201、求 m 到 n 的位与

等价于 求 m 与 n 二进制编码中 同为 1 的前缀.

```
int rangeBitwiseAnd(int m, int n)
if m==0 return 0
moveFactor=0
while m≠n
    m=m>>1
    n=n>>1
    moveFactor++
return m<<moveFactor//此时 m 为前缀部分，将其还原到其原有的位置上
```

```
int rangeBitwiseAnd(int m, int n)
r=Integer.MAX_VALUE(111111...111)//32 个 1
while m&r≠n&r
    r=r<<1
return n&r
```

202、Happy Number

boolean isHappy(int n)

```
let set be a new Set<Integer>
while set.add(n)
    if n==1 return true
    n=Aux(n)
return false
```

int Aux(int n)

```
res=0
while n≠0
    remain=n%10
    res=res+remain*remain
    n=n/10
return res
```

203、删除链表中指定值的元素

ListNode removeElements(ListNode head, int val)

```
if head==null return null
let pseudohead be a new List with any value
pseudohead.next=head
pre=pseudohead,cur=head
while cur≠null
    if cur.val==val pre.next=cur.next
    else pre=cur
    cur=cur.next
return pseudohead.next
```

204、求所有小于 n 的素数

利用 O(n)的额外空间来提升运算效率，从 O(n^2)->O(n)

int countPrimes(int n)

```
let notPrime[1...n] be a new Array stored boolean initialized to false
count=0
for i=2 to n-1
    if notPrime[i]==false
        count++
        for j=2 to $\lfloor n/i \rfloor$//i*j<n
            notPrime[i*j]=true
return count
```

205、判断两个字符串是否同构

"add" 与 "egg"同构

```
boolean isIsomorphic(String s, String t)
if s==null or t==null throw Exception
if s.length≠t.length return false
let m1[1...128] m2[1...128] be two new Array//128 代表 128 个 ASCII
for i=1 to s.length
    if m1[s[i]] ≠m2[t[i]] return false
    m1[s[i]]=i//代码中要写成 i+1，以区分数组初始化原始的 0
    m2[t[i]]=i
return true
```

注意点，m1[i]表示字符 i 出现的最后一次的位置

206、反转链表

public ListNode reverseList(ListNode head)

```
let pseudohead be a new ListNode with arbitrary value
cur=head
while cur≠null
    tem=cur.next
    cur.next=pseudohead.next
    pseudohead.next=cur
    cur=tem
return pseudohead.next
```

207、能否完成课程（即判断有向图是否存在环结构）

DFS 法：

```
boolean canFinish(int numCourses, int[][] prerequisites)
1 let Graph[1...numCourses] be a new Array stored Adjacent List initialized to empty List
2 for each edge of prerequisites
3         Graph[edge[2]].add(edge[1])//from course edge[2] point to course edge[1]
4 let visiting[1...numCourses] visited[1...numCourses] be Arrays stored boolean initialized to false
5 for i=1 to numCourses
6         if not visited[i]
7                 if not DFS(Graph,visiting,visited,i) return false
8 return true
```

```
boolean DFS(ArrayList<Integer>[] Graph,boolean[] visiting,boolean[] visited,int dex)
1 if visiting[dex] return false
2 curNodeAdj=Graph[dex]
3 visiting[dex]=true
4 for each i of curNodeAdj
5         if not visited[i]
6                 if not DFS(Graph,visiting,visited,i) return false
7 visiting[dex]=false
8 visited[dex]=true//必须放在最后，否则 Line5 可能有问题
9 return true
```

visited 代表 DFS 中代表颜色的参数，true 代表已经搜索过

visiting 代表当前搜索中已经经过的节点，需要在 DFS 遍历 Adj 后重置为 false

BSF 法：

```
boolean canFinish(int numCourses, int[][] prerequisites)
let Graph[1...numCourses] be a new Array stored Adjacent List initialized to empty List
let Degree[1...numCourses] be a new Array stored int initialized to zero
//Build Graph G(V,E)
for each edge of prerequisites
    Degree[edge[1]]++
    Graph[edge[2]].add(edge[1])
//Begin BFS from those nodes with zero Degree
let queue be a new Queue
visitcnt=1
for i=1 to numCourses
    if Degree[i]==0
        queue.add(i)
        visitcnt ++
while not queue.isEmpty()
    curnode=queue.poll()
    curAdj=Graph[curnode]
    for each edgeEnd:Adj
        Degree[edgeEnd]--
        if Degree[edgeEnd]==0
            queue.offer(edgeEnd)
            visitcnt++
return visitcnt==numCouses+1?true:false
```

Degree[i]存储的是第 i 个课程的先修课程的个数，即以该节点为

208、单词查找树

```
class TrieNode
    public TrieNode[] Children
    boolean IsWord=false
    char val
    public TrieNode()
    this.val='\0'
    let Children[1...26] be a new Array stored TrieNode
    TrieNode(char val)
    this.val=val
    let Children[1...26] be a new Array stored TrieNode
public class Trie
    private TrieNode root
    public Trie()
    let root be a new TrieNode()
    public void insert(String word)
    curNode=root
    for i=1 to word.length
        curChar=word[i]
        if curNode.Children[curChar-'a'+1]==null
            let curNode.Children[curChar-'a'+1] be a new TrieNode(curChar)
        curNode=curNode.Children[curChar-'a'+1]
    curNode.IsWord=true
    public boolean search(String word)
    curNode=root
    for i=1 to word.length
        curChar=word[i]
        if curNode.Children[curChar-'a'+1]==null return false
        curNode=curNode.Children[curChar-'a'+1]
    return curNode.IsWord
    public boolean startsWith(String prefix)
    curNode=root
    for i=1 to word.length
        curChar=word[i]
        if curNode.Children[curChar-'a']==null return false
        curNode=curNode.Children[curChar-'a']
    return AnyWord(curNode)
    private boolean AnyWord(TrieNode cur)
    if cur.IsWord return true
    for each trienode:cur.Children)
        if trienode ==null continue
        if AnyWord(trienode) return true
    return false
```

209、求长度最小的子数组，满足子数组的和不小于指定值

public int minSubArrayLen(int s, int[] nums)

```
if nums==null or nums.length==0 return 0
begin=1,end=1,sum=0 minimum=+∞
while end≤nums.length
    while end≤nums.length and sum<s
        sum=sum+nums[end++]
    while begin<end and sum≥s
        minimum=min(minimum,(end-1)-begin+1)
        if minimum==1 return 1
        sum=sum-nums[begin++]
return minimum==+∞?0:minimum
```

任意时刻,sum 为子数组 nums[begin...end-1]的和

210、有向无环图的 Toplogical sort

BFS 算法：

```
public int[] findOrder(int numCourses, int[][] prerequisites)
let Graph[1...numCourses] be a new Array stored Adjacent List initialized to empty List
let Degree[1...numCourses] be a new Array stored int initialized to zero
//Build Graph G(V,E)
for each edge of prerequisites
    Degree[edge[1]]++
    Graph[edge[2]].add(edge[1])
//Begin BFS from those nodes with zero Degree
let Res[1...numCourses] be a new Array
let queue be a new Queue
visitcnt=1
for i=1 to numCourses
    if Degree[i]==0
        queue.offer(i)
        Res[visitcnt++]=i
while not queue.isEmtpy()
    curnode=queue.poll()
    curAdj=Graph[curnode]
    for each edgeEnd:Adj
        Degree[edgeEnd]--
        if Degree[edgeEnd]==0
            queue.offer(edgeEnd)
            Res[visitcnt++]=edgeEnd
return visitcnt==numCouses+1?Res:null
```

