By mnmlist,2015-9-2 9:35:19

>

腾讯面试题目之一

描述：

假如给定一个150个整数大小的整数数组nums[]，整数的范围1~100，请问用最快的速度如何对该数组进行排序？

思路：

基数排序？假设我不用基数排序呢。。。

那我用快排、归并排序和堆排序，时间复杂度都是O(nlogn)，哪个方法更快些呢？归并排序。有什么依据么？好吧，是我猜的，囧~

回去跟同学探讨了一下，其实就是考查用空间换时间罢了。一个很经典的例子，相似的例子比如对一个堆身高数据进行排序，身高肯定就非常集中了50cm-240cm左右吧，不外乎开辟一个长度为241的整数数组count[]，遍历数组，并将count[nums[i]]++;即可，最后输出的时候按count里元素的个数输出相应的身高即可。

代码：

```
public void sortNumers(int nums[])
{
 //assume the scope of  the numbers is between 1 and 100
 if(nums==null||nums.length==0)
  return ;
 int count[]=new int[101];
 for(int i=0;i<nums.length;i++)
  count[nums[i]]++;
 for(int i=1;i<count.length;i++)
 {
  for(int j=0;j<count[i];j++)
   System.out.print(i+" ");
 }
 System.out.println();
}
```

By mnmlist,2015-9-2 9:35:19

leetcode_Palindrome Linked List

描述：

Given a singly linked list, determine if it is a palindrome.

Follow up:
Could you do it in O(n) time and O(1) space?

思路：

1.如何判断一个链表是否是回文的？很简单将链表中的元素遍历出来并放进ArrayList中，然后可以像数组一样来判断该元素是否为回文的，时间复杂度O(n)，空间复杂度O(n)，可如何用O(n)的时间复杂度和O(1)的空间复杂度来解决呢？

2.是不是可以考虑 将链表反转？可反转后还是链表啊，要是将链表分为前后两个部分呢，分为两个部分还是无法判断该链表是否为回文链表啊，那要是再将其中一个链表反转一下呢，It's done!好多时候，多想一步容易，再多想一步就困难了。

代码：

```
public boolean isPalindrome(ListNode head) {
        if(head==null||head.next==null)
            return true;
        ListNode p=head,temp=head,quick=head;
        while(temp!=null&&quick!=null)
        {
            temp=temp.next;
            if(quick.next==null)
                break;
            quick=quick.next.next;
        }
        temp=reverseList(temp);
        p=head;
        while(temp!=null&&p!=null)
        {
            if(temp.val!=p.val)
                return false;
```

By mnmlist,2015-9-2 9:35:19

```
            temp=temp.next;
            p=p.next;
        }
        return true;
    }
    public ListNode reverseList(ListNode head)
    {
        ListNode tempHead=new ListNode(-1);
        ListNode p=head.next,q=null;
        tempHead.next=head;
        head.next=null;
        while(p!=null)
        {
            q=p;
            p=p.next;
            q.next=tempHead.next;
            tempHead.next=q;
        }
        return tempHead.next;
    }
```


leetcode_Jump Game II

描述：

Given an array of non-negative integers, you are initially positioned at the first index of the array.
Each element in the array represents your maximum jump length at that position.
Your goal is to reach the last index in the minimum number of jumps.
For example:
Given array A = [2,3,1,1,4]
The minimum number of jumps to reach the last index is 2. (Jump 1 step from index 0 to 1, then 3 steps to the last index.)

思路：

1.Jump Game思路：和求Max Subarray类似，维护一个当前元素可以跳至的最大值，每循环一次更新reach=Math.max(nums[i]+1,reach),当i>reach或i>=nums.length的时候循环终止，最后看循环是否到达了最后，到达最后则返回true,否则，返回false.

By mnmlist,2015-9-2 9:35:19

2.和Jump Game不同的是，Jump Game II 让求的是跳过所有的元素至少需要几步，这需要维护一个局部变量edge为上一个reach，当i<=reach时，每次仍然通过Math.max(nums[i]+i,reach)获得最大的reach，当i>edge时，只需要更新一个edge为当前reach即可，并将minStep赋值为minStep+1。最后，当到达最后一个元素的时候说明可以到达最后，范围最少的步骤即可。

代码：

```
public int jump(int[] nums)
{
 int edge=0,reach=0;
 int minStep=0,i=0;
 for(;i<nums.length&&i<=reach;i++)
 {
  if(edge<i)
  {
   edge=reach;
   minStep=minStep+1;
  }
  reach=Math.max(nums[i]+i, reach);
 }
 if(i==nums.length)
  return minStep;
 return -1;
}
```


leetcode_Jump Game

描述：

Given an array of non-negative integers, you are initially positioned at the first index of the array.
Each element in the array represents your maximum jump length at that position.
Determine if you are able to reach the last index.
For example:
A = [2,3,1,1,4], return true.
A = [3,2,1,0,4], return false.

思路：

By mnmlist,2015-9-2 9:35:19

和求Max Subarray类似，维护一个当前元素可以跳至的最大值，每循环一次更新reach=Math.max(nums[i]+1,reach),当i>reach或i>=nums.length的时候循环终止，最后看循环是否到达了最后，到达最后则返回true,否则，返回false.

代码：

```
public boolean canJump(int[] nums) {
       int i=0;
       for(int reach=0;i<nums.length&&i<=reach;i++)
       {
        reach=Math.max(nums[i]+i,reach);
       }
       return i==nums.length;
   }
```


leetcode_Decode Ways

描述：

A message containing letters from A-Z is being encoded to numbers using the following mapping:
'A' -> 1
'B' -> 2
...
'Z' -> 26
Given an encoded message containing digits, determine the total number of ways to decode it.
For example,
Given encoded message "12", it could be decoded as "AB" (1 2) or "L" (12).
The number of ways decoding "12" is 2.

思路：

直接从后面开始算，如果str.substring(i,i+2)代表的数值小于26，那

By mnmlist,2015-9-2 9:35:19

么，memo[i]=memo[i+1]+memo[i+2];否则，memo[i]=memo[i+1]，虽然看起来很简单，却是很难想，一般都是从前面开始，然后比较字符的值再做判断，这样的话控制逻辑会非常复杂。

代码：

```
public int numDecodings(String s)
{
 int strLen=s.length();
 if(strLen==0)
  return 0;
 int memo[]=new int[strLen+1];
 memo[strLen]=1;
 memo[strLen-1]=s.charAt(strLen-1)!='0'?1:0;
 for(int i=strLen-2;i>=0;i--)
 {
  if(s.charAt(i)=='0')
   continue;
  else {
   memo[i]=(Integer.parseInt(s.substring(i,i+2)))<=26?memo[i+1]+memo[i+2]:memo[i+1];
  }
 }
 return memo[0];
}
```


leetcode_Maximum Product Subarray

描述：

Find the contiguous subarray within an array (containing at least one number) which has the largest product.
For example, given the array [2,3,-2,4],
the contiguous subarray [2,3] has the largest product = 6.

思路：

1.一种思路是将数组分为被数字0分割的子数组，然后统计负数出现的次数，如果 是偶数，直接取子数组的乘积，如果是奇数，取去掉前面第一个奇数出现之前的 部分或去掉最后一个奇数及其后面数组部分后剩余部分的乘积即可，然后取maxProduct为字数组乘

By mnmlist,2015-9-2 9:35:19

积最大的一段。程序的控制流程很麻烦。

2.每次取部分数组乘积的最大值或最小值，然后跟num[i]做乘法运算，再取最大值或最小值即可，取最大值理所当然，取最小值是因为最小值的绝对值 可能大于最大值，碰到负数就变最大值了。

代码：

```
public int maxProduct(int[] nums)
{
 if (nums == null || nums.length == 0)
  return 0;
 int maxProduct=nums[0];
 int minProduct=nums[0];
 int maxSofar=nums[0];
 int tempMaxProduct,tempMinProduct;
 int numMax,numMin;
 for(int i=1;i<nums.length;i++)
 {
  numMax=maxProduct*nums[i];
  numMin=minProduct*nums[i];
  tempMaxProduct=Math.max(Math.max(numMax,numMin ), nums[i]);
  tempMinProduct=Math.min(Math.min(numMax, numMin), nums[i]);
  maxSofar=Math.max(tempMaxProduct, maxSofar);
  maxProduct=tempMaxProduct;
  minProduct=tempMinProduct;
 }
 return maxSofar;
}
```


leetcode_Sort List

描述：

By mnmlist,2015-9-2 9:35:19

Sort a linked list in O(n log n) time using constant space complexity.

思路：

1.链表的排序一般每遇到过，让用O(nlogn)解决该问题就更不知如何下手了

2.通过参考网上的思路才知道用归并排序，采用递归的方法解决该问题还是可以的，就是理解起来有点费劲

3.重要步骤：递归，归并，查找数组有效范围内的中间节点，有序数组合并

代码：

```
public ListNode sortList(ListNode head)
{
 if(head==null||head.next==null)
  return head;
 ListNode midNode=getMidNode(head);
 ListNode newHead=midNode.next;
 midNode.next=null;
 head=mergeLists(sortList(newHead), sortList(head));
 return head;
}

public ListNode mergeLists(ListNode head1, ListNode head2)
{
 if(head1==null&&head2==null)
  return head1;
 if(head1==null)
  return head2;
 if(head2==null)
  return head1;
 ListNode newHead=new ListNode(-1);
 ListNode curListNode=newHead;
 while(head1!=null&&head2!=null)
 {
  if(head1.val<head2.val)
  {
   curListNode.next=head1;
   head1=head1.next;
  }else
  {
   curListNode.next=head2;
   head2=head2.next;
  }
  curListNode=curListNode.next;
 }
```

By mnmlist,2015-9-2 9:35:19

```
 if(head1!=null)curListNode.next=head1;
 if(head2!=null)curListNode.next=head2;
 return newHead.next;
}

public ListNode getMidNode(ListNode head)
{
 if(head==null||head.next==null)
  return head;
 ListNode p=head,q=head;
 while(q.next!=null&&q.next.next!=null)
 {
  p=p.next;
  q=q.next.next;

 }
 return p;
}
```


leetcode_Integer to Roman

描述：

Given an integer, convert it to a roman numeral.

Input is guaranteed to be within the range from 1 to 3999.

思路：

1.由于罗马数字是字符的形式堆叠而成的，所以不妨将数字的每一位转换成字符并结合相应的量级合成罗马字符串

2.相应的，对于每个字符数字在不考虑量级的情况下大概分三种情况可以解决阿拉伯数字到罗马数字的转换，分(number>=1&&number<=3)、(number>=4&&number<=8)、(number==9)来考虑，具体的实现见代码。

代码：

```
public String intToRoman(int num)
```

By mnmlist,2015-9-2 9:35:19

```
{
 StringBuilder sBuilder=new StringBuilder();
 char chArr[] ={ 'I', 'V', 'X', 'L', 'C', 'D', 'M' };
 int numArr[] ={ 1, 5, 10, 50, 100, 500, 1000 };
 HashMap<Integer,Character> map = new HashMap< Integer,Character>();
 HashMap<Character, Integer> priorityMap = new HashMap<Character, Integer>();
 for (int i = 0; i < chArr.length; i++)
 {
  map.put(numArr[i], chArr[i]);
  priorityMap.put(chArr[i], i);
 }
 String numString=String.valueOf(num);
 int tempNum=1;
 int number=0;
 int numLen=numString.length();
 for(int i=1;i<numLen;i++)
  tempNum*=10;
 char ch='0';
 for(int i=0;i<numLen;i++)
 {
  number=numString.charAt(i)-'0';
  if(number>=1&&number<=3)
  {
   ch=map.get(tempNum);
   for(int j=0;j<number;j++)
    sBuilder.append(ch);
  }
  else if(number>=4&&number<=8)
  {
   ch=map.get(tempNum);
   int pri=priorityMap.get(ch);
   char newChar=chArr[pri+1];
   for(int j=0;j<5-number;j++)
    sBuilder.append(ch);
   sBuilder.append(newChar);
   for(int j=6;j<=number;j++)
    sBuilder.append(ch);

  }else if(number==9)
  {
   ch=map.get(tempNum);
   int pri=priorityMap.get(ch);
   char newChar=chArr[pri+2];
   sBuilder.append(ch);
   sBuilder.append(newChar);
  }
  tempNum/=10;
 }

 return sBuilder.toString();
}
```

By mnmlist,2015-9-2 9:35:19

leetcode_Roman to Integer

描述：

Given a roman numeral, convert it to an integer.

Input is guaranteed to be within the range from 1 to 3999.

思路：

1.就像脑筋急转弯一样，这一题还真的搞了好久，主要是没想明白思路，现在再回过头去看代码，显而易见，或许这才是制造和创造的区别吧

2.大致过程就是一个循环，当currentChar的量级小于nextChar的量级时，直接加上nextChar的量级和currentChar的量级之差即可，其它的情况直接加上currentChar的量级，罗马数字就类似字符串的形式

3.另：这也就是有计算机才可以这么算，这对普通人来说得多麻烦啊，罗马数字很啰嗦。

代码：

```
public int romanToInt(String s) {
      char chArr[]={'I','V','X','L','C','D','M','i','v','x','l','c','d','m'};
       int numArr[]={1,5,10,50,100,500,1000,1,5,10,50,100,500,1000};
       HashMap<Character, Integer>map=new HashMap<Character, Integer>();
       for(int i=0;i<chArr.length;i++ )
        map.put(chArr[i], numArr[i]);
       int j=0;
       int strLen=s.length();
       char curCh,nextCh;
       int sum=0;
       for(int i=0;i<strLen;i++)
       {
        curCh=s.charAt(i);
        j=i+1;
        if(j<strLen)
        {
         nextCh=s.charAt(j);
         if(map.get(curCh)<map.get(nextCh))//
         {
          sum+=map.get(nextCh)-map.get(curCh);
          i=j;//
         }else {
```

By mnmlist,2015-9-2 9:35:19

```
            sum+=map.get(curCh);
    }

         }else {
    sum+=map.get(curCh);//
   }

        }
        return sum;
    }
```


leetcode_Repeated DNA Sequences

描述：

All DNA is composed of a series of nucleotides abbreviated as A, C, G, and T, for example: "ACGAATTCCG". When studying DNA, it is sometimes useful to identify repeated sequences within the DNA.

Write a function to find all the 10-letter-long sequences (substrings) that occur more than once in a DNA molecule.

For example,

> Given s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT",
>
> Return:
> ["AAAAACCCCC", "CCCCCAAAAA"].

思路：

1.很显然，暴力求解也是一种方法，尽管该方法是不可能的。

2.我们首先来看字母 " A" "C" "G" "T" 的ASCII码，分别是65, 67, 71, 84，二进制表示为 1000001, 1000011, 1000111, 1010100。可以看到它们的后三位是不同，所以用后三位就可以区分这四个字母。一个字母用3bit来区分，那么10个字母用30bit就够了。用int的第29~0位分表表示这0~9个字符，然后把30bit转化为int作为这个子串的key，放入到HashTable中，以判断该子串是否出现过。

代码：

By mnmlist,2015-9-2 9:35:19

```
public List<String> findRepeatedDnaSequences(String s)
{
 List<String>list=new ArrayList<String>();
 int strLen=s.length();
 if(strLen<=10)
  return list;
 HashMap<Integer, Integer>map=new HashMap<Integer,Integer>();
 int key=0;
 for(int i=0;i<strLen;i++)
 {
  key=((key<<3)|(s.charAt(i)&0x7))&0x3fffffff;//k<<3,key3
  //s.charAt(i)&0x7)s.charAt(i)3
  //&0x3fffffffkey
  if(i<9)continue;
  if(map.get(key)==null)//map
   map.put(key, 1);
  else if(map.get(key)==1)//list
  {
   list.add(s.substring(i-9,i+1));
   map.put(key, 2);//
  }
 }
 return list;
}
```


leetcode_Product of Array Except Self

描述：

Given an array of n integers where n > 1, `nums`, return an array `output` such that `output[i]` is equal to the product of all the elements of`nums` except `nums[i]`.

Solve it without division and in O(n).

For example, given `[1,2,3,4]`, return `[24,12,8,6]`.

Follow up:
Could you solve it with constant space complexity? (Note: The output array does not count as extra space for the purpose of space complexity analysis.)

思路：

1.该题目的要求返回一个数组，该数组的product[i]=num[0]*num[1]*.....num[i-1]*num[i+1]*...*num[num.length-1];

By mnmlist,2015-9-2 9:35:19

2.由于要求出除i位置的元素num[i]的其它所有元素的乘积，假如每次都这么循环遍历一次并作乘法运算，当到num[i]的时候跳过，这样时间复杂度就是O(n*n)

3.另外一种思路就是求出所有的元素的乘积productTotal，然后具体求某个product[i]时，直接product[i]=productTotal/num[i]

4.但3中的思路有一个问题，就是productTotal有可能溢出，为处理这种情况，将productTotal初始化为long类型，OK！

代码：

```
public int[] productExceptSelf(int[] nums)
{
 if (nums == null || nums.length == 0)
  return nums;
 long result = 1;
 for (int num : nums)
  result *= num;
 if(result!=0)
 {
  for (int i = 0; i < nums.length; i++)
   nums[i] = (int) (result / nums[i]);
 }else
 {
  int newArr[]=new int[nums.length];
  newArr=Arrays.copyOf(nums, nums.length);
  for (int i = 0; i < nums.length; i++)
  {
   if(newArr[i]!=0)
    nums[i]=0;
   else
   {
    int tempProduct=1;
    for(int j=0;j<nums.length;j++)
    {
     if(j==i)
      continue;
     else
      tempProduct*=newArr[j];

    }
    nums[i]=tempProduct;
   }
  }
 }
 return nums;
```

```
By mnmlist,2015-9-2 9:35:19
}
```


leetcode_Combination Sum III

描述：

Find all possible combinations of k numbers that add up to a number n, given that only numbers from 1 to 9 can be used and each combination should be a unique set of numbers.

Ensure that numbers within the set are sorted in ascending order.

Example 1:

Input: k = 3, n = 7

Output:

```
[[1,2,4]]
```

Example 2:

Input: k = 3, n = 9

Output:

```
[[1,2,6], [1,3,5], [2,3,4]]
```

思路：

1.和Combination Sum II思路类似，只是在其基础上判断一下当有满足条件的结果时，其序列长度是否为k

代码：

By mnmlist,2015-9-2 9:35:19

```
public List<List<Integer>> combinationSum3(int k, int n) {
 int candidates[]={1,2,3,4,5,6,7,8,9};
 List<List<Integer>>listResult=new ArrayList<List<Integer>>();
 ArrayList<Integer>list=new ArrayList<Integer>();
 Arrays.sort(candidates);
 combination(listResult, list, candidates, k,n,0);
 return listResult;
}
public void combination(List<List<Integer>>listResult,
  ArrayList<Integer>list,int[] candidates,int k, int target,int begin)
{
 if(target==0)
 { if(list.size()==k)<span style="font-size: 13.3333339691162px; font-family: Arial, Helv
   listResult.add(list);
  return;
 }
 for(int i=begin;i<candidates.length&&target>=candidates[i];i++)
 {
  if(i==begin||candidates[i]!=candidates[i-1])//<span style="font-size: 13.3333339691162p
  {
   ArrayList<Integer>copy=new ArrayList<Integer>(list);
   copy.add(candidates[i]);
   combination(listResult, copy, candidates,k,target- candidates[i],i+1);
  }
 }
}
```


leetcode_Combination Sum II

描述：

Given a collection of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums toT.

Each number in C may only be used once in the combination.

Note:

- All numbers (including target) will be positive integers.
- Elements in a combination (a1, a2, … , ak) must be in non-descending order. (ie, a1 ≤ a2 ≤ … ≤ ak).
- The solution set must not contain duplicate combinations.

