# 剑指Offer

## 1. [用两个栈实现队列](https://leetcode-cn.com/problems/yong-liang-ge-zhan-shi-xian-dui-lie-lcof/)

用两个栈实现一个队列。队列的声明如下，请实现它的两个函数 appendTail 和 deleteHead ，分别完成在队列尾部插入整数和在队列头部删除整数的功能。(若队列中没有元素，deleteHead 操作返回 -1 )

### 思路：

两栈当作队列，栈A和栈B，栈A当作队尾，栈B当作队头；入栈的时候不断向栈尾(栈A)加入，当取数的时候从栈B取，栈B为空的时候栈A全部入栈，则指针指向的就是栈头；不为空的时候直接取出即可；

### 代码：

class CQueue {

    Stack<Integer> Instack;

    Stack<Integer> Outstack;

    int size=0;

    public CQueue() {

        Instack =new Stack<>();

        Outstack =new Stack<>();

    }

    public void appendTail(int value) {

        Instack.push(value);

    }

    public int deleteHead() {

        if(Outstack.isEmpty()){

            if(Instack.isEmpty()) return -1;

            while(!Instack.isEmpty()){

                Outstack.push(Instack.pop());

            }

            return Outstack.pop();

        }

        return Outstack.pop();

    }

}

### 小结：

注意一下Stack的方法，peek,pop,push，还有它的初始化。

## 2. [包含min函数的栈](https://leetcode-cn.com/problems/bao-han-minhan-shu-de-zhan-lcof/)

定义栈的数据结构，请在该类型中实现一个能够得到栈的最小元素的 min 函数在该栈中，调用 min、push 及 pop 的时间复杂度都是 O(1)。

### 思路：

主要问题在于min方法的O（1）时间复杂度，这种取最小类的，通常都是设置两个栈，两个栈的高度都是平行的，栈A是用来记录真实的数据，栈B是用来记录最小的数据，当入栈的数据大于最小值（栈B的栈顶数据）时，则栈B入栈的数据则是栈B的栈顶数据。

### 代码

class MinStack {

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

    Stack<Integer> Stack1;

    Stack<Integer> Stack2;

    public MinStack() {

        Stack1=new Stack<>();

        Stack2=new Stack<>();

    }

    public void push(int x) {

        Stack1.push(x);

        if(Stack2.isEmpty()){

            Stack2.push(x);

        }

        else{

         if(Stack2.peek()<x){

            Stack2.push(Stack2.peek());

        }

          else{

           Stack2.push(x);

        }

        }

    }

    public void pop() {

        Stack1.pop();

        Stack2.pop();

    }

    public int top() {

        return Stack1.peek();

    }

    public int min() {

        return Stack2.peek();

    }

}

/\*\*

 \* Your MinStack object will be instantiated and called as such:

 \* MinStack obj = new MinStack();

 \* obj.push(x);

 \* obj.pop();

 \* int param\_3 = obj.top();

 \* int param\_4 = obj.min();

 \*/

### 小结

本题的难点在于查询时间复杂度为O（1），比较精妙的是用两个平行的栈，一个记录真实的数据，另一个记录最小的数据。

## 3. [有效的括号](https://leetcode-cn.com/problems/valid-parentheses/)

给定一个只包括 '('，')'，'{'，'}'，'['，']' 的字符串 s ，判断字符串是否有效。

有效字符串需满足：

1. 左括号必须用相同类型的右括号闭合。
2. 左括号必须以正确的顺序闭合。

### 思路：

括号的匹配，首先判断长度是否为偶数；然后再用栈的形式，为左括号的时候直接入栈，如果是右括号，首先判断栈是不是空，再判断是否匹配，匹配的方式用ASCII码判断；

### 代码

class Solution {

    public boolean isValid(String s) {

        if(s.length()%2!=0) return false;

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

        char[] c=s.toCharArray();

        char ch;

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

            if(St.isEmpty()){

                if(c[i]==']'||c[i]=='}'||c[i]==')') return false;

            }

            if(c[i]=='['||c[i]=='{'||c[i]=='(') St.push(c[i]);

            else {

                ch=St.pop();

                if(Math.abs(c[i]-ch)<=2) continue;

                else

                return false;

            }

        }

        return St.isEmpty();

    }

}

### 小结

主要是对JAVA的方法总结，首先是String 类型，想要对各个字符处理需要转化为char[],方法为toCharArray();然后abs方法需要静态方法Math，M大写。

## 4.[从尾到头打印链表](https://leetcode-cn.com/problems/cong-wei-dao-tou-da-yin-lian-biao-lcof/)

### 思路

反转用stack作为中介

### 代码

/\*\*

 \* Definition for singly-linked list.

 \* public class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode(int x) { val = x; }

 \* }

 \*/

class Solution {

    public int[] reversePrint(ListNode head) {

        Stack<ListNode> s=new Stack<>();

        while(head!=null){

            s.push(head);

            head=head.next;

        }

        int size=s.size(),nsize=0;

        int[] num=new int[size];

        while(!s.isEmpty()){

            num[nsize++]=s.pop().val;

        }

        return num;

    }

}

### 小结

主要是JAVA里方法的应用，一个是ListNode,next方法没有（）；另一个是取值的时候加上.val

## 5. [反转链表](https://leetcode-cn.com/problems/fan-zhuan-lian-biao-lcof/)

### 思路

最基本的是加入一个Stack,然后将Stack的内容插入链表的val即可

### 代码

/\*\*

 \* Definition for singly-linked list.

 \* public class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode(int x) { val = x; }

 \* }

 \*/

class Solution {

    public ListNode reverseList(ListNode head) {

        ListNode temp=head;

        Stack<Integer> s=new Stack<>();

        while(temp!=null){

            s.push(temp.val);

            temp=temp.next;

        }

        temp=head;

        while(!s.isEmpty()){

            temp.val=s.pop();

            temp=temp.next;

        }

        return head;

    }

}

### 小结

指针的应用，另一个是Stack<Linklist>容易出问题；看了别的，应该还有更加简单的答案

## 6. [回文链表](https://leetcode-cn.com/problems/palindrome-linked-list/)

给你一个单链表的头节点 head ，请你判断该链表是否为回文链表。如果是，返回 true ；否则，返回 false 。

### 思路

用Stack进行比较，回文问题一般比较一半即可

### 代码

/\*\*

 \* Definition for singly-linked list.

 \* public class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode() {}

 \*     ListNode(int val) { this.val = val; }

 \*     ListNode(int val, ListNode next) { this.val = val; this.next = next; }

 \* }

 \*/

class Solution {

    public boolean isPalindrome(ListNode head) {

        Stack<Integer> s=new Stack<>();

        int num=0;

        ListNode p=head;

        while(p!=null) {

           s.push(p.val);

            p=p.next;

            num++;

        }

        for(int i=0;i<num/2;i++){

            if(s.pop()!=head.val) return false;

            head=head.next;

        }

        return true;

    }

}

### 小结

简单题，下次二刷的时候看看别人的答案，有没有什么好方法

## 7. [二叉树的中序遍历](https://leetcode-cn.com/problems/binary-tree-inorder-traversal/)

给定一个二叉树的根节点 root ，返回它的 **中序** 遍历。

### 思路

经典的递归题，构造一个方法，方法用来递归的，首先判断指针是否为空，然后再根据题目要求遍历，图的递归遍历也是如此

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode() {}

 \*     TreeNode(int val) { this.val = val; }

 \*     TreeNode(int val, TreeNode left, TreeNode right) {

 \*         this.val = val;

 \*         this.left = left;

 \*         this.right = right;

 \*     }

 \* }

 \*/

class Solution {

    List<Integer> ans;

    public void mid(TreeNode root,List<Integer> ans){

        if(root==null) return ;

            mid(root.left,ans);

            ans.add(root.val);

            mid(root.right,ans);

    }

    public List<Integer> inorderTraversal(TreeNode root) {

        ans=new ArrayList();

        mid(root,ans);

        return ans;

    }

}

### 小结

无小结，下次用栈的方法做。

## 8. [对称二叉树](https://leetcode-cn.com/problems/symmetric-tree/)

给定一个二叉树，检查它是否是镜像对称的

### 思路

对称主要是，如果为空，那么就是对称，如果左右有一个不存在一个存在，那么就不是对称，再接下来去判断他们的val值是否相等，同时对A的左子树和B的右子树以及A的右子树和B的左子树做相同的判断

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode() {}

 \*     TreeNode(int val) { this.val = val; }

 \*     TreeNode(int val, TreeNode left, TreeNode right) {

 \*         this.val = val;

 \*         this.left = left;

 \*         this.right = right;

 \*     }

 \* }

 \*/

class Solution {

    boolean same(TreeNode leftnode,TreeNode rightnode){

        if(leftnode==null&&rightnode==null) return true;

        else if(leftnode==null||rightnode==null) return false;

        return leftnode.val==rightnode.val&&same(leftnode.left,rightnode.right)&&same(leftnode.right,rightnode.left);

    }

    public boolean isSymmetric(TreeNode root) {

        return same(root,root);

    }

}

### 小结

对于树递归的题，基本都得判断指针是否为空，然后根据题意继续判断。

## 9. [替换空格](https://leetcode-cn.com/problems/ti-huan-kong-ge-lcof/)

请实现一个函数，把字符串 s 中的每个空格替换成"%20"。

### 思路

用一个char[]数组一个个分析过来，在char[]长度的设置上，最大的情况是都为空格，那么长度相对于String为三倍

### 代码

class Solution {

    public String replaceSpace(String s) {

        int length=s.length(),count=0;

        char[] temp=new char[length\*3];

        for(int i=0;i<length;i++){

            char tempc=s.charAt(i);

            if(tempc==' '){

                temp[count++]='%';

                temp[count++]='2';

                temp[count++]='0';

            }

            else temp[count++]=tempc;

        }

        String ans= new String(temp,0,count);

        return ans;

    }

}

### 小结

主要是对JAVA方法的分析，charAt()，length(),最后是String ans=new String(char[],int,int)，这个是将char数组转化为String，第一个参数带转化的char数组，第二个参数是从char[]哪里开始,第三个参数是长度。

## 10. [左旋转字符串](https://leetcode-cn.com/problems/zuo-xuan-zhuan-zi-fu-chuan-lcof/)

字符串的左旋转操作是把字符串前面的若干个字符转移到字符串的尾部。请定义一个函数实现字符串左旋转操作的功能。比如，输入字符串"abcdefg"和数字2，该函数将返回左旋转两位得到的结果"cdefgab"。

### 思路

用C++就是先将整个String反转，然后在中间响应的位置reverse即可，但是Java不行，还是得加入两个Queue，新建数组然后加入，但是这样复杂度比较高，内存消耗也大。

### 代码

class Solution {

    public String reverseLeftWords(String s, int n) {

        Queue<Character> q1=new LinkedList<Character>(),q2=new LinkedList<Character>();

        int size=s.length(),count=0;

        char[] temp=new char[size];

        for(int i=0;i<size;i++){

            char tempc=s.charAt(i);

            if(i<n) q1.offer(tempc);

            else q2.offer(tempc);

        }

        while(q2.peek()!=null){

            temp[count++]=q2.poll();

        }

        while(q1.peek()!=null){

            temp[count++]=q1.poll();

        }

        String ans=new String(temp,0,size);

        return ans;

    }

}

### 小结

一个是像Stack和Queue中的<>必须是封装的，不能是基础类型；然后Queue的初始化是LinkedList,再然后是Queue的加入用offer，add容易出错

## 11. [只出现一次的数字](https://leetcode-cn.com/problems/single-number/)

给定一个**非空**整数数组，除了某个元素只出现一次以外，其余每个元素均出现两次。找出那个只出现了一次的元素。

### 思路

主要通过HashMap和Set完成

### 代码

class Solution {

    public int singleNumber(int[] nums) {

        HashMap<Integer,Integer> m=new HashMap<>();

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

        int size=nums.length;

        for(int i=0;i<size;i++){

            if(m.get(nums[i])==null)m.put(nums[i],1);

            else m.put(nums[i],2);

            s.add(nums[i]);

        }

        for(int temps:s){

            if(m.get(temps)==1) return temps;

        }

        return 0;

    }

}

### 小结

主要是对JAVA里HashMap和Set的方法的总结，当初始化一个变量的时候，如果前面的和后面New的变量名不一样，则<>中要说明；HashMap中有get，用于判断映射是否存在；put方法用于加入

## 12. [比特位计数](https://leetcode-cn.com/problems/counting-bits/)

给你一个整数 n ，对于 0 <= i <= n 中的每个 i ，计算其二进制表示中 **1 的个数** ，返回一个长度为 n + 1 的数组 ans 作为答案。

### 思路

简单题

### 代码

class Solution {

    public int[] countBits(int n) {

        int[] ans=new int[n+1];

        for(int i=0;i<=n;i++){

            int temp=i;

            while(temp!=0){

                if(temp%2==1) ans[i]++;

                temp/=2;

            }

        }

        return ans;

    }

}

### 小结

无

## 13. [二叉树的最大深度](https://leetcode-cn.com/problems/maximum-depth-of-binary-tree/)

给定一个二叉树，找出其最大深度。

二叉树的深度为根节点到最远叶子节点的最长路径上的节点数。

**说明:** 叶子节点是指没有子节点的节点。

### 思路

经典的二叉树最大深度问题，用递归的方法

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode() {}

 \*     TreeNode(int val) { this.val = val; }

 \*     TreeNode(int val, TreeNode left, TreeNode right) {

 \*         this.val = val;

 \*         this.left = left;

 \*         this.right = right;

 \*     }

 \* }

 \*/

class Solution {

    public int maxDepth(TreeNode root) {

        if(root==null){

            return 0;

        }

        int lefthigh=maxDepth(root.left);

        int righthigh=maxDepth(root.right);

        return Math.max(lefthigh,righthigh)+1;

    }

}

### 小结

递归，当访问到空节点时候返回高度0，然后左右对比，取左右最大值且+1；

## 14. [翻转二叉树](https://leetcode-cn.com/problems/invert-binary-tree/)

翻转一棵二叉树。

### 思路

用递归的思想，对于左右子树一颗颗下去，然后到底的时候，如果是null的话，那么返回null;然后对它的左右子树进行反转，从底层反转到高层，即可完成反转

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode() {}

 \*     TreeNode(int val) { this.val = val; }

 \*     TreeNode(int val, TreeNode left, TreeNode right) {

 \*         this.val = val;

 \*         this.left = left;

 \*         this.right = right;

 \*     }

 \* }

 \*/

class Solution {

    public TreeNode invertTree(TreeNode root) {

        if(root==null) return null;

        TreeNode leftnode=invertTree(root.left);

        TreeNode rightnode=invertTree(root.right);

        root.left=rightnode;

        root.right=leftnode;

        return root;

    }

}

### 小结

树的题都可以用递归

## 15[二叉树的直径](https://leetcode-cn.com/problems/diameter-of-binary-tree/)

给定一棵二叉树，你需要计算它的直径长度。一棵二叉树的直径长度是任意两个结点路径长度中的最大值。这条路径可能穿过也可能不穿过根结点。

示例 :

给定二叉树

1

/ \

2 3

/ \

4 5

返回 3, 它的长度是路径 [4,2,1,3] 或者 [5,2,1,3]。

### 思路

主要是遍历来查高度，值得注意的是遍历的时候设置一个路径最长的全变量，每次新的路径出来跟全变量对比一下，谁大选谁

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode() {}

 \*     TreeNode(int val) { this.val = val; }

 \*     TreeNode(int val, TreeNode left, TreeNode right) {

 \*         this.val = val;

 \*         this.left = left;

 \*         this.right = right;

 \*     }

 \* }

 \*/

class Solution {

    int ans=0;

    int finddeep(TreeNode root){

        if(root==null) return 0;

        int left=finddeep(root.left),right=finddeep(root.right);

        ans=Math.max(left+right,ans);

        return Math.max(left,right)+1;

    }

    public int diameterOfBinaryTree(TreeNode root) {

        finddeep(root);

        return ans;

    }

}

### 小结

值得思考

## 16. [合并二叉树](https://leetcode-cn.com/problems/merge-two-binary-trees/)

给定两个二叉树，想象当你将它们中的一个覆盖到另一个上时，两个二叉树的一些节点便会重叠。

你需要将他们合并为一个新的二叉树。合并的规则是如果两个节点重叠，那么将他们的值相加作为节点合并后的新值，否则不为 NULL 的节点将直接作为新二叉树的节点。

### 思路

用递归调用的方式去遍历，如果一个为空的话，直接把另一个的子树传递过去就好了，但是这里有一个细节，就是必须对treeA.left 的形式连接，否则只是浅拷贝，无法连接，这个就是难点

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode() {}

 \*     TreeNode(int val) { this.val = val; }

 \*     TreeNode(int val, TreeNode left, TreeNode right) {

 \*         this.val = val;

 \*         this.left = left;

 \*         this.right = right;

 \*     }

 \* }

 \*/

class Solution {

    public TreeNode mergeTrees(TreeNode root1, TreeNode root2) {

        if(root1==null) return root2;

        else if(root2==null)return root1;

        root1.val+=root2.val;

        root1.left=mergeTrees(root1.left,root2.left);

        root1.right=mergeTrees(root1.right,root2.right);

        return root1;

    }

}

### 小结

连接的地方要注意

## 17. [数组中重复的数字](https://leetcode-cn.com/problems/shu-zu-zhong-zhong-fu-de-shu-zi-lcof/)

### 思路

HashMap解决

### 代码

class Solution {

    public int findRepeatNumber(int[] nums) {

        HashMap<Integer,Integer> m=new HashMap<Integer,Integer>();

        int ans=0;

        for(int i:nums){

            if(m.get(i)==null) m.put(i,1);

            else {

                ans=i;break;

            }

        }

        return ans;

    }

}

### 小结

空间复杂度好像不行，下次二刷的时候尝试较优的解法

## 18. [在排序数组中查找数字 I](https://leetcode-cn.com/problems/zai-pai-xu-shu-zu-zhong-cha-zhao-shu-zi-lcof/)

统计一个数字在排序数组中出现的次数。

### 思路

用map，但是应该有更好的办法，应该是二分，下次再做。

### 代码

class Solution {

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

        HashMap<Integer,Integer> m=new HashMap<>();

        for(int i:nums){

            if(m.get(i)==null) m.put(i,1);

            else m.put(i,m.get(i)+1);

        }

        if(m.get(target)==null) return 0;

        else return m.get(target);

    }

}

### 小结

二刷的时候用更好的办法

## 19. [0～n-1中缺失的数字](https://leetcode-cn.com/problems/que-shi-de-shu-zi-lcof/)

一个长度为n-1的递增排序数组中的所有数字都是唯一的，并且每个数字都在范围0～n-1之内。在范围0～n-1内的n个数字中有且只有一个数字不在该数组中，请找出这个数字。

### 思路

时间复杂度比较小的话应该是二分法，但是我用了顺序查找

### 代码

class Solution {

    public int missingNumber(int[] nums) {

        int ans=0;

        for(int i=0;i<nums.length&&nums[i]==ans;ans++,i++);

        return ans;

    }

}

### 小结

复习了二分法再回头做做

## 20. [合并两个有序链表](https://leetcode-cn.com/problems/merge-two-sorted-lists/)

将两个升序链表合并为一个新的 **升序** 链表并返回。新链表是通过拼接给定的两个链表的所有节点组成的。

### 思路

双指针从前往后排序，如果遇到空的直接将节点加入即可。

### 代码

/\*\*

 \* Definition for singly-linked list.

