**查找出现次数最多的子串：**

let s='aba'

let a=[]

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

    for(let j=i+1;j<=s.length;j++){

        a.push(s.substring(i,j))

    }

}

console.log(a)

//[ 'a', 'ab', 'abc', 'b', 'bc', 'c' ]

字母的排序：

let str='bcad'

arr=str.split("")

arr.sort((a,b)=>**a.charCodeAt()-b.charCodeAt()**)

console.log(arr)

3. 无重复字符的最长子串

给定一个字符串 s ，请你找出其中不含有重复字符的 最长子串 的长度。

**利用队列的思想，如果arr中有相同的字符，while（）移除第一个元素shift()**

var lengthOfLongestSubstring = function(s) {

    let arr=[]

    let max=0

    for(let n of s){

        while(arr.indexOf(n)>=0){

            arr.shift()

        }

        arr.push(n)

        max=Math.max(max,arr.length)

    }

    return max

};

20. 有效的括号

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：

左括号必须用相同类型的右括号闭合。

左括号必须以正确的顺序闭合。

题解：**利用对象的键值对保存左右括号，键为左括号，值为右括号**

**遍历字符串，如果是左括号（k in map）就加入数组arr,如果不是左括号，就判断map[arr.pop()]的值（右括号）是否等于当前字符串**

var isValid = function(s) {

    let arr=[]

    let map={

        "{":"}",

        "[":"]",

        "(":")"

    }

    //保存左括号

    for(let k of s){

        //左括号

        if(k in map){

            arr.push(k)

            continue

        }

        //判断是否等于右括号

        if(map[arr.pop()]!==k){

            return false

        }

    }

    return !arr.length

};

71,简化路径

题解：以/划分字符串，如果字符==”..”,栈顶pop()，如果字符不为空且不等于“.”,则加入栈中，最后输出“/”+join(“/”)

var simplifyPath = function(path) {

    let arr=path.split("/")

    let s=[]

    for(let n of arr){

        if(n=='..'){

            if(s.length){

                s.pop()

            }

        }else if(n.length&&n!=='.'){

            s.push(n)

        }

    }

    return "/"+s.join("/")

};

1. 合并两个有序数组

**题解：双指针指向两个数组的尾部，遍历长数组，从后往前填，谁大填谁，如果p1或p2等于-1，直接填写另一个数组**

for(let i=nums1.length-1;i>=0;i--){

        if(p1===-1){

             nums1[i]=nums2[p2]

            p2--

        }else if(p2===-1){

            nums1[i]=nums1[p1]

            p1--

        }else if(nums1[p1]>nums2[p2]){

            nums1[i]=nums1[p1]

            p1--

        }else{

            nums1[i]=nums2[p2]

            p2--

        }

    }

#### [剑指 Offer 46. 把数字翻译成字符串](https://leetcode-cn.com/problems/ba-shu-zi-fan-yi-cheng-zi-fu-chuan-lcof/)

var translateNum = function(num) {

    const str = num.toString()

    const n = str.length

    const dp = new Array(n + 1)

    dp[0] = 1

    dp[1] = 1

    for(let i = 2; i < n + 1; i++) {

        const temp = Number(str[i-2] + str[i-1])

        if(temp >= 10 && temp <= 25) {

            dp[i] = dp[i-1] + dp[i-2]

        }else {

            dp[i] = dp[i-1]

        }

    }

    return dp[n]

};

1. 比较版本号

**题解：以“.”划分字符串，从左到右遍历两个数组，arr[i]存在则转为数值（parseInt）,不存在为0，判断两个数值的大小，移动指针i**

var compareVersion = function(version1, version2) {

    let arr1=version1.split(".")

    let len1=arr1.length

    let arr2=version2.split(".")

    let len2=arr2.length

    let i=0

    while(i<len1||i<len2){

        let num1=arr1[i] ? parseInt(arr1[i]):0

        let num2=arr2[i] ? parseInt(arr2[i]):0

        if(num1>num2){

            return 1

        }

        if(num1<num2){

            return -1

        }

        i++

    }

    return 0

};

1. 两数之和 （**map.has**）

**题解：值为键，下标为value，遍历数组，判断map中是否含有map.has（target-nums[i]）**

for(let i=0;i<nums.length;i++){

        if(map.has(target-nums[i])){

            return [map.get(target-nums[i]),i]

        }else{

            map.set(nums[i],i)

        }

    }

1. 三数之和

**题解：定义三个指针，注意，三个指针都要去重，所以数组需要先进行排序**

**如果当前和等于target,把当前三个数组成数组加入arr,并且需要while()去重左右指针**

if(sum===0){

                arr.push([nums[i],nums[left],nums[right]])

                while(left<right&&nums[left]==nums[left+1]){

                    left++

                }

                while(left<right&&nums[right]==nums[right-1]){

                    right--

                }

                left++

                right--

16. 最接近的三数之和

**题解：数组需要排序，但是指针不需要去重，在移动left和right过程中，记录最小的差值，把当前sum保存在result中，如果sum==target,则返回sum,否则最后返回result**

let sum=nums[i]+nums[left]+nums[right]

            let diff=Math.abs(sum-target)

            if(diff<min){

                min=diff

                result=sum

            }

1. 字符串相加

**题解：从后往前加，遍历两个数组，取最后一位相加的和，取个位数+total，如过和大于9，则add=1(否则add=0)**

var addStrings = function(num1, num2) {

    let len1=num1.length-1

    let len2=num2.length-1

    let result=''

    let add=0

    while(len1>=0||len2>=0||add){

        let sum=(num1[len1]\*1||0)+(num2[len2]\*1||0)+add

        result=sum%10+result

        add=sum>9?1:0

        len1--

        len2--

    }

    return result

};

