# 

[**1. Two Sum**](#_c1shqcpbdft1) **4**

[**2. Add Two Numbers**](#_9v1b2paxy2b6) **4**

[**3. longest substring without repeating characters**](#_ee7nobsh30i) **6**

[**4. Median of Two Sorted Arrays**](#_s0lbh97zmrnj) **7**

[**5. Longest Palindromic Substring**](#_u73zy9v3j8j1) **9**

[**6. Zigzag Conversion**](#_gqysvdu3nb2u) **10**

[**7. Reverse Integer**](#_5dzkhabrrb5b) **11**

[**8. String to Integer (atoi)**](#_8g2mcm6odvo6) **11**

[**9. Palindrome Number**](#_nspomwx1r13a) **12**

[**10. Regular Expression Matching**](#_9un3e23amjuw) **13**

[**11. Container With Most Water**](#_o2ikophf7soc) **14**

[**14. Longest Common Prefix**](#_kyo0d5ybqu8c) **15**

[**15. 3Sum**](#_i01a1rmsu99u) **16**

[**16. 3Sum Closest**](#_l3juj9ua1daw) **16**

[**259. 3Sum Smaller**](#_crjzzqyr7402) **16**

[**17. Letter Combinations of a Phone Number**](#_5n4o408v121w) **19**

[**19. Remove Nth Node From End of List**](#_fql9x5ituio1) **20**

[**20. Valid Parentheses**](#_d04ojn2vl0i8) **20**

[**21. Merge Two Sorted Lists**](#_ip0d2f5x6rrb) **21**

[**23. Merge k Sorted Lists**](#_n5szxeu9mavs) **22**

[**13. Roman to Integer**](#_2cn9feqoodr) **22**

[**24 Swap Nodes in Pairs**](#_bm8iiut4s5qx) **23**

[**25 Reverse Nodes in k-Group**](#_yryfm23fuzy3) **24**

[**26 Remove Duplicates from Sorted Array**](#_tp9hom4w1vhg) **25**

[**29 Divide Two Integers**](#_pr4h4ut6016q) **26**

[**31 Next Permutation**](#_brgtsdgl529m) **26**

[**46. Permutations**](#_1uri4p54zdj0) **28**

[**32 Longest Valid Parentheses**](#_2l92fo3a35kh) **29**

[**33 Search in Rotated Sorted Array**](#_tpvwhbsw002x) **31**

[**34 Find First and Last Position of Element in Sorted Array**](#_1dqinnkhznsj) **32**

[**35 Search Insert Position**](#_947xnr64ccjr) **33**

[**36 Valid Sudoku**](#_xx3691x4a78) **34**

[**39 Combination Sum**](#_ndc1baxk915x) **35**

[**40 Combination Sum II**](#_yixjmrk57mq2) **36**

[**41 First Missing Positive**](#_vrynnl4x4i8) **37**

[**42. Trapping Rain Water**](#_ujt651a277kp) **38**

[**43. Multiply Strings**](#_adwui369d63z) **39**

[**44. Wildcard Matching**](#_rmq5c2eomu6t) **40**

[**55. Jump Game**](#_qdm52bqmecc5) **42**

[**45. Jump Game II**](#_3m6p8mn8hyap) **42**

[**1306. Jump Game III**](#_svgxc1jtme6y) **44**

[**43 Multiply Strings**](#_8x575vodla0a) **45**

[**792. Number of Matching Subsequences**](#_841yr6ga8kx3) **46**

[**51 N-Queens**](#_t4oggot1m39c) **47**

[**48 Rotate Image**](#_8nq3lcsjaldn) **49**

[**49 Group Anagrams**](#_f3znmibqqawr) **50**

[**50 Pow(x, n)**](#_gj40ayg97rx6) **50**

[**53 Maximum Subarray**](#_cnj3kroddh93) **50**

[**54 Spiral Matrix**](#_q9vnhgo5zw9i) **50**

[**1849. Splitting a String Into Descending Consecutive Values**](#_xcnffhtpygxi) **51**

[**1110 Delete Nodes And Return Forest**](#_t06tw29jgx7p) **53**

[**56 Merge Intervals**](#_dqcdq2uhipba) **55**

[**57 Insert Interval**](#_cfk317cx03l1) **56**

[**60 Permutation Sequence**](#_si67oy5issb2) **57**

[**62 Unique Paths**](#_xp9ktfi4smge) **57**

[**63 Unique Paths II**](#_otk3uhf6kiw5) **57**

[**65 Valid Number**](#_tyq94jpxj4l5) **57**

[**66 Plus One**](#_by0dj45jizp0) **58**

[**67 Add Binary**](#_8wkjgqufsxu4) **59**

[**767 Reorganize String**](#_ma3doienanc8) **61**

[**209. Minimum Size Subarray Sum**](#_4ttjp55tml2q) **62**

[**380 Insert Delete GetRandom O(1)**](#_mb3dvqm9su9v) **63**

[**398 Random Pick Index**](#_kq6adl6bcgk2) **64**

[**528. Random Pick with Weight**](#_wm72j8a5mrsi) **67**

[**78 Subsets**](#_h60zy8pe6l9v) **67**

[**815 Bus Routes**](#_p0zihe9zcis6) **68**

[**76 Minimum Window Substring**](#_ywirn4p2w4lz) **70**

[**146. LRU Cache**](#_ys6a37me6wiz) **72**

[**212. Word Search**](#_htxq23e6lg5) **74**

[**1057. Campus Bikes**](#_7tuimykd2xj2) **76**

[**726. Number of Atoms**](#_j5en6sx73l5v) **77**

[**1597. Build Binary Expression Tree From Infix Expression**](#_q4azv1fxq40m) **79**

[Related solution for 772. Basic Calculator III](#_qzma8yf6o6jj) 80

[**1231. Divide Chocolate**](#_utitnuc7f83g) **81**

[**875. Koko Eating Bananas**](#_chf67hurlip4) **82**

[**935. Knight Dialer**](#_b2tglaeo55wr) **83**

[**616. Add Bold Tag in String**](#_mrc8zkq01zdh) **85**

[**126. Word Ladder II**](#_8toa6jabzaf5) **85**

[**127. Word Ladder**](#_eyj5f46si527) **87**

[**102. Binary Tree Level Order Traversal**](#_al1zaylw92c8) **88**

[**103. Binary Tree Zigzag Level Order Traversal**](#_qavmsml521v4) **89**

[108. Convert Sorted Array to Binary Search Tree](#_lxgnd0bg9g8q) **90**

[**979. Distribute Coins in Binary Tree**](#_laybq13bec12) **91**

[**股票买卖专题**](#_6d5so045mwkh) **92**

[121. Best Time to Buy and Sell Stock (only one time transaction)](#_enukbh5ainwy) 92

[122. Best Time to Buy and Sell Stock II (no limit on transaction times)](#_hjsayf9d0kjz) 94

[714. Best Time to Buy and Sell Stock with Transaction Fee (122 + transaction fee)](#_6wmzfn2nnfjb) 96

[123. Best Time to Buy and Sell Stock III (at most 2 times transaction, hard)](#_vlfr6aog7xha) 98

[188. Best Time to Buy and Sell Stock IV (at most K times transaction, hard)](#_7lcslf25n4ra) 99

[309. Best Time to Buy and Sell Stock with Cooldown](#_4gyu6jvps179) 101

[**437. Path Sum III**](#_h05awufz42zo) **102**

[96. Unique Binary Search Trees](#_fmr9m33mtp7u) **103**

[**1120. Maximum Average Subtree**](#_n4hwpjiwuqva) **104**

[**1339. Maximum Product of Splitted Binary Tree**](#_kf79ctrcvs5h) **104**

[**151 Reverse Words in a String**](#_w3dy7e67yqk5) **105**

[**152 Maximum Product Subarray**](#_2srgzv9b5ezs) **106**

[**158 Read N Characters Given read4 II - Call Multiple Times**](#_srsl8spp1qp2) **107**

[**Lowest Common Ancestors 专题**](#_o9tnqwdruwwf) **107**

[236. Lowest Common Ancestor of a Binary Tree](#_gaiox9izz77j) 107

[1644. Lowest Common Ancestor of a Binary Tree II](#_66k76zkekw4d) 109

[1650. Lowest Common Ancestor of a Binary Tree III](#_jdj5yund139o) 111

[1676. Lowest Common Ancestor of a Binary Tree IV](#_tanyquvgvt4e) 114

# 

# 

# 

# 

# 1. Two Sum

(Wu)

python

class Solution:

def twoSum(self, nums: List[int], target: int) -> List[int]:

s = {}

for i, e in enumerate(nums):

if target - e in s:

return [i, s[target-e]]

s[e] = i

follow up: 如果数据量很大怎么办？

Step 1: Ask if the data is already sorted

Step 2: If sorted - 从硬盘load 最前面和最后面到内存里，使用two pointers的方法，如果找不到再load前面的下一部分，和后面的前一部分

Step 3: If not sorted , 先用硬盘外排的方法sort所有数据，再按step2做two pointers的方法

Step 4: 外排的方法是 - 分成k份，分别到内存里排序，再merge k sorted array

# 2. Add Two Numbers

（Wu）

python

class Solution:

def addTwoNumbers(self, l1: Optional[ListNode], l2: Optional[ListNode]) -> Optional[ListNode]:

dummy = temp = ListNode()

carry = 0

while l1 or l2 or carry:

val1 = val2 = 0

if l1:

val1 = l1.val

l1 = l1.next

if l2:

val2 = l2.val

l2 = l2.next

temp.next = ListNode((val1 + val2 + carry)%10)

carry = (val1 + val2 + carry)//10

temp = temp.next

return dummy.next

follow up：减法

java

class Solution {

public ListNode addTwoNumbers(ListNode l1, ListNode l2) {

return addTwoNumbers(l1, l2, false);

}

public ListNode addTwoNumbers(ListNode l1, ListNode l2, boolean flip) {

ListNode dummyHead = new ListNode(0);

boolean allEqual = false;

//ListNode prev = dummyHead;

ListNode p = l1, q = l2, curr = dummyHead;

int carry = 0;

int last = 0;

while (p != null || q != null) {

if (p == null) {

return addTwoNumbers(l2, l1, true);

}

int x = (p != null) ? p.val : 0;

int y = (q != null) ? q.val : 0;

if (x == y) {

allEqual = true;

} else {

allEqual = false;

}

int sum = x + carry - y;

carry = 0;

while (p.next != null && sum < 0) {

carry--;

sum += 10;

}

last = sum;

if (p.next != null) {

curr.next = new ListNode(sum);

curr = curr.next;

}

if (p != null) p = p.next;

if (q != null) q = q.next;

}

if (carry < 0) {

last = carry;

}

if (carry == 0 && allEqual) {

return new ListNode(0);

}

if (last < 0 && !flip) {

return addTwoNumbers(l2, l1, true);

}

if (flip)

curr.next = new ListNode(-last);

else

curr.next = new ListNode(last);

return dummyHead.next;

}

}

// 342 - 465 = -123

// 9999999 - 9999 = 9990000

// 999 - 999 = 0

// 9999 - 9999999= -9990000

# 

# 3. longest substring without repeating characters

High Level: 双指针, 慢指针跳着走

(huang)

C#

public class Solution {

public int LengthOfLongestSubstring(string s) {

Dictionary<int, int> slow\_map = new Dictionary<int, int>();

int slow = 0;

int maxlen = 0;

for(int fast = 0; fast < s.Length; fast ++)

{

char ch = s[fast];

if (slow\_map.ContainsKey(ch)){

slow = Math.Max(slow, slow\_map[ch] + 1);

}

maxlen = Math.Max(maxlen, fast - slow + 1);

slow\_map[ch] = fast;

}

return maxlen;

}

}

private static List<String> longestSubstr(String s) {

List<String> res = new ArrayList<>();

int start = 0;

int end = 0;

int maxLen = 0, counter = 0;

int[] map = new int[128]; // to check the char exists or not

while ( end < s.length()) {

char c1 = s.charAt(end);

if (map[c1] > 0) counter++;

map[c1]++; // set c1 from 0 to 1

end++;

while (counter > 0) {

char c2 = s.charAt(start);

if (map[c2] > 1) counter--; // c1 == c2, decrease the counter

map[c2]--;

start++; // reset start to new pos

}

maxLen = Math.max(maxLen, end - start);

String substr = s.substring(start, end); // candidates

res.add(substr);

}

// maxLen, find all substr in res which len == maxLen

List<String> subset = new ArrayList<>();

for (String str : res) {

if (str.length() == maxLen) {

subset.add(str);

}

}

return subset;

}

}

# 4. Median of Two Sorted Arrays

High Level: 找第k个数 + recursion

(huang)

| public double median(int[] a, int[] b) {  // Write your solution here  //time : O(nlogn)  //space : O(1)  Arrays.sort(a);  Arrays.sort(b);  int length = a.length + b.length;  int mid0 = (length - 1 )/ 2;  int mid1 = length / 2;  int left = 0;  int right = length - 1;  while(left < mid0 || right > mid1) {//quick select + binary search or whatever  int pivot = quickSelectIndex(a, b, left, right);  if(pivot <= mid0) {  left = pivot + 1;  } else {  right = pivot - 1;  }  }  return (double)(num(a, b, mid0) + num(a, b, mid1)) /2;  }  private int num(int[] a, int[] b, int i) {//2-array mapping function  return i < a.length ? a[i] : b[i - a.length];  }  private void swap(int[] a, int[] b, int i, int j) {//2-array mapping and swap function  int[] numi = i < a.length ? a : b;  int[] numj = j < a.length ? a : b;  i = i < a.length ? i : i - a.length;  j = j < a.length ? j : j - a.length;  int tmp = numi[i];  numi[i] = numj[j];  numj[j] = tmp;  return;  }  private int quickSelectIndex(int[] a, int[] b, int left, int right) {  int pivot = left + (int)(Math.random() \* (right + 1 - left) );  int i = left;  int j = right - 1;  swap(a, b, pivot, right);  while(i <= j) {  if(num(a, b, i) > num(a, b, right)) {  if(num(a, b, j) >= num(a, b, right)) {  j--;  } else {  swap(a, b, i, j);  i++;  j--;  }  } else {  i++;  }  }  swap(a, b, i, right);  return i ;  } |
| --- |

# 5. Longest Palindromic Substring

（tang）

High Level：

Java:

// TC: O(n^2) SC: O(1) 空间优化的方法

| class Solution { public String longestPalindrome(String s) {  if (s == null || s.length() < 1) return "";  int start = 0, end = 0;  for (int i = 0; i < s.length(); i++) {  int len1 = expandAroundCenter(s, i, i);  int len2 = expandAroundCenter(s, i, i + 1);  int len = Math.max(len1, len2);  if (len > end - start) {  start = i - (len - 1) / 2;  end = i + len / 2;  }  }  return s.substring(start, end + 1); }  private int expandAroundCenter(String s, int left, int right) {  int L = left, R = right;  while (L >= 0 && R < s.length() && s.charAt(L) == s.charAt(R)) {  L--;  R++; // (L, R)  }  return R - L - 1; } }  // dp的方法 SC O(n^2) class Solution { public String longestPalindrome(String s) {  if (s == null || s.length() < 1) return "";  int start = 0, end = 0;  for (int i = 0; i < s.length(); i++) {  int len1 = expandAroundCenter(s, i, i);  int len2 = expandAroundCenter(s, i, i + 1);  int len = Math.max(len1, len2);  if (len > end - start) {  start = i - (len - 1) / 2;  end = i + len / 2;  }  }  return s.substring(start, end + 1); }  private int expandAroundCenter(String s, int left, int right) {  int L = left, R = right;  while (L >= 0 && R < s.length() && s.charAt(L) == s.charAt(R)) {  L--;  R++; // (L, R)  }  return R - L - 1; } } |
| --- |

# 6. Zigzag Conversion

（tang）

High Level:

StringBuilders

boolean goingdown, 换方向

TC: O(1)

class Solution {

public String longestPalindrome(String s) {

if (s == null || s.length() < 1) return "";

int start = 0, end = 0;

for (int i = 0; i < s.length(); i++) {

int len1 = expandAroundCenter(s, i, i);

int len2 = expandAroundCenter(s, i, i + 1);

int len = Math.max(len1, len2);

if (len > end - start) {

start = i - (len - 1) / 2;

end = i + len / 2;

}

}

return s.substring(start, end + 1);

}

private int expandAroundCenter(String s, int left, int right) {

int L = left, R = right;

while (L >= 0 && R < s.length() && s.charAt(L) == s.charAt(R)) {

L--;

R++; // (L, R)

}

return R - L - 1;

}

}

# 7. Reverse Integer

(lynn)

需要处理边界 - 用long

或者每次需要乘以0时，判断是否valid

Time : O(1) max should be the digit number of Integer\_MAX\_VALUE

Space : O(1)

class Solution {

public int reverse(int x) {

int result = 0;

long longRes = 0;

boolean negative = x > 0 ? false : true;

x = x > 0 ? x : -1 \* x;

while (x != 0) {

int digit = x % 10;

int curResult = result \* 10 + digit;

//curResult might be overflow

//check overflow, if we calculate back to match result failed, represent there is an overflow

if ((curResult - digit) / 10 != result) {

return 0;

}

result = curResult;

x = x / 10;

}

return negative ? -1 \* result : result;

}

}

# 8. String to Integer (atoi)

(lynn)

Java:

Time O(n) n is the length of input string

Space : O(1)

class Solution {

public int myAtoi(String s) {

if (s == null || s.length() == 0) {

return 0;

}

//maintain an index to loop the char in string

int index = 0;

//Step1: remove leading space

while (index < s.length() && s.charAt(index) == ' ') {

index++;

}

//Step2: check the sign

int sign = 1;

if (index < s.length() && (s.charAt(index) == '-' || s.charAt(index) == '+')) {

sign = s.charAt(index) == '-' ? -1 : 1;

index++;

}

//Step3: read the digit until non-digit

int result = 0;

while (index < s.length() && s.charAt(index) >= '0' && s.charAt(index) <= '9') {

int digit = s.charAt(index) - '0';