For example, given candidate set 10,1,2,7,6,1,5 and target 8,
A solution set is:
[1, 7]

By mnmlist,2015-9-2 9:35:19

[1, 2, 5]
[2, 6]
[1, 1, 6]

思路：

1.基本思路和Combination Sum类似。

2.Combination Sum的思路：这种题目一般要用到递归或回溯两种方法，用回溯法试过，代码规模总是越来越庞大，但最终还是没能通过所有的测试用例，^-^!用递归的话这题目看着要容易理解的多，每递归一次target要变为target=target-candidates[i]，并将开始index赋值为i，当target==0时，条件满足，如果target<candidates[i]，这轮循环结束，方法出栈，当前的i++，继续循环。

3.作为Combination Sum的升级版本，本题需要做的就是对于排序后的数组同样的数组元素，不会在重复的数组元素上做相同的求Combination Sum的运算，这可以通过(i==begin||candidates[i]!=candidates[i-1])来保证。

代码：

```
public List<List<Integer>> combinationSum2(int[] candidates, int target)
{
 List<List<Integer>>listResult=new ArrayList<List<Integer>>();
 ArrayList<Integer>list=new ArrayList<Integer>();
 Arrays.sort(candidates);
 combination(listResult, list, candidates, target,0);
 return listResult;

}
public void combination(List<List<Integer>>listResult,
  ArrayList<Integer>list,int[] candidates, int target,int begin)
{
 if(target==0)
 {
  listResult.add(list);
  return;
 }
 for(int i=begin;i<candidates.length&&target>=candidates[i];i++)
 {
  if(i==begin||candidates[i]!=candidates[i-1])
  {
   ArrayList<Integer>copy=new ArrayList<Integer>(list);
   copy.add(candidates[i]);
   combination(listResult, copy, candidates,target- candidates[i],i+1);
  }
 }
}
```

By mnmlist,2015-9-2 9:35:19

leetcode_Combination Sum

描述：

Given a set of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

The same repeated number may be chosen from C unlimited number of times.

Note:

- All numbers (including target) will be positive integers.
- Elements in a combination (a1, a2, … , ak) must be in non-descending order. (ie, a1 ≤ a2 ≤ … ≤ ak).
- The solution set must not contain duplicate combinations.

For example, given candidate set `2,3,6,7` and target `7`,
A solution set is:
`[7]`
`[2, 2, 3]`

思路：

这种题目，一般要用到递归或回溯两种方法，用回溯法试过，代码规模总是越来越庞大，但最终还是没能通过所有的测试用例，^-^!

用递归的话这题目看着要容易理解的多，每递归一次target要变为target=target-candidates[i]，并将开始index赋值为i，当target==0时，条件满足，如果target<candidates[i]，这轮循环结束，方法出栈，当前的i++，继续循环。

总之，用递归来解决问题，难想到，也不太容易被看懂。

代码：

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

By mnmlist,2015-9-2 9:35:19

```
 List<List<Integer>>listResult=new ArrayList<List<Integer>>();
 ArrayList<Integer>list=new ArrayList<Integer>();
 Arrays.sort(candidates);
 combination(listResult, list, candidates, target,0);
 return listResult;

}
public void combination(List<List<Integer>>listResult,
  ArrayList<Integer>list,int[] candidates, int target,int begin)
{
 if(target==0)
 {
  listResult.add(list);
  return;
 }
 for(int i=begin;i<candidates.length&&target>=candidates[i];i++)
 {
  ArrayList<Integer>copy=new ArrayList<Integer>(list);
  copy.add(candidates[i]);
  combination(listResult, copy, candidates,target- candidates[i],i);
 }
}
```


leetcode_Majority Element II

描述：

Given an integer array of size n, find all elements that appear more than ? n/3 ? times. The algorithm should run in linear time and in O(1) space.

思路：

1.跟让求一个数组中出现次数超过一半的元素类似，这次让求出现次数超过n/3次数的元素，原则上是类似的，但难度要大好多

2.因为是类似的，所以总体思路应该还是一样的，但具体怎么做呢？这次维护一个curNum1和curNum2两个锚点并用count1和count2来计算锚点出现的次数，因为求超过n/3次数的元素，所以数组中可能存在两个这样的元素，其它的就和Majority Element类似了，只有nums[i]与curNum1和curNum2均不相等时才会count1--，count2--

By mnmlist,2015-9-2 9:35:19

3.和Majority Element不同，最后剩下的curNum1和curNum2有可能不是出现次数超过n/3的那个数，所以还需要统计下curNum1和curNum2出现的真实次数，但这并不等于说会漏掉出现次数超过n/3的数字

代码：

```
public List<Integer> majorityElement(int[] nums) {
 List<Integer>list=new ArrayList<Integer>();
 if(nums==null)
  return list;
       int count1=0,curNum1=0;
       int count2=0,curNum2=1;
       for(int num:nums)
       {
        if(num==curNum1)
         count1++;
        else if(num==curNum2)
         count2++;
        else if(count1==0)
        {
         curNum1=num;
         count1=1;
        }else if(count2==0)
        {
         curNum2=num;
         count2=1;
        }else {
   count1--;
   count2--;
  }
       }
       count1=0;
       count2=0;
       for(int num:nums)
       {
        if(num==curNum1)
         count1++;
        else if(num==curNum2)
         count2++;
       }
       if(count1>nums.length/3)list.add(curNum1);
       if(count2>nums.length/3)list.add(curNum2);
       return list;
   }
```

By mnmlist,2015-9-2 9:35:19

leetcode_Implement Stack using Queues

描述：

Implement the following operations of a stack using queues.

push(x) -- Push element x onto stack.
pop() -- Removes the element on top of the stack.
top() -- Get the top element.
empty() -- Return whether the stack is empty.
Notes:
You must use only standard operations of a queue -- which means only push to back, peek/pop from front, size, and is empty operations are valid.
Depending on your language, queue may not be supported natively. You may simulate a queue by using a list or deque (double-ended queue), as long as you use only standard operations of a queue.
You may assume that all operations are valid (for example, no pop or top operations will be called on an empty stack).

思路：

1.跟用栈实现队列不同，我感觉用队列去实现栈要困难的多，以至于根本就想不起来，参考了网络上的思路才算是有了写头绪，原来是这个这个样子。。。

2.如果用栈来实现队列还算可以理解的话，但用队列来实现栈就只有两个字来形容:no zuo no die!，下面我就来描述下这种奇葩的思路：

3.用两个队列queue1和queue2来模拟栈，具体怎么模拟呢？queue1是操作队列，先进先出，queue2是中转队列，每次取元素时，将0~size-2个元素先中转到queue2中，然后取出queue1的最后一个元素，然后，对，然后在将queue1中的元素再依次放回queue2中，直至stack2为空且没有新的元素进栈，循环终止

4.好有想法的一个思路，希望还有更简洁明快的思路出现。

代码：

By mnmlist,2015-9-2 9:35:19

```
class MyStack {
   Deque<Integer>queue1=new LinkedList<Integer>();
Deque<Integer>queue2=new LinkedList<Integer>();
// Push element x onto stack.
public void push(int x)
{
 queue1.addLast(x);;
}

// Removes the element on top of the stack.
public void pop()
{
 if(!this.empty())
 {
  int size=queue1.size()-1;
  for(int i=0;i<size;i++)
   queue2.addLast(queue1.pop());
  queue1.pop();
 }
 queue1.addAll(queue2);
 queue2.clear();
}

// Get the top element.
public int top()
{
 int num=0;
 if(!this.empty())
 {
  int size=queue1.size()-1;
  for(int i=0;i<size;i++)
   queue2.addLast(queue1.pop());
  num=queue1.pop();
  queue1.addAll(queue2);
  queue1.addLast(num);
  queue2.clear();
 }
 return num;
}

// Return whether the stack is empty.
public boolean empty()
{
 if(queue1.size()==0)
  return true;
 return false;
}
}
```

By mnmlist,2015-9-2 9:35:19

leetcode_Majority Element

描述：

Given an array of size n, find the majority element. The majority element is the element that appears more than ? n/2 ? times.
You may assume that the array is non-empty and the majority element always exist in the array.

思路：

1.对于这种问题，相信很多人都会想起来直接对数组拍下序，然后取中间值即可return nums[nums.length/2];地球人都知道，哈哈。。。Time:O(n(log(n)),Space:O(k)

2.啥？还有更好的方法？对，其实还真有，先找第一个数字curNum作为锚，count统计curNum目前为止出现的次数，依次遍历数组nums[i],如果
nums[i]==curNum，count++；如果不等呢，count--，此时若count==0；再换锚点为curNum=nums[i]，并令count==1

3.重复2中的步骤直至遍历完所有的数组元素，那么最后剩下的nagecurNum就是在数组中出现次数大于一半的数字。

代码：

```
public int majorityElement(int[] nums) {
       int count=1,curNum=nums[0];
       for(int i=1;i<nums.length;i++)
       {
           if(nums[i]==curNum)
               count++;
           else
           {
               count--;
               if(count==0)
               {
                   curNum=nums[i];
                   count=1;
               }
```

By mnmlist,2015-9-2 9:35:19

```
            }
        }
        return curNum;
    }
```


leetcode_Summary Ranges

描述：

Given a sorted integer array without duplicates, return the summary of its ranges.

For example, given `[0,1,2,4,5,7]`, return `["0->2","4->5","7"].`

思路：

类似字符串的压缩，就是统计下连续数组元素并判断该数组元素是否连续递增，将连续递增的系列元素用startNum->endNum进行压缩，当然对于孤立的数字点，直接原样压缩即可。

代码：

```
public List<String> summaryRanges(int[] nums) {
        List<String>listStr=new ArrayList<String>();
        if(nums==null||nums.length==0)
            return listStr;
        int start=0,end=0,i=0;
        while(i<nums.length)
        {
            start=nums[i];
            while(i+1<nums.length&&nums[i+1]-nums[i]==1)
                i=i+1;
            end=nums[i];
            if(start==end)
                listStr.add(String.valueOf(start));
            else
```

```
By mnmlist,2015-9-2 9:35:19
                listStr.add(""+start+"->"+end);//jdk1.7StringBuilderZzz
            i++;
        }
        return listStr;

    }
```


leetcode_Implement Queue using Stacks

描述：

Implement the following operations of a queue using stacks.

- push(x) -- Push element x to the back of queue.
- pop() -- Removes the element from in front of queue.
- peek() -- Get the front element.
- empty() -- Return whether the queue is empty.

Notes:

- You must use only standard operations of a stack -- which means only push to top, peek/pop from top, size, and is emptyoperations are valid.
- Depending on your language, stack may not be supported natively. You may simulate a stack by using a list or deque (double-ended queue), as long as you use only standard operations of a stack.
- You may assume that all operations are valid (for example, no pop or peek operations will be called on an empty queue).

思路：

1.用栈来实现一个队列，也就是用后进先出的栈实现先进先出的队列

2.这个还是很难想的，但总之还是比用队列来实现栈容易想，大概就是用两个栈stack1和stack2来模拟队列

3.所有的元素都从stack1进栈，所有元素都从stack2出栈，当stack2为空的时候，将stack1中的所有元素出栈并全部push到stack2中去，然后从stack2出栈

By mnmlist,2015-9-2 9:35:19

4.由于栈是后进先出的，两次后进先出的操作就实现了队列的功能

代码：

```
class MyQueue {
   Stack<Integer>stack1=new Stack<Integer>();
Stack<Integer>stack2=new Stack<Integer>();
// Push element x to the back of queue.
   public void push(int x) {
       stack1.push(x);
   }

   // Removes the element from in front of queue.
   public void pop() {
       if(!stack2.empty())
        stack2.pop();
       else
       {
  while(!stack1.empty())
  {
   stack2.push(stack1.pop());
  }
  stack2.pop();
 }
   }

   // Get the front element.
   public int peek() {
    int num=0;
    if(!stack2.empty())
        num= stack2.peek();
       else
       {
  while(!stack1.empty())
  {
   stack2.push(stack1.pop());
  }
  num= stack2.peek();
 }
       return num;
   }

   // Return whether the queue is empty.
   public boolean empty() {
       if(stack1.empty()&&stack2.empty())
        return true;
       return false;
   }
}
```

By mnmlist,2015-9-2 9:35:19

leetcode_Invert Binary Tree

描述：

Invert a binary tree.

```
  4
 / \
 2  7
/\ /\
1 3 6 9
```

to

```
  4
 / \
 7  2
/\ /\
9 6 3 1
```

Trivia:
This problem was inspired by this original tweet by Max Howell:

> Google: 90% of our engineers use the software you wrote (Homebrew), but you can’ t invert a binary tree on a whiteboard so fuck off.

思路：

1.记得做树的问题时，有两题类似的，一题是判断二叉树是否镜像对称，另外一题是判断两个二叉树节点的值是否相等，同样都是用递归的方式来做

2.这一题跟镜像对称哪一题类似，从上至下，交换同一个父节点的左右子节点即可。

代码：

By mnmlist,2015-9-2 9:35:19

```
public TreeNode invertTree(TreeNode root) {
       if(root==null)
           return root;

       if(root.left!=null||root.right!=null)
       {
        LinkedList<TreeNode> list=new LinkedList<TreeNode>();
        list.addLast(root);
        TreeNode curNode=null;
           while(!list.isEmpty())
           {
            curNode=list.removeFirst();
            changeTree(curNode);
            if(curNode.left!=null)
             list.addLast(curNode.left);
            if(curNode.right!=null)
             list.addLast(curNode.right);
           }
       }
       return root;
   }

public void changeTree(TreeNode root)
{
 if (root.left != null || root.right != null)
 {
  TreeNode temp = root.left;
  root.left = root.right;
  root.right = temp;
 }
}
```


leetcode_Kth Smallest Element in a BST

描述：

Given a binary search tree, write a function kthSmallest to find the kth smallest element in it.
Note:
You may assume k is always valid, 1 ≤ k ≤ BST's total elements.
Follow up:

By mnmlist,2015-9-2 9:35:19

What if the BST is modified (insert/delete operations) often and you need to find the kth smallest frequently? How would you optimize the kthSmallest routine?
Hint:
Try to utilize the property of a BST.Show More Hint

思路：

1.一般涉及到树的问题用树的遍历方式来解决的可能性比较大，或者就是递归了

2.对于本题，一种很朴素的解决方式就是不是二叉查找树么，中序遍历时不就是一个有序的序列么，直接上中序遍历，遍历到第K个数即是本题要求的Kth Smallest Element in a BST

代码：

```
public int kthSmallest(TreeNode root, int k) {
  int count=0;
     Stack<TreeNode>st=new Stack<TreeNode>();
     st.push(root);
     TreeNode top=null;
     while(!st.empty())
     {
      top=st.peek();
      while(top.left!=null)
      {
       st.push(top.left);
       top=top.left;
      }
      while(top.right==null)
      {
       count++;
       if(count==k)
          return top.val;
       st.pop();
       if(!st.empty())
        top=st.peek();
       else
        break;
      }
      if(!st.empty())
      {
       count++;
       if(count==k)
          return top.val;
       st.pop();
       st.push(top.right);
```

```
By mnmlist,2015-9-2 9:35:19
```

```
      }

     }
     return -1;
   }
```


leetcode_Palindrome Linked List

描述：

Given a singly linked list, determine if it is a palindrome.
Follow up:
Could you do it in O(n) time and O(1) space?

思路：

1.有大概两种思路，第一种将linkedLIst表中的数据读取到一个ArrayList中，然后从ArrayList前后开始比较即可。Time:O(n),Space:O(n)

2.第二种用快慢指针找到中间节点，然后将链表断成两个表，并将后面的链表逆转，然后顺序比较两个链表即可。Time:O(n),Space:O(1)

3.显然只有第二种方法符合题意。

代码：

```
public boolean isPalindrome(ListNode head) {
       if(head==null||head.next==null)
           return true;
       ListNode p=head,temp=head,quick=head;
       while(temp!=null&&quick!=null)
```

By mnmlist,2015-9-2 9:35:19

```
        {
            temp=temp.next;
            if(quick.next==null)
                break;
            quick=quick.next.next;
        }
        temp=reverseList(temp);
        p=head;
        while(temp!=null&&p!=null)
        {
            if(temp.val!=p.val)
                return false;
            temp=temp.next;
            p=p.next;
        }
        return true;
    }
    public ListNode reverseList(ListNode head)
    {
        ListNode tempHead=new ListNode(-1);
        ListNode p=head.next,q=null;
        tempHead.next=head;
        head.next=null;
        while(p!=null)
        {
            q=p;
            p=p.next;
            q.next=tempHead.next;
            tempHead.next=q;
        }
        return tempHead.next;
    }
```


leetcode_Power of Two

描述：

Given an integer, write a function to determine if it is a power of two.

By mnmlist,2015-9-2 9:35:19

思路：

1.很有意思的一道题目，大致有三种方法：

2.第一种对num进行向右移位，每次移动一个Bit位并和0x1作&运算，共需移位31次，统计num中1出现的次数，有且仅出现一次的话就返回true

3.第二种方法是令tempNum=1，让num和tempNum作&运算，每次移位结束将tempNum左移位，工需移位31次，1有且仅出现一次则返回true

4.第三种方法有点惨无人道，boolean flag=((num&(num-1))==0);flag为真的话则返回true

代码：

仅贴出第三种方法的代码：

```
public boolean isPowerOfTwo(int n)
{
    if(n<=0)
        return false;
    return (n&(n-1))==0;
}
```


leetcode_Median of Two Sorted Arrays

描述：

There are two sorted arrays nums1 and nums2 of size m and n respectively. Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

By mnmlist,2015-9-2 9:35:19

思路：

1.题意大概为查找两个数组 的中位数，这个不难，O（N）的时间复杂度就可以解决，但题目的要求是用 O(log (m+n))的时间复杂度来解决该问题！

2.由于时间复杂度为 O(log (m+n))，一般会用到二分法，每次取数组nums1和nums2所剩下的部分的中间值mid1和mid2，然后比较两个数组的中间值大小，较大的一个取数组较小的一部分，较小的mid数组取较大的一部分。由于每次都要多虑到一部分不符合条件的，所以刚开始要求第k=（len1+len2)/2个数的k每次都要更新。

3.重负步骤2，直至每个数组剩下的有效部分为0，则另一个数组的nums[start+k]即为中位数；另外一种可能是最后当k==0时，两个数组有效部分的开始的那个数肯定为要求的数。

哎，表述能力不行啊，自己都快被绕晕了(^-^)!

代码：

```
public double findMedianSortedArrays(int[] nums1, int[] nums2) {
       if((nums1==null||nums1.length==0)&&(nums2==null||nums2.length==0))
           return 0;
       if(nums1==null||nums1.length==0)
           return getMidOfOneArray(nums2);
       if(nums2==null||nums2.length==0)
           return getMidOfOneArray(nums1);
 int totalLen = nums1.length + nums2.length;
 if ((totalLen & 0x1) == 1)
  return findMid(nums1, 0, nums1.length - 1, nums2, 0,
    nums2.length - 1, totalLen / 2);
 else
  return (findMid(nums1, 0, nums1.length - 1, nums2, 0,
    nums2.length - 1, totalLen / 2) + findMid(nums1, 0,
    nums1.length - 1, nums2, 0, nums2.length - 1,
    totalLen / 2 - 1)) * 0.5;
}

public double findMid(int nums1[], int start1, int end1, int nums2[],
  int start2, int end2, int k) {
 int aLen = end1 - start1 + 1;
 int bLen = end2 - start2 + 1;
 if (aLen == 0)
  return nums2[start2 + k];
 if (bLen == 0)
  return nums1[start1 + k];
 if (k == 0)
  return nums1[start1] < nums2[start2] ? nums1[start1]
    : nums2[start2];
 int aMid = aLen * k / (aLen + bLen);
```

By mnmlist,2015-9-2 9:35:19

```
 int bMid = k - aMid - 1;
 aMid = aMid + start1;
 bMid = bMid + start2;
 if (nums1[aMid] > nums2[bMid]) {
  k = k - (bMid - start2 + 1);
  end1 = aMid;
  start2 = bMid + 1;
 } else {
  k = k - (aMid - start1 + 1);
  end2 = bMid;
  start1 = aMid + 1;
 }
 return findMid(nums1, start1, end1, nums2, start2, end2, k);
}
public double getMidOfOneArray(int arr[])
{
    if((arr.length&0x1)==0)
        return (arr[arr.length/2]+arr[arr.length/2-1])*0.5;
    return arr[arr.length/2];
}
```


leetcode_Valid Anagram

描述：

Given two strings s and t, write a function to determine if t is an anagram of s.

For example,
s = "anagram", t = "nagaram", return true.
s = "rat", t = "car", return false.

Note:
You may assume the string contains only lowercase alphabets.

