【第十九周】月度刷题测试题

1673. 找出最具竞争力的子序列

https://leetcode-cn.com/problems/find-the-most-competitive-subsequence/

1、维护单调递增的栈，若当前的元素小于栈顶，则弹出栈顶至空或栈顶元素小于当前元素

2、n-i： 当前剩下的元素个数，k-stk.length：仍需要的元素个数。 若 n-1 < k-stk.length 即当前所剩元素个数小于需要的个数，则全部放进栈里，无须进入while 循环

3、若栈内元素个数多于k个，则弹出至只剩k个。 样例： 【1，2，3，4】， k = 2

```
/**
 * @param {number[]} nums
 * @param {number} k
 * @return {number[]}
 */
//  维护单调递增栈
var mostCompetitive = function(nums, k) {
    let stack = [],n = nums.length;
    for(let i = 0; i < n ;i++){
        while(stack.length > 0 && nums[i] < stack[stack.length-1] && n - i > k -
stack.length){
            stack.pop();
        }
        stack.push(nums[i]);
    }
    while(stack.length > k) stack.pop();
    return stack;
};
```

66. 加一

https://leetcode-cn.com/problems/plus-one/

1.数组是非空的,且每个数组元素都是非负的
2.每个数组元素的取值在 0~9之间
3.我们要返回一个数组让其是原数组的最后一位加上1

在这里将会出现3种情况

(1)如 123 + 1 = 124 , 数组长度不变,数组最后一位元素加一且不需要进一位
(2)如 129 + 1 = 130 , 数组长度不变,数组最后一位元素加一且需要进一位
(3)如 99 + 1 = 100 , 数组长度加一,数组最后一位元素加一且需要进一位

递归

4.我们一般的思路是先让数组最后一个元素 +1 若不等于10则在此基础上 +1 并return,反之则保留个位并进一,再判断倒数第二位元素 +1 是否等于10,以此类推

5.我们只需要判断 digits[i] + 1 是否等于0,若等于则传入倒数第二个下标进入递归,直到 digits[i] + 1 !== 10 || i == -1结束递归

```
 /**
 * @param {number[]} digits
 * @return {number[]}
 */
//  用算法模拟加1往前是否进位的操作逻辑
var plusOne = function(digits) {
    // 找到当前数组的最后一个元素的下标
    let index = digits.length - 1;
    // 进入递归，传入数组，和数组的倒数第一个数
    addOne(digits,index);
    return digits;
};
function addOne(arr,index){
    // 如果数组的第一位+1，也等于10，在数组的头部插入1；例子：99+1= 100
    if(index === -1) return arr.unshift(1);
    // 当前个位 +1 等于10 往前进一位，个位等于0；  例子：129+1= 130
    if(arr[index] + 1 === 10){
        arr[index] = 0;
        addOne(arr,index - 1)
    }else{ //加完1 不等于10，照常加1 返回；  例子：123+1=124
        arr[index] += 1;
        return arr;
    }
}
```

268. 丢失的数字

https://leetcode-cn.com/problems/missing-number/

1.计算数组总和sum
2.遍历数组，sum减去每个数组值,剩下的值就是结果。

```
 /**
 * @param {number[]} nums
 * @return {number}
 */
var missingNumber = function(nums) {
    let n = nums.length;
    let dis = (1+n) * n / 2;
    for(let i = 0; i < n; i++){
        dis = dis - nums[i];
    }
    return dis;
};
```