**if (result > Integer.MAX\_VALUE / 10**

**|| (result == Integer.MAX\_VALUE / 10 && digit > Integer.MAX\_VALUE % 10)) {**

**return sign == 1 ? Integer.MAX\_VALUE : Integer.MIN\_VALUE;**

**}**

result = result \* 10 + digit;;

index++;

}

return sign \* result;

}

}

# 9. Palindrome Number

(zhang)

High Level: 取%，取/

Java:

//方法一：

class Solution {

public boolean isPalindrome(int x) {

if (x < 0) return false;

int rev = 0;

int y = x;

while ( y != 0) {

rev = rev \* 10 + y % 10;

System.out.println(rev);

y = y / 10;

}

return rev == x; // -1126087180 overflow return false;

}

}

//方法二：只做一半，不需要考虑overflow的情况，更好

c#

public class Solution {

public bool IsPalindrome(int x) {

if(x < 0 || (x % 10 == 0 && x != 0)) {

return false;

}

int half = 0;

while( x > half){

half = half \*10 + x % 10;

x /= 10;

}

return half == x || ( x == half /10);

}

}

# 10. Regular Expression Matching

(luo)

/\*

dp的方法

TC： O（mn）

SC: O(mn)

https://www.youtube.com/watch?v=bSdw9rJYf-I

\*/

// dp的方法：复杂度更好

class Solution {

public boolean isMatch(String s, String p) {

int lenS = s.length();

int lenP = p.length();

char[] sArray = s.toCharArray();

char[] pArray = p.toCharArray();

boolean[][] dp = new boolean[lenS + 1][lenP + 1];

dp[0][0] = true;

//当s的长度为0的情况

for (int i = 2; i <= lenP; i++) {

dp[0][i] = pArray[i - 1] == '\*' ? dp[0][i - 2] : false;

}

for (int i = 1; i <= lenS; i++) {

for (int j = 1; j <= lenP; j++) {

char sc = sArray[i - 1];

char pc = pArray[j - 1];

if (sc == pc || pc == '.') {

dp[i][j] = dp[i - 1][j - 1];

} else {

if (pc == '\*') { // 前面的字母重复0次，直接看j-2的位置

if (dp[i][j - 2]) {

dp[i][j] = true;

} else if (sc == p.charAt(j - 2) || p.charAt(j - 2) == '.') {// 前面的字母重复1次

dp[i][j] = dp[i - 1][j];

}

}

}

}

}

return dp[lenS][lenP];

}

}

// recursion

/\*

recursion的方法

先看第一个字母是否match

再看剩下的

TC: O((lenS+lenP) \* 2^(lenS + lenP))

\*/

// recursion的方法，代码简短

class Solution {

public boolean isMatch(String s, String p) {

// corner case check, if p length is 0 but s length is not 0, return false

if (p.length() == 0) return s.length() == 0;

boolean firstMatch = s.length() > 0 && (s.charAt(0) == p.charAt(0) || p.charAt(0) == '.');

if (p.length() >= 2 && p.charAt(1) == '\*' ) {

return isMatch(s, p.substring(2)) || (firstMatch && isMatch(s.substring(1), p));

} else {

return firstMatch && isMatch(s.substring(1), p.substring(1));

}

}

}

# 11. Container With Most Water

（Aye）

双指针

public int maxArea(int[] height) {

if (height == null || height.length == 0) {

return 0;

}

int start = 0;

int end = height.length - 1;

int maxArea = 0;

while (start < end) {

int current = Math.min(height[start], height[end]);

maxArea = Math.max(maxArea, current \* (end - start));

while (start < end && height[start] <= current) {

start++;

}

while (start < end && height[end] <= current) {

end--;

}

}

return maxArea;

}

# 14. Longest Common Prefix

(zhang)

class Solution {

// Horizontal scanning

public String longestCommonPrefix(String[] strs) {

if (strs.length == 0) return "";

String prefix = strs[0];

for (int i = 1; i < strs.length; i++)

while (strs[i].indexOf(prefix) != 0) {

prefix = prefix.substring(0, prefix.length() - 1);

if (prefix.isEmpty()) return "";

}

return prefix;

}

// Vertical scanning

public String longestCommonPrefix\_Vertical(String[] strs) {

if (strs == null || strs.length == 0) return "";

for (int i = 0; i < strs[0].length() ; i++){

char c = strs[0].charAt(i);

for (int j = 1; j < strs.length; j ++) {

if (i == strs[j].length() || strs[j].charAt(i) != c)

return strs[0].substring(0, i);

}

}

return strs[0];

}

public String longestCommonPrefix\_sort(String[] strs) {

if (strs == null || strs.length == 0) return "";

Arrays.sort(strs);

int len = Math.min(strs[0].length(), strs[strs.length - 1].length());

int i = 0;

while (i < len && strs[0].charAt(i) == strs[strs.length - 1].charAt(i))

i++;

return strs[0].substring(0, i);

}

}

# 

# 15. 3Sum

（Aye）

TC: O(n^2)

SC: O(1)

注意remove duplicates

1 + 2sum

Java:

# 16. 3Sum Closest

(Aye)

# 259. 3Sum Smaller

(Aye), (huang), (tang)

先sort，再 two pointers

| // TC： O（n^2） class Solution {  public int threeSumSmaller(int[] nums, int target) {  Arrays.sort(nums);  int sum = 0;  for (int i = 0; i < nums.length - 2; i++) {  sum += twoSumSmaller(nums, i + 1, target - nums[i]);  }  return sum;  }   private int twoSumSmaller(int[] nums, int startIndex, int target) {  int sum = 0;  int left = startIndex;  int right = nums.length - 1;  while (left < right) {  if (nums[left] + nums[right] < target) {  sum += right - left; //这里是关键  left++;  } else {  right--;  }  }  return sum;  } } |
| --- |

| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Hybrid\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ class Solution {  public int threeSumSmaller(int[] nums, int target) {  Arrays.sort(nums);  int sum = 0;  for (int i = 0; i < nums.length - 2; i++) {  sum += twoSumSmaller(nums, i + 1, target - nums[i]);  }  return sum;  }   private int twoSumSmaller(int[] nums, int startIndex, int target) {  int sum = 0;  int left = startIndex;  int right = nums.length - 1;  while (left < right) {  if (nums[left] + nums[right] < target) {  /\*   L   [left,nL]?? right   | | |  -3 -2 -1 0 1 3 t: 2 [left, right - 1], newTarget: target - nums[right]  each point (L, right]: right-left, right - nL      \*/  int newLeft = binarySearch(nums, left, right - 1, target - nums[right]);  if (newLeft == -1) {  return sum;  }  sum += ((right - left) + (right - newLeft)) \* (newLeft - left + 1) / 2;  left = newLeft;  left++;  } else {  int newRight = binarySearch(nums, left + 1, right, target - nums[left]);  if (newRight == -1) {  return sum;  }  right = newRight;  }  }  return sum;  }    private int binarySearch(int[] arr, int left, int right, int target) {  int mid;  int left0 = left;  if (arr[right] < target) {  return right;  }  if (right >= left + 1 && arr[left+1] < target) {  while (left < right - 1) {  mid = left + (right - left) / 2;  if (arr[mid] >= target) {  right = mid;  } else {  left = mid;  }  }  if (arr[right] < target) {  return right;  }  if (arr[left] < target) {  return left;  }   }   if (arr[left0] < target) {  return left;  }  return -1;   } } /\* Runtime: 5 ms, faster than 98.30% of Java online submissions for 3Sum Smaller. Memory Usage: 38.2 MB, less than 96.10% of Java online submissions for 3Sum Smaller. \*/ |
| --- |

# 17. Letter Combinations of a Phone Number

（huang）

dfs

C#

//(luo) DFS

class Solution {

//TC: O(4^n)

// SC: O(n)

public List<String> letterCombinations(String digits) {

if (digits == null || digits.length() == 0) {

return new ArrayList<>();

}

String[] numChar = {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};

char[] input = digits.toCharArray();

List<String> result = new ArrayList<>();

StringBuilder sb = new StringBuilder();

helper(input, numChar, sb, result, 0);

return result;

}

private void helper(char[] input, String[] numChar, StringBuilder sb, List<String> result, int index) {

if (index == input.length) {

result.add(sb.toString());

return;

}

char[] chars = numChar[input[index] - '0'].toCharArray(); // 注意这里

for (int i = 0; i < chars.length; i++) {

sb.append(chars[i]);

helper(input, numChar, sb, result, index + 1);

sb.deleteCharAt(sb.length() - 1);

}

}

}

# 19. Remove Nth Node From End of List

(tang)

| // 快慢指针 class Solution {  public ListNode removeNthFromEnd(ListNode head, int n) {  ListNode dummy = new ListNode(0);  dummy.next = head;  ListNode slow = head;  ListNode fast = head;  int count = 0;  while (count < n && fast.next != null) {  fast = fast.next;  count++;  }  if (count < n) {  return head.next;  }  while (fast.next != null) {  fast = fast.next;  slow = slow.next;  }  slow.next = slow.next.next;  return dummy.next;  } } /\* n = 2 count = 2;  s  | |  d>1,2,3> 5  o(2n) o(1) \*/ |
| --- |

# 20. Valid Parentheses

(Lynn)

High Level: stack

public boolean isValid(String s) {

if (s == null || s.length() == 0) {

return true;

}

int index = 0;

Deque<Character> stack = new ArrayDeque<>();

while (index < s.length()) {

char c = s.charAt(index);

if (c == '(' || c =='{' || c == '[') {

stack.push(c);

} else if (c == ')') {

if (stack.isEmpty() || stack.pop() != '(') {

return false;

}

} else if (c == ']') {

if (stack.isEmpty() || stack.pop() != '[') {

return false;

}

} else if (c == '}') {

if (stack.isEmpty() || stack.pop() != '{') {

return false;

}

}

index++;

}

return stack.isEmpty();

}

# 21. Merge Two Sorted Lists

(Lynn)

public ListNode mergeTwoLists(ListNode l1, ListNode l2) {

if (l1 == null || l2 == null) {

return l1 == null ? l2 : l1;

}

ListNode dummy = new ListNode(0);

ListNode cur = dummy;

while (l1 != null && l2 != null) {

**if (l1.val <= l2.val) {**

**cur.next = l1;**

**l1 = l1.next;**

**} else {**

**cur.next = l2;**

**l2 = l2.next;**

**}**

cur = cur.next;

}

if (l1 != null) {

cur.next = l1;

}

if (l2 != null) {

cur.next = l2;

}

return dummy.next;

}

# 23. Merge k Sorted Lists

(Lynn)

public ListNode mergeKLists(ListNode[] lists) {

ListNode dummy = new ListNode(0);

ListNode cur = dummy;

**PriorityQueue<ListNode> minHeap = new PriorityQueue<>((a, b) -> a.val - b.val);**

for (ListNode head : lists) {

if (head != null) {

minHeap.offer(head);

}

}

while (!minHeap.isEmpty()) {

**//pop top out and add to result**

ListNode node = minHeap.poll();

cur.next = node;

cur = cur.next;

**//offer a next node to the queue**

if (node.next != null) {

minHeap.offer(node.next);

}

}

return dummy.next;

}

# 13. Roman to Integer

(luo)

/\*<https://www.youtube.com/watch?v=dlATMslQ6Uc>

从右往前依次处理

如果当前char比后一位的char，在map里的值小，需要减去当前char代表的value；

如果当前char比后一位的char，在map里的值大，则是正常的从左到右，从小到大的顺序，则只需要加上当前的char代表的value到result中

TC：O（n）从右到左一遍

SC：O（1）

\*/

class Solution {

public int romanToInt(String s) {

Map<Character, Integer> map = new HashMap<>();

map.put('I', 1);

map.put('V', 5);

map.put('X', 10);

map.put('L', 50);

map.put('C', 100);

map.put('D', 500);

map.put('M', 1000);

int n = s.length();

int result = map.get(s.charAt(n - 1));

for (int i = n - 2; i >= 0; i--) {

int currentValue = map.get(s.charAt(i));

if (currentValue < map.get(s.charAt(i + 1))) {

result -= currentValue;

} else {

result += currentValue;

}

}

return result;

}

}

# 

# 24 Swap Nodes in Pairs

(Huang)

// (Luo) TC: O(n) SC: O(1) recursively swap the nodes

class Solution {

public ListNode swapPairs(ListNode head) {

if (head == null || head.next == null) {

return head;

}

ListNode node1 = head;

ListNode node2 = head.next;

ListNode node3 = head.next.next;

ListNode node = swapPairs(node3);

node2.next = node1;

node1.next = node;

return node2;

}

}

# 25 Reverse Nodes in k-Group

| (Tang)  public ListNode reverseKGroup2(ListNode head, int k) {  ListNode dummy = new ListNode(0), start = dummy;  dummy.next = head;  ListNode p = start, c, n = p; // 0  while(n != null) {  p = start;   n = p;  start = p.next; // 1  for(int i = 0; i < k && n != null; i++) n = n.next; // 2  if(n == null) break;  for(int i = 0; i < k-1; i++) {  c = p.next;  p.next = c.next;  c.next = n.next;  n.next = c;  }  }  return dummy.next;  }   /\*--------k == 3----------------  \* n(tail)  \* p(prev)   \* start  \*dummy 1 2 3 4 5  \* cur  \*----------------------------\* 0/   /\*----------------------------  \* n(tail)  \* p(prev)   \* start   \*dummy 1 2 3 4 5  \* cur  \*----------------------------\* 1/      /\*----------------------------  \* n(tail)  \* p(prev)   \* start  \*dummy 1 2 3 4 5  \* cur  \*----------------------------\* 2/        /\*----------------------------  \* n(tail)  \* p(prev)   \* start  \*dummy (1) 2 3 4 5  \* cur  \*----------------------------\* 3-0/    /\*----------------------------  \* n(tail)  \* p(prev)   \* start  \*dummy 2 3 （1） 4 5  \* cur  \*----------------------------\* 3-1/      /\*----------------------------  \* n(tail)  \* p(prev)   \* start  \*dummy 3 2 1 4 5  \* cur  \*----------------------------\* 3-2/    /\*-----------top again--------------  \* n(tail)  \* p(prev)   \* start  \*dummy 3 2 1 4 5  \* cur  \*----------------------------\* 4/ |
| --- |

# 26 Remove Duplicates from Sorted Array

// code by Luo, 快慢指针

class Solution {

public int removeDuplicates(int[] nums) {

if (nums.length <= 1) {

return nums.length;

}

int slow = 1; // **[0...slow-1] to keep**

for (int i = 1; i < nums.length; i++) {**// fast pointer to traverse**

if (nums[i] != nums[slow - 1]) {

nums[slow] = nums[i];

slow++;

}

}

return slow;

}

}

# 29 Divide Two Integers

(zhang)

// TC: O(32)

High Level: bit operation

public int divide(int A, int B) {

if (A == 1 << 31 && B == -1) return (1 << 31) - 1;

int a = Math.abs(A), b = Math.abs(B), res = 0;

for (int x = 31; x >= 0; x--)

if ((a >>> x) - b >= 0) {

res += 1 << x;

a -= b << x;

}

return (A > 0) == (B > 0) ? res : -res;

}

# 31 Next Permutation

(Aye)



reference: <https://leetcode-cn.com/problems/next-permutation/solution/xia-yi-ge-pai-lie-suan-fa-xiang-jie-si-lu-tui-dao-/>

| // Java code by Tang  public void nextPermutation(int[] nums) {   int i = nums.length - 2;  while (i >= 0 && nums[i + 1] <= nums[i]) {  i--;  }  if (i >= 0) {  int j = nums.length - 1;  while (nums[j] <= nums[i]) {  j--;  }  swap(nums, i, j);  }  reverse(nums, i + 1);  }   private void reverse(int[] nums, int start) {  int i = start, j = nums.length - 1;  while (i < j) {  swap(nums, i, j);  i++;  j--;  }  }   private void swap(int[] nums, int i, int j) {  int temp = nums[i];  nums[i] = nums[j];  nums[j] = temp;  } |
| --- |

# 

# 46. Permutations

| Java code by Tang  public List<List<Integer>> permute(int[] nums) {  List<List<Integer>> res = new ArrayList<List<Integer>>();  if (nums == null || nums.length == 0) {  return res;  }  helper(nums, 0, res);  return res;  }  private void helper(int[] nums, int idx, List<List<Integer>> res) {  if (idx == nums.length - 1) {  List<Integer> tmp = toList(nums);  res.add(tmp);  return;  }  for (int i = idx; i < nums.length; i++) {  swap(nums, idx, i);  helper(nums, idx + 1, res);  swap(nums, idx, i);   }  }  private List<Integer> toList(int[] nums) {  List<Integer> res = new ArrayList<Integer>();  for (int i = 0; i < nums.length; i++) {  res.add(nums[i]);  }  return res;  }  private void swap(int[] arr, int i, int j) {  int tmp = arr[i];  arr[i] = arr[j];  arr[j] = tmp;  } /\* dfs recur tree o(n!) - tc;   ||  1 2 3 4 5   0, i -[0, .., n-1]      / | \ \  (1)2345 (2)xxx (3)xxx (4)xxxx (5)XXXXX n  /||\   (12)xxx (13)xx n-1  /||\   - - - - - n 1  |0  1 -[1,n]  ------ n-1  |1    n 1   ------  |  n   tc: n \* (n-1) \* (n-2) \*... \* 1 = n! sc: o(n)   \*/ |
| --- |

# 32 Longest Valid Parentheses

| (huang) Java code by Tang /\* \*\*\*\*\*\*\*\*\*\*\*\*Method 1: two times traversal with o(1) space complexity \*\*\*\*\*\*\*\*\*\*\*\*/  public int longestValidParentheses(String s) {  int left = 0, right = 0, maxlength = 0;  for (int i = 0; i < s.length(); i++) {  if (s.charAt(i) == '(') {  left++;  } else {  right++;  }  if (left == right) {  maxlength = Math.max(maxlength, 2 \* right);  } else if (right >= left) {  left = right = 0;  }  }  left = right = 0;  for (int i = s.length() - 1; i >= 0; i--) {  if (s.charAt(i) == '(') {  left++;  } else {  right++;  }  if (left == right) {  maxlength = Math.max(maxlength, 2 \* left);  } else if (left >= right) {  left = right = 0;  }  }  return maxlength;  } /\* \*\*\*\*\*\*\*\*\*\*\*\*Method 2: Stack method with o(n) space complexity \*\*\*\*\*\*\*\*\*\*\*\*/ /\* \*\*\*\*\*\*\*\*\*\*\*\*common stack setting with consideration on corner cases \*\*\*\*\*\*\*\*\*\*\*\*/  public int longestValidParentheses(String s) {  int max = 0;  Deque<Integer> stack = new ArrayDeque<>();  for (int i = 0; i < s.length(); i++) {  char c = s.charAt(i);  if (c == '(') {  stack.offerLast(i);  } else {  if (!stack.isEmpty() && s.charAt(stack.peekLast()) == '(') {   stack.pollLast();  int left = stack.isEmpty() ? -1 : Math.abs(stack.peekLast());  max = Math.max(max, i - left); // (peekLast(), i]  } else { // don't forgot to add ')' if necessary  stack.offerLast(i);  }  }  }  return max;  } |
| --- |

# 33 Search in Rotated Sorted Array

Java code by Tang

public int search(int[] nums, int target) { // code by Tang

if (nums == null || nums.length == 0) {

return -1;

}

int lo = 0;

int hi = nums.length - 1;

int mid;

while (lo < hi - 1) {

mid = lo + (hi - lo) / 2;

// if you use double side binary search protocol

if (nums[lo] == target) {

return lo;

} else if (nums[hi] == target) {

return hi;

} else if (nums[mid] == target) {

return mid;

}

// boundary must be check to avoid error, e.g. [1,2,3], target 1

if (nums[mid] > nums[lo]) { // left mono-increasing

if (target > nums[lo] && target < nums[mid]) {

hi = mid;

} else {

lo = mid;

}

} else {

if (target > nums[mid] && target < nums[hi]) {

lo = mid;

} else {

hi = mid;

}

}

}

if (nums[lo] == target) {

return lo;

}

if (nums[hi] == target) {

return hi;

}

return -1;

}

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

//(Zhang)

public int[] searchRange(int[] A, int target) {

int start = firstGreaterEqual(A, target);

if (start == A.length || A[start] != target) {

return new int[]{-1, -1};

}

int second = firstGreaterEqual(A, target + 1) - 1;

return new int[]{start, second};

}

//find the first number that is greater than or equal to target.

