If it breaks, we know with certainty that F = 1.
If it didn't break, then we know with certainty F = 2.
Hence, we needed 2 moves in the worst case to know what F is with certainty.
```

Example 2:

```
Input: K = 2, N = 6
Output: 3
```

Example 3:

```
Input: K = 3, N = 14
Output: 4
```

Note:

1. `1 <= K <= 100`
2. `1 <= N <= 10000`

题目大意

你将获得 K 个鸡蛋，并可以使用一栋从 1 到 N 共有 N 层楼的建筑。每个蛋的功能都是一样的，如果一个蛋碎了，你就不能再把它掉下去。你知道存在楼层 F ，满足 0 <= F <= N 任何从高于 F 的楼层落下的鸡蛋都会碎，从 F 楼层或比它低的楼层落下的鸡蛋都不会破。每次移动，你可以取一个鸡蛋（如果你有完整的鸡蛋）并把它从任一楼层 X 扔下（满足 1 <= X <= N）。你的目标是确切地知道 F 的值是多少。无论 F 的初始值如何，你确定 F 的值的最小移动次数是多少?

提示：

1. 1 <= K <= 100
2. 1 <= N <= 10000

解题思路

- 给出 `K` 个鸡蛋，`N` 层楼，要求确定安全楼层 `F` 需要最小步数 `t`。
- 这一题是微软的经典面试题。拿到题最容易想到的是二分搜索。但是仔细分析以后会发现单纯的二分是不对的。不断的二分确实能找到最终安全的楼层，但是这里没有考虑到 `K` 个鸡蛋。鸡蛋数的限制会导致二分搜索无法找到最终楼层。题目要求要在保证能找到最终安全楼层的情况下，找到最小步数。所以单纯的二分搜索并不能解答这道题。
- 这一题如果按照题意正向考虑，动态规划的状态转移方程是 `searchTime(K, N) = max( searchTime(K-1, X-1), searchTime(K, N-X) )`。其中 `X` 是丢鸡蛋的楼层。随着 `X` 从 `[1,N]`，都能计算出一个 `searchTime` 的值，在所有这 `N` 个值之中，取最小值就是本题的答案了。这个解法可以 AC 这道题。不过这个解法不细展开了。时间复杂度 `O(k*N^2)`。

{{< katex display >}}
dp(K,N) = MIN \begin{bmatrix} \, \, MAX(dp(K-1,X-1),dp(K,N-X))\, \, \end{bmatrix} ,1\leqslant x \leqslant N \
{{< /katex >}}

- 换个角度来看这个问题，定义 `dp[k][m]` 代表 `K` 个鸡蛋，`M` 次移动能检查的最大楼层。考虑某一步 `t` 应该在哪一层丢鸡蛋呢？一个正确的选择是在 `dp[k-1][t-1] + 1` 层丢鸡蛋，结果分两种情况：
  1. 如果鸡蛋碎了，我们首先排除了该层以上的所有楼层（不管这个楼有多高），而对于剩下的 `dp[k-1][t-1]` 层楼，我们一定能用 `k-1` 个鸡蛋在 `t-1` 步内求解。因此这种情况下，我们总共可以求解无限高的楼层。可见，这是一种非常好的情况，但并不总是发生。
  2. 如果鸡蛋没碎，我们首先排除了该层以下的 `dp[k-1][t-1]` 层楼，此时我们还有 `k` 个蛋和 `t-1` 步，那么我们去该层以上的楼层继续测得 `dp[k][t-1]` 层楼。因此这种情况下，我们总共可以求解 `dp[k-1][t-1] + 1 + dp[k][t-1]` 层楼。
- 在所有 `m` 步中只要有一次出现了第一种情况，那么我们就可以求解无限高的楼层。但题目要求我们能保证一定能找到安全楼层，所以每次丢鸡蛋的情况应该按照最差情况来，即每次都是第二种情况。于是得到转状态转移方程: `dp[k][m] = dp[k-1][m-1] + dp[k][m-1] + 1` 。这个方程可以压缩到一维，因为每个新的状态只和上一行和左一列有关。那么每一行从右往左更新，即 `dp[i] += 1 + dp[i-1]`。时间复杂度 `O(K * log N)`，空间复杂度 `O(N)`。
- 可能会有人有疑问，如果最初选择不在 `dp[k-1][t-1] + 1` 层丢鸡蛋会怎么样呢？选择在更低的层或者更高的层丢鸡蛋会怎样呢?
  1. 如果在更低的楼层丢鸡蛋也能保证找到安全楼层。那么得到的结果一定不是最小步数。因为这次丢鸡蛋没有充分的展现鸡蛋和移动次数的潜力，最终求解一定会有鸡蛋和步数剩余，即不是能探测的最大楼层了。
  2. 如果在更高的楼层丢鸡蛋，假设是第 `dp[k-1][t-1] + 2` 层丢鸡蛋，如果这次鸡蛋碎了，剩下 `k-1` 个鸡蛋和 `t-1` 步只能保证验证 `dp[k-1][t-1]` 的楼层，这里还剩**第** `dp[k-1][t-1]+ 1` 的楼层，不能保证最终一定能找到安全楼层了。
- 用反证法就能得出每一步都应该在第 `dp[k-1][t-1] + 1` 层丢鸡蛋。
- 这道题还可以用二分搜索来解答。回到上面分析的状态转移方程：`dp[k][m] = dp[k-1][m-1] + dp[k][m-1] + 1` 。用数学方法来解析这个递推关系。令 `f(t,k)` 为 `t` 和 `k` 的函数，题目所要求能测到最大楼层是 `N` 的最小步数，即要求出 `f(t,k) ≥ N` 时候的最小 `t`。由状态转移方程可以知道：`f(t,k) = f(t-1,k) + f(t-1,k-1) + 1`，当 `k = 1` 的时候，对应一个鸡蛋的情况，`f(t,1) = t`，当 `t = 1` 的时候，对应一步的情况，`f(1,k) = 1`。有状态转移方程得：
  {{< katex display >}}
  \begin{aligned} f(t,k) &= 1 + f(t-1,k-1) + f(t-1,k) \\ f(t,k-1) &= 1 + f(t-1,k-2) + f(t-1,k-1) \\ \end{aligned}
  {{< /katex >}}
  令 `g(t,k) = f(t,k) - f(t,k-1)`，可以得到：
  {{< katex display >}}
  g(t,k) = g(t-1,k) + g(t-1,k-1)
  {{< /katex >}}
  可以知道 `g(t,k)` 是一个杨辉三角，即二项式系数：
  {{< katex display >}}
  g(t,k) = \binom{t}{k+1} = C*{t}^{k+1}*
  {{< /katex >}}
  利用裂项相消的方法：

{{< katex display >}}
\begin{aligned} g(t,x) &= f(t,x) - f(t,x-1) \\ g(t,x-1) &= f(t,x-1) - f(t,x-2) \\ g(t,x-2) &= f(t,x-2) - f(t,x-3) \\ \begin{matrix} .\\ .\\ .\end{matrix}\\ g(t,2) &= f(t,2) - f(t,1) \\ g(t,1) &= f(t,1) - f(t,0) \\ \end{aligned}
{{< /katex >}}
于是可以得到：
{{< katex display >}}
\begin{aligned} f(t,k) &= \sum{1}^{k}g(t,x) = \sum{0}^{k} \binom{t}{x} \\ &= C{t}^{0} + C{t}^{1} + C{t}^{2} + ... + C{t}^{k} \\ \end{aligned}
{{< /katex >}}
其中：
{{< katex display >}}
\begin{aligned} C{t}^{k} \cdot \frac{n-k}{k+1} &= C{t}^{k+1} \\ C{t}^{k} &= C{t}^{k-1} \cdot \frac{t-k+1}{k} \\ \end{aligned}
{{< /katex >}}
于是针对每一项的二项式常数，都可以由前一项乘以一个分数得到下一项。
{{< katex display >}}
\begin{aligned} C{t}^{0} &= 1 \\ C{t}^{1} &= C{t}^{0} \cdot \frac{t-1+1}{1} \\ C{t}^{2} &= C{t}^{1} \cdot \frac{t-2+1}{2} \\ C{t}^{3} &= C{t}^{2} \cdot \frac{t-3+1}{3} \\ \begin{matrix} .\\ .\\ .\end{matrix}\\ C{t}^{k} &= C{t}^{k-1} \cdot \frac{t-k+1}{k} \\ \end{aligned}
{{< /katex >}}
利用二分搜索，不断的二分 `t`，直到逼近找到 `f(t,k) ≥ N` 时候最小的 `t`。时间复杂度 `O(K * log N)`，空间复杂度 `O(1)`。

代码

```
package leetcode

// 解法一 二分搜索
func superEggDrop(K int, N int) int {
  low, high := 1, N
  for low < high {
    mid := low + (high-low)>>1
    if counterF(K, N, mid) >= N {
      high = mid
    } else {
      low = mid + 1
    }
  }
  return low
}

// 计算二项式和，特殊的第一项 C(t,0) = 1
func counterF(k, n, mid int) int {
  res, sum := 1, 0
  for i := 1; i <= k && sum < n; i++ {
```

```
    res *= mid - i + 1
    res /= i
    sum += res
  }
  return sum
}

// 解法二 动态规划 DP
func superEggDrop1(K int, N int) int {
  dp, step := make([]int, K+1), 0
  for ; dp[K] < N; step++ {
    for i := K; i > 0; i-- {
      dp[i] = (1 + dp[i] + dp[i-1])
    }
  }
  return step
}
```

888. Fair Candy Swap

题目

Alice and Bob have candy bars of different sizes: `A[i]` is the size of the `i`-th bar of candy that Alice has, and `B[j]` is the size of the `j`-th bar of candy that Bob has.

Since they are friends, they would like to exchange one candy bar each so that after the exchange, they both have the same total amount of candy. (*The total amount of candy a person has is the sum of the sizes of candy bars they have*.)

Return an integer array `ans` where `ans[0]` is the size of the candy bar that Alice must exchange, and `ans[1]` is the size of the candy bar that Bob must exchange.

If there are multiple answers, you may return any one of them. It is guaranteed an answer exists.

Example 1:

```
Input: A = [1,1], B = [2,2]
Output: [1,2]
```

Example 2:

```
Input: A = [1,2], B = [2,3]
Output: [1,2]
```

Example 3:

```
Input: A = [2], B = [1,3]
Output: [2,3]
```

Example 4:

```
Input: A = [1,2,5], B = [2,4]
Output: [5,4]
```

Note:

- `1 <= A.length <= 10000`
- `1 <= B.length <= 10000`
- `1 <= A[i] <= 100000`
- `1 <= B[i] <= 100000`
- It is guaranteed that Alice and Bob have different total amounts of candy.
- It is guaranteed there exists an answer.

题目大意

爱丽丝和鲍勃有不同大小的糖果棒：A[i] 是爱丽丝拥有的第 i 块糖的大小，B[j] 是鲍勃拥有的第 j 块糖的大小。因为他们是朋友，所以他们想交换一个糖果棒，这样交换后，他们都有相同的糖果总量。（一个人拥有的糖果总量是他们拥有的糖果棒大小的总和。）返回一个整数数组 ans，其中 ans[0] 是爱丽丝必须交换的糖果棒的大小，ans[1] 是 Bob 必须交换的糖果棒的大小。如果有多个答案，你可以返回其中任何一个。保证答案存在。

提示：

- 1 <= A.length <= 10000
- 1 <= B.length <= 10000
- 1 <= A[i] <= 100000
- 1 <= B[i] <= 100000
- 保证爱丽丝与鲍勃的糖果总量不同。
- 答案肯定存在。

解题思路

- 两人交换糖果，使得两人糖果相等。要求输出一个数组，里面分别包含两人必须交换的糖果大小。
- 首先这一题肯定了一定有解，其次只允许交换一次。有了这两个前提，使本题变成简单题。先计算出为了使得交换以后两个相同的糖果数，A 需要增加或者减少的糖果数 diff。然后遍历 B ，看 A 中是否存在一个元素，能使得 B 做了对应交换 diff 以后，两人糖果相等。(此题前提保证了一定能找到)。最后输出 A 中的这个元素和遍历到 B 的这个元素，即是两人要交换的糖果数。

代码

```go
package leetcode

func fairCandySwap(A []int, B []int) []int {
  hDiff, aMap := diff(A, B)/2, make(map[int]int, len(A))
```

```go
  for _, a := range A {
    aMap[a] = a
  }
  for _, b := range B {
    if a, ok := aMap[hDiff+b]; ok {
      return []int{a, b}
    }
  }
  return nil
}

func diff(A []int, B []int) int {
  diff, maxLen := 0, max(len(A), len(B))
  for i := 0; i < maxLen; i++ {
    if i < len(A) {
      diff += A[i]
    }
    if i < len(B) {
      diff -= B[i]
    }
  }
  return diff
}

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

890. Find and Replace Pattern

题目

Given a list of strings `words` and a string `pattern`, return *a list of* `words[i]` *that match* `pattern`. You may return the answer in **any order**.

A word matches the pattern if there exists a permutation of letters `p` so that after replacing every letter `x` in the pattern with `p(x)`, we get the desired word.

Recall that a permutation of letters is a bijection from letters to letters: every letter maps to another letter, and no two letters map to the same letter.

Example 1:

```
Input: words = ["abc","deq","mee","aqq","dkd","ccc"], pattern = "abb"
Output: ["mee","aqq"]
Explanation: "mee" matches the pattern because there is a permutation {a -> m, b -> e,
...}.
"ccc" does not match the pattern because {a -> c, b -> c, ...} is not a permutation,
since a and b map to the same letter.
```

Example 2:

```
Input: words = ["a","b","c"], pattern = "a"
Output: ["a","b","c"]
```

Constraints:

- `1 <= pattern.length <= 20`
- `1 <= words.length <= 50`
- `words[i].length == pattern.length`
- `pattern` and `words[i]` are lowercase English letters.

题目大意

你有一个单词列表 words 和一个模式 pattern，你想知道 words 中的哪些单词与模式匹配。如果存在字母的排列 p ，使得将模式中的每个字母 x 替换为 p(x) 之后，我们就得到了所需的单词，那么单词与模式是匹配的。（回想一下，字母的排列是从字母到字母的双射：每个字母映射到另一个字母，没有两个字母映射到同一个字母。）返回 words 中与给定模式匹配的单词列表。你可以按任何顺序返回答案。

解题思路

- 按照题目要求，分别映射两个字符串，words 字符串数组中的字符串与 pattern 字符串每个字母做映射。这里用 map 存储。题目还要求不存在 2 个字母映射到同一个字母的情况，所以再增加一个 map，用来判断当前字母是否已经被映射过了。以上 2 个条件都满足即代表模式匹配上了。最终将所有满足模式匹配的字符串输出即可。

代码

```go
package leetcode

func findAndReplacePattern(words []string, pattern string) []string {
  res := make([]string, 0)
  for _, word := range words {
    if match(word, pattern) {
      res = append(res, word)
    }
  }
  return res
}
```

```go
func match(w, p string) bool {
  if len(w) != len(p) {
    return false
  }
  m, used := make(map[uint8]uint8), make(map[uint8]bool)
  for i := 0; i < len(w); i++ {
    if v, ok := m[p[i]]; ok {
      if w[i] != v {
        return false
      }
    } else {
      if used[w[i]] {
        return false
      }
      m[p[i]] = w[i]
      used[w[i]] = true
    }
  }
  return true
}
```

891. Sum of Subsequence Widths

题目

Given an array of integers A, consider all non-empty subsequences of A.

For any sequence S, let the width of S be the difference between the maximum and minimum element of S.

Return the sum of the widths of all subsequences of A.

As the answer may be very large, return the answer modulo 10^9 + 7.

Example 1:

```
Input: [2,1,3]
Output: 6
Explanation:
Subsequences are [1], [2], [3], [2,1], [2,3], [1,3], [2,1,3].
The corresponding widths are 0, 0, 0, 1, 1, 2, 2.
The sum of these widths is 6.
```

Note:

- 1 <= A.length <= 20000
- 1 <= A[i] <= 20000

题目大意

给定一个整数数组 A ， 考虑 A 的所有非空子序列。对于任意序列 S ， 设 S 的宽度是 S 的最大元素和最小元素的差。返回 A 的所有子序列的宽度之和。由于答案可能非常大，请返回答案模 10^9+7。

解题思路

- 理解题意以后，可以发现，数组内元素的顺序并不影响最终求得的所有子序列的宽度之和。

  ```
  [2,1,3]:[1],[2],[3],[2,1],[2,3],[1,3],[2,1,3]
  [1,2,3]:[1],[2],[3],[1,2],[2,3],[1,3],[1,2,3]
  ```

  针对每个 A[i] 而言，A[i] 对最终结果的贡献是在子序列的左右两边的时候才有贡献，当 A[i] 位于区间中间的时候，不影响最终结果。先对 A[i] 进行排序，排序以后，有 i 个数 <= A[i]，有 n - i - 1 个数 >= A[i]。所以 A[i] 会在 2^i 个子序列的右边界出现，2^(n-i-1) 个左边界出现。那么 A[i] 对最终结果的贡献是 A[i] * 2^i - A[i] * 2^(n-i-1) 。举个例子，[1,4,5,7]，A[2] = 5，那么 5 作为右边界的子序列有 2^2 = 4 个，即 [5],[1,5],[4,5],[1,4,5]，5 作为左边界的子序列有 2^(4-2-1) = 2 个，即 [5],[5,7]。A[2] = 5 对最终结果的影响是 5 * 2^2 - 5 * 2^(4-2-1) = 10 。
- 题目要求所有子序列的宽度之和，也就是求每个区间最大值减去最小值的总和。那么 `Ans = SUM{ A[i]*2^i - A[n-i-1] * 2^(n-i-1) }`，其中 `0 <= i < n`。需要注意的是 2^i 可能非常大，所以在计算中就需要去 mod 了，而不是最后计算完了再 mod。注意取模的结合律：`(a * b) % c = (a % c) * (b % c) % c`。

代码

```go
package leetcode

import (
  "sort"
)

func sumSubseqWidths(A []int) int {
  sort.Ints(A)
  res, mod, n, p := 0, 1000000007, len(A), 1
  for i := 0; i < n; i++ {
    res = (res + (A[i]-A[n-1-i])*p) % mod
    p = (p << 1) % mod
  }
  return res
}
```

892. Surface Area of 3D Shapes

题目

On a `N * N` grid, we place some `1 * 1 * 1` cubes.

Each value `v = grid[i][j]` represents a tower of `v` cubes placed on top of grid cell `(i, j)`.

Return the total surface area of the resulting shapes.

Example 1:

```
Input: [[2]]
Output: 10
```

Example 2:

```
Input: [[1,2],[3,4]]
Output: 34
```

Example 3:

```
Input: [[1,0],[0,2]]
Output: 16
```

Example 4:

```
Input: [[1,1,1],[1,0,1],[1,1,1]]
Output: 32
```

Example 5:

```
Input: [[2,2,2],[2,1,2],[2,2,2]]
Output: 46
```

Note:

- `1 <= N <= 50`
- `0 <= grid[i][j] <= 50`

题目大意

在 N * N 的网格上，我们放置一些 1 * 1 * 1 的立方体。每个值 v = grid[i][j] 表示 v 个正方体叠放在对应单元格 (i, j) 上。请你返回最终形体的表面积。

解题思路

- 给定一个网格数组，数组里面装的是立方体叠放在所在的单元格，求最终这些叠放的立方体的表面积。
- 简单题。按照题目意思，找到叠放时，重叠的面，然后用总表面积减去这些重叠的面积即为最终答案。

代码

```go
package leetcode

func surfaceArea(grid [][]int) int {
  area := 0
  for i := 0; i < len(grid); i++ {
    for j := 0; j < len(grid[0]); j++ {
      if grid[i][j] == 0 {
        continue
      }
      area += grid[i][j]*4 + 2
      // up
      if i > 0 {
        m := min(grid[i][j], grid[i-1][j])
        area -= m
      }
      // down
      if i < len(grid)-1 {
        m := min(grid[i][j], grid[i+1][j])
        area -= m
      }
      // left
      if j > 0 {
        m := min(grid[i][j], grid[i][j-1])
        area -= m
      }
      // right
      if j < len(grid[i])-1 {
        m := min(grid[i][j], grid[i][j+1])
        area -= m
      }
    }
  }
  return area
}

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

895. Maximum Frequency Stack

题目

Implement FreqStack, a class which simulates the operation of a stack-like data structure.

FreqStack has two functions:

push(int x), which pushes an integer x onto the stack.
pop(), which removes and returns the most frequent element in the stack.
If there is a tie for most frequent element, the element closest to the top of the stack is removed and returned.

Example 1:

```
Input:
["FreqStack","push","push","push","push","push","push","pop","pop","pop","pop"],
[[],[5],[7],[5],[7],[4],[5],[],[],[],[]]
Output: [null,null,null,null,null,null,null,5,7,5,4]
Explanation:
After making six .push operations, the stack is [5,7,5,7,4,5] from bottom to top.
Then:

pop() -> returns 5, as 5 is the most frequent.
The stack becomes [5,7,5,7,4].

pop() -> returns 7, as 5 and 7 is the most frequent, but 7 is closest to the top.
The stack becomes [5,7,5,4].

pop() -> returns 5.
The stack becomes [5,7,4].

pop() -> returns 4.
The stack becomes [5,7].
```

Note:

- Calls to FreqStack.push(int x) will be such that 0 <= x <= 10^9.
- It is guaranteed that FreqStack.pop() won't be called if the stack has zero elements.
- The total number of FreqStack.push calls will not exceed 10000 in a single test case.
- The total number of FreqStack.pop calls will not exceed 10000 in a single test case.
- The total number of FreqStack.push and FreqStack.pop calls will not exceed 150000 across all test cases.

题目大意

实现 FreqStack，模拟类似栈的数据结构的操作的一个类。

FreqStack 有两个函数：

- push(int x)，将整数 x 推入栈中。
- pop()，它移除并返回栈中出现最频繁的元素。如果最频繁的元素不只一个，则移除并返回最接近栈顶的元素。

解题思路

FreqStack 里面保存频次的 map 和相同频次 group 的 map。push 的时候动态的维护 x 的频次，并更新到对应频次的 group 中。pop 的时候对应减少频次字典里面的频次，并更新到对应频次的 group 中。

代码

```go
package leetcode

type FreqStack struct {
  freq    map[int]int
  group   map[int][]int
  maxfreq int
}

func Constructor895() FreqStack {
  hash := make(map[int]int)
  maxHash := make(map[int][]int)
  return FreqStack{freq: hash, group: maxHash}
}

func (this *FreqStack) Push(x int) {
  if _, ok := this.freq[x]; ok {
    this.freq[x]++
  } else {
    this.freq[x] = 1
  }
  f := this.freq[x]
  if f > this.maxfreq {
    this.maxfreq = f
  }

  this.group[f] = append(this.group[f], x)
}

func (this *FreqStack) Pop() int {
  tmp := this.group[this.maxfreq]
  x := tmp[len(tmp)-1]
  this.group[this.maxfreq] = this.group[this.maxfreq][:len(this.group[this.maxfreq])-1]
  this.freq[x]--
  if len(this.group[this.maxfreq]) == 0 {
    this.maxfreq--
  }
  return x
}
```

```
/**
 * Your FreqStack object will be instantiated and called as such:
 * obj := Constructor();
 * obj.Push(x);
 * param_2 := obj.Pop();
 */
```

896. Monotonic Array

题目

An array is *monotonic* if it is either monotone increasing or monotone decreasing.

An array `A` is monotone increasing if for all `i <= j`, `A[i] <= A[j]`. An array `A` is monotone decreasing if for all `i <= j`, `A[i] >= A[j]`.

Return `true` if and only if the given array `A` is monotonic.

Example 1:

```
Input: [1,2,2,3]
Output: true
```

Example 2:

```
Input: [6,5,4,4]
Output: true
```

Example 3:

```
Input: [1,3,2]
Output: false
```

Example 4:

```
Input: [1,2,4,5]
Output: true
```

Example 5:

```
Input: [1,1,1]
Output: true
```

Note:

1. `1 <= A.length <= 50000`

2. `-100000 <= A[i] <= 100000`

题目大意

如果数组是单调递增或单调递减的，那么它是单调的。如果对于所有 i <= j，A[i] <= A[j]，那么数组 A 是单调递增的。 如果对于所有 i <= j，A[i]> = A[j]，那么数组 A 是单调递减的。当给定的数组 A 是单调数组时返回 true，否则返回 false。

解题思路

- 判断给定的数组是不是单调(单调递增或者单调递减)的。
- 简单题，按照题意循环判断即可。

代码

```
package leetcode

func isMonotonic(A []int) bool {
  if len(A) <= 1 {
    return true
  }
  if A[0] < A[1] {
    return inc(A[1:])
  }
  if A[0] > A[1] {
    return dec(A[1:])
  }
  return inc(A[1:]) || dec(A[1:])
}

func inc(A []int) bool {
  for i := 0; i < len(A)-1; i++ {
    if A[i] > A[i+1] {
      return false
    }
  }
  return true
}

func dec(A []int) bool {
  for i := 0; i < len(A)-1; i++ {
    if A[i] < A[i+1] {
      return false
    }
  }
  return true
}
```

897. Increasing Order Search Tree

题目

Given a binary search tree, rearrange the tree in **in-order** so that the leftmost node in the tree is now the root of the tree, and every node has no left child and only 1 right child.

Example 1:

```
Input: [5,3,6,2,4,null,8,1,null,null,null,7,9]

       5
      / \
    3    6
   / \    \
  2   4    8
 /        / \
1        7   9

Output: [1,null,2,null,3,null,4,null,5,null,6,null,7,null,8,null,9]

 1
  \
   2
    \
     3
      \
       4
        \
         5
          \
           6
            \
             7
              \
               8
                \
                 9
```

Note:

1. The number of nodes in the given tree will be between 1 and 100.
2. Each node will have a unique integer value from 0 to 1000.

题目大意

给定一个树，按中序遍历重新排列树，使树中最左边的结点现在是树的根，并且每个结点没有左子结点，只有一个右子结点。

提示：

- 给定树中的结点数介于 1 和 100 之间。
- 每个结点都有一个从 0 到 1000 范围内的唯一整数值。

解题思路

- 给出一颗树，要求把树的所有孩子都排列到右子树上。
- 按照题意，可以先中根遍历原树，然后按照中根遍历的顺序，把所有节点都放在右子树上。见解法二。
- 上一种解法会重新构造一颗新树，有没有办法可以直接更改原有的树呢？节约存储空间。虽然平时软件开发过程中不建议更改原有的值，但是算法题中追求空间和时间的最优，可以考虑一下。**树可以看做是有多个孩子的链表**。这一题可以看成是链表的类似反转的操作。这一点想通以后，就好做了。先找到左子树中最左边的节点，这个节点是新树的根节点。然后依次往回遍历，不断的记录下上一次遍历的最后节点 tail，边遍历，边把后续节点串起来。最终“反转”完成以后，就得到了题目要求的样子了。代码实现见解法一。

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

// 解法一 链表思想
func increasingBST(root *TreeNode) *TreeNode {
  var head = &TreeNode{}
  tail := head
  recBST(root, tail)
  return head.Right
}

func recBST(root, tail *TreeNode) *TreeNode {
  if root == nil {
    return tail
  }
  tail = recBST(root.Left, tail)
  root.Left = nil                // 切断 root 与其 Left 的连接，避免形成环
  tail.Right, tail = root, root // 把 root 接上 tail，并保持 tail 指向尾部
  tail = recBST(root.Right, tail)
  return tail
}

// 解法二 模拟
```

```go
func increasingBST1(root *TreeNode) *TreeNode {
  list := []int{}
  inorder(root, &list)
  if len(list) == 0 {
    return root
  }
  newRoot := &TreeNode{Val: list[0], Left: nil, Right: nil}
  cur := newRoot
  for index := 1; index < len(list); index++ {
    tmp := &TreeNode{Val: list[index], Left: nil, Right: nil}
    cur.Right = tmp
    cur = tmp
  }
  return newRoot
}

func inorder(root *TreeNode, output *[]int) {
  if root != nil {
    inorder(root.Left, output)
    *output = append(*output, root.Val)
    inorder(root.Right, output)
  }
}
```

898. Bitwise ORs of Subarrays

题目

We have an array `A` of non-negative integers.

For every (contiguous) subarray `B = [A[i], A[i+1], ..., A[j]]` (with `i <= j`), we take the bitwise OR of all the elements in `B`, obtaining a result `A[i] | A[i+1] | ... | A[j]`.

Return the number of possible results. (Results that occur more than once are only counted once in the final answer.)

Example 1:

```
Input: [0]
Output: 1
Explanation:
There is only one possible result: 0.
```

Example 2:

```
Input: [1,1,2]
Output: 3
Explanation:
The possible subarrays are [1], [1], [2], [1, 1], [1, 2], [1, 1, 2].
These yield the results 1, 1, 2, 1, 3, 3.
There are 3 unique values, so the answer is 3.
```

Example 3:

```
Input: [1,2,4]
Output: 6
Explanation:
The possible results are 1, 2, 3, 4, 6, and 7.
```

Note:

1. `1 <= A.length <= 50000`
2. `0 <= A[i] <= 10^9`

题目大意

我们有一个非负整数数组 A。对于每个（连续的）子数组 B = [A[i], A[i+1], ..., A[j]] （ i <= j），我们对 B 中的每个元素进行按位或操作，获得结果 A[i] | A[i+1] | ... | A[j]。返回可能结果的数量。（多次出现的结果在最终答案中仅计算一次。）

解题思路

- 给出一个数组，要求求出这个数组所有的子数组中，每个集合内所有数字取 `|` 运算以后，不同结果的种类数。
- 这道题可以这样考虑，第一步，先考虑所有的子数组如何得到，以 `[001, 011, 100, 110, 101]` 为例，所有的子数组集合如下：

```
[001]
[001 011] [011]
[001 011 100] [011 100] [100]
[001 011 100 110] [011 100 110] [100 110] [110]
[001 011 100 110 101] [011 100 110 101] [100 110 101] [110 101] [101]
```

可以发现，从左往右遍历原数组，每次新来的一个元素，依次加入到之前已经生成过的集合中，再以自己为单独集合。这样就可以生成原数组的所有子集。

- 第二步，将每一行的子集内的所有元素都进行 `|` 运算，得到：

```
    001
    011 011
    111 111 100
    111 111 110 110
    111 111 111 111 101
```

- 第三步，去重：

```
    001
    011
    111 100
    111 110
    111 101
```

由于二进制位不超过 32 位，所以这里每一行最多不会超过 32 个数。所以最终时间复杂度不会超过 O(32 N)，即 O(K * N)。最后将这每一行的数字都放入最终的 map 中去重即可。

代码

```go
package leetcode

// 解法一 array 优化版
func subarrayBitwiseORs(A []int) int {
  res, cur, isInMap := []int{}, []int{}, make(map[int]bool)
  cur = append(cur, 0)
  for _, v := range A {
    var cur2 []int
    for _, vv := range cur {
      tmp := v | vv
      if !inSlice(cur2, tmp) {
        cur2 = append(cur2, tmp)
      }
    }
    if !inSlice(cur2, v) {
      cur2 = append(cur2, v)
    }
    cur = cur2
    for _, vv := range cur {
      if _, ok := isInMap[vv]; !ok {
        isInMap[vv] = true
        res = append(res, vv)
      }
    }
  }
  return len(res)
}
```

```go
func inSlice(A []int, T int) bool {
  for _, v := range A {
    if v == T {
      return true
    }
  }
  return false
}

// 解法二 map 版
func subarrayBitwiseORs1(A []int) int {
  res, t := map[int]bool{}, map[int]bool{}
  for _, num := range A {
    r := map[int]bool{}
    r[num] = true
    for n := range t {
      r[(num | n)] = true
    }
    t = r
    for n := range t {
      res[n] = true
    }
  }
  return len(res)
}
```

901. Online Stock Span

题目

Write a class StockSpanner which collects daily price quotes for some stock, and returns the span of that stock's price for the current day.

The span of the stock's price today is defined as the maximum number of consecutive days (starting from today and going backwards) for which the price of the stock was less than or equal to today's price.

For example, if the price of a stock over the next 7 days were [100, 80, 60, 70, 60, 75, 85], then the stock spans would be [1, 1, 1, 2, 1, 4, 6].

Example 1:

```
Input: ["StockSpanner","next","next","next","next","next","next","next"], [[],[100],[80],[60],[70],[60],[75],[85]]
Output: [null,1,1,1,2,1,4,6]
Explanation:
First, S = StockSpanner() is initialized.  Then:
```

```
S.next(100) is called and returns 1,
S.next(80) is called and returns 1,
S.next(60) is called and returns 1,
S.next(70) is called and returns 2,
S.next(60) is called and returns 1,
S.next(75) is called and returns 4,
S.next(85) is called and returns 6.

Note that (for example) S.next(75) returned 4, because the last 4 prices
(including today's price of 75) were less than or equal to today's price.
```

Note:

1. Calls to StockSpanner.next(int price) will have 1 <= price <= 10^5.
2. There will be at most 10000 calls to StockSpanner.next per test case.
3. There will be at most 150000 calls to StockSpanner.next across all test cases.
4. The total time limit for this problem has been reduced by 75% for C++, and 50% for all other languages.

题目大意

编写一个 StockSpanner 类，它收集某些股票的每日报价，并返回该股票当日价格的跨度。

今天股票价格的跨度被定义为股票价格小于或等于今天价格的最大连续日数（从今天开始往回数，包括今天）。

例如，如果未来7天股票的价格是 [100, 80, 60, 70, 60, 75, 85]，那么股票跨度将是 [1, 1, 1, 2, 1, 4, 6]。

解题思路

这一题就是单调栈的题目。维护一个单调递增的下标。

总结

单调栈类似的题

496. Next Greater Element I
497. Next Greater Element II
498. Daily Temperatures
499. Sum of Subarray Minimums
500. Largest Rectangle in Histogram

代码

```go
package leetcode

import "fmt"
```

```go
// node pair
type Node struct {
  Val int
  res int
}

// slice
type StockSpanner struct {
  Item []Node
}

func Constructor901() StockSpanner {
  stockSpanner := StockSpanner{make([]Node, 0)}
  return stockSpanner
}

// need refactor later
func (this *StockSpanner) Next(price int) int {
  res := 1
  if len(this.Item) == 0 {
    this.Item = append(this.Item, Node{price, res})
    return res
  }
  for len(this.Item) > 0 && this.Item[len(this.Item)-1].Val <= price {
    res = res + this.Item[len(this.Item)-1].res
    this.Item = this.Item[:len(this.Item)-1]
  }
  this.Item = append(this.Item, Node{price, res})
  fmt.Printf("this.Item = %v\n", this.Item)
  return res
}

/**
 * Your StockSpanner object will be instantiated and called as such:
 * obj := Constructor();
 * param_1 := obj.Next(price);
 */
```

904. Fruit Into Baskets

题目

In a row of trees, the i-th tree produces fruit with type tree[i].

You start at any tree of your choice, then repeatedly perform the following steps:

1. Add one piece of fruit from this tree to your baskets. If you cannot, stop.
2. Move to the next tree to the right of the current tree. If there is no tree to the right, stop.

Note that you do not have any choice after the initial choice of starting tree: you must perform step 1, then step 2, then back to step 1, then step 2, and so on until you stop.

You have two baskets, and each basket can carry any quantity of fruit, but you want each basket to only carry one type of fruit each.

What is the total amount of fruit you can collect with this procedure?

Example 1:

```
Input: [1,2,1]
Output: 3
Explanation: We can collect [1,2,1].
```

Example 2:

```
Input: [0,1,2,2]
Output: 3
Explanation: We can collect [1,2,2].
If we started at the first tree, we would only collect [0, 1].
```

Example 3:

```
Input: [1,2,3,2,2]
Output: 4
Explanation: We can collect [2,3,2,2].
If we started at the first tree, we would only collect [1, 2].
```

Example 4:

```
Input: [3,3,3,1,2,1,1,2,3,3,4]
Output: 5
Explanation: We can collect [1,2,1,1,2].
If we started at the first tree or the eighth tree, we would only collect 4 fruits.
```

Note:

- 1 <= tree.length <= 40000
- 0 <= tree[i] < tree.length

题目大意

这道题考察的是滑动窗口的问题。

给出一个数组，数组里面的数字代表每个果树上水果的种类，1 代表一号水果，不同数字代表的水果不同。现在有 2 个篮子，每个篮子只能装一个种类的水果，这就意味着只能选 2 个不同的数字。摘水果只能从左往右摘，直到右边没有水果可以摘就停下。问可以连续摘水果的最长区间段的长度。

解题思路

简化一下题意，给出一段数字，要求找出包含 2 个不同数字的最大区间段长度。这个区间段内只能包含这 2 个不同数字，可以重复，但是不能包含其他数字。

用典型的滑动窗口的处理方法处理即可。

代码

```go
package leetcode

func totalFruit(tree []int) int {
  if len(tree) == 0 {
    return 0
  }
  left, right, counter, res, freq := 0, 0, 1, 1, map[int]int{}
  freq[tree[0]]++
  for left < len(tree) {
    if right+1 < len(tree) && ((counter > 0 && tree[right+1] != tree[left]) ||
(tree[right+1] == tree[left] || freq[tree[right+1]] > 0)) {
      if counter > 0 && tree[right+1] != tree[left] {
        counter--
      }
      right++
      freq[tree[right]]++
    } else {
      if counter == 0 || (counter > 0 && right == len(tree)-1) {
        res = max(res, right-left+1)
      }
      freq[tree[left]]--
      if freq[tree[left]] == 0 {
        counter++
      }
      left++
    }
  }
  return res
}
```

907. Sum of Subarray Minimums

题目

Given an array of integers A, find the sum of min(B), where B ranges over every (contiguous) subarray of A.

Since the answer may be large, return the answer modulo 10^9 + 7.

Example 1:

```
Input: [3,1,2,4]
Output: 17
Explanation: Subarrays are [3], [1], [2], [4], [3,1], [1,2], [2,4], [3,1,2], [1,2,4],
[3,1,2,4].
Minimums are 3, 1, 2, 4, 1, 1, 2, 1, 1, 1.  Sum is 17.
```

Note:

1. 1 <= A.length <= 30000
2. 1 <= A[i] <= 30000

题目大意

给定一个整数数组 A，找到 min(B) 的总和，其中 B 的范围为 A 的每个（连续）子数组。

由于答案可能很大，因此返回答案模 10^9 + 7。

解题思路

- 首先想到的是暴力解法，用两层循环，分别枚举每个连续的子区间，区间内用一个元素记录区间内最小值。每当区间起点发生变化的时候，最终结果都加上上次遍历区间找出的最小值。当整个数组都扫完一遍以后，最终结果模上 10^9+7。
- 上面暴力解法时间复杂度特别大，因为某个区间的最小值可能是很多区间的最小值，但是我们暴力枚举所有区间，导致要遍历的区间特别多。优化点就在如何减少遍历的区间。第二种思路是用 2 个单调栈。想得到思路是 `res = sum(A[i] * f(i))`，其中 f(i) 是子区间的数，A[i] 是这个子区间内的最小值。为了得到 f(i) 我们需要找到 left[i] 和 right[i]，left[i] 是 A[i] 左边严格大于 A[i]的区间长度。right[i] 是 A[i] 右边非严格大于(>= 关系)的区间长度。left[i] + 1 等于以 A[i] 结尾的子数组数目，A[i] 是唯一的最小值；right[i] + 1 等于以 A[i] 开始的子数组数目，A[i] 是第一个最小值。于是有 `f(i) = (left[i] + 1) * (right[i] + 1)`。例如对于 [3,1,4,2,5,3,3,1] 中的“2”，我们找到的串就为[4,2,5,3,3]，2 左边有 1 个数比 2 大且相邻，2 右边有 3 个数比 2 大且相邻，所以 2 作为最小值的串有 2 * 4 = 8 种。用排列组合的思维也能分析出来，2 的左边可以拿 0，1，..... m 个，总共 (m + 1) 种，同理右边可以拿 0，1，..... n 个，总共 (n + 1) 种，所以总共 (m + 1)(n + 1) 种。只要计算出了 f(i)，这个题目就好办了。以 [3,1,2,4] 为例，left[i] + 1 = [1,2,1,1]，right[i] + 1 = [1,3,2,1]，对应 i 位的乘积是 f[i] = [1 * 1，2 * 3，1 * 2，1 * 1] = [1，6，2，1]，最终要求的最小值的总和 res = 3 * 1 + 1 * 6 + 2 * 2 + 4 * 1 = 17。

- **看到这种 mod1e9+7 的题目，首先要想到的就是dp**。最终的优化解即是利用 DP + 单调栈。单调栈维护数组中的值逐渐递增的对应下标序列。定义 `dp[i + 1]` 代表以 A[i] 结尾的子区间内最小值的总和。状态转移方程是 `dp[i + 1] = dp[prev + 1] + (i - prev) * A[i]`，其中 prev 是比 A[i] 小的前一个数，由于我们维护了一个单调栈，所以 prev 就是栈顶元素。(i - prev) * A[i] 代表在还没有出现 prev 之前，这些区间内都是 A[i] 最小，那么这些区间有 i - prev 个，所以最小值总和应该是 (i - prev) * A[i]。再加上 dp[prev + 1] 就是 dp[i + 1] 的最小值总和了。以 [3, 1, 2, 4, 3] 为例，当 i = 4, 所有以 A[4] 为结尾的子区间有:

  ```
  [3]
  [4, 3]
  [2, 4, 3]
  [1, 2, 4, 3]
  [3, 1, 2, 4, 3]
  ```

  在这种情况下, stack.peek() = 2, A[2] = 2。前两个子区间 [3] and [4, 3], 最小值的总和 = (i - stack.peek()) * A[i] = 6。后 3 个子区间是 [2, 4, 3], [1, 2, 4, 3] 和 [3, 1, 2, 4, 3], 它们都包含 2，2 是比 3 小的前一个数，所以 dp[i + 1] = dp[stack.peek() + 1] = dp[2 + 1] = dp[3] = dp[2 + 1]。即需要求 i = 2 的时候 dp[i + 1] 的值。继续递推，比 2 小的前一个值是 1，A[1] = 1。dp[3] = dp[1 + 1] + (2 - 1) * A[2]= dp[2] + 2。dp[2] = dp[1 + 1]，当 i = 1 的时候，prev = -1，即没有人比 A[1] 更小了，所以 dp[2] = dp[1 + 1] = dp[-1 + 1] + (1 - (-1)) * A[1] = 0 + 2 * 1 = 2。迭代回去，dp[3] = dp[2] + 2 = 2 + 2 = 4。dp[stack.peek() + 1] = dp[2 + 1] = dp[3] = 4。所以 dp[i + 1] = 4 + 6 = 10。
- 与这一题相似的解题思路的题目有第 828 题，第 891 题。

代码

```go
package leetcode

// 解法一 最快的解是 DP + 单调栈
func sumSubarrayMins(A []int) int {
  stack, dp, res, mod := []int{}, make([]int, len(A)+1), 0, 1000000007
  stack = append(stack, -1)

  for i := 0; i < len(A); i++ {
    for stack[len(stack)-1] != -1 && A[i] <= A[stack[len(stack)-1]] {
      stack = stack[:len(stack)-1]
    }
    dp[i+1] = (dp[stack[len(stack)-1]+1] + (i-stack[len(stack)-1])*A[i]) % mod
    stack = append(stack, i)
    res += dp[i+1]
    res %= mod
  }
  return res
}

type pair struct {
  val   int
  count int
```

```go
}

// 解法二 用两个单调栈
func sumSubarrayMins1(A []int) int {
  res, n, mod := 0, len(A), 1000000007
  lefts, rights, leftStack, rightStack := make([]int, n), make([]int, n), []*pair{},
[]*pair{}
  for i := 0; i < n; i++ {
    count := 1
    for len(leftStack) != 0 && leftStack[len(leftStack)-1].val > A[i] {
      count += leftStack[len(leftStack)-1].count
      leftStack = leftStack[:len(leftStack)-1]
    }
    leftStack = append(leftStack, &pair{val: A[i], count: count})
    lefts[i] = count
  }

  for i := n - 1; i >= 0; i-- {
    count := 1
    for len(rightStack) != 0 && rightStack[len(rightStack)-1].val >= A[i] {
      count += rightStack[len(rightStack)-1].count
      rightStack = rightStack[:len(rightStack)-1]
    }
    rightStack = append(rightStack, &pair{val: A[i], count: count})
    rights[i] = count
  }

  for i := 0; i < n; i++ {
    res = (res + A[i]*lefts[i]*rights[i]) % mod
  }
  return res
}

// 解法三 暴力解法，中间很多重复判断子数组的情况
func sumSubarrayMins2(A []int) int {
  res, mod := 0, 1000000007
  for i := 0; i < len(A); i++ {
    stack := []int{}
    stack = append(stack, A[i])
    for j := i; j < len(A); j++ {
      if stack[len(stack)-1] >= A[j] {
        stack = stack[:len(stack)-1]
        stack = append(stack, A[j])
      }
      res += stack[len(stack)-1]
    }
  }
  return res % mod
}
```

910. Smallest Range II

题目

Given an array `A` of integers, for each integer `A[i]` we need to choose **either `x = -K` or `x = K`**, and add `x` to `A[i]` **(only once)**.

After this process, we have some array `B`.

Return the smallest possible difference between the maximum value of `B` and the minimum value of `B`.

Example 1:

```
Input: A = [1], K = 0
Output: 0
Explanation: B = [1]
```

Example 2:

```
Input: A = [0,10], K = 2
Output: 6
Explanation: B = [2,8]
```

Example 3:

```
Input: A = [1,3,6], K = 3
Output: 3
Explanation: B = [4,6,3]
```

Note:

1. `1 <= A.length <= 10000`
2. `0 <= A[i] <= 10000`
3. `0 <= K <= 10000`

题目大意

给你一个整数数组 A，对于每个整数 A[i]，可以选择 x = -K 或是 x = K （K 总是非负整数），并将 x 加到 A[i] 中。在此过程之后，得到数组 B。返回 B 的最大值和 B 的最小值之间可能存在的最小差值。

解题思路

- 简单题。先排序，找出 A 数组中最大的差值。然后循环扫一遍数组，利用双指针，选择 x = -K 或是 x = K ，每次选择都更新一次最大值和最小值之间的最小差值。循环一次以后便可以找到满足题意的答案。

代码

```
package leetcode

import "sort"

func smallestRangeII(A []int, K int) int {
  n := len(A)
  sort.Ints(A)
  res := A[n-1] - A[0]
  for i := 0; i < n-1; i++ {
    a, b := A[i], A[i+1]
    high := max(A[n-1]-K, a+K)
    low := min(A[0]+K, b-K)
    res = min(res, high-low)
  }
  return res
}

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

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

911. Online Election

题目

In an election, the `i`-th vote was cast for `persons[i]` at time `times[i]`.

Now, we would like to implement the following query function: `TopVotedCandidate.q(int t)` will return the number of the person that was leading the election at time `t`.

Votes cast at time `t` will count towards our query. In the case of a tie, the most recent vote (among tied candidates) wins.

Example 1:

```
Input: ["TopVotedCandidate","q","q","q","q","q","q"], [[[0,1,1,0,0,1,0],
[0,5,10,15,20,25,30]],[3],[12],[25],[15],[24],[8]]
Output: [null,0,1,1,0,0,1]
Explanation:
At time 3, the votes are [0], and 0 is leading.
At time 12, the votes are [0,1,1], and 1 is leading.
At time 25, the votes are [0,1,1,0,0,1], and 1 is leading (as ties go to the most
recent vote.)
This continues for 3 more queries at time 15, 24, and 8.
```

Note:

1. `1 <= persons.length = times.length <= 5000`
2. `0 <= persons[i] <= persons.length`
3. `times` is a strictly increasing array with all elements in `[0, 10^9]`.
4. `TopVotedCandidate.q` is called at most `10000` times per test case.
5. `TopVotedCandidate.q(int t)` is always called with `t >= times[0]`.

题目大意

在选举中，第 i 张票是在时间为 times[i] 时投给 persons[i] 的。

现在，我们想要实现下面的查询函数： TopVotedCandidate.q(int t) 将返回在 t 时刻主导选举的候选人的编号。

在 t 时刻投出的选票也将被计入我们的查询之中。在平局的情况下，最近获得投票的候选人将会获胜。

提示：

1. 1 <= persons.length = times.length <= 5000
2. 0 <= persons[i] <= persons.length
3. times 是严格递增的数组，所有元素都在 [0, 10^9] 范围中。
4. 每个测试用例最多调用 10000 次 TopVotedCandidate.q。
5. TopVotedCandidate.q(int t) 被调用时总是满足 t >= times[0]。

解题思路

- 给出一个 2 个数组，分别代表第 `i` 人在第 `t` 时刻获得的票数。需要实现一个查询功能的函数，查询在任意 `t` 时刻，输出谁的选票领先。
- `persons[]` 数组里面装的是获得选票人的编号，`times[]` 数组里面对应的是每个选票的时刻。`times[]` 数组默认是有序的，从小到大排列。先计算出每个时刻哪个人选票领先，放在一个数组中，实现查询函数的时候，只需要先对 `times[]` 数组二分搜索，找到比查询时间 `t` 小的最大时刻 `i`，再在选票领先的数组里面输出对应时刻领先的人的编号即可。

代码

```
package leetcode
```

```go
import (
  "sort"
)

// TopVotedCandidate define
type TopVotedCandidate struct {
  persons []int
  times   []int
}

// Constructor911 define
func Constructor911(persons []int, times []int) TopVotedCandidate {
  leaders, votes := make([]int, len(persons)), make([]int, len(persons))
  leader := persons[0]
  for i := 0; i < len(persons); i++ {
    p := persons[i]
    votes[p]++
    if votes[p] >= votes[leader] {
      leader = p
    }
    leaders[i] = leader
  }
  return TopVotedCandidate{persons: leaders, times: times}
}

// Q define
func (tvc *TopVotedCandidate) Q(t int) int {
  i := sort.Search(len(tvc.times), func(p int) bool { return tvc.times[p] > t })
  return tvc.persons[i-1]
}

/**
 * Your TopVotedCandidate object will be instantiated and called as such:
 * obj := Constructor(persons, times);
 * param_1 := obj.Q(t);
 */
```

914. X of a Kind in a Deck of Cards

题目

In a deck of cards, each card has an integer written on it.

Return `true` if and only if you can choose `X >= 2` such that it is possible to split the entire deck into 1 or more groups of cards, where:

- Each group has exactly `X` cards.

- All the cards in each group have the same integer.

Example 1:

```
Input: deck = [1,2,3,4,4,3,2,1]
Output: true
Explanation: Possible partition [1,1],[2,2],[3,3],[4,4].
```

Example 2:

```
Input: deck = [1,1,1,2,2,2,3,3]
Output: false´
Explanation: No possible partition.
```

Example 3:

```
Input: deck = [1]
Output: false
Explanation: No possible partition.
```

Example 4:

```
Input: deck = [1,1]
Output: true
Explanation: Possible partition [1,1].
```

Example 5:

```
Input: deck = [1,1,2,2,2,2]
Output: true
Explanation: Possible partition [1,1],[2,2],[2,2].
```

Constraints:

- `1 <= deck.length <= 10^4`
- `0 <= deck[i] < 10^4`

题目大意

给定一副牌，每张牌上都写着一个整数。此时，你需要选定一个数字 X，使我们可以将整副牌按下述规则分成 1 组或更多组：

- 每组都有 X 张牌。
- 组内所有的牌上都写着相同的整数。

仅当你可选的 X >= 2 时返回 true。

解题思路

- 给定一副牌，要求选出数字 X，使得每组都有 X 张牌，每组牌的数字都相同。当 X ≥ 2 的时候，输出 true。
- 通过分析题目，我们可以知道，只有当 X 为所有 count 的约数，即所有 count 的最大公约数的约数时，才存在可能的分组。因此我们只要求出所有 count 的最大公约数 g，判断 g 是否大于等于 2 即可，如果大于等于 2，则满足条件，否则不满足。
- 时间复杂度：O(NlogC)，其中 N 是卡牌的个数，C 是数组 deck 中数的范围，在本题中 C 的值为 10000。求两个数最大公约数的复杂度是 O(logC)，需要求最多 N - 1 次。空间复杂度：O(N + C) 或 O(N)。

代码

```go
package leetcode

func hasGroupsSizeX(deck []int) bool {
  if len(deck) < 2 {
    return false
  }
  m, g := map[int]int{}, -1
  for _, d := range deck {
    m[d]++
  }
  for _, v := range m {
    if g == -1 {
      g = v
    } else {
      g = gcd(g, v)
    }
  }
  return g >= 2
}

func gcd(a, b int) int {
  if a == 0 {
    return b
  }
  return gcd(b%a, a)
}
```

916. Word Subsets

题目

We are given two arrays `A` and `B` of words. Each word is a string of lowercase letters.

Now, say that word `b` is a subset of word `a` if every letter in `b` occurs in `a`, **including multiplicity**. For example, `"wrr"` is a subset of `"warrior"`, but is not a subset of `"world"`.

Now say a word `a` from `A` is *universal* if for every `b` in `B`, `b` is a subset of `a`.

Return a list of all universal words in `A`. You can return the words in any order.

Example 1:

```
Input:A = ["amazon","apple","facebook","google","leetcode"], B = ["e","o"]
Output:["facebook","google","leetcode"]
```

Example 2:

```
Input:A = ["amazon","apple","facebook","google","leetcode"], B = ["l","e"]
Output:["apple","google","leetcode"]
```

Example 3:

```
Input:A = ["amazon","apple","facebook","google","leetcode"], B = ["e","oo"]
Output:["facebook","google"]
```

Example 4:

```
Input:A = ["amazon","apple","facebook","google","leetcode"], B = ["lo","eo"]
Output:["google","leetcode"]
```

Example 5:

```
Input:A = ["amazon","apple","facebook","google","leetcode"], B = ["ec","oc","ceo"]
Output:["facebook","leetcode"]
```

Note:

1. `1 <= A.length, B.length <= 10000`
2. `1 <= A[i].length, B[i].length <= 10`
3. `A[i]` and `B[i]` consist only of lowercase letters.
4. All words in `A[i]` are unique: there isn't `i != j` with `A[i] == A[j]`.

题目大意

我们给出两个单词数组 A 和 B。每个单词都是一串小写字母。现在，如果 b 中的每个字母都出现在 a 中，包括重复出现的字母，那么称单词 b 是单词 a 的子集。 例如，“wrr” 是 “warrior” 的子集，但不是 “world” 的子集。如果对 B 中的每一个单词 b，b 都是 a 的子集，那么我们称 A 中的单词 a 是通用的。你可以按任意顺序以列表形式返回 A 中所有的通用单词。

解题思路

- 简单题。先统计出 B 数组中单词每个字母的频次，再在 A 数组中依次判断每个单词是否超过了这个频次，如果超过了即输出。

代码

```go
package leetcode

func wordSubsets(A []string, B []string) []string {
  var counter [26]int
  for _, b := range B {
    var m [26]int
    for _, c := range b {
      j := c - 'a'
      m[j]++
    }
    for i := 0; i < 26; i++ {
      if m[i] > counter[i] {
        counter[i] = m[i]
      }
    }
  }
  var res []string
  for _, a := range A {
    var m [26]int
    for _, c := range a {
      j := c - 'a'
      m[j]++
    }
    ok := true
    for i := 0; i < 26; i++ {
      if m[i] < counter[i] {
        ok = false
        break
      }
    }
    if ok {
      res = append(res, a)
    }
  }
  return res
}
```

918. Maximum Sum Circular Subarray

题目

Given a **circular array C** of integers represented by `A`, find the maximum possible sum of a non-empty subarray of **C**.

Here, a *circular array* means the end of the array connects to the beginning of the array. (Formally, `C[i] = A[i]` when `0 <= i < A.length`, and `C[i+A.length] = C[i]` when `i >= 0`.)

Also, a subarray may only include each element of the fixed buffer `A` at most once. (Formally, for a subarray `C[i], C[i+1], ..., C[j]`, there does not exist `i <= k1, k2 <= j` with `k1 % A.length = k2 % A.length`.)

Example 1:

```
Input: [1,-2,3,-2]
Output: 3
Explanation: Subarray [3] has maximum sum 3
```

Example 2:

```
Input: [5,-3,5]
Output: 10
Explanation: Subarray [5,5] has maximum sum 5 + 5 = 10
```

Example 3:

```
Input: [3,-1,2,-1]
Output: 4
Explanation: Subarray [2,-1,3] has maximum sum 2 + (-1) + 3 = 4
```

Example 4:

```
Input: [3,-2,2,-3]
Output: 3
Explanation: Subarray [3] and [3,-2,2] both have maximum sum 3
```

Example 5:

```
Input: [-2,-3,-1]
Output: -1
Explanation: Subarray [-1] has maximum sum -1
```

Note:

1. `-30000 <= A[i] <= 30000`
2. `1 <= A.length <= 30000`

题目大意

给定一个由整数数组 A 表示的环形数组 C，求 C 的非空子数组的最大可能和。在此处，环形数组意味着数组的末端将会与开头相连呈环状。（形式上，当0 <= i < A.length 时 C[i] = A[i]，而当 i >= 0 时 C[i+A.length] = C[i]）

此外，子数组最多只能包含固定缓冲区 A 中的每个元素一次。（形式上，对于子数组 C[i], C[i+1], ..., C[j]，不存在 i <= k1, k2 <= j 其中 k1 % A.length = k2 % A.length）

提示：

- -30000 <= A[i] <= 30000
- 1 <= A.length <= 30000

解题思路

- 给出一个环形数组，要求出这个环形数组中的连续子数组的最大和。
- 拿到这题最先想到的思路是把这个数组再拼接一个，在这两个数组中查找连续子数组的最大和。这种做法是错误的，例如在 `[5,-3,5]` 这个数组中会得出 `7` 的结果，但是实际结果是 `10` 。那么这题怎么做呢？仔细分析可以得到，环形数组的最大连续子段和有两种情况，第一种情况是这个连续子段就出现在数组中， 不存在循环衔接的情况。针对这种情况就比较简单，用 `kadane` 算法(也是动态规划的思想)，`O(n)` 的时间复杂度就可以求出结果。第二种情况是这个连续的子段出现在跨数组的情况，即会出现首尾相连的情况。要想找到这样一个连续子段，可以反向考虑。想找到跨段的连续子段，那么这个数组剩下的这一段就是不跨段的连续子段。想要跨段的子段和最大，那么剩下的这段连续子段和最小。如果能找到这个数组的每个元素取相反数组成的数组中的最大连续子段和，那么反过来就能找到原数组的连续子段和最小。举个例子：`[1, 2, -3, -4, 5]` ，取它的每个元素的相反数 `[-1, -2, 3, 4, -5]`，构造的数组中最大连续子段和是 `3 + 4 = 7`，由于取了相反数，所以可以得到原数组中最小连续子段和是 `-7` 。所以跨段的最大连续子段和就是剩下的那段 `[1,2,5]`。
- 还有一些边界的情况，例如，`[1, 2, -2, -3, 5, 5, -4, 6]` 和 `[1, 2, -2, -3, 5, 5, -4, 8]`，所以还需要比较一下情况一和情况二的值，它们两者最大值才是最终环形数组的连续子数组的最大和。

代码

```go
package leetcode

import "math"

func maxSubarraySumCircular(A []int) int {
  n, sum := len(A), 0
  for _, v := range A {
    sum += v
  }
  kad := kadane(A)
  for i := 0; i < n; i++ {
    A[i] = -A[i]
  }
  negativeMax := kadane(A)
  if sum+negativeMax <= 0 {
    return kad
  }
  return max(kad, sum+negativeMax)
}

func kadane(a []int) int {
  n, MaxEndingHere, maxSoFar := len(a), a[0], math.MinInt32
  for i := 1; i < n; i++ {
    MaxEndingHere = max(a[i], MaxEndingHere+a[i])
```

```
    maxSoFar = max(MaxEndingHere, maxSoFar)
  }
  return maxSoFar
}
```

920. Number of Music Playlists

题目

Your music player contains `N` different songs and she wants to listen to `L` (not necessarily different) songs during your trip. You create a playlist so that:

- Every song is played at least once
- A song can only be played again only if `K` other songs have been played

Return the number of possible playlists. **As the answer can be very large, return it modulo `10^9 + 7`**.

Example 1:

```
Input: N = 3, L = 3, K = 1
Output: 6
Explanation: There are 6 possible playlists. [1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1].
```

Example 2:

```
Input: N = 2, L = 3, K = 0
Output: 6
Explanation: There are 6 possible playlists. [1, 1, 2], [1, 2, 1], [2, 1, 1], [2, 2, 1], [2, 1, 2], [1, 2, 2]
```

Example 3:

```
Input: N = 2, L = 3, K = 1
Output: 2
Explanation: There are 2 possible playlists. [1, 2, 1], [2, 1, 2]
```

Note:

1. `0 <= K < N <= L <= 100`

题目大意

你的音乐播放器里有 N 首不同的歌，在旅途中，你的旅伴想要听 L 首歌（不一定不同，即，允许歌曲重复）。请你为她按如下规则创建一个播放列表：

- 每首歌至少播放一次。

- 一首歌只有在其他 K 首歌播放完之后才能再次播放。

返回可以满足要求的播放列表的数量。由于答案可能非常大，请返回它模 10^9 + 7 的结果。

提示：

- 0 <= K < N <= L <= 100

解题思路

- 简化抽象一下题意，给 N 个数，要求从这 N 个数里面组成一个长度为 L 的序列，并且相同元素的间隔不能小于 K 个数。问总共有多少组组成方法。
- 一拿到题，会觉得这一题是三维 DP，因为存在 3 个变量，但是实际考虑一下，可以降一维。我们先不考虑 K 的限制，只考虑 N 和 L。定义 `dp[i][j]` 代表播放列表里面有 `i` 首歌，其中包含 `j` 首不同的歌曲，那么题目要求的最终解存在 `dp[L][N]` 中。考虑 `dp[i][j]` 的递归公式，音乐列表当前需要组成 `i` 首歌，有 2 种方式可以得到，由 `i - 1` 首歌的列表中添加一首列表中**不存在**的新歌曲，或者由 `i - 1` 首歌的列表中添加一首列表中**已经存在**的歌曲。即，`dp[i][j]` 可以由 `dp[i - 1][j - 1]` 得到，也可以由 `dp[i - 1][j]` 得到。如果是第一种情况，添加一首新歌，那么新歌有 N - (j - 1) 首，如果是第二种情况，添加一首已经存在的歌，歌有 j 首，所以状态转移方程是 `dp[i][j] = dp[i - 1][j - 1] * ( N - ( j - 1 ) ) + dp[i - 1][j] * j` 。但是这个方程是在不考虑 K 的限制条件下得到的，距离满足题意还差一步。接下来需要考虑加入 K 这个限制条件以后，状态转移方程该如何推导。
- 如果是添加一首新歌，是不受 K 限制的，所以 `dp[i - 1][j - 1] * ( N - ( j - 1 ) )` 这里不需要变化。如果是添加一首存在的歌曲，这个时候就会受到 K 的限制了。如果当前播放列表里面的歌曲有 `j` 首，并且 `j > K`，那么选择歌曲只能从 `j - K` 里面选，因为不能选择 `j - 1` 到 `j - k` 的这些歌，选择了就不满足重复的歌之间间隔不能小于 `K` 的限制条件了。那 j ≤ K 呢？这个时候一首歌都不能选，因为歌曲数都没有超过 K，当然不能再选择重复的歌曲。(选择了就再次不满足重复的歌之间间隔不能小于 `K` 的限制条件了)。
  经过上述分析，可以得到最终的状态转移方程：
  {{< katex display >}}
  dp[i][j]= \begin{matrix} \left{ \begin{array}{lr} dp[i - 1][j - 1] * (N - (j - 1)) + dp[i - 1][j] * (j - k) , & {j > k}\ dp[i - 1][j - 1] * (N - (j - 1)), & {j \leq k} \end{array} \right. \end{matrix}
  {{< /katex >}}
- 上面的式子可以合并简化成下面这个式子：`dp[i][j] = dp[i - 1][j - 1]*(N - (j - 1)) + dp[i-1][j]*max(j-K, 0)`，递归初始值 `dp[0][0] = 1`。

代码

```go
package leetcode

func numMusicPlaylists(N int, L int, K int) int {
  dp, mod := make([][]int, L+1), 1000000007
  for i := 0; i < L+1; i++ {
    dp[i] = make([]int, N+1)
  }
  dp[0][0] = 1
  for i := 1; i <= L; i++ {
```

```
    for j := 1; j <= N; j++ {
      dp[i][j] = (dp[i-1][j-1] * (N - (j - 1))) % mod
      if j > K {
        dp[i][j] = (dp[i][j] + (dp[i-1][j]*(j-K))%mod) % mod
      }
    }
  }
  return dp[L][N]
}
```

921. Minimum Add to Make Parentheses Valid

题目

Given a string S of '(' and ')' parentheses, we add the minimum number of parentheses ('(' or ')', and in any positions) so that the resulting parentheses string is valid.

Formally, a parentheses string is valid if and only if:

- It is the empty string, or
- It can be written as AB (A concatenated with B), where A and B are valid strings, or
- It can be written as (A), where A is a valid string.
  Given a parentheses string, return the minimum number of parentheses we must add to make the resulting string valid.

Example 1:

```
Input: "())"
Output: 1
```

Example 2:

```
Input: "((("
Output: 3
```

Example 3:

```
Input: "()"
Output: 0
```

Example 4:

```
Input: "()))(("
Output: 4
```

Note:

1. S.length <= 1000
2. S only consists of '(' and ')' characters.

题目大意

给一个括号的字符串，如果能在这个括号字符串中的任意位置添加括号，问能使得这串字符串都能完美匹配的最少添加数是多少。

解题思路

这题也是栈的题目，利用栈进行括号匹配。最后栈里剩下几个括号，就是最少需要添加的数目。

代码

```go
package leetcode

func minAddToMakeValid(S string) int {
  if len(S) == 0 {
    return 0
  }
  stack := make([]rune, 0)
  for _, v := range S {
    if v == '(' {
      stack = append(stack, v)
    } else if (v == ')') && len(stack) > 0 && stack[len(stack)-1] == '(' {
      stack = stack[:len(stack)-1]
    } else {
      stack = append(stack, v)
    }
  }
  return len(stack)
}
```

922. Sort Array By Parity II

题目

Given an array A of non-negative integers, half of the integers in A are odd, and half of the integers are even.

Sort the array so that whenever A[i] is odd, i is odd; and whenever A[i] is even, i is even.

You may return any answer array that satisfies this condition.

Example 1:

```
Input: [4,2,5,7]
Output: [4,5,2,7]
Explanation: [4,7,2,5], [2,5,4,7], [2,7,4,5] would also have been accepted.
```

Note:

- 2 <= A.length <= 20000
- A.length % 2 == 0
- 0 <= A[i] <= 1000

题目大意

要求数组中奇数下标位置上放奇数，偶数下标位置上放偶数。

解题思路

这题比较简单，用两个下标控制奇数，偶数放置在哪个下标即可。奇数奇数之间，偶数偶数之间的顺序可以是无序的。

代码

```go
package leetcode

func sortArrayByParityII(A []int) []int {
  if len(A) == 0 || len(A)%2 != 0 {
    return []int{}
  }
  res := make([]int, len(A))
  oddIndex := 1
  evenIndex := 0
  for i := 0; i < len(A); i++ {
    if A[i]%2 == 0 {
      res[evenIndex] = A[i]
```

```
      evenIndex += 2
    } else {
      res[oddIndex] = A[i]
      oddIndex += 2
    }
  }
  return res
}
```

923. 3Sum With Multiplicity

题目

Given an integer array A, and an integer target, return the number of tuples i, j, k such that i < j < k and A[i] + A[j] + A[k] == target.

As the answer can be very large, return it modulo 10^9 + 7.

Example 1:

```
Input: A = [1,1,2,2,3,3,4,4,5,5], target = 8
Output: 20
Explanation:
Enumerating by the values (A[i], A[j], A[k]):
(1, 2, 5) occurs 8 times;
(1, 3, 4) occurs 8 times;
(2, 2, 4) occurs 2 times;
(2, 3, 3) occurs 2 times.
```

Example 2:

```
Input: A = [1,1,2,2,2,2], target = 5
Output: 12
Explanation:
A[i] = 1, A[j] = A[k] = 2 occurs 12 times:
We choose one 1 from [1,1] in 2 ways,
and two 2s from [2,2,2,2] in 6 ways.
```

Note:

- 3 <= A.length <= 3000
- 0 <= A[i] <= 100
- 0 <= target <= 300

题目大意

这道题是第 15 题的升级版。给出一个数组，要求找到 3 个数相加的和等于 target 的解组合的个数，并且要求 i < j < k。解的组合个数不需要去重，相同数值不同下标算不同解(这里也是和第 15 题的区别)

解题思路

这一题大体解法和第 15 题一样的，只不过算所有解组合的时候需要一点排列组合的知识，如果取 3 个一样的数，需要计算 C n 3，去 2 个相同的数字的时候，计算 C n 2，取一个数字就正常计算。最后所有解的个数都加起来就可以了。

代码

```go
package leetcode

import (
  "sort"
)

func threeSumMulti(A []int, target int) int {
  mod := 1000000007
  counter := map[int]int{}
  for _, value := range A {
    counter[value]++
  }

  uniqNums := []int{}
  for key := range counter {
    uniqNums = append(uniqNums, key)
  }
  sort.Ints(uniqNums)

  res := 0
  for i := 0; i < len(uniqNums); i++ {
    ni := counter[uniqNums[i]]
    if (uniqNums[i]*3 == target) && counter[uniqNums[i]] >= 3 {
      res += ni * (ni - 1) * (ni - 2) / 6
    }
    for j := i + 1; j < len(uniqNums); j++ {
      nj := counter[uniqNums[j]]
      if (uniqNums[i]*2+uniqNums[j] == target) && counter[uniqNums[i]] > 1 {
        res += ni * (ni - 1) / 2 * nj
      }
      if (uniqNums[j]*2+uniqNums[i] == target) && counter[uniqNums[j]] > 1 {
        res += nj * (nj - 1) / 2 * ni
      }
      c := target - uniqNums[i] - uniqNums[j]
```

```
        if c > uniqNums[j] && counter[c] > 0 {
          res += ni * nj * counter[c]
        }
      }
    }
    return res % mod
  }
```

924. Minimize Malware Spread

题目

In a network of nodes, each node `i` is directly connected to another node `j` if and only if `graph[i][j] = 1`.

Some nodes `initial` are initially infected by malware. Whenever two nodes are directly connected and at least one of those two nodes is infected by malware, both nodes will be infected by malware. This spread of malware will continue until no more nodes can be infected in this manner.

Suppose `M(initial)` is the final number of nodes infected with malware in the entire network, after the spread of malware stops.

We will remove one node from the initial list. Return the node that if removed, would minimize `M(initial)`. If multiple nodes could be removed to minimize `M(initial)`, return such a node with the smallest index.

Note that if a node was removed from the `initial` list of infected nodes, it may still be infected later as a result of the malware spread.

Example 1:

```
Input: graph = [[1,1,0],[1,1,0],[0,0,1]], initial = [0,1]
Output: 0
```

Example 2:

```
Input: graph = [[1,0,0],[0,1,0],[0,0,1]], initial = [0,2]
Output: 0
```

Example 3:

```
Input: graph = [[1,1,1],[1,1,1],[1,1,1]], initial = [1,2]
Output: 1
```

Note:

1. `1 < graph.length = graph[0].length <= 300`

2. `0 <= graph[i][j] == graph[j][i] <= 1`
3. `graph[i][i] = 1`
4. `1 <= initial.length < graph.length`
5. `0 <= initial[i] < graph.length`

题目大意

在节点网络中，只有当 graph[i][j] = 1 时，每个节点 i 能够直接连接到另一个节点 j。一些节点 initial 最初被恶意软件感染。只要两个节点直接连接，且其中至少一个节点受到恶意软件的感染，那么两个节点都将被恶意软件感染。这种恶意软件的传播将继续，直到没有更多的节点可以被这种方式感染。假设 M(initial) 是在恶意软件停止传播之后，整个网络中感染恶意软件的最终节点数。我们可以从初始列表中删除一个节点。如果移除这一节点将最小化 M(initial)， 则返回该节点。如果有多个节点满足条件，就返回索引最小的节点。请注意，如果某个节点已从受感染节点的列表 initial 中删除，它以后可能仍然因恶意软件传播而受到感染。

提示：

- 1 < graph.length = graph[0].length <= 300
- 0 <= graph[i][j] == graph[j][i] <= 1
- graph[i][i] = 1
- 1 <= initial.length < graph.length
- 0 <= initial[i] < graph.length

解题思路

- 给出一个节点之间的关系图，如果两个节点是连通的，那么病毒软件就会感染到连通的所有节点。现在如果想移除一个病毒节点，能最大减少感染，请问移除哪个节点？如果多个节点都能减少感染量，优先移除序号偏小的那个节点。
- 这一题一看就是考察的并查集。利用节点的连通关系，把题目中给的所有节点都 `union()` 起来，然后依次统计每个集合内有多少个点。最后扫描一次 initial 数组，选出这个数组中节点小的并且所在集合节点多，这个节点就是最终答案。

代码

```
package leetcode

import (
  "math"

  "github.com/halfrost/LeetCode-Go/template"
)

func minMalwareSpread(graph [][]int, initial []int) int {
  if len(initial) == 0 {
    return 0
  }
  uf, maxLen, maxIdx, uniqInitials, compMap := template.UnionFindCount{},
math.MinInt32, -1, map[int]int{}, map[int][]int{}
```

```go
  uf.Init(len(graph))
  for i := range graph {
    for j := i + 1; j < len(graph); j++ {
      if graph[i][j] == 1 {
        uf.Union(i, j)
      }
    }
  }
  for _, i := range initial {
    compMap[uf.Find(i)] = append(compMap[uf.Find(i)], i)
  }
  for _, v := range compMap {
    if len(v) == 1 {
      uniqInitials[v[0]] = v[0]
    }
  }
  if len(uniqInitials) == 0 {
    smallestIdx := initial[0]
    for _, i := range initial {
      if i < smallestIdx {
        smallestIdx = i
      }
    }
    return smallestIdx
  }
  for _, i := range initial {
    if _, ok := uniqInitials[i]; ok {
      size := uf.Count()[uf.Find(i)]
      if maxLen < size {
        maxLen, maxIdx = size, i
      } else if maxLen == size {
        if i < maxIdx {
          maxIdx = i
        }
      }
    }
  }
  return maxIdx
}
```

925. Long Pressed Name

题目

Your friend is typing his name into a keyboard. Sometimes, when typing a character c, the key might get long pressed, and the character will be typed 1 or more times.

You examine the typed characters of the keyboard. Return True if it is possible that it was your friends name, with some characters (possibly none) being long pressed.

Example 1:

```
Input: name = "alex", typed = "aaleex"
Output: true
Explanation: 'a' and 'e' in 'alex' were long pressed.
```

Example 2:

```
Input: name = "saeed", typed = "ssaaedd"
Output: false
Explanation: 'e' must have been pressed twice, but it wasn't in the typed output.
```

Example 3:

```
Input: name = "leelee", typed = "lleeelee"
Output: true
```

Example 4:

```
Input: name = "laiden", typed = "laiden"
Output: true
Explanation: It's not necessary to long press any character.
```

Note:

1. name.length <= 1000
2. typed.length <= 1000
3. The characters of name and typed are lowercase letters.

题目大意

给定 2 个字符串，后者的字符串中包含前者的字符串。比如在打字的过程中，某个字符会多按了几下。判断后者字符串是不是比前者字符串存在这样的“长按”键盘的情况。

解题思路

- 这一题也可以借助滑动窗口的思想。2 个字符串一起比较，如果遇到有相同的字符串，窗口继续往后滑动。直到遇到了第一个不同的字符，如果遇到两个字符串不相等的情况，可以直接返回 false。具体实现见代码。
- 这一题的测试用例修改过一次，需要注意我这里写的第二组测试用例，当 name 结束以后，如果 typed 还有多余的不同的字符，这种情况要输出 false 的。具体见 test 文件里面的第二组，第三组，第四组测试用例。

代码

```go
package leetcode

func isLongPressedName(name string, typed string) bool {
  if len(name) == 0 && len(typed) == 0 {
    return true
  }
  if (len(name) == 0 && len(typed) != 0) || (len(name) != 0 && len(typed) == 0) {
    return false
  }
  i, j := 0, 0
  for i < len(name) && j < len(typed) {
    if name[i] != typed[j] {
      return false
    }
    for i < len(name) && j < len(typed) && name[i] == typed[j] {
      i++
      j++
    }
    for j < len(typed) && typed[j] == typed[j-1] {
      j++
    }
  }
  return i == len(name) && j == len(typed)
}
```

927. Three Equal Parts

题目

Given an array `A` of `0`s and `1`s, divide the array into 3 non-empty parts such that all of these parts represent the same binary value.

If it is possible, return **any** `[i, j]` with `i+1 < j`, such that:

- `A[0], A[1], ..., A[i]` is the first part;
- `A[i+1], A[i+2], ..., A[j-1]` is the second part, and
- `A[j], A[j+1], ..., A[A.length - 1]` is the third part.
- All three parts have equal binary value.

If it is not possible, return `[-1, -1]`.

Note that the entire part is used when considering what binary value it represents. For example, `[1,1,0]` represents `6` in decimal, not `3`. Also, leading zeros are allowed, so `[0,1,1]` and `[1,1]` represent the same value.

Example 1:

```
Input: [1,0,1,0,1]
Output: [0,3]
```

Example 2:

```
Input: [1,1,0,1,1]
Output: [-1,-1]
```

Note:

1. `3 <= A.length <= 30000`
2. `A[i] == 0` or `A[i] == 1`

题目大意

给定一个由 0 和 1 组成的数组 A，将数组分成 3 个非空的部分，使得所有这些部分表示相同的二进制值。如果可以做到，请返回任何 [i, j]，其中 i+1 < j，这样一来：

- A[0], A[1], ..., A[i] 组成第一部分；
- A[i+1], A[i+2], ..., A[j-1] 作为第二部分；
- A[j], A[j+1], ..., A[A.length - 1] 是第三部分。
- 这三个部分所表示的二进制值相等。

如果无法做到，就返回 [-1, -1]。

注意，在考虑每个部分所表示的二进制时，应当将其看作一个整体。例如，[1,1,0] 表示十进制中的 6，而不会是 3。此外，前导零也是被允许的，所以 [0,1,1] 和 [1,1] 表示相同的值。

提示：

1. 3 <= A.length <= 30000
2. A[i] == 0 或 A[i] == 1

解题思路

- 给出一个数组，数组里面只包含 0 和 1，要求找到 2 个分割点，使得分成的 3 个子数组的二进制是完全一样

的。

- 这一题的解题思路不难，按照题意模拟即可。先统计 1 的个数 total，然后除以 3 就是每段 1 出现的个数。有一些特殊情况需要额外判断一下，例如没有 1 的情况，那么只能首尾分割。1 个个数不是 3 的倍数，也无法分割成满足题意。然后找到第一个 1 的下标，然后根据 total/3 找到 mid，第一个分割点。再往后移动，找到第二个分割点。找到这 3 个点以后，同步的移动这 3 个点，移动中判断这 3 个下标对应的数值是否相等，如果都相等，并且最后一个点能移动到末尾，就算找到了满足题意的解了。

代码

```go
package leetcode

func threeEqualParts(A []int) []int {
  n, ones, i, count := len(A), 0, 0, 0
  for _, a := range A {
    ones += a
  }
  if ones == 0 {
    return []int{0, n - 1}
  }
  if ones%3 != 0 {
    return []int{-1, -1}
  }
  k := ones / 3
  for i < n {
    if A[i] == 1 {
      break
    }
    i++
  }
  start, j := i, i
  for j < n {
    count += A[j]
    if count == k+1 {
      break
    }
    j++
  }
  mid := j
  j, count = 0, 0
  for j < n {
    count += A[j]
    if count == 2*k+1 {
      break
    }
    j++
  }
  end := j
  for end < n && A[start] == A[mid] && A[mid] == A[end] {
```

```
    start++
    mid++
    end++
  }
  if end == n {
    return []int{start - 1, mid}
  }
  return []int{-1, -1}
}
```

928. Minimize Malware Spread II

题目

(This problem is the same as *Minimize Malware Spread*, with the differences bolded.)

In a network of nodes, each node `i` is directly connected to another node `j` if and only if `graph[i][j] = 1`.

Some nodes `initial` are initially infected by malware. Whenever two nodes are directly connected and at least one of those two nodes is infected by malware, both nodes will be infected by malware. This spread of malware will continue until no more nodes can be infected in this manner.

Suppose `M(initial)` is the final number of nodes infected with malware in the entire network, after the spread of malware stops.

We will remove one node from the initial list, **completely removing it and any connections from this node to any other node**. Return the node that if removed, would minimize `M(initial)`. If multiple nodes could be removed to minimize `M(initial)`, return such a node with the smallest index.

Example 1:

```
Input: graph = [[1,1,0],[1,1,0],[0,0,1]], initial = [0,1]
Output: 0
```

Example 2:

```
Input: graph = [[1,1,0],[1,1,1],[0,1,1]], initial = [0,1]
Output: 1
```

Example 3:

```
Input: graph = [[1,1,0,0],[1,1,1,0],[0,1,1,1],[0,0,1,1]], initial = [0,1]
Output: 1
```

Note:

1. `1 < graph.length = graph[0].length <= 300`
2. `0 <= graph[i][j] == graph[j][i] <= 1`
3. `graph[i][i] = 1`
4. `1 <= initial.length < graph.length`
5. `0 <= initial[i] < graph.length`

题目大意

(这个问题与 尽量减少恶意软件的传播 是一样的，不同之处用粗体表示。)在节点网络中，只有当 graph[i][j] = 1 时，每个节点 i 能够直接连接到另一个节点 j。一些节点 initial 最初被恶意软件感染。只要两个节点直接连接，且其中至少一个节点受到恶意软件的感染，那么两个节点都将被恶意软件感染。这种恶意软件的传播将继续，直到没有更多的节点可以被这种方式感染。假设 M(initial) 是在恶意软件停止传播之后，整个网络中感染恶意软件的最终节点数。我们可以从初始列表中删除一个节点，并完全移除该节点以及从该节点到任何其他节点的任何连接。如果移除这一节点将最小化 M(initial)， 则返回该节点。如果有多个节点满足条件，就返回索引最小的节点。

提示：

- 1 < graph.length = graph[0].length <= 300
- 0 <= graph[i][j] == graph[j][i] <= 1
- graph[i][i] = 1
- 1 <= initial.length < graph.length
- 0 <= initial[i] < graph.length

解题思路

- 这一题是第 924 题的加强版。给出一个节点之间的关系图，如果两个节点是连通的，那么病毒软件就会感染到连通的所有节点。现在如果想**完全彻底**移除一个病毒节点，能最大减少感染，请问移除哪个节点？如果多个节点都能减少感染量，优先移除序号偏小的那个节点。这一题的输入输出要求和第 924 题是完全一样的，区别在于第 924 题实际上是要求把一个病毒节点变成非病毒节点，而这道题是完全删除一个病毒节点以及它连接的所有边。
- 这一题考察的是并查集。当然用 DFS 也可以解答这一题。并查集的做法如下，首先先将所有的病毒节点去掉，然后将所有连通块合并成一个节点。因为一个连通集合中的节点，要么全部被感染，要么全部不被感染，所以可以把每个集合整体考虑。然后统计所有集合直接相邻的病毒节点的个数。对于一个集合来说：
    1. 如果直接相邻的病毒节点的个数为 0，则一定不会被感染，忽略这种情况；
    2. 如果直接相邻的病毒节点的个数为 1，则将该病毒节点删除后，整个连通块就可以避免被感染，这种情况是我们寻找的答案；
    3. 如果直接相邻的病毒节点的个数大于等于2，则不管删除哪个病毒节点，该连通块都仍会被感染，忽略这种情况；
- 所以只需在所有第二种连通块（直接相邻的病毒节点的个数为 1 的连通块）中，找出节点个数最多的连通块，与它相邻的病毒节点就是我们要删除的节点；如果有多个连通块节点个数相同，再找出与之对应的编号最小的病毒节点即可。

代码

```
package leetcode
```

```go
import (
  "math"

  "github.com/halfrost/LeetCode-Go/template"
)

func minMalwareSpread2(graph [][]int, initial []int) int {
  if len(initial) == 0 {
    return 0
  }
  uf, minIndex, count, countMap, malwareMap, infectMap := template.UnionFind{},
initial[0], math.MinInt64, map[int]int{}, map[int]int{}, map[int]map[int]int{}
  for _, v := range initial {
    malwareMap[v]++
  }
  uf.Init(len(graph))
  for i := range graph {
    for j := range graph[i] {
      if i == j {
        break
      }
      if graph[i][j] == 1 && malwareMap[i] == 0 && malwareMap[j] == 0 {
        uf.Union(i, j)
      }
    }
  }
  for i := 0; i < len(graph); i++ {
    countMap[uf.Find(i)]++
  }
  // 记录每个集合和直接相邻病毒节点的个数
  for _, i := range initial {
    for j := 0; j < len(graph); j++ {
      if malwareMap[j] == 0 && graph[i][j] == 1 {
        p := uf.Find(j)
        if _, ok := infectMap[p]; ok {
          infectMap[p][i] = i
        } else {
          tmp := map[int]int{}
          tmp[i] = i
          infectMap[p] = tmp
        }
      }
    }
  }
  // 选出病毒节点中序号最小的
  for _, v := range initial {
    minIndex = min(minIndex, v)
  }
  for i, v := range infectMap {
```

```
    // 找出只和一个病毒节点相连通的
    if len(v) == 1 {
      tmp := countMap[uf.Find(i)]
      keys := []int{}
      for k := range v {
        keys = append(keys, k)
      }
      if count == tmp && minIndex > keys[0] {
        minIndex = keys[0]
      }
      if count < tmp {
        minIndex = keys[0]
        count = tmp
      }
    }
  }
  return minIndex
}
```

930. Binary Subarrays With Sum

题目

In an array A of 0s and 1s, how many non-empty subarrays have sum S?

Example 1:

```
Input: A = [1,0,1,0,1], S = 2
Output: 4
Explanation:
The 4 subarrays are bolded below:
[1,0,1,0,1]
[1,0,1,0,1]
[1,0,1,0,1]
[1,0,1,0,1]
```

Note:

- A.length <= 30000
- 0 <= S <= A.length
- A[i] is either 0 or 1.

题目大意

给定一个数组，数组里面的元素只有 0 和 1 两种。问这个数组有多少个和为 S 的子数组。

解题思路

这道题也是滑动窗口的题目。不断的加入右边的值，直到总和等于 S。[i,j] 区间内的和可以等于 [0,j] 的和减去 [0,i-1] 的和。在 freq 中不断的记下能使得和为 sum 的组合方法数，例如 freq[1] = 2 ，代表和为 1 有两种组合方法，(可能是 1 和 1，0 或者 0，1，这道题只管组合总数，没要求输出具体的组合对)。这道题的做法就是不断的累加，如果遇到比 S 多的情况，多出来的值就在 freq 中查表，看多出来的值可能是由几种情况构成的。一旦和与 S 相等以后，之后比 S 多出来的情况会越来越多(因为在不断累积，总和只会越来越大)，不断的查 freq 表就可以了。

代码

```go
package leetcode

import "fmt"

func numSubarraysWithSum(A []int, S int) int {
  freq, sum, res := make([]int, len(A)+1), 0, 0
  freq[0] = 1
  for _, v := range A {
    t := sum + v - S
    if t >= 0 {
      // 总和有多余的，需要减去 t，除去的方法有 freq[t] 种
      res += freq[t]
    }
    sum += v
    freq[sum]++
    fmt.Printf("freq = %v sum = %v res = %v t = %v\n", freq, sum, res, t)
  }
  return res
}
```

933. Number of Recent Calls

题目

Write a class `RecentCounter` to count recent requests.

It has only one method: `ping(int t)`, where t represents some time in milliseconds.

Return the number of `ping`s that have been made from 3000 milliseconds ago until now.

Any ping with time in `[t - 3000, t]` will count, including the current ping.

It is guaranteed that every call to `ping` uses a strictly larger value of `t` than before.

Example 1:

```
Input: inputs = ["RecentCounter","ping","ping","ping","ping"], inputs = [[],[1],[100],
[3001],[3002]]
Output: [null,1,2,3,3]
```

Note:

1. Each test case will have at most `10000` calls to `ping`.
2. Each test case will call `ping` with strictly increasing values of `t`.
3. Each call to ping will have `1 <= t <= 10^9`.

题目大意

写一个 RecentCounter 类来计算最近的请求。它只有一个方法：ping(int t)，其中 t 代表以毫秒为单位的某个时间。返回从 3000 毫秒前到现在的 ping 数。任何处于 [t - 3000, t] 时间范围之内的 ping 都将会被计算在内，包括当前（指 t 时刻）的 ping。保证每次对 ping 的调用都使用比之前更大的 t 值。

提示：

- 每个测试用例最多调用 10000 次 ping。
- 每个测试用例会使用严格递增的 t 值来调用 ping。
- 每次调用 ping 都有 1 <= t <= 10^9。

解题思路

- 要求设计一个类，可以用 `ping(t)` 的方法，计算 [t-3000, t] 区间内的 ping 数。t 是毫秒。
- 这一题比较简单，`ping()` 方法用二分搜索即可。

代码

```go
type RecentCounter struct {
  list []int
}

func Constructor933() RecentCounter {
  return RecentCounter{
    list: []int{},
  }
}

func (this *RecentCounter) Ping(t int) int {
  this.list = append(this.list, t)
  index := sort.Search(len(this.list), func(i int) bool { return this.list[i] >= t-3000
})
  if index < 0 {
    index = -index - 1
  }
  return len(this.list) - index
```

```
}

/**
 * Your RecentCounter object will be instantiated and called as such:
 * obj := Constructor();
 * param_1 := obj.Ping(t);
 */
```

938. Range Sum of BST

题目

Given the `root` node of a binary search tree, return *the sum of values of all nodes with a value in the range* `[low, high]`.

Example 1:

```
Input: root = [10,5,15,3,7,null,18], low = 7, high = 15
Output: 32
```

Example 2:

```
Input: root = [10,5,15,3,7,13,18,1,null,6], low = 6, high = 10
Output: 23
```

Constraints:

- The number of nodes in the tree is in the range `[1, 2 * 10^4]`.
- `1 <= Node.val <= 10^5`
- `1 <= low <= high <= 10^5`
- All `Node.val` are **unique**.

题目大意

给定二叉搜索树的根结点 root，返回值位于范围 [low, high] 之间的所有结点的值的和。

解题思路

- 简单题。因为二叉搜索树的有序性，先序遍历即为有序。遍历过程中判断节点值是否位于区间范围内，在区间内就累加，不在区间内节点就不管。最终输出累加和。

代码

```
package leetcode
```

```go
import (
  "github.com/halfrost/LeetCode-Go/structures"
)

// TreeNode define
type TreeNode = structures.TreeNode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

func rangeSumBST(root *TreeNode, low int, high int) int {
  res := 0
  preOrder(root, low, high, &res)
  return res
}

func preOrder(root *TreeNode, low, high int, res *int) {
  if root == nil {
    return
  }
  if low <= root.Val && root.Val <= high {
    *res += root.Val
  }
  preOrder(root.Left, low, high, res)
  preOrder(root.Right, low, high, res)
}
```

942. DI String Match

题目

Given a string `S` that **only** contains "I" (increase) or "D" (decrease), let `N = S.length`.

Return **any** permutation `A` of `[0, 1, ..., N]` such that for all `i = 0, ..., N-1`:

- If `S[i] == "I"`, then `A[i] < A[i+1]`
- If `S[i] == "D"`, then `A[i] > A[i+1]`

Example 1:

```
Input: "IDID"
Output: [0,4,1,3,2]
```

Example 2:

```
Input: "III"
Output: [0,1,2,3]
```

Example 3:

```
Input: "DDI"
Output: [3,2,0,1]
```

Note:

1. `1 <= S.length <= 10000`
2. `S` only contains characters `"I"` or `"D"`.

题目大意

给定只含 "I"（增大）或 "D"（减小）的字符串 S ， 令 N = S.length。返回 [0, 1, ..., N] 的任意排列 A 使得对于所有 i = 0, ..., N-1， 都有：

- 如果 S[i] == "I"，那么 A[i] < A[i+1]
- 如果 S[i] == "D"，那么 A[i] > A[i+1]

解题思路

- 给出一个字符串，字符串中只有字符 `"I"` 和字符 `"D"`。字符 `"I"` 代表 `A[i] < A[i+1]`，字符 `"D"` 代表 `A[i] > A[i+1]` ，要求找到满足条件的任意组合。
- 这一题也是水题，取出字符串长度即是最大数的数值，然后按照题意一次排出最终数组即可。

代码

```go
package leetcode

func diStringMatch(S string) []int {
  result, maxNum, minNum, index := make([]int, len(S)+1), len(S), 0, 0
  for _, ch := range S {
    if ch == 'I' {
      result[index] = minNum
      minNum++
    } else {
```

```
      result[index] = maxNum
      maxNum--
    }
    index++
  }
  result[index] = minNum
  return result
}
```

946. Validate Stack Sequences

题目

Given two sequences pushed and popped with distinct values, return true if and only if this could have been the result of a sequence of push and pop operations on an initially empty stack.

Example 1:

```
Input: pushed = [1,2,3,4,5], popped = [4,5,3,2,1]
Output: true
Explanation: We might do the following sequence:
push(1), push(2), push(3), push(4), pop() -> 4,
push(5), pop() -> 5, pop() -> 3, pop() -> 2, pop() -> 1
```

Example 2:

```
Input: pushed = [1,2,3,4,5], popped = [4,3,5,1,2]
Output: false
Explanation: 1 cannot be popped before 2.
```

Note:

1. 0 <= pushed.length == popped.length <= 1000
2. 0 <= pushed[i], popped[i] < 1000
3. pushed is a permutation of popped.
4. pushed and popped have distinct values.

题目大意

给 2 个数组，一个数组里面代表的是 push 的顺序，另一个数组里面代表的是 pop 的顺序。问按照这样的顺序操作以后，最终能否把栈清空?

解题思路

这一题也是靠栈操作的题目，按照 push 数组的顺序先把压栈，然后再依次在 pop 里面找栈顶元素，找到了就 pop，直到遍历完 pop 数组，最终如果遍历完了 pop 数组，就代表清空了整个栈了。

代码

```go
package leetcode

func validateStackSequences(pushed []int, popped []int) bool {
  stack, j, N := []int{}, 0, len(pushed)
  for _, x := range pushed {
    stack = append(stack, x)
    for len(stack) != 0 && j < N && stack[len(stack)-1] == popped[j] {
      stack = stack[0 : len(stack)-1]
      j++
    }
  }
  return j == N
}
```

947. Most Stones Removed with Same Row or Column

题目

On a 2D plane, we place stones at some integer coordinate points. Each coordinate point may have at most one stone.

Now, a *move* consists of removing a stone that shares a column or row with another stone on the grid.

What is the largest possible number of moves we can make?

Example 1:

```
Input: stones = [[0,0],[0,1],[1,0],[1,2],[2,1],[2,2]]
Output: 5
```

Example 2:

```
Input: stones = [[0,0],[0,2],[1,1],[2,0],[2,2]]
Output: 3
```

Example 3:

```
Input: stones = [[0,0]]
Output: 0
```

Note:

1. `1 <= stones.length <= 1000`
2. `0 <= stones[i][j] < 10000`

题目大意

在二维平面上，我们将石头放置在一些整数坐标点上。每个坐标点上最多只能有一块石头。现在，move 操作将会移除与网格上的某一块石头共享一列或一行的一块石头。我们最多能执行多少次 move 操作?

提示：

- 1 <= stones.length <= 1000
- 0 <= stones[i][j] < 10000

解题思路

- 给出一个数组，数组中的元素是一系列的坐标点。现在可以移除一些坐标点，移除必须满足：移除的这个点，在相同的行或者列上有一个点。问最终可以移除多少个点。移除到最后必然有些点独占一行，那么这些点都不能被移除。
- 这一题的解题思路是并查集。把所有共行或者共列的点都 `union()` 起来。不同集合之间是不能相互移除的。反证法：如果能移除，代表存在共行或者共列的情况，那么肯定是同一个集合了，这样就不满足不同集合了。最终剩下的点就是集合的个数，每个集合只会留下一个点。所以移除的点就是点的总数减去集合的个数 `len(stones) - uf.totalCount()`。
- 如果暴力合并集合，时间复杂度也非常差，可以由优化的地方。再遍历所有点的过程中，可以把遍历过的行和列存起来。这里可以用 map 来记录，key 为行号，value 为上一次遍历过的点的序号。同样，列也可以用 map 存起来，key 为列号，value 为上一次遍历过的点的序号。经过这样的优化以后，时间复杂度会提高不少。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/template"
)

func removeStones(stones [][]int) int {
  if len(stones) <= 1 {
    return 0
  }
  uf, rowMap, colMap := template.UnionFind{}, map[int]int{}, map[int]int{}
  uf.Init(len(stones))
  for i := 0; i < len(stones); i++ {
```

```
    if _, ok := rowMap[stones[i][0]]; ok {
      uf.Union(rowMap[stones[i][0]], i)
    } else {
      rowMap[stones[i][0]] = i
    }
    if _, ok := colMap[stones[i][1]]; ok {
      uf.Union(colMap[stones[i][1]], i)
    } else {
      colMap[stones[i][1]] = i
    }
  }
  return len(stones) - uf.TotalCount()
}
```

949. Largest Time for Given Digits

题目

Given an array of 4 digits, return the largest 24 hour time that can be made.

