# 第二章

## 1.数组中重复的数字\*

Number[index]==index，当前值是否在值索引处

**package** cn.lzw.jianzhioffer.chapter2;

**import** java.util.Scanner;

/\*\*题目：在一个长度为n的数组里的所有数字都在0到n-1的范围内。 数组中某些数字是重复的，

\* 但不知道有几个数字是重复的。也不知道每个数字重复几次。请找出数组中任意一个重复的数字。

\* 例如，如果输入长度为7的数组{2,3,1,0,2,5,3}，那么对应的输出是第一个重复的数字2。

\* 思路：1.个人，采用额外o(n)空间的数组，以下标为标识，每读到一个数，它对应的下标值加1，

\* 一旦发现该下标的值为2，即为重复数字。

\* 2.遍历数组，按照排序的思想，首先，每个下标位置的数字都应该等于下标。如果不是，

\* 每个数字就应交换到它对应值的下标位置，如果该位置已经有与它相等的数

\* 就找到重复的，如果没有就交换位置。

\* 尽管有两次循环，但是每个数字最多只要交换2次，因此时间复杂度为O(N).

\*

\* **@author** lzw

\* **@date** 2018年1月2日

\* **@version** version1.0

\*/

**public** **class** DuplicateNumsInArray {

**public** **static** **void** main(String[] args) {

Scanner sc=**new** Scanner(System.***in***);

String str=sc.nextLine();

String[] strAry=str.split("\\s+");

**int**[] numbers=**new** **int**[strAry.length];

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

numbers[i]=Integer.*valueOf*(strAry[i]);

}

**int**[] d=**new** **int**[1];

**boolean** valid=*duplicate*(numbers,numbers.length,d);

System.***out***.println(valid+" "+d[0]);

}

/\*

public boolean duplicate(int numbers[],int length,int [] duplication) {

if(numbers==null||length<1){

return false;

}

int[] times=new int[length];

boolean valid=false;

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

times[numbers[i]]++;

if(times[numbers[i]]==2){

valid=true;

duplication[0]=numbers[i];

break;

}

}

return valid;

}

\*/

**public** **static** **boolean** duplicate(**int** numbers[],**int** length,**int** [] duplication) {

**if**(numbers==**null**||length<1){

**return** **false**;

}

**boolean** valid=**false**;

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

**while**(i!=numbers[i]){

**if**(numbers[numbers[i]]==numbers[i]){

valid=**true**;

duplication[0]=numbers[i];

**break**;

}**else**{

*swap*(numbers,i,numbers[i]);

}

}

**if**(valid){

**break**;

}

}

**return** valid;

}

**public** **static** **void** swap(**int**[] ary,**int** i, **int** j){

**int** tmp=ary[i];

ary[i]=ary[j];

ary[j]=tmp;

}

}

## 2.二维数组的查找\*\*

以右上角数字为基准，比较是左移还是下移

**package** cn.lzw.jianzhioffer.chapter2;

/\*\*题目：在一个二维数组中，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：从右上角查询，若比目标大，就列减1往左移，若比目标小，就行加1往下移，直到遇到相等 返回true，否则返回false

\*

\* **@author** lzw

\* **@date** 2018年1月3日

\* **@version** version1.0

\*/

**public** **class** FindNumIn2DArray {

**public** **boolean** Find(**int** target, **int** [][] array) {

**if**(array==**null**||array.length<1||array[0].length<1){

**return** **false**;

}

**int** j=array[0].length-1;

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

**for**(;j>=0;j--){

**if**(array[i][j]==target){

**return** **true**;

}**else** **if**(array[i][j]>target){

**continue**;

}**else**{

**break**;

}

}

}

**return** **false**;

}

}

## 3.替换空格\*\*

计算空格出现的次数，构建新长度数组，从后往前插入

**package** cn.lzw.jianzhioffer.chapter2;

/\*\*题目：请实现一个函数，将一个字符串中的空格替换成“%20”。

\* 例如，当字符串为We Are Happy.则经过替换之后的字符串为We%20Are%20Happy。

\* 思路：1.我的，利用StringBuffer的insert函数，判断某个位置是空格，则插入字符数组{%20}，

\* 但要注意，循环条件的更改，不能是str.length()，因为长度一直在变，会出现异常。

\* 2.书本，遍历一次统计空格次数cnt，构建一个长度为cnt\*2+原始数组长度的数组。然后两个后边指针

\* 往前移，一旦遇到空格，P2就添加（%20）,否则就直接复制。

\* 注意 ' '是空格，而'\0'是空字符

\*

\* **@author** lzw

\* **@date** 2018年1月3日

\* **@version** version1.0

\*/

**public** **class** ReplaceAllspace {

/\*

public String replaceSpace(StringBuffer str) {

char[] c={'%','2','0'};

int length=str.length();