//could return A.length if target is greater than A[A.length-1].

//actually this is the same as lower\_bound in C++ STL.

private int firstGreaterEqual(int[] A, int target) {

int low = 0, high = A.length;

while (low < high) {

int mid = low + ((high - low) >> 1);

//low <= mid < high

if (A[mid] < target) {

low = mid + 1;

} else {

//should not be mid-1 when A[mid]==target.

//could be mid even if A[mid]>target because mid<high.

high = mid;

}

}

return low;

}

}

// double-sided binary search by Tang

public int[] searchRange(int[] nums, int target) {

int[] res = new int[]{-1, -1};

if (nums == null || nums.length == 0) {

return res;

}

int left = firstBS(nums, target, 0, nums.length - 1);

if (left == -1) {

return res;

}

int right = lastBS(nums, target, left, nums.length - 1);

return new int[]{left, right};

}

private int firstBS(int[] array, int target, int lo, int hi) {

int mid;

while (lo < hi - 1) {

mid = lo + (hi - lo) / 2;

if (array[mid] >= target) {

hi = mid;

} else {

lo = mid;

}

}

if (array[lo] == target) {

return lo;

}

if (array[hi] == target) {

return hi;

}

return -1;

}

private int lastBS(int[] array, int target, int lo, int hi) {

int mid;

while (lo < hi - 1) {

mid = lo + (hi - lo) / 2;

if (array[mid] <= target) {

lo = mid;

} else {

hi = mid;

}

}

if (array[hi] == target) {

return hi;

}

if (array[lo] == target) {

return lo;

}

return -1;

}

# 35 Search Insert Position

(luo)

/\*

Binary Search 找target，或者找应该插入的位置

\*/

class Solution {

public int searchInsert(int[] nums, int target) {

// pre-process提前处理target小于最小值，大于最大值的情况

if (target < nums[0]) {

return 0;

}

if (target > nums[nums.length - 1]) {

return nums.length;

}

int left = 0;

int right = nums.length - 1;

while (left + 1 < right) {

int mid = left + (right - left) / 2;

if (nums[mid] == target) {

return mid;

} else if (nums[mid] < target) {

left = mid;

} else {

right = mid;

}

}

if (target == nums[left]) return left;

if (target == nums[right]) return right;

return left + 1;

}

}

# 36 Valid Sudoku

// code by Luo, 分别看行，列，和每一个box

class Solution {

public boolean isValidSudoku(char[][] board) {

// check each row

for (int i = 0; i < 9; i++) {

Set<Character> set = new HashSet<>();

for (int j = 0; j < 9; j++) {

if (board[i][j] != '.' && !set.add(board[i][j])) {

return false;

}

}

}

// check each column

for (int i = 0; i < 9; i++) {

Set<Character> set = new HashSet<>();

for (int j = 0; j < 9; j++) {

if (board[j][i] != '.' && !set.add(board[j][i])) {

return false;

}

}

}

// check each box

for (int i = 0; i < 9; i++) { // 大格

Set<Character> set = new HashSet<>();

for (int j = 0; j < 9; j++) { // 小格的index

if (board[3 \* (i / 3) + j / 3][3 \* (i % 3) + j % 3] != '.' && !set.add(board[3 \* (i / 3) + j / 3][3 \* (i % 3) + j % 3])) {

return false;

}

}

}

return true;

}

}

# 39 Combination Sum

(Peiyu) DFS: find all combinations

Example: [2, 3, 5] target 8

leve = target / min(nums)

N = len(nums)

TC: a loose upper bound O(N^level) - total nodes of a N-arry tree of height level

SC: O(level) - dfs stack depth

8 (target)

(2) / (3) | \(5)

6 5 3

(2) / (3)| \(5) (3)/ \(5) \(5)

4 3 1 2 0

(2) /|(3)\(5) |(3) |(5)

2 1 -1 0

(2)/

0

public List<List<Integer>> combinationSum(int[] nums, int target) {

List<List<Integer>> res = new ArrayList();

helper(nums, target, res, new ArrayList(), 0);

return res;

}

private void helper(int[] nums, int target, List<List<Integer>> res, List<Integer> path, int idx) {

if (target == 0) {

res.add(new ArrayList(path));

}

for (int i = idx; i < nums.length; i++) {

if (nums[i] > target) continue;

path.add(nums[i]);

helper(nums, target - nums[i], res, path, i); // note: don’t increase index as nums[i] can be reused

path.remove(path.size() - 1);

}

}

# 40 Combination Sum II

(Peiyu) Difference is nums can contain **duplicate** numbers and each number can be **used only once**

1. **Sort array [9,1,1,1,3] -> [0,0,1,3,9]**
2. **If a number is the same as previous one, then no need to explore the combination starting from it**

**Ex. [1,1,1,3] target = 8**

**First 1, idx = 0, dfs will explore all possibilities starting from 0, i.e, based on [1,1,1,3]**

**Second 1, idx = 1, dfs will explore all possibilities starting from 1, i.e, based on [1,1,3]**

**Whatever combination found in [1,1,3] can be found with [1,1,1,3], so save it**

N = len(nums)

TC: Same as subset, each element, either pick or not pick it, total combinations O(2^n)

SC: O(N) for stack

8 (target)

(2) / (3) | \(5)

6 5 3

(3) / \(5) |(5)

3 1 0

(5) /

-2

public List<List<Integer>> combinationSum2(int[] nums, int target) {

List<List<Integer>> res = new ArrayList();

Arrays.sort(nums);

helper(res, new ArrayList(), nums, target, 0);

return res;

}

private void helper(List<List<Integer>> res, List<Integer> path, int[] nums, int target, int idx) {

if (target == 0) {

res.add(new ArrayList(path));

return;

}

if (idx >= nums.length) {

return;

}

for (int i = idx; i < nums.length; i++) {

if (nums[i] > target) continue; //early termination

if (i > idx && nums[i] == nums[i - 1]) continue;

path.add(nums[i]);

helper(res, path, nums, target - nums[i], i + 1); //can't reuse

path.remove(path.size() - 1);

}

}

# 41 First Missing Positive

/\*

step1: 看1是否在，如果不在，直接return 1

Step2: 把负数和0全部改成1， 把大于array.length的数也都改成1，因为答案不会是他们

Step3: 把出现过的数，的index那边标成负数，这样一圈下来，所以都被标负了

Step4: 最小的那个没有被标负的index，就是缺的数

\*/

// code by Luo

class Solution {

public int firstMissingPositive(int[] nums) {

int n = nums.length;

// base case

int contains = 0;

for (int i = 0; i < n; i++) {

if (nums[i] == 1) {

contains++;

break;

}

}

if (contains == 0) {

return 1;

}

for (int i = 0; i < n; i++) {

if (nums[i] <= 0 || nums[i] > n) {

nums[i] = 1;

}

}

for (int i = 0; i < n; i++) {

int val = Math.abs(nums[i]); // 这个位置的value

if (val == n) {

nums[0] = -Math.abs(nums[0]);

} else {

nums[val] = -Math.abs(nums[val]);

}

}

for (int i = 1; i < n; i++) {

if (nums[i] > 0) {

return i;

}

}

if (nums[0] > 0) {

return n;

}

return n + 1;

}

}

# 42. Trapping Rain Water

| /\* \*\*\*\*\*\*\*\*\*\*\*\*Method 1: Two pointer method with o(1) space complexity \*\*\*\*\*\*\*\*\*\*\*\*/  public int trap(int[] height) {  // time : O(n)  // space : O(1)  if (height.length<3) return 0;   int left = 0, right = height.length-1;   int leftMax=0, rightMax=0;   int ans = 0;   while (left < right) {  leftMax = Math.max(leftMax, height[left]);   rightMax = Math.max(rightMax, height[right]);  if (leftMax < rightMax) {  ans += Math.max(0, leftMax-height[left]);   left++;   } else {  ans += Math.max(0, rightMax-height[right]);   right--;   }  }  return ans;   }  /\* \*\*\*\*\*\*\*\*\*\*\*\*Method 2: Stack method with o(n) space complexity \*\*\*\*\*\*\*\*\*\*\*\*/  public int trap2(int[] height) {  // time : O(n)  // space : O(n)  if (height == null || height.length < 2) return 0;    Stack<Integer> stack = new Stack<>();  int water = 0, i = 0;  while (i < height.length) {  if (stack.isEmpty() || height[i] <= height[stack.peek()]) {  stack.push(i++);  } else {  int pre = stack.pop(); // (pre, i)  if (!stack.isEmpty()) {  // find the smaller height between the two sides  int minHeight = Math.min(height[stack.peek()], height[i]);  // calculate the area  water += (minHeight - height[pre]) \* (i - stack.peek() - 1);  }  }  }  return water;  } |
| --- |

# 43. Multiply Strings

| /\* \*\*\*\*\*\*\*\*\*\*\*\* Method 2: Binary string multiply code added and tested in leetcode \*\*\*\*\*\*\*\*\*\*\*\*/  public String multiply(String num1, String num2) {  int m = num1.length(), n = num2.length();  String num1Binary = Integer.toBinaryString(Integer.parseInt(num1));  String num2Binary = Integer.toBinaryString(Integer.parseInt(num2));  String binRes = multiplyBinary(num1Binary, num2Binary);  System.out.print("String Binary Mutiply : " + binRes + ", ");  int[] pos = new int[m + n];    for(int i = m - 1; i >= 0; i--) {  for(int j = n - 1; j >= 0; j--) {  int mul = (num1.charAt(i) - '0') \* (num2.charAt(j) - '0');   int p1 = i + j, p2 = i + j + 1;  int sum = mul + pos[p2];    pos[p1] += sum / 10;  pos[p2] = (sum) % 10;  }  }     StringBuilder sb = new StringBuilder();  for(int p : pos) if(!(sb.length() == 0 && p == 0)) sb.append(p);  String res = sb.length() == 0 ? "0" : sb.toString(); // special cases for empty  System.out.println( Integer.parseInt(binRes, 2) );  return res;  }   public String multiplyBinary(String num1, String num2) { // binary string multiply !!!  int m = num1.length(), n = num2.length();  int[] pos = new int[m + n];    for(int i = m - 1; i >= 0; i--) {  for(int j = n - 1; j >= 0; j--) {  int mul = (num1.charAt(i) - '0') \* (num2.charAt(j) - '0');   int p1 = i + j, p2 = i + j + 1;  int sum = mul + pos[p2];    pos[p1] += sum / 2;  pos[p2] = (sum) % 2;  }  }     StringBuilder sb = new StringBuilder();  for(int p : pos) if(!(sb.length() == 0 && p == 0)) sb.append(p);  return sb.length() == 0 ? "0" : sb.toString(); // special cases for empty  } |
| --- |

# 44. Wildcard Matching

//(zhang) recursion

class Solution {

public HashMap<String, Boolean> memoMap = new HashMap<>();

public boolean isMatch(String s, String p) {

if (s == null) return false;

boolean match = dfs(s, 0, p, 0);

return match;

}

private boolean dfs(String s, int i, String p, int j) {

String key = i + "#" + j;

if (memoMap.containsKey(key)) {

return memoMap.get(key);

}

boolean match = false;

if (j == p.length() && i == s.length())

return true;

if (i > s.length())

return false;

if (i < s.length() && j < p.length() && (s.charAt(i) == p.charAt(j) || p.charAt(j) == '?')) {

match = dfs(s, i + 1, p, j + 1);

}

if (j < p.length() && p.charAt(j) == '\*') {

match = dfs(s, i + 1, p, j) || dfs(s, i, p, j + 1);

}

memoMap.put(key, match);

return match;

}

}

//(Luo) DP

class Solution {

public boolean isMatch(String s, String p) {

// s 可能为空，且只包含从 a-z 的小写字母。

// p 可能为空，且只包含从 a-z 的小写字母，以及字符 ? 和 \*。

int lens = s.length();

int lenp = p.length();

char[] sArray = s.toCharArray();

char[] pArray = p.toCharArray();

boolean[][] dp = new boolean[lens + 1][lenp + 1];

dp[0][0] = true;

//当s的长度为0的情况

for (int i = 1; i <= lenp; i++) {

dp[0][i] = pArray[i - 1] == '\*' ? dp[0][i - 1] : false;

}

for (int i = 1; i <= lens; i++) {

for (int j = 1; j <= lenp; j++) {

char sc = sArray[i - 1];

char pc = pArray[j - 1];

if (sc == pc || pc == '?') {

dp[i][j] = dp[i - 1][j - 1];// 当前字符匹配,当前s[0..i-1]p[0..j-1]是否匹配取决于之前dp[i - 1][j - 1]

} else {

if (pc == '\*') {

if (dp[i][j - 1] || dp[i - 1][j - 1] || dp[i - 1][j]) { // 这里是填表格的关键

dp[i][j] = true;

}

}

}

}

}

return dp[lens][lenp];

}

}

/\*

i

0 1 2 3 4 5

0 a c d c b

0 t f f f f f

1 a f t f f f f

2 \* f t t t t t

3 c f f t f t f

4 ? f f f t f t

5 b f f f f f f

0 a a

0 t f f

\* t

\*/

# 55. Jump Game

(Luo)

// DP, 从后往前，dp

class Solution {

public boolean canJump(int[] nums) {

if (nums == null || nums.length <= 1) return true;

int n = nums.length;

boolean[] canJump = new boolean[n];

canJump[n - 1] = true;

for (int i = n - 2; i >= 0; i--) {

if (i + nums[i] >= n - 1) {

canJump[i] = true;

//continue;

} else {

for (int j = 1; j <= nums[i]; j++) {

if (canJump[i + j]) {

canJump[i] = true;

break;

}

}

}

}

return canJump[0];

}

}

//TC: O(n \* max value of items)

//SC: O(n)

# 45. Jump Game II

(luo)

// method 1: DP O(n^2)

// method 2: 类似BFS O（n）

// dp

class Solution {

public int jump(int[] nums) {

int n = nums.length;

int[] dp = new int[n]; // dp[i] represents the min steps to reach the last index from index i

Arrays.fill(dp, -1);

dp[n - 1] = 0;// base case

for (int i = n - 2; i >= 0; i--) {

// 直接能到

if (nums[i] + i >= n) {

dp[i] = 1;

} else {

//不能直接到

//找它所能到的所有的地方，最近的，加1，就是dp[i]

for (int j = 1; j <= nums[i]; j++) {

if (dp[j + i] == -1) {

continue;

}

if (dp[i] == -1) {

dp[i] = dp[i + j] + 1;

} else {

dp[i] = Math.min(dp[i], dp[j + i] + 1);

}

}

}

}

return dp[0];

}

}

// 类似BFS，O(n)

class Solution {

public int jump(int[] nums) {

int jumps = 0;

int currentJumpEnd = 0;

int farthest = 0;

for (int i = 0; i < nums.length - 1; i++) {

// we continuously find the how far we can reach in the current jump

farthest = Math.max(farthest, i + nums[i]);

// if we have come to the end of the current jump,

// we need to make another jump

if (i == currentJumpEnd) {

jumps++;

currentJumpEnd = farthest;

}

}

return jumps;

}

}

# 1306. Jump Game III

(luo)

/\*

graph的最短路径问题

从start，到length -1的位置，最短几步可以到达。

BFS1求最短路径，用queue和visited

TC: O(V + E) -> O(n) where n is the length of input, each node visited once

SC: O(n) for the queue

0 1 2 3 4 5 6

[4,2,3,0,3,1,2]

i

DFS 也可以

\*/

class Solution {

public boolean canReach(int[] arr, int start) {

Queue<Integer> queue = new ArrayDeque<>();

