139 Word Break

class Solution {

public boolean wordBreak(String s, List<String> wordDict) {

if(s == null){

return false;

}

if(wordDict.size() == 0){

return false;

}

HashSet<String> set = new HashSet<String>();

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

set.add(wordDict.get(i));

}

boolean[] canBreak = new boolean[s.length() + 1];

canBreak[0] = true;

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

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

if(canBreak[j] && set.contains(s.substring(j,i + 1))){

canBreak[i + 1] = true;

}

}

}

return canBreak[s.length()];

}

}

160 intersection of two linked list

public class Solution {

public ListNode getIntersectionNode(ListNode headA, ListNode headB) {

if (headA == null || headB == null) {

return null;

}

// get the tail of list A.

ListNode node = headA;

while (node.next != null) {

node = node.next;

}

node.next = headB;

ListNode result = listCycleII(headA);

node.next = null;

return result;

}

private ListNode listCycleII(ListNode head) {

ListNode slow = head, fast = head.next;

while (slow != fast) {

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

return null;

}

slow = slow.next;

fast = fast.next.next;

}

slow = head;

fast = fast.next;

while (slow != fast) {

slow = slow.next;

fast = fast.next;

}

return slow;

}

}

56 Merge Interval

class Solution {

public List<Interval> merge(List<Interval> intervals) {

ArrayList<Interval> result = new ArrayList<>();

if(intervals.size() == 0){

return result;

}

Collections.sort(intervals, new Comparator<Interval>(){

public int compare(Interval v1, Interval v2){

return v1.start - v2.start;

}

});

int start = intervals.get(0).start;

int end = intervals.get(0).end;

for(int i = 1; i < intervals.size(); i++){

if(intervals.get(i).start <= end){

start = Math.min(intervals.get(i).start,start);

end = Math.max(intervals.get(i).end, end);

} else{

result.add(new Interval(start,end));

start = intervals.get(i).start;

end = intervals.get(i).end;

}

}

result.add(new Interval(start,end));

return result;

}

}

257 binary tree path

class Solution {

public List<String> binaryTreePaths(TreeNode root) {

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

if(root == null){

return result;

}

traverse(result, root, "");

return result;

}

private void traverse(ArrayList<String> result, TreeNode root, String subset){

subset += root.val + "->";

if(root.left == null && root.right == null){

subset = subset.substring(0, subset.length() - 2);

result.add(subset);

return;

} else if(root.left != null && root.right == null ){

traverse(result, root.left, subset);

} else if(root.left == null && root.right != null){

traverse(result, root.right, subset);

} else {

traverse(result, root.left, subset);

traverse(result, root.right, subset);

}

}

}

477 total hamming distance

class Solution {

public int totalHammingDistance(int[] nums) {

if(nums.length == 0){

return 0;

}

int[] hammingDis = new int[32];

int result = 0;

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

for(int j = 0; j < 32; j++){

int k = (nums[i] & 1);

hammingDis[j] += k;

nums[i] >>= 1;

}

}

for(int i = 0;i < 32;i++){

result += hammingDis[i] \* (nums.length - hammingDis[i]);

}

return result;

}

}

94 Inorder binary tree

class Solution {

public List<Integer> inorderTraversal(TreeNode root) {

if(root == null){

return new ArrayList<Integer>();

}

ArrayList<Integer> result = new ArrayList<>();

traverse(root, result);

return result;

}

private void traverse(TreeNode root, ArrayList<Integer> result){

if(root == null){

return ;

}

traverse(root.left, result);

result.add(root.val);

traverse(root.right,result);

}

}

31 next permutation

public class Solution {

public void swapItem(int[] nums, int i, int j) {

int temp = nums[i];

nums[i] = nums[j];

nums[j] = temp;

}

public void swapList(int[] nums, int i, int j) {

while (i < j) {

swapItem(nums, i, j);

i ++; j --;

}

}

public int[] nextPermutation(int[] nums) {

int len = nums.length;

if ( len <= 1)

return nums;

int i = len - 1;

while (i > 0 && nums[i] <= nums[i - 1])

i --;

swapList(nums, i, len - 1);

if (i != 0) {

int j = i;

while (nums[j] <= nums[i - 1]) j++;

swapItem(nums, j, i-1);

}

return nums;

}

}

34 Search for range

class Solution {

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

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

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

int start = 0;

int end = nums.length - 1;

int left = 0;

int right = 0;