DFS1:

```
class Vertex{
        static int time=0
        int end=0
        int dex
        boolean visited=false
        boolean visiting=false
        public Vertex(int dex){this.dex=dex}
}
int[] findOrder(int numCourses, int[][] prerequisites)
1 let Vertexs[1...numCourses] be a new Array stored Vertex
2 for i=1 to numCourses
3         Vertexs[i]=new Vertex(i)
4 let Graph[1...numCourses] be a new Array stored Ajacent List initialized to empty List
5 for each edge: prerequisites
6         Graph[edge[2]].add(Vertexs[edge[1]])
7 for i=1 to numCourses
8         if not Vertexs[i].visited
9                 if not DFS(Graph,Vertexs,i) return new int[0]
10 Descend Sort Vertexs by end
12 let Res[1...numCourses] be a new Array
13 cnt=0
14 for each v:Vertexs
15         Res[cnt++]=v.dex
16 return Res

boolean DFS(ArrayList<Vertex>[] Graph,Vertex[] Courses,int dex)
1 if Vertexs[dex].visiting return false
2 Vertexs[dex].visiting=true
3 curAdj=Graph[dex]
4 for Vertex v:curAdj
5         if not v.visited
6                 if not DFS(Graph,Vertexs,v.dex)
7                         return false
8 Vertexs[dex].visiting=false
9 Vertexs[dex].visited=true//必须放在最后，否则 Line 可能有问题
10 Vertex.time++
11 Vertexs[dex].end=Vertex.time
12 return true
```

211、单词查找树 2

```
public class WordDictionary {
    private class TrieNode{
        TrieNode[] children;
        boolean isWord;
        TrieNode()
        children=new TrieNode[26];
        isWord=false;
    }
    private TrieNode root;
    WordDictionary()
    root=new TrieNode();

    // Adds a word into the data structure.
    public void addWord(String word)
    TrieNode iter=root;
    for each c of word
        index=c-'a';
        if iter.children[index]==null
            iter.children[index]=new TrieNode();
        iter=iter.children[index];
    iter.isWord=true;

    // Returns if the word is in the data structure. A word could
    // contain the dot character '.' to represent any one letter.
    public boolean search(String word)
    return searchHelper(word,root);

    private boolean searchHelper(String subWord,TrieNode root)
    if subWord.equals("") return root.isWord;
    first=subWord[1]
    if first=='.'
        for int i=1 to 26
            if root.children[i]≠null
                if searchHelper(substring[2...end],root.children[i])    return true;
        return false;
    else
        index=first-'a';
        if root.children[index]≠null
             return searchHelper(substring[2...end],root.children[index]);
        else
            return false;
```

212、

213、小偷偷房子（房子首尾相接）

拆分成两个部分[1...n-1] 或 [2...n]

```
public int rob(int[] nums)
if nums.length==1 return nums[0]
return max(Aux(nums,1,nums.length-1),Aux(nums,2,nums.length))
```

```
private int Aux(int[] num, int begin, int end)
preInclude=0,preExclude=0
for i=begin to end
    tem1=preInclude,tem2=preExclude
    preInclude=tem2+num[i]
    preExclude=max(tem1,tem2)
return max(preInclude,preExclude)
```

DP 算法见 198

214、补充一个字符串，使其为回文序列，且长度最短

KMP 算法：十分巧妙

sNew:aabbcde#edcbbaa

π　:010000000000012

KMP 算法可以求出一个字符串的从字符串起始位置开始的最长回文序列

```
String shortestPalindrome(String s)
sNew=s+'#'+s.reverse()
let π[1...sNew.length] be a new Array
π[1]=0//首字符定义为 0（longgest pre-postfix 不包括自身）
k= π[1]
for q=2 to sNew.length
    while k>0 and sNew[q]≠sNew[k+1]
        k= π[k]
    if sNew[q]==sNew[k+1]
        k=k+1
    π[q]=k
return s[π[sNew.length]+1...s.length].reverse+s
```

Recursive Solution:

215、线性时间选择算法

int findKthLargest(int[] nums, int k)

```
return Select(nums,1,nums.length,nums.length-k+1)
```

int Select(int[] nums,int p,int r,int k)

```
if p==r return nums[p]
q=Partition(nums,p,r)
n=q-p+1
if n==k return nums[q]
if k<n
    return Select(nums,p,q-1,k)
else
    return Select(nums,q+1,r,k-n)
```

216、指定长度子集和问题

```
public List<List<Integer>> combinationSum3(int k, int target)
let Res be a new List<List<Integer>>
let Pre be a new List<Integer>
nums={1,2,3,4,5,6,7,8,9}
Aux(nums,k,1,target,Res,Pre)
return Res
```

```
void Aux(int[] nums,int k,int dex,int target,List<List<Integer>> Res,List<Integer> Pre)
if Pre.size()==k and target==0
    Res.add(Copy(Pre))
    return
for start=dex to nums.length
    Pre.add(nums[start])
    Aux(nums,k,start+1,target-nums[start],Res,Pre)
    Pre.remove(Pre.size())
```

217、判断是否含有相同的元素**(**元素大小与数组长度无关）

public boolean containsDuplicate(int[] nums)

```
let set be a new Set<Integer>
for i=1 to nums.length
    if not set.add(nums[i]) return true
return false
```

218、城市天际线问题

Point 的第一种存储方式

```
public List<int[]> getSkyline(int[][] buildings)
1 let skyLine be a new ArrayList<int[]>
2 let points be a new ArrayList<int[]>
//一个建筑出现时为正，消失时为负
3 for each building:buildings
4       points.add({building[1],building[3]})
5       points add({building[2],-building[3]})
6 sort(points,new Comparator<int[]>(){
      public int compare(int[] obj1,int[] obj2){
          if obj1[1] ≠obj2[1] return obj1[1]-obj2[1]
          else return obj2[2]-obj1[2]
      }
7 let maxHeap be a MAXHEAP
8 curHeight=0,preHeight=0
9 for each point:points
10      if point[2]>0//当出现正值时，说明有一个建筑需要进入队列
11          maxHeap.offer(point[2])
12      else maxHeap.remove(-point[2])//出现负值说明该建筑已经到右边缘，将其退出队列
13      curHeight=maxHeap.peek()==null?0:maxHeap.peek()
14      if curHeight≠preHeight
15          skyLine.add({point[1],curHeight})
16          preHeight=curHeight
17 return skyLine
```

Line6:为什么当 obj[1]相同时，要把高度大的放在前面？

情况 1：若都为正数，即都是大楼刚出现的时刻，孰先孰后无关紧要

情况 2：若一正一负，若负在先，先将楼弹出最大堆，会导致以下情况

输入(0,2,3) 和 (2,5,3) 输出为(0,3) (2,0) (2,3) (5,0)

