253 Meeting room 2

Given an array of meeting time intervals consisting of start and end times [[s1,e1],[s2,e2],...] (si < ei), find the minimum number of conference rooms required.

For example,

Given [[0, 30],[5, 10],[15, 20]],

return 2.

/\*\*

\* Definition for an interval.

\* public class Interval {

\* int start;

\* int end;

\* Interval() { start = 0; end = 0; }

\* Interval(int s, int e) { start = s; end = e; }

\* }

\*/

class Point{

int time;

int flag;

Point(int t, int s){

this.time = t;

this.flag = s;

}

public static Comparator<Point> PointComparator = new Comparator<Point>(){

public int compare(Point p1, Point p2){

if(p1.time == p2.time) return p1.flag - p2.flag;

else return p1.time - p2.time;

}

};

}

public class Solution {

public int minMeetingRooms(Interval[] intervals) {

List<Point> list = new ArrayList<>(intervals.length\*2);

for(Interval i : intervals){

list.add(new Point(i.start, 1));

list.add(new Point(i.end, 0));

}

Collections.sort(list,Point.PointComparator );

int count = 0, ans = 0;

for(Point p : list){

if(p.flag == 1) {

count++;

}

else {

count--;

}

ans = Math.max(ans, count);

}

return ans;

}

}

314. Binary Tree Vertical Order Traversal

class Solution {

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

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

if(root == null){

return result;

}

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

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

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

int min = 0; //min & max may not be 0

int max = 0;

queue.offer(root);

index.offer(0);

while(!queue.isEmpty()){ // zero not deal with separately, all will be done when it is poll();

TreeNode t = queue.poll();

int i = index.poll();

ArrayList list = null;

if(map.containsKey(i)){

list = map.get(i);

list.add(t.val);

map.put(i, list);

} else {

list = new ArrayList<>();

list.add(t.val);

map.put(i, list);

}

if(t.left != null){

int subIndex = i - 1;

min = Math.min(min, subIndex);

queue.offer(t.left);

index.offer(subIndex);

}

if(t.right != null){

int subIndex = i + 1;

max = Math.max(max, subIndex);

queue.offer(t.right);

index.offer(subIndex);

}

}

for(int i = min; i <= max; i++){

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

result.add(map.get(i));

}

}

return result;

}

}

297. Serialize and Deserialize Binary Tree

class Solution {

/\*\*

\* This method will be invoked first, you should design your own algorithm

\* to serialize a binary tree which denote by a root node to a string which

\* can be easily deserialized by your own "deserialize" method later.

\*/

public String serialize(TreeNode root) {

if (root == null) {

return "{}";

}

ArrayList<TreeNode> queue = new ArrayList<TreeNode>();

queue.add(root);

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

TreeNode node = queue.get(i);

if (node == null) {

continue;

}

queue.add(node.left);

queue.add(node.right);

}

while (queue.get(queue.size() - 1) == null) {

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

}

StringBuilder sb = new StringBuilder();

sb.append("{");

sb.append(queue.get(0).val);

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

if (queue.get(i) == null) {

sb.append(",#");

} else {

sb.append(",");

sb.append(queue.get(i).val);

}

}

sb.append("}");

return sb.toString();

}

/\*\*

\* This method will be invoked second, the argument data is what exactly

\* you serialized at method "serialize", that means the data is not given by

\* system, it's given by your own serialize method. So the format of data is

\* designed by yourself, and deserialize it here as you serialize it in

\* "serialize" method.

\*/

public TreeNode deserialize(String data) {

if (data.equals("{}")) {

return null;

}

String[] vals = data.substring(1, data.length() - 1).split(",");

ArrayList<TreeNode> queue = new ArrayList<TreeNode>();

TreeNode root = new TreeNode(Integer.parseInt(vals[0]));

queue.add(root);

int index = 0;

boolean isLeftChild = true;

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

if (!vals[i].equals("#")) {

TreeNode node = new TreeNode(Integer.parseInt(vals[i]));

if (isLeftChild) {

queue.get(index).left = node;

} else {

queue.get(index).right = node;

}

queue.add(node);

}

if (!isLeftChild) {

index++;

}

isLeftChild = !isLeftChild;