//search for left bound

while(start + 1 < end){

int mid = start + (end - start) / 2;

if(nums[mid] == target){

end = mid;

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

start = mid;

} else {

end = mid;

}

}

if(nums[start] == target){

left = start;

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

left = end;

} else {

left = -1;

}

//search for right bound

start = 0;

end = nums.length - 1;

while(start + 1 < end){

int mid = start + (end - start) / 2;

if(nums[mid] == target){

start = mid;

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

start = mid;

} else{

end = mid;

}

}

if(nums[end] == target){

right = end;

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

right = start;

} else {

right = -1;

}

return new int[]{left, right};

}

}

114 flatten bst to linked list

public class Solution {

private TreeNode lastNode = null;

public void flatten(TreeNode root) {

if (root == null) {

return;

}

if (lastNode != null) {

lastNode.left = null;

lastNode.right = root;

}

lastNode = root;

TreeNode right = root.right;

flatten(root.left);

flatten(right);

}

}

non – recursive

public void flatten(TreeNode root) {

if (root == null) {

return;

}

Stack<TreeNode> stack = new Stack<>();

stack.push(root);

while (!stack.empty()) {

TreeNode node = stack.pop();

if (node.right != null) {

stack.push(node.right);

}

if (node.left != null) {

stack.push(node.left);

}

// connect

node.left = null;

if (stack.empty()) {

node.right = null;

} else {

node.right = stack.peek();

}

}

}

BST to double linked list

348. Design Tic-Tac-Toe

public class TicTacToe {

private int[] rows;

private int[] cols;

private int diagonal;

private int antiDiagonal;

/\*\* Initialize your data structure here. \*/

public TicTacToe(int n) {

rows = new int[n];

cols = new int[n];

}

/\*\* Player {player} makes a move at ({row}, {col}).

@param player The player, can be either 1 or 2.

@return The current winning condition, can be either:

0: No one wins.

1: Player 1 wins.

2: Player 2 wins. \*/

public int move(int row, int col, int player) {

int toAdd = player == 1 ? 1 : -1;

rows[row] += toAdd;

cols[col] += toAdd;

if (row == col)

{

diagonal += toAdd;

}

if (col == (cols.length - row - 1))

{

antiDiagonal += toAdd;

}

int size = rows.length;

if (Math.abs(rows[row]) == size ||

Math.abs(cols[col]) == size ||

Math.abs(diagonal) == size ||

Math.abs(antiDiagonal) == size)

{

return player;

}

return 0;

}

}

157 Read N Characters Given Read4

public class Solution extends Reader4 {

/\*\*

\* @param buf Destination buffer

\* @param n Maximum number of characters to read

\* @return The number of characters read

\*/

public int read(char[] buf, int n) {

//1. n < 4

//2. n == 4

//3. n > 4

int index = 0;

while(n > 0){

int size = n > 4 ? 4 : n;

char[] cArray = new char[4];

int count = read4(cArray);

size = size > count ? count : size;

for(int i = 0; i < size; i++, index++){

buf[index] = cArray[i];

}

n -= count;

if(count < 4){

break;

}

}

return index;

}

}

42 trap rain water

public class Solution {

/\*\*

\* @param heights: an array of integers

\* @return: a integer

\*/

public int trapRainWater(int[] heights) {

// write your code here

int left = 0, right = heights.length - 1;

int res = 0;

if(left >= right)

return res;

int leftheight = heights[left];

int rightheight = heights[right];

while(left < right) {

if(leftheight < rightheight) {

left ++;

if(leftheight > heights[left]) {

res += (leftheight - heights[left]);

} else {

leftheight = heights[left];

}

} else {

right --;

if(rightheight > heights[right]) {

res += (rightheight - heights[right]);

} else {

rightheight = heights[right];

}

}

}

return res;

}

}

407 trap rain water 2

class Cell{

public int x,y, h;

Cell(){}

Cell(int xx,int yy, int hh){

x = xx;

y = yy;

h = hh;

}

}

class CellComparator implements Comparator<Cell> {

@Override

public int compare(Cell x, Cell y)

{

if(x.h > y.h)

return 1;

else if(x.h == y.h){

return 0;

}

else {

return -1;

}

}

}

public class Solution {

int []dx = {1,-1,0,0};

int []dy = {0,0,1,-1};

public int trapRainWater(int[][] heights) {

// write your code here

if(heights.length == 0)

return 0;