思路：

百度的意思anagram：由颠倒字母顺序而构成的字。

有两种解决方案，一个对字符串中的字符进行排序并判断排序后的字符串序列是否相同，另外一种方法是直接统计所有字符出现的次数并判断所有字符出现的次数是否相同，相同则返回true

By mnmlist,2015-9-2 9:35:19

1.将字符串中字符进行排序肯定涉及到String转换位charArray，然后用nlogn的算法对字符进行排序，既用到额外存储空间，用得话费nlogn的时间

2.直接生成一个26个整型数组，统计所有字符出现的次数并比较各个字符出现的次数是否相同。O（k）的空间复杂度，O（n）的时间复杂度

3.综上所述，方法2更胜一筹。

代码：

```
public boolean isAnagram(String s, String t) {
       if(s==null&&t==null)
           return true;
       if(s==null||t==null)
           return false;
       int lens=s.length();
       int lent=t.length();
       if(lens!=lent)
           return false;
       int charCount1[]=new int[26];
       int charCount2[]=new int[26];
       for(int i=0;i<lens;i++)
       {
           ++charCount1[s.charAt(i)-'a'];
           ++charCount2[t.charAt(i)-'a'];
       }
       for(int i=0;i<26;i++)
       {
           if(charCount1[i]!=charCount2[i])
               return false;
       }
       return true;

   }
```


leetcode_Sqrt(x)

描述：

By mnmlist,2015-9-2 9:35:19

Implement int sqrt(int x).
Compute and return the square root of x.

思路：

按题意来说是实现一个api方法，对整数开根号

1.用二分的方法来对一个整数开根号，每次循环求start和end之间的值mid来计算，若mid*mid>x，则end变为mid，若mid*mid<x,则start取mid，直至mid*mid=x

2.对于1中的思路有两个不足的地方，假如x为99呢，mid*mid永远不可能为99，因为sqrt(99)==9，而当mid变的很大时mid*mid直接就溢出了

3.对于2.种提到的不足之处，可用(mid==start||mid==end)来弥补，当mid==start或mid==end任何一个条件满足即可。而mid*mid溢出的问题可以用x/mid mid来解决

4.对整数开根号能不能升级为对float、double类型的数据开根号呢，当然可以，判断条件只需要x/mid-mid小于某个数(也就是精度)即可。

代码：

Demo版本：

```
public int mySqrt(int x)
{
 if (x < 0)
  return -1;
 int start = 0, end = x, mid=x/2;
 if(x*x==x)//to judge the number 1
  return x;
 long tempNum = 0,tempMid=0;
 while (mid!= start && mid!= end)
 {
  tempMid=mid;
  tempNum=tempMid*tempMid;
  if (tempNum<x)
  {
   start=mid;
```

By mnmlist,2015-9-2 9:35:19

```
   mid =(mid+ end) / 2  ;
  }
  else if (tempNum>x)
  {
   end=mid;
   mid =(mid+ start) / 2 ;
  }else
   break;
 }
 return mid;
}
```

改进版本：

```
public int mySqrt(int x)
{
 if (x < 0)
  return -1;
 int start = 0, end = x, mid=x/2;
 if(x*x==x)//to judge the number 1
  return x;
 int tempNum = 0,tempMid=0;
 while (mid!= start && mid!= end)
 {
  tempMid=mid;
  tempNum=x/tempMid-tempMid;
  if (tempNum>0)
  {
   start=mid;
   mid =(mid+ end) / 2  ;

  }
  else if (tempNum<0)
  {
   end=(int)mid;
   mid =(mid+ start) / 2 ;
  }else if(tempNum==0) {
   break;
  }

 }
 return mid;
}
```

By mnmlist,2015-9-2 9:35:19

leetcode_Count and Say

描述：

The count-and-say sequence is the sequence of integers beginning as follows:
1, 11, 21, 1211, 111221, ...

1 is read off as "one 1" or 11.
11 is read off as "two 1s" or 21.
21 is read off as "one 2, then one 1" or 1211.

Given an integer n, generate the nth sequence.

Note: The sequence of integers will be represented as a string.

思路：

很有意思的一个题目，由于题目描述的比较简单，刚开始竟然没有读懂。。。

1.题目的意思是给你一个数n，让你输出前n个按照给定规则生成的数字串

2.而生成该数字串的规则为后一个串是前一个串的读法

3.而字符串是如何读的呢？是将该数字串种连续相同的数字进行压缩编码即111读作3个1，即31,11读作2个1，即21，以此类推。

代码：

```
public String countAndSay(int n)
{
 if (n <= 0)
  return new String("");
 String strNum = "1";
 for (int i = 1; i < n; i++)//"1"
 {
  strNum = getString(strNum);//
 }
 return strNum;
```

```
By mnmlist,2015-9-2 9:35:19
}

public String getString(String strNum)
{
 int numLen = strNum.length();
 StringBuilder sb = new StringBuilder();
 int indexStart = 0, indexEnd = 0;
 char ch;
 while (indexStart < numLen)//
 {
  ch = strNum.charAt(indexStart);
  indexEnd = indexStart + 1;
  while (indexEnd < numLen && ch == strNum.charAt(indexEnd))
   indexEnd++;
  sb.append(indexEnd - indexStart);
  sb.append(ch);
  indexStart = indexEnd;
 }
 return sb.toString();
}
```


leetcode_Rectangle Area

描述：

Find the total area covered by two rectilinear rectangles in a 2D plane.

Each rectangle is defined by its bottom left corner and top right corner as shown in the figure.

By mnmlist,2015-9-2 9:35:19

Assume that the total area is never beyond the maximum possible value of int.

思路：

1.本题目的意思是，求totalAreage=areage(A)||areage(B)，就是求两个矩形的并集。

2.这题有点绕，解决问题的关键是求出两个矩形相交部分的面积，然后两个矩形的面积之和减去两个矩形相交的部分即是最后要求的结果

3.但是如何求两个矩形相交的部分呢，假设两个矩形相交，那就要求两个矩形相交部分的左边的最大值，右边的最小值，上边的最小值，下边的最大值

4.然后再根据矩形相交时的位置（比如左上右下方向，左下右上等来判断要减去的面积）

代码：

```
public int computeArea(int A, int B, int C, int D, int E, int F, int G,
  int H)
{
 int totalArea = (D - B) * (C - A) + (H - F) * (G - E);
 int left, bottom, top, right;
 left = getLeft(A, E);
 bottom = getBottom(B, F);
 top = getTop(D, H);
 right = getRight(C, G);
 if(left<right&&bottom<top)//
 {
  if(A<=E||B<=F)
  {
      if(!(C<E&&D<F))
    totalArea -= (right - left) * (top - bottom);
  }
  if(E<A&&F<B)//
  {
   if(!(G<A&&H<B))
    totalArea -= (right - left) * (top - bottom);
  }
```

By mnmlist,2015-9-2 9:35:19

```
 }
 return totalArea;
}

public int getLeft(int y1, int y2)
{
 return y1 > y2 ? y1 : y2;
}

public int getBottom(int x1, int x2)
{
 return x1 > x2 ? x1 : x2;
}

public int getTop(int x1, int x2)
{
 return x1 < x2 ? x1 : x2;
}

public int getRight(int y1, int y2)
{
 return y1 < y2 ? y1 : y2;
}
```


leetcode_Delete Node in a Linked List

描述：

Write a function to delete a node (except the tail) in a singly linked list, given only access to that node.
Supposed the linked list is 1 -> 2 -> 3 -> 4 and you are given the third node with value 3, the linked list should become 1 -> 2 -> 4 after calling your function.

思路：

1.若不是最后一个，将当前节点的下一个节点的值赋值给当前节点，然后直接删除下一个节点即可。

2.若是最后一个，即currentNode.next==null，这就需要从头遍历一下当前链表了，找到倒数第二个节点，直接node.next=node.next.next；即可完成删除工作。

By mnmlist,2015-9-2 9:35:19

当然，本题目要求的仅仅是将当前节点删除即可。

代码：

```
public void deleteNode(ListNode node) {
       if(node==null)
           return;
       node.val=node.next.val;
       node.next=node.next.next;
   }
```


leetcode_Evaluate Reverse Polish Notation

描述：

Evaluate the value of an arithmetic expression in Reverse Polish Notation.
Valid operators are +, -, *, /. Each operand may be an integer or another expression.
Some examples:
["2", "1", "+", "3", "*"] -> ((2 + 1) * 3) -> 9
["4", "13", "5", "/", "+"] -> (4 + (13 / 5)) -> 6

思路：

1.就像题目所描述的，计算逆波兰数学表达式的结果，循环访问字符串数组类型的表达式

2.遍历的字符是数字，将数字进栈，是数学符号的时，将连续的两个数字出栈并计算出结果，然后将结果再入栈

By mnmlist,2015-9-2 9:35:19

3.重复步骤2直至所有的数组元素被访问完毕。

代码：

//代码通俗易懂，略冗余。

```
public int evalRPN(String[] tokens) {
       if(tokens==null||tokens.length==0)
           return 0;
       if(tokens.length==1)
           return Integer.parseInt(tokens[0]);
       int parseNum=0;
       int num1=0,num2=0;
       Stack<Integer>st=new Stack<Integer>();
       for(int i=0;i<tokens.length;i++)//iterate the item of the array
       {
        if(isOperator(tokens[i]))//if the item is operator,caculate the numbers
        {
         num2=st.peek();
         st.pop();
         num1=st.peek();
         st.pop();
         parseNum=evaluteNums(num1,num2,tokens[i]);
         if(i+1==tokens.length)
          return parseNum;
         st.push(parseNum);
        }else {
   st.push(Integer.parseInt(tokens[i]));//if the item is number,push to the stack
  }

       }
       return parseNum;
   }
public int evaluteNums(int num1,int num2,String operator)
{
 if(operator.equals("+"))
  return num1+num2;
 else if(operator.equals("-"))
  return num1-num2;
 else if(operator.equals("*"))
  return num1*num2;
 else
  return num1/num2;
}
public boolean isOperator(String str)
{
 char operator=str.charAt(0);
 if(operator=='+'||operator=='-'||operator=='*'||operator=='/')
```

By mnmlist,2015-9-2 9:35:19

```
 {
  if(str.length()==1)
   return true;
 }
 return false;
}
```


leetcode_Evaluate Reverse Polish Notation

描述：

Evaluate the value of an arithmetic expression in Reverse Polish Notation.

Valid operators are +, -, *, /. Each operand may be an integer or another expression.

Some examples:
["2", "1", "+", "3", "*"] -> ((2 + 1) * 3) -> 9
["4", "13", "5", "/", "+"] -> (4 + (13 / 5)) -> 6

思路：

1.就像题目所描述的，计算逆波兰数学表达式的结果，循环访问字符串数组类型的表达式

2.遍历的字符是数字，将数字进栈，是数学符号的时，将连续的两个数字出栈并计算出结果，然后将结果再入栈

3.重复步骤2直至所有的数组元素被访问完毕。

代码：

//代码通俗易懂，略冗余。

By mnmlist,2015-9-2 9:35:19

```
public int evalRPN(String[] tokens) {
       if(tokens==null||tokens.length==0)
           return 0;
       if(tokens.length==1)
           return Integer.parseInt(tokens[0]);
       int parseNum=0;
       int num1=0,num2=0;
       Stack<Integer>st=new Stack<Integer>();
       for(int i=0;i<tokens.length;i++)//iterate the item of the array
       {
        if(isOperator(tokens[i]))//if the item is operator,caculate the numbers
        {
         num2=st.peek();
         st.pop();
         num1=st.peek();
         st.pop();
         parseNum=evaluteNums(num1,num2,tokens[i]);
         if(i+1==tokens.length)
          return parseNum;
         st.push(parseNum);
        }else {
   st.push(Integer.parseInt(tokens[i]));//if the item is number,push to the stack
  }

       }
       return parseNum;
   }
public int evaluteNums(int num1,int num2,String operator)
{
 if(operator.equals("+"))
  return num1+num2;
 else if(operator.equals("-"))
  return num1-num2;
 else if(operator.equals("*"))
  return num1*num2;
 else
  return num1/num2;
}
public boolean isOperator(String str)
{
 char operator=str.charAt(0);
 if(operator=='+'||operator=='-'||operator=='*'||operator=='/')
 {
  if(str.length()==1)
   return true;
 }
 return false;
}
```

By mnmlist,2015-9-2 9:35:19

leetcode_Contains Duplicate

描述：

Given an array of integers, find if the array contains any duplicates. Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.

思路：

给定一堆数，让判断是否有重复的数字，无外乎将之前记录的标记下来，但是标记历史记录肯定要用到O（n）大小的存储空间，而用多余的存储空间又是面试时很忌讳的。所以我想到了，用两个HashSet来标记是否出现了重复数字，2个？没错，虽然是2个，但是HashSet是用1bit来表示标记一个数，总体来讲还是比开辟一个int[]更节省内存空间的。

代码：

```
public class Solution {
   public boolean containsDuplicate(int[] nums) {
       if(nums==null||nums.length==0)
           return false;
       BitSet  set1=new BitSet();
       BitSet  set2=new BitSet();
       int len=nums.length;
       int num=0;
       for(int i=0;i<len;i++)
       {
           if(nums[i]>=0)
           {
               if(!set1.get(nums[i]))
                   set1.set(nums[i]);
                else
                   return true;
           }else
           {
               num=-nums[i];
                if(!set2.get(num))
                   set2.set(num);
                else
```

```
                    return true;
            }

        }
        return false;
    }
}
```


leetcode_Valid Sudoku

描述：

Determine if a Sudoku is valid, according to: Sudoku Puzzles - The Rules.

The Sudoku board could be partially filled, where empty cells are filled with the character '.'.

A partially filled sudoku which is valid.

Note:
A valid Sudoku board (partially filled) is not necessarily solvable. Only the filled cells need to be validated.

思路：

和一圈又一圈由外而内打印数字一样，考察的也就是程序的执行流程和边界值的把握。而这一题就更简单，由题意可知，本题考查的是每行每列和9个板块之间是不是都符合要求，和八皇后有点像，但要简单的多。

代码：

```
public class Solution {
    public boolean isValidSudoku(char[][] board) {
        BitSet set=new BitSet();
        int i,j,m,k,index_i,index_j;
```

By mnmlist,2015-9-2 9:35:19

```
int num=0;
//to evaluate if the 9 blocks are OK
for( i=0;i<board.length;i+=3)
{
    for( j=0;j<board[0].length;j+=3)
    {
        set.clear();
        index_i=i+3;
        index_j=j+3;
        for( m=i;m<index_i;m++)
        {
            for( k=j;k<index_j;k++)
            {
                if(board[m][k]!='.')
                {
                    num=board[m][k]-'0';
                    if(!set.get(num))
                        set.set(num);
                    else
                        return false;
                }
            }
        }
    }
}
//to evaluate the rows
for(i=0;i<board.length;i++)
{
    set.clear();
    for(j=0;j<board[0].length;j++)
    {
        if(board[i][j]!='.')
        {
            num=board[i][j]-'0';
             if(!set.get(num))
                set.set(num);
            else
                return false;
        }

    }
}
//to evalueate the colums
for(j=0;j<board[0].length;j++)
{
    set.clear();
    for( i=0;i<board.length;i++)
    {
        if(board[i][j]!='.')
        {
            num=board[i][j]-'0';
            if(!set.get(num))
             set.set(num);
            else
             return false;
        }
```

By mnmlist,2015-9-2 9:35:19

```
            }
        }
        return true;
    }
}
```


leetcode_Insertion Sort List

描述：

Sort a linked list using insertion sort.

思路：

实现对链表的插入排序，显然，只能从头开始对链表进行插入排序了，时间复杂度O（n*n）

代码：

```
/**
* Definition for singly-linked list.
* public class ListNode {
*     int val;
*     ListNode next;
*     ListNode(int x) { val = x; }
* }
*/
public class Solution {
    public ListNode insertionSortList(ListNode head) {
        if(head==null||head.next==null)
            return head;
        ListNode newHead=new ListNode(0);
        newHead.next=head;
        ListNode temp=null,p=head,q=null;
        while(p.next!=null)
        {
            if(p.val<=p.next.val)
```

By mnmlist,2015-9-2 9:35:19

```
            {
                p=p.next;
            }else
            {
                temp=p.next;
                p.next=p.next.next;
                 q=newHead;
                 while(q!=p)
                {
                 if(temp.val<q.next.val)
                {
                    temp.next=q.next;
                    q.next=temp;
                    break;
                 }
                 q=q.next;

                }
            }

        }
        return newHead.next;
    }
}
```


leetcode_Word Break

描述：
Given a string s and a dictionary of words dict, determine if s can be segmented into a space-separated sequence of one or more dictionary words.
For example, given
s = “leetcode”,
dict = [“leet”, “code”].
Return true because “leetcode” can be segmented as “leet code”.

思路：
刚开始拿到这题时，直接用循环的方式来做，即用一个
while（（index=s.indexOf(str)）!=-1）将所有符合要求的str穷举出来，一边穷举一边删除已经出现的字符串，但是，对s=”aaaaaaa”,setArr={“aaaa”,”aa”,”aaa”}这种测试用例就没招了，很显然到”aa”时已经将s删的只剩下”a”了，结果肯定不行。
这种方法的缺点是不能记忆已经产生的结果并推倒重来。
关于推倒重来，回溯法倒是不错，就像八皇后问题那样，但是问题不是变得越来越复杂了

By mnmlist,2015-9-2 9:35:19

么。
最后，还是用动态规划解决了该问题，先从第一个字符开始比对以第一个字符开始的字符串和set里面的字符串进行比对，如果有相同的字符串就将（i+str.length())处标记为true，依次类推，但只有在第i位已经被标记为true的位置才可以作为比对的开始位置，就像接力赛一样。最后能接力到最后的话就表示字符串可以分解为字典集合里面的元素。

感觉还是代码更好理解，直接上代码：
代码：

```
public  boolean wordBreak(String s, Set<String> wordDict) {
       if(s==null)
           return true;
       BitSet set=new BitSet();
       set.set(0);//
       int len=s.length();
       int tempLen=0,endIndex=0;
       for(int i=0;i<len;i++)
       {
           //
           if(!set.get(i))
               continue;
           //
           for(String str:wordDict)
           {
               tempLen=str.length();
               endIndex=i+tempLen;
               if(endIndex>len)//
                   continue;
               if(set.get(endIndex))
                   continue;
               if(s.subSequence(i, endIndex).equals(str))
                   set.set(endIndex);
           }
       }
       //
       return set.get(len);

   }
```


leetcode_Valid Parentheses

描述：

By mnmlist,2015-9-2 9:35:19

Given a string containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

The brackets must close in the correct order, "()" and "()[]{}" are all valid but "(]" and "([)]" are not.

思路：

很简单的字符串匹配问题，直接用栈就可以实现，为使程序更加高效，当要匹配的字符种类比较多的时候可以考虑用HashMap来保存匹配对

代码：

用if else来比较匹配对

```
public  boolean isValid(String s) {
       if(s==null||s.length()==0)
           return true;
       Stack<Character> st=new Stack<Character>();
       char ch;
       int len=s.length();
       for(int i=0;i<len;i++)
       {
           ch=s.charAt(i);
           if(!st.empty())
           {
               if(ch==')'&&st.peek()=='(')
               {
                   st.pop();
               }else if(ch==']'&&st.peek()=='[')
               {
                   st.pop();
               }
               else if(ch=='}'&&st.peek()=='{')
               {
                   st.pop();
               }else
                   st.push(ch);
           }else
               st.push(ch);

       }
       if(!st.empty())
           return false;
       return true;

   }
```

By mnmlist,2015-9-2 9:35:19

用HashMap来保存匹配对

```
public  boolean isValid(String s) {
 if (s == null || s.length() == 0)
  return true;
 Stack<Character> st = new Stack<Character>();
 Map<Character, Character> map = new HashMap<Character, Character>();
 map.put('(', ')');
 map.put('{', '}');
 map.put('[', ']');
 int len=s.length();
 for (int i = 0; i < len; i++) {
  if (!st.empty()&&map.containsKey(st.peek())&&s.charAt(i) == map.get(st.peek())) {
   st.pop();
  }else
   st.push(s.charAt(i));
 }
 return st.empty();
}
```


leetcode_Longest Common Prefix

描述：

Write a function to find the longest common prefix string amongst an array of strings.