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

if(str.charAt(i)==' '){

str.deleteCharAt(i);

str.insert(i,c);

i+=2;

length+=2;

}

}

return str.toString();

}

\*/

**public** String replaceSpace(StringBuffer str) {

String s=str.toString();

**char**[] c=s.toCharArray();

**int** cnt=0;

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

**if**(c[i]==' '){

cnt++;

}

}

**char**[] ch=**new** **char**[cnt\*2+c.length];

**int** k=ch.length-1;

**for**(**int** j=c.length-1;j>=0;j--){

**if**(c[j]==' '){

ch[k--]='0';

ch[k--]='2';

ch[k--]='%';

}**else**{

ch[k--]=c[j];

}

}

**return** String.*valueOf*(ch);

}

}

## 4.从尾到头打印链表\*\*

先进后出采用递归方式实现（高级），栈

**package** cn.lzw.jianzhioffer.chapter2;

**import** java.util.ArrayList;

/\*\*题目：输入一个链表，从尾到头打印链表每个节点的值。

\* 思路：利用栈或者递归

\*

\* **@author** lzw

\* **@date** 2018年1月3日

\* **@version** version1.0

\*/

**public** **class** PrintListFromTailToHead {

/\*

public ArrayList<Integer> printListFromTailToHead(ListNode listNode) {

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

ArrayList<Integer> list=new ArrayList<Integer>();

while(listNode!=null){

stack.push(listNode.val);

listNode=listNode.next;

}

while(!stack.empty()){

list.add(stack.pop());

}

return list;

}

\*/

**public** ArrayList<Integer> printListFromTailToHead(ListNode listNode) {

ArrayList<Integer> list=**new** ArrayList<Integer>();

recursionList(list,listNode);

**return** list;

}

**public** **void** recursionList( ArrayList<Integer> list,ListNode node){

**if**(node!=**null**){

recursionList(list,node.next);

list.add(node.val);

}

}

}

**class** ListNode {

**int** val;

ListNode next = **null**;

ListNode(**int** val) {

**this**.val = val;

}

}

## 5.重建二叉树（前序遍历和中序遍历）\*\*\*\*

在中序遍历中匹配前序遍历的首个指引，然后把中序分为左右子树，接着递归

**package** cn.lzw.jianzhioffer.chapter2;

/\*\*题目：给定二叉树的前序遍历和中序遍历数组，要求重建二叉树

\* 思路：因为前序遍历的每个节点都是中节点，因此若找到对应中序遍历的位置，左边即为左子树，右边即为右子树。

\* 每次都重新定位各索引位置，因此采用递归思想。

\* 注意递归函数的参数，是前序，前序开始索引，前序终止索引，中序，中序开始索引，中序终止索引。

\* 并且，有左子树就是leftLength>0,有右子树就是leftLength<preEnd-preStart

\* 特殊测试，前序和中序不匹配。因此当遍历完中序也找不到对应的索引，就要抛出异常

\*

\* **@author** lzw

\* **@date** 2018年1月3日

\* **@version** version1.0

\*/

**public** **class** ReConstructBinaryTree {

**public** TreeNode reConstructBinaryTree(**int** [] pre,**int** [] in)**throws** Exception {

**if**(pre==**null**||in==**null**||pre.length<1||in.length<1){

**return** **null**;

}

**return** reConstructBinaryTreeCore(pre,0,pre.length-1,in,0,in.length-1);

}

**public** TreeNode reConstructBinaryTreeCore(**int**[] pre,**int** preStart,**int** preEnd,**int**[] in,**int** inStart,**int** inEnd)**throws** Exception{

TreeNode node=**new** TreeNode(pre[preStart]);

node.left=**null**;

node.right=**null**;

**int** rootIndex=inStart;

**while**(rootIndex<=inEnd&&in[rootIndex]!=pre[preStart]){

rootIndex++;