PriorityQueue<Cell> q = new PriorityQueue<Cell>(new CellComparator());

int n = heights.length;

int m = heights[0].length;

int [][]visit = new int[n][m];

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

q.offer(new Cell(i,0,heights[i][0]));

q.offer(new Cell(i,m-1,heights[i][m-1]));

visit[i][0] = 1;

visit[i][m-1] = 1;

}

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

q.offer(new Cell(0,i,heights[0][i]));

q.offer(new Cell(n-1,i,heights[n-1][i]));

visit[0][i] = 1;

visit[n-1][i] = 1;

}

int ans = 0 ;

while(!q.isEmpty()) {

Cell now = q.poll();

for(int i = 0; i < 4; i++) {

int nx = now.x + dx[i];

int ny = now.y + dy[i];

if(0<=nx && nx < n && 0 <= ny && ny < m && visit[nx][ny] == 0) {

visit[nx][ny] = 1;

q.offer(new Cell(nx,ny,Math.max(now.h,heights[nx][ny])));

ans = ans + Math.max(0,now.h - heights[nx][ny]);

}

}

}

return ans;

}

}

49 Group Anagrams

class Solution {

public List<List<String>> groupAnagrams(String[] strs) {

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

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

return result;

}

HashMap<String, ArrayList<String>> map = new HashMap<>();

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

char[] array = strs[i].toCharArray();

Arrays.sort(array);

String s = String.valueOf(array);

ArrayList<String> list;

if(map.containsKey(s)){

list = map.get(s);

} else {

list = new ArrayList<>();

}

list.add(strs[i]);

map.put(s, list);

}

for(ArrayList<String> list : map.values()){

result.add(list);

}

return result;

}

}

210 course schedule 2

public class Solution {

public int[] findOrder(int numCourses, int[][] prerequisites) {

// Write your code here

List[] edges = new ArrayList[numCourses];

int[] degree = new int[numCourses];

for (int i = 0;i < numCourses; i++)

edges[i] = new ArrayList<Integer>();

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

degree[prerequisites[i][0]] ++ ;

edges[prerequisites[i][1]].add(prerequisites[i][0]);

}

Queue queue = new LinkedList();

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

if (degree[i] == 0) {

queue.add(i);

}

}

int count = 0;

int[] order = new int[numCourses];

while(!queue.isEmpty()){

int course = (int)queue.poll();

order[count] = course;

count ++;

int n = edges[course].size();

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

int pointer = (int)edges[course].get(i);

degree[pointer]--;

if (degree[pointer] == 0) {

queue.add(pointer);

}

}

}

if (count == numCourses)

return order;

return new int[0];

}

}

46 permutation

class Solution {

public List<List<Integer>> permute(int[] nums) {

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

if(nums == null){

return result;

}

if(nums.length == 0){

result.add(new ArrayList<Integer>());

return result;

}

// nums, result, subset, set

ArrayList<Integer> permutation = new ArrayList<>();

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

helper(nums, result, permutation, set);

return result;

}

private void helper(int[] nums,

ArrayList<List<Integer>> result,

ArrayList<Integer> permutation,

HashSet<Integer> set){

if(permutation.size() == nums.length){

result.add(new ArrayList<Integer>(permutation));

return;

}

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

if(!set.add(nums[i])){

continue;

}

permutation.add(nums[i]);

helper(nums, result, permutation, set);

set.remove(nums[i]);

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

}

}

}

282 expression add operators

class Solution {

public:

vector<string> addOperators(string num, int target) {

vector<string> res;

addOperatorsDFS(num, target, 0, 0, "", res);

return res;

}

void addOperatorsDFS(string num, int target, long long diff, long long curNum, string out, vector<string> &res) {

if (num.size() == 0 && curNum == target) {

res.push\_back(out);

}

for (int i = 1; i <= num.size(); ++i) {

string cur = num.substr(0, i);

if (cur.size() > 1 && cur[0] == '0') return;

string next = num.substr(i);

if (out.size() > 0) {

addOperatorsDFS(next, target, stoll(cur), curNum + stoll(cur), out + "+" + cur, res);

addOperatorsDFS(next, target, -stoll(cur), curNum - stoll(cur), out + "-" + cur, res);

addOperatorsDFS(next, target, diff \* stoll(cur), (curNum - diff) + diff \* stoll(cur), out + "\*" + cur, res);