#### [43. 字符串相乘](https://leetcode-cn.com/problems/multiply-strings/)

给定两个以字符串形式表示的非负整数 num1 和 num2，返回 num1 和 num2 的乘积，它们的乘积也表示为字符串形式。

**题解：从后往前遍历两个字符串，取最后一位字符相乘，结果加上arr[i+j+1]的值，取个位数存arr[i+j+1],arr[i+j]=math.floor(p/10)+arr[i+j]**

var multiply = function(num1, num2) {

    if(num1[0]=='0'||num2[0]=='0')return '0'

    let len1=num1.length

    let len2=num2.length

    let arr=new Array(len1+len2).fill(0)

    for(let i=len1-1;i>=0;i--){

        for(let j=len2-1;j>=0;j--){

            let p=num1[i]\*num2[j]

            p+=arr[i+j+1]    //加上原有的

            arr[i+j+1]=p%10   //存个位数

            arr[i+j]=(Math.floor(p/10))+arr[i+j]   //存十位数，加上原有的

        }

    }

    let i=0

    while(i<arr.length){

        if(arr[i]==0){

            arr.shift()

        }else{

            break

        }

    }

    return arr.join('')

};

1. 数组中的第K个最大元素

**题解：遍历数组，把元素保存至小堆，如果堆的长度大于k，则堆.pop(),最后返回堆顶元素**

**注意：父节点：（i-1）>>1**

**Insert() if(i==0)return**

**Pop():this.s[0]=this.s[s.length-1]**

var findKthLargest = function(nums, k) {

    class min1{

        constructor(){

            this.s=[]

        }

        swap(i,j){

            [this.s[i],this.s[j]]=[this.s[j],this.s[i]]

        }

        getleft(i){

            return i\*2+1

        }

        getright(i){

            return i\*2+2

        }

        getf(i){

            return (i-1)>>1

        }

        up(i){

            if(i==0) return

            let f=this.getf(i)

            if(this.s[i]<this.s[f]){

                this.swap(i,f)

                this.up(f)

            }

        }

        down(i){

            let left=this.getleft(i)

            let right=this.getright(i)

            if(this.s[left]<this.s[i]){

                this.swap(left,i)

                this.down(left)

            }

            if(this.s[right]<this.s[i]){

                this.swap(right,i)

                this.down(right)

            }

        }

        insert(value){

            this.s.push(value)

            this.up(this.s.length-1)

        }

        pop(){

            this.s[0]=this.s.pop()

            this.down(0)

        }

        top(){

            return this.s[0]

        }

        size(){

            return this.s.length

        }

    }

    const min=new min1()

    nums.forEach(item=>{

        min.insert(item)

        if(min.size()>k){

            min.pop()

        }

    })

    return min.top()

};

#### [46. 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。

**标记：use=true,回溯则use=false,如果use=true,continue,use放在递归中携带**

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

给定一个可包含重复数字的序列 nums ，**按任意顺序** 返回所有不重复的全排列。

**题解：use=true,回溯则use=false,如果use=true,continue,use放在递归中携带**

**需要排序，如果nums[i]==nums[i-1]且use【i-1】=false，continue**

if(i>0&&nums[i]==nums[i-1]&&use[i-1]==false){

                continue

            }

            if(use[i]){

                continue

            }

#### [141. 环形链表](https://leetcode-cn.com/problems/linked-list-cycle/) (判断是否有环)

**题解：注意循环条件while(k&&k.next) //k是保证当前不为空，k.next是保证下一个有值**

#### [142. 环形链表 II](https://leetcode-cn.com/problems/linked-list-cycle-ii/) （找出环的入口）

**题解：当指针相等（s==k）,重新建立两个指针，一个在相遇节点，一个在头节点，while(p1!==p2)next....**

while(p1!==p2){

                p1=p1.next

                p2=p2.next

            }

            return p1

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

**题解：遍历数组，遇到1，小岛数量count加1，并调用沉默函数sink(i,j)**

**Sink（）函数：如果i，j越界或者g[i][j]等于0，直接return,标记当前元素，递归处理周边元素**

var numIslands = function(grid) {

    let count=0

    const sink=function(row,col){

        //如果越界或者值为0，直接return

        if(row<0||row>=grid.length||col<0||col>=grid[0].length||grid[row][col]==="0"){

            return

        }

        //当前元素标记为“0”

        grid[row][col]="0"

        //递归处理其周围的元素

        sink(row,col+1)

        sink(row,col-1)

        sink(row+1,col)

        sink(row-1,col)

    }

    //遍历数组，遇到1，小岛数量加1，并调用沉默函数

    for(let i=0;i<grid.length;i++){

        for(let j=0;j<grid[0].length;j++){

            if(grid[i][j]==="1"){

                count++

                sink(i,j)

            }

        }

    }

    return count

};

#### [695. 岛屿的最大面积](https://leetcode-cn.com/problems/max-area-of-island/)

**题解：记录每个岛的数量**

var maxAreaOfIsland = function(grid) {

    let result=0

    const sink=function(i,j){

        if(i<0||i>=grid.length||j<0||j>=grid[0].length||grid[i][j]===0){

            return 0

        }

        grid[i][j]=0

        let count=1

        count+=sink(i,j+1)

        count+=sink(i,j-1)

        count+=sink(i+1,j)

        count+=sink(i-1,j)

        return count

    }

    for(let i=0;i<grid.length;i++){

        for(let j=0;j<grid[0].length;j++){

            if(grid[i][j]===1){

                let count=sink(i,j)

                result=Math.max(result,count)

            }

        }

    }

    return result

};

#### [54. 螺旋矩阵](https://leetcode-cn.com/problems/spiral-matrix/)

**题解：循环条件是不越界：第一个条件是《=** 第二个是**《**

while(left<=right&&top<=bottom)，if(left<right&&top<bottom)

#### [剑指 Offer 22. 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/)

# 单链表循环条件中while(p->next)和while(p)有什么区别

**如果是p,在尾节点还是true,但是执行p=p.next为空**

**如果是p.next,处理到最后一个节点p=p.next不为空,且不会再进入循环**

**题解：利用快慢指针，快指针先走（k-1）步**

var getKthFromEnd = function(head, k) {

    let p1=head

    let p2=head

    let num=k-1

    while(num--){

        p1=p1.next

    }

    while(p1.next){

        p1=p1.next

        p2=p2.next

    }

    return p2

};