思路：

有两种思路：

By mnmlist,2015-9-2 9:35:19

1.从0向最大的公共前缀长度进行，i=0,即每次从0循环至strs.length，所有的字符都相等，则count++，直至有一个字符不相同为止，循环终止
2.假设 longest common prefix 等于字符串数组的最短字符串的长度，从0循环至strs.length，在前面最长公共最大长度的基础上比较相邻两个串的最大公共子串

个人感觉还是第一种思路更好，用到的额外存储空间更少，时间也更短

代码：

思路1：从0向最大的公共前缀长度进行，i=0,即每次从0循环至strs.length，所有的字符都相等，则count++，直至有一个字符不相同为止，循环终止

```
public String longestCommonPrefix(String[] strs) {
       if(strs==null)
           return null;
       String str=new String("");
       if(strs.length==0)
           return str;
       if(strs.length==1)
           return strs[0];
       int min=strs[0].length();
       int len=0;
       for(int i=1;i<strs.length;i++)
       {
           len=strs[i].length();
           min=min<len?min:len;
       }
       char temp='0';
       int count=0,i=0,j=0;
       boolean flag=true;
       for(i=0;i<min;i++)
       {
           temp=strs[0].charAt(i);
           flag=true;
           for(j=1;j<strs.length;j++)
           {
               if(temp!=strs[j].charAt(i))
               {
                   flag=false;
                   break;
               }
           }
           if(flag)
               count++;
           else
               break;
       }
       return strs[0].substring(0,count);

   }
```

By mnmlist,2015-9-2 9:35:19

思路二：假设 longest common prefix 等于字符串数组的最短字符串的长度，从0循环至strs.length，在前面最长公共最大长度的基础上比较相邻两个串的最大公共子串

```
public String longestCommonPrefix(String[] strs) {
       if(strs==null)
           return null;
       String str=new String("");
       if(strs.length==0)
           return str;
       if(strs.length==1)
           return strs[0];
       int min=strs[0].length();
       int len=0;
       for(int i=1;i<strs.length;i++)
       {
           len=strs[i].length();
           min=min<len?min:len;
       }
       int count=min,i=0,j=0;
       for(i=1;i<strs.length;i++)
       {
           for(j=min-1;j>=0;j--)
           {
                 if(!strs[i-1].substring(0,j+1).equals(strs[i].substring(0,j+1)))
                 {
                     count--;
                 }else
                   break;
           }
           min=count;
       }
       return strs[0].substring(0,count);

   }
```


java多线程问题中死锁的一个实现

1.直接上代码：

```
class LockDemo{
public static final Object A_LOCK=new Object();
public static final Object B_LOCK=new Object();
}
```

By mnmlist,2015-9-2 9:35:19

```
public class ThreadLockDemo implements Runnable{
public boolean flag;
public ThreadLockDemo(boolean flag)
{
 this.flag=flag;
}
public void run()
{
 if(flag)
 {
  synchronized (LockDemo.A_LOCK) {
   System.out.println("if  A_LOCK");
   synchronized (LockDemo.B_LOCK) {
    System.out.println("B_LOCK");
   }

  }
 }
 else
 {
  synchronized (LockDemo.B_LOCK) {
   System.out.println("else B_LOCK");
   synchronized (LockDemo.A_LOCK) {
    System.out.println("A_LOCK");
   }
  }
 }
}
public static void main(String[] args) {
 // TODO Auto-generated method stub
 ThreadLockDemo td1=new ThreadLockDemo(false);
 ThreadLockDemo td2=new ThreadLockDemo(true);
 Thread t1=new Thread(td1);
 Thread t2=new Thread(td2);
 t1.start();
 t2.start();
}

}
```

2.有图有真相

leetcode_Largest Number

By mnmlist,2015-9-2 9:35:19

描述：

Given a list of non negative integers, arrange them such that they form the largest number.
For example, given [3, 30, 34, 5, 9], the largest formed number is 9534330.
Note: The result may be very large, so you need to return a string instead of an integer.

思路：

1.将数字转换为字符串

2.对字符串数组进行升序排序

3.具体的排序规则为将两个字符串以str1+str2和str2+str1的方式拼接起来，然后比较两个拼接后的字符串即可

4.将排好序的字符串数组拼接起来即为要求的最大整数字符串

ps：刚开始走了好多弯路以比较短的字符串为基准和比较长的字符串循环比较，直至有不同大小的段为止，吃力不讨好，最后也没有算对^-^!

代码：

```
public class Solution implements Comparator<String>{
   public int compare(String str1,String str2)
   {
       String newStr1=str1+str2;
       String newStr2=str2+str1;
       return newStr2.compareTo(newStr1);
   }
   public String largestNumber(int[] nums) {
       if(nums==null||nums.length==0)
           return new String("");
       if(nums.length==1)
           return String.valueOf(nums[0]);
       int len=nums.length;
       String arr[]=new String[len];
       for(int i=0;i<len;i++)
        arr[i]=String.valueOf(nums[i]);
```

```
By mnmlist,2015-9-2 9:35:19
```

```
        Arrays.sort(arr,new Solution());
        StringBuilder sb=new StringBuilder();
        for(int i=0;i<len;i++)
            sb.append(arr[i]);
        if(sb.charAt(0)=='0')
            return "0";
        return sb.toString();
    }
}
```


leetcode_Container With Most Water

描述：

Given n non-negative integers a1, a2, ..., an, where each represents a point at coordinate (i, ai). n vertical lines are drawn such that the two endpoints of line i is at (i, ai) and (i, 0). Find two lines, which together with x-axis forms a container, such that the container contains the most water.

Note: You may not slant the container.

思路：

1.先获取两段的垂线和x轴组成的容器可以容纳的水量tempSum，并使得maxSum=tempSum;

2.然后再去除两段较小的垂线段，组成新的容器，获得新的可容纳的水量tempSum，并更新maxSum

3.循环直至start==end

代码：

```
public int maxArea(int[] height) {
```

By mnmlist,2015-9-2 9:35:19

```
    if(height==null||height.length==1)
         return 0;
    int end=height.length-1,start=0;
    int sum=0;
    while(start<end)
    {
        sum=Math.max(sum,Math.min(height[start],height[end])*(end-start));
        if(height[start]<height[end])
             start++;
         else
             end--;
    }
    return sum;
}
```


leetcode_Minimum Size Subarray Sum

描述：

Given an array of n positive integers and a positive integer s, find the minimal length of a subarray of which the sum ≥ s. If there isn't one, return 0 instead.

For example, given the array [2,3,1,2,4,3] and s = 7,
the subarray [4,3] has the minimal length under the problem constraint.

click to show more practice.

More practice:
If you have figured out the O(n) solution, try coding another solution of which the time complexity is O(n log n).

思路：

1.找到第一符合条件的长度

2.先加上后面的一个元素

3.如果减去前面的一个元素后sum小于target，转到2

By mnmlist,2015-9-2 9:35:19

3.减去前面的n个元素后符合条件&&减去前面的n+1个元素后不符合条件，获得一个新的长度，跟最小长度相比，小于minLen，更新minLen=newLen

4.若果start<end&&end<nums.length，转至2

代码：

```
public int minSubArrayLen(int s, int[] nums) {
       if(nums==null||nums.length==0)
           return 0;
       int start=0,end=-1;
       int sum=0,min=0;
       int temp=0;
       while(sum<s&&end<nums.length-1)//
        sum+=nums[++end];
       if(sum<s)
           return 0;
       min=end-start+1;
       while(start<end&&end<nums.length)
       {
        if(end+1<nums.length)
         sum+=nums[++end];
           if(sum-nums[start]<s)
           {
            if(end==nums.length-1)//
                break;
            else
             continue;//s
           }
           while(start<end&&sum-nums[start]>=s)////s
               sum-=nums[start++];
           temp=end-start+1;
           min=min<temp?min:temp;
       }
       return min;
   }
```


leetcode_Maximum Product Subarray

By mnmlist,2015-9-2 9:35:19

描述：

Find the contiguous subarray within an array (containing at least one number) which has the largest product.

For example, given the array [2,3,-2,4],
the contiguous subarray [2,3] has the largest product = 6.

思路：

0．动态规划问题，和求最大连续和maximum subarray类似，但感觉比求最大连续和复杂的多

1．以0为分割元素获得一系列的区间

2．对每一个区间求最大值

3．具体到每一个区间，顺序查找一遍寻找最大的序列，逆序查找一遍寻找最大的序列，求顺序或逆序查找的最大值

4．注意：(tempCount1&1) == 1)可以节省好多时间，用%2==1就不行

代码：

```
public int maxProduct(int[] nums) {
 if (nums == null)
  return 0;
 if (nums.length == 0)
  return 0;
 if (nums.length == 1)
  return nums[0];
 List<Integer> list = new ArrayList<Integer>();
 list.add(-1);
 for (int i = 0; i < nums.length; i++) {//0
  if (nums[i] == 0)
   list.add(i);
 }
 list.add(nums.length);
 int len = list.size();
 int max = 0, tempMax = 0;
 for (int i = 0; i < len - 1; i++) {//
```

By mnmlist,2015-9-2 9:35:19

```
  tempMax = getMax(nums, list.get(i), list.get(i + 1));
  max = max > tempMax ? max : tempMax;
 }
 return max;
}

public int getMax(int nums[], int start, int end) {
 if (end - start <= 2)//==20<20
 {
  if(start!=nums.length-1)
   return nums[start + 1];
  else
   return nums[start];//
 }
 int sum = 1;
 int count = 0;
 for (int i = start + 1; i < end; i++) {
  sum *= nums[i];
  if (nums[i] < 0)
   count++;
 }
 int tempSum = sum;
 int tempCount = count;
 for (int i = start + 1; i < end; i++) {//
  if (tempSum < 0) {
   if (nums[i] < 0 &&(tempCount&1) == 1) {//(tempCount1&1) == 1)%2==1
    tempSum /= nums[i];
    break;
   } else {
    tempSum /= nums[i];
   }
  } else
   break;
 }
 int tempSum1 = sum;
 int tempCount1 = count;
 for (int i = end - 1; i > start; i--) {//
  if (tempSum1 < 0) {
   if (nums[i] < 0 && (tempCount1&1) == 1) {//(tempCount1&1) == 1)%2==1
    tempSum1 /= nums[i];
    break;
   } else {
    tempSum1 /= nums[i];
   }
  } else
   break;
 }
 int max = tempSum > tempSum1 ? tempSum : tempSum1;//
 return max;
}
```

By mnmlist,2015-9-2 9:35:19

leetcode_Copy List with Random Pointer

描述：

A linked list is given such that each node contains an additional random pointer which could point to any node in the list or null.

Return a deep copy of the list.

思路：

1.首先根据旧链表的值创建一个新的链表并分别将旧链表和新链表存储到listOld和listNew中。

2.然后根据旧链表中index位置的结点的random指针所指向的位置，找出旧链表指针所指向结点在listOld中的index，也即listNew中的newIndex

3.把新链表中的index位置的结点指向newIndex位置的结点，问题得解！

代码：

```
/**
* Definition for singly-linked list with a random pointer.
* class RandomListNode {
*     int label;
*     RandomListNode next, random;
*     RandomListNode(int x) { this.label = x; }
* };
*/
public class Solution {
   public RandomListNode copyRandomList(RandomListNode head) {
 if(head==null)
  return null;
 List<RandomListNode>listOld=new ArrayList<RandomListNode>();//store the old node
 List<RandomListNode>listNew=new ArrayList<RandomListNode>();//store the new node
 RandomListNode newHead=new RandomListNode(0);
 RandomListNode pListNode=head,curListNode=newHead;
       while(pListNode!=null)//create the new LinkList && store the old node to listOld&s
       {
```

By mnmlist,2015-9-2 9:35:19

```
        listOld.add(pListNode);//store the old node
        RandomListNode qListNode=new RandomListNode(pListNode.label );
        listNew.add(qListNode);//store the new node
        curListNode.next=qListNode;
        curListNode=qListNode;
        pListNode=pListNode.next;
       }
       RandomListNode qListNode=newHead.next,temp=null;
       pListNode=head;
       int indexOld=0;
       while(pListNode!=null)//get the index of pListNode.random ,then put the node of ir
       {
        temp=pListNode.random;
        if(temp!=null)
        {
         indexOld=listOld.indexOf(pListNode.random);//get the index of pListNode.random
            qListNode.random=listNew.get(indexOld);//put the node of indexOld to che rand
        }
        else {
   qListNode.random=null;
  }
        pListNode=pListNode.next;
        qListNode=qListNode.next;
       }

       return newHead.next;
   }
}
```


leetcode_Rotate Array

描述：

Rotate an array of n elements to the right by k steps.

For example, with n = 7 and k = 3, the array [1,2,3,4,5,6,7] is rotated to [5,6,7,1,2,3,4].

Note:
Try to come up as many solutions as you can, there are at least 3 different ways to solve this problem.

[show hint]

Hint:
Could you do it in-place with O(1) extra space?
Related problem: Reverse Words in a String II

By mnmlist,2015-9-2 9:35:19

思路：

当然，一个很简单且容易想到的思路就是直接循环移位k位即可，但每次都要移动n个元素，即总共需要移动k*n个元素

和Reverse Words in a String II题目类似，还有一种通过改变固定数目的元素就可以实现移位数组的功能，即先将1~len-k，len-k~len之间的元素逆置，最后将1~len之间的元素逆置，可以实现最后的旋转数组的目的。

代码：

```
public void rotate(int[] nums, int k) {
       if(nums==null)
           return;
 int len=nums.length;
 k=k%len;
 if(k==0)
     return;
 int mid=len-k;
 int temp=0;
 int index=mid/2;
 for(int i=0;i<index;i++)
 {
     temp=nums[i];
     nums[i]=nums[mid-1-i];
     nums[mid-1-i]=temp;
 }
 index=(mid+len)/2;
 for(int i=mid;i<index;i++)
 {
     temp=nums[i];
     nums[i]=nums[mid+len-1-i];
     nums[mid+len-1-i]=temp;
 }
 index=len/2;
 for(int i=0;i<index;i++)
 {
     temp=nums[i];
     nums[i]=nums[len-1-i];
     nums[len-1-i]=temp;
 }
  }
```

By mnmlist,2015-9-2 9:35:19

leetcode_Search for a Range

描述：

Given a sorted array of integers, find the starting and ending position of a given target value.
Your algorithm's runtime complexity must be in the order of O(log n).
If the target is not found in the array, return `[-1, -1]`.
For example,
Given `[5, 7, 7, 8, 8, 10]` and target value 8,
return `[3, 4]`.

思路：

1.二分查找，找到value为target的一个元素的位置，找不到返回[-1,-1]

2.假设找到位置，且为index，分别向两边拓展即可。

代码：

```
public int[] searchRange(int[] A, int target) {
        int arr[]=new int[2];
        Arrays.fill(arr, -1);
        if(A==null||A.length==0)
         return arr;
        int len=A.length;
        int start=0,end=len-1;
        int mid=0;
        int index=-1;
        while(start<=end)
        {
         mid=(start+end)/2;
         if(A[mid]==target)
         {
          index=mid;
          break;
         }else if(A[mid]<target)
          start=mid+1;
         else
```

By mnmlist,2015-9-2 9:35:19

```
            end=mid-1;
        }
        if(index==-1)
         return arr;
        int i=1,temp=index-i;
        while(temp>=0&&A[temp]==target)
        {
         i++;
         temp=index-i;
        }
        arr[0]=index-i+1;
        i=1;
        temp=index+i;
        while(temp<len&&A[temp]==target)
        {
         i++;
         temp=index+i;
        }
        arr[1]=index+i-1;
        return arr;
    }
```


leetcode_Search Insert Position

描述：

Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

Here are few examples.
[1,3,5,6], 5 → 2
[1,3,5,6], 2 → 1
[1,3,5,6], 7 → 4
[1,3,5,6], 0 → 0

思路：

1.变形的二分查找问题

2.最后A[mid]=target，index=mid

By mnmlist,2015-9-2 9:35:19

3.最后A[mid]!=target，又分A[mid]<target和A[mid]>target两种情况，

4.if(A[mid]<target),index=mid+1;if(A[mid]>target),index=mid;

代码：

```
public int searchInsert(int[] A, int target) {
 int len=A.length;
    int start=0,end=len-1;
    int mid=0;
    int index=-1;//if A[mid]=target
    int index2=-1;//if A[mid]!=target
    while(start<=end)
    {
     mid=(start+end)/2;
     if(A[mid]==target)
     {
      index=mid;
      break;
     }else if(A[mid]<target)
      start=mid+1;
     else
      end=mid-1;
    }
    if(index!=-1)
     return index;
    else
    {
     if(A[mid]<target)
      index2=mid+1;
     else {
   index2=mid;
  }
    }
    return index2;
   }
```


leetcode_Count Primes

By mnmlist,2015-9-2 9:35:19

描述：

Count the number of prime numbers less than a non-negative number, n

思路：

判断一个数是否为质数有两种方法，一种是判断它能否被 2 ~ (int)sqrt(n)+1之间的数整除，能被整除为合数，否则为质数

但是，当n非常大的时候这种方法是非常费时间的。

另外一种改进的方法是仅用n除以2~(int)sqrt(n)+1之间的所有质数，而2~(int)sqrt(n)+1之间的质数从何而来？先计算出来

测试用例需要的小于num的所有质数的num大概为1200就能通过本题的测试用例

代码：

```
public int countPrimes(int n) {
        n--;
        if(n<2)
            return 0;
        if(n==2)
            return 1;
        int primArr[]={2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89
        int count=1;
        int temp=0;
        for(int i=3;i<=n;i++)
        {
            count++;
            temp=(int)Math.sqrt(i)+1;
            for(int j=0;j<primArr.length;j++)
            {
                if(temp>primArr[j])
                {
                    if(i%primArr[j]==0)
                    {
                        count--;
                        break;
                    }
                }else
```

```
                    break;
                }
            }
            return count;
        }
```


leetcode_Reverse Linked List

描述：

Reverse a singly linked list.

思路：

感觉还是非递归方法更简洁明快，创建一个头节点并链接到第一个结点前面，从第二个结点开始依次用前插法插到前面，最后得到倒序的结点。

代码：

非递归方法：

```
public ListNode reverseList(ListNode head) {
        if(head==null||head.next==null)
            return head;
        ListNode headNode=new ListNode(0);
        ListNode pNode,qNode;
        headNode.next=head;
        pNode=head.next;
        head.next=null;
        while(pNode!=null)
        {
            qNode=pNode;
            pNode=pNode.next;
            qNode.next=headNode.next;
            headNode.next=qNode;
        }
```

By mnmlist,2015-9-2 9:35:19

```
        return headNode.next;
    }
```

递归方法：

```
public ListNode reverseList(ListNode head) {
        if(head==null)
            return head;
        ListNode newHead,curNode,tempNode;
        if(head.next==null)
            return head;
        else
        {
            curNode=head;
            tempNode=head.next;
            newHead=reverseList(tempNode);
        }
        curNode.next=null;
        tempNode.next=curNode;
        return newHead;
    }
```


leetcode_Rotate Image

描述：

You are given an n x n 2D matrix representing an image.

Rotate the image by 90 degrees (clockwise).

Follow up:
Could you do this in-place?

思路：

对于这种题目，就应该在纸上画画写写，由简单到一般，一般是可以发现规律的。当然了，只是有规律，newArr[j][rows-i-1]=matrix[i][j];

By mnmlist,2015-9-2 9:35:19

代码：

```
public void rotate(int[][] matrix) {
       int rows=matrix.length;
       int colums=matrix[0].length;
      int newArr[][]=new int[rows][colums];
      for(int i=0;i<rows;i++)
      {
          for(int j=0;j<colums;j++)
               newArr[j][rows-i-1]=matrix[i][j];
      }
      for(int i=0;i<rows;i++)
      {
          for(int j=0;j<colums;j++)
               matrix[i][j]=newArr[i][j];
      }

   }
```


leetcode_Remove Element

描述:

Given an array and a value, remove all instances of that value in place and return the new length.

The order of elements can be changed. It doesn't matter what you leave beyond the new length.

思路：

将值和该元素相等的元素略过即可

代码：

By mnmlist,2015-9-2 9:35:19

```
public int removeElement(int[] A, int elem) {
       if(A==null||A.length==0)
           return 0;
       int index=0,i=0;
       int len=A.length;
       while(i<len)
       {
           if(A[i]!=elem)
               A[index++]=A[i];
           i++;
       }
       return index;
   }
```


leetcode_Remove Duplicates from Sorted Array II

描述：

Follow up for "Remove Duplicates":
What if duplicates are allowed at most twice?

For example,
Given sorted array nums = [1,1,1,2,2,3],

Your function should return length = 5, with the first five elements of nums being 1, 1, 2, 2 and 3. It doesn't matter what you leave beyond the new length.