} else {

addOperatorsDFS(next, target, stoll(cur), stoll(cur), cur, res);

}

}

}

};

108 Convert Sorted Array to Binary Search Tree

class Solution {

public TreeNode sortedArrayToBST(int[] nums) {

if(nums.length == 0){

return null;

}

int start = 0;

int end = nums.length - 1;

TreeNode root = generateTree(start,end,nums);

return root;

}

private TreeNode generateTree(int start, int end, int[]nums){

if(start > end){

return null;

}

int mid = start + (end - start) / 2;

TreeNode n = new TreeNode(nums[mid]);

n.left = generateTree(start, mid - 1, nums);

n.right = generateTree(mid + 1, end, nums);

return n;

}

}

639 Decode ways ii

public class Solution {

public int numDecodings(String s) {

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

return 0;

}

final int mod = 1000000007;

int n = s.length();

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

f[0] = 1;

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

f[i] = 0;

if (s.charAt(i - 1) == '\*') {

f[i] = (int)((f[i] + 9L \* f[i - 1]) % mod);

if (i >= 2) {

if (s.charAt(i - 2) == '\*') {

f[i] = (int)((f[i] + 15L \* f[i - 2]) % mod);

}

else if (s.charAt(i - 2) == '1') {

f[i] = (int)((f[i] + 9L \* f[i - 2]) % mod);

}

else if (s.charAt(i - 2) == '2') {

f[i] = (int)((f[i] + 6L \* f[i - 2]) % mod);

}

}

}

else {

if (s.charAt(i - 1) != '0') {

f[i] = (f[i] + f[i - 1]) % mod;

}

if (i >= 2) {

if (s.charAt(i - 2) == '\*'){

if (s.charAt(i - 1) <= '6') {

f[i] = (int)((f[i] + 2L \* f[i - 2]) % mod);

}

else {

f[i] = (f[i] + f[i - 2]) % mod;

}

}

else {

int twoDigits = (s.charAt(i - 2) - '0') \* 10 + s.charAt(i - 1) - '0';

if (twoDigits >= 10 && twoDigits <= 26) {

f[i] = (f[i] + f[i - 2]) % mod;

}

}

}

}

}

return f[n];

}

}

141 Linked List cycle

public class Solution {

public boolean hasCycle(ListNode head) {

if(head == null){

return false;

}

ListNode slow = head;

ListNode fast = head;

while(fast!= null && fast.next !=null){

slow = slow.next;

fast = fast.next.next;

if(slow == fast){

return true;

}

}

return false;

}

}

124 Binary Tree Maximum Path Sum

public class Solution {

private class ResultType {

// singlePath: 从root往下走到任意点的最大路径，这条路径可以不包含任何点

// maxPath: 从树中任意到任意点的最大路径，这条路径至少包含一个点

int singlePath, maxPath;

ResultType(int singlePath, int maxPath) {

this.singlePath = singlePath;

this.maxPath = maxPath;

}

}

private ResultType helper(TreeNode root) {

if (root == null) {

return new ResultType(0, Integer.MIN\_VALUE);

}

// Divide

ResultType left = helper(root.left);

ResultType right = helper(root.right);

// Conquer

int singlePath = Math.max(left.singlePath, right.singlePath) + root.val;

singlePath = Math.max(singlePath, 0);

int maxPath = Math.max(left.maxPath, right.maxPath);

maxPath = Math.max(maxPath, left.singlePath + right.singlePath + root.val);

return new ResultType(singlePath, maxPath);

}

public int maxPathSum(TreeNode root) {

ResultType result = helper(root);

return result.maxPath;

}

}

398 random pick index

class Solution {

int[] nums;

Random random;

public Solution(int[] nums) {

this.nums= nums;

this.random = new Random();

}

public int pick(int target) {

int count = 0;

int result = 0;

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

if(nums[i] != target){

continue;

} else {

count++;

if(random.nextInt(count) == 0){

result = i;

}

}

}

return result;

}

}

20 valid parentheses

public boolean isValidParentheses(String s) {

// Write your code here

Stack<Character> stack = new Stack<>();

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

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

stack.push(c);

}

if (c == ')') {

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

return false;

}

}

if (c == ']') {

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

return false;

}

}

if (c == '}') {

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

return false;

}

}

}

return stack.isEmpty();