#### [209. 长度最小的子数组](https://leetcode-cn.com/problems/minimum-size-subarray-sum/) （滑动窗口）

给定一个含有 n个正整数的数组和一个正整数 target **。**

找出该数组中满足其和≥ target的长度最小的 **连续子数组** [numsl, numsl+1, ..., numsr-1, numsr] ，并返回其长度**。**如果不存在符合条件的子数组，返回 0 。

**题解：用for循环数组，长度为i-start+1**

var minSubArrayLen = function(target, nums) {

    let res=Infinity

    let sum=0

    let start=0

    for(let i=0;i<nums.length;i++){

        sum+=nums[i]

        while(sum>=target){

            res=Math.min(res,i-start+1)

            sum-=nums[start]

            start++

        }

    }

    return res>nums.length?0:res

};

#### [155. 最小栈](https://leetcode-cn.com/problems/min-stack/)

**题解：用一个节点node保存当前加入的val,并保存当前栈中的最小值min,每次push操作的时候，判断栈是否为空，空则加入当前节点（val,val），不为空则需要把当前val与栈中的最后一个节点的min比较，取最小值**

function Node(val,min){

    this.val=val||null

    this.min=min||null

}

MinStack.prototype.push = function(val) {

    if(!this.s.length){

        let node=new Node(val,val)

        this.s.push(node)

    }else{

        let min=this.s[this.s.length-1].min

        let node=new Node(val,Math.min(val,min))

        this.s.push(node)

    }

};

#### [剑指 Offer 62. 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/)

var lastRemaining = function(n, m) {

    let index=0

**for(let i=2;i!=n+1;i++){**

**index=(index+m)%i**

**}**

    return index

};

#### [42. 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。

#### **题解：双指针循环条件（l<r）,获取左右边界的最大值，**

**如果左边界最大值<右边界最大值，则left++**

**累积雨水值等于左边最大值减去当前左指针的值**

**反之：right--**

var trap = function(height) {

    let len=height.length-1

    let left=0

    let right=len

    let maxl=0

    let maxr=0

    let result=0

    while(left<right){

        //获取最大的边界

        maxl=Math.max(maxl,height[left])

        maxr=Math.max(maxr,height[right])

        //左边小于右边

        if(maxl<maxr){

            result+=maxl-height[left]

            left++

        }else{

            result+=maxr-height[right]

            right--

        }

    }

    return result

};

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

**题解：get（）:判断是否在map中（has）,如果存在则取出删除再加入 （get,delete,set）**

**Put(): 如果存在map中（has）则删除，在添加set,如果长度（size）大 于capacity则删除第一个（map.keys().next().value）**

var LRUCache = function(capacity) {

    this.capacity=capacity

    this.map=new Map()

};

LRUCache.prototype.get = function(key) {

    if(this.map.has(key)){

        let temp=this.map.get(key)

**this.map.delete(key)**

**this.map.set(key,temp)**

        return temp

    }else{

        return -1

    }

};

LRUCache.prototype.put = function(key, value) {

    if(this.map.has(key)){

        this.map.delete(key)

    }

    this.map.set(key,value)

    if(this.map.size>this.capacity){

**this.map.delete(this.map.keys().next().value)**

    }

};

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

**题解：建立一个虚拟头节点，指针p指向该头节点，遍历两个链表，小的值则加入p.next**

var mergeTwoLists = function(list1, list2) {

    let node= new ListNode()

    let p=node

    let l1=list1

    let l2=list2

    while(l1&&l2){

        if(l1.val<l2.val){

            p.next=l1

            l1=l1.next

        }else{

            p.next=l2

            l2=l2.next

        }

        p=p.next

    }

**p.next=l1===null?l2:l1**

    return node.next

};

#### [93. 复原 IP 地址](https://leetcode-cn.com/problems/restore-ip-addresses/)

**题解：回溯算法：退出递归的条件（p.length==4&&index==len）或者p.length>4**

**剪枝的条件：截取的字符串（slice(index,i+1)）长度不能大于3，且值不能大于255，且第一个不能为0**(str.length>1&&str[0]=='0')

var restoreIpAddresses = function(s) {

    let arr=[]

    let p=[]

    let len=s.length

    const dd=function(index){

        if(**p.length>4**){

            return

        }

        if(**p.length==4&&index===len**){

            arr.push(p.join("."))

            return

        }

        for(let i=index;i<len;i++){

            let str=**s.slice(index,i+1)**

            if(**str.length>3||+str>255||(str.length>1&&str[0]==='0'))**{

**break**

            }

            p.push(str)

            dd(i+1)

            p.pop()

        }

    }

    dd(0)

    return arr

};

#### [31. 下一个排列](https://leetcode-cn.com/problems/next-permutation/)

**题解：找到降序的开始位置start，然后在start位置之后，需要排序，找到第一个比start大的值，与start交换**

var nextPermutation = function(nums) {

    //排序

    const reverse=function(nums,left,right){

        while(left<=right){

            [nums[left],nums[right]]=[nums[right],nums[left]]

            left++

            right--

        }

    }

    let len=nums.length

    for(let i=len-1;i>=0;i--){

        if(i==0){

            return nums.sort((a,b)=>a-b)

        }else{

            //找到降序

            if(nums[i]>nums[i-1]){

                reverse(nums,i,len-1)

                for(let k=i;k<len;k++){

                    if(nums[k]>nums[i-1]){

                        let temp=nums[i-1]

                        nums[i-1]=nums[k]

                        nums[k]=temp

                        return nums

                    }

                }

            }

        }

    }

};