 \* public class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode() {}

 \*     ListNode(int val) { this.val = val; }

 \*     ListNode(int val, ListNode next) { this.val = val; this.next = next; }

 \* }

 \*/

class Solution {

    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {

        ListNode l=new ListNode(),p=l;

        while(l1!=null||l2!=null){

            if(l1==null){

                p.next=l2;

                break;

            }

            else if(l2==null) {

                p.next=l1;

                break;

            }

            ListNode tl=new ListNode();

            if(l1.val<=l2.val){

                tl.val=l1.val;

                p.next=tl;

                p=p.next;

                l1=l1.next;

            }

            else{

                tl.val=l2.val;

                p.next=tl;

                p=p.next;

                l2=l2.next;

            }

        }

        return l.next;

    }

}

### 小结

主要是卡在那个（l1!=null||l2!=null）写成了（l1!=null&&l2!=null）了

## 21. [反转链表](https://leetcode-cn.com/problems/reverse-linked-list/)

给你单链表的头节点 head ，请你反转链表，并返回反转后的链表。

### 思路

加一个栈，然后倒叙输出。

### 代码

/\*\*

 \* Definition for singly-linked list.

 \* public class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode() {}

 \*     ListNode(int val) { this.val = val; }

 \*     ListNode(int val, ListNode next) { this.val = val; this.next = next; }

 \* }

 \*/

class Solution {

    public ListNode reverseList(ListNode head) {

        Stack<ListNode> s=new Stack<>();

        while(head!=null){

            s.push(head);

            head=head.next;

        }

        ListNode ans=new ListNode(),p=ans;

        while(!s.isEmpty()){

            ListNode t=s.pop();

            t.next=null;

            p.next=t;

            p=p.next;

        }

        return ans.next;

    }

}

### 小结

最笨的方法，应该还有简单的

## 22. [找到所有数组中消失的数字](https://leetcode-cn.com/problems/find-all-numbers-disappeared-in-an-array/)

给你一个含 n 个整数的数组 nums ，其中 nums[i] 在区间 [1, n] 内。请你找出所有在 [1, n] 范围内但没有出现在 nums 中的数字，并以数组的形式返回结果。

### 思路

新建一个数组，长度n+1,然后为0的位置的下标就是要寻找的

### 代码

class Solution {

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

        int[] temp=new int[nums.length+1];

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

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

            temp[nums[i]]++;

        }

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

            if(temp[i]==0) ans.add(i);

        }

        return ans;

    }

}

### 小结

以空间换时间

## 23. [283. 移动零](https://leetcode-cn.com/problems/move-zeroes/)

给定一个数组 nums，编写一个函数将所有 0 移动到数组的末尾，同时保持非零元素的相对顺序。

### 思路

有一个特别巧妙的思路，就是遍历整个数组，遇到非零的，从第一，第二，第三直接往下排，同时记录那个下标；遍历结束后从那个下标开始，一直补零补到最后。

### 代码

class Solution {

    public void moveZeroes(int[] nums) {

        int nowp=0,tempp=0;

        while(tempp<nums.length){

            if(nums[tempp]!=0){

                nums[nowp++]=nums[tempp];

            }

            tempp++;

        }

        while(nowp<nums.length){

            nums[nowp++]=0;

        }

    }

}

### 小结

很巧妙的思路，值得学习

## [24. 多数元素](https://leetcode-cn.com/problems/majority-element/)

给定一个大小为 n 的数组，找到其中的多数元素。多数元素是指在数组中出现次数 大于 ⌊ n/2 ⌋ 的元素。

你可以假设数组是非空的，并且给定的数组总是存在多数元素。

### 思路

用一个map来记录数据（因为不知道数组中的最大值），然后特殊情况单独拿出来

### 代码

class Solution {

    public int majorityElement(int[] nums) {

        if(nums.length==1) return nums[0];

        HashMap<Integer,Integer> m=new HashMap<>();

        int ans=0;

        for(int i:nums){

            if(m.get(i)==null) m.put(i,1);

            else if((m.get(i)+1)>(int)nums.length/2){

                ans=i;break;

            }

            else m.put(i,m.get(i)+1);

        }

        return ans;

    }

}

### 总结

应该还有更好的办法，下次刷的时候再写

## [25 环形链表](https://leetcode-cn.com/problems/linked-list-cycle/)

### 思路

简答题，用一个map映射一下之前有没有出现过这个Node即可

### 代码

/\*\*

 \* Definition for singly-linked list.

 \* class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode(int x) {

 \*         val = x;

 \*         next = null;

 \*     }

 \* }

 \*/

public class Solution {

    public boolean hasCycle(ListNode head) {

        HashMap<ListNode,Boolean> m=new HashMap<>();

        while(head!=null) {

            if(m.get(head)==null) m.put(head,true);

            head=head.next;

            if(m.get(head)!=null) return true;

        }

        return false;

    }

}

### 小结

简单题

## 26. [相交链表](https://leetcode-cn.com/problems/intersection-of-two-linked-lists/)

给你两个单链表的头节点 headA 和 headB ，请你找出并返回两个单链表相交的起始节点。如果两个链表没有交点，返回 null 。

### 思路

用HashMap记录出现的Node，然后用两个链表依次访问HashMap有无这个节点，有的话就返回这个节点，没有的话就插入

### 代码

/\*\*

 \* Definition for singly-linked list.

 \* public class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode(int x) {

 \*         val = x;

 \*         next = null;

 \*     }

 \* }

 \*/

public class Solution {

    public ListNode getIntersectionNode(ListNode headA, ListNode headB) {

        ListNode ans=null;

        HashMap<ListNode,Boolean> m=new HashMap<>();

        while(headA!=null||headB!=null){

            if(headA!=null&&m.get(headA)!=null){

                ans=headA;break;

            }

            else if(headB!=null&&m.get(headB)!=null) {

                ans=headB;break;

            }

           if(headA!=null&&m.get(headA)==null) {

               m.put(headA,true);

                headA=headA.next;

            }

            if(headB!=null&&m.get(headB)==null) {

               m.put(headB,true);

                headB=headB.next;

            }

        }

        return ans;

    }

}

### 小结

两条链表的题，while循环中用（||）

## 27[. 汉明距离](https://leetcode-cn.com/problems/hamming-distance/)

两个整数之间的 [汉明距离](https://baike.baidu.com/item/%E6%B1%89%E6%98%8E%E8%B7%9D%E7%A6%BB) 指的是这两个数字对应二进制位不同的位置的数目。

给你两个整数 x 和 y，计算并返回它们之间的汉明距离。

### 思路

简单题

### 代码

class Solution {

    public int hammingDistance(int x, int y) {

        Stack<Integer> S1=new Stack<>(),S2=new Stack<>();

        while(x!=0||y!=0){

            if(x!=0) {

                S1.push(x%2);

                x/=2;

            }

            else S1.push(0);

            if(y!=0){

                S2.push(y%2);

                y/=2;

            }

            else S2.push(0);

        }

        int count=0;

        while(!S1.isEmpty()){

            int t1=S1.pop(),t2=S2.pop();

            if(t1!=t2) count++;

        }

        return count;

    }

}

### 小结

## 28. [第一个只出现一次的字符](https://leetcode-cn.com/problems/di-yi-ge-zhi-chu-xian-yi-ci-de-zi-fu-lcof/)

在字符串 s 中找出第一个只出现一次的字符。如果没有，返回一个单空格。 s 只包含小写字母。

### 思路

用Map记录出现过的次数，用Queue记录出现过的字母。

### 代码

class Solution {

    public char firstUniqChar(String s) {

        char ans=' ';

        Queue<Character> Q=new LinkedList<Character>();

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

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

            char temp=s.charAt(i);

            if(m.get(temp)==null) {

                m.put(temp,1);

            }

            else {

                m.put(temp,m.get(temp)+1);

            }

            Q.offer(temp);

        }

        for(Character i:Q){

            if(m.get(i)==1){

                ans=i;break;

            }

        }

        return ans;

    }

}

### 小结

常规题

## 29. [两数相加](https://leetcode-cn.com/problems/add-two-numbers/)

给你两个 非空 的链表，表示两个非负的整数。它们每位数字都是按照 逆序 的方式存储的，并且每个节点只能存储 一位 数字。

请你将两个数相加，并以相同形式返回一个表示和的链表。

你可以假设除了数字 0 之外，这两个数都不会以 0 开头。

### 思路

看起来简单，其实里面藏着不少东西，首先他不能转化成Int Long来做，因为他们俩大小的限制肯定无法满足题意的；然后是这种链表相加，字符串相加，最好的办法就是补零，补成一样的长度；最后一点是对于LinkList,它不能通过本身等于 new LinkList来连接，只能通过.next来连接

### 代码

/\*\*

 \* Definition for singly-linked list.

 \* public class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode() {}

 \*     ListNode(int val) { this.val = val; }

 \*     ListNode(int val, ListNode next) { this.val = val; this.next = next; }

 \* }

 \*/

class Solution {

    public ListNode addTwoNumbers(ListNode l1, ListNode l2) {

        boolean flag=false;ListNode p=l1,q=l2;

        int n1=0,n2=0;

        while(p.next!=null){

            n1++;

            p=p.next;

        }

        while(q.next!=null){

            n2++;

            q=q.next;

        }

        if(n1>n2){

            int count=n1-n2;

            while(count>0){

                q.next=new ListNode(0);

                q=q.next;

                count--;

            }

        }

        if(n2>n1){

            int count=n2-n1;

            while(count>0){

                p.next=new ListNode(0);

                p=p.next;

                count--;

            }

        }

        ListNode ans=l1;

        while(l1!=null){

            int temp=0;

                temp=l1.val+l2.val;

                if(flag) temp++;

                l1.val=temp%10;

                if(temp>9)  flag=true;

                else flag=false;

                if(l1.next==null) break;

                l1=l1.next;

                l2=l2.next;

        }

        if(flag) l1.next=new ListNode(1);

            return ans;

    }

}

### 小结

这题跟以前做的字符串相加有点类似但有不同的地方，其他的解法二刷的时候再去学。

## 30. [从上到下打印二叉树](https://leetcode-cn.com/problems/cong-shang-dao-xia-da-yin-er-cha-shu-lcof/)

从上到下打印出二叉树的每个节点，同一层的节点按照从左到右的顺序打印

### 思路

就是层次遍历

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode(int x) { val = x; }

 \* }

 \*/

class Solution {

    public int[] levelOrder(TreeNode root) {

        if(root==null) return new int[0];

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

        Q.offer(root);

        Stack<Integer> s=new Stack<>();

        int count=1;

        while(!Q.isEmpty()){

            TreeNode temp=Q.poll();

            s.push(temp.val);

            if(temp.left!=null){

                Q.offer(temp.left);

                count++;

            }

            if(temp.right!=null){

                Q.offer(temp.right);

                count++;

            }

        }

        int[] ans=new int[count];

        count=0;

        for(Integer i:s){

            ans[count++]=i;

        }

        return ans;

    }

}

### 小结

无

## 31. [从上到下打印二叉树 II](https://leetcode-cn.com/problems/cong-shang-dao-xia-da-yin-er-cha-shu-ii-lcof/)

从上到下按层打印二叉树，同一层的节点按从左到右的顺序打印，每一层打印到一行。

例如:

给定二叉树: [3,9,20,null,null,15,7],

3

/ \

9 20

/ \

15 7

返回其层次遍历结果：

[

[3],

[9,20],

[15,7]

]

### 思路

这题比较有意思，返回的是二重数组，因此在细节上发生了变动，但它仍然是层次遍历，因此还是需要queue,然后比较巧妙地是Queue遍历的时候，需要一个for循环，for循环的次数就是这层的节点数（Queuec此时的长度），然后添加到一个List(一重数组)，再将List加到二重数组内。

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode(int x) { val = x; }

 \* }

 \*/

class Solution {

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

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

        if(root==null) return ans;

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

        Q.offer(root);

        while(!Q.isEmpty()){

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

            for(int i=Q.size();i>0;i--){

                TreeNode tempT=Q.poll();

                temp.add(tempT.val);

                if(tempT.left!=null){

                    Q.offer(tempT.left);

                }

                if(tempT.right!=null){

                    Q.offer(tempT.right);

                }

            }

            ans.add(temp);

        }

        return ans;

    }

}

### 小结

非常有意思的变种，值得关注

## 32. [从上到下打印二叉树 III](https://leetcode-cn.com/problems/cong-shang-dao-xia-da-yin-er-cha-shu-iii-lcof/)

请实现一个函数按照之字形顺序打印二叉树，即第一行按照从左到右的顺序打印，第二层按照从右到左的顺序打印，第三行再按照从左到右的顺序打印，其他行以此类推。

### 思路

打印二叉树的变种，思路大差不差，就是加入一个Stack,如果是需要从右到左，则加入到Stack,反之直接加入LinkedList即可。

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode(int x) { val = x; }

 \* }

 \*/

class Solution {

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

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

        List<List<Integer>> ans=new LinkedList<>();

        if(root==null) return ans;

        Q.offer(root);

        boolean flag=false;

        while(!Q.isEmpty()){

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

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

            for(int i=Q.size();i>0;i-- ){

                TreeNode tempt=Q.poll();

             if(tempt.left!=null){

                        Q.offer(tempt.left);

                    }

                    if(tempt.right!=null){

                        Q.offer(tempt.right);

                    }

                if(flag){

                    temps.push(tempt);

                }

                else{

                    templ.add(tempt.val);

                }

            }

            while(!temps.isEmpty()){

                templ.add(temps.pop().val);

            }

            if(flag) flag=false;

            else flag=true;

            ans.add(templ);

        }

        return ans;

    }

}

### 小结

变种题

## 33. [删除链表的倒数第 N 个结点](https://leetcode-cn.com/problems/remove-nth-node-from-end-of-list/)

给你一个链表，删除链表的倒数第 n 个结点，并且返回链表的头结点。

### 思路

最笨的方法，扫两次，然后特殊情况特殊处理。

### 代码

/\*\*

 \* Definition for singly-linked list.

 \* public class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode() {}

 \*     ListNode(int val) { this.val = val; }

 \*     ListNode(int val, ListNode next) { this.val = val; this.next = next; }

 \* }

 \*/

class Solution {

    public ListNode removeNthFromEnd(ListNode head, int n) {

        ListNode p=head,q;

        Stack<ListNode> s=new Stack<>();

        int count=0,one=0;

        while(p!=null){

            s.push(p);

            count++;

            p=p.next;

        }

        int the=count-n;

        if(count==1) return null;

        if(the==0) return head.next;

        p=head;

        count=0;

        while(p!=null){

            count++;

            if(count==the){

                p.next=p.next.next;

                break;

            }

            p=p.next;

        }

        return head;

    }

}

### 小结

方法太垃圾了，二刷的时候找点好的

## 34.子集

给你一个整数数组 nums ，数组中的元素 **互不相同** 。返回该数组所有可能的子集（幂集）。

解集 **不能** 包含重复的子集。你可以按 **任意顺序** 返回解集。

### 思路

经典回溯题，写一个全局变量，然后主函数调用方法，方法的格式是for(int i=j;i<xx.length;i++),新增元素，加入全局变量,递归调用方法；剔除刚刚加入的元素

### 代码

class Solution {

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

    void visit(int index,int[] nums,LinkedList<Integer> temp){

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

            temp.add(nums[i]);

            ans.add(new ArrayList(temp));

            visit(++index,nums,temp);

            temp.removeLast();

        }

    }

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

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

         ans.add(temp);

        visit(0,nums,temp);

        return ans;

    }

}

### 小结

比较简单的回溯题

## 35. [树的子结构](https://leetcode-cn.com/problems/shu-de-zi-jie-gou-lcof/)

输入两棵二叉树A和B，判断B是不是A的子结构。(约定空树不是任意一个树的子结构)

B是A的子结构， 即 A中有出现和B相同的结构和节点值。

### 思路

回溯题，这题求子结构，分为两个，一个是主函数，就是树的起点出现null的话，无论是A还是B直接返回false;然后是对这个A树的左子树和B树的主函数（而非isright函数）进行调用；只要有一个满足那么子树就能找到了；

然后是调用方法，方法的话是对该节点的子节点进行查询，如果B树为null了，则说明查询结束且前面的查询都是对上了，因此回true,然后是对节点的数值对比以及节点的左节点和右节点进行对比。

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode(int x) { val = x; }

 \* }

 \*/

class Solution {

    public boolean isSubStructure(TreeNode A, TreeNode B) {

        if(A==null||B==null) return false;

        return isright(A,B)||isSubStructure(A.left,B)||isSubStructure(A.right,B);

    }

     boolean isright(TreeNode A, TreeNode B){

        if(B==null) return true;

        if(A==null||A.val!=B.val) return false;

        return isright(A.left,B.left)&&isright(A.right,B.right);

    }

}

### 小结

很好的递归题，值得反复思考

## 36. [二叉树的镜像](https://leetcode-cn.com/problems/er-cha-shu-de-jing-xiang-lcof/)

请完成一个函数，输入一个二叉树，该函数输出它的镜像。

### 思路

一直递归后根遍历，如果访问到null则返回，然后将它的左右节点对换即可。

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode(int x) { val = x; }

 \* }

 \*/

class Solution {

    void bianli(TreeNode root){

        if(root==null) return ;

        bianli(root.left);

        bianli(root.right);

        TreeNode temp=root.left;

        root.left=root.right;root.right=temp;

    }

    public TreeNode mirrorTree(TreeNode root) {

        bianli(root);

        return root;

    }

}

### 小结

后跟遍历的变型题

## 37. [对称的二叉树](https://leetcode-cn.com/problems/dui-cheng-de-er-cha-shu-lcof/)

请实现一个函数，用来判断一棵二叉树是不是对称的。如果一棵二叉树和它的镜像一样，那么它是对称的。

### 思路

比较左子树的左边和右子树的右边，左子树的右边和右子树的左边即可

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode(int x) { val = x; }

 \* }

 \*/

class Solution {

    boolean issame(TreeNode rl,TreeNode rr){

        if(rl==null&&rr==null) return true;

        if(rl==null||rr==null||rl.val!=rr.val) return false;

        return issame(rl.left,rr.right)&&issame(rl.right,rr.left);