}

return root;

}

}

543. Diameter of Binary Tree

class Solution {

int max = 0;

public int diameterOfBinaryTree(TreeNode root) {

if(root == null){

return 0;

}

int result = helper(root); //this should write separately

return Math.max(max, result);

}

private int helper(TreeNode root){

if(root == null){

return -1;

}

int left = helper(root.left) + 1;

int right = helper(root.right) + 1;

max = Math.max(left + right, max);

return Math.max(left, right);

}

}

75 sort colors

class Solution {

public void sortColors(int[] nums) {

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

return ;

}

int left = 0;

int mid = 0;

int right = nums.length - 1;

while(left <= right){ // this should be left <= right, becasue right haven't done anything

if(nums[left] == 0){

if(left != mid){

System.out.println(left + " " +mid);

swap(nums, left, mid);

}

left++;

mid++;

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

left++;

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

swap(nums,left, right);

right--; // only right -- here

}

}

}

private void swap(int[] nums, int left, int right){

int temp = nums[right];

nums[right] = nums[left];

nums[left] = temp;

}

}

273 integer to English word

class Solution {

public:

string numberToWords(int num) {

string res = convertHundred(num % 1000);

vector<string> v = {"Thousand", "Million", "Billion"};

for (int i = 0; i < 3; ++i) {

num /= 1000;

res = num % 1000 ? convertHundred(num % 1000) + " " + v[i] + " " + res : res;

}

while (res.back() == ' ') res.pop\_back();

return res.empty() ? "Zero" : res;

}

string convertHundred(int num) {

vector<string> v1 = {"", "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Eleven", "Twelve", "Thirteen", "Fourteen", "Fifteen", "Sixteen", "Seventeen", "Eighteen", "Nineteen"};

vector<string> v2 = {"", "", "Twenty", "Thirty", "Forty", "Fifty", "Sixty", "Seventy", "Eighty", "Ninety"};

string res;

int a = num / 100, b = num % 100, c = num % 10;

res = b < 20 ? v1[b] : v2[b / 10] + (c ? " " + v1[c] : "");

if (a > 0) res = v1[a] + " Hundred" + (b ? " " + res : "");

return res;

}

};

133 Clone graph

public class Solution {

public UndirectedGraphNode cloneGraph(UndirectedGraphNode node) {

if(node == null){

return node;

}

HashMap<UndirectedGraphNode,UndirectedGraphNode> map = new HashMap<>();

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

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

queue.offer(node);

set.add(node);

while(!queue.isEmpty()){

UndirectedGraphNode n = queue.poll();

UndirectedGraphNode newNode = new UndirectedGraphNode(n.label);

map.put(n, newNode);

for(UndirectedGraphNode neighbor : n.neighbors){

if(!map.containsKey(neighbor)){

queue.offer(neighbor);

}

}

}

for(UndirectedGraphNode n : map.keySet()){

for(UndirectedGraphNode neighbor: n.neighbors){

UndirectedGraphNode newNeighbor = map.get(neighbor); // should have the new node not the old node!!!

map.get(n).neighbors.add(newNeighbor);

}

}

return map.get(node);

}

}

349 intersection of two array

class Solution {

public int[] intersection(int[] nums1, int[] nums2) {

if(nums1 == null || nums1.length == 0 || nums2 == null || nums2.length == 0){

return new int[0];

}

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

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

for(int i : nums1){

set.add(i);

}

for(int i : nums2){

if(set.contains(i)){

result.add(i);

}

}

int[] res = new int[result.size()];

int k = 0;

for(Integer i: result){

res[k] = i;

k++;

}

return res;

}

}

33 Search in a rotated sorted array

class Solution {

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

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

return -1;

}

int start = 0;

int end = nums.length - 1;

int mid = 0;

while(start + 1 < end){

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

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

if(nums[start] <= target && target <= nums[mid]){

end = mid;

} else {

start = mid;

}

} else {

if(nums[mid] <= target && target <= nums[end]){

start = mid;

} else {

end = mid;

}

}

}

if(nums[start] == target){

return start;

}

if(nums[end] == target){

return end;

}

return -1;