The smallest 24 hour time is 00:00, and the largest is 23:59. Starting from 00:00, a time is larger if more time has elapsed since midnight.

Return the answer as a string of length 5. If no valid time can be made, return an empty string.

Example 1:

```
Input: [1,2,3,4]
Output: "23:41"
```

Example 2:

```
Input: [5,5,5,5]
Output: ""
```

Note:

1. `A.length == 4`
2. `0 <= A[i] <= 9`

题目大意

给定一个由 4 位数字组成的数组，返回可以设置的符合 24 小时制的最大时间。最小的 24 小时制时间是 00:00，而最大的是 23:59。从 00:00 （午夜）开始算起，过得越久，时间越大。以长度为 5 的字符串返回答案。如果不能确定有效时间，则返回空字符串。

解题思路

- 给出 4 个数字，要求返回一个字符串，代表由这 4 个数字能组成的最大 24 小时制的时间。
- 简单题，这一题直接暴力枚举就可以了。依次检查给出的 4 个数字每个排列组合是否是时间合法的。例如检查 10 * A[i] + A[j] 是不是小于 24， 10 * A[k] + A[l] 是不是小于 60。如果合法且比目前存在的最大时间更大，就更新这个最大时间。

代码

```go
package leetcode

import "fmt"

func largestTimeFromDigits(A []int) string {
  flag, res := false, 0
  for i := 0; i < 4; i++ {
    for j := 0; j < 4; j++ {
      if i == j {
        continue
      }
      for k := 0; k < 4; k++ {
        if i == k || j == k {
          continue
        }
        l := 6 - i - j - k
        hour := A[i]*10 + A[j]
        min := A[k]*10 + A[l]
        if hour < 24 && min < 60 {
          if hour*60+min >= res {
            res = hour*60 + min
            flag = true
          }
        }
      }
    }
  }
  if flag {
    return fmt.Sprintf("%02d:%02d", res/60, res%60)
  } else {
    return ""
  }
}
```

952. Largest Component Size by Common Factor

题目

Given a non-empty array of unique positive integers `A`, consider the following graph:

- There are `A.length` nodes, labelled `A[0]` to `A[A.length - 1];`
- There is an edge between `A[i]` and `A[j]` if and only if `A[i]` and `A[j]` share a common factor greater than 1.

Return the size of the largest connected component in the graph.

Example 1:

```
Input: [4,6,15,35]
Output: 4
```

Example 2:

```
Input: [20,50,9,63]
Output: 2
```

Example 3:

```
Input: [2,3,6,7,4,12,21,39]
Output: 8
```

Note:

1. `1 <= A.length <= 20000`
2. `1 <= A[i] <= 100000`

题目大意

给定一个由不同正整数的组成的非空数组 A，考虑下面的图：

有 A.length 个节点，按从 A[0] 到 A[A.length - 1] 标记；
只有当 A[i] 和 A[j] 共用一个大于 1 的公因数时，A[i] 和 A[j] 之间才有一条边。
返回图中最大连通组件的大小。

提示：

1. 1 <= A.length <= 20000
2. 1 <= A[i] <= 100000

解题思路

- 给出一个数组，数组中的元素如果每两个元素有公约数，那么这两个元素可以算有关系。所有有关系的数可以放在一个集合里，问这个数组里面有关系的元素组成的集合里面最多有多少个元素。
- 这一题读完题直觉就是用并查集来解题。首先可以用暴力的解法尝试。用 2 层循环，两两比较有没有公约

数，如果有公约数就 `union()` 到一起。提交以后出现 TLE，其实看一下数据规模就知道会超时，`1 <= A.length <= 20000`。注意到 `1 <= A[i] <= 100000`，开根号以后最后才 316.66666，这个规模的数不大。所以把每个数小于根号自己的因子都找出来，例如 `6 = 2 * 3`，`15 = 3 * 5`，那么把 6 和 2，6 和 3 都 `union()`，15 和 3，15 和 5 都 `union()`，最终遍历所有的集合，找到最多元素的集合，输出它包含的元素值。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/template"
)

// 解法一 并查集 UnionFind
func largestComponentSize(A []int) int {
  maxElement, uf, countMap, res := 0, template.UnionFind{}, map[int]int{}, 1
  for _, v := range A {
    maxElement = max(maxElement, v)
  }
  uf.Init(maxElement + 1)
  for _, v := range A {
    for k := 2; k*k <= v; k++ {
      if v%k == 0 {
        uf.Union(v, k)
        uf.Union(v, v/k)
      }
    }
  }
  for _, v := range A {
    countMap[uf.Find(v)]++
    res = max(res, countMap[uf.Find(v)])
  }
  return res
}

// 解法二 UnionFindCount
func largestComponentSize1(A []int) int {
  uf, factorMap := template.UnionFindCount{}, map[int]int{}
  uf.Init(len(A))
  for i, v := range A {
    for k := 2; k*k <= v; k++ {
      if v%k == 0 {
        if _, ok := factorMap[k]; !ok {
          factorMap[k] = i
        } else {
          uf.Union(i, factorMap[k])
```

```
        }
        if _, ok := factorMap[v/k]; !ok {
          factorMap[v/k] = i
        } else {
          uf.Union(i, factorMap[v/k])
        }
      }
    }
    if _, ok := factorMap[v]; !ok {
      factorMap[v] = i
    } else {
      uf.Union(i, factorMap[v])
    }
  }
  return uf.MaxUnionCount()
}
```

953. Verifying an Alien Dictionary

题目

In an alien language, surprisingly they also use english lowercase letters, but possibly in a different `order`. The `order` of the alphabet is some permutation of lowercase letters.

Given a sequence of `words` written in the alien language, and the `order` of the alphabet, return `true` if and only if the given `words` are sorted lexicographicaly in this alien language.

Example 1:

```
Input: words = ["hello","leetcode"], order = "hlabcdefgijkmnopqrstuvwxyz"
Output: true
Explanation: As 'h' comes before 'l' in this language, then the sequence is sorted.
```

Example 2:

```
Input: words = ["word","world","row"], order = "worldabcefghijkmnpqstuvxyz"
Output: false
Explanation: As 'd' comes after 'l' in this language, then words[0] > words[1], hence
the sequence is unsorted.
```

Example 3:

```
Input: words = ["apple","app"], order = "abcdefghijklmnopqrstuvwxyz"
Output: false
Explanation: The first three characters "app" match, and the second string is shorter
(in size.) According to lexicographical rules "apple" > "app", because 'l' > '∅', where
'∅' is defined as the blank character which is less than any other character (More
info).
```

Note:

1. `1 <= words.length <= 100`
2. `1 <= words[i].length <= 20`
3. `order.length == 26`
4. All characters in `words[i]` and `order` are english lowercase letters.

题目大意

某种外星语也使用英文小写字母，但可能顺序 order 不同。字母表的顺序（order）是一些小写字母的排列。给定一组用外星语书写的单词 words，以及其字母表的顺序 order，只有当给定的单词在这种外星语中按字典序排列时，返回 true；否则，返回 false。

解题思路

- 这一题是简单题。给出一个字符串数组，判断把字符串数组里面字符串是否是按照 order 的排序排列的。order 是给出个一个字符串排序。这道题的解法是把 26 个字母的顺序先存在 map 中，然后依次遍历判断字符串数组里面字符串的大小。

代码

```go
package leetcode

func isAlienSorted(words []string, order string) bool {
  if len(words) < 2 {
    return true
  }
  hash := make(map[byte]int)
  for i := 0; i < len(order); i++ {
    hash[order[i]] = i
  }
  for i := 0; i < len(words)-1; i++ {
    pointer, word, wordplus := 0, words[i], words[i+1]
    for pointer < len(word) && pointer < len(wordplus) {
      if hash[word[pointer]] > hash[wordplus[pointer]] {
        return false
      }
      if hash[word[pointer]] < hash[wordplus[pointer]] {
```

```
        break
      } else {
        pointer = pointer + 1
      }
    }
    if pointer < len(word) && pointer >= len(wordplus) {
      return false
    }
  }
  return true
}
```

959. Regions Cut By Slashes

题目

In a N x N `grid` composed of 1 x 1 squares, each 1 x 1 square consists of a `/`, `\`, or blank space. These characters divide the square into contiguous regions.

(Note that backslash characters are escaped, so a `\` is represented as `"\\"`.)

Return the number of regions.

Example 1:

```
Input:
[
  " /",
  "/ "
]
Output: 2
Explanation: The 2x2 grid is as follows:
```

Example 2:

```
Input:
[
  " /",
  "  "
]
Output: 1
Explanation: The 2x2 grid is as follows:
```

Example 3:

```
Input:
[
  "\\/",
  "/\\"
]
Output: 4
Explanation: (Recall that because \ characters are escaped, "\\/" refers to \/, and
"/\\" refers to /\.)
The 2x2 grid is as follows:
```

Example 4:

```
Input:
[
  "/\\",
  "\\/"
]
Output: 5
Explanation: (Recall that because \ characters are escaped, "/\\" refers to /\, and
"\\/" refers to \/.)
The 2x2 grid is as follows:
```

Example 5:

```
Input:
[
  "//",
  "/ "
]
Output: 3
Explanation: The 2x2 grid is as follows:
```

Note:

1. `1 <= grid.length == grid[0].length <= 30`
2. `grid[i][j]` is either `'/'`, `'\'`, or `' '`.

题目大意

在由 1 x 1 方格组成的 N x N 网格 grid 中，每个 1 x 1 方块由 /、\ 或空格构成。这些字符会将方块划分为一些共边的区域。(请注意，反斜杠字符是转义的，因此 \ 用 "\" 表示)返回区域的数目。

提示：

- 1 <= grid.length == grid[0].length <= 30
- grid[i][j] 是 '/'、"、或 ' '。

解题思路

- 给出一个字符串，代表的是 `N x N` 正方形中切分的情况，有 2 种切分的情况 `'\'` 和 `'/'`，即从左上往右下切和从右上往左下切。问按照给出的切分方法，能把 `N x N` 正方形切成几部分？
- 这一题解题思路是并查集。先将每个 `1*1` 的正方形切分成下图的样子。分成 4 小块。然后按照题目给的切分图来合并各个小块。

- 遇到 `'\\'`，就把第 0 块和第 1 块 `union()` 起来，第 2 块和第 3 块 `union()` 起来；遇到 `'/'`，就把第 0 块和第 3 块 `union()` 起来，第 2 块和第 1 块 `union()` 起来；遇到 `' '`，就把第 0 块和第 1 块 `union()` 起来，第 2 块和第 1 块 `union()` 起来，第 2 块和第 3 块 `union()` 起来，即 4 块都 `union()` 起来；最后还需要记得上一行和下一行还需要 `union()`，即本行的第 2 块和下一行的第 0 块 `union()` 起来；左边一列和右边一列也需要 `union()`。即本列的第 1 块和右边一列的第 3 块 `union()` 起来。最后计算出集合总个数就是最终答案了。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/template"
)

func regionsBySlashes(grid []string) int {
  size := len(grid)
  uf := template.UnionFind{}
  uf.Init(4 * size * size)
  for i := 0; i < size; i++ {
    for j := 0; j < size; j++ {
      switch grid[i][j] {
      case '\\':
        uf.Union(getFaceIdx(size, i, j, 0), getFaceIdx(size, i, j, 1))
        uf.Union(getFaceIdx(size, i, j, 2), getFaceIdx(size, i, j, 3))
      case '/':
        uf.Union(getFaceIdx(size, i, j, 0), getFaceIdx(size, i, j, 3))
        uf.Union(getFaceIdx(size, i, j, 2), getFaceIdx(size, i, j, 1))
      case ' ':
        uf.Union(getFaceIdx(size, i, j, 0), getFaceIdx(size, i, j, 1))
        uf.Union(getFaceIdx(size, i, j, 2), getFaceIdx(size, i, j, 1))
        uf.Union(getFaceIdx(size, i, j, 2), getFaceIdx(size, i, j, 3))
      }
      if i < size-1 {
        uf.Union(getFaceIdx(size, i, j, 2), getFaceIdx(size, i+1, j, 0))
      }
      if j < size-1 {
        uf.Union(getFaceIdx(size, i, j, 1), getFaceIdx(size, i, j+1, 3))
      }
```

```
    }
  }
  count := 0
  for i := 0; i < 4*size*size; i++ {
    if uf.Find(i) == i {
      count++
    }
  }
  return count
}

func getFaceIdx(size, i, j, k int) int {
  return 4*(i*size+j) + k
}
```

961. N-Repeated Element in Size 2N Array

题目

In a array `A` of size `2N`, there are `N+1` unique elements, and exactly one of these elements is repeated N times.

Return the element repeated `N` times.

Example 1:

```
Input: [1,2,3,3]
Output: 3
```

Example 2:

```
Input: [2,1,2,5,3,2]
Output: 2
```

Example 3:

```
Input: [5,1,5,2,5,3,5,4]
Output: 5
```

Note:

1. `4 <= A.length <= 10000`
2. `0 <= A[i] < 10000`
3. `A.length` is even

题目大意

在大小为 2N 的数组 A 中有 N+1 个不同的元素，其中有一个元素重复了 N 次。返回重复了 N 次的那个元素。

解题思路

- 简单题。数组中有 2N 个数，有 N + 1 个数是不重复的，这之中有一个数重复了 N 次，请找出这个数。解法非常简单，把所有数都存入 map 中，如果遇到存在的 key 就返回这个数。

代码

```
package leetcode

func repeatedNTimes(A []int) int {
  kv := make(map[int]struct{})
  for _, val := range A {
    if _, ok := kv[val]; ok {
      return val
    }
    kv[val] = struct{}{}
  }
  return 0
}
```

966. Vowel Spellchecker

题目

Given a `wordlist`, we want to implement a spellchecker that converts a query word into a correct word.

For a given `query` word, the spell checker handles two categories of spelling mistakes:

- Capitalization: If the query matches a word in the wordlist (**case-insensitive**), then the query word is returned with the same case as the case in the wordlist.
  - Example: `wordlist = ["yellow"]`, `query = "YellOw"`: `correct = "yellow"`
  - Example: `wordlist = ["Yellow"]`, `query = "yellow"`: `correct = "Yellow"`
  - Example: `wordlist = ["yellow"]`, `query = "yellow"`: `correct = "yellow"`
- Vowel Errors: If after replacing the vowels ('a', 'e', 'i', 'o', 'u') of the query word with any vowel individually, it matches a word in the wordlist (**case-insensitive**), then the query word is returned with the same case as the match in the wordlist.
  - Example: `wordlist = ["YellOw"]`, `query = "yollow"`: `correct = "YellOw"`
  - Example: `wordlist = ["YellOw"]`, `query = "yeellow"`: `correct = ""` (no match)
  - Example: `wordlist = ["YellOw"]`, `query = "yllw"`: `correct = ""` (no match)

In addition, the spell checker operates under the following precedence rules:

- When the query exactly matches a word in the wordlist (**case-sensitive**), you should return the same

word back.
- When the query matches a word up to capitlization, you should return the first such match in the wordlist.
- When the query matches a word up to vowel errors, you should return the first such match in the wordlist.
- If the query has no matches in the wordlist, you should return the empty string.

Given some `queries`, return a list of words `answer`, where `answer[i]` is the correct word for `query = queries[i]`.

Example 1:

```
Input:wordlist = ["KiTe","kite","hare","Hare"], queries =
["kite","Kite","KiTe","Hare","HARE","Hear","hear","keti","keet","keto"]
Output:["kite","KiTe","KiTe","Hare","hare","","","KiTe","","KiTe"]
```

Note:

- `1 <= wordlist.length <= 5000`
- `1 <= queries.length <= 5000`
- `1 <= wordlist[i].length <= 7`
- `1 <= queries[i].length <= 7`
- All strings in `wordlist` and `queries` consist only of **english** letters.

题目大意

在给定单词列表 wordlist 的情况下，我们希望实现一个拼写检查器，将查询单词转换为正确的单词。

对于给定的查询单词 query，拼写检查器将会处理两类拼写错误：

- 大小写：如果查询匹配单词列表中的某个单词（不区分大小写），则返回的正确单词与单词列表中的大小写相同。
    - 例如：wordlist = ["yellow"], query = "YellOw": correct = "yellow"
    - 例如：wordlist = ["Yellow"], query = "yellow": correct = "Yellow"
    - 例如：wordlist = ["yellow"], query = "yellow": correct = "yellow"
- 元音错误：如果在将查询单词中的元音（‘a’、‘e’、‘i’、‘o’、‘u’）分别替换为任何元音后，能与单词列表中的单词匹配（不区分大小写），则返回的正确单词与单词列表中的匹配项大小写相同。
    - 例如：wordlist = ["YellOw"], query = "yollow": correct = "YellOw"
    - 例如：wordlist = ["YellOw"], query = "yeellow": correct = "" （无匹配项）
    - 例如：wordlist = ["YellOw"], query = "yllw": correct = "" （无匹配项）

此外，拼写检查器还按照以下优先级规则操作：

- 当查询完全匹配单词列表中的某个单词（区分大小写）时，应返回相同的单词。
- 当查询匹配到大小写问题的单词时，您应该返回单词列表中的第一个这样的匹配项。
- 当查询匹配到元音错误的单词时，您应该返回单词列表中的第一个这样的匹配项。
- 如果该查询在单词列表中没有匹配项，则应返回空字符串。

给出一些查询 queries，返回一个单词列表 answer，其中 answer[i] 是由查询 query = queries[i] 得到的正确单词。

解题思路

- 读完题，很明显需要用 `map` 来解题。依题意分为 3 种情况，查询字符串完全匹配；查询字符串只是大小写不同；查询字符串有元音错误。第一种情况用 `map` `key` 直接匹配即可。第二种情况，利用 `map` 将单词从小写形式转换成原单词正确的大小写形式。第三种情况，利用 `map` 将单词从忽略元音的小写形式换成原单词正确形式。最后注意一下题目最后给的 4 个优先级规则即可。

代码

```
package leetcode

import "strings"

func spellchecker(wordlist []string, queries []string) []string {
  wordsPerfect, wordsCap, wordsVowel := map[string]bool{}, map[string]string{},
map[string]string{}
  for _, word := range wordlist {
    wordsPerfect[word] = true
    wordLow := strings.ToLower(word)
    if _, ok := wordsCap[wordLow]; !ok {
      wordsCap[wordLow] = word
    }
    wordLowVowel := devowel(wordLow)
    if _, ok := wordsVowel[wordLowVowel]; !ok {
      wordsVowel[wordLowVowel] = word
    }
  }
  res, index := make([]string, len(queries)), 0
  for _, query := range queries {
    if _, ok := wordsPerfect[query]; ok {
      res[index] = query
      index++
      continue
    }
    queryL := strings.ToLower(query)
    if v, ok := wordsCap[queryL]; ok {
      res[index] = v
      index++
      continue
    }

    queryLV := devowel(queryL)
    if v, ok := wordsVowel[queryLV]; ok {
      res[index] = v
      index++
```

```
        continue
      }
      res[index] = ""
      index++
    }
    return res

  }

  func devowel(word string) string {
    runes := []rune(word)
    for k, c := range runes {
      if c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u' {
        runes[k] = '*'
      }
    }
    return string(runes)
  }
```

968. Binary Tree Cameras

题目

Given a binary tree, we install cameras on the nodes of the tree.

Each camera at a node can monitor **its parent, itself, and its immediate children**.

Calculate the minimum number of cameras needed to monitor all nodes of the tree.

Example 1:

```
Input: [0,0,null,0,0]
Output: 1
Explanation: One camera is enough to monitor all nodes if placed as shown.
```

Example 2:

```
Input: [0,0,null,0,null,0,null,null,0]
Output: 2
Explanation: At least two cameras are needed to monitor all nodes of the tree. The
above image shows one of the valid configurations of camera placement.
```

Note:

1. The number of nodes in the given tree will be in the range `[1, 1000]`.
2. **Every** node has value 0.

题目大意

给定一个二叉树，我们在树的节点上安装摄像头。节点上的每个摄影头都可以监视其父对象、自身及其直接子对象。计算监控树的所有节点所需的最小摄像头数量。

提示：

1. 给定树的节点数的范围是 [1, 1000]。
2. 每个节点的值都是 0。

解题思路

- 给出一棵树，要求在这个树上面放摄像头，一个摄像头最多可以监视 4 个节点，2 个孩子，本身节点，还有父亲节点。问最少放多少个摄像头可以覆盖树上的所有节点。
- 这一题可以用贪心思想来解题。先将节点分为 3 类，第一类，叶子节点，第二类，包含叶子节点的节点，第三类，其中一个叶子节点上放了摄像头的。按照这个想法，将树的每个节点染色，如下图。

- 所有包含叶子节点的节点，可以放一个摄像头，这个可以覆盖至少 3 个节点，如果还有父节点的话，可以覆盖 4 个节点。所以贪心的策略是从最下层的叶子节点开始往上“染色”，先把最下面的一层 1 染色。标 1 的节点都是要放一个摄像头的。如果叶子节点中包含 1 的节点，那么再将这个节点染成 2 。如下图的黄色节点。黄色节点代表的是不用放摄像头的节点，因为它被叶子节点的摄像头覆盖了。出现了 2 的节点以后，再往上的节点又再次恢复成叶子节点 0 。如此类推，直到推到根节点。

- 最后根节点还需要注意多种情况，根节点可能是叶子节点 0，那么最终答案还需要 + 1，因为需要在根节点上放一个摄像头，不然根节点覆盖不到了。根节点也有可能是 1 或者 2，这两种情况都不需要增加摄像头了，以为都覆盖到了。按照上述的方法，递归即可得到答案。

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
type status int

const (
  isLeaf status = iota
  parentofLeaf
  isMonitoredWithoutCamera
)
```

```go
func minCameraCover(root *TreeNode) int {
  res := 0
  if minCameraCoverDFS(root, &res) == isLeaf {
    res++
  }
  return res
}

func minCameraCoverDFS(root *TreeNode, res *int) status {
  if root == nil {
    return 2
  }
  left, right := minCameraCoverDFS(root.Left, res), minCameraCoverDFS(root.Right, res)
  if left == isLeaf || right == isLeaf {
    *res++
    return parentofLeaf
  } else if left == parentofLeaf || right == parentofLeaf {
    return isMonitoredWithoutCamera
  } else {
    return isLeaf
  }
}
```

969. Pancake Sorting

题目

Given an array A, we can perform a pancake flip: We choose some positive integer k <= A.length, then reverse the order of the first k elements of A. We want to perform zero or more pancake flips (doing them one after another in succession) to sort the array A.

Return the k-values corresponding to a sequence of pancake flips that sort A. Any valid answer that sorts the array within 10 * A.length flips will be judged as correct.

Example 1:

```
Input: [3,2,4,1]
Output: [4,2,4,3]
Explanation:
We perform 4 pancake flips, with k values 4, 2, 4, and 3.
Starting state: A = [3, 2, 4, 1]
After 1st flip (k=4): A = [1, 4, 2, 3]
After 2nd flip (k=2): A = [4, 1, 2, 3]
After 3rd flip (k=4): A = [3, 2, 1, 4]
After 4th flip (k=3): A = [1, 2, 3, 4], which is sorted.
```

Example 2:

```
Input: [1,2,3]
Output: []
Explanation: The input is already sorted, so there is no need to flip anything.
Note that other answers, such as [3, 3], would also be accepted.
```

Note:

- 1 <= A.length <= 100
- A[i] is a permutation of [1, 2, ..., A.length]

题目大意

给定一个数组，要求输出“煎饼排序”的步骤，使得最终数组是从小到大有序的。“煎饼排序”，每次排序都反转前 n 个数，n 小于数组的长度。

解题思路

这道题的思路是，每次找到当前数组中无序段中最大的值，（初始的时候，整个数组相当于都是无序段），将最大值的下标 i 进行“煎饼排序”，前 i 个元素都反转一遍。这样最大值就到了第一个位置了。然后紧接着再进行一次数组总长度 n 的“煎饼排序”，目的是使最大值到数组最后一位，这样它的位置就归位了。那么数组的无序段为 n-1 。然后用这个方法不断的循环，直至数组中每个元素都到了排序后最终的位置下标上了。最终数组就有序了。

这道题有一个特殊点在于，数组里面的元素都是自然整数，那么最终数组排序完成以后，数组的长度就是最大值。所以找最大值也不需要遍历一次数组了，直接取出长度就是最大值。

代码

```go
package leetcode

func pancakeSort(A []int) []int {
  if len(A) == 0 {
```

```go
    return []int{}
  }
  right := len(A)
  var (
    ans []int
  )
  for right > 0 {
    idx := find(A, right)
    if idx != right-1 {
      reverse969(A, 0, idx)
      reverse969(A, 0, right-1)
      ans = append(ans, idx+1, right)
    }
    right--
  }

  return ans
}

func reverse969(nums []int, l, r int) {
  for l < r {
    nums[l], nums[r] = nums[r], nums[l]
    l++
    r--
  }
}

func find(nums []int, t int) int {
  for i, num := range nums {
    if num == t {
      return i
    }
  }
  return -1
}
```

970. Powerful Integers

题目

Given two positive integers `x` and `y`, an integer is *powerful* if it is equal to `x^i + y^j` for some integers `i >= 0` and `j >= 0`.

Return a list of all *powerful* integers that have value less than or equal to `bound`.

You may return the answer in any order. In your answer, each value should occur at most once.

Example 1:

```
Input: x = 2, y = 3, bound = 10
Output: [2,3,4,5,7,9,10]
Explanation:
2 = 2^0 + 3^0
3 = 2^1 + 3^0
4 = 2^0 + 3^1
5 = 2^1 + 3^1
7 = 2^2 + 3^1
9 = 2^3 + 3^0
10 = 2^0 + 3^2
```

Example 2:

```
Input: x = 3, y = 5, bound = 15
Output: [2,4,6,8,10,14]
```

Note:

- `1 <= x <= 100`
- `1 <= y <= 100`
- `0 <= bound <= 10^6`

题目大意

给定两个正整数 x 和 y，如果某一整数等于 x^i + y^j，其中整数 i >= 0 且 j >= 0，那么我们认为该整数是一个强整数。返回值小于或等于 bound 的所有强整数组成的列表。你可以按任何顺序返回答案。在你的回答中，每个值最多出现一次。

解题思路

- 简答题，题目要求找出满足 `x^i + y^j ≤ bound` 条件的所有解。题目要求输出中不能重复，所以用 map 来去重。剩下的就是 `n^2` 暴力循环枚举所有解。

代码

```go
package leetcode

import "math"

func powerfulIntegers(x int, y int, bound int) []int {
  if x == 1 && y == 1 {
    if bound < 2 {
      return []int{}
    }
    return []int{2}
  }
```

```go
  if x > y {
    x, y = y, x
  }
  visit, result := make(map[int]bool), make([]int, 0)
  for i := 0; ; i++ {
    found := false
    for j := 0; pow(x, i)+pow(y, j) <= bound; j++ {
      v := pow(x, i) + pow(y, j)
      if !visit[v] {
        found = true
        visit[v] = true
        result = append(result, v)
      }
    }
    if !found {
      break
    }
  }
  return result
}

func pow(x, i int) int {
  return int(math.Pow(float64(x), float64(i)))
}
```

971. Flip Binary Tree To Match Preorder Traversal

题目

You are given the `root` of a binary tree with `n` nodes, where each node is uniquely assigned a value from `1` to `n`. You are also given a sequence of `n` values `voyage`, which is the **desired** [pre-order traversal](#) of the binary tree.

Any node in the binary tree can be **flipped** by swapping its left and right subtrees. For example, flipping node 1 will have the following effect:

Flip the **smallest** number of nodes so that the **pre-order traversal** of the tree **matches** `voyage`.

Return *a list of the values of all **flipped** nodes. You may return the answer in **any order**. If it is **impossible** to flip the nodes in the tree to make the pre-order traversal match* `voyage`*, return the list* `[-1]`.

Example 1:

```
Input: root = [1,2], voyage = [2,1]
Output: [-1]
Explanation: It is impossible to flip the nodes such that the pre-order traversal
matches voyage.
```

Example 2:

```
Input: root = [1,2,3], voyage = [1,3,2]
Output: [1]
Explanation: Flipping node 1 swaps nodes 2 and 3, so the pre-order traversal matches
voyage.
```

Example 3:

```
Input: root = [1,2,3], voyage = [1,2,3]
Output: []
Explanation: The tree's pre-order traversal already matches voyage, so no nodes need to
be flipped.
```

Constraints:

- The number of nodes in the tree is `n`.
- `n == voyage.length`
- `1 <= n <= 100`
- `1 <= Node.val, voyage[i] <= n`
- All the values in the tree are **unique**.
- All the values in `voyage` are **unique**.

题目大意

给你一棵二叉树的根节点 root ，树中有 n 个节点，每个节点都有一个不同于其他节点且处于 1 到 n 之间的值。另给你一个由 n 个值组成的行程序列 voyage ，表示 预期 的二叉树 先序遍历 结果。通过交换节点的左右子树，可以翻转 该二叉树中的任意节点。请翻转 最少 的树中节点，使二叉树的 先序遍历 与预期的遍历行程 voyage 相匹配。如果可以，则返回 翻转的 所有节点的值的列表。你可以按任何顺序返回答案。如果不能，则返回列表 [-1]。

解题思路

- 题目要求翻转最少树中节点，利用贪心的思想，应该从根节点开始从上往下依次翻转，这样翻转的次数是最少的。对树进行深度优先遍历，如果遍历到某一个节点的时候，节点值不能与行程序列匹配，那么答案一定是 [-1]。否则，当下一个期望数字 `voyage[i]` 与即将遍历的子节点的值不同的时候，就要翻转一下当前这个节点的左右子树，继续 DFS。递归结束可能有 2 种情况，一种是找出了所有要翻转的节点，另一种情况是没有需要翻转的，即原树先序遍历的结果与 `voyage` 是完全一致的。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/structures"
)

// TreeNode define
type TreeNode = structures.TreeNode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

func flipMatchVoyage(root *TreeNode, voyage []int) []int {
  res, index := make([]int, 0, len(voyage)), 0
  if travelTree(root, &index, voyage, &res) {
    return res
  }
  return []int{-1}
}

func travelTree(root *TreeNode, index *int, voyage []int, res *[]int) bool {
  if root == nil {
    return true
  }
  if root.Val != voyage[*index] {
    return false
  }
  *index++
  if root.Left != nil && root.Left.Val != voyage[*index] {
    *res = append(*res, root.Val)
    return travelTree(root.Right, index, voyage, res) && travelTree(root.Left, index,
voyage, res)
  }
  return travelTree(root.Left, index, voyage, res) && travelTree(root.Right, index,
voyage, res)
}
```

973. K Closest Points to Origin

题目

We have a list of points on the plane. Find the K closest points to the origin (0, 0).

(Here, the distance between two points on a plane is the Euclidean distance.)

You may return the answer in any order. The answer is guaranteed to be unique (except for the order that it is in.)

Example 1:

```
Input: points = [[1,3],[-2,2]], K = 1
Output: [[-2,2]]
Explanation:
The distance between (1, 3) and the origin is sqrt(10).
The distance between (-2, 2) and the origin is sqrt(8).
Since sqrt(8) < sqrt(10), (-2, 2) is closer to the origin.
We only want the closest K = 1 points from the origin, so the answer is just [[-2,2]].
```

Example 2:

```
Input: points = [[3,3],[5,-1],[-2,4]], K = 2
Output: [[3,3],[-2,4]]
(The answer [[-2,4],[3,3]] would also be accepted.)
```

Note:

- 1 <= K <= points.length <= 10000
- -10000 < points[i][0] < 10000
- -10000 < points[i][1] < 10000

题目大意

找出 K 个距离坐标原点最近的坐标点。

解题思路

这题也是排序题，先将所有点距离坐标原点的距离都算出来，然后从小到大排序。取前 K 个即可。

代码

```go
package leetcode

import "sort"

// KClosest define
func KClosest(points [][]int, K int) [][]int {
  sort.Slice(points, func(i, j int) bool {
```

```
    return points[i][0]*points[i][0]+points[i][1]*points[i][1] <
      points[j][0]*points[j][0]+points[j][1]*points[j][1]
  })
  ans := make([][]int, K)
  for i := 0; i < K; i++ {
    ans[i] = points[i]
  }
  return ans
}
```

976. Largest Perimeter Triangle

题目

Given an array A of positive lengths, return the largest perimeter of a triangle with non-zero area, formed from 3 of these lengths.

If it is impossible to form any triangle of non-zero area, return 0.

Example 1:

```
Input: [2,1,2]
Output: 5
```

Example 2:

```
Input: [1,2,1]
Output: 0
```

Example 3:

```
Input: [3,2,3,4]
Output: 10
```

Example 4:

```
Input: [3,6,2,3]
Output: 8
```

Note:

- 3 <= A.length <= 10000
- 1 <= A[i] <= 10^6

题目大意

找到可以组成三角形三条边的长度，要求输出三条边之和最长的，即三角形周长最长。

解题思路

这道题也是排序题，先讲所有的长度进行排序，从大边开始往前找，找到第一个任意两边之和大于第三边(满足能构成三角形的条件)的下标，然后输出这 3 条边之和即可，如果没有找到输出 0 。

代码

```
package leetcode

func largestPerimeter(A []int) int {
  if len(A) < 3 {
    return 0
  }
  quickSort164(A, 0, len(A)-1)
  for i := len(A) - 1; i >= 2; i-- {
    if (A[i]+A[i-1] > A[i-2]) && (A[i]+A[i-2] > A[i-1]) && (A[i-2]+A[i-1] > A[i]) {
      return A[i] + A[i-1] + A[i-2]
    }
  }
  return 0
}
```

977. Squares of a Sorted Array

题目

Given an array of integers A sorted in non-decreasing order, return an array of the squares of each number, also in sorted non-decreasing order.

Example 1:

```
Input: [-4,-1,0,3,10]
Output: [0,1,9,16,100]
```

Example 2:

```
Input: [-7,-3,2,3,11]
Output: [4,9,9,49,121]
```

Note:

1. 1 <= A.length <= 10000
2. -10000 <= A[i] <= 10000
3. A is sorted in non-decreasing order.

题目大意

给一个已经有序的数组，返回的数组也必须是有序的，且数组中的每个元素是由原数组中每个数字的平方得到的。

解题思路

这一题由于原数组是有序的，所以要尽量利用这一特点来减少时间复杂度。

最终返回的数组，最后一位，是最大值，这个值应该是由原数组最大值，或者最小值得来的，所以可以从数组的最后一位开始排列最终数组。用 2 个指针分别指向原数组的首尾，分别计算平方值，然后比较两者大小，大的放在最终数组的后面。然后大的一个指针移动。直至两个指针相撞，最终数组就排列完成了。

代码

```go
package leetcode

import "sort"

// 解法一
func sortedSquares(A []int) []int {
  ans := make([]int, len(A))
  for i, k, j := 0, len(A)-1, len(ans)-1; i <= j; k-- {
    if A[i]*A[i] > A[j]*A[j] {
      ans[k] = A[i] * A[i]
      i++
    } else {
      ans[k] = A[j] * A[j]
```

```
        j--
      }
    }
    return ans
}

// 解法二
func sortedSquares1(A []int) []int {
    for i, value := range A {
      A[i] = value * value
    }
    sort.Ints(A)
    return A
}
```

978. Longest Turbulent Subarray

题目

A subarray `A[i], A[i+1], ..., A[j]` of `A` is said to be *turbulent* if and only if:

- For `i <= k < j`, `A[k] > A[k+1]` when `k` is odd, and `A[k] < A[k+1]` when `k` is even;
- **OR**, for `i <= k < j`, `A[k] > A[k+1]` when `k` is even, and `A[k] < A[k+1]` when `k` is odd.

That is, the subarray is turbulent if the comparison sign flips between each adjacent pair of elements in the subarray.

Return the **length** of a maximum size turbulent subarray of A.

Example 1:

```
Input: [9,4,2,10,7,8,8,1,9]
Output: 5
Explanation: (A[1] > A[2] < A[3] > A[4] < A[5])
```

Example 2:

```
Input: [4,8,12,16]
Output: 2
```

Example 3:

```
Input: [100]
Output: 1
```

Note:

1. `1 <= A.length <= 40000`
2. `0 <= A[i] <= 10^9`

题目大意

当 A 的子数组 A[i], A[i+1], ..., A[j] 满足下列条件时，我们称其为湍流子数组：

若 i <= k < j，当 k 为奇数时， A[k] > A[k+1]，且当 k 为偶数时，A[k] < A[k+1]；
或 若 i <= k < j，当 k 为偶数时，A[k] > A[k+1] ，且当 k 为奇数时， A[k] < A[k+1]。
也就是说，如果比较符号在子数组中的每个相邻元素对之间翻转，则该子数组是湍流子数组。

返回 A 的最大湍流子数组的长度。

提示：

- 1 <= A.length <= 40000
- 0 <= A[i] <= 10^9

解题思路

- 给出一个数组，要求找出“摆动数组”的最大长度。所谓“摆动数组”的意思是，元素一大一小间隔的。
- 这一题可以用滑动窗口来解答。用一个变量记住下次出现的元素需要大于还是需要小于前一个元素。也可以用模拟的方法，用两个变量分别记录上升和下降数字的长度。一旦元素相等了，上升和下降数字长度都置为 1，其他时候按照上升和下降的关系增加队列长度即可，最后输出动态维护的最长长度。

代码

```go
package leetcode

// 解法一 模拟法
func maxTurbulenceSize(A []int) int {
  inc, dec := 1, 1
  maxLen := min(1, len(A))
  for i := 1; i < len(A); i++ {
    if A[i-1] < A[i] {
      inc = dec + 1
      dec = 1
    } else if A[i-1] > A[i] {
      dec = inc + 1
      inc = 1
    } else {
      inc = 1
      dec = 1
    }
    maxLen = max(maxLen, max(inc, dec))
  }
  return maxLen
```

```
}

// 解法二 滑动窗口
func maxTurbulenceSize1(A []int) int {
  if len(A) == 1 {
    return 1
  }
  // flag > 0 代表下一个数要大于前一个数，flag < 0 代表下一个数要小于前一个数
  res, left, right, flag, lastNum := 0, 0, 0, A[1]-A[0], A[0]
  for left < len(A) {
    if right < len(A)-1 && ((A[right+1] > lastNum && flag > 0) || (A[right+1] < lastNum
&& flag < 0) || (right == left)) {
      right++
      flag = lastNum - A[right]
      lastNum = A[right]
    } else {
      if flag != 0 {
        res = max(res, right-left+1)
      }
      left++
    }
  }
  return max(res, 1)
}
```

979. Distribute Coins in Binary Tree

题目

Given the `root` of a binary tree with `N` nodes, each `node` in the tree has `node.val` coins, and there are `N` coins total.

In one move, we may choose two adjacent nodes and move one coin from one node to another. (The move may be from parent to child, or from child to parent.)

Return the number of moves required to make every node have exactly one coin.

Example 1:

```
Input: [3,0,0]
Output: 2
Explanation: From the root of the tree, we move one coin to its left child, and one
coin to its right child.
```

Example 2:

```
Input: [0,3,0]
Output: 3
Explanation: From the left child of the root, we move two coins to the root [taking two moves].  Then, we move one coin from the root of the tree to the right child.
```

Example 3:

```
Input: [1,0,2]
Output: 2
```

Example 4:

```
Input: [1,0,0,null,3]
Output: 4
```

Note:

1. `1<= N <= 100`
2. `0 <= node.val <= N`

题目大意

给定一个有 N 个结点的二叉树的根结点 root，树中的每个结点上都对应有 node.val 枚硬币，并且总共有 N 枚硬币。在一次移动中，我们可以选择两个相邻的结点，然后将一枚硬币从其中一个结点移动到另一个结点。(移动可以是从父结点到子结点，或者从子结点移动到父结点。)。返回使每个结点上只有一枚硬币所需的移动次数。

提示：

1. 1<= N <= 100
2. 0 <= node.val <= N

解题思路

- 给出一棵树，有 N 个节点。有 N 个硬币分散在这 N 个节点中，问经过多少次移动以后，所有节点都有一枚硬币。
- 这一题乍一看比较难分析，仔细一想，可以用贪心和分治的思想来解决。一个树的最小单元是一个根节点和两个孩子。在这种情况下，3 个节点谁的硬币多就可以分给没有硬币的那个节点，这种移动方法也能保证移动步数最少。不难证明，硬币由相邻的节点移动过来的步数是最少的。那么一棵树从最下一层开始往上推，逐步从下往上把硬币移动上去，直到最后根节点也都拥有硬币。多余 1 枚的节点记为 `n -1`，没有硬币的节点记为 -1 。例如，下图中左下角的 3 个节点，有 4 枚硬币的节点可以送出 3 枚硬币，叶子节点有 0 枚硬币的节点需要接收 1 枚硬币。根节点有 0 枚硬币，左孩子给了 3 枚，右孩子需要 1 枚，自己本身还要留一枚，所以最终还能剩 1 枚。

- 所以每个节点移动的步数应该是 `left + right + root.Val - 1`。最后递归求解即可。

代码

```go
package leetcode

/**
 * Definition for a binary tree root.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func distributeCoins(root *TreeNode) int {
  res := 0
  distributeCoinsDFS(root, &res)
  return res
}

func distributeCoinsDFS(root *TreeNode, res *int) int {
  if root == nil {
    return 0
  }
  left, right := distributeCoinsDFS(root.Left, res), distributeCoinsDFS(root.Right,
res)
  *res += abs(left) + abs(right)
  return left + right + root.Val - 1
}
```

980. Unique Paths III

题目

On a 2-dimensional `grid`, there are 4 types of squares:

- `1` represents the starting square. There is exactly one starting square.
- `2` represents the ending square. There is exactly one ending square.
- `0` represents empty squares we can walk over.
- `-1` represents obstacles that we cannot walk over.

Return the number of 4-directional walks from the starting square to the ending square, that **walk over every non-obstacle square exactly once**.

Example 1:

```
Input: [[1,0,0,0],[0,0,0,0],[0,0,2,-1]]
Output: 2
Explanation: We have the following two paths:
1. (0,0),(0,1),(0,2),(0,3),(1,3),(1,2),(1,1),(1,0),(2,0),(2,1),(2,2)
2. (0,0),(1,0),(2,0),(2,1),(1,1),(0,1),(0,2),(0,3),(1,3),(1,2),(2,2)
```

Example 2:

```
Input: [[1,0,0,0],[0,0,0,0],[0,0,0,2]]
Output: 4
Explanation: We have the following four paths:
1. (0,0),(0,1),(0,2),(0,3),(1,3),(1,2),(1,1),(1,0),(2,0),(2,1),(2,2),(2,3)
2. (0,0),(0,1),(1,1),(1,0),(2,0),(2,1),(2,2),(1,2),(0,2),(0,3),(1,3),(2,3)
3. (0,0),(1,0),(2,0),(2,1),(2,2),(1,2),(1,1),(0,1),(0,2),(0,3),(1,3),(2,3)
4. (0,0),(1,0),(2,0),(2,1),(1,1),(0,1),(0,2),(0,3),(1,3),(1,2),(2,2),(2,3)
```

Example 3:

```
Input: [[0,1],[2,0]]
Output: 0
Explanation:
There is no path that walks over every empty square exactly once.
Note that the starting and ending square can be anywhere in the grid.
```

Note:

1. `1 <= grid.length * grid[0].length <= 20`

题目大意

在二维网格 grid 上，有 4 种类型的方格：

- 1 表示起始方格。且只有一个起始方格。
- 2 表示结束方格，且只有一个结束方格。
- 0 表示我们可以走过的空方格。
- -1 表示我们无法跨越的障碍。

返回在四个方向（上、下、左、右）上行走时，从起始方格到结束方格的不同路径的数目，**每一个无障碍方格都要通过一次**。

解题思路

- 这一题也可以按照第 79 题的思路来做。题目要求输出地图中从起点到终点的路径条数。注意路径要求必须走满所有空白的格子。
- 唯一需要注意的一点是，空白的格子并不是最后走的总步数，`总步数 = 空白格子数 + 1`，因为要走到终点，走到终点也算一步。

代码

```
package leetcode

var dir = [][]int{
  {-1, 0},
  {0, 1},
  {1, 0},
  {0, -1},
}

func uniquePathsIII(grid [][]int) int {
  visited := make([][]bool, len(grid))
  for i := 0; i < len(visited); i++ {
    visited[i] = make([]bool, len(grid[0]))
  }
  res, empty, startx, starty, endx, endy, path := 0, 0, 0, 0, 0, 0, []int{}
  for i, v := range grid {
    for j, vv := range v {
      switch vv {
      case 0:
        empty++
      case 1:
        startx, starty = i, j
      case 2:
        endx, endy = i, j
      }
    }
  }
  findUniquePathIII(grid, visited, path, empty+1, startx, starty, endx, endy, &res) //
可走的步数要加一，因为终点格子也算一步，不然永远走不到终点！
  return res
}

func isInPath(board [][]int, x, y int) bool {
  return x >= 0 && x < len(board) && y >= 0 && y < len(board[0])
}

func findUniquePathIII(board [][]int, visited [][]bool, path []int, empty, startx,
starty, endx, endy int, res *int) {
  if startx == endx && starty == endy {
    if empty == 0 {
      *res++
    }
    return
  }
  if board[startx][starty] >= 0 {
    visited[startx][starty] = true
```

```
    empty--
    path = append(path, startx)
    path = append(path, starty)
    for i := 0; i < 4; i++ {
      nx := startx + dir[i][0]
      ny := starty + dir[i][1]
      if isInPath(board, nx, ny) && !visited[nx][ny] {
        findUniquePathIII(board, visited, path, empty, nx, ny, endx, endy, res)
      }
    }
    visited[startx][starty] = false
    //empty++ 这里虽然可以还原这个变量值，但是赋值没有意义，干脆不写了
    path = path[:len(path)-2]
  }
  return
}
```

981. Time Based Key-Value Store

题目

Create a timebased key-value store class `TimeMap`, that supports two operations.

1. `set(string key, string value, int timestamp)`

- Stores the `key` and `value`, along with the given `timestamp`.

2. `get(string key, int timestamp)`

- Returns a value such that `set(key, value, timestamp_prev)` was called previously, with `timestamp_prev <= timestamp`.
- If there are multiple such values, it returns the one with the largest `timestamp_prev`.
- If there are no values, it returns the empty string (`""`).

Example 1:

```
Input: inputs = ["TimeMap","set","get","get","set","get","get"], inputs = [[],
["foo","bar",1],["foo",1],["foo",3],["foo","bar2",4],["foo",4],["foo",5]]
Output: [null,null,"bar","bar",null,"bar2","bar2"]
Explanation:
TimeMap kv;
kv.set("foo", "bar", 1); // store the key "foo" and value "bar" along with timestamp =
1
kv.get("foo", 1);  // output "bar"
kv.get("foo", 3); // output "bar" since there is no value corresponding to foo at
timestamp 3 and timestamp 2, then the only value is at timestamp 1 ie "bar"
kv.set("foo", "bar2", 4);
kv.get("foo", 4); // output "bar2"
kv.get("foo", 5); //output "bar2"
```

Example 2:

```
Input: inputs = ["TimeMap","set","set","get","get","get","get","get"], inputs = [[],
["love","high",10],["love","low",20],["love",5],["love",10],["love",15],["love",20],
["love",25]]
Output: [null,null,null,"","high","high","low","low"]
```

Note:

1. All key/value strings are lowercase.
2. All key/value strings have length in the range `[1, 100]`
3. The `timestamps` for all `TimeMap.set` operations are strictly increasing.
4. `1 <= timestamp <= 10^7`
5. `TimeMap.set` and `TimeMap.get` functions will be called a total of `120000` times (combined) per test case.

题目大意

创建一个基于时间的键值存储类 TimeMap，它支持下面两个操作：

1. set(string key, string value, int timestamp)

- 存储键 key、值 value，以及给定的时间戳 timestamp。

2. get(string key, int timestamp)

- 返回先前调用 set(key, value, timestamp_prev) 所存储的值，其中 timestamp_prev <= timestamp。
- 如果有多个这样的值，则返回对应最大的 timestamp_prev 的那个值。
- 如果没有值，则返回空字符串（""）。

提示：

1. 所有的键/值字符串都是小写的。
2. 所有的键/值字符串长度都在 [1, 100] 范围内。
3. 所有 TimeMap.set 操作中的时间戳 timestamps 都是严格递增的。
4. 1 <= timestamp <= 10^7

5. TimeMap.set 和 TimeMap.get 函数在每个测试用例中将（组合）调用总计 120000 次。

解题思路

- 要求设计一个基于时间戳的 `kv` 存储。`set()` 操作里面会会包含一个时间戳。get() 操作的时候查找时间戳小于等于 `timestamp` 的 `key` 对应的 `value`，如果有多个解，输出满足条件的最大时间戳对应的 `value` 值。
- 这一题可以用二分搜索来解答，用 `map` 存储 `kv` 数据，`key` 对应的就是 `key`，`value` 对应一个结构体，里面包含 `value` 和 `timestamp`。执行 `get()` 操作的时候，先取出 `key` 对应的结构体数组，然后在这个数组里面根据 `timestamp` 进行二分搜索。由于题意是要找小于等于 `timestamp` 的最大 `timestamp`，这会有很多满足条件的解，变换一下，先找 `> timestamp` 的最小解，然后下标减一即是满足题意的最大解。
- 另外题目中提到“`TimeMap.set` 操作中的 `timestamp` 是严格递增的”。所以在 `map` 中存储 `value` 结构体的时候，不需要排序了，天然有序。

代码

```go
package leetcode

import "sort"

type data struct {
  time  int
  value string
}

// TimeMap is a timebased key-value store
// TimeMap define
type TimeMap map[string][]data

// Constructor981 define
func Constructor981() TimeMap {
  return make(map[string][]data, 1024)
}

// Set define
func (t TimeMap) Set(key string, value string, timestamp int) {
  if _, ok := t[key]; !ok {
    t[key] = make([]data, 1, 1024)
  }
  t[key] = append(t[key], data{
    time:  timestamp,
    value: value,
  })
}

// Get define
func (t TimeMap) Get(key string, timestamp int) string {
  d := t[key]
```

```
  i := sort.Search(len(d), func(i int) bool {
    return timestamp < d[i].time
  })
  i--
  return t[key][i].value
}

/**
 * Your TimeMap object will be instantiated and called as such:
 * obj := Constructor();
 * obj.Set(key,value,timestamp);
 * param_2 := obj.Get(key,timestamp);
 */
```

984. String Without AAA or BBB

题目

Given two integers `A` and `B`, return **any** string `S` such that:

- `S` has length `A + B` and contains exactly `A` `'a'` letters, and exactly `B` `'b'` letters;
- The substring `'aaa'` does not occur in `S`;
- The substring `'bbb'` does not occur in `S`.

Example 1:

```
Input: A = 1, B = 2
Output: "abb"
Explanation: "abb", "bab" and "bba" are all correct answers.
```

Example 2:

```
Input: A = 4, B = 1
Output: "aabaa"
```

Note:

1. `0 <= A <= 100`
2. `0 <= B <= 100`
3. It is guaranteed such an `S` exists for the given `A` and `B`.

题目大意

给定两个整数 A 和 B，返回任意字符串 S，要求满足：

- S 的长度为 A + B，且正好包含 A 个 'a' 字母与 B 个 'b' 字母；
- 子串 'aaa' 没有出现在 S 中；

- 子串 'bbb' 没有出现在 S 中。

提示：

- 0 <= A <= 100
- 0 <= B <= 100
- 对于给定的 A 和 B，保证存在满足要求的 S。

解题思路

- 给出 A 和 B 的个数，要求组合出字符串，不能出现 3 个连续的 A 和 3 个连续的 B。这题由于只可能有 4 种情况，暴力枚举就可以了。假设 B 的个数比 A 多(如果 A 多，就交换一下 A 和 B)，最终可能的情况只可能是这 4 种情况中的一种：`ba`，`bbabb`，`bbabbabb`，`bbabbabbabbabababa`。

代码

```
package leetcode

func strWithout3a3b(A int, B int) string {
  ans, a, b := "", "a", "b"
  if B < A {
    A, B = B, A
    a, b = b, a
  }
  Dif := B - A
  if A == 1 && B == 1 { // ba
    ans = b + a
  } else if A == 1 && B < 5 { // bbabb
    for i := 0; i < B-2; i++ {
      ans = ans + b
    }
    ans = b + b + a + ans
  } else if (B-A)/A >= 1 { //bbabbabb
    for i := 0; i < A; i++ {
      ans = ans + b + b + a
      B = B - 2
    }
    for i := 0; i < B; i++ {
      ans = ans + b
    }
  } else { //bbabbabbabbabababa
    for i := 0; i < Dif; i++ {
      ans = ans + b + b + a
      B -= 2
      A--
    }
    for i := 0; i < B; i++ {
      ans = ans + b + a
```

```
    }
  }
  return ans
}
```

985. Sum of Even Numbers After Queries

题目

We have an array `A` of integers, and an array `queries` of queries.

For the `i`-th query `val = queries[i][0], index = queries[i][1]`, we add val to `A[index]`. Then, the answer to the `i`-th query is the sum of the even values of `A`.

(Here, the given `index = queries[i][1]` is a 0-based index, and each query permanently modifies the array `A`.)

Return the answer to all queries. Your `answer` array should have `answer[i]` as the answer to the `i`-th query.

Example 1:

```
Input: A = [1,2,3,4], queries = [[1,0],[-3,1],[-4,0],[2,3]]
Output: [8,6,2,4]
Explanation:
At the beginning, the array is [1,2,3,4].
After adding 1 to A[0], the array is [2,2,3,4], and the sum of even values is 2 + 2 + 4
= 8.
After adding -3 to A[1], the array is [2,-1,3,4], and the sum of even values is 2 + 4 =
6.
After adding -4 to A[0], the array is [-2,-1,3,4], and the sum of even values is -2 + 4
= 2.
After adding 2 to A[3], the array is [-2,-1,3,6], and the sum of even values is -2 + 6
= 4.
```

Note:

1. `1 <= A.length <= 10000`
2. `-10000 <= A[i] <= 10000`
3. `1 <= queries.length <= 10000`
4. `-10000 <= queries[i][0] <= 10000`
5. `0 <= queries[i][1] < A.length`

题目大意

给出一个整数数组 A 和一个查询数组 queries。

对于第 i 次查询，有 val = queries[i][0], index = queries[i][1]，我们会把 val 加到 A[index] 上。然后，第 i 次查询的答案是 A 中偶数值的和。（此处给定的 index = queries[i][1] 是从 0 开始的索引，每次查询都会永久修改数组 A。）返回所有查询的答案。你的答案应当以数组 answer 给出，answer[i] 为第 i 次查询的答案。

解题思路

- 给出一个数组 A 和 query 数组。要求每次 query 操作都改变数组 A 中的元素值，并计算此次操作结束数组 A 中偶数值之和。
- 简单题，先计算 A 中所有偶数之和。再每次 query 操作的时候，动态维护这个偶数之和即可。

代码

```go
package leetcode

func sumEvenAfterQueries(A []int, queries [][]int) []int {
  cur, res := 0, []int{}
  for _, v := range A {
    if v%2 == 0 {
      cur += v
    }
  }
  for _, q := range queries {
    if A[q[1]]%2 == 0 {
      cur -= A[q[1]]
    }
    A[q[1]] += q[0]
    if A[q[1]]%2 == 0 {
      cur += A[q[1]]
    }
    res = append(res, cur)
  }
  return res
}
```

986. Interval List Intersections

题目

Given two lists of closed intervals, each list of intervals is pairwise disjoint and in sorted order.

Return the intersection of these two interval lists.

(Formally, a closed interval [a, b] (with a <= b) denotes the set of real numbers x with a <= x <= b. The intersection of two closed intervals is a set of real numbers that is either empty, or can be represented as a closed interval. For example, the intersection of [1, 3] and [2, 4] is [2, 3].)

Example 1:

```
Input: A = [[0,2],[5,10],[13,23],[24,25]], B = [[1,5],[8,12],[15,24],[25,26]]
Output: [[1,2],[5,5],[8,10],[15,23],[24,24],[25,25]]
Reminder: The inputs and the desired output are lists of Interval objects, and not
arrays or lists.
```

Note:

- 0 <= A.length < 1000
- 0 <= B.length < 1000
- 0 <= A[i].start, A[i].end, B[i].start, B[i].end < 10^9

Note: input types have been changed on April 15, 2019. Please reset to default code definition to get new method signature.

题目大意

这道题考察的是滑动窗口的问题。

给出 2 个数组 A 和数组 B。要求求出这 2 个数组的交集数组。题意见图。

解题思路

交集的左边界应该为，`start := max(A[i].Start, B[j].Start)`，右边界为，`end := min(A[i].End, B[j].End)`，如果 `start <= end`，那么这个就是一个满足条件的交集，放入最终数组中。如果 `A[i].End <= B[j].End`，代表 B 数组范围比 A 数组大，A 的游标右移。如果 `A[i].End > B[j].End`，代表 A 数组范围比 B 数组大，B 的游标右移。

代码

```go
package leetcode

/**
 * Definition for an interval.
 * type Interval struct {
 *     Start int
 *     End   int
 * }
 */
func intervalIntersection(A []Interval, B []Interval) []Interval {
  res := []Interval{}
  for i, j := 0, 0; i < len(A) && j < len(B); {
    start := max(A[i].Start, B[j].Start)
    end := min(A[i].End, B[j].End)
    if start <= end {
      res = append(res, Interval{Start: start, End: end})
```

```
    }
    if A[i].End <= B[j].End {
      i++
    } else {
      j++
    }
  }
  return res
}
```

987. Vertical Order Traversal of a Binary Tree

题目

Given the `root` of a binary tree, calculate the **vertical order traversal** of the binary tree.

For each node at position `(row, col)`, its left and right children will be at positions `(row + 1, col - 1)` and `(row + 1, col + 1)` respectively. The root of the tree is at `(0, 0)`.

The **vertical order traversal** of a binary tree is a list of top-to-bottom orderings for each column index starting from the leftmost column and ending on the rightmost column. There may be multiple nodes in the same row and same column. In such a case, sort these nodes by their values.

Return *the **vertical order traversal** of the binary tree.*

Example 1:

```
Input: root = [3,9,20,null,null,15,7]
Output: [[9],[3,15],[20],[7]]
Explanation:
Column -1: Only node 9 is in this column.
Column 0: Nodes 3 and 15 are in this column in that order from top to bottom.
Column 1: Only node 20 is in this column.
Column 2: Only node 7 is in this column.
```

Example 2:

```
Input: root = [1,2,3,4,5,6,7]
Output: [[4],[2],[1,5,6],[3],[7]]
Explanation:
Column -2: Only node 4 is in this column.
Column -1: Only node 2 is in this column.
Column 0: Nodes 1, 5, and 6 are in this column.
          1 is at the top, so it comes first.
          5 and 6 are at the same position (2, 0), so we order them by their value, 5
before 6.
Column 1: Only node 3 is in this column.
Column 2: Only node 7 is in this column.
```

Example 3:

```
Input: root = [1,2,3,4,6,5,7]
Output: [[4],[2],[1,5,6],[3],[7]]
Explanation:
This case is the exact same as example 2, but with nodes 5 and 6 swapped.
Note that the solution remains the same since 5 and 6 are in the same location and
should be ordered by their values.
```

Constraints:

- The number of nodes in the tree is in the range `[1, 1000]`.
- `0 <= Node.val <= 1000`

题目大意

给你二叉树的根结点 root ，请你设计算法计算二叉树的 垂序遍历 序列。

对位于 (row, col) 的每个结点而言，其左右子结点分别位于 (row + 1, col - 1) 和 (row + 1, col + 1) 。树的根结点位于 (0, 0) 。二叉树的 垂序遍历 从最左边的列开始直到最右边的列结束，按列索引每一列上的所有结点，形成一个按出现位置从上到下排序的有序列表。如果同行同列上有多个结点，则按结点的值从小到大进行排序。返回二叉树的 垂序遍历 序列。

解题思路

- 题目要求按照一列一列的遍历二叉树。需要解决 2 个问题。第一个问题，二叉树上每个结点的二维坐标如何计算。第二个问题，同一个二维坐标点上摞起来多个结点，需要按照从小到大的顺序排序，如例子二和例子三，同一个二维坐标点 (2，0) 上，摞了 2 个不同的结点。
- 先解决第一个问题，由于题目要求根结点是 (0，0) ，即根结点是坐标原点，它的左子树的 x 坐标都是负数，它的右子树的 x 坐标都是正数。按照先序遍历，就可以将这些结点的二维坐标计算出来。再进行一次排序，按照 x 坐标从小到大排序，坐标相同的情况对应着结点摞起来的情况，摞起来的结点按照 val 值的大小从小到大排序。这样在 x 轴方向，所有结点就排列好了。排序完成，也顺便解决了第二个问题。
- 最后一步只需要扫描一遍这个排好序的数组，按照列的顺序，依次将同一列的结点打包至一个一维数组中。最终输出的二维数组即为题目所求。

代码

```go
package leetcode

import (
  "sort"

  "github.com/halfrost/LeetCode-Go/structures"
)

// TreeNode define
type TreeNode = structures.TreeNode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

type node struct {
  x, y, val int
}

func verticalTraversal(root *TreeNode) [][]int {
  nodes := []*node{}
  inorder(root, 0, 0, &nodes)
  sort.Slice(nodes, func(i, j int) bool {
    if nodes[i].y == nodes[j].y {
      if nodes[i].x < nodes[j].x {
        return true
      } else if nodes[i].x > nodes[j].x {
        return false
      }
      return nodes[i].val < nodes[j].val
    }
    return nodes[i].y < nodes[j].y
  })
  res, currY, currColumn := [][]int{}, nodes[0].y, []int{nodes[0].val}
  for i := 1; i < len(nodes); i++ {
    if currY == nodes[i].y {
      currColumn = append(currColumn, nodes[i].val)
    } else {
      res = append(res, currColumn)
      currColumn = []int{nodes[i].val}
      currY = nodes[i].y
```

```
    }
  }
  res = append(res, currColumn)
  return res
}

func inorder(root *TreeNode, x, y int, nodes *[]*node) {
  if root != nil {
    *nodes = append(*nodes, &node{x, y, root.Val})
    inorder(root.Left, x+1, y-1, nodes)
    inorder(root.Right, x+1, y+1, nodes)
  }
}
```

989. Add to Array-Form of Integer

题目

For a non-negative integer `X`, the *array-form of* `X` is an array of its digits in left to right order. For example, if `X = 1231`, then the array form is `[1,2,3,1]`.

Given the array-form `A` of a non-negative integer `X`, return the array-form of the integer `X+K`.

Example 1:

```
Input: A = [1,2,0,0], K = 34
Output: [1,2,3,4]
Explanation: 1200 + 34 = 1234
```

Example 2:

```
Input: A = [2,7,4], K = 181
Output: [4,5,5]
Explanation: 274 + 181 = 455
```

Example 3:

```
Input: A = [2,1,5], K = 806
Output: [1,0,2,1]
Explanation: 215 + 806 = 1021
```

Example 4:

```
Input: A = [9,9,9,9,9,9,9,9,9,9], K = 1
Output: [1,0,0,0,0,0,0,0,0,0,0]
Explanation: 9999999999 + 1 = 10000000000
```

Note:

1. `1 <= A.length <= 10000`
2. `0 <= A[i] <= 9`
3. `0 <= K <= 10000`
4. If `A.length > 1`, then `A[0] != 0`

题目大意

对于非负整数 X 而言，X 的数组形式是每位数字按从左到右的顺序形成的数组。例如，如果 X = 1231，那么其数组形式为 [1,2,3,1]。给定非负整数 X 的数组形式 A，返回整数 X+K 的数组形式。

解题思路

- 简单题，计算 2 个非负整数的和。累加过程中不断的进位，最终输出到数组中记得需要逆序，即数字的高位排在数组下标较小的位置。

代码

```go
package leetcode

func addToArrayForm(A []int, K int) []int {
  res := []int{}
  for i := len(A) - 1; i >= 0; i-- {
    sum := A[i] + K%10
    K /= 10
    if sum >= 10 {
      K++
      sum -= 10
    }
    res = append(res, sum)
  }
  for ; K > 0; K /= 10 {
    res = append(res, K%10)
  }
  reverse(res)
  return res
}

func reverse(A []int) {
  for i, n := 0, len(A); i < n/2; i++ {
    A[i], A[n-1-i] = A[n-1-i], A[i]
  }
}
```

990. Satisfiability of Equality Equations

题目

Given an array equations of strings that represent relationships between variables, each string `equations[i]` has length `4` and takes one of two different forms: `"a==b"` or `"a!=b"`. Here, `a` and `b` are lowercase letters (not necessarily different) that represent one-letter variable names.

Return `true` if and only if it is possible to assign integers to variable names so as to satisfy all the given equations.

Example 1:

```
Input: ["a==b","b!=a"]
Output: false
Explanation: If we assign say, a = 1 and b = 1, then the first equation is satisfied,
but not the second.  There is no way to assign the variables to satisfy both equations.
```

Example 2:

```
Input: ["b==a","a==b"]
Output: true
Explanation: We could assign a = 1 and b = 1 to satisfy both equations.
```

Example 3:

```
Input: ["a==b","b==c","a==c"]
Output: true
```

Example 4:

```
Input: ["a==b","b!=c","c==a"]
Output: false
```

Example 5:

```
Input: ["c==c","b==d","x!=z"]
Output: true
```

Note:

1. `1 <= equations.length <= 500`
2. `equations[i].length == 4`
3. `equations[i][0]` and `equations[i][3]` are lowercase letters
4. `equations[i][1]` is either `'='` or `'!'`
5. `equations[i][2]` is `'='`

题目大意

给定一个由表示变量之间关系的字符串方程组成的数组，每个字符串方程 equations[i] 的长度为 4，并采用两种不同的形式之一："a==b" 或 "a!=b"。在这里，a 和 b 是小写字母（不一定不同），表示单字母变量名。只有当可以将整数分配给变量名，以便满足所有给定的方程时才返回 true，否则返回 false。

提示：

1. 1 <= equations.length <= 500
2. equations[i].length == 4
3. equations[i][0] 和 equations[i][3] 是小写字母
4. equations[i][1] 要么是 '='，要么是 '!'
5. equations[i][2] 是 '='

解题思路

- 给出一个字符串数组，数组里面给出的是一些字母的关系，只有 `'=='` 和 `'! ='` 两种关系。问给出的这些关系中是否存在悖论?
- 这一题是简单的并查集的问题。先将所有 `'=='` 关系的字母 `union()` 起来，然后再一一查看 `'! ='` 关系中是否有 `'=='` 关系的组合，如果有，就返回 `false`，如果遍历完都没有找到，则返回 `true`。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/template"
)

func equationsPossible(equations []string) bool {
  if len(equations) == 0 {
    return false
  }
  uf := template.UnionFind{}
  uf.Init(26)
  for _, equ := range equations {
    if equ[1] == '=' && equ[2] == '=' {
      uf.Union(int(equ[0]-'a'), int(equ[3]-'a'))
    }
  }
  for _, equ := range equations {
    if equ[1] == '!' && equ[2] == '=' {
      if uf.Find(int(equ[0]-'a')) == uf.Find(int(equ[3]-'a')) {
        return false
      }
    }
```

```
    }
    return true
}
```

991. Broken Calculator

题目

On a broken calculator that has a number showing on its display, we can perform two operations:

- **Double**: Multiply the number on the display by 2, or;
- **Decrement**: Subtract 1 from the number on the display.

Initially, the calculator is displaying the number `X`.

Return the minimum number of operations needed to display the number `Y`.

Example 1:

```
Input: X = 2, Y = 3
Output: 2
Explanation: Use double operation and then decrement operation {2 -> 4 -> 3}.
```

Example 2:

```
Input: X = 5, Y = 8
Output: 2
Explanation: Use decrement and then double {5 -> 4 -> 8}.
```

Example 3:

```
Input: X = 3, Y = 10
Output: 3
Explanation:  Use double, decrement and double {3 -> 6 -> 5 -> 10}.
```

Example 4:

```
Input: X = 1024, Y = 1
Output: 1023
Explanation: Use decrement operations 1023 times.
```

Note:

1. `1 <= X <= 10^9`
2. `1 <= Y <= 10^9`

题目大意

在显示着数字的坏计算器上，我们可以执行以下两种操作：

- 双倍（Double）：将显示屏上的数字乘 2；
- 递减（Decrement）：将显示屏上的数字减 1 。

最初，计算器显示数字 X。返回显示数字 Y 所需的最小操作数。

解题思路

- 看到本题的数据规模非常大，`10^9`，算法只能采用 `O(sqrt(n))`、`O(log n)`、`O(1)` 的算法。`O(sqrt(n))` 和 `O(1)` 在本题中是不可能的。所以按照数据规模来估计，本题只能尝试 `O(log n)` 的算法。`O(log n)` 的算法有二分搜索，不过本题不太符合二分搜索算法背景。题目中明显出现乘 2，这很明显是可以达到 `O(log n)` 的。最终确定解题思路是数学方法，循环中会用到乘 2 或者除 2 的计算。
- 既然出现了乘 2 和减一的操作，很容易考虑到奇偶性上。题目要求最小操作数，贪心思想，应该尽可能多的使用除 2 操作，使得 Y 和 X 大小差不多，最后再利用加一操作微调。只要 Y 比 X 大就执行除法操作。当然这里要考虑一次奇偶性，如果 Y 是奇数，先加一变成偶数再除二；如果 Y 是偶数，直接除二。如此操作直到 Y 不大于 X，最后执行 `X-Y` 次加法操作微调即可。

代码

```go
package leetcode

func brokenCalc(X int, Y int) int {
  res := 0
  for Y > X {
    res++
    if Y&1 == 1 {
      Y++
    } else {
      Y /= 2
    }
  }
  return res + X - Y
}
```

992. Subarrays with K Different Integers

题目

Given an array A of positive integers, call a (contiguous, not necessarily distinct) subarray of A good if the number of different integers in that subarray is exactly K.

(For example, [1,2,3,1,2] has 3 different integers: 1, 2, and 3.)

Return the number of good subarrays of A.

Example 1:

```
Input: A = [1,2,1,2,3], K = 2
Output: 7
Explanation: Subarrays formed with exactly 2 different integers: [1,2], [2,1], [1,2],
[2,3], [1,2,1], [2,1,2], [1,2,1,2].
```

Example 2:

```
Input: A = [1,2,1,3,4], K = 3
Output: 3
Explanation: Subarrays formed with exactly 3 different integers: [1,2,1,3], [2,1,3],
[1,3,4].
```

Note:

- 1 <= A.length <= 20000
- 1 <= A[i] <= A.length
- 1 <= K <= A.length

题目大意

这道题考察的是滑动窗口的问题。

给出一个数组 和 K，K 代表窗口能能包含的不同数字的个数。K = 2 代表窗口内只能有 2 个不同的数字。求数组中满足条件 K 的窗口个数。

解题思路

如果只是单纯的滑动窗口去做，会错过一些解。比如在例子 1 中，滑动窗口可以得到 [1,2], [1,2,1], [1,2,1,2], [2,1,2], [1,2], [2,3], 会少 [2,1] 这个解，原因是右边窗口滑动到最右边了，左边窗口在缩小的过程中，右边窗口不会再跟着动了。有同学可能会说，每次左边窗口移动的时候，右边窗口都再次从左边窗口的位置开始重新滑动。这样做确实可以，但是这样做完会发现超时。因为中间包含大量的重复计算。

这道题就需要第 3 个指针。原有滑动窗口的 2 个指针，右窗口保留这个窗口里面最长的子数组，正好有 K 个元素，左窗口右移的逻辑不变。再多用一个指针用来标识正好有 K - 1 个元素的位置。那么正好有 K 个不同元素的解就等于 ans = atMostK(A, K) - atMostK(A, K - 1)。最多有 K 个元素减去最多有 K - 1 个元素得到的窗口中正好有 K 个元素的解。

以例子 1 为例，先求最多有 K 个元素的窗口个数。

```
[1]
[1,2], [2]
[1,2,1], [2,1], [1]
[1,2,1,2], [2,1,2], [1,2], [2]
[2,3], [3]
```

每当窗口滑动到把 K 消耗为 0 的时候，res = right - left + 1 。为什么要这么计算，right - left + 1 代表的含义是，终点为 right，至多为 K 个元素的窗口有多少个。[left,right], [left + 1,right], [left + 2,right] ..... [right,right]。这样算出来的解是包含这道题最终求得的解的，还多出了一部分解。多出来的部分减掉即可，即减掉最多为 K - 1 个元素的解。

最多为 K - 1 个元素的解如下：

```
[1]
[2]
[1]
[2]
[3]
```

两者相减以后得到的结果就是最终结果：

```
[1,2]
[1,2,1], [2,1]
[1,2,1,2], [2,1,2], [1,2]
[2,3]
```

代码

```
package leetcode

func subarraysWithKDistinct(A []int, K int) int {
  return subarraysWithKDistinctSlideWindow(A, K) - subarraysWithKDistinctSlideWindow(A, K-1)
}

func subarraysWithKDistinctSlideWindow(A []int, K int) int {
  left, right, counter, res, freq := 0, 0, K, 0, map[int]int{}
  for right = 0; right < len(A); right++ {
    if freq[A[right]] == 0 {
      counter--
    }
    freq[A[right]]++
    for counter < 0 {
      freq[A[left]]--
      if freq[A[left]] == 0 {
        counter++
      }
      left++
    }
    res += right - left + 1
  }
```

```
    return res
}
```

993. Cousins in Binary Tree

题目

In a binary tree, the root node is at depth `0`, and children of each depth `k` node are at depth `k+1`.

Two nodes of a binary tree are *cousins* if they have the same depth, but have **different parents**.

We are given the `root` of a binary tree with unique values, and the values `x` and `y` of two different nodes in the tree.

Return `true` if and only if the nodes corresponding to the values `x` and `y` are cousins.

Example 1:

```
Input: root = [1,2,3,4], x = 4, y = 3
Output: false
```

Example 2:

```
Input: root = [1,2,3,null,4,null,5], x = 5, y = 4
Output: true
```

Example 3:

```
Input: root = [1,2,3,null,4], x = 2, y = 3
Output: false
```

Note:

1. The number of nodes in the tree will be between `2` and `100`.
2. Each node has a unique integer value from `1` to `100`.

题目大意

在二叉树中，根节点位于深度 0 处，每个深度为 k 的节点的子节点位于深度 k+1 处。如果二叉树的两个节点深度相同，但父节点不同，则它们是一对堂兄弟节点。我们给出了具有唯一值的二叉树的根节点 root，以及树中两个不同节点的值 x 和 y。只有与值 x 和 y 对应的节点是堂兄弟节点时，才返回 true。否则，返回 false。

解题思路

- 给出一个二叉树，和 x ，y 两个值，要求判断这两个值是不是兄弟结点。兄弟结点的定义：都位于同一层，并且父结点是同一个结点。
- 这一题有 3 种解题方法，DFS、BFS、递归。思路都不难。

代码

```
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

// 解法一 递归
func isCousins(root *TreeNode, x int, y int) bool {
  if root == nil {
    return false
  }
  levelX, levelY := findLevel(root, x, 1), findLevel(root, y, 1)
  if levelX != levelY {
    return false
  }
  return !haveSameParents(root, x, y)
}

func findLevel(root *TreeNode, x, level int) int {
  if root == nil {
    return 0
  }
  if root.Val != x {
    leftLevel, rightLevel := findLevel(root.Left, x, level+1), findLevel(root.Right, x,
level+1)
    if leftLevel == 0 {
      return rightLevel
    }
    return leftLevel
  }
  return level
}

func haveSameParents(root *TreeNode, x, y int) bool {
  if root == nil {
    return false
  }
```

```go
  if (root.Left != nil && root.Right != nil && root.Left.Val == x && root.Right.Val ==
y) ||
    (root.Left != nil && root.Right != nil && root.Left.Val == y && root.Right.Val ==
x) {
    return true
  }
  return haveSameParents(root.Left, x, y) || haveSameParents(root.Right, x, y)
}

// 解法二 BFS
type mark struct {
  prev  int
  depth int
}

func isCousinsBFS(root *TreeNode, x int, y int) bool {
  if root == nil {
    return false
  }
  queue := []*TreeNode{root}
  visited := [101]*mark{}
  visited[root.Val] = &mark{prev: -1, depth: 1}

  for len(queue) > 0 {
    node := queue[0]
    queue = queue[1:]
    depth := visited[node.Val].depth
    if node.Left != nil {
      visited[node.Left.Val] = &mark{prev: node.Val, depth: depth + 1}
      queue = append(queue, node.Left)
    }
    if node.Right != nil {
      visited[node.Right.Val] = &mark{prev: node.Val, depth: depth + 1}
      queue = append(queue, node.Right)
    }
  }
  if visited[x] == nil || visited[y] == nil {
    return false
  }
  if visited[x].depth == visited[y].depth && visited[x].prev != visited[y].prev {
    return true
  }
  return false
}

// 解法三 DFS
func isCousinsDFS(root *TreeNode, x int, y int) bool {
  var depth1, depth2, parent1, parent2 int
  dfsCousins(root, x, 0, -1, &parent1, &depth1)
```

```
  dfsCousins(root, y, 0, -1, &parent2, &depth2)
  return depth1 > 1 && depth1 == depth2 && parent1 != parent2
}

func dfsCousins(root *TreeNode, val, depth, last int, parent, res *int) {
  if root == nil {
    return
  }
  if root.Val == val {
    *res = depth
    *parent = last
    return
  }
  depth++
  dfsCousins(root.Left, val, depth, root.Val, parent, res)
  dfsCousins(root.Right, val, depth, root.Val, parent, res)
}
```

995. Minimum Number of K Consecutive Bit Flips

题目

In an array `A` containing only 0s and 1s, a `K`-bit flip consists of choosing a (contiguous) subarray of length `K` and simultaneously changing every 0 in the subarray to 1, and every 1 in the subarray to 0.

Return the minimum number of `K`-bit flips required so that there is no 0 in the array. If it is not possible, return `-1`.

Example 1:

```
Input: A = [0,1,0], K = 1
Output: 2
Explanation: Flip A[0], then flip A[2].
```

Example 2:

```
Input: A = [1,1,0], K = 2
Output: -1
Explanation: No matter how we flip subarrays of size 2, we can't make the array become [1,1,1].
```

Example 3:

```
Input: A = [0,0,0,1,0,1,1,0], K = 3
Output: 3
Explanation:
Flip A[0],A[1],A[2]: A becomes [1,1,1,1,0,1,1,0]
Flip A[4],A[5],A[6]: A becomes [1,1,1,1,1,0,0,0]
Flip A[5],A[6],A[7]: A becomes [1,1,1,1,1,1,1,1]
```

Note:

1. `1 <= A.length <= 30000`
2. `1 <= K <= A.length`

题目大意

在仅包含 0 和 1 的数组 A 中，一次 K 位翻转包括选择一个长度为 K 的（连续）子数组，同时将子数组中的每个 0 更改为 1，而每个 1 更改为 0。返回所需的 K 位翻转的次数，以便数组没有值为 0 的元素。如果不可能，返回 -1。

提示：

1. 1 <= A.length <= 30000
2. 1 <= K <= A.length

解题思路

- 给出一个数组，数组里面的元素只有 0 和 1。给一个长度为 K 的窗口，在这个窗口内的所有元素都会 0-1 翻转。问最后需要翻转几次，使得整个数组都为 1 。如果不能翻转使得最后数组元素都为 1，则输出 -1。
- 拿到这一题首先想到的是贪心算法。例如第 765 题，这类题的描述都是这样的：在一个数组中或者环形数组中通过交换位置，或者翻转变换，达到最终结果，要求找到最少步数。贪心能保证是最小步数(证明略)。按照贪心的思想，这一题也这样做，从数组 0 下标开始往后扫，依次翻转每个 K 大小的窗口内元素。
- 由于窗口大小限制了，所以这题滑动窗口只需要一个边界坐标，用左边界就可以判断了。每一个 `A[i]` 是否需要翻转，是由于 `[ i-k+1, i ]`、`[ i-k+2, i+1 ]`、`[ i-k+3, i+2 ]`.....`[ i-1, i+k ]` 这一系列的窗口翻转`累积影响`的。那如何之前这些窗口`累积`到 `A[i]` 上翻转的次数呢？可以动态的维护一个翻转次数，当 `i` 摆脱了上一次翻转窗口 `K` 的时候，翻转次数就 `-1` 。举个例子：

  ```
  A = [0 0 0 1 0 1 1 0] K = 3

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

当判断 `A[i]` 是否需要翻转的时候，只需要留意每个宽度为 `K` 窗口的左边界。会影响到 A[i] 的窗口的左边界分别是 `i-k+1`、`i-k+2`、`i-k+3`、..... `i-1`，只需要分别看这些窗口有没有翻转就行。这里可以用特殊标记来记录这些窗口的左边界是否被翻转了。如果翻转过，则把窗口左边界的那个数字标记为 2 (为什么要标记为 2 呢？其实设置成什么都可以，只要不是 0 和 1 ，和原有的数字区分开就行)。当 `i≥k` 的时候，代表 `i` 已经脱离了 `i-k` 的这个窗口，因为能影响 `A[i]` 的窗口是从 `i-k+1` 开始的，如果 `A[i-k] == 2` 代表 `i-k` 窗口已经翻转过了，现在既然脱离了它的窗口影响，那么就要把累积的 `flippedTime - 1` 。这样就维护了累积 `flippedTime` 和滑动窗口中累积影响的关系。

- 接下来还需要处理的是 `flippedTime` 与当前 `A[i]` 是否翻转的问题。如果 `flippedTime` 是偶数次，原来的 0 还是 0，就需要再次翻转，如果 `flippedTime` 是奇数次，原来的 0 变成了 1 就不需要翻转了。总结成一条结论就是 `A[i]` 与 `flippedTime` 同奇偶性的时候就要翻转。当 `i + K` 比 `len(A)` 大的时候，代表剩下的这些元素肯定不能在一个窗口里面翻转，则输出 -1 。

代码

```go
package leetcode

func minKBitFlips(A []int, K int) int {
  flippedTime, count := 0, 0
  for i := 0; i < len(A); i++ {
    if i >= K && A[i-K] == 2 {
      flippedTime--
    }
    // 下面这个判断包含了两种情况:
    // 如果 flippedTime 是奇数，且 A[i] == 1 就需要翻转
    // 如果 flippedTime 是偶数，且 A[i] == 0 就需要翻转
    if flippedTime%2 == A[i] {
      if i+K > len(A) {
        return -1
      }
      A[i] = 2
      flippedTime++
      count++
    }
  }
  return count
}
```

996. Number of Squareful Arrays

题目

Given an array `A` of non-negative integers, the array is *squareful* if for every pair of adjacent elements, their sum is a perfect square.

Return the number of permutations of A that are squareful. Two permutations `A1` and `A2` differ if and only if there is some index `i` such that `A1[i] != A2[i]`.

Example 1:

```
Input: [1,17,8]
Output: 2
Explanation:
[1,8,17] and [17,8,1] are the valid permutations.
```

Example 2:

```
Input: [2,2,2]
Output: 1
```

Note:

1. `1 <= A.length <= 12`
2. `0 <= A[i] <= 1e9`

题目大意

给定一个非负整数数组 A，如果该数组每对相邻元素之和是一个完全平方数，则称这一数组为正方形数组。

返回 A 的正方形排列的数目。两个排列 A1 和 A2 不同的充要条件是存在某个索引 i，使得 A1[i] != A2[i]。

解题思路

- 这一题是第 47 题的加强版。第 47 题要求求出一个数组的所有不重复的排列。这一题要求求出一个数组的所有不重复，且相邻两个数字之和都为完全平方数的排列。
- 思路和第 47 题完全一致，只不过增加判断相邻两个数字之和为完全平方数的判断，注意在 DFS 的过程中，需要剪枝，否则时间复杂度很高，会超时。

代码

```go
package leetcode

import (
  "math"
  "sort"
)

func numSquarefulPerms(A []int) int {
```

```go
  if len(A) == 0 {
    return 0
  }
  used, p, res := make([]bool, len(A)), []int{}, [][]int{}
  sort.Ints(A) // 这里是去重的关键逻辑
  generatePermutation996(A, 0, p, &res, &used)
  return len(res)
}

func generatePermutation996(nums []int, index int, p []int, res *[][]int, used *[]bool)
{
  if index == len(nums) {
    checkSquareful := true
    for i := 0; i < len(p)-1; i++ {
      if !checkSquare(p[i] + p[i+1]) {
        checkSquareful = false
        break
      }
    }
    if checkSquareful {
      temp := make([]int, len(p))
      copy(temp, p)
      *res = append(*res, temp)
    }
    return
  }
  for i := 0; i < len(nums); i++ {
    if !(*used)[i] {
      if i > 0 && nums[i] == nums[i-1] && !(*used)[i-1] { // 这里是去重的关键逻辑
        continue
      }
      if len(p) > 0 && !checkSquare(nums[i]+p[len(p)-1]) { // 关键的剪枝条件
        continue
      }
      (*used)[i] = true
      p = append(p, nums[i])
      generatePermutation996(nums, index+1, p, res, used)
      p = p[:len(p)-1]
      (*used)[i] = false
    }
  }
  return
}

func checkSquare(num int) bool {
  tmp := math.Sqrt(float64(num))
  if int(tmp)*int(tmp) == num {
    return true
  }
```

```
    return false
}
```

999. Available Captures for Rook

题目

On an 8 x 8 chessboard, there is one white rook. There also may be empty squares, white bishops, and black pawns. These are given as characters 'R', '.', 'B', and 'p' respectively. Uppercase characters represent white pieces, and lowercase characters represent black pieces.

The rook moves as in the rules of Chess: it chooses one of four cardinal directions (north, east, west, and south), then moves in that direction until it chooses to stop, reaches the edge of the board, or captures an opposite colored pawn by moving to the same square it occupies. Also, rooks cannot move into the same square as other friendly bishops.