思路：

直接略过重复元素

代码：

By mnmlist,2015-9-2 9:35:19

```
public int removeDuplicates(int[] A) {
        if(A==null)
           return 0;
       if(A.length<3)
           return A.length;
      int len=A.length;
      int index=2;//new length
      int i=2;//to traverse the array
      int temp=0;
      while(i<len)
      {
          temp=index-2;
          while(i<len&&A[i]==A[temp])
               i++;
           if(i<len)
               A[index++]=A[i];
           i++;
      }
      return index;
   }
```


leetcode_Remove Duplicates from Sorted Array

描述：

Given a sorted array, remove the duplicates in place such that each element appear only once and return the new length.

Do not allocate extra space for another array, you must do this in place with constant memory.

For example,
Given input array nums = [1,1,2],

Your function should return length = 2, with the first two elements of nums being 1 and 2 respectively. It doesn't matter what you leave beyond the new length.

思路：

直接略过重复的元素即可

By mnmlist,2015-9-2 9:35:19

代码：

```
public int removeDuplicates(int[] A) {
       if(A==null||A.length==0)
           return 0;
      int len=A.length;
      int index=1;//new length
      int i=1;//to traverse the array
      int temp=0;
      while(i<len)
      {
          temp=i-1;
          while(i<len&&A[i]==A[temp])
               i++;
           if(i<len)
               A[index++]=A[i];
           i++;
      }
      return index;
   }
```


leetcode_Plus One

描述：

Given a non-negative number represented as an array of digits, plus one to the number.
The digits are stored such that the most significant digit is at the head of the list.

思路：

很简单的字符串加法，当然要考虑到最后的进位

代码：

By mnmlist,2015-9-2 9:35:19

```
public int[] plusOne(int[] digits) {
       if(digits==null||digits.length==0)
           return digits;
       int len=digits.length;
       int flowNum=1;
       int sum=0;
       for(int i=len-1;i>=0;i--)
       {
           sum=digits[i]+flowNum;//the first time flowNum is the addNum 1
           digits[i]=sum%10;
           flowNum=sum/10;
       }
       if(flowNum!=0)
       {
           int newArr[]=new int[len+1];
           newArr[0]=flowNum;
           for(int i=1;i<newArr.length;i++)
               newArr[i]=digits[i-1];
           return newArr;
       }
       return digits;
   }
```


leetcode_Majority Element

描述：

Given an array of size n, find the majority element. The majority element is the element that appears more than ? n/2 ? times.

You may assume that the array is non-empty and the majority element always exist in the array.

思路：

令temp=arr[0],count=1;如果下一个元素跟temp相等，count++；若果不等，count--；若果count==0；temp换做当前元素，count=1；由于majority element出现的次数大于一半长度，所以最后剩下的元素肯定为majority element

By mnmlist,2015-9-2 9:35:19

代码：

```
public int majorityElement(int[] num) {
    int temp=num[0],count=1;
    int len=num.length;
    for(int i=1;i<len;i++)
    {
        if(num[i]==temp)
            count++;
        else
        {
            count--;
            if(count==0)
            {
                temp=num[i];
                count=1;
            }
        }

    }
    return temp;
  }
```


leetcode_Merge Sorted Array

描述：

Given two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one sorted array.

Note:
You may assume that nums1 has enough space (size that is greater or equal to m + n) to hold additional elements from nums2. The number of elements initialized in nums1and nums2 are m and n respectively.

思路：

1.最省力的一种方式从后面开始比较，较大的一个放到num1的后方，直至其中一个元素

By mnmlist,2015-9-2 9:35:19

比较完

2.假设num2种仍然存在没有比较完的元素，依次将num2种的元素放置到num1中

代码：

```
public void merge(int A[], int m, int B[], int n) {
       int index=m+n-1;
       int i=m-1,j=n-1;
       while(i>=0&&j>=0)
       {
           if(A[i]<B[j])
               A[index--]=B[j--];
           else
               A[index--]=A[i--];
       }
       if(j>=0)
       {
           while(j>=0)
           {
               A[index--]=B[j--];
           }
       }
   }
```


leetcode_Search in Rotated Sorted Array II

描述：

Follow up for "Search in Rotated Sorted Array":
What if duplicates are allowed?

Would this affect the run-time complexity? How and why?

By mnmlist,2015-9-2 9:35:19

Write a function to determine if a given target is in the array.

思路：

本题木的特点是数组初始有序，然后循环移位了。
由于是循环移位，所以数组前一半或后一半至少有一半元素是有序的，而找到其中一半有序的元素正式本题的题眼。
1.初始start=0，end=len-1
2.mid=(start+end)/2;如果arr[mid]==target，找到元素
3.如果arr[start]<arr[mid]，则前半部分有序，如果arr[start]<=target<arr[mid],则target存在的话肯定在前半部分，end=mid-1，如果target>arr[mid]，则target存在的话肯定在后半部分，start=mid+1;
4.如果arr[start]>arr[mid]，则后半部分有序，若果。。。。推理过程同3
5.如果arr[start]==arr[mid]，start=start+1

代码：

```
public boolean search(int[] A, int target) {
       boolean hasNum=false;
       if(A==null||A.length==0)
        return hasNum;
       int start=0,end=A.length-1,mid=0;
       while(start<=end)
       {
        mid=(start+end)/2;
        if(A[mid]==target)
        {
         hasNum=true;
         break;
        }else
        {
         if(A[start]<A[mid])
            {
             if(A[mid]>target&&A[start]<=target)
              end=mid-1;
             else
    start=mid+1;
            }else if(A[start]>A[mid])
            {
             if(target>A[mid]&&target<=A[end])
              start=mid+1;
             else
    end=mid-1;
            }else
   start++;
  }

       }
       return hasNum;
   }
```

By mnmlist,2015-9-2 9:35:19

Search in Rotated Sorted Array

描述：

Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., `0 1 2 4 5 6 7` might become `4 5 6 7 0 1 2`).

You are given a target value to search. If found in the array return its index, otherwise return -1.

You may assume no duplicate exists in the array.

思路：

本题木的特点是数组初始有序，然后循环移位了。

由于是循环移位，所以数组前一半或后一半至少有一半元素是有序的，而找到其中一半有序的元素正式本题的题眼。

1.初始start=0，end=len-1

2.mid=(start+end)/2;如果arr[mid]==target，找到元素

3.如果arr[start]<arr[mid]，则前半部分有序，如果arr[start]<=target<arr[mid],则target存在的话肯定在前半部分，end=mid-1，如果target>arr[mid]，则target存在的话肯定在后半部分，start=mid+1;

4.如果arr[start]>arr[mid]，则后半福分有序，若果。。。。推理过程同3

代码：

By mnmlist,2015-9-2 9:35:19

```
public int search(int[] A, int target) {
       if(A==null||A.length==0)
        return -1;
       int index=-1;
       int start=0,end=A.length-1,mid=0;
       while(start<=end)
       {
        mid=(start+end)/2;
        if(A[mid]==target)
        {
         index=mid;
         break;
        }else
        {
         if(A[start]<A[mid])
            {
             if(A[mid]>target&&A[start]<=target)
              end=mid-1;
             else
     start=mid+1;
            }else if(A[start]>A[mid])
            {
             if(target>A[mid]&&target<=A[end])
              start=mid+1;
             else
     end=mid-1;
            }else
    start++;
  }

       }
       return index;
   }
```


leetcode_Pascal's Triangle II

描述：

Given an index k, return the kth row of the Pascal's triangle.

For example, given k = 3,

By mnmlist,2015-9-2 9:35:19

Return [1,3,3,1].

思路：

就是类似杨辉三角的问题，用两个数组存储相邻的两行然后用第一行来算第二行即可，将计算到第k行的数据存储到list中即可。

代码：

```
public List<Integer> getRow(int rowIndex) {
 List<Integer> subList = new ArrayList<Integer>();
 int numRows = rowIndex + 1;
 if (numRows <= 0)
  return subList;
 int arr[][] = new int[numRows][];
 arr[0] = new int[1];
 Arrays.fill(arr[0], 1);
 if (numRows > 1) {
  arr[1] = new int[2];
  Arrays.fill(arr[1], 1);
 }
 for (int i = 2; i < numRows; i++) {
  arr[i] = new int[i + 1];
  arr[i][0] = 1;
  for (int j = 1; j < i; j++)
   arr[i][j] = arr[i - 1][j] + arr[i - 1][j - 1];// to caculate the
               // piscal
               // triganle
  arr[i][i] = 1;
 }

 for (int num : arr[rowIndex])
  subList.add(num);
 return subList;
}
```


leetcode_Valid Number

描述：

By mnmlist,2015-9-2 9:35:19

Validate if a given string is numeric.

Some examples:
`"0"` => `true`
`" 0.1 "` => `true`
`"abc"` => `false`
`"1 a"` => `false`
`"2e10"` => `true`

Note: It is intended for the problem statement to be ambiguous. You should gather all requirements up front before implementing one.

Update (2015-02-10):
The signature of the `C++` function had been updated. If you still see your function signature accepts a `const char *` argument, please click the reload button to reset your code definition.

思路：

这道题目还是挺复杂的，复杂的原因是题目并没有给出有效数字的定义，比如说 .1 ，2.1e10.2 ，2e3.2 和2.1e5 有效，但10.1.1非法而这些数字的判断又是很麻烦的

代码：

```
public boolean isNumber(String s) {
 s = s.trim();
 if (s.startsWith("+"))
  s = s.substring(1);
 else if (s.startsWith("-"))
  s = s.substring(1);
 boolean isInterger = false;
 boolean isFloater = false;
 boolean isSciencer = false;
 if (isInt(s))
  isInterger = true;
 int indexFloat = -1;
 int indexScience1 = -1;
 int indexScience2 = -1;
 indexFloat = s.indexOf('.');
 indexScience1 = s.indexOf('e');
 indexScience2 = s.indexOf('E');
 String nums[] = null;
 if (indexFloat != -1) {
  int index1=s.indexOf('.');
  int index2=s.lastIndexOf('.');
  if(index1!=index2)
   isFloater=false;
  else
   isFloater = isFloat(s);
 }
```

By mnmlist,2015-9-2 9:35:19

```
 if (indexScience1 + indexScience2 != -2) {
  if (indexScience1 != -1)
  {
   int index1=s.indexOf('e');
   int index2=s.lastIndexOf('e');
   if(index1!=index2)
    isSciencer=false;
   else
   {
    nums = s.split("e");
    isSciencer = isScientific(nums);
   }
  }

  if (indexScience2 != -1)
  {
   int index1=s.indexOf('E');
   int index2=s.lastIndexOf('E');
   if(index1!=index2)
    isSciencer=false;
   else
   {
    nums = s.split("E");
    isSciencer = isScientific(nums);
   }

  }


 }
 if (isFloater || isInterger || isSciencer)
  return true;
 return false;
}

public boolean isScientific(String num[]) //
{
 if(num.length<2)
  return false;
 if (num[1].startsWith("+") || num[1].startsWith("-"))
  num[1] = num[1].substring(1);
 boolean flag1 = num[0].contains(".");
 if (flag1)
  flag1 = isFloat(num[0]);
 else
  flag1 = isInt(num[0]);
 boolean flag2 = isInt(num[1]);
 if (flag1 && flag2)
  return true;
 return false;
}

public boolean isFloatic(String str[]) //
{
 if (isInt(str[0]) && isInt(str[1]))
  return true;
 return false;
```

```
By mnmlist,2015-9-2 9:35:19
}

public boolean isFloat(String s) //
{
 boolean isTrue = false;
 String nums[] = null;
 if (s.startsWith("."))
  isTrue = isInt(s.substring(1));
 else if(s.endsWith("."))
  isTrue = isInt(s.substring(0,s.length()-1));
 else {
  nums = s.split("\\.");
  if (nums.length != 2)
   return false;
  isTrue = isInt(nums[0]) && isInt(nums[1]);
 }
 return isTrue;
}

public boolean isInt(String str) //
{
 if (str == null || str.length() == 0)
  return false;
 boolean isLeg = true;
 char ch;
 for (int i = 0; i < str.length(); i++) {
  ch = str.charAt(i);
  if (ch > '9' || ch < '0') {
   isLeg = false;
   break;
  }
 }
 return isLeg;
}
```


leetcode_Valid Palindrome

描述：

Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

For example,
"A man, a plan, a canal: Panama" is a palindrome.
"race a car" is not a palindrome.

By mnmlist,2015-9-2 9:35:19

Note:
Have you consider that the string might be empty? This is a good question to ask during an interview.

For the purpose of this problem, we define empty string as valid palindrome.

思路：

很简单，搞一个while循环即可，比较前后的字符是否相同，直至start>=end

代码：

```
String str=null;
   public boolean isPalindrome(String s) {
       boolean isPali=true;
       str=s;
       int i=0,j=s.length()-1;
       while(i<j)
       {
           while(i<j&&!isLegal(getChar(i)))
               i++;
           while(i<j&&!isLegal(getChar(j)))
               j--;
           if(i<j)
           {
                if(getChar(i)!=getChar(j))
               {
                    isPali=false;
                    break;
               }
           }
           i++;
           j--;
       }
       return isPali;
   }
   public char getChar(int index)
   {
       char ch=str.charAt(index);
       if(ch>='A'&&ch<='Z')
           ch+='a'-'A';
       return ch;
   }
   public boolean isLegal(char ch)
   {
       if((ch>='A'&&ch<='Z')||(ch>='a'&&ch<='z')||(ch>='0'&&ch<='9'))
           return true;
       return false;
   }
```

By mnmlist,2015-9-2 9:35:19

leetcode_Spiral Matrix

描述：

Given a matrix of m x n elements (m rows, n columns), return all elements of the matrix in spiral order.

For example,
Given the following matrix:

```
[
[ 1, 2, 3 ],
[ 4, 5, 6 ],
[ 7, 8, 9 ]
]
```

You should return [1,2,3,6,9,8,7,4,5].

思路：

这题看起来是很复杂，做起来也确实挺复杂的。但是呢，这题并不是非常非常难，只是控制逻辑让人很抓狂罢了。
colStart,colEnd,rowStart,rowEnd,num=0
1.colStart<colEnd 输出arr[row][colStart]~arr[row][colEnd-1]的值
2.rowStart<rowEnd 输出arr[col][rowStart]~ar[col][rowEnd-1]的值
3.colEnd>colStart 输出arr[rowLen-1-row][colEnd]~arr[rowLen-1-row][colStart+1]的值
4.rowEnd>rowStart 输出arr[colLen-1-col][rowEnd]~arr[colLen-1-col][rowStart+1]的值

代码：

```
public static List<Integer> spiralOrder(int[][] matrix) {
       List<Integer>list=new ArrayList<Integer>();
       if(matrix==null||matrix.length==0)
        return list;
       int rows=matrix.length,cols=matrix[0].length;
       if(rows==1)
```

By mnmlist,2015-9-2 9:35:19

```
        {
         for(int num:matrix[0])
          list.add(num);
         return list;
        }else if(matrix[0].length==1)
        {
         for(int i=0;i<rows;i++)
          list.add(matrix[i][0]);
         return list;
        }
        int x=0,y=0;
        while(x<cols-x&&y<rows-y)
        {
         int end=cols-x-1;
         if(x<end)
         {
          for(int i=x;i<end;i++)
               list.add(matrix[y][i]);
         }

         end=rows-y-1;
         if(y<end)
         {
          for(int j=y;j<end;j++)
               list.add(matrix[j][cols-x-1]);
         }
         int start=cols-x-1;
         if(start>x)
         {
          for(int i=start;i>x;i--)
               list.add(matrix[rows-y-1][i]);
         }
         start=rows-y-1;
         if(start>y)
         {
          for(int j=start;j>y;j--)
               list.add(matrix[j][x]);
         }
         if(2*x+1==cols&&2*y+1==rows)
          list.add(matrix[y][x]);
         x++;
         y++;
        }
        return list;
    }
```


leetcode_Spiral Matrix II

By mnmlist,2015-9-2 9:35:19

描述：

Given an integer n, generate a square matrix filled with elements from 1 to n^2 in spiral order.

For example,
Given n = 3,

You should return the following matrix:

```
[
[ 1, 2, 3 ],
[ 8, 9, 4 ],
[ 7, 6, 5 ]
]
```

思路：

这题看起来是很复杂，做起来也确实挺复杂的。但是呢，这题并不是非常非常难，只是控制逻辑让人很抓狂罢了。

colStart,colEnd,rowStart,rowEnd,num=0

1.colStart<colEnd 为arr[row][colStart]~arr[row][colEnd-1]赋值num++;

2.rowStart<rowEnd 为arr[col][rowStart]~ar[col][rowEnd-1]赋值num++;

3.colEnd>colStart 为arr[rowLen-1-row][colEnd]~arr[rowLen-1-row][colStart+1]赋值num++;

4.rowEnd>rowStart 为arr[colLen-1-col][rowEnd]~arr[colLen-1-col][rowStart+1]赋值num++;

代码：

```
public int[][] generateMatrix(int n) {
 int matrix[][]=null;
 if(n<0)
```

By mnmlist,2015-9-2 9:35:19

```
        return matrix;
 else if(n==0)
 {
  matrix=new int [1][];
  matrix[0]=new int[]{};
 }
 matrix=new int[n][];
 for(int i=0;i<n;i++)
  matrix[i]=new int[n];
       int startNum=1;
       int rows=n,cols=n;
       int x=0,y=0;
       while(x<cols-x&&y<rows-y)
       {
        int end=cols-x-1;
        if(x<end)
        {
         for(int i=x;i<end;i++)
             matrix[y][i]=startNum++;
        }

        end=rows-y-1;
        if(y<end)
        {
         for(int j=y;j<end;j++)
             matrix[j][cols-x-1]=startNum++;;
        }
        int start=cols-x-1;
        if(start>x)
        {
         for(int i=start;i>x;i--)
             matrix[rows-y-1][i]=startNum++;;
        }
        start=rows-y-1;
        if(start>y)
        {
         for(int j=start;j>y;j--)
             matrix[j][x]=startNum++;
        }
        if(2*x+1==cols&&2*y+1==rows)
         matrix[y][x]=startNum++;;
        x++;
        y++;
       }
       return matrix;
   }
```


leetcode_Unique Paths II

By mnmlist,2015-9-2 9:35:19

描述：

Follow up for "Unique Paths":

Now consider if some obstacles are added to the grids. How many unique paths would there be?

An obstacle and empty space is marked as `1` and `0` respectively in the grid.

For example,

There is one obstacle in the middle of a 3x3 grid as illustrated below.

```
[
 [0,0,0],
 [0,1,0],
 [0,0,0]
]
```

The total number of unique paths is `2`.

Note: m and n will be at most 100.

思路：

1.考虑障碍的情况下，给每个元素arrNum[i]赋初值
2.如果第一行或第一列某处有障碍，从该处开始改行的每个元素arrNum[i]赋值为0

3.考虑到有障碍的情况，给每个元素赋值

代码：

```
public int uniquePathsWithObstacles(int[][] obstacleGrid) {
  if(obstacleGrid==null)
   return 0;
  if(obstacleGrid.length==0)
   return 0;
  if(obstacleGrid[0][0]==1)
   return 0;
 int m=obstacleGrid.length;
 int n=obstacleGrid[0].length;
 boolean flagRow=false;
 boolean flagCol=false;
 int arrNum[][]=new int[m][];
```

By mnmlist,2015-9-2 9:35:19

```
for(int i=0;i<m;i++)
 arrNum[i]=new int[n];
for(int i=0;i<m;i++)//arrNum[i]
//arrNum[i]0
{
 if(obstacleGrid[i][0]==1)
 {
  flagRow=true;
  while(i<m)
  {
   arrNum[i][0]=0;
   i++;
  }
  break;
 }else
  arrNum[i][0]=1;
}
for(int i=0;i<n;i++)//arrNum[i]
//arrNum[i]0
{
 if(obstacleGrid[0][i]==1)
 {
  flagCol=true;
  while(i<n)
  {
   arrNum[0][i]=0;
   i++;
  }
  break;
 }else
  arrNum[0][i]=1;
}
if(m==1)
{
 if(flagCol)
  return 0;
 else
  return 1;
}
if(n==1)
{
 if(flagRow)
  return 0;
 else
  return 1;
}
for(int i=1;i<m;i++)//
{
 for(int j=1;j<n;j++)
 {
  if(obstacleGrid[i][j]==1)
   arrNum[i][j]=0;
  else
   arrNum[i][j]=arrNum[i-1][j]+arrNum[i][j-1];
 }
```

By mnmlist,2015-9-2 9:35:19

```
 }
 return arrNum[m-1][n-1];
}
```


leetcode_Unique Paths

描述：

A robot is located at the top-left corner of a m x n grid (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

How many possible unique paths are there?