面试题 17.12. BiNode

https://leetcode-cn.com/problems/binode-lcci/

利用的二叉树的遍历

1、新建一个节点

2、中序遍历二叉搜索树

3、边遍历边操作一次，改变一次位置

4、操作一次有3个步骤：

（1）当前根节点的左节点赋为null；

（2）上一个节点的右节点指向当前节点；

（3）更新上一个节点，以便下次操作

5、最后返回新建节点的右节点

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val) {
 *     this.val = val;
 *     this.left = this.right = null;
 * }
 */
/**
 * @param {TreeNode} root
 * @return {TreeNode}
 */
//  1.将当前根节点的左节点 赋值为null；
//  2.上一个节点的右节点指向当前节点
//  3.更新上一个节点，方便我们下一步操作
var convertBiNode = function(root) {
    //新建一个节点，作为初始空节点的上一个节点
    let preNode = new TreeNode(0);
    const res = preNode;
    const inOrder = root =>{
        if(!root) return null;
        inOrder(root.left);
        // 将当前根节点的左节点 赋值为null；
        root.left = null;
        // 上一个节点的右节点指向当前节点
        preNode.right = root;
        // 更新上一个节点，方便我们下一步操作
        preNode = root;
        inOrder(root.right);
    }
    inOrder(root);
    return res.right;
};
```

328. 奇偶链表

https://leetcode-cn.com/problems/odd-even-linked-list/

这个题类似于 讲过的第一节课【链表】里面的【分隔链表】，做法参考分隔链表

```
/**
 * Definition for singly-linked list.
```

```
 * function ListNode(val, next) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.next = (next===undefined ? null : next)
 * }
 */
/**
 * @param {ListNode} head
 * @return {ListNode}
 */
var oddEvenList = function(head) {
    if(head === null) {
        return head;
    }
    let evenHead = head.next;
    let odd = head,even = evenHead;
    while(even !== null && even.next !== null){
        odd.next = even.next;
        odd = odd.next;
        even.next = odd.next;
        even = even.next;
    }
    odd.next = evenHead;
    return head;
};
```

第二种方法:

时间复杂度为O(1/2n)也就是O(n) 解法:
1.使用快慢指针;每次更新快慢指针跳过1个节点
2.用奇数指针的最后一项连接偶数指针
3.最终就得到原来的结果

```
var oddEvenList = function (head) {
  if (!head || !head.next) return head;
  let odd = head; // 当前项为奇数项
  let even = head.next; // 当前项为偶数项
  let targetOdd = new ListNode(); // 当前为奇数链表
  let oddNode = targetOdd; // 当前为奇数链表当前的节点
  let targetEven = new ListNode(); // 当前为偶数链表
  let evenNode = targetEven; // 当前为偶数链表
  // 这里的时间复杂度O(1/2n)
  while (odd || even) {
    // 分别增加奇数链表以及偶数链表的节点
    if (odd) {
      oddNode.next = new ListNode(odd.val);
      oddNode = oddNode.next;
    }
    if (even) {
      evenNode.next = new ListNode(even.val);
      evenNode = evenNode.next;
    }
    // 查找下一个奇数或者偶数节点,当下一个节点的时为null时
    // 把当前的节点置为null防止死循环
    if (odd) {
      if (odd.next) {
        odd = odd.next.next;
```

```
      } else {
        odd = null;
      }
    }
    if (even) {
      if (even.next) {
        even = even.next.next;
      } else {
        even = null;
      }
    }
  }
  // 剩下的为最后一个节点连接偶数节点最终返回的就是所需要的
  oddNode.next = targetEven.next;
  return targetOdd.next;
};
```

781. 森林中的兔子

https://leetcode-cn.com/problems/rabbits-in-forest/

解题思路:

1.如果一只兔子说还有n只兔子与它有相同颜色，那么必然存在n + 1只兔子。
2.answers中并不包含所有兔子，也就是说[2]、[2, 2]、[2, 2, 2]三种情况其实是一样的。
3.如果answers = [2, 2, 2]，只有当兔子数量为3只时，才能满足任意一只能告诉你还有2只与它有相同颜色。
4.如果answers = [2, 2, 2, 2]，此时回答可以拆分成两组，分别是[2, 2, 2]和[2]，共有6只兔子。
5.因此问题就转换为，将answers按照回答数量分类，并统计所有分类的兔子数量。
6.如果每类回答ans的数量有count个，那么兔子一共可以分为Math.ceil(count / (ans + 1))组，每组ans + 1只。
7.因此每类回答对应的兔子数量为Math.ceil(count / (ans + 1)) * (ans + 1)。