    }

    public boolean isSymmetric(TreeNode root) {

        if(root==null) return true;

        return issame(root.left,root.right);

    }

}

### 小结

经典题

## 38. [旋转图像](https://leetcode-cn.com/problems/rotate-image/)

给定一个 n × n 的二维矩阵 matrix 表示一个图像。请你将图像顺时针旋转 90 度。

你必须在 原地 旋转图像，这意味着你需要直接修改输入的二维矩阵。请不要 使用另一个矩阵来旋转图像。

### 思路

用List<List<Integer>> 记录数组数据，然后再赋值。

### 代码

class Solution {

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

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

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

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

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

                templ.add(matrix[i][j]);

            }

            temp.add(templ);

        }

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

            List<Integer> templ=temp.get(matrix.length-i-1);

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

                matrix[j][i]=templ.get(j);

            }

        }

    }

}

### 小结

ArrayList用get(i)的方法

## 39. [组合总和](https://leetcode-cn.com/problems/combination-sum/)

给定一个无重复元素的正整数数组 candidates 和一个正整数 target ，找出 candidates 中所有可以使数字和为目标数 target 的唯一组合。

candidates 中的数字可以无限制重复被选取。如果至少一个所选数字数量不同，则两种组合是唯一的。

对于给定的输入，保证和为 target 的唯一组合数少于 150 个。

### 思路

回溯题，首先要对数据从小到大排序；然后调用递归函数，递归函数中，判断部分一定要放在for循环外面（重要），然后还有一点是这里的数据是可以重复使用的，这点要注意。还有一点是加入的时候，加入New ArrayList(temp)，不然加入指针就会有问题。LinkedList才有removeLast方法的。

### 代码

class Solution {

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

    void huisu(int[] c,int rear,LinkedList<Integer> temp, int j){

        if(rear<0) return ;

        if(rear==0) {

                ans.add(new ArrayList(temp));return ;

            }

        for(int i=j;i<c.length;i++){

            temp.add(c[i]);

            huisu(c,rear-c[i],temp,i);

            temp.removeLast();

        }

    }

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

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

        Arrays.sort(candidates);

        huisu(candidates,target,temp,0);

        return ans;

    }

}

### 小结

草泥马，细节真的烦死我了。

## 40.[全排列](https://leetcode-cn.com/problems/permutations/)

给定一个不含重复数字的数组 nums ，返回其 **所有可能的全排列** 。你可以 **按任意顺序** 返回答案。

### 思路

还是回溯，但是细节仍要注意，一个是添加的时候要加入New LinkedList;另一个是根据题意，看加入数是否满足，可以在for 里面也可以是外面

### 代码

class Solution {

    List<List<Integer>> ans=new LinkedList<>();

    void huisu(HashMap m,int[] nums,int ns,LinkedList<Integer> temp){

        if(nums.length==ns) {

            ans.add(new LinkedList(temp));

            return ;

        }

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

            if(m.get(nums[i])!=null) continue;

            m.put(nums[i],1);

            temp.add(nums[i]);

            huisu(m,nums,ns+1,temp);

            m.remove(nums[i]);

            temp.removeLast();

        }

    };

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

        HashMap<Integer,Integer> m=new HashMap<>();

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

        huisu(m,nums,0,temp);

        return ans;

    }

}

### 小结

虽然卡了下，但是还好

## 41. [括号生成](https://leetcode-cn.com/problems/generate-parentheses/)

数字 n 代表生成括号的对数，请你设计一个函数，用于能够生成所有可能的并且 **有效的**括号组合。

有效括号组合需满足：左括号必须以正确的顺序闭合。

### 思路

方法还是回溯，首先第一点是在匹配的过程中右括号的数目不能大于左括号数目，且左括号的数目不能大于n，基于以上的条件，当右括号的数目为n的时候，匹配成功

### 代码

class Solution {

    List<String> ans=new LinkedList<>();

    public List<String> generateParenthesis(int n) {

        String temp="";

        huisu(0,0,n,temp);

        return ans;

    }

    void huisu(int countleft,int countright,int n,String temp){

        if(countleft<countright||countleft>n) return;

        if(countright==n){

            String temps=new String(temp);

            ans.add(temps);

            return;

        }

        for(int i=0;i<2;i++){

            if(i==0){

                countleft++;

                temp+='(';

            }

            else {

                countright++;

                temp+=')';

            }

            huisu(countleft,countright,n,temp);

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

            if(i==0) countleft--;

            else countright--;

        }

    }

}

### 小结

主要是对方法的小结，首先string是可以直接加char的，然后是string的截取是string=string.substring(int start, string.length()-1);

## 42. [根据身高重建队列](https://leetcode-cn.com/problems/queue-reconstruction-by-height/)

假设有打乱顺序的一群人站成一个队列，数组 people 表示队列中一些人的属性（不一定按顺序）。每个 people[i] = [hi, ki] 表示第 i 个人的身高为 hi ，前面 正好 有 ki 个身高大于或等于 hi 的人。

### 思路

非常经典的二维数组排序问题，首先对一个元素进行排序，然后对另一个元素进行排序；从本题来讲就是先从身高从高到矮排序，如果遇到身高相同的，先把K比较大的排后面；

然后排序完之后，全部都是从高到低的，然后通过K值依次插入相应的K位置

### 代码

class Solution {

    public int[][] reconstructQueue(int[][] people) {

        Arrays.sort(people,(i1,i2)->i1[0]==i2[0]?i1[1]-i2[1]:i2[0]-i1[0]);

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

        for(int[] person:people){

            temp.add(person[1],person);

        }

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

    }

}

### 小结

非常经典非常典型的二维数组排序题，PAT上面也不少，但是那时候用的是JAVA，因此方法要注意下，是Arrays.sort(二维数组，（第一维数组，第二维数组）->判定条件)，后面那个按K值插入也是非常有意思的

## 43. [电话号码的字母组合](https://leetcode-cn.com/problems/letter-combinations-of-a-phone-number/)

给定一个仅包含数字 2-9 的字符串，返回所有它能表示的字母组合。答案可以按 任意顺序 返回。

给出数字到字母的映射如下（与电话按键相同）。注意 1 不对应任何字母。

### 思路

经典的回溯题

### 代码

class Solution {

    List<String> ans=new LinkedList<String>();

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

    void huisu(String temp,String str){

        if(temp.length()>str.length()) return ;

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

            ans.add(new String(temp));

            return;

        }

        int tempc=str.charAt(temp.length())-'0';

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

            temp+=tele[tempc].charAt(i);

            huisu(temp,str);

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

        }

    }

    public List<String> letterCombinations(String digits) {

        if(digits.length()==0) return ans;

        String temp="";

        huisu(temp,digits);

        return ans;

}

### 小结

经典练手题

## 44. [二叉树展开为链表](https://leetcode-cn.com/problems/flatten-binary-tree-to-linked-list/)

给你二叉树的根结点 root ，请你将它展开为一个单链表：

展开后的单链表应该同样使用 TreeNode ，其中 right 子指针指向链表中下一个结点，而左子指针始终为 null 。

展开后的单链表应该与二叉树 先序遍历 顺序相同。

### 思路

一种就是，找root.left的最右边的子节点pre（前提是root.left存在），然后将root的右子树加入到pre的右子树，再将root.right=root.left,然后root=root.right ,开始新一轮的展开。

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode() {}

 \*     TreeNode(int val) { this.val = val; }

 \*     TreeNode(int val, TreeNode left, TreeNode right) {

 \*         this.val = val;

 \*         this.left = left;

 \*         this.right = right;

 \*     }

 \* }

 \*/

class Solution {

    public void flatten(TreeNode root) {

        if(root==null) return ;

        while(root!=null){

            TreeNode pre=root.left;

            if(pre!=null){

                while(pre.right!=null) pre=pre.right;

                pre.right=root.right;

                root.right=root.left;

                root.left=null;

            }

            root=root.right;

        }

    }

}

### 小结

还有别的思路，下次二刷的时候再看

## 45. [除自身以外数组的乘积](https://leetcode-cn.com/problems/product-of-array-except-self/)

给你一个长度为 n 的整数数组 nums，其中 n > 1，返回输出数组 output ，其中 output[i] 等于 nums 中除 nums[i] 之外其余各元素的乘积。

请**不要使用除法，**且在 O(n) 时间复杂度内完成此题。

### 思路

不能使用除法，先从左边开始算，ans[0]是1，ans[1]是nums[0],nums[2]是nums[0]-nums[1];然后再从右边开始算，ans[4]=nums[4]\*ans[4];描述的不是很正确，具体看代码

### 代码

class Solution {

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

        int[] ans=new int[nums.length];

        int q=1,p=1;

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

            ans[i]=q;

            q\*=nums[i];

        }

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

            p\*=nums[i];

            ans[i-1]\*=p;

        }

        return ans;

    }

}

### 小结

一开始思路是对的，但是用了for(){for()}超时了，稍微修改下就好了。

## 46. [每日温度](https://leetcode-cn.com/problems/daily-temperatures/)

请根据每日 气温 列表 temperatures ，请计算在每一天需要等几天才会有更高的温度。如果气温在这之后都不会升高，请在该位置用 0 来代替。

### 思路

很好的题，如果用两层for循环就超时了，用一个栈，栈内记录的是下标，for循环温度数组，当栈不为空的时候，while检查当前的下标的温度是否大于栈顶下标的温度，如果大于，则弹出，并且记录弹出栈的下标和当前下标的差加入ans数组（注意，这里是while循环而不是if,因为当前的温度可能是好多天温度的较大温度）,在while循环外面加入当前下标。当最后一天时，直接为0,不用查了

### 代码

class Solution {

    public int[] dailyTemperatures(int[] temperatures) {

        int[] ans=new int[temperatures.length];

        Stack<Integer> s=new Stack<>();

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

            while(!s.isEmpty()&&temperatures[i]>temperatures[s.peek()]){

                ans[s.peek()]=i-s.pop();

            }

            s.push(i);

        }

        return ans;

    }

}

### 小结

挺好的题，下次二刷争取会做

## 47. [字母异位词分组](https://leetcode-cn.com/problems/group-anagrams/)

给你一个字符串数组，请你将 字母异位词 组合在一起。可以按任意顺序返回结果列表。

字母异位词 是由重新排列源单词的字母得到的一个新单词，所有源单词中的字母都恰好只用一次。

### 思路

将Strs中的str重新排列，字母从大到小；然后构建一个map<String,LinkedList>,如果存在则加入；不存在则新建

### 代码

class Solution {

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

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

        List<List<String>> ans=new LinkedList<>();

        HashMap<String,List<String>> m=new HashMap<>();

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

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

            Arrays.sort(temp);

            String s=new String(temp);

            if(m.get(s)==null) {

                List<String> ts=new LinkedList<>();

                ts.add(strs[i]);

                m.put(s,ts);

            }

            else m.get(s).add(strs[i]);

        }

        return new ArrayList<List<String>>(m.values());

    }

}

### 小结

这里主要是对方法的小结，首先第一点，想到map，但是没想到里面用LinkedList 作为value;然后是方法，toCharArray可以将字符串转为数组；Arrays.sort可以对char[]直接排序；最后是想要将map的value加入，可以用values()方法。

## 48. [寻找重复数](https://leetcode-cn.com/problems/find-the-duplicate-number/)

给定一个包含 n + 1 个整数的数组 nums ，其数字都在 1 到 n 之间（包括 1 和 n），可知至少存在一个重复的整数。

假设 nums 只有 一个重复的整数 ，找出 这个重复的数 。

你设计的解决方案必须不修改数组 nums 且只用常量级 O(1) 的额外空间。

### 思路

一个数组，1~n,且里面有重复数字，那么肯定存在一个环，就是快慢指针题，

设置一个慢指针，一次走一步，再设置一个快指针，一次走两步，就像跑道里，快的总会追上慢的。当追上时，记录那个位置（但是不知道是下标还是值）；再从pre=0开始，一个个往下找值，另一个继续在环中循环，当相等的时候，便是那个值。

### 代码

class Solution {

    public int findDuplicate(int[] nums) {

        int slow=nums[0],fast=nums[nums[0]];

        while(fast!=slow){

            slow=nums[slow];

            fast=nums[nums[fast]];

        }

        int pre=0,ppre=slow;

        while(pre!=ppre){

            pre=nums[pre];

            ppre=nums[ppre];

        }

        return pre;

    }

}

### 小结

经典快慢指针题，值得思考

## 49. [数组中的第K个最大元素](https://leetcode-cn.com/problems/kth-largest-element-in-an-array/)

给定整数数组 nums 和整数 k，请返回数组中第 **k** 个最大的元素。

请注意，你需要找的是数组排序后的第 k 个最大的元素，而不是第 k 个不同的元素。

### 思路

快排，然后倒数找

### 代码

class Solution {

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

        Arrays.sort(nums);

        return nums[nums.length-k];

    }

}

### 小结

调用库了，题目的意思应该是让你用快排

## 50. [二叉树的层序遍历](https://leetcode-cn.com/problems/binary-tree-level-order-traversal/)

给你一个二叉树，请你返回其按 **层序遍历** 得到的节点值。 （即逐层地，从左到右访问所有节点）。

### 思路

前面做过

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode() {}

 \*     TreeNode(int val) { this.val = val; }

 \*     TreeNode(int val, TreeNode left, TreeNode right) {

 \*         this.val = val;

 \*         this.left = left;

 \*         this.right = right;

 \*     }

 \* }

 \*/

class Solution {

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

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

        List<List<Integer>> ans=new LinkedList<>();

        if(root==null) return ans;

        q.offer(root);

        while(!q.isEmpty()){

            LinkedList<Integer> temp=new LinkedList();

            int length=q.size();

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

                TreeNode tempt=q.poll();

                temp.offer(tempt.val);

                if(tempt.left!=null) q.offer(tempt.left);

                if(tempt.right!=null) q.offer(tempt.right);

            }

            ans.add(temp);

        }

        return ans;

    }

}

### 小结

做过

## 51. [颜色分类](https://leetcode-cn.com/problems/sort-colors/)

给定一个包含红色、白色和蓝色，一共 n 个元素的数组，原地对它们进行排序，使得相同颜色的元素相邻，并按照红色、白色、蓝色顺序排列。

此题中，我们使用整数 0、 1 和 2 分别表示红色、白色和蓝色。

### 思路

荷兰国旗题，我用调方法快排做的，但是不大好，题解的双指针不是特别理解，下次二刷的时候在学习吧

### 代码

class Solution {

public void sortColors(int[] nums) {

int n = nums.length;

int p0 = 0, p1 = 0;

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

if (nums[i] == 1) {

int temp = nums[i];

nums[i] = nums[p1];

nums[p1] = temp;

++p1;

} else if (nums[i] == 0) {

int temp = nums[i];

nums[i] = nums[p0];

nums[p0] = temp;

if (p0 < p1) {

temp = nums[i];

nums[i] = nums[p1];

nums[p1] = temp;

}

++p0;

++p1;

}

}

}

}

### 小结

下次二刷的时候再学习

## 52. [岛屿数量](https://leetcode-cn.com/problems/number-of-islands/)

给你一个由 '1'（陆地）和 '0'（水）组成的的二维网格，请你计算网格中岛屿的数量。

岛屿总是被水包围，并且每座岛屿只能由水平方向和/或竖直方向上相邻的陆地连接形成。

此外，你可以假设该网格的四条边均被水包围。

### 思路

典型的图题，这题求数量，其实就是dfs访问一个点之后把它设置为0即可。

### 代码

class Solution {

    boolean dfs(char[][] grid,int x,int y){

        if(x<0||x>=grid[0].length||y<0||y>=grid.length) return false;

        if(grid[y][x]=='0') return false;

        grid[y][x]='0';

        dfs(grid,x+1,y);

        dfs(grid,x,y+1);

        dfs(grid,x-1,y);

        dfs(grid,x,y-1);

        return true;

    }

    public int numIslands(char[][] grid) {

        int count=0;

        for(int x=0;x<grid[0].length;x++){

            for(int y=0;y<grid.length;y++){

                if(dfs(grid,x,y)) count++;

            }

        }

        return count;

    }

}

### 小结

图题里的最简单的

## [53 目标和](https://leetcode-cn.com/problems/target-sum/)

给你一个整数数组 nums 和一个整数 target 。

向数组中的每个整数前添加 '+' 或 '-' ，然后串联起所有整数，可以构造一个 表达式 ：

例如，nums = [2, 1] ，可以在 2 之前添加 '+' ，在 1 之前添加 '-' ，然后串联起来得到表达式 "+2-1" 。

返回可以通过上述方法构造的、运算结果等于 target 的不同 表达式 的数目。

### 思路

这里采用的是回溯的想法，能够满足计算量，但是有一点注意的是在huisu(n+1,nums[n])尽量使用n+1而非n++,因为可能会使后面的n发生变化

### 代码

class Solution {

    int ans=0;

    void huisu(int n,int count,int[] nums,int target){

        if(n==nums.length-1) {

            if(count==target) {

                ans++;

            }

            return ;

        }

        huisu(n+1,count+nums[n+1],nums,target);

        huisu(n+1,count-nums[n+1],nums,target);

    }

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

        huisu(0,nums[0],nums,target);

        huisu(0,-1\*nums[0],nums,target);

        return ans;

    }

}

### 小结

注意n++和n+1

## 54. [单词搜索](https://leetcode-cn.com/problems/word-search/)

给定一个 m x n 二维字符网格 board 和一个字符串单词 word 。如果 word 存在于网格中，返回 true ；否则，返回 false 。

单词必须按照字母顺序，通过相邻的单元格内的字母构成，其中“相邻”单元格是那些水平相邻或垂直相邻的单元格。同一个单元格内的字母不允许被重复使用。

### 思路

经典的图的遍历回溯题，需要注意的是，visit[][]当作一个全局变量哪怕不是在全局申明的，在for循环之后需要将访问过的点去除

### 代码

class Solution {

    int max=0,xlength=0,ylength=0;

    boolean flag=false;

    String str=new String();

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

    void dfs(int y,int x,int[][] visit,String temp,char[][] board){

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

            if(str.equals(temp)){

                flag=true;

            }

            return ;

        }

        if(flag||y<0||x<0||x>=xlength||y>=ylength||visit[y][x]==1) return ;

        visit[y][x] =1;

        temp+=String.valueOf(board[y][x]);

        for(int i=0;i<4;i++){

            dfs(y+direct[i][0],x+direct[i][1],visit,temp,board);

        }

        visit[y][x] =0;