因此要求若在某一处即是某栋楼的开始，又是某栋楼的结束，先压入新楼，再弹出旧楼

Point 的另一种存储方式

```
public List<int[]> getSkyline(int[][] buildings)
1 let skyLine be a new ArrayList<int[]>
2 let points be a new ArrayList<int[]>
//一个建筑出现时为负，消失时为正
3 for each building:buildings
4       points.add({building[1],-building[3]})
5       points add({building[2],building[3]})
6 sort(points,new Comparator<int[]>(){
        public int compare(int[] obj1,int[] obj2){
            if obj1[1] ≠obj2[1] return obj1[1]-obj2[1]
            else return obj1[2]-obj2[2]
        }
7 let maxHeap be a MAXHEAP
8 curHeight=0,preHeight=0
9 for each point:points
10      if point[2]<0//当出现负值时，说明有一个建筑需要进入队列
11          maxHeap.offer(-point[2])
12      else maxHeap.remove(point[2]) //出现正值说明该建筑已经到右边缘，将其退出队列
13      curHeight=maxHeap.peek()==null?0:maxHeap.peek()
14      if curHeight≠preHeight
15          skyLine.add({point[1],curHeight})
16          preHeight=curHeight
17 return skyLine
```

Line6:为什么当 obj[1]相同时，要把高度小的放在前面？

情况 1：若都为负数，即都是大楼刚出现的时刻，孰先孰后无关紧要

情况 2：若一正一负，若正在先，先将楼弹出最大堆，会导致以下情况

输入(0,2,3) 和 (2,5,3) 输出为(0,3) (2,0) (2,3) (5,0)

因此要求若在某一处即是某栋楼的开始，又是某栋楼的结束，先压入新楼，再弹出旧楼

219、判断是否含有距离不超过 k 的相同元素

public boolean containsNearbyDuplicate(int[] nums, int k)

```
let set be a new Set<Integer>
for i=1 to nums.length
    if i>k set.remove(nums[i-k-1])
    if not set.add(nums[i]) return true
return false
```

220、判断是否含有距离不超过 k 的且差值不超过 t 的一对元素(share solution)

boolean containsNearbyAlmostDuplicate(int[] nums, int k, int t)

```
if nums==null or nums.length<2 or k<1 or t<0 return false
Long maximum=Integer.MIN_VALUE,minimum=Integer.MAX.VALUE
for each i:nums
      maximum=max(maximum,i)
      minimum=min(minimum,i)
Long numBucket=(maximum-minimum)/(t==0?1:t)+1
let buckets be a new Map<Long,LinkedList<Integer>> buckets
let queueNumBucket be a new Queue stroed Long
for i=1 to numBucket
      buckets.put(i,a new LinkedList)
for i=1 to nums.length
       Long dexBucket=(nums[i]-minimum)/(t==0?1:t)
       queueNumBucket.offer(dexBucket)
       if queueNumBucket.size()>k+1//队列中有 k+2 个元素时，弹出队头元素
            buckets[queueNumBucket.poll()].clear()
       curbucket=buckets[dexBucket]
       if curbucket.size()≠0 return true
       curbucket.add(nums[i])
       if dexBucket>1 and buckets[dexBucket-1].size()≠0
         and |buckets[dexBucket-1].getFirst()-long(nums[i])|≤t
            return true
      if dexBucket<numBucket and buckets[dexBucket+1].size()≠0
         and |buckets[dexBucket+1].getFirst()-long(nums[i])|≤t
           return true
return false
```

类似利用桶的还有 Code164

第 i 个桶的范围:[min+(i-1)*interval,min+i*interval)

nums[1...n]

A、**规定桶的个数为 n 个**：那么间隔为[(max-min)/(num.length-1)]

因为只有最大值会放到第 n 个桶中（特殊情况）

B、**规定桶的长度为 len**：那么桶的个数为□(max-min)/len□**+1**

其中+1 是为了补偿 max-min 恰能被 len 整除的时候，由于桶的范围是左闭右开的，需要将最大值放入额外的一个桶

221、最大的正方形区域（标记为'1'的区域）

public int maximalSquare(char[][] matrix)

```
if matrix==null or matrix.length==0 or matrix[1].length==0 return 0
maximum=0
for row=1 to matrix.length
    for col=1 to matrix[1].length
        if matrix[row][col]=='0' continue
        maximum=max(maximum,Aux(maximum,row,col)
```

private int Aux(char[][] matrix,int row,int col)

```
i=row+1,j=col+1
while i<matrix.length and j<matrix[1].length
    for k=col to j
        if matrix[i][k]=='0' return ((i-1)-row+1)* ((i-1)-row+1)
    for k=row to i-1
        if matrix[k][j]=='0' return((i-1)-row+1)* ((i-1)-row+1)
    i++ j++
return ((i-1)-row+1)* ((i-1)-row+1)
```

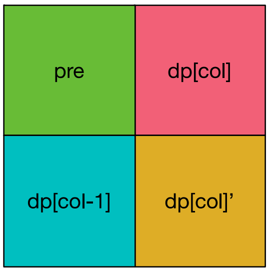
动态规划：

dp[row][col]=min(dp[row-1][col-1],dp[row][col-1],dp[row-1][col])+1

dp[row][col]存储的是，以(row,col)为右下端点的正方形的边长

```
public int maximalSquare(char[][] matrix)
rows=matrix.length
cols=rows>0?matrix[1].length:0
let dp[0...rows][0...cols] be a new array
res=0
for row=1 to rows
      for col=1 to cols
            if matrix[row][col]=='1'
                  dp[row][col]=min(dp[row-1][col-1],dp[row-1][col],dp[row][col-1])+1
            res=max(res,dp[row][col]*dp[row][col])
return res
```

```
public int maximalSquare(char[][] matrix)
rows=matrix.length
cols=rows>0?matrix[1].length:0
let dp[0...cols] be a new array
res=0,pre=0
for row=1 to rows
      for col=1 to cols
            tmp=dp[col]
            if matrix[row][col]=='1'
                  dp[col]=min(pre,dp[col-1],dp[col])
            else dp[col]=0
            res=Math.max(res,dp[col]*dp[col])
            pre=tmp
return res
```



pre	dp[col]
dp[col-1]	dp[col]’

1、每次迭代会从左到右计算 dp[col]，在更新 dp[col]之前，dp[col]的值为上一行该列的值，即代表了红色的区域

2、而 dp[col-1]已经是本行该列的新值，因此无法通过 dp 来表示 pre 部分，而 pre 却是 dp[col-1]更新之前的值，因此用额外的一个量来存储即可

222、完全二叉树节点个数(share solution) 复杂度 0(lg^2(n))

```
public int countNodes(TreeNode root)
LeftMostHeight=0
iter=root
while iter≠null
        LeftMostHeight++
        iter=iter.left
RightMostHeight=0
iter=root
while iter≠null
        RightMostHeight++
        iter=iter.right
if LeftMostHeight==RightMostHeight return (1<<LeftMostHeight)-1
left=0,right=(1<<RightMostHeight)-1
leafNum=Aux(root,left,right,RightMostHeight)
notleafNum=(1<<RightMostHeight)-1
return leafNum+notleafNum

private int Aux(TreeNode root,int left,int right,int len)
if left==right return left+1
if left==right-1 return left+1
mid=⌊(left+right)/2⌋
midNode=root
for i=1 to len
        midNode=((mid>>(len-i))&1)==0?midNode.left:midNode.right
if midNode==null return Aux(root,left,mid,len)
else return Aux(root,mid,right,len)

public int countNodes(TreeNode root)
LeftMostHeight=0
iter=root
while iter≠null
        LeftMostHeight++
        iter=iter.left
RightMostHeight=0
iter=root
while iter≠null
        RightMostHeight++
        iter=iter.right
if LeftMostHeight==RightMostHeight
         return (1<<LeftMostHeight)-1
else return 1+countNode(root.left)+countNodes(root.right)
```