Return the number of pawns the rook can capture in one move.

Example 1:

```
Input: [[".",".",".",".",".",".",".",".],[".",".",".","p",".",".",".",".],
[".",".",".","R",".",".",".","p"],[".",".",".",".",".",".",".",".],
[".",".",".",".",".",".",".",".],[".",".",".","p",".",".",".",".],
[".",".",".",".",".",".",".",".],[".",".",".",".",".",".",".",.]]
Output: 3
Explanation:
In this example the rook is able to capture all the pawns.
```

Example 2:

```
Input: [[".",".",".",".",".",".",".",.],[".","p","p","p","p","p",".",.],
[".","p","p","B","p","p",".",.],[".","p","B","R","B","p",".",.],
[".","p","p","B","p","p",".",.],[".","p","p","p","p","p",".",.],
[".",".",".",".",".",".",".",.],[".",".",".",".",".",".",.,.]]
Output: 0
Explanation:
Bishops are blocking the rook to capture any pawn.
```

Example 3:

```
Input: [[".",".",".",".",".",".",".",".],[".",".",".","p",".",".",".",".],
[".",".",".","p",".",".",".",".],["p","p",".","R",".","p","B",".],
[".",".",".",".",".",".",".",".],[".",".",".","B",".",".",".",".],
[".",".",".","p",".",".",".",".],[".",".",".",".",".",".",".",.]]
Output: 3
Explanation:
The rook can capture the pawns at positions b5, d6 and f5.
```

Note:

1. `board.length == board[i].length == 8`
2. `board[i][j]` is either `'R'`, `'.'`, `'B'`, or `'p'`
3. There is exactly one cell with `board[i][j] == 'R'`

题目大意

在一个 8 x 8 的棋盘上，有一个白色的车（Rook），用字符 'R' 表示。棋盘上还可能存在空方块，白色的象（Bishop）以及黑色的卒（pawn），分别用字符 '.'，'B' 和 'p' 表示。不难看出，大写字符表示的是白棋，小写字符表示的是黑棋。车按国际象棋中的规则移动。东，西，南，北四个基本方向任选其一，然后一直向选定的方向移动，直到满足下列四个条件之一：

- 棋手选择主动停下来。
- 棋子因到达棋盘的边缘而停下。
- 棋子移动到某一方格来捕获位于该方格上敌方（黑色）的卒，停在该方格内。
- 车不能进入/越过已经放有其他友方棋子（白色的象）的方格，停在友方棋子前。

你现在可以控制车移动一次，请你统计有多少敌方的卒处于你的捕获范围内（即，可以被一步捕获的棋子数）。

解题思路

- 按照国际象棋的规则移动车，要求输出只移动一次，有多少个卒在车的捕获范围之内
- 简单题，按照国际象棋车的移动规则， 4 个方向分别枚举即可。

代码

```go
package leetcode

func numRookCaptures(board [][]byte) int {
  num := 0
  for i := 0; i < len(board); i++ {
    for j := 0; j < len(board[i]); j++ {
      if board[i][j] == 'R' {
        num += caputure(board, i-1, j, -1, 0) // Up
        num += caputure(board, i+1, j, 1, 0)  // Down
        num += caputure(board, i, j-1, 0, -1) // Left
        num += caputure(board, i, j+1, 0, 1)  // Right
      }
    }
```

```
  }
  return num
}

func caputure(board [][]byte, x, y int, bx, by int) int {
  for x >= 0 && x < len(board) && y >= 0 && y < len(board[x]) && board[x][y] != 'B' {
    if board[x][y] == 'p' {
      return 1
    }
    x += bx
    y += by
  }
  return 0
}
```

1002. Find Common Characters

题目

Given an array `A` of strings made only from lowercase letters, return a list of all characters that show up in all strings within the list **(including duplicates)**. For example, if a character occurs 3 times in all strings but not 4 times, you need to include that character three times in the final answer.

You may return the answer in any order.

Example 1:

```
Input: ["bella","label","roller"]
Output: ["e","l","l"]
```

Example 2:

```
Input: ["cool","lock","cook"]
Output: ["c","o"]
```

Note:

1. `1 <= A.length <= 100`
2. `1 <= A[i].length <= 100`
3. `A[i][j]` is a lowercase letter

题目大意

给定仅有小写字母组成的字符串数组 A，返回列表中的每个字符串中都显示的全部字符（包括重复字符）组成的列表。例如，如果一个字符在每个字符串中出现 3 次，但不是 4 次，则需要在最终答案中包含该字符 3 次。你可以按任意顺序返回答案。

解题思路

- 简单题。给出一个字符串数组 A，要求找出这个数组中每个字符串都包含字符，如果字符出现多次，在最终结果中也需要出现多次。这一题可以用 map 来统计每个字符串的频次，但是如果用数组统计会更快。题目中说了只有小写字母，那么用 2 个 26 位长度的数组就可以统计出来了。遍历字符串数组的过程中，不过的缩小每个字符在每个字符串中出现的频次(因为需要找所有字符串公共的字符，公共的频次肯定就是最小的频次)，得到了最终公共字符的频次数组以后，按顺序输出就可以了。

代码

```go
package leetcode

import "math"

func commonChars(A []string) []string {
  cnt := [26]int{}
  for i := range cnt {
    cnt[i] = math.MaxUint16
  }
  cntInWord := [26]int{}
  for _, word := range A {
    for _, char := range []byte(word) { // compiler trick - here we will not allocate
new memory
      cntInWord[char-'a']++
    }
    for i := 0; i < 26; i++ {
      // 缩小频次，使得统计的公共频次更加准确
      if cntInWord[i] < cnt[i] {
        cnt[i] = cntInWord[i]
      }
    }
    // 重置状态
    for i := range cntInWord {
      cntInWord[i] = 0
    }
  }
  result := make([]string, 0)
  for i := 0; i < 26; i++ {
    for j := 0; j < cnt[i]; j++ {
      result = append(result, string(i+'a'))
    }
  }
  return result
}
```

1003. Check If Word Is Valid After Substitutions

题目

We are given that the string "abc" is valid.

From any valid string V, we may split V into two pieces X and Y such that X + Y (X concatenated with Y) is equal to V. (X or Y may be empty.) Then, X + "abc" + Y is also valid.

If for example S = "abc", then examples of valid strings are: "abc", "aabcbc", "abcabc", "abcabcababcc". Examples of invalid strings are: "abccba", "ab", "cababc", "bac".

Return true if and only if the given string S is valid.

Example 1:

```
Input: "aabcbc"
Output: true
Explanation:
We start with the valid string "abc".
Then we can insert another "abc" between "a" and "bc", resulting in "a" + "abc" + "bc"
which is "aabcbc".
```

Example 2:

```
Input: "abcabcababcc"
Output: true
Explanation:
"abcabcabc" is valid after consecutive insertings of "abc".
Then we can insert "abc" before the last letter, resulting in "abcabcab" + "abc" + "c"
which is "abcabcababcc".
```

Example 3:

```
Input: "abccba"
Output: false
```

Example 4:

```
Input: "cababc"
Output: false
```

Note:

1. 1 <= S.length <= 20000
2. S[i] is 'a', 'b', or 'c'

题目大意

假设 abc 是有效的字符串，对于任何 字符串 V，如果用 abc 把字符串 V 切成 2 半，X 和 Y，组成 X + abc + Y 的字符串，X + abc + Y 的这个字符串依旧是有效的。X 和 Y 可以是空字符串。

例如，"abc"("" + "abc" + ""), "aabcbc"("a" + "abc" + "bc"), "abcabc"("" + "abc" + "abc"), "abcabcababcc"("abc" + "abc" + "ababcc"，其中 "ababcc" 也是有效的，"ab" + "abc" + "c") 都是有效的字符串。

"abccba"("" + "abc" + "cba"，"cba" 不是有效的字符串), "ab"("ab" 也不是有效字符串), "cababc"("c" + "abc" + "bc"，"c"，"bc" 都不是有效字符串), "bac" ("bac" 也不是有效字符串)这些都不是有效的字符串。

任意给一个字符串 S ，要求判断它是否有效，如果有效则输出 true。

解题思路

这一题可以类似括号匹配问题，因为 "abc" 这样的组合就代表是有效的，类似于括号匹配，遇到 "a" 就入栈，当遇到 "b" 字符的时候判断栈顶是不是 "a"，当遇到 "c" 字符的时候需要判断栈顶是不是 "a" 和 "b"。最后如果栈都清空了，就输出 true。

代码

```go
package leetcode

func isValid1003(S string) bool {
  if len(S) < 3 {
    return false
  }
  stack := []byte{}
  for i := 0; i < len(S); i++ {
    if S[i] == 'a' {
      stack = append(stack, S[i])
    } else if S[i] == 'b' {
      if len(stack) > 0 && stack[len(stack)-1] == 'a' {
        stack = append(stack, S[i])
      } else {
        return false
      }
    } else {
      if len(stack) > 1 && stack[len(stack)-1] == 'b' && stack[len(stack)-2] == 'a' {
```

```
        stack = stack[:len(stack)-2]
      } else {
        return false
      }
    }
  }
  return len(stack) == 0
}
```

1004. Max Consecutive Ones III

题目

Given an array A of 0s and 1s, we may change up to K values from 0 to 1.

Return the length of the longest (contiguous) subarray that contains only 1s.

Example 1:

```
Input: A = [1,1,1,0,0,0,1,1,1,1,0], K = 2
Output: 6
Explanation:
[1,1,1,0,0,1,1,1,1,1,1]
Bolded numbers were flipped from 0 to 1.  The longest subarray is underlined.
```

Example 2:

```
Input: A = [0,0,1,1,0,0,1,1,1,0,1,1,0,0,0,1,1,1,1], K = 3
Output: 10
Explanation:
[0,0,1,1,1,1,1,1,1,1,1,1,0,0,0,1,1,1,1]
Bolded numbers were flipped from 0 to 1.  The longest subarray is underlined.
```

Note:

- 1 <= A.length <= 20000
- 0 <= K <= A.length
- A[i] is 0 or 1

题目大意

这道题考察的是滑动窗口的问题。

给出一个数组，数组中元素只包含 0 和 1 。再给一个 K，代表能将 0 变成 1 的次数。要求出经过变换以后，1 连续的最长长度。

解题思路

按照滑动窗口的思路处理即可，不断的更新和维护最大长度。

代码

```
package leetcode

func longestOnes(A []int, K int) int {
  res, left, right := 0, 0, 0
  for left < len(A) {
    if right < len(A) && ((A[right] == 0 && K > 0) || A[right] == 1) {
      if A[right] == 0 {
        K--
      }
      right++
    } else {
      if K == 0 || (right == len(A) && K > 0) {
        res = max(res, right-left)
      }
      if A[left] == 0 {
        K++
      }
      left++
    }
  }
  return res
}
```

1005. Maximize Sum Of Array After K Negations

题目

Given an array A of integers, we must modify the array in the following way: we choose an i and replace A[i] with -A[i], and we repeat this process K times in total. (We may choose the same index i multiple times.)

Return the largest possible sum of the array after modifying it in this way.

Example 1:

```
Input: A = [4,2,3], K = 1
Output: 5
Explanation: Choose indices (1,) and A becomes [4,-2,3].
```

Example 2:

```
Input: A = [3,-1,0,2], K = 3
Output: 6
Explanation: Choose indices (1, 2, 2) and A becomes [3,1,0,2].
```

Example 3:

```
Input: A = [2,-3,-1,5,-4], K = 2
Output: 13
Explanation: Choose indices (1, 4) and A becomes [2,3,-1,5,4].
```

Note:

- 1 <= A.length <= 10000
- 1 <= K <= 10000
- -100 <= A[i] <= 100

题目大意

将数组中的元素变成它的相反数，这种操作执行 K 次之后，求出数组中所有元素的总和最大。

解题思路

这一题可以用最小堆来做，构建最小堆，每次将最小的元素变成它的相反数。然后最小堆调整，再将新的最小元素变成它的相反数。执行 K 次以后求数组中所有的值之和就是最大值。

这道题也可以用排序来实现。排序一次，从最小值开始往后扫，依次将最小值变为相反数。这里需要注意一点，负数都改变成正数以后，接着不是再改变这些变成正数的负数，而是接着改变顺序的正数。因为这些正数是比较小的正数。负数越小，变成正数以后值越大。正数越小，变成负数以后对总和影响最小。具体实现见代码。

代码

```go
package leetcode

import (
  "sort"
```

```go
)

func largestSumAfterKNegations(A []int, K int) int {
  sort.Ints(A)
  minIdx := 0
  for i := 0; i < K; i++ {
    A[minIdx] = -A[minIdx]
    if A[minIdx+1] < A[minIdx] {
      minIdx++
    }
  }
  sum := 0
  for _, a := range A {
    sum += a
  }
  return sum
}
```

1006. Clumsy Factorial

题目

Normally, the factorial of a positive integer `n` is the product of all positive integers less than or equal to `n`. For example, `factorial(10) = 10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1`.

We instead make a *clumsy factorial:* using the integers in decreasing order, we swap out the multiply operations for a fixed rotation of operations: multiply (*), divide (/), add (+) and subtract (-) in this order.

For example, `clumsy(10) = 10 * 9 / 8 + 7 - 6 * 5 / 4 + 3 - 2 * 1`. However, these operations are still applied using the usual order of operations of arithmetic: we do all multiplication and division steps before any addition or subtraction steps, and multiplication and division steps are processed left to right.

Additionally, the division that we use is *floor division* such that `10 * 9 / 8` equals `11`. This guarantees the result is an integer.

`Implement the clumsy` function as defined above: given an integer `N`, it returns the clumsy factorial of `N`.

Example 1:

```
Input:4
Output: 7
Explanation: 7 = 4 * 3 / 2 + 1
```

Example 2:

```
Input:10
Output:12
Explanation:12 = 10 * 9 / 8 + 7 - 6 * 5 / 4 + 3 - 2 * 1
```

Note:

1. `1 <= N <= 10000`
2. `2^31 <= answer <= 2^31 - 1` (The answer is guaranteed to fit within a 32-bit integer.)

题目大意

通常，正整数 n 的阶乘是所有小于或等于 n 的正整数的乘积。例如，factorial(10) = 10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1。相反，我们设计了一个笨阶乘 clumsy：在整数的递减序列中，我们以一个固定顺序的操作符序列来依次替换原有的乘法操作符：乘法(*)，除法(/)，加法(+)和减法(-)。例如，clumsy(10) = 10 * 9 / 8 + 7 - 6 * 5 / 4 + 3 - 2 * 1。然而，这些运算仍然使用通常的算术运算顺序：我们在任何加、减步骤之前执行所有的乘法和除法步骤，并且按从左到右处理乘法和除法步骤。另外，我们使用的除法是地板除法（floor division），所以 10 * 9 / 8 等于 11。这保证结果是一个整数。实现上面定义的笨函数：给定一个整数 N，它返回 N 的笨阶乘。

解题思路

- 按照题意，由于本题没有括号，所以先乘除后加减。4 个操作一组，先算乘法，再算除法，再算加法，最后算减法。减法也可以看成是加法，只是带负号的加法。

代码

```go
package leetcode

func clumsy(N int) int {
  res, count, tmp, flag := 0, 1, N, true
  for i := N - 1; i > 0; i-- {
    count = count % 4
    switch count {
    case 1:
      tmp = tmp * i
    case 2:
      tmp = tmp / i
    case 3:
      res = res + tmp
      flag = true
      tmp = -1
      res = res + i
    case 0:
      flag = false
      tmp = tmp * (i)
    }
    count++
  }
  if !flag {
```

```
    res = res + tmp
  }
  return res
}
```

1011. Capacity To Ship Packages Within D Days

题目

A conveyor belt has packages that must be shipped from one port to another within `D` days.

The `i`-th package on the conveyor belt has a weight of `weights[i]`. Each day, we load the ship with packages on the conveyor belt (in the order given by `weights`). We may not load more weight than the maximum weight capacity of the ship.

Return the least weight capacity of the ship that will result in all the packages on the conveyor belt being shipped within `D` days.

Example 1:

```
Input: weights = [1,2,3,4,5,6,7,8,9,10], D = 5
Output: 15
Explanation:
A ship capacity of 15 is the minimum to ship all the packages in 5 days like this:
1st day: 1, 2, 3, 4, 5
2nd day: 6, 7
3rd day: 8
4th day: 9
5th day: 10

Note that the cargo must be shipped in the order given, so using a ship of capacity 14
and splitting the packages into parts like (2, 3, 4, 5), (1, 6, 7), (8), (9), (10) is
not allowed.
```

Example 2:

```
Input: weights = [3,2,2,4,1,4], D = 3
Output: 6
Explanation:
A ship capacity of 6 is the minimum to ship all the packages in 3 days like this:
1st day: 3, 2
2nd day: 2, 4
3rd day: 1, 4
```

Example 3:

```
Input: weights = [1,2,3,1,1], D = 4
Output: 3
Explanation:
1st day: 1
2nd day: 2
3rd day: 3
4th day: 1, 1
```

Note:

1. `1 <= D <= weights.length <= 50000`
2. `1 <= weights[i] <= 500`

题目大意

传送带上的包裹必须在 D 天内从一个港口运送到另一个港口。

传送带上的第 i 个包裹的重量为 weights[i]。每一天，我们都会按给出重量的顺序往传送带上装载包裹。我们装载的重量不会超过船的最大运载重量。

返回能在 D 天内将传送带上的所有包裹送达的船的最低运载能力。

提示：

- 1 <= D <= weights.length <= 50000
- 1 <= weights[i] <= 500

解题思路

- 给出一个数组和天数 D，要求正好在 D 天把数组中的货物都运完。求传输带上能承受的最小货物重量是多少。
- 这一题和第 410 题完全一样，只不过换了一个题面。代码完全不变。思路解析见第 410 题。

代码

```go
func shipWithinDays(weights []int, D int) int {
  maxNum, sum := 0, 0
  for _, num := range weights {
    sum += num
    if num > maxNum {
      maxNum = num
    }
  }
  if D == 1 {
    return sum
  }
  low, high := maxNum, sum
  for low < high {
    mid := low + (high-low)>>1
```

```go
    if calSum(mid, D, weights) {
      high = mid
    } else {
      low = mid + 1
    }
  }
  return low
}

func calSum(mid, m int, nums []int) bool {
  sum, count := 0, 0
  for _, v := range nums {
    sum += v
    if sum > mid {
      sum = v
      count++
      // 分成 m 块，只需要插桩 m -1 个
      if count > m-1 {
        return false
      }
    }
  }
  return true
}
```

1017. Convert to Base -2

题目

Given a number `N`, return a string consisting of `"0"`s and `"1"`s that represents its value in base `-2` (negative two).

The returned string must have no leading zeroes, unless the string is `"0"`.

Example 1:

```
Input: 2
Output: "110"
Explantion: (-2) ^ 2 + (-2) ^ 1 = 2
```

Example 2:

```
Input: 3
Output: "111"
Explantion: (-2) ^ 2 + (-2) ^ 1 + (-2) ^ 0 = 3
```

Example 3:

```
Input: 4
Output: "100"
Explantion: (-2) ^ 2 = 4
```

Note:

1. `0 <= N <= 10^9`

题目大意

给出数字 N，返回由若干 "0" 和 "1"组成的字符串，该字符串为 N 的负二进制（base -2）表示。除非字符串就是 "0"，否则返回的字符串中不能含有前导零。

提示：

- 0 <= N <= 10^9

解题思路

- 给出一个十进制的数，要求转换成 -2 进制的数
- 这一题仿造十进制转二进制的思路，短除法即可。

代码

```go
package leetcode

import "strconv"

func baseNeg2(N int) string {
  if N == 0 {
    return "0"
  }
  res := ""
  for N != 0 {
    remainder := N % (-2)
    N = N / (-2)
    if remainder < 0 {
      remainder += 2
      N++
    }
    res = strconv.Itoa(remainder) + res
  }
  return res
}
```

1018. Binary Prefix Divisible By 5

题目

Given an array `A` of `0`s and `1`s, consider `N_i`: the i-th subarray from `A[0]` to `A[i]` interpreted as a binary number (from most-significant-bit to least-significant-bit.)

Return a list of booleans `answer`, where `answer[i]` is `true` if and only if `N_i` is divisible by 5.

Example 1:

```
Input: [0,1,1]
Output: [true,false,false]
Explanation:
The input numbers in binary are 0, 01, 011; which are 0, 1, and 3 in base-10.  Only the first number is divisible by 5, so answer[0] is true.
```

Example 2:

```
Input: [1,1,1]
Output: [false,false,false]
```

Example 3:

```
Input: [0,1,1,1,1,1]
Output: [true,false,false,false,true,false]
```

Example 4:

```
Input: [1,1,1,0,1]
Output: [false,false,false,false,false]
```

Note:

1. `1 <= A.length <= 30000`
2. `A[i]` is `0` or `1`

题目大意

给定由若干 0 和 1 组成的数组 A。我们定义 N_i：从 A[0] 到 A[i] 的第 i 个子数组被解释为一个二进制数（从最高有效位到最低有效位）。返回布尔值列表 answer，只有当 N_i 可以被 5 整除时，答案 answer[i] 为 true，否则为 false。

解题思路

- 简单题。每扫描数组中的一个数字，累计转换成二进制数对 5 取余，如果余数为 0，则存入 true，否则存入 false。

代码

```go
package leetcode

func prefixesDivBy5(a []int) []bool {
  res, num := make([]bool, len(a)), 0
  for i, v := range a {
    num = (num<<1 | v) % 5
    res[i] = num == 0
  }
  return res
}
```

1019. Next Greater Node In Linked List

题目

We are given a linked list with head as the first node. Let's number the nodes in the list: node_1, node_2, node_3, ... etc.

Each node may have a next larger value: for node_i, next_larger(node_i) is the node_j.val such that j > i, node_j.val > node_i.val, and j is the smallest possible choice. If such a j does not exist, the next larger value is 0.

Return an array of integers answer, where answer[i] = next_larger(node_{i+1}).

Note that in the example inputs (not outputs) below, arrays such as [2,1,5] represent the serialization of a linked list with a head node value of 2, second node value of 1, and third node value of 5.

Example 1:

```
Input: [2,1,5]
Output: [5,5,0]
```

Example 2:

```
Input: [2,7,4,3,5]
Output: [7,0,5,5,0]
```

Example 3:

```
Input: [1,7,5,1,9,2,5,1]
Output: [7,9,9,9,0,5,0,0]
```

Note:

- 1 <= node.val <= 10^9 for each node in the linked list.
- The given list has length in the range [0, 10000].

题目大意

给出一个链表，要求找出每个结点后面比该结点值大的第一个结点，如果找不到这个结点，则输出 0 。

解题思路

这一题和第 739 题、第 496 题、第 503 题类似。也有 2 种解题方法。先把链表中的数字存到数组中，整道题的思路就和第 739 题完全一致了。普通做法就是 2 层循环。优化的做法就是用单调栈，维护一个单调递减的栈即可。

代码

```go
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */
// 解法一 单调栈
func nextLargerNodes(head *ListNode) []int {
  res, indexes, nums := make([]int, 0), make([]int, 0), make([]int, 0)
  p := head
  for p != nil {
    nums = append(nums, p.Val)
    p = p.Next
  }
```

```
  for i := 0; i < len(nums); i++ {
    res = append(res, 0)
  }
  for i := 0; i < len(nums); i++ {
    num := nums[i]
    for len(indexes) > 0 && nums[indexes[len(indexes)-1]] < num {
      index := indexes[len(indexes)-1]
      res[index] = num
      indexes = indexes[:len(indexes)-1]
    }
    indexes = append(indexes, i)
  }
  return res
}
```

1020. Number of Enclaves

题目

Given a 2D array `A`, each cell is 0 (representing sea) or 1 (representing land)

A move consists of walking from one land square 4-directionally to another land square, or off the boundary of the grid.

Return the number of land squares in the grid for which we **cannot** walk off the boundary of the grid in any number of moves.

Example 1:

```
Input: [[0,0,0,0],[1,0,1,0],[0,1,1,0],[0,0,0,0]]
Output: 3
Explanation:
There are three 1s that are enclosed by 0s, and one 1 that isn't enclosed because its
on the boundary.
```

Example 2:

```
Input: [[0,1,1,0],[0,0,1,0],[0,0,1,0],[0,0,0,0]]
Output: 0
Explanation:
All 1s are either on the boundary or can reach the boundary.
```

Note:

1. `1 <= A.length <= 500`
2. `1 <= A[i].length <= 500`

3. `0 <= A[i][j] <= 1`
4. All rows have the same size.

题目大意

给出一个二维数组 A，每个单元格为 0（代表海）或 1（代表陆地）。移动是指在陆地上从一个地方走到另一个地方（朝四个方向之一）或离开网格的边界。返回网格中无法在任意次数的移动中离开网格边界的陆地单元格的数量。

提示：

- 1 <= A.length <= 500
- 1 <= A[i].length <= 500
- 0 <= A[i][j] <= 1
- 所有行的大小都相同

解题思路

- 给出一个地图，要求输出不和边界连通的 1 的个数。
- 这一题可以用 DFS 也可以用并查集解答。DFS 的思路是深搜的过程中把和边界连通的点都覆盖成 0，最后遍历一遍地图，输出 1 的个数即可。并查集的思路就比较直接了，把能和边界连通的放在一个集合中，剩下的就是不能和边界连通的都在另外一个集合中，输出这个集合里面元素的个数即可。
- 这一题和第 200 题，第 1254 题，第 695 题类似。可以放在一起练习。

代码

```go
func numEnclaves(A [][]int) int {
  m, n := len(A), len(A[0])
  for i := 0; i < m; i++ {
    for j := 0; j < n; j++ {
      if i == 0 || i == m-1 || j == 0 || j == n-1 {
        if A[i][j] == 1 {
          dfsNumEnclaves(A, i, j)
        }
      }
    }
  }
  count := 0
  for i := 0; i < m; i++ {
    for j := 0; j < n; j++ {
      if A[i][j] == 1 {
        count++
      }
    }
  }
  return count
}

func dfsNumEnclaves(A [][]int, x, y int) {
```

```
  if !isInGrid(A, x, y) || A[x][y] == 0 {
    return
  }
  A[x][y] = 0
  for i := 0; i < 4; i++ {
    nx := x + dir[i][0]
    ny := y + dir[i][1]
    dfsNumEnclaves(A, nx, ny)
  }
}
```

1021. Remove Outermost Parentheses

题目

A valid parentheses string is either empty (""), "(" + A + ")", or A + B, where A and B are valid parentheses strings, and + represents string concatenation. For example, "", "()", "(())()", and "(()(()))" are all valid parentheses strings.

A valid parentheses string S is primitive if it is nonempty, and there does not exist a way to split it into S = A+B, with A and B nonempty valid parentheses strings.

Given a valid parentheses string S, consider its primitive decomposition: S = P_1 + P_2 + ... + P_k, where P_i are primitive valid parentheses strings.

Return S after removing the outermost parentheses of every primitive string in the primitive decomposition of S.

Example 1:

```
Input: "(()())(())"
Output: "()()()"
Explanation:
The input string is "(()())(())", with primitive decomposition "(()())" + "(())".
After removing outer parentheses of each part, this is "()()" + "()" = "()()()".
```

Example 2:

```
Input: "(()())(())(()(()))"
Output: "()()()()(())"
Explanation:
The input string is "(()())(())(()(()))", with primitive decomposition "(()())" + "
(())" + "(()(()))".
After removing outer parentheses of each part, this is "()()" + "()" + "()(())" = "()()
()()(())".
```

Example 3:

```
Input: "()()"
Output: ""
Explanation:
The input string is "()()", with primitive decomposition "()" + "()".
After removing outer parentheses of each part, this is "" + "" = "".
```

Note:

- S.length <= 10000
- S[i] is "(" or ")"
- S is a valid parentheses string

题目大意

题目要求去掉最外层的括号。

解题思路

用栈模拟即可。

代码

```go
package leetcode

// 解法一
func removeOuterParentheses(S string) string {
  now, current, ans := 0, "", ""
  for _, char := range S {
    if string(char) == "(" {
      now++
```

```
    } else if string(char) == ")" {
      now--
    }
    current += string(char)
    if now == 0 {
      ans += current[1 : len(current)-1]
      current = ""
    }
  }
  return ans
}

// 解法二
func removeOuterParentheses1(S string) string {
  stack, res, counter := []byte{}, "", 0
  for i := 0; i < len(S); i++ {
    if counter == 0 && len(stack) == 1 && S[i] == ')' {
      stack = stack[1:]
      continue
    }
    if len(stack) == 0 && S[i] == '(' {
      stack = append(stack, S[i])
      continue
    }
    if len(stack) > 0 {
      switch S[i] {
      case '(':
        {
          counter++
          res += "("
        }
      case ')':
        {
          counter--
          res += ")"
        }
      }
    }
  }
  return res
}
```

1025. Divisor Game

题目

Alice and Bob take turns playing a game, with Alice starting first.

Initially, there is a number `N` on the chalkboard. On each player's turn, that player makes a *move* consisting of:

- Choosing any `x` with `0 < x < N` and `N % x == 0`.
- Replacing the number `N` on the chalkboard with `N - x`.

Also, if a player cannot make a move, they lose the game.

Return `True` if and only if Alice wins the game, assuming both players play optimally.

Example 1:

```
Input: 2
Output: true
Explanation: Alice chooses 1, and Bob has no more moves.
```

Example 2:

```
Input: 3
Output: false
Explanation: Alice chooses 1, Bob chooses 1, and Alice has no more moves.
```

Note:

1. `1 <= N <= 1000`

题目大意

爱丽丝和鲍勃一起玩游戏，他们轮流行动。爱丽丝先手开局。最初，黑板上有一个数字 N 。在每个玩家的回合，玩家需要执行以下操作：

- 选出任一 x，满足 0 < x < N 且 N % x == 0 。
- 用 N - x 替换黑板上的数字 N 。

如果玩家无法执行这些操作，就会输掉游戏。只有在爱丽丝在游戏中取得胜利时才返回 True，否则返回 false。假设两个玩家都以最佳状态参与游戏。

解题思路

- 两人相互玩一个游戏，游戏初始有一个数 N，开始游戏的时候，任一方选择一个数 x，满足 `0 < x < N` 并且 `N % x == 0` 的条件，然后 `N-x` 为下一轮开始的数。此轮结束，该另外一个人继续选择数字，两人相互轮流选择。直到某一方再也没法选择数字的时候，输掉游戏。问如果你先手开始游戏，给出 N 的时候，能否直到这局你是否会必胜或者必输？
- 这一题当 `N = 1` 的时候，那一轮的人必输。因为没法找到一个数字能满足 `0 < x < N` 并且 `N % x == 0` 的条件了。必胜策略就是把对方逼至 `N = 1` 的情况。题目中假设了对手也是一个很有头脑的人。初始如果 `N 为偶数`，我就选择 x = 1，对手拿到的数字就是奇数。只要最终能让对手拿到奇数，他就会输。初始如果 `N 为奇数`，N = 1 的时候直接输了，N 为其他奇数的时候，我们也只能选择一个奇数 x，(因为 `N % x == 0` ，N 为奇数，x 一定不会是偶数，因为偶数就能被 2 整除了)，对手由于是一个很有头脑的人，当我们选完 N - x 是偶数的时候，他就选择 1，那么轮到我们拿到的数字又是奇数。对手只要一直保证我们拿到奇数，最终肯定会

逼着我们拿到 1，最终他就会获得胜利。所以经过分析可得，初始数字如果是偶数，有必胜策略，如果初始数字是奇数，有必输的策略。

代码

```
package leetcode

func divisorGame(N int) bool {
  return N%2 == 0
}
```

1026. Maximum Difference Between Node and Ancestor

题目

Given the `root` of a binary tree, find the maximum value `V` for which there exists **different** nodes `A` and `B` where `V = |A.val - B.val|` and `A` is an ancestor of `B`.

(A node A is an ancestor of B if either: any child of A is equal to B, or any child of A is an ancestor of B.)

Example 1:

```
Input: [8,3,10,1,6,null,14,null,null,4,7,13]
Output: 7
Explanation:
We have various ancestor-node differences, some of which are given below :
|8 - 3| = 5
|3 - 7| = 4
|8 - 1| = 7
|10 - 13| = 3
Among all possible differences, the maximum value of 7 is obtained by |8 - 1| = 7.
```

Note:

1. The number of nodes in the tree is between `2` and `5000`.
2. Each node will have value between `0` and `100000`.

题目大意

给定二叉树的根节点 root，找出存在于不同节点 A 和 B 之间的最大值 V，其中 V = |A.val - B.val|，且 A 是 B 的祖先。（如果 A 的任何子节点之一为 B，或者 A 的任何子节点是 B 的祖先，那么我们认为 A 是 B 的祖先）

提示：

- 树中的节点数在 2 到 5000 之间。
- 每个节点的值介于 0 到 100000 之间。

解题思路

- 给出一颗树，要求找出祖先和孩子的最大差值。
- DPS 深搜即可。每个节点和其所有孩子的`最大值`来自于 3 个值，节点本身，递归遍历左子树的最大值，递归遍历右子树的最大值；每个节点和其所有孩子的`最小值`来自于 3 个值，节点本身，递归遍历左子树的最小值，递归遍历右子树的最小值。依次求出自身节点和其所有孩子节点的最大差值，深搜的过程中动态维护最大差值即可。

代码

```go
func maxAncestorDiff(root *TreeNode) int {
  res := 0
  dfsAncestorDiff(root, &res)
  return res
}

func dfsAncestorDiff(root *TreeNode, res *int) (int, int) {
  if root == nil {
    return -1, -1
  }
  leftMax, leftMin := dfsAncestorDiff(root.Left, res)
  if leftMax == -1 && leftMin == -1 {
    leftMax = root.Val
    leftMin = root.Val
  }
  rightMax, rightMin := dfsAncestorDiff(root.Right, res)
  if rightMax == -1 && rightMin == -1 {
    rightMax = root.Val
    rightMin = root.Val
  }
  *res = max(*res, max(abs(root.Val-min(leftMin, rightMin)), abs(root.Val-max(leftMax,
rightMax))))
  return max(leftMax, max(rightMax, root.Val)), min(leftMin, min(rightMin, root.Val))
}
```

1028. Recover a Tree From Preorder Traversal

题目

We run a preorder depth first search on the `root` of a binary tree.

At each node in this traversal, we output `D` dashes (where `D` is the *depth* of this node), then we output the value of this node. *(If the depth of a node is `D`, the depth of its immediate child is `D+1`. The depth of the root node is `0`.)*

If a node has only one child, that child is guaranteed to be the left child.

Given the output `S` of this traversal, recover the tree and return its `root`.

Example 1:

```
Input: "1-2--3--4-5--6--7"
Output: [1,2,5,3,4,6,7]
```

Example 2:

```
Input: "1-2--3---4-5--6---7"
Output: [1,2,5,3,null,6,null,4,null,7]
```

Example 3:

```
Input: "1-401--349---90--88"
Output: [1,401,null,349,88,90]
```

Note:

- The number of nodes in the original tree is between `1` and `1000`.
- Each node will have a value between `1` and `10^9`.

题目大意

我们从二叉树的根节点 root 开始进行深度优先搜索。

在遍历中的每个节点处，我们输出 D 条短划线（其中 D 是该节点的深度），然后输出该节点的值。（如果节点的深度为 D，则其直接子节点的深度为 D + 1。根节点的深度为 0）。如果节点只有一个子节点，那么保证该子节点为左子节点。给出遍历输出 S，还原树并返回其根节点 root。

提示：

- 原始树中的节点数介于 1 和 1000 之间。
- 每个节点的值介于 1 和 10 ^ 9 之间。

解题思路

- 给出一个字符串，字符串是一个树的先根遍历的结果，其中破折号的个数代表层数。请根据这个字符串生成对应的树。
- 这一题解题思路比较明确，用 DFS 就可以解题。边深搜字符串，边根据破折号的个数判断当前节点是否属于

本层。如果不属于本层，回溯到之前的根节点，添加叶子节点以后再继续深搜。需要注意的是每次深搜时，扫描字符串的 index 需要一直保留，回溯也需要用到这个 index。

代码

```
package leetcode

import (
  "strconv"
)

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func recoverFromPreorder(S string) *TreeNode {
  if len(S) == 0 {
    return &TreeNode{}
  }
  root, index, level := &TreeNode{}, 0, 0
  cur := root
  dfsBuildPreorderTree(S, &index, &level, cur)
  return root.Right
}

func dfsBuildPreorderTree(S string, index, level *int, cur *TreeNode) (newIndex *int) {
  if *index == len(S) {
    return index
  }
  if *index == 0 && *level == 0 {
    i := 0
    for i = *index; i < len(S); i++ {
      if !isDigital(S[i]) {
        break
      }
    }
    num, _ := strconv.Atoi(S[*index:i])
    tmp := &TreeNode{Val: num, Left: nil, Right: nil}
    cur.Right = tmp
    nLevel := *level + 1
    index = dfsBuildPreorderTree(S, &i, &nLevel, tmp)
    index = dfsBuildPreorderTree(S, index, &nLevel, tmp)
  }
  i := 0
```

```go
    for i = *index; i < len(S); i++ {
      if isDigital(S[i]) {
        break
      }
    }
    if *level == i-*index {
      j := 0
      for j = i; j < len(S); j++ {
        if !isDigital(S[j]) {
          break
        }
      }
      num, _ := strconv.Atoi(S[i:j])
      tmp := &TreeNode{Val: num, Left: nil, Right: nil}
      if cur.Left == nil {
        cur.Left = tmp
        nLevel := *level + 1
        index = dfsBuildPreorderTree(S, &j, &nLevel, tmp)
        index = dfsBuildPreorderTree(S, index, level, cur)
      } else if cur.Right == nil {
        cur.Right = tmp
        nLevel := *level + 1
        index = dfsBuildPreorderTree(S, &j, &nLevel, tmp)
        index = dfsBuildPreorderTree(S, index, level, cur)
      }
    }
    return index
  }
```

1030. Matrix Cells in Distance Order

题目

We are given a matrix with `R` rows and `C` columns has cells with integer coordinates `(r, c)`, where `0 <= r < R` and `0 <= c < C`.

Additionally, we are given a cell in that matrix with coordinates `(r0, c0)`.

Return the coordinates of all cells in the matrix, sorted by their distance from `(r0, c0)` from smallest distance to largest distance. Here, the distance between two cells `(r1, c1)` and `(r2, c2)` is the Manhattan distance, `|r1 - r2| + |c1 - c2|`. (You may return the answer in any order that satisfies this condition.)

Example 1:

```
Input: R = 1, C = 2, r0 = 0, c0 = 0
Output: [[0,0],[0,1]]
Explanation: The distances from (r0, c0) to other cells are: [0,1]
```

Example 2:

```
Input: R = 2, C = 2, r0 = 0, c0 = 1
Output: [[0,1],[0,0],[1,1],[1,0]]
Explanation: The distances from (r0, c0) to other cells are: [0,1,1,2]
The answer [[0,1],[1,1],[0,0],[1,0]] would also be accepted as correct.
```

Example 3:

```
Input: R = 2, C = 3, r0 = 1, c0 = 2
Output: [[1,2],[0,2],[1,1],[0,1],[1,0],[0,0]]
Explanation: The distances from (r0, c0) to other cells are: [0,1,1,2,2,3]
There are other answers that would also be accepted as correct, such as [[1,2],[1,1],
[0,2],[1,0],[0,1],[0,0]].
```

Note:

1. `1 <= R <= 100`
2. `1 <= C <= 100`
3. `0 <= r0 < R`
4. `0 <= c0 < C`

题目大意

给出 R 行 C 列的矩阵，其中的单元格的整数坐标为 (r, c)，满足 0 <= r < R 且 0 <= c < C。另外，我们在该矩阵中给出了一个坐标为 (r0, c0) 的单元格。

返回矩阵中的所有单元格的坐标，并按到 (r0, c0) 的距离从最小到最大的顺序排，其中，两单元格(r1, c1) 和 (r2, c2) 之间的距离是曼哈顿距离，|r1 - r2| + |c1 - c2|。（你可以按任何满足此条件的顺序返回答案。）

解题思路

- 按照题意计算矩阵内给定点到其他每个点的距离即可

代码

```go
package leetcode

func allCellsDistOrder(R int, C int, r0 int, c0 int) [][]int {
```

```
  longRow, longCol, result := max(abs(r0-0), abs(R-r0)), max(abs(c0-0), abs(C-c0)),
make([][]int, 0)
  maxDistance := longRow + longCol
  bucket := make([][][]int, maxDistance+1)
  for i := 0; i <= maxDistance; i++ {
    bucket[i] = make([][]int, 0)
  }
  for r := 0; r < R; r++ {
    for c := 0; c < C; c++ {
      distance := abs(r-r0) + abs(c-c0)
      tmp := []int{r, c}
      bucket[distance] = append(bucket[distance], tmp)
    }
  }
  for i := 0; i <= maxDistance; i++ {
    for _, buk := range bucket[i] {
      result = append(result, buk)
    }
  }
  return result
}
```

1037. Valid Boomerang

题目

A *boomerang* is a set of 3 points that are all distinct and **not** in a straight line.

Given a list of three points in the plane, return whether these points are a boomerang.

Example 1:

```
Input: [[1,1],[2,3],[3,2]]
Output: true
```

Example 2:

```
Input: [[1,1],[2,2],[3,3]]
Output: false
```

Note:

1. `points.length == 3`
2. `points[i].length == 2`
3. `0 <= points[i][j] <= 100`

题目大意

回旋镖定义为一组三个点，这些点各不相同且不在一条直线上。给出平面上三个点组成的列表，判断这些点是否可以构成回旋镖。

解题思路

- 判断给出的 3 组点能否满足回旋镖。
- 简单题。判断 3 个点组成的 2 条直线的斜率是否相等。由于斜率的计算是除法，还可能遇到分母为 0 的情况，那么可以转换成乘法，交叉相乘再判断是否相等，就可以省去判断分母为 0 的情况了，代码也简洁成一行了。

代码

```
package leetcode

func isBoomerang(points [][]int) bool {
  return (points[0][0]-points[1][0])*(points[0][1]-points[2][1]) != (points[0][0]-points[2][0])*(points[0][1]-points[1][1])
}
```

1038. Binary Search Tree to Greater Sum Tree

题目

Given the `root` of a Binary Search Tree (BST), convert it to a Greater Tree such that every key of the original BST is changed to the original key plus sum of all keys greater than the original key in BST.

As a reminder, a *binary search tree* is a tree that satisfies these constraints:

- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

Note: This question is the same as 538: https://leetcode.com/problems/convert-bst-to-greater-tree/

Example 1:

```
Input: root = [4,1,6,0,2,5,7,null,null,null,3,null,null,null,8]
Output: [30,36,21,36,35,26,15,null,null,null,33,null,null,null,8]
```

Example 2:

```
Input: root = [0,null,1]
Output: [1,null,1]
```

Example 3:

```
Input: root = [1,0,2]
Output: [3,3,2]
```

Example 4:

```
Input: root = [3,2,4,1]
Output: [7,9,4,10]
```

Constraints:

- The number of nodes in the tree is in the range `[1, 100]`.
- `0 <= Node.val <= 100`
- All the values in the tree are **unique**.

- `root` is guaranteed to be a valid binary search tree.

题目大意

给出二叉 搜索 树的根节点，该树的节点值各不相同，请你将其转换为累加树（Greater Sum Tree），使每个节点 node 的新值等于原树中大于或等于 node.val 的值之和。

提醒一下，二叉搜索树满足下列约束条件：

- 节点的左子树仅包含键 小于 节点键的节点。
- 节点的右子树仅包含键 大于 节点键的节点。
- 左右子树也必须是二叉搜索树。

解题思路

- 根据二叉搜索树的有序性，想要将其转换为累加树，只需按照 右节点 - 根节点 - 左节点的顺序遍历，并累加和即可。
- 此题同第 538 题。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/structures"
)

// TreeNode define
type TreeNode = structures.TreeNode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */

func bstToGst(root *TreeNode) *TreeNode {
  if root == nil {
    return root
  }
  sum := 0
  dfs1038(root, &sum)
  return root
}

func dfs1038(root *TreeNode, sum *int) {
```

```
  if root == nil {
    return
  }
  dfs1038(root.Right, sum)
  root.Val += *sum
  *sum = root.Val
  dfs1038(root.Left, sum)
}
```

1040. Moving Stones Until Consecutive II

题目

On an **infinite** number line, the position of the i-th stone is given by `stones[i]`. Call a stone an *endpoint stone* if it has the smallest or largest position.

Each turn, you pick up an endpoint stone and move it to an unoccupied position so that it is no longer an endpoint stone.

In particular, if the stones are at say, `stones = [1,2,5]`, you **cannot** move the endpoint stone at position 5, since moving it to any position (such as 0, or 3) will still keep that stone as an endpoint stone.

The game ends when you cannot make any more moves, ie. the stones are in consecutive positions.

When the game ends, what is the minimum and maximum number of moves that you could have made? Return the answer as an length 2 array: `answer = [minimum_moves, maximum_moves]`

Example 1:

```
Input: [7,4,9]
Output: [1,2]
Explanation:
We can move 4 -> 8 for one move to finish the game.
Or, we can move 9 -> 5, 4 -> 6 for two moves to finish the game.
```

Example 2:

```
Input: [6,5,4,3,10]
Output: [2,3]
We can move 3 -> 8 then 10 -> 7 to finish the game.
Or, we can move 3 -> 7, 4 -> 8, 5 -> 9 to finish the game.
Notice we cannot move 10 -> 2 to finish the game, because that would be an illegal
move.
```

Example 3:

```
Input: [100,101,104,102,103]
Output: [0,0]
```

Note:

1. `3 <= stones.length <= 10^4`
2. `1 <= stones[i] <= 10^9`
3. `stones[i]` have distinct values.

题目大意

在一个长度无限的数轴上，第 i 颗石子的位置为 stones[i]。如果一颗石子的位置最小/最大，那么该石子被称作端点石子。每个回合，你可以将一颗端点石子拿起并移动到一个未占用的位置，使得该石子不再是一颗端点石子。值得注意的是，如果石子像 stones = [1,2,5] 这样，你将无法移动位于位置 5 的端点石子，因为无论将它移动到任何位置（例如 0 或 3），该石子都仍然会是端点石子。当你无法进行任何移动时，即，这些石子的位置连续时，游戏结束。

要使游戏结束，你可以执行的最小和最大移动次数分别是多少？ 以长度为 2 的数组形式返回答案：answer = [minimum_moves, maximum_moves] 。

提示：

1. 3 <= stones.length <= 10^4
2. 1 <= stones[i] <= 10^9
3. stones[i] 的值各不相同。

解题思路

- 给出一个数组，数组里面代表的是石头的坐标。要求移动石头，最终使得这些石头的坐标是一个连续的自然数列。但是规定，当一个石头是端点的时候，是不能移动的，例如 [1,2,5]，5 是端点，不能把 5 移到 3 或者 0 的位置，因为移动之后，这个石头仍然是端点。最终输出将所有石头排成连续的自然数列所需的最小步数和最大步数。
- 这道题的关键就是如何保证端点石头不能再次移动到端点的限制。例如，[5,6,8,9,20]，20 是端点，但是 20 就可以移动到 7 的位置，最终形成 [5,6,7,8,9] 的连续序列。但是 [5,6,7,8,20]，这种情况 20 就不能移动到 9 了，只能让 8 移动到 9，20 再移动到 8 的位置，最终还是形成了 [5,6,7,8,9]，但是步数需要 2 步。经过上述分析，可以得到，端点石头只能往中间空挡的位置移动，如果中间没有空挡，那么需要借助一个石头先制造一个空挡，然后端点石头再插入到中间，这样最少是需要 2 步。
- 再来考虑极值的情况。先看最大步数，最大步数肯定慢慢移动，一次移动一格，并且移动的格数最多。这里有两个极端情况，把数组里面的数全部都移动到最左端点，把数组里面的数全部都移动到最右端点。每次只移动一格。例如，全部都移到最右端点：

```
[3,4,5,6,10] // 初始状态，连续的情况
[4,5,6,7,10] // 第一步，把 3 挪到右边第一个可以插入的位置，即 7
[5,6,7,8,10] // 第二步，把 4 挪到右边第一个可以插入的位置，即 8
[6,7,8,9,10] // 第三步，把 5 挪到右边第一个可以插入的位置，即 9
```

```
[1,3,5,7,10] // 初始状态，不连续的情况
[3,4,5,7,10] // 第一步，把 1 挪到右边第一个可以插入的位置，即 4
[4,5,6,7,10] // 第二步，把 3 挪到右边第一个可以插入的位置，即 6
[5,6,7,8,10] // 第三步，把 4 挪到右边第一个可以插入的位置，即 8
[6,7,8,9,10] // 第四步，把 5 挪到右边第一个可以插入的位置，即 9
```

挪动的过程类似翻滚，最左边的石头挪到右边第一个可以放下的地方。然后不断的往右翻滚。把数组中的数全部都移动到最左边也同理。对比这两种情况的最大值，即是移动的最大步数。

- 再看最小步数。这里就涉及到了滑动窗口了。由于最终是要形成连续的自然数列，所以滑动窗口的大小已经固定成 n 了，从数组的 0 下标可以往右滑动窗口，这个窗口中能包含的数字越多，代表窗口外的数字越少，那么把这些数字放进窗口内的步数也最小。于是可以求得最小步数。这里有一个比较坑的地方就是题目中的那个 `“端点不能移动以后还是端点”` 的限制。针对这种情况，需要额外的判断。如果当前窗口内有 n-1 个元素了，即只有一个端点在窗口外，并且窗口右边界的值减去左边界的值也等于 n-1，代表这个窗口内已经都是连续数字了。这种情况端点想融合到这个连续数列中，最少需要 2 步(上文已经分析过了)。
- 注意一些边界情况。如果窗口从左往右滑动，窗口右边界滑到最右边了，但是窗口右边界的数字减去左边界的数字还是小于窗口大小 n，代表已经滑到头了，可以直接 break 出去。为什么滑到头了呢？由于数组经过从小到大排序以后，数字越往右边越大，当前数字是小值，已经满足了 `stones[right]-stones[left] < n`，左边界继续往右移动只会使得 `stones[left]` 更大，就更加小于 n 了。而我们需要寻找的是 `stones[right]-stones[left] >= n` 的边界点，肯定再也找不到了。

代码

```go
package leetcode

import (
  "math"
  "sort"
)

func numMovesStonesII(stones []int) []int {
  if len(stones) == 0 {
    return []int{0, 0}
  }
  sort.Ints(stones)
  n := len(stones)
  maxStep, minStep, left, right := max(stones[n-1]-stones[1]-n+2, stones[n-2]-
stones[0]-n+2), math.MaxInt64, 0, 0
  for left < n {
    if right+1 < n && stones[right]-stones[left] < n {
      right++
    } else {
      if stones[right]-stones[left] >= n {
        right--
      }
      if right-left+1 == n-1 && stones[right]-stones[left]+1 == n-1 {
```

```
        minStep = min(minStep, 2)
      } else {
        minStep = min(minStep, n-(right-left+1))
      }
      if right == n-1 && stones[right]-stones[left] < n {
        break
      }
      left++
    }
  }
  return []int{minStep, maxStep}
}
```

1047. Remove All Adjacent Duplicates In String

题目

Given a string S of lowercase letters, a duplicate removal consists of choosing two adjacent and equal letters, and removing them.

We repeatedly make duplicate removals on S until we no longer can.

Return the final string after all such duplicate removals have been made. It is guaranteed the answer is unique.

Example 1:

```
Input: "abbaca"
Output: "ca"
Explanation:
For example, in "abbaca" we could remove "bb" since the letters are adjacent and equal,
and this is the only possible move.  The result of this move is that the string is
"aaca", of which only "aa" is possible, so the final string is "ca".
```

Note:

1. 1 <= S.length <= 20000
2. S consists only of English lowercase letters.

题目大意

给出由小写字母组成的字符串 S，重复项删除操作会选择两个相邻且相同的字母，并删除它们。在 S 上反复执行重复项删除操作，直到无法继续删除。在完成所有重复项删除操作后返回最终的字符串。答案保证唯一。

解题思路

用栈模拟，类似“对对碰”，一旦新来的字符和栈顶的字符一样的话，就弹出栈顶字符，直至扫完整个字符串。栈中剩下的字符串就是最终要输出的结果。

代码

```go
package leetcode

func removeDuplicates1047(S string) string {
  stack := []rune{}
  for _, s := range S {
    if len(stack) == 0 || len(stack) > 0 && stack[len(stack)-1] != s {
      stack = append(stack, s)
    } else {
      stack = stack[:len(stack)-1]
    }
  }
  return string(stack)
}
```

1048. Longest String Chain

题目

Given a list of words, each word consists of English lowercase letters.

Let's say `word1` is a predecessor of `word2` if and only if we can add exactly one letter anywhere in `word1` to make it equal to `word2`. For example, `"abc"` is a predecessor of `"abac"`.

A *word chain* is a sequence of words `[word_1, word_2, ..., word_k]` with `k >= 1`, where `word_1` is a predecessor of `word_2`, `word_2` is a predecessor of `word_3`, and so on.

Return the longest possible length of a word chain with words chosen from the given list of `words`.

Example 1:

```
Input: words = ["a","b","ba","bca","bda","bdca"]
Output: 4
Explanation: One of the longest word chain is "a","ba","bda","bdca".
```

Example 2:

```
Input: words = ["xbc","pcxbcf","xb","cxbc","pcxbc"]
Output: 5
```

Constraints:

- `1 <= words.length <= 1000`
- `1 <= words[i].length <= 16`
- `words[i]` only consists of English lowercase letters.

题目大意

给出一个单词列表，其中每个单词都由小写英文字母组成。如果我们可以在 word1 的任何地方添加一个字母使其变成 word2，那么我们认为 word1 是 word2 的前身。例如，"abc" 是 "abac" 的前身。词链是单词 [word_1, word_2, ..., word_k] 组成的序列，k >= 1，其中 word_1 是 word_2 的前身，word_2 是 word_3 的前身，依此类推。从给定单词列表 words 中选择单词组成词链，返回词链的最长可能长度。

解题思路

- 从这题的数据规模上分析，可以猜出此题是 DFS 或者 DP 的题。简单暴力的方法是以每个字符串为链条的起点开始枚举之后的字符串，两两判断能否构成满足题意的前身字符串。这种做法包含很多重叠子问题，例如 a 和 b 能构成前身字符串，以 c 为起点的字符串链条可能用到 a 和 b，以 d 为起点的字符串链条也可能用到 a 和 b。顺其自然，考虑用 DP 的思路解题。
- 先将 words 字符串数组排序，然后用 poss 数组记录下每种长度字符串的在排序数组中的起始下标。然后逆序往前递推。因为初始条件只能得到以最长字符串为起始的字符串链长度为 1 。每选择一个起始字符串，从它的长度 + 1 的每个字符串 j 开始比较，是否能为其前身字符串。如果能构成前身字符串，那么 dp[i] = max(dp[i], 1+dp[j])。最终递推到下标为 0 的字符串。最终输出整个递推过程中的最大长度即为所求。

代码

```
package leetcode

import "sort"

func longestStrChain(words []string) int {
  sort.Slice(words, func(i, j int) bool { return len(words[i]) < len(words[j]) })
  poss, res := make([]int, 16+2), 0
  for i, w := range words {
    if poss[len(w)] == 0 {
      poss[len(w)] = i
    }
  }
  dp := make([]int, len(words))
  for i := len(words) - 1; i >= 0; i-- {
    dp[i] = 1
    for j := poss[len(words[i])+1]; j < len(words) && len(words[j]) == len(words[i])+1;
j++ {
      if isPredecessor(words[j], words[i]) {
        dp[i] = max(dp[i], 1+dp[j])
      }
    }
    res = max(res, dp[i])
```

```
  }
  return res
}

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

func isPredecessor(long, short string) bool {
  i, j := 0, 0
  wasMismatch := false
  for j < len(short) {
    if long[i] != short[j] {
      if wasMismatch {
        return false
      }
      wasMismatch = true
      i++
      continue
    }
    i++
    j++
  }
  return true
}
```

1049. Last Stone Weight II

题目

We have a collection of rocks, each rock has a positive integer weight.

Each turn, we choose **any two rocks** and smash them together. Suppose the stones have weights `x` and `y` with `x <= y`. The result of this smash is:

- If `x == y`, both stones are totally destroyed;
- If `x != y`, the stone of weight `x` is totally destroyed, and the stone of weight `y` has new weight `y-x`.

At the end, there is at most 1 stone left. Return the **smallest possible** weight of this stone (the weight is 0 if there are no stones left.)

Example 1:

```
Input: [2,7,4,1,8,1]
Output: 1
Explanation:
We can combine 2 and 4 to get 2 so the array converts to [2,7,1,8,1] then,
we can combine 7 and 8 to get 1 so the array converts to [2,1,1,1] then,
we can combine 2 and 1 to get 1 so the array converts to [1,1,1] then,
we can combine 1 and 1 to get 0 so the array converts to [1] then that's the optimal
value.
```

Note:

1. `1 <= stones.length <= 30`
2. `1 <= stones[i] <= 100`

题目大意

有一堆石头，每块石头的重量都是正整数。每一回合，从中选出任意两块石头，然后将它们一起粉碎。假设石头的重量分别为 x 和 y，且 x <= y。那么粉碎的可能结果如下：

如果 x == y，那么两块石头都会被完全粉碎；
如果 x != y，那么重量为 x 的石头将会完全粉碎，而重量为 y 的石头新重量为 y-x。
最后，最多只会剩下一块石头。返回此石头最小的可能重量。如果没有石头剩下，就返回 0。

提示：

1. 1 <= stones.length <= 30
2. 1 <= stones[i] <= 1000

解题思路

- 给出一个数组，数组里面的元素代表的是石头的重量。现在要求两个石头对碰，如果重量相同，两个石头都消失，如果一个重一个轻，剩下的石头是两者的差值。问经过这样的多次碰撞以后，能剩下的石头的重量最轻是多少?
- 由于两两石头要发生碰撞，所以可以将整个数组可以分为两部分，如果这两部分的石头重量总和相差不大，那么经过若干次碰撞以后，剩下的石头重量一定是最小的。现在就需要找到这样两堆总重量差不多的两堆石头。这个问题就可以转化为 01 背包问题。从数组中找到 `sum/2` 重量的石头集合，如果一半能尽量达到 `sum/2`，那么另外一半和 `sum/2` 的差是最小的，最好的情况就是两堆石头的重量都是 `sum/2`，那么两两石头对碰以后最后都能消失。01 背包的经典模板可以参考第 416 题。

代码

```go
package leetcode

func lastStoneWeightII(stones []int) int {
  sum := 0
  for _, v := range stones {
```

```
    sum += v
  }
  n, c, dp := len(stones), sum/2, make([]int, sum/2+1)
  for i := 0; i <= c; i++ {
    if stones[0] <= i {
      dp[i] = stones[0]
    } else {
      dp[i] = 0
    }
  }
  for i := 1; i < n; i++ {
    for j := c; j >= stones[i]; j-- {
      dp[j] = max(dp[j], dp[j-stones[i]]+stones[i])
    }
  }
  return sum - 2*dp[c]
}
```

1051. Height Checker

题目

Students are asked to stand in non-decreasing order of heights for an annual photo.

Return the minimum number of students that must move in order for all students to be standing in non-decreasing order of height.

Notice that when a group of students is selected they can reorder in any possible way between themselves and the non selected students remain on their seats.

Example 1:

```
Input: heights = [1,1,4,2,1,3]
Output: 3
Explanation:
Current array : [1,1,4,2,1,3]
Target array  : [1,1,1,2,3,4]
On index 2 (0-based) we have 4 vs 1 so we have to move this student.
On index 4 (0-based) we have 1 vs 3 so we have to move this student.
On index 5 (0-based) we have 3 vs 4 so we have to move this student.
```

Example 2:

```
Input: heights = [5,1,2,3,4]
Output: 5
```

Example 3:

```
Input: heights = [1,2,3,4,5]
Output: 0
```

Constraints:

- `1 <= heights.length <= 100`
- `1 <= heights[i] <= 100`

题目大意

学校在拍年度纪念照时，一般要求学生按照 非递减 的高度顺序排列。请你返回能让所有学生以 非递减 高度排列的最小必要移动人数。注意，当一组学生被选中时，他们之间可以以任何可能的方式重新排序，而未被选中的学生应该保持不动。

解题思路

- 给定一个高度数组，要求输出把这个数组按照非递减高度排列所需移动的最少次数。
- 简单题，最少次数意味着每次移动，一步到位，一步就移动到它所在的最终位置。那么用一个辅助排好序的数组，一一比对计数即可。

代码

```go
package leetcode

func heightChecker(heights []int) int {
  result, checker := 0, []int{}
  checker = append(checker, heights...)
  sort.Ints(checker)
  for i := 0; i < len(heights); i++ {
    if heights[i] != checker[i] {
      result++
    }
  }
  return result
}
```

1052. Grumpy Bookstore Owner

题目

Today, the bookstore owner has a store open for `customers.length` minutes. Every minute, some number of customers (`customers[i]`) enter the store, and all those customers leave after the end of that minute.

On some minutes, the bookstore owner is grumpy. If the bookstore owner is grumpy on the i-th minute, `grumpy[i] = 1`, otherwise `grumpy[i] = 0`. When the bookstore owner is grumpy, the customers of that minute are not satisfied, otherwise they are satisfied.

The bookstore owner knows a secret technique to keep themselves not grumpy for `X` minutes straight, but can only use it once.

Return the maximum number of customers that can be satisfied throughout the day.

Example 1:

```
Input: customers = [1,0,1,2,1,1,7,5], grumpy = [0,1,0,1,0,1,0,1], X = 3
Output: 16
Explanation: The bookstore owner keeps themselves not grumpy for the last 3 minutes.
The maximum number of customers that can be satisfied = 1 + 1 + 1 + 1 + 7 + 5 = 16.
```

Note:

- `1 <= X <= customers.length == grumpy.length <= 20000`
- `0 <= customers[i] <= 1000`
- `0 <= grumpy[i] <= 1`

题目大意

今天，书店老板有一家店打算试营业 customers.length 分钟。每分钟都有一些顾客（customers[i]）会进入书店，所有这些顾客都会在那一分钟结束后离开。在某些时候，书店老板会生气。 如果书店老板在第 i 分钟生气，那么 grumpy[i] = 1，否则 grumpy[i] = 0。 当书店老板生气时，那一分钟的顾客就会不满意，不生气则他们是满意的。书店老板知道一个秘密技巧，能抑制自己的情绪，可以让自己连续 X 分钟不生气，但却只能使用一次。请你返回这一天营业下来，最多有多少客户能够感到满意的数量。

提示：

1. 1 <= X <= customers.length == grumpy.length <= 20000
2. 0 <= customers[i] <= 1000
3. 0 <= grumpy[i] <= 1

解题思路

- 给出一个顾客入店时间表和书店老板发脾气的时间表。两个数组的时间是一一对应的，即相同下标对应的相同的时间。书店老板可以控制自己在 X 分钟内不发火，但是只能控制一次。问有多少顾客能在书店老板不发火的时候在书店里看书。抽象一下，给出一个价值数组和一个装着 0 和 1 的数组，当价值数组的下标对应另外一个数组相同下标的值是 0 的时候，那么这个价值可以累加，当对应是 1 的时候，就不能加上这个价值。现在可以让装着 0 和 1 的数组中连续 X 个数都变成 0，问最终价值最大是多少?
- 这道题是典型的滑动窗口的题目。最暴力的解法是滑动窗口右边界，当与左边界的距离等于 X 的时候，计算此刻对应的数组的总价值。当整个宽度为 X 的窗口滑过整个数组以后，输出维护的最大值即可。这个方法耗时比较长。因为每次计算数组总价值的时候都要遍历整个数组。这里是可以优化的地方。
- 每次计算数组总价值的时候，其实目的是为了找到宽度为 X 的窗口对应里面为 1 的数累加和最大，因为如果把这个窗口里面的 1 都变成 0 以后，那么对最终价值的影响也最大。所以用一个变量 `customer0` 专门记录

脾气数组中为 0 的对应的价值，累加起来。因为不管怎么改变，为 0 的永远为 0，唯一变化的是 1 变成 0 。用 `customer1` 专门记录脾气数组中为 1 的对应的价值。在窗口滑动过程中找到 `customer1` 的最大值。最终要求的最大值就是 `customer0 + maxCustomer1`。

代码

```
package leetcode

// 解法一 滑动窗口优化版
func maxSatisfied(customers []int, grumpy []int, X int) int {
  customer0, customer1, maxCustomer1, left, right := 0, 0, 0, 0, 0
  for ; right < len(customers); right++ {
    if grumpy[right] == 0 {
      customer0 += customers[right]
    } else {
      customer1 += customers[right]
      for right-left+1 > X {
        if grumpy[left] == 1 {
          customer1 -= customers[left]
        }
        left++
      }
      if customer1 > maxCustomer1 {
        maxCustomer1 = customer1
      }
    }
  }
  return maxCustomer1 + customer0
}

// 解法二 滑动窗口暴力版
func maxSatisfied1(customers []int, grumpy []int, X int) int {
  left, right, res := 0, -1, 0
  for left < len(customers) {
    if right+1 < len(customers) && right-left < X-1 {
      right++
    } else {
      if right-left+1 == X {
        res = max(res, sumSatisfied(customers, grumpy, left, right))
      }
      left++
    }
  }
  return res
}

func sumSatisfied(customers []int, grumpy []int, start, end int) int {
```

```
  sum := 0
  for i := 0; i < len(customers); i++ {
    if i < start || i > end {
      if grumpy[i] == 0 {
        sum += customers[i]
      }
    } else {
      sum += customers[i]
    }
  }
  return sum
}
```

1054. Distant Barcodes

题目

In a warehouse, there is a row of barcodes, where the `i`-th barcode is `barcodes[i]`.

Rearrange the barcodes so that no two adjacent barcodes are equal. You may return any answer, and it is guaranteed an answer exists.

Example 1:

```
Input: [1,1,1,2,2,2]
Output: [2,1,2,1,2,1]
```

Example 2:

```
Input: [1,1,1,1,2,2,3,3]
Output: [1,3,1,3,2,1,2,1]
```

Note:

1. `1 <= barcodes.length <= 10000`
2. `1 <= barcodes[i] <= 10000`

题目大意

在一个仓库里，有一排条形码，其中第 i 个条形码为 barcodes[i]。请你重新排列这些条形码，使其中两个相邻的条形码 不能 相等。 你可以返回任何满足该要求的答案，此题保证存在答案。

解题思路

- 这一题和第 767 题原理是完全一样的。第 767 题是 Google 的面试题。
- 解题思路比较简单，先按照每个数字的频次从高到低进行排序，注意会有频次相同的数字。排序以后，分别从

第 0 号位和中间的位置开始往后取数，取完以后即为最终解。

代码

```
package leetcode

import "sort"

func rearrangeBarcodes(barcodes []int) []int {
  bfs := barcodesFrequencySort(barcodes)
  if len(bfs) == 0 {
    return []int{}
  }
  res := []int{}
  j := (len(bfs)-1)/2 + 1
  for i := 0; i <= (len(bfs)-1)/2; i++ {
    res = append(res, bfs[i])
    if j < len(bfs) {
      res = append(res, bfs[j])
    }
    j++
  }
  return res
}

func barcodesFrequencySort(s []int) []int {
  if len(s) == 0 {
    return []int{}
  }
  sMap := map[int]int{}   // 统计每个数字出现的频次
  cMap := map[int][]int{} // 按照频次作为 key 排序
  for _, b := range s {
    sMap[b]++
  }
  for key, value := range sMap {
    cMap[value] = append(cMap[value], key)
  }
  var keys []int
  for k := range cMap {
    keys = append(keys, k)
  }
  sort.Sort(sort.Reverse(sort.IntSlice(keys)))
  res := make([]int, 0)
  for _, k := range keys {
    for i := 0; i < len(cMap[k]); i++ {
      for j := 0; j < k; j++ {
        res = append(res, cMap[k][i])
      }
```

```
    }
  }
  return res
}
```

1073. Adding Two Negabinary Numbers

题目

Given two numbers `arr1` and `arr2` in base **-2**, return the result of adding them together.

Each number is given in *array format*: as an array of 0s and 1s, from most significant bit to least significant bit. For example, `arr = [1,1,0,1]` represents the number `(-2)^3 + (-2)^2 + (-2)^0 = -3`. A number `arr` in *array format* is also guaranteed to have no leading zeros: either `arr == [0]` or `arr[0] == 1`.

Return the result of adding `arr1` and `arr2` in the same format: as an array of 0s and 1s with no leading zeros.

Example 1:

```
Input: arr1 = [1,1,1,1,1], arr2 = [1,0,1]
Output: [1,0,0,0,0]
Explanation: arr1 represents 11, arr2 represents 5, the output represents 16.
```

Note:

1. `1 <= arr1.length <= 1000`
2. `1 <= arr2.length <= 1000`
3. `arr1` and `arr2` have no leading zeros
4. `arr1[i]` is `0` or `1`
5. `arr2[i]` is `0` or `1`

题目大意

给出基数为 -2 的两个数 arr1 和 arr2，返回两数相加的结果。数字以 数组形式 给出：数组由若干 0 和 1 组成，按最高有效位到最低有效位的顺序排列。例如，arr = [1,1,0,1] 表示数字 (-2)^3 + (-2)^2 + (-2)^0 = -3。数组形式 的数字也同样不含前导零：以 arr 为例，这意味着要么 arr == [0]，要么 arr[0] == 1。

返回相同表示形式的 arr1 和 arr2 相加的结果。两数的表示形式为：不含前导零、由若干 0 和 1 组成的数组。

提示：

- 1 <= arr1.length <= 1000
- 1 <= arr2.length <= 1000
- arr1 和 arr2 都不含前导零
- arr1[i] 为 0 或 1
- arr2[i] 为 0 或 1

解题思路

- 给出两个 -2 进制的数，要求计算出这两个数的和，最终表示形式还是 -2 进制。
- 这一题最先想到的思路是先把两个 -2 进制的数转成 10 进制以后做加法，然后把结果表示成 -2 进制。这个思路可行，但是在提交以后会发现数据溢出 int64 了。在第 257 / 267 组测试数据会出现 WA。测试数据见 test 文件。另外换成 big.Add 也不是很方便。所以考虑换一个思路。
- 这道题实际上就是求两个 -2 进制数的加法，为什么还要先转到 10 进制再换回 -2 进制呢？为何不直接进行 -2 进制的加法。所以开始尝试直接进行加法运算。加法是从低位到高位依次累加，遇到进位要从低往高位进位。所以从两个数组的末尾往前扫，模拟低位相加的过程。关键的是进位问题。进位分 3 种情况，依次来讨论：

1. 进位到高位 k ，高位 k 上的两个数数字分别是 0 和 0 。这种情况最终 k 位为 1 。

```
        证明：由于进位是由 k - 1 位进过来的，所以 k - 1 位是 2 个 1 。现在 k 位是 2 个 0，
        所以加起来的和是 2 * (-2)^(k - 1)。
        当 k 为奇数的时候，2 * (-2)^(k - 1) = (-1)^(k - 1)* 2 * 2^(k - 1) = 2^k
        当 k 为偶数的时候，2 * (-2)^(k - 1) = (-1)^(k - 1)* 2 * 2^(k - 1) = -2^k
        综合起来就是 (-2)^k，所以最终 k 位上有一个 1
```

2. 进位到高位 k ，高位 k 上的两个数数字分别是 0 和 1 。这种情况最终 k 位为 0 。

```
        证明：由于进位是由 k - 1 位进过来的，所以 k - 1 位是 2 个 1。现在 k 位是 1 个 0 和 1 个
1，
        所以加起来的和是 (-2)^k + 2 * (-2)^(k - 1)。
        当 k 为奇数的时候，(-2)^k + 2 * (-2)^(k - 1) = -2^k + 2^k = 0
        当 k 为偶数的时候，(-2)^k + 2 * (-2)^(k - 1) = 2^k - 2^k = 0
        综合起来就是 0，所以最终 k 位上有一个 0
```

3. 进位到高位 k ，高位 k 上的两个数数字分别是 1 和 1 。这种情况最终 k 位为 1 。

```
        证明：由于进位是由 k - 1 位进过来的，所以 k - 1 位是 2 个 1 。现在 k 位是 2 个 1，
        所以加起来的和是 2 * (-2)^k + 2 * (-2)^(k - 1)。
        当 k 为奇数的时候，2 * (-2)^k + 2 * (-2)^(k - 1) = -2^(k + 1) + 2^k = 2^k*(1 - 2)
= -2^k
        当 k 为偶数的时候，2 * (-2)^k + 2 * (-2)^(k - 1) = 2^(k + 1) - 2^k = 2^k*(2 - 1) =
2^k
        综合起来就是 (-2)^k，所以最终 k 位上有一个 1
```

- 所以综上所属，-2 进制的进位和 2 进制的进位原理是完全一致的，只不过 -2 进制的进位是 -1，而 2 进制的进位是 1 。由于进位可能在 -2 进制上出现前导 0 ，所以最终结果需要再去除前导 0 。

代码

```
package leetcode
```

```go
// 解法一 模拟进位
func addNegabinary(arr1 []int, arr2 []int) []int {
  carry, ans := 0, []int{}
  for i, j := len(arr1)-1, len(arr2)-1; i >= 0 || j >= 0 || carry != 0; {
    if i >= 0 {
      carry += arr1[i]
      i--
    }
    if j >= 0 {
      carry += arr2[j]
      j--
    }
    ans = append([]int{carry & 1}, ans...)
    carry = -(carry >> 1)
  }
  for idx, num := range ans { // 去掉前导 0
    if num != 0 {
      return ans[idx:]
    }
  }
  return []int{0}
}

// 解法二 标准的模拟，但是这个方法不能 AC，因为测试数据超过了 64 位，普通数据类型无法存储
func addNegabinary1(arr1 []int, arr2 []int) []int {
  return intToNegabinary(negabinaryToInt(arr1) + negabinaryToInt(arr2))
}

func negabinaryToInt(arr []int) int {
  if len(arr) == 0 {
    return 0
  }
  res := 0
  for i := 0; i < len(arr)-1; i++ {
    if res == 0 {
      res += (-2) * arr[i]
    } else {
      res = res * (-2)
      res += (-2) * arr[i]
    }
  }
  return res + 1*arr[len(arr)-1]
}

func intToNegabinary(num int) []int {
  if num == 0 {
    return []int{0}
  }
```

```
  res := []int{}

  for num != 0 {
    remainder := num % (-2)
    num = num / (-2)
    if remainder < 0 {
      remainder += 2
      num++
    }
    res = append([]int{remainder}, res...)
  }
  return res
}
```

1074. Number of Submatrices That Sum to Target

题目

Given a `matrix`, and a `target`, return the number of non-empty submatrices that sum to target.

A submatrix `x1, y1, x2, y2` is the set of all cells `matrix[y]` with `x1 <= x <= x2` and `y1 <= y <= y2`.

Two submatrices `(x1, y1, x2, y2)` and `(x1', y1', x2', y2')` are different if they have some coordinate that is different: for example, if `x1 != x1'`.

Example 1:

```
Input: matrix = [[0,1,0],[1,1,1],[0,1,0]], target = 0
Output: 4
Explanation: The four 1x1 submatrices that only contain 0.
```

Example 2:

```
Input: matrix = [[1,-1],[-1,1]], target = 0
Output: 5
Explanation: The two 1x2 submatrices, plus the two 2x1 submatrices, plus the 2x2
submatrix.
```

Note:

1. `1 <= matrix.length <= 300`
2. `1 <= matrix[0].length <= 300`
3. `-1000 <= matrix[i] <= 1000`
4. `-10^8 <= target <= 10^8`

题目大意

给出矩阵 matrix 和目标值 target，返回元素总和等于目标值的非空子矩阵的数量。

子矩阵 x1, y1, x2, y2 是满足 x1 <= x <= x2 且 y1 <= y <= y2 的所有单元 matrix[x][y] 的集合。

如果 (x1, y1, x2, y2) 和 (x1', y1', x2', y2') 两个子矩阵中部分坐标不同（如：x1 != x1'），那么这两个子矩阵也不同。

提示：

1. 1 <= matrix.length <= 300
2. 1 <= matrix[0].length <= 300
3. -1000 <= matrix[i] <= 1000
4. -10^8 <= target <= 10^8

解题思路

- 给出一个矩阵，要求在这个矩阵中找出子矩阵的和等于 target 的矩阵个数。
- 这一题读完题感觉是滑动窗口的二维版本。如果把它拍扁，在一维数组中，求连续的子数组和为 target，这样就很好做。如果这题不降维，纯暴力解是 O(n^6)。如何优化降低时间复杂度呢？
- 联想到第 1 题 Two Sum 问题，可以把 2 个数求和的问题优化到 O(n)。这里也用类似的思想，用一个 map 来保存行方向上曾经出现过的累加和，相减就可以得到本行的和。这里可能读者会有疑惑，为什么不能每一行都单独保存呢？为什么一定要用累加和相减的方式来获取每一行的和呢？因为这一题要求子矩阵所有解，如果只单独保存每一行的和，只能求得小的子矩阵，子矩阵和子矩阵组成的大矩阵的情况会漏掉(当然再循环一遍，把子矩阵累加起来也可以，但是这样就多了一层循环了)，例如子矩阵是 1*4 的，但是 2 个这样的子矩阵摞在一起形成 2 * 4 也能满足条件，如果不用累加和的办法，只单独存每一行的和，最终还要有组合的步骤。经过这样的优化，可以从 O(n^6) 优化到 O(n^4)，能 AC 这道题，但是时间复杂度太高了。如何优化？
- 首先，子矩阵需要上下左右 4 个边界，4 个变量控制循环就需要 O(n^4)，行和列的区间累加还需要 O(n^2)。行和列的区间累加可以通过 preSum 来解决。例如 `sum[i,j] = sum[j] - sum[i - 1]`，其中 sum[k] 中存的是从 0 到 K 的累加和： {{< katex display >}}
  \sum_{0}^{k} matrix[i]
  {{< /katex >}}
  那么一个区间内的累加和可以由这个区间的右边界减去区间左边界左边的那个累加和得到(由于是闭区间，所需要取左边界左边的和)。经过这样的处理，列方向的维度就被我们拍扁了。
- 再来看看行方向的和，现在每一列的和都可以通过区间相减的方法得到。那么这道题就变成了第 1 题 Two Sum 的问题了。Two Sum 问题只需要 O(n) 的时间复杂度求解，这一题由于是二维的，所以两个列的边界还需要循环，所以最终优化下来的时间复杂度是 O(n^3)。计算 presum 可以直接用原数组，所以空间复杂度只有一个 O(n) 的字典。
- 类似思路的题目有第 560 题，第 304 题。

代码

```
package leetcode

func numSubmatrixSumTarget(matrix [][]int, target int) int {
```

```go
  m, n, res := len(matrix), len(matrix[0]), 0
  for row := range matrix {
    for col := 1; col < len(matrix[row]); col++ {
      matrix[row][col] += matrix[row][col-1]
    }
  }
  for i := 0; i < n; i++ {
    for j := i; j < n; j++ {
      counterMap, sum := make(map[int]int, m), 0
      counterMap[0] = 1 // 题目保证一定有解，所以这里初始化是 1
      for row := 0; row < m; row++ {
        if i > 0 {
          sum += matrix[row][j] - matrix[row][i-1]
        } else {
          sum += matrix[row][j]
        }
        res += counterMap[sum-target]
        counterMap[sum]++
      }
    }
  }
  return res
}

// 暴力解法 O(n^4)
func numSubmatrixSumTarget1(matrix [][]int, target int) int {
  m, n, res, sum := len(matrix), len(matrix[0]), 0, 0
  for i := 0; i < n; i++ {
    for j := i; j < n; j++ {
      counterMap := map[int]int{}
      counterMap[0] = 1 // 题目保证一定有解，所以这里初始化是 1
      sum = 0
      for row := 0; row < m; row++ {
        for k := i; k <= j; k++ {
          sum += matrix[row][k]
        }
        res += counterMap[sum-target]
        counterMap[sum]++
      }
    }
  }
  return res
}

// 暴力解法超时！ O(n^6)
func numSubmatrixSumTarget2(matrix [][]int, target int) int {
  res := 0
  for startx := 0; startx < len(matrix); startx++ {
    for starty := 0; starty < len(matrix[startx]); starty++ {
```

```
      for endx := startx; endx < len(matrix); endx++ {
        for endy := starty; endy < len(matrix[startx]); endy++ {
          if sumSubmatrix(matrix, startx, starty, endx, endy) == target {
            //fmt.Printf("startx = %v, starty = %v, endx = %v, endy = %v\n", startx,
starty, endx, endy)
            res++
          }
        }
      }
    }
  }
  return res
}

func sumSubmatrix(matrix [][]int, startx, starty, endx, endy int) int {
  sum := 0
  for i := startx; i <= endx; i++ {
    for j := starty; j <= endy; j++ {
      sum += matrix[i][j]
    }
  }
  return sum
}
```

1078. Occurrences After Bigram

题目

Given words `first` and `second`, consider occurrences in some `text` of the form "`first second third`", where `second` comes immediately after `first`, and `third` comes immediately after `second`.

For each such occurrence, add "`third`" to the answer, and return the answer.

Example 1:

```
Input: text = "alice is a good girl she is a good student", first = "a", second =
"good"
Output: ["girl","student"]
```

Example 2:

```
Input: text = "we will we will rock you", first = "we", second = "will"
Output: ["we","rock"]
```

Note:

1. `1 <= text.length <= 1000`

2. `text` consists of space separated words, where each word consists of lowercase English letters.
3. `1 <= first.length, second.length <= 10`
4. `first` and `second` consist of lowercase English letters.

题目大意

给出第一个词 first 和第二个词 second，考虑在某些文本 text 中可能以 "first second third" 形式出现的情况，其中 second 紧随 first 出现，third 紧随 second 出现。对于每种这样的情况，将第三个词 "third" 添加到答案中，并返回答案。

解题思路

- 简单题。给出一个 text，要求找出紧接在 first 和 second 后面的那个字符串，有多个就输出多个。解法很简单，先分解出 words 每个字符串，然后依次遍历进行字符串匹配。匹配到 first 和 second 以后，输出之后的那个字符串。

代码

```go
package leetcode

import "strings"

func findOcurrences(text string, first string, second string) []string {
  var res []string
  words := strings.Split(text, " ")
  if len(words) < 3 {
    return []string{}
  }
  for i := 2; i < len(words); i++ {
    if words[i-2] == first && words[i-1] == second {
      res = append(res, words[i])
    }
  }
  return res
}
```

1079. Letter Tile Possibilities

题目

You have a set of `tiles`, where each tile has one letter `tiles[i]` printed on it. Return the number of possible non-empty sequences of letters you can make.

Example 1:

```
Input: "AAB"
Output: 8
Explanation: The possible sequences are "A", "B", "AA", "AB", "BA", "AAB", "ABA",
"BAA".
```

Example 2:

```
Input: "AAABBC"
Output: 188
```

Note:

1. `1 <= tiles.length <= 7`
2. `tiles` consists of uppercase English letters.

题目大意

你有一套活字字模 tiles，其中每个字模上都刻有一个字母 tiles[i]。返回你可以印出的非空字母序列的数目。提示：

- 1 <= tiles.length <= 7
- tiles 由大写英文字母组成

解题思路

- 题目要求输出所有非空字母序列的数目。这一题是排列和组合的结合题目。组合是可以选择一个字母，二个字母，..... n 个字母。每个组合内是排列问题。比如选择 2 个字母，字母之间相互排序不同是影响最终结果的，不同的排列顺序是不同的解。
- 这道题目由于不需要输出所有解，所以解法可以优化，例如我们在递归计算解的时候，不需要真的遍历原字符串，只需要累加一些字母的频次就可以。当然如果要输出所有解，就需要真实遍历原字符串了(见解法二)。简单的做法是每次递归按照频次累加。因为每次增加一个字母一定是 26 个大写字母中的一个。这里需要注意的是，增加的只能是 26 个字母里面还能取出“机会”的字母，例如递归到到第 3 轮了，A 用完了，这个时候只能取频次还不为 0 的字母拼上去。

代码

```go
package leetcode

// 解法一 DFS
func numTilePossibilities(tiles string) int {
  m := make(map[byte]int)
  for i := range tiles {
    m[tiles[i]]++
  }
  arr := make([]int, 0)
  for _, v := range m {
    arr = append(arr, v)
```

```go
  }
  return numTileDFS(arr)
}

func numTileDFS(arr []int) (r int) {
  for i := 0; i < len(arr); i++ {
    if arr[i] == 0 {
      continue
    }
    r++
    arr[i]--
    r += numTileDFS(arr)
    arr[i]++
  }
  return
}

// 解法二 DFS 暴力解法
func numTilePossibilities1(tiles string) int {
  res, tmp, tMap, used := 0, []byte{}, make(map[string]string, 0), make([]bool,
len(tiles))
  findTile([]byte(tiles), tmp, &used, 0, &res, tMap)
  return res
}

func findTile(tiles, tmp []byte, used *[]bool, index int, res *int, tMap
map[string]string) {
  flag := true
  for _, v := range *used {
    if v == false {
      flag = false
      break
    }
  }
  if flag {
    return
  }
  for i := 0; i < len(tiles); i++ {
    if (*used)[i] == true {
      continue
    }
    tmp = append(tmp, tiles[i])
    (*used)[i] = true
    if _, ok := tMap[string(tmp)]; !ok {
      //fmt.Printf("i = %v tiles = %v 找到了结果 = %v\n", i, string(tiles), string(tmp))
      *res++
    }
    tMap[string(tmp)] = string(tmp)
```

```
    findTile([]byte(tiles), tmp, used, i+1, res, tMap)
    tmp = tmp[:len(tmp)-1]
    (*used)[i] = false
  }
}
```

1089. Duplicate Zeros

题目

Given a fixed length array `arr` of integers, duplicate each occurrence of zero, shifting the remaining elements to the right.

Note that elements beyond the length of the original array are not written.

Do the above modifications to the input array **in place**, do not return anything from your function.

Example 1:

```
Input: [1,0,2,3,0,4,5,0]
Output: null
Explanation: After calling your function, the input array is modified to:
[1,0,0,2,3,0,0,4]
```

Example 2:

```
Input: [1,2,3]
Output: null
Explanation: After calling your function, the input array is modified to: [1,2,3]
```

Note:

1. `1 <= arr.length <= 10000`
2. `0 <= arr[i] <= 9`

题目大意

给你一个长度固定的整数数组 arr，请你将该数组中出现的每个零都复写一遍，并将其余的元素向右平移。注意：请不要在超过该数组长度的位置写入元素。要求：请对输入的数组 就地 进行上述修改，不要从函数返回任何东西。

解题思路

- 给一个固定长度的数组，把数组元素为 0 的元素都往后复制一遍，后面的元素往后移，超出数组长度的部分删除。
- 简单题，按照题意，用 append 和 slice 操作即可。

代码

```
package leetcode

func duplicateZeros(arr []int) {
  for i := 0; i < len(arr); i++ {
    if arr[i] == 0 && i+1 < len(arr) {
      arr = append(arr[:i+1], arr[i:len(arr)-1]...)
      i++
    }
  }
}
```

1091. Shortest Path in Binary Matrix

题目

In an N by N square grid, each cell is either empty (0) or blocked (1).

A *clear path from top-left to bottom-right* has length `k` if and only if it is composed of cells `C_1, C_2, ..., C_k` such that:

- Adjacent cells `C_i` and `C_{i+1}` are connected 8-directionally (ie., they are different and share an edge or corner)
- `C_1` is at location `(0, 0)` (ie. has value `grid[0][0]`)
- `C_k` is at location `(N-1, N-1)` (ie. has value `grid[N-1][N-1]`)
- If `C_i` is located at `(r, c)`, then `grid[r][c]` is empty (ie. `grid[r][c] == 0`).

Return the length of the shortest such clear path from top-left to bottom-right. If such a path does not exist, return -1.

Example 1:

```
Input: [[0,1],[1,0]]
Output: 2
```

Example 2:

```
Input: [[0,0,0],[1,1,0],[1,1,0]]
Output: 4
```

Note:

1. `1 <= grid.length == grid[0].length <= 100`
2. `grid[r][c]` is `0` or `1`

题目大意

在一个 N × N 的方形网格中，每个单元格有两种状态：空（0）或者阻塞（1）。一条从左上角到右下角、长度为 k 的畅通路径，由满足下述条件的单元格 C_1, C_2, ..., C_k 组成：

- 相邻单元格 C_i 和 C*{i+1} 在八个方向之一上连通（此时，C_i 和* C{i+1} 不同且共享边或角）
- C_1 位于 (0, 0)（即，值为 grid[0][0]）
- C_k 位于 (N-1, N-1)（即，值为 grid[N-1][N-1]）
- 如果 C_i 位于 (r, c)，则 grid[r][c] 为空（即，grid[r][c] == 0）

返回这条从左上角到右下角的最短畅通路径的长度。如果不存在这样的路径，返回 -1 。

解题思路

- 这一题是简单的找最短路径。利用 BFS 从左上角逐步扩展到右下角，便可以很容易求解。注意每轮扩展需要考虑 8 个方向。

代码

```go
var dir = [][]int{
  {-1, -1},
  {-1, 0},
  {-1, 1},
  {0, 1},
  {0, -1},
  {1, -1},
  {1, 0},
  {1, 1},
}

func shortestPathBinaryMatrix(grid [][]int) int {
  visited := make([][]bool, 0)
  for range make([]int, len(grid)) {
    visited = append(visited, make([]bool, len(grid[0])))
  }
  dis := make([][]int, 0)
  for range make([]int, len(grid)) {
    dis = append(dis, make([]int, len(grid[0])))
  }
  if grid[0][0] == 1 {
    return -1
  }
  if len(grid) == 1 && len(grid[0]) == 1 {
    return 1
  }
```

```go
  queue := []int{0}
  visited[0][0], dis[0][0] = true, 1
  for len(queue) > 0 {
    cur := queue[0]
    queue = queue[1:]
    curx, cury := cur/len(grid[0]), cur%len(grid[0])
    for d := 0; d < 8; d++ {
      nextx := curx + dir[d][0]
      nexty := cury + dir[d][1]
      if isInBoard(grid, nextx, nexty) && !visited[nextx][nexty] && grid[nextx][nexty]
== 0 {
        queue = append(queue, nextx*len(grid[0])+nexty)
        visited[nextx][nexty] = true
        dis[nextx][nexty] = dis[curx][cury] + 1
        if nextx == len(grid)-1 && nexty == len(grid[0])-1 {
          return dis[nextx][nexty]
        }
      }
    }
  }
  return -1
}

func isInBoard(board [][]int, x, y int) bool {
  return x >= 0 && x < len(board) && y >= 0 && y < len(board[0])
}
```

1093. Statistics from a Large Sample

题目

We sampled integers between `0` and `255`, and stored the results in an array `count`: `count[k]` is the number of integers we sampled equal to `k`.

Return the minimum, maximum, mean, median, and mode of the sample respectively, as an array of **floating point numbers**. The mode is guaranteed to be unique.

(Recall that the median of a sample is:

- *The middle element, if the elements of the sample were sorted and the number of elements is odd;*
- *The average of the middle two elements, if the elements of the sample were sorted and the number of elements is even.)*

Example 1:

```
Input: count =
[0,1,3,4,0,0,0
```

- 中位数的处理相对麻烦一些，因为需要分非 0 的 count 之和是奇数还是偶数两种情况。先假设非 0 的 count 和为 cnt，那么如果 cnt 是奇数的话，只需要找到 cnt/2 的位置即可，通过不断累加 count 的值，直到累加和超过 ≥ cnt/2。如果 cnt 是偶数的话，需要找到 cnt/2 + 1 和 cnt/2 的位置，找法和奇数情况相同，不过需要找两次(可以放到一个循环中做两次判断)。

代码