Above is a 3 x 7 grid. How many possible unique paths are there?

Note: m and n will be at most 100.

思路：

这是一个类似斐波那契数列的问题，由于robot仅可以向 右走或向下走，所以arrNum[i][j]=arrNum[i-1][j]+arrNum[i][j-1];

代码：

```
public int uniquePaths(int m, int n) {
 if(m==1||n==1)
  return 1;
 int arrNum[][]=new int[m][];
 for(int i=0;i<m;i++)
  arrNum[i]=new int[n];
 for(int arr[]:arrNum)
  Arrays.fill(arr, 1);
 for(int i=1;i<m;i++)
 {
```

By mnmlist,2015-9-2 9:35:19

```
  for(int j=1;j<n;j++)
   arrNum[i][j]=arrNum[i-1][j]+arrNum[i][j-1];
 }
 return arrNum[m-1][n-1];
}
```


leetcode_Excel Sheet Column Title

描述：

Given a positive integer, return its corresponding column title as appear in an Excel sheet.

For example:

```
1 -> A
2 -> B
3 -> C
...
26 -> Z
27 -> AA
28 -> AB
```

思路：

比字母串转换成数字稍稍复杂些，每次获取字符的时候temp=n-1;是一个需要留意的地方，否则很容易搞错

代码：

```
public String convertToTitle(int n) {
       char ch;
       StringBuilder sb=new StringBuilder();
       int temp=0;
       while(n!=0)
       {
```

By mnmlist,2015-9-2 9:35:19

```
        temp=n-1;
            ch=(char)(temp%26+'A');
            n=temp/26;
            sb.insert(0, ch);
        }
        return sb.toString();
    }
```


leetcode_Excel Sheet Column Number

描述：

Related to question Excel Sheet Column Title
Given a column title as appear in an Excel sheet, return its corresponding column number.
For example:

```
A -> 1
B -> 2
C -> 3
...
Z -> 26
AA -> 27
AB -> 28
```

思路：

很简单，就是类似数字字符串转换成整数的那种逻辑

代码：

```
public int titleToNumber(String s) {
        int titleNum=0;
        int len=s.length();
        for(int i=0;i<len;i++)
```

By mnmlist,2015-9-2 9:35:19

```
        {
            titleNum=titleNum*26+s.charAt(i)-'A'+1;
        }
        return titleNum;
    }
```


leetcode_Compare Version Numbers

描述：

Compare two version numbers version1 and version2.
If version1 > version2 return 1, if version1 < version2 return -1, otherwise return 0.

You may assume that the version strings are non-empty and contain only digits and the . character.
The . character does not represent a decimal point and is used to separate number sequences.
For instance, 2.5 is not "two and a half" or "half way to version three", it is the fifth second-level revision of the second first-level revision.

Here is an example of version numbers ordering:

```
0.1 < 1.1 < 1.2 < 13.37
```

思路：

1.用split将' . '分隔的数字提取出并放到一个数组中

2.然后分别比较数组中的各位数字即可。

代码：

```
public int compareVersion(String version1, String version2) {
        String Arr1[]=version1.split("\\.");
        String Arr2[]=version2.split("\\.");
        int len1=Arr1.length;
```

By mnmlist,2015-9-2 9:35:19

```
        int len2=Arr2.length;
        int min=len1<len2?len1:len2;
        int num1=0,num2=0;
        for(int i=0;i<min;i++)
        {
            num1=Integer.valueOf(Arr1[i]);
            num2=Integer.valueOf(Arr2[i]);
            if(num1!=num2)
            {
                if(num1>num2)
                    return 1;
                else
                    return -1;
            }
        }
        if(len1>len2)
        {
            for(int i=min;i<len1;i++)
            {
                num1=Integer.valueOf(Arr1[i]);
                if(num1!=0)
                    return 1;
            }
        }
        else if(len1<len2)
        {
            for(int i=min;i<len2;i++)
            {
                num1=Integer.valueOf(Arr2[i]);
                if(num1!=0)
                    return -1;
            }
        }
        return 0;
    }
```


leetcode_Gas Station

描述：

There are N gas stations along a circular route, where the amount of gas at station i is gas[i].

You have a car with an unlimited gas tank and it costs cost[i] of gas to travel from station i to its next station (i+1). You begin the journey with an empty tank at one of the gas stations.

By mnmlist,2015-9-2 9:35:19

Return the starting gas station's index if you can travel around the circuit once, otherwise return -1.

Note:
The solution is guaranteed to be unique.

思路：

这是一个动态规划的题目，假设能跑完全程，最后邮箱中的油肯定还有剩余，由于是一个环形的路线，所以理论来讲，可以从任何地方开始。所以，可以用两个变量来表示这道题目，sum表示全程的油的剩余，而index来表示从何处开始。
1.从任何地方可以开始本次的行程，而sum+=gas[i]-cost[i],若sum<0,说明从index开始是不合适的,换成从i+1开始，直至结尾
2.若最后sum>=0,说明路线存在且以index为起点的路线为一个可行的路线。

代码：

```
public int canCompleteCircuit(int[] gas, int[] cost) {
        int sum=0,tempSum=0,num=0,stationNum=0;
        int len=gas.length;
        for(int i=0;i<len;i++)
        {
            num=gas[i]-cost[i];
            sum+=num;
            tempSum+=num;
            if(tempSum<0)
            {
                stationNum=(i+1)%len;
                tempSum=0;
            }
        }
        if(sum>=0)
            return stationNum;
        return -1;
    }
```


leetcode_Length of Last Word

By mnmlist,2015-9-2 9:35:19

描述：

Given a string s consists of upper/lower-case alphabets and empty space characters `' '`, return the length of last word in the string.

If the last word does not exist, return 0.

Note: A word is defined as a character sequence consists of non-space characters only.

For example,
Given s = `"Hello World"`,
return `5`.

思路：

1.s==null return 0

2.s=s.trim(),s.length()==0 return 0;

3.从后面开始计算字母的数量，直到遇到空格为止

代码：

```
public int lengthOfLastWord(String s) {
        if(s==null)
            return 0;
        s=s.trim();
        int len=s.length();
        if(len==0)
            return 0;
        int i=0;
        for(i=len-1;i>=0;i--)
        {
            if(!isAlp(s.charAt(i)))
                break;
        }
        return len-1-i;
    }
    public boolean isAlp(char ch)
    {
        if((ch>='A'&&ch<='Z')||(ch>='a'&&ch<='z'))
            return true;
        return false;
    }
```

By mnmlist,2015-9-2 9:35:19

leetcode_Find Minimum in Rotated Sorted Array

描述：

Suppose a sorted array is rotated at some pivot unknown to you beforehand.
(i.e., `0 1 2 4 5 6 7` might become `4 5 6 7 0 1 2`).
Find the minimum element.
You may assume no duplicate exists in the array.

思路：

这道题由于要求对数的时间复杂度，所以考虑用二分查找，而二分查找需要数组是有序的。而本文所给的数组是循环移位后的有序数组，所以假设能够确定数字的顺寻就可以应用二分查找，本文的题眼也就在这里。由于所给的数据是循环移位后的有序数组，所以本数组至少有一半的元素是有序的，找出有序的那一般即可应用二分查找。

1.首先取mid=（start+end）/2,将arr[start]和arr[mid]比较，如果arr[start]==arr[mid]，找到元素

2.如果arr[start]<arr[mid]，则前半部分元素是有序的，由于前半部分有序，若arr[start]<=target<arr[mid],则假设元素存在则一定在前半部分，end=mid-1；若target>arr[mid],则元素如果存在，则存在后半部分，start=mid+1

3.如果arr[start]>arr[mid]，则后半部分肯定有有序的，推理过程通上，balabala。。。

4.while（start<=end)，继续循环

By mnmlist,2015-9-2 9:35:19

代码：

leetcode_Find Minimum in Rotated Sorted Array II

描述：

> Follow up for "Find Minimum in Rotated Sorted Array":
> What if duplicates are allowed?
>
> Would this affect the run-time complexity? How and why?

Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., `0 1 2 4 5 6 7` might become `4 5 6 7 0 1 2`).

Find the minimum element.

The array may contain duplicates.

思路：

这道题由于要求对数的时间复杂度，所以考虑用二分查找，而二分查找需要数组是有序的。而本文所给的数组是循环移位后的有序数组，所以假设能够确定数字的顺寻就可以应用二分查找，本文的题眼也就在这里。由于所给的数据是循环移位后的有序数组，所以本数组至少有一半的元素是有序的，找出有序的那一般即可应用二分查找。

1.首先取mid=（start+end）/2,将arr[start]和arr[mid]比较，如果arr[start]==arr[mid]，找到元素

2.如果arr[start]<arr[mid]，则前半部分元素是有序的，由于前半部分有序，若arr[start]<=target<arr[mid],则假设元素存在则一定在前半部分，end=mid-1；若target>arr[mid],则元素如果存在，则存在后半部分，start=mid+1

By mnmlist,2015-9-2 9:35:19

3.如果arr[start]>arr[mid]，则后半部分肯定有有序的，推理过程通上，balabala。。。

4.arr[start]==arr[mid]，说明存在相同的元素，start=start+1；

5.while（start<=end)，继续循环

代码：

```
public int findMin(List<Integer>nums) {
 int len = nums.size();
 int start = 0, end = len - 1;
 int mid = (start + end) / 2;
 int target = nums.get(mid);
 int min = findMinNum(nums, start, end, target);
 return min;
}

public int findMinNum(List<Integer>nums, int start, int end, int target) {
    int startNum= nums.get(start),midNum=0;
 if (start == end)
  return target < startNum ? target : startNum ;
 int mid = (start + end) / 2;
 midNum=nums.get(mid);
 target=target<midNum?target:midNum;
 if (startNum  <midNum) {
  target = target < startNum? target : startNum ;
  start = mid + 1;
 } else if(startNum>midNum)
 {
     midNum=nums.get(mid+1);
  if(mid+1<=end)
   target=target<midNum?target:midNum;
  end=mid-1;
 }else
     start=start+1;
 return findMinNum(nums, start, end, target);
}
```

By mnmlist,2015-9-2 9:35:19

leetcode_Candy

描述：

There are N children standing in a line. Each child is assigned a rating value.

You are giving candies to these children subjected to the following requirements:

- Each child must have at least one candy.
- Children with a higher rating get more candies than their neighbors.

What is the minimum candies you must give?

思路：

1.新建一个数组arr，给每个元素赋值为1，即每个孩子的糖果至少为1个

2.从左至右，假设rating[i]<rating[i+1]，则arr[i+1]=arr[i]+1;

3.从右至左，若rating[i]<rating[i-1]，则比较arr[i-1]=max(arr[i-1],arr[i]+1)

4.从左至右，统计糖果的数量即可。

代码：

```
public int candy(int[] ratings) {
       if(ratings==null)
           return 0;
       if(ratings.length<=1)
           return ratings.length;
       int count=0;
       int len=ratings.length;
       int candyNum[]=new int[len];
       Arrays.fill(candyNum, 1);
       for(int i=1;i<len;i++)
       {
        if(ratings[i]>ratings[i-1])
         candyNum[i]=candyNum[i-1]+1;
       }
       for(int i=len-1;i>=1;i--)
       {
```

By mnmlist,2015-9-2 9:35:19

```
        if(ratings[i]<ratings[i-1])
        {
         if(candyNum[i-1]<candyNum[i]+1)
          candyNum[i-1]=candyNum[i]+1;
        }


       }
       for(int i=0;i<len;i++)
        count+=candyNum[i];
       return count;
   }
```


leetcode_Divide Two Integers

描述：

Divide two integers without using multiplication, division and mod operator.
If it is overflow, return MAX_INT.

思路：

用减法

代码：

```
public class Solution {
   int count=0;
long dividendTemp=0,divisorTemp=0;
public int divide(int dividend, int divisor) {
       if(divisor==0)
           return 0;
       if(dividend==Integer.MIN_VALUE&&divisor==-1)
           return Integer.MAX_VALUE;
       int countFlag=0;
       long dividend1=dividend;
```

By mnmlist,2015-9-2 9:35:19

```
        long divisor1=divisor;
        if(dividend1<0)
        {
           countFlag++;
           dividend1=-dividend1;
        }
        if(divisor1<0)
        {
            countFlag++;
            divisor1=-divisor1;
        }
        dividendTemp=dividend1;
        divisorTemp=divisor1;
        while(dividend1>=divisor1+divisor1)
        {
         dividend1=getDivid(dividend1);
        }
        if(dividend1>=divisor1)
         count++;
        if(countFlag==1)
            return -count;
        return count;
    }
public long getDivid(long dividend)
{
 int temp=1;
 while(dividend>=(divisorTemp<<temp))
  temp++;
 temp--;
 dividend-=(divisorTemp<<temp);
 count+=1<<temp;
 return dividend;
}
}
```


leetcode_Trapping Rain Water

描述：

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

For example,
Given [0,1,0,2,1,0,1,3,2,1,2,1], return 6.

By mnmlist,2015-9-2 9:35:19

The above elevation map is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped. Thanks Marcos for contributing this image!

思路：

动态规划题目

1.从左至右算出每个点的左方的最大值

2.从右至左算出每个点的右方的最大值

3.从左至右循环sum+=min(leftMax[i]+rightMax[i])-arr[i]

4.sum的值就是所能储存的最大水量

代码：

```
public int trap(List<Integer> height) {
        int count=0;
        if(height==null)
            return 0;
        if(height.size()==0)
            return 0;
        int len=height.size();
        int arrLeft[]=new int[len];
        int arrRight[]=new int[len];
        int max=0;
        int heigh=0;
        for(int i=0;i<len;i++)
        {
            heigh=height.get(i);
            if(heigh>max)
                max=heigh;
            arrLeft[i]=max;
        }
        max=0;
        heigh=0;
        for(int i=len-1;i>=0;i--)
        {
            heigh=height.get(i);
            if(heigh>max)
```

By mnmlist,2015-9-2 9:35:19

```
                max=heigh;
            arrRight[i]=max;
        }
        int newLen=len-1;
        int min=0;
        for(int i=1;i<newLen;i++)
        {
            heigh=height.get(i);
            min=arrLeft[i]<arrRight[i]?arrLeft[i]:arrRight[i];
            if(min>heigh)
                count+=min-heigh;
        }
        return count;
    }
```


leetcode_Reverse Words in a String

描述：

Given an input string, reverse the string word by word.

For example,
Given s = "the sky is blue",
return "blue is sky the".

Update (2015-02-12):
For C programmers: Try to solve it in-place in O(1) space.

click to show clarification.

Clarification:

- What constitutes a word?
  A sequence of non-space characters constitutes a word.
- Could the input string contain leading or trailing spaces?
  Yes. However, your reversed string should not contain leading or trailing spaces.
- How about multiple spaces between two words?
  Reduce them to a single space in the reversed string.

思路：

1.trim（ ）+去除串内多余的空格

By mnmlist,2015-9-2 9:35:19

2.将串内每个单词反转

3.将整个句子反转

代码：

```
public String reverseWords(String s) {
 s=s.trim();
 if(!s.equals(""))
 {
  char arr[]=s.toCharArray();
  int index=1;
  for(int i=1;i<arr.length;i++)
  {
   if(arr[i-1]==arr[i]&&arr[i]==' ')
    continue;
   else
    arr[index++]=arr[i];

  }
  s=new String(arr,0,index);
 }else {
  return "";
 }
 StringBuilder sBuilder=new StringBuilder(s);
 char ch;
 int i=0,j=s.length()-1;
 int start=0,end=0;
 for(i=0;i<sBuilder.length();i++)
 {
  start=i;
  while(sBuilder.charAt(i)!=' '&&i!=j)
   i++;
  end=i;
  if(end!=j)
   end--;
  while(start<end)
  {
   ch=sBuilder.charAt(start);
   sBuilder.setCharAt(start, sBuilder.charAt(end));
   sBuilder.setCharAt(end, ch);
   start++;
   end--;
  }
 }
 sBuilder.reverse();
       return sBuilder.toString();
   }
```

By mnmlist,2015-9-2 9:35:19

leetcode_Gray Code

描述：

The gray code is a binary numeral system where two successive values differ in only one bit.

Given a non-negative integer n representing the total number of bits in the code, print the sequence of gray code. A gray code sequence must begin with 0.

For example, given n = 2, return [0,1,3,2]. Its gray code sequence is:

```
00 - 0
01 - 1
11 - 3
10 - 2
```

Note:
For a given n, a gray code sequence is not uniquely defined.

For example, [0,2,3,1] is also a valid gray code sequence according to the above definition.

For now, the judge is able to judge based on one instance of gray code sequence. Sorry about that.

思路：

1.n位格雷码对应有1<<n个数

2.每个数字对应的格雷码的对应的计算公式为：i^(i/2)

代码：

```
public List<Integer> grayCode(int n) {
       List<Integer>list=new ArrayList<Integer>();
       if(n<0)
```

By mnmlist,2015-9-2 9:35:19

```
            return list;
        n=1<<n;
        for(int i=0;i<n;i++)
            list.add((i^(i/2)));
        return list;
    }
```


leetcode_Happy Number

描述：

Write an algorithm to determine if a number is "happy".

A happy number is a number defined by the following process: Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1. Those numbers for which this process ends in 1 are happy numbers.

Example: 19 is a happy number

- $1^2 + 9^2 = 82$
- $8^2 + 2^2 = 68$
- $6^2 + 8^2 = 100$
- $1^2 + 0^2 + 0^2 = 1$

思路：

很简单的思路，用一个HashSet存储计算的结果直至出现循环，剩下的就是计算一个整数的各个位数的数字和对这些数字进行求和了。

代码：

```
public class Solution {
    int digitsCount=0;
    public boolean isHappy(int n) {
        int digits[]=new int[15];
```

By mnmlist,2015-9-2 9:35:19

```
        int digitsCount=0;
        int num=0;
        Set<Integer>set=new HashSet<Integer>();
        while(true)
        {
            digitsCount=getDigits(n,digits);
            num=getSum(digits,digitsCount);
            if(num==1)
                return true;
            else
            {
                if(set.contains(num))
                    break;
                else
                    set.add(num);
            }
            n=num;
        }
        return false;
    }
    public int getDigits(int n,int digits[])
    {
        int temp=0;
        int count=0;
        while(n!=0)
        {
            temp=n%10;
            n/=10;
            if(temp!=0)
                digits[count++]=temp;
        }
        return count;
    }
    public int getSum(int digits[],int n)
    {
        int sum=0;
        for(int i=0;i<n;i++)
            sum+=digits[i]*digits[i];
        return sum;
    }
}
```


leetcode_Remove Linked List Elements

By mnmlist,2015-9-2 9:35:19

描述：

Remove all elements from a linked list of integers that have value val.

Example
Given: 1 --> 2 --> 6 --> 3 --> 4 --> 5 --> 6, val = 6
Return: 1 --> 2 --> 3 --> 4 --> 5

思路：

直接创建一个头结点即可，从头节点开始，while(p.next!=null)
if（p.next.data=val)p.next=p.next.next;

代码：

```
/**
* Definition for singly-linked list.
* public class ListNode {
*     int val;
*     ListNode next;
*     ListNode(int x) { val = x; }
* }
*/
public class Solution {
   public ListNode removeElements(ListNode head, int val) {
        if(head==null)
            return head;
        ListNode preNode=new ListNode(0);
        preNode.next=head;
        ListNode pNode=preNode;
        while(pNode.next!=null)
        {
            if(pNode.next.val==val)
                pNode.next=pNode.next.next;
            else
                pNode=pNode.next;
        }
        return preNode.next;
   }
}
```

By mnmlist,2015-9-2 9:35:19

leetcode_Best Time to Buy and Sell Stock III

描述：

Say you have an array for which the ith element is the price of a given stock on day i.

Design an algorithm to find the maximum profit. You may complete at most two transactions.