2 的 n 次幂: 1<<n

223、两个矩形的面积

public int computeArea(int A, int B, int C, int D, int E, int F, int G, int H)

```
left=max(A,E)
bottom=max(B,F)
right=min(C,G)
top=min(D,H)
if left≥right or bottom≥top overlap=0
else overlap=(right-left)*(top-bottom)
areaA=(C-A)*(D-B),areaB=(G-E)*(H-F)
return areaA+areaB-overlap
```

224、含有加减法的运算表达式的实现(brillient!)

public int calculate(String s)

```
res=0
sign='+'
let stackVal and stackSign be two Stacks
for i=1 to s.length()
    if s[i]==' ' continue
    elseif Character.isDigit(s[i])
        curVal=0
        while i≤s.length() and Character.isDigit(s[i])
            curVal=curVal*10+s[i++]-'0'
        i--
        if sign=='+' res=res+curVal
        elseif sign=='-' res=res-curVal
    elseif s[i]=='('
        stackVal.push(res)
        stackSign.push(sign)
        res=0
        sign='+'
    elseif s[i]==')'
        res=stackVal.poll()+(stackSign.poll()=='+'?1:-1)*res
    else sign=s[i]
return res
```

优化方案：

```
public int calculate(String s)
sign=1,res=0//sign 最初赋值为 1，即正号
let stack be a new Stack
for i=1 to s.length()
    if Character.isDigit(s[i])
        curVal=0
        while i≤s.length() and Character.isDigit(s[i])
            curVal=curVal*10+(s[i++]-'0')
        i--
        res=res+curVal*sign
    elseif s[i]=='+' sign=1
    elseif s[i]=='-' sign=-1
    elseif s[i]=='('
        stack.push(res)
        stack.push(sign)
        res=0
        sign=1
    elseif s[i]==')'
        res=res*stack.pop()+stack.pop()
//空格部分什么也不做，直接跳过
return res
```

225、用队列实现栈

思路：在压入时，使得满足栈的性质

不变式：压入开始前，队列满足栈的性质，即队列头尾栈顶元素，队列尾为栈底元素
　　　　压入结束后，队列头元素为刚压入的元素，也满足栈的性质

```
class MyStack {
    //one Queue solution
    private Queue<Integer> queue = new LinkedList<Integer>()
    // Push element x onto stack.

    public void push(int x)
    queue.add(x)
    for i=2 to queue.size()
        queue.add(queue.poll())

    // Removes the element on top of the stack.
    public void pop()
    queue.poll()

    // Get the top element.
    public int top()
    return queue.peek()

    // Return whether the stack is empty.
    public boolean empty()
    return queue.isEmpty()
}
```

226、反转一颗 BST（二叉搜索树）

public TreeNode invertTree(TreeNode root)

```
helper(root)
return root
```

private void helper(TreeNode cur)

```
if cur≠null
    tem=cur.left
    cur.left=cur.right
    cur.right=tem
    helper(cur.left)
    helper(cur.right)
```

227、加减乘除运算表达式的计算（不含括号）类似的有 224

Solution1: Using Stack

```
public int calculate(String s)
if s==null or s.length==0 return 0
let stack be a new Stack
sign='+'
for i=1 to s.length()
    if s[i]==' ' continue
    elseif Character.isDigit(s[i])
        curVal=0
        while i≤s.length() and Character.isDigit(s[i])
            curVal=curVal*10+s[i++]-'0'
        i--
        if sign=='+'
            stack.push(curVal)
        elseif sign=='-'
            stack.push(-curVal)
        elseif sign=='*'
            stack.push(stack.pop()*curVal)
        elseif sign=='/'
            stack.push(stack.pop()/curVal)
    else sign=s[i]
res=0
while not stack.isEmtpy()
    res=res+stack.pop()
return res
```

给第一个数附上初始的符号'+'，将 符号-数字 视为一对

当数字出现后，根据与该数字配对的符号进行计算，将当前计算结果压进栈

另外当为乘法除法时，当前结果与上一次的结果有关，一次与栈顶元素计算后再压入栈

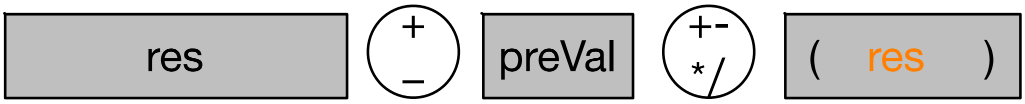
Solution 2:Not Using Stack

```
public int calculate(String s)
if s==null or s.length==0 return 0
res=0
preVal=0,curVal=0
sign='+'
for i=1 to s.length()
    if s[i]==' ' continue
    elseif Character.isDigit(s[i])
        curVal=0
        while i≤s.length() and Character.isDigit(s[i])
            curVal=curVal*10+s[i++]-'0'
        i--
        if sign=='+'//当前数值与前一个数值无关，将前一个数值更新到总和中
            res=res+preVal
            preVal=curVal
        elseif sign=='-'//当前数值与前一个数值无关，将前一个数值更新到总和中
            res=res+preVal
            preVal=-curVal
        elseif sign=='*'//当前数值与前一个数值有关，不更新总和
            preVal=preVal*curVal
        elseif sign=='/'//当前数值与前一个数值有关，不更新总和
            preVal==preVal/curVal
    else sign=s[i]
res=res+preVal//需要加上最后一个结果
return res
```

带有括号的加减乘除表达式的运算

```
public int calculate(String s)
1 let stkVal,stkRes be two new Stack stored int
2 let stkSign be new Stack stored sign
3 preVal=0
4 res=0
5 sign='+'
6 for i=1 to s.length()
7       if s[i]==' ' continue
8       elseif Character.isDigit(s[i])
9           curVal=0
10          while i≤s.length() Character.isDigit(s[i])
11              curVal=curVal*10+s[i++]-'0'
12          i--
13          if sign=='+'//当前数值与前一个数值无关，将前一个数值更新到总和中
14              res=res+preVal,preVal=curVal
15          elseif sign=='-' //当前数值与前一个数值无关，将前一个数值更新到总和中
16              res=res+preVal,preVal=-curVal
17          elseif sign=='*' preVal=preVal*curVal//当前数值与前一个数值有关，不更新总和
18          elseif sign=='/' preVal/curVal//当前数值与前一个数值有关，不更新总和
19      elseif s[i]=='('
20          stkRes.push(res)
21          stkVal.push(preVal)
22          stkSign.push(sign)
23          preVal=0,sign='+',res=0
24      elseif s[i]==')'
25          res=res+preVal//括号内是一个完整表达式，需要加上括号内最后一个数值
26          if stkSign.peek()=='+'//当前数值与前一个数值无关，将前一个数值更新到总和中
27              preVal=res,res=stkRes.poll()+stkVal.poll()
28          elseif stkSign.peek()=='-'//当前数值与前一个数值无关，将前一个数值更新到总和
29              preVal=-res,res=stkRes.poll()+stkVal.poll()
30          elseif stkSign.peek()=='*'//当前数值与前一个数值有关，不更新总和
31              preVal=stkVal.poll()*res,res=stkRes.poll()
32          elseif stkSign.peek()=='/'//当前数值与前一个数值有关，不更新总和
33              preVal=stkVal.poll()/res,res=stkRes.poll()
34          stkSign.pop()
35      else sign=s[i]
36 return res+preVal
```