    }

    public boolean exist(char[][] board, String word) {

        ylength=board.length;

        xlength=board[0].length;

        int[][] visit=new int[ylength][xlength];

        str=word;

        for(int i=0;i<ylength;i++){

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

                visit[i][j]=0;

            }

        }

        for(int i=0;i<ylength;i++){

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

                if(flag) break;

                dfs(i,j,visit,new String(),board);

            }

        }

        return flag;

    }

}

### 小结

Visit访问过的点去除这个卡了我太久了

## [55. 最小路径和](https://leetcode-cn.com/problems/minimum-path-sum/)

给定一个包含非负整数的 *m* x *n* 网格 grid ，请找出一条从左上角到右下角的路径，使得路径上的数字总和为最小。

### 思路

经典动态规划题，动态规划。边界点，方程处理好即可，用回溯会超时

### 代码

class Solution {

    public int minPathSum(int[][] grid) {

        int[][] step=new int[grid.length][grid[0].length];

        step[0][0]=grid[0][0];

        for(int x=0;x<grid[0].length;x++){

            for(int y=0;y<grid.length;y++){

                if(y==0&&x==0) continue;

                else if(y==0){

                    step[y][x]=grid[y][x]+step[y][x-1];

                    continue;

                }

                else if(x==0){

                    step[y][x]=step[y-1][x]+grid[y][x];

                    continue;

                }

                else {

                    step[y][x]=Math.min(step[y][x-1],step[y-1][x])+grid[y][x];

                }

            }

        }

        return step[grid.length-1][grid[0].length-1];

    }

}

### 小结

动态规划

## 56. [不同路径](https://leetcode-cn.com/problems/unique-paths/)

一个机器人位于一个 m x n 网格的左上角 （起始点在下图中标记为 “Start” ）。

机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为 “Finish” ）。

问总共有多少条不同的路径？

### 思路

简单的动态规划题

### 代码

class Solution {

    public int uniquePaths(int m, int n) {

        int[][] step=new int[m][n];

        step[0][0]=1;

        for(int y=0;y<m;y++){

            for(int x=0;x<n;x++){

                if(y==0&&x==0) continue;

                else if(y==0||x==0) step[y][x]=1;

                else  step[y][x]=step[y-1][x]+step[y][x-1];

            }

        }

        return step[m-1][n-1];

    }

}

### 小结

动态规划

## 57. [完全平方数](https://leetcode-cn.com/problems/perfect-squares/)

给定正整数 n，找到若干个完全平方数（比如 1, 4, 9, 16, ...）使得它们的和等于 n。你需要让组成和的完全平方数的个数最少。

给你一个整数 n ，返回和为 n 的完全平方数的 最少数量 。

完全平方数 是一个整数，其值等于另一个整数的平方；换句话说，其值等于一个整数自乘的积。例如，1、4、9 和 16 都是完全平方数，而 3 和 11 不是

### 思路

动态规划，这题的超越方程这样思考，比如说数字17，它想要最小，里面含有一个最大平方数16，去除这个平方数后就是num[1]+一个平方数（平方数为1）；如果是3的话，最大平方数为1，剩下就是num[2],所以num[2]+1

### 代码

class Solution {

    public int numSquares(int n) {

        int[] num=new int[n+1];

        num[0]=0;num[1]=1;

        for(int i=2;i<=n;i++){

            int temp=Integer.MAX\_VALUE;

            for(int j=1;j\*j<=i;j++){

                temp=Math.min(temp,num[i-j\*j]);

            }

            num[i]=temp+1;

        }

        return num[n];

    }

}

### 小结

转移方程比较难想到

## 58.[最佳买卖股票时机含冷冻期](https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/)

给定一个整数数组，其中第 i 个元素代表了第 i 天的股票价格 。​

设计一个算法计算出最大利润。在满足以下约束条件下，你可以尽可能地完成更多的交易（多次买卖一支股票）:

你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。

卖出股票后，你无法在第二天买入股票 (即冷冻期为 1 天)。

### 思路

动态规划，其中每天的状态有三种，分别是持有股票，不持有且冷却时间，不持有且不冷却。

对于持有股票，要么是前一天不持有且不冷却然后今天买入，要么就是昨天买入持有；对于不持有且冷却，就是今天把股票卖出去了，那么状态就是前天持有且卖出；需要注意的是，第零天的时候，持有股票的状态直接设置为-股票价，不然会出问题。

### 代码

class Solution {

    public int maxProfit(int[] prices) {

        int[][] money=new int[prices.length+1][3];

        money[0][0]=-prices[0];

        money[0][1]=0;

        money[0][2]=0;

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

            money[i][0]=Math.max(money[i-1][0],money[i-1][1]-prices[i-1]);//持有股票

            money[i][1]=Math.max(money[i-1][1],money[i-1][2]);//没持有且不冷冻

            money[i][2]=money[i-1][0]+prices[i-1];//没有且冷冻

        }

        return Math.max(money[prices.length][1],money[prices.length][2]);

    }

}

### 小结

好题，值得思考的动态规划

## 59. [买卖股票的最佳时机](https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock/)

给定一个数组 prices ，它的第 i 个元素 prices[i] 表示一支给定股票第 i 天的价格。

你只能选择 某一天 买入这只股票，并选择在 未来的某一个不同的日子 卖出该股票。设计一个算法来计算你所能获取的最大利润。

返回你可以从这笔交易中获取的最大利润。如果你不能获取任何利润，返回 0 。

### 思路

上一题的简化版，补充一点，确认状态有几种，这种状态是上一个哪个状态导致的，做了什么动作

### 代码

class Solution {

    public int maxProfit(int[] prices) {

        int[][] buy=new int[prices.length+1][2];

        buy[0][0]=-prices[0];

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

            buy[i][0]=Math.max(buy[i-1][0],-prices[i-1]);//有票 是因为之前买了或则刚刚买了

            buy[i][1]=Math.max(buy[i-1][1],buy[i-1][0]+prices[i-1]);//没票 因为没买或则卖了

        }

        return Math.max(buy[prices.length][0],buy[prices.length][1]);

    }

}

### 小结

动态规划

## 60[最大子序和](https://leetcode-cn.com/problems/maximum-subarray/)

给定一个整数数组 nums ，找到一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。

### 思路

经典动态规划题

### 代码

class Solution {

    public int maxSubArray(int[] nums) {

        int max=Integer.MIN\_VALUE;

        if(nums.length==1) return nums[0];

        int[] n=new int[nums.length+1];

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

            n[i]=Math.max(nums[i-1],nums[i-1]+n[i-1]); //当前那个nums 上一个

            if(n[i]>max) max=n[i];

        }

        return max;

    }

}

### 小结

经典三部曲

## 61[环形链表 II](https://leetcode-cn.com/problems/linked-list-cycle-ii/)

给定一个链表，返回链表开始入环的第一个节点。 如果链表无环，则返回 null。

为了表示给定链表中的环，我们使用整数 pos 来表示链表尾连接到链表中的位置（索引从 0 开始）。 如果 pos 是 -1，则在该链表中没有环。注意，pos 仅仅是用于标识环的情况，并不会作为参数传递到函数中。

说明：不允许修改给定的链表。

### 思路

经典快慢回环题

### 代码

/\*\*

 \* Definition for singly-linked list.

 \* class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode(int x) {

 \*         val = x;

 \*         next = null;

 \*     }

 \* }

 \*/

public class Solution {

    public ListNode detectCycle(ListNode head) {

        ListNode fast=head,slow=head;

        while(true){

            if(fast==null||fast.next==null) return null;

            fast=fast.next.next;

            slow=slow.next;

            if(fast==slow) break;

        }

        fast=head;

        while(fast!=slow){

            slow=slow.next;

            fast=fast.next;

        }

        return fast;

    }

}

### 小结

经典题

## 62[爬楼梯](https://leetcode-cn.com/problems/climbing-stairs/)

假设你正在爬楼梯。需要 *n* 阶你才能到达楼顶。

每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢？

**注意：**给定 *n* 是一个正整数。

### 思路

最简单的动态规划

### 代码

class Solution {

    public int climbStairs(int n) {

        int[] step=new int[n+1];

        step[0]=1;

        step[1]=1;

        for(int i=2;i<=n;i++){

            step[i]=step[i-1]+step[i-2];

        }

        return step[n];

    }

}

### 小结

最简单的动态规划

## 63. [最长连续序列](https://leetcode-cn.com/problems/longest-consecutive-sequence/)

给定一个未排序的整数数组 nums ，找出数字连续的最长序列（不要求序列元素在原数组中连续）的长度。

请你设计并实现时间复杂度为 O(n) 的算法解决此问题。

### 思路

直接用快排后就很清晰了。

### 代码

class Solution {

    public int longestConsecutive(int[] nums) {

        if(nums.length==0) return 0;

        Arrays.sort(nums);

        int[] n=new int[nums.length];

        n[0]=1;

        int max=1;

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

            if(nums[i]==nums[i-1]) {

                n[i]=n[i-1];continue;

            }

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

            else n[i]=1;

            if(max<n[i]) max=n[i];

        }

        return max;

    }

}

### 小结

简单题

## 63. [最长递增子序列](https://leetcode-cn.com/problems/longest-increasing-subsequence/)

给你一个整数数组 nums ，找到其中最长严格递增子序列的长度。

子序列是由数组派生而来的序列，删除（或不删除）数组中的元素而不改变其余元素的顺序。例如，[3,6,2,7] 是数组 [0,3,1,6,2,2,7] 的子序列。

### 思路

动态规划

### 代码

class Solution {

    public int lengthOfLIS(int[] nums) {

        int[] dp=new int[nums.length];

        int max=0;

        for(int i=0;i<dp.length;i++){

            dp[i]=1;

            for(int j=i-1;j>=0;j--){

                if(nums[j]<nums[i]){

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

                }

            }

            if(dp[i]>max) max=dp[i];

        }

        return max;

    }

}

### 小结

动态规划

## 64. [打家劫舍](https://leetcode-cn.com/problems/house-robber/)

你是一个专业的小偷，计划偷窃沿街的房屋。每间房内都藏有一定的现金，影响你偷窃的唯一制约因素就是相邻的房屋装有相互连通的防盗系统，**如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警**。

给定一个代表每个房屋存放金额的非负整数数组，计算你**不触动警报装置的情况下**，一夜之内能够偷窃到的最高金额。

### 思路

动态规划简单题

### 代码

class Solution {

    public int rob(int[] nums) {

        int[][] dp=new int[nums.length][2];

        dp[0][1]=nums[0];

        for(int i=1;i<dp.length;i++){

            dp[i][1]=dp[i-1][0]+nums[i];//偷了 之前没偷

            dp[i][0]=Math.max(dp[i-1][0],dp[i-1][1]);//没偷  之前没偷,之前偷了

        }

        return Math.max(dp[nums.length-1][1],dp[nums.length-1][0]);

    }

}

### 小结

动态规划

## 65. [跳跃游戏](https://leetcode-cn.com/problems/jump-game/)

给定一个非负整数数组 nums ，你最初位于数组的 **第一个下标** 。

数组中的每个元素代表你在该位置可以跳跃的最大长度。

判断你是否能够到达最后一个下标。

### 思路

简单的贪心，和前面的动态规划不同的是，前面的题是这个状态是前面哪些动作带来的，这题是这个状态带来后面什么变化。

### 代码

class Solution {

    public boolean canJump(int[] nums) {

        boolean[] dp=new boolean[nums.length];

        dp[0]=true;

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

            if(dp[i]){

                int count=0;

                for(int j=i+1;j<nums.length&&count<nums[i];j++,count++)

                    dp[j]=true;

            }

        }

        return dp[nums.length-1];

    }

}

### 小结

简单题

## 66. [零钱兑换](https://leetcode-cn.com/problems/coin-change/)

给你一个整数数组 coins ，表示不同面额的硬币；以及一个整数 amount ，表示总金额。

计算并返回可以凑成总金额所需的 最少的硬币个数 。如果没有任何一种硬币组合能组成总金额，返回 -1 。

你可以认为每种硬币的数量是无限的。

### 思路

背包问题，用贪心不一定能得到最优解，这种背包问题有点类似于前面的平方数那题，剩余多少，该怎么从前面得到，然后每个数字设置为最大

### 代码

class Solution {

    public int coinChange(int[] coins, int amount) {

        int[] dp=new int[amount+1];

        Arrays.fill(dp,amount+1);

        dp[0]=0;

        for(int i=1;i<=amount;i++){

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

                if(i>=coins[j])

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

            }

        }

        return dp[amount]==amount+1?-1:dp[amount];

    }

}

### 小结

背包问题

## 67. [斐波那契数列](https://leetcode-cn.com/problems/fei-bo-na-qi-shu-lie-lcof/)

写一个函数，输入 n ，求斐波那契（Fibonacci）数列的第 n 项（即 F(N)）。斐波那契数列的定义如下：

F(0) = 0,   F(1) = 1

F(N) = F(N - 1) + F(N - 2), 其中 N > 1.

斐波那契数列由 0 和 1 开始，之后的斐波那契数就是由之前的两数相加而得出。

答案需要取模 1e9+7（1000000007），如计算初始结果为：1000000008，请返回 1。

### 思路

方程已经放在那里了，唯一要注意的是取余，加入的时候就取余了

### 代码

class Solution {

    public int fib(int n) {

        int[] dp=new int[n+1];

        dp[0]=0;

        if(n>=1) dp[1]=1;

        else return 0;

        for(int i=2;i<=n;i++){

            dp[i]=(dp[i-1]+dp[i-2])%1000000007;

        }

        return dp[n];

    }

}

### 小结

最简单的动态规划题

## 68. [青蛙跳台阶问题](https://leetcode-cn.com/problems/qing-wa-tiao-tai-jie-wen-ti-lcof/)

一只青蛙一次可以跳上1级台阶，也可以跳上2级台阶。求该青蛙跳上一个 n 级的台阶总共有多少种跳法。

答案需要取模 1e9+7（1000000007），如计算初始结果为：1000000008，请返回 1。

### 思路

简单的动态规划题

### 代码

class Solution {

    public int numWays(int n) {

        int[] dp=new int[n+1];

        if(n==0) return 1;

        dp[0]=0;

        for(int i=1;i<=n;i++){

            if(i==1) dp[i]=1;

            else if(i==2) dp[i]=2;

            else dp[i]=(dp[i-1]+dp[i-2])%1000000007;

        }

        return dp[n];

    }

}

### 小结

简单的动态规划

## 69. [股票的最大利润](https://leetcode-cn.com/problems/gu-piao-de-zui-da-li-run-lcof/)

假设把某股票的价格按照时间先后顺序存储在数组中，请问买卖该股票一次可能获得的最大利润是多少？

### 思路

其实可以不用动态规划做，就是按着那个顺序排下来，记录前面的最便宜的，然后相减，记录利润

### 代码

class Solution {

    public int maxProfit(int[] prices) {

        if(prices.length==0) return 0;

        int[] dp=new int[prices.length+1];

        int cost=0;

        int max=0;

         dp[0]=0;

        cost=prices[0];

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

                dp[i]=Math.max(dp[i-1],prices[i-1]-cost);

             cost=Math.min(prices[i-1],cost);

             if(dp[i]>max) max=dp[i];

            }

        return max;

    }

}

### 小结

简单题

## 70. [连续子数组的最大和](https://leetcode-cn.com/problems/lian-xu-zi-shu-zu-de-zui-da-he-lcof/)

输入一个整型数组，数组中的一个或连续多个整数组成一个子数组。求所有子数组的和的最大值。

要求时间复杂度为O(n)。

### 思路

动态规划简单题

### 代码

class Solution {

    public int maxSubArray(int[] nums) {

        int[] dp=new int[nums.length+1];

        dp[0]=0;

        int max=Integer.MIN\_VALUE;

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

            dp[i]=Math.max(dp[i-1]+nums[i-1],nums[i-1]);

            if(dp[i]>max) max=dp[i];

        }

        return max;

    }

}

### 小结

简单题

## 71. [礼物的最大价值](https://leetcode-cn.com/problems/li-wu-de-zui-da-jie-zhi-lcof/)

在一个 m\*n 的棋盘的每一格都放有一个礼物，每个礼物都有一定的价值（价值大于 0）。你可以从棋盘的左上角开始拿格子里的礼物，并每次向右或者向下移动一格、直到到达棋盘的右下角。给定一个棋盘及其上面的礼物的价值，请计算你最多能拿到多少价值的礼物？

### 思路

做过，简单题

### 代码

class Solution {

    public int maxValue(int[][] grid) {

        int dp[][]=new int[grid.length][grid[0].length];

        for(int y=0;y<grid.length;y++){

            for(int x=0;x<grid[0].length;x++){

                if(x==0&&y==0) dp[0][0]=grid[0][0];

                else if(x==0) dp[y][x]=grid[y][x]+dp[y-1][x];

                else if(y==0) dp[y][x]=grid[y][x]+dp[y][x-1];

                else dp[y][x]=Math.max(dp[y-1][x],dp[y][x-1])+grid[y][x];

            }

        }

        return dp[grid.length-1][grid[0].length-1];

    }

}

### 小结

动态规划

## 72. [把数字翻译成字符串](https://leetcode-cn.com/problems/ba-shu-zi-fan-yi-cheng-zi-fu-chuan-lcof/)

给定一个数字，我们按照如下规则把它翻译为字符串：0 翻译成 “a” ，1 翻译成 “b”，……，11 翻译成 “l”，……，25 翻译成 “z”。一个数字可能有多个翻译。请编程实现一个函数，用来计算一个数字有多少种不同的翻译方法。

### 思路

动态规划，需要注意的点挺多的，特别注意的是506中间出现0的；剩余的用转移方程，一个个字符对应过来看状态即可。

### 代码

class Solution {

    public int translateNum(int num) {

        String str=Integer.toString(num);

        int[] dp=new int[str.length()+1];

        dp[0]=1;

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

            boolean flag=false;

            if(i-2>=0&&(str.charAt(i-1)-'0')+(str.charAt(i-2)-'0')\*10<26&&(str.charAt(i-2)-'0')!=0)

              flag=true;

            dp[i]=dp[i-1];

            if(flag) dp[i]+=dp[i-2];

        }

        return dp[str.length()];

    }

}

### 小结

挺好的题，然后整数字符串是Integer.toString();

## 73. [最长不含重复字符的子字符串](https://leetcode-cn.com/problems/zui-chang-bu-han-zhong-fu-zi-fu-de-zi-zi-fu-chuan-lcof/)

请从字符串中找出一个最长的不包含重复字符的子字符串，计算该最长子字符串的长度。

### 思路

首先是dp，然后是因为出现不重复，所以使用map,然后从头开始遍历，如果是没有加入到map的数字，那么加入map，dp[i]=dp[i-1]+1;如果是已经存在了的，那么left要么是目前值，要么是map里的下标，两者取最大值，然后有效字串是i-left；最后记录一下最大值

### 代码

class Solution {

    public int lengthOfLongestSubstring(String s) {

        int[] dp=new int[s.length()+1];

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

        int max=0,left=0;

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

            if(m.get(s.charAt(i-1))==null){

                m.put(s.charAt(i-1),i);

                dp[i]=dp[i-1]+1;

            }

            else{

                left=Math.max(left,m.get(s.charAt(i-1)));

                dp[i]=i-left;

                 m.put(s.charAt(i-1),i);

            }

            if(dp[i]>max) max=dp[i];

        }

        return max;

    }

}

### 小结

比较好的题目，用到map,动态规划，其中最为难就是left,left可以是上一个同一个字母的下标，也可以是最近更新的left，两者取其大，这个是最难想到的

## 74. [删除链表的节点](https://leetcode-cn.com/problems/shan-chu-lian-biao-de-jie-dian-lcof/)

给定单向链表的头指针和一个要删除的节点的值，定义一个函数删除该节点。

返回删除后的链表的头节点。

### 思路

简单题

### 代码

/\*\*

 \* Definition for singly-linked list.