}

**if**(rootIndex==inEnd&&in[rootIndex]!=pre[preStart]){

**throw** **new** Exception("invalid input");

}

**int** leftLength=rootIndex-inStart;

**if**(leftLength>0){

node.left= reConstructBinaryTreeCore(pre,preStart+1,preStart+leftLength,in,inStart,rootIndex-1);

}

**if**(leftLength<preEnd-preStart){

node.right= reConstructBinaryTreeCore(pre,preStart+leftLength+1,preEnd,in,rootIndex+1,inEnd);

}

**return** node;

}

}

**class** TreeNode {

**int** val;

TreeNode left;

TreeNode right;

TreeNode(**int** x) { val = x; }

}

## 6.二叉树的下一个节点\*\*\*

给定一个二叉树和其中的一个结点，请找出中序遍历顺序的下一个结点并且返回。注意，树中的结点不仅包含左右子结点，同时包含指向父结点的指针

解题：有右子树，则递归找左儿子；无右子树，则递归找左爸爸

**package** cn.lzw.jianzhioffer.chapter2;

/\*\*题目：给定一个二叉树和其中的一个结点，请找出中序遍历顺序的下一个结点并且返回。

\* 注意，树中的结点不仅包含左右子结点，同时包含指向父结点的指针。

\* 思路：1.首先判断有没有右子树。有的话 递归找到右子树对应的第一个左子树，如果没有就返回当前的右子树。

\* 没有的话。递归找到第一个是爸爸的左子树，有的话返回爸爸。没有的话就返回null

\* **@author** lzw

\* **@date** 2018年1月1日

\* **@version** version1.0

\*/

**public** **class** BinaryTreeNextNode {

**public** TreeLinkNode GetNext(TreeLinkNode pNode)

{

**if**(pNode==**null**){

**return** **null**;

}

**if**(pNode.right!=**null**){

**return** getLeftSon(pNode.right);

}**else**{

**return** getLeftFather(pNode);

}

}

**public** TreeLinkNode getLeftSon(TreeLinkNode node){

TreeLinkNode left=node.left;

**if**(left==**null**){

**return** node;

}**else**{

**return** getLeftSon(left);

}

}

**public** TreeLinkNode getLeftFather(TreeLinkNode node){

TreeLinkNode father=node.next;

**if**(father==**null**){

**return** **null**;

}

**if**(node==father.left){

**return** father;

}**else**{

**return** getLeftFather(father);

}

}

}

**class** TreeLinkNode {

**int** val;

TreeLinkNode left = **null**;

TreeLinkNode right = **null**;

TreeLinkNode next = **null**;

TreeLinkNode(**int** val) {

**this**.val = val;

}

}

## 7.用两个栈实现队列\*

一个栈1只作压入，一个栈2在有数据直接弹出，没数据的话先把栈1的数据全部弹出压进来

**package** cn.lzw.jianzhioffer.chapter2;

**import** java.util.Stack;

/\*\*题目：两个栈实现队列

\* 思路：入队：栈1直接压栈

\* 出队：栈2若为空，则把栈1的数据都弹出压到栈2， 最后弹出栈2

\* **@author** lzw

\* **@date** 2018年1月3日

\* **@version** version1.0

\*/

**public** **class** TwoStackToQueue {

Stack<Integer> stack1 = **new** Stack<Integer>();

Stack<Integer> stack2 = **new** Stack<Integer>();

**public** **void** push(**int** node) {

stack1.push(node);

}

**public** **int** pop() {

**if**(stack2.empty()){

**while**(!stack1.empty()){

stack2.push(stack1.pop());

}

}

**return** stack2.pop();

}

}

## 8.斐波那契额数列\*

动态规划，第三项等于前一项与后一项之和。

**package** cn.lzw.jianzhioffer.chapter2;

/\*\*题目：大家都知道斐波那契数列，现在要求输入一个整数n，请你输出斐波那契数列的第n项。n<=39

\* 思路：递归

\* 数列 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233，377，610，987，1597，2584，4181，6765，10946，17711，28657，46368........