#### [912. 排序数组](https://leetcode-cn.com/problems/sort-an-array/)

给你一个整数数组 nums，请你将该数组升序排列。

题解：快速排序：**选取基准：splice**

**let index=Math.floor(nums.length/2)**

**let base=nums.splice(index,1)[0]**

var sortArray = function(nums) {

    if(nums.length<2){

        return nums

    }

    let index=Math.floor(nums.length/2)

    let base=nums.splice(index,1)[0]

    let left=[]

    let right=[]

    for(let i=0;i<nums.length;i++){

        if(nums[i]>base){

            right.push(nums[i])

        }else{

            left.push(nums[i])

        }

    }

    return [...sortArray(left),base,...sortArray(right)]

};

var sortArray = function(nums) {

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

        // i表示已经排序好的个数

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

            if(nums[j] > nums[j+1]) {

                [nums[j], nums[j+1]] = [nums[j+1], nums[j]]

            }

        }

    }

    return nums

};

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

**题解：贪心算法，按起点排序，遍历数组，**

**如果当前起点，小于等于上一个终点，则end=max(end,nums[i][1])**

**如果当前起点，大于上一个终点，则更新start和end （arr.push([start,end])）**

**注意：！！return 前，执行一次arr.push([start,end])**

var merge = function(ins) {

    //按起点排序

    ins.sort((a,b)=>{

        return a[0]-b[0]

    })

    let len=ins.length

    let arr=[]

    let start=ins[0][0]

    let end=ins[0][1]

    for(let i=1;i<len;i++){

        if(ins[i][0]<=end){

            end=Math.max(end,ins[i][1])

        }else{

            arr.push([start,end])

            start=ins[i][0]

            end=ins[i][1]

        }

    }

**//把最后更新的start,end加入**

**arr.push([start,end])**

    return arr

};

#### [224. 基本计算器](https://leetcode-cn.com/problems/basic-calculator/)

给你一个字符串表达式 s ，请你实现一个基本计算器来计算并返回它的值。

**用栈保存符号，用op标识当前的符号位，**

**如果是+，则取sign栈顶符号，如果是-，栈顶符号取反，**

**如果遇到（，则把当前op加入栈中，**

**如果当前），则pop栈顶元素**

var calculate = function(s) {

    let sign=[1]

    let n=0

    let sum=0

    let op=1

    for(let i=0;i<s.length;i++){

        if(s[i]===' ')continue

        if(s[i]-'0'>=0&&s[i]-'0'<=9){

            n=n\*10+s[i]\*1

            continue

        }

        sum+=op\*n

        n=0

        if(s[i]==="+") op=sign[sign.length-1]

        if(s[i]==='-') op=-sign[sign.length-1]

        if(s[i]==="(") sign.push(op)

        if(s[i]===')') sign.pop()

    }

**sum+=op\*n**

    return sum

};

#### [227. 基本计算器 II](https://leetcode-cn.com/problems/basic-calculator-ii/)

给你一个字符串表达式 s ，请你实现一个基本计算器来计算并返回它的值。

var calculate = function(s) {

    let arr=[]

    let op="+"

    let num=0

    for(let i=0;i<s.length;i++){

        if(!isNaN(Number(s[i]))&&s[i]!==' '){

            num=num\*10+s[i]\*1

        }

        if(isNaN(Number(s[i]))||i===s.length-1){

            switch(op){

                case "+":arr.push(num); break

                case"-":arr.push(-num);break

                case"\*":arr.push(arr.pop()\*num);break

                case"/":arr.push(~~(arr.pop()/num));break

            }

            op=s[i]

            num=0

        }

    }

    return arr.reduce((a,b)=>a+b)

};

#### [160. 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。如果两个链表不存在相交节点，返回 null 。

var getIntersectionNode = function(headA, headB) {

    let p1=headA

    let len1=1

    let p2=headB

    let len2=1

    while(p1.next){

        p1=p1.next

        len1++

    }

    while(p2.next){

        p2=p2.next

        len2++

    }

    let diff=Math.abs(len1-len2)

    let s1

    let s2

    if(len1>len2){

        s1=headA

        s2=headB

    }else{

        s1=headB

        s2=headA

    }

    while(diff){

        s1=s1.next

        diff--

    }

**while(s1){**

        if(s1===s2){

            return s1

        }else{

            s1=s1.next

            s2=s2.next

        }

    }

    return null

};

#### [48. 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。

**题解：斜对角的元素交换[i][j]==[j][i],开始的起点为【i】【i】所以j初始化等于i**

**最后再每个数组元素，翻转一下**

var rotate = function(m) {

    let len=m.length

    for(let i=0;i<len;i++){

        for(let **j=i**;j<len;j++){

            let temp=m[i][j]

            m[i][j]=m[j][i]

            m[j][i]=temp

        }

    }

    return m.map(item=>item.reverse())

};

#### [14. 最长公共前缀](https://leetcode-cn.com/problems/longest-common-prefix/)

编写一个函数来查找字符串数组中的最长公共前缀。

如果不存在公共前缀，返回空字符串 ""。

**题解：数组排序，取第一个和最后一个对比，如果不相等，则break**

**截取字符串strs[0].substr(0,i)**

**charAt(i)根据下标找字符：**

var longestCommonPrefix = function (strs) {

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

    strs.sort();

    let len = strs[0].length;

    let i;

    for (i = 0; i < len; i++) {

        if(strs[strs.length - 1].**charAt(i)** !== strs[0].**charAt(i)**)break;

    }

    return strs[0].**substring(0, i)**;

};

substr（a,b） 第二个参数是截取[字符串](https://so.csdn.net/so/search?q=%E5%AD%97%E7%AC%A6%E4%B8%B2&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/maxiaotengzhang/article/details/_blank)的长度

substring（a,b） 第二个参数是截取字符串最终的下标

#### [349. 两个数组的交集](https://leetcode-cn.com/problems/intersection-of-two-arrays/)

var intersection = function(nums1, nums2) {

    let arr=nums1.filter(item=>{

        return nums2.indexOf(item)>-1

    })

    return [...new Set(arr)]