Line 27-33 黄色的 res 的值是括号内表达式的值



228、归纳数字的范围

[0,1,2,3,5,6,7,9]->   ["0->3","5->7","9"]

public List<String> summaryRanges(int[] nums)

```
let res be a new LinkedList<String>
if nums==null or nums.length==0 return res
begin=1,end=2
while end≤nums.length
    if nums[end] ≠nums[end-1]+1
        if end-1-begin+1==1
            res.add(Integer.toString(nums[begin]))
        else
            res.add(Integer.toString(nums[begin])+"->"+ Integer.toString(nums[end-1])
    end++
if end-1-begin+1==1
    res.add(Integer.toString(nums[begin]))
else
    res.add(Integer.toString(nums[begin])+"->"+ Integer.toString(nums[end-1])
return res
```

229、找出主元（出现次数多于⌊n/3⌋)

思路：主元最多只有两个

```
public List<Integer> majorityElement(int[] nums)
1 element1=∞,element2=∞,cnt1=0,cnt2=0
2 for each i:nums
3         if i==element1 cnt1++
4         elseif i==element2 cnt2++
5         elseif cnt1==0 element1=i,cnt1=1
6         elseif cnt2==0 element2=i,cnt2=1
7         else cnt1--,cnt2--
8 cnt1=cnt2=0
9 for each i:nums
10        if i==element1 cnt1++
11        elseif i==element2 cnt2++
12 let res be a new List
13 if cnt1>nums.length/3 res.add(element1)
14 if cnt2>nums.length/3 res.add(element2)
15 return res
```

Line 5、6 行保证每次只更替一个

230、二叉搜索树的第 k 顺序数

```
int cnt=0
int res=0
boolean founded=false
public int kthSmallest(TreeNode root, int k)
helper(root,k)
return res

private void helper(TreeNode cur,int k){
if not founded and cur≠null
        helper(cur.left,k)
        cnt++
        if cnt==k res=cur.val,founded=true,return
        helper(cur.right,k)
```

231、判断是否是 2 的幂次

public boolean isPowerOfTwo(int n)

```
if n≤0 return false
cnt=0
for i=0 to 30
    cnt=cnt+(n>>i&1)
return cnt==1? true:false
```

232、用堆栈实现队列

思路：与利用队列实现栈一样，在压入时，使得其满足队列的性质

不变式：压入前，栈中元素满足队列的性质（栈顶为队列头，栈底为队列尾）
　　　　压入后，压入的元素位于栈底（也就是队列尾），满足队列的性质

```
class MyQueue {
    LinkedList<Integer> queue = new LinkedList<Integer>()
    // Push element x to the back of queue.
    public void push(int x)
    LinkedList<Integer> temp = new LinkedList<Integer>()
    while not queue.isEmpty()
        temp.push(queue.pop())
    queue.push(x)
    while not temp.isEmpty())
        queue.push(temp.pop())

    // Removes the element from in front of queue.
    public void pop()
    queue.pop()

    // Get the front element.
    public int peek()
    return queue.peek()

    // Return whether the queue is empty.
    public boolean empty()
    return queue.isEmpty()
}
```

233、1 的计数???

public int countDigitOne(int n)

```
onts=0
m=1
while m≤n
    ones=ones+(n/m+8)/10*m+(n/m%10==1?n%m+1:0)
    m=m*10
return ones
```

234、判断链表是否为 Palindrome O(n)复杂度 O(1)空间

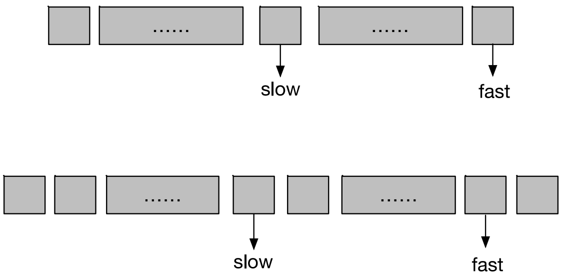
public boolean isPalindrome(ListNode head)

```
if head==null return false
slow=head,fast=head
while fast.next≠null and fast.next.next≠null//寻找中点的判断条件
        fast=fast.next.next
        slow=slow.next
slow=reverse(slow.next) //无论长度为奇数还是偶数，slow.next 是又半部分的开始
iter1=head,iter2=slow
while iter2≠null
        if iter1.val≠iter2.val return false
        iter1=iter1.next
        iter2=iter2.next
return true
```

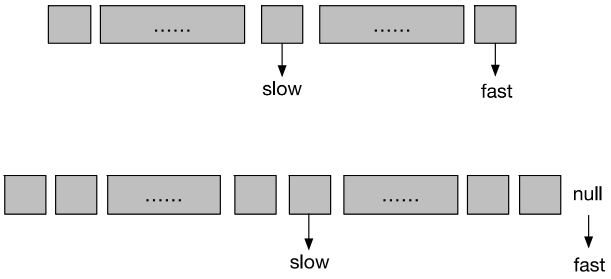
ListNode reverse(ListNode head)

```
headNew=null
iter=head
while iter≠null
        tem=iter.next
        iter.next=headNew
        headNew=iter
        iter=tem
return headNew
```

循环条件为 fast.next≠null **and** fast.next.next≠null



循环条件为 fast ≠null **and** fast.next ≠null



235、搜索二叉树两个节点的公共祖先

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

```
iter=root
parent=root
while iter≠null
    if iter.val==p.val or iter.val=q.val return iter
    parent=iter
    if iter.val>p.val and iter.val>q.val
    elseif iter.val<p.val and iter.val<q.val
    else break
return parent
```

236、一般二叉树两个节点的公共祖先

```
TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q)
let path1,path2 be two new empty List<TreeNode>
findPath(root,p,path1)
findPath(root,q,path2)
i=0
parent=root
while i≤path1.size() and i≤path2.size()
        if path1[i].val≠path2[i].val break
        parent=path1[i++]
return parent
```

```
private boolean findPath(TreeNode cur,TreeNode p,List<TreeNode> path)
if cur==null return false
path.add(cur)
if cur==p return true
if findPath(cur.left,p,path) return true
if findPath(cur.right,p,path) return true
path.remove(path.size())
return false
```

```
public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q)
if root==null or root==p or root==q return root
left= lowestCommonAncestor(root.left, p, q)
right= lowestCommonAncestor(root.right, p, q)
if left≠ null and right≠null return root//左右都不为空说明，p 和 q 分别位于 root 的左右子树中，因此他们的公共祖先 LSA 为 root
return left≠null? left:right//左为空说明，q、p 不存在与 root 的左子树中,右为空说明 p、q 不存在与 root 的右子树中
//注意，left，right 并不是 root.left 和 root.right，而是 left 位于 root.left，right 位于 root.right
```

237、删除链表中指定的节点（只给了这个节点）

由于无法获取当前节点的前一个节点，因此只能通过改变节点的值来达到删除的目的

```
public void deleteNode(ListNode node)
if node==null return
iter=node,pre=null
while iter.next≠null
    pre=iter
    iter.val=iter.next.val
    iter=iter.next
pre.next=null
```

238、计算数组中每一项的积（除了该项之外的积）

例如：{1，2，3，4}-->{2*3*4，1*3*4，1*2*4，1*2*3}

public int[] productExceptSelf(int[] nums)

```
let res[1...nums.length] be a new Array
tolProduct=help(nums,1)
res[1]=tolProduct
for i=2 to nums.length
    if nums[i]==0 //recalculate tolProduct
        tolProduct=helper(nums,i,)
    else
        tolProduct=tolProduct*nums[i-1]/nums[i]
    res[i]=tolProduct
return res
```

private int helper(int[] nums,int dex)

```
tolProduct=1
for i=1 to nums.length
    if i==dex continue
    if nums[i]==0
        tolProduct=0,break
    else tolProduct=tolProduct*nums[i]
return tolProduct
```