}

}

102 Binary Tree Level Order Traversal

class Solution {

public List<List<Integer>> levelOrder(TreeNode root) {

if(root == null){

return new ArrayList<List<Integer>>();

}

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

Queue<TreeNode> queue = new LinkedList<>();

queue.offer(root);

while(!queue.isEmpty()){

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

int size = queue.size();

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

TreeNode t = queue.poll();

if(t.left != null){

queue.offer(t.left);

}

if(t.right != null){

queue.offer(t.right);

}

list.add(t.val);

}

result.add(list);

}

return result;

}

}

143 reorder the list

public class Solution {

private ListNode reverse(ListNode head) {

ListNode newHead = null;

while (head != null) {

ListNode temp = head.next;

head.next = newHead;

newHead = head;

head = temp;

}

return newHead;

}

private void merge(ListNode head1, ListNode head2) {

int index = 0;

ListNode dummy = new ListNode(0);

while (head1 != null && head2 != null) {

if (index % 2 == 0) {

dummy.next = head1;

head1 = head1.next;

} else {

dummy.next = head2;

head2 = head2.next;

}

dummy = dummy.next;

index ++;

}

if (head1 != null) {

dummy.next = head1;

} else {

dummy.next = head2;

}

}

private ListNode findMiddle(ListNode head) {

ListNode slow = head, fast = head.next;

while (fast != null && fast.next != null) {

fast = fast.next.next;

slow = slow.next;

}

return slow;

}

public void reorderList(ListNode head) {

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

return;

}

ListNode mid = findMiddle(head);

ListNode tail = reverse(mid.next);

mid.next = null;

merge(head, tail);

}

}

654 maximum bst

class Solution {

public TreeNode constructMaximumBinaryTree(int[] nums) {

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

return null;

}

int left = 0;

int right = nums.length - 1;

TreeNode root = helper(nums, left, right);

return root;

}

private TreeNode helper(int[] nums, int left, int right){

if(left > right){

return null;

}

int max = left;

for(int i = left + 1;i <= right; i++){

max = nums[max] > nums[i] ? max : i;

}

TreeNode root = new TreeNode(nums[max]);

root.left = helper(nums, left, max - 1);

root.right = helper(nums, max + 1, right);

return root;

}

}

71 simplify the path

class Solution {

public String simplifyPath(String path) {

String result = "/";

String[] stubs = path.split("/+");

ArrayList<String> paths = new ArrayList<String>();

for (String s : stubs){

if(s.equals("..")){

if(paths.size() > 0){

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

}

}

else if (!s.equals(".") && !s.equals("")){

paths.add(s);

}

}

for (String s : paths){

result += s + "/";

}

if (result.length() > 1)

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

return result;

}

}

88 Merge the sorted array

class Solution {

public void merge(int[] nums1, int m, int[] nums2, int n) {

if(nums1.length == 0){

nums1 = nums2;

return;

} else if(nums2.length == 0){

return;

}

while(m > 0 || n > 0){

int a = m - 1 >= 0 ? nums1[m - 1] : Integer.MIN\_VALUE;

int b = n - 1 >= 0 ? nums2[n - 1] : Integer.MIN\_VALUE;

if(a >= b){

nums1[m + n - 1] = a;

m--;

} else {

nums1[m + n - 1] = b;

n--;

}

}

}

}

41 First Missing Positive

public class Solution {

public int firstMissingPositive(int[] A) {

if (A == null) {

return 1;

}

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

while (A[i] > 0 && A[i] <= A.length && A[i] != (i+1)) {

int tmp = A[A[i]-1];

if (tmp == A[i]) {

break;

}

A[A[i]-1] = A[i];

A[i] = tmp;

}

}

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

if (A[i] != i + 1) {

return i + 1;

}

}

return A.length + 1;

}

}

261 graph valid tree

class Solution {

public boolean validTree(int n, int[][] edges) {

if (n == 0) {

return false;

}

if (edges.length != n - 1) {

return false;

}

Map<Integer, Set<Integer>> graph = initializeGraph(n, edges);

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

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

queue.offer(0);

hash.add(0);

while (!queue.isEmpty()) {

int node = queue.poll();

for (Integer neighbor : graph.get(node)) {

if (hash.contains(neighbor)) {

continue;

}

hash.add(neighbor);

queue.offer(neighbor);

}

}

return (hash.size() == n);