};

#### [剑指 Offer 10- I. 斐波那契数列](https://leetcode-cn.com/problems/fei-bo-na-qi-shu-lie-lcof/)

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

var fib = function(n) {

    //初始化

    let dp=[0,1]

    //遍历

    for(let i=2;i<=n;i++){

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

    }

    return dp[n]

};

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

题解：集合中有重复的元素（排序去重），每个数字只能使用一次（i+1）

var combinationSum2 = function(ca, target) {

    let arr=[]

    let p=[]

    ca.sort()

    const dd=function(index,sum){

        if(sum>target)return

        if(sum==target){

            arr.push([...p])

            return

        }

        for(let i=index;i<ca.length;i++){

            if(i>index&&ca[i]==ca[i-1]){

                continue

            }

            sum+=ca[i]

            p.push(ca[i])

            dd(i+1,sum)

            p.pop()

            sum-=ca[i]

        }

    }

    dd(0,0)

    return arr

};

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

题解：集合无重复元素（不需要去重），可以无限选择（i）

var combinationSum = function(ca, target) {

    let arr=[]

    let p=[]

    const dd=function(index,sum){

        if(sum>target)return

        if(sum==target){

            arr.push([...p])

            return

        }

        for(let i=index;i<ca.length;i++){

            sum+=ca[i]

            p.push(ca[i])

            dd(i,sum)

            p.pop()

            sum-=ca[i]

        }

    }

    dd(0,0)

    return arr

};

#### [25. K 个一组翻转链表](https://leetcode-cn.com/problems/reverse-nodes-in-k-group/)

题解：设置虚拟头节点，找到end,start,保存下一轮的起点next(**start.next=next**)

var reverseKGroup = function(head, k) {

    if(head==null)return head

    //虚拟头节点

    let d=new ListNode(0,head)

    let pre=d,end=d

    while(end.next){

        let i=0

        //找到第一段的结尾

        while(i<k&&end){

            i++

            end=end.next

        }

        //如果不够k个，直接退出

        if(end==null) break

        //保存下一段的开头

        let next=end.next

        end.next=null   //置空

        let start=pre.next

        pre.next=reverse(start)  //反转start到end

        //重置起点

        start.next=next   //start为空节点，最后start=pre.next

        end=start

        pre=end

    }

    return d.next

};

//返回新链表的头节点

var reverse=function(head){

    let cur=head

    let pre=null

    while(cur){

        let temp=cur.next

        cur.next=pre

        //更新

        pre=cur

        cur=temp

    }

    return pre

}

#### [468. 验证IP地址](https://leetcode-cn.com/problems/validate-ip-address/)

**题解：**

**ipv4:以’.’分割，长度为4，如果arr[i]==‘’（错），转化为数字（parseInt(arr[i],10)）,再转化为字符串，如果长度改变了（错），且值小于255，大于0**

**Ipv6 :长度为8，每个arr[i]长度1到4，match/[a-fA-F0-9]/g的长度如果不等于原长度（错）match的数组必须存在**

var validIPAddress = function(queryIP) {

    // 先以ipv4分割

    let arr = queryIP.split(".");

    let flag=true

    // 不是ipv4

    if(arr.length == 1){

        arr = queryIP.split(":");

       flag=false

    }

    for(let i = 0; i < arr.length; i++) {

        if(flag) {

            if(arr.length !== 4) return "Neither";

            // 验证ipv4

            if(arr[i] == "") return "Neither";

            let t = parseInt(arr[i], 10);

            if(t.toString().length !== arr[i].length || t < 0 || t > 255)return "Neither";

        } else {

            if(arr.length !== 8) return "Neither";

            // 验证ipv6

            if(arr[i].length<1 || arr[i].length > 4)return "Neither";

            let t = arr[i].match(/[a-fA-F0-9]/g);

            if(!t || t.length !== arr[i].length)return "Neither";

        }

    }

    return flag?'IPv4':'IPv6'

};

#### [443. 压缩字符串](https://leetcode-cn.com/problems/string-compression/)

join() 方法用于把数组中的所有元素放入一个字符串

**题解：转化为字符转(join)，再截取连续相同**match(/(.)\1\*/g) || []

用reduce遍历，如果pre+=item.length>1 (如果长度大于1)

Pre=item[0]+item.length 否则为item

转化为数组：split(“”) 以空格划分

修改原数组：arr.splice(0,arr.length,...res) 从0开始删除len个，再添加res

var compress = function(chars) {

    //转化为字符串

    let str=chars.join('')

    //获取连续相同的

    let arr=str.match(/(.)\1\*/g)

    //处理次数

    let res=arr.reduce((p,item)=>{

        p+=item.length>1?item[0]+item.length:item

        return p

    },'').split('')

    chars.splice(0,chars.length,...res)

};

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

**输入：**s = "3[a]2[bc]"**输出：**"aaabcbc"

var decodeString = function(s) {

    const reg = /(\d+)\[([a-z]+)\]/g

    while (s.includes('[')) {

        s = s.replace(reg, ($, $1, $2) => $2.repeat($1))

    }

    return s

};

#### [41. 缺失的第一个正数](https://leetcode-cn.com/problems/first-missing-positive/)

**题解:n放在第n位**

**对原数组进行遍历，**

 //nums[i]是否在nums[i]-1的位置上

**While(Nums[i]>=1 &&nums[i]<=length&&nums[i]!==nums[nums[i]-1])**

**Swap(nums,i,nums[i[]-1)**

**遍历依旧排序好的数组，如果第i个元素的值，不是i nums[i] !=i+1 return i+1**

var firstMissingPositive = function(nums) {

    for(let i=0;i<nums.length;i++){

       while(nums[i]>=1&&nums[i]<=nums.length&&**nums[i]!=nums[nums[i]-1]**){

**swap(nums,i,nums[i]-1)**

        }

    }

    for(let i=0;i<nums.length;i++){

**if(nums[i]!=i+1){**

**return i+1**

        }

    }

    return nums.length+1

};

var swap=function(nums,i,j){

    [nums[i],nums[j]]=[nums[j],nums[i]]