Brilliant Solution:

public int[] productExceptSelf(int[] nums)

```
res[0]=1
for i=2 to nums.length
    res[i]=res[i-1]*nums[i-1]
right=1
for i=nums.length downto 1
    res[i]=res[i]*right
    right=right*nums[i]
return res
```

239、移动窗口的最大值数组

例如数组{1,2,3,4,5} 窗口长度为 3

窗口第一次在{1,2,3}，最大值 3，第二次在{2,3,4}，最大值 4，第三次在{3,4,5}，最大值 5

输出为{3,4,5}，即每次窗口最大值所构成的数组

```
public int[] maxSlidingWindow(int[] nums, int k)
if nums==null or nums.length==0 return nums
let queue be a PriorityQueue<Integer>
res[1...nums.length-k+1] be a new Array
for i=1 to k
    queue.offer(nums[i])
res[1]=queue.peek()
for i=k+1 to nums.length
    queue.remove(nums[i-k])
    queue.offer(nums[i])
    res[i-k+1]=queue.peel()
return res
```

Brilliant Solution:

```
public int[] maxSlidingWindow(int[] nums, int k)
if nums==null or nums.length==0 return nums
res[1...nums.length-k+1] be a new Array
dex=1
let queue be a new Queue<Integer>
for i=1 to nums.length
// remove numbers out of range k
    while not queue.isEmtpy() and queue.peek()<i-k+1
        queue.poll()
// remove smaller numbers in k range as they are useless
//保证队列的尾部（当前值）是队列的最小值
    while not queue.isEmtpy() and nums[queue.getLast()]<nums[i]
        queue.removeLast()
    queue.offer(i)
    if i≥k
        res[dex++]=nums[queue.peek()]
return res
```

始终保持队头为当前窗口值最大的索引

240、搜索有序矩阵中搜索给定值

public boolean searchMatrix(int[][] matrix, int target)

```
if matrix==null or matrix.length==0 or matrix[0].length==0 return false
col=matrix[1].length
row=1
while col≥1 and row ≤matrix.length
    if target==matrix[row][col] return true
    elseif target<matrix[row][col] col--
    elseif target>matrix[row][col] row++
return false
```

241、求一个包含"**+　-　***"的运算表达式添括号所得的所有结果（结果可能重复）

问题转化为：两个子问题（两个子问题 由 选择先算哪个运算符号所得）

public List<Integer> diffWaysToCompute(String input)

```
let res be a new empty List<Integer>
if input==null or input.length==0 return res
isSingleValue=true
for i=1 to input.length
    if "+-*".indexOf(input[i]) ≠-1
        isSingleValue=false
        String inputLeft=input[1...i-1]
        String inputRight=input[i+1...end]
        resLeft= diffWaysToCompute(inputLeft)
        resRight= diffWaysToCompute(inputRight)
        for each Integer:resLeft
            for each Integer:resRight
                if input[i]=='+'
                    res.add(left+right)
                elseif input[i]=='-'
                    res.add(left-right)
                else
                    res.add(left*right)
if isSingleValue res.add(Integer.parseInt(input))
return res
```

242、判断两个字符串是否包含相同的元素

public boolean isAnagram(String s, String t)

```
if s==null or t==null return false
if s.length≠t.length return false
aryS=s.toCharArray()
aryT=t.toCharArray()
sort(aryS),sort(aryT)
for i=1 to aryS.length
    if aryS[i] ≠aryT[i] return false
return true
```

257、二叉树的所有路径（从根到叶节点）

```
public List<String> binaryTreePaths(TreeNode root)
let res be a new ArrayList<String>
if root≠null helper(root,"",res)
return res
```

```
void helper(TreeNode root, String path, List<String> res)
if root.left==null and root.right==null res.add(path+root.val)
if root.left≠null helper(root.left,path+root.val+"->",res)
if root.right≠null helper(root.right,path+root.val+"->",res)
```

关键：1. "->"放置位置

由于第一个节点会直接填写，而其余节点均要用->链接
path+root.val 作为通用表达式，既要适用于根节点，又要适用于其他节点，因此"->"应该放在 path 中

258、数字各个位相加，直至和为个位数

```
public int addDigits(int num)
while $num \geq 10$
    num=helper(num)
return num

private int helper(int num)
res=0
while $num \neq 0$
    res=res+num%10
    num=num/10
return res
```

260、找到两个只出现过一次的数（在一个数组中，其余元素均出现两次）

public int[] singleNumber(int[] nums)

```
1 xor=0
2 for each num of nums
3       xor=xor^num
4 offset=0
5 while (xor&(1<<offset))==0
6       offset++
* 4 offset=31
* 5 while (xor&(1<<offset))==0
* 6       offset--
7 divide=1<<offset
8 rets={0,0}
9 for each num of nums
10      if num&divide==0
11          rets[0]=rets[0]^num
12      else
13          rets[1]=rets[1]^num
14 return rets
```

由于 xor 最后只是这两个只出现一次的元素的异或结果，由于这两个元素不同，因此异或结果 xor 必然包含 1（bit）任选其中一个 1bit 作为两组元素的分隔

在两组中各自进行异或，每一组只含有一个出现一次的元素，因此异或结果就是该元素

263、判断是否是为 ugly 数字

ugly 满足其因式分解只包含 2、3、5，且定义 1 为 ugly 数字

```
public boolean isUgly(int num)
if num==1 return true
if num==0 return false
while num≠1
    if num%2==0
        num=num/2
    elseif num%3==0
        num=num/3
    elseif num%5==0
        num=num/5
    else return false
return true
```

264、第 n 个 ugly 数字

1*2　2*2　3*2　4*2　5*2...
1*3　2*3　3*3　4*3　5*3...
1*5　2*5　3*5　4*5　5*5...

DP:

public int nthUglyNumber(int n)

```
let ugly[1...n] be a new array
ugly[1]=1
index2=1,index3=1,index5=1
factor2=2,factor3=3,factor5=5
for i=2 to n
    minimum=min(min(factor2,factor3),factor5)
    ugly[i]=minimum
    if factor2==minimum//并列的判断
        factor2=2*ugly[++index2]
    if factor3==minimum//并列的判断
        factor3=3*ugly[++index3]
    if factor5==minimum//并列的判断
        factor5=5*ugly[++index5]
return ugly[n]
```

往前推进的方式并不是(1 2 3 4 5 6 7...)*2(or3or5): 2*(++index2)
而是下一个 ugly 数字乘以 2(or3or5): 2*ugly[++index2]

268、丢失的数字

public int missingNumber(int[] nums)

```
res=nums.length
i=0
for each num of nums
    res=res^num
    res=res^i++
return res
```