 \* public class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode(int x) { val = x; }

 \* }

 \*/

class Solution {

    public ListNode deleteNode(ListNode head, int val) {

        ListNode p1=head,p2=head.next;

        if(p1.val==val) return p1.next;

        while(p2!=null){

            if(p2.val==val){

                p1.next=p2.next;

                break;

            }

            p1=p1.next;

            p2=p2.next;

        }

        return head;

    }

}

### 小结

简单题

## 75. [链表中倒数第k个节点](https://leetcode-cn.com/problems/lian-biao-zhong-dao-shu-di-kge-jie-dian-lcof/)

输入一个链表，输出该链表中倒数第k个节点。为了符合大多数人的习惯，本题从1开始计数，即链表的尾节点是倒数第1个节点。

例如，一个链表有 6 个节点，从头节点开始，它们的值依次是 1、2、3、4、5、6。这个链表的倒数第 3 个节点是值为 4 的节点。

### 思考

简单题

### 代码

/\*\*

 \* Definition for singly-linked list.

 \* public class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode(int x) { val = x; }

 \* }

 \*/

class Solution {

    public ListNode getKthFromEnd(ListNode head, int k) {

        ListNode p=head;

        int count=0;

        while(p!=null){

            p=p.next;

            count++;

        }

        for(int i=0;i<count-k;i++) head=head.next;

        return head;

    }

}

### 小结

简单题

## 76. [调整数组顺序使奇数位于偶数前面](https://leetcode-cn.com/problems/diao-zheng-shu-zu-shun-xu-shi-qi-shu-wei-yu-ou-shu-qian-mian-lcof/)

输入一个整数数组，实现一个函数来调整该数组中数字的顺序，使得所有奇数位于数组的前半部分，所有偶数位于数组的后半部分。

### 思路

先把奇数放前面，用p记录位置，偶数加入队列，然后结束后，从p开始，队列全部加入

### 代码

class Solution {

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

        int p=0;

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

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

            if(nums[i]%2==0) q.offer(nums[i]);

            else nums[p++]=nums[i];

        }

        while(!q.isEmpty()){

            nums[p++]=q.poll();

        }

        return nums;

    }

}

### 小结

简单题

## 77. [合并两个排序的链表](https://leetcode-cn.com/problems/he-bing-liang-ge-pai-xu-de-lian-biao-lcof/)

输入两个递增排序的链表，合并这两个链表并使新链表中的节点仍然是递增排序的。

### 思路

简单题

### 代码

/\*\*

 \* Definition for singly-linked list.

 \* public class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode(int x) { val = x; }

 \* }

 \*/

class Solution {

    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {

        ListNode head=new ListNode(),p=head;

        while(l1!=null||l2!=null){

            if(l1==null){

                p.next=l2;break;

            }

            else if(l2==null) {

                p.next=l1;break;

            }

            else {

                ListNode temp=new ListNode();

                if(l1.val>=l2.val) {

                    temp.val=l2.val;

                    l2=l2.next;

                }

                else {

                    temp.val=l1.val;

                    l1=l1.next;

                }

                p.next=temp;

            }

            p=p.next;

        }

        return head.next;

    }

}

### 小结

简单题

## 78. [两个链表的第一个公共节点](https://leetcode-cn.com/problems/liang-ge-lian-biao-de-di-yi-ge-gong-gong-jie-dian-lcof/)

输入两个链表，找出它们的第一个公共节点。

### 思路

用Hash Map

### 代码

/\*\*

 \* Definition for singly-linked list.

 \* public class ListNode {

 \*     int val;

 \*     ListNode next;

 \*     ListNode(int x) {

 \*         val = x;

 \*         next = null;

 \*     }

 \* }

 \*/

public class Solution {

    public ListNode getIntersectionNode(ListNode headA, ListNode headB) {

        HashMap<ListNode,Integer> m=new HashMap<>();

        ListNode p1=headA,p2=headB;

        while(p1!=null){

            m.put(p1,1);

            p1=p1.next;

        }

        while(p2!=null){

            if(m.get(p2)!=null) return p2;

            p2=p2.next;

        }

        return null;

    }

}

### 小结

简单题

## 79. [和为s的两个数字](https://leetcode-cn.com/problems/he-wei-sde-liang-ge-shu-zi-lcof/)

输入一个递增排序的数组和一个数字s，在数组中查找两个数，使得它们的和正好是s。如果有多对数字的和等于s，则输出任意一对即可。

### 思路

简单题

### 代码

class Solution {

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

        int right=nums.length-1;

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

            if(nums[i]>=target){

                right=i;break;

            }

        }

        int p1=0,p2=right;

        int[] ans=new int[2];

        while(p1<p2){

            if(nums[p1]+nums[p2]==target){

                ans[0]=nums[p1];

                ans[1]=nums[p2];

                break;

            }

            else if(nums[p1]+nums[p2]>target)p2--;

            else p1++;

        }

        return ans;

    }

}

### 小结

简单题

## 80. [翻转单词顺序](https://leetcode-cn.com/problems/fan-zhuan-dan-ci-shun-xu-lcof/)

输入一个英文句子，翻转句子中单词的顺序，但单词内字符的顺序不变。为简单起见，标点符号和普通字母一样处理。例如输入字符串"I am a student. "，则输出"student. a am I"。

### 思路

简单题，就是一些方法记一下，一个是char转为数组，String.valueOf;一个是字符串去除首尾空格，String.trim;一个是判读是否为字母String.isLetter

### 代码

class Solution {

    public String reverseWords(String s) {

        s=s.trim();

        Stack<String> st=new Stack<>();

        String temp=new String();

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

            char tc=s.charAt(i);

            if(tc!=' '){

                String t=String.valueOf(tc);

                temp+=t;

                continue;

            }

            else if(temp!=""){

                st.push(temp);

                temp="";

            }

        }

        if(!temp.equals("")) st.push(temp);

        int count=0;

        String ans=new String();

        while(!st.isEmpty()){

            if(count!=0) ans+=" ";

            ans+=st.pop();

            count++;

        }

        return ans;

    }

}

### 小结

简单题

## 81. [矩阵中的路径](https://leetcode-cn.com/problems/ju-zhen-zhong-de-lu-jing-lcof/)

给定一个 m x n 二维字符网格 board 和一个字符串单词 word 。如果 word 存在于网格中，返回 true ；否则，返回 false 。

单词必须按照字母顺序，通过相邻的单元格内的字母构成，其中“相邻”单元格是那些水平相邻或垂直相邻的单元格。同一个单元格内的字母不允许被重复使用。

### 思路

Str的比较最好用第k个字符和String对比，如果用string和string对比容易超时

### 代码

class Solution {

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

    boolean flag=false;

    void huisu(int k,int y,int x,String word,char[][] board,int[][] visit){

        if(x<0||y<0||y>=board.length||x>=board[0].length||flag||visit[y][x]==1||word.charAt(k)!=board[y][x])return ;

        if(k==word.length()-1) flag=true;

        visit[y][x]=1;

        for(int i=0;i<4;i++){

            huisu(k+1,y+direct[i][0],x+direct[i][1],word,board,visit);

        }

        visit[y][x]=0;

    }

    public boolean exist(char[][] board, String word) {

        int[][] visit=new int[board.length][board[0].length];

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

            for(int x=0;x<board[0].length;x++) {

                huisu(0,y,x,word,board,visit);

                if(flag) break;

            }

        }

        return flag;

    }

}

### 小结

三部曲得记清楚

## 82. [机器人的运动范围](https://leetcode-cn.com/problems/ji-qi-ren-de-yun-dong-fan-wei-lcof/)

地上有一个m行n列的方格，从坐标 [0,0] 到坐标 [m-1,n-1] 。一个机器人从坐标 [0, 0] 的格子开始移动，它每次可以向左、右、上、下移动一格（不能移动到方格外），也不能进入行坐标和列坐标的数位之和大于k的格子。例如，当k为18时，机器人能够进入方格 [35, 37] ，因为3+5+3+7=18。但它不能进入方格 [35, 38]，因为3+5+3+8=19。请问该机器人能够到达多少个格子？

### 思路

经典回溯题

### 代码

class Solution {

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

    void huisu(int y,int x,int m,int n,int k,int[][] visit){

        if(x<0||y<0||x>=n||y>=m||visit[y][x]==1) return ;

        int tx=x,ty=y,count=0;

        while(tx!=0||ty!=0){

            count+=tx%10;

            count+=ty%10;

            tx/=10;

            ty/=10;

        }

        if(count>k) return;

        visit[y][x]=1;

        for(int i=0;i<4;i++){

            huisu(y+direct[i][0],x+direct[i][1],m,n,k,visit);

        }

    }

    public int movingCount(int m, int n, int k) {

        int[][] visit=new int[m][n];

        int ans=0;

        huisu(0,0,m,n,k,visit);

        for(int i=0;i<m;i++){

            for(int j=0;j<n;j++){

                if(visit[i][j]==1) ans++;

            }

        }

        return ans;

    }

}

### 小结

简单题

## 83. [二叉树中和为某一值的路径](https://leetcode-cn.com/problems/er-cha-shu-zhong-he-wei-mou-yi-zhi-de-lu-jing-lcof/)

输入一棵二叉树和一个整数，打印出二叉树中节点值的和为输入整数的所有路径。从树的根节点开始往下一直到叶节点所经过的节点形成一条路径。

### 思路

很典型的回溯题，但是需要注意的是，回溯的时候加入ListLinked的时候记得是new ，当不满足的时候也不要return

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode() {}

 \*     TreeNode(int val) { this.val = val; }

 \*     TreeNode(int val, TreeNode left, TreeNode right) {

 \*         this.val = val;

 \*         this.left = left;

 \*         this.right = right;

 \*     }

 \* }

 \*/

class Solution {

    List<List<Integer>> ans=new LinkedList<>();

    void huisu(TreeNode root,int target,LinkedList temp,int count){

        if(root==null) return ;

        count+=root.val;

        temp.add(root.val);

        if(count==target&&root.left==null&&root.right==null)

        {

            ans.add(new LinkedList<Integer>(temp));

        }

        huisu(root.left,target,temp,count);

        huisu(root.right,target,temp,count);

        temp.removeLast();

    }

    public List<List<Integer>> pathSum(TreeNode root, int target) {

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

        huisu(root,target,temp,0);

        return ans;

    }

}

### 小结

简单题，但是细节没处理好

## 84. [二叉搜索树的第k大节点](https://leetcode-cn.com/problems/er-cha-sou-suo-shu-de-di-kda-jie-dian-lcof/)

给定一棵二叉搜索树，请找出其中第k大的节点。

### 思路

二叉搜索树的中序遍历是递增的，逆中序遍历的话是递减

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode(int x) { val = x; }

 \* }

 \*/

class Solution {

    int ans=0,kk=0;

    void dfs(TreeNode root){

        if(root==null) return ;

        dfs(root.right);

        if(kk==0) return ;

        if(--kk==0) ans=root.val;

        dfs(root.left);

    }

    public int kthLargest(TreeNode root, int k) {

        kk=k;

        dfs(root);

        return ans;

    }

}

### 小结

心思灵巧的题目

## 85. [转换字符串的最少操作次数](https://leetcode-cn.com/problems/minimum-moves-to-convert-string/)

给你一个字符串 s ，由 n 个字符组成，每个字符不是 'X' 就是 'O' 。

一次**操作** 定义为从 s 中选出 **三个连续字符**并将选中的每个字符都转换为 'O' 。注意，如果字符已经是 'O' ，只需要保持 **不变** 。

返回将 s 中所有字符均转换为 'O' 需要执行的 **最少** 操作次数。

### 思路

我用了dp，但是贪心更优（遇到X就次数+1，然后跳过后两个）

### 代码

class Solution {

    public int minimumMoves(String s) {

        int[] dp=new int[s.length()+1];

        Arrays.fill(dp,0);

        boolean flag=false;

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

            if(i<=3) {

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

                    flag=true;

                    }

                if(flag)dp[i]=1;

            }

            else {

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

                    int min=dp[i-3]>dp[i-2]?dp[i-2]:dp[i-3];

                    dp[i]=(min>dp[i-1]?dp[i-1]:min)+1;

                }

                else dp[i]=dp[i-1];

            }

        }

        return dp[s.length()];

    }

}

### 小结

很明显 贪心更优解

## 86. [找出缺失的观测数据](https://leetcode-cn.com/problems/find-missing-observations/)

现有一份 n + m 次投掷单个**六面** 骰子的观测数据，骰子的每个面从 1 到 6 编号。观测数据中缺失了 n 份，你手上只拿到剩余 m 次投掷的数据。幸好你有之前计算过的这 n + m 次投掷数据的 **平均值** 。

给你一个长度为 m 的整数数组 rolls ，其中 rolls[i] 是第 i 次观测的值。同时给你两个整数 mean 和 n 。

返回一个长度为 n 的数组，包含所有缺失的观测数据，且满足这 n + m 次投掷的 **平均值** 是 mean 。如果存在多组符合要求的答案，只需要返回其中任意一组即可。如果不存在答案，返回一个空数组。

k 个数字的 **平均值** 为这些数字求和后再除以 k 。

注意 mean 是一个整数，所以 n + m 次投掷的总和需要被 n + m 整除。

### 思路

变相的贪心算法，用平均值去填

### 代码

class Solution {

    public int[] missingRolls(int[] rolls, int mean, int n) {

        int count=n+rolls.length;

        int all=mean\*count;

        for(int i=0;i<count-n;i++){

            all-=rolls[i];

        }

        int[] ans=new int[n];

        int index=0;

        while(n>0&&index<ans.length){

            if(all/n>6||all/n<=0) return new int[0];

            else {

                ans[index++]=all/n;

                all-=all/n;

                n--;

            }

        }

        if(n==0) return ans;

        else return new int[0];

    }

}

### 小结

好题

## 87. [二叉搜索树与双向链表](https://leetcode-cn.com/problems/er-cha-sou-suo-shu-yu-shuang-xiang-lian-biao-lcof/)

输入一棵二叉搜索树，将该二叉搜索树转换成一个排序的循环双向链表。要求不能创建任何新的节点，只能调整树中节点指针的指向。

### 思路

二叉搜索树，用中序遍历记录顺序，然后记录开头和结尾，然后按顺序连接即可

### 代码

/\*

// Definition for a Node.

class Node {

    public int val;

    public Node left;

    public Node right;

    public Node() {}

    public Node(int \_val) {

        val = \_val;

    }

    public Node(int \_val,Node \_left,Node \_right) {

        val = \_val;

        left = \_left;

        right = \_right;

    }

};

\*/

class Solution {

    Queue<Node> q=new LinkedList<Node>();

    void dfs(Node root){

        if(root==null) return ;

        dfs(root.left);

        q.offer(root);

        dfs(root.right);

    }

    public Node treeToDoublyList(Node root) {

        if(root==null) return root;

        dfs(root);

        Node head=q.peek(),rear=new Node();

        while(!q.isEmpty()){

            Node temp=q.poll();

            if(q.peek()!=null){

                temp.right=q.peek();

                q.peek().left=temp;

            }

            else{

                rear=temp;

            }

        }

        head.left=rear;

        rear.right=head;

        return head;

    }

}

### 小结

二叉排序树经典题

## 88. [把数组排成最小的数](https://leetcode-cn.com/problems/ba-shu-zu-pai-cheng-zui-xiao-de-shu-lcof/)

输入一个非负整数数组，把数组里所有数字拼接起来排成一个数，打印能拼接出的所有数字中最小的一个。

### 思路

核心算法是两个字符串拼接起来，如果x+y>y+x，那么就用y+x;拼接的方法是Arrays.sort(temp,(x,y)->(x+y).compareTo(y+x)),其中compareTo方法是比较字符串的ASSIC码，apped是StringBuilder方法

### 代码

class Solution {

    public String minNumber(int[] nums) {

        StringBuilder ans=new StringBuilder();

        String[] tempn=new String[nums.length];

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

            tempn[i]=String.valueOf(nums[i]);

        }

        Arrays.sort(tempn,(x,y)->(x+y).compareTo(y+x));

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

            ans.append(tempn[i]);

        }

        return ans.toString();

    }

}

### 小结

好题

89.