}

private Map<Integer, Set<Integer>> initializeGraph(int n, int[][] edges) {

Map<Integer, Set<Integer>> graph = new HashMap<>();

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

graph.put(i, new HashSet<Integer>());

}

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

int u = edges[i][0];

int v = edges[i][1];

graph.get(u).add(v);

graph.get(v).add(u);

}

return graph;

}

}

23 merge k sorted list

public class Solution {

private Comparator<ListNode> ListNodeComparator = new Comparator<ListNode>() {

public int compare(ListNode left, ListNode right) {

return left.val - right.val;

}

};

public ListNode mergeKLists(List<ListNode> lists) {

if (lists == null || lists.size() == 0) {

return null;

}

Queue<ListNode> heap = new PriorityQueue<ListNode>(lists.size(), ListNodeComparator);

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

if (lists.get(i) != null) {

heap.add(lists.get(i));

}

}

ListNode dummy = new ListNode(0);

ListNode tail = dummy;

while (!heap.isEmpty()) {

ListNode head = heap.poll();

tail.next = head;

tail = head;

if (head.next != null) {

heap.add(head.next);

}

}

return dummy.next;

}

}

689 Maximum Sum of 3 Non-Overlapping Subarrays

class Solution {

public int[] maxSumOfThreeSubarrays(int[] nums, int k) {

int n = nums.length;

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

int[] left = new int[n];

int[] right = new int[n];

int[] ret = new int[3];

// First get the prefix sum of nums.

// Prefix sum enables us to get the sum of k consecutive element in O(1) time

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

sum[i + 1] = sum[i] + nums[i];

}

// DP for the left intetval max sum

for (int i = k, tot = sum[k] - sum[0]; i < n; i++) {

if (sum[i + 1] - sum[i - k + 1] > tot) {

tot = sum[i + 1] - sum[i - k + 1];

left[i] = i - k + 1;

} else {

left[i] = left[i - 1];

}

}

// DP for the right interval max sum

right[n - k] = n - k;

for (int i = n - 1 - k, tot = sum[n] - sum[n - k]; i >= 0; i--) {

if (sum[i + k] - sum[i] >= tot) {

tot = sum[i + k] - sum[i];

right[i] = i;

} else {

right[i] = right[i + 1];

}

}

// Find the max sum by iterating through the middle interval index based on above 2 cache.

int maxSum = 0;

for (int i = k; i <= n - 2 \* k; i++) {

int l = left[i - 1], r = right[i + k];

int tot = sum[l + k] - sum[l] + sum[r + k] - sum[r] + sum[i + k] - sum[i];

if (tot > maxSum) {

ret[0] = l;

ret[1] = i;

ret[2] = r;

maxSum = tot;

}

}

return ret;

}

}

209 Minimum Size Subarray Sum

class Solution {

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

if(s < 0){

return 0;

}

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

return 0;

}

int left = 0;

int right = 0;

int sum = 0;

int minLen = 0;

while(right < nums.length){

while(sum < s && right < nums.length){

sum += nums[right];

right++;

}

while(sum >= s){

if(minLen == 0){

minLen = right - left;

}

minLen = Math.min(minLen, right - left);

sum -= nums[left];

left++;

}

}

return minLen;

}

}

100 same tree

public boolean isSameTree(TreeNode p, TreeNode q) {

if(p == null && q == null){

return true;

} else if(p == null || q == null){

return false;

} else {

boolean left = isSameTree(p.left, q.left);

boolean right = isSameTree(p.right, q.right);

boolean result;

if(p.val == q.val) {

result = true;

} else {

result = false;

}

if(left == true && right == true && result == true) {

return true;

} else if(left == false || right == false || result == false){

return false;

}

return false;

}

}

259 3sum smaller

class Solution {

public:

int threeSumSmaller(vector<int>& nums, int target) {

if (nums.size() < 3) return 0;

int res = 0;

sort(nums.begin(), nums.end());

for (int i = 0; i < nums.size() - 2; ++i) {

int left = i + 1, right = nums.size() - 1;

while (left < right) {

if (nums[i] + nums[left] + nums[right] < target) {

res += right - left;

++left;

} else {

--right;

}

}

}

return res;

}

};

81 Search in Rotated Sorted Array II

public class Solution {

// 这个问题在面试中不会让实现完整程序

// 只需要举出能够最坏情况的数据是 [1,1,1,1... 1] 里有一个0即可。

// 在这种情况下是无法使用二分法的，复杂度是O(n)