\* **@author** lzw

\* **@date** 2018年1月3日

\* **@version** version1.0

\*/

**public** **class** Fibonacci {

**public** **int** Fibonacci1(**int** n) {

**if**(n<1){

**return** 0;

}

**if**(n==1||n==2){

**return** 1;

}

**int** cur=1;

**int** last=1;

**for**(**int** i=2;i<n;i++){

**int** tmp=cur;

cur=cur+last;

last=tmp;

}

**return** cur;

}

}

## 9.旋转数组的最小数字\*\*\*

利用二分查找，两个指针，头尾。因为非递减，因此中间的数若比头大，则最小的数在右边，中间的数比尾小，则最小的数在左边，直到两个指针差值为1，最小的数在第二个指针。另外，当三者相同时，应该按照顺序查找

**package** cn.lzw.jianzhioffer.chapter2;

/\*\*题目：把一个数组最开始的若干个元素搬到数组的末尾，我们称之为数组的旋转。 输入一个非递减排序的数组的一个旋转，

\* 输出旋转数组的最小元素。 例如数组{3,4,5,1,2}为{1,2,3,4,5}的一个旋转，该数组的最小值为1。

\* NOTE：给出的所有元素都大于0，若数组大小为0，请返回0。

\* 思路：利用二分查找，两个指针，头尾。因为非递减，因此中间的数若比头大，则最小的数在右边，

\* 中间的数比尾小，则最小的数在左边，直到两个指针差值为1，最小的数在第二个指针。

\* 另外，当三者相同时，应该按照顺序查找。

\*

\* **@author** lzw

\* **@date** 2018年1月3日

\* **@version** version1.0

\*/

**public** **class** MinNumsInRotateAry {

**public** **int** minNumberInRotateArray(**int** [] array) {

**if**(array==**null**||array.length<1){

**return** 0;

}

**int** start=0;

**int** end=array.length-1;

**int** mid=start;

**while**(array[start]>=array[end]){

**if**(end-start==1){

mid=end;

**break**;

}

mid=(start+end)/2;

**if**(array[mid]==array[start]&&array[mid]==array[end]){

**return** MinInOrder(array,start,end);

}

**if**(array[mid]>=array[start]){

start=mid;

}**else** **if**(array[mid]<=array[end]){

end=mid;

}

}

**return** array[mid];

}

**public** **int** MinInOrder(**int**[] ary,**int** start,**int** end){

**int** result=ary[start];

**for**(**int** i=start;i<=end;i++){

**if**(result>ary[i]){

result=ary[i];

**break**;

}

}

**return** result;

}

}

## 10.矩阵中的路径\*\*\*\*

题目：请设计一个函数，用来判断在一个矩阵中是否存在一条包含某字符串所有字符的路径。路径可以从矩阵中的任意一个格子开始，每一步可以在矩阵中向左，向右，向上，向下移动一个格子。如果一条路径经过了矩阵中的某一个格子，则该路径不能再进入该格子。例如 a b c e s f c s a d e e 矩阵中包含一条字符串"bcced"的路径，但是矩阵中不包含"abcb"路径，因为字符串的第一个字符b占据了矩阵中的第一行第二个格子之后，路径不能再次进入该格子。

思路：回溯算法，约束条件是当前的位置在格子范围内，且符合理论要的字符，并且没有访问过，就可以继续上下左右的走。如果找不到就回溯，回溯的时候要把当前的字符的长度减1并且访问标志置为false.要建立一个一样大的visited数组。

**package** cn.lzw.jianzhioffer.chapter2;

/\*\*题目：请设计一个函数，用来判断在一个矩阵中是否存在一条包含某字符串所有字符的路径。

\* 路径可以从矩阵中的任意一个格子开始，每一步可以在矩阵中向左，向右，向上，向下移动一个格子。

\* 如果一条路径经过了矩阵中的某一个格子，则该路径不能再进入该格子。

\* 例如 a b c e s f c s a d e e 矩阵中包含一条字符串"bcced"的路径，但是矩阵中不包含"abcb"路径，

\* 因为字符串的第一个字符b占据了矩阵中的第一行第二个格子之后，路径不能再次进入该格子。

\* 思路：回溯算法，约束条件是当前的位置在格子范围内，且符合理论要的字符，并且没有访问过，就可以继续上下左右的走。

\* 如果找不到就回溯，回溯的时候要把当前的字符的长度减1并且访问标志置为false.