## 89[扑克牌中的顺子](https://leetcode-cn.com/problems/bu-ke-pai-zhong-de-shun-zi-lcof/)

从**若干副扑克牌**中随机抽 5 张牌，判断是不是一个顺子，即这5张牌是不是连续的。2～10为数字本身，A为1，J为11，Q为12，K为13，而大、小王为 0 ，可以看成任意数字。A 不能视为 14。

### 思路

这个题，首先看有没有重复的牌（set解决），set的fangf方法有.add(),contain();然后因为总共五张牌，如果是顺子的话，最大值减去最小值应该小于5（0除外）

### 代码

class Solution {

    public boolean isStraight(int[] nums) {

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

       int max=Integer.MIN\_VALUE,min=Integer.MAX\_VALUE;

       for(int i=0;i<5;i++){

           if(nums[i]==0) continue;

           if(nums[i]>max) max=nums[i];

           if(nums[i]<min) min=nums[i];

           if(s.contains(nums[i])) return false;

           s.add(nums[i]);

       }

       return (max-min)<5;

    }

}

### 小结

思路理解了就是easy

## 90. [二叉树的深度](https://leetcode-cn.com/problems/er-cha-shu-de-shen-du-lcof/)

输入一棵二叉树的根节点，求该树的深度。从根节点到叶节点依次经过的节点（含根、叶节点）形成树的一条路径，最长路径的长度为树的深度。

例如：

给定二叉树 [3,9,20,null,null,15,7]，

### 思路

简单题，层次遍历

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode(int x) { val = x; }

 \* }

 \*/

class Solution {

    public int maxDepth(TreeNode root) {

        if(root==null) return 0;

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

        q.offer(root);

        int ans=0;

        while(!q.isEmpty()){

            int length=q.size();

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

                TreeNode temp=q.poll();

                if(temp.left!=null)q.offer(temp.left);

                if(temp.right!=null)q.offer(temp.right);

            }

            ans++;

        }

        return ans;

    }

}

### 小结

简单题

## 91. [平衡二叉树](https://leetcode-cn.com/problems/ping-heng-er-cha-shu-lcof/)

输入一棵二叉树的根节点，判断该树是不是平衡二叉树。如果某二叉树中任意节点的左右子树的深度相差不超过1，那么它就是一棵平衡二叉树。

### 思路

递归

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode(int x) { val = x; }

 \* }

 \*/

class Solution {

    boolean flag=true;

    int dfs(TreeNode root){

        if(root==null) return 0;

        if(!flag) return 0;

        int left=dfs(root.left)+1;

        int right=dfs(root.right)+1;

        if(Math.abs(left-right)>1) flag=false;

        return Math.max(left,right);

    }

    public boolean isBalanced(TreeNode root) {

        dfs(root);

        return flag;

    }

}

### 小结

简单题

## 92. [二叉搜索树的最近公共祖先](https://leetcode-cn.com/problems/er-cha-sou-suo-shu-de-zui-jin-gong-gong-zu-xian-lcof/)

给定一个二叉搜索树, 找到该树中两个指定节点的最近公共祖先。

百度百科中最近公共祖先的定义为：“对于有根树 T 的两个结点 p、q，最近公共祖先表示为一个结点 x，满足 x 是 p、q 的祖先且 x 的深度尽可能大（一个节点也可以是它自己的祖先）。”

例如，给定如下二叉搜索树:  root = [6,2,8,0,4,7,9,null,null,3,5]

### 思路

如果是公共祖先，那么在二叉排序树里一定在两边；如果不是公共祖先，那么就会在左（右）子树里

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode(int x) { val = x; }

 \* }

 \*/

class Solution {

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

        while(root!=null){

            if(p.val<root.val&&q.val<root.val){

                root=root.left;

            }

            else if(p.val>root.val&&q.val>root.val){

                root=root.right;

            }

            else break;

        }

        return root;

    }

}

### 小结

简单题

## 93. [II. 二叉树的最近公共祖先](https://leetcode-cn.com/problems/er-cha-shu-de-zui-jin-gong-gong-zu-xian-lcof/)

给定一个二叉树, 找到该树中两个指定节点的最近公共祖先。

百度百科中最近公共祖先的定义为：“对于有根树 T 的两个结点 p、q，最近公共祖先表示为一个结点 x，满足 x 是 p、q 的祖先且 x 的深度尽可能大（一个节点也可以是它自己的祖先）。”

### 思路

思路也是类似的，但是它不是搜索二叉树，所以从它的左子树开始找，如果找能找到p||q,那么就在，然后从它的右子树开始找，如果能找到p||q，那么它在右子树，那么当前的root就是它要最深祖先；如果有一个为null的话，那么找到的那个就是它的祖先

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode(int x) { val = x; }

 \* }

 \*/

class Solution {

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

       if(root==null) return null;

       if(p.val==root.val||q.val==root.val) return root;

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

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

       if(left!=null&&right!=null) return root;

       if(left==null) return right;

       else return left;

    }

}

### 小结

上一题的翻版，但是更加的有一般意义

## 94. [实现 Trie (前缀树)](https://leetcode-cn.com/problems/implement-trie-prefix-tree/)

Trie（发音类似 "try"）或者说 前缀树 是一种树形数据结构，用于高效地存储和检索字符串数据集中的键。这一数据结构有相当多的应用情景，例如自动补完和拼写检查。

请你实现 Trie 类：

Trie() 初始化前缀树对象。

void insert(String word) 向前缀树中插入字符串 word 。

boolean search(String word) 如果字符串 word 在前缀树中，返回 true（即，在检索之前已经插入）；否则，返回 false 。

boolean startsWith(String prefix) 如果之前已经插入的字符串 word 的前缀之一为 prefix ，返回 true ；否则，返回 false 。

### 思路

用类的方法，用Trie[] child记录下一个单词，用end记录是否有这个单词

### 代码

class Trie {

    private Trie[] child;

    private boolean end;

    public Trie() {

         child=new Trie[26];

         end=false;

    }

    public void insert(String word) {

        Trie node=this;

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

            int temp =word.charAt(i)-'a';

            if(node.child[temp]==null){

                node.child[temp]=new Trie();

            }

            node=node.child[temp];

        }

        node.end=true;

    }

    public boolean search(String word) {

        Trie node=this;

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

            if(node.child[word.charAt(i)-'a']==null) return false;

            node=node.child[word.charAt(i)-'a'];

        }

        return node.end;

    }

    public boolean startsWith(String prefix) {

        Trie node=this;

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

            if(node.child[prefix.charAt(i)-'a']==null) return false;

            node=node.child[prefix.charAt(i)-'a'];

        }

        return true;

    }

}

### 小结

挺好的题目

## 95. [把二叉搜索树转换为累加树](https://leetcode-cn.com/problems/convert-bst-to-greater-tree/)

给出二叉 搜索 树的根节点，该树的节点值各不相同，请你将其转换为累加树（Greater Sum Tree），使每个节点 node 的新值等于原树中大于或等于 node.val 的值之和。

### 思路

主要需要理解什么是累加树，就是节点比它小的节点之和加它本身。又因为他是二叉树，所以逆序搜索就得到从大到小的序列

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode() {}

 \*     TreeNode(int val) { this.val = val; }

 \*     TreeNode(int val, TreeNode left, TreeNode right) {

 \*         this.val = val;

 \*         this.left = left;

 \*         this.right = right;

 \*     }

 \* }

 \*/

class Solution {

    Stack<Integer> s=new Stack<Integer>();

    int temp=0;

    void dfs(TreeNode root){

        if(root.right!=null) dfs(root.right);

        s.push(temp);

        temp+=root.val;

        if(root.left!=null) dfs(root.left);

    }

    void add(TreeNode root){

        if(root.left!=null) add(root.left);

        root.val+=s.pop();

        if(root.right!=null) add(root.right);

    }

    public TreeNode convertBST(TreeNode root) {

        if(root==null) return root;

        dfs(root);

        add(root);

        return root;

    }

}

### 小结

简单题

## 96. [不同的二叉搜索树](https://leetcode-cn.com/problems/unique-binary-search-trees/)

给你一个整数 n ，求恰由 n 个节点组成且节点值从 1 到 n 互不相同的 **二叉搜索树** 有多少种？返回满足题意的二叉搜索树的种数。

### 思路

这题想到动态规划不容易，因为每一个节点都可能轮到当根节点，然后查找二叉树它又比较特别，左边比根节点小，右边比根节点大；那么右边的子树可能性\*右边子树可能性的它的可能性，第一个for循环是节点的个数，第二个for循环是轮流根节点

### 代码

class Solution {

    public int numTrees(int n) {

        int[] dp=new int[n+1];

        dp[0]=1;

        dp[1]=1;

        for(int i=2;i<=n;i++){

            for(int j=0;j<i;j++){

                dp[i]+=dp[j]\*dp[i-j-1];

            }

        }

        return dp[n];

    }

}

### 小结

比较好的动态规划

## 97. [回文子串](https://leetcode-cn.com/problems/palindromic-substrings/)

给你一个字符串 s ，请你统计并返回这个字符串中 回文子串 的数目。

回文字符串 是正着读和倒过来读一样的字符串。

子字符串 是字符串中的由连续字符组成的一个序列。

具有不同开始位置或结束位置的子串，即使是由相同的字符组成，也会被视作不同的子串。

### 思路

用动态规划，dp[][]，二重数组是用来记左右下标的，如果String.charAt(i)==String.charAt(j)时，然后i-j<=2时候可以直接判断为回文；有两重for(for)，第一重i是右下标，第二重j是左下标，dp[i]是随着数字的增加，所以每当i增加1时，就是新一轮判断；然后引入了dp[i-1][j+1]因为在二重for循环时候，会出现循环判断

### 代码

class Solution {

    public int countSubstrings(String s) {

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

        int count=0;

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

            for(int j=0;j<=i;j++){

                if(s.charAt(j)==s.charAt(i)&&(i-j<=2||dp[i-1][j+1])){

                    dp[i][j]=true;

                    count++;

                }

                else

                    dp[i][j]=false;

            }

        }

        return count;

    }

}

### 小结

很好的题，dp[][]用来左右下标，还有重复出现的问题

## 98. [打家劫舍 III](https://leetcode-cn.com/problems/house-robber-iii/)

在上次打劫完一条街道之后和一圈房屋后，小偷又发现了一个新的可行窃的地区。这个地区只有一个入口，我们称之为“根”。 除了“根”之外，每栋房子有且只有一个“父“房子与之相连。一番侦察之后，聪明的小偷意识到“这个地方的所有房屋的排列类似于一棵二叉树”。 如果两个直接相连的房子在同一天晚上被打劫，房屋将自动报警。

计算在不触动警报的情况下，小偷一晚能够盗取的最高金额。

### 思路

树形的dp,但是是从叶节点开始往上偷，分为两个状态，偷了和没偷，然后节点的权值记录金额；然后又有左右两个叶子节点，所以要对两个考虑。

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode() {}

 \*     TreeNode(int val) { this.val = val; }

 \*     TreeNode(int val, TreeNode left, TreeNode right) {

 \*         this.val = val;

 \*         this.left = left;

 \*         this.right = right;

 \*     }

 \* }

 \*/

class Solution {

    int[] dp(TreeNode root){

        if(root==null) return new int[2];

        int[] ans=new int[2];

        int[] left=dp(root.left);

        int[] right=dp(root.right);

        int notrob=Math.max(left[0],left[1])+Math.max(right[0],right[1]);//0是没抢，1是抢了

        int rob=root.val+left[0]+right[0];

        ans[0]=notrob;ans[1]=rob;

        return ans;

    }

    public int rob(TreeNode root) {

        int[] ans=dp(root);

        return Math.max(ans[0],ans[1]);

    }

}

### 小结

比较变种的动态规划题

## 99. [路径总和 III](https://leetcode-cn.com/problems/path-sum-iii/)

给定一个二叉树的根节点 root ，和一个整数 targetSum ，求该二叉树里节点值之和等于 targetSum 的 路径 的数目。

路径 不需要从根节点开始，也不需要在叶子节点结束，但是路径方向必须是向下的（只能从父节点到子节点）。

### 思路

前缀和，把每一个节点加上之前路径的和（temp），然后用一个map记录之前路径的和，如果出现temp+root.val-target 在map.getOrDefault 里有值，那么ans+上这个值，然后再将m.put(temp+root.val) 需要注意的是，map以形参传递，但是每次都改的是同一个值，因此回溯结束后需要-1；另一个注意的点是root,root是比较特殊的，需要在回溯之前就有m.put(0,1),否则将会遗漏

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode() {}

 \*     TreeNode(int val) { this.val = val; }

 \*     TreeNode(int val, TreeNode left, TreeNode right) {

 \*         this.val = val;

 \*         this.left = left;

 \*         this.right = right;

 \*     }

 \* }

 \*/

class Solution {

    int rv,ans=0;

    void dfs(TreeNode root,int temp,HashMap<Integer,Integer> m){

        if(root==null) return ;

        temp+=root.val;

        ans+=m.getOrDefault(temp-rv,0);

        m.put(temp,m.getOrDefault(temp,0)+1);

        dfs(root.left,temp,m);

        dfs(root.right,temp,m);

        m.put(temp,m.getOrDefault(temp,0)-1);

    };

    public int pathSum(TreeNode root, int targetSum) {

        HashMap<Integer,Integer> m=new HashMap<>();

        rv=targetSum;

        m.put(0,1);

        dfs(root,0,m);

        return ans;

    }

}

### 小结

好题

## 100. [接雨水](https://leetcode-cn.com/problems/trapping-rain-water/)

给定 n 个非负整数表示每个宽度为 1 的柱子的高度图，计算按此排列的柱子，下雨之后能接多少雨水。

### 思路

双指针的题目，比较好的思路是，设立两个数组，从左到右取最大（前面最高和当前值），就是假设右边是最高的，然后雨水能全接满；再从右到左取最大，假设左边是最高的，雨水能全接满；最后一次遍历，就是left和right数组取最小值-high[i].就是当前位置雨水的数量，累加一下即可。

### 代码

class Solution {

    public int trap(int[] height) {

        int ans=0;

        int[] left=new int[height.length+1],right=new int[height.length+1];

        left[0]=height[0];

        right[height.length]=height[height.length-1];

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

            left[i]=Math.max(left[i-1],height[i-1]);

        }

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

            right[i]=Math.max(right[i+1],height[i-1]);

        }

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

            ans+=Math.min(right[i],left[i])-height[i-1];

        }

        return ans;

    }

}

### 小结

双指针的比较好的题，面积之减

## 101. [字符串解码](https://leetcode-cn.com/problems/decode-string/)

给定一个经过编码的字符串，返回它解码后的字符串。

编码规则为: k[encoded\_string]，表示其中方括号内部的 encoded\_string 正好重复 k 次。注意 k 保证为正整数。

你可以认为输入字符串总是有效的；输入字符串中没有额外的空格，且输入的方括号总是符合格式要求的。

此外，你可以认为原始数据不包含数字，所有的数字只表示重复的次数 k ，例如不会出现像 3a 或 2[4] 的输入。

### 思路

用递归，每一次的 [ 表示递归的入口

### 代码

class Solution {

    int i=0;

    public String decodeString(String s) {

        return digui(s,0);

    }

    String  digui(String s,int tempn){

        StringBuilder temp=new StringBuilder();

        int n=0;

        while(i<s.length()){

           if(s.charAt(i)>='0'&&s.charAt(i)<='9'){

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

           }

           else if(s.charAt(i)>='a'&&s.charAt(i)<='z'){

               temp.append(s.charAt(i++));

           }

           else if(s.charAt(i)=='['){

               i++;

               temp.append(digui(s,n));

               n=0;

           }

           else if(s.charAt(i++)==']'){

               StringBuilder temps=new StringBuilder();

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

                   temps.append(temp.toString());

               }

               return temps.toString();

           }

       }

       return temp.toString();

    }

}

### 小结

好题

## 102. [LRU 缓存机制](https://leetcode-cn.com/problems/lru-cache/)

运用你所掌握的数据结构，设计和实现一个  LRU (最近最少使用) 缓存机制 。

实现 LRUCache 类：

LRUCache(int capacity) 以正整数作为容量 capacity 初始化 LRU 缓存

int get(int key) 如果关键字 key 存在于缓存中，则返回关键字的值，否则返回 -1 。

void put(int key, int value) 如果关键字已经存在，则变更其数据值；如果关键字不存在，则插入该组「关键字-值」。当缓存容量达到上限时，它应该在写入新数据之前删除最久未使用的数据值，从而为新的数据值留出空间。

### 思路

用hashmap和双链表，双链表来记录node，map记录这个节点存在不存在，然后node需要自己写一个类，具体的情况，具体的修改参考要求

### 代码

class LRUCache {

    class LRU{

        int key;

        int value;

        LRU pre;

        LRU next;

        LRU(){}

        LRU(int key,int value){

            this.key=key;

            this.value=value;

        }

    }

    HashMap<Integer,LRU> m=new HashMap<>();

    int capacity;

    int size;

    LRU head,tail;

    public LRUCache(int capacity) {

        size=0;

        this.capacity=capacity;

        head=new LRU();

        tail=new LRU();

        head.next=tail;

        tail.pre=head;

    }

    public int get(int key) {

        LRU target=m.get(key);

        if(target==null) return -1;

        else {

            removenode(target);

            movetohead(target);

        }

        return target.value;

    }

    public void put(int key, int value) {

        LRU target=m.get(key);

        if(target!=null){

            target.value=value;

            removenode(target);

            movetohead(target);

            return ;

        }

        else{

            LRU temp=new LRU(key,value);

            size++;

            if(size>capacity){

                size--;

                m.put(tail.pre.key,null);

                removetail();

            }

            movetohead(temp);

            m.put(key,temp);

        }

    }

    void movetohead(LRU temp){

        head.next.pre=temp;

        temp.pre=head;

        temp.next=head.next;

        head.next=temp;

    }

    void removetail(){

        removenode(tail.pre);

    }

    void removenode(LRU temp){

        temp.pre.next=temp.next;

        temp.next.pre=temp.pre;

    }

}

/\*\*

 \* Your LRUCache object will be instantiated and called as such:

 \* LRUCache obj = new LRUCache(capacity);

 \* int param\_1 = obj.get(key);

 \* obj.put(key,value);

 \*/

### 小结

明白思路还是简单的，但是细节特别麻烦

## 103. [找到字符串中所有字母异位词](https://leetcode-cn.com/problems/find-all-anagrams-in-a-string/)

给定两个字符串 s 和 p，找到 s 中所有 p 的 异位词 的子串，返回这些子串的起始索引。不考虑答案输出的顺序。

异位词 指由相同字母重排列形成的字符串（包括相同的字符串）。

### 思路

滑动窗口，数组记录，下标动，然后Arrays.equals()可以比较数组是否相同

### 代码

class Solution {

    public List<Integer> findAnagrams(String s, String p) {

        LinkedList ans=new LinkedList();

        int m=s.length(),n=p.length();

        int[] a=new int[26],b=new int[26];

        if(n>m) return ans;

        for(int i=0;i<n;i++){

            a[s.charAt(i)-'a']+=1;

            b[p.charAt(i)-'a']+=1;

        }

        if(Arrays.equals(a,b)) ans.add(0);

        for(int i=n;i<m;i++){

            a[s.charAt(i)-'a']+=1;

            a[s.charAt(i-n)-'a']-=1;

            if(Arrays.equals(a,b)) ans.add(i-n+1);

        }

        return ans;

    }

}

### 小结

滑动窗口，第一次碰见

## [104. 和为 K 的子数组](https://leetcode-cn.com/problems/subarray-sum-equals-k/)

给你一个整数数组 nums 和一个整数 k ，请你统计并返回该数组中和为 k的连续子数组的个数。

### 思路

连续子序列之和为前缀和

### 代码

class Solution {

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

        int ans=0,pre=0;

        HashMap<Integer,Integer> m=new HashMap<>();

        m.put(0,1);

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

            pre+=nums[i];

            ans+=m.getOrDefault(pre-k,0);

            m.put(pre,m.getOrDefault(pre,0)+1);

        }

        return ans;

    }

}

### 小结

前缀和

## 105[合并区间](https://leetcode-cn.com/problems/merge-intervals/)

以数组 intervals 表示若干个区间的集合，其中单个区间为 intervals[i] = [starti, endi] 。请你合并所有重叠的区间，并返回一个不重叠的区间数组，该数组需恰好覆盖输入中的所有区间。