```go
package leetcode

func sampleStats(count []int) []float64 {
  res := make([]float64, 5)
  res[0] = 255
  sum := 0
  for _, val := range count {
    sum += val
  }
  left, right := sum/2, sum/2
  if (sum % 2) == 0 {
    right++
  }
  pre, mode := 0, 0
  for i, val := range count {
    if val > 0 {
      if i < int(res[0]) {
        res[0] = float64(i)
      }
      res[1] = float64(i)
    }
    res[2] += float64(i*val) / float64(sum)
    if pre < left && pre+val >= left {
      res[3] += float64(i) / 2.0
    }
    if pre < right && pre+val >= right {
      res[3] += float64(i) / 2.0
    }
    pre += val

    if val > mode {
      mode = val
      res[4] = float64(i)
    }
  }
  return res
}
```

1105. Filling Bookcase Shelves

题目

We have a sequence of `books`: the `i`-th book has thickness `books[i][0]` and height `books[i][1]`.

We want to place these books **in order** onto bookcase shelves that have total width `shelf_width`.

We choose some of the books to place on this shelf (such that the sum of their thickness is `<= shelf_width`), then build another level of shelf of the bookcase so that the total height of the bookcase has increased by the maximum height of the books we just put down. We repeat this process until there are no more books to place.

Note again that at each step of the above process, the order of the books we place is the same order as the given sequence of books. For example, if we have an ordered list of 5 books, we might place the first and second book onto the first shelf, the third book on the second shelf, and the fourth and fifth book on the last shelf.

Return the minimum possible height that the total bookshelf can be after placing shelves in this manner.

Example 1:

```
Input: books = [[1,1],[2,3],[2,3],[1,1],[1,1],[1,1],[1,2]], shelf_width = 4
Output: 6
Explanation:
The sum of the heights of the 3 shelves are 1 + 3 + 2 = 6.
Notice that book number 2 does not have to be on the first shelf.
```

Constraints:

- `1 <= books.length <= 1000`
- `1 <= books[i][0] <= shelf_width <= 1000`
- `1 <= books[i][1] <= 1000`

题目大意

附近的家居城促销，你买回了一直心仪的可调节书架，打算把自己的书都整理到新的书架上。你把要摆放的书 books 都整理好，叠成一摞：从上往下，第 i 本书的厚度为 books[i][0]，高度为 books[i][1]。按顺序将这些书摆放到总宽度为 shelf_width 的书架上。

先选几本书放在书架上（它们的厚度之和小于等于书架的宽度 shelf_width），然后再建一层书架。重复这个过程，直到把所有的书都放在书架上。

需要注意的是，在上述过程的每个步骤中，摆放书的顺序与你整理好的顺序相同。 例如，如果这里有 5 本书，那么可能的一种摆放情况是：第一和第二本书放在第一层书架上，第三本书放在第二层书架上，第四和第五本书放在最后一层书架上。每一层所摆放的书的最大高度就是这一层书架的层高，书架整体的高度为各层高之和。以这种方式布置书架，返回书架整体可能的最小高度。

解题思路

- 给出一个数组，数组里面每个元素代表的是一个本书的宽度和高度。要求按照书籍的顺序，把书摆到宽度为 `shelf_width` 的书架上。问最终放下所有书籍以后，书架的最小高度。
- 这一题的解题思路是动态规划。`dp[i]` 代表放置前 `i` 本书所需要的书架最小高度。初始值 dp[0] = 0，其他为最大值 1000*1000。遍历每一本书，把当前这本书作为书架最后一层的最后一本书，将这本书之前的书向后调整，看看是否可以减少之前的书架高度。状态转移方程为 `dp[i] = min(dp[i] , dp[j - 1] + h)`，其中 `j` 表示最后一层所能容下书籍的索引，`h` 表示最后一层最大高度。`j` 调整完一遍以后就能找出书架最小高度值了。时间复杂度 O(n^2)。

代码

```
package leetcode

func minHeightShelves(books [][]int, shelfWidth int) int {
  dp := make([]int, len(books)+1)
  dp[0] = 0
  for i := 1; i <= len(books); i++ {
    width, height := books[i-1][0], books[i-1][1]
    dp[i] = dp[i-1] + height
    for j := i - 1; j > 0 && width+books[j-1][0] <= shelfWidth; j-- {
      height = max(height, books[j-1][1])
      width += books[j-1][0]
      dp[i] = min(dp[i], dp[j-1]+height)
    }
  }
  return dp[len(books)]
}
```

1108. Defanging an IP Address

题目

Given a valid (IPv4) IP `address`, return a defanged version of that IP address.

A *defanged IP address* replaces every period `"."` with `"[.]"`.

Example 1:

```
Input: address = "1.1.1.1"
Output: "1[.]1[.]1[.]1"
```

Example 2:

```
Input: address = "255.100.50.0"
Output: "255[.]100[.]50[.]0"
```

Constraints:

- The given `address` is a valid IPv4 address.

题目大意

给你一个有效的 IPv4 地址 address，返回这个 IP 地址的无效化版本。所谓无效化 IP 地址，其实就是用 "[.]" 代替了每个 "."。

提示：

- 给出的 address 是一个有效的 IPv4 地址

解题思路

- 给出一个 IP 地址，要求把点替换成 `[.]`。
- 简单题，按照题意替换即可。

代码

```
package leetcode

import "strings"

func defangIPaddr(address string) string {
  return strings.Replace(address, ".", "[.]", -1)
}
```

1110. Delete Nodes And Return Forest

题目

Given the `root` of a binary tree, each node in the tree has a distinct value.

After deleting all nodes with a value in `to_delete`, we are left with a forest (a disjoint union of trees).

Return the roots of the trees in the remaining forest. You may return the result in any order.

Example 1:

```
Input: root = [1,2,3,4,5,6,7], to_delete = [3,5]
Output: [[1,2,null,4],[6],[7]]
```

Constraints:

- The number of nodes in the given tree is at most `1000`.
- Each node has a distinct value between `1` and `1000`.
- `to_delete.length <= 1000`
- `to_delete` contains distinct values between `1` and `1000`.

题目大意

给出二叉树的根节点 root，树上每个节点都有一个不同的值。如果节点值在 to_delete 中出现，我们就把该节点从树上删去，最后得到一个森林（一些不相交的树构成的集合）。返回森林中的每棵树。你可以按任意顺序组织答案。

提示：

- 树中的节点数最大为 1000。
- 每个节点都有一个介于 1 到 1000 之间的值，且各不相同。
- to_delete.length <= 1000
- to_delete 包含一些从 1 到 1000、各不相同的值。

解题思路

- 给出一棵树，再给出一个数组，要求删除数组中相同元素值的节点。输出最终删除以后的森林。
- 简单题。边遍历树，边删除数组中的元素。这里可以先把数组里面的元素放入 map 中，加速查找。遇到相同的元素就删除节点。这里需要特殊判断的是当前删除的节点是否是根节点，如果是根节点需要根据条件置空它的左节点或者右节点。

代码

```go
func delNodes(root *TreeNode, toDelete []int) []*TreeNode {
  if root == nil {
    return nil
  }
  res, deleteMap := []*TreeNode{}, map[int]bool{}
  for _, v := range toDelete {
    deleteMap[v] = true
  }
  dfsDelNodes(root, deleteMap, true, &res)
  return res
}

func dfsDelNodes(root *TreeNode, toDel map[int]bool, isRoot bool, res *[]*TreeNode)
bool {
  if root == nil {
    return false
  }
  if isRoot && !toDel[root.Val] {
    *res = append(*res, root)
  }
```

```
  isRoot = false
  if toDel[root.Val] {
    isRoot = true
  }
  if dfsDelNodes(root.Left, toDel, isRoot, res) {
    root.Left = nil
  }
  if dfsDelNodes(root.Right, toDel, isRoot, res) {
    root.Right = nil
  }
  return isRoot
}
```

1111. Maximum Nesting Depth of Two Valid Parentheses Strings

题目

A string is a *valid parentheses string* (denoted VPS) if and only if it consists of `"("` and `")"` characters only, and:

- It is the empty string, or
- It can be written as `AB` (`A` concatenated with `B`), where `A` and `B` are VPS's, or
- It can be written as `(A)`, where `A` is a VPS.

We can similarly define the *nesting depth* `depth(S)` of any VPS `S` as follows:

- `depth("") = 0`
- `depth(A + B) = max(depth(A), depth(B))`, where `A` and `B` are VPS's
- `depth("(" + A + ")") = 1 + depth(A)`, where `A` is a VPS.

For example, `""`, `"()()"`, and `"()(()())"` are VPS's (with nesting depths 0, 1, and 2), and `")("` and `"(()"` are not VPS's.

Given a VPS seq, split it into two disjoint subsequences `A` and `B`, such that `A` and `B` are VPS's (and `A.length + B.length = seq.length`).

Now choose **any** such `A` and `B` such that `max(depth(A), depth(B))` is the minimum possible value.

Return an `answer` array (of length `seq.length`) that encodes such a choice of `A` and `B`: `answer[i] = 0` if `seq[i]` is part of `A`, else `answer[i] = 1`. Note that even though multiple answers may exist, you may return any of them.

Example 1:

```
Input: seq = "(()())"
Output: [0,1,1,1,1,0]
```

Example 2:

```
Input: seq = "()(())()"
Output: [0,0,0,1,1,0,1,1]
```

Constraints:

- `1 <= seq.size <= 10000`

题目大意

有效括号字符串 仅由 "(" 和 ")" 构成，并符合下述几个条件之一：

- 空字符串
- 连接，可以记作 AB（A 与 B 连接），其中 A 和 B 都是有效括号字符串
- 嵌套，可以记作 (A)，其中 A 是有效括号字符串

类似地，我们可以定义任意有效括号字符串 s 的 嵌套深度 depth(S)：

- s 为空时，depth("") = 0
- s 为 A 与 B 连接时，depth(A + B) = max(depth(A), depth(B))，其中 A 和 B 都是有效括号字符串
- s 为嵌套情况，depth("(" + A + ")") = 1 + depth(A)，其中 A 是有效括号字符串

例如："""，"()()"，和 "()(()())" 都是有效括号字符串，嵌套深度分别为 0，1，2，而 ")(" 和 "(()" 都不是有效括号字符串。

给你一个有效括号字符串 seq，将其分成两个不相交的子序列 A 和 B，且 A 和 B 满足有效括号字符串的定义（注意：A.length + B.length = seq.length）。

现在，你需要从中选出 任意 一组有效括号字符串 A 和 B，使 max(depth(A), depth(B)) 的可能取值最小。

返回长度为 seq.length 答案数组 answer ，选择 A 还是 B 的编码规则是：如果 seq[i] 是 A 的一部分，那么 answer[i] = 0。否则，answer[i] = 1。即便有多个满足要求的答案存在，你也只需返回 一个。

解题思路

- 给出一个括号字符串。选出 A 部分和 B 部分，使得 `max(depth(A), depth(B))` 值最小。在最终的数组中输出 0 和 1，0 标识是 A 部分，1 标识是 B 部分。
- 这一题想要 `max(depth(A), depth(B))` 值最小，可以使用贪心思想。如果 A 部分和 B 部分都尽快括号匹配，不深层次嵌套，那么总的层次就会变小。只要让嵌套的括号中属于 A 的和属于 B 的间隔排列即可。例如：“`((((()))))`”，上面的字符串的嵌套深度是 4，按照上述的贪心思想，则标记为 0101 1010。
- 这一题也可以用二分的思想来解答。把深度平分给 A 部分和 B 部分。
  - 第一次遍历，先计算最大深度
  - 第二次遍历，把深度小于等于最大深度一半的括号标记为 0(给 A 部分)，否则标记为 1(给 B 部分)

代码

```go
package leetcode

// 解法一 二分思想
func maxDepthAfterSplit(seq string) []int {
  stack, maxDepth, res := 0, 0, []int{}
  for _, v := range seq {
    if v == '(' {
      stack++
      maxDepth = max(stack, maxDepth)
    } else {
      stack--
    }
  }
  stack = 0
  for i := 0; i < len(seq); i++ {
    if seq[i] == '(' {
      stack++
      if stack <= maxDepth/2 {
        res = append(res, 0)
      } else {
        res = append(res, 1)
      }
    } else {
      if stack <= maxDepth/2 {
        res = append(res, 0)
      } else {
        res = append(res, 1)
      }
      stack--
    }
  }
  return res
}

// 解法二 模拟
func maxDepthAfterSplit1(seq string) []int {
  stack, top, res := make([]int, len(seq)), -1, make([]int, len(seq))
  for i, r := range seq {
    if r == ')' {
      res[i] = res[stack[top]]
      top--
      continue
    }
    top++
    stack[top] = i
    res[i] = top % 2
  }
  return res
```

```
}
```

1122. Relative Sort Array

题目

Given two arrays `arr1` and `arr2`, the elements of `arr2` are distinct, and all elements in `arr2` are also in `arr1`.

Sort the elements of `arr1` such that the relative ordering of items in `arr1` are the same as in `arr2`. Elements that don't appear in `arr2` should be placed at the end of `arr1` in **ascending** order.

Example 1:

```
Input: arr1 = [2,3,1,3,2,4,6,7,9,2,19], arr2 = [2,1,4,3,9,6]
Output: [2,2,2,1,4,3,3,9,6,7,19]
```

Constraints:

- `arr1.length, arr2.length <= 1000`
- `0 <= arr1[i], arr2[i] <= 1000`
- Each `arr2[i]` is distinct.
- Each `arr2[i]` is in `arr1`.

题目大意

给你两个数组，arr1 和 arr2，

- arr2 中的元素各不相同
- arr2 中的每个元素都出现在 arr1 中

对 arr1 中的元素进行排序，使 arr1 中项的相对顺序和 arr2 中的相对顺序相同。未在 arr2 中出现过的元素需要按照升序放在 arr1 的末尾。

提示：

- arr1.length, arr2.length <= 1000
- 0 <= arr1[i], arr2[i] <= 1000
- arr2 中的元素 arr2[i] 各不相同
- arr2 中的每个元素 arr2[i] 都出现在 arr1 中

解题思路

- 给出 2 个数组 A 和 B，A 中包含 B 中的所有元素。要求 A 按照 B 的元素顺序排序，B 中没有的元素再接着排在后面，从小到大排序。
- 这一题有多种解法。一种暴力的解法就是依照题意，先把 A 中的元素都统计频次放在 map 中，然后 按照 B 的顺序输出，接着再把剩下的元素排序接在后面。还有一种桶排序的思想，由于题目限定了 A 的大小是

1000，这个数量级很小，所以可以用 1001 个桶装所有的数，把数都放在桶里，这样默认就已经排好序了。接下来的做法和前面暴力解法差不多，按照频次输出。B 中以外的元素就按照桶的顺序依次输出即可。

代码

```
package leetcode

import "sort"

// 解法一 桶排序，时间复杂度 O(n^2)
func relativeSortArray(A, B []int) []int {
  count := [1001]int{}
  for _, a := range A {
    count[a]++
  }
  res := make([]int, 0, len(A))
  for _, b := range B {
    for count[b] > 0 {
      res = append(res, b)
      count[b]--
    }
  }
  for i := 0; i < 1001; i++ {
    for count[i] > 0 {
      res = append(res, i)
      count[i]--
    }
  }
  return res
}

// 解法二 模拟，时间复杂度 O(n^2)
func relativeSortArray1(arr1 []int, arr2 []int) []int {
  leftover, m, res := []int{}, make(map[int]int), []int{}
  for _, v := range arr1 {
    m[v]++
  }
  for _, s := range arr2 {
    count := m[s]
    for i := 0; i < count; i++ {
      res = append(res, s)
    }
    m[s] = 0
  }
  for v, count := range m {
    for i := 0; i < count; i++ {
      leftover = append(leftover, v)
    }
```

```
    }
    sort.Ints(leftover)
    res = append(res, leftover...)
    return res
}
```

1123. Lowest Common Ancestor of Deepest Leaves

题目

Given a rooted binary tree, return the lowest common ancestor of its deepest leaves.

Recall that:

- The node of a binary tree is a *leaf* if and only if it has no children
- The *depth* of the root of the tree is 0, and if the depth of a node is `d`, the depth of each of its children is `d+1`.
- The *lowest common ancestor* of a set `S` of nodes is the node `A` with the largest depth such that every node in S is in the subtree with root `A`.

Example 1:

```
Input: root = [1,2,3]
Output: [1,2,3]
Explanation:
The deepest leaves are the nodes with values 2 and 3.
The lowest common ancestor of these leaves is the node with value 1.
The answer returned is a TreeNode object (not an array) with serialization "[1,2,3]".
```

Example 2:

```
Input: root = [1,2,3,4]
Output: [4]
```

Example 3:

```
Input: root = [1,2,3,4,5]
Output: [2,4,5]
```

Constraints:

- The given tree will have between 1 and 1000 nodes.
- Each node of the tree will have a distinct value between 1 and 1000.

题目大意

给你一个有根节点的二叉树，找到它最深的叶节点的最近公共祖先。

回想一下：

- 叶节点 是二叉树中没有子节点的节点
- 树的根节点的 深度 为 0，如果某一节点的深度为 d，那它的子节点的深度就是 d+1
- 如果我们假定 A 是一组节点 S 的 最近公共祖先，S 中的每个节点都在以 A 为根节点的子树中，且 A 的深度达到此条件下可能的最大值。

提示：

- 给你的树中将有 1 到 1000 个节点。
- 树中每个节点的值都在 1 到 1000 之间。

解题思路

- 给出一颗树，找出最深的叶子节点的最近公共祖先 LCA。
- 这一题思路比较直接。先遍历找到最深的叶子节点，如果左右子树的最深的叶子节点深度相同，那么当前节点就是它们的最近公共祖先。如果左右子树的最深的深度不等，那么需要继续递归往下找符合题意的 LCA。如果最深的叶子节点没有兄弟，那么公共父节点就是叶子本身，否则返回它的 LCA。
- 有几个特殊的测试用例，见测试文件。特殊的点就是最深的叶子节点没有兄弟节点的情况。

代码

```go
package leetcode

/**
 * Definition for a binary tree node.
 * type TreeNode struct {
 *     Val int
 *     Left *TreeNode
 *     Right *TreeNode
 * }
 */
func lcaDeepestLeaves(root *TreeNode) *TreeNode {
  if root == nil {
    return nil
  }
  lca, maxLevel := &TreeNode{}, 0
  lcaDeepestLeavesDFS(&lca, &maxLevel, 0, root)
  return lca
}

func lcaDeepestLeavesDFS(lca **TreeNode, maxLevel *int, depth int, root *TreeNode) int
{
  *maxLevel = max(*maxLevel, depth)
  if root == nil {
    return depth
  }
```

```
  depthLeft := lcaDeepestLeavesDFS(lca, maxLevel, depth+1, root.Left)
  depthRight := lcaDeepestLeavesDFS(lca, maxLevel, depth+1, root.Right)
  if depthLeft == *maxLevel && depthRight == *maxLevel {
    *lca = root
  }
  return max(depthLeft, depthRight)
}
```

1128. Number of Equivalent Domino Pairs

题目

Given a list of `dominoes`, `dominoes[i] = [a, b]` is *equivalent* to `dominoes[j] = [c, d]` if and only if either (`a==c` and `b==d`), or (`a==d` and `b==c`) - that is, one domino can be rotated to be equal to another domino.

Return the number of pairs `(i, j)` for which `0 <= i < j < dominoes.length`, and `dominoes[i]` is equivalent to `dominoes[j]`.

Example 1:

```
Input: dominoes = [[1,2],[2,1],[3,4],[5,6]]
Output: 1
```

Constraints:

- `1 <= dominoes.length <= 40000`
- `1 <= dominoes[i][j] <= 9`

题目大意

给你一个由一些多米诺骨牌组成的列表 dominoes。如果其中某一张多米诺骨牌可以通过旋转 0 度或 180 度得到另一张多米诺骨牌，我们就认为这两张牌是等价的。形式上，dominoes[i] = [a, b] 和 dominoes[j] = [c, d] 等价的前提是 ac 且 bd，或是 ad 且 bc。

在 0 <= i < j < dominoes.length 的前提下，找出满足 dominoes[i] 和 dominoes[j] 等价的骨牌对 (i, j) 的数量。

提示：

- 1 <= dominoes.length <= 40000
- 1 <= dominoes[i][j] <= 9

解题思路

- 给出一组多米诺骨牌，求出这组牌中相同牌的个数。牌相同的定义是：牌的 2 个数字相同(正序或者逆序相同都算相同)
- 简单题。由于牌是 2 个数，所以将牌的 2 个数 hash 成一个 2 位数，比较大小即可，正序和逆序都 hash 成 2

位数，然后在桶中比较是否已经存在，如果不存在，跳过，如果存在，计数。

代码

```
package leetcode

func numEquivDominoPairs(dominoes [][]int) int {
  if dominoes == nil || len(dominoes) == 0 {
    return 0
  }
  result, buckets := 0, [100]int{}
  for _, dominoe := range dominoes {
    key, rotatedKey := dominoe[0]*10+dominoe[1], dominoe[1]*10+dominoe[0]
    if dominoe[0] != dominoe[1] {
      if buckets[rotatedKey] > 0 {
        result += buckets[rotatedKey]
      }
    }
    if buckets[key] > 0 {
      result += buckets[key]
      buckets[key]++
    } else {
      buckets[key]++
    }
  }
  return result
}
```

1137. N-th Tribonacci Number

题目

The Tribonacci sequence Tn is defined as follows:

T0 = 0, T1 = 1, T2 = 1, and Tn+3 = Tn + Tn+1 + Tn+2 for n >= 0.

Given `n`, return the value of Tn.

Example 1:

```
Input: n = 4
Output: 4
Explanation:
T_3 = 0 + 1 + 1 = 2
T_4 = 1 + 1 + 2 = 4
```

Example 2:

```
Input: n = 25
Output: 1389537
```

Constraints:

- `0 <= n <= 37`
- The answer is guaranteed to fit within a 32-bit integer, ie. `answer <= 2^31 - 1`.

题目大意

泰波那契序列 Tn 定义如下：

T0 = 0, T1 = 1, T2 = 1, 且在 n >= 0 的条件下 Tn+3 = Tn + Tn+1 + Tn+2

给你整数 n，请返回第 n 个泰波那契数 Tn 的值。

提示：

- 0 <= n <= 37
- 答案保证是一个 32 位整数，即 answer <= 2^31 - 1。

解题思路

- 求泰波那契数列中的第 n 个数。
- 简单题，按照题意定义计算即可。

代码

```go
package leetcode

func tribonacci(n int) int {
  if n < 2 {
    return n
  }
  trib, prev, prev2 := 1, 1, 0
  for n > 2 {
    trib, prev, prev2 = trib+prev+prev2, trib, prev
    n--
  }
  return trib
}
```

1143. Longest Common Subsequence

题目

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.

题目大意

给定两个字符串 text1 和 text2，返回这两个字符串的最长 公共子序列 的长度。如果不存在 公共子序列 ，返回 0 。一个字符串的 子序列 是指这样一个新的字符串：它是由原字符串在不改变字符的相对顺序的情况下删除某些字符（也可以不删除任何字符）后组成的新字符串。例如，"ace" 是 "abcde" 的子序列，但 "aec" 不是 "abcde" 的子序列。两个字符串的 公共子序列 是这两个字符串所共同拥有的子序列。

解题思路

- 这一题是经典的最长公共子序列的问题。解题思路是二维动态规划。假设字符串 `text1` 和 `text2` 的长度分别为 `m` 和 `n`，创建 `m+1` 行 `n+1` 列的二维数组 `dp`，定义 `dp[i][j]` 表示长度为 i 的 `text1[0:i-1]` 和长度为 j 的 `text2[0:j-1]` 的最长公共子序列的长度。先考虑边界条件。当 `i = 0` 时，`text1[]` 为空字符串，它与任何字符串的最长公共子序列的长度都是 `0`，所以 `dp[0][j] = 0`。同理当 `j = 0` 时，`text2[]`

为空字符串，它与任何字符串的最长公共子序列的长度都是 `0`，所以 `dp[i][0] = 0`。由于二维数组的大小特意增加了 `1`，即 `m+1` 和 `n+1`，并且默认值是 `0`，所以不需要再初始化赋值了。

- 当 `text1[i-1] = text2[j-1]` 时，将这两个相同的字符称为公共字符，考虑 `text1[0:i-1]` 和 `text2[0:j-1]` 的最长公共子序列，再增加一个字符（即公共字符）即可得到 `text1[0:i]` 和 `text2[0:j]` 的最长公共子序列，所以 `dp[i][j]=dp[i-1][j-1]+1`。当 `text1[i-1] != text2[j-1]` 时，最长公共子序列一定在 `text[0:i-1], text2[0:j]` 和 `text[0:i], text2[0:j-1]` 中取得。即 `dp[i][j] = max(dp[i-1][j], dp[i][j-1])`。所以状态转移方程如下：

  {{< katex display >}}
  dp[i][j] = \left{\begin{matrix}dp[i-1][j-1]+1 &,text1[i-1]=text2[j-1]\max(dp[i-1][j],dp[i][j-1])&,text1[i-1]\neq text2[j-1]\end{matrix}\right.
  {{< /katex >}}

- 最终结果存储在 `dp[len(text1)][len(text2)]` 中。时间复杂度 `O(mn)`，空间复杂度 `O(mn)`，其中 `m` 和 `n` 分别是 `text1` 和 `text2` 的长度。

代码

```go
package leetcode

func longestCommonSubsequence(text1 string, text2 string) int {
  if len(text1) == 0 || len(text2) == 0 {
    return 0
  }
  dp := make([][]int, len(text1)+1)
  for i := range dp {
    dp[i] = make([]int, len(text2)+1)
  }
  for i := 1; i < len(text1)+1; i++ {
    for j := 1; j < len(text2)+1; 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[len(text1)][len(text2)]
}

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

1145. Binary Tree Coloring Game

题目

Two players play a turn based game on a binary tree. We are given the `root` of this binary tree, and the number of nodes `n` in the tree. `n` is odd, and each node has a distinct value from `1` to `n`.

Initially, the first player names a value `x` with `1 <= x <= n`, and the second player names a value `y` with `1 <= y <= n` and `y != x`. The first player colors the node with value `x` red, and the second player colors the node with value `y` blue.

Then, the players take turns starting with the first player. In each turn, that player chooses a node of their color (red if player 1, blue if player 2) and colors an **uncolored** neighbor of the chosen node (either the left child, right child, or parent of the chosen node.)

If (and only if) a player cannot choose such a node in this way, they must pass their turn. If both players pass their turn, the game ends, and the winner is the player that colored more nodes.

You are the second player. If it is possible to choose such a `y` to ensure you win the game, return `true`. If it is not possible, return `false`.

Example 1:

```
Input: root = [1,2,3,4,5,6,7,8,9,10,11], n = 11, x = 3
Output: true
Explanation: The second player can choose the node with value 2.
```

Constraints:

- `root` is the root of a binary tree with `n` nodes and distinct node values from `1` to `n`.
- `n` is odd.
- `1 <= x <= n <= 100`

题目大意

有两位极客玩家参与了一场「二叉树着色」的游戏。游戏中，给出二叉树的根节点 root，树上总共有 n 个节点，且 n 为奇数，其中每个节点上的值从 1 到 n 各不相同。游戏从「一号」玩家开始（「一号」玩家为红色，「二号」玩家为蓝色），最开始时，

- 「一号」玩家从 [1, n] 中取一个值 x（1 <= x <= n）；
- 「二号」玩家也从 [1, n] 中取一个值 y（1 <= y <= n）且 y != x。
- 「一号」玩家给值为 x 的节点染上红色，而「二号」玩家给值为 y 的节点染上蓝色。

之后两位玩家轮流进行操作，每一回合，玩家选择一个他之前涂好颜色的节点，将所选节点一个 未着色 的邻节点（即左右子节点、或父节点）进行染色。如果当前玩家无法找到这样的节点来染色时，他的回合就会被跳过。若两个玩家都没有可以染色的节点时，游戏结束。着色节点最多的那位玩家获得胜利 ✌️。现在，假设你是「二号」玩家，根据所给出的输入，假如存在一个 y 值可以确保你赢得这场游戏，则返回 true；若无法获胜，就请返回

false。

提示：

- 二叉树的根节点为 root，树上由 n 个节点，节点上的值从 1 到 n 各不相同。
- n 为奇数。
- 1 <= x <= n <= 100

解题思路

- 2 个人参加二叉树着色游戏。二叉树节点数为奇数。1 号玩家和 2 号玩家分别在二叉树上选项一个点着色。每一回合，玩家选择一个他之前涂好颜色的节点，将所选节点一个 未着色 的邻节点（即左右子节点、或父节点）进行染色。当有人不能选点着色的时候，他的那个回合会被跳过。双方都没法继续着色的时候游戏结束。着色多的人获胜。问二号玩家是否存在必胜策略?

- 如图所示，当一号玩家选择了一个红色的结点，可能会将二叉树切割为 3 个部分（连通分量），如果选择的是根结点，则可能是 2 个部分或 1 个部分，如果选择叶结点，则是 1 个部分。不过无论哪种情况都无关紧要，我们都可以当成 3 个部分来对待，例如一号玩家选择了一个叶结点，我们也可以把叶结点的左右两个空指针看成大小为 0 的两个部分。
- 那么二号玩家怎样选择蓝色结点才是最优呢？答案是：选择离红色结点最近，且所属连通分量规模最大的那个点。也就是示例图中的 1 号结点。如果我们选择了 1 号结点为蓝色结点，那么可以染成红色的点就只剩下 6 号点和 7 号点了，而蓝色可以把根结点和其左子树全部占据。
- 如何确定蓝色是否有必胜策略，就可以转换为，被红色点切割的三个连通分量中，是否存在一个连通分量，大小大于所有结点数目的一半。统计三个连通分量大小的过程，可以用深度优先搜索（DFS）来实现。当遍历到某一结点，其结点值等于选定的红色结点时，我们统计这个结点的左子树 `red_left` 和右子树 `red_right` 的大小，那么我们就已经找到两个连通分量的大小了，最后一个父结点连通分量的大小，可以用结点总数减去这两个连通分量大小，再减去红色所占结点，即 `parent = n - red_left - red_right - 1`。

代码

```
func btreeGameWinningMove(root *TreeNode, n int, x int) bool {
  var left, right int
  dfsBtreeGameWinningMove(root, &left, &right, x)
  up := n - left - right - 1
  n /= 2
  return left > n || right > n || up > n
}

func dfsBtreeGameWinningMove(node *TreeNode, left, right *int, x int) int {
  if node == nil {
    return 0
  }
  l, r := dfsBtreeGameWinningMove(node.Left, left, right, x),
dfsBtreeGameWinningMove(node.Right, left, right, x)
  if node.Val == x {
    *left, *right = l, r
  }
```

```
    return l + r + 1
}
```

1154. Day of the Year

题目

Given a string `date` representing a Gregorian calendar date formatted as `YYYY-MM-DD`, return the day number of the year.

Example 1:

```
Input: date = "2019-01-09"
Output: 9
Explanation: Given date is the 9th day of the year in 2019.
```

Example 2:

```
Input: date = "2019-02-10"
Output: 41
```

Example 3:

```
Input: date = "2003-03-01"
Output: 60
```

Example 4:

```
Input: date = "2004-03-01"
Output: 61
```

Constraints:

- `date.length == 10`
- `date[4] == date[7] == '-'`, and all other `date[i]`'s are digits
- `date` represents a calendar date between Jan 1st, 1900 and Dec 31, 2019.

题目大意

实现一个 MajorityChecker 的类，它应该具有下述几个 API：

- MajorityChecker(int[] arr) 会用给定的数组 arr 来构造一个 MajorityChecker 的实例。
- int query(int left, int right, int threshold) 有这么几个参数：
  - 0 <= left <= right < arr.length 表示数组 arr 的子数组的长度。
  - 2 * threshold > right - left + 1，也就是说阈值 threshold 始终比子序列长度的一半还要大。

每次查询 query(...) 会返回在 arr[left], arr[left+1], ..., arr[right] 中至少出现阈值次数 threshold 的元素，如果不存在这样的元素，就返回 -1。

提示：

- 1 <= arr.length <= 20000
- 1 <= arr[i] <= 20000
- 对于每次查询，0 <= left <= right < len(arr)
- 对于每次查询，2 * threshold > right - left + 1
- 查询次数最多为 10000

解题思路

- 给出一个时间字符串，求出这一天是这一年当中的第几天。
- 简单题。依照题意处理即可。

代码

```go
package leetcode

import "time"

func dayOfYear(date string) int {
  first := date[:4] + "-01-01"
  firstDay, _ := time.Parse("2006-01-02", first)
  dateDay, _ := time.Parse("2006-01-02", date)
  duration := dateDay.Sub(firstDay)
  return int(duration.Hours())/24 + 1
}
```

1157. Online Majority Element In Subarray

题目

Implementing the class `MajorityChecker`, which has the following API:

- `MajorityChecker(int[] arr)` constructs an instance of MajorityChecker with the given array `arr`;
- `int query(int left, int right, int threshold)` has arguments such that:
  - `0 <= left <= right < arr.length` representing a subarray of `arr`;
  - `2 * threshold > right - left + 1`, ie. the threshold is always a strict majority of the length of the subarray

Each `query(...)` returns the element in `arr[left], arr[left+1], ..., arr[right]` that occurs at least `threshold` times, or `-1` if no such element exists.

Example:

```
MajorityChecker majorityChecker = new MajorityChecker([1,1,2,2,1,1]);
majorityChecker.query(0,5,4); // returns 1
majorityChecker.query(0,3,3); // returns -1
majorityChecker.query(2,3,2); // returns 2
```

Constraints:

- `1 <= arr.length <= 20000`
- `1 <= arr[i] <= 20000`
- For each query, `0 <= left <= right < len(arr)`
- For each query, `2 * threshold > right - left + 1`
- The number of queries is at most `10000`

题目大意

实现一个 MajorityChecker 的类，它应该具有下述几个 API：

- MajorityChecker(int[] arr) 会用给定的数组 arr 来构造一个 MajorityChecker 的实例。
- int query(int left, int right, int threshold) 有这么几个参数：
- 0 <= left <= right < arr.length 表示数组 arr 的子数组的长度。
- 2 * threshold > right - left + 1，也就是说阈值 threshold 始终比子序列长度的一半还要大。

每次查询 query(...) 会返回在 arr[left], arr[left+1], ..., arr[right] 中至少出现阈值次数 threshold 的元素，如果不存在这样的元素，就返回 -1。

提示：

- 1 <= arr.length <= 20000
- 1 <= arr[i] <= 20000
- 对于每次查询，0 <= left <= right < len(arr)
- 对于每次查询，2 * threshold > right - left + 1
- 查询次数最多为 10000

解题思路

- 设计一个数据结构，能在任意的一个区间内，查找是否存在众数，众数的定义是：该数字出现的次数大于区间的一半。如果存在众数，一定唯一。如果在给定的区间内找不到众数，则输出 -1 。
- 这一题有一个很显眼的“暗示”，`2 * threshold > right - left + 1`，这个条件就是摩尔投票算法的前提条件。摩尔投票的思想可以见第 169 题。这一题又要在区间内查询，所以选用线段树这个数据结构来实现。经过分析，可以确定此题的解题思路，摩尔投票 + 线段树。

- 摩尔投票的思想是用两个变量，candidate 和 count，用来记录待被投票投出去的元素，和候选人累积没被投出去的轮数。如果候选人累积没有被投出去的轮数越多，那么最终成为众数的可能越大。从左往右扫描整个数组，先去第一个元素为 candidate，如果遇到相同的元素就累加轮数，如果遇到不同的元素，就把 candidate 和不同的元素一起投出去。当轮数变成 0 了，再选下一个元素作为 candidate。从左扫到右，就能找到众数了。那怎么和线段树结合起来呢？
- 线段树是把一个大的区间拆分成很多个小区间，那么考虑这样一个问题。每个小区间内使用摩尔投票，最终把所有小区间合并起来再用一次摩尔投票，得到的结果和对整个区间使用一次摩尔投票的结果是一样的么？答案是一样的。可以这样想，众数总会在一个区间内被选出来，那么其他区间的摩尔投票都是起“中和”作用的，即两两元素一起出局。这个问题想通以后，说明摩尔投票具有可加的性质。既然满足可加，就可以和线段树结合，因为线段树每个线段就是加起来，最终合并成大区间的。
- 举个例子，arr = [1,1,2,2,1,1]，先构造线段树，如下左图。

  现在每个线段树的节点不是只存一个 int 数字了，而是存 candidate 和 count。每个节点的 candidate 和 count 分别代表的是该区间内摩尔投票的结果。初始化的时候，先把每个叶子都填满，candidate 是自己，count = 1 。即右图绿色节点。然后在 pushUp 的时候，进行摩尔投票：

  ```
  mc.merge = func(i, j segmentItem) segmentItem {
      if i.candidate == j.candidate {
        return segmentItem{candidate: i.candidate, count: i.count + j.count}
      }
      if i.count > j.count {
        return segmentItem{candidate: i.candidate, count: i.count - j.count}
      }
      return segmentItem{candidate: j.candidate, count: j.count - i.count}
    }
  ```

  直到根节点的 candidate 和 count 都填满。**注意，这里的 count 并不是元素出现的总次数，而是摩尔投票中坚持没有被投出去的轮数**。当线段树构建完成以后，就可以开始查询任意区间内的众数了，candidate 即为众数。接下来还要确定众数是否满足 `threshold` 的条件。
- 用一个字典记录每个元素在数组中出现位置的下标，例如上述这个例子，用 map 记录下标：count = map[1:[0 1 4 5] 2:[2 3]]。由于下标在记录过程中是递增的，所以满足二分查找的条件。利用这个字典就可以查出在任意区间内，指定元素出现的次数。例如这里要查找 1 在 [0,5] 区间内出现的个数，那么利用 2 次二分查找，分别找到 `lowerBound` 和 `upperBound`，在 [lowerBound，upperBound) 区间内，都是元素 1 ，那么区间长度即是该元素重复出现的次数，和 `threshold` 比较，如果 ≥ `threshold` 说明找到了答案，否则没有找到就输出 -1 。

代码

```go
package leetcode

import (
  "sort"
)

type segmentItem struct {
```

```go
  candidate int
  count     int
}

// MajorityChecker define
type MajorityChecker struct {
  segmentTree []segmentItem
  data        []int
  merge       func(i, j segmentItem) segmentItem
  count       map[int][]int
}

// Constructor1157 define
func Constructor1157(arr []int) MajorityChecker {
  data, tree, mc, count := make([]int, len(arr)), make([]segmentItem, 4*len(arr)),
MajorityChecker{}, make(map[int][]int)
  // 这个 merge 函数就是摩尔投票算法
  mc.merge = func(i, j segmentItem) segmentItem {
    if i.candidate == j.candidate {
      return segmentItem{candidate: i.candidate, count: i.count + j.count}
    }
    if i.count > j.count {
      return segmentItem{candidate: i.candidate, count: i.count - j.count}
    }
    return segmentItem{candidate: j.candidate, count: j.count - i.count}
  }
  for i := 0; i < len(arr); i++ {
    data[i] = arr[i]
  }
  for i := 0; i < len(arr); i++ {
    if _, ok := count[arr[i]]; !ok {
      count[arr[i]] = []int{}
    }
    count[arr[i]] = append(count[arr[i]], i)
  }
  mc.data, mc.segmentTree, mc.count = data, tree, count
  if len(arr) > 0 {
    mc.buildSegmentTree(0, 0, len(arr)-1)
  }
  return mc
}

func (mc *MajorityChecker) buildSegmentTree(treeIndex, left, right int) {
  if left == right {
    mc.segmentTree[treeIndex] = segmentItem{candidate: mc.data[left], count: 1}
    return
  }
  leftTreeIndex, rightTreeIndex := mc.leftChild(treeIndex), mc.rightChild(treeIndex)
  midTreeIndex := left + (right-left)>>1
```

```go
  mc.buildSegmentTree(leftTreeIndex, left, midTreeIndex)
  mc.buildSegmentTree(rightTreeIndex, midTreeIndex+1, right)
  mc.segmentTree[treeIndex] = mc.merge(mc.segmentTree[leftTreeIndex],
mc.segmentTree[rightTreeIndex])
}

func (mc *MajorityChecker) leftChild(index int) int {
  return 2*index + 1
}

func (mc *MajorityChecker) rightChild(index int) int {
  return 2*index + 2
}

// Query define
func (mc *MajorityChecker) query(left, right int) segmentItem {
  if len(mc.data) > 0 {
    return mc.queryInTree(0, 0, len(mc.data)-1, left, right)
  }
  return segmentItem{candidate: -1, count: -1}
}

func (mc *MajorityChecker) queryInTree(treeIndex, left, right, queryLeft, queryRight
int) segmentItem {
  midTreeIndex, leftTreeIndex, rightTreeIndex := left+(right-left)>>1,
mc.leftChild(treeIndex), mc.rightChild(treeIndex)
  if queryLeft <= left && queryRight >= right { // segment completely inside range
    return mc.segmentTree[treeIndex]
  }
  if queryLeft > midTreeIndex {
    return mc.queryInTree(rightTreeIndex, midTreeIndex+1, right, queryLeft, queryRight)
  } else if queryRight <= midTreeIndex {
    return mc.queryInTree(leftTreeIndex, left, midTreeIndex, queryLeft, queryRight)
  }
  // merge query results
  return mc.merge(mc.queryInTree(leftTreeIndex, left, midTreeIndex, queryLeft,
midTreeIndex),
    mc.queryInTree(rightTreeIndex, midTreeIndex+1, right, midTreeIndex+1, queryRight))
}

// Query define
func (mc *MajorityChecker) Query(left int, right int, threshold int) int {
  res := mc.query(left, right)
  if _, ok := mc.count[res.candidate]; !ok {
    return -1
  }
  start := sort.Search(len(mc.count[res.candidate]), func(i int) bool { return left <=
mc.count[res.candidate][i] })
```

```
  end := sort.Search(len(mc.count[res.candidate]), func(i int) bool { return right <
mc.count[res.candidate][i] }) - 1
  if (end - start + 1) >= threshold {
    return res.candidate
  }
  return -1
}

/**
 * Your MajorityChecker object will be instantiated and called as such:
 * obj := Constructor(arr);
 * param_1 := obj.Query(left,right,threshold);
 */
```

1160. Find Words That Can Be Formed by Characters

题目

You are given an array of strings `words` and a string `chars`.

A string is *good* if it can be formed by characters from `chars` (each character can only be used once).

Return the sum of lengths of all good strings in `words`.

Example 1:

```
Input: words = ["cat","bt","hat","tree"], chars = "atach"
Output: 6
Explanation:
The strings that can be formed are "cat" and "hat" so the answer is 3 + 3 = 6.
```

Example 2:

```
Input: words = ["hello","world","leetcode"], chars = "welldonehoneyr"
Output: 10
Explanation:
The strings that can be formed are "hello" and "world" so the answer is 5 + 5 = 10.
```

Note:

1. `1 <= words.length <= 1000`
2. `1 <= words[i].length, chars.length <= 100`
3. All strings contain lowercase English letters only.

题目大意

给你一份『词汇表』（字符串数组） words 和一张『字母表』（字符串） chars。假如你可以用 chars 中的『字母』（字符）拼写出 words 中的某个『单词』（字符串），那么我们就认为你掌握了这个单词。注意：每次拼写时，chars 中的每个字母都只能用一次。返回词汇表 words 中你掌握的所有单词的 长度之和。

提示：

1. 1 <= words.length <= 1000
2. 1 <= words[i].length, chars.length <= 100
3. 所有字符串中都仅包含小写英文字母

解题思路

- 给出一个字符串数组 `words` 和一个字符串 `chars`，要求输出 `chars` 中能构成 `words` 字符串的字符数总数。
- 简单题。先分别统计 `words` 和 `chars` 里面字符的频次。然后针对 `words` 中每个 `word` 判断能够能由 `chars` 构成，如果能构成，最终结果加上这个 `word` 的长度。

代码

```go
package leetcode

func countCharacters(words []string, chars string) int {
  count, res := make([]int, 26), 0
  for i := 0; i < len(chars); i++ {
    count[chars[i]-'a']++
  }
  for _, w := range words {
    if canBeFormed(w, count) {
      res += len(w)
    }
  }
  return res
}
func canBeFormed(w string, c []int) bool {
  count := make([]int, 26)
  for i := 0; i < len(w); i++ {
    count[w[i]-'a']++
    if count[w[i]-'a'] > c[w[i]-'a'] {
      return false
    }
  }
  return true
}
```

1170. Compare Strings by Frequency of the Smallest Character

题目

Let's define a function `f(s)` over a non-empty string `s`, which calculates the frequency of the smallest character in `s`. For example, if `s = "dcce"` then `f(s) = 2` because the smallest character is `"c"` and its frequency is 2.

Now, given string arrays `queries` and `words`, return an integer array `answer`, where each `answer[i]` is the number of words such that `f(queries[i])` < `f(W)`, where `W` is a word in `words`.

Example 1:

```
Input: queries = ["cbd"], words = ["zaaaz"]
Output: [1]
Explanation: On the first query we have f("cbd") = 1, f("zaaaz") = 3 so f("cbd") <
f("zaaaz").
```

Example 2:

```
Input: queries = ["bbb","cc"], words = ["a","aa","aaa","aaaa"]
Output: [1,2]
Explanation: On the first query only f("bbb") < f("aaaa"). On the second query both
f("aaa") and f("aaaa") are both > f("cc").
```

Constraints:

- `1 <= queries.length <= 2000`
- `1 <= words.length <= 2000`
- `1 <= queries[i].length, words[i].length <= 10`
- `queries[i][j]`, `words[i][j]` are English lowercase letters.

题目大意

我们来定义一个函数 f(s)，其中传入参数 s 是一个非空字符串；该函数的功能是统计 s 中（按字典序比较）最小字母的出现频次。

例如，若 s = "dcce"，那么 f(s) = 2，因为最小的字母是 "c"，它出现了 2 次。

现在，给你两个字符串数组待查表 queries 和词汇表 words，请你返回一个整数数组 answer 作为答案，其中每个 answer[i] 是满足 f(queries[i]) < f(W) 的词的数目，W 是词汇表 words 中的词。

提示：

- 1 <= queries.length <= 2000
- 1 <= words.length <= 2000
- 1 <= queries[i].length, words[i].length <= 10

- queries[i][j], words[i][j] 都是小写英文字母

解题思路

- 给出 2 个数组，`queries` 和 `words`，针对每一个 `queries[i]` 统计在 `words[j]` 中满足 `f(queries[i]) < f(words[j])` 条件的 `words[j]` 的个数。`f(string)` 的定义是 `string` 中字典序最小的字母的频次。
- 先依照题意，构造出 `f()` 函数，算出每个 `words[j]` 的 `f()` 值，然后排序。再依次计算 `queries[i]` 的 `f()` 值。针对每个 `f()` 值，在 `words[j]` 的 `f()` 值中二分搜索，查找比它大的值的下标 `k`，`n-k` 即是比 `queries[i]` 的 `f()` 值大的元素个数。依次输出到结果数组中即可。

代码

```go
package leetcode

import "sort"

func numSmallerByFrequency(queries []string, words []string) []int {
  ws, res := make([]int, len(words)), make([]int, len(queries))
  for i, w := range words {
    ws[i] = countFunc(w)
  }
  sort.Ints(ws)
  for i, q := range queries {
    fq := countFunc(q)
    res[i] = len(words) - sort.Search(len(words), func(i int) bool { return fq < ws[i]
})
  }
  return res
}

func countFunc(s string) int {
  count, i := [26]int{}, 0
  for _, b := range s {
    count[b-'a']++
  }
  for count[i] == 0 {
    i++
  }
  return count[i]
}
```

1171. Remove Zero Sum Consecutive Nodes from Linked List

题目

Given the `head` of a linked list, we repeatedly delete consecutive sequences of nodes that sum to `0` until there are no such sequences.

After doing so, return the head of the final linked list. You may return any such answer.

(Note that in the examples below, all sequences are serializations of `ListNode` objects.)

Example 1:

```
Input: head = [1,2,-3,3,1]
Output: [3,1]
**Note**: The answer [1,2,1] would also be accepted.
```

Example 2:

```
Input: head = [1,2,3,-3,4]
Output: [1,2,4]
```

Example 3:

```
Input: head = [1,2,3,-3,-2]
Output: [1]
```

Constraints:

- The given linked list will contain between `1` and `1000` nodes.
- Each node in the linked list has `-1000 <= node.val <= 1000`.

题目大意

给你一个链表的头节点 head，请你编写代码，反复删去链表中由 总和 值为 0 的连续节点组成的序列，直到不存在这样的序列为止。删除完毕后，请你返回最终结果链表的头节点。你可以返回任何满足题目要求的答案。

（注意，下面示例中的所有序列，都是对 ListNode 对象序列化的表示。）

提示：

- 给你的链表中可能有 1 到 1000 个节点。
- 对于链表中的每个节点，节点的值：-1000 <= node.val <= 1000.

解题思路

- 给出一个链表，要求把链表中和为 0 的结点都移除。
- 由于链表的特性，不能随机访问。所以从链表的头开始往后扫，把累加和存到字典中。当再次出现相同的累加和的时候，代表这中间的一段和是 0，于是要删除这一段。删除这一段的过程中，也要删除这一段在字典中存过的累加和。有一个特殊情况需要处理，即整个链表的总和是 0，那么最终结果是空链表。针对这个特殊情

况，字典中先预存入 0 值。

代码

```
package leetcode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */

// 解法一
func removeZeroSumSublists(head *ListNode) *ListNode {
  // 计算累加和，和作为 key 存在 map 中，value 存那个节点的指针。如果字典中出现了重复的和，代表出现了和为 0 的段。
  sum, sumMap, cur := 0, make(map[int]*ListNode), head
  // 字典中增加 0 这个特殊值，是为了防止最终链表全部消除完
  sumMap[0] = nil
  for cur != nil {
    sum = sum + cur.Val
    if ptr, ok := sumMap[sum]; ok {
      // 在字典中找到了重复的和，代表 [ptr, tmp] 中间的是和为 0 的段，要删除的就是这一段。
      // 同时删除 map 中中间这一段的和
      if ptr != nil {
        iter := ptr.Next
        tmpSum := sum + iter.Val
        for tmpSum != sum {
          // 删除中间为 0 的那一段，tmpSum 不断的累加删除 map 中的和
          delete(sumMap, tmpSum)
          iter = iter.Next
          tmpSum = tmpSum + iter.Val
        }
        ptr.Next = cur.Next
      } else {
        head = cur.Next
        sumMap = make(map[int]*ListNode)
        sumMap[0] = nil
      }
    } else {
      sumMap[sum] = cur
    }
    cur = cur.Next
  }
  return head
}
```

```go
// 解法二 暴力解法
func removeZeroSumSublists1(head *ListNode) *ListNode {
  if head == nil {
    return nil
  }
  h, prefixSumMap, sum, counter, lastNode := head, map[int]int{}, 0, 0, &ListNode{Val: 1010}
  for h != nil {
    for h != nil {
      sum += h.Val
      counter++
      if v, ok := prefixSumMap[sum]; ok {
        lastNode, counter = h, v
        break
      }
      if sum == 0 {
        head = h.Next
        break
      }
      prefixSumMap[sum] = counter
      h = h.Next
    }
    if lastNode.Val != 1010 {
      h = head
      for counter > 1 {
        counter--
        h = h.Next
      }
      h.Next = lastNode.Next
    }
    if h == nil {
      break
    } else {
      h, prefixSumMap, sum, counter, lastNode = head, map[int]int{}, 0, 0, &ListNode{Val: 1010}
    }
  }
  return head
}
```

1175. Prime Arrangements

题目

Return the number of permutations of 1 to `n` so that prime numbers are at prime indices (1-indexed.)

(Recall that an integer is prime if and only if it is greater than 1, and cannot be written as a product of two positive integers both smaller than it.)

Since the answer may be large, return the answer **modulo** `10^9 + 7`.

Example 1:

```
Input: n = 5
Output: 12
Explanation: For example [1,2,5,4,3] is a valid permutation, but [5,2,3,4,1] is not because the prime number 5 is at index 1.
```

Example 2:

```
Input: n = 100
Output: 682289015
```

Constraints:

- `1 <= n <= 100`

题目大意

请你帮忙给从 1 到 n 的数设计排列方案，使得所有的「质数」都应该被放在「质数索引」（索引从 1 开始）上；你需要返回可能的方案总数。让我们一起来回顾一下「质数」：质数一定是大于 1 的，并且不能用两个小于它的正整数的乘积来表示。由于答案可能会很大，所以请你返回答案 模 mod 10^9 + 7 之后的结果即可。

提示：

- 1 <= n <= 100

解题思路

- 给出一个数 n，要求在 1-n 这 n 个数中，素数在素数索引下标位置上的全排列个数。
- 由于这一题的 `n` 小于 100，所以可以用打表法。先把小于 100 个素数都打表打出来。然后对小于 n 的素数进行全排列，即 n！，然后再对剩下来的非素数进行全排列，即 (n-c)！。两个的乘积即为最终答案。

代码

```go
package leetcode

import "sort"

var primes = []int{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97}

func numPrimeArrangements(n int) int {
  primeCount := sort.Search(25, func(i int) bool { return primes[i] > n })
```

```
  return factorial(primeCount) * factorial(n-primeCount) % 1000000007
}

func factorial(n int) int {
  if n == 1 || n == 0 {
    return 1
  }
  return n * factorial(n-1) % 1000000007
}
```

1178. Number of Valid Words for Each Puzzle

题目

With respect to a given `puzzle` string, a `word` is *valid* if both the following conditions are satisfied:

- `word` contains the first letter of `puzzle`.
- For each letter in `word`, that letter is in `puzzle`.For example, if the puzzle is "abcdefg", then valid words are "faced", "cabbage", and "baggage"; while invalid words are "beefed" (doesn't include "a") and "based" (includes "s" which isn't in the puzzle).

Return an array `answer`, where `answer[i]` is the number of words in the given word list `words` that are valid with respect to the puzzle `puzzles[i]`.

Example :

```
Input:
words = ["aaaa","asas","able","ability","actt","actor","access"],
puzzles = ["aboveyz","abrodyz","abslute","absoryz","actresz","gaswxyz"]
Output: [1,1,3,2,4,0]
Explanation:
1 valid word for "aboveyz" : "aaaa"
1 valid word for "abrodyz" : "aaaa"
3 valid words for "abslute" : "aaaa", "asas", "able"
2 valid words for "absoryz" : "aaaa", "asas"
4 valid words for "actresz" : "aaaa", "asas", "actt", "access"
There're no valid words for "gaswxyz" cause none of the words in the list contains
letter 'g'.
```

Constraints:

- `1 <= words.length <= 10^5`
- `4 <= words[i].length <= 50`
- `1 <= puzzles.length <= 10^4`
- `puzzles[i].length == 7`
- `words[i][j]`, `puzzles[i][j]` are English lowercase letters.

- Each `puzzles[i]` doesn't contain repeated characters.

题目大意

外国友人仿照中国字谜设计了一个英文版猜字谜小游戏，请你来猜猜看吧。

字谜的迷面 puzzle 按字符串形式给出，如果一个单词 word 符合下面两个条件，那么它就可以算作谜底：

- 单词 word 中包含谜面 puzzle 的第一个字母。
- 单词 word 中的每一个字母都可以在谜面 puzzle 中找到。
  例如，如果字谜的谜面是 "abcdefg"，那么可以作为谜底的单词有 "faced", "cabbage", 和 "baggage"；而 "beefed"（不含字母 "a"）以及 "based"（其中的 "s" 没有出现在谜面中）都不能作为谜底。

返回一个答案数组 answer，数组中的每个元素 answer[i] 是在给出的单词列表 words 中可以作为字谜迷面 puzzles[i] 所对应的谜底的单词数目。

提示：

- 1 <= words.length <= 10^5
- 4 <= words[i].length <= 50
- 1 <= puzzles.length <= 10^4
- puzzles[i].length == 7
- words[i][j], puzzles[i][j] 都是小写英文字母。
- 每个 puzzles[i] 所包含的字符都不重复。

解题思路

- 首先题目中两个限制条件非常关键：**puzzles[i].length == 7，每个 puzzles[i] 所包含的字符都不重复**。也就是说穷举每个puzzle的子串的搜索空间就是2^7=128，而且不用考虑去重问题。
- 因为谜底的判断只跟字符是否出现有关，跟字符的个数无关，另外都是小写的英文字母，所以可以用 `bitmap` 来表示单词(word)。
- 利用 `map` 记录不同状态的单词(word)的个数。
- 根据题意，如果某个单词(word)是某个字谜(puzzle)的谜底，那么 `word` 的 `bitmap` 肯定对应于 `puzzle` 某个子串的 `bitmap` 表示，且 `bitmap` 中包含 `puzzle` 的第一个字母的 `bit` 占用。
- 问题就转换为：求每一个 `puzzle` 的每一个子串，然后求和这个子串具有相同 `bitmap` 表示且 `word` 中包含 `puzzle` 的第一个字母的 `word` 的个数。

代码

```go
package leetcode

/*
  匹配跟单词中的字母顺序，字母个数都无关，可以用bitmap压缩
  1. 记录word中 利用map记录各种bit标示的个数
  2. puzzles 中各个字母都不相同！ 记录bitmap，然后搜索子空间中各种bit标识的个数的和
     因为puzzles长度最长是7，所以搜索空间 2^7
*/
func findNumOfValidWords(words []string, puzzles []string) []int {
  wordBitStatusMap, res := make(map[uint32]int, 0), []int{}
```

```go
  for _, w := range words {
    wordBitStatusMap[toBitMap([]byte(w))]++
  }
  for _, p := range puzzles {
    var bitMap uint32
    var totalNum int
    bitMap |= (1 << (p[0] - 'a')) //work中要包含 p 的第一个字母 所以这个bit位上必须是1
    findNum([]byte(p)[1:], bitMap, &totalNum, wordBitStatusMap)
    res = append(res, totalNum)
  }
  return res
}

func toBitMap(word []byte) uint32 {
  var res uint32
  for _, b := range word {
    res |= (1 << (b - 'a'))
  }
  return res
}

//利用dfs 搜索 pussles的子空间
func findNum(puzzles []byte, bitMap uint32, totalNum *int, m map[uint32]int) {
  if len(puzzles) == 0 {
    *totalNum = *totalNum + m[bitMap]
    return
  }
  //不包含puzzles[0],即puzzles[0]对应bit是0
  findNum(puzzles[1:], bitMap, totalNum, m)
  //包含puzzles[0],即puzzles[0]对应bit是1
  bitMap |= (1 << (puzzles[0] - 'a'))
  findNum(puzzles[1:], bitMap, totalNum, m)
  bitMap ^= (1 << (puzzles[0] - 'a')) //异或 清零
  return
}
```

1184. Distance Between Bus Stops

题目

A bus has `n` stops numbered from `0` to `n - 1` that form a circle. We know the distance between all pairs of neighboring stops where `distance[i]` is the distance between the stops number `i` and `(i + 1) % n`.

The bus goes along both directions i.e. clockwise and counterclockwise.

Return the shortest distance between the given `start` and `destination` stops.

Example 1:

```
Input: distance = [1,2,3,4], start = 0, destination = 1
Output: 1
Explanation: Distance between 0 and 1 is 1 or 9, minimum is 1.
```

Example 2:

```
Input: distance = [1,2,3,4], start = 0, destination = 2
Output: 3
Explanation: Distance between 0 and 2 is 3 or 7, minimum is 3.
```

Example 3:

```
Input: distance = [1,2,3,4], start = 0, destination = 3
Output: 4
Explanation: Distance between 0 and 3 is 6 or 4, minimum is 4.
```

Constraints:

- `1 <= n <= 10^4`
- `distance.length == n`
- `0 <= start, destination < n`
- `0 <= distance[i] <= 10^4`

题目大意

环形公交路线上有 n 个站，按次序从 0 到 n - 1 进行编号。我们已知每一对相邻公交站之间的距离，distance[i] 表示编号为 i 的车站和编号为 (i + 1) % n 的车站之间的距离。环线上的公交车都可以按顺时针和逆时针的方向行驶。返回乘客从出发点 start 到目的地 destination 之间的最短距离。

提示：

- 1 <= n <= 10^4
- distance.length == n
- 0 <= start, destination < n
- 0 <= distance[i] <= 10^4

解题思路

- 给出一个数组，代表的是公交车站每站直接的距离。距离是按照数组下标的顺序给出的，公交车可以按照顺时针行驶，也可以按照逆时针行驶。问行驶的最短距离是多少。
- 按照题意，分别算出顺时针和逆时针的行驶距离，比较两者距离，取出小值就是结果。

代码

```
package leetcode
```

```go
func distanceBetweenBusStops(distance []int, start int, destination int) int {
  clockwiseDis, counterclockwiseDis, n := 0, 0, len(distance)
  for i := start; i != destination; i = (i + 1) % n {
    clockwiseDis += distance[i]
  }
  for i := destination; i != start; i = (i + 1) % n {
    counterclockwiseDis += distance[i]
  }
  if clockwiseDis < counterclockwiseDis {
    return clockwiseDis
  }
  return counterclockwiseDis
}
```

1185. Day of the Week

题目

Given a date, return the corresponding day of the week for that date.

The input is given as three integers representing the `day`, `month` and `year` respectively.

Return the answer as one of the following values `{"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"}`.

Example 1:

```
Input: day = 31, month = 8, year = 2019
Output: "Saturday"
```

Example 2:

```
Input: day = 18, month = 7, year = 1999
Output: "Sunday"
```

Example 3:

```
Input: day = 15, month = 8, year = 1993
Output: "Sunday"
```

Constraints:

- The given dates are valid dates between the years `1971` and `2100`.

题目大意

给你一个日期，请你设计一个算法来判断它是对应一周中的哪一天。输入为三个整数：day、month 和 year，分别表示日、月、年。

您返回的结果必须是这几个值中的一个 {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"}。

提示：

- 给出的日期一定是在 1971 到 2100 年之间的有效日期。

解题思路

- 给出一个日期，要求算出这一天是星期几。
- 简单题，按照常识计算即可。

代码

```
package leetcode

import "time"

func dayOfTheWeek(day int, month int, year int) string {
  return time.Date(year, time.Month(month), day, 0, 0, 0, 0,
time.Local).Weekday().String()
}
```

1189. Maximum Number of Balloons

题目

Given a string `text`, you want to use the characters of `text` to form as many instances of the word **"balloon"** as possible.

You can use each character in `text` **at most once**. Return the maximum number of instances that can be formed.

Example 1:

```
Input: text = "nlaebolko"
Output: 1
```

Example 2:

```
Input: text = "loonbalxballpoon"
Output: 2
```

Example 3:

```
Input: text = "leetcode"
Output: 0
```

Constraints:

- `1 <= text.length <= 10^4`
- `text` consists of lower case English letters only.

题目大意

给你一个字符串 text，你需要使用 text 中的字母来拼凑尽可能多的单词 "balloon"（气球）。字符串 text 中的每个字母最多只能被使用一次。请你返回最多可以拼凑出多少个单词 "balloon"。

提示：

- 1 <= text.length <= 10^4
- text 全部由小写英文字母组成

解题思路

- 给出一个字符串，问这个字符串里面的数组能组成多少个 **balloon** 这个单词。
- 简单题，先统计 26 个字母每个字母的频次，然后取出 balloon 这 5 个字母出现频次最小的值就是结果。

代码

```go
package leetcode

func maxNumberOfBalloons(text string) int {
  fre := make([]int, 26)
  for _, t := range text {
    fre[t-'a']++
  }
  // 字符 b 的频次是数组下标 1 对应的元素值
  // 字符 a 的频次是数组下标 0 对应的元素值
  // 字符 l 的频次是数组下标 11 对应的元素值，这里有 2 个 l，所以元素值需要除以 2
  // 字符 o 的频次是数组下标 14 对应的元素值，这里有 2 个 o，所以元素值需要除以 2
  // 字符 n 的频次是数组下标 13 对应的元素值
  return min(fre[1], min(fre[0], min(fre[11]/2, min(fre[14]/2, fre[13]))))
}
```

1190. Reverse Substrings Between Each Pair of Parentheses

题目

You are given a string `s` that consists of lower case English letters and brackets.

Reverse the strings in each pair of matching parentheses, starting from the innermost one.

Your result should **not** contain any brackets.

Example 1:

```
Input: s = "(abcd)"
Output: "dcba"
```

Example 2:

```
Input: s = "(u(love)i)"
Output: "iloveu"
Explanation: The substring "love" is reversed first, then the whole string is reversed.
```

Example 3:

```
Input: s = "(ed(et(oc))el)"
Output: "leetcode"
Explanation: First, we reverse the substring "oc", then "etco", and finally, the whole
string.
```

Example 4:

```
Input: s = "a(bcdefghijkl(mno)p)q"
Output: "apmnolkjihgfedcbq"
```

Constraints:

- `0 <= s.length <= 2000`
- `s` only contains lower case English characters and parentheses.
- It's guaranteed that all parentheses are balanced.

题目大意

给出一个字符串 s（仅含有小写英文字母和括号）。请你按照从括号内到外的顺序，逐层反转每对匹配括号中的字符串，并返回最终的结果。注意，您的结果中 不应 包含任何括号。

解题思路

- 本题最容易想到的思路是利用栈将每对括号里面的字符串入栈，当遇到 ")" 括号时出栈并逆序。由于用到了栈的数据结构，多层括号嵌套的问题也不用担心。这种边入栈出栈，逆序字符串的方法，时间复杂度是 O(n^2)，有没有可能进一步降低时间复杂度呢？
- 上述解法中，存在重复遍历的情况。扫描原字符串的时候，入栈出栈已经扫描了一次，在 ")" 括号出栈时，逆

序又会扫一遍已经入栈的字符串。这部分重复遍历的过程可以优化掉。第一次循环先标记出逆序区间。例如遇到 "(" 的时候，入栈并记录下它的下标，当遇到 ")" 的时候，意味着这一对括号匹配上了，所以将 ")" 的下标和之前入栈 "(" 的下标交换。此次遍历将逆序区间标记出来了。再遍历一次，根据逆序区间逆序字符串。不在逆序区间的字符串正常 append。如果在逆序区间内的，逆序遍历，添加到最终结果字符串中。这样做，时间复杂度仅为 O(n)。具体实现见下面代码。

代码

```
package leetcode

func reverseParentheses(s string) string {
  pair, stack := make([]int, len(s)), []int{}
  for i, b := range s {
    if b == '(' {
      stack = append(stack, i)
    } else if b == ')' {
      j := stack[len(stack)-1]
      stack = stack[:len(stack)-1]
      pair[i], pair[j] = j, i
    }
  }
  res := []byte{}
  for i, step := 0, 1; i < len(s); i += step {
    if s[i] == '(' || s[i] == ')' {
      i = pair[i]
      step = -step
    } else {
      res = append(res, s[i])
    }
  }
  return string(res)
}
```

1200. Minimum Absolute Difference

题目

Given an array of **distinct** integers `arr`, find all pairs of elements with the minimum absolute difference of any two elements.

Return a list of pairs in ascending order(with respect to pairs), each pair `[a, b]` follows

- `a, b` are from `arr`
- `a < b`
- `b - a` equals to the minimum absolute difference of any two elements in `arr`

Example 1:

```
Input: arr = [4,2,1,3]
Output: [[1,2],[2,3],[3,4]]
Explanation: The minimum absolute difference is 1. List all pairs with difference equal to 1 in ascending order.
```

Example 2:

```
Input: arr = [1,3,6,10,15]
Output: [[1,3]]
```

Example 3:

```
Input: arr = [3,8,-10,23,19,-4,-14,27]
Output: [[-14,-10],[19,23],[23,27]]
```

Constraints:

- `2 <= arr.length <= 10^5`
- `-10^6 <= arr[i] <= 10^6`

题目大意

给出一个数组，要求找出所有满足条件的数值对 [a,b]： `a<b` 并且 `b-a` 的差值是数组中所有两个元素差值的最小值。

解题思路

- 给出一个数组，要求找出所有满足条件的数值对 [a,b]： `a<b` 并且 `b-a` 的差值是数组中所有两个元素差值的最小值。
- 简单题，按照题意先排序，然后依次求出两个相邻元素的差值，求出最小的差值。最后遍历一遍数组，把所有等于最小差值的数值对都输出。

代码

```go
package leetcode

import (
  "math"
  "sort"
)

func minimumAbsDifference(arr []int) [][]int {
  minDiff, res := math.MaxInt32, [][]int{}
  sort.Ints(arr)
  for i := 1; i < len(arr); i++ {
    if arr[i]-arr[i-1] < minDiff {
```

```
      minDiff = arr[i] - arr[i-1]
    }
    if minDiff == 1 {
      break
    }
  }
  for i := 1; i < len(arr); i++ {
    if arr[i]-arr[i-1] == minDiff {
      res = append(res, []int{arr[i-1], arr[i]})
    }
  }
  return res
}
```

1201. Ugly Number III

题目

Write a program to find the `n`-th ugly number.

Ugly numbers are **positive integers** which are divisible by `a` **or** `b` **or** `c`.

Example 1:

```
Input: n = 3, a = 2, b = 3, c = 5
Output: 4
Explanation: The ugly numbers are 2, 3, 4, 5, 6, 8, 9, 10... The 3rd is 4.
```

Example 2:

```
Input: n = 4, a = 2, b = 3, c = 4
Output: 6
Explanation: The ugly numbers are 2, 3, 4, 6, 8, 9, 10, 12... The 4th is 6.
```

Example 3:

```
Input: n = 5, a = 2, b = 11, c = 13
Output: 10
Explanation: The ugly numbers are 2, 4, 6, 8, 10, 11, 12, 13... The 5th is 10.
```

Example 4:

```
Input: n = 1000000000, a = 2, b = 217983653, c = 336916467
Output: 1999999984
```

Constraints:

- `1 <= n, a, b, c <= 10^9`
- `1 <= a * b * c <= 10^18`
- It's guaranteed that the result will be in range `[1, 2 * 10^9]`

题目大意

请你帮忙设计一个程序，用来找出第 n 个丑数。丑数是可以被 a 或 b 或 c 整除的 正整数。

提示：

- 1 <= n, a, b, c <= 10^9
- 1 <= a * b * c <= 10^18
- 本题结果在 [1, 2 * 10^9] 的范围内

解题思路

- 给出 4 个数字，a，b，c，n。要求输出可以整除 a 或者整除 b 或者整除 c 的第 n 个数。
- 这一题限定了解的范围，`[1, 2 * 10^9]`，所以直接二分搜索来求解。逐步二分逼近 low 值，直到找到能满足条件的 low 的最小值，即为最终答案。
- 这一题的关键在如何判断一个数是第几个数。一个数能整除 a，能整除 b，能整除 c，那么它应该是第 `num/a + num/b + num/c - num/ab - num/bc - num/ac + num/abc` 个数。这个就是韦恩图。需要注意的是，求 `ab`、`bc`、`ac`、`abc` 的时候需要再除以各自的最大公约数 `gcd()`。

代码

```go
package leetcode

func nthUglyNumber(n int, a int, b int, c int) int {
  low, high := int64(0), int64(2*1e9)
  for low < high {
    mid := low + (high-low)>>1
    if calNthCount(mid, int64(a), int64(b), int64(c)) < int64(n) {
      low = mid + 1
    } else {
      high = mid
    }
  }
  return int(low)
}

func calNthCount(num, a, b, c int64) int64 {
  ab, bc, ac := a*b/gcd(a, b), b*c/gcd(b, c), a*c/gcd(a, c)
  abc := a * bc / gcd(a, bc)
  return num/a + num/b + num/c - num/ab - num/bc - num/ac + num/abc
}

func gcd(a, b int64) int64 {
```

```
  for b != 0 {
    a, b = b, a%b
  }
  return a
}
```

1202. Smallest String With Swaps

题目

You are given a string `s`, and an array of pairs of indices in the string `pairs` where `pairs[i] = [a, b]` indicates 2 indices(0-indexed) of the string.

You can swap the characters at any pair of indices in the given `pairs` **any number of times**.

Return the lexicographically smallest string that `s` can be changed to after using the swaps.

Example 1:

```
Input: s = "dcab", pairs = [[0,3],[1,2]]
Output: "bacd"
Explaination:
Swap s[0] and s[3], s = "bcad"
Swap s[1] and s[2], s = "bacd"
```

Example 2:

```
Input: s = "dcab", pairs = [[0,3],[1,2],[0,2]]
Output: "abcd"
Explaination:
Swap s[0] and s[3], s = "bcad"
Swap s[0] and s[2], s = "acbd"
Swap s[1] and s[2], s = "abcd"
```

Example 3:

```
Input: s = "cba", pairs = [[0,1],[1,2]]
Output: "abc"
Explaination:
Swap s[0] and s[1], s = "bca"
Swap s[1] and s[2], s = "bac"
Swap s[0] and s[1], s = "abc"
```

Constraints:

- `1 <= s.length <= 10^5`
- `0 <= pairs.length <= 10^5`

- `0 <= pairs[i][0], pairs[i][1] < s.length`
- `s` only contains lower case English letters.

题目大意

给你一个字符串 s，以及该字符串中的一些「索引对」数组 pairs，其中 pairs[i] = [a, b] 表示字符串中的两个索引（编号从 0 开始）。你可以 任意多次交换 在 pairs 中任意一对索引处的字符。返回在经过若干次交换后，s 可以变成的按字典序最小的字符串。

提示：

- 1 <= s.length <= 10^5
- 0 <= pairs.length <= 10^5
- 0 <= pairs[i][0], pairs[i][1] < s.length
- s 中只含有小写英文字母

解题思路

- 给出一个字符串和一个字符串里可交换的下标。要求交换以后字典序最小的字符串。
- 这一题可以用并查集来解题，先把可交换下标都 `Union()` 起来，每个集合内，按照字典序从小到大排列。最后扫描原有字符串，从左到右依次找到各自对应的集合里面最小的字符进行替换，每次替换完以后，删除集合中该字符(防止下次重复替换)。最终得到的字符就是最小字典序的字符。

代码

```go
package leetcode

import (
  "sort"

  "github.com/halfrost/LeetCode-Go/template"
)

func smallestStringWithSwaps(s string, pairs [][]int) string {
  uf, res, sMap := template.UnionFind{}, []byte(s), map[int][]byte{}
  uf.Init(len(s))
  for _, pair := range pairs {
    uf.Union(pair[0], pair[1])
  }
  for i := 0; i < len(s); i++ {
    r := uf.Find(i)
    sMap[r] = append(sMap[r], s[i])
  }
  for _, v := range sMap {
    sort.Slice(v, func(i, j int) bool {
      return v[i] < v[j]
```

```
    })
  }
  for i := 0; i < len(s); i++ {
    r := uf.Find(i)
    bytes := sMap[r]
    res[i] = bytes[0]
    sMap[r] = bytes[1:len(bytes)]
  }
  return string(res)
}
```

1203. Sort Items by Groups Respecting Dependencies

题目

There are `n` items each belonging to zero or one of `m` groups where `group[i]` is the group that the `i`-th item belongs to and it's equal to `-1` if the `i`-th item belongs to no group. The items and the groups are zero indexed. A group can have no item belonging to it.

Return a sorted list of the items such that:

- The items that belong to the same group are next to each other in the sorted list.
- There are some relations between these items where `beforeItems[i]` is a list containing all the items that should come before the `i` th item in the sorted array (to the left of the `i` th item).

Return any solution if there is more than one solution and return an **empty list** if there is no solution.

Example 1:

```
Input: n = 8, m = 2, group = [-1,-1,1,0,0,1,0,-1], beforeItems = [[],[6],[5],[6],[3,6],
[],[],[]]
Output: [6,3,4,1,5,2,0,7]
```

Example 2:

```
Input: n = 8, m = 2, group = [-1,-1,1,0,0,1,0,-1], beforeItems = [[],[6],[5],[6],[3],
[],[4],[]]
Output: []
Explanation: This is the same as example 1 except that 4 needs to be before 6 in the
sorted list.
```

Constraints:

- `1 <= m <= n <= 3 * 104`
- `group.length == beforeItems.length == n`
- `1 <= group[i] <= m - 1`
- `0 <= beforeItems[i].length <= n - 1`
- `0 <= beforeItems[i][j] <= n - 1`
- `i != beforeItems[i][j]`
- `beforeItems[i]` does not contain duplicates elements.

题目大意

有 n 个项目，每个项目或者不属于任何小组，或者属于 m 个小组之一。group[i] 表示第 i 个小组所属的小组，如果第 i 个项目不属于任何小组，则 group[i] 等于 -1。项目和小组都是从零开始编号的。可能存在小组不负责任何项目，即没有任何项目属于这个小组。

请你帮忙按要求安排这些项目的进度，并返回排序后的项目列表：

- 同一小组的项目，排序后在列表中彼此相邻。
- 项目之间存在一定的依赖关系，我们用一个列表 beforeItems 来表示，其中 beforeItems[i] 表示在进行第 i 个项目前（位于第 i 个项目左侧）应该完成的所有项目。

如果存在多个解决方案，只需要返回其中任意一个即可。如果没有合适的解决方案，就请返回一个 空列表 。

解题思路

- 读完题能确定这一题是拓扑排序。但是和单纯的拓扑排序有区别的是，同一小组内的项目需要彼此相邻。用 2 次拓扑排序即可解决。第一次拓扑排序排出组间的顺序，第二次拓扑排序排出组内的顺序。为了实现方便，用 map 给虚拟分组标记编号。如下图，将 3，4，6 三个任务打包到 0 号分组里面，将 2，5 两个任务打包到 1 号分组里面，其他任务单独各自为一组。组间的依赖是 6 号任务依赖 1 号任务。由于 6 号任务封装在 0 号分组里，所以 3 号分组依赖 0 号分组。先组间排序，确定分组顺序，再组内拓扑排序，排出最终顺序。

- 上面的解法可以 AC，但是时间太慢了。因为做了一些不必要的操作。有没有可能只用一次拓扑排序呢？将必须要在一起的结点统一依赖一个虚拟结点，例如下图中的虚拟结点 8 和 9 。3，4，6 都依赖 8 号任务，2 和 5 都依赖 9 号任务。1 号任务本来依赖 6 号任务，由于 6 由依赖 8 ，所以添加 1 依赖 8 的边。通过增加虚拟结点，增加了需要打包在一起结点的入度。构建出以上关系以后，按照入度为 0 的原则，依次进行 DFS。8 号和 9 号两个虚拟结点的入度都为 0 ，对它们进行 DFS，必定会使得与它关联的节点都被安排在一起，这样就满足了题意：同一小组的项目，排序后在列表中彼此相邻。一遍扫完，满足题意的顺序就排出来了。这个解法 beat 100%！

代码

```go
package leetcode

// 解法一 拓扑排序版的 DFS
func sortItems(n int, m int, group []int, beforeItems [][]int) []int {
  groups, inDegrees := make([][]int, n+m), make([]int, n+m)
  for i, g := range group {
    if g > -1 {
      g += n
```

```go
      groups[g] = append(groups[g], i)
      inDegrees[i]++
    }
  }
  for i, ancestors := range beforeItems {
    gi := group[i]
    if gi == -1 {
      gi = i
    } else {
      gi += n
    }
    for _, ancestor := range ancestors {
      ga := group[ancestor]
      if ga == -1 {
        ga = ancestor
      } else {
        ga += n
      }
      if gi == ga {
        groups[ancestor] = append(groups[ancestor], i)
        inDegrees[i]++
      } else {
        groups[ga] = append(groups[ga], gi)
        inDegrees[gi]++
      }
    }
  }
  res := []int{}
  for i, d := range inDegrees {
    if d == 0 {
      sortItemsDFS(i, n, &res, &inDegrees, &groups)
    }
  }
  if len(res) != n {
    return nil
  }
  return res
}

func sortItemsDFS(i, n int, res, inDegrees *[]int, groups *[][]int) {
  if i < n {
    *res = append(*res, i)
  }
  (*inDegrees)[i] = -1
  for _, ch := range (*groups)[i] {
    if (*inDegrees)[ch]--; (*inDegrees)[ch] == 0 {
      sortItemsDFS(ch, n, res, inDegrees, groups)
    }
  }
```

```go
}

// 解法二 二维拓扑排序 时间复杂度 O(m+n)，空间复杂度 O(m+n)
func sortItems1(n int, m int, group []int, beforeItems [][]int) []int {
  groupItems, res := map[int][]int{}, []int{}
  for i := 0; i < len(group); i++ {
    if group[i] == -1 {
      group[i] = m + i
    }
    groupItems[group[i]] = append(groupItems[group[i]], i)
  }
  groupGraph, groupDegree, itemGraph, itemDegree := make([][]int, m+n), make([]int,
m+n), make([][]int, n), make([]int, n)
  for i := 0; i < len(beforeItems); i++ {
    for j := 0; j < len(beforeItems[i]); j++ {
      if group[beforeItems[i][j]] != group[i] {
        // 不同组项目，确定组间依赖关系
        groupGraph[group[beforeItems[i][j]]] = append(groupGraph[group[beforeItems[i]
[j]]], group[i])
        groupDegree[group[i]]++
      } else {
        // 同组项目，确定组内依赖关系
        itemGraph[beforeItems[i][j]] = append(itemGraph[beforeItems[i][j]], i)
        itemDegree[i]++
      }
    }
  }
  items := []int{}
  for i := 0; i < m+n; i++ {
    items = append(items, i)
  }
  // 组间拓扑
  groupOrders := topSort(groupGraph, groupDegree, items)
  if len(groupOrders) < len(items) {
    return nil
  }
  for i := 0; i < len(groupOrders); i++ {
    items := groupItems[groupOrders[i]]
    // 组内拓扑
    orders := topSort(itemGraph, itemDegree, items)
    if len(orders) < len(items) {
      return nil
    }
    res = append(res, orders...)
  }
  return res
}

func topSort(graph [][]int, deg, items []int) (orders []int) {
```

```
  q := []int{}
  for _, i := range items {
    if deg[i] == 0 {
      q = append(q, i)
    }
  }
  for len(q) > 0 {
    from := q[0]
    q = q[1:]
    orders = append(orders, from)
    for _, to := range graph[from] {
      deg[to]--
      if deg[to] == 0 {
        q = append(q, to)
      }
    }
  }
  return
}
```

1207. Unique Number of Occurrences

题目

Given an array of integers `arr`, write a function that returns `true` if and only if the number of occurrences of each value in the array is unique.

Example 1:

```
Input: arr = [1,2,2,1,1,3]
Output: true
Explanation: The value 1 has 3 occurrences, 2 has 2 and 3 has 1. No two values have the same number of occurrences.
```

Example 2:

```
Input: arr = [1,2]
Output: false
```

Example 3:

```
Input: arr = [-3,0,1,-3,1,1,1,-3,10,0]
Output: true
```

Constraints:

- `1 <= arr.length <= 1000`

- `-1000 <= arr[i] <= 1000`

题目大意

给你一个整数数组 arr，请你帮忙统计数组中每个数的出现次数。如果每个数的出现次数都是独一无二的，就返回 true；否则返回 false。

提示：

- 1 <= arr.length <= 1000
- -1000 <= arr[i] <= 1000

解题思路

- 给出一个数组，先统计每个数字出现的频次，判断在这个数组中是否存在相同的频次。
- 简单题，先统计数组中每个数字的频次，然后用一个 map 判断频次是否重复。

代码

```go
package leetcode

func uniqueOccurrences(arr []int) bool {
  freq, m := map[int]int{}, map[int]bool{}
  for _, v := range arr {
    freq[v]++
  }
  for _, v := range freq {
    if _, ok := m[v]; !ok {
      m[v] = true
    } else {
      return false
    }
  }
  return true
}
```

1208. Get Equal Substrings Within Budget

题目

You are given two strings `s` and `t` of the same length. You want to change `s` to `t`. Changing the `i`-th character of `s` to `i`-th character of `t` costs `|s[i] - t[i]|` that is, the absolute difference between the ASCII values of the characters.

You are also given an integer `maxCost`.

Return the maximum length of a substring of `s` that can be changed to be the same as the corresponding substring of `t` with a cost less than or equal to `maxCost`.

If there is no substring from `s` that can be changed to its corresponding substring from `t`, return `0`.

Example 1:

```
Input: s = "abcd", t = "bcdf", maxCost = 3
Output: 3
Explanation: "abc" of s can change to "bcd". That costs 3, so the maximum length is 3.
```

Example 2:

```
Input: s = "abcd", t = "cdef", maxCost = 3
Output: 1
Explanation: Each character in s costs 2 to change to charactor in t, so the maximum
length is 1.
```

Example 3:

```
Input: s = "abcd", t = "acde", maxCost = 0
Output: 1
Explanation: You can't make any change, so the maximum length is 1.
```

Constraints:

- `1 <= s.length, t.length <= 10^5`
- `0 <= maxCost <= 10^6`
- `s` and `t` only contain lower case English letters.

题目大意

给你两个长度相同的字符串，s 和 t。将 s 中的第 i 个字符变到 t 中的第 i 个字符需要 |s[i] - t[i]| 的开销（开销可能为 0），也就是两个字符的 ASCII 码值的差的绝对值。

用于变更字符串的最大预算是 maxCost。在转化字符串时，总开销应当小于等于该预算，这也意味着字符串的转化可能是不完全的。如果你可以将 s 的子字符串转化为它在 t 中对应的子字符串，则返回可以转化的最大长度。如果 s 中没有子字符串可以转化成 t 中对应的子字符串，则返回 0。

提示：

- 1 <= s.length, t.length <= 10^5
- 0 <= maxCost <= 10^6
- s 和 t 都只含小写英文字母。

解题思路

- 给出 2 个字符串 `s` 和 `t` 和一个“预算”，要求把“预算”尽可能的花完，`s` 中最多连续有几个字母能变成 `t` 中的字母。“预算”的定义是：|s[i] - t[i]| 。

- 这一题是滑动窗口的题目，滑动窗口右边界每移动一格，就减少一定的预算，直到预算不能减少，再移动滑动窗口的左边界，这个时候注意要把预算还原回去。当整个窗口把字符 `s` 或 `t` 都滑动完了的时候，取出滑动过程中窗口的最大值即为结果。

代码

```
package leetcode

func equalSubstring(s string, t string, maxCost int) int {
  left, right, res := 0, -1, 0
  for left < len(s) {
    if right+1 < len(s) && maxCost-abs(int(s[right+1]-'a')-int(t[right+1]-'a')) >= 0 {
      right++
      maxCost -= abs(int(s[right]-'a') - int(t[right]-'a'))
    } else {
      res = max(res, right-left+1)
      maxCost += abs(int(s[left]-'a') - int(t[left]-'a'))
      left++
    }
  }
  return res
}
```

1209. Remove All Adjacent Duplicates in String II

题目

Given a string `s`, a *k duplicate removal* consists of choosing `k` adjacent and equal letters from `s` and removing them causing the left and the right side of the deleted substring to concatenate together.

We repeatedly make `k` duplicate removals on `s` until we no longer can.

Return the final string after all such duplicate removals have been made.

It is guaranteed that the answer is unique.

Example 1:

```
Input: s = "abcd", k = 2
Output: "abcd"
Explanation:There's nothing to delete.
```

Example 2:

```
Input: s = "deeedbbcccbdaa", k = 3
Output: "aa"
Explanation:
First delete "eee" and "ccc", get "ddbbbdaa"
Then delete "bbb", get "dddaa"
Finally delete "ddd", get "aa"
```

Example 3:

```
Input: s = "pbbcggttciiippooaais", k = 2
Output: "ps"
```

Constraints:

- `1 <= s.length <= 10^5`
- `2 <= k <= 10^4`
- `s` only contains lower case English letters.

题目大意

给你一个字符串 s，「k 倍重复项删除操作」将会从 s 中选择 k 个相邻且相等的字母，并删除它们，使被删去的字符串的左侧和右侧连在一起。你需要对 s 重复进行无限次这样的删除操作，直到无法继续为止。在执行完所有删除操作后，返回最终得到的字符串。本题答案保证唯一。

解题思路

- 暴力解法。每增加一个字符，就往前扫描 `k` 位，判断是否存在 `k` 个连续相同的字符。消除了 `k` 个相同字符后，重新组成的字符串还可能再次产出 `k` 位相同的字符，（类似消消乐，`k` 个相同的字符碰到一起就“消除”），还需要继续消除。最差情况要再次扫描一次字符串。时间复杂度 O(n^2)，空间复杂度 O(n)。
- 暴力解法的低效在于重复统计字符频次，如果每个字符的频次统计一次就好了。按照这个思路，利用 stack，每个栈元素存 2 个值，一个是字符，一个是该字符对应的频次。有了栈顶字符频次信息，就不需要重复往前扫描了。只要栈顶字符频次到达了 `k`，就弹出该字符。如此反复，最终剩下的字符串为所求。时间复杂度 O(n)，空间复杂度 O(n)。

代码

```go
package leetcode

// 解法一 stack
func removeDuplicates(s string, k int) string {
  stack, arr := [][2]int{}, []byte{}
  for _, c := range s {
    i := int(c - 'a')
    if len(stack) > 0 && stack[len(stack)-1][0] == i {
      stack[len(stack)-1][1]++
      if stack[len(stack)-1][1] == k {
        stack = stack[:len(stack)-1]
```

```go
        }
      } else {
        stack = append(stack, [2]int{i, 1})
      }
    }
    for _, pair := range stack {
      c := byte(pair[0] + 'a')
      for i := 0; i < pair[1]; i++ {
        arr = append(arr, c)
      }
    }
    return string(arr)
}

// 解法二 暴力
func removeDuplicates1(s string, k int) string {
    arr, count, tmp := []rune{}, 0, '#'
    for _, v := range s {
      arr = append(arr, v)
      for len(arr) > 0 {
        count = 0
        tmp = arr[len(arr)-1]
        for i := len(arr) - 1; i >= 0; i-- {
          if arr[i] != tmp {
            break
          }
          count++
        }
        if count == k {
          arr = arr[:len(arr)-k]
        } else {
          break
        }
      }
    }
    return string(arr)
}
```

1217. Minimum Cost to Move Chips to The Same Position

题目

There are some chips, and the i-th chip is at position `chips[i]`.

You can perform any of the two following types of moves **any number of times** (possibly zero) **on any chip**:

- Move the `i`-th chip by 2 units to the left or to the right with a cost of **0**.
- Move the `i`-th chip by 1 unit to the left or to the right with a cost of **1**.

There can be two or more chips at the same position initially.

Return the minimum cost needed to move all the chips to the same position (any position).

Example 1:

```
Input: chips = [1,2,3]
Output: 1
Explanation: Second chip will be moved to positon 3 with cost 1. First chip will be
moved to position 3 with cost 0. Total cost is 1.
```

Example 2:

```
Input: chips = [2,2,2,3,3]
Output: 2
Explanation: Both fourth and fifth chip will be moved to position two with cost 1.
Total minimum cost will be 2.
```

Constraints:

- `1 <= chips.length <= 100`
- `1 <= chips[i] <= 10^9`

题目大意

数轴上放置了一些筹码，每个筹码的位置存在数组 chips 当中。你可以对 任何筹码 执行下面两种操作之一（不限操作次数，0 次也可以）：

- 将第 i 个筹码向左或者右移动 2 个单位，代价为 0。
- 将第 i 个筹码向左或者右移动 1 个单位，代价为 1。

最开始的时候，同一位置上也可能放着两个或者更多的筹码。返回将所有筹码移动到同一位置（任意位置）上所需要的最小代价。

提示：

- 1 <= chips.length <= 100
- 1 <= chips[i] <= 10^9

解题思路

- 给出一个数组，数组的下标代表的是数轴上的坐标点，数组的元素代表的是砝码大小。砝码移动规则，左右移动 2 格，没有代价，左右移动 1 个，代价是 1 。问最终把砝码都移动到一个格子上，最小代价是多少。
- 先解读砝码移动规则：偶数位置的到偶数位置的没有代价，奇数到奇数位置的没有代价。利用这个规则，我们可以把所有的砝码**无代价**的摞在一个奇数的位置上和一个偶数的位置上。这样我们只用关心这两个位置了。并且这两个位置可以连续在一起。最后一步即将相邻的这两摞砝码合并到一起。由于左右移动一个代价是 1，所以最小代价的操作是移动最少砝码的那一边。奇数位置上砝码少就移动奇数位置上的，偶数位置上砝码少就移动偶数位置上的。所以这道题解法变的异常简单，遍历一次数组，找到其中有多少个奇数和偶数位置的砝码，

取其中比较少的，就是最终答案。

代码

```
package leetcode

func minCostToMoveChips(chips []int) int {
  odd, even := 0, 0
  for _, c := range chips {
    if c%2 == 0 {
      even++
    } else {
      odd++
    }
  }
  return min(odd, even)
}
```

1221. Split a String in Balanced Strings

题目

Balanced strings are those who have equal quantity of 'L' and 'R' characters.