```
/**
 * @param {number[]} answers
 * @return {number}
 */
var numRabbits = function(answers) {
    // 用map缓存每一种回答的数量
    let map = new Map();
    // 缓存结果 最少的兔子数量
    let result = 0;
    // 遍历所有回答，统计每一种回答出现的次数
    for(const ans of answers){
        map.set(ans,map.has(ans) ? map.get(ans) + 1 : 1);
    }
    // 根据每一种回答的次数，计算兔子🐰的数量
    for(const [ans,count] of map){
        // 统一每一类回答对应的兔子的数量
        result +=
        // 计算每一类回答可以分为几组
        Math.ceil(count / (ans + 1)) *
        // 每一组兔子🐰的数量
        (ans + 1)
```

```
    }
    return result;

};
```

136. 只出现一次的数字

https://leetcode-cn.com/problems/single-number/

1.用sort方法对原数组里面各个元素的出现的次数，做一个从小到大的升序

2.接着去遍历数组里面每一个元素的值，如果当前元素值等于当前元素值的下一个，累加；否则返回就是我们想要的结果

```
/**
 * @param {number[]} nums
 * @return {number}
 */
var singleNumber = function(nums) {
    nums.sort((a,b) => a - b);
    for(let i = 0,len = nums.length;i < len; i++){
        if(nums[i] === nums[i+1]) i++;
        else return nums[i];
    }
};
```

543. 二叉树的直径

https://leetcode-cn.com/problems/diameter-of-binary-tree/

做法：DFS

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val, left, right) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.left = (left===undefined ? null : left)
 *     this.right = (right===undefined ? null : right)
 * }
 */
/**
 * @param {TreeNode} root
 * @return {number}
 */
//  dfs
var diameterOfBinaryTree = function(root) {
    // 默认为1 是因为默认根节点自身的路径长度
    let ans = 1;
    function depth(rootNode){
        if(!rootNode){
            // 因为不存在根节点，所以深度即是0
            return 0;
        }
```

```
        let L = depth(rootNode.left);
        let R = depth(rootNode.right);
        // 获取树的最长路径
        // L + R + 1 = 左子树深度（节点个数） + 右子树深度（节点个数） + 1个根节点；
        ans = Math.max(ans,L + R + 1);
        // 已经知道因为根节点的左右子树的深度，则左右子树深度的最大值+1，即是以根节点为主的最大深度
        return Math.max(L,R) + 1;
    }
    depth(root);
    // 由于depth函数中已经默认加上自身跟节点路径，所以最后减去1
    return ans - 1;
};
```

剑指 Offer 38. 字符串的排列

https://leetcode-cn.com/problems/zi-fu-chuan-de-pai-lie-lcof/

长度为n的字符串全排列的全部情况共n!种，循环遍历每个字符，并把该字符当成接下来结果中的字符串首个字符，然后递归找到剩余字符的所有排列组合情况再进行拼接

```
/**
 * @param {string} s
 * @return {string[]}
 */
var permutation = function (s) {
    let set = new Set()//set集合去重
    if (s.length == 1)
        return [s]
    for (let i = 0; i < s.length; i++) {
        let char = s[i]
        let ss = s.slice(0, i) + s.slice(i + 1, s.length)//拼接其余字符
        let l = permutation(ss)//递归找到其余元素的所有排列方式
        for (let j = 0; j < l.length; j++) {
            set.add(char + l[j])//将遍历的字符char拼接到剩余字符全排列的头部并存储到集合中
        }
    }
    return [...set]//集合转化为数组
};
```

169. 多数元素

https://leetcode-cn.com/problems/majority-element/

当 元素和栈顶元素相等 或 空栈 时入栈，则出栈，最后栈中剩下的必然都是是大于一半的那个元素

```
/**
 * @param {number[]} nums
```

```
 * @return {number}
 */
var majorityElement = function(nums) {
    let stack = [];
    for(let n of nums){
        // let m = stack.length;
        // if(stack[m-1] === n || !m){
        // 注释的两行等于下一行的if条件
        if(!stack.length || n === stack[stack.length - 1]){
            stack.push(n);
        }else{
            stack.pop();
        }
    }
    return stack[0];
};
```