}

}

211 Add and Search Word - Data structure design

public class WordDictionary {

class TrieNode {

TrieNode[] child = new TrieNode[26];

boolean isWord = false;

}

TrieNode root = new TrieNode();

public void addWord(String word) {

TrieNode p = root;

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

if (p.child[c - 'a'] == null) p.child[c - 'a'] = new TrieNode();

p = p.child[c - 'a'];

}

p.isWord = true;

}

public boolean search(String word) {

return helper(word, 0, root);

}

private boolean helper(String s, int index, TrieNode p) {

if (index >= s.length()) return p.isWord;

char c = s.charAt(index);

if (c == '.') {

for (int i = 0; i < p.child.length; i++)

if (p.child[i] != null && helper(s, index + 1, p.child[i]))

return true;

return false;

} else return (p.child[c - 'a'] != null && helper(s, index + 1, p.child[c - 'a']));

}

}

278 first bad version

public class Solution extends VersionControl {

public int firstBadVersion(int n) {

if(n <= 0){

return 0;

}

int left = 1;

int right = n;

int mid = 0;

while(left + 1 < right){

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

if(isBadVersion(mid)){ // do not need to think about the previous one

right = mid;

} else{

left = mid;

}

}

if(isBadVersion(left)){

return left;

}

return right;

}

}

29 Divide two integer

public class Solution {

/\*\*

\* @param dividend the dividend

\* @param divisor the divisor

\* @return the result

\*/

public int divide(int dividend, int divisor) {

if (divisor == 0) {

return dividend >= 0? Integer.MAX\_VALUE : Integer.MIN\_VALUE;

}

if (dividend == 0) {

return 0;

}

if (dividend == Integer.MIN\_VALUE && divisor == -1) {

return Integer.MAX\_VALUE;

}

boolean isNegative = (dividend < 0 && divisor > 0) ||

(dividend > 0 && divisor < 0);

long a = Math.abs((long)dividend);

long b = Math.abs((long)divisor);

int result = 0;

while(a >= b){

int shift = 0;

while(a >= (b << shift)){

shift++;

}

a -= b << (shift - 1);

result += 1 << (shift - 1);

}

return isNegative? -result: result;

}

}

215 Kth Largest Element in an Array

class Solution {

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

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

return 0;

}

int start = 0;

int end = nums.length - 1;

while(true){

int position = partition(nums, start, end);

if(position == k - 1){

return nums[position];

} else if(position < k - 1){

start = position + 1;

} else {

end = position - 1;

}

}

}

private int partition(int[] nums, int start, int end){

int pivot = nums[start];

int i = start;

int j = end;

while(i < j){

while(i < j && nums[j] <= pivot){

j--;

}

nums[i] = nums[j];

while(i < j && nums[i] >= pivot){

i++;

}

nums[j] = nums[i];

}

nums[j] = pivot;

return j;

}

}

236 Lowest Common Ancestor of a Binary Tree

class Solution {

public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {

if(root == null || p == root || q == root){

return root;

}

TreeNode left = lowestCommonAncestor(root.left, p, q);

TreeNode right = lowestCommonAncestor(root.right, p, q);

if(left != null && right != null){

return root;

} else {

return left != null ? left :right;

}

}

235 Lowest Common Ancestor of a Binary Search Tree

class Solution {

public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {

if(root == null){

return null;

}

while(true){

if(root.val > Math.max(p.val, q.val)){

root = root.left;

} else if(root.val < Math.min(p.val, q.val)){

root = root.right;

} else {

break;

}

}

return root;

}

}

121 Best Time to Buy and Sell Stock

class Solution {

public int maxProfit(int[] prices) {

if(prices.length == 0){

return 0;

}

int min = prices[0];

int max = 0;

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

if(max < prices[i] - min){

max = prices[i] - min;

}

if(prices[i] < min){

min = prices[i];

}

}

return max;

}

}

76 Minimum Window Substring

class Solution {

public String minWindow(String s, String t) {

if(s == null || t == null){

return "";

}

if(s.length() < t.length()){

return "";

}

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

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

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

}

int left = 0;

int right = 0;

int count = map.size(); // map.size() to avoid duplicate in t

int len = 0;

int index = 0;

while(right < s.length()){

char c = s.charAt(right);

if(map.containsKey(c)){

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

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

count--;