输入只能缺少一个数字，而不能为一串

若没有缺少数字，即第一个丢失的数字为下一个数字 n

0 1 2 3...n-1

若丢失了数字 k

那么数组为

0 1 2 ...k-1 k+1...n-1 n

对比两个数组，除了数字 k 与数字 n 之外，剩余数字出现了 2 次，用异或便可滤去这些数字

273、阿拉伯数字变英文表达

```
public String numberToWords(int num)
if num==0 return "Zero"
return helper(num)

private String helper(int num)
if num<1000 return lessThousandToWord(num)
if num<1000*1000 return lessThousandToWord(⍰num/1000⍰)+
                " Thousand"+(num%1000==0?"":" "+helper(num%1000))
if num<1000*1000*1000 return lessThousandToWord(⍰num/(1000*1000) ⍰)+
                " Million"+(num%(1000*1000)==0?"":" "+helper(num%(1000*1000)))
return lessThousandToWord(⍰num/(1000*1000*1000) ⍰)+
                " Billion"+(num%(1000*1000*1000)==0?"":" "+helper(num%(1000*1000*1000)))

private String lessThousandToWord(int n){
if n<10 return digitToWord(n)
if n≥10&&n≤19 return tenToWord(n)
if n≥20&&n<100 return tensToWord(n)+(n%10==0?"":" "+digitToWord(n%10))
return digitToWord(⍰n/100⍰)+" Hundred"+(n%100==0?"":" "+lessThousandToWord(n%100))

private String digitToWord(int n)
switch(n)
case 1:return "One"
case 2:return "Two"
case 3:return "Three"
case 4:return "Four"
case 5:return "Five"
case 6:return "Six"
case 7:return "Seven"
case 8:return "Eight"
case 9:return "Nine"
default: return ""
```

private String tensToWord(int n)

```
switch(n/10)
case 2: return "Twenty"
case 3: return "Thirty"
case 4: return "Forty"
case 5: return "Fifty"
case 6: return "Sixty"
case 7: return "Seventy"
case 8: return "Eighty"
case 9: return "Ninety"
default: return ""
```

private String tenToWord(int n)

```
switch(n)
case 10: return "Ten"
case 11: return "Eleven"
case 12: return "Twelve"
case 13: return "Thirteen"
case 14: return "Fourteen"
case 15: return "Fifteen"
case 16: return "Sixteen"
case 17: return "Seventeen"
case 18: return "Eighteen"
case 19: return "Nineteen"
default: return ""
```

274、论文 h 索引

学者的文论索引 h 定义如下：

N 篇 paper 中，至少被引用 h 次的论文有 h 篇，剩余 N-h 篇论文被引用次数不多于 h 次

public int hIndex(int[] citations)

```
1 len=citations.length
2 let count[0...len] be a new array//利用了线性时间排序算法的思路
3 for each c of citations
4       if c≥len count[len]++
5       else count[c]++
6 total=0
7 for i=len downto 0
8       total=total+count[i]
9       if total≥i
10          return i
11 return 0
```

为什么第 9 行是大于等于而不是等于：

{1,1}当 i 为 2 是 total 为 0，total 为 1 时 total 为 2

7-9 行迭代过程中：若当前迭代 total 没有发生改变，而 i 相比于上次迭代减少了 1，此时若条件成立则一定是取等号

275、论文 h 索引（对于已排序的情况）（二分法）

```
public int hIndex(int[] citations)
left=0,len=citations.length,right=len-1
while left<right
    mid=⌊(left+right)/2⌋
    if citations[mid] <len-mid left=mid+1
    else right=mid-1
if len==0 return 0
return citations[left] <len-left?len-left-1:len-left
```

```
public int hIndex(int[] citations)
left=0,len=citations.length,right=len-1
while left≤right
    mid=⌊(left+right)/2⌋
    if citations[mid] <len-mid left=mid+1
    else right=mid-1
return len-left
```

278、找到第一个错误的版本（二分搜索）

```
public int firstBadVersion(int n)
left=1,right=n
while left<right
    mid=⌊(left+right)/2⌋
    if not isBadVersion(mid) left=mid+1
    else right=mid-1
return isBadVersion(left)?left:left+1
```

```
public int firstBadVersion(int n)
left=1,right=n
while left≤right
    mid=⌊(left+right)/2⌋
    if not isBadVersion(mid) left=mid+1
    else right=mid-1
return left
```

细节：判断条件为 left≤right 时，其结果往往不需要讨论
判断条件为 left<right 时，其结果往往需要讨论

共性：进行二分时，要么取右边，要么取左边，都不包括 mid，换言之，新的迭代区域的边界情况（若取右边，[mid+1,right]中的 mid+1，若取左边[left,mid-1]中的 mid-1）并不确定
原因：当 left<right 时，终止时 left==right，但这点的情况并不明确，因此需要对该点进行判断后才能输出最后结果；当 left≤right 时，终止时 left>right，并且 left 的情况是明确的，因为当 left==right 时的迭代会使得 left 处于正确的位置

279、整数分解成平方数的最少个数

DP: dp[i+j*j]=min(dp[i+j*j],dp[i]+1)

public int numSquares(int n)

```
let dp[0...n] be a new array initialized to +∞
dp[0]=0//使得平方数也是正确的结果：1
for i=0 to n
        for j=1 to i+j*j≤n
                dp[i+j*j]=min(dp[i+j*j],dp[i]+1)
return dp[n]
```

很神奇的一点，在你求 **dp[i+j*j]**时，所访问的 **dp[i]**一定是值为 **i** 时的最小值，也就是说更新一定会发生在被调用之前

被更新时为 $i_1+j_1*j_1$，被调用时为 i_2，满足 $i_2= i_1+j_1*j_1$

当被调用时,$i==i_2$，因此 $i_2+j*j>i_2$，因此不可能再被更新了

282、结果为设定值得运算表达式

```
public List<String> addOperators(String num, int target)
let res be a new List<String>
let sb be a new StringBuilder
if num==null or num.length==0 return res
helper(res,num,target,sb,1,0,0)
return res

void helper(List<String> res,String num,int target,StringBuilder sb, int pos,long sum,long preVal)
if pos==num.length+1
    sum+=preVal
    if sum==target
        res.add(sb.toString())
    return
for i=pos to num.length
    if i≠pos and num[pos]=='0' break //首位是 0 的话不可以多个数合并成一个数，只能单独作为一位
    curVal=Long.parseLong(num[pos...i])
    originalLength=sb.length()
    if pos==1//整个表达式的第一个数字不需要携带符号
        sb.append(curVal)
        helper(res,num,target,sb,i+1,sum+preVal,curVal)
        sb.setLength(originalLength)
    else
        sb.append("+").append(curVal)
        helper(res,num,target,sb,i+1,sum+preVal,curVal)
        sb.setLength(originalLength)

        sb.append("-").append(curVal)
        helper(res,num,target,sb,i+1,sum+preVal,-curVal)
        sb.setLength(originalLength)

        sb.append("*").append(curVal)
        helper(res,num,target,sb,i+1,sum,preVal*curVal)
        sb.setLength(originalLength)
```

关键：将[+-*/][digit] 作为一个整体

283、将零元素移动到最后面，并且保持其余元素相对顺序

public void moveZeroes(int[] nums)

```
moves=0
for i=1 to nums.length
    if nums[i]==0 moves++
    else nums[i-moves]=nums[i]
for i=nums.length+1-moves to nums.length
    nums[i]=0
```

284、定义带有 peek 功能的迭代器

```
class PeekingIterator implements Iterator<Integer> {
    private Integer peek
    private Iterator<Integer> iterator
    public PeekingIterator(Iterator<Integer> iterator)
    this.iterator=iterator
    peek=null

    public Integer peek()
    if peek==null
    peek=iterator.next()
    return peek

    public Integer next()
    if peek≠null
        int res=peek
        peek=null
        return res
    else return iterator.next()

    public boolean hasNext() {
    if peek≠null return true
    else return iterator.hasNext()
```

287、找到重复的元素（n+1 个元素，[1-n]）

public int findDuplicate(int[] nums)

```
for i=1 to nums.length
    if nums[i]==i continue
    while nums[i]≠i
        if nums[nums[i]]==nums[i] return nums[i]
        tem=nums[i]
        nums[i]=nums[nums[i]]
        nums[tem]=tem//此时 nums[i]已经改变了
return 0//if input is right,never return this
```

289、生死游戏

Rules:

1.Any live cell with fewer than two live neighbors dies, as if caused by under-population.