\* 要建立一个一样大的visited数组。

\*

\* **@author** lzw

\* **@date** 2018年1月3日

\* **@version** version1.0

\*/

**public** **class** MatrixHasPath {

**public** **boolean** hasPath(**char**[] matrix, **int** rows, **int** cols, **char**[] str)

{

**if**(matrix==**null**||rows<1||cols<1||str==**null**){

**return** **false**;

}

**boolean**[] visited=**new** **boolean**[rows\*cols];

**int** pathLength=0;

**for**(**int** row=0;row<rows;row++){

**for**(**int** col=0;col<cols;col++){

**if**(hasCore(matrix,rows,cols, str,row,col,pathLength,visited)){

**return** **true**;

}

}

}

**return** **false**;}

**public** **boolean** hasCore(**char**[] matrix, **int** rows, **int** cols, **char**[] str,**int** row,**int** col,**int** pathLength,**boolean**[] visited){

**if**(pathLength==str.length){

**return** **true**;

}

**boolean** hasPath=**false**;

**if**(row>=0&&row<rows

&&col>=0&&col<cols

&&matrix[row\*cols+col]==str[pathLength]

&&!visited[row\*cols+col]){

pathLength++;

visited[row\*cols+col]=**true**;

hasPath=hasCore(matrix,rows,cols, str,row,col-1,pathLength,visited)||

hasCore(matrix,rows,cols, str,row-1,col,pathLength,visited)||

hasCore(matrix,rows,cols, str,row,col+1,pathLength,visited)||

hasCore(matrix,rows,cols, str,row+1,col,pathLength,visited)

;

**if**(!hasPath){

pathLength--;

visited[row\*cols+col]=**false**;

}

}

**return** hasPath;

}

}

## 11机器人的运动范围\*\*\*\*

同样是回溯法，当满足约束条件步数阈值||没有访问过，在格子范围内，就继续左右上下递归。每成功1次计数加1，否则返回0

**package** cn.lzw.jianzhioffer.chapter2;

/\*\*题目：地上有一个m行和n列的方格。一个机器人从坐标0,0的格子开始移动，每一次只能向左，右，上，下四个方向移动一格，

\* 但是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。但是，它不能进入方格（35,38），因为3+5+3+8 = 19。

\* 请问该机器人能够达到多少个格子？

\* 思路：同样是回溯法，当满足约束条件步数阈值||没有访问过，在格子范围内，就继续左右上下递归。

\* 每成功1次计数加1，否则返回0

\*

\* **@author** lzw

\* **@date** 2018年1月3日

\* **@version** version1.0

\*/

**public** **class** MovingCounts {

**public** **int** movingCount(**int** threshold, **int** rows, **int** cols)

{

**if**(rows<1||cols<1){

**return** 0;

}

**int** steps=0;

**boolean**[] visited=**new** **boolean**[rows\*cols];

steps=movingSteps(threshold,rows,cols,0,0,visited);

**return** steps;

}

**public** **int** movingSteps(**int** threshold, **int** rows, **int** cols,**int** row,**int** col,**boolean**[] visited){

**if**(threshold<bitCount(row)+bitCount(col)){

**return** 0;

}

**int** steps=0;

**if**(row>=0&&row<rows&&

col>=0&&col<cols&&

!visited[row\*cols+col]){

visited[row\*cols+col]=**true**;

steps+=1+movingSteps(threshold,rows,cols,row-1,col,visited)+movingSteps(threshold,rows,cols,row,col-1,visited)

+movingSteps(threshold,rows,cols,row+1,col,visited)+movingSteps(threshold,rows,cols,row,col+1,visited);

}

**return** steps;

}

**public** **int** bitCount(**int** num){

**int** sum=0;

**do**{

sum+=num%10;

num=num/10;

}**while**(num!=0);

**return** sum;

}

}

## 12.二进制中1的个数\*

采用无符号右移

**package** cn.lzw.jianzhioffer.chapter2;

/\*\*题目：输入一个整数，输出该数二进制表示中1的个数。其中负数用补码表示。

\* 思路：判断是否为1，用与1操作。然后无符号右移1位。

\* **@author** lzw

\* **@date** 2018年1月3日

\* **@version** version1.0

\*/

**public** **class** NumbersOf1 {

**public** **int** NumberOf1(**int** n) {

**int** cnt=0;

**while**(n!=0){

**if**((n&1)==1){

cnt++;

}

n=n>>>1;

}

**return** cnt;

}

}

# 第三章

## 13.数值的整数方\*

注意指数为负，底数为0

**package** cn.lzw.jianzhioffer.chapter3;

/\*\*题目：给定一个double类型的浮点数base和int类型的整数exponent。求base的exponent次方。

\* 不考虑大数问题

\* 思路：注意指数是0、负数的情况。

\*

\* **@author** lzw

\* **@date** 2018年1月3日

\* **@version** version1.0

\*/

**public** **class** NumPower {

**public** **double** Power(**double** base, **int** exponent) {

**if**(exponent==0){

**return** 1;

}

**double** result;

**if**(exponent<0){

result=*getPower*(base,exponent\*-1);

result=1/result;