Set<Integer> visited = new HashSet<>();

queue.offer(start);

visited.add(start);

while (!queue.isEmpty()) {

int cur = queue.poll();

visited.add(cur);

int value = arr[cur];

if (value == 0) {

return true;

}

int right = cur + value;

int left = cur - value;

if (right < arr.length && !visited.contains(right)) {

queue.offer(right);

}

if (left >= 0 && !visited.contains(left)) {

queue.offer(left);

}

}

return false;

}

}

# 43 Multiply Strings

(Lynn)

class Solution {

public String multiply(String num1, String num2) {

int m = num1.length();

int n = num2.length();

int[] res = new int[m + n];

// multiply each digit from two string, put the result in array

for (int i = m - 1; i >= 0; i--) {

for (int j = n - 1; j >= 0; j--) {

int digit1 = num1.charAt(i) - '0';

int digit2 = num2.charAt(j) - '0';

int value = digit1 \* digit2;

res[i + j + 1] += value % 10;

res[i + j] += value / 10;

}

}

// deal with digit overflow

for (int i = res.length - 1; i >= 0; i--) {

int val = res[i];

if (val >= 10) {

res[i] = val % 10;

if (i > 0) {

res[i - 1] += val / 10;

}

}

}

// convert int array into string to represent integer, avoid leading zero

StringBuilder sb = new StringBuilder();

for (int i = 0; i < res.length; i++) {

if (sb.length() == 0 && res[i] == 0) {

continue;

}

sb.append(res[i]);

}

return sb.length() == 0? "0" : sb.toString();

}

}

# 

# 792. Number of Matching Subsequences

public int numMatchingSubseq(String s, String[] words) {

**//Solution1 : HashMap to maintain each char and its index**

// Map<Character, List<Integer>> map = new HashMap<>();

// for (int i = 0; i < s.length(); i++) {

// if (!map.containsKey(s.charAt(i))) {

// map.put(s.charAt(i), new ArrayList<>());

// }

// //index of letter should be unique and sorted in acsending order

// map.get(s.charAt(i)).add(i);

// }

// int count = 0;

// for (String word : words) {

// int preIndex = -1;

// int i = 0;

// for (; i < word.length(); i++) {

// char c = word.charAt(i);

// if (!map.containsKey(c)) break;

// List<Integer> indice = map.get(c);

// boolean foundValideIndex = false;

// for (int k = 0; k < indice.size(); k++) {

// if (indice.get(k) > preIndex) {

// preIndex = indice.get(k);

// foundValideIndex = true;

// break;

// }

// }

// if (!foundValideIndex) break;

// }

// if (i == word.length()) count++;

// }

// return count;

**//Solution2 : optimization**

if (s == null || s.length() == 0) {

return 0;

}

int count = 0;

List<int[]>[] map = new List[26];

for (int i = 0; i < 26; i++) {

map[i] = new ArrayList<>();

}

for (int i = 0; i < words.length; i++) {

String word = words[i];

char c = word.charAt(0);

**//position in map should be c - 'a', position in words list should be i**

map[c - 'a'].add(new int[] {i, 1});

}

for (char c : s.toCharArray()) { //m

List<int[]> listOfIndexInWords = map[c - 'a'];

map[c - 'a'] = new ArrayList<>();

for (int[] index : listOfIndexInWords) { //

int indexOfWords = index[0];

int indexOfNextLetter = index[1];

if (indexOfNextLetter == words[indexOfWords].length()) {

count++;

} else {

map[words[indexOfWords].charAt(indexOfNextLetter) - 'a'].add(new int[] {indexOfWords, indexOfNextLetter + 1});

}

}

}

return count;

}

# 51 N-Queens

(Luo)

/\*

TC: O(n \* n!) n for isValid, n! for DFS

n

n(n - 1)

n(n - 1)(n - 2)

..

n!

\*/

class Solution {

public List<List<String>> solveNQueens(int n) {

char[][] board = getBoard(n);

List<List<String>> res = new ArrayList<>();

helper(board, n, 0, res);

return res;

}

private char[][] getBoard(int n) {

char[][] board = new char[n][n];

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

board[i][j] = '.';

}

}

return board;

}

private void helper(char[][] board, int n, int row, List<List<String>> res) {

if (row == n) {

res.add(convert(board));

return;

}

for (int col = 0; col < n; col++) {

if (isValid(board, row, col, n)) {

board[row][col] = 'Q';

helper(board, n, row + 1, res);

board[row][col] = '.';

}

}

}

private List<String> convert(char[][] board) {

List<String> res = new ArrayList<>();

for (int i = 0; i < board.length; i++) {

StringBuilder sb = new StringBuilder();

for (int j = 0; j < board.length; j++) {

sb.append(board[i][j]);

}

res.add(sb.toString());

}

return res;

}

//根据现有的board，再放[row, col]这个位置是否valid

private boolean isValid(char[][] board, int row, int col, int n) {

for (int i = 0; i <= row; i++) {

//same col

if (board[i][col] == 'Q') {

return false;

}

//same diagonal

if ((row - i) >= 0 && (col - i) >= 0 && board[row - i][col - i] == 'Q') {

return false;

}

//same anti-diagonal

if ((row - i) >= 0 && (col + i) < n && board[row - i][col + i] == 'Q') {

return false;

}

}

return true;

}

}

# 48 Rotate Image

// code by Luo, 分层做交换，每次换4个

class Solution {

public void rotate(int[][] matrix) {

int n = matrix.length;

int offset = n / 2;

for (int i = 0; i < offset; i++) { // i represents offset, which level

int left = i;

int right = n - i - 2;

for (int j = left; j <= right; j++) { // j represents from left to right, from first to last element

int temp = matrix[left][j]; // upper left position

matrix[left][j] = matrix[n - 1 - j][left];

matrix[n - 1 - j][left] = matrix[n - 1 - left][n - 1 - j];

matrix[n - 1- left][n - 1 - j] = matrix[j][n - 1- left];

matrix[j][n - 1 - left] = temp;

}

}

}

}

# 49 Group Anagrams

# 50 Pow(x, n)

# 

# 53 Maximum Subarray

# 54 Spiral Matrix

| public List<Integer> spiralOrder(int[][] matrix) {  List ans = new ArrayList();  if (matrix.length == 0)  return ans;  int r1 = 0, r2 = matrix.length - 1;  int c1 = 0, c2 = matrix[0].length - 1;  while (r1 <= r2 && c1 <= c2) {  for (int c = c1; c <= c2; c++) ans.add(matrix[r1][c]);  for (int r = r1 + 1; r <= r2; r++) ans.add(matrix[r][c2]);  if (r1 < r2 && c1 < c2) {  for (int c = c2 - 1; c > c1; c--) ans.add(matrix[r2][c]);  for (int r = r2; r > r1; r--) ans.add(matrix[r][c1]);  }  r1++;  r2--;  c1++;  c2--;  }  return ans;  } |
| --- |

36 Valid Sudoku

(Aye)

public boolean isValidSudoku(char[][] board) {

for (int i = 0; i < 9; i++) {

Set<Character> dedupRow = new HashSet<>();

Set<Character> dedupCol = new HashSet<>();

Set<Character> dedupCube = new HashSet<>();

for (int j = 0; j < 9; j++) {

if (board[i][j] != '.' && !dedupRow.add(board[i][j])) {

return false;

}

if (board[j][i] != '.' && !dedupCol.add(board[j][i])) {

return false;

}

if (board[j/3 + i/3\*3][j%3 + i%3\*3] != '.' && !dedupCube.add(board[j/3 + i/3\*3][j%3 + i%3\*3])) {

// System.out.print((i/3 + j/3) + " " + (j%3 + j%3));

return false;

}

// System.out.println((j/3 + i/3\*3) + " " + (j%3 + i%3\*3) + " ");

}

// System.out.println("-----");

}

return true;

}

# 1849. Splitting a String Into Descending Consecutive Values

// （wu） O（n^2） ??

def splitString(self, s: str) -> bool:

n = len(s)

start = 0

# 枚举第一个子字符串对应的初始值

# 第一个子字符串不能包含整个字符串

for i in range(n - 1):

start = 10 \* start + int(s[i])

# 循环验证当前的初始值是否符合要求

pval = start

cval = 0

cidx = i + 1

for j in range(i + 1, n):

if pval == 1:

# 如果上一个值为 1，那么剩余字符串对应的数值只能为 0

if all(s[k] == '0' for k in range(cidx, n)):

return True

else:

break

cval = 10 \* cval + int(s[j])

if cval > pval - 1:

# 不符合要求，提前结束

break

elif cval == pval - 1:

if j + 1 == n:

# 已经遍历到末尾

return True

pval = cval

cval = 0

cidx = j + 1

return False

//(Peiyu) 上面的java版本

public boolean splitString(String s) {

int n = s.length();

long start = 0;

// 枚举第一个子字符串对应的初始值

// 第一个子字符串不能包含整个字符串

for (int i = 0; i < n; i++) { //O(N)

start = 10 \* start + s.charAt(i) - '0';

// 循环验证当前的初始值是否符合要求

long prev = start;

long current = 0;

for (int j = i + 1; j < n; j++) { //O(N)

if (prev == 1) {

// 如果上一个值为 1，那么剩余字符串对应的数值只能为 0

if (s.charAt(j) != '0') break;

else if (j == n - 1) return true;

else continue;

}

current = 10 \* current + s.charAt(j) - '0';

if (current > prev - 1) {

// 不符合要求，提前结束

break;

} else if (current == prev - 1) {

if (j + 1 == n)

// 已经遍历到末尾

return true;

prev = current;

current = 0;

}

}

}

return false;

}

(Peiyu) DFS

//1|0009998

// 0|009998

// 0|09998 00|9998 009|998 0099|98 00999|8 009998

// 00|09998

// 0|9998 09|998 099|98 0999|8 09998

// 000|9998

// 9|998 99|98 999|8 9998

// 0009|998 00099|98 000999|8 0009998

//10|009998

// 0|09998 00|9998 009|998 0099|98 00999|8 009998

// 9|98 99|8 998

//100|09998

// 0|9998 09|998 099|98 -> found 1, return true

public boolean splitString(String s) {

return helper(s, null);

}

private boolean helper(String s, Long prev) {

long current = 0;

for (int i = 0; i < s.length(); i++) {

current = current \* 10 + s.charAt(i) - '0';

if (prev == null) {

if (helper(s.substring(i + 1), current)){

return true;

}

} else {

if (prev == current + 1 && (i == s.length() - 1 || helper(s.substring(i + 1), current))) {

return true;

}

}

}

return false;

}

# 1110 Delete Nodes And Return Forest

（luo）

// TC: O(n)

// SC: O(height)

class Solution {

public List<TreeNode> delNodes(TreeNode root, int[] to\_delete) {

Set<Integer> set = new HashSet<>();

for (int i : to\_delete) {

set.add(i);

}

List<TreeNode> res = new ArrayList<>();

// if (!set.contains(root.val)) {

// res.add(root); //先单独判断root

// }

dfs(root, true, set, res);

return res;

}

//这个dfs，return的是自己，但是是在把自己和set对比之后，把新的自己return回去,就是自己或者null

private TreeNode dfs(TreeNode node, boolean isRoot, Set<Integer> set, List<TreeNode> res) {

if (node == null) {

return null;

}

node.left = dfs(node.left, false, set, res);

node.right = dfs(node.right, false, set, res);

if (set.contains(node.val)) {

if (node.left != null) {

res.add(node.left); //

}

if (node.right != null) {

res.add(node.right);

}

return null;

}

if (isRoot) { //把root的情况在这里cover了

res.add(node);

}

return node;

}

}

//（huang） 不需要把root单独出来，思路比较清晰的方法

class Solution:

def delNodes(self, root: TreeNode, to\_delete: List[int]) -> List[TreeNode]:

to\_delete = set(to\_delete)

ans = []

self.findforest(root, False, to\_delete, ans)

return ans

def findforest(self, root, parent\_exist, to\_delete, ans):

if root == None:

return None

if root.val in to\_delete:

root.left = self.findforest(root.left, False, to\_delete, ans)

root.right = self.findforest(root.right, False, to\_delete, ans)

return None

else:

if not parent\_exist:

ans.append(root)

root.left = self.findforest(root.left, True, to\_delete, ans)

root.right = self.findforest(root.right, True, to\_delete, ans)

return root

# 56 Merge Intervals

<https://leetcode.com/list/9rjh2ka1/> 扫描线几个高频题list

// pq sort

public int[][] merge(int[][] intervals) {

if (intervals.length <= 1)

return intervals;

// Sort by ascending starting point

Arrays.sort(intervals, (i1, i2) -> Integer.compare(i1[0], i2[0]));

List<int[]> result = new ArrayList<>();

int[] newInterval = intervals[0];

result.add(newInterval);

for (int[] interval : intervals) {

if(interval[0] <= newInterval[1]) // Overlapping intervals, move the end if need

newInterval[1] = Math.max(newInterval[1], interval[1]); // [1, 6]

else { // Disjoint intervals, add the new interval to the list

newInterval = interval;

result.add(newInterval);

}

}

return result.toArray(new int[result.size()][]);

}

//(luo) 扫描线的方法

/\*

1 1 -1. -1. 1. -1. 1. -1

1 2 3 6 8 10 15. 18

---------

1. 2. 1. 0

\*/

class Solution {

public int[][] merge(int[][] intervals) {

List<int[]> result = new ArrayList<>();

List<Boundary> boundaries = new ArrayList<>();

for (int[] interval : intervals) {

boundaries.add(new Boundary(interval[0], 1));

boundaries.add(new Boundary(interval[1], -1));

}

Collections.sort(boundaries, new MyComparator());

int isMatched = 0;

int left = 0;

int right = 0;

for (Boundary boundary : boundaries) {

if (isMatched == 0) {

left = boundary.num;

}

isMatched += boundary.type;

if (isMatched == 0) {

right = boundary.num;

result.add(new int[] {left, right}); //如何用单独的数字变成list

}

}

return result.toArray(new int[result.size()][]); // 如何用list of int[] 变成int[][]

}

class Boundary {

int num;

int type; // 1 is start, -1 is end

public Boundary(int num, int type) {

this.num = num;

this.type = type;

}

}

public class MyComparator implements Comparator<Boundary> {

@Override

public int compare(Boundary one, Boundary two) {

if (one.num == two.num) {

return one.type > two.type ? -1 : 1;

}

return one.num < two.num ? -1: 1;

}

}

}

# 57 Insert Interval

解法1：标准的插入

将intervals的所有元素全部遍历一遍，可以想见会依次遇到这些情况：

intervals[i]如果整体都在newInterval之前，则可以直接将intervals[i]加入results;

intervals[i]如果和newInterval有交集，则与之融合生成新的newInterval；这样的融合可能会有若干次；

intervals[i]如果整体都在newInterval之后，则将newInterval（可能经历了融合）加入results，并把未遍历的intervals[i]也都加入results;

public int[][] insert(int[][] intervals, int[] newInterval) {

LinkedList<int[]> result = new LinkedList<int[]>();

int i = 0;

// add all the intervals ending before newInterval starts

while (i < intervals.length && intervals[i][1] < newInterval[0])

result.add(intervals[i++]);

// merge all overlapping intervals to one considering newInterval

while (i < intervals.length && intervals[i][0] <= newInterval[1]) {

newInterval[0] = Math.min(newInterval[0], intervals[i][0]);

newInterval[1] = Math.max(newInterval[1], intervals[i][1]);

i++;

}

result.add(newInterval); // add the union of intervals we got

// add all the rest

while (i < intervals.length)

result.add(intervals[i++]);

return result.toArray(new int[result.size()][2]);

}

# 60 Permutation Sequence

# 62 Unique Paths

# 63 Unique Paths II

# 65 Valid Number

(Aye)

| class Solution {  public boolean isNumber(String s) {  s = s.trim();  boolean pointSeen = false;  boolean eSeen = false;  boolean numberSeen = false;  for(int i=0; i<s.length(); i++) {  if('0' <= s.charAt(i) && s.charAt(i) <= '9') {  numberSeen = true;  } else if(s.charAt(i) == '.') {  if(eSeen || pointSeen)  return false;  pointSeen = true;  } else if(s.charAt(i) == 'e' || s.charAt(i) == 'E') {  if(eSeen || !numberSeen)  return false;  numberSeen = false;  eSeen = true;  } else if(s.charAt(i) == '-' || s.charAt(i) == '+') {  if(i != 0 && s.charAt(i-1) != 'e')  return false;  } else  return false;  }  return numberSeen;  } } |
| --- |

# 66 Plus One

(luo)

class Solution {

public int[] plusOne(int[] digits) {

int n = digits.length;

boolean addDigit = false;

for (int i = digits.length - 1; i >= 0; i--) {

digits[i] += 1;

if (digits[i] <= 9) {

addDigit = false;

break;

} else {

digits[i] = digits[i] % 10;

addDigit = true;

}

}

if (!addDigit) {

return digits;

}

int[] res = new int[n + 1];

res[0] = 1;

for (int i = 1; i < res.length; i++) {

res[i] = digits[i - 1];

}

return res;

}

}

# 67 Add Binary

(luo)

/\*

11:19

从右往前加

用StringBuilder往上粘，最后reverse sb

\*/

class Solution {

public String addBinary(String a, String b) {

int i = a.length() - 1;

int j = b.length() - 1;

int sum = 0;

StringBuilder sb = new StringBuilder();

while (i >= 0 || j >= 0 || sum != 0) {

if (i >= 0) {

sum += a.charAt(i) - '0';

i--;

}

if (j >= 0) {

sum += b.charAt(j) - '0';

j--;

}

sb.append(sum % 2);

sum = sum / 2;

}

return sb.reverse().toString();