}

}

right++; //remember to add right;

while(count == 0){

char cTemp = s.charAt(left);

if(map.containsKey(cTemp)){

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

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

count++;

}

}

if(len == 0 || len > right - left){ // remember the first one;

len = right - left;

index = left;

}

left++; //remember to add left;

}

}

return s.substring(index, index + len);

}

}

146 LRU cache

public class LRUCache {

private class Node{

Node prev;

Node next;

int key;

int value;

public Node(int key, int value) {

this.key = key;

this.value = value;

this.prev = null;

this.next = null;

}

}

private int capacity;

private HashMap<Integer, Node> hs = new HashMap<Integer, Node>();

private Node head = new Node(-1, -1);

private Node tail = new Node(-1, -1);

public LRUCache(int capacity) {

this.capacity = capacity;

tail.prev = head;

head.next = tail;

}

public int get(int key) {

if( !hs.containsKey(key)) {

return -1;

}

// remove current

Node current = hs.get(key);

current.prev.next = current.next;

current.next.prev = current.prev;

// move current to tail

move\_to\_tail(current);

return hs.get(key).value;

}

public void set(int key, int value) {

// this internal `get` method will update the key's position in the linked list.

if (get(key) != -1) {

hs.get(key).value = value;

return;

}

if (hs.size() == capacity) {

hs.remove(head.next.key);

head.next = head.next.next;

head.next.prev = head;

}

Node insert = new Node(key, value);

hs.put(key, insert);

move\_to\_tail(insert);

}

private void move\_to\_tail(Node current) {

current.prev = tail.prev;

tail.prev = current;

current.prev.next = current;

current.next = tail;

}

}

125 Valid Palindrome

class Solution {

public boolean isPalindrome(String s) {

if(s == null){

return false;

}

if(s.length() == 1)

{

return true;

}

s = s.toLowerCase();

int start = 0;

int end = s.length() - 1;

while(start < end){

char a = s.charAt(start);

char b = s.charAt(end);

int i = 0;

int j = 0;

while(start < end && !isValid(a)){

a = s.charAt(++start);

}

while(start < end && !isValid(b)){

b = s.charAt(--end);

}

if(a != b){

return false;

}

start++;

end--;

}

return true;

}

private boolean isValid(char c){

return Character.isLetter(c)||Character.isDigit(c);

}

}

311 Sparse Matrix Multiplication

public class Solution {

/\*\*

\* @param A a sparse matrix

\* @param B a sparse matrix

\* @return the result of A \* B

\*/

public int[][] multiply(int[][] A, int[][] B) {

// Write your code here

int n = A.length;

int m = A[0].length;

int k = B[0].length;

int[][] C = new int[n][k];

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

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

if (A[i][j] != 0)

for (int l = 0; l < k; ++l)

C[i][l] += A[i][j] \* B[j][l];

return C;

}

}

21 Merge two sorted list

class Solution {

public ListNode mergeTwoLists(ListNode l1, ListNode l2) {

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

return null;

}

if(l1 == null){

return l2;

}

if(l2 == null){

return l1;

}

ListNode dummy = new ListNode(0);

ListNode head;

head = dummy;

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

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

head.next = l1;

l1 = l1.next;

} else {

head.next = l2;

l2 = l2.next;

}

head = head.next;

}

while(l1 != null){

head.next = l1;

head = head.next;

l1 = l1.next;

}

while(l2 != null){

head.next = l2;

head = head.next;

l2 = l2.next;

}

return dummy.next;

}

}

meeting room I

/\*\*

\* Definition for an interval.

\* public class Interval {

\* int start;

\* int end;

\* Interval() { start = 0; end = 0; }

\* Interval(int s, int e) { start = s; end = e; }

\* }

\*/

class Solution {

public boolean canAttendMeetings(Interval[] intervals) {

if(intervals.length == 0){

return true;

}

if(intervals.length == 1){

return true;

}

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

public int compare(Interval i1, Interval i2){

return i1.start - i2.start;

}

});

int end = intervals[0].end;

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

if(intervals[i].start < end) {

return false;

}

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

}

return true;

}

}