思路：

给两次交易机会，让求最大的利润，先从左向右求到当前节点的最大利润并将其存储到一个数组profit1[ ]中，然后从右向左求到当前节点的最大利润并将其存储到一个数组profit2[ ]中，然后从前向后遍历求到某个节点时其左右部分的利润相加，然后和当前最大利润相比从而得出最大利润。

代码：

```
public int maxProfit(int[] prices) {
        if(prices==null)
            return 0;
        int len=prices.length;
        if(len==0)
            return 0;
        int maxProfit=0;
        int tempPre=prices[0];
        int tempCur=0;
        prices[0]=0;
        for(int i=1;i<len;i++)
        {
         tempCur=prices[i];
         prices[i]=prices[i]-tempPre;
         tempPre=tempCur;
        }
  maxProfit=getMaxProfit(prices);
        return maxProfit;
    }
```

By mnmlist,2015-9-2 9:35:19

```
    public int getMaxProfit(int profit[])
    {
        int len=profit.length;
        int profitFromEnd[]=new int[len];
        for(int i=0;i<len;i++)
            profitFromEnd[i]=profit[i];
        for(int i=1;i<len;i++)
        {
            if(profit[i-1]>0)
                profit[i]=profit[i-1]+profit[i];
        }
        for(int i=len-2;i>=0;i--)
        {
            if(profitFromEnd[i+1]>0)
                profitFromEnd[i]=profitFromEnd[i]+profitFromEnd[i+1];
        }
        int max=profit[0];
        for(int i=1;i<len;i++)
        {
            if(max<profit[i])
                max=profit[i];
            profit[i]=max;
        }
        max=profitFromEnd[len-1];
        for(int i=len-2;i>=0;i--)
        {
            if(max<profitFromEnd[i])
                max=profitFromEnd[i];
            profitFromEnd[i]=max;
        }
        int maxProfit=0;
        int tempMaxProfit=0;
        int count=len-1;
        for(int i=0;i<count;i++)
        {
            tempMaxProfit=profit[i]+profitFromEnd[i+1];
            if(maxProfit<tempMaxProfit)
                maxProfit=tempMaxProfit;
        }
        return maxProfit;
    }
```


leetcode_Best Time to Buy and Sell Stock II

By mnmlist,2015-9-2 9:35:19

描述：

Say you have an array for which the ith element is the price of a given stock on day i.

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times). However, you may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

思路：

由于交易次数不限，所以将其所有大于0的利润相加即可，为何要这样出题？

代码：

```
public int maxProfit(int[] prices) {
       if(prices==null)
           return 0;
       int len=prices.length;
       if(len==0)
           return 0;
       int temp=0;
       int maxProfit=0;
       for(int i=1;i<len;i++)
       {
           temp=prices[i]-prices[i-1];
        if(temp>0)
               maxProfit+=temp;
       }
       return maxProfit;
   }
```


leetcode_Best Time to Buy and Sell Stock

描述：

By mnmlist,2015-9-2 9:35:19

Say you have an array for which the ith element is the price of a given stock on day i.

If you were only permitted to complete at most one transaction (ie, buy one and sell one share of the stock), design an algorithm to find the maximum profit.

思路：

简而言之，这就是一个求数组最大和的问题，动态规划

代码：

```
public int maxProfit(int[] prices) {
       if(prices==null)
           return 0;
       int len=prices.length;
       if(len==0)
           return 0;
       int maxProfit=0;
       int tempPre=prices[0];
       int tempCur=0;
       prices[0]=0;
       for(int i=1;i<len;i++)//caculate the profit of a day
       {
        tempCur=prices[i];
        prices[i]=prices[i]-tempPre;
        tempPre=tempCur;
       }
       for(int i=1;i<len;i++)//caculate the max total profit
       {
           if(prices[i-1]>=0)
            prices[i]=prices[i-1]+prices[i];
           if(prices[i]>maxProfit)
               maxProfit=prices[i];
       }
       return maxProfit;
   }
```


leetcode_Factorial Trailing Zeroes

By mnmlist,2015-9-2 9:35:19

描述：

Given an integer n, return the number of trailing zeroes in n!.

Note: Your solution should be in logarithmic time complexity.

思路：

一般来讲，.5*2 .5 *4 .5*8 .....可以获得末位是0的情况，同理25*2 25*4 ...所以仅需递归地求出从1到n中，5的个数，25的个数，125的个数，625的个数。。。即可

代码：

```
public int trailingZeroes(int n) {
       int count=0;
       while(n!=0)
       {
           count+=n/5;
           n/=5;
       }
       return count;
   }
```


leetcode_Reverse Bits

描述：

Reverse bits of a given 32 bits unsigned integer.

For example, given input 43261596 (represented in binary as 00000010100101000001111010011100), return 964176192 (represented in binary as 00111001011110000010100101000000).

Follow up:
If this function is called many times, how would you optimize it?

By mnmlist,2015-9-2 9:35:19

Related problem: Reverse Integer

思路：

和反转整数不太相同的是将32位无符号整数按二进制位的形式反转没法直接用%和/方法，但是对于二进制位我们可以直接用位操作，从而达到类似整数用%和/的效果。

代码：

```
public int reverseBits(int n) {
 if(n==0)
        return 0;
       boolean flag=false;
       int temp=0;
       int num=0;
       for(int i=0;i<=31;i++)
       {
          temp= n&(1<<i);
          if(!flag&&temp!=0)
            flag=true;
        if(flag)
        {
         if(temp!=0)
          num=num*2+1;
         else
          num=num*2;
        }

       }
       return num;
   }
```


leetcode_Bitwise AND of Numbers Range

描述：

Given a range [m, n] where 0 <= m <= n <= 2147483647, return the bitwise AND of all numbers in this range, inclusive.

By mnmlist,2015-9-2 9:35:19

For example, given the range [5, 7], you should return 4.

思路：

由于相邻的两个数最低位肯定有0有1，所以直接and肯定为0，所以可以通过直接and来和向右移位获得一个区间内的相同的位数，最后再通过向左移位获得一个区间所有数字相与的结果。

代码：

```
public int rangeBitwiseAnd(int m, int n) {
        int offset=0;
        while(m!=n)
        {
            m>>=1;
            n>>=1;
            offset++;
        }
        return m<<offset;
    }
```


leetcode_Find Peak Element

描述：

A peak element is an element that is greater than its neighbors.

Given an input array where `num[i] ≠ num[i+1]`, find a peak element and return its index.

The array may contain multiple peaks, in that case return the index to any one of the peaks is fine.

You may imagine that `num[-1] = num[n] = -∞`.

For example, in array `[1, 2, 3, 1]`, 3 is a peak element and your function should return the index number 2.

click to show spoilers.

Note:
Your solution should be in logarithmic complexity.

Credits:
Special thanks to @ts for adding this problem and creating all test cases.

By mnmlist,2015-9-2 9:35:19

思路：

开始感觉很难下手，数组不是有序的，时间要求是log(n)，最后想到一种方法，mid=(left+right)/2;if(num[mid]<num[mid+1])left=mid+1;else right=mid,这种方式还真的可以找到其中的一个peak，所以根据时间复杂度的提示反而提醒了思路。

代码：

```
public int findPeakElement(int[] num) {
        int left=0,right=num.length-1;
        int mid=0;
        int num1=0,num2=0;
        while(left<=right){
            if(left==right)
                return left;
            mid=(left+right)/2;
            if(num[mid]<num[mid+1])
                left=mid+1;
            else
                right=mid;
        }
        return 0;
    }
```


leetcode_Isomorphic Strings

描述：

Given two strings s and t, determine if they are isomorphic.

Two strings are isomorphic if the characters in s can be replaced to get t.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character but a character may map to itself.

For example,
Given "egg", "add", return true.

Given "foo", "bar", return false.

By mnmlist,2015-9-2 9:35:19

Given "paper", "title", return true.

Note:
You may assume both s and t have the same length.

思路：

题目的大意是用s中字母对应的字母可以组成t中的串，简而言之，就是s中的字母对应在t中的字母是唯一的，而t中的字母对应在s中也是唯一的，问题解决。

代码：

```
public boolean isIsomorphic(String s, String t) {
        if(s==null)
            return true;
        int len=s.length();
        if(len==1)
            return true;
        boolean flag=true;
        flag=isIso(s, t);//the letter in s must correspond to a unique letter in t
        if(flag)
         flag=isIso(t, s);//also,the letter in t must correspond to a unique letter in s
        return flag;
    }
public boolean isIso(String s, String t)
{
 int len=s.length();
 char arr[]=new char[257];
 Arrays.fill(arr, '%');
        char ch_s,ch_t,temp;
        for(int i=0;i<len;i++)
        {
            ch_s=s.charAt(i);
            ch_t=t.charAt(i);
            if(arr[ch_s]!='%')//judge if ch_s has already appeared or not
            {
                temp=arr[ch_s];
                if(temp!=ch_t)//if  has appeared but the only ch_s corresponded to differe
                    return false;
            }else
             arr[ch_s]=ch_t;
        }
        return true;
}
```


leetcode_Add Binary

By mnmlist,2015-9-2 9:35:19

描述：

Given two binary strings, return their sum (also a binary string).

For example,
a = "11"
b = "1"
Return "100".

思路：

将字符串从后向前进行相加，最后有进位的话再创造新的位数，最后将字符串反转，输出即可。

代码：

```
public String addBinary(String a, String b) {
        int len1=a.length();
        int len2=b.length();
        String string=null;
        if(len1>len2)
        {
         string=a;
         a=b;
         b=string;
        }
        len1=a.length();
        len2=b.length();
        StringBuilder sBuilder=new StringBuilder();
        int temp=0;
        boolean flag=false;
        int span=len2-len1;
        int indexA=0;
        for(int i=len2-1;i>=0;i--)
        {
         temp=0;
         indexA=i-span;
         if(indexA>=0)
          temp+=a.charAt(indexA)-'0';
         temp+=b.charAt(i)-'0';
         if(flag)
         {
             temp+=1;
             flag=false;
         }
         sBuilder.append(temp%2);
         if(temp>=2)
          flag=true;
        }
        if(flag)
```

By mnmlist,2015-9-2 9:35:19

```
        sBuilder.append('1');
       return sBuilder.reverse().toString();
   }
```


leetcode_Letter Combinations of a Phone Number

描述：

Given a digit string, return all possible letter combinations that the number could represent.
A mapping of digit to letters (just like on the telephone buttons) is given below.

```
Input:Digit string "23"
Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].
```

Note:
Although the above answer is in lexicographical order, your answer could be in any order you want.

思路：

本题目要实现的效果就是产生一系列广义有序的字符串序列，本来想假如知道digits字符串长度的话，可以搞一个n层循环，但是digits的长度是未知的。想了好久，想到了用递归来实现本题目要实现的功能，每个数字循环到数字所代表字符的个数大小时，返回上一级递归，然后再在上一级的循环层次计数上再加1即可。

当字符的长度等于digits的长度时，说明已经产生一组有效的字符串，存储起来，然后将最后一个字符删除，继续循环。。。。

最后，当index<0时，说明整个循环已经结束，over！但是在具体实现的过程中，递归理解起来比较复杂，具体如何结束，这个判断条件需要特别注意下，假如只是简单的return可能达不到应有的效果。

By mnmlist,2015-9-2 9:35:19

总之，这题目搞了两个多小时，感觉有点乱。

代码：

```
boolean flag=true;
int digitsLength=0;
int arr[];
List<String>list=new ArrayList<String>();
StringBuilder sBuilder=new StringBuilder();
Map<Integer, String>map=new HashMap<Integer, String>();
public List<String> letterCombinations(String digits) {
 if(digits==null||digits.length()==0)
  return list;
       digitsLength=digits.length();
       arr=new int[digitsLength];
       int digitsArr[]=new int[]{2,3,4,5,6,7,8,9};
       String strArr[]=new String[]{"abc","def","ghi","jkl","mno","pqrs","tuv","wxyz"};
       int len=digitsArr.length;
       Arrays.fill(arr, 0);
       for(int i=0;i<len;i++)
        map.put(digitsArr[i], strArr[i]);//
       combinations(digits,0);
       return list;
   }
public void combinations(String digits,int index)
{
 if(!flag)//flagfalse
  return;
 if(index<0)
 {
  flag=false;//index<0
  return;
 }
 int number=digits.charAt(index)-'0';
 String string=map.get(number);//
 int len=string.length();
 int upLevelIndex=0;
 if(index<digitsLength-1)//
 {
  if(arr[index]==len)//
  {
   arr[index]=0;
   upLevelIndex=sBuilder.length()-1;//
   if(upLevelIndex>=0)
   {
    sBuilder.deleteCharAt(upLevelIndex);
    combinations(digits,index-1);
   }else {
    flag=false;//
    return;
```

By mnmlist,2015-9-2 9:35:19

```
   }
  }else{
   sBuilder.append(string.charAt(arr[index]));
   arr[index]++;
   combinations(digits,index+1);
  }
 }
 else//
 {
  for(int i=0;i<len;i++)
  {
   sBuilder.append(string.charAt(i));
   list.add(sBuilder.toString());//
   sBuilder.deleteCharAt(sBuilder.length()-1);//
  }
  upLevelIndex=sBuilder.length()-1;//
  if(upLevelIndex>=0)
  {
   sBuilder.deleteCharAt(upLevelIndex);
   combinations(digits,index-1);
  }else {
   flag=false;//
   return;
  }

 }
}
```

结果：

leetocde_Binary Tree Right Side View

描述：

Given a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom.

For example:
Given the following binary tree,

```
 1      <---
/ \
```

By mnmlist,2015-9-2 9:35:19

```
2   3     <---
\   \
 5   4    <---
```

You should return [1, 3, 4].

思路：

按层序遍历的思路，每次只保存该层的最后一个元素即可。

代码：

```
public List<Integer> rightSideView(TreeNode root) {
       List<Integer>listReturn=new ArrayList<Integer>();
       if(root==null)
        return listReturn;
       List<TreeNode>list=new ArrayList<TreeNode>();
       List<TreeNode>temp=new ArrayList<TreeNode>();
       list.add(root);
       TreeNode node=null;
       listReturn.add(root.val);
       while(list.size()!=0)
       {
        for(int i=0;i<list.size();i++)
        {
         node=list.get(i);
         if(node.left!=null)
          temp.add(node.left);
         if(node.right!=null)
          temp.add(node.right);
        }
        if(temp.size()>0)
         listReturn.add(temp.get(temp.size()-1).val);
        list.clear();
           list=temp;
        temp=new ArrayList<TreeNode>();
       }
    return listReturn;
  }
```

结果：

By mnmlist,2015-9-2 9:35:19

leetcode_Number of 1 Bits

描述：

Write a function that takes an unsigned integer and returns the number of ’ 1' bits it has (also known as the Hamming weight).

For example, the 32-bit integer ’ 11' has binary representation 00000000000000000000000000001011, so the function should return 3.

Credits:
Special thanks to @ts for adding this problem and creating all test cases.

思路：

java中如何表示无符号整数呢，很伤，那就用C写吧。二进制与运算和二进制循环移位搞定

代码：

```
int hammingWeight(uint32_t n) {
   int count=0;
   while(n>0)
   {
       count+=(n&1);
       n=n>>1;
   }
   return count;
}
```

结果：

By mnmlist,2015-9-2 9:35:19

leetcode_Climbing Stairs

描述：

You are climbing a stair case. It takes n steps to reach to the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

思路：

变形的斐波那契

代码：

```
public int climbStairs(int n) {
       if(n<3)
           return n;
       int arr[]=new int[n+1];
       arr[1]=1;
       arr[2]=2;
       for(int i=3;i<=n;i++)
           arr[i]=arr[i-1]+arr[i-2];
       return arr[n];
   }
```

结果：

leetcode_Min Stack

By mnmlist,2015-9-2 9:35:19

描述：

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

- push(x) -- Push element x onto stack.
- pop() -- Removes the element on top of the stack.
- top() -- Get the top element.
- getMin() -- Retrieve the minimum element in the stack.

思路：

本题目的解法是用到了两个栈，一个用来存元素，另一个用来存最小元素，元素入栈时和minStack栈里面的栈顶元素相比，小于栈顶元素则存入，大于栈顶元素则栈顶元素（当前元素中的最小值）入栈。其中，需要注意的是元素出栈时，要随时更新当前栈中的最小元素min=minStack.top()

代码：

```
Stack<Integer> st = new Stack<Integer>();
Stack<Integer> stMin = new Stack<Integer>();
int min = 0;

public void push(int x) {
 if (!st.isEmpty()) {
  if (min > x) {
   st.push(x);
   stMin.push(x);
   min = x;
  } else {
   st.push(x);
   stMin.push(min);
  }

 } else {
  st.push(x);
  stMin.push(x);
  min = x;
 }
}

public void pop() {
 st.pop();
 stMin.pop();
 if (!stMin.isEmpty())//
  min = stMin.peek();

}

public int top() {
 return st.peek();
}
```

By mnmlist,2015-9-2 9:35:19

```
public int getMin() {
 return stMin.peek();
}
```

结果：

leetcode_Validate Binary Search Tree

描述：

Given a binary tree, determine if it is a valid binary search tree (BST).

Assume a BST is defined as follows:

- The left subtree of a node contains only nodes with keys less than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- Both the left and right subtrees must also be binary search trees.

confused what `"{1,#,2,3}"` means? > read more on how binary tree is serialized on OJ.

OJ's Binary Tree Serialization:
The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.

Here's an example:

```
  1
 / \
2   3
   /
  4
   \
    5
```

The above binary tree is serialized as `"{1,2,3,#,#,4,#,#,5}"`.

思路：

由于二叉排序树的中序遍历序列是有序的，所以考虑在遍历的过程中通过判断遍历序列是

By mnmlist,2015-9-2 9:35:19

否有序从而来判断该排序树是否有效，但这又涉及到第一个元素的问题，所以可以设一个比Integer.MAX_INT还小的值作为参考值或者设一个flag来判断是否是第一个值，第一个值直接跳过即可。

代码：

```
public boolean isValidBST(TreeNode root) {
     if(root==null)
   return true;
     Stack<TreeNode>st=new Stack<TreeNode>();
     st.push(root);
     TreeNode top=null;
     double  temp=-2147483649.0;
     while(!st.empty())
     {
      top=st.peek();
      while(top.left!=null)
      {
       st.push(top.left);
       top=top.left;
      }
      while(top.right==null)
      {
       if(temp<top.val)
        temp=top.val;
       else
        return false;
       st.pop();
       if(!st.empty())
        top=st.peek();
       else
        break;
      }
      if(!st.empty())
      {
       if(temp<top.val)
        temp=top.val;
       else
        return false;
       st.pop();
       st.push(top.right);
      }

     }
     return true;
   }
```

结果：

By mnmlist,2015-9-2 9:35:19

leetcode_110_Balanced Binary Tree

描述：

Given a binary tree, determine if it is height-balanced.

For this problem, a height-balanced binary tree is defined as a binary tree in which the depth of the two subtrees of every node never differ by more than 1.

思路：

走了好多弯路，最后才发现直接根据定义来就可以了，左右子树的深度不超过1且左右子树都是平衡二叉树，就是这么简洁明快。

代码：

```
/**
* Definition for binary tree
* public class TreeNode {
*     int val;
*     TreeNode left;
*     TreeNode right;
*     TreeNode(int x) { val = x; }
* }
*/
public class Solution {
    public boolean isBalanced(TreeNode root) {
 if(root==null)
  return true;
 if(root.left==null&&root.right==null)
  return true;
 int len=maxDepth(root.left)-maxDepth(root.right);
 if(Math.abs(len)<=1&&isBalanced(root.left)&&isBalanced(root.right))
  return true;
 return false;
   }
public int maxDepth(TreeNode root) {
       if(root==null)
        return 0;
       if(root.left==null&&root.right==null)
        return 1;
```

By mnmlist,2015-9-2 9:35:19

```
        int num1=maxDepth(root.left),num2=maxDepth(root.right);
        int num=(num1>=num2?num1:num2)+1;
        return num;
    }
}
```

结果：

leetcode_96_Unique Binary Search Trees

描述：

Given n, how many structurally unique BST's (binary search trees) that store values 1...n?

For example,
Given n = 3, there are a total of 5 unique BST's.

```
 1     3   3    2   1
 \    /   /    /\   \
  3   2   1    1 3   2
 /   /    \         \
 2   1     2         3
```

思路：

这一道题呢，刚开始完全没有思路，拿笔在草纸上这样演算，那样演算，但是没有思路。这是一个二叉查找树，所以题眼很可能就是在这里，在网上一查还真是这样。由于这是一棵二叉查找树，所以当一棵树形态固定下来后，该树是唯一的。由于根节点的值大于左子树上的任何一个结点的值，小于右子树上任何一个结点的值，所以，当某个节点被当作树的根节点时，树的形态个数有num(root.left)*num(root.right)个，从num_0,num_1,num_2开始迭代计算即可。

代码：

```
public int numTrees(int n) {
 int arr[]=new int[n+1];
 int len=0;
 Arrays.fill(arr, 0);
```

By mnmlist,2015-9-2 9:35:19

```
 arr[0]=1;
 arr[1]=1;
 for(int i=2;i<=n;i++)
 {
  len=i-1;
  for(int j=0;j<i;j++)
  {
   arr[i]+=arr[j]*arr[len-j];
  }
 }
 return arr[n];
   }
```

结果：

leetcode_173_Binary Search Tree Iterator

描述：

Implement an iterator over a binary search tree (BST). Your iterator will be initialized with the root node of a BST.