### 思路

先将左区间从小到大排序，如果一样右区间选小值，然后用for循环合并

### 代码

class Solution {

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

        // if(intervals.length==1) return {{intervals[0][0],intervals[0][1]}};

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

        Arrays.sort(intervals,(a,b)->a[0]==b[0]?a[1]-b[1]:a[0]-b[0]);

        int left=0,right=1;

        while(right<=intervals.length){

            int leftbound=intervals[left][0];

            int rightbound=intervals[left][1];

            for(;right<intervals.length;right++){

                if(intervals[right][0]<=rightbound){

                    rightbound=Math.max(rightbound,intervals[right][1]);

                }

                else break;

            }

            list.add(new int[]{leftbound,rightbound});

            if(right==intervals.length) break;

            left=right;

        }

        int[][] ans=new int[list.size()][2];

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

            ans[i][0]=list.get(i)[0];

            ans[i][1]=list.get(i)[1];

        }

        return ans;

    }

}

### 小结

典型题

## 106[二叉树中的最大路径和](https://leetcode-cn.com/problems/binary-tree-maximum-path-sum/)

**路径** 被定义为一条从树中任意节点出发，沿父节点-子节点连接，达到任意节点的序列。同一个节点在一条路径序列中 **至多出现一次** 。该路径**至少包含一个**节点，且不一定经过根节点。

**路径和** 是路径中各节点值的总和。

给你一个二叉树的根节点 root ，返回其 **最大路径和** 。

### 思路

一开始给难度hard吓到了，其实很简单，就是dfs，从叶子节点往上，如果<0当作0处理，然后加一加，大于max就替代，一直往上

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode() {}

 \*     TreeNode(int val) { this.val = val; }

 \*     TreeNode(int val, TreeNode left, TreeNode right) {

 \*         this.val = val;

 \*         this.left = left;

 \*         this.right = right;

 \*     }

 \* }

 \*/

class Solution {

    int ans=Integer.MIN\_VALUE;

    public int maxPathSum(TreeNode root) {

        dfs(root);

        return ans;

    }

    int dfs(TreeNode root){

        if(root==null) return 0;

        int left=Math.max(dfs(root.left),0);

        int right=Math.max(dfs(root.right),0);

        int temp=root.val+left+right;

        ans=Math.max(ans,temp);

        return root.val+Math.max(left,right);

    }

}

### 小结

不知道的还以为是难题

## 107. [最大正方形](https://leetcode-cn.com/problems/maximal-square/)

在一个由 '0' 和 '1' 组成的二维矩阵内，找到只包含 '1' 的最大正方形，并返回其面积。

### 思路

动态规划，dp[][]取边长，和dp[i-1][j],dp[i][j-1],dp[i-1][j-1]作比较

### 代码

class Solution {

    public int maximalSquare(char[][] matrix) {

        int area=0;

        if(matrix.length==0||matrix[0].length==0) return area;

        int y=matrix.length, x=matrix[0].length;

        int[][] dp=new int[y][x];

        for(int i=0;i<y;i++){

            for(int j=0;j<x;j++){

                if(matrix[i][j]=='1'){

                    if(i==0||j==0) dp[i][j]=1;

                    else{

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

                    }

                    area=Math.max(dp[i][j],area);

                }

            }

        }

        return area\*area;

    }

}

### 小结

那个傻逼char真的傻逼

## [108 滑动窗口最大值](https://leetcode-cn.com/problems/sliding-window-maximum/)

给你一个整数数组 nums，有一个大小为 k 的滑动窗口从数组的最左侧移动到数组的最右侧。你只可以看到在滑动窗口内的 k 个数字。滑动窗口每次只向右移动一位。

返回滑动窗口中的最大值。

### 思路

本来是堆加一个map可以很简单的解决了，但是java里堆没有类，所以用一个双向队列来，双向队列记录的是下标（因为通过下标可以查值），然后如果滑动窗口踢出了最大值的下标，pollFirst,如果新加的数字大于队尾数字，则将队尾踢出

### 代码

class Solution {

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

        Deque<Integer> q=new ArrayDeque<Integer>();

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

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

            while(!q.isEmpty()&&i-k+1>q.getFirst()) q.pollFirst();

            while(!q.isEmpty()&&nums[i]>nums[q.getLast()]) q.pollLast();

            q.offer(i);

            if(i>=k-1) ans[j++]=nums[q.getFirst()];

        }

        return ans;

     }

}

### 小结

滑动窗口

## 109[乘积最大子数组](https://leetcode-cn.com/problems/maximum-product-subarray/)

给你一个整数数组 nums ，请你找出数组中乘积最大的连续子数组（该子数组中至少包含一个数字），并返回该子数组所对应的乘积。

### 思路

动态规划，但是需要注意的是负数×负数的话会变正，因此需要一个dpmin记录最小值

### 代码

class Solution {

    public int maxProduct(int[] nums) {

        int[] dpmax=new int[nums.length+1];

        int[] dpmin=new int[nums.length+1];

        dpmax[0]=1;dpmin[0]=1;

        int max=Integer.MIN\_VALUE;

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

            dpmax[i]=Math.max(Math.max(dpmin[i-1]\*nums[i-1],dpmax[i-1]\*nums[i-1]),nums[i-1]);

            dpmin[i]=Math.min(Math.min(dpmin[i-1]\*nums[i-1],dpmax[i-1]\*nums[i-1]),nums[i-1]);

            max=Math.max(dpmax[i],max);

        }

        return max;

    }

}

### 小结

动态规划

## 110. [最短无序连续子数组](https://leetcode-cn.com/problems/shortest-unsorted-continuous-subarray/)

给你一个整数数组 nums ，你需要找出一个 连续子数组 ，如果对这个子数组进行升序排序，那么整个数组都会变为升序排序。

请你找出符合题意的 最短 子数组，并输出它的长度。

### 思路

复制一个数组，然后快排排序；然后和元素组对比，记录不同的位置

### 代码

class Solution {

    public int findUnsortedSubarray(int[] nums) {

        int[] temp=new int[nums.length];

        boolean flag=true;

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

            temp[i]=nums[i];

        }

        Arrays.sort(temp);

        if(Arrays.equals(nums,temp)) return 0;

        int left=0,right=0;

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

            if(nums[i]!=temp[i]){

                if(flag){

                    left=i;right=i;

                    flag=false;

                }

                else {

                    right=i;

                }

            }

        }

        return right-left+1;

    }

}

### 小结

简单题

## 111. [验证二叉搜索树](https://leetcode-cn.com/problems/validate-binary-search-tree/)

给你一个二叉树的根节点 root ，判断其是否是一个有效的二叉搜索树。

**有效** 二叉搜索树定义如下：

* 节点的左子树只包含**小于**当前节点的数。
* 节点的右子树只包含 **大于** 当前节点的数。
* 所有左子树和右子树自身必须也是二叉搜索树。

### 思路

中序遍历，然后就是数字的范围

### 代码

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode() {}

 \*     TreeNode(int val) { this.val = val; }

 \*     TreeNode(int val, TreeNode left, TreeNode right) {

 \*         this.val = val;

 \*         this.left = left;

 \*         this.right = right;

 \*     }

 \* }

 \*/

class Solution {

    LinkedList<Long> l=new LinkedList<>();

    void dfs(TreeNode root){

        if(root==null) return ;

        dfs(root.left);

        l.add((long)root.val);

        dfs(root.right);

    }

    public boolean isValidBST(TreeNode root) {

        dfs(root);

        Long temp=Long.MIN\_VALUE;

        for(Long i:l){

            if(i<=temp) return false;

            temp=i;

        }

        return true;

    }

}

### 小结

简单题

## 112. [最长回文子串](https://leetcode-cn.com/problems/longest-palindromic-substring/)

给你一个字符串 s，找到 s 中最长的回文子串。

### 思路

同之前的题，但是有一点需要注意的是substring会忽略重复的部分，因此输出字符串的时候需要先输入到char[]再转化为String

### 代码

class Solution {

    public String longestPalindrome(String s) {

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

        int begin=0,length=0;

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

            for(int j=0;j<=i;j++){

                if(s.charAt(j)==s.charAt(i)&&(i-j<=2||dp[j+1][i-1])){

                    dp[j][i]=true;

                    if(length<i-j+1){

                        begin=j;

                        length=i-j+1;

                    }

                }

                else dp[j][i]=false;

            }

        }

        char[] temp=new char[length];

        for(int i=0;i<length;i++){

            temp[i]=s.charAt(begin+i);

        }

        return  new String(temp);

    }

}

### 小结

Substring卡了我好久，气死了

## 113[三数之和](https://leetcode-cn.com/problems/3sum/)

给你一个包含 n 个整数的数组 nums，判断 nums 中是否存在三个元素 a，b，c ，使得 a + b + c = 0 ？请你找出所有和为 0 且不重复的三元组。

注意：答案中不可以包含重复的三元组。

### 思路

先快排，这题的精髓在于去重，第一个是遍历i的时候去重，然后是相等的时候，左右指针left和right去重，精华在这里；还有一点是如果i为下标时候就大于0时，后面没有比较的必要性了

### 代码

class Solution {

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

        List<List<Integer>> ans= new LinkedList<List<Integer>>();

        if(nums.length<3) return ans;

        Arrays.sort(nums);

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

            if(nums[i]>0) break;

            if(i>0&&nums[i]==nums[i-1]) continue;

            int left=i+1,right=nums.length-1;

            while(left<right){

                int sum=nums[left]+nums[right]+nums[i];

                if(sum==0){

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

                    temp.add(nums[i]);

                    temp.add(nums[left]);

                    temp.add(nums[right]);

                    ans.add(temp);

                while(left<right&&nums[left]==nums[left+1]) left++;

                while(left<right&&nums[right]==nums[right-1]) right--;

                    left++;

                    right--;

                }

                if(sum<0) left++;

                if(sum>0) right--;

            }

            }

            return ans;

        }

        }

### 思路

好题

## 114. [前 K 个高频元素](https://leetcode-cn.com/problems/top-k-frequent-elements/)

给你一个整数数组 nums 和一个整数 k ，请你返回其中出现频率前 k 高的元素。你可以按 **任意顺序** 返回答案。

### 思路

一开始的是想用最小堆，堆太难写了用桶排；桶排的话是map记录所有的数字和出现的频率，用keySet()可以遍历所有的key；然后再写一个List[]链表数组，下标是记录发生的次数，链表是内容。用数组列表不能记录发生相同次数的数字

### 代码

class Solution {

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

        HashMap<Integer,Integer> m=new HashMap<Integer,Integer>();

        for(int i:nums){

            m.put(i,m.getOrDefault(i,0)+1);

        }

        List<Integer>[] temp=new List[nums.length+1];int[] ans=new int[k];

        for(Integer i:m.keySet()){

            int v=m.get(i);

            if(temp[v]==null) temp[v]=new LinkedList<Integer>();

            temp[v].add(i);

        }

        for(int count=0, i=nums.length;i>=0&&count<k;i--){

            if(temp[i]!=null) {

                for(int t:temp[i]){

                    ans[count++]=t;

                }

            }

        }

        return ans;

    }

}

### 小结

第一次用桶排写题

## 115. [编辑距离](https://leetcode-cn.com/problems/edit-distance/)

给你两个单词 word1 和 word2，请你计算出将 word1 转换成 word2所使用的最少操作数 。

你可以对一个单词进行如下三种操作：

* 插入一个字符
* 删除一个字符
* 替换一个字符

### 思路

这道题用dp比较难想，dp[i][j]就是word1的前i个字符和word2前[j]个字符转化为需要多少次，如果是相同，那么就是dp[i-1][j-1]次；如果不同，那么就是dp[i-1][j] （word1的前i-1个字符转化，第i个字符增加） dp[i][j-1] (word1的前i-1个字符转化，第i个删除) dp[i-1][j-1]word 前i-1个字符转化，第i个字符转化的基础上再加一步，最后取dp[word1.length][word2.length]即可

### 代码

class Solution {

    public int minDistance(String word1, String word2) {

        int[][] dp=new int[word1.length()+1][word2.length()+1];

        for(int i=1;i<=word1.length();i++) dp[i][0]=dp[i-1][0]+1;

        for(int i=1;i<=word2.length();i++) dp[0][i]=dp[0][i-1]+1;

        for(int i=1;i<=word1.length();i++){

            for(int j=1;j<=word2.length();j++){

                if(word1.charAt(i-1)==word2.charAt(j-1)) dp[i][j]=dp[i-1][j-1];

                else {

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

                }

            }

        }

        return dp[word1.length()][word2.length()];

    }

}

### 小结

Dp题 ，没想到

## 116. [任务调度器](https://leetcode-cn.com/problems/task-scheduler/)

给你一个用字符数组 tasks 表示的 CPU 需要执行的任务列表。其中每个字母表示一种不同种类的任务。任务可以以任意顺序执行，并且每个任务都可以在 1 个单位时间内执行完。在任何一个单位时间，CPU 可以完成一个任务，或者处于待命状态。

然而，两个**相同种类** 的任务之间必须有长度为整数n的冷却时间，因此至少有连续 n 个单位时间内 CPU 在执行不同的任务，或者在待命状态。

你需要计算完成所有任务所需要的**最短时间** 。

### 思路

填桶题，记录下出现最多的任务，然后它的数量就是任务序列出现的次数，然后一个任务序列的长度是n+1（因为有冷却时间），然后相乘，但是需要主义的是最后一排，最后一排是不需要冷却时间的，因此需要另外计算，就是和最大数量出现过的字母相同的次数，即last，出现一个++；最后一点需要注意的是，如果种类太多了，导致冷却时间都无法满足所有的，此时的时间就是任务的长度

### 代码

class Solution {

    public int leastInterval(char[] tasks, int n) {

        int[] count=new int[26];

        int max=Integer.MIN\_VALUE;

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

            count[tasks[i]-'A']+=1;

            max=Math.max(max,count[tasks[i]-'A']);

        }

        int last=0;

        for(int i:count){

            if(max==i) last++;

        }

        return Math.max((n+1)\*(max-1)+last,tasks.length);

    }

}

### 小结

贪心算法，挺好的题

## 117[单词拆分](https://leetcode-cn.com/problems/word-break/)

给定一个**非空**字符串 s 和一个包含**非空**单词的列表 wordDict，判定 s 是否可以被空格拆分为一个或多个在字典中出现的单词。

### 思路

动态规划，根据题目是完全拆分的，然后对于每一个I,遍历是否存在一个单词使得它dp[i-length]==true 如果存在那么它为true;

### 代码

class Solution {

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

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

        dp[0]=true;

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

            for(String temp:wordDict){

                int length=temp.length();

                if(i>=length&&s.substring(i-length,i).equals(temp))

                    {

                        dp[i]=dp[i-length];

                    }

                if(dp[i]==true) break;

            }

        }

        return dp[s.length()];

    }

}

### 小结

一般的动态规划题

## 118. [分割等和子集](https://leetcode-cn.com/problems/partition-equal-subset-sum/)

给你一个 **只包含正整数**的 **非空**数组 nums 。请你判断是否可以将这个数组分割成两个子集，使得两个子集的元素和相等。

### 思路

动态规划，dp[i][j] i表示前i个数，j表示当前的数字

### 代码

class Solution {

    public boolean canPartition(int[] nums) {

        int count=0;

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

            count+=nums[i];

        }

        if(count%2!=0) return false;;

        count/=2;

        Arrays.sort(nums);

        boolean[][] dp=new boolean[nums.length+1][count+1];

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

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

                dp[i][j]=dp[i-1][j];

                if(nums[i-1]==j) {

                    dp[i][j]=true;

                    continue;

                }

                if(nums[i-1]<=j) dp[i][j]=dp[i][j]||dp[i-1][j-nums[i-1]];

                if(dp[i][count]) return true;

            }

        }

        return dp[nums.length][count];

    }

}

### 小结

我好菜

## 119. [搜索旋转排序数组](https://leetcode-cn.com/problems/search-in-rotated-sorted-array/)

整数数组 nums 按升序排列，数组中的值 互不相同 。

在传递给函数之前，nums 在预先未知的某个下标 k（0 <= k < nums.length）上进行了 旋转，使数组变为 [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]（下标 从 0 开始 计数）。例如， [0,1,2,4,5,6,7] 在下标 3 处经旋转后可能变为 [4,5,6,7,0,1,2] 。

给你 旋转后 的数组 nums 和一个整数 target ，如果 nums 中存在这个目标值 target ，则返回它的下标，否则返回 -1 。

### 思路

旋转后，部分的排序还是升序的，那么用nums[mid]和num[left]比较查哪部分是升序的，然后再升序部分里查是否存在target,如果存在，则用二分；不存在那么就对零部分查看；最后注意=号

### 代码

class Solution {

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

        if(nums.length==0) return -1;

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

        while(left<=right){

            int mid=(left+right)/2;

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

            if(nums[mid]>=nums[left]){

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

                    right=mid-1;

                }

                else {

                    left=mid+1;

                }

            }

            else if(nums[mid]<nums[right]){

                if(target>nums[mid]&&target<=nums[right]) left=mid+1;

                else right=mid-1;

            }

        }

    return -1;

    }

}

### 小结

感觉不难但是又不好做

## 220. [最小覆盖子串](https://leetcode-cn.com/problems/minimum-window-substring/)

给你一个字符串 s 、一个字符串 t 。返回 s 中涵盖 t 所有字符的最小子串。如果 s 中不存在涵盖 t 所有字符的子串，则返回空字符串 "" 。

注意：

对于 t 中重复字符，我们寻找的子字符串中该字符数量必须不少于 t 中该字符数量。

如果 s 中存在这样的子串，我们保证它是唯一的答案。

### 思路

滑动窗口，用一个temp数组和count来记录子串是否满足要求，然后开始遍历，如果满足的话，左指针开始移动直至不满足，然后和最短长度比较，比它还小则记录最小的长度和起始位置。

### 代码

class Solution {

    public String minWindow(String s, String t) {

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

        int[] temp=new int[256];

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

            temp[t.charAt(i)]+=1;

        }

        int l=0,r=0,length=Integer.MAX\_VALUE,start=0,count=t.length();

        while(r<s.length()){

            if(temp[s.charAt(r)]>0) count--;

            temp[s.charAt(r)]-=1;

            if(count==0){

                while(count==0){

                    temp[s.charAt(l)]+=1;

                    if(temp[s.charAt(l)]>0) count++;

                    l++;

                }

                if(r-l+2<length){

                    length=r-l+2;

                    start=l-1;

                }

            }

            r++;

        }

        if(length==Integer.MAX\_VALUE) return "";

        return s.substring(start,start+length);