}

#### [171. Excel 表列序号](https://leetcode-cn.com/problems/excel-sheet-column-number/)

**题解：获取与A的偏移量charCodeAt(),从后往前遍历，值累加在number上，每次乘上进制，进制每次调用\*26**

var titleToNumber = function(columnTitle) {

    let number = 0;

    let multiple = 1;

    for (let i = columnTitle.length - 1; i >= 0; i--) {

        //编码减去A的编码+1，等于在A之上的偏移量

        const k = columnTitle[i].charCodeAt() - 'A'.charCodeAt() + 1;

        number += k \* multiple;  //多位数的结果

        multiple \*= 26;  //下一位等于原值\*26

    }

    return number;

};

#### [498. 对角线遍历](https://leetcode-cn.com/problems/diagonal-traverse/)

**题解：先计算所有对角线，再通过反转，最后扁平化输出**

var findDiagonalOrder = function(mat) {

    let arr=[]

    let row=mat.length

    let col=mat[0].length

    for(let k=0;k<col;k++){

        let p=[]

        for(let i=0,j=k;i<row&&j>=0;i++,j--){

            p.push(mat[i][j])

        }

        arr.push([...p])

    }

    for(let k=1;k<row;k++){

        let p=[]

        for(let i=k,j=col-1;i<row&&j>=0;i++,j--){

            p.push(mat[i][j])

        }

        arr.push([...p])

    }

    for(let i=1;i<arr.length;i++){

        if(i%2==0){

            arr[i].reverse()

        }

    }

    return arr.flat(2)

};

**动态规划：**

1. 最大子数组和

**题解：贪心算法，局部最优=>全局最优，标记最大值，如果加入当前值小于0，直接更新sum=0**

 for(let i=0;i<len;i++){

       sum+=nums[i]

       result=Math.max(result,sum)

       if(sum<0){

           sum=0

       }

   }

**动态规划：**

let dp=new Array(len).fill(0)

    dp[0]=nums[0]

    let result=dp[0]

    for(let i=1;i<len;i++){

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

        result=Math.max(result,dp[i])

    }

1. 买卖股票的最佳时机（只买卖一次）

**题解：定义两个状态**

**买0：今天买入-p[i],之前买入dp[i-1][0]，**

**卖1: 今天卖出dp[i-1][0]+p[i],之前卖出：dp[i-1][1]**

1. 买卖股票的最佳时机II ：（可以多次买卖）

**题解：获取每天的正利润**

**题解：买0：今天买入：dp[i-1][1]-p[i], 之前买入：dp[i-1][0]**

**卖1：今天卖出：dp[i-1][0]+p[i], 之前卖出：dp[i-1][1]**

123,买卖股票的最佳时机III ：（最多可以完成两笔交易）

**题解：定义5个状态**

**0不操作：dp[i-1][0]**

**1买入：-p[i]，dp[i-1][1]**

**2卖出：dp[i-1][1]+p[i] , dp[i-1][2]**

**3买入：dp[i-1][2]-p[i], dp[i-1][3]**

**4卖出：dp[i-1][3]+p[i], dp[i-1][4]**

188,买卖股票的最佳时机IV ：（最多可以完成 k 笔交易）

**题解：定义2个状态**

**（奇数）买入：dp[i-1][j]-p[i], dp[i-1][j+1]**

**（偶数）卖出：dp[i-1][j+1]+p[i], dp[i-1][j+2]**

初始化：

 let dp=Array(len).fill(0).map(()=>Array(2\*k+1).fill(0))

    for(let i=1;i<2\*k;i+=2){

        dp[0][i]=-p[0]

    }

遍历：

 for(let j=0;j<2\*k;j+=2)

1. 买卖股票的最佳时机含手续费,（可以买卖多次）

**题解：定义两个状态,卖出的时候计算手续费**

**0买入：dp[i-1][1]-p[i] , dp[i-1][0]**

**1卖出：dp[i-1][0]+p[i]-fee , dp[i-1][1]**

#### [5. 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。

**题解：遍历数组，可能存在当前元素是奇数（i,i）或偶数（i，i+1）的情况**

**函数dd:while（）左右指针不越界，且值相等，则指针移动**

**保存长度更大的字符串s.substr(left+1,right-left-1)**

var longestPalindrome = function(s) {

    let res=''

    const dd=function(left,right){

        while(left>=0&&right<s.length&&s[left]===s[right]){

            left--

            right++

        }

        if(res.length<right-left-1){

            res=s.substr(left+1,right-left-1)

        }

    }

    for(let i=0;i<s.length;i++){

        //奇数

        dd(i,i)

        //偶数

        dd(i,i+1)

    }

    return res

};

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

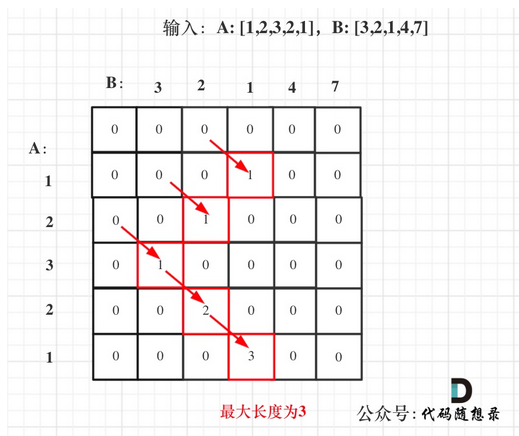
假设你正在爬楼梯。需要 n 阶你才能到达楼顶。

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？

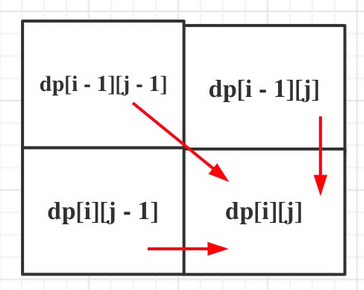
**题解：dp[i]=dp[i-1]+dp[i-2]**

#### [718. 最长重复子数组](https://leetcode-cn.com/problems/maximum-length-of-repeated-subarray/)

**题解：两个数组，连续相同，斜角数+1 dp[i][j]=dp[i-1][j-1]+1**



# 1143. 最长公共子序列（JavaScript）



var longestCommonSubsequence = function(t1, t2) {

    let len1=t1.length

    let len2=t2.length

    let dp=Array(len1+1).fill(0).map(()=>Array(len2+1).fill(0))

   for(let i=1;i<=len1;i++){

       for(let j=1;j<=len2;j++){

           if(t1[i-1]==t2[j-1]){

               dp[i][j]=dp[i-1][j-1]+1

           }else{

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

           }

       }

   }

   return dp[len1][len2]