}

}

68. Text Justification

| class Solution { // ood,   // 1. find out line, line function -> list of <string> desired format -> append format  public List<String> fullJustify(String[] words, int L) {  List<String> lines = new ArrayList<String>();    int index = 0; // cur word index  while (index < words.length) {  int count = words[index].length(); // # char count in cur line (most compact)  int last = index + 1; // next word  while (last < words.length) {  if (words[last].length() + count + 1 > L) break;  count += words[last].length() + 1; // words + '\_'  last++;  }    StringBuilder builder = new StringBuilder();  int diff = last - index - 1; // [index, last), spaces # between words, #words - 1;  // if last line or number of words in the line is 1, left-justified  if (last == words.length || diff == 0) {  for (int i = index; i < last; i++) {  builder.append(words[i] + " ");  }  builder.deleteCharAt(builder.length() - 1);  for (int i = builder.length(); i < L; i++) {  builder.append(" ");  }  } else {  // middle justified  int spaces = (L - count) / diff; // extra space number for every space  int r = (L - count) % diff; // extra space on the right  for (int i = index; i < last; i++) {  builder.append(words[i]);  if (i < last - 1) {  for (int j = 0; j <= (spaces + ((i - index) < r ? 1 : 0)); j++) { //?  builder.append(" ");  }  }  }  }  lines.add(builder.toString());  index = last;  }      return lines;  }  } |
| --- |

71. Simplify Path

| class Solution {  public String simplifyPath(String path) {   // Initialize a stack  Deque<String> stack = new ArrayDeque<String>();  String[] components = path.split("/");   // Split the input string on "/" as the delimiter  // and process each portion one by one  for (String directory : components) {   // A no-op for a "." or an empty string  if (directory.equals(".") || directory.isEmpty()) {  continue;  } else if (directory.equals("..")) {   // If the current component is a "..", then  // we pop an entry from the stack if it's non-empty  if (!stack.isEmpty()) {  stack.pollLast();  }  } else {   // Finally, a legitimate directory name, so we add it  // to our stack  stack.offerLast(directory);  }  }   // Stich together all the directory names together  StringBuilder result = new StringBuilder();  for (String dir : stack) {  result.append("/");  result.append(dir);  }  return result.length() > 0 ? result.toString() : "/" ;  } } |
| --- |

# 767 Reorganize String

（luo）

//TC: O(n), SC: O(n)

//先数出个数最多的字母，隔着能放下的话，剩下的挨着放就可以。

class Solution {

public String reorganizeString(String s) {

// step 1.1 : load char to int array

int[] count = new int[26];

for (char c : s.toCharArray()) {

count[c - 'a']++;

}

// step 1.2 : find the most count char

int maxIndex = 0;

for (int i = 1; i < count.length; i++) {

if (count[i] > count[maxIndex]) {

maxIndex = i;

}

}

// step 2. put the max index char to result array, 间隔放

if (count[maxIndex] > (s.length() + 1) / 2) {

return "";

}

char[] res = new char[s.length()];

int index = 0;

// 放最多的字母

while (count[maxIndex] > 0) {

res[index] = (char) (maxIndex + 'a');

index += 2;

count[maxIndex]--;

}

for (int i = 0; i < count.length; i++) {

while (count[i] > 0) {

if (index >= res.length) {

index = 1;

}

res[index] = (char) (i + 'a');

index += 2;

count[i]--;

}

}

return new String(res);

}

}

# 209. Minimum Size Subarray Sum

(luo)

/\*

subarray sum, 本题采用 sliding window

2 3 1 2 4 3

s

f

T: O(n)

S: O(1)

\*/

class Solution {

public int minSubArrayLen(int target, int[] nums) {

int sum = 0; // sliding window sum

int min = Integer.MAX\_VALUE;

int left = 0; // left pointer

for (int i = 0; i < nums.length; i++) { // right pointer

sum += nums[i];

while (left <= i && sum >= target) {

min = Math.min(min, i - left + 1);

sum -= nums[left];

left++;

}

}

return min == Integer.MAX\_VALUE ? 0 : min;

}

}

366. Find Leaves of Binary Tree

| class Solution {  public List<List<Integer>> findLeaves(TreeNode root) {  List<List<Integer>> res = new ArrayList<>();  getHeight(root, res);  return res;  }  private int getHeight(TreeNode root, List<List<Integer>> res) {  if (root == null) {  return -1;  }  int left = getHeight(root.left, res);  int right = getHeight(root.right, res);  int curHeight = Math.max(left, right) + 1;  if (curHeight == res.size()) {  res.add(new ArrayList<Integer>());  }  res.get(curHeight).add(root.val);  // root.left = null;  // root.right = null;  return curHeight;  } } |
| --- |

# 380 Insert Delete GetRandom O(1)

(Luo)

/\*

思路：需要的是insert,remove 可以用hashmap的得到O(1)

getRandom需要一个index来看顺序，但是HashMap没有顺序

HashMap ： value : index

ArrayList: index : value

\*/

class RandomizedSet {

Map<Integer, Integer> dict; // value to index pair

List<Integer> list; // value

Random rand;

public RandomizedSet() {

dict = new HashMap<>();

list = new ArrayList<>();

rand = new Random();

}

public boolean insert(int val) {

if (dict.containsKey(val)) {

return false;

}

//下面两句别写反了

dict.put(val, list.size()); // 新的value的index是list.size() 因为是下一个

list.add(val);

return true;

}

public boolean remove(int val) {

if (!dict.containsKey(val)) {

return false;

}

Integer idx = dict.get(val);

int lastElement = list.get(list.size() - 1);

list.set(idx, lastElement);

dict.put(lastElement, idx); // remove index上的value后，从map里把val的entry删去，也要更新lastElement的index，在map里

list.remove(list.size() - 1);

dict.remove(val);

return true;

}

public int getRandom() {

return list.get(rand.nextInt(list.size()));

}

}

# 398 Random Pick Index

(Luo)

/\*

HashMap - value: list<index>

(1: [0])

(2: [1])

(3, [2, 3, 4])

Runtime: 64 ms, faster than 69.62% of Java online submissions for Random Pick Index.

Memory Usage: 49.3 MB, less than 54.89% of Java online submissions for Random Pick Index.

\*/

class Solution {

HashMap<Integer, List<Integer>> map;

Random rand = new Random();

public Solution(int[] nums) {

map = new HashMap<>();

for (int i = 0; i < nums.length; i++) {

map.putIfAbsent(nums[i], new ArrayList<>());

map.get(nums[i]).add(i);

}

}

public int pick(int target) {

List<Integer> list = map.get(target);

return list.get(rand.nextInt(list.size()));

}

}

493. Reverse Pairs

(Tang)

| class Solution {  int[] helper;  public int reversePairs(int[] nums) {  this.helper = new int[nums.length];  return mergeSort(nums, 0, nums.length-1);  }  private int mergeSort(int[] nums, int s, int e){  if(s>=e) return 0;   int mid = s + (e-s)/2;   int cnt = mergeSort(nums, s, mid) + mergeSort(nums, mid+1, e);   for(int i = s, j = mid+1; i<=mid; i++){  while(j<=e && nums[i]/2.0 > nums[j]) j++; //  cnt += j-(mid+1);   }  //Arrays.sort(nums, s, e+1);   myMerge(nums, s, mid, e);  return cnt;   }    private void myMerge(int[] nums, int s, int mid, int e){  for(int i = s; i<=e; i++) helper[i] = nums[i];  int p1 = s;//pointer for left part  int p2 = mid+1;//pointer for right part  int i = s;//pointer for sorted array  while(p1<=mid || p2<=e){  if(p1>mid || (p2<=e && helper[p1] >= helper[p2])){  nums[i++] = helper[p2++];  }else{  nums[i++] = helper[p1++];  }  }  } } /\*  0 2 3 4  1,3,2,|3, 1  / \ 1,3,2 3, 1  / \ / \ 1, 3 2 3 1  / \ 1 3   i j i j 1, 3 | 2 | 3, 1 cnt: 1  +1  \ /   1, 2, 3 1, 3  i j  +1 cnt: 2  \ /  1, 1, 2, 3, 3     MergeSort  Explanation: In each round, we divide our array into two parts and sort them. So after "int cnt = mergeSort(nums, s, mid) + mergeSort(nums, mid+1, e); ", the left part and the right part are sorted and now our only job is to count how many pairs of number (leftPart[i], rightPart[j]) satisfies leftPart[i] <= 2\*rightPart[j]. For example, left: 4 6 8 right: 1 2 3 so we use two pointers to travel left and right. For each leftPart[i], if j<=e && nums[i]/2.0 > nums[j], we just continue to move j to the end, to increase rightPart[j], until it is valid. Like in our example, left's 4 can match 1 and 2; left's 6 can match 1, 2, 3, and left's 8 can match 1, 2, 3. So in this particular round, there are 8 pairs found, so we increase our total by 8.  \*/ |
| --- |

# 528. Random Pick with Weight

(luo)

class Solution {

private int[] prefixSums;

private int totalSum;

public Solution(int[] w) {

prefixSums = new int[w.length]; // constructor

int prefixSum = 0;

for (int i = 0; i < w.length; i++) {

prefixSum += w[i];

prefixSums[i] = prefixSum;

}

totalSum = prefixSum;

}

public int pickIndex() {

double target = totalSum \* Math.random();// 0...4 -2.4

// run a binary search to find the target zone

int low = 0, high = prefixSums.length; // 0..2

while (low < high) {

// better to avoid the overflow

int mid = low + (high - low) / 2; // mid = 1

if (target > prefixSums[mid])

low = mid + 1;

else

high = mid;

}

return low;

}

}

# 78 Subsets

(Lynn)

public void sortColors(int[] nums) {

if (nums == null || nums.length == 0) {

return;

}

int i = 0;

int j = 0;

int k = nums.length - 1;

while (j <= k) {

if (nums[j] == 0) {

//temp is 1, because i points to known part, we could just use 1 instead of temp

int temp = nums[i];

nums[i] = nums[j];

nums[j] = temp; // now j points to 1, we move j forward, let j points to unknown

i++;

j++;

} else if (nums[j] == 1) {

j++;

} else if (nums[j] == 2) {

//at this point, we dont know what num k points to, so we need to recotd by temp

int temp = nums[k];

nums[k] = nums[j];

nums[j] = temp;

k--;

//we dont move j because the number swaped to j is still unknown

}

}

}

# 815 Bus Routes

(huang)

// c#

public class Solution

{

private bool findStopToRoutes(int[][] routes, int source, int target, Dictionary <int, List<int>> stopToroutes, HashSet<int> startRoute, HashSet<int> endRoute)

{

for(int i = 0; i < routes.Length; i++)

{

bool sourceFlag = false;

bool targetFlag = false;

foreach (var stop in routes[i])

{

if (stop == source)

{

sourceFlag = true;

startRoute.Add(i);

}

if (stop == target)

{

targetFlag = true;

endRoute.Add(i);

}

if (!stopToroutes.ContainsKey(stop))

{

stopToroutes[stop] = new List<int>();

}

stopToroutes[stop].Add(i);

}

if (sourceFlag && targetFlag)

{

return true;

}

}

return false;

}

private void findRouteGraph(Dictionary <int, List<int>> stopToroutes, Dictionary <int, HashSet<int>> routeG)

{

foreach( var keyValue in stopToroutes)

{

var key = keyValue.Key;

var value = keyValue.Value;

// Console.WriteLine("key in: {0}", key);

for(int i = 0; i < value.Count; i ++)

{

int start = value[i];

for (int j = 0; j < value.Count; j ++)

{

if (i != j)

{

int end = value[j];

if ( ! routeG.ContainsKey(start))

{

routeG[start] = new HashSet<int>();

}

routeG[start].Add(end);

}

}

}

}

}

public int NumBusesToDestination(int[][] routes, int source, int target)

{

// bus stop to busroute number

if (source == target)

{

return 0;

}

Dictionary <int, List<int>> stopToroutes = new Dictionary <int, List<int>>();

HashSet<int> startRoute = new HashSet<int>();

HashSet<int> endRoute = new HashSet<int>();

if (findStopToRoutes(routes, source, target, stopToroutes, startRoute, endRoute))

{

return 1;

}

Dictionary <int, HashSet<int>> routeG = new Dictionary <int, HashSet<int>>();

findRouteGraph(stopToroutes, routeG);

Dictionary <int, int> distance = new Dictionary <int, int>();

Queue<int> Q = new Queue<int>();

foreach(int route in startRoute)

{

Q.Enqueue(route);

distance.Add(route, 1);

}

while (Q.Count != 0)

{

int route = Q.Dequeue();

//Console.WriteLine("route : {0}",route);

if (endRoute.Contains(route))

{

return distance[route];

}

//Console.WriteLine("route : {0}",route);

if (!routeG.ContainsKey(route))

{

break;

}

foreach (var nextroute in routeG[route])

{

if (distance.ContainsKey(nextroute))

{

continue;

}

Q.Enqueue(nextroute);

distance[nextroute] = distance[route] + 1;

}

}

return -1;

}

}

# 76 Minimum Window Substring

(code by Luo)

/\*

sliding window

\*/

class Solution {

public String minWindow(String s, String t) {

Map<Character, Integer> tMap = new HashMap<>();

for (char c : t.toCharArray()) {

tMap.putIfAbsent(c, 0);

tMap.put(c, tMap.get(c) + 1);

}

int counter = t.length(); //用于记录需要多少个match才能成功

int slow = 0;

int minLength = Integer.MAX\_VALUE;

int left = 0;

int right = 0;

for (int i = 0; i < s.length(); i++) { // fast pointer

char c = s.charAt(i);

if (tMap.containsKey(c)) {

tMap.put(c, tMap.get(c) - 1);

if (tMap.get(c) >= 0) { //如果被减成了负数，则counter不用--，说明进来的多了。

counter--; // 注意这里

}

}

while (counter == 0) {

if (minLength > i - slow + 1) {

minLength = i - slow + 1;

left = slow;

right = i;

}

char ch = s.charAt(slow);

if (tMap.containsKey(ch)) {

tMap.put(ch, tMap.get(ch) + 1);

if (tMap.get(ch) > 0) {//如果ch在tMap里目前是负的，说明目前的substring里的本char多了，则counter不需要++

counter++;

}

}

slow++;

}

}

return minLength == Integer.MAX\_VALUE? "" : s.substring(left, right + 1);

}

}

# 146. LRU Cache

(Luo)

| class LRUCache {  Map<Integer, Node> map;  DDList list;  int capacity;   public LRUCache(int capacity) {  map = new HashMap<>();  list = new DDList();  this.capacity = capacity;  }    public int get(int key) {  // get value for key from map  // update dd list position  Node node = map.get(key);  if (node == null) {  return -1;  }  int value = node.getValue();  list.update(node); // remove node, add to head of list  return value;  }    public void put(int key, int value) {  // if exist, update dd list and node value  // if not exist, create, add to map, adjust size, add to head of dd list  Node node = map.get(key);  if (node != null) {  map.get(key).value = value;  list.update(node);  } else {  node = new Node(key, value);  if (list.getSize() == capacity) {    // map.remove(list.dummyTail.prev.key); //???  int tailKey = list.removeTail();  map.remove(tailKey);  }  map.put(key, node);  list.addHead(node);  }    } }  class Node {  int key, value;  Node prev, next;  Node (int key, int value) {  this.key = key;  this.value = value;  }  public int getValue() {  return value;  } }  class DDList {  int size;  Node dummyHead;  Node dummyTail;    public DDList() {  dummyHead = new Node(0, 0);  dummyTail = new Node(0, 0);  dummyHead.next = dummyTail;  dummyTail.prev = dummyHead;  }  public void update(Node node) { // remove node, add to head  remove(node);  addHead(node);  }  public void addHead(Node node) {  size++;  Node head = dummyHead.next;  dummyHead.next = node;  node.prev = dummyHead;  node.next = head;  head.prev = node;    }  public int removeTail() {  Node tail = dummyTail.prev;  remove(tail);  return tail.key;  }  private void remove(Node node) {  size--;  Node prev = node.prev;  Node next = node.next;  prev.next = next;  next.prev = prev;  }  public int getSize() {  return size;  } } |
| --- |

# 212. Word Search

(Luo)

| class Solution {    public List<String> findWords(char[][] board, String[] words) {    Trie trie = new Trie(words);    int m = board.length;  int n = board[0].length;   StringBuilder sb = new StringBuilder();  Set<String> res = new HashSet<>();  boolean[][] visited = new boolean[m][n];   for (int i = 0; i < m; i++) {  for (int j = 0; j < n; j++) {  helper(board, i, j, trie.root, sb, res, visited);  }  }  return new ArrayList<>(res);  }   private void helper(char[][] board, int i, int j, TrieNode root, StringBuilder sb, Set<String> res, boolean[][] visited) {  // base case  if (i < 0 || j < 0 || i >= board.length || j >= board[0].length || visited[i][j]) {  return;  }  // recursion rule  char ch = board[i][j];  int index = ch - 'a';    if (root.children[index] == null) {  return;  }   sb.append(ch);  root = root.children[index];   if (root.isWord) {  res.add(sb.toString());  }   visited[i][j] = true;   helper(board, i + 1, j, root, sb, res, visited);  helper(board, i - 1, j, root, sb, res, visited);  helper(board, i, j + 1, root, sb, res, visited);  helper(board, i, j - 1, root, sb, res, visited);    sb.deleteCharAt(sb.length() - 1);  visited[i][j] = false;  }   }  class TrieNode {  TrieNode[] children = new TrieNode[26]; // index indicates which Character, node is children  boolean isWord;   }  class Trie {  TrieNode root;    public Trie(String[] words) {  root = new TrieNode();    buildTrie(words);  }    public void buildTrie(String[] words) {  for (String word : words) {  addWord(word, root);  }  }    private void addWord(String word, TrieNode root) {  TrieNode cur = root;  for (char c : word.toCharArray()) {  int index = c - 'a';  TrieNode node = cur.children[index];  if (node == null) {  cur.children[index] = new TrieNode();  }  cur = cur.children[index];    }  cur.isWord = true;  }  } |
| --- |

# 1057. Campus Bikes

class Solution {

public int[] assignBikes(int[][] workers, int[][] bikes) {

int[] result = new int[workers.length];

List<Node> list = new ArrayList();

//for every worker compute the distances to every bike O(n\*m) and add them to the list

for(int i = 0;i<workers.length;i++){

for(int j = 0;j<bikes.length;j++) {

list.add(new Node(i,j,getManhattan(workers[i],bikes[j])));