    }

}

### 小结

小问题让我抓狂

## 221. [在排序数组中查找元素的第一个和最后一个位置](https://leetcode-cn.com/problems/find-first-and-last-position-of-element-in-sorted-array/)

给定一个按照升序排列的整数数组 nums，和一个目标值 target。找出给定目标值在数组中的开始位置和结束位置。

如果数组中不存在目标值 target，返回 [-1, -1]。

### 思路

二分查，查到后再找区间

### 代码

class Solution {

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

        int start=-1,end=-1,left=0,right=nums.length-1,find=-1;

        while(left<=right){

            int mid=(left+right)/2;

            if(nums[mid]==target){

                find=mid;

                break;

            }

            if(target<nums[mid]){

                right=mid-1;

            }

            else left=mid+1;

        }

        int[] ans={-1,-1};

        if(find==-1) return ans;

        while(find>=0&&nums[find]==target) find--;

        start=++find;

        while(find<nums.length&&nums[find]==target) find++;

        end=--find;

        ans[0]=start;

        ans[1]=end;

        return ans;

    }

}

### 小结

简单题

## 222. [下一个排列](https://leetcode-cn.com/problems/next-permutation/)

实现获取 下一个排列 的函数，算法需要将给定数字序列重新排列成字典序中下一个更大的排列（即，组合出下一个更大的整数）。

如果不存在下一个更大的排列，则将数字重新排列成最小的排列（即升序排列）。

必须 原地 修改，只允许使用额外常数空间。

### 思路

这个就是一个固定思路，先从后往前找升序的i和i-1，然后第一个大于nums[i]的j，然后交换i和j位置，再将从i到尾巴进行升序排列

### 代码

class Solution {

    public void nextPermutation(int[] nums) {

        if(nums.length<=1) return;

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

            if(nums[i]>nums[i-1]){

                for(int j=nums.length-1;j>=i;j--){

                    if(nums[j]>nums[i-1]){

                        int temp=nums[i-1];

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

                        nums[j]=temp;break;

                    }

                }

                Arrays.sort(nums,i,nums.length);return;

            }

        }

        Arrays.sort(nums);

    }

}

### 小结

固定套路题

## 223. [二维数组中的查找](https://leetcode-cn.com/problems/er-wei-shu-zu-zhong-de-cha-zhao-lcof/)

在一个 n \* m 的二维数组中，每一行都按照从左到右递增的顺序排序，每一列都按照从上到下递增的顺序排序。请完成一个高效的函数，输入这样的一个二维数组和一个整数，判断数组中是否含有该整数。

### 思路

逆时针旋转45%，发现是类似于查找二叉树

### 代码

class Solution {

    public boolean findNumberIn2DArray(int[][] matrix, int target) {

        if(matrix.length==0||matrix[0].length==0) return false;

        int x=0,y=matrix[0].length-1;

        while(x<matrix.length&&y>=0){

            if(matrix[x][y]>target) y--;

            else if(matrix[x][y]<target) x++;

            else return true;

        }

        return false;

    }

}

### 小结

很巧妙的题

## 224. [数组中出现次数超过一半的数字](https://leetcode-cn.com/problems/shu-zu-zhong-chu-xian-ci-shu-chao-guo-yi-ban-de-shu-zi-lcof/)

数组中有一个数字出现的次数超过数组长度的一半，请找出这个数字。

你可以假设数组是非空的，并且给定的数组总是存在多数元素。

### 思路

简单题

### 代码

class Solution {

    public int majorityElement(int[] nums) {

        HashMap<Integer,Integer> m=new HashMap<>();

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

            m.put(nums[i],m.getOrDefault(nums[i],0)+1);

        }

        for(int i:m.keySet()){

            if(m.get(i)>nums.length/2) return i;

        }

        return 0;

    }

}

### 小结

简单题

## 225. [构建乘积数组](https://leetcode-cn.com/problems/gou-jian-cheng-ji-shu-zu-lcof/)

给定一个数组 A[0,1,…,n-1]，请构建一个数组 B[0,1,…,n-1]，其中 B[i] 的值是数组 A 中除了下标 i 以外的元素的积, 即 B[i]=A[0]×A[1]×…×A[i-1]×A[i+1]×…×A[n-1]。不能使用除法。

### 思路

建立两个数组

代码class Solution {

    public int[] constructArr(int[] a) {

        int[] left=new int[a.length+1],right=new int[a.length+2],ans=new int[a.length];

        left[0]=1;

        right[a.length+1]=1;

        for(int i=1;i<=a.length;i++) left[i]=left[i-1]\*a[i-1];

        for(int i=a.length;i>=1;i--) right[i]=right[i+1]\*a[i-1];

        for(int i=1;i<=a.length;i++){

            ans[i-1]=left[i-1]\*right[i+1];

        }

        return ans;

    }

}

### 小结

太累了，简单题都想半天

## 226. [I. 剪绳子](https://leetcode-cn.com/problems/jian-sheng-zi-lcof/)

给你一根长度为 n 的绳子，请把绳子剪成整数长度的 m 段（m、n都是整数，n>1并且m>1），每段绳子的长度记为 k[0],k[1]...k[m-1] 。请问 k[0]\*k[1]\*...\*k[m-1] 可能的最大乘积是多少？例如，当绳子的长度是8时，我们把它剪成长度分别为2、3、3的三段，此时得到的最大乘积是18。

### 思路

贪心算法，非整除时取平均值或则平均值+1为长度，最后乘上剩余长度；当整除时直接算即可

### 代码

class Solution {

    public int cuttingRope(int n) {

        int max=Integer.MIN\_VALUE;

        for(int m=2;m<=n;m++){

            int temp=n/m,count=1;

            if(n%m==0){

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

                    count\*=temp;

                }

                max=Math.max(count,max);

            }

            else {

                int t=n;

                for(int i=1;i<m;i++){

                    count\*=temp;

                    t-=temp;

                }

                max=Math.max(count\*t,max);

                t=n;

                count=1;

                for(int i=1;i<m&&t>0;i++){

                    count\*=(temp+1);

                    t-=(temp+1);

                }

                if(t<=0) continue;

                max=Math.max(count\*t,max);

            }

        }

        return max;

    }

}

### 小结

贪心算法

## 227. [和为s的连续正数序列](https://leetcode-cn.com/problems/he-wei-sde-lian-xu-zheng-shu-xu-lie-lcof/)

输入一个正整数 target ，输出所有和为 target 的连续正整数序列（至少含有两个数）。

序列内的数字由小到大排列，不同序列按照首个数字从小到大排列。

### 思路

枚举法

### 代码

class Solution {

    public int[][] findContinuousSequence(int target) {

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

        for(int i=1;i<=target/2+1;i++){

            int sum=0;

            for(int j=i;sum<=target;j++){

                sum+=j;

                if(sum==target){

                    int[] temp=new int[j-i+1];

                    for(int k=i;k<=j;k++) temp[k-i]=k;

                    ans.add(temp);

                }

            }

        }

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

    }

}

### 小结

主要是二维链表转化为二维数组，用LinkedList<int[]> ans记录，最后ans.toArray(new int[ans.size()][])即可

## 228. [圆圈中最后剩下的数字](https://leetcode-cn.com/problems/yuan-quan-zhong-zui-hou-sheng-xia-de-shu-zi-lcof/)

0,1,···,n-1这n个数字排成一个圆圈，从数字0开始，每次从这个圆圈里删除第m个数字（删除后从下一个数字开始计数）。求出这个圆圈里剩下的最后一个数字。

例如，0、1、2、3、4这5个数字组成一个圆圈，从数字0开始每次删除第3个数字，则删除的前4个数字依次是2、0、4、1，因此最后剩下的数字是3。

### 思路

约瑟夫环，可以建一个链表，然后找到一个排除一个

### 代码

class Solution {

    public int lastRemaining(int n, int m) {

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

      for(int i=0;i<n;i++){

          l.add(i);

      }

      int start=0;

      while(n>1){

          start=(start+m-1)%n;

          l.remove(start);

          n--;

      }

    return l.get(0);

    }

}

### 小结

ArrayList的效率大于LinkedList

## 229. [顺时针打印矩阵](https://leetcode-cn.com/problems/shun-shi-zhen-da-yin-ju-zhen-lcof/)

输入一个矩阵，按照从外向里以顺时针的顺序依次打印出每一个数字。

### 思路

简单模拟题

### 代码

class Solution {

    public int[] spiralOrder(int[][] matrix) {

        if(matrix.length==0||matrix[0].length==0) return new int[0];

        int x=matrix[0].length,y=matrix.length,start=0,count=0,startx=-1,starty=0;

        int[] ans=new int[x\*y];

        while(start<x\*y){

            for(startx+=1;startx<x-count&&start<x\*y;startx++) ans[start++]=matrix[starty][startx];

            startx--;

            for(starty+=1;starty<y-count&&start<x\*y;starty++) ans[start++]=matrix[starty][startx];

            starty--;

            for(startx-=1;startx>=count&&start<x\*y;startx--) ans[start++]=matrix[starty][startx];

            startx++;

            for(starty-=1;starty>count&&start<x\*y;starty--) ans[start++]=matrix[starty][startx];

            starty++;

            count++;

        }

        return ans;

    }

}

### 小结

主要是逆时针最后一圈的时候 ，注意y>count而非y>=count，并且y++不能省略

## 330. [栈的压入、弹出序列](https://leetcode-cn.com/problems/zhan-de-ya-ru-dan-chu-xu-lie-lcof/)

输入两个整数序列，第一个序列表示栈的压入顺序，请判断第二个序列是否为该栈的弹出顺序。假设压入栈的所有数字均不相等。例如，序列 {1,2,3,4,5} 是某栈的压栈序列，序列 {4,5,3,2,1} 是该压栈序列对应的一个弹出序列，但 {4,3,5,1,2} 就不可能是该压栈序列的弹出序列。

### 思路

用栈和HashMap来模拟，HashMap用来查找是否存在

### 代码

class Solution {

    public boolean validateStackSequences(int[] pushed, int[] popped) {

        HashMap<Integer,Integer> m=new HashMap<>();

        Stack<Integer> s=new Stack<>();

        int p=0;

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

            if(m.get(popped[i])==null){

                while(p<pushed.length&&m.get(popped[i])==null){

                    m.put(pushed[p],m.getOrDefault(pushed[p],0)+1);

                    s.push(pushed[p++]);

                }

                if(m.get(popped[i])!=null){

                    m.put(popped[i],null);

                    s.pop();

                }

                else return false;

            }

            else {

                if(s.pop()!=popped[i]) return false;

                continue;

            }

        }

        return true;

    }

}

### 小结

简单题，就是时间和空间比较垃圾

## 331. [把字符串转换成整数](https://leetcode-cn.com/problems/ba-zi-fu-chuan-zhuan-huan-cheng-zheng-shu-lcof/)

写一个函数 StrToInt，实现把字符串转换成整数这个功能。不能使用 atoi 或者其他类似的库函数。

首先，该函数会根据需要丢弃无用的开头空格字符，直到寻找到第一个非空格的字符为止。

当我们寻找到的第一个非空字符为正或者负号时，则将该符号与之后面尽可能多的连续数字组合起来，作为该整数的正负号；假如第一个非空字符是数字，则直接将其与之后连续的数字字符组合起来，形成整数。

该字符串除了有效的整数部分之后也可能会存在多余的字符，这些字符可以被忽略，它们对于函数不应该造成影响。

注意：假如该字符串中的第一个非空格字符不是一个有效整数字符、字符串为空或字符串仅包含空白字符时，则你的函数不需要进行转换。

在任何情况下，若函数不能进行有效的转换时，请返回 0。

说明：

假设我们的环境只能存储 32 位大小的有符号整数，那么其数值范围为 [−231,  231 − 1]。如果数值超过这个范围，请返回  INT\_MAX (231 − 1) 或 INT\_MIN (−231) 。

### 思路

题目不难，就是特殊数字比较烦

### 代码

class Solution {

    public int strToInt(String str) {

        long ans=0;

        boolean start=false,fuhao=false;

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

            if(i==0&&!Character.isDigit(str.charAt(i))&&str.charAt(i)!=' '&&str.charAt(i)!='-'&&str.charAt(i)!='+') return 0;

            else if(str.charAt(i)==' '&&!start) continue;

            else if(str.charAt(i)=='-'&&!start) {

                start=true;

                fuhao=true;

            }

            else if(str.charAt(i)=='+'&&!start) {

                start=true;

                fuhao=false;

            }

            else if(Character.isDigit(str.charAt(i))){

                ans=ans\*10+str.charAt(i)-'0';

                start=true;

            }

            else if(!Character.isDigit(str.charAt(i))) break;

            if(ans>Integer.MAX\_VALUE){

                if(fuhao) return Integer.MIN\_VALUE;

                else return Integer.MAX\_VALUE;

            }

        }

        if(fuhao) ans=-1\*ans;

        return (int)ans;

    }

}

### 小结

Character.isDigit这个方法记一下

## 332. [II. 队列的最大值](https://leetcode-cn.com/problems/dui-lie-de-zui-da-zhi-lcof/)

请定义一个队列并实现函数 max\_value 得到队列里的最大值，要求函数max\_value、push\_back 和 pop\_front 的均摊时间复杂度都是O(1)。

### 思路

队列找最大值用双向队列

### 代码

class MaxQueue {

    Deque<Integer> dq;

    Queue<Integer> q;

    public MaxQueue() {

        dq=new ArrayDeque<Integer>();

        q=new LinkedList<Integer>();

    }

    public int max\_value() {

        if(q.isEmpty()) return -1;

        return dq.getFirst();

    }

    public void push\_back(int value) {

        q.offer(value);

        while(!dq.isEmpty()&&dq.getLast()<value) dq.removeLast();

        dq.offer(value);

    }

    public int pop\_front() {

        if(q.isEmpty()) return -1;

        int temp=q.poll();

        if(temp==dq.getFirst()) dq.removeFirst();

        return temp;

    }

}

/\*\*

 \* Your MaxQueue object will be instantiated and called as such:

 \* MaxQueue obj = new MaxQueue();

 \* int param\_1 = obj.max\_value();

 \* obj.push\_back(value);

 \* int param\_3 = obj.pop\_front();

 \*/

### 小结

找最大值用双向队列

## 333. [字符串的排列](https://leetcode-cn.com/problems/zi-fu-chuan-de-pai-lie-lcof/)

输入一个字符串，打印出该字符串中字符的所有排列。

你可以以任意顺序返回这个字符串数组，但里面不能有重复元素。

### 思路

这种for(int i=0)的需要在for循环内写visit和temp[] 退出的判断在for外，主函数不需要for,因为是从0开始的

### 代码

class Solution {

    LinkedList<String> tempans=new LinkedList<>();

    HashMap<String,Integer> m=new HashMap<>();

    int[] visit;

    void huisu(char[] temp,String s,int n){

        if(n==s.length()) {

            if(m.get(new String(temp))==null){

                m.put(new String(temp),1);

                tempans.add(new String(temp));

            }

            return ;

        }

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

            if(visit[j]==1) continue ;

            visit[j]=1;

            temp[n]=s.charAt(j);

            huisu(temp,s,n+1);

            visit[j]=0;

        }

    }

    public String[] permutation(String s) {

        visit=new int[s.length()];

        huisu(new char[s.length()],s,0);

        return tempans.toArray(new String[tempans.size()]);

    }

}

### 小结

区别和别的种类的回溯

## 334. [n个骰子的点数](https://leetcode-cn.com/problems/nge-tou-zi-de-dian-shu-lcof/)

把n个骰子扔在地上，所有骰子朝上一面的点数之和为s。输入n，打印出s的所有可能的值出现的概率。

你需要用一个浮点数数组返回答案，其中第 i 个元素代表这 n 个骰子所能掷出的点数集合中第 i 小的那个的概率。

### 思路

动态规划，第一个是取double的话，赋值时需要/6.0而非/6,第二个是如果投掷两次 那么和为1的就不可能了，输出答案时需要注意

### 代码

class Solution {

    public double[] dicesProbability(int n) {

        double[][] dp=new double[n+1][n\*6+1];

        for(int i=0;i<=6;i++) dp[1][i]=(1.0/6.0);

        for(int i=1;i<=n;i++){

            for(int j=i;j<=i\*6;j++){

                for(int k=1;k<=6;k++){

                    if(i==1||j-k<=0) break;

                    dp[i][j]+=(double)(dp[i-1][j-k]/6.0);

                }

            }

        }

        double[] ans=new double[n\*5+1];

        for(int i=1,j=n;i<=n\*5+1;i++,j++){

            ans[i-1]=dp[n][j]/1.0;        }

        return ans;

    }

}

### 小结

好题

## 335. [丑数](https://leetcode-cn.com/problems/chou-shu-lcof/)

我们把只包含质因子 2、3 和 5 的数称作丑数（Ugly Number）。求按从小到大的顺序的第 n 个丑数。

 5

### 思路

一开始用质数除，然后超时了，最后用动态规划的答案去试，因为丑数肯定是/2 /3 /5肯定是丑数，所以它可以用另一个丑数推导出来的，因此用动态规划

### 代码

class Solution {

    public int nthUglyNumber(int n) {

       int[] dp=new int[n+1];

       int a=0,b=0,c=0;

       dp[0]=1;

       for(int i=1;i<=n;i++){

           int n2=dp[a]\*2,n3=dp[b]\*3,n5=dp[c]\*5;

           dp[i]=Math.min(n2,Math.min(n3,n5));

           if(dp[i]==n2) a++;

           if(dp[i]==n3) b++;

           if(dp[i]==n5) c++;

       }

        return dp[n-1];

    }

}

### 小结

比较新巧的方法

## 336. [复杂链表的复制](https://leetcode-cn.com/problems/fu-za-lian-biao-de-fu-zhi-lcof/)

请实现 copyRandomList 函数，复制一个复杂链表。在复杂链表中，每个节点除了有一个 next 指针指向下一个节点，还有一个 random 指针指向链表中的任意节点或者 null。

### 思路

这是一个复杂链表的复制，一开始用map<int,node>存储，但是会有相同的int会重复，于是看了题解是map<node,node> 老节点和新节点的映射，然后老节点.next就是m.get(老节点)。Next 这样的

### 代码

/\*

// Definition for a Node.

class Node {

    int val;

    Node next;

    Node random;

    public Node(int val) {

        this.val = val;

        this.next = null;

        this.random = null;

    }

}

\*/

class Solution {

    public Node copyRandomList(Node head) {

        if(head==null) return null;

        Node q=head;

        HashMap<Node,Node> m=new HashMap<Node,Node>();

        while(q!=null){

            m.put(q,new Node(q.val));

            q=q.next;

        }

        q=head;

        while(q!=null){

            m.get(q).next=m.get(q.next);

            m.get(q).random=m.get(q.random);

            q=q.next;

        }

        return m.get(head);

    }

}

### 小结

很好的新思路