};

#### [1312. 让字符串成为回文串的最少插入次数](https://leetcode-cn.com/problems/minimum-insertion-steps-to-make-a-string-palindrome/)

给你一个字符串 s ，每一次操作你都可以在字符串的任意位置插入任意字符。

请你返回让 s 成为回文串的 **最少操作次数** 。

「回文串」是正读和反读都相同的字符串。

**题解：dp表赋值为0，当相等的时候不需要操作dp[i][j]等于斜下角的值（[i+1][j-1]）**

**如果不相等，dp[i][j]等于同行或同列最小值加1**

**遍历顺序：行：倒数第二行开始，i=len-2**

**列：j=i+1**

var minInsertions = function(s) {

    let len=s.length

    let dp=new Array(len).fill(0).map(()=>Array(len).fill(0))

    for(let i=len-2;i>=0;i--){//行

        for(let j=i+1;j<len;j++){

            if(s[i]==s[j]){

                dp[i][j]=dp[i+1][j-1]

            }else{

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

            }

        }

    }

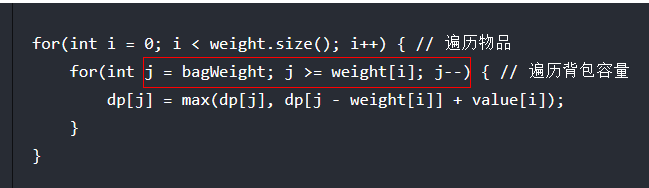
    return dp[0][len-1]

};

**01背包：**

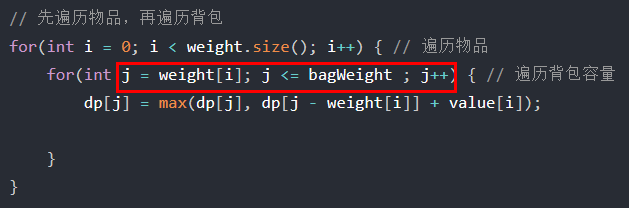
**一个物品只有取或不取两种可能，先遍历物品，再遍历背包容量**

01背包内嵌的循环是从大到小遍历，为了保证每个物品仅被添加一次。



#### 完全背包，可以取无数次：

完全背包的物品是可以添加多次的，所以要从小到大去遍历：



**完全背包强调的顺序问题：**

1. 先遍历物品，则每个元素只出现一次，不会出现重复的情况
2. 先遍历背包容量，则会出现{1, 5} 和 {5, 1}两种情况。

**当如果**顺序不同的序列被视作不同的组合，**需要先遍历背包容量**

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

**完全背包问题，先遍历物品，再遍历背包，起始值为con[i],小于等于最大容量amount,**

**凑满j的金额，至少需要多少个**

var coinChange = function(coins, amount) {

    let dp=new Array(amount+1).fill(Infinity)

    dp[0]=0

    for(let i=0;i<coins.length;i++){

        for(let j=coins[i];j<=amount;j++){

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

        }

    }

return dp[amount]==Infinity?-1:dp[amount]};

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

**题解：当前节点最长的子序列长度，是和i之前的元素进行比较，如果num[i]>num[j],**

**则dp[i]=max(dp[i],dp[j]+1)**

var lengthOfLIS = function(nums) {

    let max=1

    let len=nums.length

    let dp=new Array(len+1).fill(1)

    for(let i=1;i<len;i++){

        for(let j=0;j<i;j++){   //对比i之前的数字，比i小则dp[j]+1

            if(nums[i]>nums[j]){

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

            }

        }

        max=Math.max(dp[i],max)

    }

    return max

};

**二叉树**

#### [236. 二叉树的最近公共祖先](https://leetcode-cn.com/problems/lowest-common-ancestor-of-a-binary-tree/)

**题解：后序递归，如果当前节点为null,或者是目标值，则返回当前节点，获得左右子节点的状态，如果都为true，则返回当前节点，不然就返回left||right**

var lowestCommonAncestor = function(root, p, q) {

    let left=lowestCommonAncestor(root.left,p,q)

    let right=lowestCommonAncestor(root.right,p,q)

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

        return root

    }

    if(**left&&right**){

        return root

    }

**return left||right**

};

112. 路径总和

给你二叉树的根节点 root 和一个表示目标和的整数 targetSum 。判断该树中是否存在 根节点到叶子节点 的路径，这条路径上所有节点值相加等于目标和 targetSum 。如果存在，返回 true ；否则，返回 false 。

叶子节点 是指没有子节点的节点。

题解：**如果是叶子节点且是target,返回true,当前节点是否为true，取决于左右子节点的返回值，只有有一个是true，则返回true**

var hasPathSum = function(root, targetSum) {

    if(root==null){

        return false

    }

    const dd=function(root,sum){

        sum+=root.val

        //叶子节点和为target

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

            if(sum==targetSum){

                return true

            }else{

                return false

            }

        }

        if(root.left&&dd(root.left,sum)||

           root.right&&dd(root.right,sum) ){

               return true

           }

        return false

    }

    return dd(root,0)

};