}

}

//sort by distance, sort by worker index, sort by bike

// O(n\*m\*log(n\*m))

Collections.sort(list,(a,b)-> {

if(a.distance==b.distance) {

if(a.worker==b.worker) {

return a.bike-b.bike;

}

return a.worker-b.worker;

} else {return a.distance-b.distance;}

});

//keep track of the workers that already have bike and of the bikes that have already been used

// O(n\*m)

Set<Integer> bikesSet = new HashSet();

Set<Integer> workersSet = new HashSet();

for(int i = 0;i< list.size();i++) {

Node curr = list.get(i);

if (bikesSet.size() == bikes.length) break;

if(bikesSet.contains(curr.bike) || workersSet.contains(curr.worker)) continue;

result[curr.worker] = curr.bike;

bikesSet.add(curr.bike);

workersSet.add(curr.worker);

}

return result;

}

private int getManhattan(int[] p1, int[] p2) {

return Math.abs(p1[0]-p2[0]) + Math.abs(p1[1]-p2[1]);

}

private class Node{

public int worker;

public int bike;

public int distance;

public Node(int w, int b, int d) {

worker = w;

bike = b;

distance = d;

}

}

}

# 726. Number of Atoms

(luo)

/\*

主要是getName 和 getCount 两个核心的函数，用于被调用

遇到左括号时就可以进入recursion，注意出来时需要跟当前的map合并

遇到右括号就结束recursion，return当前层

https://www.youtube.com/watch?v=6nQ2jfs7a7I

TC: O(n) for main method, nlogn for sort

\*/

| class Solution {   public String countOfAtoms(String formula) {  int[] index = new int[1];  char[] input = formula.toCharArray();   Map<String, Integer> map = countOfAtoms(input, index);    StringBuilder sb = new StringBuilder();  List<String> resList = new ArrayList<>();  for (String s : map.keySet()) {  if (map.get(s) > 1) {  resList.add(s + map.get(s));  } else {  resList.add(s);  }    }  Collections.sort(resList);  for (String s : resList) {  sb.append(s);  }  return sb.toString();  }  private Map<String, Integer> countOfAtoms(char[] input, int[] index) {  Map<String, Integer> res = new HashMap<>();    if (index[0] == input.length) {  return res;  }    while (index[0] != input.length) { // 这里一定要要，why？    char cur = input[index[0]];  if (cur == '(') {  index[0]++;  Map<String, Integer> tmp = countOfAtoms(input, index);  int count = getCount(input, index);  for (Map.Entry<String, Integer> entry : tmp.entrySet()) {  res.put(entry.getKey(), res.getOrDefault(entry.getKey(), 0) + entry.getValue() \* count);  }  } else if (cur == ')') {  index[0]++;  return res;  } else {  String name = getName(input, index);  res.put(name, res.getOrDefault(name, 0) + getCount(input, index));  }  }  return res;  }  private String getName(char[] input, int[] index) {  StringBuilder sb = new StringBuilder();  sb.append(input[index[0]]);    index[0]++;    if (index[0] < input.length && input[index[0]] >= 'a' && input[index[0]] <= 'z') {  sb.append(input[index[0]]);  index[0]++;  }    return sb.toString();  }  private int getCount(char[] input, int[] index) {  int count = 0;  while (index[0] < input.length && input[index[0]] >= '0' && input[index[0]] <= '9') {  count = count \* 10 + input[index[0]] - '0';  index[0]++;  }  return count == 0? 1 : count;  } } |
| --- |

# 1597. Build Binary Expression Tree From Infix Expression

(Tang)

| /\*\*  \* Definition for a binary tree node.  \* class Node {  \* char val;  \* Node left;  \* Node right;  \* Node() {this.val = ' ';}  \* Node(char val) { this.val = val; }  \* Node(char val, Node left, Node right) {  \* this.val = val;  \* this.left = left;  \* this.right = right;  \* }  \* }  \*/ class Solution {  public Node expTree(String s) {  s = '(' + s + ')';  Deque<Node> nodes = new LinkedList<>();  Deque<Character> ops = new LinkedList<>();  Map<Character, Integer> priority = Map.of('+', 0, '-', 0, '\*', 1, '/', 1);   for (char c : s.toCharArray())  if (Character.isDigit(c)) {  nodes.push(new Node(c));  } else if (c == '(') {  ops.push(c);  } else if (c == ')') {  while (ops.peek() != '(')  nodes.push(buildNode(ops.pop(), nodes.pop(), nodes.pop()));  ops.pop(); // remove '('  } else { // c == '+' || c == '-' || c == '\*' || c == '/'  while (ops.peek() != '(' && priority.get(ops.peek()) >= priority.get(c))  nodes.push(buildNode(ops.pop(), nodes.pop(), nodes.pop()));  ops.push(c);  }   return nodes.peek();  }   private Node buildNode(char op, Node right, Node left) {  return new Node(op, left, right);  } } |
| --- |

## Related solution for 772. Basic Calculator III

| class Solution {  public int calculate(String s) {  if (s == null || s.length() == 0) return 0;  Stack<Integer> nums = new Stack<>();   Stack<Character> ops = new Stack<>();  int num = 0;  Map<Character, Integer> priority = Map.of('+', 0, '-', 0, '\*', 1, '/', 1, '(', 2, ')', 2);  for (int i = 0; i < s.length(); i++) {  char c = s.charAt(i);  if (c == ' ') {  continue;  }   if (Character.isDigit(c)) {  num = c - '0';  while (i + 1 < s.length() && Character.isDigit(s.charAt(i + 1))) {  num \*= 10;  num += (s.charAt(i+1) - '0');  i++;  }  nums.push(num);  num = 0;  } else if (c == '(') {  ops.push(c);  } else if (c == ')') {  while (!ops.isEmpty() && ops.peek() != '(') {  nums.push(calc(ops.pop(), nums.pop(), nums.pop()));  }  ops.pop();  } else if (c == '+' || c == '-' || c == '\*' || c == '/') {  while (!ops.isEmpty() && ops.peek() != '(' && priority.get(ops.peek()) >= priority.get(c)) {  nums.push(calc(ops.pop(), nums.pop(), nums.pop()));  }   ops.push(c);  }  }  while (!ops.isEmpty()) {  nums.push(calc(ops.pop(), nums.pop(), nums.pop()));  }  return nums.peek();  }    private int calc(Character op, int num1, int num2) {  switch (op) {  case '+': return num2 + num1;  case '-': return num2 - num1;  case '\*': return num2 \* num1;  case '/': return num2 / num1;  }  throw new IllegalArgumentException();  } } |
| --- |

# 1231. Divide Chocolate

public int maximizeSweetness(int[] sweetness, int k) {

if (sweetness == null || sweetness.length == 0) {

return 0;

}

k = k + 1; // add self

int min = sweetness[0];

int max = 0;

for (int i = 0; i < sweetness.length; i++) {

min = Math.min(min, sweetness[i]);

max += sweetness[i];

}

while (min <= max) {

int mid = (max + min) / 2;

int cut = countCut(sweetness, mid);

if (cut > k) {

min = mid + 1;

} else if (cut < k){

max = mid - 1;

} else {

min = mid + 1;

}

}

return max;

}

private int countCut(int[] sweetness, int target) {

//we want to cut arr into subarray with sum >= target

//count how many cuts

int cutCount = 0;

int subArrSum = 0;

for (int i = 0; i < sweetness.length; i++) {

subArrSum += sweetness[i];

if (subArrSum >= target) {

subArrSum = 0;

cutCount += 1;

}

}

return cutCount;

}

# 875. Koko Eating Bananas

//M \* logN

public int minEatingSpeed(int[] piles, int h) {

if (piles == null || piles.length == 0 || h < piles.length) return -1;

int maxNum = piles[0];

// int totalSum = 0;

for (int i = 0; i < piles.length; i++) {

// totalSum += piles[i];

maxNum = Math.max(maxNum, piles[i]);

}

int minK = 1;

int maxK = maxNum;

//logN, N the max number of banana

while (minK <= maxK) {

int mid = minK + ((maxK - minK) / 2);

if (canFinish(piles, mid, h)) {

maxK = mid - 1;

} else {

minK = mid + 1;

}

}

return minK;

}

// M , M length of piles

private boolean canFinish(int[] piles, int k, int h) {

// if (k == 0) return true;

int hourCost = 0;

for (int num : piles) {

hourCost += num / k;

if (num % k > 0) hourCost += 1;

}

return hourCost <= h;

}

# 935. Knight Dialer

// dp , n \* 10

class Solution {

public int knightDialer(int n) {

int MOD = 1000000007;

int paths[][] = {{4, 6}, {6, 8}, {7, 9}, {4, 8}, {0, 3, 9}, {}, {0, 1, 7}, {2, 6}, {1, 3}, {2, 4}};

// Previous moves of knight-> For instance, if a knight is at 0, it reached from either 4 or 6. Similarly if it is at 1, it is reached from 7 or 9 & so on

double dp[][] = new double[n + 1][10]; // rows -> no of steps taken to reach row i cols-> no of digits

for (int j = 0; j < 10; j++)

dp[1][j] = 1; //populate the base case for n =1

for (int i = 2; i < n + 1; i++) { // no of steps taken by knight to reach i

for (int j = 0; j < 10; j++) { // no of digits

for (int p : paths[j]) { // Previous move of knight in order to reach digit j

dp[i][j] += dp[i - 1][p]; // cumulatively add from the previous knight move. For instance., F(2, 0) -> F(1,4) + F(1,6) F(2,6) -> F(1,0) + F(1,1) + F(1,7)

}

dp[i][j] %= MOD;

}

}

double sum = 0d;

for (int j = 0; j < 10; j++)

sum += dp[n][j];

return (int) (sum % MOD);

}

}

// dfs

| class Solution {  int mod = 1000000007;  int[][] moves = {{-1, -2}, {-2, - 1}, {-2, 1}, {-1, 2}, {1, -2}, {2, -1}, {1, 2}, {2, 1}};  public int knightDialer(int n) {  int[][][] memo = new int[4][3][n + 1];  int res = 0;  for (int i = 0; i < 4; i++) {  for (int j = 0; j < 3; j++) {  res = (res + helper(i, j, n, memo) % mod ) % mod;  }  }  return res;  }      // 在i，j位置上，还剩n步可走，有多少种distinct的phone number  private int helper(int x, int y, int n, int[][][] memo) {  if (x < 0 || y < 0 || x >= 4 || y >= 3 || (x == 3 && y != 1)) {  return 0;  }  // baes case  if (n == 1) {  return 1;  }    if (memo[x][y][n] != 0) {  return memo[x][y][n];  }    int cur = 0;  for (int i = 0; i < 8; i++) {  int xx = x + moves[i][0];  int yy = y + moves[i][1];  cur = (cur + (helper(xx, yy, n - 1, memo) % mod)) % mod;  }    memo[x][y][n] = cur;    return cur;  } } |
| --- |

# 616. Add Bold Tag in String

/\*

TC: O(length of all characters in dict \* length of s string)

SC: O(length of input string)

\*/

public class Solution {

public String addBoldTag(String s, String[] dict) {

boolean[] bold = new boolean[s.length()];

for(String substr : dict) { // O(length of dict array)

int start=0;

while(start >= 0) {

start = s.indexOf(substr,start); // O(n, length of input string)

if(start<0) break;

int end = start+substr.length();

for(int i=start; i<end; i++) {

bold[i]=true;

}

start++; // Just start from next index, instead of iterating through entire string

}

}

StringBuilder sb = new StringBuilder();

// O(n)

for(int i=0; i<s.length(); i++) {

if(bold[i] && (i-1<0 || !bold[i-1])) {

sb.append("<b>");

}

sb.append(s.charAt(i)); // Just go character by character rather than cutting up the string

if(bold[i] && (i+1==s.length() || !bold[i+1])) {

sb.append("</b>");

}

}

return sb.toString();

}

}

# 126. Word Ladder II

| class Solution {  public List<List<String>> findLadders(String beginWord, String endWord, List<String> wordList) {  List<List<String>> ans = new ArrayList<>();  Set<String> wordSet = new HashSet<>(wordList);  if (!wordSet.contains(endWord)) return ans;    Queue<List<String>> queue = new LinkedList<>(); // each element in queue is a path  queue.offer(Arrays.asList(beginWord));  Set<String> visited = new HashSet<>();  visited.add(beginWord);  // list(hit->)    while (!queue.isEmpty()) {   int sz = queue.size();  while (sz-- > 0) { // m\*n steps (26k2)  List<String> currPath = queue.poll();  String lastWord = currPath.get(currPath.size()-1);  List<String> neighbors = getNeighbors(lastWord, wordSet); //26k2  for (String neigh : neighbors) {  List<String> newPath = new ArrayList<>(currPath);  newPath.add(neigh);  visited.add(neigh);  if (neigh.equals(endWord)) {  ans.add(newPath);  } else {  queue.offer(newPath);  }  }  }  for (String s : visited) // remove used words from wordSet to avoid going back  wordSet.remove(s);  }    return ans;  }    private List<String> getNeighbors(String word, Set<String> wordSet) { // 26k2  List<String> neighbors = new LinkedList<>();  for (int i = 0; i < word.length(); i++) {  char[] ch = word.toCharArray();  for (char c = 'a'; c <= 'z'; c++) {  ch[i] = c;  String str = new String(ch);  if (wordSet.contains(str)) // only get valid neighbors  neighbors.add(str);  }  }  return neighbors;  } } |
| --- |

# 127. Word Ladder

| class Solution {  public int ladderLength(String beginWord, String endWord, List<String> wordList) {  Set<String> dict = new HashSet<>(wordList);  Queue<String> queue = new ArrayDeque<>();  Set<String> visited = new HashSet<>();  int level = 0;  // 1. init  queue.offer(beginWord);  visited.add(beginWord);  while (!queue.isEmpty()) { // n(26k2)  int size = queue.size();  while (size > 0) {  String tmp = queue.poll();  size--;  if (tmp.equals(endWord)) {  return level + 1;  }  // generate next word  List<String> nei = getNei(tmp, visited, dict);  for (String neiWord : nei) {  queue.offer(neiWord);  visited.add(neiWord);  }  }  level++;  }  return 0;  }  private List<String> getNei(String tmp, Set<String> visited, Set<String> dict) { // 26k2   List<String> nei = new ArrayList<>(); //   char[] arr = tmp.toCharArray(); // k    for (int i = 0; i < arr.length; i++) { // k  char orig = arr[i];  for (char j = 'a'; j <= 'z'; j++) { // 26  if (j != orig) {  arr[i] = j;  String newStr = new String(arr); // k  if (dict.contains(newStr) && !visited.contains(newStr)) { //   nei.add(newStr);  }  arr[i] = orig;  }  }  }  return nei;  } } |
| --- |

# 102. Binary Tree Level Order Traversal

| class Solution {   List<List<Integer>> levels = new ArrayList<List<Integer>>();   public void helper(TreeNode node, int level) {  // start the current level  if (levels.size() == level)  levels.add(new ArrayList<Integer>());   // fulfill the current level  levels.get(level).add(node.val);   // process child nodes for the next level  if (node.left != null)  helper(node.left, level + 1);  if (node.right != null)  helper(node.right, level + 1);  }    public List<List<Integer>> levelOrder(TreeNode root) {  if (root == null) return levels;  helper(root, 0);  return levels;  } }  // Method 2: iterative  public List<List<Integer>> levelOrder(TreeNode root) {  Queue<TreeNode> queue = new LinkedList<TreeNode>();  List<List<Integer>> wrapList = new LinkedList<List<Integer>>();  if(root == null) return wrapList;  queue.offer(root);  while(!queue.isEmpty()){  int levelNum = queue.size();  List<Integer> subList = new LinkedList<Integer>();  for(int i=0; i<levelNum; i++) {  if(queue.peek().left != null) queue.offer(queue.peek().left);  if(queue.peek().right != null) queue.offer(queue.peek().right);  subList.add(queue.poll().val);  }  wrapList.add(subList);  }  return wrapList;  } |
| --- |

# 103. Binary Tree Zigzag Level Order Traversal

(Tang)

| // dfs recursion: tc/sc: o(n)  protected void DFS(TreeNode node, int level, List<List<Integer>> results) {  if (level == results.size()) {  LinkedList<Integer> newLevel = new LinkedList<Integer>();  results.add(newLevel);  }   if (level % 2 == 0)  results.get(level).add(node.val);  else  results.get(level).add(0, node.val);     if (node.left != null) DFS(node.left, level + 1, results);  if (node.right != null) DFS(node.right, level + 1, results);  }   public List<List<Integer>> zigzagLevelOrder(TreeNode root) {  List<List<Integer>> results = new ArrayList<List<Integer>>();  if (root == null) {  return results;  }  DFS(root, 0, results);  return results;  }   // iterative, bfs, tc/sc: o(n) class Solution {  public List<List<Integer>> zigzagLevelOrder(TreeNode root) {  List<List<Integer>> res = new ArrayList<>();  if (root == null) return res;  Queue<TreeNode> queue = new LinkedList<>();  queue.add(root);  boolean zigzag = false;  while (!queue.isEmpty()) {  List<Integer> level = new ArrayList<>();  int cnt = queue.size();  for (int i = 0; i < cnt; i++) {  TreeNode node = queue.poll();  if (zigzag) {  level.add(0, node.val);  } else {  level.add(node.val);  }  if (node.left != null) {  queue.add(node.left);  }  if (node.right != null) {  queue.add(node.right);  }  }  res.add(level);  zigzag = !zigzag;  }  return res;  } } |
| --- |

# **108. Convert Sorted Array to Binary Search Tree**

(Tang)

/\*

ambiguity of choosing root happens when the number of input is even.