Given a balanced string `s` split it in the maximum amount of balanced strings.

Return the maximum amount of splitted balanced strings.

Example 1:

```
Input: s = "RLRRLLRLRL"
Output: 4
Explanation: s can be split into "RL", "RRLL", "RL", "RL", each substring contains same
number of 'L' and 'R'.
```

Example 2:

```
Input: s = "RLLLLRRRLR"
Output: 3
Explanation: s can be split into "RL", "LLLRRR", "LR", each substring contains same
number of 'L' and 'R'.
```

Example 3:

```
Input: s = "LLLLRRRR"
Output: 1
Explanation: s can be split into "LLLLRRRR".
```

Constraints:

- `1 <= s.length <= 1000`
- `s[i] = 'L' or 'R'`

题目大意

在一个「平衡字符串」中，'L' 和 'R' 字符的数量是相同的。给出一个平衡字符串 s，请你将它分割成尽可能多的平衡字符串。返回可以通过分割得到的平衡字符串的最大数量。

提示：

- 1 <= s.length <= 1000
- s[i] = 'L' 或 'R'

解题思路

- 给出一个字符串，要求把这个字符串切成一些子串，这些子串中 R 和 L 的字符数是相等的。问能切成多少个满足条件的子串。
- 这道题是简单题，按照题意模拟即可。从左往右扫，遇到 `R` 就加一，遇到 `L` 就减一，当计数是 `0` 的时候就是平衡的时候，就切割。

代码

```go
package leetcode

func balancedStringSplit(s string) int {
  count, res := 0, 0
  for _, r := range s {
    if r == 'R' {
      count++
    } else {
      count--
    }
    if count == 0 {
      res++
    }
  }
  return res
}
```

1232. Check If It Is a Straight Line

题目

You are given an array `coordinates`, `coordinates[i] = [x, y]`, where `[x, y]` represents the coordinate of a point. Check if these points make a straight line in the XY plane.

Example 1:

```
Input: coordinates = [[1,2],[2,3],[3,4],[4,5],[5,6],[6,7]]
Output: true
```

Example 2:

```
Input: coordinates = [[1,1],[2,2],[3,4],[4,5],[5,6],[7,7]]
Output: false
```

Constraints:

- `2 <= coordinates.length <= 1000`
- `coordinates[i].length == 2`
- `-10^4 <= coordinates[i][0], coordinates[i][1] <= 10^4`
- `coordinates` contains no duplicate point.

题目大意

在一个 XY 坐标系中有一些点，我们用数组 coordinates 来分别记录它们的坐标，其中 coordinates[i] = [x, y] 表示横坐标为 x、纵坐标为 y 的点。

请你来判断，这些点是否在该坐标系中属于同一条直线上，是则返回 true，否则请返回 false。

提示：

- 2 <= coordinates.length <= 1000
- coordinates[i].length == 2
- -10^4 <= coordinates[i][0], coordinates[i][1] <= 10^4
- coordinates 中不含重复的点

解题思路

- 给出一组坐标点，要求判断这些点是否在同一直线上。
- 按照几何原理，依次计算这些点的斜率是否相等即可。斜率需要做除法，这里采用一个技巧是换成乘法。例如 `a/b = c/d` 换成乘法是 `a*d  = c*d` 。

代码

```
package leetcode
```

```go
func checkStraightLine(coordinates [][]int) bool {
  dx0 := coordinates[1][0] - coordinates[0][0]
  dy0 := coordinates[1][1] - coordinates[0][1]
  for i := 1; i < len(coordinates)-1; i++ {
    dx := coordinates[i+1][0] - coordinates[i][0]
    dy := coordinates[i+1][1] - coordinates[i][1]
    if dy*dx0 != dy0*dx { // check cross product
      return false
    }
  }
  return true
}
```

1234. Replace the Substring for Balanced String

题目

You are given a string containing only 4 kinds of characters `'Q', 'W', 'E'` and `'R'`.

A string is said to be **balanced** **if each of its characters appears `n/4` times where `n` is the length of the string.

Return the minimum length of the substring that can be replaced with **any** other string of the same length to make the original string `s` **balanced**.

Return 0 if the string is already **balanced**.

Example 1:

```
Input: s = "QWER"
Output: 0
Explanation: s is already balanced.
```

Example 2:

```
Input: s = "QQWE"
Output: 1
Explanation: We need to replace a 'Q' to 'R', so that "RQWE" (or "QRWE") is balanced.
```

Example 3:

```
Input: s = "QQQW"
Output: 2
Explanation: We can replace the first "QQ" to "ER".
```

Example 4:

```
Input: s = "QQQQ"
Output: 3
Explanation: We can replace the last 3 'Q' to make s = "QWER".
```

Constraints:

- `1 <= s.length <= 10^5`
- `s.length` is a multiple of `4`
- `s` contains only `'Q'`, `'W'`, `'E'` and `'R'`.

题目大意

有一个只含有 'Q', 'W', 'E', 'R' 四种字符，且长度为 n 的字符串。假如在该字符串中，这四个字符都恰好出现 n/4 次，那么它就是一个「平衡字符串」。给你一个这样的字符串 s，请通过「替换一个子串」的方式，使原字符串 s 变成一个「平衡字符串」。你可以用和「待替换子串」长度相同的 任何 其他字符串来完成替换。请返回待替换子串的最小可能长度。如果原字符串自身就是一个平衡字符串，则返回 0。

提示：

- 1 <= s.length <= 10^5
- s.length 是 4 的倍数
- s 中只含有 'Q', 'W', 'E', 'R' 四种字符

解题思路

- 给出一个字符串，要求输出把这个字符串变成“平衡字符串”的最小替换字符串的长度(替换只能替换一串，不能单个字母替换)。“平衡字符串”的定义是：字符串中，`‘Q’`，`‘W’`，`‘E’`，`‘R’`，出现的次数当且仅当只有 `len(s)/4` 次。
- 这一题是滑动窗口的题目。先统计 4 个字母的频次并计算出 `k = len(s)/4` 。滑动窗口向右滑动一次，对应右窗口的那个字母频次减 1，直到滑到所有字母的频次都 `≤ k` 的地方停止。此时，窗口外的字母的频次都 `≤ k` 了。这是只要变换窗口内字符串即可。但是这个窗口内还可能包含本来频次就小于 `k` 的字母，如果能把它们剔除掉，窗口可以进一步的减少。所以继续移动左边界，试探移动完左边界以后，是否所有字母频次都 `≤ k`。在所有窗口移动过程中取出最小值，即为最终答案。
- 举个例子：`"WQWRQQQW"`。`W` 有 3 个，`Q` 有 4 个，`R` 有 1 个，`E` 有 0 个。最后平衡状态是每个字母 2 个，那么我们需要拿出 1 个 `W` 和 2 个 `Q` 替换掉。即要找到一个最短的字符串包含 1 个 `W` 和 2 个 `Q`。滑动窗口正好可以解决这个问题。向右滑到 `"WQWRQ"` 停止，这时窗口外的所有字母频次都 `≤ k` 了。这个窗口包含了多余的 1 个 `W`，和 1 个 `R`。`W` 可以踢除掉，那么要替换的字符串是 `"QWRQ"`。`R` 不能踢除了(因为要找包含 1 个 `W` 和 2 个 `Q` 的字符串) 。窗口不断的滑动，直到结束。这个例子中最小的字符串其实位于末尾，`"QQW"`。

代码

```
```

```go
package leetcode

func balancedString(s string) int {
  count, k := make([]int, 128), len(s)/4
  for _, v := range s {
    count[int(v)]++
  }
  left, right, res := 0, -1, len(s)
  for left < len(s) {
    if count['Q'] > k || count['W'] > k || count['E'] > k || count['R'] > k {
      if right+1 < len(s) {
        right++
        count[s[right]]--
      } else {
        break
      }
    } else {
      res = min(res, right-left+1)
      count[s[left]]++
      left++
    }
  }
  return res
}
```

1235. Maximum Profit in Job Scheduling

题目

We have `n` jobs, where every job is scheduled to be done from `startTime[i]` to `endTime[i]`, obtaining a profit of `profit[i]`.

You're given the `startTime` , `endTime` and `profit` arrays, you need to output the maximum profit you can take such that there are no 2 jobs in the subset with overlapping time range.

If you choose a job that ends at time `X` you will be able to start another job that starts at time `X`.

Example 1:

```
Input: startTime = [1,2,3,3], endTime = [3,4,5,6], profit = [50,10,40,70]
Output: 120
Explanation: The subset chosen is the first and fourth job.
Time range [1-3]+[3-6] , we get profit of 120 = 50 + 70.
```

Example 2:

```
Input: startTime = [1,2,3,4,6], endTime = [3,5,10,6,9], profit = [20,20,100,70,60]
Output: 150
Explanation: The subset chosen is the first, fourth and fifth job.
Profit obtained 150 = 20 + 70 + 60.
```

Example 3:

```
Input: startTime = [1,1,1], endTime = [2,3,4], profit = [5,6,4]
Output: 6
```

Constraints:

- `1 <= startTime.length == endTime.length == profit.length <= 5 * 10^4`
- `1 <= startTime[i] < endTime[i] <= 10^9`
- `1 <= profit[i] <= 10^4`

题目大意

你打算利用空闲时间来做兼职工作赚些零花钱。这里有 n 份兼职工作，每份工作预计从 startTime[i] 开始到 endTime[i] 结束，报酬为 profit[i]。给你一份兼职工作表，包含开始时间 startTime，结束时间 endTime 和预计报酬 profit 三个数组，请你计算并返回可以获得的最大报酬。注意，时间上出现重叠的 2 份工作不能同时进行。如果你选择的工作在时间 X 结束，那么你可以立刻进行在时间 X 开始的下一份工作。

提示：

- 1 <= startTime.length == endTime.length == profit.length <= 5 * 10^4
- 1 <= startTime[i] < endTime[i] <= 10^9
- 1 <= profit[i] <= 10^4

解题思路

- 给出一组任务，任务有开始时间，结束时间，和任务收益。一个任务开始还没有结束，中间就不能再安排其他任务。问如何安排任务，能使得最后收益最大?
- 一般任务的题目，区间的题目，都会考虑是否能排序。这一题可以先按照任务的结束时间从小到大排序，如果结束时间相同，则按照收益从小到大排序。`dp[i]` 代表前 `i` 份工作能获得的最大收益。初始值，`dp[0] = job[1].profit` 。对于任意一个任务 `i` ，看能否找到满足 `jobs[j].enTime <= jobs[j].startTime && j < i` 条件的 `j`，即查找 `upper_bound` 。由于 `jobs` 被我们排序了，所以这里可以使用二分搜索来查找。如果能找到满足条件的任务 j，那么状态转移方程是：`dp[i] = max(dp[i-1], jobs[i].profit)`。如果能找到满足条件的任务 j，那么状态转移方程是：`dp[i] = max(dp[i-1], dp[low]+jobs[i].profit)`。最终求得的解在 `dp[len(startTime)-1]` 中。

代码

```
package leetcode
```

```go
import "sort"

type job struct {
  startTime int
  endTime   int
  profit    int
}

func jobScheduling(startTime []int, endTime []int, profit []int) int {
  jobs, dp := []job{}, make([]int, len(startTime))
  for i := 0; i < len(startTime); i++ {
    jobs = append(jobs, job{startTime: startTime[i], endTime: endTime[i], profit:
profit[i]})
  }
  sort.Sort(sortJobs(jobs))
  dp[0] = jobs[0].profit
  for i := 1; i < len(jobs); i++ {
    low, high := 0, i-1
    for low < high {
      mid := low + (high-low)>>1
      if jobs[mid+1].endTime <= jobs[i].startTime {
        low = mid + 1
      } else {
        high = mid
      }
    }
    if jobs[low].endTime <= jobs[i].startTime {
      dp[i] = max(dp[i-1], dp[low]+jobs[i].profit)
    } else {
      dp[i] = max(dp[i-1], jobs[i].profit)
    }
  }
  return dp[len(startTime)-1]
}

type sortJobs []job

func (s sortJobs) Len() int {
  return len(s)
}
func (s sortJobs) Less(i, j int) bool {
  if s[i].endTime == s[j].endTime {
    return s[i].profit < s[j].profit
  }
  return s[i].endTime < s[j].endTime
}
func (s sortJobs) Swap(i, j int) {
  s[i], s[j] = s[j], s[i]
```

```
}
```

1239. Maximum Length of a Concatenated String with Unique Characters

题目

Given an array of strings `arr`. String `s` is a concatenation of a sub-sequence of `arr` which have **unique characters**.

Return *the maximum possible length* of `s`.

Example 1:

```
Input: arr = ["un","iq","ue"]
Output: 4
Explanation: All possible concatenations are "","un","iq","ue","uniq" and "ique".
Maximum length is 4.
```

Example 2:

```
Input: arr = ["cha","r","act","ers"]
Output: 6
Explanation: Possible solutions are "chaers" and "acters".
```

Example 3:

```
Input: arr = ["abcdefghijklmnopqrstuvwxyz"]
Output: 26
```

Constraints:

- `1 <= arr.length <= 16`
- `1 <= arr[i].length <= 26`
- `arr[i]` contains only lower case English letters.

题目大意

给定一个字符串数组 arr，字符串 s 是将 arr 某一子序列字符串连接所得的字符串，如果 s 中的每一个字符都只出现过一次，那么它就是一个可行解。请返回所有可行解 s 中最长长度。

解题思路

- 每个字符串数组可以想象为 26 位的 0101 二进制串。出现的字符对应的位上标记为 1，没有出现的字符对应的位上标记为 0 。如果一个字符串中包含重复的字符，那么它所有 1 的个数一定不等于字符串的长度。如果 2 个字符串每个字母都只出现了一次，那么它们俩对应的二进制串 mask 相互与运算的结果一定为 0 ，即 0，

1 互补了。利用这个特点，深搜所有解，保存出最长可行解的长度即可。

代码

```
package leetcode

import (
  "math/bits"
)

func maxLength(arr []string) int {
  c, res := []uint32{}, 0
  for _, s := range arr {
    var mask uint32
    for _, c := range s {
      mask = mask | 1<<(c-'a')
    }
    if len(s) != bits.OnesCount32(mask) { // 如果字符串本身带有重复的字符，需要排除
      continue
    }
    c = append(c, mask)
  }
  dfs(c, 0, 0, &res)
  return res
}

func dfs(c []uint32, index int, mask uint32, res *int) {
  *res = max(*res, bits.OnesCount32(mask))
  for i := index; i < len(c); i++ {
    if mask&c[i] == 0 {
      dfs(c, i+1, mask|c[i], res)
    }
  }
  return
}

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

1249. Minimum Remove to Make Valid Parentheses

题目

Given a string s of `'('` , `')'` and lowercase English characters.

Your task is to remove the minimum number of parentheses (`'('` or `')'`, in any positions) so that the resulting *parentheses string* is valid and return **any** valid string.

Formally, a *parentheses string* is valid if and only if:

- It is the empty string, contains only lowercase characters, or
- It can be written as `AB` (`A` concatenated with `B`), where `A` and `B` are valid strings, or
- It can be written as `(A)`, where `A` is a valid string.

Example 1:

```
Input: s = "lee(t(c)o)de)"
Output: "lee(t(c)o)de"
Explanation: "lee(t(co)de)" , "lee(t(c)ode)" would also be accepted.
```

Example 2:

```
Input: s = "a)b(c)d"
Output: "ab(c)d"
```

Example 3:

```
Input: s = "))(("
Output: ""
Explanation: An empty string is also valid.
```

Example 4:

```
Input: s = "(a(b(c)d)"
Output: "a(b(c)d)"
```

Constraints:

- `1 <= s.length <= 10^5`
- `s[i]` is one of `'('` , `')'` and lowercase English letters`.`

题目大意

给你一个由 '('、')' 和小写字母组成的字符串 s。你需要从字符串中删除最少数目的 '(' 或者 ')' （可以删除任意位置的括号），使得剩下的「括号字符串」有效。请返回任意一个合法字符串。有效「括号字符串」应当符合以下 任意一条 要求：

- 空字符串或只包含小写字母的字符串

- 可以被写作 AB（A 连接 B）的字符串，其中 A 和 B 都是有效「括号字符串」
- 可以被写作 (A) 的字符串，其中 A 是一个有效的「括号字符串」

解题思路

- 最容易想到的思路是利用栈判断括号匹配是否有效。这个思路可行，时间复杂度也只是 O(n)。
- 不用栈，可以 2 次循环遍历，正向遍历一次，标记出多余的 `'('`，逆向遍历一次，再标记出多余的 `')'`，最后将所有这些标记多余的字符删掉即可。这种解法写出来的代码也很简洁，时间复杂度也是 O(n)。
- 针对上面的解法再改进一点。正向遍历的时候不仅标记出多余的 `'('`，还可以顺手把多余的 `')'` 删除。这样只用循环一次。最后再删除掉多余的 `'('` 即可。时间复杂度还是 O(n)。

代码

```go
package leetcode

func minRemoveToMakeValid(s string) string {
  res, opens := []byte{}, 0
  for i := 0; i < len(s); i++ {
    if s[i] == '(' {
      opens++
    } else if s[i] == ')' {
      if opens == 0 {
        continue
      }
      opens--
    }
    res = append(res, s[i])
  }
  for i := len(res) - 1; i >= 0; i-- {
    if res[i] == '(' && opens > 0 {
      opens--
      res = append(res[:i], res[i+1:]...)
    }
  }
  return string(res)
}
```

1252. Cells with Odd Values in a Matrix

题目

Given `n` and `m` which are the dimensions of a matrix initialized by zeros and given an array `indices` where `indices[i] = [ri, ci]`. For each pair of `[ri, ci]` you have to increment all cells in row `ri` and column `ci` by 1.

Return *the number of cells with odd values* in the matrix after applying the increment to all `indices`.

Example 1:

```
Input: n = 2, m = 3, indices = [[0,1],[1,1]]
Output: 6
Explanation: Initial matrix = [[0,0,0],[0,0,0]].
After applying first increment it becomes [[1,2,1],[0,1,0]].
The final matrix will be [[1,3,1],[1,3,1]] which contains 6 odd numbers.
```

Example 2:

```
Input: n = 2, m = 2, indices = [[1,1],[0,0]]
Output: 0
Explanation: Final matrix = [[2,2],[2,2]]. There is no odd number in the final matrix.
```

Constraints:

- `1 <= n <= 50`
- `1 <= m <= 50`
- `1 <= indices.length <= 100`
- `0 <= indices[i][0] < n`
- `0 <= indices[i][1] < m`

题目大意

给你一个 n 行 m 列的矩阵，最开始的时候，每个单元格中的值都是 0。另有一个索引数组 indices，indices[i] = [ri, ci] 中的 ri 和 ci 分别表示指定的行和列（从 0 开始编号）。你需要将每对 [ri, ci] 指定的行和列上的所有单元格的值加 1。请你在执行完所有 indices 指定的增量操作后，返回矩阵中 「奇数值单元格」 的数目。

提示：

- 1 <= n <= 50
- 1 <= m <= 50
- 1 <= indices.length <= 100
- 0 <= indices[i][0] < n
- 0 <= indices[i][1] < m

解题思路

- 给出一个 n * m 的矩阵，和一个数组，数组里面包含一些行列坐标，并在指定坐标上 + 1，问最后 n * m 的矩阵中奇数的总数。
- 暴力方法按照题意模拟即可。

代码

```go
package leetcode

// 解法一 暴力法
```

```go
func oddCells(n int, m int, indices [][]int) int {
  matrix, res := make([][]int, n), 0
  for i := range matrix {
    matrix[i] = make([]int, m)
  }
  for _, indice := range indices {
    for i := 0; i < m; i++ {
      matrix[indice[0]][i]++
    }
    for j := 0; j < n; j++ {
      matrix[j][indice[1]]++
    }
  }
  for _, m := range matrix {
    for _, v := range m {
      if v&1 == 1 {
        res++
      }
    }
  }
  return res
}

// 解法二 暴力法
func oddCells1(n int, m int, indices [][]int) int {
  rows, cols, count := make([]int, n), make([]int, m), 0
  for _, pair := range indices {
    rows[pair[0]]++
    cols[pair[1]]++
  }
  for i := 0; i < n; i++ {
    for j := 0; j < m; j++ {
      if (rows[i]+cols[j])%2 == 1 {
        count++
      }
    }
  }
  return count
}
```

1254. Number of Closed Islands

题目

Given a 2D `grid` consists of `0s` (land) and `1s` (water). An *island* is a maximal 4-directionally connected group of `0s` and a *closed island* is an island **totally** (all left, top, right, bottom) surrounded by `1s.`

Return the number of *closed islands*.

Example 1:

```
Input: grid = [[1,1,1,1,1,1,1,0],[1,0,0,0,0,1,1,0],[1,0,1,0,1,1,1,0],[1,0,0,0,0,1,0,1],
[1,1,1,1,1,1,1,0]]
Output: 2
Explanation:
Islands in gray are closed because they are completely surrounded by water (group of
1s).
```

Example 2:

```
Input: grid = [[0,0,1,0,0],[0,1,0,1,0],[0,1,1,1,0]]
Output: 1
```

Example 3:

```
Input: grid = [[1,1,1,1,1,1,1],
           [1,0,0,0,0,0,1],
           [1,0,1,1,1,0,1],
           [1,0,1,0,1,0,1],
           [1,0,1,1,1,0,1],
           [1,0,0,0,0,0,1],
           [1,1,1,1,1,1,1]]
Output: 2
```

Constraints:

- `1 <= grid.length, grid[0].length <= 100`
- `0 <= grid[i][j] <=1`

题目大意

有一个二维矩阵 grid ，每个位置要么是陆地（记号为 0 ）要么是水域（记号为 1 ）。我们从一块陆地出发，每次可以往上下左右 4 个方向相邻区域走，能走到的所有陆地区域，我们将其称为一座「岛屿」。如果一座岛屿 完全 由水域包围，即陆地边缘上下左右所有相邻区域都是水域，那么我们将其称为 「封闭岛屿」。请返回封闭岛屿的数目。

提示：

- 1 <= grid.length, grid[0].length <= 100
- 0 <= grid[i][j] <=1

解题思路

- 给出一个地图，1 代表海水，0 代表陆地。要求找出四周都是海水的陆地的总个数。
- 这一题和第 200 题解题思路完全一致。只不过这一题要求必须四周都是海水，第 200 题的陆地可以是靠着地

图边缘的。在此题中，靠着地图边缘的陆地不能最终计数到结果中。

代码

```
package leetcode

func closedIsland(grid [][]int) int {
  m := len(grid)
  if m == 0 {
    return 0
  }
  n := len(grid[0])
  if n == 0 {
    return 0
  }
  res, visited := 0, make([][]bool, m)
  for i := 0; i < m; i++ {
    visited[i] = make([]bool, n)
  }
  for i := 0; i < m; i++ {
    for j := 0; j < n; j++ {
      isEdge := false
      if grid[i][j] == 0 && !visited[i][j] {
        checkIslands(grid, &visited, i, j, &isEdge)
        if !isEdge {
          res++
        }

      }
    }
  }
  return res
}

func checkIslands(grid [][]int, visited *[][]bool, x, y int, isEdge *bool) {
  if (x == 0 || x == len(grid)-1 || y == 0 || y == len(grid[0])-1) && grid[x][y] == 0 {
    *isEdge = true
  }
  (*visited)[x][y] = true
  for i := 0; i < 4; i++ {
    nx := x + dir[i][0]
    ny := y + dir[i][1]
    if isIntInBoard(grid, nx, ny) && !(*visited)[nx][ny] && grid[nx][ny] == 0 {
      checkIslands(grid, visited, nx, ny, isEdge)
    }
  }
  *isEdge = *isEdge || false
}
```

```go
func isIntInBoard(board [][]int, x, y int) bool {
  return x >= 0 && x < len(board) && y >= 0 && y < len(board[0])
}
```

1260. Shift 2D Grid

题目

Given a 2D `grid` of size `m x n` and an integer `k`. You need to shift the `grid` `k` times.

In one shift operation:

- Element at `grid[i][j]` moves to `grid[i][j + 1]`.
- Element at `grid[i][n - 1]` moves to `grid[i + 1][0]`.
- Element at `grid[m - 1][n - 1]` moves to `grid[0][0]`.

Return the *2D grid* after applying shift operation `k` times.

Example 1:

```
Input: grid = [[1,2,3],[4,5,6],[7,8,9]], k = 1
Output: [[9,1,2],[3,4,5],[6,7,8]]
```

Example 2:

```
Input: grid = [[3,8,1,9],[19,7,2,5],[4,6,11,10],[12,0,21,13]], k = 4
Output: [[12,0,21,13],[3,8,1,9],[19,7,2,5],[4,6,11,10]]
```

Example 3:

```
Input: grid = [[1,2,3],[4,5,6],[7,8,9]], k = 9
Output: [[1,2,3],[4,5,6],[7,8,9]]
```

Constraints:

- `m == grid.length`
- `n == grid[i].length`
- `1 <= m <= 50`
- `1 <= n <= 50`
- `-1000 <= grid[i][j] <= 1000`
- `0 <= k <= 100`

题目大意

给你一个 m 行 n 列的二维网格 grid 和一个整数 k。你需要将 grid 迁移 k 次。每次「迁移」操作将会引发下述活动：

- 位于 grid[i][j] 的元素将会移动到 grid[i][j + 1]。
- 位于 grid[i][n - 1] 的元素将会移动到 grid[i + 1][0]。
- 位于 grid[m - 1][n - 1] 的元素将会移动到 grid[0][0]。

请你返回 k 次迁移操作后最终得到的 二维网格。

解题思路

- 给一个矩阵和一个移动步数 k，要求把矩阵每个元素往后移动 k 步，最后的元素移动头部，循环移动，最后输出移动结束的矩阵。
- 简单题，按照题意循环移动即可，注意判断边界情况。

代码

```
package leetcode

func shiftGrid(grid [][]int, k int) [][]int {
  x, y := len(grid[0]), len(grid)
  newGrid := make([][]int, y)
  for i := 0; i < y; i++ {
    newGrid[i] = make([]int, x)
  }
  for i := 0; i < y; i++ {
    for j := 0; j < x; j++ {
      ny := (k / x) + i
      if (j + (k % x)) >= x {
        ny++
      }
      newGrid[ny%y][(j+(k%x))%x] = grid[i][j]
    }
  }
  return newGrid
}
```

1266. Minimum Time Visiting All Points

题目

On a plane there are `n` points with integer coordinates `points[i] = [xi, yi]`. Your task is to find the minimum time in seconds to visit all points.

You can move according to the next rules:

- In one second always you can either move vertically, horizontally by one unit or diagonally (it means to

move one unit vertically and one unit horizontally in one second).
- You have to visit the points in the same order as they appear in the array.

Example 1:

```
Input: points = [[1,1],[3,4],[-1,0]]
Output: 7
Explanation: One optimal path is [1,1] -> [2,2] -> [3,3] -> [3,4] -> [2,3] -> [1,2] ->
[0,1] -> [-1,0]
Time from [1,1] to [3,4] = 3 seconds
Time from [3,4] to [-1,0] = 4 seconds
Total time = 7 seconds
```

Example 2:

```
Input: points = [[3,2],[-2,2]]
Output: 5
```

Constraints:

- `points.length == n`
- `1 <= n <= 100`
- `points[i].length == 2`
- `-1000 <= points[i][0], points[i][1] <= 1000`

题目大意

平面上有 n 个点，点的位置用整数坐标表示 points[i] = [xi, yi]。请你计算访问所有这些点需要的最小时间（以秒为单位）。你可以按照下面的规则在平面上移动：

- 每一秒沿水平或者竖直方向移动一个单位长度，或者跨过对角线（可以看作在一秒内向水平和竖直方向各移动一个单位长度）。
- 必须按照数组中出现的顺序来访问这些点。

提示：

- points.length == n
- 1 <= n <= 100
- points[i].length == 2
- -1000 <= points[i][0], points[i][1] <= 1000

解题思路

- 在直角坐标系上给出一个数组，数组里面的点是飞机飞行经过的点。飞机飞行只能沿着水平方向、垂直方向、45°方向飞行。问飞机经过所有点的最短时间。
- 简单的数学问题。依次遍历数组，分别计算 x 轴和 y 轴上的差值，取最大值即是这两点之间飞行的最短时

间。最后累加每次计算的最大值就是最短时间。

代码

```
package leetcode

func minTimeToVisitAllPoints(points [][]int) int {
  res := 0
  for i := 1; i < len(points); i++ {
    res += max(abs(points[i][0]-points[i-1][0]), abs(points[i][1]-points[i-1][1]))
  }
  return res
}
```

1268. Search Suggestions System

题目

Given an array of strings `products` and a string `searchWord`. We want to design a system that suggests at most three product names from `products` after each character of `searchWord` is typed. Suggested products should have common prefix with the searchWord. If there are more than three products with a common prefix return the three lexicographically minimums products.

Return *list of lists* of the suggested `products` after each character of `searchWord` is typed.

Example 1:

```
Input: products = ["mobile","mouse","moneypot","monitor","mousepad"], searchWord =
"mouse"
Output: [
["mobile","moneypot","monitor"],
["mobile","moneypot","monitor"],
["mouse","mousepad"],
["mouse","mousepad"],
["mouse","mousepad"]
]
Explanation: products sorted lexicographically =
["mobile","moneypot","monitor","mouse","mousepad"]
After typing m and mo all products match and we show user
["mobile","moneypot","monitor"]
After typing mou, mous and mouse the system suggests ["mouse","mousepad"]
```

Example 2:

```
Input: products = ["havana"], searchWord = "havana"
Output: [["havana"],["havana"],["havana"],["havana"],["havana"],["havana"]]
```

Example 3:

```
Input: products = ["bags","baggage","banner","box","cloths"], searchWord = "bags"
Output: [["baggage","bags","banner"],["baggage","bags","banner"],["baggage","bags"],
["bags"]]
```

Example 4:

```
Input: products = ["havana"], searchWord = "tatiana"
Output: [[],[],[],[],[],[],[]]
```

Constraints:

- `1 <= products.length <= 1000`
- There are no repeated elements in `products`.
- `1 <= Σ products[i].length <= 2 * 10^4`
- All characters of `products[i]` are lower-case English letters.
- `1 <= searchWord.length <= 1000`
- All characters of `searchWord` are lower-case English letters.

题目大意

给你一个产品数组 products 和一个字符串 searchWord ， products 数组中每个产品都是一个字符串。请你设计一个推荐系统，在依次输入单词 searchWord 的每一个字母后，推荐 products 数组中前缀与 searchWord 相同的最多三个产品。如果前缀相同的可推荐产品超过三个，请按字典序返回最小的三个。请你以二维列表的形式，返回在输入 searchWord 每个字母后相应的推荐产品的列表。

解题思路

- 由于题目要求返回的答案要按照字典序输出，所以先排序。有序字符串又满足了二分搜索的条件，于是可以用二分搜索。sort.SearchStrings 返回的是满足搜索条件的第一个起始下标。末尾不满足条件的字符串要切掉。所以要搜 2 次，第一次二分搜索先将不满足目标串前缀的字符串筛掉。第二次二分搜索再搜索出最终满足题意的字符串。

代码

```go
package leetcode

import (
  "sort"
)

func suggestedProducts(products []string, searchWord string) [][]string {
```

```
  sort.Strings(products)
  searchWordBytes, result := []byte(searchWord), make([][]string, 0, len(searchWord))
  for i := 1; i <= len(searchWord); i++ {
    searchWordBytes[i-1]++
    products = products[:sort.SearchStrings(products, string(searchWordBytes[:i]))]
    searchWordBytes[i-1]--
    products = products[sort.SearchStrings(products, searchWord[:i]):]
    if len(products) > 3 {
      result = append(result, products[:3])
    } else {
      result = append(result, products)
    }
  }
  return result
}
```

1275. Find Winner on a Tic Tac Toe Game

题目

Tic-tac-toe is played by two players *A* and *B* on a 3 x 3 grid.

Here are the rules of Tic-Tac-Toe:

- Players take turns placing characters into empty squares (" ").
- The first player *A* always places "X" characters, while the second player *B* always places "O" characters.
- "X" and "O" characters are always placed into empty squares, never on filled ones.
- The game ends when there are 3 of the same (non-empty) character filling any row, column, or diagonal.
- The game also ends if all squares are non-empty.
- No more moves can be played if the game is over.

Given an array `moves` where each element is another array of size 2 corresponding to the row and column of the grid where they mark their respective character in the order in which *A* and *B* play.

Return the winner of the game if it exists (*A* or *B*), in case the game ends in a draw return "Draw", if there are still movements to play return "Pending".

You can assume that `moves` is **valid** (It follows the rules of Tic-Tac-Toe), the grid is initially empty and *A* will play **first**.

Example 1:

```
Input: moves = [[0,0],[2,0],[1,1],[2,1],[2,2]]
Output: "A"
Explanation: "A" wins, he always plays first.
"X  "    "X  "    "X  "    "X  "    "X  "
"   " -> "   " -> " X " -> " X " -> " X "
"   "    "O  "    "O  "    "OO "    "OOX"
```

Example 2:

```
Input: moves = [[0,0],[1,1],[0,1],[0,2],[1,0],[2,0]]
Output: "B"
Explanation: "B" wins.
"X  "    "X  "    "XX "    "XXO"    "XXO"    "XXO"
"   " -> " O " -> " O " -> " O " -> "XO " -> "XO "
"   "    "   "    "   "    "   "    "   "    "O  "
```

Example 3:

```
Input: moves = [[0,0],[1,1],[2,0],[1,0],[1,2],[2,1],[0,1],[0,2],[2,2]]
Output: "Draw"
Explanation: The game ends in a draw since there are no moves to make.
"XXO"
"OOX"
"XOX"
```

Example 4:

```
Input: moves = [[0,0],[1,1]]
Output: "Pending"
Explanation: The game has not finished yet.
"X  "
" O "
"   "
```

Constraints:

- `1 <= moves.length <= 9`
- `moves[i].length == 2`
- `0 <= moves[i][j] <= 2`
- There are no repeated elements on `moves`.
- `moves` follow the rules of tic tac toe.

题目大意

A 和 B 在一个 3 x 3 的网格上玩井字棋。井字棋游戏的规则如下：

- 玩家轮流将棋子放在空方格 (" ") 上。
- 第一个玩家 A 总是用 "X" 作为棋子，而第二个玩家 B 总是用 "O" 作为棋子。
- "X" 和 "O" 只能放在空方格中，而不能放在已经被占用的方格上。
- 只要有 3 个相同的（非空）棋子排成一条直线（行、列、对角线）时，游戏结束。
- 如果所有方块都放满棋子（不为空），游戏也会结束。
- 游戏结束后，棋子无法再进行任何移动。

给你一个数组 moves，其中每个元素是大小为 2 的另一个数组（元素分别对应网格的行和列），它按照 A 和 B 的行动顺序（先 A 后 B）记录了两人各自的棋子位置。如果游戏存在获胜者（A 或 B），就返回该游戏的获胜者；如果游戏以平局结束，则返回 "Draw"；如果仍会有行动（游戏未结束），则返回 "Pending"。你可以假设 moves 都有效（遵循井字棋规则），网格最初是空的，A 将先行动。

提示：

- 1 <= moves.length <= 9
- moves[i].length == 2
- 0 <= moves[i][j] <= 2
- moves 里没有重复的元素。
- moves 遵循井字棋的规则。

解题思路

- 两人玩 3*3 井字棋，A 先走，B 再走。谁能获胜就输出谁，如果平局输出 “Draw”，如果游戏还未结束，输出 “Pending”。游戏规则：谁能先占满行、列或者对角线任意一条线，谁就赢。
- 简单题。题目给定 move 数组最多 3 步，而要赢得比赛，必须走满 3 步，所以可以先模拟，按照给的步数数组把 A 和 B 的步数都放在棋盘上。然后依次判断行、列，对角线的三种情况。如果都判完了，剩下的情况就是平局和死局的情况。

代码

```go
package leetcode

func tictactoe(moves [][]int) string {
  board := [3][3]byte{}
  for i := 0; i < len(moves); i++ {
    if i%2 == 0 {
      board[moves[i][0]][moves[i][1]] = 'X'
    } else {
      board[moves[i][0]][moves[i][1]] = 'O'
    }
  }
  for i := 0; i < 3; i++ {
    if board[i][0] == 'X' && board[i][1] == 'X' && board[i][2] == 'X' {
      return "A"
    }
    if board[i][0] == 'O' && board[i][1] == 'O' && board[i][2] == 'O' {
      return "B"
```

```
    }
    if board[0][i] == 'X' && board[1][i] == 'X' && board[2][i] == 'X' {
      return "A"
    }
    if board[0][i] == 'O' && board[1][i] == 'O' && board[2][i] == 'O' {
      return "B"
    }
  }
  if board[0][0] == 'X' && board[1][1] == 'X' && board[2][2] == 'X' {
    return "A"
  }
  if board[0][0] == 'O' && board[1][1] == 'O' && board[2][2] == 'O' {
    return "B"
  }
  if board[0][2] == 'X' && board[1][1] == 'X' && board[2][0] == 'X' {
    return "A"
  }
  if board[0][2] == 'O' && board[1][1] == 'O' && board[2][0] == 'O' {
    return "B"
  }
  if len(moves) < 9 {
    return "Pending"
  }
  return "Draw"
}
```

1281. Subtract the Product and Sum of Digits of an Integer

题目

Given an integer number `n`, return the difference between the product of its digits and the sum of its digits.

Example 1:

```
Input: n = 234
Output: 15
Explanation:
Product of digits = 2 * 3 * 4 = 24
Sum of digits = 2 + 3 + 4 = 9
Result = 24 - 9 = 15
```

Example 2:

```
Input: n = 4421
Output: 21
Explanation:
Product of digits = 4 * 4 * 2 * 1 = 32
Sum of digits = 4 + 4 + 2 + 1 = 11
Result = 32 - 11 = 21
```

Constraints:

- `1 <= n <= 10^5`

题目大意

给你一个整数 n，请你帮忙计算并返回该整数「各位数字之积」与「各位数字之和」的差。

提示：

- 1 <= n <= 10^5

解题思路

- 给出一个数，计算这个数每位数字乘积减去每位数字累加的差值。
- 简单题，按照题意输入输出即可。

代码

```go
func subtractProductAndSum(n int) int {
  sum, product := 0, 1
  for ; n > 0; n /= 10 {
    sum += n % 10
    product *= n % 10
  }
  return product - sum
}
```

1283. Find the Smallest Divisor Given a Threshold

题目

Given an array of integers `nums` and an integer `threshold`, we will choose a positive integer divisor and divide all the array by it and sum the result of the division. Find the **smallest** divisor such that the result mentioned above is less than or equal to `threshold`.

Each result of division is rounded to the nearest integer greater than or equal to that element. (For example: 7/3 = 3 and 10/2 = 5).

It is guaranteed that there will be an answer.

Example 1:

```
Input: nums = [1,2,5,9], threshold = 6
Output: 5
Explanation: We can get a sum to 17 (1+2+5+9) if the divisor is 1.
If the divisor is 4 we can get a sum to 7 (1+1+2+3) and if the divisor is 5 the sum will be 5 (1+1+1+2).
```

Example 2:

```
Input: nums = [2,3,5,7,11], threshold = 11
Output: 3
```

Example 3:

```
Input: nums = [19], threshold = 5
Output: 4
```

Constraints:

- `1 <= nums.length <= 5 * 10^4`
- `1 <= nums[i] <= 10^6`
- `nums.length <= threshold <= 10^6`

题目大意

给你一个整数数组 nums 和一个正整数 threshold ，你需要选择一个正整数作为除数，然后将数组里每个数都除以它，并对除法结果求和。请你找出能够使上述结果小于等于阈值 threshold 的除数中 最小 的那个。每个数除以除数后都向上取整，比方说 7/3 = 3 ， 10/2 = 5 。题目保证一定有解。

提示：

- 1 <= nums.length <= 5 * 10^4
- 1 <= nums[i] <= 10^6
- nums.length <= threshold <= 10^6

解题思路

- 给出一个数组和一个阈值，要求找到一个除数，使得数组里面每个数和这个除数的商之和不超过这个阈值。求除数的最小值。
- 这一题是典型的二分搜索的题目。根据题意，在 [1, 1000000] 区间内搜索除数，针对每次 `mid`，计算一次商的累加和。如果和比 `threshold` 小，说明除数太大，所以缩小右区间；如果和比 `threshold` 大，说明除数太小，所以缩小左区间。最终找到的 `low` 值就是最求的最小除数。

代码

```go
func smallestDivisor(nums []int, threshold int) int {
  low, high := 1, 1000000
  for low < high {
    mid := low + (high-low)>>1
    if calDivisor(nums, mid, threshold) {
      high = mid
    } else {
      low = mid + 1
    }
  }
  return low
}

func calDivisor(nums []int, mid, threshold int) bool {
  sum := 0
  for i := range nums {
    if nums[i]%mid != 0 {
      sum += nums[i]/mid + 1
    } else {
      sum += nums[i] / mid
    }
  }
  if sum <= threshold {
    return true
  }
  return false
}
```

1287. Element Appearing More Than 25% In Sorted Array

题目

Given an integer array **sorted** in non-decreasing order, there is exactly one integer in the array that occurs more than 25% of the time.

Return that integer.

Example 1:

```
Input: arr = [1,2,2,6,6,6,6,7,10]
Output: 6
```

Constraints:

- `1 <= arr.length <= 10^4`

- `0 <= arr[i] <= 10^5`

题目大意

给你一个非递减的 有序 整数数组，已知这个数组中恰好有一个整数，它的出现次数超过数组元素总数的 25%。请你找到并返回这个整数。

提示：

- 1 <= arr.length <= 10^4
- 0 <= arr[i] <= 10^5

解题思路

- 给出一个非递减的有序数组，要求输出出现次数超过数组元素总数 25% 的元素。
- 简单题，由于已经非递减有序了，所以只需要判断 `arr[i] == arr[i+n/4]` 是否相等即可。

代码

```go
func findSpecialInteger(arr []int) int {
  n := len(arr)
  for i := 0; i < n-n/4; i++ {
    if arr[i] == arr[i+n/4] {
      return arr[i]
    }
  }
  return -1
}
```

1290. Convert Binary Number in a Linked List to Integer

题目

Given `head` which is a reference node to a singly-linked list. The value of each node in the linked list is either 0 or 1. The linked list holds the binary representation of a number.

Return the *decimal value* of the number in the linked list.

Example 1:

```
Input: head = [1,0,1]
Output: 5
Explanation: (101) in base 2 = (5) in base 10
```

Example 2:

```
Input: head = [0]
Output: 0
```

Example 3:

```
Input: head = [1]
Output: 1
```

Example 4:

```
Input: head = [1,0,0,1,0,0,1,1,1,0,0,0,0,0,0]
Output: 18880
```

Example 5:

```
Input: head = [0,0]
Output: 0
```

Constraints:

- The Linked List is not empty.
- Number of nodes will not exceed `30`.
- Each node's value is either `0` or `1`.

题目大意

给你一个单链表的引用结点 head。链表中每个结点的值不是 0 就是 1。已知此链表是一个整数数字的二进制表示形式。请你返回该链表所表示数字的 十进制值 。

提示：

- 链表不为空。
- 链表的结点总数不超过 30。
- 每个结点的值不是 0 就是 1。

解题思路

- 给出一个链表，链表从头到尾表示的数是一个整数的二进制形式，要求输出这个整数的十进制。
- 简单题，从头到尾遍历一次链表，边遍历边累加二进制位。

代码

```go
func getDecimalValue(head *ListNode) int {
  sum := 0
  for head != nil {
    sum = sum*2 + head.Val
    head = head.Next
  }
  return sum
}
```

1295. Find Numbers with Even Number of Digits

题目

Given an array `nums` of integers, return how many of them contain an **even number** of digits.

Example 1:

```
Input: nums = [12,345,2,6,7896]
Output: 2
Explanation:
12 contains 2 digits (even number of digits).
345 contains 3 digits (odd number of digits).
2 contains 1 digit (odd number of digits).
6 contains 1 digit (odd number of digits).
7896 contains 4 digits (even number of digits).
Therefore only 12 and 7896 contain an even number of digits.
```

Example 2:

```
Input: nums = [555,901,482,1771]
Output: 1
Explanation:
Only 1771 contains an even number of digits.
```

Constraints:

- `1 <= nums.length <= 500`
- `1 <= nums[i] <= 10^5`

题目大意

给你一个整数数组 nums，请你返回其中位数为 偶数 的数字的个数。

提示：

- 1 <= nums.length <= 500
- 1 <= nums[i] <= 10^5

解题思路

- 给你一个整数数组，要求输出位数为偶数的数字的个数。
- 简单题，把每个数字转换为字符串判断长度是否是偶数即可。

代码

```
func findNumbers(nums []int) int {
  res := 0
  for _, n := range nums {
    res += 1 - len(strconv.Itoa(n))%2
  }
  return res
}
```

1299. Replace Elements with Greatest Element on Right Side

题目

Given an array `arr`, replace every element in that array with the greatest element among the elements to its right, and replace the last element with `-1`.

After doing so, return the array.

Example 1:

```
Input: arr = [17,18,5,4,6,1]
Output: [18,6,6,6,1,-1]
```

Constraints:

- `1 <= arr.length <= 10^4`
- `1 <= arr[i] <= 10^5`

题目大意

给你一个数组 arr ，请你将每个元素用它右边最大的元素替换，如果是最后一个元素，用 -1 替换。完成所有替换操作后，请你返回这个数组。

提示：

- 1 <= arr.length <= 10^4
- 1 <= arr[i] <= 10^5

解题思路

- 给出一个数组，要求把所有元素都替换成自己右边最大的元素，最后一位补上 -1 。最后输出变化以后的数组。
- 简单题，按照题意操作即可。

代码

```
func replaceElements(arr []int) []int {
  j, temp := -1, 0
  for i := len(arr) - 1; i >= 0; i-- {
    temp = arr[i]
    arr[i] = j
    j = max(j, temp)
  }
  return arr
}
```

1300. Sum of Mutated Array Closest to Target

题目

Given an integer array `arr` and a target value `target`, return the integer `value` such that when we change all the integers larger than `value` in the given array to be equal to `value`, the sum of the array gets as close as possible (in absolute difference) to `target`.

In case of a tie, return the minimum such integer.

Notice that the answer is not neccesarilly a number from `arr`.

Example 1:

```
Input: arr = [4,9,3], target = 10
Output: 3
Explanation: When using 3 arr converts to [3, 3, 3] which sums 9 and that's the optimal answer.
```

Example 2:

```
Input: arr = [2,3,5], target = 10
Output: 5
```

Example 3:

```
Input: arr = [60864,25176,27249,21296,20204], target = 56803
Output: 11361
```

Constraints:

- `1 <= arr.length <= 10^4`
- `1 <= arr[i], target <= 10^5`

题目大意

给你一个整数数组 arr 和一个目标值 target ，请你返回一个整数 value ，使得将数组中所有大于 value 的值变成 value 后，数组的和最接近 target （最接近表示两者之差的绝对值最小）。如果有多种使得和最接近 target 的方案，请你返回这些整数中的最小值。请注意，答案不一定是 arr 中的数字。

提示：

- 1 <= arr.length <= 10^4
- 1 <= arr[i], target <= 10^5

解题思路

- 给出一个数组 arr 和 target。能否找到一个 value 值，使得将数组中所有大于 value 的值变成 value 后，数组的和最接近 target。如果有多种方法，输出 value 值最小的。
- 这一题可以用二分搜索来求解。最后输出的唯一解有 2 个限制条件，一个是变化后的数组和最接近 target 。另一个是输出的 value 是所有可能方法中最小值。二分搜索最终的 value 值。mid 就是尝试的 value 值，每选择一次 mid，就算一次总和，和 target 比较。由于数组里面每个数和 mid 差距各不相同，所以每次调整 mid 有可能出现 mid 选小了以后，距离 target 反而更大了；mid 选大了以后，距离 target 反而更小了。这里的解决办法是，把 value 上下方可能的值都拿出来比较一下。

代码

```go
func findBestValue(arr []int, target int) int {
  low, high := 0, 100000
  for low < high {
    mid := low + (high-low)>>1
    if calculateSum(arr, mid) < target {
      low = mid + 1
    } else {
      high = mid
    }
  }
  if high == 100000 {
    res := 0
    for _, num := range arr {
      if res < num {
        res = num
```

```
      }
    }
    return res
  }
  // 比较阈值线分别定在 left - 1 和 left 的时候与 target 的接近程度
  sum1, sum2 := calculateSum(arr, low-1), calculateSum(arr, low)
  if target-sum1 <= sum2-target {
    return low - 1
  }
  return low
}

func calculateSum(arr []int, mid int) int {
  sum := 0
  for _, num := range arr {
    sum += min(num, mid)
  }
  return sum
}

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

1302. Deepest Leaves Sum

题目

Given a binary tree, return the sum of values of its deepest leaves.

Example 1:

```
Input: root = [1,2,3,4,5,null,6,7,null,null,null,null,8]
Output: 15
```

Constraints:

- The number of nodes in the tree is between `1` and `10^4`.
- The value of nodes is between `1` and `100`.

题目大意

给你一棵二叉树，请你返回层数最深的叶子节点的和。

提示：

- 树中节点数目在 1 到 10^4 之间。
- 每个节点的值在 1 到 100 之间。

解题思路

- 给你一棵二叉树，请你返回层数最深的叶子节点的和。
- 这一题不难，DFS 遍历把最底层的叶子节点和都加起来即可。

代码

```
func deepestLeavesSum(root *TreeNode) int {
  maxLevel, sum := 0, 0
  dfsDeepestLeavesSum(root, 0, &maxLevel, &sum)
  return sum
}

func dfsDeepestLeavesSum(root *TreeNode, level int, maxLevel, sum *int) {
  if root == nil {
    return
  }
  if level > *maxLevel {
    *maxLevel, *sum = level, root.Val
  } else if level == *maxLevel {
    *sum += root.Val
  }
  dfsDeepestLeavesSum(root.Left, level+1, maxLevel, sum)
  dfsDeepestLeavesSum(root.Right, level+1, maxLevel, sum)
}
```

1304. Find N Unique Integers Sum up to Zero

题目

Given an integer `n`, return **any** array containing `n` **unique** integers such that they add up to 0.

Example 1:

```
Input: n = 5
Output: [-7,-1,1,3,4]
Explanation: These arrays also are accepted [-5,-1,1,2,3] , [-3,-1,2,-2,4].
```

Example 2:

```
Input: n = 3
Output: [-1,0,1]
```

Example 3:

```
Input: n = 1
Output: [0]
```

Constraints:

- `1 <= n <= 1000`

题目大意

给你一个整数 n，请你返回 任意 一个由 n 个 各不相同 的整数组成的数组，并且这 n 个数相加和为 0 。

提示：

- 1 <= n <= 1000

解题思路

- 给出一个数 n，输出一个有 n 个数的数组，里面元素之和为 0 。
- 简单题，简单循环即可。

代码

```go
func sumZero(n int) []int {
  res, left, right, start := make([]int, n), 0, n-1, 1
  for left < right {
    res[left] = start
    res[right] = -start
    start++
    left = left + 1
    right = right - 1
  }
  return res
}
```

1305. All Elements in Two Binary Search Trees

题目

Given two binary search trees `root1` and `root2`.

Return a list containing *all the integers* from *both trees* sorted in **ascending** order.

Example 1:

```
Input: root1 = [2,1,4], root2 = [1,0,3]
Output: [0,1,1,2,3,4]
```

Example 2:

```
Input: root1 = [0,-10,10], root2 = [5,1,7,0,2]
Output: [-10,0,0,1,2,5,7,10]
```

Example 3:

```
Input: root1 = [], root2 = [5,1,7,0,2]
Output: [0,1,2,5,7]
```

Example 4:

```
Input: root1 = [0,-10,10], root2 = []
Output: [-10,0,10]
```

Example 5:

```
Input: root1 = [1,null,8], root2 = [8,1]
Output: [1,1,8,8]
```

Constraints:

- Each tree has at most `5000` nodes.
- Each node's value is between `[-10^5, 10^5]`.

题目大意

给你 root1 和 root2 这两棵二叉搜索树。请你返回一个列表，其中包含 两棵树 中的所有整数并按 升序 排序。.

提示：

- 每棵树最多有 5000 个节点。
- 每个节点的值在 [-10^5, 10^5] 之间。

解题思路

- 给出 2 棵二叉排序树，要求将 2 棵树所有节点的值按照升序排序。
- 这一题最暴力简单的方法就是把 2 棵树的节点都遍历出来，然后放在一个数组里面从小到大排序即可。这样做虽然能 AC，但是时间复杂度高。因为题目中给的二叉排序树这一条件没有用上。由于树中节点本来已经有序了，所以题目实质想要我们合并 2 个有序数组。利用中根遍历，把 2 个二叉排序树的所有节点值都遍历出

来，遍历出来以后就是有序的。接下来再合并这两个有序数组即可。合并 2 个有序数组是第 88 题。

代码

```
// 解法一 合并排序
func getAllElements(root1 *TreeNode, root2 *TreeNode) []int {
  arr1 := inorderTraversal(root1)
  arr2 := inorderTraversal(root2)
  arr1 = append(arr1, make([]int, len(arr2))...)
  merge(arr1, len(arr1)-len(arr2), arr2, len(arr2))
  return arr1
}

// 解法二 暴力遍历排序，时间复杂度高
func getAllElements1(root1 *TreeNode, root2 *TreeNode) []int {
  arr := []int{}
  arr = append(arr, preorderTraversal(root1)...)
  arr = append(arr, preorderTraversal(root2)...)
  sort.Ints(arr)
  return arr
}
```

1306. Jump Game III

题目

Given an array of non-negative integers `arr`, you are initially positioned at `start` index of the array. When you are at index `i`, you can jump to `i + arr[i]` or `i - arr[i]`, check if you can reach to **any** index with value 0.

Notice that you can not jump outside of the array at any time.

Example 1:

```
Input: arr = [4,2,3,0,3,1,2], start = 5
Output: true
Explanation:
All possible ways to reach at index 3 with value 0 are:
index 5 -> index 4 -> index 1 -> index 3
index 5 -> index 6 -> index 4 -> index 1 -> index 3
```

Example 2:

```
Input: arr = [4,2,3,0,3,1,2], start = 0
Output: true
Explanation:
One possible way to reach at index 3 with value 0 is:
index 0 -> index 4 -> index 1 -> index 3
```

Example 3:

```
Input: arr = [3,0,2,1,2], start = 2
Output: false
Explanation: There is no way to reach at index 1 with value 0.
```

Constraints:

- `1 <= arr.length <= 5 * 10^4`
- `0 <= arr[i] < arr.length`
- `0 <= start < arr.length`

题目大意

这里有一个非负整数数组 arr，你最开始位于该数组的起始下标 start 处。当你位于下标 i 处时，你可以跳到 i + arr[i] 或者 i - arr[i]。请你判断自己是否能够跳到对应元素值为 0 的 任一 下标处。注意，不管是什么情况下，你都无法跳到数组之外。

提示：

- 1 <= arr.length <= 5 * 10^4
- 0 <= arr[i] < arr.length
- 0 <= start < arr.length

解题思路

- 给出一个非负数组和一个起始下标 `start`。站在 `start`，每次可以跳到 `i + arr[i]` 或者 `i - arr[i]`。要求判断能否跳到元素值为 0 的下标处。
- 这一题考察的是递归。每一步都需要判断 3 种可能：
  1. 当前是否站在元素值为 0 的目标点上。
  2. 往前跳 arr[start]，是否能站在元素值为 0 的目标点上。
  3. 往后跳 arr[start]，是否能站在元素值为 0 的目标点上。

  第 2 种可能和第 3 种可能递归即可，每一步都判断这 3 种可能是否有一种能跳到元素值为 0 的下标处。
- `arr[start] += len(arr)` 这一步仅仅只是为了标记此下标已经用过了，下次判断的时候该下标会被 `if` 条件筛选掉。

代码

```go
func canReach(arr []int, start int) bool {
  if start >= 0 && start < len(arr) && arr[start] < len(arr) {
    jump := arr[start]
    arr[start] += len(arr)
    return jump == 0 || canReach(arr, start+jump) || canReach(arr, start-jump)
  }
  return false
}
```

1310. XOR Queries of a Subarray

题目

Given the array `arr` of positive integers and the array `queries` where `queries[i] = [Li,Ri]`, for each query `i` compute the **XOR** of elements from `Li` to `Ri` (that is, `arr[Li]xor arr[Li+1]xor ...xor arr[Ri]`). Return an array containing the result for the given `queries`.

Example 1:

```
Input: arr = [1,3,4,8], queries = [[0,1],[1,2],[0,3],[3,3]]
Output: [2,7,14,8]
Explanation:
The binary representation of the elements in the array are:
1 = 0001
3 = 0011
4 = 0100
8 = 1000
The XOR values for queries are:
[0,1] = 1 xor 3 = 2
[1,2] = 3 xor 4 = 7
[0,3] = 1 xor 3 xor 4 xor 8 = 14
[3,3] = 8
```

Example 2:

```
Input: arr = [4,8,2,10], queries = [[2,3],[1,3],[0,0],[0,3]]
Output: [8,0,4,4]
```

Constraints:

- `1 <= arr.length <= 3 * 10^4`
- `1 <= arr[i] <= 10^9`
- `1 <= queries.length <= 3 * 10^4`
- `queries[i].length == 2`

- `0 <= queries[i][0] <= queries[i][1] < arr.length`

题目大意

有一个正整数数组 arr，现给你一个对应的查询数组 queries，其中 queries[i] = [Li, Ri]。对于每个查询 i，请你计算从 Li 到 Ri 的 XOR 值（即 arr[Li] xor arr[Li+1] xor ... xor arr[Ri]）作为本次查询的结果。并返回一个包含给定查询 queries 所有结果的数组。

解题思路

- 此题求区间异或，很容易让人联想到区间求和。区间求和利用前缀和，可以使得 query 从 O(n) 降为 O(1)。区间异或能否也用类似前缀和的思想呢？答案是肯定的。利用异或的两个性质，x ^ x = 0，x ^ 0 = x。那么有：（由于 LaTeX 中异或符号 ^ 是特殊字符，笔者用 {{< katex >}} \oplus {{< /katex >}} 代替异或）

  {{< katex display >}}
  \begin{aligned}Query(left,right) &=arr[left] \oplus \cdots \oplus arr[right]\&=(arr[0] \oplus \cdots \oplus arr[left-1]) \oplus (arr[0] \oplus \cdots \oplus arr[left-1]) \oplus (arr[left] \oplus \cdots \oplus arr[right])\ &=(arr[0] \oplus \cdots \oplus arr[left-1]) \oplus (arr[0] \oplus \cdots \oplus arr[right])\ &=xors[left] \oplus xors[right+1]\ \end{aligned}
  {{< /katex >}}

  按照这个思路解题，便可以将 query 从 O(n) 降为 O(1)，总的时间复杂度为 O(n)。

代码

```go
package leetcode

func xorQueries(arr []int, queries [][]int) []int {
  xors := make([]int, len(arr))
  xors[0] = arr[0]
  for i := 1; i < len(arr); i++ {
    xors[i] = arr[i] ^ xors[i-1]
  }
  res := make([]int, len(queries))
  for i, q := range queries {
    res[i] = xors[q[1]]
    if q[0] > 0 {
      res[i] ^= xors[q[0]-1]
    }
  }
  return res
}
```

1313. Decompress Run-Length Encoded List

题目

We are given a list `nums` of integers representing a list compressed with run-length encoding.

Consider each adjacent pair of elements `[freq, val] = [nums[2*i], nums[2*i+1]]` (with `i >= 0`). For each such pair, there are `freq` elements with value `val` concatenated in a sublist. Concatenate all the sublists from left to right to generate the decompressed list.

Return the decompressed list.

Example 1:

```
Input: nums = [1,2,3,4]
Output: [2,4,4,4]
Explanation: The first pair [1,2] means we have freq = 1 and val = 2 so we generate the array [2].
The second pair [3,4] means we have freq = 3 and val = 4 so we generate [4,4,4].
At the end the concatenation [2] + [4,4,4] is [2,4,4,4].
```

Example 2:

```
Input: nums = [1,1,2,3]
Output: [1,3,3]
```

Constraints:

- `2 <= nums.length <= 100`
- `nums.length % 2 == 0`
- `1 <= nums[i] <= 100`

题目大意

给你一个以行程长度编码压缩的整数列表 nums 。考虑每对相邻的两个元素 [freq, val] = [nums[2*i], nums[2*i+1]] （其中 i >= 0 ），每一对都表示解压后子列表中有 freq 个值为 val 的元素，你需要从左到右连接所有子列表以生成解压后的列表。请你返回解压后的列表。

解题思路

- 给定一个带编码长度的数组，要求解压这个数组。
- 简单题。按照题目要求，下标从 0 开始，奇数位下标为前一个下标对应元素重复次数，那么就把这个元素 append 几次。最终输出解压后的数组即可。

代码

```go
package leetcode

func decompressRLElist(nums []int) []int {
  res := []int{}
  for i := 0; i < len(nums); i += 2 {
    for j := 0; j < nums[i]; j++ {
      res = append(res, nums[i+1])
    }
  }
  return res
}
```

1317. Convert Integer to the Sum of Two No-Zero Integers

题目

Given an integer `n`. No-Zero integer is a positive integer which **doesn't contain any 0** in its decimal representation.

Return *a list of two integers* `[A, B]` where:

- `A` and `B` are No-Zero integers.
- `A + B = n`

It's guarateed that there is at least one valid solution. If there are many valid solutions you can return any of them.

Example 1:

```
Input: n = 2
Output: [1,1]
Explanation: A = 1, B = 1. A + B = n and both A and B don't contain any 0 in their
decimal representation.
```

Example 2:

```
Input: n = 11
Output: [2,9]
```

Example 3:

```
Input: n = 10000
Output: [1,9999]
```

Example 4:

```
Input: n = 69
Output: [1,68]
```

Example 5:

```
Input: n = 1010
Output: [11,999]
```

Constraints:

- `2 <= n <= 10^4`

题目大意

「无零整数」是十进制表示中 不含任何 0 的正整数。给你一个整数 n，请你返回一个 由两个整数组成的列表 [A, B]，满足：

- A 和 B 都是无零整数
- A + B = n

题目数据保证至少有一个有效的解决方案。如果存在多个有效解决方案，你可以返回其中任意一个。

解题思路

- 给定一个整数 n，要求把它分解为 2 个十进制位中不含 0 的正整数且这两个正整数之和为 n。
- 简单题。在 [1, n/2] 区间内搜索，只要有一组满足条件的解就 break。题目保证了至少有一组解，并且多组解返回任意一组即可。

代码

```go
package leetcode

func getNoZeroIntegers(n int) []int {
  noZeroPair := []int{}
  for i := 1; i <= n/2; i++ {
    if isNoZero(i) && isNoZero(n-i) {
      noZeroPair = append(noZeroPair, []int{i, n - i}...)
      break
    }
  }
  return noZeroPair
}

func isNoZero(n int) bool {
  for n != 0 {
    if n%10 == 0 {
```

```
        return false
      }
      n /= 10
    }
    return true
  }
```

1319. Number of Operations to Make Network Connected

题目

There are `n` computers numbered from `0` to `n-1` connected by ethernet cables `connections` forming a network where `connections[i] = [a, b]` represents a connection between computers `a` and `b`. Any computer can reach any other computer directly or indirectly through the network.

Given an initial computer network `connections`. You can extract certain cables between two directly connected computers, and place them between any pair of disconnected computers to make them directly connected. Return the *minimum number of times* you need to do this in order to make all the computers connected. If it's not possible, return -1.

Example 1:

```
Input: n = 4, connections = [[0,1],[0,2],[1,2]]
Output: 1
Explanation: Remove cable between computer 1 and 2 and place between computers 1 and 3.
```

Example 2:

```
Input: n = 6, connections = [[0,1],[0,2],[0,3],[1,2],[1,3]]
Output: 2
```

Example 3:

```
Input: n = 6, connections = [[0,1],[0,2],[0,3],[1,2]]
Output: -1
Explanation: There are not enough cables.
```

Example 4:

```
Input: n = 5, connections = [[0,1],[0,2],[3,4],[2,3]]
Output: 0
```

Constraints:

- `1 <= n <= 10^5`
- `1 <= connections.length <= min(n*(n-1)/2, 10^5)`
- `connections[i].length == 2`
- `0 <= connections[i][0], connections[i][1] < n`
- `connections[i][0] != connections[i][1]`
- There are no repeated connections.
- No two computers are connected by more than one cable.

题目大意

用以太网线缆将 n 台计算机连接成一个网络，计算机的编号从 0 到 n-1。线缆用 connections 表示，其中 connections[i] = [a, b] 连接了计算机 a 和 b。网络中的任何一台计算机都可以通过网络直接或者间接访问同一个网络中其他任意一台计算机。给你这个计算机网络的初始布线 connections，你可以拔开任意两台直连计算机之间的线缆，并用它连接一对未直连的计算机。请你计算并返回使所有计算机都连通所需的最少操作次数。如果不可能，则返回 -1 。

解题思路

- 很明显这题的解题思路是并查集。先将每个 connections 构建出并查集。构建中需要累加冗余的连接。例如 2 个节点已经连通，再连接这个集合中的任意 2 个节点就算冗余连接。冗余连接的线都可以移动，去连接还没有连通的节点。计算出冗余连接数，再根据并查集的集合总数，即可得出答案。
- 这一题答案有 3 种可能。第一种，所有点都在一个集合内，即全部连通，这时输出 0 。第二种，冗余的连接不够串起所有点，这时输出 -1 。第三种情况是可以连通的情况。 m 个集合需要连通，最少需要 m - 1 条线。如果冗余连接数大于 m - 1，则输出 m - 1 即可。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/template"
)

func makeConnected(n int, connections [][]int) int {
  if n-1 > len(connections) {
    return -1
  }
  uf, redundance := template.UnionFind{}, 0
  uf.Init(n)
  for _, connection := range connections {
    if uf.Find(connection[0]) == uf.Find(connection[1]) {
      redundance++
    } else {
      uf.Union(connection[0], connection[1])
    }
  }
  if uf.TotalCount() == 1 || redundance < uf.TotalCount()-1 {
    return 0
```

```
  }
  return uf.TotalCount() - 1
}
```

1329. Sort the Matrix Diagonally

题目

A **matrix diagonal** is a diagonal line of cells starting from some cell in either the topmost row or leftmost column and going in the bottom-right direction until reaching the matrix's end. For example, the **matrix diagonal** starting from `mat[2][0]`, where `mat` is a `6 x 3` matrix, includes cells `mat[2][0]`, `mat[3][1]`, and `mat[4][2]`.

Given an `m x n` matrix `mat` of integers, sort each **matrix diagonal** in ascending order and return *the resulting matrix*.

Example 1:

```
Input: mat = [[3,3,1,1],[2,2,1,2],[1,1,1,2]]
Output: [[1,1,1,1],[1,2,2,2],[1,2,3,3]]
```

Constraints:

- `m == mat.length`
- `n == mat[i].length`
- `1 <= m, n <= 100`
- `1 <= mat[i][j] <= 100`

题目大意

给你一个 m * n 的整数矩阵 mat ，请你将同一条对角线上的元素（从左上到右下）按升序排序后，返回排好序的矩阵。

解题思路

- 这道题思路很简单。按照对角线，把每条对角线的元素读取出来放在数组中。这里可以利用 map 保存这些数组。再将这些数组排序。最后按照对角线还原矩阵即可。

代码

```go
package leetcode

func diagonalSort(mat [][]int) [][]int {
  m, n, diagonalsMap := len(mat), len(mat[0]), make(map[int][]int)
  for i := 0; i < m; i++ {
    for j := 0; j < n; j++ {
      diagonalsMap[i-j] = append(diagonalsMap[i-j], mat[i][j])
```

```
    }
  }
  for _, v := range diagonalsMap {
    sort.Ints(v)
  }
  for i := 0; i < m; i++ {
    for j := 0; j < n; j++ {
      mat[i][j] = diagonalsMap[i-j][0]
      diagonalsMap[i-j] = diagonalsMap[i-j][1:]
    }
  }
  return mat
}
```

1332. Remove Palindromic Subsequences

题目

Given a string `s` consisting only of letters `'a'` and `'b'`. In a single step you can remove one palindromic **subsequence** from `s`.

Return the minimum number of steps to make the given string empty.

A string is a subsequence of a given string, if it is generated by deleting some characters of a given string without changing its order.

A string is called palindrome if is one that reads the same backward as well as forward.

Example 1:

```
Input: s = "ababa"
Output: 1
Explanation: String is already palindrome
```

Example 2:

```
Input: s = "abb"
Output: 2
Explanation: "abb" -> "bb" -> "".
Remove palindromic subsequence "a" then "bb".
```

Example 3:

```
Input: s = "baabb"
Output: 2
Explanation: "baabb" -> "b" -> "".
Remove palindromic subsequence "baab" then "b".
```

Example 4:

```
Input: s = ""
Output: 0
```

Constraints:

- `0 <= s.length <= 1000`
- `s` only consists of letters 'a' and 'b'

题目大意

给你一个字符串 s，它仅由字母 'a' 和 'b' 组成。每一次删除操作都可以从 s 中删除一个回文 子序列。返回删除给定字符串中所有字符（字符串为空）的最小删除次数。

「子序列」定义：如果一个字符串可以通过删除原字符串某些字符而不改变原字符顺序得到，那么这个字符串就是原字符串的一个子序列。

「回文」定义：如果一个字符串向后和向前读是一致的，那么这个字符串就是一个回文。

解题思路

- 笔者读完题以为是第 5 题的加强版。在字符串中每次都找到最长的回文子串删除，一直删除到找不到回文子串结束，删除的总次数 + 剩余字母数 = 最小删除次数。提交以后 `wrong answer` 了，在 `bbaabaaa` 这组测试用例出错了。如果按照找最长回文字符串的思路，先找到最长回文子串 `aabaa`，剩余 `bba`，还需要再删除 2 次，`bb` 和 `a`。总共删除次数是 3 。为什么出错误了呢？仔细再读题，题目中说的是子序列，这不是连续的，再加上这道题是 `easy` 难度，其实很简单。
- 这道题的答案只可能是 0，1，2 。空串对应的 0 。如果有一个字母，单个字母可以构成回文，所以是 1，如果字符串长度大于等于 2，即 `a` 和 `b` 都有，第一步先删除所有 `a`，因为所有的 `a` 构成了回文子序列。第二步删除所有的 `b`，因为所有的 `b` 构成了回文子序列。经过这样两步，一定能删除所有字符。

代码

```go
package leetcode

func removePalindromeSub(s string) int {
  if len(s) == 0 {
    return 0
  }
  for i := 0; i < len(s)/2; i++ {
    if s[i] != s[len(s)-1-i] {
      return 2
    }
  }
  return 1
}
```

1337. The K Weakest Rows in a Matrix

题目

Given a `m * n` matrix `mat` of *ones* (representing soldiers) and *zeros* (representing civilians), return the indexes of the `k` weakest rows in the matrix ordered from the weakest to the strongest.

A row ***i*** is weaker than row ***j***, if the number of soldiers in row ***i*** is less than the number of soldiers in row ***j***, or they have the same number of soldiers but ***i*** is less than ***j***. Soldiers are **always** stand in the frontier of a row, that is, always *ones* may appear first and then *zeros*.

Example 1:

```
Input: mat =
[[1,1,0,0,0],
 [1,1,1,1,0],
 [1,0,0,0,0],
 [1,1,0,0,0],
 [1,1,1,1,1]],
k = 3
Output: [2,0,3]
Explanation:
The number of soldiers for each row is:
row 0 -> 2
row 1 -> 4
row 2 -> 1
row 3 -> 2
row 4 -> 5
Rows ordered from the weakest to the strongest are [2,0,3,1,4]
```

Example 2:

```
Input: mat =
[[1,0,0,0],
 [1,1,1,1],
 [1,0,0,0],
 [1,0,0,0]],
k = 2
Output: [0,2]
Explanation:
The number of soldiers for each row is:
row 0 -> 1
row 1 -> 4
row 2 -> 1
row 3 -> 1
Rows ordered from the weakest to the strongest are [0,2,3,1]
```

Constraints:

- `m == mat.length`
- `n == mat[i].length`
- `2 <= n, m <= 100`
- `1 <= k <= m`
- `matrix[i][j]` is either 0 **or** 1.

题目大意

给你一个大小为 m * n 的矩阵 mat，矩阵由若干军人和平民组成，分别用 1 和 0 表示。请你返回矩阵中战斗力最弱的 k 行的索引，按从最弱到最强排序。如果第 i 行的军人数量少于第 j 行，或者两行军人数量相同但 i 小于 j，那么我们认为第 i 行的战斗力比第 j 行弱。军人 总是 排在一行中的靠前位置，也就是说 1 总是出现在 0 之前。

解题思路

- 简单题。第一个能想到的解题思路是，先统计每一行 1 的个数，然后将结果进行排序，按照 1 的个数从小到大排序，如果 1 的个数相同，再按照行号从小到大排序。排好序的数组取出前 K 位即为答案。
- 此题还有第二种解法。在第一种解法中，并没有用到题目中“军人 总是 排在一行中的靠前位置，也就是说 1 总是出现在 0 之前。”这一条件。由于有了这个条件，使得如果按照列去遍历，最先出现 0 的行，则是最弱的行。行号小的先被遍历到，所以相同数量 1 的行，行号小的会排在前面。最后记得再添加上全 1 的行。同样，最终输出取出前 K 位即为答案。此题解法二才是最优雅最高效的解法。

代码

```go
package leetcode

func kWeakestRows(mat [][]int, k int) []int {
  res := []int{}
  for j := 0; j < len(mat[0]); j++ {
    for i := 0; i < len(mat); i++ {
      if mat[i][j] == 0 && ((j == 0) || (mat[i][j-1] != 0)) {
        res = append(res, i)
      }
    }
  }
  for i := 0; i < len(mat); i++ {
    if mat[i][len(mat[0])-1] == 1 {
      res = append(res, i)
    }
  }
  return res[:k]
}
```

1353. Maximum Number of Events That Can Be Attended

题目

Given an array of `events` where `events[i] = [startDayi, endDayi]`. Every event `i` starts at `startDayi` and ends at `endDayi`.

You can attend an event `i` at any day `d` where `startTimei <= d <= endTimei`. Notice that you can only attend one event at any time `d`.

Return *the maximum number of events* you can attend.

Example 1:

```
Input: events = [[1,2],[2,3],[3,4]]
Output: 3
Explanation: You can attend all the three events.
One way to attend them all is as shown.
Attend the first event on day 1.
Attend the second event on day 2.
Attend the third event on day 3.
```

Example 2:

```
Input: events= [[1,2],[2,3],[3,4],[1,2]]
Output: 4
```

Example 3:

```
Input: events = [[1,4],[4,4],[2,2],[3,4],[1,1]]
Output: 4
```

Example 4:

```
Input: events = [[1,100000]]
Output: 1
```

Example 5:

```
Input: events = [[1,1],[1,2],[1,3],[1,4],[1,5],[1,6],[1,7]]
Output: 7
```

Constraints:

- `1 <= events.length <= 10^5`
- `events[i].length == 2`
- `1 <= startDayi <= endDayi <= 10^5`

题目大意

给你一个数组 events，其中 events[i] = [startDayi, endDayi] ，表示会议 i 开始于 startDayi ，结束于 endDayi 。你可以在满足 startDayi <= d <= endDayi 中的任意一天 d 参加会议 i 。注意，一天只能参加一个会议。请你返回你可以参加的 最大 会议数目。

解题思路

- 关于会议安排，活动安排这类题，第一直觉是贪心问题。先按照会议开始时间从小到大排序，如果开始时间相同，再按照结束时间从小到大排序。贪心策略是，优先选择参加早结束的会议。因为一个结束时间晚的会议，代表这个会议持续时间长，先参加马上要结束的会议，这样可以参加更多的会议。
- 注意题目给的数据代表的是天数。比较大小的时候最好转换成坐标轴上的坐标点。例如 [1,2] 代表这个会议持续 2 天，如果在坐标轴上表示，是 [0,2]，0-1 表示第一天，1-2 表示第二天。所以比较会议时需要把开始时间减一。选定了这个会议以后记得要把这一天排除，例如选择了第二天，那么下次对比起始时间需要从坐标 2 开始，因为第二天的时间范围是 1-2，所以下一轮比较会议前需要把开始时间加一。从左往右依次扫描各个会议时间段，选择结束时间大于起始时间的会议，不断累加次数，扫描完所有会议，最终结果即为可参加的最大会议数。
- 测试数据中有一组很恶心的数据，见 test 文件中最后一组数据。这组数据在同一天叠加了多个会议，并且起始时间完全一致。这种特殊情况需要加判断条件排除，见下面代码 continue 条件。

代码

```go
package leetcode

import (
  "sort"
)

func maxEvents(events [][]int) int {
  sort.Slice(events, func(i, j int) bool {
    if events[i][0] == events[j][0] {
      return events[i][1] < events[j][1]
    }
    return events[i][0] < events[j][0]
  })
  attended, current := 1, events[0]
  for i := 1; i < len(events); i++ {
    prev, event := events[i-1], events[i]
    if event[0] == prev[0] && event[1] == prev[1] && event[1] == event[0] {
      continue
    }
    start, end := max(current[0], event[0]-1), max(current[1], event[1])
    if end-start > 0 {
      current[0] = start + 1
```

```
      current[1] = end
      attended++
    }
  }
  return attended
}

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

1380. Lucky Numbers in a Matrix

题目

Given a `m * n` matrix of **distinct** numbers, return all lucky numbers in the matrix in **any** order.

A lucky number is an element of the matrix such that it is the minimum element in its row and maximum in its column.

Example 1:

```
Input: matrix = [[3,7,8],[9,11,13],[15,16,17]]
Output: [15]
Explanation: 15 is the only lucky number since it is the minimum in its row and the maximum in its column
```

Example 2:

```
Input: matrix = [[1,10,4,2],[9,3,8,7],[15,16,17,12]]
Output: [12]
Explanation: 12 is the only lucky number since it is the minimum in its row and the maximum in its column.
```

Example 3:

```
Input: matrix = [[7,8],[1,2]]
Output: [7]
```

Constraints:

- `m == mat.length`
- `n == mat[i].length`
- `1 <= n, m <= 50`

- `1 <= matrix[i][j] <= 10^5`.
- All elements in the matrix are distinct.

题目大意

给你一个 m * n 的矩阵，矩阵中的数字 各不相同 。请你按 任意 顺序返回矩阵中的所有幸运数。幸运数是指矩阵中满足同时下列两个条件的元素：

- 在同一行的所有元素中最小
- 在同一列的所有元素中最大

解题思路

- 找出矩阵中的幸运数。幸运数的定义：同时满足 2 个条件，在同一行的所有元素中最小并且在同一列的所有元素中最大。
- 简单题。按照题意遍历矩阵，找到同时满足 2 个条件的数输出即可。

代码

```go
package leetcode

func luckyNumbers(matrix [][]int) []int {
  t, r, res := make([]int, len(matrix[0])), make([]int, len(matrix[0])), []int{}
  for _, val := range matrix {
    m, k := val[0], 0
    for j := 0; j < len(matrix[0]); j++ {
      if val[j] < m {
        m = val[j]
        k = j
      }
      if t[j] < val[j] {
        t[j] = val[j]
      }
    }

    if t[k] == m {
      r[k] = m
    }
  }
  for k, v := range r {
    if v > 0 && v == t[k] {
      res = append(res, v)
    }
  }
  return res
}
```

1383. Maximum Performance of a Team

题目

You are given two integers `n` and `k` and two integer arrays `speed` and `efficiency` both of length `n`. There are `n` engineers numbered from `1` to `n`. `speed[i]` and `efficiency[i]` represent the speed and efficiency of the `ith` engineer respectively.

Choose **at most** `k` different engineers out of the `n` engineers to form a team with the maximum **performance**.

The performance of a team is the sum of their engineers' speeds multiplied by the minimum efficiency among their engineers.

Return *the maximum performance of this team*. Since the answer can be a huge number, return it **modulo** `109 + 7`.

Example 1:

```
Input: n = 6, speed = [2,10,3,1,5,8], efficiency = [5,4,3,9,7,2], k = 2
Output: 60
Explanation:
We have the maximum performance of the team by selecting engineer 2 (with speed=10 and
efficiency=4) and engineer 5 (with speed=5 and efficiency=7). That is, performance =
(10 + 5) * min(4, 7) = 60.
```

Example 2:

```
Input: n = 6, speed = [2,10,3,1,5,8], efficiency = [5,4,3,9,7,2], k = 3
Output: 68
Explanation:
This is the same example as the first but k = 3. We can select engineer 1, engineer 2
and engineer 5 to get the maximum performance of the team. That is, performance = (2 +
10 + 5) * min(5, 4, 7) = 68.
```

Example 3:

```
Input: n = 6, speed = [2,10,3,1,5,8], efficiency = [5,4,3,9,7,2], k = 4
Output: 72
```

Constraints:

- `1 <= <= k <= n <= 105`
- `speed.length == n`
- `efficiency.length == n`
- `1 <= speed[i] <= 105`

- `1 <= efficiency[i] <= 108`

题目大意

公司有编号为 1 到 n 的 n 个工程师，给你两个数组 speed 和 efficiency ，其中 speed[i] 和 efficiency[i] 分别代表第 i 位工程师的速度和效率。请你返回由最多 k 个工程师组成的 最大团队表现值 ，由于答案可能很大，请你返回结果对 10^9 + 7 取余后的结果。团队表现值 的定义为：一个团队中「所有工程师速度的和」乘以他们「效率值中的最小值」。

解题思路

- 题目要求返回最大团队表现值，表现值需要考虑速度的累加和，和效率的最小值。即使速度快，效率的最小值很小，总的表现值还是很小。先将效率从大到小排序。从效率高的工程师开始选起，遍历过程中维护一个大小为 k 的速度最小堆。每次遍历都计算一次团队最大表现值。扫描完成，最大团队表现值也筛选出来了。具体实现见下面的代码。

代码

```go
package leetcode

import (
  "container/heap"
  "sort"
)

func maxPerformance(n int, speed []int, efficiency []int, k int) int {
  indexes := make([]int, n)
  for i := range indexes {
    indexes[i] = i
  }
  sort.Slice(indexes, func(i, j int) bool {
    return efficiency[indexes[i]] > efficiency[indexes[j]]
  })
  ph := speedHeap{}
  heap.Init(&ph)
  speedSum := 0
  var max int64
  for _, index := range indexes {
    if ph.Len() == k {
      speedSum -= heap.Pop(&ph).(int)
    }
    speedSum += speed[index]
    heap.Push(&ph, speed[index])
    max = Max(max, int64(speedSum)*int64(efficiency[index]))
  }
  return int(max % (1e9 + 7))
}
```

```go
type speedHeap []int

func (h speedHeap) Less(i, j int) bool  { return h[i] < h[j] }
func (h speedHeap) Swap(i, j int)       { h[i], h[j] = h[j], h[i] }
func (h speedHeap) Len() int            { return len(h) }
func (h *speedHeap) Push(x interface{}) { *h = append(*h, x.(int)) }
func (h *speedHeap) Pop() interface{} {
  res := (*h)[len(*h)-1]
  *h = (*h)[:h.Len()-1]
  return res
}

func Max(a, b int64) int64 {
  if a > b {
    return a
  }
  return b
}
```

1385. Find the Distance Value Between Two Arrays

题目

Given two integer arrays `arr1` and `arr2`, and the integer `d`, *return the distance value between the two arrays*.

The distance value is defined as the number of elements `arr1[i]` such that there is not any element `arr2[j]` where `|arr1[i]-arr2[j]| <= d`.

Example 1:

```
Input: arr1 = [4,5,8], arr2 = [10,9,1,8], d = 2
Output: 2
Explanation:
For arr1[0]=4 we have:
|4-10|=6 > d=2
|4-9|=5 > d=2
|4-1|=3 > d=2
|4-8|=4 > d=2
For arr1[1]=5 we have:
|5-10|=5 > d=2
|5-9|=4 > d=2
|5-1|=4 > d=2
|5-8|=3 > d=2
For arr1[2]=8 we have:
|8-10|=2 <= d=2
|8-9|=1 <= d=2
```

```
|8-1|=7 > d=2
|8-8|=0 <= d=2
```

Example 2:

```
Input: arr1 = [1,4,2,3], arr2 = [-4,-3,6,10,20,30], d = 3
Output: 2
```

Example 3:

```
Input: arr1 = [2,1,100,3], arr2 = [-5,-2,10,-3,7], d = 6
Output: 1
```

Constraints:

- `1 <= arr1.length, arr2.length <= 500`
- `-10^3 <= arr1[i], arr2[j] <= 10^3`
- `0 <= d <= 100`

题目大意

给你两个整数数组 arr1 ， arr2 和一个整数 d ，请你返回两个数组之间的 距离值 。「距离值」 定义为符合此距离要求的元素数目：对于元素 arr1[i] ，不存在任何元素 arr2[j] 满足 |arr1[i]-arr2[j]| <= d 。

提示：

- 1 <= arr1.length, arr2.length <= 500
- -10^3 <= arr1[i], arr2[j] <= 10^3
- 0 <= d <= 100

解题思路

- 计算两个数组之间的距离，距离值的定义：满足对于元素 arr1[i] ，不存在任何元素 arr2[j] 满足 |arr1[i]-arr2[j]| <= d 这一条件的元素数目。
- 简单题，按照距离值的定义，双层循环计数即可。

代码

```go
package leetcode

func findTheDistanceValue(arr1 []int, arr2 []int, d int) int {
  res := 0
  for i := range arr1 {
    for j := range arr2 {
      if abs(arr1[i]-arr2[j]) <= d {
        break
      }
```

```
      if j == len(arr2)-1 {
        res++
      }
    }
  }
  return res
}

func abs(a int) int {
  if a < 0 {
    return -1 * a
  }
  return a
}
```

1389. Create Target Array in the Given Order

题目

Given two arrays of integers `nums` and `index`. Your task is to create *target* array under the following rules:

- Initially *target* array is empty.
- From left to right read nums[i] and index[i], insert at index `index[i]` the value `nums[i]` in *target* array.
- Repeat the previous step until there are no elements to read in `nums` and `index.`

Return the *target* array.

It is guaranteed that the insertion operations will be valid.

Example 1:

```
Input: nums = [0,1,2,3,4], index = [0,1,2,2,1]
Output: [0,4,1,3,2]
Explanation:
nums       index     target
0            0        [0]
1            1        [0,1]
2            2        [0,1,2]
3            2        [0,1,3,2]
4            1        [0,4,1,3,2]
```

Example 2:

```
Input: nums = [1,2,3,4,0], index = [0,1,2,3,0]
Output: [0,1,2,3,4]
Explanation:
nums       index     target
1            0        [1]
2            1        [1,2]
3            2        [1,2,3]
4            3        [1,2,3,4]
0            0        [0,1,2,3,4]
```

Example 3:

```
Input: nums = [1], index = [0]
Output: [1]
```

Constraints:

- `1 <= nums.length, index.length <= 100`
- `nums.length == index.length`
- `0 <= nums[i] <= 100`
- `0 <= index[i] <= i`

题目大意

给你两个整数数组 nums 和 index。你需要按照以下规则创建目标数组：

- 目标数组 target 最初为空。
- 按从左到右的顺序依次读取 nums[i] 和 index[i]，在 target 数组中的下标 index[i] 处插入值 nums[i] 。
- 重复上一步，直到在 nums 和 index 中都没有要读取的元素。

请你返回目标数组。题目保证数字插入位置总是存在。

解题思路

- 给定 2 个数组，分别装的是待插入的元素和待插入的位置。最后输出操作完成的数组。
- 简单题，按照题意插入元素即可。

代码

```go
package leetcode

func createTargetArray(nums []int, index []int) []int {
  result := make([]int, len(nums))
  for i, pos := range index {
    copy(result[pos+1:i+1], result[pos:i])
    result[pos] = nums[i]
  }
  return result
}
```

1396. Design Underground System

题目

Implement the `UndergroundSystem` class:

- `void checkIn(int id, string stationName, int t)`
  - A customer with a card id equal to `id`, gets in the station `stationName` at time `t`.
  - A customer can only be checked into one place at a time.
- `void checkOut(int id, string stationName, int t)`
  - A customer with a card id equal to `id`, gets out from the station `stationName` at time `t`.
- `double getAverageTime(string startStation, string endStation)`
  - Returns the average time to travel between the `startStation` and the `endStation`.
  - The average time is computed from all the previous traveling from `startStation` to `endStation` that happened **directly**.
  - Call to `getAverageTime` is always valid.

You can assume all calls to `checkIn` and `checkOut` methods are consistent. If a customer gets in at time **t1** at some station, they get out at time **t2** with **t2 > t1**. All events happen in chronological order.

Example 1:

```
Input
["UndergroundSystem","checkIn","checkIn","checkIn","checkOut","checkOut","checkOut","getAverageTime","getAverageTime","checkIn","getAverageTime","checkOut","getAverageTime"]
[[],[45,"Leyton",3],[32,"Paradise",8],[27,"Leyton",10],[45,"Waterloo",15],[27,"Waterloo",20],[32,"Cambridge",22],["Paradise","Cambridge"],["Leyton","Waterloo"],[10,"Leyton",24],["Leyton","Waterloo"],[10,"Waterloo",38],["Leyton","Waterloo"]]

Output
[null,null,null,null,null,null,null,14.00000,11.00000,null,11.00000,null,12.00000]

Explanation
```

```
UndergroundSystem undergroundSystem = new UndergroundSystem();
undergroundSystem.checkIn(45, "Leyton", 3);
undergroundSystem.checkIn(32, "Paradise", 8);
undergroundSystem.checkIn(27, "Leyton", 10);
undergroundSystem.checkOut(45, "Waterloo", 15);
undergroundSystem.checkOut(27, "Waterloo", 20);
undergroundSystem.checkOut(32, "Cambridge", 22);
undergroundSystem.getAverageTime("Paradise", "Cambridge");       // return 14.00000.
There was only one travel from "Paradise" (at time 8) to "Cambridge" (at time 22)
undergroundSystem.getAverageTime("Leyton", "Waterloo");          // return 11.00000.
There were two travels from "Leyton" to "Waterloo", a customer with id=45 from time=3
to time=15 and a customer with id=27 from time=10 to time=20. So the average time is (
(15-3) + (20-10) ) / 2 = 11.00000
undergroundSystem.checkIn(10, "Leyton", 24);
undergroundSystem.getAverageTime("Leyton", "Waterloo");          // return 11.00000
undergroundSystem.checkOut(10, "Waterloo", 38);
undergroundSystem.getAverageTime("Leyton", "Waterloo");          // return 12.00000
```

Example 2:

```
Input
["UndergroundSystem","checkIn","checkOut","getAverageTime","checkIn","checkOut","getAve
rageTime","checkIn","checkOut","getAverageTime"]
[[],[10,"Leyton",3],[10,"Paradise",8],["Leyton","Paradise"],[5,"Leyton",10],
[5,"Paradise",16],["Leyton","Paradise"],[2,"Leyton",21],[2,"Paradise",30],
["Leyton","Paradise"]]

Output
[null,null,null,5.00000,null,null,5.50000,null,null,6.66667]

Explanation
UndergroundSystem undergroundSystem = new UndergroundSystem();
undergroundSystem.checkIn(10, "Leyton", 3);
undergroundSystem.checkOut(10, "Paradise", 8);
undergroundSystem.getAverageTime("Leyton", "Paradise"); // return 5.00000
undergroundSystem.checkIn(5, "Leyton", 10);
undergroundSystem.checkOut(5, "Paradise", 16);
undergroundSystem.getAverageTime("Leyton", "Paradise"); // return 5.50000
undergroundSystem.checkIn(2, "Leyton", 21);
undergroundSystem.checkOut(2, "Paradise", 30);
undergroundSystem.getAverageTime("Leyton", "Paradise"); // return 6.66667
```

Constraints:

- There will be at most `20000` operations.
- `1 <= id, t <= 106`
- All strings consist of uppercase and lowercase English letters, and digits.
- `1 <= stationName.length <= 10`

- Answers within `105` of the actual value will be accepted as correct.