113. 路径总和ii :给定一个二叉树和一个目标和，找到所有从根节点到叶子节点路径总和等于给定目标和的路径。

**题解：遇到叶子节点且为target,加入路径，处理其左右子节点需要单独处理回溯**

if(root.left){

            p.push(root.left.val)

            dd(root.left,p,sum)

            p.pop()

        }

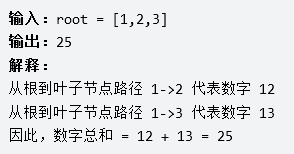
257. 二叉树的所有路径

给你一个二叉树的根节点 root ，按 任意顺序 ，返回所有从根节点到叶子节点的路径。

**输入：**root = [1,2,3,null,5]**输出：**["1->2->5","1->3"]

**题解：如果是叶子节点，p+=root.val,如果不是叶子节点：p+=root.val+”->”**

129. 求根节点到叶节点数字之和



**题解：获得所有从根节点到叶子节点的路径，转化为字符串保存在arr中，最后把arr中的字符串转化为数字相加**

var sumNumbers = function(root) {

    let arr=[]

    let p=[root.val]

    const dd=function(root,p){

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

            arr.push(p.join(""))

            return

        }

        if(root.left){

            p.push(root.left.val)

            dd(root.left,p)

            p.pop()

        }

        if(root.right){

            p.push(root.right.val)

            dd(root.right,p)

            p.pop()

        }

    }

    dd(root,p)

    return arr.reduce((a,b)=>(+a)+(+b))

};

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

给你二叉树的根节点 root ，返回其节点值的 **层序遍历** 。 （即逐层地，从左到右访问所有节点）。

**题解：利用队列：移除一个头节点，就要把其子节点加入队列**

**在while()中初始化p=[],用于保存当前层次的值**

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

**题解：统一迭代：取出当前节点pop()，判断其是不是为null,如果是，则需要处理节点**

while(s.length){

        let node=s.pop()

        if(node===null){

            arr.push(**s.pop().val**)

        }else{

            node.right&&s.push(node.right)

**s.push(node)**

**s.push(null)**

            node.left&&s.push(node.left)

        }

    }

#### [429. N 叉树的层序遍历](https://leetcode-cn.com/problems/n-ary-tree-level-order-traversal/)

**题解：遍历其子节点：node.children.length**

for(let n=0;n<node.children.length;n++){

                if(node.children[n]){

                    qq.push(node.children[n])

                }

            }

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

给你一棵二叉树的根节点 root ，翻转这棵二叉树，并返回其根节点。

**题解：采用层序遍历,交换两个节点，交换结束需要重新赋值给根节点**

var invertTree = function(root) {

    if(root==null){

        return root

    }

    const dd=function(root,left,right){

        let temp=left

        left=right

        right=temp

        root.left=left

        root.right=right

    }

    let s=[root]

    while(s.length){

        let len=s.length

        for(let i=0;i<len;i++){

            let node=s.shift()

            //处理

            dd(node,node.left,node.right)

            node.left&&s.push(node.left)

            node .right&&s.push(node.right)

        }

    }

    return root

};

#### [230. 二叉搜索树中第K小的元素](https://leetcode-cn.com/problems/kth-smallest-element-in-a-bst/)

给定一个二叉搜索树的根节点 root ，和一个整数 k ，请你设计一个算法查找其中第 k个最小元素（**从 1 开始计数**）。

**题解：中序遍历二叉搜索树，加入数组，返回arr[k-1]**

#### [剑指 Offer 54. 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 大的节点的值。

**题解：中序遍历二叉搜索树，加入数组，返回arr[arr.length-k]**

#### [103. 二叉树的锯齿形层序遍历](https://leetcode-cn.com/problems/binary-tree-zigzag-level-order-traversal/)

给你二叉树的根节点 root ，返回其节点值的 **锯齿形层序遍历** 。（即先从左往右，再从右往左进行下一层遍历，以此类推，层与层之间交替进行）。

**题解：层序遍历，用flag标识添加到path时的顺序，再处理结束当前层次，flag取反**

**flag=true, p.push() flag=false, p.unshift()**

let node=s.shift()

            if(flag){

                 p.push(node.val)

            }else{

                 p.unshift(node.val)

            }

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

**题解：递归或者层序遍历**

var maxDepth = function(root) {

    if(root==null){

        return 0

    }

    return 1+Math.max(maxDepth(root.left),maxDepth(root.right))

};

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

给你一个二叉树的根节点 root ， 检查它是否轴对称。

**题解：迭代法：建立一个队列，把左右节点加入队列，每次取出（shift）两个进行比较,如果两个都为空，continue，如果其中一个为空，或者值不相等，返回false**

var isSymmetric = function(root) {

   let s=[]

   s.push(root.left)

   s.push(root.right)

   while(s.length){

       let left=s.shift()

       let right=s.shift()

       if(left==null&&right==null){

           continue

       }

       if(left==null||right==null||left.val!=right.val){

           return false

       }

       s.push(left.left)

       s.push(right.right)

       s.push(left.right)

       s.push(right.left)

   }

   return true

}

#### [199. 二叉树的右视图](https://leetcode-cn.com/problems/binary-tree-right-side-view/)

**题解：采用层序遍历，在添加子节点的时候，每次记录最后一个子节点**

#### [105. 从前序与中序遍历序列构造二叉树](https://leetcode-cn.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/)

**题解：根据前序的根节点（**shift**），创建节点，在中序中查找下标**indexOf

**（以下标划分），然后再对齐左右子树建树**

**跳出递归：如果前序数组为空，返回null**

var buildTree = function(preorder, inorder) {

    if(!preorder.length){

        return null

    }

    let md=preorder.shift()

    let index=inorder.indexOf(md)

    const root=new TreeNode(md)

    root.left=buildTree(preorder.slice(0,index),inorder.slice(0,index))

    root.right=buildTree(preorder.slice(index),inorder.slice(index+1))

    return root

};