\*/

| class Solution { // tc: o(n), sc: o(height), height = log(n)   int[] nums;   public TreeNode helper(int left, int right) {  if (left > right) return null;   // always choose left middle node as a root  int p = (left + right) / 2;   // always choose right middle node as a root  // if ((left + right) % 2 == 1) ++p;  // choose random middle node as a root  // if ((left + right) % 2 == 1) p += rand.nextInt(2);    // preorder traversal: node -> left -> right  TreeNode root = new TreeNode(nums[p]);  root.left = helper(left, p - 1);  root.right = helper(p + 1, right);  return root;  }   public TreeNode sortedArrayToBST(int[] nums) {  this.nums = nums;  return helper(0, nums.length - 1);  } } |
| --- |
|  |
|  |

# 979. Distribute Coins in Binary Tree

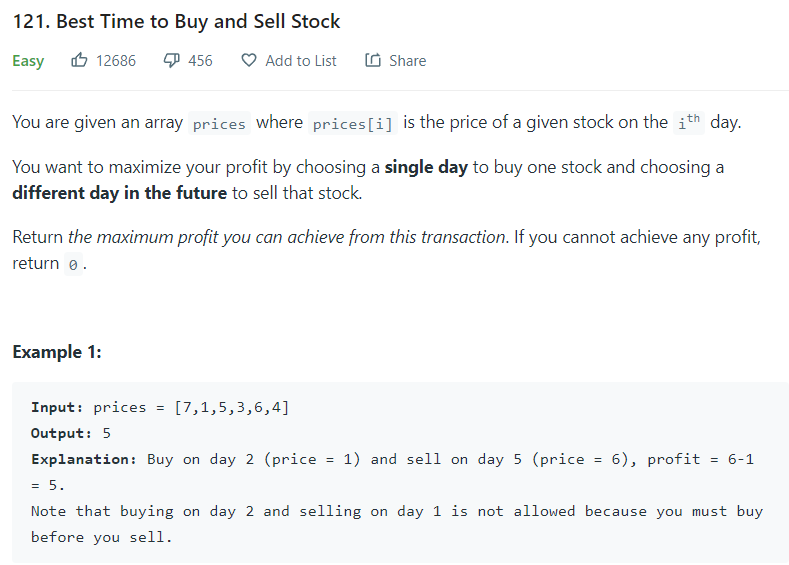
(Tang)

| class Solution {  int ans;  public int distributeCoins(TreeNode root) {  ans = 0;  dfs(root);  return ans;  }   public int dfs(TreeNode node) { // dfs 当前节点需要 拿给子节点的金币个数  if (node == null) return 0;  int L = dfs(node.left);  int R = dfs(node.right);  ans += Math.abs(L) + Math.abs(R);  return node.val + L + R - 1;  }  // https://leetcode-cn.com/problems/distribute-coins-in-binary-tree/solution/zai-er-cha-shu-zhong-fen-pei-ying-bi-by-leetcode/  /\*  node.val + L + R - 1 表示当前节点需要 拿给子节点的金币个数 或者 从子节点拿给自己的金币的个数，即当前节点的金币的移动次数。因为是后序遍历，所以能保证子节点已经都得到了需要的金币，同时 ans += Math.abs(L) + Math.abs(R);记录了子节点的移动次数，故累加后即是最终结果。    0(0+2-1-1=0) +2+|-1| = 3  / \  +0 3(3+0+0-1=2) 0(0+0+0-1=-1) +0     \*/ } |
| --- |

# 股票买卖专题

|  | k transaction | no limit transactions | cooldown time | transaction fee |
| --- | --- | --- | --- | --- |
| 121 | x(k == 1) |  |  |  |
| 122 |  | x |  |  |
| 714 |  | x |  | x |
| 123 | x(k<=2) |  |  |  |
| 188 | x(<= n-1) |  |  |  |
| 309 |  | x | x |  |

## [121. Best Time to Buy and Sell Stock](https://leetcode.com/problems/best-time-to-buy-and-sell-stock/) (only one time transaction)

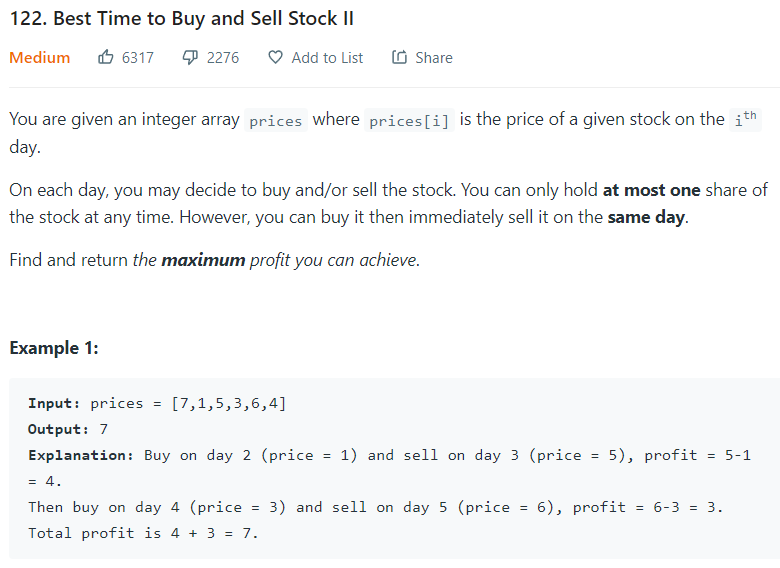


| public int maxProfit(int[] prices) {  if (prices == null || prices.length == 0) {  return 0;  }  int len = prices.length;  int[] buy = new int[len]; // max profit at i  int[] sell = new int[len];  buy[0] = -prices[0];  for (int i = 1; i < len; i++) {  buy[i] = Math.max(buy[i-1], -prices[i]);  sell[i] = Math.max(sell[i-1], buy[i-1] + prices[i]);  }  return sell[len - 1];  } |
| --- |

public int maxProfit(int[] prices) {  
 if (prices == null || prices.length == 0) {  
 return 0;  
 }  
 int len = prices.length;  
 int buy = -prices[0];  
 int sell = 0;  
 for (int i = 1; i < len; i++) {  
 int tmp = buy;  
 buy = Math.max(buy, -prices[i]);  
 sell = Math.max(sell, tmp + prices[i]);  
 }  
 return sell;  
 }

| public class Solution {  public int maxProfit(int prices[]) {  int minprice = Integer.MAX\_VALUE;  int maxprofit = 0;  for (int i = 0; i < prices.length; i++) {  if (prices[i] < minprice)  minprice = prices[i];  else if (prices[i] - minprice > maxprofit)  maxprofit = prices[i] - minprice;  }  return maxprofit;  } } // tc: o(n), sc: o(1)  public int maxProfit(int[] prices) {  if (prices == null || prices.length == 0) {  return 0;  }  int len = prices.length;  int buy = -prices[0];  int sell = 0;  for (int i = 1; i < len; i++) {  //int tmp = buy;  buy = Math.max(buy, -prices[i]);  sell = Math.max(sell, buy + prices[i]);  }  return sell;  } |
| --- |

## [122. Best Time to Buy and Sell Stock II](https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/) (no limit on transaction times)



| class Solution {  public int maxProfit(int[] prices) {  int maxprofit = 0;  for (int i = 1; i < prices.length; i++) {  if (prices[i] > prices[i - 1])  maxprofit += prices[i] - prices[i - 1];  }  return maxprofit;  } } |
| --- |

DP method

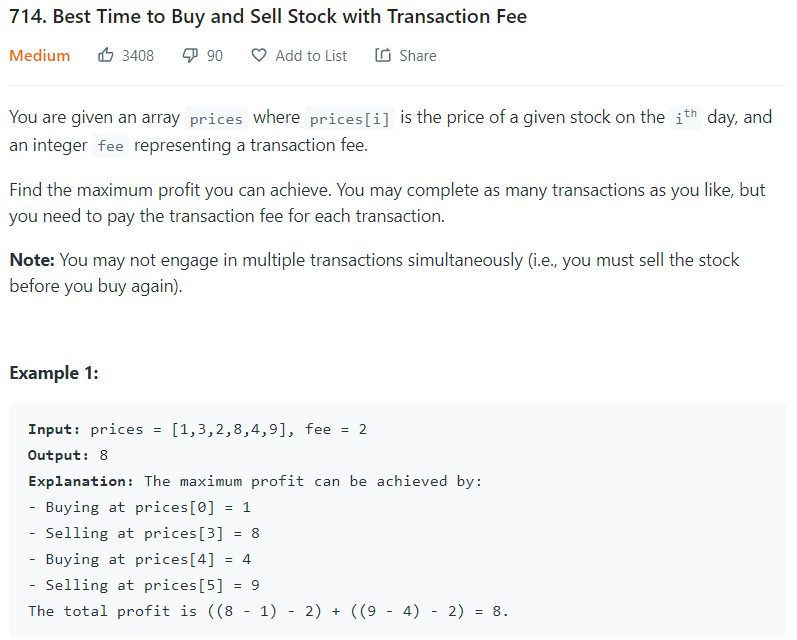
| /\* dp[i][0] 表示第 ii 天交易完后手里没有股票的最大利润， dp[i][1] 表示第 ii 天交易完后手里持有一支股票的最大利润（i从 00 开始）有第i天。 \*/ |
| --- |

| public int maxProfit(int[] prices) {  int n = prices.length;  int[][] dp = new int[n][2];  dp[0][0] = 0;  dp[0][1] = -prices[0];  for (int i = 1; i < n; ++i) {  dp[i][0] = Math.max(dp[i - 1][0], dp[i - 1][1] + prices[i]);  dp[i][1] = Math.max(dp[i - 1][1], dp[i - 1][0] - prices[i]);  }  return dp[n - 1][0];  } |
| --- |

/\*  
space optimization  
\*/

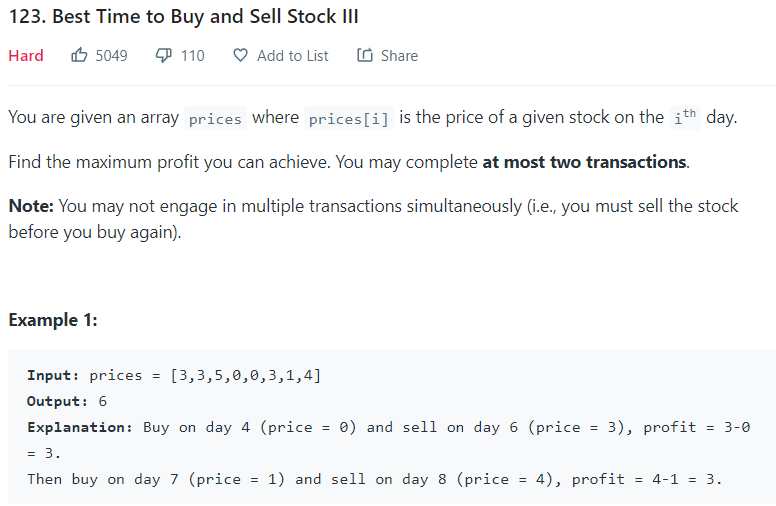
| class Solution {  public int maxProfit(int[] prices) {  int n = prices.length;  int dp0 = 0, dp1 = -prices[0];  for (int i = 1; i < n; ++i) {  int newDp0 = Math.max(dp0, dp1 + prices[i]);  int newDp1 = Math.max(dp1, dp0 - prices[i]);  dp0 = newDp0;  dp1 = newDp1;  }  return dp0;  } } |
| --- |

## [714. Best Time to Buy and Sell Stock with Transaction Fee](https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-transaction-fee/) (122 + transaction fee)



| class Solution { /\* dp[i][0] 表示第 ii 天交易完后手里没有股票的最大利润， dp[i][1] 表示第 ii 天交易完后手里持有一支股票的最大利润（i从 00 开始）。 \*/  public int maxProfit0(int[] prices, int fee) {  int n = prices.length;  int[][] dp = new int[n][2];  dp[0][0] = 0;  dp[0][1] = -prices[0];  for (int i = 1; i < n; ++i) {  dp[i][0] = Math.max(dp[i - 1][0], dp[i - 1][1] + prices[i] - fee);  dp[i][1] = Math.max(dp[i - 1][1], dp[i - 1][0] - prices[i]);  }  return dp[n - 1][0];  }  /\* space optimization, greedy method  proofs from dp to greedy: when current i is minimum so far, buy1 will update, while sell1 cannot be larger than history largest. Example: see code 121 \*/  public int maxProfit(int[] prices, int fee) {  int n = prices.length;  int sell = 0, buy = -prices[0];  for (int i = 1; i < n; ++i) {  buy = Math.max(buy, sell - prices[i]);  sell = Math.max(sell, buy + prices[i] - fee);  }  return sell;  }  /\* space optimization, correct \*/  public int maxProfit(int[] prices, int fee) {  int n = prices.length;  int sell = 0, buy = -prices[0], tmp = 0;  for (int i = 1; i < n; ++i) {  tmp = sell;  sell = Math.max(sell, buy + prices[i] - fee);  buy = Math.max(buy, tmp - prices[i]);  }  return sell;  }  } |
| --- |
|  |

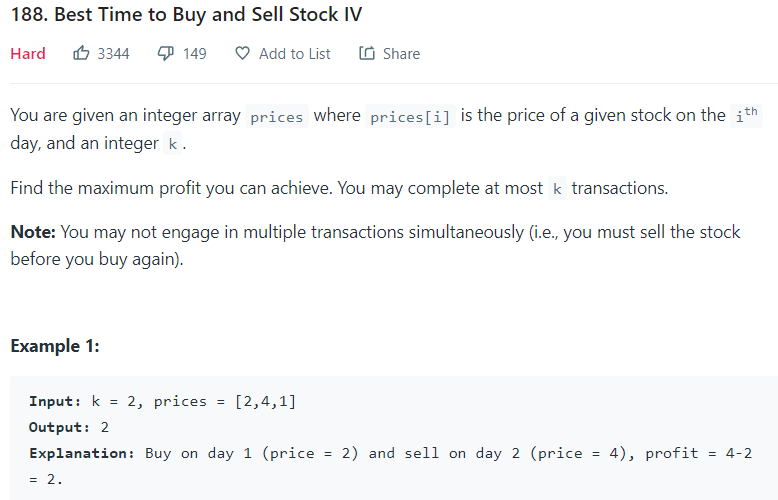
## [123. Best Time to Buy and Sell Stock III](https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iii/) (at most 2 times transaction, hard)



| class Solution { // greedy, DP method see 188  public int maxProfit(int[] prices) {  int n = prices.length;  int buy1 = -prices[0], sell1 = 0; // profit  int buy2 = -prices[0], sell2 = 0;  for (int i = 1; i < n; ++i) {  buy1 = Math.max(buy1, -prices[i]); // min prices [0, i)  sell1 = Math.max(sell1, buy1 + prices[i]); // hist, sell at ith d  buy2 = Math.max(buy2, sell1 - prices[i]); // hist, buy at ith d  sell2 = Math.max(sell2, buy2 + prices[i]); // hist, sell at ith d  }  return sell2;  } } |
| --- |

| /\* Ambiguity resolved, DP method with dimension optimization,  proofs from dp to greedy: when current i is minimum so far, buy1 will update, while sell1 cannot be larger than history largest. Example: see code 121 \*/  public int maxProfit(int[] prices) {  int n = prices.length;  int buy1 = -prices[0], sell1 = 0, pbuy1 = -prices[0], psell1 = 0;  int buy2 = -prices[0], sell2 = 0, pbuy2 = -prices[0], psell2 = 0;  for (int i = 1; i < n; ++i) {   buy1 = Math.max(pbuy1, -prices[i]);  sell1 = Math.max(psell1, pbuy1 + prices[i]);  buy2 = Math.max(pbuy2, sell1 - prices[i]);  sell2 = Math.max(psell2, pbuy2 + prices[i]);     pbuy1 = buy1;  pbuy2 = buy2;  psell1 = sell1;  psell2 = sell2;  }  return sell2;  } |
| --- |

## [188. Best Time to Buy and Sell Stock IV](https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iv/) (at most K times transaction, hard)



| /\*\*  \* dp[i, j] represents the max profit up until prices[j] using at most i transactions. (times <= i)   \* dp[i, j] = max(dp[i, j-1], prices[j] - prices[jj] + dp[i-1, jj]) { jj in range of [0, j-1] }  \* = max(dp[i, j-1], prices[j] + max(dp[i-1, jj] - prices[jj]))  \* dp[0, j] = 0; 0 transactions makes 0 profit  \* dp[i, 0] = 0; if there is only one price data point you can't make any transaction.  \*/  class Solution {  public int maxProfit(int k, int[] prices) {  int n = prices.length;  if (n <= 1)  return 0;   int[][] dp = new int[k+1][n];  for (int i = 1; i <= k; i++) {  int localMax = dp[i-1][0] - prices[0]; // -cost [0, i-1]  for (int j = 1; j < n; j++) {  dp[i][j] = Math.max(dp[i][j-1], prices[j] + localMax);  localMax = Math.max(localMax, dp[i-1][j] - prices[j]);  // profit at j, buy at j  }  }  return dp[k][n-1];  } } |
| --- |

// Time complexity: O(n\*k). Space complexity: O(k).

class Solution {

public int maxProfit(int k, int[] prices) {

if (k == 0) return 0;

int[] profit = new int[k+1];

int[] cost = new int[k+1];

profit[0] = 0;

Arrays.fill(cost, Integer.MAX\_VALUE);

for (int price: prices) {

for (int i = 0; i < k; i++) {

cost[i+1] = Math.min(cost[i+1], price - profit[i]);

profit[i+1] = Math.max(profit[i+1], price - cost[i+1]);