}**else**{

result=*getPower*(base,exponent);

}

**return** result;

}

**public** **static** **double** getPower(**double** base,**int** exponent){

**double** result=1.0;

**for**(**int** i=0;i<exponent;i++){

result\*=base;

}

**return** result;

}

}

## 14.打印从1到最大的N位数\*\*\*\*

细节题：1）采用char数组来表示数字

2）过程分为char类型加法和把数字打印出来

3）加法的时候采用char字符相减得到差值与引入进位，每位字符数字的nSum计算，直到第一位的nSum>=10时达到终止条件。

4）打印的时候注意判断前面是否为0，是0就不打印

While(!inCrement(Nums)){

printNum(Num);}

**package** cn.lzw.jianzhioffer.chapter3;

**import** java.util.Scanner;

/\*\*

\* ��ӡ��1������nλ��

\* **@author** Administrator

\* **@date** 2017��12��8��

\* **@version** version1.0

\*/

**public** **class** PrintMaxOfDigits {

**public** **static** **void** main(String[] args) {

Scanner sc=**new** Scanner(System.***in***);

**int** n=sc.nextInt();

*printMaxN*(n);

}

**private** **static** **void** printMaxN(**int** n) {

**if**(n<=0) {

**return**;

}

**char**[] Nums=**new** **char**[n];

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

Nums[i]='0';

}

**while**(!*inCrement*(Nums)) {

*printNums*(Nums);

}

}

**private** **static** **void** printNums(**char**[] nums) {

**boolean** isBegin0=**true**;

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

**if**(nums[i]!='0') {

isBegin0=**false**;

}

**if**(!isBegin0)

{

System.***out***.print(nums[i]);

}

}

System.***out***.println();

}

**private** **static** **boolean** inCrement(**char**[] nums) {

**boolean** overflow=**false**;

**int** overtake=0;

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

**int** nSum=nums[i]-'0'+overtake;

**if**(i==nums.length-1) {

nSum++;

}

**if**(nSum>=10) {

**if**(i==0) {

overflow=**true**;

}**else** {

nSum-=10;

overtake=1;

nums[i]=(**char**) ('0'+nSum);

}

}**else** {

nums[i]=(**char**) ('0'+nSum);

**break**;

}

}

**return** overflow;

}

}

## 15.删除链表的重复节点\*\*\*\*

需要三个节点，first下一节点指向头节点，tmp当前节点，last当前节点的前一节点

**package** cn.lzw.jianzhioffer.chapter3;

/\*\*题目：在一个排序的链表中，存在重复的结点，请删除该链表中重复的结点，重复的结点不保留，返回链表头指针。

\* 例如，链表1->2->3->3->4->4->5 处理后为 1->2->5

\* 思路：定义一个first节点，它的下一个结点指向头结点，防止一开始就重复。最后返回first.next

\* 定义一个节点Last指向重复节点前一个。一旦重复就把后边的都删除。

\*

\* **@author** lzw

\* **@date** 2018年1月3日

\* **@version** version1.0

\*/

**public** **class** DeleteDuplication {

**public** ListNode deleteDuplication(ListNode pHead)

{

ListNode first=**new** ListNode(-1);

first.next=pHead;

ListNode tmp=pHead;

ListNode last=first;

**while**(tmp!=**null** && tmp.next != **null**){

ListNode next=tmp.next;

**if**(next.val==tmp.val){

**int** val=tmp.val;

tmp.next=next.next;

**while**(tmp.next!=**null**&&tmp.next.val==val){

tmp.next=tmp.next.next;

}

last.next=tmp.next;

tmp=last;

}**else**{

last=tmp;

tmp=tmp.next;

}

}

**return** first.next;

}

}

## 16.正则表达式匹配\*\*\*\*

题目：请实现一个函数用来匹配包括'.'和'\*'的正则表达式。模式中的字符'.'表示任意一个字符而'\*'表示它前面的字符可以出现任意次（包含0次）。 在本题中，匹配是指字符串的所有字符匹配整个模式。例如，字符串"aaa"与模式"a.a"和"ab\*ac\*a"匹配，但是与"aa.a"和"ab\*a"均不匹配