题目大意

请你实现一个类 UndergroundSystem ， 它支持以下 3 种方法：

- 1. checkIn(int id, string stationName, int t)
    - 编号为 id 的乘客在 t 时刻进入地铁站 stationName 。
    - 一个乘客在同一时间只能在一个地铁站进入或者离开。
- 2. checkOut(int id, string stationName, int t)
    - 编号为 id 的乘客在 t 时刻离开地铁站 stationName 。
- 3. getAverageTime(string startStation, string endStation)
    - 返回从地铁站 startStation 到地铁站 endStation 的平均花费时间。
    - 平均时间计算的行程包括当前为止所有从 startStation 直接到达 endStation 的行程。
    - 调用 getAverageTime 时，询问的路线至少包含一趟行程。

你可以假设所有对 checkIn 和 checkOut 的调用都是符合逻辑的。也就是说，如果一个顾客在 t1 时刻到达某个地铁站，那么他离开的时间 t2 一定满足 t2 > t1 。所有的事件都按时间顺序给出。

解题思路

- 维护 2 个 `map`。一个 `mapA` 内部存储的是乘客 `id` 与（入站时间，站名）的对应关系。另外一个 `mapB` 存储的是起点站与终点站花费总时间与人数总数的关系。每当有人 `checkin()`，就更新 `mapA` 中的信息。每当有人 `checkout()`，就更新 `mapB` 中的信息，并删除 `mapA` 对应乘客 `id` 的键值对。最后调用 `getAverageTime()` 函数的时候根据 `mapB` 中存储的信息计算即可。

代码

```
package leetcode

type checkin struct {
  station string
  time    int
}

type stationTime struct {
  sum, count float64
}

type UndergroundSystem struct {
  checkins     map[int]*checkin
  stationTimes map[string]map[string]*stationTime
}

func Constructor() UndergroundSystem {
  return UndergroundSystem{
    make(map[int]*checkin),
    make(map[string]map[string]*stationTime),
  }
```

```go
}

func (s *UndergroundSystem) CheckIn(id int, stationName string, t int) {
  s.checkins[id] = &checkin{stationName, t}
}

func (s *UndergroundSystem) CheckOut(id int, stationName string, t int) {
  checkin := s.checkins[id]
  destination := s.stationTimes[checkin.station]
  if destination == nil {
    s.stationTimes[checkin.station] = make(map[string]*stationTime)
  }
  st := s.stationTimes[checkin.station][stationName]
  if st == nil {
    st = new(stationTime)
    s.stationTimes[checkin.station][stationName] = st
  }
  st.sum += float64(t - checkin.time)
  st.count++
  delete(s.checkins, id)
}

func (s *UndergroundSystem) GetAverageTime(startStation string, endStation string)
float64 {
  st := s.stationTimes[startStation][endStation]
  return st.sum / st.count
}

/**
 * Your UndergroundSystem object will be instantiated and called as such:
 * obj := Constructor();
 * obj.CheckIn(id,stationName,t);
 * obj.CheckOut(id,stationName,t);
 * param_3 := obj.GetAverageTime(startStation,endStation);
 */
```

1423. Maximum Points You Can Obtain from Cards

题目

There are several cards **arranged in a row**, and each card has an associated number of points The points are given in the integer array `cardPoints`.

In one step, you can take one card from the beginning or from the end of the row. You have to take exactly `k` cards.

Your score is the sum of the points of the cards you have taken.

Given the integer array `cardPoints` and the integer `k`, return the *maximum score* you can obtain.

Example 1:

```
Input: cardPoints = [1,2,3,4,5,6,1], k = 3
Output: 12
Explanation: After the first step, your score will always be 1. However, choosing the rightmost card first will maximize your total score. The optimal strategy is to take the three cards on the right, giving a final score of 1 + 6 + 5 = 12.
```

Example 2:

```
Input: cardPoints = [2,2,2], k = 2
Output: 4
Explanation: Regardless of which two cards you take, your score will always be 4.
```

Example 3:

```
Input: cardPoints = [9,7,7,9,7,7,9], k = 7
Output: 55
Explanation: You have to take all the cards. Your score is the sum of points of all cards.
```

Example 4:

```
Input: cardPoints = [1,1000,1], k = 1
Output: 1
Explanation: You cannot take the card in the middle. Your best score is 1.
```

Example 5:

```
Input: cardPoints = [1,79,80,1,1,1,200,1], k = 3
Output: 202
```

Constraints:

- `1 <= cardPoints.length <= 10^5`
- `1 <= cardPoints[i] <= 10^4`
- `1 <= k <= cardPoints.length`

题目大意

几张卡牌 排成一行，每张卡牌都有一个对应的点数。点数由整数数组 cardPoints 给出。每次行动，你可以从行的开头或者末尾拿一张卡牌，最终你必须正好拿 k 张卡牌。你的点数就是你拿到手中的所有卡牌的点数之和。给你一个整数数组 cardPoints 和整数 k，请你返回可以获得的最大点数。

解题思路

- 这一题是滑动窗口题的简化题。从卡牌两边取 K 张牌，可以转换成在中间连续取 n-K 张牌。从两边取牌的点数最大，意味着剩下来中间牌的点数最小。扫描一遍数组，在每一个窗口大小为 n-K 的窗口内计算累加和，记录下最小的累加和。题目最终求的最大点数等于牌的总和减去中间最小的累加和。

代码

```go
package leetcode

func maxScore(cardPoints []int, k int) int {
  windowSize, sum := len(cardPoints)-k, 0
  for _, val := range cardPoints[:windowSize] {
    sum += val
  }
  minSum := sum
  for i := windowSize; i < len(cardPoints); i++ {
    sum += cardPoints[i] - cardPoints[i-windowSize]
    if sum < minSum {
      minSum = sum
    }
  }
  total := 0
  for _, pt := range cardPoints {
    total += pt
  }
  return total - minSum
}
```

1437. Check If All 1's Are at Least Length K Places Away

题目

Given an array `nums` of 0s and 1s and an integer `k`, return `True` if all 1's are at least `k` places away from each other, otherwise return `False`.

Example 1:

```
Input: nums = [1,0,0,0,1,0,0,1], k = 2
Output: true
Explanation: Each of the 1s are at least 2 places away from each other.
```

Example 2:

```
Input: nums = [1,0,0,1,0,1], k = 2
Output: false
Explanation: The second 1 and third 1 are only one apart from each other.
```

Example 3:

```
Input: nums = [1,1,1,1,1], k = 0
Output: true
```

Example 4:

```
Input: nums = [0,1,0,1], k = 1
Output: true
```

Constraints:

- `1 <= nums.length <= 10^5`
- `0 <= k <= nums.length`
- `nums[i]` is `0` or `1`

题目大意

给你一个由若干 0 和 1 组成的数组 nums 以及整数 k。如果所有 1 都至少相隔 k 个元素，则返回 True ；否则，返回 False 。

解题思路

- 简单题。扫描一遍数组，遇到 1 的时候比较前一个 1 的下标索引，如果相隔小于 k 则返回 false。如果大于等于 k 就更新下标索引，继续循环。循环结束输出 true 即可。

代码

```go
package leetcode

func kLengthApart(nums []int, k int) bool {
  prevIndex := -1
  for i, num := range nums {
    if num == 1 {
      if prevIndex != -1 && i-prevIndex-1 < k {
        return false
      }
      prevIndex = i
    }
  }
  return true
}
```

1438. Longest Continuous Subarray With Absolute Diff Less Than or Equal to Limit

题目

Given an array of integers `nums` and an integer `limit`, return the size of the longest **non-empty** subarray such that the absolute difference between any two elements of this subarray is less than or equal to `limit`.

Example 1:

```
Input: nums = [8,2,4,7], limit = 4
Output: 2
Explanation: All subarrays are:
[8] with maximum absolute diff |8-8| = 0 <= 4.
[8,2] with maximum absolute diff |8-2| = 6 > 4.
[8,2,4] with maximum absolute diff |8-2| = 6 > 4.
[8,2,4,7] with maximum absolute diff |8-2| = 6 > 4.
[2] with maximum absolute diff |2-2| = 0 <= 4.
[2,4] with maximum absolute diff |2-4| = 2 <= 4.
[2,4,7] with maximum absolute diff |2-7| = 5 > 4.
[4] with maximum absolute diff |4-4| = 0 <= 4.
[4,7] with maximum absolute diff |4-7| = 3 <= 4.
[7] with maximum absolute diff |7-7| = 0 <= 4.
Therefore, the size of the longest subarray is 2.
```

Example 2:

```
Input: nums = [10,1,2,4,7,2], limit = 5
Output: 4
Explanation: The subarray [2,4,7,2] is the longest since the maximum absolute diff is
|2-7| = 5 <= 5.
```

Example 3:

```
Input: nums = [4,2,2,2,4,4,2,2], limit = 0
Output: 3
```

Constraints:

- `1 <= nums.length <= 10^5`
- `1 <= nums[i] <= 10^9`
- `0 <= limit <= 10^9`

题目大意

给你一个整数数组 nums ，和一个表示限制的整数 limit，请你返回最长连续子数组的长度，该子数组中的任意两个元素之间的绝对差必须小于或者等于 limit 。如果不存在满足条件的子数组，则返回 0 。

解题思路

- 最开始想到的思路是利用滑动窗口遍历一遍数组，每个窗口内排序，取出最大最小值。滑动窗口遍历一次的时间复杂度是 O(n)，所以此题时间复杂度是否高效落在了排序算法上了。由于前后 2 个窗口数据是有关联的，仅仅只变动了 2 个数据（左窗口移出的数据和右窗口移进的数据），所以排序没有必要每次都重新排序。这里利用二叉排序树来排序，添加和删除元素时间复杂度是 O(log n)，这种方法总的时间复杂度是 O(n log n)。空间复杂度 O(n)。
- 二叉排序树的思路是否还有再优化的空间？答案是有。二叉排序树内维护了所有结点的有序关系，但是这个关系是多余的。此题只需要找到最大值和最小值，并不需要除此以外节点的有序信息。所以用二叉排序树是大材小用了。可以换成 2 个单调队列，一个维护窗口内的最大值，另一个维护窗口内的最小值。这样优化以后，时间复杂度降低到 O(n)，空间复杂度 O(n)。具体实现见代码。
- 单调栈的题还有第 42 题，第 84 题，第 496 题，第 503 题，第 739 题，第 856 题，第 901 题，第 907 题，第 1130 题，第 1425 题，第 1673 题。

代码

```go
package leetcode

func longestSubarray(nums []int, limit int) int {
  minStack, maxStack, left, res := []int{}, []int{}, 0, 0
  for right, num := range nums {
    for len(minStack) > 0 && nums[minStack[len(minStack)-1]] > num {
      minStack = minStack[:len(minStack)-1]
    }
    minStack = append(minStack, right)
    for len(maxStack) > 0 && nums[maxStack[len(maxStack)-1]] < num {
      maxStack = maxStack[:len(maxStack)-1]
    }
    maxStack = append(maxStack, right)
    if len(minStack) > 0 && len(maxStack) > 0 && nums[maxStack[0]]-nums[minStack[0]] >
limit {
      if left == minStack[0] {
        minStack = minStack[1:]
      }
      if left == maxStack[0] {
        maxStack = maxStack[1:]
      }
      left++
    }
    if right-left+1 > res {
      res = right - left + 1
    }
  }
  return res
}
```

1439. Find the Kth Smallest Sum of a Matrix With Sorted Rows

题目

You are given an `m * n` matrix, `mat`, and an integer `k`, which has its rows sorted in non-decreasing order.

You are allowed to choose exactly 1 element from each row to form an array. Return the Kth **smallest** array sum among all possible arrays.

Example 1:

```
Input: mat = [[1,3,11],[2,4,6]], k = 5
Output: 7
Explanation: Choosing one element from each row, the first k smallest sum are:
[1,2], [1,4], [3,2], [3,4], [1,6]. Where the 5th sum is 7.
```

Example 2:

```
Input: mat = [[1,3,11],[2,4,6]], k = 9
Output: 17
```

Example 3:

```
Input: mat = [[1,10,10],[1,4,5],[2,3,6]], k = 7
Output: 9
Explanation: Choosing one element from each row, the first k smallest sum are:
[1,1,2], [1,1,3], [1,4,2], [1,4,3], [1,1,6], [1,5,2], [1,5,3]. Where the 7th sum is 9.
```

Example 4:

```
Input: mat = [[1,1,10],[2,2,9]], k = 7
Output: 12
```

Constraints:

- `m == mat.length`
- `n == mat.length[i]`
- `1 <= m, n <= 40`
- `1 <= k <= min(200, n ^ m)`
- `1 <= mat[i][j] <= 5000`
- `mat[i]` is a non decreasing array.

题目大意

给你一个 m * n 的矩阵 mat，以及一个整数 k ，矩阵中的每一行都以非递减的顺序排列。你可以从每一行中选出 1 个元素形成一个数组。返回所有可能数组中的第 k 个 最小数组和。

解题思路

- 这一题是第 373 题的升级版。在第 373 题中，给定 2 个有序数组，要求分别从这 2 个数组中选出一个数组成一个数对，最终输出和最小的 K 组。这一题中给出的是 m*n 的矩阵。其实是将第 373 题的 2 个数组升级为了 m 个数组。无非外层多了一层循环。这层循环依次从每一行中选出一个数，先从第 0 行和第 1 行取数，找到前 K 小的组合以后，再从第 2 行取数，以此类推。其他做法和第 373 题一致。维护一个长度为 k 的最小堆。每次从堆中 pop 出最小的数组和 sum 和对应的下标 index，然后依次将下标向后移动一位，生成新的 sum，加入堆中。

代码

```
package leetcode

import "container/heap"

func kthSmallest(mat [][]int, k int) int {
  if len(mat) == 0 || len(mat[0]) == 0 || k == 0 {
    return 0
  }
  prev := mat[0]
  for i := 1; i < len(mat); i++ {
    prev = kSmallestPairs(prev, mat[i], k)
  }
  if k < len(prev) {
    return -1
  }
  return prev[k-1]
}

func kSmallestPairs(nums1 []int, nums2 []int, k int) []int {
  res := []int{}
  if len(nums2) == 0 {
    return res
  }
  pq := newPriorityQueue()
  for i := 0; i < len(nums1) && i < k; i++ {
    heap.Push(pq, &pddata{
      n1:    nums1[i],
      n2:    nums2[0],
      n2Idx: 0,
    })
  }
  for pq.Len() > 0 {
    i := heap.Pop(pq)
    data := i.(*pddata)
    res = append(res, data.n1+data.n2)
```

```go
    k--
    if k <= 0 {
      break
    }
    idx := data.n2Idx
    idx++
    if idx >= len(nums2) {
      continue
    }
    heap.Push(pq, &pddata{
      n1:    data.n1,
      n2:    nums2[idx],
      n2Idx: idx,
    })
  }
  return res
}

type pddata struct {
  n1    int
  n2    int
  n2Idx int
}

type priorityQueue []*pddata

func newPriorityQueue() *priorityQueue {
  pq := priorityQueue([]*pddata{})
  heap.Init(&pq)
  return &pq
}

func (pq priorityQueue) Len() int           { return len(pq) }
func (pq priorityQueue) Swap(i, j int)      { pq[i], pq[j] = pq[j], pq[i] }
func (pq priorityQueue) Less(i, j int) bool { return pq[i].n1+pq[i].n2 <
pq[j].n1+pq[j].n2 }
func (pq *priorityQueue) Pop() interface{} {
  old := *pq
  val := old[len(old)-1]
  old[len(old)-1] = nil
  *pq = old[0 : len(old)-1]
  return val
}

func (pq *priorityQueue) Push(i interface{}) {
  val := i.(*pddata)
  *pq = append(*pq, val)
}
```

1442. Count Triplets That Can Form Two Arrays of Equal XOR

题目

Given an array of integers `arr`.

We want to select three indices `i`, `j` and `k` where `(0 <= i < j <= k < arr.length)`.

Let's define `a` and `b` as follows:

- `a = arr[i] ^ arr[i + 1] ^ ... ^ arr[j - 1]`
- `b = arr[j] ^ arr[j + 1] ^ ... ^ arr[k]`

Note that **^** denotes the **bitwise-xor** operation.

Return *the number of triplets* (`i`, `j` and `k`) Where `a == b`.

Example 1:

```
Input: arr = [2,3,1,6,7]
Output: 4
Explanation: The triplets are (0,1,2), (0,2,2), (2,3,4) and (2,4,4)
```

Example 2:

```
Input: arr = [1,1,1,1,1]
Output: 10
```

Example 3:

```
Input: arr = [2,3]
Output: 0
```

Example 4:

```
Input: arr = [1,3,5,7,9]
Output: 3
```

Example 5:

```
Input: arr = [7,11,12,9,5,2,7,17,22]
Output: 8
```

Constraints:

- `1 <= arr.length <= 300`

- `1 <= arr[i] <= 10^8`

题目大意

给你一个整数数组 arr 。现需要从数组中取三个下标 i、j 和 k ，其中 (0 <= i < j <= k < arr.length) 。a 和 b 定义如下：

- a = arr[i] ^ arr[i + 1] ^ ... ^ arr[j - 1]
- b = arr[j] ^ arr[j + 1] ^ ... ^ arr[k]

注意：^ 表示 按位异或 操作。请返回能够令 a == b 成立的三元组 (i, j , k) 的数目。

解题思路

- 这一题需要用到 `x^x = 0` 这个异或特性。题目要求 `a == b`，可以等效转化为 `arr[i] ^ arr[i + 1] ^ ... ^ arr[j - 1] ^ arr[j] ^ arr[j + 1] ^ ... ^ arr[k] = 0`，这样 j 相当于可以“忽略”，专注找到所有元素异或结果为 0 的区间 [i,k] 即为答案。利用前缀和的思想，只不过此题非累加和，而是异或。又由 `x^x = 0` 这个异或特性，相同部分异或相当于消除，于是有 `prefix[i,k] = prefix[0,k] ^ prefix[0,i-1]`，找到每一个 `prefix[i,k] = 0` 的 i，k 组合，i < j <= k，那么满足条件的三元组 (i,j,k) 的个数完全取决于 j 的取值范围，(因为 i 和 k 已经固定了)，j 的取值范围为 k-i，所以累加所有满足条件的 k-i，输出即为最终答案。

代码

```go
package leetcode

func countTriplets(arr []int) int {
  prefix, num, count, total := make([]int, len(arr)), 0, 0, 0
  for i, v := range arr {
    num ^= v
    prefix[i] = num
  }
  for i := 0; i < len(prefix)-1; i++ {
    for k := i + 1; k < len(prefix); k++ {
      total = prefix[k]
      if i > 0 {
        total ^= prefix[i-1]
      }
      if total == 0 {
        count += k - i
      }
    }
  }
  return count
}
```

1455. Check If a Word Occurs As a Prefix of Any Word in a Sentence

题目

Given a `sentence` that consists of some words separated by a **single space**, and a `searchWord`.

You have to check if `searchWord` is a prefix of any word in `sentence`.

Return *the index of the word* in `sentence` where `searchWord` is a prefix of this word (**1-indexed**).

If `searchWord` is a prefix of more than one word, return the index of the first word **(minimum index)**. If there is no such word return **-1**.

A **prefix** of a string `s` is any leading contiguous substring of `s`.

Example 1:

```
Input: sentence = "i love eating burger", searchWord = "burg"
Output: 4
Explanation: "burg" is prefix of "burger" which is the 4th word in the sentence.
```

Example 2:

```
Input: sentence = "this problem is an easy problem", searchWord = "pro"
Output: 2
Explanation: "pro" is prefix of "problem" which is the 2nd and the 6th word in the
sentence, but we return 2 as it's the minimal index.
```

Example 3:

```
Input: sentence = "i am tired", searchWord = "you"
Output: -1
Explanation: "you" is not a prefix of any word in the sentence.
```

Example 4:

```
Input: sentence = "i use triple pillow", searchWord = "pill"
Output: 4
```

Example 5:

```
Input: sentence = "hello from the other side", searchWord = "they"
Output: -1
```

Constraints:

- `1 <= sentence.length <= 100`
- `1 <= searchWord.length <= 10`
- `sentence` consists of lowercase English letters and spaces.
- `searchWord` consists of lowercase English letters.

题目大意

给你一个字符串 sentence 作为句子并指定检索词为 searchWord ，其中句子由若干用 单个空格 分隔的单词组成。请你检查检索词 searchWord 是否为句子 sentence 中任意单词的前缀。

- 如果 searchWord 是某一个单词的前缀，则返回句子 sentence 中该单词所对应的下标（下标从 1 开始）。
- 如果 searchWord 是多个单词的前缀，则返回匹配的第一个单词的下标（最小下标）。
- 如果 searchWord 不是任何单词的前缀，则返回 -1 。

字符串 S 的 「前缀」是 S 的任何前导连续子字符串。

解题思路

- 给出 2 个字符串，一个是匹配串，另外一个是句子。在句子里面查找带匹配串前缀的单词，并返回第一个匹配单词的下标。
- 简单题。按照题意，扫描一遍句子，一次匹配即可。

代码

```go
package leetcode

import "strings"

func isPrefixOfWord(sentence string, searchWord string) int {
  for i, v := range strings.Split(sentence, " ") {
    if strings.HasPrefix(v, searchWord) {
      return i + 1
    }
  }
  return -1
}
```

[1461. Check If a String Contains All Binary Codes of Size K]

题目

Given a binary string `s` and an integer `k`.

Return *True* if every binary code of length `k` is a substring of `s`. Otherwise, return *False*.

Example 1:

```
Input: s = "00110110", k = 2
Output: true
Explanation: The binary codes of length 2 are "00", "01", "10" and "11". They can be
all found as substrings at indicies 0, 1, 3 and 2 respectively.
```

Example 2:

```
Input: s = "00110", k = 2
Output: true
```

Example 3:

```
Input: s = "0110", k = 1
Output: true
Explanation: The binary codes of length 1 are "0" and "1", it is clear that both exist
as a substring.
```

Example 4:

```
Input: s = "0110", k = 2
Output: false
Explanation: The binary code "00" is of length 2 and doesn't exist in the array.
```

Example 5:

```
Input: s = "0000000001011100", k = 4
Output: false
```

Constraints:

- `1 <= s.length <= 5 * 10^5`
- `s` consists of 0's and 1's only.
- `1 <= k <= 20`

题目大意

给你一个二进制字符串 `s` 和一个整数 `k` 。如果所有长度为 `k` 的二进制字符串都是 `s` 的子串，请返回 `True` ，否则请返回 `False` 。

解题思路

- 构造一个 `mask` 遮罩，依次划过整个二进制字符串，每次滑动即取出遮罩遮住的 `k` 位二进制字符。可以用 `map` 存储不同的二进制转换成的十进制数，最后判断 `len(map)` 是否等于 `k` 即可。但是用 `map` 存比较慢，此处换成 `bool` 数组。先构造一个长度为 `k` 的数组，然后每次通过 `mask` 更新这个 `bool` 数组对应十进制的 `bool` 值，并且记录剩余还缺几个二进制数。等剩余的等于 0 的时候，说明所有二进制字符串都出现了，直接输出 `true`，否则循环完以后输出 `false`。

代码

```go
package leetcode

import "math"

func hasAllCodes(s string, k int) bool {
  need := int(math.Pow(2.0, float64(k)))
  visited, mask, curr := make([]bool, need), (1<<k)-1, 0
  for i := 0; i < len(s); i++ {
    curr = ((curr << 1) | int(s[i]-'0')) & mask
    if i >= k-1 { // mask 有效位达到了 k 位
      if !visited[curr] {
        need--
        visited[curr] = true
        if need == 0 {
          return true
        }
      }
    }
  }
  return false
}
```

1463. Cherry Pickup II

题目

Given a `rows x cols` matrix `grid` representing a field of cherries. Each cell in `grid` represents the number of cherries that you can collect.

You have two robots that can collect cherries for you, Robot #1 is located at the top-left corner (0,0) , and Robot #2 is located at the top-right corner (0, cols-1) of the grid.

Return the maximum number of cherries collection using both robots by following the rules below:

- From a cell (i,j), robots can move to cell (i+1, j-1) , (i+1, j) or (i+1, j+1).
- When any robot is passing through a cell, It picks it up all cherries, and the cell becomes an empty cell (0).
- When both robots stay on the same cell, only one of them takes the cherries.

- Both robots cannot move outside of the grid at any moment.
- Both robots should reach the bottom row in the `grid`.

Example 1:

```
Input: grid = [[3,1,1],[2,5,1],[1,5,5],[2,1,1]]
Output: 24
Explanation: Path of robot #1 and #2 are described in color green and blue
respectively.
Cherries taken by Robot #1, (3 + 2 + 5 + 2) = 12.
Cherries taken by Robot #2, (1 + 5 + 5 + 1) = 12.
Total of cherries: 12 + 12 = 24.
```

Example 2:

```
Input: grid = [[1,0,0,0,0,0,1],[2,0,0,0,0,3,0],[2,0,9,0,0,0,0],[0,3,0,5,4,0,0],
[1,0,2,3,0,0,6]]
Output: 28
Explanation: Path of robot #1 and #2 are described in color green and blue
respectively.
Cherries taken by Robot #1, (1 + 9 + 5 + 2) = 17.
Cherries taken by Robot #2, (1 + 3 + 4 + 3) = 11.
Total of cherries: 17 + 11 = 28.
```

Example 3:

```
Input: grid = [[1,0,0,3],[0,0,0,3],[0,0,3,3],[9,0,3,3]]
Output: 22
```

Example 4:

```
Input: grid = [[1,1],[1,1]]
Output: 4
```

Constraints:

- `rows == grid.length`
- `cols == grid[i].length`
- `2 <= rows, cols <= 70`
- `0 <= grid[i][j] <= 100`

题目大意

给你一个 rows x cols 的矩阵 grid 来表示一块樱桃地。 grid 中每个格子的数字表示你能获得的樱桃数目。你有两个机器人帮你收集樱桃，机器人 1 从左上角格子 (0,0) 出发，机器人 2 从右上角格子 (0, cols-1) 出发。请你按照如下规则，返回两个机器人能收集的最多樱桃数目：

- 从格子 (i,j) 出发，机器人可以移动到格子 (i+1, j-1)，(i+1, j) 或者 (i+1, j+1) 。
- 当一个机器人经过某个格子时，它会把该格子内所有的樱桃都摘走，然后这个位置会变成空格子，即没有樱桃的格子。
- 当两个机器人同时到达同一个格子时，它们中只有一个可以摘到樱桃。
- 两个机器人在任意时刻都不能移动到 grid 外面。
- 两个机器人最后都要到达 grid 最底下一行。

解题思路

- 如果没有思路可以先用暴力解法 DFS 尝试。读完题可以分析出求最多樱桃数目，里面包含了很多重叠子问题，于是乎自然而然思路是用动态规划。数据规模上看，100 的数据规模最多能保证 O(n^3) 时间复杂度的算法不超时。
- 这一题的变量有 2 个，一个是行号，另外一个是机器人所在的列。具体来说，机器人每走一步的移动范围只能往下走，不能往上走，所以 2 个机器人所在行号一定相同。两个机器人的列号不同。综上，变量有 3 个，1 个行号和2 个列号。定义 `dp[i][j][k]` 代表第一个机器人从 (0,0) 走到 (i,k) 坐标，第二个机器人从 (0,n-1) 走到 (i,k) 坐标，两者最多能收集樱桃的数目。状态转移方程为 ：

  {{< katex display >}}
  dp[i][j][k] = max \begin{pmatrix}\begin{array}{lr} dp[i-1][f(j_1))][f(j_2)] + grid[i][j_1] + grid[i][j_2], j_1\neq j_2 \ dp[i-1][f(j_1))][f(j_2)] + grid[i][j_1], j_1 = j_2 \end{array} \end{pmatrix}
  {{< /katex>}}

  其中：

  {{< katex display >}}
  \left{\begin{matrix}f(j_1) \in [0,n), f(j_1) - j_1 \in [-1,0,1]\ f(j_2) \in [0,n), f(j_2) - j_2 \in [-1,0,1]\end{matrix}\right.
  {{< /katex>}}

  即状态转移过程中需要在 `[j1 - 1, j1, j1 + 1]` 中枚举 `j1`，同理，在 在 `[j2 - 1, j2, j2 + 1]` 中枚举 `j2`，每个状态转移需要枚举这 3*3 = 9 种状态。
- 边界条件 `dp[i][0][n-1] = grid[0][0] + grid[0][n-1]`，最终答案存储在 `dp[m-1]` 行中，循环找出 `dp[m-1][j1][j2]` 中的最大值，到此该题得解。

代码

```go
package leetcode

func cherryPickup(grid [][]int) int {
  rows, cols := len(grid), len(grid[0])
  dp := make([][][]int, rows)
  for i := 0; i < rows; i++ {
    dp[i] = make([][]int, cols)
    for j := 0; j < cols; j++ {
      dp[i][j] = make([]int, cols)
    }
  }
  for i := 0; i < rows; i++ {
    for j := 0; j <= i && j < cols; j++ {
```

```go
      for k := cols - 1; k >= cols-1-i && k >= 0; k-- {
        max := 0
        for a := j - 1; a <= j+1; a++ {
          for b := k - 1; b <= k+1; b++ {
            sum := isInBoard(dp, i-1, a, b)
            if a == b && i > 0 && a >= 0 && a < cols {
              sum -= grid[i-1][a]
            }
            if sum > max {
              max = sum
            }
          }
        }
        if j == k {
          max += grid[i][j]
        } else {
          max += grid[i][j] + grid[i][k]
        }
        dp[i][j][k] = max
      }
    }
  }
  count := 0
  for j := 0; j < cols && j < rows; j++ {
    for k := cols - 1; k >= 0 && k >= cols-rows; k-- {
      if dp[rows-1][j][k] > count {
        count = dp[rows-1][j][k]
      }
    }
  }
  return count
}

func isInBoard(dp [][][]int, i, j, k int) int {
  if i < 0 || j < 0 || j >= len(dp[0]) || k < 0 || k >= len(dp[0]) {
    return 0
  }
  return dp[i][j][k]
}
```

1464. Maximum Product of Two Elements in an Array

题目

Given the array of integers `nums`, you will choose two different indices `i` and `j` of that array. Return the maximum value of `(nums[i]-1)*(nums[j]-1)`.

Example 1:

```
Input: nums = [3,4,5,2]
Output: 12
Explanation: If you choose the indices i=1 and j=2 (indexed from 0), you will get the
maximum value, that is, (nums[1]-1)*(nums[2]-1) = (4-1)*(5-1) = 3*4 = 12.
```

Example 2:

```
Input: nums = [1,5,4,5]
Output: 16
Explanation: Choosing the indices i=1 and j=3 (indexed from 0), you will get the
maximum value of (5-1)*(5-1) = 16.
```

Example 3:

```
Input: nums = [3,7]
Output: 12
```

Constraints:

- `2 <= nums.length <= 500`
- `1 <= nums[i] <= 10^3`

题目大意

给你一个整数数组 nums，请你选择数组的两个不同下标 i 和 j，使 (nums[i]-1)*(nums[j]-1) 取得最大值。请你计算并返回该式的最大值。

解题思路

- 简单题。循环一次，按照题意动态维护 2 个最大值，从而也使得 `(nums[i]-1)*(nums[j]-1)` 能取到最大值。

代码

```go
package leetcode

func maxProduct(nums []int) int {
  max1, max2 := 0, 0
  for _, num := range nums {
    if num >= max1 {
      max2 = max1
```

```
      max1 = num
    } else if num <= max1 && num >= max2 {
      max2 = num
    }
  }
  return (max1 - 1) * (max2 - 1)
}
```

1465. Maximum Area of a Piece of Cake After Horizontal and Vertical Cuts

题目

Given a rectangular cake with height `h` and width `w`, and two arrays of integers `horizontalCuts` and `verticalCuts` where `horizontalCuts[i]` is the distance from the top of the rectangular cake to the `ith` horizontal cut and similarly, `verticalCuts[j]` is the distance from the left of the rectangular cake to the `jth` vertical cut.

Return the maximum area of a piece of cake after you cut at each horizontal and vertical position provided in the arrays `horizontalCuts` and `verticalCuts`. Since the answer can be a huge number, return this modulo 10^9 + 7.

Example 1:

```
Input: h = 5, w = 4, horizontalCuts = [1,2,4], verticalCuts = [1,3]
Output: 4
Explanation: The figure above represents the given rectangular cake. Red lines are the horizontal and vertical cuts. After you cut the cake, the green piece of cake has the maximum area.
```

Example 2:

```
Input: h = 5, w = 4, horizontalCuts = [3,1], verticalCuts = [1]
Output: 6
Explanation: The figure above represents the given rectangular cake. Red lines are the horizontal and vertical cuts. After you cut the cake, the green and yellow pieces of cake have the maximum area.
```

Example 3:

```
Input: h = 5, w = 4, horizontalCuts = [3], verticalCuts = [3]
Output: 9
```

Constraints:

- `2 <= h, w <= 10^9`
- `1 <= horizontalCuts.length < min(h, 10^5)`
- `1 <= verticalCuts.length < min(w, 10^5)`
- `1 <= horizontalCuts[i] < h`
- `1 <= verticalCuts[i] < w`
- It is guaranteed that all elements in `horizontalCuts` are distinct.
- It is guaranteed that all elements in `verticalCuts` are distinct.

题目大意

矩形蛋糕的高度为 h 且宽度为 w，给你两个整数数组 horizontalCuts 和 verticalCuts，其中 horizontalCuts[i] 是从矩形蛋糕顶部到第 i 个水平切口的距离，类似地， verticalCuts[j] 是从矩形蛋糕的左侧到第 j 个竖直切口的距离。请你按数组 horizontalCuts 和 verticalCuts 中提供的水平和竖直位置切割后，请你找出 面积最大 的那份蛋糕，并返回其 面积 。由于答案可能是一个很大的数字，因此需要将结果对 10^9 + 7 取余后返回。

解题思路

- 读完题比较容易想到解题思路。找到水平切口最大的差值和竖直切口最大的差值，这 4 条边构成的矩形即为最大矩形。不过有特殊情况需要判断，切口除了题目给的切口坐标以外，默认还有 4 个切口，即蛋糕原始的 4 条边。如下图二，最大的矩形其实在切口之外。所以找水平切口最大差值和竖直切口最大差值需要考虑到蛋糕原始的 4 条边。

代码

```go
package leetcode

import "sort"

func maxArea(h int, w int, hcuts []int, vcuts []int) int {
  sort.Ints(hcuts)
  sort.Ints(vcuts)
  maxw, maxl := hcuts[0], vcuts[0]
  for i, c := range hcuts[1:] {
    if c-hcuts[i] > maxw {
      maxw = c - hcuts[i]
    }
  }
  if h-hcuts[len(hcuts)-1] > maxw {
    maxw = h - hcuts[len(hcuts)-1]
  }
  for i, c := range vcuts[1:] {
```

```
    if c-vcuts[i] > maxl {
      maxl = c - vcuts[i]
    }
  }
  if w-vcuts[len(vcuts)-1] > maxl {
    maxl = w - vcuts[len(vcuts)-1]
  }
  return (maxw * maxl) % (1000000007)
}
```

1470. Shuffle the Array

题目

Given the array `nums` consisting of `2n` elements in the form `[x1,x2,...,xn,y1,y2,...,yn]`.

Return the array in the form `[x1,y1,x2,y2,...,xn,yn]`.

Example 1:

```
Input: nums = [2,5,1,3,4,7], n = 3
Output: [2,3,5,4,1,7]
Explanation: Since x1=2, x2=5, x3=1, y1=3, y2=4, y3=7 then the answer is [2,3,5,4,1,7].
```

Example 2:

```
Input: nums = [1,2,3,4,4,3,2,1], n = 4
Output: [1,4,2,3,3,2,4,1]
```

Example 3:

```
Input: nums = [1,1,2,2], n = 2
Output: [1,2,1,2]
```

Constraints:

- `1 <= n <= 500`
- `nums.length == 2n`
- `1 <= nums[i] <= 10^3`

题目大意

给你一个数组 nums ，数组中有 2n 个元素，按 [x1,x2,...,xn,y1,y2,...,yn] 的格式排列。请你将数组按 [x1,y1,x2,y2,...,xn,yn] 格式重新排列，返回重排后的数组。

解题思路

- 给定一个 2n 的数组，把后 n 个元素插空放到前 n 个元素里面。输出最终完成的数组。
- 简单题，按照题意插空即可。

代码

```
package leetcode

func shuffle(nums []int, n int) []int {
  result := make([]int, 0)
  for i := 0; i < n; i++ {
    result = append(result, nums[i])
    result = append(result, nums[n+i])
  }
  return result
}
```

1480. Running Sum of 1d Array

题目

Given an array `nums`. We define a running sum of an array as `runningSum[i] = sum(nums[0]…nums[i])`.

Return the running sum of `nums`.

Example 1:

```
Input: nums = [1,2,3,4]
Output: [1,3,6,10]
Explanation: Running sum is obtained as follows: [1, 1+2, 1+2+3, 1+2+3+4].
```

Example 2:

```
Input: nums = [1,1,1,1,1]
Output: [1,2,3,4,5]
Explanation: Running sum is obtained as follows: [1, 1+1, 1+1+1, 1+1+1+1, 1+1+1+1+1].
```

Example 3:

```
Input: nums = [3,1,2,10,1]
Output: [3,4,6,16,17]
```

Constraints:

- `1 <= nums.length <= 1000`
- `-10^6 <= nums[i] <= 10^6`

题目大意

给你一个数组 nums 。数组「动态和」的计算公式为：runningSum[i] = sum(nums[0]…nums[i]) 。请返回 nums 的动态和。

解题思路

- 简单题，按照题意依次循环计算前缀和即可。

代码

```go
package leetcode

func runningSum(nums []int) []int {
  dp := make([]int, len(nums)+1)
  dp[0] = 0
  for i := 1; i <= len(nums); i++ {
    dp[i] = dp[i-1] + nums[i-1]
  }
  return dp[1:]
}
```

1482. Minimum Number of Days to Make m Bouquets

题目

Given an integer array `bloomDay`, an integer `m` and an integer `k`.

We need to make `m` bouquets. To make a bouquet, you need to use `k` **adjacent flowers** from the garden.

The garden consists of `n` flowers, the `ith` flower will bloom in the `bloomDay[i]` and then can be used in **exactly one** bouquet.

Return *the minimum number of days* you need to wait to be able to make `m` bouquets from the garden. If it is impossible to make `m` bouquets return **-1**.

Example 1:

```
Input: bloomDay = [1,10,3,10,2], m = 3, k = 1
Output: 3
Explanation: Let's see what happened in the first three days. x means flower bloomed
and _ means flower didn't bloom in the garden.
We need 3 bouquets each should contain 1 flower.
After day 1: [x, _, _, _, _]   // we can only make one bouquet.
After day 2: [x, _, _, _, x]   // we can only make two bouquets.
After day 3: [x, _, x, _, x]   // we can make 3 bouquets. The answer is 3.
```

Example 2:

```
Input: bloomDay = [1,10,3,10,2], m = 3, k = 2
Output: -1
Explanation: We need 3 bouquets each has 2 flowers, that means we need 6 flowers. We
only have 5 flowers so it is impossible to get the needed bouquets and we return -1.
```

Example 3:

```
Input: bloomDay = [7,7,7,7,12,7,7], m = 2, k = 3
Output: 12
Explanation: We need 2 bouquets each should have 3 flowers.
Here's the garden after the 7 and 12 days:
After day 7: [x, x, x, x, _, x, x]
We can make one bouquet of the first three flowers that bloomed. We cannot make another
bouquet from the last three flowers that bloomed because they are not adjacent.
After day 12: [x, x, x, x, x, x, x]
It is obvious that we can make two bouquets in different ways.
```

Example 4:

```
Input: bloomDay = [1000000000,1000000000], m = 1, k = 1
Output: 1000000000
Explanation: You need to wait 1000000000 days to have a flower ready for a bouquet.
```

Example 5:

```
Input: bloomDay = [1,10,2,9,3,8,4,7,5,6], m = 4, k = 2
Output: 9
```

Constraints:

- `bloomDay.length == n`
- `1 <= n <= 10^5`
- `1 <= bloomDay[i] <= 10^9`
- `1 <= m <= 10^6`
- `1 <= k <= n`

题目大意

给你一个整数数组 bloomDay，以及两个整数 m 和 k 。现需要制作 m 束花。制作花束时，需要使用花园中 相邻的 k 朵花 。花园中有 n 朵花，第 i 朵花会在 bloomDay[i] 时盛开，恰好 可以用于 一束 花中。请你返回从花园中摘 m 束花需要等待的最少的天数。如果不能摘到 m 束花则返回 -1 。

解题思路

- 本题是二分搜索提醒。题目解空间固定，答案区间一定在 [0, maxDay] 中。这是单调增且有序区间，所以可以在这个解空间内使用二分搜索。在区间 [0, maxDay] 中找到第一个能满足 m 束花的解。二分搜索判断是否为 true 的条件为：从左往右遍历数组，依次统计当前日期下，花是否开了，如果连续开花 k 朵，便为 1 束，数组遍历结束如果花束总数 ≥ k 即为答案。二分搜索会返回最小的下标，即对应满足题意的最少天数。

代码

```go
package leetcode

import "sort"

func minDays(bloomDay []int, m int, k int) int {
  if m*k > len(bloomDay) {
    return -1
  }
  maxDay := 0
  for _, day := range bloomDay {
    if day > maxDay {
      maxDay = day
    }
  }
  return sort.Search(maxDay, func(days int) bool {
    flowers, bouquets := 0, 0
    for _, d := range bloomDay {
      if d > days {
        flowers = 0
      } else {
        flowers++
        if flowers == k {
          bouquets++
          flowers = 0
        }
      }
    }
    return bouquets >= m
  })
}
```

1486. XOR Operation in an Array

题目

Given an integer `n` and an integer `start`.

Define an array `nums` where `nums[i] = start + 2*i` (0-indexed) and `n == nums.length`.

Return the bitwise XOR of all elements of `nums`.

Example 1:

```
Input: n = 5, start = 0
Output: 8
Explanation:Array nums is equal to [0, 2, 4, 6, 8] where (0 ^ 2 ^ 4 ^ 6 ^ 8) = 8.
Where "^" corresponds to bitwise XOR operator.
```

Example 2:

```
Input: n = 4, start = 3
Output: 8
Explanation:Array nums is equal to [3, 5, 7, 9] where (3 ^ 5 ^ 7 ^ 9) = 8.
```

Example 3:

```
Input: n = 1, start = 7
Output: 7
```

Example 4:

```
Input: n = 10, start = 5
Output: 2
```

Constraints:

- `1 <= n <= 1000`
- `0 <= start <= 1000`
- `n == nums.length`

题目大意

给你两个整数，n 和 start 。数组 nums 定义为：nums[i] = start + 2*i（下标从 0 开始）且 n == nums.length 。请返回 nums 中所有元素按位异或（XOR）后得到的结果。

解题思路

- 简单题。按照题意，一层循环依次累积异或数组中每个元素。

代码

```go
package leetcode

func xorOperation(n int, start int) int {
  res := 0
  for i := 0; i < n; i++ {
    res ^= start + 2*i
  }
  return res
}
```

1512. Number of Good Pairs

题目

Given an array of integers `nums`.

A pair `(i,j)` is called good if `nums[i] == nums[j]` and `i < j`.

Return the number of good pairs.

Example 1:

```
Input: nums = [1,2,3,1,1,3]
Output: 4
Explanation: There are 4 good pairs (0,3), (0,4), (3,4), (2,5) 0-indexed.
```

Example 2:

```
Input: nums = [1,1,1,1]
Output: 6
Explanation: Each pair in the array are good.
```

Example 3:

```
Input: nums = [1,2,3]
Output: 0
```

Constraints:

- `1 <= nums.length <= 1000`
- `1 <= nums[i] <= 100`

题目大意

给你一个整数数组 nums。如果一组数字 (i,j) 满足 nums[i] == nums[j] 且 i < j ，就可以认为这是一组好数对。返回好数对的数目。

解题思路

- 简单题，按照题目中好数对的定义，循环遍历判断两数是否相等，累加计数即可。

代码

```go
package leetcode

func numIdenticalPairs(nums []int) int {
  total := 0
  for x := 0; x < len(nums); x++ {
    for y := x + 1; y < len(nums); y++ {
      if nums[x] == nums[y] {
        total++
      }
    }
  }
  return total
}
```

1539. Kth Missing Positive Number

题目

Given an array `arr` of positive integers sorted in a **strictly increasing order**, and an integer `k`.

Find the `kth` positive integer that is missing from this array.

Example 1:

```
Input: arr = [2,3,4,7,11], k = 5
Output: 9
Explanation: The missing positive integers are [1,5,6,8,9,10,12,13,...]. The
5th missing positive integer is 9.
```

Example 2:

```
Input: arr = [1,2,3,4], k = 2
Output: 6
Explanation: The missing positive integers are [5,6,7,...]. The 2nd missing positive
integer is 6.
```

Constraints:

- `1 <= arr.length <= 1000`
- `1 <= arr[i] <= 1000`
- `1 <= k <= 1000`
- `arr[i] < arr[j]` for `1 <= i < j <= arr.length`

题目大意

给你一个 **严格升序排列** 的正整数数组 `arr` 和一个整数 `k` 。请你找到这个数组里第 `k` 个缺失的正整数。

解题思路

- 简单题。用一个变量从 1 开始累加，依次比对数组中是否存在，不存在的话就把 k - -，直到 k 为 0 的时候即是要输出的值。特殊情况，missing positive 都在数组之外，如例子 2 。

代码

```go
package leetcode

func findKthPositive(arr []int, k int) int {
  positive, index := 1, 0
  for index < len(arr) {
    if arr[index] != positive {
      k--
    } else {
      index++
    }
    if k == 0 {
      break
    }
    positive++
  }
  if k != 0 {
    positive += k - 1
  }
  return positive
}
```

1551. Minimum Operations to Make Array Equal

题目

You have an array `arr` of length `n` where `arr[i] = (2 * i) + 1` for all valid values of `i` (i.e. `0 <= i < n`).

In one operation, you can select two indices `x` and `y` where `0 <= x, y < n` and subtract `1` from `arr[x]` and add `1` to `arr[y]` (i.e. perform `arr[x] -=1` and `arr[y] += 1`). The goal is to make all the elements of the array **equal**. It is **guaranteed** that all the elements of the array can be made equal using some operations.

Given an integer `n`, the length of the array. Return *the minimum number of operations* needed to make all the elements of arr equal.

Example 1:

```
Input: n = 3
Output: 2
Explanation: arr = [1, 3, 5]
First operation choose x = 2 and y = 0, this leads arr to be [2, 3, 4]
In the second operation choose x = 2 and y = 0 again, thus arr = [3, 3, 3].
```

Example 2:

```
Input: n = 6
Output: 9
```

Constraints:

- `1 <= n <= 10^4`

题目大意

存在一个长度为 n 的数组 arr ，其中 arr[i] = (2 * i) + 1 （ 0 <= i < n ）。一次操作中，你可以选出两个下标，记作 x 和 y （ 0 <= x, y < n ）并使 arr[x] 减去 1 、arr[y] 加上 1 （即 arr[x] -=1 且 arr[y] += 1 ）。最终的目标是使数组中的所有元素都 相等 。题目测试用例将会 保证： 在执行若干步操作后，数组中的所有元素最终可以全部相等。给你一个整数 n，即数组的长度。请你返回使数组 arr 中所有元素相等所需的 最小操作数 。

解题思路

- 这一题是数学题。题目给定的操作并不会使数组中所有元素之和变化，最终让所有元素相等，那么数组中所有元素的平均值即为最后数组中每一个元素的值。最少操作数的策略应该是以平均数为中心，中心右边的数减小，对称的中心左边的数增大。由于原数组是等差数列，两两元素之间相差 2，利用数学方法可以算出操作数。
- 数组长度分为奇数和偶数分别讨论。如果数组长度为奇数，所需要的操作数是：

  {{< katex display >}}
  \begin{aligned} &\quad 2 + 4 + \cdots + 2\cdot\left\lfloor\frac{n}{2}\right\rfloor \ &= \frac{1}{2}\left\lfloor\frac{n}{2}\right\rfloor\left(2\cdot\left\lfloor\frac{n}{2}\right\rfloor + 2\right) \ &= \left\lfloor\frac{n}{2}\right\rfloor \left(\left\lfloor\frac{n}{2}\right\rfloor + 1\right) \ &= \frac{n-1}{2}\left(\frac{n-1}{2} + 1\right) \ &= \frac{n-1}{2}\cdot\frac{n+1}{2} \ &= \frac{n^2-1}{4} \ &= \left\lfloor\frac{n^2}{4}\right\rfloor \end{aligned}
  {{< /katex >}}

  数组长度是偶数，所需要的操作数是：

{{< katex display >}}
\begin{aligned} &\quad 1 + 3 + \cdots + \left(2\cdot\left\lfloor\frac{n}{2}\right\rfloor - 1\right) \ &= \frac{1}{2}\left\lfloor\frac{n}{2}\right\rfloor\left(2\cdot\left\lfloor\frac{n}{2}\right\rfloor - 1 + 1\right)\ &= \left(\left\lfloor\frac{n}{2}\right\rfloor\right)^2 \ &= \frac{n^2}{4} \end{aligned}
{{< /katex >}}

综上所述，最小操作数是 n^2/4

代码

```
package leetcode

func minOperations(n int) int {
  return n * n / 4
}
```

1572. Matrix Diagonal Sum

题目

Given a square matrix `mat`, return the sum of the matrix diagonals.

Only include the sum of all the elements on the primary diagonal and all the elements on the secondary diagonal that are not part of the primary diagonal.

Example 1:

```
Input: mat = [[1,2,3],
              [4,5,6],
              [7,8,9]]
Output: 25
Explanation:Diagonals sum: 1 + 5 + 9 + 3 + 7 = 25
Notice that element mat[1][1] = 5 is counted only once.
```

Example 2:

```
Input: mat = [[1,1,1,1],
              [1,1,1,1],
              [1,1,1,1],
              [1,1,1,1]]
Output: 8
```

Example 3:

```
Input: mat = [[5]]
Output: 5
```

Constraints:

- `n == mat.length == mat[i].length`
- `1 <= n <= 100`
- `1 <= mat[i][j] <= 100`

题目大意

给你一个正方形矩阵 mat，请你返回矩阵对角线元素的和。请你返回在矩阵主对角线上的元素和副对角线上且不在主对角线上元素的和。

解题思路

- 简单题。根据题意，把主对角线和副对角线上的元素相加。
- 如果正方形矩阵的长度 n 为奇数，相加的结果需要减去 mat[n/2][n/2]。

代码

```go
package leetcode

func diagonalSum(mat [][]int) int {
  n := len(mat)
  ans := 0
  for pi := 0; pi < n; pi++ {
    ans += mat[pi][pi]
  }
  for si, sj := n-1, 0; sj < n; si, sj = si-1, sj+1 {
    ans += mat[si][sj]
  }
  if n%2 == 0 {
    return ans
  }
  return ans - mat[n/2][n/2]
}
```

1573. Number of Ways to Split a String

题目

Given a binary string `s` (a string consisting only of '0's and '1's), we can split `s` into 3 **non-empty** strings s1, s2, s3 (s1+ s2+ s3 = s).

Return the number of ways `s` can be split such that the number of characters '1' is the same in s1, s2, and s3.

Since the answer may be too large, return it modulo 10^9 + 7.

Example 1:

```
Input: s = "10101"
Output: 4
Explanation: There are four ways to split s in 3 parts where each part contain the same number of letters '1'.
"1|010|1"
"1|01|01"
"10|10|1"
"10|1|01"
```

Example 2:

```
Input: s = "1001"
Output: 0
```

Example 3:

```
Input: s = "0000"
Output: 3
Explanation: There are three ways to split s in 3 parts.
"0|0|00"
"0|00|0"
"00|0|0"
```

Example 4:

```
Input: s = "100100010100110"
Output: 12
```

Constraints:

- `3 <= s.length <= 10^5`
- `s[i]` is `'0'` or `'1'`.

题目大意

给你一个二进制串 s （一个只包含 0 和 1 的字符串），我们可以将 s 分割成 3 个 非空 字符串 s1, s2, s3 （s1 + s2 + s3 = s）。请你返回分割 s 的方案数，满足 s1，s2 和 s3 中字符 '1' 的数目相同。由于答案可能很大，请将它对 10^9 + 7 取余后返回。

解题思路

- 这一题是考察的排列组合的知识。根据题意，如果 1 的个数不是 3 的倍数，直接返回 -1。如果字符串里面没有 1，那么切分的方案就是组合，在 n-1 个字母里面选出 2 个位置。利用组合的计算方法，组合数是 (n-1) * (n-2) / 2 。
- 剩下的是 3 的倍数的情况。在字符串中选 2 个位置隔成 3 段。从第一段最后一个 1 到第二段第一个 1 之间的 0 的个数为 m1，从第二段最后一个 1 到第三段第一个 1 之间的 0 的个数为 m2。利用乘法原理，方案数为 m1 * m2。

代码

```
package leetcode

func numWays(s string) int {
  ones := 0
  for _, c := range s {
    if c == '1' {
      ones++
    }
  }
  if ones%3 != 0 {
    return 0
  }
  if ones == 0 {
    return (len(s) - 1) * (len(s) - 2) / 2 % 1000000007
  }
  N, a, b, c, d, count := ones/3, 0, 0, 0, 0, 0
  for i, letter := range s {
    if letter == '0' {
      continue
    }
    if letter == '1' {
      count++
    }
    if count == N {
      a = i
    }
    if count == N+1 {
      b = i
    }
    if count == 2*N {
      c = i
    }
    if count == 2*N+1 {
```

```
      d = i
    }
  }
  return (b - a) * (d - c) % 1000000007
}
```

1579. Remove Max Number of Edges to Keep Graph Fully Traversable

题目

Alice and Bob have an undirected graph of `n` nodes and 3 types of edges:

- Type 1: Can be traversed by Alice only.
- Type 2: Can be traversed by Bob only.
- Type 3: Can by traversed by both Alice and Bob.

Given an array `edges` where `edges[i] = [typei, ui, vi]` represents a bidirectional edge of type `typei` between nodes `ui` and `vi`, find the maximum number of edges you can remove so that after removing the edges, the graph can still be fully traversed by both Alice and Bob. The graph is fully traversed by Alice and Bob if starting from any node, they can reach all other nodes.

Return *the maximum number of edges you can remove, or return* `-1` *if it's impossible for the graph to be fully traversed by Alice and Bob.*

Example 1:

```
Input: n = 4, edges = [[3,1,2],[3,2,3],[1,1,3],[1,2,4],[1,1,2],[2,3,4]]
Output: 2
Explanation: If we remove the 2 edges [1,1,2] and [1,1,3]. The graph will still be
fully traversable by Alice and Bob. Removing any additional edge will not make it so.
So the maximum number of edges we can remove is 2.
```

Example 2:

```
Input: n = 4, edges = [[3,1,2],[3,2,3],[1,1,4],[2,1,4]]
Output: 0
Explanation: Notice that removing any edge will not make the graph fully traversable by
Alice and Bob.
```

Example 3:

```
Input: n = 4, edges = [[3,2,3],[1,1,2],[2,3,4]]
Output: -1
Explanation: In the current graph, Alice cannot reach node 4 from the other nodes.
Likewise, Bob cannot reach 1. Therefore it's impossible to make the graph fully
traversable.
```

Constraints:

- `1 <= n <= 10^5`
- `1 <= edges.length <= min(10^5, 3 * n * (n-1) / 2)`
- `edges[i].length == 3`
- `1 <= edges[i][0] <= 3`
- `1 <= edges[i][1] < edges[i][2] <= n`
- All tuples `(typei, ui, vi)` are distinct.

题目大意

Alice 和 Bob 共有一个无向图，其中包含 n 个节点和 3 种类型的边：

- 类型 1：只能由 Alice 遍历。
- 类型 2：只能由 Bob 遍历。
- 类型 3：Alice 和 Bob 都可以遍历。

给你一个数组 edges ，其中 edges[i] = [typei, ui, vi] 表示节点 ui 和 vi 之间存在类型为 typei 的双向边。请你在保证图仍能够被 Alice和 Bob 完全遍历的前提下，找出可以删除的最大边数。如果从任何节点开始，Alice 和 Bob 都可以到达所有其他节点，则认为图是可以完全遍历的。返回可以删除的最大边数，如果 Alice 和 Bob 无法完全遍历图，则返回 -1 。

解题思路

- 本题是第 1319 题的加强版。在第 1319 题中只有一个人，同样也是判断在保证图可连通的基础上，删掉最多边的条数。这一题只不过变成了 2 个人。解题思路依旧是并查集。
- 初始化 2 个并查集，分别表示 Alice 和 Bob。先合并公共边，每合并一条边，可删除的最大总边数便减少 1 。再合并 2 人各自的单独的边，同样是每合并一条边，每合并一条边，可删除的最大总边数便减少 1 。合并完所有的边，2 人的并查集内部集合数仍大于 1，那么则代表 2 人无法完全遍历图，则输出 -1。如果 2 人的并查集内部集合都是 1，代表整个图都连通了。输出可以删除的最大边数。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/template"
)

func maxNumEdgesToRemove(n int, edges [][]int) int {
  alice, bob, res := template.UnionFind{}, template.UnionFind{}, len(edges)
  alice.Init(n)
```

```go
  bob.Init(n)
  for _, e := range edges {
    x, y := e[1]-1, e[2]-1
    if e[0] == 3 && (!(alice.Find(x) == alice.Find(y)) || !(bob.Find(x) ==
bob.Find(y))) {
      alice.Union(x, y)
      bob.Union(x, y)
      res--
    }
  }
  ufs := [2]*template.UnionFind{&alice, &bob}
  for _, e := range edges {
    if tp := e[0]; tp < 3 && !(ufs[tp-1].Find(e[1]-1) == ufs[tp-1].Find(e[2]-1)) {
      ufs[tp-1].Union(e[1]-1, e[2]-1)
      res--
    }
  }
  if alice.TotalCount() > 1 || bob.TotalCount() > 1 {
    return -1
  }
  return res
}
```

1600. Throne Inheritance

题目

A kingdom consists of a king, his children, his grandchildren, and so on. Every once in a while, someone in the family dies or a child is born.

The kingdom has a well-defined order of inheritance that consists of the king as the first member. Let's define the recursive function `Successor(x, curOrder)`, which given a person `x` and the inheritance order so far, returns who should be the next person after `x` in the order of inheritance.

```
Successor(x, curOrder):
    if x has no children or all of x's children are in curOrder:
        if x is the king return null
        else return Successor(x's parent, curOrder)
    else return x's oldest child who's not in curOrder
```

For example, assume we have a kingdom that consists of the king, his children Alice and Bob (Alice is older than Bob), and finally Alice's son Jack.

1. In the beginning, `curOrder` will be `["king"]`.
2. Calling `Successor(king, curOrder)` will return Alice, so we append to `curOrder` to get `["king", "Alice"]`.
3. Calling `Successor(Alice, curOrder)` will return Jack, so we append to `curOrder` to get `["king", "Alice", "Jack"]`.

4. Calling `Successor(Jack, curOrder)` will return Bob, so we append to `curOrder` to get `["king", "Alice", "Jack", "Bob"]`.
5. Calling `Successor(Bob, curOrder)` will return `null`. Thus the order of inheritance will be `["king", "Alice", "Jack", "Bob"]`.

Using the above function, we can always obtain a unique order of inheritance.

Implement the `ThroneInheritance` class:

- `ThroneInheritance(string kingName)` Initializes an object of the `ThroneInheritance` class. The name of the king is given as part of the constructor.
- `void birth(string parentName, string childName)` Indicates that `parentName` gave birth to `childName`.
- `void death(string name)` Indicates the death of `name`. The death of the person doesn't affect the `Successor` function nor the current inheritance order. You can treat it as just marking the person as dead.
- `string[] getInheritanceOrder()` Returns a list representing the current order of inheritance **excluding** dead people.

Example 1:

```
Input
["ThroneInheritance", "birth", "birth", "birth", "birth", "birth", "birth",
"getInheritanceOrder", "death", "getInheritanceOrder"]
[["king"], ["king", "andy"], ["king", "bob"], ["king", "catherine"], ["andy",
"matthew"], ["bob", "alex"], ["bob", "asha"], [null], ["bob"], [null]]
Output
[null, null, null, null, null, null, null, ["king", "andy", "matthew", "bob", "alex",
"asha", "catherine"], null, ["king", "andy", "matthew", "alex", "asha", "catherine"]]

Explanation
ThroneInheritance t= new ThroneInheritance("king"); // order:king
t.birth("king", "andy"); // order: king >andy
t.birth("king", "bob"); // order: king > andy >bob
t.birth("king", "catherine"); // order: king > andy > bob >catherine
t.birth("andy", "matthew"); // order: king > andy >matthew > bob > catherine
t.birth("bob", "alex"); // order: king > andy > matthew > bob >alex > catherine
t.birth("bob", "asha"); // order: king > andy > matthew > bob > alex >asha > catherine
t.getInheritanceOrder(); // return ["king", "andy", "matthew", "bob", "alex", "asha",
"catherine"]
t.death("bob"); // order: king > andy > matthew >bob > alex > asha > catherine
t.getInheritanceOrder(); // return ["king", "andy", "matthew", "alex", "asha",
"catherine"]
```

Constraints:

- `1 <= kingName.length, parentName.length, childName.length, name.length <= 15`
- `kingName`, `parentName`, `childName`, and `name` consist of lowercase English letters only.

- All arguments `childName` and `kingName` are **distinct**.
- All `name` arguments of `death` will be passed to either the constructor or as `childName` to `birth` first.
- For each call to `birth(parentName, childName)`, it is guaranteed that `parentName` is alive.
- At most `105` calls will be made to `birth` and `death`.
- At most `10` calls will be made to `getInheritanceOrder`.

题目大意

一个王国里住着国王、他的孩子们、他的孙子们等等。每一个时间点，这个家庭里有人出生也有人死亡。这个王国有一个明确规定的皇位继承顺序，第一继承人总是国王自己。我们定义递归函数 Successor(x, curOrder) ，给定一个人 x 和当前的继承顺序，该函数返回 x 的下一继承人。

解题思路

- 这道题思路不难。先将国王每个孩子按照顺序存在一个 map 中，然后每个国王的孩子还存在父子关系，同理也按顺序存在 map 中。执行 GetInheritanceOrder() 函数时，将国王的孩子按顺序遍历，如果每个孩子还有孩子，递归遍历到底。如果把继承关系看成一棵树，此题便是多叉树的先根遍历的问题。

代码

```go
package leetcode

type ThroneInheritance struct {
  king  string
  edges map[string][]string
  dead  map[string]bool
}

func Constructor(kingName string) (t ThroneInheritance) {
  return ThroneInheritance{kingName, map[string][]string{}, map[string]bool{}}
}

func (t *ThroneInheritance) Birth(parentName, childName string) {
  t.edges[parentName] = append(t.edges[parentName], childName)
}

func (t *ThroneInheritance) Death(name string) {
  t.dead[name] = true
}

func (t *ThroneInheritance) GetInheritanceOrder() (res []string) {
  var preorder func(string)
  preorder = func(name string) {
    if !t.dead[name] {
      res = append(res, name)
    }
    for _, childName := range t.edges[name] {
      preorder(childName)
```

```
    }
  }
  preorder(t.king)
  return
}

/**
 * Your ThroneInheritance object will be instantiated and called as such:
 * obj := Constructor(kingName);
 * obj.Birth(parentName,childName);
 * obj.Death(name);
 * param_3 := obj.GetInheritanceOrder();
 */
```

1603. Design Parking System

题目

Design a parking system for a parking lot. The parking lot has three kinds of parking spaces: big, medium, and small, with a fixed number of slots for each size.

Implement the `ParkingSystem` class:

- `ParkingSystem(int big, int medium, int small)` Initializes object of the `ParkingSystem` class. The number of slots for each parking space are given as part of the constructor.
- `bool addCar(int carType)` Checks whether there is a parking space of `carType` for the car that wants to get into the parking lot. `carType` can be of three kinds: big, medium, or small, which are represented by `1`, `2`, and `3` respectively. **A car can only park in a parking space of its** `carType`. If there is no space available, return `false`, else park the car in that size space and return `true`.

Example 1:

```
Input
["ParkingSystem", "addCar", "addCar", "addCar", "addCar"]
[[1, 1, 0], [1], [2], [3], [1]]
Output
[null, true, true, false, false]

Explanation
ParkingSystem parkingSystem = new ParkingSystem(1, 1, 0);
parkingSystem.addCar(1); // return true because there is 1 available slot for a big car
parkingSystem.addCar(2); // return true because there is 1 available slot for a medium car
parkingSystem.addCar(3); // return false because there is no available slot for a small car
parkingSystem.addCar(1); // return false because there is no available slot for a big car. It is already occupied.
```

Constraints:

- `0 <= big, medium, small <= 1000`
- `carType` is `1`, `2`, or `3`
- At most `1000` calls will be made to `addCar`

题目大意

请你给一个停车场设计一个停车系统。停车场总共有三种不同大小的车位：大，中和小，每种尺寸分别有固定数目的车位。

请你实现 ParkingSystem 类：

- ParkingSystem(int big, int medium, int small) 初始化 ParkingSystem 类，三个参数分别对应每种停车位的数目。
- bool addCar(int carType) 检查是否有 carType 对应的停车位。 carType 有三种类型：大，中，小，分别用数字 1， 2 和 3 表示。一辆车只能停在 carType 对应尺寸的停车位中。如果没有空车位，请返回 false ，否则将该车停入车位并返回 true 。

解题思路

- 简单题。分别用 3 个变量表示大，中和小车位。`addCar()` 判断这 3 个变量是否还有空车位即可。

代码

```go
package leetcode

type ParkingSystem struct {
  Big    int
  Medium int
  Small  int
}

func Constructor(big int, medium int, small int) ParkingSystem {
  return ParkingSystem{
    Big:    big,
    Medium: medium,
    Small:  small,
  }
}

func (this *ParkingSystem) AddCar(carType int) bool {
  switch carType {
  case 1:
    {
      if this.Big > 0 {
        this.Big--
        return true
      }
```

```
        return false
      }
    case 2:
      {
        if this.Medium > 0 {
          this.Medium--
          return true
        }
        return false
      }
    case 3:
      {
        if this.Small > 0 {
          this.Small--
          return true
        }
        return false
      }
    }
    return false
}

/**
 * Your ParkingSystem object will be instantiated and called as such:
 * obj := Constructor(big, medium, small);
 * param_1 := obj.AddCar(carType);
 */
```

1608. Special Array With X Elements Greater Than or Equal X

题目

You are given an array `nums` of non-negative integers. `nums` is considered **special** if there exists a number `x` such that there are **exactly** `x` numbers in `nums` that are **greater than or equal to** `x`.

Notice that `x` **does not** have to be an element in `nums`.

Return `x` *if the array is **special**, otherwise, return* `-1`. It can be proven that if `nums` is special, the value for `x` is **unique**.

Example 1:

```
Input: nums = [3,5]
Output: 2
Explanation: There are 2 values (3 and 5) that are greater than or equal to 2.
```

Example 2:

```
Input: nums = [0,0]
Output: -1
Explanation: No numbers fit the criteria for x.
If x = 0, there should be 0 numbers >= x, but there are 2.
If x = 1, there should be 1 number >= x, but there are 0.
If x = 2, there should be 2 numbers >= x, but there are 0.
x cannot be greater since there are only 2 numbers in nums.
```

Example 3:

```
Input: nums = [0,4,3,0,4]
Output: 3
Explanation: There are 3 values that are greater than or equal to 3.
```

Example 4:

```
Input: nums = [3,6,7,7,0]
Output: -1
```

Constraints:

- `1 <= nums.length <= 100`
- `0 <= nums[i] <= 1000`

题目大意

给你一个非负整数数组 nums 。如果存在一个数 x ， 使得 nums 中恰好有 x 个元素 大于或者等于 x ， 那么就称 nums 是一个 特殊数组 ， 而 x 是该数组的 特征值 。（注意： x 不必 是 nums 的中的元素。）如果数组 nums 是一个 特殊数组 ， 请返回它的特征值 x 。否则，返回 -1 。可以证明的是，如果 nums 是特殊数组，那么其特征值 x 是 唯一的 。

解题思路

- 简单题。抓住题干中给的证明，特征值是唯一的。先将数组从小到大排序，下标的含义与特征值就等价了。下标 `i` 代表大于等于 `nums[i]` 的元素有 `len(nums) - i` 个，那么从第 0 个下标的元素开始遍历，如果这个元素都大于 `len(nums)`，那么后面 `len(nums)` 个元素也都大于等于它，特征值就找到了。如果特征值减一以后，仍然满足 `nums[i] >= x`，说明满足条件的值有多个，这一点不满足特征值唯一性，可以直接返回 -1 了。下标继续右移，特征值继续减一。如果最终循环结束依旧找不到特征值，返回 -1 。

代码

```go
package leetcode

import "sort"

func specialArray(nums []int) int {
  sort.Ints(nums)
```

```
  x := len(nums)
  for _, num := range nums {
    if num >= x {
      return x
    }
    x--
    if num >= x {
      return -1
    }
  }
  return -1
}
```

1614. Maximum Nesting Depth of the Parentheses

题目

A string is a **valid parentheses string** (denoted **VPS**) if it meets one of the following:

- It is an empty string `""`, or a single character not equal to `"("` or `")"`,
- It can be written as `AB` (`A` concatenated with `B`), where `A` and `B` are **VPS**'s, or
- It can be written as `(A)`, where `A` is a **VPS**.

We can similarly define the **nesting depth** `depth(S)` of any VPS `S` as follows:

- `depth("") = 0`
- `depth(C) = 0`, where `C` is a string with a single character not equal to `"("` or `")"`.
- `depth(A + B) = max(depth(A), depth(B))`, where `A` and `B` are **VPS**'s.
- `depth("(" + A + ")") = 1 + depth(A)`, where `A` is a **VPS**.

For example, `""`, `"()()"`, and `"()(()())"` are **VPS**'s (with nesting depths 0, 1, and 2), and `")("` and `"(()"` are not **VPS**'s.

Given a **VPS** represented as string `s`, return *the **nesting depth** of* `s`.

Example 1:

```
Input: s = "(1+(2*3)+((8)/4))+1"
Output: 3
Explanation: Digit 8 is inside of 3 nested parentheses in the string.
```

Example 2:

```
Input: s = "(1)+((2))+(((3)))"
Output: 3
```

Example 3:

```
Input: s = "1+(2*3)/(2-1)"
Output: 1
```

Example 4:

```
Input: s = "1"
Output: 0
```

Constraints:

- `1 <= s.length <= 100`
- `s` consists of digits `0-9` and characters `'+'`, `'-'`, `'*'`, `'/'`, `'('`, and `')'`.
- It is guaranteed that parentheses expression `s` is a **VPS**.

题目大意

如果字符串满足以下条件之一，则可以称之为 有效括号字符串（valid parentheses string，可以简写为 VPS）：

- 字符串是一个空字符串 ""，或者是一个不为 "(" 或 ")" 的单字符。
- 字符串可以写为 AB（A 与 B 字符串连接），其中 A 和 B 都是 有效括号字符串 。
- 字符串可以写为 (A)，其中 A 是一个 有效括号字符串 。

类似地，可以定义任何有效括号字符串 S 的 嵌套深度 depth(S)：

- depth("") = 0
- depth(C) = 0，其中 C 是单个字符的字符串，且该字符不是 "(" 或者 ")"
- depth(A + B) = max(depth(A), depth(B))，其中 A 和 B 都是 有效括号字符串
- depth("(" + A + ")") = 1 + depth(A)，其中 A 是一个 有效括号字符串

例如：""、"()()"、"()(()())" 都是 有效括号字符串（嵌套深度分别为 0、1、2），而 ")(" 、"(()" 都不是 有效括号字符串 。给你一个 有效括号字符串 s，返回该字符串的 s 嵌套深度 。

解题思路

- 简单题。求一个括号字符串的嵌套深度。题目给的括号字符串都是有效的，所以不需要考虑非法的情况。扫描一遍括号字符串，遇到 `(` 可以无脑 ++，并动态维护最大值，遇到 `)` 可以无脑 - - 。最后输出最大值即可。

代码

```go
package leetcode

func maxDepth(s string) int {
  res, cur := 0, 0
  for _, c := range s {
    if c == '(' {
      cur++
      res = max(res, cur)
    } else if c == ')' {
      cur--
```

```
    }
  }
  return res
}

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

1619. Mean of Array After Removing Some Elements

题目

Given an integer array `arr`, return *the mean of the remaining integers after removing the smallest* `5%` *and the largest* `5%` *of the elements.*

Answers within `10-5` of the **actual answer** will be considered accepted.

Example 1:

```
Input: arr = [1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,3]
Output: 2.00000
Explanation: After erasing the minimum and the maximum values of this array, all
elements are equal to 2, so the mean is 2.
```

Example 2:

```
Input: arr = [6,2,7,5,1,2,0,3,10,2,5,0,5,5,0,8,7,6,8,0]
Output: 4.00000
```

Example 3:

```
Input: arr =
[6,0,7,0,7,5,7,8,3,4,0,7,8,1,6,8,1,1,2,4,8,1,9,5,4,3,8,5,10,8,6,6,1,0,6,10,8,2,3,4]
Output: 4.77778
```

Example 4:

```
Input: arr =
[9,7,8,7,7,8,4,4,6,8,8,7,6,8,8,9,2,6,0,0,1,10,8,6,3,3,5,1,10,9,0,7,10,0,10,4,1,10,6,9,3
,6,0,0,2,7,0,6,7,2,9,7,7,3,0,1,6,1,10,3]
Output: 5.27778
```

Example 5:

```
Input: arr =
[4,8,4,10,0,7,1,3,7,8,8,3,4,1,6,2,1,1,8,0,9,8,0,3,9,10,3,10,1,10,7,3,2,1,4,9,10,7,6,4,0
,8,5,1,2,1,6,2,5,0,7,10,9,10,3,7,10,5,8,5,7,6,7,6,10,9,5,10,5,5,7,2,10,7,7,8,2,0,1,1]
Output: 5.29167
```

Constraints:

- `20 <= arr.length <= 1000`
- `arr.length` **is a multiple** of `20`.
- `0 <= arr[i] <= 10^5`

题目大意

给你一个整数数组 `arr` ，请你删除最小 `5%` 的数字和最大 `5%` 的数字后，剩余数字的平均值。与 **标准答案** 误差在 `10-5` 的结果都被视为正确结果。

解题思路

- 简单题。先将数组排序，然后累加中间 90% 的元素，最后计算平均值。

代码

```go
package leetcode

import "sort"

func trimMean(arr []int) float64 {
  sort.Ints(arr)
  n, sum := len(arr), 0
  for i := n / 20; i < n-(n/20); i++ {
    sum += arr[i]
  }
  return float64(sum) / float64((n - (n / 10)))
}
```

1624. Largest Substring Between Two Equal Characters

题目

Given a string `s`, return *the length of the longest substring between two equal characters, excluding the two characters.* If there is no such substring return `-1`.

A **substring** is a contiguous sequence of characters within a string.

Example 1:

```
Input: s = "aa"
Output: 0
Explanation: The optimal substring here is an empty substring between the two 'a's.
```

Example 2:

```
Input: s = "abca"
Output: 2
Explanation: The optimal substring here is "bc".
```

Example 3:

```
Input: s = "cbzxy"
Output: -1
Explanation: There are no characters that appear twice in s.
```

Example 4:

```
Input: s = "cabbac"
Output: 4
Explanation: The optimal substring here is "abba". Other non-optimal substrings include
"bb" and "".
```

Constraints:

- `1 <= s.length <= 300`
- `s` contains only lowercase English letters.

题目大意

给你一个字符串 s，请你返回 两个相同字符之间的最长子字符串的长度 ，计算长度时不含这两个字符。如果不存在这样的子字符串，返回 -1 。子字符串 是字符串中的一个连续字符序列。

解题思路

- 简单题。取每个字符，扫描一次字符串，如果在字符串中还能找到相同的字符，则返回最末尾的那个字符，计算这两个字符之间的距离。取最末尾的字符是为了让两个相同的字符距离最长。扫描字符串过程中动态维护最长长度。如果字符串中不存在 2 个相同的字符，则返回 -1 。

代码

```go
package leetcode

import "strings"
```

```go
func maxLengthBetweenEqualCharacters(s string) int {
  res := -1
  for k, v := range s {
    tmp := strings.LastIndex(s, string(v))
    if tmp > 0 {
      if res < tmp-k-1 {
        res = tmp - k - 1
      }
    }
  }
  return res
}
```

1629. Slowest Key

题目

A newly designed keypad was tested, where a tester pressed a sequence of `n` keys, one at a time.

You are given a string `keysPressed` of length `n`, where `keysPressed[i]` was the `ith` key pressed in the testing sequence, and a sorted list `releaseTimes`, where `releaseTimes[i]` was the time the `ith` key was released. Both arrays are **0-indexed**. The `0th` key was pressed at the time `0`, and every subsequent key was pressed at the **exact** time the previous key was released.

The tester wants to know the key of the keypress that had the **longest duration**. The `ith` keypress had a **duration** of `releaseTimes[i] - releaseTimes[i - 1]`, and the `0th` keypress had a duration of `releaseTimes[0]`.

Note that the same key could have been pressed multiple times during the test, and these multiple presses of the same key **may not** have had the same **duration**.

*Return the key of the keypress that had the **longest duration**. If there are multiple such keypresses, return the lexicographically largest key of the keypresses.*

Example 1:

```
Input: releaseTimes = [9,29,49,50], keysPressed = "cbcd"
Output: "c"
Explanation: The keypresses were as follows:
Keypress for 'c' had a duration of 9 (pressed at time 0 and released at time 9).
Keypress for 'b' had a duration of 29 - 9 = 20 (pressed at time 9 right after the
release of the previous character and released at time 29).
Keypress for 'c' had a duration of 49 - 29 = 20 (pressed at time 29 right after the
release of the previous character and released at time 49).
Keypress for 'd' had a duration of 50 - 49 = 1 (pressed at time 49 right after the
release of the previous character and released at time 50).
The longest of these was the keypress for 'b' and the second keypress for 'c', both
with duration 20.
'c' is lexicographically larger than 'b', so the answer is 'c'.
```

Example 2:

```
Input: releaseTimes = [12,23,36,46,62], keysPressed = "spuda"
Output: "a"
Explanation: The keypresses were as follows:
Keypress for 's' had a duration of 12.
Keypress for 'p' had a duration of 23 - 12 = 11.
Keypress for 'u' had a duration of 36 - 23 = 13.
Keypress for 'd' had a duration of 46 - 36 = 10.
Keypress for 'a' had a duration of 62 - 46 = 16.
The longest of these was the keypress for 'a' with duration 16.
```

Constraints:

- `releaseTimes.length == n`
- `keysPressed.length == n`
- `2 <= n <= 1000`
- `1 <= releaseTimes[i] <= 109`
- `releaseTimes[i] < releaseTimes[i+1]`
- `keysPressed` contains only lowercase English letters.

题目大意

LeetCode 设计了一款新式键盘，正在测试其可用性。测试人员将会点击一系列键（总计 n 个），每次一个。

给你一个长度为 n 的字符串 keysPressed ，其中 keysPressed[i] 表示测试序列中第 i 个被按下的键。releaseTimes 是一个升序排列的列表，其中 releaseTimes[i] 表示松开第 i 个键的时间。字符串和数组的 下标都从 0 开始 。第 0 个键在时间为 0 时被按下，接下来每个键都 恰好 在前一个键松开时被按下。测试人员想要找出按键持续时间最长 的键。第 i 次按键的持续时间为 releaseTimes[i] - releaseTimes[i - 1] ，第 0 次按键的持续时间为 releaseTimes[0] 。

注意，测试期间，同一个键可以在不同时刻被多次按下，而每次的持续时间都可能不同。请返回按键 持续时间最长 的键，如果有多个这样的键，则返回 按字母顺序排列最大 的那个键。

解题思路

- 题干很长，不过还是简单题。循环扫描一遍数组，计算出每个按键的持续时间。动态更新按键最长时间的键。如果持续时间最长的有多个键，需要返回字母序最大的那一个键。

代码

```go
package leetcode

func slowestKey(releaseTimes []int, keysPressed string) byte {
  longestDuration, key := releaseTimes[0], keysPressed[0]
  for i := 1; i < len(releaseTimes); i++ {
    duration := releaseTimes[i] - releaseTimes[i-1]
    if duration > longestDuration {
```

```
        longestDuration = duration
        key = keysPressed[i]
      } else if duration == longestDuration && keysPressed[i] > key {
        key = keysPressed[i]
      }
    }
    return key
  }
```

1631. Path With Minimum Effort

题目

You are a hiker preparing for an upcoming hike. You are given `heights`, a 2D array of size `rows x columns`, where `heights[row][col]` represents the height of cell `(row, col)`. You are situated in the top-left cell, `(0, 0)`, and you hope to travel to the bottom-right cell, `(rows-1, columns-1)` (i.e., **0-indexed**). You can move **up**, **down**, **left**, or **right**, and you wish to find a route that requires the minimum **effort**.

A route's **effort** is the **maximum absolute difference** in heights between two consecutive cells of the route.

Return *the minimum **effort** required to travel from the top-left cell to the bottom-right cell.*

Example 1:

```
Input: heights = [[1,2,2],[3,8,2],[5,3,5]]
Output: 2
Explanation: The route of [1,3,5,3,5] has a maximum absolute difference of 2 in
consecutive cells.
This is better than the route of [1,2,2,2,5], where the maximum absolute difference is
3.
```

Example 2:

```
Input: heights = [[1,2,3],[3,8,4],[5,3,5]]
Output: 1
Explanation: The route of [1,2,3,4,5] has a maximum absolute difference of 1 in
consecutive cells, which is better than route [1,3,5,3,5].
```

Example 3:

```
Input: heights = [[1,2,1,1,1],[1,2,1,2,1],[1,2,1,2,1],[1,2,1,2,1],[1,1,1,2,1]]
Output: 0
Explanation: This route does not require any effort.
```

Constraints:

- `rows == heights.length`
- `columns == heights[i].length`
- `1 <= rows, columns <= 100`
- `1 <= heights[i][j] <= 10^6`

题目大意

你准备参加一场远足活动。给你一个二维 `rows x columns` 的地图 `heights` ，其中 `heights[row][col]` 表示格子 `(row, col)` 的高度。一开始你在最左上角的格子 `(0, 0)` ，且你希望去最右下角的格子 `(rows-1, columns-1)` （注意下标从 0 开始编号）。你每次可以往 上，下，左，右 四个方向之一移动，你想要找到耗费 体力 最小的一条路径。一条路径耗费的 体力值 是路径上相邻格子之间 高度差绝对值 的 最大值 决定的。请你返回从左上角走到右下角的最小 体力消耗值 。

解题思路

- 此题和第 778 题解题思路完全一致。在第 778 题中求的是最短连通时间。此题求的是连通路径下的最小体力值。都是求的最小值，只是 2 个值的意义不同罢了。
- 按照第 778 题的思路，本题也有多种解法。第一种解法是 DFS + 二分。先将题目变换一个等价问法。题目要求找到最小体力消耗值，也相当于问是否存在一个体力消耗值 x，只要大于等于 x，一定能连通。利用二分搜索来找到这个临界值。体力消耗值是有序的，此处满足二分搜索的条件。题目给定柱子高度是 [1,10^6]，所以体力值一定在 [0,10^6-1] 这个区间内。判断是否取中值的条件是用 DFS 或者 BFS 搜索 (0,0) 点和 (N-1, N-1) 点之间是否连通。时间复杂度：O(mnlogC)，其中 m 和 n 分别是地图的行数和列数，C 是格子的最大高度。C 最大为 10^6，所以 logC 常数也很小。空间复杂度 O(mn)。
- 第二种解法是并查集。将图中所有边按照权值从小到大进行排序，并依次加入并查集中。直到加入一条权值为 x 的边以后，左上角到右下角连通了。最小体力消耗值也就找到了。注意加入边的时候，只加入 `i-1` 和 `i` ，`j-1` 和 `j` 这 2 类相邻的边。因为最小体力消耗意味着不走回头路。上下左右四个方向到达一个节点，只可能从上边和左边走过来。从下边和右边走过来肯定是浪费体力了。时间复杂度：O(mnlog(mn))，其中 m 和 n 分别是地图的行数和列数，图中的边数为 O(mn)。空间复杂度 O(mn)，即为存储所有边以及并查集需要的空间。

代码

```go
package leetcode

import (
  "sort"

  "github.com/halfrost/LeetCode-Go/template"
)

var dir = [4][2]int{
  {0, 1},
  {1, 0},
  {0, -1},
  {-1, 0},
```

```go
}

// 解法一 DFS + 二分
func minimumEffortPath(heights [][]int) int {
  n, m := len(heights), len(heights[0])
  visited := make([][]bool, n)
  for i := range visited {
    visited[i] = make([]bool, m)
  }
  low, high := 0, 1000000
  for low < high {
    threshold := low + (high-low)>>1
    if !hasPath(heights, visited, 0, 0, threshold) {
      low = threshold + 1
    } else {
      high = threshold
    }
    for i := range visited {
      for j := range visited[i] {
        visited[i][j] = false
      }
    }
  }
  return low
}

func hasPath(heights [][]int, visited [][]bool, i, j, threshold int) bool {
  n, m := len(heights), len(heights[0])
  if i == n-1 && j == m-1 {
    return true
  }
  visited[i][j] = true
  res := false
  for _, d := range dir {
    ni, nj := i+d[0], j+d[1]
    if ni < 0 || ni >= n || nj < 0 || nj >= m || visited[ni][nj] || res {
      continue
    }
    diff := abs(heights[i][j] - heights[ni][nj])
    if diff <= threshold && hasPath(heights, visited, ni, nj, threshold) {
      res = true
    }
  }
  return res
}

func abs(a int) int {
  if a < 0 {
    a = -a
```

```go
  }
  return a
}

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

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

// 解法二 并查集
func minimumEffortPath1(heights [][]int) int {
  n, m, edges, uf := len(heights), len(heights[0]), []edge{}, template.UnionFind{}
  uf.Init(n * m)
  for i, row := range heights {
    for j, h := range row {
      id := i*m + j
      if i > 0 {
        edges = append(edges, edge{id - m, id, abs(h - heights[i-1][j])})
      }
      if j > 0 {
        edges = append(edges, edge{id - 1, id, abs(h - heights[i][j-1])})
      }
    }
  }
  sort.Slice(edges, func(i, j int) bool { return edges[i].diff < edges[j].diff })
  for _, e := range edges {
    uf.Union(e.v, e.w)
    if uf.Find(0) == uf.Find(n*m-1) {
      return e.diff
    }
  }
  return 0
}

type edge struct {
  v, w, diff int
}
```

1636. Sort Array by Increasing Frequency

题目

Given an array of integers `nums`, sort the array in **increasing** order based on the frequency of the values. If multiple values have the same frequency, sort them in **decreasing** order.

Return the *sorted array*.

Example 1:

```
Input: nums = [1,1,2,2,2,3]
Output: [3,1,1,2,2,2]
Explanation: '3' has a frequency of 1, '1' has a frequency of 2, and '2' has a
frequency of 3.
```

Example 2:

```
Input: nums = [2,3,1,3,2]
Output: [1,3,3,2,2]
Explanation: '2' and '3' both have a frequency of 2, so they are sorted in decreasing
order.
```

Example 3:

```
Input: nums = [-1,1,-6,4,5,-6,1,4,1]
Output: [5,-1,4,4,-6,-6,1,1,1]
```

Constraints:

- `1 <= nums.length <= 100`
- `100 <= nums[i] <= 100`

题目大意

给你一个整数数组 `nums` ，请你将数组按照每个值的频率 **升序** 排序。如果有多个值的频率相同，请你按照数值本身将它们 **降序** 排序。请你返回排序后的数组。

解题思路

- 简单题。先统计每个值的频率，然后按照频率从小到大排序，相同频率的按照值的大小，从大到小排序。

代码

```go
package leetcode

import "sort"

func frequencySort(nums []int) []int {
  freq := map[int]int{}
```

```go
  for _, v := range nums {
    freq[v]++
  }
  sort.Slice(nums, func(i, j int) bool {
    if freq[nums[i]] == freq[nums[j]] {
      return nums[j] < nums[i]
    }
    return freq[nums[i]] < freq[nums[j]]
  })
  return nums
}
```

1640. Check Array Formation Through Concatenation

题目

You are given an array of **distinct** integers `arr` and an array of integer arrays `pieces`, where the integers in `pieces` are **distinct**. Your goal is to form `arr` by concatenating the arrays in `pieces` **in any order**. However, you are **not** allowed to reorder the integers in each array `pieces[i]`.

Return `true` *if it is possible to form the array* `arr` *from* `pieces`. Otherwise, return `false`.

Example 1:

```
Input: arr = [85], pieces = [[85]]
Output: true
```

Example 2:

```
Input: arr = [15,88], pieces = [[88],[15]]
Output: true
Explanation: Concatenate [15] then [88]
```

Example 3:

```
Input: arr = [49,18,16], pieces = [[16,18,49]]
Output: false
Explanation: Even though the numbers match, we cannot reorder pieces[0].
```

Example 4:

```
Input: arr = [91,4,64,78], pieces = [[78],[4,64],[91]]
Output: true
Explanation: Concatenate [91] then [4,64] then [78]
```

Example 5:

```
Input: arr = [1,3,5,7], pieces = [[2,4,6,8]]
Output: false
```

Constraints:

- `1 <= pieces.length <= arr.length <= 100`
- `sum(pieces[i].length) == arr.length`
- `1 <= pieces[i].length <= arr.length`
- `1 <= arr[i], pieces[i][j] <= 100`
- The integers in `arr` are **distinct**.
- The integers in `pieces` are **distinct** (i.e., If we flatten pieces in a 1D array, all the integers in this array are distinct).

