## 哈希：

242 有效的字母异位词

解法一：//通过调用排序函数

class Solution {

public boolean isAnagram(String s, String t) {

char[] sstr=s.toCharArray();

char[] tstr=t.toCharArray();

Arrays.sort(sstr);

Arrays.sort(tstr);

return String.valueOf(sstr).equals(String.valueOf(tstr));

}

}

解法二：//构建一个长度为26的数组，用于存储字符串中字母出现的次数，使用数组下标//表示是哪一个字母。

class Solution {

public boolean isAnagram(String s, String t) {

if (s.length() != t.length()) {

return false;

}

int[] table = new int[26];

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

table[s.charAt(i) - 'a']++;

}

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

table[t.charAt(i) - 'a']--;

if (table[t.charAt(i) - 'a'] < 0) {

return false;

}

}

return true;

}

}

49、字母异位词  
class Solution{

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

If(strs.length==0) return new ArrayList();

Map<String,List> ans=new HashMap<String,List>();

for(String s:strs){

char[] ca=s.toCharArray();

Arrays.sort(ca);

String key=String.valueOf(ca);

If(!ans.containsKey(key)) ans.put(key,new ArrayList());

ans.get(key).ans(s);

}

Return new ArrayList(ans.values());

}

}

两数之和：

解法一：暴力

class Solution {

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

int[] res=new int[2];

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

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

if((nums[i]+nums[j])==target){

res[0]=i;

res[1]=j;

return res;

}

}

}

return res;

}

}

解法二：哈希表

class Solution {

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

int[] res=new int[2];

//使用哈希表的方法实现

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

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

if(hashtable.containsKey(target-nums[i])){

return new int[]{hashtable.get(target-nums[i]),i};

}

hashtable.put(nums[i],i);

}

return new int[0];

}

}

## 堆：

剑指 offer 40 最小的k个数  
class Solution {

public int[] getLeastNumbers(int[] arr, int k) {

Arrays.sort(arr);

int[] res=new int[k];

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

res[i]=arr[i];

}

return res;

}

}

239、滑动窗口最大值  
class Solution {

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

        if(nums == null || nums.length < 2) return nums;

        // 双向队列 保存当前窗口最大值的数组位置 保证队列中数组位置的数值按从大到小排序

        LinkedList<Integer> queue = new LinkedList();

        // 结果数组

        int[] result = new int[nums.length-k+1];

        // 遍历nums数组

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

            // 保证从大到小 如果前面数小则需要依次弹出，直至满足要求

            while(!queue.isEmpty() && nums[queue.peekLast()] <= nums[i]){

                queue.pollLast();

            }

            // 添加当前值对应的数组下标

            queue.addLast(i);

            // 判断当前队列中队首的值是否有效

            if(queue.peek() <= i-k){

                queue.poll();

            }

            // 当窗口长度为k时 保存当前窗口中最大值

            if(i+1 >= k){

                result[i+1-k] = nums[queue.peek()];

            }

        }

        return result;

    }

}

## 树

leetcode 108 将有序数组转换为二叉搜索树 https://leetcode-cn.com/problems/convert-sorted-array-to-binary-search-tree/

解法一：左右等分建立左右子树，然后对左右子树进行递归建立二叉树,递归的思想

代码：

class Solution {

public TreeNode sortedArrayToBST(int[] nums) {

return dfs(nums,0,nums.length-1);

}

private TreeNode dfs(int[] nums,int low,int high){

if(low>high){//终止条件

return null;

}

int mid=low+(high-low)/2;//二分查找找到中点

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

root.left=dfs(nums,low,mid-1）;

root.right=dfs(nums,mid+1,high);

return root;

}

反思：对于递归总是模模糊糊，老是会去思考递归的执行过程，但是我又想不明白递归的过程。

leetcode 226 翻转二叉树 https://leetcode-cn.com/problems/invert-binary-tree/

解法一：使用递归的思想，

leetcode 104 二叉树的最大深度 https://leetcode-cn.com/problems/maximum-depth-of-binary-tree/

解法一：递归

leetcode 534 二叉树的直径 https://leetcode-cn.com/problems/diameter-of-binary-tree/

解法：递归

计算每个节点的左右子树的路径长度，取较大值作为节点的路径。

int res=0;

public int diameterOfBinaryTree(TreeNode root) {

if(root==null){

return 0;

}

dfs(root);

return res;

}

private int dfs(TreeNode root){

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

return 0;

}

int leftlength= root.left==null ? 0:dfs(root.left)+1;

int rightlength= root.right==null? 0:dfs(root.right)+1;

res=Math.max(res,leftlength+rightlength);

return Math.max(leftlength,rightlength);

}

}

类似题目：124 二叉树中的最大路径和 https://leetcode-cn.com/problems/binary-tree-maximum-path-sum/ 、687 最长同值路径 https://leetcode-cn.com/problems/longest-univalue-path/

94、二叉树的中序遍历  
class Solution {

public List<Integer> inorderTraversal(TreeNode root) {

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

inorder(root,res);

return res;

}

public void inorder(TreeNode root,List<Integer> res){

if(root==null){

return;

}

inorder(root.left,res);

res.add(root.val);

inorder(root.right,res);

}

}

1. 二叉树的前序遍历

class Solution {

public List<Integer> inorderTraversal(TreeNode root) {

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

inorder(root,res);

return res;

}

public void preorder(TreeNode root,List<Integer> res){

if(root==null){

return;

}  
res.add(root.val);

preorder(root.left,res);

preorder(root.right,res);

}

}

590、N叉树的后序遍历  
class Solution {

public List<Integer> postorder(Node root) {

LinkedList<Node> stack=new LinkedList<>();

LinkedList<Integer> output=new LinkedList<>();

if(root==null){

return output;

}

stack.add(root);

while(!stack.isEmpty()){

Node node=stack.pollLast();

output.addFirst(node.val);

for(Node item:node.children){

if(item!=null){

stack.add(item);

}

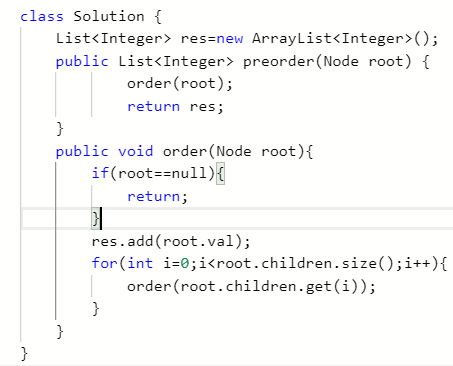
}

}

return output;

}

}

1. N叉树的前序遍历  
     
     
   429、N叉树的层序遍历  
   