思路：首先返回true条件是字符串strIndex和模板数组pIndex的索引都走到尽头，如果模板到尽头而字符串没有返回false判断的时候，先判断第二个字符是不是'\*'，如果是的话 判断是否相等 是的话字符串有可能strIndex+1,pIndex+2，或者strIndex+1,pIndex或者strIndex,pIndex+2。不相等strIndex,pIndex+2正常相等时字符串与模板值相等，或者模板值=='.'注意每次比较前判断索引会不会越界

public boolean match(char[] str, char[] pattern)

public boolean matchCore(char[] str,char[] pattern,int strIndex,int pIndex) 递归return

**package** cn.lzw.jianzhioffer.chapter3;

/\*\*题目：请实现一个函数用来匹配包括'.'和'\*'的正则表达式。模式中的字符'.'表示任意一个字符，

\* 而'\*'表示它前面的字符可以出现任意次（包含0次）。 在本题中，匹配是指字符串的所有字符匹配整个模式。

\* 例如，字符串"aaa"与模式"a.a"和"ab\*ac\*a"匹配，但是与"aa.a"和"ab\*a"均不匹配

\*思路：首先返回true条件是字符串strIndex和模板数组pIndex的索引都走到尽头，如果模板到尽头而字符串没有返回false

\* 判断的时候，先判断第二个字符是不是'\*'，如果是的话 判断是否相等 是的话字符串有可能strIndex+1,pIndex+2，

\* 或者strIndex+1,pIndex或者strIndex,pIndex+2。不相等strIndex,pIndex+2

\* 正常相等时字符串与模板值相等，或者模板值=='.'

\* 注意每次比较前判断索引会不会越界

\*

\* **@author** lzw

\* **@date** 2018年1月3日

\* **@version** version1.0

\*/

**public** **class** ELMatch {

**public** **boolean** match(**char**[] str, **char**[] pattern)

{

**if**(str==**null**||pattern==**null**){

**return** **false**;

}

**return** matchCore(str,pattern,0,0);

}

**public** **boolean** matchCore(**char**[] str,**char**[] pattern,**int** strIndex,**int** pIndex){

**if**(strIndex==str.length&&pIndex==pattern.length){

**return** **true**;

}

**if**(strIndex!=str.length&&pIndex==pattern.length){

**return** **false**;

}

**if**(pIndex+1!=pattern.length&&pattern[pIndex+1]=='\*'){

**if**(strIndex!=str.length&&(str[strIndex]==pattern[pIndex]||pattern[pIndex]=='.')){

**return** matchCore(str,pattern,strIndex+1,pIndex+2)||

matchCore(str,pattern,strIndex+1,pIndex)||

matchCore(str,pattern,strIndex,pIndex+2);

}**else**{

**return** matchCore(str,pattern,strIndex,pIndex+2);

}

}

**if**(strIndex!=str.length&&(str[strIndex]==pattern[pIndex]||pattern[pIndex]=='.')){

**return** matchCore(str,pattern,strIndex+1,pIndex+1);

}

**return** **false**;

}

}

## 17.表示数字的字符串\*\*\*\*

按数字组成结构顺序判断 首先有判断数字的函数，接着判断有无’.’，最后判断有无’e’,’E’,最后如果index==str.length并且前边都为true则返回true.

**package** cn.lzw.jianzhioffer.chapter3;

**public** **class** StrIsNumeric {

**int** index=0;

**public** **boolean** isNumeric(**char**[] str) {

**if**(str==**null**){

**return** **false**;

}

**boolean** numeric=scanNum(str);

**if**(index<str.length&&str[index]=='.'){

index++;

numeric=scanUnsignNum(str)||numeric;//ע������λ��˳��

}

**if**(index<str.length&&(str[index]=='e'||str[index]=='E')){

index++;//�ǵ�ÿ�ζ�Ҫ��һ ��һ������

numeric=numeric&&scanNum(str);//����Ҫ�������

}

**return** numeric&&str.length==index;

}

**public** **boolean** scanNum(**char**[] str){

**if**(str.length>index&&(str[index]=='-'||str[index]=='+')){

index++;

**return** scanUnsignNum(str);

}

**return** scanUnsignNum(str);

}

**public** **boolean** scanUnsignNum(**char**[] str){

**int** before=index;

**while**(str.length>index&&(str[index]>='0'&&str[index]<='9')){//ע��0-9�������������Ҫ������

index++;

}

**return** index>before;

}

}

## 18.调整数组顺序使奇数位于偶数前面