题目大意

给你一个整数数组 arr ， 数组中的每个整数 互不相同 。另有一个由整数数组构成的数组 pieces，其中的整数也 互不相同 。请你以 任意顺序 连接 pieces 中的数组以形成 arr 。但是，不允许 对每个数组 pieces[i] 中的整数重新排序。如果可以连接 pieces 中的数组形成 arr ， 返回 true ； 否则，返回 false 。

解题思路

- 简单题。题目保证了 arr 中的元素唯一，所以可以用 map 把每个元素的 index 存起来，方便查找。遍历 pieces 数组，在每个一维数组中判断元素顺序是否和原 arr 元素相对顺序一致。这个时候就用 map 查找，如果顺序是一一相连的，那么就是正确的。有一个顺序不是一一相连，或者出现了 arr 不存在的元素，都返回 false。

代码

```go
package leetcode

func canFormArray(arr []int, pieces [][]int) bool {
  arrMap := map[int]int{}
  for i, v := range arr {
    arrMap[v] = i
  }
  for i := 0; i < len(pieces); i++ {
    order := -1
    for j := 0; j < len(pieces[i]); j++ {
      if _, ok := arrMap[pieces[i][j]]; !ok {
        return false
      }
      if order == -1 {
        order = arrMap[pieces[i][j]]
      } else {
        if arrMap[pieces[i][j]] == order+1 {
```

```
            order = arrMap[pieces[i][j]]
          } else {
            return false
          }
        }
      }
    }
    return true
  }
```

1641. Count Sorted Vowel Strings

题目

Given an integer `n`, return *the number of strings of length* `n` *that consist only of vowels (*`a`*,* `e`*,* `i`*,* `o`*,* `u`*) and are* ***lexicographically sorted****.*

A string `s` is **lexicographically sorted** if for all valid `i`, `s[i]` is the same as or comes before `s[i+1]` in the alphabet.

Example 1:

```
Input: n = 1
Output: 5
Explanation: The 5 sorted strings that consist of vowels only are
["a","e","i","o","u"].
```

Example 2:

```
Input: n = 2
Output: 15
Explanation: The 15 sorted strings that consist of vowels only are
["aa","ae","ai","ao","au","ee","ei","eo","eu","ii","io","iu","oo","ou","uu"].
Note that "ea" is not a valid string since 'e' comes after 'a' in the alphabet.
```

Example 3:

```
Input: n = 33
Output: 66045
```

Constraints:

- `1 <= n <= 50`

题目大意

给你一个整数 n，请返回长度为 n 、仅由元音 (a, e, i, o, u) 组成且按 字典序排列 的字符串数量。

字符串 s 按 字典序排列 需要满足：对于所有有效的 i，s[i] 在字母表中的位置总是与 s[i+1] 相同或在 s[i+1] 之前。

解题思路

- 题目给的数据量并不大，第一个思路是利用 DFS 遍历打表法。时间复杂度 O(1)，空间复杂度 O(1)。
- 第二个思路是利用数学中的组合公式计算结果。题目等价于假设现在有 n 个字母，要求取 4 次球（可以选择不取）将字母分为 5 堆，问有几种取法。确定了取法以后，`a`，`e`，`i`，`o`，`u`，每个字母的个数就确定了，据题意要求按照字母序排序，那么最终字符串也就确定了。现在关注解决这个组合问题就可以了。把问题再转化一次，等价于，有 n+4 个字母，取 4 次，问有几种取法。+4 代表 4 个空操作，取走它们意味着不取。根据组合的数学定义，答案为 C(n+4,4)。

代码

```
package leetcode

// 解法一 打表
func countVowelStrings(n int) int {
  res := []int{1, 5, 15, 35, 70, 126, 210, 330, 495, 715, 1001, 1365, 1820, 2380, 3060, 3876, 4845, 5985, 7315, 8855, 10626, 12650, 14950, 17550, 20475, 23751, 27405, 31465, 35960, 40920, 46376, 52360, 58905, 66045, 73815, 82251, 91390, 101270, 111930, 123410, 135751, 148995, 163185, 178365, 194580, 211876, 230300, 249900, 270725, 292825, 316251}
  return res[n]
}

func makeTable() []int {
  res, array := 0, []int{}
  for i := 0; i < 51; i++ {
    countVowelStringsDFS(i, 0, []string{}, []string{"a", "e", "i", "o", "u"}, &res)
    array = append(array, res)
    res = 0
  }
  return array
}

func countVowelStringsDFS(n, index int, cur []string, vowels []string, res *int) {
  vowels = vowels[index:]
  if len(cur) == n {
    (*res)++
    return
  }
  for i := 0; i < len(vowels); i++ {
    cur = append(cur, vowels[i])
    countVowelStringsDFS(n, i, cur, vowels, res)
    cur = cur[:len(cur)-1]
  }
}
```

```go
// 解法二 数学方法 -- 组合
func countVowelStrings1(n int) int {
  return (n + 1) * (n + 2) * (n + 3) * (n + 4) / 24
}
```

1642. Furthest Building You Can Reach

题目

You are given an integer array `heights` representing the heights of buildings, some `bricks`, and some `ladders`.

You start your journey from building `0` and move to the next building by possibly using bricks or ladders.

While moving from building `i` to building `i+1` (**0-indexed**),

- If the current building's height is **greater than or equal** to the next building's height, you do **not** need a ladder or bricks.
- If the current building's height is **less than** the next building's height, you can either use **one ladder** or `(h[i+1] - h[i])` **bricks**.

Return the furthest building index (0-indexed) you can reach if you use the given ladders and bricks optimally.

Example 1:

```
Input: heights = [4,2,7,6,9,14,12], bricks = 5, ladders = 1
Output: 4
Explanation: Starting at building 0, you can follow these steps:
- Go to building 1 without using ladders nor bricks since 4 >= 2.
- Go to building 2 using 5 bricks. You must use either bricks or ladders because 2 < 7.
- Go to building 3 without using ladders nor bricks since 7 >= 6.
- Go to building 4 using your only ladder. You must use either bricks or ladders
because 6 < 9.
It is impossible to go beyond building 4 because you do not have any more bricks or
ladders.
```

Example 2:

```
Input: heights = [4,12,2,7,3,18,20,3,19], bricks = 10, ladders = 2
Output: 7
```

Example 3:

```
Input: heights = [14,3,19,3], bricks = 17, ladders = 0
Output: 3
```

Constraints:

- `1 <= heights.length <= 10^5`
- `1 <= heights[i] <= 10^6`
- `0 <= bricks <= 10^9`
- `0 <= ladders <= heights.length`

题目大意

给你一个整数数组 heights ，表示建筑物的高度。另有一些砖块 bricks 和梯子 ladders 。你从建筑物 0 开始旅程，不断向后面的建筑物移动，期间可能会用到砖块或梯子。当从建筑物 i 移动到建筑物 i+1（下标 从 0 开始 ）时：

- 如果当前建筑物的高度 大于或等于 下一建筑物的高度，则不需要梯子或砖块。
- 如果当前建筑的高度 小于 下一个建筑的高度，您可以使用 一架梯子 或 (h[i+1] - h[i]) 个砖块

如果以最佳方式使用给定的梯子和砖块，返回你可以到达的最远建筑物的下标（下标 从 0 开始 ）。

解题思路

- 这一题可能会想到贪心算法。梯子很厉害，可以无限长，所以梯子用来跨越最高的楼。遇到非最高的距离差，先用砖头。这样贪心的话不正确。例如，[1, 5, 1, 2, 3, 4, 10000] 这组数据，梯子有 1 个，4 块砖头。最大的差距在 10000 和 4 之间，贪心选择在此处用梯子。但是砖头不足以让我们走到最后两栋楼。贪心得到的结果是 3，正确的结果是 5，先用梯子，再用砖头走过 3，4，5 号楼。
- 上面的贪心解法错误在于没有“动态”的贪心，使用梯子应该选择能爬过楼里面最高的 2 个。于是顺理成章的想到了优先队列。维护一个长度为梯子个数的最小堆，当队列中元素超过梯子个数，便将队首最小值出队，出队的这个楼与楼的差距用砖头填补。所有砖头用完了，即是可以到达的最远楼号。

代码

```go
package leetcode

import (
  "container/heap"
)

func furthestBuilding(heights []int, bricks int, ladder int) int {
  usedLadder := &heightDiffPQ{}
  for i := 1; i < len(heights); i++ {
    needbricks := heights[i] - heights[i-1]
    if needbricks < 0 {
      continue
    }
    if ladder > 0 {
      heap.Push(usedLadder, needbricks)
```

```go
        ladder--
      } else {
        if len(*usedLadder) > 0 && needbricks > (*usedLadder)[0] {
          needbricks, (*usedLadder)[0] = (*usedLadder)[0], needbricks
          heap.Fix(usedLadder, 0)
        }
        if bricks -= needbricks; bricks < 0 {
          return i - 1
        }
      }
    }
    return len(heights) - 1
}

type heightDiffPQ []int

func (pq heightDiffPQ) Len() int            { return len(pq) }
func (pq heightDiffPQ) Less(i, j int) bool  { return pq[i] < pq[j] }
func (pq heightDiffPQ) Swap(i, j int)       { pq[i], pq[j] = pq[j], pq[i] }
func (pq *heightDiffPQ) Push(x interface{}) { *pq = append(*pq, x.(int)) }
func (pq *heightDiffPQ) Pop() interface{} {
  x := (*pq)[len(*pq)-1]
  *pq = (*pq)[:len(*pq)-1]
  return x
}
```

1646. Get Maximum in Generated Array

题目

You are given an integer `n`. An array `nums` of length `n + 1` is generated in the following way:

- `nums[0] = 0`
- `nums[1] = 1`
- `nums[2 * i] = nums[i]` when `2 <= 2 * i <= n`
- `nums[2 * i + 1] = nums[i] + nums[i + 1]` when `2 <= 2 * i + 1 <= n`

Return *the **maximum** integer in the array* `nums`.

Example 1:

```
Input: n = 7
Output: 3
Explanation: According to the given rules:
  nums[0] = 0
  nums[1] = 1
  nums[(1 * 2) = 2] = nums[1] = 1
  nums[(1 * 2) + 1 = 3] = nums[1] + nums[2] = 1 + 1 = 2
  nums[(2 * 2) = 4] = nums[2] = 1
  nums[(2 * 2) + 1 = 5] = nums[2] + nums[3] = 1 + 2 = 3
  nums[(3 * 2) = 6] = nums[3] = 2
  nums[(3 * 2) + 1 = 7] = nums[3] + nums[4] = 2 + 1 = 3
Hence, nums = [0,1,1,2,1,3,2,3], and the maximum is 3.
```

Example 2:

```
Input: n = 2
Output: 1
Explanation: According to the given rules, the maximum between nums[0], nums[1], and
nums[2] is 1.
```

Example 3:

```
Input: n = 3
Output: 2
Explanation: According to the given rules, the maximum between nums[0], nums[1],
nums[2], and nums[3] is 2.
```

Constraints:

- `0 <= n <= 100`

题目大意

给你一个整数 n 。按下述规则生成一个长度为 n + 1 的数组 nums：

- nums[0] = 0
- nums[1] = 1
- 当 2 <= 2 * i <= n 时，nums[2 * i] = nums[i]
- 当 2 <= 2 * i + 1 <= n 时，nums[2 * i + 1] = nums[i] + nums[i + 1]

返回生成数组 nums 中的 最大值。

解题思路

- 给出一个 n + 1 的数组，并按照生成规则生成这个数组，求出这个数组中的最大值。
- 简单题，按照题意生成数组，边生成边记录和更新最大值即可。

- 注意边界条件，当 n 为 0 的时候，数组里面只有一个元素 0 。

代码

```
package leetcode

func getMaximumGenerated(n int) int {
  if n == 0 {
    return 0
  }
  nums, max := make([]int, n+1), 0
  nums[0], nums[1] = 0, 1
  for i := 0; i <= n; i++ {
    if nums[i] > max {
      max = nums[i]
    }
    if 2*i >= 2 && 2*i <= n {
      nums[2*i] = nums[i]
    }
    if 2*i+1 >= 2 && 2*i+1 <= n {
      nums[2*i+1] = nums[i] + nums[i+1]
    }
  }
  return max
}
```

1647. Minimum Deletions to Make Character Frequencies Unique

题目

A string `s` is called **good** if there are no two different characters in `s` that have the same **frequency**.

Given a string `s`, return *the **minimum** number of characters you need to delete to make* `s` ***good**.*

The **frequency** of a character in a string is the number of times it appears in the string. For example, in the string `"aab"`, the **frequency** of `'a'` is `2`, while the **frequency** of `'b'` is `1`.

Example 1:

```
Input: s = "aab"
Output: 0
Explanation: s is already good.
```

Example 2:

```
Input: s = "aaabbbcc"
Output: 2
Explanation: You can delete two 'b's resulting in the good string "aaabcc".
Another way it to delete one 'b' and one 'c' resulting in the good string "aaabbc".
```

Example 3:

```
Input: s = "ceabaacb"
Output: 2
Explanation: You can delete both 'c's resulting in the good string "eabaab".
Note that we only care about characters that are still in the string at the end (i.e.
frequency of 0 is ignored).
```

Constraints:

- `1 <= s.length <= 105`
- `s` contains only lowercase English letters.

题目大意

如果字符串 s 中 不存在 两个不同字符 频次 相同的情况，就称 s 是 优质字符串 。

给你一个字符串 s，返回使 s 成为优质字符串需要删除的最小字符数。

字符串中字符的 频次 是该字符在字符串中的出现次数。例如，在字符串 "aab" 中，'a' 的频次是 2，而 'b' 的频次是 1 。

提示：

- `1 <= s.length <= 105`
- `s` 仅含小写英文字母

解题思路

- 给出一个字符串 s，要求输出使 s 变成“优质字符串”需要删除的最小字符数。“优质字符串”的定义是：字符串 s 中不存在频次相同的两个不同字符。
- 首先将 26 个字母在字符串中的频次分别统计出来，然后把频次从大到小排列，从频次大的开始，依次调整：例如，假设前一个和后一个频次相等，就把前一个字符删除一个，频次减一，再次排序，如果频次还相等，继续调整，如果频次不同了，游标往后移，继续调整后面的频次。直到所有的频次都不同了，就可以输出最终结果了。
- 这里需要注意频次为 0 的情况，即字母都被删光了。频次为 0 以后，就不需要再比较了。

代码

```go
package leetcode

import (
  "sort"
```

```go
)

func minDeletions(s string) int {
  frequency, res := make([]int, 26), 0
  for i := 0; i < len(s); i++ {
    frequency[s[i]-'a']++
  }
  sort.Sort(sort.Reverse(sort.IntSlice(frequency)))
  for i := 1; i <= 25; i++ {
    if frequency[i] == frequency[i-1] && frequency[i] != 0 {
      res++
      frequency[i]--
      sort.Sort(sort.Reverse(sort.IntSlice(frequency)))
      i--
    }
  }
  return res
}
```

1648. Sell Diminishing-Valued Colored Balls

题目

You have an `inventory` of different colored balls, and there is a customer that wants `orders` balls of **any** color.

The customer weirdly values the colored balls. Each colored ball's value is the number of balls **of that color** you currently have in your `inventory`. For example, if you own `6` yellow balls, the customer would pay `6` for the first yellow ball. After the transaction, there are only `5` yellow balls left, so the next yellow ball is then valued at `5` (i.e., the value of the balls decreases as you sell more to the customer).

You are given an integer array, `inventory`, where `inventory[i]` represents the number of balls of the `ith` color that you initially own. You are also given an integer `orders`, which represents the total number of balls that the customer wants. You can sell the balls **in any order**.

Return *the **maximum** total value that you can attain after selling* `orders` *colored balls*. As the answer may be too large, return it **modulo** `109 + 7`.

Example 1:

```
Input: inventory = [2,5], orders = 4
Output: 14
Explanation: Sell the 1st color 1 time (2) and the 2nd color 3 times (5 + 4 + 3).
The maximum total value is 2 + 5 + 4 + 3 = 14.
```

Example 2:

```
Input: inventory = [3,5], orders = 6
Output: 19
Explanation: Sell the 1st color 2 times (3 + 2) and the 2nd color 4 times (5 + 4 + 3 +
2).
The maximum total value is 3 + 2 + 5 + 4 + 3 + 2 = 19.
```

Example 3:

```
Input: inventory = [2,8,4,10,6], orders = 20
Output: 110
```

Example 4:

```
Input: inventory = [1000000000], orders = 1000000000
Output: 21
Explanation: Sell the 1st color 1000000000 times for a total value of
500000000500000000. 500000000500000000 modulo 109 + 7 = 21.
```

Constraints:

- `1 <= inventory.length <= 10^5`
- `1 <= inventory[i] <= 10^9`
- `1 <= orders <= min(sum(inventory[i]), 10^9)`

题目大意

你有一些球的库存 inventory ，里面包含着不同颜色的球。一个顾客想要 任意颜色 总数为 orders 的球。这位顾客有一种特殊的方式衡量球的价值：每个球的价值是目前剩下的 同色球 的数目。比方说还剩下 6 个黄球，那么顾客买第一个黄球的时候该黄球的价值为 6 。这笔交易以后，只剩下 5 个黄球了，所以下一个黄球的价值为 5 （也就是球的价值随着顾客购买同色球是递减的）

给你整数数组 inventory ，其中 inventory[i] 表示第 i 种颜色球一开始的数目。同时给你整数 orders ，表示顾客总共想买的球数目。你可以按照 任意顺序 卖球。请你返回卖了 orders 个球以后 最大 总价值之和。由于答案可能会很大，请你返回答案对 109 + 7 取余数 的结果。

提示：

- 1 <= inventory.length <= 10^5
- 1 <= inventory[i] <= 10^9
- 1 <= orders <= min(sum(inventory[i]), 10^9)

解题思路

- 给出一个 `inventory` 数组和 `orders` 次操作，要求输出数组中前 `orders` 大个元素累加和。需要注意的是，每累加一个元素 `inventory[i]`，这个元素都会减一，下次再累加的时候，需要选取更新以后的数组的最大值。

- 拿到这个题目以后很容易想到优先队列，建立大根堆以后，`pop` 出当前最大值 `maxItem`，累加，以后把 `maxItem` 减一再 `push` 回去。循环执行 `orders` 次以后即是最终结果。题目是这个意思，但是我们不能这么写代码，因为题目条件里面给出了 `orders` 的数据大小。orders 最大为 10^9。按照优先队列的这个方法一定会超时，时间复杂度为 O(orders·logn)。那就换一个思路。优先队列这个思路中，重复操作了 `orders` 次，其实在这些操作中，有一些是没有必要的废操作。这些大量的“废”操作导致了超时。试想，在 `orders` 次操作中，能否合并 `n` 个 `pop` 操作，一口气先 `pop` 掉 `n` 个前 `n` 大的数呢？这个是可行的，因为每次 `pop` 出去，元素都只会减一，这个是非常有规律的。
- 为了接下来的描述更加清晰易懂，还需要再定义 1 个值， `thresholdValue` 为操作 `n` 次以后，当前 `inventory` 数组的最大值。关于 `thresholdValue` 的理解，这里要说明一下。 `thresholdValue` 的来源有 2 种，一种是本来数组里面就有这个值，还有一种来源是 `inventory[i]` 元素减少到了 `thresholdValue` 这个值。举个例子：原始数组是 [2,3,3,4,5]， `orders` = 4，取 4 次以后，剩下的数组是 [2,2,3,3,3]。3 个 3 里面其中一个 3 就来自于 `4-1=3`，或者 `5-2=3`。
- 用二分搜索在 [0，max(`inventory`)] 区间内找到这个 `thresholdValue` 值，能满足下列不等式的最小 `thresholdValue` 值：
  {{< katex display >}}
  \sum_{inventory[i]\geqslant thresholdValue}^{} \left (inventory[i] - thresholdValue \right)\leqslant orders
  {{< /katex >}}
  `thresholdValue` 越小，不等式左边的值越大，随着 `thresholdValue` 的增大，不等式左边的值越来越小，直到刚刚能小于等于 `orders`。求出了 `thresholdValue` 值以后，还需要再判断有多少值等于 `thresholdValue - 1` 值了。

- 还是举上面的例子，原始数组是 [2,3,3,4,5]， `orders` = 4，我们可以求得 `thresholdValue` = 3 。`inventory[i]` > `thresholdValue` 的那部分 100% 的要取走， `thresholdValue` 就像一个水平面，突出水平面的那些都要拿走，每列的值按照等差数列求和公式计算即可。但是 `orders` - `thresholdValue` = 1，说明水平面以下还要拿走一个，即 `thresholdValue` 线下的虚线框里面的那 4 个球，还需要任意取走一个。最后总的结果是这 2 部分的总和，((5 + 4) + 4) + 3 = 16 。

代码

```go
package leetcode

import (
  "container/heap"
)

// 解法一 贪心 + 二分搜索
func maxProfit(inventory []int, orders int) int {
  maxItem, thresholdValue, count, res, mod := 0, -1, 0, 0, 1000000007
  for i := 0; i < len(inventory); i++ {
    if inventory[i] > maxItem {
      maxItem = inventory[i]
    }
  }
```

```
  low, high := 0, maxItem
  for low <= high {
    mid := low + ((high - low) >> 1)
    for i := 0; i < len(inventory); i++ {
      count += max(inventory[i]-mid, 0)
    }
    if count <= orders {
      thresholdValue = mid
      high = mid - 1
    } else {
      low = mid + 1
    }
    count = 0
  }
  count = 0
  for i := 0; i < len(inventory); i++ {
    count += max(inventory[i]-thresholdValue, 0)
  }
  count = orders - count
  for i := 0; i < len(inventory); i++ {
    if inventory[i] >= thresholdValue {
      if count > 0 {
        res += (thresholdValue + inventory[i]) * (inventory[i] - thresholdValue + 1) /
2
        count--
      } else {
        res += (thresholdValue + 1 + inventory[i]) * (inventory[i] - thresholdValue) /
2
      }
    }
  }
  return res % mod
}
```

1649. Create Sorted Array through Instructions

题目

Given an integer array `instructions`, you are asked to create a sorted array from the elements in `instructions`. You start with an empty container `nums`. For each element from **left to right** in `instructions`, insert it into `nums`. The **cost** of each insertion is the **minimum** of the following:

- The number of elements currently in `nums` that are **strictly less than** `instructions[i]`.
- The number of elements currently in `nums` that are **strictly greater than** `instructions[i]`.

For example, if inserting element `3` into `nums = [1,2,3,5]`, the **cost** of insertion is `min(2, 1)` (elements `1` and `2` are less than `3`, element `5` is greater than `3`) and `nums` will become `[1,2,3,3,5]`.

Return *the **total cost** to insert all elements from* `instructions` *into* `nums`. Since the answer may be large, return it **modulo** `10^9 + 7`

Example 1:

```
Input: instructions = [1,5,6,2]
Output: 1
Explanation: Begin with nums = [].
Insert 1 with cost min(0, 0) = 0, now nums = [1].
Insert 5 with cost min(1, 0) = 0, now nums = [1,5].
Insert 6 with cost min(2, 0) = 0, now nums = [1,5,6].
Insert 2 with cost min(1, 2) = 1, now nums = [1,2,5,6].
The total cost is 0 + 0 + 0 + 1 = 1.
```

Example 2:

```
Input: instructions = [1,2,3,6,5,4]
Output: 3
Explanation: Begin with nums = [].
Insert 1 with cost min(0, 0) = 0, now nums = [1].
Insert 2 with cost min(1, 0) = 0, now nums = [1,2].
Insert 3 with cost min(2, 0) = 0, now nums = [1,2,3].
Insert 6 with cost min(3, 0) = 0, now nums = [1,2,3,6].
Insert 5 with cost min(3, 1) = 1, now nums = [1,2,3,5,6].
Insert 4 with cost min(3, 2) = 2, now nums = [1,2,3,4,5,6].
The total cost is 0 + 0 + 0 + 0 + 1 + 2 = 3.
```

Example 3:

```
Input: instructions = [1,3,3,3,2,4,2,1,2]
Output: 4
Explanation: Begin with nums = [].
Insert 1 with cost min(0, 0) = 0, now nums = [1].
Insert 3 with cost min(1, 0) = 0, now nums = [1,3].
Insert 3 with cost min(1, 0) = 0, now nums = [1,3,3].
Insert 3 with cost min(1, 0) = 0, now nums = [1,3,3,3].
Insert 2 with cost min(1, 3) = 1, now nums = [1,2,3,3,3].
Insert 4 with cost min(5, 0) = 0, now nums = [1,2,3,3,3,4].
Insert 2 with cost min(1, 4) = 1, now nums = [1,2,2,3,3,3,4].
Insert 1 with cost min(0, 6) = 0, now nums = [1,1,2,2,3,3,3,4].
Insert 2 with cost min(2, 4) = 2, now nums = [1,1,2,2,2,3,3,3,4].
The total cost is 0 + 0 + 0 + 0 + 1 + 0 + 1 + 0 + 2 = 4.
```

Constraints:

- `1 <= instructions.length <= 105`
- `1 <= instructions[i] <= 105`

题目大意

给你一个整数数组 instructions ，你需要根据 instructions 中的元素创建一个有序数组。一开始你有一个空的数组 nums ，你需要 从左到右 遍历 instructions 中的元素，将它们依次插入 nums 数组中。每一次插入操作的 代价 是以下两者的 较小值：

- nums 中 严格小于 instructions[i] 的数字数目。
- nums 中 严格大于 instructions[i] 的数字数目。

比方说，如果要将 3 插入到 nums = [1,2,3,5] ，那么插入操作的 代价 为 min(2, 1) (元素 1 和 2 小于 3 ，元素 5 大于 3 ），插入后 nums 变成 [1,2,3,3,5] 。请你返回将 instructions 中所有元素依次插入 nums 后的 总最小代价 。由于答案会很大，请将它对 10^9 + 7 取余 后返回。

解题思路

- 给出一个数组，要求将其中的元素从头开始往另外一个空数组中插入，每次插入前，累加代价值 cost = min(**strictly less than**, **strictly greater than**)。最后输出累加值。
- 这一题虽然是 Hard 题，但是读完题以后就可以判定这是模板题了。可以用线段树和树状数组来解决。这里简单说说线段树的思路吧，先将待插入的数组排序，获得总的区间。每次循环做 4 步：2 次 `query` 分别得到 `strictlyLessThan` 和 `strictlyGreaterThan` ，再比较出两者中的最小值累加，最后一步就是 `update` 。
- 由于题目给的数据比较大，所以建立线段树之前记得要先离散化。这一题核心代码不超过 10 行，其他的都是模板代码。具体实现见代码。

代码

```go
package leetcode

import (
  "sort"

  "github.com/halfrost/LeetCode-Go/template"
)

// 解法一 树状数组 Binary Indexed Tree
func createSortedArray(instructions []int) int {
  bit, res := template.BinaryIndexedTree{}, 0
  bit.Init(100001)
  for i, v := range instructions {
    less := bit.Query(v - 1)
    greater := i - bit.Query(v)
    res = (res + min(less, greater)) % (1e9 + 7)
    bit.Add(v, 1)
  }
  return res
```

```go
}

// 解法二 线段树 SegmentTree
func createSortedArray1(instructions []int) int {
  if len(instructions) == 0 {
    return 0
  }
  st, res, mod := template.SegmentCountTree{}, 0, 1000000007
  numsMap, numsArray, tmpArray := discretization1649(instructions)
  // 初始化线段树，节点内的值都赋值为 0，即计数为 0
  st.Init(tmpArray, func(i, j int) int {
    return 0
  })
  for i := 0; i < len(instructions); i++ {
    strictlyLessThan := st.Query(0, numsMap[instructions[i]]-1)
    strictlyGreaterThan := st.Query(numsMap[instructions[i]]+1,
numsArray[len(numsArray)-1])
    res = (res + min(strictlyLessThan, strictlyGreaterThan)) % mod
    st.UpdateCount(numsMap[instructions[i]])
  }
  return res
}

func discretization1649(instructions []int) (map[int]int, []int, []int) {
  tmpArray, numsArray, numsMap := []int{}, []int{}, map[int]int{}
  for i := 0; i < len(instructions); i++ {
    numsMap[instructions[i]] = instructions[i]
  }
  for _, v := range numsMap {
    numsArray = append(numsArray, v)
  }
  sort.Ints(numsArray)
  for i, num := range numsArray {
    numsMap[num] = i
  }
  for i := range numsArray {
    tmpArray = append(tmpArray, i)
  }
  return numsMap, numsArray, tmpArray
}

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

1652. Defuse the Bomb

题目

You have a bomb to defuse, and your time is running out! Your informer will provide you with a **circular** array `code` of length of `n` and a key `k`.

To decrypt the code, you must replace every number. All the numbers are replaced **simultaneously**.

- If `k > 0`, replace the `ith` number with the sum of the **next** `k` numbers.
- If `k < 0`, replace the `ith` number with the sum of the **previous** `k` numbers.
- If `k == 0`, replace the `ith` number with `0`.

As `code` is circular, the next element of `code[n-1]` is `code[0]`, and the previous element of `code[0]` is `code[n-1]`.

Given the **circular** array `code` and an integer key `k`, return *the decrypted code to defuse the bomb*!

Example 1:

```
Input: code = [5,7,1,4], k = 3
Output: [12,10,16,13]
Explanation: Each number is replaced by the sum of the next 3 numbers. The decrypted code is [7+1+4, 1+4+5, 4+5+7, 5+7+1]. Notice that the numbers wrap around.
```

Example 2:

```
Input: code = [1,2,3,4], k = 0
Output: [0,0,0,0]
Explanation: When k is zero, the numbers are replaced by 0.
```

Example 3:

```
Input: code = [2,4,9,3], k = -2
Output: [12,5,6,13]
Explanation: The decrypted code is [3+9, 2+3, 4+2, 9+4]. Notice that the numbers wrap around again. If k is negative, the sum is of the previous numbers.
```

Constraints:

- `n == code.length`
- `1 <= n <= 100`
- `1 <= code[i] <= 100`
- `(n - 1) <= k <= n - 1`

题目大意

你有一个炸弹需要拆除，时间紧迫！你的情报员会给你一个长度为 n 的 循环 数组 code 以及一个密钥 k 。为了获得正确的密码，你需要替换掉每一个数字。所有数字会 同时 被替换。

- 如果 k > 0 ，将第 i 个数字用 接下来 k 个数字之和替换。
- 如果 k < 0 ，将第 i 个数字用 之前 k 个数字之和替换。
- 如果 k == 0 ，将第 i 个数字用 0 替换。

由于 code 是循环的， code[n-1] 下一个元素是 code[0] ，且 code[0] 前一个元素是 code[n-1] 。

给你 循环 数组 code 和整数密钥 k ，请你返回解密后的结果来拆除炸弹！

解题思路

- 给出一个 code 数组，要求按照规则替换每个字母。
- 简单题，按照题意描述循环即可。

代码

```
package leetcode

func decrypt(code []int, k int) []int {
  if k == 0 {
    for i := 0; i < len(code); i++ {
      code[i] = 0
    }
    return code
  }
  count, sum, res := k, 0, make([]int, len(code))
  if k > 0 {
    for i := 0; i < len(code); i++ {
      for j := i + 1; j < len(code); j++ {
        if count == 0 {
          break
        }
        sum += code[j]
        count--
      }
      if count > 0 {
        for j := 0; j < len(code); j++ {
          if count == 0 {
            break
          }
          sum += code[j]
          count--
        }
      }
      res[i] = sum
      sum, count = 0, k
    }
  }
```

```
  if k < 0 {
    for i := 0; i < len(code); i++ {
      for j := i - 1; j >= 0; j-- {
        if count == 0 {
          break
        }
        sum += code[j]
        count++
      }
      if count < 0 {
        for j := len(code) - 1; j >= 0; j-- {
          if count == 0 {
            break
          }
          sum += code[j]
          count++
        }
      }
      res[i] = sum
      sum, count = 0, k
    }
  }
  return res
}
```

1653. Minimum Deletions to Make String Balanced

题目

You are given a string `s` consisting only of characters `'a'` and `'b'`.

You can delete any number of characters in `s` to make `s` **balanced**. `s` is **balanced** if there is no pair of indices `(i,j)` such that `i < j` and `s[i] = 'b'` and `s[j]= 'a'`.

Return *the **minimum** number of deletions needed to make* `s` ***balanced***.

Example 1:

```
Input: s = "aababbab"
Output: 2
Explanation: You can either:
Delete the characters at 0-indexed positions 2 and 6 ("aababbab" -> "aaabbb"), or
Delete the characters at 0-indexed positions 3 and 6 ("aababbab" -> "aabbbb").
```

Example 2:

```
Input: s = "bbaaaaabb"
Output: 2
Explanation: The only solution is to delete the first two characters.
```

Constraints:

- `1 <= s.length <= 105`
- `s[i]` is `'a'` or `'b'`.

题目大意

给你一个字符串 s ，它仅包含字符 'a' 和 'b' 。你可以删除 s 中任意数目的字符，使得 s 平衡 。我们称 s 平衡的 当不存在下标对 (i,j) 满足 i < j 且 s[i] = 'b' 同时 s[j]= 'a' 。请你返回使 s 平衡 的 最少 删除次数。

解题思路

- 给定一个字符串，要求删除最少次数，使得字母 a 都排在字母 b 的前面。
- 很容易想到的一个解题思路是 DP。定义 `dp[i]` 为字符串下标 [0, i] 这个区间内使得字符串平衡的最少删除次数。当 `s[i] == 'a'` 的时候，有 2 种情况，一种是 `s[i]` 前面全是 `[aa……aa]` 的情况，这个时候只需要把其中的所有的字母 `b` 删除即可。还有一种情况是 `s[i]` 前面有字母 `a` 也有字母 `b`，即 `[aaa……abb……b]`，这种情况就需要考虑 `dp[i-1]` 了。当前字母是 `a`，那么肯定要删除字母 `a`，来维持前面有一段字母 `b` 的情况。当 `s[i] == 'b'` 的时候，不管是 `[aa……aa]` 这种情况，还是 `[aaa……abb……b]` 这种情况，当前字母 `b` 都可以直接附加在后面，也能保证整个字符串是平衡的。所以状态转移方程为 `dp[i+1] = min(dp[i] + 1, bCount), s[i] == 'a'`，`dp[i+1] = dp[i], s[i] == 'b'`。最终答案存在 `dp[n]` 中。由于前后项的递推关系中只用到一次前一项，所以我们还可以优化一下空间，用一个变量保存前一项的结果。优化以后的代码见解法一。
- 这一题还有一个模拟的思路。题目要求找到最小删除字数，那么就是要找到一个“临界点”，在这个临界点的左边删除所有的字母 b，在这个临界点的右边删除所有的字母 a。在所有的“临界点”中找到删除最少的次数。代码实现见解法二。

代码

```go
package leetcode

// 解法一 DP
func minimumDeletions(s string) int {
  prev, res, bCount := 0, 0, 0
  for _, c := range s {
    if c == 'a' {
      res = min(prev+1, bCount)
      prev = res
    } else {
      bCount++
    }
  }
  return res
}
```

```go
func min(a, b int) int {
  if a < b {
    return a
  }
  return b
}

// 解法二 模拟
func minimumDeletions1(s string) int {
  aCount, bCount, res := 0, 0, 0
  for i := 0; i < len(s); i++ {
    if s[i] == 'a' {
      aCount++
    }
  }
  res = aCount
  for i := 0; i < len(s); i++ {
    if s[i] == 'a' {
      aCount--
    } else {
      bCount++
    }
    res = min(res, aCount+bCount)
  }
  return res
}
```

1654. Minimum Jumps to Reach Home

题目

A certain bug's home is on the x-axis at position `x`. Help them get there from position `0`.

The bug jumps according to the following rules:

- It can jump exactly `a` positions **forward** (to the right).
- It can jump exactly `b` positions **backward** (to the left).
- It cannot jump backward twice in a row.
- It cannot jump to any `forbidden` positions.

The bug may jump forward **beyond** its home, but it **cannot jump** to positions numbered with **negative** integers.

Given an array of integers `forbidden`, where `forbidden[i]` means that the bug cannot jump to the position `forbidden[i]`, and integers `a`, `b`, and `x`, return *the minimum number of jumps needed for the bug to reach its home*. If there is no possible sequence of jumps that lands the bug on position `x`, return `1.`

Example 1:

```
Input: forbidden = [14,4,18,1,15], a = 3, b = 15, x = 9
Output: 3
Explanation: 3 jumps forward (0 -> 3 -> 6 -> 9) will get the bug home.
```

Example 2:

```
Input: forbidden = [8,3,16,6,12,20], a = 15, b = 13, x = 11
Output: -1
```

Example 3:

```
Input: forbidden = [1,6,2,14,5,17,4], a = 16, b = 9, x = 7
Output: 2
Explanation: One jump forward (0 -> 16) then one jump backward (16 -> 7) will get the
bug home.
```

Constraints:

- `1 <= forbidden.length <= 1000`
- `1 <= a, b, forbidden[i] <= 2000`
- `0 <= x <= 2000`
- All the elements in `forbidden` are distinct.
- Position `x` is not forbidden.

题目大意

有一只跳蚤的家在数轴上的位置 x 处。请你帮助它从位置 0 出发，到达它的家。

跳蚤跳跃的规则如下：

- 它可以 往前 跳恰好 a 个位置（即往右跳）。
- 它可以 往后 跳恰好 b 个位置（即往左跳）。
- 它不能 连续 往后跳 2 次。
- 它不能跳到任何 forbidden 数组中的位置。

跳蚤可以往前跳 超过 它的家的位置，但是它 不能跳到负整数 的位置。给你一个整数数组 forbidden ，其中 forbidden[i] 是跳蚤不能跳到的位置，同时给你整数 a， b 和 x ，请你返回跳蚤到家的最少跳跃次数。如果没有恰好到达 x 的可行方案，请你返回 -1 。

解题思路

- 给出坐标 x ，可以往前跳的步长 a，往后跳的步长 b。要求输出能跳回家的最少跳跃次数。
- 求最少跳跃次数，思路用 BFS 求解，最先到达坐标 x 的方案即是最少跳跃次数。对 `forbidden` 的处理是把记忆化数组里面把他们标记为 true。禁止连续往后跳 2 次的限制，要求我们在 BFS 入队的时候再记录一下跳跃方向，每次往后跳的时候判断前一跳是否是往后跳，如果是往后跳，此次就不能往后跳了。

代码

```go
package leetcode

func minimumJumps(forbidden []int, a int, b int, x int) int {
  visited := make([]bool, 6000)
  for i := range forbidden {
    visited[forbidden[i]] = true
  }
  queue, res := [][2]int{{0, 0}}, -1
  for len(queue) > 0 {
    length := len(queue)
    res++
    for i := 0; i < length; i++ {
      cur, isBack := queue[i][0], queue[i][1]
      if cur == x {
        return res
      }
      if isBack == 0 && cur-b > 0 && !visited[cur-b] {
        visited[cur-b] = true
        queue = append(queue, [2]int{cur - b, 1})
      }
      if cur+a < len(visited) && !visited[cur+a] {
        visited[cur+a] = true
        queue = append(queue, [2]int{cur + a, 0})
      }
    }
    queue = queue[length:]
  }
  return -1
}
```

1655. Distribute Repeating Integers

题目

You are given an array of `n` integers, `nums`, where there are at most `50` unique values in the array. You are also given an array of `m` customer order quantities, `quantity`, where `quantity[i]` is the amount of integers the `ith` customer ordered. Determine if it is possible to distribute `nums` such that:

- The `ith` customer gets **exactly** `quantity[i]` integers,
- The integers the `ith` customer gets are **all equal**, and
- Every customer is satisfied.

Return `true` *if it is possible to distribute* `nums` *according to the above conditions*.

Example 1:

```
Input: nums = [1,2,3,4], quantity = [2]
Output: false
Explanation: The 0th customer cannot be given two different integers.
```

Example 2:

```
Input: nums = [1,2,3,3], quantity = [2]
Output: true
Explanation: The 0th customer is given [3,3]. The integers [1,2] are not used.
```

Example 3:

```
Input: nums = [1,1,2,2], quantity = [2,2]
Output: true
Explanation: The 0th customer is given [1,1], and the 1st customer is given [2,2].
```

Example 4:

```
Input: nums = [1,1,2,3], quantity = [2,2]
Output: false
Explanation: Although the 0th customer could be given [1,1], the 1st customer cannot be
satisfied.
```

Example 5:

```
Input: nums = [1,1,1,1,1], quantity = [2,3]
Output: true
Explanation: The 0th customer is given [1,1], and the 1st customer is given [1,1,1].
```

Constraints:

- `n == nums.length`
- `1 <= n <= 105`
- `1 <= nums[i] <= 1000`
- `m == quantity.length`
- `1 <= m <= 10`
- `1 <= quantity[i] <= 105`
- There are at most `50` unique values in `nums`.

题目大意

给你一个长度为 n 的整数数组 nums ，这个数组中至多有 50 个不同的值。同时你有 m 个顾客的订单 quantity ，其中，整数 quantity[i] 是第 i 位顾客订单的数目。请你判断是否能将 nums 中的整数分配给这些顾客，且满足：

- 第 i 位顾客 恰好 有 quantity[i] 个整数。
- 第 i 位顾客拿到的整数都是 相同的 。

- 每位顾客都满足上述两个要求。

如果你可以分配 nums 中的整数满足上面的要求，那么请返回 true ，否则返回 false 。

解题思路

- 给定一个数组 nums，订单数组 quantity，要求按照订单满足顾客的需求。如果能满足输出 true，不能满足输出 false。
- 用 DFS 记忆化暴力搜索。代码实现不难。（不知道此题为什么是 Hard）

代码

```go
package leetcode

func canDistribute(nums []int, quantity []int) bool {
  freq := make(map[int]int)
  for _, n := range nums {
    freq[n]++
  }
  return dfs(freq, quantity)
}

func dfs(freq map[int]int, quantity []int) bool {
  if len(quantity) == 0 {
    return true
  }
  visited := make(map[int]bool)
  for i := range freq {
    if visited[freq[i]] {
      continue
    }
    visited[freq[i]] = true
    if freq[i] >= quantity[0] {
      freq[i] -= quantity[0]
      if dfs(freq, quantity[1:]) {
        return true
      }
      freq[i] += quantity[0]
    }
  }
  return false
}
```

1656. Design an Ordered Stream

题目

There is a stream of `n` `(id, value)` pairs arriving in an **arbitrary** order, where `id` is an integer between `1` and `n` and `value` is a string. No two pairs have the same `id`.

Design a stream that returns the values in **increasing order of their IDs** by returning a **chunk** (list) of values after each insertion. The concatenation of all the **chunks** should result in a list of the sorted values.

Implement the `OrderedStream` class:

- `OrderedStream(int n)` Constructs the stream to take `n` values.
- `String[] insert(int id, String value)` Inserts the pair `(id, value)` into the stream, then returns the **largest possible chunk** of currently inserted values that appear next in the order.

Example:

```
Input
["OrderedStream", "insert", "insert", "insert", "insert", "insert"]
[[5], [3, "ccccc"], [1, "aaaaa"], [2, "bbbbb"], [5, "eeeee"], [4, "ddddd"]]
Output
[null, [], ["aaaaa"], ["bbbbb", "ccccc"], [], ["ddddd", "eeeee"]]

Explanation
// Note that the values ordered by ID is ["aaaaa", "bbbbb", "ccccc", "ddddd", "eeeee"].
OrderedStream os = new OrderedStream(5);
os.insert(3, "ccccc"); // Inserts (3, "ccccc"), returns [].
os.insert(1, "aaaaa"); // Inserts (1, "aaaaa"), returns ["aaaaa"].
os.insert(2, "bbbbb"); // Inserts (2, "bbbbb"), returns ["bbbbb", "ccccc"].
os.insert(5, "eeeee"); // Inserts (5, "eeeee"), returns [].
os.insert(4, "ddddd"); // Inserts (4, "ddddd"), returns ["ddddd", "eeeee"].
// Concatentating all the chunks returned:
// [] + ["aaaaa"] + ["bbbbb", "ccccc"] + [] + ["ddddd", "eeeee"] = ["aaaaa", "bbbbb",
"ccccc", "ddddd", "eeeee"]
// The resulting order is the same as the order above.
```

Constraints:

- `1 <= n <= 1000`
- `1 <= id <= n`
- `value.length == 5`
- `value` consists only of lowercase letters.
- Each call to `insert` will have a unique `id.`
- Exactly `n` calls will be made to `insert`.

题目大意

有 n 个 (id, value) 对，其中 id 是 1 到 n 之间的一个整数，value 是一个字符串。不存在 id 相同的两个 (id, value) 对。

设计一个流，以 任意 顺序获取 n 个 (id, value) 对，并在多次调用时 按 id 递增的顺序 返回一些值。

实现 OrderedStream 类：

- OrderedStream(int n) 构造一个能接收 n 个值的流，并将当前指针 ptr 设为 1 。
- String[] insert(int id, String value) 向流中存储新的 (id, value) 对。存储后：
  如果流存储有 id = ptr 的 (id, value) 对，则找出从 id = ptr 开始的 最长 id 连续递增序列 ，并 按顺序 返回与这些 id 关联的值的列表。然后，将 ptr 更新为最后那个 id + 1 。
  否则，返回一个空列表。

解题思路

- 设计一个具有插入操作的 Ordered Stream。insert 操作先在指定位置插入 value，然后返回当前指针 ptr 到最近一个空位置的最长连续递增字符串。如果字符串不为空，ptr 移动到非空 value 的后一个下标位置处。
- 简单题。按照题目描述模拟即可。注意控制好 ptr 的位置。

代码

```go
package leetcode

type OrderedStream struct {
  ptr    int
  stream []string
}

func Constructor(n int) OrderedStream {
  ptr, stream := 1, make([]string, n+1)
  return OrderedStream{ptr: ptr, stream: stream}
}

func (this *OrderedStream) Insert(id int, value string) []string {
  this.stream[id] = value
  res := []string{}
  if this.ptr == id || this.stream[this.ptr] != "" {
    res = append(res, this.stream[this.ptr])
    for i := id + 1; i < len(this.stream); i++ {
      if this.stream[i] != "" {
        res = append(res, this.stream[i])
      } else {
        this.ptr = i
        return res
      }
    }
  }
  if len(res) > 0 {
    return res
  }
  return []string{}
}

/**
```

```
 * Your OrderedStream object will be instantiated and called as such:
 * obj := Constructor(n);
 * param_1 := obj.Insert(id,value);
 */
```

1657. Determine if Two Strings Are Close

题目

Two strings are considered **close** if you can attain one from the other using the following operations:

- Operation 1: Swap any two **existing** characters.
  - For example, `abcde -> aecdb`
- Operation 2: Transform **every** occurrence of one **existing** character into another **existing** character, and do the same with the other character.
  - For example, `aacabb -> bbcbaa` (all `a`'s turn into `b`'s, and all `b`'s turn into `a`'s)

You can use the operations on either string as many times as necessary.

Given two strings, `word1` and `word2`, return `true` *if* `word1` *and* `word2` *are* ***close****, and* `false` *otherwise.*

Example 1:

```
Input: word1 = "abc", word2 = "bca"
Output: true
Explanation: You can attain word2 from word1 in 2 operations.
Apply Operation 1: "abc" -> "acb"
Apply Operation 1: "acb" -> "bca"
```

Example 2:

```
Input: word1 = "a", word2 = "aa"
Output: false
Explanation: It is impossible to attain word2 from word1, or vice versa, in any number
of operations.
```

Example 3:

```
Input: word1 = "cabbba", word2 = "abbccc"
Output: true
Explanation: You can attain word2 from word1 in 3 operations.
Apply Operation 1: "cabbba" -> "caabbb"
Apply Operation 2: "caabbb" -> "baaccc"
Apply Operation 2: "baaccc" -> "abbccc"
```

Example 4:

```
Input: word1 = "cabbba", word2 = "aabbss"
Output: false
Explanation: It is impossible to attain word2 from word1, or vice versa, in any amount
of operations.
```

Constraints:

- `1 <= word1.length, word2.length <= 105`
- `word1` and `word2` contain only lowercase English letters.

题目大意

如果可以使用以下操作从一个字符串得到另一个字符串，则认为两个字符串 接近 ：

- 操作 1：交换任意两个 现有 字符。例如，abcde -> aecdb
- 操作 2：将一个 现有 字符的每次出现转换为另一个 现有 字符，并对另一个字符执行相同的操作。例如，aacabb -> bbcbaa（所有 a 转化为 b ，而所有的 b 转换为 a ）

你可以根据需要对任意一个字符串多次使用这两种操作。给你两个字符串，word1 和 word2 。如果 word1 和 word2 接近 ，就返回 true ；否则，返回 false 。

解题思路

- 判断 2 个字符串是否“接近”。“接近”的定义是能否通过交换 2 个字符或者 2 个字母互换，从一个字符串变换成另外一个字符串，如果存在这样的变换，即是“接近”。
- 先统计 2 个字符串的 26 个字母的频次，如果频次有不相同的，直接返回 false。在频次相同的情况下，再从小到大排序，再次扫描判断频次是否相同。
- 注意几种特殊情况：频次相同，再判断字母交换是否合法存在，如果字母不存在，输出 false。例如测试文件中的 case 5 。出现频次个数相同，但是频次不同。例如测试文件中的 case 6 。

代码

```go
package leetcode

import (
  "sort"
)

func closeStrings(word1 string, word2 string) bool {
  if len(word1) != len(word2) {
    return false
  }
  freqCount1, freqCount2 := make([]int, 26), make([]int, 26)
  for _, c := range word1 {
    freqCount1[c-97]++
  }
```

```
  for _, c := range word2 {
    freqCount2[c-97]++
  }
  for i := 0; i < 26; i++ {
    if (freqCount1[i] == freqCount2[i]) ||
      (freqCount1[i] > 0 && freqCount2[i] > 0) {
      continue
    }
    return false
  }
  sort.Ints(freqCount1)
  sort.Ints(freqCount2)
  for i := 0; i < 26; i++ {
    if freqCount1[i] != freqCount2[i] {
      return false
    }
  }
  return true
}
```

1658. Minimum Operations to Reduce X to Zero

题目

You are given an integer array `nums` and an integer `x`. In one operation, you can either remove the leftmost or the rightmost element from the array `nums` and subtract its value from `x`. Note that this **modifies** the array for future operations.

Return *the **minimum number** of operations to reduce* `x` *to **exactly*** `0` *if it's possible, otherwise, return* `1`.

Example 1:

```
Input: nums = [1,1,4,2,3], x = 5
Output: 2
Explanation: The optimal solution is to remove the last two elements to reduce x to
zero.
```

Example 2:

```
Input: nums = [5,6,7,8,9], x = 4
Output: -1
```

Example 3:

```
Input: nums = [3,2,20,1,1,3], x = 10
Output: 5
Explanation: The optimal solution is to remove the last three elements and the first two elements (5 operations in total) to reduce x to zero.
```

Constraints:

- `1 <= nums.length <= 105`
- `1 <= nums[i] <= 104`
- `1 <= x <= 109`

题目大意

给你一个整数数组 nums 和一个整数 x 。每一次操作时，你应当移除数组 nums 最左边或最右边的元素，然后从 x 中减去该元素的值。请注意，需要 修改 数组以供接下来的操作使用。如果可以将 x 恰好 减到 0 ，返回 最小操作数 ；否则，返回 -1 。

解题思路

- 给定一个数组 nums 和一个整数 x，要求从数组两端分别移除一些数，使得这些数加起来正好等于整数 x，要求输出最小操作数。
- 要求输出最小操作数，即数组两头的数字个数最少，并且加起来和正好等于整数 x。由于在数组的两头，用 2 个指针分别操作不太方便。我当时解题的时候的思路是把它变成循环数组，这样两边的指针就在一个区间内了。利用滑动窗口找到一个最小的窗口，使得窗口内的累加和等于整数 k。这个方法可行，但是代码挺多的。
- 有没有更优美的方法呢？有的。要想两头的长度最少，也就是中间这段的长度最大。这样就转换成直接在数组上使用滑动窗口求解，累加和等于一个固定值的连续最长的子数组。
- 和这道题类似思路的题目，209，1040(循环数组)，325。强烈推荐这 3 题。

代码

```go
package leetcode

func minOperations(nums []int, x int) int {
  total := 0
  for _, n := range nums {
    total += n
  }
  target := total - x
  if target < 0 {
    return -1
  }
  if target == 0 {
    return len(nums)
  }
  left, right, sum, res := 0, 0, 0, -1
  for right < len(nums) {
    if sum < target {
```

```
      sum += nums[right]
      right++
    }
    for sum >= target {
      if sum == target {
        res = max(res, right-left)
      }
      sum -= nums[left]
      left++
    }
  }
  if res == -1 {
    return -1
  }
  return len(nums) - res
}

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

1659. Maximize Grid Happiness

题目

You are given four integers, `m`, `n`, `introvertsCount`, and `extrovertsCount`. You have an `m x n` grid, and there are two types of people: introverts and extroverts. There are `introvertsCount` introverts and `extrovertsCount` extroverts.

You should decide how many people you want to live in the grid and assign each of them one grid cell. Note that you **do not** have to have all the people living in the grid.

The **happiness** of each person is calculated as follows:

- Introverts **start** with `120` happiness and **lose** `30` happiness for each neighbor (introvert or extrovert).
- Extroverts **start** with `40` happiness and **gain** `20` happiness for each neighbor (introvert or extrovert).

Neighbors live in the directly adjacent cells north, east, south, and west of a person's cell.

The **grid happiness** is the **sum** of each person's happiness. Return *the **maximum possible grid happiness**.*

Example 1:

```
Input: m = 2, n = 3, introvertsCount = 1, extrovertsCount = 2
Output: 240
Explanation: Assume the grid is 1-indexed with coordinates (row, column).
We can put the introvert in cell (1,1) and put the extroverts in cells (1,3) and (2,3).
- Introvert at (1,1) happiness: 120 (starting happiness) - (0 * 30) (0 neighbors) = 120
- Extrovert at (1,3) happiness: 40 (starting happiness) + (1 * 20) (1 neighbor) = 60
- Extrovert at (2,3) happiness: 40 (starting happiness) + (1 * 20) (1 neighbor) = 60
The grid happiness is 120 + 60 + 60 = 240.
The above figure shows the grid in this example with each person's happiness. The
introvert stays in the light green cell while the extroverts live on the light purple
cells.
```

Example 2:

```
Input: m = 3, n = 1, introvertsCount = 2, extrovertsCount = 1
Output: 260
Explanation: Place the two introverts in (1,1) and (3,1) and the extrovert at (2,1).
- Introvert at (1,1) happiness: 120 (starting happiness) - (1 * 30) (1 neighbor) = 90
- Extrovert at (2,1) happiness: 40 (starting happiness) + (2 * 20) (2 neighbors) = 80
- Introvert at (3,1) happiness: 120 (starting happiness) - (1 * 30) (1 neighbor) = 90
The grid happiness is 90 + 80 + 90 = 260.
```

Example 3:

```
Input: m = 2, n = 2, introvertsCount = 4, extrovertsCount = 0
Output: 240
```

Constraints:

- `1 <= m, n <= 5`
- `0 <= introvertsCount, extrovertsCount <= min(m * n, 6)`

题目大意

给你四个整数 m、n、introvertsCount 和 extrovertsCount 。有一个 m x n 网格，和两种类型的人：内向的人和外向的人。总共有 introvertsCount 个内向的人和 extrovertsCount 个外向的人。请你决定网格中应当居住多少人，并为每个人分配一个网格单元。 注意，不必 让所有人都生活在网格中。每个人的 幸福感 计算如下：

- 内向的人 开始 时有 120 个幸福感，但每存在一个邻居（内向的或外向的）他都会 失去 30 个幸福感。
- 外向的人 开始 时有 40 个幸福感，每存在一个邻居（内向的或外向的）他都会 得到 20 个幸福感。

邻居是指居住在一个人所在单元的上、下、左、右四个直接相邻的单元中的其他人。网格幸福感 是每个人幸福感的总和 。 返回 最大可能的网格幸福感 。

解题思路

- 给出 `m` x `n` 网格和两种人，要求如何安排这两种人能使得网格的得分最大。两种人有各自的初始分，相邻可能会加分也有可能减分。

- 这一题状态很多。首先每个格子有 3 种状态，那么每一行有 3^6 = 729 种不同的状态。每行行内分数变化值可能是 -60(两个内向)，+40(两个外向)，-10(一个内向一个外向)。两行行间分数变化值可能是 -60(两个内向)，+40(两个外向)，-10(一个内向一个外向)。那么我们可以把每行的状态压缩成一个三进制，那么网格就变成了一维，每两个三进制之间的关系是行间关系，每个三进制内部还需要根据内向和外向的人数决定行内最终分数。定义 `dp[lineStatusLast][row][introvertsCount][extrovertsCount]` 代表在上一行 `row - 1` 的状态是 `lineStatusLast` 的情况下，当前枚举到了第 `row` 行，内向还有 `introvertsCount` 个人，外向还有 `extrovertsCount` 个人能获得的最大分数。状态转移方程是 `dp[lineStatusLast(row-1)][row][introvertsCount][extrovertsCount] = max{dp[lineStatusLast(row)][row+1][introvertsCount - countIC(lineStatusLast(row)) ][extrovertsCount - countEC(lineStatusLast(row)) ] + scoreInner(lineStatusLast(row)) + scoreOuter(lineStatusLast(row-1),lineStatusLast(row))}` ，这里有 2 个统计函数，`countIC` 是统计当前行状态三进制里面有多少个内向人。`countEC` 是统计当前行状态三进制里面有多少个外向人。`scoreInner` 是计算当前行状态三进制的行内分数。`scoreOuter` 是计算 `row -1` 行和 `row` 行之间的行间分数。
- 由于这个状态转移方程的计算量是巨大的。所以需要预先初始化一些计算结果。比如把 729 中行状态分别对应的行内、行间的分数都计算好，在动态规划状态转移的时候，直接查表获取分数即可。这样我们在深搜的时候，利用 dp 的记忆化，可以大幅减少时间复杂度。
- 题目中还提到，人数可以不用完。如果 `introvertsCount = 0`, `extrovertsCount = 0` ，即人数都用完了的情况，这时候 `dp = 0` 。如果 `row = m`，即已经枚举完了所有行，那么不管剩下多少人，这一行的 `dp = 0` 。
- 初始化的时候，注意，特殊处理 0 的情况，0 行 0 列都初始化为 -1 。

代码

```go
package leetcode

import (
  "math"
)

func getMaxGridHappiness(m int, n int, introvertsCount int, extrovertsCount int) int {
  // lineStatus            将每一行中 3 种状态进行编码，空白 - 0，内向人 - 1，外向人 - 2，每行状态用三进制表示
  // lineStatusList[729][6]      每一行的三进制表示
  // introvertsCountInner[729]  每一个 lineStatus 包含的内向人数
  // extrovertsCountInner[729]  每一个 lineStatus 包含的外向人数
  // scoreInner[729]         每一个 lineStatus 包含的行内得分（只统计 lineStatus 本身的得分，不包括它与上一行的）
  // scoreOuter[729][729]      每一个 lineStatus 包含的行外得分
  // dp[上一行的 lineStatus][当前处理到的行][剩余的内向人数][剩余的外向人数]
  n3, lineStatus, introvertsCountInner, extrovertsCountInner, scoreInner, scoreOuter, lineStatusList, dp := math.Pow(3.0, float64(n)), 0, [729]int{}, [729]int{}, [729]int{}, [729][729]int{}, [729][6]int{}, [729][6][7][7]int{}
  for i := 0; i < 729; i++ {
    lineStatusList[i] = [6]int{}
  }
```

```
  for i := 0; i < 729; i++ {
    dp[i] = [6][7][7]int{}
    for j := 0; j < 6; j++ {
      dp[i][j] = [7][7]int{}
      for k := 0; k < 7; k++ {
        dp[i][j][k] = [7]int{-1, -1, -1, -1, -1, -1, -1}
      }
    }
  }
  // 预处理
  for lineStatus = 0; lineStatus < int(n3); lineStatus++ {
    tmp := lineStatus
    for i := 0; i < n; i++ {
      lineStatusList[lineStatus][i] = tmp % 3
      tmp /= 3
    }
    introvertsCountInner[lineStatus], extrovertsCountInner[lineStatus],
scoreInner[lineStatus] = 0, 0, 0
    for i := 0; i < n; i++ {
      if lineStatusList[lineStatus][i] != 0 {
        // 个人分数
        if lineStatusList[lineStatus][i] == 1 {
          introvertsCountInner[lineStatus]++
          scoreInner[lineStatus] += 120
        } else if lineStatusList[lineStatus][i] == 2 {
          extrovertsCountInner[lineStatus]++
          scoreInner[lineStatus] += 40
        }
        // 行内分数
        if i-1 >= 0 {
          scoreInner[lineStatus] += closeScore(lineStatusList[lineStatus][i],
lineStatusList[lineStatus][i-1])
        }
      }
    }
  }
  // 行外分数
  for lineStatus0 := 0; lineStatus0 < int(n3); lineStatus0++ {
    for lineStatus1 := 0; lineStatus1 < int(n3); lineStatus1++ {
      scoreOuter[lineStatus0][lineStatus1] = 0
      for i := 0; i < n; i++ {
        scoreOuter[lineStatus0][lineStatus1] += closeScore(lineStatusList[lineStatus0]
[i], lineStatusList[lineStatus1][i])
      }
    }
  }
  return dfs(0, 0, introvertsCount, extrovertsCount, m, int(n3), &dp,
&introvertsCountInner, &extrovertsCountInner, &scoreInner, &scoreOuter)
}
```

```go
// 如果 x 和 y 相邻，需要加上的分数
func closeScore(x, y int) int {
  if x == 0 || y == 0 {
    return 0
  }
  // 两个内向的人，每个人要 -30，一共 -60
  if x == 1 && y == 1 {
    return -60
  }
  if x == 2 && y == 2 {
    return 40
  }
  return -10
}

// dfs(上一行的 lineStatus，当前处理到的行，剩余的内向人数，剩余的外向人数)
func dfs(lineStatusLast, row, introvertsCount, extrovertsCount, m, n3 int, dp *[729][6]
[7][7]int, introvertsCountInner, extrovertsCountInner, scoreInner *[729]int, scoreOuter
*[729][729]int) int {
  // 边界条件：如果已经处理完，或者没有人了
  if row == m || introvertsCount+extrovertsCount == 0 {
    return 0
  }
  // 记忆化
  if dp[lineStatusLast][row][introvertsCount][extrovertsCount] != -1 {
    return dp[lineStatusLast][row][introvertsCount][extrovertsCount]
  }
  best := 0
  for lineStatus := 0; lineStatus < n3; lineStatus++ {
    if introvertsCountInner[lineStatus] > introvertsCount ||
extrovertsCountInner[lineStatus] > extrovertsCount {
      continue
    }
    score := scoreInner[lineStatus] + scoreOuter[lineStatus][lineStatusLast]
    best = max(best, score+dfs(lineStatus, row+1, introvertsCount-
introvertsCountInner[lineStatus], extrovertsCount-extrovertsCountInner[lineStatus], m,
n3, dp, introvertsCountInner, extrovertsCountInner, scoreInner, scoreOuter))
  }
  dp[lineStatusLast][row][introvertsCount][extrovertsCount] = best
  return best
}

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

1662. Check If Two String Arrays are Equivalent

题目

Given two string arrays `word1` and `word2`, return `true` ***if the two arrays** represent* *the same string, and* `false` *otherwise.*

A string is **represented** by an array if the array elements concatenated **in order** forms the string.

Example 1:

```
Input: word1 = ["ab", "c"], word2 = ["a", "bc"]
Output: true
Explanation:
word1 represents string "ab" + "c" -> "abc"
word2 represents string "a" + "bc" -> "abc"
The strings are the same, so return true.
```

Example 2:

```
Input: word1 = ["a", "cb"], word2 = ["ab", "c"]
Output: false
```

Example 3:

```
Input: word1  = ["abc", "d", "defg"], word2 = ["abcddefg"]
Output: true
```

Constraints:

- `1 <= word1.length, word2.length <= 103`
- `1 <= word1[i].length, word2[i].length <= 103`
- `1 <= sum(word1[i].length), sum(word2[i].length) <= 103`
- `word1[i]` and `word2[i]` consist of lowercase letters.

题目大意

给你两个字符串数组 word1 和 word2 。如果两个数组表示的字符串相同，返回 true ；否则，返回 false 。数组表示的字符串 是由数组中的所有元素 按顺序 连接形成的字符串。

解题思路

- 简单题，依次拼接 2 个数组内的字符串，然后比较 str1 和 str2 是否相同即可。

代码

```go
package leetcode

func arrayStringsAreEqual(word1 []string, word2 []string) bool {
  str1, str2 := "", ""
  for i := 0; i < len(word1); i++ {
    str1 += word1[i]
  }
  for i := 0; i < len(word2); i++ {
    str2 += word2[i]
  }
  return str1 == str2
}
```

1663. Smallest String With A Given Numeric Value

题目

The **numeric value** of a **lowercase character** is defined as its position `(1-indexed)` in the alphabet, so the numeric value of `a` is `1`, the numeric value of `b` is `2`, the numeric value of `c` is `3`, and so on.

The **numeric value** of a **string** consisting of lowercase characters is defined as the sum of its characters' numeric values. For example, the numeric value of the string `"abe"` is equal to `1 + 2 + 5 = 8`.

You are given two integers `n` and `k`. Return *the **lexicographically smallest string** with **length** equal to `n` and **numeric value** equal to `k`.*

Note that a string `x` is lexicographically smaller than string `y` if `x` comes before `y` in dictionary order, that is, either `x` is a prefix of `y`, or if `i` is the first position such that `x[i] != y[i]`, then `x[i]` comes before `y[i]` in alphabetic order.

Example 1:

```
Input: n = 3, k = 27
Output: "aay"
Explanation: The numeric value of the string is 1 + 1 + 25 = 27, and it is the smallest string with such a value and length equal to 3.
```

Example 2:

```
Input: n = 5, k = 73
Output: "aaszz"
```

Constraints:

- `1 <= n <= 105`
- `n <= k <= 26 * n`

题目大意

小写字符 的 数值 是它在字母表中的位置（从 1 开始），因此 a 的数值为 1 ，b 的数值为 2 ，c 的数值为 3 ，以此类推。字符串由若干小写字符组成，字符串的数值 为各字符的数值之和。例如，字符串 "abe" 的数值等于 1 + 2 + 5 = 8 。给你两个整数 n 和 k 。返回 长度 等于 n 且 数值 等于 k 的 字典序最小 的字符串。注意，如果字符串 x 在字典排序中位于 y 之前，就认为 x 字典序比 y 小，有以下两种情况：

- x 是 y 的一个前缀；
- 如果 i 是 x[i] != y[i] 的第一个位置，且 x[i] 在字母表中的位置比 y[i] 靠前。

解题思路

- 给出 n 和 k，要求找到字符串长度为 n，字母在字母表内位置总和为 k 的最小字典序字符串。
- 这一题笔者读完题，比赛的时候直接用 DFS 撸了一版。赛后看了时间复杂度马马虎虎，感觉还有优化的空间。DFS 会遍历出所有的解，实际上这一题只要求最小字典序，所以 DFS 剪枝的时候要加上判断字典序的判断，如果新添加进来的字母比已经保存的字符串的相应位置上的字母字典序大，那么就直接 return，这个答案一定不会是最小字典序。代码见解法二
- 想到这里，其实 DFS 不必要，直接用 for 循环就可找到最小字典序的字符串。代码见解法一。

代码

```go
package leetcode

// 解法一 贪心
func getSmallestString(n int, k int) string {
    str, i, j := make([]byte, n), 0, 0
    for i = n-1; i <= k-26; i, k = i-1, k-26 {
        str[i] = 'z'
    }
    if i >= 0 {
        str[i] = byte('a' + k-1-i)
        for ; j < i; j++ {
            str[j] = 'a'
        }
    }
    return string(str)
}

// 解法二 DFS
func getSmallestString1(n int, k int) string {
  if n == 0 {
    return ""
  }
  res, c := "", []byte{}
  findSmallestString(0, n, k, 0, c, &res)
  return res
}
```

```go
func findSmallestString(value int, length, k, index int, str []byte, res *string) {
  if len(str) == length && value == k {
    tmp := string(str)
    if (*res) == "" {
      *res = tmp
    }
    if tmp < *res && *res != "" {
      *res = tmp
    }
    return
  }
  if len(str) >= index && (*res) != "" && str[index-1] > (*res)[index-1] {
    return
  }
  for j := 0; j < 26; j++ {
    if k-value > (length-len(str))*26 || value > k {
      return
    }
    str = append(str, byte(int('a')+j))
    value += j + 1
    findSmallestString(value, length, k, index+1, str, res)
    str = str[:len(str)-1]
    value -= j + 1

  }
}
```

1664. Ways to Make a Fair Array

题目

You are given an integer array `nums`. You can choose **exactly one** index (**0-indexed**) and remove the element. Notice that the index of the elements may change after the removal.

For example, if `nums = [6,1,7,4,1]`:

- Choosing to remove index `1` results in `nums = [6,7,4,1]`.
- Choosing to remove index `2` results in `nums = [6,1,4,1]`.
- Choosing to remove index `4` results in `nums = [6,1,7,4]`.

An array is **fair** if the sum of the odd-indexed values equals the sum of the even-indexed values.

Return the ***number** of indices that you could choose such that after the removal,* `nums` *is **fair**.*

Example 1:

```
Input: nums = [2,1,6,4]
Output: 1
Explanation:
Remove index 0: [1,6,4] -> Even sum: 1 + 4 = 5. Odd sum: 6. Not fair.
Remove index 1: [2,6,4] -> Even sum: 2 + 4 = 6. Odd sum: 6. Fair.
Remove index 2: [2,1,4] -> Even sum: 2 + 4 = 6. Odd sum: 1. Not fair.
Remove index 3: [2,1,6] -> Even sum: 2 + 6 = 8. Odd sum: 1. Not fair.
There is 1 index that you can remove to make nums fair.
```

Example 2:

```
Input: nums = [1,1,1]
Output: 3
Explanation: You can remove any index and the remaining array is fair.
```

Example 3:

```
Input: nums = [1,2,3]
Output: 0
Explanation: You cannot make a fair array after removing any index.
```

Constraints:

- `1 <= nums.length <= 105`
- `1 <= nums[i] <= 104`

题目大意

给你一个整数数组 nums 。你需要选择 恰好 一个下标（下标从 0 开始）并删除对应的元素。请注意剩下元素的下标可能会因为删除操作而发生改变。

比方说，如果 nums = [6,1,7,4,1] ，那么：

- 选择删除下标 1 ，剩下的数组为 nums = [6,7,4,1] 。
- 选择删除下标 2 ，剩下的数组为 nums = [6,1,4,1] 。
- 选择删除下标 4 ，剩下的数组为 nums = [6,1,7,4] 。

如果一个数组满足奇数下标元素的和与偶数下标元素的和相等，该数组就是一个 平衡数组 。请你返回删除操作后，剩下的数组 nums 是 平衡数组 的 方案数 。

解题思路

- 给定一个数组 nums，要求输出仅删除一个元素以后能使得整个数组平衡的方案数。平衡的定义是奇数下标元素总和等于偶数下标元素总和。
- 这一题如果暴力解答，会超时。原因是每次删除元素以后，都重新计算奇偶数位总和比较耗时。应该利用前面计算过的累加和，推导出此次删除元素以后的情况。这样修改以后就不超时了。具体的，如果删除的是元素是奇数位，这个下标的前缀和不变，要变化的是后面的。删除元素后面，原来偶数位的总和变成了奇数位了，原来奇数位的总和变成偶数位了。删除元素后面这半段的总和可以用前缀和计算出来，奇数位的总和减去删除元素的前缀和，就得到了删除元素后面的后缀和。通过这个办法就可以得到删除元素后面的，奇数位总和，偶数

位总和。注意这个后缀和是包含了删除元素的。所以最后需要判断删除元素是奇数位还是偶数位，如果是奇数位，那么在计算出来的偶数和上再减去这个删除元素；如果是偶数位，就在计算出来的奇数和上再减去这个删除元素。代码见解法二。

- 这一题还有一种更简洁的写法，就是解法一了。通过了解法二的思考，我们可以知道，每次变换以后的操作可以抽象出来，即三步，减去一个数，判断是否相等，再加上一个数。只不过这三步在解法二中都去判断了奇偶性。如果我们不判断奇偶性，那么代码就可以写成解法一的样子。为什么可以不用管奇偶性呢？因为每次删除一个元素以后，下次再删除，奇偶就发生颠倒了，上次的奇数和到了下次就是偶数和了。想通这一点就可以把代码写成解法一的样子。

代码

```go
// 解法一 超简洁写法
func waysToMakeFair(nums []int) int {
  sum, res := [2]int{}, 0
  for i := 0; i < len(nums); i++ {
    sum[i%2] += nums[i]
  }
  for i := 0; i < len(nums); i++ {
    sum[i%2] -= nums[i]
    if sum[i%2] == sum[1-(i%2)] {
      res++
    }
    sum[1-(i%2)] += nums[i]
  }
  return res
}

// 解法二 前缀和，后缀和
func waysToMakeFair1(nums []int) int {
  evenPrefix, oddPrefix, evenSuffix, oddSuffix, res := 0, 0, 0, 0, 0
  for i := 0; i < len(nums); i++ {
    if i%2 == 0 {
      evenSuffix += nums[i]
    } else {
      oddSuffix += nums[i]
    }
  }
  for i := 0; i < len(nums); i++ {
    if i%2 == 0 {
      evenSuffix -= nums[i]
    } else {
      oddSuffix -= nums[i]
    }
    if (evenPrefix + oddSuffix) == (oddPrefix + evenSuffix) {
      res++
    }
    if i%2 == 0 {
      evenPrefix += nums[i]
```

```
    } else {
      oddPrefix += nums[i]
    }
  }
  return res
}
```

1665. Minimum Initial Energy to Finish Tasks

题目

You are given an array `tasks` where `tasks[i] = [actuali, minimumi]`:

- `actuali` is the actual amount of energy you **spend to finish** the `ith` task.
- `minimumi` is the minimum amount of energy you **require to begin** the `ith` task.

For example, if the task is `[10, 12]` and your current energy is `11`, you cannot start this task. However, if your current energy is `13`, you can complete this task, and your energy will be `3` after finishing it.

You can finish the tasks in **any order** you like.

Return *the **minimum** initial amount of energy you will need to finish all the tasks*.

Example 1:

```
Input: tasks = [[1,2],[2,4],[4,8]]
Output: 8
Explanation:
Starting with 8 energy, we finish the tasks in the following order:
    - 3rd task. Now energy = 8 - 4 = 4.
    - 2nd task. Now energy = 4 - 2 = 2.
    - 1st task. Now energy = 2 - 1 = 1.
Notice that even though we have leftover energy, starting with 7 energy does not work
because we cannot do the 3rd task.
```

Example 2:

```
Input: tasks = [[1,3],[2,4],[10,11],[10,12],[8,9]]
Output: 32
Explanation:
Starting with 32 energy, we finish the tasks in the following order:
    - 1st task. Now energy = 32 - 1 = 31.
    - 2nd task. Now energy = 31 - 2 = 29.
    - 3rd task. Now energy = 29 - 10 = 19.
    - 4th task. Now energy = 19 - 10 = 9.
    - 5th task. Now energy = 9 - 8 = 1.
```

Example 3:

```
Input: tasks = [[1,7],[2,8],[3,9],[4,10],[5,11],[6,12]]
Output: 27
Explanation:
Starting with 27 energy, we finish the tasks in the following order:
    - 5th task. Now energy = 27 - 5 = 22.
    - 2nd task. Now energy = 22 - 2 = 20.
    - 3rd task. Now energy = 20 - 3 = 17.
    - 1st task. Now energy = 17 - 1 = 16.
    - 4th task. Now energy = 16 - 4 = 12.
    - 6th task. Now energy = 12 - 6 = 6.
```

Constraints:

- `1 <= tasks.length <= 105`
- `1 <= actualⁱ <= minimumⁱ <= 104`

题目大意

给你一个任务数组 tasks ， 其中 tasks[i] = [actualⁱ, minimumⁱ]：

- actual i 是完成第 i 个任务 需要耗费 的实际能量。
- minimum i 是开始第 i 个任务前需要达到的最低能量。

比方说，如果任务为 [10, 12] 且你当前的能量为 11 ， 那么你不能开始这个任务。如果你当前的能量为 13 ， 你可以完成这个任务，且完成它后剩余能量为 3 。你可以按照 任意顺序 完成任务。请你返回完成所有任务的 最少 初始能量。

解题思路

- 给出一个 task 数组，每个元素代表一个任务，每个任务有实际消费能量值和开始这个任务需要的最低能量。要求输出能完成所有任务的最少初始能量。
- 这一题直觉是贪心。先将任务按照 `minimum - actual` 进行排序。先完成差值大的任务，那么接下来的能量能最大限度的满足接下来的任务。这样可能完成所有任务的可能性越大。循环任务数组的时候，保存当前能量在 `cur` 中，如果当前能量不够开启下一个任务，那么这个差值就是需要弥补的，这些能量就是最少初始能量中的，所以加上这些差值能量。如果当前能量可以开启下一个任务，那么就更新当前能量，减去实际消耗的能量以后，再继续循环。循环结束就能得到最少初始能量了。

代码

```go
package leetcode

import (
  "sort"
)

func minimumEffort(tasks [][]int) int {
  sort.Sort(Task(tasks))
```

```go
  res, cur := 0, 0
  for _, t := range tasks {
    if t[1] > cur {
      res += t[1] - cur
      cur = t[1] - t[0]
    } else {
      cur -= t[0]
    }
  }
  return res
}

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

// Task define
type Task [][]int

func (task Task) Len() int {
  return len(task)
}

func (task Task) Less(i, j int) bool {
  t1, t2 := task[i][1]-task[i][0], task[j][1]-task[j][0]
  if t1 != t2 {
    return t2 < t1
  }
  return task[j][1] < task[i][1]
}

func (task Task) Swap(i, j int) {
  t := task[i]
  task[i] = task[j]
  task[j] = t
}
```

1668. Maximum Repeating Substring

题目

For a string `sequence`, a string `word` is **`k`-repeating** if `word` concatenated `k` times is a substring of `sequence`. The `word`'s **maximum `k`-repeating value** is the highest value `k` where `word` is `k`-repeating in `sequence`. If `word` is not a substring of `sequence`, `word`'s maximum `k`-repeating value is `0`.

Given strings `sequence` and `word`, return *the* ***maximum*** `k` ***-repeating value*** *of* `word` *in* `sequence`.

Example 1:

```
Input: sequence = "ababc", word = "ab"
Output: 2
Explanation: "abab" is a substring in "ababc".
```

Example 2:

```
Input: sequence = "ababc", word = "ba"
Output: 1
Explanation: "ba" is a substring in "ababc". "baba" is not a substring in "ababc".
```

Example 3:

```
Input: sequence = "ababc", word = "ac"
Output: 0
Explanation: "ac" is not a substring in "ababc".
```

Constraints:

- `1 <= sequence.length <= 100`
- `1 <= word.length <= 100`
- `sequence` and `word` contains only lowercase English letters.

题目大意

给你一个字符串 sequence ， 如果字符串 word 连续重复 k 次形成的字符串是 sequence 的一个子字符串，那么单词 word 的 重复值为 k 。单词 word 的 最大重复值 是单词 word 在 sequence 中最大的重复值。如果 word 不是 sequence 的子串，那么重复值 k 为 0 。给你一个字符串 sequence 和 word ， 请你返回 最大重复值 k 。

解题思路

- 循环叠加构造 `word`，每次构造出新的 `word` 都在 `sequence` 查找一次，如果找到就输出叠加次数，否则继续叠加构造，直到字符串长度和 `sequence` 一样长，最终都没有找到则输出 0 。

代码

```go
package leetcode

import (
  "strings"
)

func maxRepeating(sequence string, word string) int {
  for i := len(sequence) / len(word); i >= 0; i-- {
```

```
    tmp := ""
    for j := 0; j < i; j++ {
      tmp += word
    }
    if strings.Contains(sequence, tmp) {
      return i
    }
  }
  return 0
}
```

1669. Merge In Between Linked Lists

题目

You are given two linked lists: `list1` and `list2` of sizes `n` and `m` respectively.

Remove `list1`'s nodes from the `ath` node to the `bth` node, and put `list2` in their place.

The blue edges and nodes in the following figure incidate the result:

Build the result list and return its head.

Example 1:

```
Input: list1 = [0,1,2,3,4,5], a = 3, b = 4, list2 = [1000000,1000001,1000002]
Output: [0,1,2,1000000,1000001,1000002,5]
Explanation: We remove the nodes 3 and 4 and put the entire list2 in their place. The
blue edges and nodes in the above figure indicate the result.
```

Example 2:

```
Input: list1 = [0,1,2,3,4,5,6], a = 2, b = 5, list2 =
[1000000,1000001,1000002,1000003,1000004]
Output: [0,1,1000000,1000001,1000002,1000003,1000004,6]
Explanation: The blue edges and nodes in the above figure indicate the result.
```

Constraints:

- `3 <= list1.length <= 104`
- `1 <= a <= b < list1.length - 1`
- `1 <= list2.length <= 104`

题目大意

给你两个链表 list1 和 list2 ，它们包含的元素分别为 n 个和 m 个。请你将 list1 中第 a 个节点到第 b 个节点删除，并将list2 接在被删除节点的位置。

解题思路

- 简单题，考查链表的基本操作。此题注意 a == b 的情况。

代码

```
func mergeInBetween(list1 *ListNode, a int, b int, list2 *ListNode) *ListNode {
  n := list1
  var startRef, endRef *ListNode
  for i := 0; i <= b; i++ {
    if i == a-1 {
      startRef = n
    }
    if i == b {
      endRef = n
    }
    n = n.Next
  }
  startRef.Next = list2
  n = list2
  for n.Next != nil {
    n = n.Next
  }
  n.Next = endRef.Next
  return list1
}
```

1670. Design Front Middle Back Queue

题目

Design a queue that supports `push` and `pop` operations in the front, middle, and back.

Implement the `FrontMiddleBack` class:

- `FrontMiddleBack()` Initializes the queue.
- `void pushFront(int val)` Adds `val` to the **front** of the queue.
- `void pushMiddle(int val)` Adds `val` to the **middle** of the queue.
- `void pushBack(int val)` Adds `val` to the **back** of the queue.
- `int popFront()` Removes the **front** element of the queue and returns it. If the queue is empty, return `1`.
- `int popMiddle()` Removes the **middle** element of the queue and returns it. If the queue is empty, return `1`.
- `int popBack()` Removes the **back** element of the queue and returns it. If the queue is empty,

return `1`.

Notice that when there are **two** middle position choices, the operation is performed on the **frontmost** middle position choice. For example:

- Pushing `6` into the middle of `[1, 2, 3, 4, 5]` results in `[1, 2, 6, 3, 4, 5]`.
- Popping the middle from `[1, 2, 3, 4, 5, 6]` returns `3` and results in `[1, 2, 4, 5, 6]`.

Example 1:

```
Input:
["FrontMiddleBackQueue", "pushFront", "pushBack", "pushMiddle", "pushMiddle",
"popFront", "popMiddle", "popMiddle", "popBack", "popFront"]
[[], [1], [2], [3], [4], [], [], [], [], []]
Output:
[null, null, null, null, null, 1, 3, 4, 2, -1]

Explanation:
FrontMiddleBackQueue q = new FrontMiddleBackQueue();
q.pushFront(1);   // [1]
q.pushBack(2);    // [1, 2]
q.pushMiddle(3);  // [1, 3, 2]
q.pushMiddle(4);  // [1, 4, 3, 2]
q.popFront();     // return 1 -> [4, 3, 2]
q.popMiddle();    // return 3 -> [4, 2]
q.popMiddle();    // return 4 -> [2]
q.popBack();      // return 2 -> []
q.popFront();     // return -1 -> [] (The queue is empty)
```

Constraints:

- `1 <= val <= 109`
- At most `1000` calls will be made to `pushFront`, `pushMiddle`, `pushBack`, `popFront`, `popMiddle`, and `popBack`.

题目大意

请你设计一个队列，支持在前，中，后三个位置的 push 和 pop 操作。

请你完成 FrontMiddleBack 类：

- FrontMiddleBack() 初始化队列。
- void pushFront(int val) 将 val 添加到队列的 最前面 。
- void pushMiddle(int val) 将 val 添加到队列的 正中间 。
- void pushBack(int val) 将 val 添加到队里的 最后面 。
- int popFront() 将 最前面 的元素从队列中删除并返回值，如果删除之前队列为空，那么返回 -1 。
- int popMiddle() 将 正中间 的元素从队列中删除并返回值，如果删除之前队列为空，那么返回 -1 。
- int popBack() 将 最后面 的元素从队列中删除并返回值，如果删除之前队列为空，那么返回 -1 。

请注意当有 两个 中间位置的时候，选择靠前面的位置进行操作。比方说：

- 将 6 添加到 [1, 2, 3, 4, 5] 的中间位置，结果数组为 [1, 2, 6, 3, 4, 5] 。
- 从 [1, 2, 3, 4, 5, 6] 的中间位置弹出元素，返回 3 ，数组变为 [1, 2, 4, 5, 6] 。

解题思路

- 简单题，利用 go 原生的双向队列 list 的实现，可以轻松实现这个“前中后队列”。
- 具体实现见代码，几组特殊测试用例见测试文件。

代码

```go
package leetcode

import (
  "container/list"
)

type FrontMiddleBackQueue struct {
  list   *list.List
  middle *list.Element
}

func Constructor() FrontMiddleBackQueue {
  return FrontMiddleBackQueue{list: list.New()}
}

func (this *FrontMiddleBackQueue) PushFront(val int) {
  e := this.list.PushFront(val)
  if this.middle == nil {
    this.middle = e
  } else if this.list.Len()%2 == 0 && this.middle.Prev() != nil {
    this.middle = this.middle.Prev()
  }
}

func (this *FrontMiddleBackQueue) PushMiddle(val int) {
  if this.middle == nil {
    this.PushFront(val)
  } else {
    if this.list.Len()%2 != 0 {
      this.middle = this.list.InsertBefore(val, this.middle)
    } else {
      this.middle = this.list.InsertAfter(val, this.middle)
    }
  }
}

func (this *FrontMiddleBackQueue) PushBack(val int) {
```

```go
  e := this.list.PushBack(val)
  if this.middle == nil {
    this.middle = e
  } else if this.list.Len()%2 != 0 && this.middle.Next() != nil {
    this.middle = this.middle.Next()
  }
}

func (this *FrontMiddleBackQueue) PopFront() int {
  if this.list.Len() == 0 {
    return -1
  }
  e := this.list.Front()
  if this.list.Len() == 1 {
    this.middle = nil
  } else if this.list.Len()%2 == 0 && this.middle.Next() != nil {
    this.middle = this.middle.Next()
  }
  return this.list.Remove(e).(int)
}

func (this *FrontMiddleBackQueue) PopMiddle() int {
  if this.middle == nil {
    return -1
  }
  e := this.middle
  if this.list.Len()%2 != 0 {
    this.middle = e.Prev()
  } else {
    this.middle = e.Next()
  }
  return this.list.Remove(e).(int)
}

func (this *FrontMiddleBackQueue) PopBack() int {
  if this.list.Len() == 0 {
    return -1
  }
  e := this.list.Back()
  if this.list.Len() == 1 {
    this.middle = nil
  } else if this.list.Len()%2 != 0 && this.middle.Prev() != nil {
    this.middle = this.middle.Prev()
  }
  return this.list.Remove(e).(int)
}

/**
 * Your FrontMiddleBackQueue object will be instantiated and called as such:
```

```
 * obj := Constructor();
 * obj.PushFront(val);
 * obj.PushMiddle(val);
 * obj.PushBack(val);
 * param_4 := obj.PopFront();
 * param_5 := obj.PopMiddle();
 * param_6 := obj.PopBack();
 */
```

1672. Richest Customer Wealth

题目

You are given an `m x n` integer grid `accounts` where `accounts[i][j]` is the amount of money the `ith` customer has in the `jth` bank. Return *the **wealth** that the richest customer has.*

A customer's **wealth** is the amount of money they have in all their bank accounts. The richest customer is the customer that has the maximum **wealth**.

Example 1:

```
Input: accounts = [[1,2,3],[3,2,1]]
Output: 6
Explanation:1st customer has wealth = 1 + 2 + 3 = 6
2nd customer has wealth = 3 + 2 + 1 = 6
Both customers are considered the richest with a wealth of 6 each, so return 6.
```

Example 2:

```
Input: accounts = [[1,5],[7,3],[3,5]]
Output: 10
Explanation:
1st customer has wealth = 6
2nd customer has wealth = 10
3rd customer has wealth = 8
The 2nd customer is the richest with a wealth of 10.
```

Example 3:

```
Input: accounts = [[2,8,7],[7,1,3],[1,9,5]]
Output: 17
```

Constraints:

- `m == accounts.length`
- `n == accounts[i].length`
- `1 <= m, n <= 50`

- `1 <= accounts[i][j] <= 100`

题目大意

给你一个 m x n 的整数网格 accounts ， 其中 accounts[i][j] 是第 i 位客户在第 j 家银行托管的资产数量。返回最富有客户所拥有的 资产总量 。客户的 资产总量 就是他们在各家银行托管的资产数量之和。最富有客户就是 资产总量 最大的客户。

解题思路

- 简单题。计算二维数组中每个一位数组的元素总和，然后动态维护这些一位数组和的最大值即可。

代码

```go
package leetcode

func maximumWealth(accounts [][]int) int {
  res := 0
  for _, banks := range accounts {
    sAmount := 0
    for _, amount := range banks {
      sAmount += amount
    }
    if sAmount > res {
      res = sAmount
    }
  }
  return res
}
```

1673. Find the Most Competitive Subsequence

题目

Given an integer array `nums` and a positive integer `k`, return *the most **competitive** subsequence of* `nums` *of size* `k`.

An array's subsequence is a resulting sequence obtained by erasing some (possibly zero) elements from the array.

We define that a subsequence `a` is more **competitive** than a subsequence `b` (of the same length) if in the first position where `a` and `b` differ, subsequence `a` has a number **less** than the corresponding number in `b`. For example, `[1,3,4]` is more competitive than `[1,3,5]` because the first position they differ is at the final number, and `4` is less than `5`.

Example 1:

```
Input: nums = [3,5,2,6], k = 2
Output: [2,6]
Explanation: Among the set of every possible subsequence: {[3,5], [3,2], [3,6], [5,2],
[5,6], [2,6]}, [2,6] is the most competitive.
```

Example 2:

```
Input: nums = [2,4,3,3,5,4,9,6], k = 4
Output: [2,3,3,4]
```

Constraints:

- `1 <= nums.length <= 105`
- `0 <= nums[i] <= 109`
- `1 <= k <= nums.length`

题目大意

给你一个整数数组 nums 和一个正整数 k ，返回长度为 k 且最具 竞争力 的 nums 子序列。数组的子序列是从数组中删除一些元素（可能不删除元素）得到的序列。

在子序列 a 和子序列 b 第一个不相同的位置上，如果 a 中的数字小于 b 中对应的数字，那么我们称子序列 a 比子序列 b（相同长度下）更具 竞争力 。 例如，[1,3,4] 比 [1,3,5] 更具竞争力，在第一个不相同的位置，也就是最后一个位置上， 4 小于 5 。

解题思路

- 这一题是单调栈的典型题型。利用单调栈，可以保证原数组中元素相对位置不变，这满足题意中删除元素但不移动元素的要求。单调栈又能保证每次进栈，元素是最小的。
- 类似的题目还有第 42 题，第 84 题，第 496 题，第 503 题，第 856 题，第 901 题，第 907 题，第 1130 题，第 1425 题，第 1673 题。

代码