// 因此写个for循环最坏也是O(n)，那就写个for循环就好了

// 如果你觉得，不是每个情况都是最坏情况，你想用二分法解决不是最坏情况的情况，那你就写一个二分吧。

// 反正面试考的不是你在这个题上会不会用二分法。这个题的考点是你想不想得到最坏情况。

public boolean search(int[] A, int target) {

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

if (A[i] == target) {

return true;

}

}

return false;

}

}

340 Longest Substring with At Most K Distinct Characters

class Solution {

public int lengthOfLongestSubstringKDistinct(String s, int k) {

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

return 0;

}

if(k <= 0){

return 0;

}

int left = 0;

int right = 0;

int count = 0;

int result = 0;

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

while(right < s.length()){

char c = s.charAt(right);

map.put(c, map.getOrDefault(c, 0) + 1);

if(map.get(c) == 1){

count++;

}

right++;

while(count > k){

char cTemp = s.charAt(left);

map.put(cTemp, map.get(cTemp) - 1);

if(map.get(cTemp) == 0){

count--;

}

left++;

}

result = Math.max(result, right - left);

}

return result;

}

}

173 Binary Search Tree Iterator

public class BSTIterator {

private Stack<TreeNode> stack = new Stack<>();

TreeNode next = null;

void AddNodeToStack(TreeNode root) {

while (root != null) {

stack.push(root);

root = root.left;

}

}

// @param root: The root of binary tree.

public BSTIterator(TreeNode root) {

next = root;

}

//@return: True if there has next node, or false

public boolean hasNext() {

if (next != null) {

AddNodeToStack(next);

next = null;

}

return !stack.isEmpty();

}

//@return: return next node

public TreeNode next() {

if (!hasNext()) {

return null;

}

TreeNode cur = stack.pop();

next = cur.right;

return cur;

}

}

4 Median of Two Sorted Arrays

public class Solution {

public double findMedianSortedArrays(int A[], int B[]) {

int len = A.length + B.length;

if (len % 2 == 1) {

return findKth(A, 0, B, 0, len / 2 + 1);

}

return (

findKth(A, 0, B, 0, len / 2) + findKth(A, 0, B, 0, len / 2 + 1)

) / 2.0;

}

// find kth number of two sorted array

public static int findKth(int[] A, int A\_start,

int[] B, int B\_start,

int k){

if (A\_start >= A.length) {

return B[B\_start + k - 1];

}

if (B\_start >= B.length) {

return A[A\_start + k - 1];

}

if (k == 1) {

return Math.min(A[A\_start], B[B\_start]);

}

int A\_key = A\_start + k / 2 - 1 < A.length

? A[A\_start + k / 2 - 1]

: Integer.MAX\_VALUE;

int B\_key = B\_start + k / 2 - 1 < B.length

? B[B\_start + k / 2 - 1]

: Integer.MAX\_VALUE;

if (A\_key < B\_key) {

return findKth(A, A\_start + k / 2, B, B\_start, k - k / 2);

} else {

return findKth(A, A\_start, B, B\_start + k / 2, k - k / 2);

}

}

}

153 Find Minimum in Rotated Sorted Array

class Solution {

public int findMin(int[] nums) {

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

return 0;

}

int start = 0;

int end = nums.length - 1;

while(start + 1 < end){

int mid = start + (end - start) / 2;

if(nums[mid] < nums[mid - 1]){

return nums[mid];

} else if(nums[mid] > nums[nums.length - 1]){

start = mid;

} else if(nums[mid] <= nums[nums.length - 1]){

end = mid;

}

}

if(nums[end] < nums[start]){

return nums[end];

}

return nums[start];

}

}

636 Exclusive Time of Functions

class Solution {

public:

vector<int> exclusiveTime(int n, vector<string>& logs) {

vector<int> res(n, 0);

stack<int> st;

int preTime = 0;

for (string log : logs) {

int found1 = log.find(":");

int found2 = log.find\_last\_of(":");

int idx = stoi(log.substr(0, found1));

string type = log.substr(found1 + 1, found2 - found1 - 1);

int time = stoi(log.substr(found2 + 1));

if (!st.empty()) {

res[st.top()] += time - preTime;

}

preTime = time;

if (type == "start") st.push(idx);

else {

auto t = st.top(); st.pop();

++res[t];

++preTime;

}

}

return res;

}

};