}

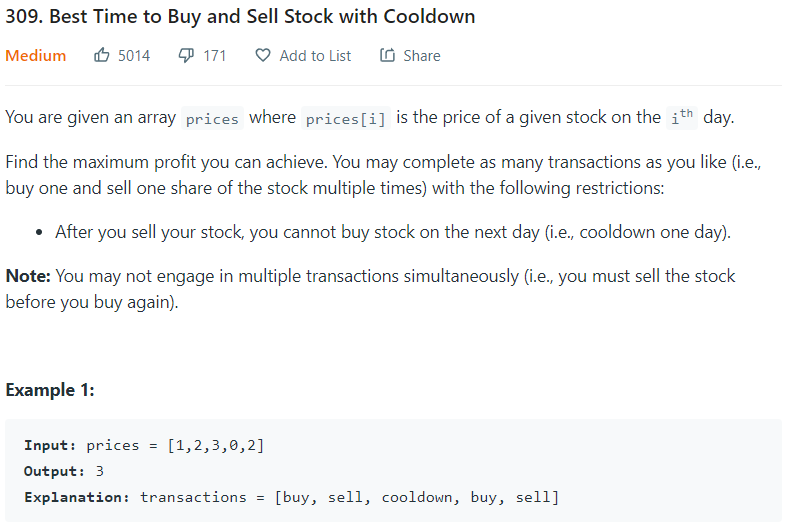
}

return profit[k];

}

}

## [309. Best Time to Buy and Sell Stock with Cooldown](https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/)



| class Solution {  public int maxProfit0(int[] prices) {  if (prices.length == 0) {  return 0;  }   int n = prices.length;  // f[i][0]: 手上持有股票的最大收益  // f[i][1]: 手上不持有股票，并且处于冷冻期中的累计最大收益  // f[i][2]: 手上不持有股票，并且不在冷冻期中的累计最大收益  int[][] f = new int[n][3];  f[0][0] = -prices[0];  for (int i = 1; i < n; ++i) {  f[i][0] = Math.max(f[i - 1][0], f[i - 1][2] - prices[i]);  f[i][1] = f[i - 1][0] + prices[i];  f[i][2] = Math.max(f[i - 1][1], f[i - 1][2]);  }  return Math.max(f[n - 1][1], f[n - 1][2]);  }  // Optimization w space  public int maxProfit(int[] prices) {  int sold = Integer.MIN\_VALUE, held = Integer.MIN\_VALUE, reset = 0;  for (int price : prices) {  int preSold = sold;    sold = held + price;  held = Math.max(held, reset - price);  reset = Math.max(reset, preSold);  }    return Math.max(sold, reset);  }   } |
| --- |

# 437. Path Sum III

public int pathSum(TreeNode root, int targetSum) {

if (root == null) return 0;

Map<Integer, Integer> prefixSum = new HashMap<>();

//map root to cur node prefixsum, frequency

int[] count = new int[1];

//initial with a pair of (0, 1) in case we have valid path from root to current node

prefixSum.put(0,1);

dfs(root, 0, targetSum, prefixSum, count);

return count[0];

}

/\*

Time : O(n)

Space: O(n)

\*/

private void dfs(TreeNode root, int pathSum, int targetSum, Map<Integer, Integer> prefixSum, int[] count) {

if (root == null) return;

pathSum = root.val + pathSum;

if (prefixSum.containsKey(pathSum - targetSum)) {

count[0] += prefixSum.get(pathSum - targetSum);

}

prefixSum.put(pathSum, prefixSum.getOrDefault(pathSum, 0) + 1);

dfs(root.left, pathSum, targetSum, prefixSum, count);

dfs(root.right, pathSum, targetSum, prefixSum, count);

//prefixSum at least have key of pathSum with value 1, just remove by deducting 1

prefixSum.put(pathSum, prefixSum.get(pathSum) - 1);

}

# 96. Unique Binary Search Trees

class Solution {

/\*\*

\* 1, 2, 3, 4 .... n

\* root = 1 root.left = null and root.right = [2, .... n]

\* root = 2 root.left = [1] and root.right = [3, 4, ...n]

\* root = 3 root.left = [1, 2] and root.right = [4, ...n]

\* ...

\* root = i root.left = [1, .. i -1] and root.right = [i+1 .. n]

\* ...

\* root = n root.left = [1, ... n-1] and root.right = null

\*

\* f(n) = f(0) \* f(n - 1) + f(1)\*f(n-2) + f(2)\*f(n - 2) +......+ f(n-1) \* f(0)

\* unit case f(0) = 1 [multiple] f(1) = 1

\*\*/

public int numTrees(int n) {

if (n < 1) {

throw new IllegalStateException();

}

int[] dp = new int[n+1];

dp[0] = 1;

dp[1] = 1;

for (int i = 2; i <= n; i++) {

int start = 0;

int end = i - 1;

while (start <= end) {

if (start == end) {

dp[i] += (dp[start]\*dp[end]);

} else {

dp[i] += (dp[start]\*dp[end]\*2);

}

start++;

end--;

}

}

return dp[n];

}

}

# 1120. Maximum Average Subtree

| class Solution {  public double maximumAverageSubtree(TreeNode root) {  double[] maxAvg = {Integer.MIN\_VALUE};  helper(root, maxAvg);  return maxAvg[0];  }    private double[] helper(TreeNode root, double[] maxAvg) {  if (root == null) {  return new double[]{0.0, 0.0};  }  double[] left, right;  left = helper(root.left, maxAvg);  right = helper(root.right, maxAvg);  double counts = left[0] + right[0] + 1.0;  double sum = left[1] + right[1] + (double) root.val;  maxAvg[0] = Math.max(maxAvg[0], (double) sum / counts);  return new double[]{counts, sum};  } } |
| --- |

# 1339. Maximum Product of Splitted Binary Tree

| class Solution { // tc/sc: O(n)  private static final int MOD = 1000000007;  public int maxProduct(TreeNode root) {  List<Integer> sumVals = new ArrayList<>();  long totSum = helper(root, sumVals);    long maxProd = Integer.MIN\_VALUE;  for (int sum : sumVals) {  maxProd = Math.max(maxProd, sum \* (totSum - sum));  }  return (int) (maxProd % MOD);  }    private int helper(TreeNode root, List<Integer> sumVals) {  if (root == null) {  return 0;  }  int left = helper(root.left, sumVals);  int right = helper(root.right, sumVals);  int curSum = left + right + root.val;  sumVals.add(curSum);  return curSum % MOD;  } } |
| --- |

# 151 Reverse Words in a String

//luo

step 1: remove spaces

step 2: reverse overall

step 3: reverse each word

// TC: O(n) SC: O(1) 除了换char array外没有额外空间

class Solution {  
 public String reverseWords(String s) {  
 char[] array = s.toCharArray();  
 int slow = 0; // 0..slow - 1 to keep\*  
 for (int i = 0; i < array.length; i++) { // fast pointer  
 char cur = array[i];  
 if (cur != ' ') {  
 array[slow++] = cur;  
 } else if (i >= 1 && array[i - 1] != ' ') {  
 array[slow++] = cur;  
 }  
 }  
 if (slow - 1 < array.length && array[slow - 1] == ' ') { //1. 这里是slow-1，不是slow  
 slow--;  
 }  
   
 int start = 0;  
 int end = 0;  
 for (int i = 0; i < slow; i++) {  
 if (i == 0 || array[i - 1] == ' ') {  
 start = i;  
 }  
 if (i == slow - 1 || array[i + 1] == ' ') {  
 end = i;  
 reverse(array, start, end);  
 }  
 }  
 reverse(array, 0, slow - 1);  
 return new String(array, 0, slow);  
 }  
 private void reverse(char[] array, int start, int end) {  
 while (start < end) { // 2.  
 char tmp = array[start];  
 array[start] = array[end];  
 array[end] = tmp;   
 start++;  
 end--;  
 }  
  
 }  
}

# 152 Maximum Product Subarray

//luo

// TC: O(n) SC:O(1)

| class Solution {  public int maxProduct(int[] nums) {  int n = nums.length; // int[] dpMax = new int[n]; // int[] dpMin = new int[n];   // dpMax[0] = nums[0]; // dpMin[0] = nums[0];    int dpMax = nums[0];  int dpMin = nums[0];  int max = nums[0];    for (int i = 1; i < n; i++) {  // dpMax[i] = Math.max(nums[i], Math.max(dpMax[i - 1] \* nums[i], dpMin[i - 1] \* nums[i]));  // dpMin[i] = Math.min(nums[i], Math.min(dpMin[i - 1] \* nums[i], dpMax[i - 1] \* nums[i]));  int tmpMax = dpMax;  dpMax = Math.max(nums[i], Math.max(dpMax \* nums[i], dpMin \* nums[i]));  dpMin = Math.min(nums[i], Math.min(tmpMax \* nums[i], dpMin \* nums[i]));  max = Math.max(max, dpMax);  }  return max;  } } |
| --- |

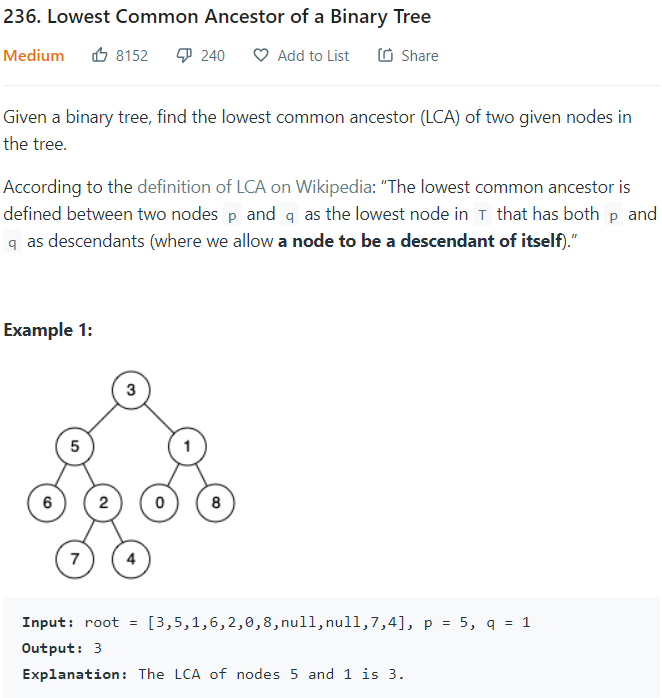
# 158 Read N Characters Given read4 II - Call Multiple Times

# Lowest Common Ancestors 专题

|  | two existed nodes | two nodes | tree node w parent | k nodes LCA |
| --- | --- | --- | --- | --- |
| 236 | x |  |  |  |
| 1644 |  | x |  |  |
| 1650 |  |  | x |  |
| 1676 |  |  |  | x |

## [236. Lowest Common Ancestor of a Binary Tree](https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/)

(Tang)



| class Solution { // tc/sc: o(N)  public TreeNode lowestCommonAncestor0(TreeNode root, TreeNode p, TreeNode q) {  if(root == null || root == p || root == q) return root;  TreeNode left = lowestCommonAncestor(root.left, p, q);  TreeNode right = lowestCommonAncestor(root.right, p, q);  if(left != null && right != null) return root;  return left != null ? left : right;  }   public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {  // Stack for tree traversal  Deque<TreeNode> stack = new ArrayDeque<>();   // 1. HashMap for parent pointers  Map<TreeNode, TreeNode> parent = new HashMap<>();   parent.put(root, null);  stack.push(root);   // Iterate until we find both the nodes p and q  while (!parent.containsKey(p) || !parent.containsKey(q)) {   TreeNode node = stack.pop();   // While traversing the tree, keep saving the parent pointers.  if (node.left != null) {  parent.put(node.left, node);  stack.push(node.left);  }  if (node.right != null) {  parent.put(node.right, node);  stack.push(node.right);  }  }   // 2. Ancestors set() for node p.  Set<TreeNode> ancestors = new HashSet<>();   // Process all ancestors for node p using parent pointers.  while (p != null) {  ancestors.add(p);  p = parent.get(p);  }   // 3. The first ancestor of q which appears in  // p's ancestor set() is their lowest common ancestor.  while (!ancestors.contains(q))  q = parent.get(q);  return q;  }  } |
| --- |

## [1644. Lowest Common Ancestor of a Binary Tree II](https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree-ii/)

| /\* https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree-ii/discuss/933835/Java.-Difference-from-236-is-you-need-to-search-the-entire-tree.  Time Complexity: O(N) Space Complexity: O(H), H is the height of the tree  This question is similar to 236. Last Common Ancestor of Binary Tree. Question 236 has two important premises:  1. It is guaranteed that both p and q are in the tree.  2. A node can be a descendant of itself.  But for this question, the premises are different:  It is NOT guaranteed that both p and q are in the tree. A node can still be a descendant of itself. Hence,  We need a way to record if we've seen both p and q We need to traverse the entire tree even after we've found one of them. \*/  class Solution0 {  boolean pFound = false;  boolean qFound = false;   public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {  TreeNode LCA = LCA(root, p, q);  return pFound && qFound ? LCA : null;  }    public TreeNode LCA(TreeNode root, TreeNode p, TreeNode q) {  if (root == null) return root;  TreeNode left = LCA(root.left, p, q);   TreeNode right = LCA(root.right, p, q);  if (root == p) {  pFound = true;  return root;  }  if (root == q) {  qFound = true;  return root;  }  return left == null ? right : right == null ? left : root;  } }   class Solution {  int count = 0;    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {  TreeNode LCA = LCA(root, p, q);  return count == 2 ? LCA : null;  }    public TreeNode LCA(TreeNode root, TreeNode p, TreeNode q) {  if (root == null) return root;  TreeNode left = LCA(root.left, p, q);   TreeNode right = LCA(root.right, p, q);  if (root == p || root == q) {  count++;  return root;  }  return left == null ? right : right == null ? left : root;  } } |
| --- |

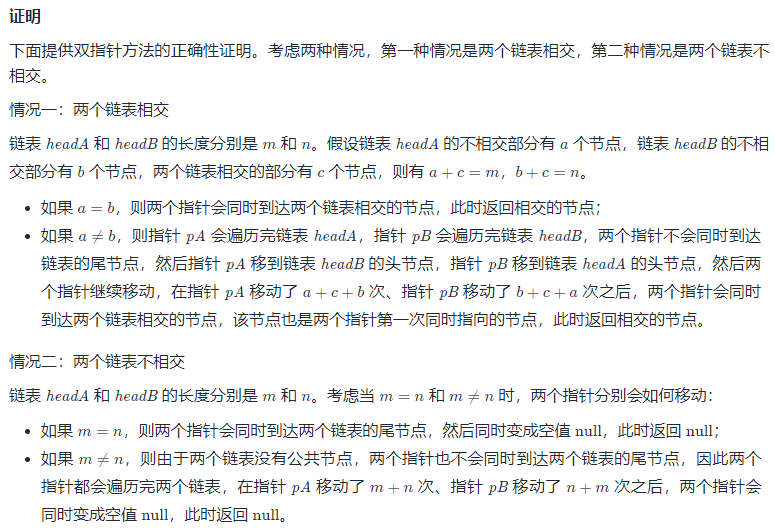
| /\* Iterative solution \*/  class Solution {  public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {  Map<TreeNode, TreeNode> parents = new HashMap<>();   Stack<TreeNode> stack = new Stack<>();  parents.put(root, null);   stack.push(root);  while (!stack.isEmpty() && (!parents.containsKey(p) || !parents.containsKey(q))) {  TreeNode curr = stack.pop();  if (curr.left != null) {  parents.put(curr.left, curr); stack.push(curr.left);  }  if (curr.right != null) {  parents.put(curr.right, curr); stack.push(curr.right);  }  }  if(!parents.containsKey(q) || !parents.containsKey(q)){  return null;  }  Set<TreeNode> pAns = new HashSet<>();  while (p != null) {  pAns.add(p);  p = parents.get(p);  }  while (!pAns.contains(q)) {  q = parents.get(q);  if (q == null) break; // prevent the infinite loop  }  return q;  } } |
| --- |

## [1650. Lowest Common Ancestor of a Binary Tree III](https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree-iii/)

| /\* // Definition for a Node. class Node {  public int val;  public Node left;  public Node right;  public Node parent; }; \*/ |
| --- |

| class Solution {  public Node lowestCommonAncestor(Node p, Node q) {  int pDepth = getDepth(p);   int qDepth = getDepth(q);    Node x = pDepth < qDepth ? p : q;  Node y = pDepth < qDepth ? q : p;    int diff = Math.abs(qDepth - pDepth);  while(diff-- > 0) {  y = y.parent;  }    while(x != y) {  x = x.parent;  y = y.parent;  }    return x;  }    private int getDepth(Node node) {  if (node == null) return 0;  int count = 0;   while(node != null) {  node = node.parent;  count++;  }    return count;  } } |
| --- |

| class Solution {  public Node lowestCommonAncestor0(Node p, Node q) {  Set<Node> set = new HashSet();  while(p!=null){  if(set.contains(p)) return p;  set.add(p);  p = p.parent;  Node t = p;  p = q;  q = t;  }    while(q!=null){  if(set.contains(q)) return q;  // set.add(q);  q = q.parent;  }  return null;  }  /\* Proofs: Leetcode 160, two pointers \*/  public Node lowestCommonAncestor(Node p, Node q) {  Node a = p, b = q;  while (a != b) {  a = a == null? q : a.parent;  b = b == null? p : b.parent;   }  return a;  } } |
| --- |



| class Solution { // DFS  public Node lowestCommonAncestor(Node p, Node q) {  Node root = p;  while(root.parent != null) {  root = root.parent;  }  return helper(root, p, q);  }  Node helper(Node root, Node p, Node q) {  if (root == null) return null;  if (root == p || root == q) return root;    Node left = helper(root.left, p, q);  Node right = helper(root.right, p, q);  if (left != null && right != null) return root;  if (left == null && right != null) return right;  if (left != null && right == null) return left;  return null;  } } |
| --- |

## [1676. Lowest Common Ancestor of a Binary Tree IV](https://leetcode.ca/2020-07-02-1676-Lowest-Common-Ancestor-of-a-Binary-Tree-IV/)

| class Solution { // need iterative method  public TreeNode lowestCommonAncestor(TreeNode root, TreeNode[] nodes) {  Set<TreeNode> set = new HashSet<TreeNode>();  for (TreeNode n : nodes) set.add(n);  return helper(root, set);  }  private TreeNode helper(TreeNode root, Set<TreeNode> set) {  // base case  if (root == null || set.contains(root)) {  return root;  }  TreeNode lr = helper(root.left, set);  TreeNode rr = helper(root.right, set);  if (lr != null && rr != null) {  return root;  }  return lr != null ? lr : rr;  } } |
| --- |