```go
package leetcode

// 单调栈
func mostCompetitive(nums []int, k int) []int {
  stack := make([]int, 0, len(nums))
  for i := 0; i < len(nums); i++ {
    for len(stack)+len(nums)-i > k && len(stack) > 0 && nums[i] < stack[len(stack)-1] {
      stack = stack[:len(stack)-1]
    }
    stack = append(stack, nums[i])
  }
  return stack[:k]
}
```

1674. Minimum Moves to Make Array Complementary

题目

You are given an integer array `nums` of **even** length `n` and an integer `limit`. In one move, you can replace any integer from `nums` with another integer between `1` and `limit`, inclusive.

The array `nums` is **complementary** if for all indices `i` (**0-indexed**), `nums[i] + nums[n - 1 - i]` equals the same number. For example, the array `[1,2,3,4]` is complementary because for all indices `i`, `nums[i] + nums[n - 1 - i] = 5`.

Return the ***minimum** number of moves required to make* `nums` ***complementary***.

Example 1:

```
Input: nums = [1,2,4,3], limit = 4
Output: 1
Explanation: In 1 move, you can change nums to [1,2,2,3] (underlined elements are
changed).
nums[0] + nums[3] = 1 + 3 = 4.
nums[1] + nums[2] = 2 + 2 = 4.
nums[2] + nums[1] = 2 + 2 = 4.
nums[3] + nums[0] = 3 + 1 = 4.
Therefore, nums[i] + nums[n-1-i] = 4 for every i, so nums is complementary.
```

Example 2:

```
Input: nums = [1,2,2,1], limit = 2
Output: 2
Explanation: In 2 moves, you can change nums to [2,2,2,2]. You cannot change any number
to 3 since 3 > limit.
```

Example 3:

```
Input: nums = [1,2,1,2], limit = 2
Output: 0
Explanation: nums is already complementary.
```

Constraints:

- `n == nums.length`
- `2 <= n <= 105`
- `1 <= nums[i] <= limit <= 105`
- `n` is even.

题目大意

给你一个长度为 偶数 n 的整数数组 nums 和一个整数 limit 。每一次操作，你可以将 nums 中的任何整数替换为 1 到 limit 之间的另一个整数。

如果对于所有下标 i（下标从 0 开始），nums[i] + nums[n - 1 - i] 都等于同一个数，则数组 nums 是 互补的 。例如，数组 [1,2,3,4] 是互补的，因为对于所有下标 i ，nums[i] + nums[n - 1 - i] = 5 。

返回使数组 互补 的 最少 操作次数。

解题思路

- 这一题考察的是差分数组。通过分析题意，可以得出，针对每一个 `sum` 的取值范围是 `[2, 2* limt]`，定义 `a = min(nums[i], nums[n - i - 1])`，`b = max(nums[i], nums[n - i - 1])`，在这个区间内，又可以细分成 5 个区间，`[2, a + 1)`，`[a + 1, a + b)`，`[a + b + 1, a + b + 1)`，`[a + b + 1, b + limit + 1)`，`[b + limit + 1, 2 * limit)`，在这 5 个区间内使得数组互补的最小操作次数分别是 `2(减少 a, 减少 b)`，`1(减少 b)`，`0(不用操作)`，`1(增大 a)`，`+2(增大 a, 增大 b)`，换个表达方式，按照扫描线从左往右扫描，在这 5 个区间内使得数组互补的最小操作次数叠加变化分别是 `+2(减少 a, 减少 b)`，`-1(减少 a)`，`-1(不用操作)`，`+1(增大 a)`，`+1(增大 a, 增大 b)`，利用这前后两个区间的关系，就可以构造一个差分数组。差分数组反应的是前后两者的关系。如果想求得 0 ~ n 的总关系，只需要求一次前缀和即可。
- 这道题要求输出最少的操作次数，所以利用差分数组 + 前缀和，累加前缀和的同时维护最小值。从左往右扫描完一遍以后，输出最小值即可。

代码

```go
package leetcode

func minMoves(nums []int, limit int) int {
  diff := make([]int, limit*2+2) // nums[i] <= limit, b+limit+1 is maximum
limit+limit+1
  for j := 0; j < len(nums)/2; j++ {
    a, b := min(nums[j], nums[len(nums)-j-1]), max(nums[j], nums[len(nums)-j-1])
    // using prefix sum: most interesting point, and is the key to reduce complexity
    diff[2] += 2
```

```
    diff[a+1]--
    diff[a+b]--
    diff[a+b+1]++
    diff[b+limit+1]++
  }
  cur, res := 0, len(nums)
  for i := 2; i <= 2*limit; i++ {
    cur += diff[i]
    res = min(res, cur)
  }
  return res
}

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
}
```

1675. Minimize Deviation in Array

题目

You are given an array `nums` of `n` positive integers.

You can perform two types of operations on any element of the array any number of times:

- If the element is **even**, **divide** it by `2`.
  - For example, if the array is `[1,2,3,4]`, then you can do this operation on the last element, and the array will be `[1,2,3,2].`
- If the element is **odd**, **multiply** it by `2`.
  - For example, if the array is `[1,2,3,4]`, then you can do this operation on the first element, and the array will be `[2,2,3,4].`

The **deviation** of the array is the **maximum difference** between any two elements in the array.

Return *the **minimum deviation** the array can have after performing some number of operations.*

Example 1:

```
Input: nums = [1,2,3,4]
Output: 1
Explanation: You can transform the array to [1,2,3,2], then to [2,2,3,2], then the deviation will be 3 - 2 = 1.
```

Example 2:

```
Input: nums = [4,1,5,20,3]
Output: 3
Explanation: You can transform the array after two operations to [4,2,5,5,3], then the deviation will be 5 - 2 = 3.
```

Example 3:

```
Input: nums = [2,10,8]
Output: 3
```

Constraints:

- `n == nums.length`
- `2 <= n <= 105`
- `1 <= nums[i] <= 10^9`

题目大意

给你一个由 n 个正整数组成的数组 nums 。你可以对数组的任意元素执行任意次数的两类操作：

- 如果元素是 偶数 ，除以 2。例如，如果数组是 [1,2,3,4] ，那么你可以对最后一个元素执行此操作，使其变成 [1,2,3,2]
- 如果元素是 奇数 ，乘上 2。例如，如果数组是 [1,2,3,4] ，那么你可以对第一个元素执行此操作，使其变成 [2,2,3,4]
  数组的 偏移量 是数组中任意两个元素之间的 最大差值 。

返回数组在执行某些操作之后可以拥有的 最小偏移量 。

解题思路

- 要找到最小偏移量，即需要令最大值变小，最小值变大。要想达到这个要求，需要对奇数偶数做乘法和除法。这里特殊的是，奇数一旦乘以 2 以后，就变成偶数了。偶数除以 2 以后可能是奇数也可能是偶数。所以可以先将所有的奇数都乘以 2 统一变成偶数。
- 第二轮不断的将最大值除 2，直到最大值为奇数，不能再操作了。每轮循环中把比 min 值大的偶数也都除以 2 。这里除以 2 有 2 个目的，一个目的是将第一步奇数乘 2 还原回去，另一个目的是将本来的偶数除以 2 。可能有人有疑问，为什么只把最大值变小，没有将最小值变大呢？如果最小值是奇数，那么它一定是由上一个偶数除以 2 变过来的，我们在上一个状态已经计算过这个偶数了，因此没必要扩大它；如果最小值是偶数，那么它一定会在某一轮的除 2 操作中，不操作，即它不会满足 `min <= nums[i]/2` 这个条件。每次循环都更新该次循环的最大值和最小值，并记录偏移量。不断的循环，直到最大值为奇数，退出循环。最终输出最小偏移量。

代码

```go
package leetcode

func minimumDeviation(nums []int) int {
  min, max := 0, 0
  for i := range nums {
    if nums[i]%2 == 1 {
      nums[i] *= 2
    }
    if i == 0 {
      min = nums[i]
      max = nums[i]
    } else if nums[i] < min {
      min = nums[i]
    } else if max < nums[i] {
      max = nums[i]
    }
  }
  res := max - min
  for max%2 == 0 {
    tmax, tmin := 0, 0
    for i := range nums {
      if nums[i] == max || (nums[i]%2 == 0 && min <= nums[i]/2) {
        nums[i] /= 2
      }
      if i == 0 {
        tmin = nums[i]
        tmax = nums[i]
      } else if nums[i] < tmin {
        tmin = nums[i]
      } else if tmax < nums[i] {
        tmax = nums[i]
      }
    }
    if tmax-tmin < res {
      res = tmax - tmin
    }
    min, max = tmin, tmax
  }
  return res
}
```

1678. Goal Parser Interpretation

题目

You own a **Goal Parser** that can interpret a string `command`. The `command` consists of an alphabet of `"G"`, `"()"` and/or `"(al)"` in some order. The Goal Parser will interpret `"G"` as the string `"G"`, `"()"` as the string `"o"`, and `"(al)"` as the string `"al"`. The interpreted strings are then concatenated in the original order.

Given the string `command`, return *the **Goal Parser**'s interpretation of* `command`.

Example 1:

```
Input: command = "G()(al)"
Output: "Goal"
Explanation: The Goal Parser interprets the command as follows:
G -> G
() -> o
(al) -> al
The final concatenated result is "Goal".
```

Example 2:

```
Input: command = "G()()()()(al)"
Output: "Gooooal"
```

Example 3:

```
Input: command = "(al)G(al)()()G"
Output: "alGalooG"
```

Constraints:

- `1 <= command.length <= 100`
- `command` consists of `"G"`, `"()"`, and/or `"(al)"` in some order.

题目大意

请你设计一个可以解释字符串 command 的 Goal 解析器 。command 由 "G"、"()" 和/或 "(al)" 按某种顺序组成。Goal 解析器会将 "G" 解释为字符串 "G"、"()" 解释为字符串 "o" ， "(al)" 解释为字符串 "al" 。然后，按原顺序将经解释得到的字符串连接成一个字符串。给你字符串 command ， 返回 Goal 解析器 对 command 的解释结果。

解题思路

- 简单题，按照题意修改字符串即可。由于是简单题，这一题也不用考虑嵌套的情况。

代码

```go
package leetcode

func interpret(command string) string {
  if command == "" {
```

```
    return ""
  }
  res := ""
  for i := 0; i < len(command); i++ {
    if command[i] == 'G' {
      res += "G"
    } else {
      if command[i] == '(' && command[i+1] == 'a' {
        res += "al"
        i += 3
      } else {
        res += "o"
        i ++
      }
    }
  }
  return res
}
```

1679. Max Number of K-Sum Pairs

题目

You are given an integer array `nums` and an integer `k`.

In one operation, you can pick two numbers from the array whose sum equals `k` and remove them from the array.

Return *the maximum number of operations you can perform on the array*.

Example 1:

```
Input: nums = [1,2,3,4], k = 5
Output: 2
Explanation: Starting with nums = [1,2,3,4]:
- Remove numbers 1 and 4, then nums = [2,3]
- Remove numbers 2 and 3, then nums = []
There are no more pairs that sum up to 5, hence a total of 2 operations.
```

Example 2:

```
Input: nums = [3,1,3,4,3], k = 6
Output: 1
Explanation: Starting with nums = [3,1,3,4,3]:
- Remove the first two 3's, then nums = [1,4,3]
There are no more pairs that sum up to 6, hence a total of 1 operation.
```

Constraints:

- `1 <= nums.length <= 105`
- `1 <= nums[i] <= 109`
- `1 <= k <= 109`

题目大意

给你一个整数数组 nums 和一个整数 k 。每一步操作中，你需要从数组中选出和为 k 的两个整数，并将它们移出数组。返回你可以对数组执行的最大操作数。

解题思路

- 读完题第一感觉这道题是 TWO SUM 题目的加强版。需要找到所有满足和是 k 的数对。先考虑能不能找到两个数都是 k/2 ，如果能找到多个这样的数，可以先移除他们。其次在利用 TWO SUM 的思路，找出和为 k 的数对。利用 TWO SUM 里面 map 的做法，时间复杂度 O(n)。

代码

```go
package leetcode

// 解法一 优化版
func maxOperations(nums []int, k int) int {
  counter, res := make(map[int]int), 0
  for _, n := range nums {
    counter[n]++
  }
  if (k & 1) == 0 {
    res += counter[k>>1] >> 1
    // 能够由 2 个相同的数构成 k 的组合已经都排除出去了，剩下的一个单独的也不能组成 k 了
    // 所以这里要把它的频次置为 0 。如果这里不置为 0，下面代码判断逻辑还需要考虑重复使用数字的情况
    counter[k>>1] = 0
  }
  for num, freq := range counter {
    if num <= k/2 {
      remain := k - num
      if counter[remain] < freq {
        res += counter[remain]
      } else {
        res += freq
      }
    }
  }
  return res
}

// 解法二
func maxOperations_(nums []int, k int) int {
  counter, res := make(map[int]int), 0
  for _, num := range nums {
```

```
    counter[num]++
    remain := k - num
    if num == remain {
      if counter[num] >= 2 {
        res++
        counter[num] -= 2
      }
    } else {
      if counter[remain] > 0 {
        res++
        counter[remain]--
        counter[num]--
      }
    }
  }
  return res
}
```

1680. Concatenation of Consecutive Binary Numbers

题目

Given an integer `n`, return *the **decimal value** of the binary string formed by concatenating the binary representations of* `1` *to* `n` *in order, **modulo*** `109 + 7`.

Example 1:

```
Input: n = 1
Output: 1
Explanation: "1" in binary corresponds to the decimal value 1.
```

Example 2:

```
Input: n = 3
Output: 27
Explanation: In binary, 1, 2, and 3 corresponds to "1", "10", and "11".
After concatenating them, we have "11011", which corresponds to the decimal value 27.
```

Example 3:

```
Input: n = 12
Output: 505379714
Explanation: The concatenation results in "110111001011101111000100110101011110 0".
The decimal value of that is 118505380540.
After modulo 109 + 7, the result is 505379714.
```

Constraints:

- `1 <= n <= 10^5`

题目大意

给你一个整数 n ， 请你将 1 到 n 的二进制表示连接起来，并返回连接结果对应的 十进制 数字对 10^9 + 7 取余的结果。

解题思路

- 理解题意以后，先找到如何拼接最终二进制数的规律。假设 `f(n)` 为最终变换以后的十进制数。那么根据题意，`f(n) = f(n-1) << shift + n` 这是一个递推公式。`shift` 左移的位数就是 `n` 的二进制对应的长度。`shift` 的值是随着 `n` 变化而变化的。由二进制进位规律可以知道，2 的整数次幂的时候，对应的二进制长度会增加 1 位。这里可以利用位运算来判断是否是 2 的整数次幂。
- 这道题另外一个需要处理的是模运算的法则。此题需要用到模运算的加法法则。

  ```
  模运算与基本四则运算有些相似，但是除法例外。
  (a + b) % p = (a % p + b % p) % p  (1)
  (a - b) % p = (a % p - b % p) % p  (2)
  (a * b) % p = (a % p * b % p) % p  (3)
  a ^ b % p = ((a % p)^b) % p  (4)
  结合律:
  ((a+b) % p + c) % p = (a + (b+c) % p) % p  (5)
  ((a*b) % p * c)% p = (a * (b*c) % p) % p  (6)
  交换律:
  (a + b) % p = (b+a) % p  (7)
  (a * b) % p = (b * a) % p  (8)
  分配律:
  ((a +b)% p * c) % p = ((a * c) % p + (b * c) % p) % p  (9)
  ```

  这一题需要用到模运算的加法运算法则。

代码

```go
package leetcode

import (
  "math/bits"
)

// 解法一 模拟
func concatenatedBinary(n int) int {
  res, mod, shift := 0, 1000000007, 0
  for i := 1; i <= n; i++ {
    if (i & (i - 1)) == 0 {
      shift++
    }
```

```go
    res = ((res << shift) + i) % mod
  }
  return res
}

// 解法二 位运算
func concatenatedBinary1(n int) int {
  res := 0
  for i := 1; i <= n; i++ {
    res = (res<<bits.Len(uint(i)) | i) % (1e9 + 7)
  }
  return res
}
```

1681. Minimum Incompatibility

题目

You are given an integer array `nums` and an integer `k`. You are asked to distribute this array into `k` subsets of **equal size** such that there are no two equal elements in the same subset.

A subset's **incompatibility** is the difference between the maximum and minimum elements in that array.

Return *the **minimum possible sum of incompatibilities** of the `k` subsets after distributing the array optimally, or return `-1` if it is not possible.*

A subset is a group integers that appear in the array with no particular order.

Example 1:

```
Input: nums = [1,2,1,4], k = 2
Output: 4
Explanation: The optimal distribution of subsets is [1,2] and [1,4].
The incompatibility is (2-1) + (4-1) = 4.
Note that [1,1] and [2,4] would result in a smaller sum, but the first subset contains
2 equal elements.
```

Example 2:

```
Input: nums = [6,3,8,1,3,1,2,2], k = 4
Output: 6
Explanation: The optimal distribution of subsets is [1,2], [2,3], [6,8], and [1,3].
The incompatibility is (2-1) + (3-2) + (8-6) + (3-1) = 6.
```

Example 3:

```
Input: nums = [5,3,3,6,3,3], k = 3
Output: -1
Explanation: It is impossible to distribute nums into 3 subsets where no two elements are equal in the same subset.
```

Constraints:

- `1 <= k <= nums.length <= 16`
- `nums.length` is divisible by `k`
- `1 <= nums[i] <= nums.length`

题目大意

给你一个整数数组 nums 和一个整数 k 。你需要将这个数组划分到 k 个相同大小的子集中，使得同一个子集里面没有两个相同的元素。一个子集的 不兼容性 是该子集里面最大值和最小值的差。

请你返回将数组分成 k 个子集后，各子集 **不兼容性** 的 **和** 的 **最小值** ，如果无法分成分成 k 个子集，返回 -1 。子集的定义是数组中一些数字的集合，对数字顺序没有要求。

解题思路

- 读完题最直白的思路就是 DFS。做法类似第 77 题。这里就不赘述了。可以见第 77 题题解。
- 这一题还需要用到贪心的思想。每次取数都取最小的数。这样可以不会让最大数和最小数在一个集合中。由于每次取数都是取最小的，那么能保证不兼容性每次都尽量最小。于是在 order 数组中定义取数的顺序。然后再把数组从小到大排列。这样每次按照 order 顺序取数，都是取的最小值。
- 正常的 DFS 写完提交，耗时是很长的。大概是 1532ms。如何优化到极致呢？这里需要加上 2 个剪枝条件。第一个剪枝条件比较简单，如果累计 sum 比之前存储的 res 大，那么直接 return，不需要继续递归了。第二个剪枝条件就非常重要了，可以一下子减少很多次递归。每次取数产生新的集合的时候，要从第一个最小数开始取，一旦取了，后面就不需要再循环递归了。举个例子，[1,2,3,4]，第一个数如果取 2，集合可以是 [[2,3], [1,4]] 或 [[2,4], [1,3]], 这个集合和[[1,3],[2,4]]、[[1,4], [2,3]] 情况一样。可以看到如果取出第一个最小值以后，后面的循环是不必要的了。所以在取下标为 0 的数的时候，递归到底层以后，返回就可以直接 break，不用接下去的循环了，接下去的循环和递归是不必要的。每组组内的顺序我们并不关心，只要最大值和最小值在分组内即可。另外组间顺序我们也不关心。所以可以把排列问题 O(n!) 时间复杂度降低到组合问题 O(2^n)。加了这 2 个剪枝条件以后，耗时就变成了 0ms 了。beats 100%

代码

```go
package leetcode

import (
  "math"
  "sort"
)

func minimumIncompatibility(nums []int, k int) int {
  sort.Ints(nums)
  eachSize, counts := len(nums)/k, make([]int, len(nums)+1)
```

```go
  for i := range nums {
    counts[nums[i]]++
    if counts[nums[i]] > k {
      return -1
    }
  }
  orders := []int{}
  for i := range counts {
    orders = append(orders, i)
  }
  sort.Ints(orders)
  res := math.MaxInt32
  generatePermutation1681(nums, counts, orders, 0, 0, eachSize, &res, []int{})
  if res == math.MaxInt32 {
    return -1
  }
  return res
}

func generatePermutation1681(nums, counts, order []int, index, sum, eachSize int, res
*int, current []int) {
  if len(current) > 0 && len(current)%eachSize == 0 {
    sum += current[len(current)-1] - current[len(current)-eachSize]
    index = 0
  }
  if sum >= *res {
    return
  }
  if len(current) == len(nums) {
    if sum < *res {
      *res = sum
    }
    return
  }
  for i := index; i < len(counts); i++ {
    if counts[order[i]] == 0 {
      continue
    }
    counts[order[i]]--
    current = append(current, order[i])
    generatePermutation1681(nums, counts, order, i+1, sum, eachSize, res, current)
    current = current[:len(current)-1]
    counts[order[i]]++
    // 这里是关键的剪枝
    if index == 0 {
      break
    }
  }
}
```

1684. Count the Number of Consistent Strings

题目

You are given a string `allowed` consisting of **distinct** characters and an array of strings `words`. A string is **consistent** if all characters in the string appear in the string `allowed`.

Return *the number of **consistent** strings in the array* `words`.

Example 1:

```
Input: allowed = "ab", words = ["ad","bd","aaab","baa","badab"]
Output: 2
Explanation: Strings "aaab" and "baa" are consistent since they only contain characters
'a' and 'b'.
```

Example 2:

```
Input: allowed = "abc", words = ["a","b","c","ab","ac","bc","abc"]
Output: 7
Explanation: All strings are consistent.
```

Example 3:

```
Input: allowed = "cad", words = ["cc","acd","b","ba","bac","bad","ac","d"]
Output: 4
Explanation: Strings "cc", "acd", "ac", and "d" are consistent.
```

Constraints:

- `1 <= words.length <= 104`
- `1 <= allowed.length <= 26`
- `1 <= words[i].length <= 10`
- The characters in `allowed` are **distinct**.
- `words[i]` and `allowed` contain only lowercase English letters.

题目大意

给你一个由不同字符组成的字符串 `allowed` 和一个字符串数组 `words` 。如果一个字符串的每一个字符都在 `allowed` 中，就称这个字符串是 一致字符串 。

请你返回 `words` 数组中 一致字符串 的数目。

解题思路

- 简单题。先将 `allowed` 转化成 map。将 `words` 数组中每个单词的字符都在 map 中查找一遍，如果都存在就累加 res。如果有不存在的字母，不累加。最终输出 res 即可。

代码

```
package leetcode

func countConsistentStrings(allowed string, words []string) int {
  allowedMap, res, flag := map[rune]int{}, 0, true
  for _, str := range allowed {
    allowedMap[str]++
  }
  for i := 0; i < len(words); i++ {
    flag = true
    for j := 0; j < len(words[i]); j++ {
      if _, ok := allowedMap[rune(words[i][j])]; !ok {
        flag = false
        break
      }
    }
    if flag {
      res++
    }
  }
  return res
}
```

1685. Sum of Absolute Differences in a Sorted Array

题目

You are given an integer array `nums` sorted in **non-decreasing** order.

Build and return *an integer array* `result` *with the same length as* `nums` *such that* `result[i]` *is equal to the* ***summation of absolute differences*** *between* `nums[i]` *and all the other elements in the array.*

In other words, `result[i]` is equal to `sum(|nums[i]-nums[j]|)` where `0 <= j < nums.length` and `j != i` (**0-indexed**).

Example 1:

```
Input: nums = [2,3,5]
Output: [4,3,5]
Explanation: Assuming the arrays are 0-indexed, then
result[0] = |2-2| + |2-3| + |2-5| = 0 + 1 + 3 = 4,
result[1] = |3-2| + |3-3| + |3-5| = 1 + 0 + 2 = 3,
result[2] = |5-2| + |5-3| + |5-5| = 3 + 2 + 0 = 5.
```

Example 2:

```
Input: nums = [1,4,6,8,10]
Output: [24,15,13,15,21]
```

Constraints:

- `2 <= nums.length <= 105`
- `1 <= nums[i] <= nums[i + 1] <= 104`

题目大意

给你一个 非递减 有序整数数组 `nums` 。请你建立并返回一个整数数组 `result`，它跟 `nums` 长度相同，且 `result[i]` 等于 `nums[i]` 与数组中所有其他元素差的绝对值之和。换句话说， `result[i]` 等于 `sum(|nums[i]-nums[j]|)` ，其中 `0 <= j < nums.length` 且 `j != i` （下标从 0 开始）。

解题思路

- 利用前缀和思路解题。题目中说明了是有序数组，所以在计算绝对值的时候可以拆开绝对值符号。假设要计算当前 `result[i]`，以 `i` 为界，把原数组 `nums` 分成了 3 段。`nums[0 ~ i-1]` 和 `nums[i+1 ~ n]`，前面一段 `nums[0 ~ i-1]` 中的每个元素都比 `nums[i]` 小，拆掉绝对值以后，`sum(|nums[i]-nums[j]|) = nums[i] * i - prefixSum[0 ~ i-1]`，后面一段 `nums[i+1 ~ n]` 中的每个元素都比 `nums[i]` 大，拆掉绝对值以后，`sum(|nums[i]-nums[j]|) = prefixSum[i+1 ~ n] - nums[i] * (n - 1 - i)`。特殊的情况，`i = 0` 和 `i = n` 的情况特殊处理一下就行。

代码

```go
package leetcode

//解法一 优化版 prefixSum + sufixSum
func getSumAbsoluteDifferences(nums []int) []int {
  size := len(nums)
  sufixSum := make([]int, size)
  sufixSum[size-1] = nums[size-1]
  for i := size - 2; i >= 0; i-- {
    sufixSum[i] = sufixSum[i+1] + nums[i]
  }
  ans, preSum := make([]int, size), 0
  for i := 0; i < size; i++ {
    // 后面可以加到的值
```

```
    res, sum := 0, sufixSum[i]-nums[i]
    res += (sum - (size-i-1)*nums[i])
    // 前面可以加到的值
    res += (i*nums[i] - preSum)
    ans[i] = res
    preSum += nums[i]
  }
  return ans
}

// 解法二 prefixSum
func getSumAbsoluteDifferences1(nums []int) []int {
  preSum, res, sum := []int{}, []int{}, nums[0]
  preSum = append(preSum, nums[0])
  for i := 1; i < len(nums); i++ {
    sum += nums[i]
    preSum = append(preSum, sum)
  }
  for i := 0; i < len(nums); i++ {
    if i == 0 {
      res = append(res, preSum[len(nums)-1]-preSum[0]-nums[i]*(len(nums)-1))
    } else if i > 0 && i < len(nums)-1 {
      res = append(res, preSum[len(nums)-1]-preSum[i]-preSum[i-1]+nums[i]*i-nums[i]*
(len(nums)-1-i))
    } else {
      res = append(res, nums[i]*len(nums)-preSum[len(nums)-1])
    }
  }
  return res
}
```

1688. Count of Matches in Tournament

题目

You are given an integer `n`, the number of teams in a tournament that has strange rules:

- If the current number of teams is **even**, each team gets paired with another team. A total of `n / 2` matches are played, and `n / 2` teams advance to the next round.
- If the current number of teams is **odd**, one team randomly advances in the tournament, and the rest gets paired. A total of `(n - 1) / 2` matches are played, and `(n - 1) / 2 + 1` teams advance to the next round.

Return *the number of matches played in the tournament until a winner is decided.*

Example 1:

```
Input: n = 7
Output: 6
Explanation: Details of the tournament:
- 1st Round: Teams = 7, Matches = 3, and 4 teams advance.
- 2nd Round: Teams = 4, Matches = 2, and 2 teams advance.
- 3rd Round: Teams = 2, Matches = 1, and 1 team is declared the winner.
Total number of matches = 3 + 2 + 1 = 6.
```

Example 2:

```
Input: n = 14
Output: 13
Explanation: Details of the tournament:
- 1st Round: Teams = 14, Matches = 7, and 7 teams advance.
- 2nd Round: Teams = 7, Matches = 3, and 4 teams advance.
- 3rd Round: Teams = 4, Matches = 2, and 2 teams advance.
- 4th Round: Teams = 2, Matches = 1, and 1 team is declared the winner.
Total number of matches = 7 + 3 + 2 + 1 = 13.
```

Constraints:

- `1 <= n <= 200`

题目大意

给你一个整数 n ，表示比赛中的队伍数。比赛遵循一种独特的赛制：

- 如果当前队伍数是 偶数 ，那么每支队伍都会与另一支队伍配对。总共进行 n / 2 场比赛，且产生 n / 2 支队伍进入下一轮。
- 如果当前队伍数为 奇数 ，那么将会随机轮空并晋级一支队伍，其余的队伍配对。总共进行 (n - 1) / 2 场比赛，且产生 (n - 1) / 2 + 1 支队伍进入下一轮。

返回在比赛中进行的配对次数，直到决出获胜队伍为止。

解题思路

- 简单题，按照题目的规则模拟。
- 这一题还有更加简洁的代码，见解法一。n 个队伍，一个冠军，需要淘汰 n-1 个队伍。每一场比赛淘汰一个队伍，因此进行了 n-1 场比赛。所以共有 n-1 个配对。

代码

```go
package leetcode

// 解法一
func numberOfMatches(n int) int {
  return n - 1
}
```

```go
// 解法二 模拟
func numberOfMatches1(n int) int {
  sum := 0
  for n != 1 {
    if n&1 == 0 {
      sum += n / 2
      n = n / 2
    } else {
      sum += (n - 1) / 2
      n = (n-1)/2 + 1
    }
  }
  return sum
}
```

1689. Partitioning Into Minimum Number Of Deci-Binary Numbers

题目

A decimal number is called **deci-binary** if each of its digits is either `0` or `1` without any leading zeros. For example, `101` and `1100` are **deci-binary**, while `112` and `3001` are not.

Given a string `n` that represents a positive decimal integer, return *the **minimum** number of positive **deci-binary** numbers needed so that they sum up to* `n`.

Example 1:

```
Input: n = "32"
Output: 3
Explanation: 10 + 11 + 11 = 32
```

Example 2:

```
Input: n = "82734"
Output: 8
```

Example 3:

```
Input: n = "27346209830709182346"
Output: 9
```

Constraints:

- `1 <= n.length <= 105`
- `n` consists of only digits.

- `n` does not contain any leading zeros and represents a positive integer.

题目大意

如果一个十进制数字不含任何前导零，且每一位上的数字不是 0 就是 1 ，那么该数字就是一个 十-二进制数 。例如，101 和 1100 都是 十-二进制数，而 112 和 3001 不是。给你一个表示十进制整数的字符串 n ，返回和为 n 的 十-二进制数 的最少数目。

解题思路

- 这一题也算是简单题，相通了以后，代码就 3 行。
- 要想由 01 组成的十进制数组成 n，只需要在 n 这个数的各个数位上依次排上 0 和 1 即可。例如 n = 23423723，这是一个 8 位数。最大数字是 7，所以至少需要 7 个数累加能得到这个 n。这 7 个数的百位都为 1，其他数位按需求取 0 和 1 即可。例如万位是 2，那么这 7 个数中任找 2 个数的万位是 1 ，其他 5 个数的万位是 0 即可。

代码

```go
package leetcode

func minPartitions(n string) int {
  res := 0
  for i := 0; i < len(n); i++ {
    if int(n[i]-'0') > res {
      res = int(n[i] - '0')
    }
  }
  return res
}
```

1690. Stone Game VII

题目

Alice and Bob take turns playing a game, with **Alice starting first**.

There are `n` stones arranged in a row. On each player's turn, they can **remove** either the leftmost stone or the rightmost stone from the row and receive points equal to the **sum** of the remaining stones' values in the row. The winner is the one with the higher score when there are no stones left to remove.

Bob found that he will always lose this game (poor Bob, he always loses), so he decided to **minimize the score's difference**. Alice's goal is to **maximize the difference** in the score.

Given an array of integers `stones` where `stones[i]` represents the value of the `ith` stone **from the left**, return *the **difference** in Alice and Bob's score if they both play **optimally**.*

Example 1:

```
Input: stones = [5,3,1,4,2]
Output: 6
Explanation:
- Alice removes 2 and gets 5 + 3 + 1 + 4 = 13 points. Alice = 13, Bob = 0, stones =
[5,3,1,4].
- Bob removes 5 and gets 3 + 1 + 4 = 8 points. Alice = 13, Bob = 8, stones = [3,1,4].
- Alice removes 3 and gets 1 + 4 = 5 points. Alice = 18, Bob = 8, stones = [1,4].
- Bob removes 1 and gets 4 points. Alice = 18, Bob = 12, stones = [4].
- Alice removes 4 and gets 0 points. Alice = 18, Bob = 12, stones = [].
The score difference is 18 - 12 = 6.
```

Example 2:

```
Input: stones = [7,90,5,1,100,10,10,2]
Output: 122
```

Constraints:

- `n == stones.length`
- `2 <= n <= 1000`
- `1 <= stones[i] <= 1000`

题目大意

石子游戏中，爱丽丝和鲍勃轮流进行自己的回合，爱丽丝先开始 。有 n 块石子排成一排。每个玩家的回合中，可以从行中 移除 最左边的石头或最右边的石头，并获得与该行中剩余石头值之 和 相等的得分。当没有石头可移除时，得分较高者获胜。鲍勃发现他总是输掉游戏（可怜的鲍勃，他总是输），所以他决定尽力 减小得分的差值 。爱丽丝的目标是最大限度地 扩大得分的差值 。

给你一个整数数组 stones ，其中 stones[i] 表示 从左边开始 的第 i 个石头的值，如果爱丽丝和鲍勃都 发挥出最佳水平 ，请返回他们 得分的差值 。

解题思路

- 首先考虑 Bob 缩小分值差距意味着什么，意味着他想让他和 Alice 相对分数最小。Bob 已经明确肯定是输，所以他的分数一定比 Alice 小，那么 Bob - Alice 分数相减一定是负数。相对分数越小，意味着差值越大。负数越大，差值越小。-50 和 -10，-10 数值大，相差小。所以 Bob 的操作是让相对分数越大。Alice 的目的也是这样，要让 Alice - Bob 的相对分数越大，这里是正数的越大。综上，两者的目的相同，都是让相对分数最大化。
- 定义 `dp[i][j]` 代表在当前 `stone[i ~ j]` 区间内能获得的最大分差。状态转移方程为：

  ```
  dp[i][j] = max(
                  sum(i + 1, j) - dp[i + 1][j],   // 这一局取走 `stone[i]`，获得 sum(i
  + 1, j) 分数，再减去剩下对手能获得的分数，即是此局能获得的最大分差。
                  sum(i, j - 1) - dp[i][j - 1]    // 这一局取走 `stone[j]`，获得 sum(i,
  j - 1) 分数，再减去剩下对手能获得的分数，即是此局能获得的最大分差。
                )
  ```

计算 `sum(i + 1, j) = stone[i + 1] + stone[i + 2] + …… + stone[j]` 利用前缀和计算区间和。

- 解法二是正常思路解答出来的代码。解法一是压缩了 DP 数组，在 DP 状态转移的时候，生成下一个 `dp[j]` 实际上是有规律的。利用这个规律可以少存一维数据，压缩空间。解法一的代码直接写出来，比较难想。先写出解法二的代码，然后找到递推规律，优化空间压缩一维，再写出解法一的代码。

代码

```
package leetcode

// 解法一 优化空间版 DP
func stoneGameVII(stones []int) int {
  n := len(stones)
  sum := make([]int, n)
  dp := make([]int, n)
  for i, d := range stones {
    sum[i] = d
  }
  for i := 1; i < n; i++ {
    for j := 0; j+i < n; j++ {
      if (n-i)%2 == 1 {
        d0 := dp[j] + sum[j]
        d1 := dp[j+1] + sum[j+1]
        if d0 > d1 {
          dp[j] = d0
        } else {
          dp[j] = d1
        }
      } else {
        d0 := dp[j] - sum[j]
        d1 := dp[j+1] - sum[j+1]
        if d0 < d1 {
          dp[j] = d0
        } else {
          dp[j] = d1
        }
      }
      sum[j] = sum[j] + stones[i+j]
    }
  }
  return dp[0]
}

// 解法二 常规 DP
func stoneGameVII1(stones []int) int {
  prefixSum := make([]int, len(stones))
  for i := 0; i < len(stones); i++ {
    if i == 0 {
      prefixSum[i] = stones[i]
```

```
    } else {
      prefixSum[i] = prefixSum[i-1] + stones[i]
    }
  }
  dp := make([][]int, len(stones))
  for i := range dp {
    dp[i] = make([]int, len(stones))
    dp[i][i] = 0
  }
  n := len(stones)
  for l := 2; l <= n; l++ {
    for i := 0; i+l <= n; i++ {
      dp[i][i+l-1] = max(prefixSum[i+l-1]-prefixSum[i+1]+stones[i+1]-dp[i+1][i+l-1],
prefixSum[i+l-2]-prefixSum[i]+stones[i]-dp[i][i+l-2])
    }
  }
  return dp[0][n-1]
}

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

1691. Maximum Height by Stacking Cuboids

题目

Given `n` `cuboids` where the dimensions of the `ith` cuboid is `cuboids[i] = [widthi, lengthi, heighti]` (**0-indexed**). Choose a **subset** of `cuboids` and place them on each other.

You can place cuboid `i` on cuboid `j` if `widthi <= widthj` and `lengthi <= lengthj` and `heighti <= heightj`. You can rearrange any cuboid's dimensions by rotating it to put it on another cuboid.

Return *the **maximum height** of the stacked* `cuboids`.

Example 1:

```
Input: cuboids = [[50,45,20],[95,37,53],[45,23,12]]
Output: 190
Explanation:
Cuboid 1 is placed on the bottom with the 53x37 side facing down with height 95.
Cuboid 0 is placed next with the 45x20 side facing down with height 50.
Cuboid 2 is placed next with the 23x12 side facing down with height 45.
The total height is 95 + 50 + 45 = 190.
```

Example 2:

```
Input: cuboids = [[38,25,45],[76,35,3]]
Output: 76
Explanation:
You can't place any of the cuboids on the other.
We choose cuboid 1 and rotate it so that the 35x3 side is facing down and its height is 76.
```

Example 3:

```
Input: cuboids = [[7,11,17],[7,17,11],[11,7,17],[11,17,7],[17,7,11],[17,11,7]]
Output: 102
Explanation:
After rearranging the cuboids, you can see that all cuboids have the same dimension.
You can place the 11x7 side down on all cuboids so their heights are 17.
The maximum height of stacked cuboids is 6 * 17 = 102.
```

Constraints:

- `n == cuboids.length`
- `1 <= n <= 100`
- `1 <= widthi, lengthi, heighti <= 100`

题目大意

给你 n 个长方体 cuboids ， 其中第 i 个长方体的长宽高表示为 cuboids[i] = [widthi, lengthi, heighti]（下标从 0 开始）。请你从 cuboids 选出一个 子集 ， 并将它们堆叠起来。如果 widthi <= widthj 且 lengthi <= lengthj 且 heighti <= heightj ， 你就可以将长方体 i 堆叠在长方体 j 上。你可以通过旋转把长方体的长宽高重新排列，以将它放在另一个长方体上。返回 堆叠长方体 cuboids 可以得到的 最大高度 。

解题思路

- 这一题是 LIS 最长递增子序列系列问题的延续。一维 LIS 问题是第 300 题。二维 LIS 问题是 354 题。这一题是三维的 LIS 问题。
- 题目要求最终摞起来的长方体尽可能的高，那么把长宽高中最大的值旋转为高。这是针对单个方块的排序。多个方块间还要排序，因为他们摞起来有要求，大的方块必须放在下面。所以针对多个方块，按照长，宽，高的顺序进行排序。两次排序完成以后，可以用动态规划找出最大值了。定义 `dp[i]` 为以 `i` 为最后一块砖块所能堆叠的最高高度。由于长和宽已经排好序了。所以只需要在 [0, i - 1] 这个区间内动态更新 dp 最大值。

代码

```go
package leetcode

import "sort"

func maxHeight(cuboids [][]int) int {
```

```go
  n := len(cuboids)
  for i := 0; i < n; i++ {
    sort.Ints(cuboids[i]) // 立方体三边内部排序
  }
  // 立方体排序，先按最短边，再到最长边
  sort.Slice(cuboids, func(i, j int) bool {
    if cuboids[i][0] != cuboids[j][0] {
      return cuboids[i][0] < cuboids[j][0]
    }
    if cuboids[i][1] != cuboids[j][1] {
      return cuboids[i][1] < cuboids[j][1]
    }
    return cuboids[i][2] < cuboids[j][2]
  })
  res := 0
  dp := make([]int, n)
  for i := 0; i < n; i++ {
    dp[i] = cuboids[i][2]
    res = max(res, dp[i])
  }
  for i := 1; i < n; i++ {
    for j := 0; j < i; j++ {
      if cuboids[j][0] <= cuboids[i][0] && cuboids[j][1] <= cuboids[i][1] && cuboids[j]
[2] <= cuboids[i][2] {
        dp[i] = max(dp[i], dp[j]+cuboids[i][2])
      }
    }
    res = max(res, dp[i])
  }
  return res
}

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

1694. Reformat Phone Number

题目

You are given a phone number as a string `number`. `number` consists of digits, spaces `' '`, and/or dashes `'-'`.

You would like to reformat the phone number in a certain manner. Firstly, **remove** all spaces and dashes. Then, **group** the digits from left to right into blocks of length 3 **until** there are 4 or fewer digits. The final digits are then grouped as follows:

- 2 digits: A single block of length 2.
- 3 digits: A single block of length 3.
- 4 digits: Two blocks of length 2 each.

The blocks are then joined by dashes. Notice that the reformatting process should **never** produce any blocks of length 1 and produce **at most** two blocks of length 2.

Return *the phone number after formatting.*

Example 1:

```
Input: number = "1-23-45 6"
Output: "123-456"
Explanation: The digits are "123456".
Step 1: There are more than 4 digits, so group the next 3 digits. The 1st block is
"123".
Step 2: There are 3 digits remaining, so put them in a single block of length 3. The
2nd block is "456".
Joining the blocks gives "123-456".
```

Example 2:

```
Input: number = "123 4-567"
Output: "123-45-67"
Explanation: The digits are "1234567".
Step 1: There are more than 4 digits, so group the next 3 digits. The 1st block is
"123".
Step 2: There are 4 digits left, so split them into two blocks of length 2. The blocks
are "45" and "67".
Joining the blocks gives "123-45-67".
```

Example 3:

```
Input: number = "123 4-5678"
Output: "123-456-78"
Explanation: The digits are "12345678".
Step 1: The 1st block is "123".
Step 2: The 2nd block is "456".
Step 3: There are 2 digits left, so put them in a single block of length 2. The 3rd
block is "78".
Joining the blocks gives "123-456-78".
```

Example 4:

```
Input: number = "12"
Output: "12"
```

Example 5:

```
Input: number = "--17-5 229 35-39475 "
Output: "175-229-353-94-75"
```

Constraints:

- `2 <= number.length <= 100`
- `number` consists of digits and the characters `'-'` and `' '`.
- There are at least **two** digits in `number`.

题目大意

给你一个字符串形式的电话号码 number 。number 由数字、空格 ' '、和破折号 '-' 组成。

请你按下述方式重新格式化电话号码。

- 首先，删除 所有的空格和破折号。
- 其次，将数组从左到右 每 3 个一组 分块，直到 剩下 4 个或更少数字。剩下的数字将按下述规定再分块：
  - 2 个数字：单个含 2 个数字的块。
  - 3 个数字：单个含 3 个数字的块。
  - 4 个数字：两个分别含 2 个数字的块。

最后用破折号将这些块连接起来。注意，重新格式化过程中 不应该 生成仅含 1 个数字的块，并且 最多 生成两个含 2 个数字的块。返回格式化后的电话号码。

解题思路

- 简单题。先判断号码是不是 2 位和 4 位，如果是，单独输出这 2 种情况。剩下的都是 3 位以上了，取余，判断剩余的数字是 2 个还是 4 个。这时不可能存在剩 1 位数的情况。除 3 余 1，即剩 4 位的情况，末尾 4 位需要 2 个一组输出。除 3 余 2，即剩 2 位的情况。处理好末尾，再逆序每 3 个一组连接 "-" 即可。可能需要注意的 case 见 test 文件。

代码

```go
package leetcode

import (
  "strings"
)
```

```go
func reformatNumber(number string) string {
  parts, nums := []string{}, []rune{}
  for _, r := range number {
    if r != '-' && r != ' ' {
      nums = append(nums, r)
    }
  }
  threeDigits, twoDigits := len(nums)/3, 0
  switch len(nums) % 3 {
  case 1:
    threeDigits--
    twoDigits = 2
  case 2:
    twoDigits = 1
  default:
    twoDigits = 0
  }
  for i := 0; i < threeDigits; i++ {
    s := ""
    s += string(nums[0:3])
    nums = nums[3:]
    parts = append(parts, s)
  }
  for i := 0; i < twoDigits; i++ {
    s := ""
    s += string(nums[0:2])
    nums = nums[2:]
    parts = append(parts, s)
  }
  return strings.Join(parts, "-")
}
```

1695. Maximum Erasure Value

题目

You are given an array of positive integers `nums` and want to erase a subarray containing **unique elements**. The **score** you get by erasing the subarray is equal to the **sum** of its elements.

Return *the **maximum score** you can get by erasing **exactly one** subarray.*

An array `b` is called to be a subarray of `a` if it forms a contiguous subsequence of `a`, that is, if it is equal to `a[l],a[l+1],...,a[r]` for some `(l,r)`.

Example 1:

```
Input: nums = [4,2,4,5,6]
Output: 17
Explanation: The optimal subarray here is [2,4,5,6].
```

Example 2:

```
Input: nums = [5,2,1,2,5,2,1,2,5]
Output: 8
Explanation: The optimal subarray here is [5,2,1] or [1,2,5].
```

Constraints:

- `1 <= nums.length <= 105`
- `1 <= nums[i] <= 104`

题目大意

给你一个正整数数组 nums ， 请你从中删除一个含有 若干不同元素 的子数组。删除子数组的 得分 就是子数组各元素之 和 。返回 只删除一个 子数组可获得的 最大得分 。如果数组 b 是数组 a 的一个连续子序列，即如果它等于 a[l],a[l+1],...,a[r] ， 那么它就是 a 的一个子数组。

解题思路

- 读完题立马能识别出这是经典的滑动窗口题。利用滑动窗口从左往右滑动窗口，滑动过程中统计频次，如果是不同元素，右边界窗口又移，否则左边窗口缩小。每次移动更新 max 值。最终扫完一遍以后，max 值即为所求。

代码

```go
package leetcode

func maximumUniqueSubarray(nums []int) int {
  if len(nums) == 0 {
    return 0
  }
  result, left, right, freq := 0, 0, -1, map[int]int{}
  for left < len(nums) {
    if right+1 < len(nums) && freq[nums[right+1]] == 0 {
      freq[nums[right+1]]++
      right++
    } else {
      freq[nums[left]]--
      left++
    }
    sum := 0
    for i := left; i <= right; i++ {
      sum += nums[i]
    }
```

```
    result = max(result, sum)
  }
  return result
}

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

1696. Jump Game VI

题目

You are given a **0-indexed** integer array `nums` and an integer `k`.

You are initially standing at index `0`. In one move, you can jump at most `k` steps forward without going outside the boundaries of the array. That is, you can jump from index `i` to any index in the range `[i + 1, min(n - 1, i + k)]` **inclusive**.

You want to reach the last index of the array (index `n - 1`). Your **score** is the **sum** of all `nums[j]` for each index `j` you visited in the array.

Return *the **maximum score** you can get*.

Example 1:

```
Input: nums = [1,-1,-2,4,-7,3], k = 2
Output: 7
Explanation: You can choose your jumps forming the subsequence [1,-1,4,3] (underlined above). The sum is 7.
```

Example 2:

```
Input: nums = [10,-5,-2,4,0,3], k = 3
Output: 17
Explanation: You can choose your jumps forming the subsequence [10,4,3] (underlined above). The sum is 17.
```

Example 3:

```
Input: nums = [1,-5,-20,4,-1,3,-6,-3], k = 2
Output: 0
```

Constraints:

- `1 <= nums.length, k <= 10^5`
- `10^4 <= nums[i] <= 10^4`

题目大意

给你一个下标从 0 开始的整数数组 nums 和一个整数 k 。一开始你在下标 0 处。每一步，你最多可以往前跳 k 步，但你不能跳出数组的边界。也就是说，你可以从下标 i 跳到 [i + 1， min(n - 1, i + k)] 包含 两个端点的任意位置。你的目标是到达数组最后一个位置（下标为 n - 1 ），你的 得分 为经过的所有数字之和。请你返回你能得到的 最大得分 。

解题思路

- 首先能想到的解题思路是动态规划。定义 `dp[i]` 为跳到第 `i` 个位子能获得的最大分数。题目要求的是 `dp[n-1]`，状态转移方程是：`dp[i] = nums[i] + max(dp[j]), max(0, i - k ) <= j < i`，这里需要注意 `j` 的下界，题目中说到不能跳到负数区间，所以左边界下界为 0 。求 `max(dp[j])` 需要遍历一次求得最大值，所以这个解法整体时间复杂度是 O((n - k) * k)，但是提交以后提示超时了。
- 分析一下超时原因。每次都要在 `[max(0, i - k ), i)` 区间内扫描找到最大值，下一轮的区间是 `[max(0, i - k + 1), i + 1)`，前后这两轮扫描的区间存在大量重合部分 `[max(0, i - k + 1), i)`，正是这部分反反复复的扫描导致算法低效。如何高效的在一个区间内找到最大值是本题的关键。利用单调队列可以完成此题。单调队列里面存一个区间内最大值的下标。这里单调队列有 2 个性质。性质一，队列的队首永远都是最大值，队列从大到小降序排列。如果来了一个比队首更大的值的下标，需要将单调队列清空，只存这个新的最大值的下标。性质二，队列的长度为 k。从队尾插入新值，并把队头的最大值“挤”出队首。拥有了这个单调队列以后，再进行 DP 状态转移，效率就很高了。每次只需取出队首的最大值即可。具体代码见下面。

代码

```go
package leetcode

import (
  "math"
)

// 单调队列
func maxResult(nums []int, k int) int {
  dp := make([]int, len(nums))
  dp[0] = nums[0]
  for i := 1; i < len(dp); i++ {
    dp[i] = math.MinInt32
  }
  window := make([]int, k)
  for i := 1; i < len(nums); i++ {
    dp[i] = nums[i] + dp[window[0]]
    for len(window) > 0 && dp[window[len(window)-1]] <= dp[i] {
      window = window[:len(window)-1]
    }
    for len(window) > 0 && i-k >= window[0] {
```

```go
      window = window[1:]
    }
    window = append(window, i)
  }
  return dp[len(nums)-1]
}

// 超时
func maxResult1(nums []int, k int) int {
  dp := make([]int, len(nums))
  if k > len(nums) {
    k = len(nums)
  }
  dp[0] = nums[0]
  for i := 1; i < len(dp); i++ {
    dp[i] = math.MinInt32
  }
  for i := 1; i < len(nums); i++ {
    left, tmp := max(0, i-k), math.MinInt32
    for j := left; j < i; j++ {
      tmp = max(tmp, dp[j])
    }
    dp[i] = nums[i] + tmp
  }
  return dp[len(nums)-1]
}

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

1700. Number of Students Unable to Eat Lunch

题目

The school cafeteria offers circular and square sandwiches at lunch break, referred to by numbers `0` and `1` respectively. All students stand in a queue. Each student either prefers square or circular sandwiches.

The number of sandwiches in the cafeteria is equal to the number of students. The sandwiches are placed in a **stack**. At each step:

- If the student at the front of the queue **prefers** the sandwich on the top of the stack, they will **take it** and leave the queue.

- Otherwise, they will **leave it** and go to the queue's end.

This continues until none of the queue students want to take the top sandwich and are thus unable to eat.

You are given two integer arrays `students` and `sandwiches` where `sandwiches[i]` is the type of the `ith` sandwich in the stack (`i = 0` is the top of the stack) and `students[j]` is the preference of the `jth` student in the initial queue (`j = 0` is the front of the queue). Return *the number of students that are unable to eat.*

Example 1:

```
Input: students = [1,1,0,0], sandwiches = [0,1,0,1]
Output: 0
Explanation:
- Front student leaves the top sandwich and returns to the end of the line making
students = [1,0,0,1].
- Front student leaves the top sandwich and returns to the end of the line making
students = [0,0,1,1].
- Front student takes the top sandwich and leaves the line making students = [0,1,1]
and sandwiches = [1,0,1].
- Front student leaves the top sandwich and returns to the end of the line making
students = [1,1,0].
- Front student takes the top sandwich and leaves the line making students = [1,0] and
sandwiches = [0,1].
- Front student leaves the top sandwich and returns to the end of the line making
students = [0,1].
- Front student takes the top sandwich and leaves the line making students = [1] and
sandwiches = [1].
- Front student takes the top sandwich and leaves the line making students = [] and
sandwiches = [].
Hence all students are able to eat.
```

Example 2:

```
Input: students = [1,1,1,0,0,1], sandwiches = [1,0,0,0,1,1]
Output: 3
```

Constraints:

- `1 <= students.length, sandwiches.length <= 100`
- `students.length == sandwiches.length`
- `sandwiches[i]` is `0` or `1`.
- `students[i]` is `0` or `1`.

题目大意

学校的自助午餐提供圆形和方形的三明治，分别用数字 0 和 1 表示。所有学生站在一个队列里，每个学生要么喜欢圆形的要么喜欢方形的。
餐厅里三明治的数量与学生的数量相同。所有三明治都放在一个 栈 里，每一轮：

- 如果队列最前面的学生 喜欢 栈顶的三明治，那么会 拿走它 并离开队列。
- 否则，这名学生会 放弃这个三明治 并回到队列的尾部。
  这个过程会一直持续到队列里所有学生都不喜欢栈顶的三明治为止。

给你两个整数数组 students 和 sandwiches ，其中 sandwiches[i] 是栈里面第 i 个三明治的类型（i = 0 是栈的顶部），students[j] 是初始队列里第 j 名学生对三明治的喜好（j = 0 是队列的最开始位置）。请你返回无法吃午餐的学生数量。

解题思路

- 简单题。按照题意，学生不管怎么轮流领三明治，如果数量够，经过多轮循环，总能领到。问题可以等价为，学生依次到队列前面领取三明治。2 个种类的三明治都摆好放在那里了。最终领不到三明治的学生都是因为喜欢的三明治不够发放了。按照这个思路，先统计 2 种三明治的总个数，然后减去学生对三明治的需求总数，剩下的学生即都是无法满足的。

代码

```go
package leetcode

func countStudents(students []int, sandwiches []int) int {
  tmp, n, i := [2]int{}, len(students), 0
  for _, v := range students {
    tmp[v]++
  }
  for i < n && tmp[sandwiches[i]] > 0 {
    tmp[sandwiches[i]]--
    i++
  }
  return n - i
}
```

1704. Determine if String Halves Are Alike

题目

You are given a string `s` of even length. Split this string into two halves of equal lengths, and let `a` be the first half and `b` be the second half.

Two strings are **alike** if they have the same number of vowels (`'a'`, `'e'`, `'i'`, `'o'`, `'u'`, `'A'`, `'E'`, `'I'`, `'O'`, `'U'`). Notice that `s` contains uppercase and lowercase letters.

Return `true` *if* `a` *and* `b` *are* ***alike***. Otherwise, return `false`.

Example 1:

```
Input: s = "book"
Output: true
Explanation: a = "bo" and b = "ok". a has 1 vowel and b has 1 vowel. Therefore, they are alike.
```

Example 2:

```
Input: s = "textbook"
Output: false
Explanation: a = "text" and b = "book". a has 1 vowel whereas b has 2. Therefore, they are not alike.
Notice that the vowel o is counted twice.
```

Example 3:

```
Input: s = "MerryChristmas"
Output: false
```

Example 4:

```
Input: s = "AbCdEfGh"
Output: true
```

Constraints:

- `2 <= s.length <= 1000`
- `s.length` is even.
- `s` consists of **uppercase and lowercase** letters.

题目大意

给你一个偶数长度的字符串 s 。将其拆分成长度相同的两半，前一半为 a ，后一半为 b 。两个字符串 相似 的前提是它们都含有相同数目的元音（'a'，'e'，'i'，'o'，'u'，'A'，'E'，'I'，'O'，'U'）。注意，s 可能同时含有大写和小写字母。如果 a 和 b 相似，返回 true ；否则，返回 false 。

解题思路

- 简单题。依题意，分别统计前半段元音字母的个数和后半段元音字母的个数，个数相同则输出 true，不同就输出 false。

代码

```go
package leetcode

func halvesAreAlike(s string) bool {
  return numVowels(s[len(s)/2:]) == numVowels(s[:len(s)/2])
}
```

```go
func numVowels(x string) int {
  res := 0
  for _, c := range x {
    switch c {
    case 'a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U':
      res++
    }
  }
  return res
}
```

1710. Maximum Units on a Truck

题目

You are assigned to put some amount of boxes onto **one truck**. You are given a 2D array `boxTypes`, where `boxTypes[i] = [numberOfBoxesi, numberOfUnitsPerBoxi]`:

- `numberOfBoxesi` is the number of boxes of type `i`.
- `numberOfUnitsPerBoxi` is the number of units in each box of the type `i`.

You are also given an integer `truckSize`, which is the **maximum** number of **boxes** that can be put on the truck. You can choose any boxes to put on the truck as long as the number of boxes does not exceed `truckSize`.

Return *the **maximum** total number of **units** that can be put on the truck.*

Example 1:

```
Input: boxTypes = [[1,3],[2,2],[3,1]], truckSize = 4
Output: 8
Explanation: There are:
- 1 box of the first type that contains 3 units.
- 2 boxes of the second type that contain 2 units each.
- 3 boxes of the third type that contain 1 unit each.
You can take all the boxes of the first and second types, and one box of the third type.
The total number of units will be = (1 * 3) + (2 * 2) + (1 * 1) = 8.
```

Example 2:

```
Input: boxTypes = [[5,10],[2,5],[4,7],[3,9]], truckSize = 10
Output: 91
```

Constraints:

- `1 <= boxTypes.length <= 1000`

- `1 <= numberOfBoxesi, numberOfUnitsPerBoxi <= 1000`
- `1 <= truckSize <= 106`

题目大意

请你将一些箱子装在 一辆卡车 上。给你一个二维数组 boxTypes ， 其中 boxTypes[i] = [numberOfBoxesi, numberOfUnitsPerBoxi]：

- numberOfBoxesi 是类型 i 的箱子的数量。-
- numberOfUnitsPerBoxi 是类型 i 每个箱子可以装载的单元数量。

整数 truckSize 表示卡车上可以装载 箱子 的 最大数量 。只要箱子数量不超过 truckSize ， 你就可以选择任意箱子装到卡车上。返回卡车可以装载 单元 的 最大 总数。

解题思路

- 简单题。先将箱子按照单元数量从大到小排序。要想卡车装载单元数最大，那么需要尽量装单元数多的箱子。所以排序以后从单元数量多的箱子开始取。一直取至 truckSize 没有空间。累积的单元数即最大总数。

代码

```go
package leetcode

import "sort"

func maximumUnits(boxTypes [][]int, truckSize int) int {
  sort.Slice(boxTypes, func(i, j int) bool {
    return boxTypes[i][1] > boxTypes[j][1]
  })
  res := 0
  for i := 0; truckSize > 0 && i < len(boxTypes); i++ {
    if truckSize >= boxTypes[i][0] {
      truckSize -= boxTypes[i][0]
      res += (boxTypes[i][1] * boxTypes[i][0])
    } else {
      res += (truckSize * boxTypes[i][1])
      truckSize = 0
    }
  }
  return res
}
```

1716. Calculate Money in Leetcode Bank

题目

Hercy wants to save money for his first car. He puts money in the Leetcode bank **every day**.

He starts by putting in `$1` on Monday, the first day. Every day from Tuesday to Sunday, he will put in `$1` more than the day before. On every subsequent Monday, he will put in `$1` more than the **previous Monday**.

Given `n`, return *the total amount of money he will have in the Leetcode bank at the end of the* `nth` *day.*

Example 1:

```
Input: n = 4
Output: 10
Explanation: After the 4th day, the total is 1 + 2 + 3 + 4 = 10.
```

Example 2:

```
Input: n = 10
Output: 37
Explanation: After the 10th day, the total is (1 + 2 + 3 + 4 + 5 + 6 + 7) + (2 + 3 + 4)
= 37. Notice that on the 2nd Monday, Hercy only puts in $2.
```

Example 3:

```
Input: n = 20
Output: 96
Explanation: After the 20th day, the total is (1 + 2 + 3 + 4 + 5 + 6 + 7) + (2 + 3 + 4
+ 5 + 6 + 7 + 8) + (3 + 4 + 5 + 6 + 7 + 8) = 96.
```

Constraints:

- `1 <= n <= 1000`

题目大意

Hercy 想要为购买第一辆车存钱。他 每天 都往力扣银行里存钱。最开始，他在周一的时候存入 1 块钱。从周二到周日，他每天都比前一天多存入 1 块钱。在接下来每一个周一，他都会比 前一个周一 多存入 1 块钱。给你 n ，请你返回在第 n 天结束的时候他在力扣银行总共存了多少块钱。

解题思路

- 简单题。按照题意写 2 层循环即可。

代码

```go
package leetcode

func totalMoney(n int) int {
  res := 0
  for tmp, count := 1, 7; n > 0; tmp, count = tmp+1, 7 {
    for m := tmp; n > 0 && count > 0; m++ {
      res += m
      n--
      count--
    }
  }
  return res
}
```

1720. Decode XORed Array

题目

There is a **hidden** integer array `arr` that consists of `n` non-negative integers.

It was encoded into another integer array `encoded` of length `n - 1`, such that `encoded[i] = arr[i] XOR arr[i + 1]`. For example, if `arr = [1,0,2,1]`, then `encoded = [1,2,3]`.

You are given the `encoded` array. You are also given an integer `first`, that is the first element of `arr`, i.e. `arr[0]`.

Return *the original array* `arr`. It can be proved that the answer exists and is unique.

Example 1:

```
Input: encoded = [1,2,3], first = 1
Output: [1,0,2,1]
Explanation: If arr = [1,0,2,1], then first = 1 and encoded = [1 XOR 0, 0 XOR 2, 2 XOR 1] = [1,2,3]
```

Example 2:

```
Input: encoded = [6,2,7,3], first = 4
Output: [4,2,0,7,4]
```

Constraints:

- `2 <= n <= 104`
- `encoded.length == n - 1`
- `0 <= encoded[i] <= 105`
- `0 <= first <= 10^5`

题目大意

未知 整数数组 arr 由 n 个非负整数组成。经编码后变为长度为 n - 1 的另一个整数数组 encoded ，其中 encoded[i] = arr[i] XOR arr[i + 1] 。例如，arr = [1,0,2,1] 经编码后得到 encoded = [1,2,3] 。给你编码后的数组 encoded 和原数组 arr 的第一个元素 first（arr[0]）。请解码返回原数组 arr 。可以证明答案存在并且是唯一的。

解题思路

- 简单题。按照题意，求返回解码以后的原数组，即将编码后的数组前后两两元素依次做异或 `XOR` 运算。

代码

```go
package leetcode

func decode(encoded []int, first int) []int {
  arr := make([]int, len(encoded)+1)
  arr[0] = first
  for i, val := range encoded {
    arr[i+1] = arr[i] ^ val
  }
  return arr
}
```

1721. Swapping Nodes in a Linked List

题目

You are given the `head` of a linked list, and an integer `k`.

Return *the head of the linked list after **swapping** the values of the `kth` node from the beginning and the `kth` node from the end (the list is **1-indexed**).*

Example 1:

```
Input: head = [1,2,3,4,5], k = 2
Output: [1,4,3,2,5]
```

Example 2:

```
Input: head = [7,9,6,6,7,8,3,0,9,5], k = 5
Output: [7,9,6,6,8,7,3,0,9,5]
```

Example 3:

```
Input: head = [1], k = 1
Output: [1]
```

Example 4:

```
Input: head = [1,2], k = 1
Output: [2,1]
```

Example 5:

```
Input: head = [1,2,3], k = 2
Output: [1,2,3]
```

Constraints:

- The number of nodes in the list is `n`.
- `1 <= k <= n <= 10^5`
- `0 <= Node.val <= 100`

题目大意

给你链表的头节点 `head` 和一个整数 `k` 。**交换** 链表正数第 `k` 个节点和倒数第 `k` 个节点的值后，返回链表的头节点（链表 **从 1 开始索引**）。

解题思路

- 这道题虽然是 medium，但是实际非常简单。题目要求链表中 2 个节点的值，无非是先找到这 2 个节点，然后再交换即可。链表查询节点需要 O(n)，2 次循环找到对应的 2 个节点，交换值即可。

代码

```go
package leetcode

import (
  "github.com/halfrost/LeetCode-Go/structures"
)

// ListNode define
type ListNode = structures.ListNode

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */
func swapNodes(head *ListNode, k int) *ListNode {
  count := 1
  var a, b *ListNode
  for node := head; node != nil; node = node.Next {
```

```
    if count == k {
      a = node
    }
    count++
  }
  length := count
  count = 1
  for node := head; node != nil; node = node.Next {
    if count == length-k {
      b = node
    }
    count++
  }
  a.Val, b.Val = b.Val, a.Val
  return head
}
```

1725. Number Of Rectangles That Can Form The Largest Square

题目

You are given an array `rectangles` where `rectangles[i] = [li, wi]` represents the `ith` rectangle of length `li` and width `wi`.

You can cut the `ith` rectangle to form a square with a side length of `k` if both `k <= li` and `k <= wi`. For example, if you have a rectangle `[4,6]`, you can cut it to get a square with a side length of at most `4`.

Let `maxLen` be the side length of the **largest** square you can obtain from any of the given rectangles.

Return *the **number** of rectangles that can make a square with a side length of* `maxLen`.

Example 1:

```
Input: rectangles = [[5,8],[3,9],[5,12],[16,5]]
Output: 3
Explanation: The largest squares you can get from each rectangle are of lengths
[5,3,5,5].
The largest possible square is of length 5, and you can get it out of 3 rectangles.
```

Example 2:

```
Input: rectangles = [[2,3],[3,7],[4,3],[3,7]]
Output: 3
```

Constraints:

- `1 <= rectangles.length <= 1000`

- `rectangles[i].length == 2`
- `1 <= li, wi <= 10^9`
- `li != wi`

题目大意

给你一个数组 rectangles ，其中 rectangles[i] = [li, wi] 表示第 i 个矩形的长度为 li 、宽度为 wi 。如果存在 k 同时满足 k <= li 和 k <= wi ，就可以将第 i 个矩形切成边长为 k 的正方形。例如，矩形 [4,6] 可以切成边长最大为 4 的正方形。设 maxLen 为可以从矩形数组 rectangles 切分得到的 最大正方形 的边长。返回可以切出边长为 maxLen 的正方形的矩形 数目 。

解题思路

- 简单题。扫描数组中的每一个矩形，先找到边长较小的那条边，作为正方形的边长。扫描过程中动态更新最大的正方形边长，并累加计数。循环一遍结束，输出最终计数值即可。

代码

```go
package leetcode

func countGoodRectangles(rectangles [][]int) int {
  minLength, count := 0, 0
  for i, _ := range rectangles {
    minSide := 0
    if rectangles[i][0] <= rectangles[i][1] {
      minSide = rectangles[i][0]
    } else {
      minSide = rectangles[i][1]
    }
    if minSide > minLength {
      minLength = minSide
      count = 1
    } else if minSide == minLength {
      count++
    }
  }
  return count
}
```

1732. Find the Highest Altitude

题目

There is a biker going on a road trip. The road trip consists of `n + 1` points at different altitudes. The biker starts his trip on point `0` with altitude equal `0`.

You are given an integer array `gain` of length `n` where `gain[i]` is the **net gain in altitude** between points `i` and `i + 1` for all (`0 <= i < n`). Return *the **highest altitude** of a point.*

Example 1:

```
Input: gain = [-5,1,5,0,-7]
Output: 1
Explanation: The altitudes are [0,-5,-4,1,1,-6]. The highest is 1.
```

Example 2:

```
Input: gain = [-4,-3,-2,-1,4,3,2]
Output: 0
Explanation: The altitudes are [0,-4,-7,-9,-10,-6,-3,-1]. The highest is 0.
```

Constraints:

- `n == gain.length`
- `1 <= n <= 100`
- `100 <= gain[i] <= 100`

题目大意

有一个自行车手打算进行一场公路骑行，这条路线总共由 n + 1 个不同海拔的点组成。自行车手从海拔为 0 的点 0 开始骑行。给你一个长度为 n 的整数数组 gain ，其中 gain[i] 是点 i 和点 i + 1 的 净海拔高度差（0 <= i < n）。请你返回 最高点的海拔 。

解题思路

- 简单题。循环数组依次从第一个海拔点开始还原每个海拔点，动态记录最大高度。循环结束输出最大高度即可。

代码

```go
package leetcode

func largestAltitude(gain []int) int {
  max, height := 0, 0
  for _, g := range gain {
    height += g
    if height > max {
      max = height
    }
  }
  return max
}
```

1734. Decode XORed Permutation

题目

There is an integer array `perm` that is a permutation of the first `n` positive integers, where `n` is always **odd**.

It was encoded into another integer array `encoded` of length `n - 1`, such that `encoded[i] = perm[i] XOR perm[i + 1]`. For example, if `perm = [1,3,2]`, then `encoded = [2,1]`.

Given the `encoded` array, return *the original array* `perm`. It is guaranteed that the answer exists and is unique.

Example 1:

```
Input: encoded = [3,1]
Output: [1,2,3]
Explanation: If perm = [1,2,3], then encoded = [1 XOR 2,2 XOR 3] = [3,1]
```

Example 2:

```
Input: encoded = [6,5,4,6]
Output: [2,4,1,5,3]
```

Constraints:

- `3 <= n < 10^5`
- `n` is odd.
- `encoded.length == n - 1`

题目大意

给你一个整数数组 perm ，它是前 n 个正整数的排列，且 n 是个奇数 。它被加密成另一个长度为 n - 1 的整数数组 encoded ，满足 encoded[i] = perm[i] XOR perm[i + 1] 。比方说，如果 perm = [1,3,2] ，那么 encoded = [2,1] 。给你 encoded 数组，请你返回原始数组 perm 。题目保证答案存在且唯一。

解题思路

- 这一题与第 136 题和第 137 题思路类似，借用 `x ^ x = 0` 这个性质解题。依题意，原数组 perm 是 n 个正整数，即取值在 `[1,n+1]` 区间内，但是排列顺序未知。可以考虑先将 `[1,n+1]` 区间内的所有数异或得到 total。再将 encoded 数组中奇数下标的元素异或得到 odd：

  {{< katex display >}}
  \begin{aligned}odd &= encoded[1] + encoded[3] + ... + encoded[n-1]\&= (perm[1] \,\, XOR \,\, perm[2]) + (perm[3] \,\, XOR \,\, perm[4]) + ... + (perm[n-1] \,\, XOR \,\, perm[n])\end{aligned}
  {{< /katex >}}

total 是 n 个正整数异或全集，odd 是 `n-1` 个正整数异或集。两者异或 `total ^ odd` 得到的值必定是 perm[0]，因为 `x ^ x = 0`，那么重复出现的元素被异或以后消失了。算出 perm[0] 就好办了。

{{< katex display >}}
\begin{aligned}encoded[0] &= perm[0] \,\, XOR \,\, perm[1]\perm[0] \,\, XOR \,\, encoded[0] &= perm[0] \,\, XOR \,\, perm[0] \,\, XOR \,\, perm[1] = perm[1]\perm[1] \,\, XOR \,\, encoded[1] &= perm[1] \,\, XOR \,\, perm[1] \,\, XOR \,\, perm[2] = perm[2]\...\perm[n-1] \,\, XOR \,\, encoded[n-1] &= perm[n-1] \,\, XOR \,\, perm[n-1] \,\, XOR \,\, perm[n] = perm[n]\\end{aligned}
{{< /katex >}}

依次类推，便可以推出原数组 perm 中的所有数。

代码

```go
package leetcode

func decode(encoded []int) []int {
  n, total, odd := len(encoded), 0, 0
  for i := 1; i <= n+1; i++ {
    total ^= i
  }
  for i := 1; i < n; i += 2 {
    odd ^= encoded[i]
  }
  perm := make([]int, n+1)
  perm[0] = total ^ odd
  for i, v := range encoded {
    perm[i+1] = perm[i] ^ v
  }
  return perm
}
```

1736. Latest Time by Replacing Hidden Digits

题目

You are given a string `time` in the form of `hh:mm`, where some of the digits in the string are hidden (represented by `?`).

The valid times are those inclusively between `00:00` and `23:59`.

Return *the latest valid time you can get from* `time` *by replacing the hidden digits*.

Example 1:

```
Input: time = "2?:?0"
Output: "23:50"
Explanation: The latest hour beginning with the digit '2' is 23 and the latest minute ending with the digit '0' is 50.
```

Example 2:

```
Input: time = "0?:3?"
Output: "09:39"
```

Example 3:

```
Input: time = "1?:22"
Output: "19:22"
```

Constraints:

- `time` is in the format `hh:mm`.
- It is guaranteed that you can produce a valid time from the given string.

题目大意

给你一个字符串 time ，格式为 hh:mm（小时：分钟），其中某几位数字被隐藏（用 ? 表示）。有效的时间为 00:00 到 23:59 之间的所有时间，包括 00:00 和 23:59 。替换 time 中隐藏的数字，返回你可以得到的最晚有效时间。

解题思路

- 简单题。根据题意，需要找到最晚的有效时间。枚举时间 4 个位置即可。如果第 3 个位置是 ? ，那么它最晚时间是 5；如果第 4 个位置是 ? ，那么它最晚时间是 9；如果第 2 个位置是 ? ，那么它最晚时间是 9；如果第 1 个位置是 ? ，根据第 2 个位置判断，如果第 2 个位置是大于 3 的数，那么第一个位置最晚时间是 1，如果第 2 个位置是小于 3 的数那么第一个位置最晚时间是 2 。按照上述规则即可还原最晚时间。

代码

```go
package leetcode

func maximumTime(time string) string {
  timeb := []byte(time)
  if timeb[3] == '?' {
    timeb[3] = '5'
  }
  if timeb[4] == '?' {
    timeb[4] = '9'
  }
  if timeb[0] == '?' {
    if int(timeb[1]-'0') > 3 && int(timeb[1]-'0') < 10 {
      timeb[0] = '1'
    } else {
      timeb[0] = '2'
    }
  }
  if timeb[1] == '?' {
```

```
    timeb[1] = '9'
  }
  if timeb[0] == '2' && timeb[1] == '9' {
    timeb[1] = '3'
  }
  return string(timeb)
}
```

1738. Find Kth Largest XOR Coordinate Value

题目

You are given a 2D `matrix` of size `m x n`, consisting of non-negative integers. You are also given an integer `k`.

The **value** of coordinate `(a, b)` of the matrix is the XOR of all `matrix[i][j]` where `0 <= i <= a < m` and `0 <= j <= b < n` **(0-indexed)**.

Find the `kth` largest value **(1-indexed)** of all the coordinates of `matrix`.

Example 1:

```
Input: matrix = [[5,2],[1,6]], k = 1
Output: 7
Explanation: The value of coordinate (0,1) is 5 XOR 2 = 7, which is the largest value.
```

Example 2:

```
Input: matrix = [[5,2],[1,6]], k = 2
Output: 5
Explanation:The value of coordinate (0,0) is 5 = 5, which is the 2nd largest value.
```

Example 3:

```
Input: matrix = [[5,2],[1,6]], k = 3
Output: 4
Explanation: The value of coordinate (1,0) is 5 XOR 1 = 4, which is the 3rd largest value.
```

Example 4:

```
Input: matrix = [[5,2],[1,6]], k = 4
Output: 0
Explanation: The value of coordinate (1,1) is 5 XOR 2 XOR 1 XOR 6 = 0, which is the 4th largest value.
```

Constraints:

- `m == matrix.length`
- `n == matrix[i].length`
- `1 <= m, n <= 1000`
- `0 <= matrix[i][j] <= 10^6`
- `1 <= k <= m * n`

题目大意

给你一个二维矩阵 matrix 和一个整数 k ，矩阵大小为 m x n 由非负整数组成。矩阵中坐标 (a, b) 的 值 可由对所有满足 0 <= i <= a < m 且 0 <= j <= b < n 的元素 matrix[i][j]（下标从 0 开始计数）执行异或运算得到。请你找出 matrix 的所有坐标中第 k 大的值（k 的值从 1 开始计数）。

解题思路

- 区间异或结果类比于区间二维前缀和。只不过需要注意 x^x = 0 这一性质。举例：

  通过简单推理，可以得出区间二维前缀和 preSum 的递推式。具体代码见解法二。
- 上面的解法中，preSum 用二维数组计算的。能否再优化空间复杂度，降低成 O(n)？答案是可以的。通过观察可以发现。preSum 可以按照一行一行来生成。先生成 preSum 前一行，下一行生成过程中会用到前一行的信息，异或计算以后，可以覆盖原数据(前一行的信息)，对之后的计算没有影响。这个优化空间复杂度的方法和优化 DP 空间复杂度是完全一样的思路和方法。

  具体代码见解法一。
- 计算出了 preSum，还需要考虑如何输出第 k 大的值。有 3 种做法，第一种是排序，第二种是优先队列，第三种是第 215 题中的 O(n) 的 partition 方法。时间复杂度最低的当然是 O(n)。但是经过实际测试，runtime 最优的是排序的方法。所以笔者以下两种方法均采用了排序的方法。

代码

```go
package leetcode

import "sort"

// 解法一 压缩版的前缀和
func kthLargestValue(matrix [][]int, k int) int {
  if len(matrix) == 0 || len(matrix[0]) == 0 {
    return 0
  }
  res, prefixSum := make([]int, 0, len(matrix)*len(matrix[0])), make([]int,
len(matrix[0]))
  for i := range matrix {
    line := 0
    for j, v := range matrix[i] {
      line ^= v
      prefixSum[j] ^= line
      res = append(res, prefixSum[j])
    }
  }
```

```go
  sort.Ints(res)
  return res[len(res)-k]
}

// 解法二 前缀和
func kthLargestValue1(matrix [][]int, k int) int {
  nums, prefixSum := []int{}, make([][]int, len(matrix)+1)
  prefixSum[0] = make([]int, len(matrix[0])+1)
  for i, row := range matrix {
    prefixSum[i+1] = make([]int, len(matrix[0])+1)
    for j, val := range row {
      prefixSum[i+1][j+1] = prefixSum[i+1][j] ^ prefixSum[i][j+1] ^ prefixSum[i][j] ^
val
      nums = append(nums, prefixSum[i+1][j+1])
    }
  }
  sort.Ints(nums)
  return nums[len(nums)-k]
}
```

1742. Maximum Number of Balls in a Box

题目

You are working in a ball factory where you have `n` balls numbered from `lowLimit` up to `highLimit` **inclusive** (i.e., `n == highLimit - lowLimit + 1`), and an infinite number of boxes numbered from `1` to `infinity`.

Your job at this factory is to put each ball in the box with a number equal to the sum of digits of the ball's number. For example, the ball number `321` will be put in the box number `3 + 2 + 1 = 6` and the ball number `10` will be put in the box number `1 + 0 = 1`.

Given two integers `lowLimit` and `highLimit`, return *the number of balls in the box with the most balls.*

Example 1:

```
Input: lowLimit = 1, highLimit = 10
Output: 2
Explanation:
Box Number:  1 2 3 4 5 6 7 8 9 10 11 ...
Ball Count:  2 1 1 1 1 1 1 1 1 0  0  ...
Box 1 has the most number of balls with 2 balls.
```

Example 2:

```
Input: lowLimit = 5, highLimit = 15
Output: 2
Explanation:
Box Number:  1 2 3 4 5 6 7 8 9 10 11 ...
Ball Count:  1 1 1 1 2 2 1 1 1 0  0  ...
Boxes 5 and 6 have the most number of balls with 2 balls in each.
```

Example 3:

```
Input: lowLimit = 19, highLimit = 28
Output: 2
Explanation:
Box Number:  1 2 3 4 5 6 7 8 9 10 11 12 ...
Ball Count:  0 1 1 1 1 1 1 1 1 2  0  0  ...
Box 10 has the most number of balls with 2 balls.
```

Constraints:

- `1 <= lowLimit <= highLimit <= 10^5`

题目大意

你在一家生产小球的玩具厂工作，有 n 个小球，编号从 lowLimit 开始，到 highLimit 结束（包括 lowLimit 和 highLimit ，即 n == highLimit - lowLimit + 1）。另有无限数量的盒子，编号从 1 到 infinity 。你的工作是将每个小球放入盒子中，其中盒子的编号应当等于小球编号上每位数字的和。例如，编号 321 的小球应当放入编号 3 + 2 + 1 = 6 的盒子，而编号 10 的小球应当放入编号 1 + 0 = 1 的盒子。

给你两个整数 lowLimit 和 highLimit ，返回放有最多小球的盒子中的小球数量。如果有多个盒子都满足放有最多小球，只需返回其中任一盒子的小球数量。

解题思路

- 简单题。循环遍历一遍数组，依次计算出所有小球的编号各位数字累加和，并且动态维护放有小球最多的数目。循环结束，输出最多小球个数即可。

代码

```go
package leetcode

func countBalls(lowLimit int, highLimit int) int {
  buckets, maxBall := [46]int{}, 0
  for i := lowLimit; i <= highLimit; i++ {
    t := 0
    for j := i; j > 0; {
      t += j % 10
      j = j / 10
```

```
    }
    buckets[t]++
    if buckets[t] > maxBall {
      maxBall = buckets[t]
    }
  }
  return maxBall
}
```

1744. Can You Eat Your Favorite Candy on Your Favorite Day?

题目

You are given a **(0-indexed)** array of positive integers `candiesCount` where `candiesCount[i]` represents the number of candies of the `ith` type you have. You are also given a 2D array `queries` where `queries[i] = [favoriteTypei, favoriteDayi, dailyCapi]`.

You play a game with the following rules:

- You start eating candies on day **`0`**.
- You **cannot** eat **any** candy of type `i` unless you have eaten **all** candies of type `i - 1`.
- You must eat **at least one** candy per day until you have eaten all the candies.

Construct a boolean array `answer` such that `answer.length == queries.length` and `answer[i]` is `true` if you can eat a candy of type `favoriteTypei` on day `favoriteDayi` without eating **more than** `dailyCapi` candies on **any** day, and `false` otherwise. Note that you can eat different types of candy on the same day, provided that you follow rule 2.

Return *the constructed array* `answer`.

Example 1:

```
Input: candiesCount = [7,4,5,3,8], queries = [[0,2,2],[4,2,4],[2,13,1000000000]]
Output: [true,false,true]
Explanation:
1- If you eat 2 candies (type 0) on day 0 and 2 candies (type 0) on day 1, you will eat a candy of type 0 on day 2.
2- You can eat at most 4 candies each day.
   If you eat 4 candies every day, you will eat 4 candies (type 0) on day 0 and 4 candies (type 0 and type 1) on day 1.
   On day 2, you can only eat 4 candies (type 1 and type 2), so you cannot eat a candy of type 4 on day 2.
3- If you eat 1 candy each day, you will eat a candy of type 2 on day 13.
```

Example 2:

```
Input: candiesCount = [5,2,6,4,1], queries = [[3,1,2],[4,10,3],[3,10,100],[4,100,30],
[1,3,1]]
Output: [false,true,true,false,false]
```

Constraints:

- `1 <= candiesCount.length <= 105`
- `1 <= candiesCount[i] <= 105`
- `1 <= queries.length <= 105`
- `queries[i].length == 3`
- `0 <= favoriteTypei < candiesCount.length`
- `0 <= favoriteDayi <= 109`
- `1 <= dailyCapi <= 109`

题目大意

给你一个下标从 0 开始的正整数数组 candiesCount ，其中 candiesCount[i] 表示你拥有的第 i 类糖果的数目。同时给你一个二维数组 queries ，其中 queries[i] = [favoriteTypei, favoriteDayi, dailyCapi] 。你按照如下规则进行一场游戏：

- 你从第 0 天开始吃糖果。
- 你在吃完 所有 第 i - 1 类糖果之前，不能 吃任何一颗第 i 类糖果。
- 在吃完所有糖果之前，你必须每天 至少 吃 一颗 糖果。

请你构建一个布尔型数组 answer ，满足 answer.length == queries.length 。answer[i] 为 true 的条件是：在每天吃 不超过 dailyCapi 颗糖果的前提下，你可以在第 favoriteDayi 天吃到第 favoriteTypei 类糖果；否则 answer[i] 为 false 。注意，只要满足上面 3 条规则中的第二条规则，你就可以在同一天吃不同类型的糖果。请你返回得到的数组 answer 。

解题思路

- 每天吃糖个数的下限是 1 颗，上限是 dailyCap。针对每一个 query 查询在第 i 天能否吃到 i 类型的糖果，要想吃到 i 类型的糖果，必须吃完 i-1 类型的糖果。意味着在 [favoriteDayi + 1, (favoriteDayi+1)×dailyCapi] 区间内能否包含一颗第 favoriteTypei 类型的糖果。如果能包含则输出 true，不能包含则输出 false。吃的糖果数是累积的，所以这里利用前缀和计算出累积吃糖果数所在区间 [sum[favoriteTypei−1]+1, sum[favoriteTypei]]。最后判断 query 区间和累积吃糖果数的区间是否有重叠即可。如果重叠即输出 true。
- 判断两个区间是否重合，情况有好几种：内包含，全包含，半包含等等。没有交集的情况比较少，所以可以用排除法。对于区间 [x1, y1] 以及 [x2, y2]，它们没有交集当且仅当 x1 > y2 或者 y1 < x2。

代码

```go
package leetcode

func canEat(candiesCount []int, queries [][]int) []bool {
  n := len(candiesCount)
  prefixSum := make([]int, n)
  prefixSum[0] = candiesCount[0]
  for i := 1; i < n; i++ {
```

```
    prefixSum[i] = prefixSum[i-1] + candiesCount[i]
  }
  res := make([]bool, len(queries))
  for i, q := range queries {
    favoriteType, favoriteDay, dailyCap := q[0], q[1], q[2]
    x1 := favoriteDay + 1
    y1 := (favoriteDay + 1) * dailyCap
    x2 := 1
    if favoriteType > 0 {
      x2 = prefixSum[favoriteType-1] + 1
    }
    y2 := prefixSum[favoriteType]
    res[i] = !(x1 > y2 || y1 < x2)
  }
  return res
}
```

1748. Sum of Unique Elements

题目

You are given an integer array `nums`. The unique elements of an array are the elements that appear **exactly once** in the array.

Return *the **sum** of all the unique elements of* `nums`.

Example 1:

```
Input: nums = [1,2,3,2]
Output: 4
Explanation: The unique elements are [1,3], and the sum is 4.
```

Example 2:

```
Input: nums = [1,1,1,1,1]
Output: 0
Explanation: There are no unique elements, and the sum is 0.
```

Example 3:

```
Input: nums = [1,2,3,4,5]
Output: 15
Explanation: The unique elements are [1,2,3,4,5], and the sum is 15.
```

Constraints:

- `1 <= nums.length <= 100`
- `1 <= nums[i] <= 100`

题目大意

给你一个整数数组 `nums` 。数组中唯一元素是那些只出现 **恰好一次** 的元素。请你返回 `nums` 中唯一元素的 **和** 。

解题思路

- 简单题。利用 map 统计出每个元素出现的频次。再累加所有频次为 1 的元素，最后输出累加和即可。

代码

```go
package leetcode

func sumOfUnique(nums []int) int {
  freq, res := make(map[int]int), 0
  for _, v := range nums {
    if _, ok := freq[v]; !ok {
      freq[v] = 0
    }
    freq[v]++
  }
  for k, v := range freq {
    if v == 1 {
      res += k
    }
  }
  return res
}
```

1752. Check if Array Is Sorted and Rotated

题目

Given an array `nums`, return `true` *if the array was originally sorted in non-decreasing order, then rotated **some** number of positions (including zero)*. Otherwise, return `false`.

There may be **duplicates** in the original array.

Note: An array `A` rotated by `x` positions results in an array `B` of the same length such that `A[i] == B[(i+x) % A.length]`, where `%` is the modulo operation.

Example 1:

```
Input: nums = [3,4,5,1,2]
Output: true
Explanation: [1,2,3,4,5] is the original sorted array.
You can rotate the array by x = 3 positions to begin on the the element of value 3: [3,4,5,1,2].
```

Example 2:

```
Input: nums = [2,1,3,4]
Output: false
Explanation: There is no sorted array once rotated that can make nums.
```

Example 3:

```
Input: nums = [1,2,3]
Output: true
Explanation: [1,2,3] is the original sorted array.
You can rotate the array by x = 0 positions (i.e. no rotation) to make nums.
```

Example 4:

```
Input: nums = [1,1,1]
Output: true
Explanation: [1,1,1] is the original sorted array.
You can rotate any number of positions to make nums.
```

Example 5:

```
Input: nums = [2,1]
Output: true
Explanation: [1,2] is the original sorted array.
You can rotate the array by x = 5 positions to begin on the element of value 2: [2,1].
```

Constraints:

- `1 <= nums.length <= 100`
- `1 <= nums[i] <= 100`

题目大意

给你一个数组 nums 。nums 的源数组中，所有元素与 nums 相同，但按非递减顺序排列。如果 nums 能够由源数组轮转若干位置（包括 0 个位置）得到，则返回 true ；否则，返回 false 。源数组中可能存在 重复项 。

解题思路

- 简单题。从头扫描一遍数组，找出相邻两个元素递减的数对。如果递减的数对只有 1 个，则有可能是轮转得来的，超过 1 个，则返回 false。题干里面还提到可能有多个重复元素，针对这一情况还需要判断一下 `nums[0]` 和 `nums[len(nums)-1]` 。如果是相同元素，`nums[0] < nums[len(nums)-1]`，并且数组中间还存在一对递减的数对，这时候也是 false。判断好上述这 2 种情况，本题得解。

代码

```
package leetcode
```

```go
func check(nums []int) bool {
  count := 0
  for i := 0; i < len(nums)-1; i++ {
    if nums[i] > nums[i+1] {
      count++
      if count > 1 || nums[0] < nums[len(nums)-1] {
        return false
      }
    }
  }
  return true
}
```

1758. Minimum Changes To Make Alternating Binary String

题目

You are given a string `s` consisting only of the characters `'0'` and `'1'`. In one operation, you can change any `'0'` to `'1'` or vice versa.

The string is called alternating if no two adjacent characters are equal. For example, the string `"010"` is alternating, while the string `"0100"` is not.

Return *the **minimum** number of operations needed to make* `s` *alternating*.

Example 1:

```
Input: s = "0100"
Output: 1
Explanation: If you change the last character to '1', s will be "0101", which is
alternating.
```

Example 2:

```
Input: s = "10"
Output: 0
Explanation: s is already alternating.
```

Example 3:

```
Input: s = "1111"
Output: 2
Explanation: You need two operations to reach "0101" or "1010".
```

Constraints:

- `1 <= s.length <= 104`
- `s[i]` is either `'0'` or `'1'`.

题目大意

你将得到一个仅包含字符“ 0”和“ 1”的字符串 `s`。 在一项操作中，你可以将任何 `'0'` 更改为 `'1'`，反之亦然。 如果两个相邻字符都不相等，则该字符串称为交替字符串。 例如，字符串“ 010”是交替的，而字符串“ 0100”则不是。 返回使 `s` 交替所需的最小操作数。

解题思路

- 简单题。利用数组下标奇偶交替性来判断交替字符串。交替字符串有 2 种，一个是 `'01010101……'` 还有一个是 `'1010101010……'`，这两个只需要计算出一个即可，另外一个利用 `len(s) - res` 就是答案。

代码

```
package leetcode

func minOperations(s string) int {
  res := 0
  for i := 0; i < len(s); i++ {
    if int(s[i]-'0') != i%2 {
      res++
    }
  }
  return min(res, len(s)-res)
}

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