2.Any live cell with two or three live neighbors lives on to the next generation.

3.Any live cell with more than three live neighbors dies, as if by over-population..

4.Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

状态-->状态转移-->状态

* **live-live:2**
* **live-dead:3**
* **dead-live:4**
* **dead-dead:5**

public void gameOfLife(int[][] board)

```
if board==null or board.length==0 or board[1].length==0
for i=1 to board.length
    for j=1 to board[1].length
        board[i][j]=state(board,i,j)
for i=1 to board.length
    for j=1 to board[1].length
        if board[i][j]==2 or board[i][j]==4
            board[i][j]=1
        elseif board[i][j]==3 or board[i][j]==5
            board[i][j]=0
```

private int state(int[][] board,int row,int col)

```
left=row>1? row-1:row
right=row<board.length?row+1:row
top=col>1? col-1:col
bottom=col<board[1].length? col+1:col
liveCnt=0
for i=left to right
    for j=top to bottom
        if i==row and j==col continue
        else liveCnt+= (board[i][j]==1||board[i][j]==2||board[i][j]==3)?1:0
if board[row][col]==1
   if liveCnt<2 or liveCnt>3 return 3 //rule1、3
   else return 2//rule 2
else
   if liveCnt==3 return 4 //rule4
   else return 5
```

290、模式匹配（同构）与205类似

public boolean wordPattern(String pattern, String str)

```
strAry=str.split(" ")
if pattern.length()≠strAry.length return false
let charPos be a new HashMap<Character,Integer>
let wordPos be a new HashMap<String,Integer>
for i=1 to pattern.length
    c=pattern[i]
    word=strAry[i]
    if charPos.containsKey(c)
        lastPos=charPos.get(c)
        if (not wordPos.containsKey(word)) or wordPos.get(word) ≠lastPos return false
        charPos.put(c,i)
        wordPos.put(word,i)
    else
        if wordPos.containsKey(word) return false
        charPos.put(c,i)
        wordPos.put(word,i)
return true
```

292、Nim 游戏（搬运石块）

```
public boolean canWinNim(int n)
if n<3 return true
let dp[1...n] be a new array stored boolean
dp[1]=dp[2]=dp[3]=true
for i=4 to n
    dp[i]=(dp[i-1]&&dp[i-2]&&dp[i-3])? false:true
return dp[n]
```

(dp[i-1]&&dp[i-2]&&dp[i-3])表示对于 i-1,i-2,i-3 块石头，先手必赢时，那么 i 块石头必输

```
public boolean canWinNim(int n)
if n%4==0 return false
else return true
```

295、实现可以输出中位数的容器

让较大的一半与较小的一半分开存储

```
class MedianFinder
    Queue<Integer> large= new PriorityQueue<Integer>()
    Queue<Integer> small= new PriorityQueue<Integer>(Collections.reverseOrder())

    public void addNum(int num)
    large.add(num)
    small.add(large.poll())//这里弹出的是 large 中的最小值
    if (large.size()<small.size())
        large.add(small.poll())//保证 large 的元素个数不少于 small 的元素个数

    public double findMedian()
    return large.size()>small.size()?
        large.peek():(large.peek()+small.peek()) / 2.0
```

296、实现树到字符串，字符串到树的转化(对应关系可自己定义，确保可逆即可）

方法 1：若一个节点只有左子树,val+"R"
若一个节点只有右子树,val+"L"
若一个节点为叶节点,val+"A"

```
private String final leftNull="L",rightNull="R",allNull="A"
public String serialize(TreeNode root)
let sb be a new StringBuilder
if root≠null helper1(root,sb) //R,L,A 必须通过存在的节点的孩子节点的状况进行判断
return sb.toString()

private void helper1(TreeNode root,StringBuilder sb)//需要保证 root 不为 null
if root.left==null and root.right==null
    sb.append(root.val).append(',').append(allNull).append(',')
elseif root.left==null
    sb.append(root.val).append(',').append(leftNull).append(',')
    helper1(root.right,sb)
elseif root.right=null
    sb.append(root.val).append(',').append(rightNull).append(',')
    helper1(root.left,sb)
else sb.append(root.val).append(',')
    helper1(root.left,sb)
    helper2(root.right,sb)

private int iter//基本类型无法像类对象可以以引用的方式传递，共享唯一一份数据
public TreeNode deserialize(String data)
iter=0
if data.equals("") return null
return helper2(data.split(","))//可以保证第一个字符串为数字

private TreeNode helper2(String[] strAry)//需要保证 iter 当前指向的字符串是数子
curVal=Integer.parseInt(strAry[iter++])
root=new TreeNode(curVal)
if strAry[iter].equals(allNull)
    iter++//skip "A"
elseif strAry[iter].equals(leftNull)
    iter++
    root.right=helper2(strAry)
elseif strAry[iter].equals(rightNull)
    iter++
    root.left=helper2(strAry)
else root.left=helper2(strAry)
    root.right=helper2(strAry)
return root
```

另一种思路按前序遍历，当前节点是 null 就返回添加'X'

```
private final String spliter=",",Null="N"
public String serialize(TreeNode root)
let sb be a new StringBuilder
helper1(root,sb)
return sb.toString()

private void helper1(TreeNode root, StringBuilder sb)
if root==null
    sb.append(Null).append(spliter)
else
    sb.append(root.val).append(spliter)
    helper1(root.left,sb)
    helpter2(root.right,sb)

public TreeNode deserialize(String data)
iter=0
return helper2(data.split(spliter))

private int iter//基本类型无法像类对象可以以引用的方式传递，共享唯一一份数据
private TreeNode helper2(String[] strAry)
if strAry[iter].equals(Null)
    iter++
    return null
else
    root=new TreeNode(Integer.parseInt(strAry[iter++]))
    root.left=helper2(strAry)
    root.right=helper2(strAry)
    return root
```

只有前序遍历无法构建唯一二叉树，但是给出 Null 节点后便可以了

299、指出完全相同（数值和位置）的数字个数，以及数字相同位置不同的数字个数

public String getHint(String secret, String guess)

```
let secretAry[1...10] guessAry[1...10] be new Arrays
countA=0
for i=1 to secret.length
    if secret[i]==guess[i] countA++
    secret[secret[i]-'0']++
    guess[guess[i]-'0']++
countB=0
for i=1 to 10
    countB+=min(secretAry[i],guessAry[i])
countB=countB-countA//countB 包含所有数值相同的数字个数，要减去位置也相同的才是位置不同数值相同的数字个数
return countA+"A"+countB+"B"
```

300、最长单调递增子序列

$O(n^2)$

public int lengthOfLIS(int[] nums)

```
if nums==null or nums.length==0 return 0
let dp[1...nums.length] be a new Array initialized to 1
maximum=1
for i=2 to nums.length
    for j=1 to i-1
        if nums[i]>nums[j] dp[i]=max(dp[i],dp[j]+1)
    maximum=max(maximum,dp[i])
return maximum
```

O(nlgn)

public int lengthOfLIS(int[] nums)

```
let dp[0...nums.length-1] be a new Array
len=0
for each n of nums
    i=binarySearch(dp,0,len,n)
    if i<0 i=-i-1
    dp[i]=n
    if i==len len++
return len
```