Calling `next()` will return the next smallest number in the BST.

Note: `next()` and `hasNext()` should run in average O(1) time and uses O(h) memory, where h is the height of the tree.

Credits:
Special thanks to @ts for adding this problem and creating all test cases.

思路：

这道题想了好久，知道用中序遍历来解决，用一个list将遍历的元素存储起来一下就解决了，但是空间复杂度不行。具体怎么解决，如何控制程序的终止困扰了我好久。知道我想起来至多用O(h) memory，我想到了直接把一趟遍历后返回开始之前的所有元素存储起来不就正好符合题目要求了么，bravo！

代码：

```
/**
* Definition for binary tree
```

By mnmlist,2015-9-2 9:35:19

```
* public class TreeNode {
*     int val;
*     TreeNode left;
*     TreeNode right;
*     TreeNode(int x) { val = x; }
* }
*/

public class BSTIterator {

List<Integer> list = new ArrayList<Integer>();//
Stack<TreeNode> st = new Stack<TreeNode>();//

public BSTIterator(TreeNode root) {
 if (root != null)
  st.push(root);
}

/** @return whether we have a next smallest number */
public boolean hasNext() {
 boolean flag1 = list.isEmpty(), flag2 = st.isEmpty();
 if (!flag1)
  return true;
 if (flag1 && !flag2)
  return iteratorDemo();
 return false;
}

/** @return the next smallest number */
public int next() {
 int num = list.get(0);
 list.remove(0);
 return num;
}
//
public boolean iteratorDemo() {
 TreeNode top = null;
 while (!st.empty()) {
  top = st.peek();
  while (top.left != null) {
   st.push(top.left);
   top = top.left;
  }
  while (top.right == null) {
   list.add(top.val);
   st.pop();
   if (!st.empty())
    top = st.peek();
   else
    break;
  }
  if (!st.empty()) {
   list.add(top.val);
   st.pop();
   st.push(top.right);
   break;
```

By mnmlist,2015-9-2 9:35:19

```
  }
 }
 if(list.isEmpty())
  return false;
 return true;
}
}


/**
* Your BSTIterator will be called like this:
* BSTIterator i = new BSTIterator(root);
* while (i.hasNext()) v[f()] = i.next();
*/
```

结果：

leetcode_106_Construct Binary Tree from Inorder and Postorder Traversal

描述：

Given inorder and postorder traversal of a tree, construct the binary tree.

Note:
You may assume that duplicates do not exist in the tree.

思路：

1.将中序遍历序列和其对应的下标存储到一个map中，方便下面的查找
2.递归选取后序序列的倒数第一个元素作为树的根节点，然后查找根节点在后序序列中位置inorderIndex，endInorder-inorderIndex可以得到右子树的长度
3.根据右子树的长度和endPreOrder可以求出后序序列中右子树的起始位置
4.从上面可以求出左右子树的后序序列和中序序列的起始位置，递归调用建树过程即可。

代码：

```
/**
* Definition for binary tree
* public class TreeNode {
```

By mnmlist,2015-9-2 9:35:19

```
*     int val;
*     TreeNode left;
*     TreeNode right;
*     TreeNode(int x) { val = x; }
* }
*/
public class Solution {
   int[] ArrPostorder;
int[] ArrInorder;
Map<Integer, Integer> mapInorder = new HashMap<Integer, Integer>();
public TreeNode buildTree(int[] inorder, int[] postorder) {
 if (inorder.length == 0 || inorder == null)
  return null;
 ArrPostorder = postorder;
 ArrInorder = inorder;
 for (int i = 0; i < inorder.length; i++)
  mapInorder.put(inorder[i], i);
 int start = 0, end = postorder.length - 1;
 TreeNode root = new TreeNode(0);
 createTree(root, start, end, start, end);
 return root;
}

public void createTree(TreeNode root, int start1, int end1, int start2,int end2) {
 int subStart1,subStart2,subEnd1,subEnd2;
 int target = ArrPostorder[end2];
 int indexInOrder = mapInorder.get(target);
 int len = end1-indexInOrder;
 int indexPostOrder = end2-len;
 if (start1 <= indexInOrder - 1) {
  subEnd1 = indexInOrder-1;
  subEnd2 = indexPostOrder - 1;
  root.left = new TreeNode(0);
  createTree(root.left, start1, subEnd1, start2, subEnd2);
 }
 root.val = target;
 if (indexInOrder + 1 <= end1) {
  subStart1 = indexInOrder + 1;
  subStart2 = indexPostOrder;
  subEnd2=end2-1;
  root.right = new TreeNode(0);
  createTree(root.right, subStart1, end1, subStart2, subEnd2);
 }
 return;
}
}
```

结果：

By mnmlist,2015-9-2 9:35:19

leetcode_105_Construct Binary Tree from Preorder and Inorder Traversal

描述：

Given preorder and inorder traversal of a tree, construct the binary tree.

Note:
You may assume that duplicates do not exist in the tree.

思路：

1.将中序遍历序列和其对应的下标存储到一个map中，方便下面的查找
2.递归选取前序序列的第一个元素作为树的根节点，然后查找根节点在前序序列中位置inorderIndex，inorderIndex-startInorder可以得到左子树的长度
3.根据左子树的长度和startPreOrder可以求出前序序列中左子树的起始位置
4.从上面可以求出左右子树的前序序列和中序序列的起始位置，递归调用建树过程即可。
PS：其实，对于这道题，有更简单的方法，可根据按前序序列元素出现的顺序依次作为树的根节点进行前序建树，这样要简单多了，根本不用分别计算前序和中序序列的左右子树的起始位置，哪天把这种方法的代码给补上。我的方法好繁呐，但是，我感觉更直接，

而且，前序中序后序建树均可。

代码：

```
/**
* Definition for binary tree
* public class TreeNode {
*     int val;
*     TreeNode left;
*     TreeNode right;
*     TreeNode(int x) { val = x; }
* }
*/
public class Solution {
   int []ArrPreorder;
int []ArrInorder;
   Map<Integer, Integer>mapInorder=new HashMap<Integer, Integer>();
public TreeNode buildTree(int[] preorder, int[] inorder) {
 if(preorder.length==0||preorder==null)
  return null;
 ArrPreorder=preorder;
```

By mnmlist,2015-9-2 9:35:19

```
    ArrInorder=inorder;
    for(int i=0;i<inorder.length;i++)
     mapInorder.put(inorder[i], i);
       int start=0, end=preorder.length-1;
       TreeNode root =new TreeNode(0);
       createTree(root,start,end,start, end);
       return root;
   }
public void createTree(TreeNode root, int start1,int end1,int start2,int end2)
{
 int subStart1=start1,subStart2=start2,subEnd1=end1,subEnd2=end2;
 int target=ArrPreorder[start1];
 int indexInOrder=mapInorder.get(target);
 int len=indexInOrder-start2;
 int indexPreOrder=start1+len;
 if(start2<=indexInOrder-1)
 {
  subStart1=start1+1;
  subEnd1=indexPreOrder;
  subEnd2=indexInOrder-1;
  root.left=new TreeNode(0);
  createTree(root.left,subStart1,start2, subStart2, subEnd2);
 }
 root.val=target;
 if(indexInOrder+1<=end2)
 {
  subStart1=indexPreOrder+1;
  subStart2=indexInOrder+1;
  root.right=new TreeNode(0);
  createTree(root.right,subStart1,end1, subStart2, end2);
 }
}
}
```

结果：

leetcode_99_Recover Binary Search Tree

描述：

Two elements of a binary search tree (BST) are swapped by mistake.

By mnmlist,2015-9-2 9:35:19

Recover the tree without changing its structure.

Note:
A solution using O(n) space is pretty straight forward. Could you devise a constant space solution?

confused what `"{1,#,2,3}"` means? > read more on how binary tree is serialized on OJ.

OJ's Binary Tree Serialization:
The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.

Here's an example:

```
  1
 /\
2  3
  /
 4
  \
   5
```

The above binary tree is serialized as `"{1,2,3,#,#,4,#,#,5}"`.

思路：

首先中序遍历二叉查找树并将遍历的节点存储到一个list中，然后对list中的值进行比较，查找出位置出现变化的两个结点，将两个结点的值进行互换，完成本题的要求。

但是呢，对于如何发现位置出现变化的两个结点是本题的重点和难点，具体判断条件可以参见下面的程序，最后对查到的结点进行取舍也是一大问题，一般符合判断条件的结点会出现三个，我取的是第一个和第三个，这不好讲清楚，具体可自行推敲。

代码：

```
/**
* Definition for binary tree
* public class TreeNode {
*     int val;
*     TreeNode left;
*     TreeNode right;
*     TreeNode(int x) { val = x; }
* }
*/
public class Solution {
   public void recoverTree(TreeNode root) {
```

By mnmlist,2015-9-2 9:35:19

```
 List<TreeNode>list=new ArrayList<TreeNode>();
     if(root==null)
   return;
     Stack<TreeNode>st=new Stack<TreeNode>();
     st.push(root);
     TreeNode top=null;
     while(!st.empty())
     {
      top=st.peek();
      while(top.left!=null)
      {
       st.push(top.left);
       top=top.left;
      }
      while(top.right==null)
      {
       list.add(top);
       st.pop();
       if(!st.empty())
        top=st.peek();
       else
        break;
      }
      if(!st.empty())
      {
       list.add(top);
       st.pop();
       st.push(top.right);
      }

     }
     int len=list.size()-1;
     List<Integer>indexList=new ArrayList<Integer>();
     if(list.get(0).val>list.get(1).val)
      indexList.add(0);
     for(int i=1;i<len;i++)
     {
      if(list.get(i).val<list.get(i-1).val||list.get(i).val>list.get(i+1).val)
       indexList.add(i);
     }
     if(list.get(len).val<list.get(len-1).val)
      indexList.add(len);
     TreeNode node1=list.get(indexList.get(0));
     TreeNode node2=list.get(indexList.get(indexList.size()-1));
     int temp=node1.val;
     node1.val=node2.val;
     node2.val=temp;

   }
}
```

结果：

By mnmlist,2015-9-2 9:35:19

leetcode_117_Populating Next Right Pointers in Each Node II

描述：

Follow up for problem "Populating Next Right Pointers in Each Node".
What if the given tree could be any binary tree? Would your previous solution still work?
Note:

- You may only use constant extra space.

For example,
Given the following binary tree,

```
    1
  / \
 2   3
/\   \
4 5   7
```

After calling your function, the tree should look like:

```
    1 -> NULL
  / \
 2 -> 3 -> NULL
/\   \
4-> 5 -> 7 -> NULL
```

思路：

就是用两个list存储相邻两层的结点，把第i+1层结点存储到list中的同时，分别将第i层的各个节点指向下一个，跟116题Populating Next Right Pointers in Each Node I一个思路，就不重复写了。

By mnmlist,2015-9-2 9:35:19

代码：

```
/**
* Definition for binary tree with next pointer.
* public class TreeLinkNode {
*     int val;
*     TreeLinkNode left, right, next;
*     TreeLinkNode(int x) { val = x; }
* }
*/
public class Solution {
    public void connect(TreeLinkNode root)
 {
      if(root==null)
       return ;
      List<TreeLinkNode>list=new ArrayList<TreeLinkNode>();
      List<TreeLinkNode>temp=new ArrayList<TreeLinkNode>();
      TreeLinkNode node=null,tempNode=null;
      list.add(root);
      while(true)
      {
       tempNode=list.get(0);
       if(tempNode.left!=null)
        temp.add(tempNode.left);
       if(tempNode.right!=null)
        temp.add(tempNode.right);
       for(int i=1;i<list.size();i++)
       {
        node=list.get(i);
        if(node.left!=null)
         temp.add(node.left);
        if(node.right!=null)
         temp.add(node.right);
        tempNode.next=node;
        tempNode=node;
       }
       if(temp.size()!=0)
       {
       list=temp;
       temp=new ArrayList<TreeLinkNode>();
       }
       else
        break;
      }
      return;
 }
}
```

By mnmlist,2015-9-2 9:35:19

结果：

使用Java中Comparator接口实现自定义排序

一般情况下，自己动手写一个简单排序程序还是没有问题的，但是你能保证写出来的排序程序的时间复杂度吗？你能保证程序的正确性吗，鲁棒性呢，还有程序结构的清晰性，可维护性.....综上所述，学习一下排序接口来实现对复杂对象的排序还是很有必要的。Java中有两个用来实现排序的接口Comparator和Comparable接口，本人比较喜欢使用java的Comparator接口，在程序里实现Comparator接口里的compare(Object o1,Object o2)方法，然后在程序中通过调用Arrays.sort(array,new DemoClass())或者Collections.sort(collection,new DemoClass())来实现对对象数组的排序，直接上一栗子：

描述：

Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

思路：

本来想模仿合并两个链表的方法，查找所有的lists链表一遍找出一个最小值，并将最小值链接到结果链表中，并将最小结点删除，将最小结点的next结点替代之，直至为null，将链表删去。
但是当lists.size()比较大时，时间复杂度就上来了，而且每次只能选一个最小结点，O(N*N)的节奏。。。
所以，想到不就是排序么，将所有的结点集合到一起，然后对其进行快速排序不就可以了么，啊哈，真的是这样，bravo！
很显然，这种方法肯定不是最优方法，但是时间复杂度也只是O(N*logN)，也是可以接受的。

代码：

By mnmlist,2015-9-2 9:35:19

```
public class Solution implements Comparator<ListNode> {
public int compare(ListNode t1, ListNode t2) {
 return t1.val - t2.val;
}

public ListNode mergeKLists(List<ListNode> lists) {
 if (lists == null || lists.size() == 0)
  return null;
 if (lists.size() == 1)// just have one linkedlist
  return lists.get(0);
 int i=0,len=0;
 List<ListNode> returnList = new ArrayList<ListNode>();
 ListNode cur = null;
 for (i = 0; i < lists.size(); i++) {
  cur = lists.get(i);
  while (cur != null) {
   returnList.add(cur);
   cur = cur.next;
  }
 }
 Collections.sort(returnList, new Solution());
 ListNode pre = returnList.get(0);
 cur = null;
 len = returnList.size();
 for (i = 1; i < len; i++) {
  cur = returnList.get(i);
  pre.next = cur;
  pre = cur;
 }
 returnList.get(len - 1).next = null;
 return returnList.get(0);
}
}
```


leetcode_98_Validate Binary Search Tree

描述：

Given a binary tree, determine if it is a valid binary search tree (BST).

By mnmlist,2015-9-2 9:35:19

Assume a BST is defined as follows:

- The left subtree of a node contains only nodes with keys less than the node's key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- Both the left and right subtrees must also be binary search trees.

confused what `"{1,#,2,3}"` means? > read more on how binary tree is serialized on OJ.

OJ's Binary Tree Serialization:
The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.

Here's an example:

```
 1
/\
2  3
 /
 4
  \
   5
```

The above binary tree is serialized as `"{1,2,3,#,#,4,#,#,5}"`.

思路：

由于二叉排序树和对二叉树的中序遍历所形成的值是有序的是充分必要条件，所以仅需对二叉树进行中序遍历即可，并将遍历的结点的值存储到一个list中，然后依次比较list中的值，是有序的则二叉树为二叉排序树，否则则不是。

当然，一个更好的方法是用一个temp暂存上一个结点的值，然后依次进行比较即可。

代码：

```
public boolean isValidBST(TreeNode root) {
 boolean flag=true;
 List<Integer>list=new ArrayList<Integer>();
     if(root==null)
   return flag;
     Stack<TreeNode>st=new Stack<TreeNode>();
     st.push(root);
     TreeNode top=null;
     while(!st.empty())
     {
```

By mnmlist,2015-9-2 9:35:19

```
      top=st.peek();
      while(top.left!=null)
      {
       st.push(top.left);
       top=top.left;
      }
      while(top.right==null)
      {
       list.add(top.val);
       st.pop();
       if(!st.empty())
        top=st.peek();
       else
        break;
      }
      if(!st.empty())
      {
       list.add(top.val);
       st.pop();
       st.push(top.right);
      }

     }
     int len=list.size();
     int num=list.get(0),temp=0;
     for(int i=1;i<len;i++)
     {
      temp=list.get(i);
      if(num>temp)
      {
       flag=false;
       break;
      }
      num=temp;
     }
 return flag;
   }
```

结果：

leetcode_129_Sum Root to Leaf Numbers

By mnmlist,2015-9-2 9:35:19

描述：

Given a binary tree containing digits from `0-9` only, each root-to-leaf path could represent a number.

An example is the root-to-leaf path `1->2->3` which represents the number `123`.

Find the total sum of all root-to-leaf numbers.

For example,

```
 1
/\
2  3
```

The root-to-leaf path `1->2` represents the number `12`.
The root-to-leaf path `1->3` represents the number `13`.

Return the sum = 12 + 13 = `25`.

思路：

和pathsum系列是一个思路，后序遍历至某个节点，将root-to-leaf的所有结点的值表示成一个数和sum相加，直至遍历完所有的leaf结点，返回sum。

代码：

```
public int sumNumbers(TreeNode root) {
 if (root == null)
  return 0;
 int temp=0,sum=0;
 LinkedList<TreeNode>linkedList=new LinkedList<TreeNode>();
 Stack<TreeNode> st1 = new Stack<TreeNode>();
 linkedList.addLast(root);
 TreeNode top = null;
 while (!linkedList.isEmpty()) {
  top=linkedList.getLast();
  while(top.left!=null)//
  {
   linkedList.addLast(top.left);
   top=top.left;
  }
  while(!linkedList.isEmpty())
  {
   top=linkedList.getLast();
   if(top.right==null)//
```

By mnmlist,2015-9-2 9:35:19

```
   {
    if(top.left==null)
    {
     temp=0;
     for(int i=0;i<linkedList.size();i++)
      temp=temp*10+linkedList.get(i).val;
     sum+=temp;
    }
    linkedList.removeLast();
   }else
   {
    if(st1.empty()||(!st1.empty()&&st1.peek()!=top))//st1
    {
     st1.push(top);
     linkedList.addLast(top.right);
     break;//
    }
    else//
    {
     st1.pop();
     linkedList.removeLast();
    }
   }

  }
 }
 return sum;
   }
```

结果：

leetcode_108_Convert Sorted Array to Binary Search Tree

描述：

Given an array where elements are sorted in ascending order, convert it to a height balanced BST.

思路：

By mnmlist,2015-9-2 9:35:19

思路就是按二分查找的思路进行中序建树，递归建树。

代码：

```
public TreeNode sortedArrayToBST(int[] num) {
 if(num.length==0||num==null)
  return null;
        TreeNode rootNode =new TreeNode(0);
        int start=0,end=num.length-1;
        createTree(rootNode, num, start, end);
        return rootNode;
   }
public void createTree(TreeNode root,int []num, int start,int end)
{
 if(start>end)
 {
  root=null;
  return;
 }
 int mid=(start+end)/2;
 if(start<=mid-1)
 {
  root.left=new TreeNode(0);
  createTree(root.left, num, start, mid-1);
 }
 root.val=num[mid];
 if(mid+1<=end)
 {
  root.right=new TreeNode(0);
  createTree(root.right, num, mid+1, end);
 }

}
```

结果：

leetcode_113_Path Sum II

描述：

Given a binary tree and a sum, find all root-to-leaf paths where each path's sum equals the

By mnmlist,2015-9-2 9:35:19

given sum.

For example:
Given the below binary tree and `sum = 22`,

```
      5
     /\
    4  8
   /  /\
  11 13 4
 / \  /\
7  2 5  1
```

return

```
[
 [5,4,11,2],
 [5,8,4,5]
]
```

思路：

大致思路就是将pathsum 1中的存储节点的栈stack换成具有栈功能的LinkedList，这样每匹配依次将各个节点的值存储到一个list中，遍历完整棵树后返回一个满足条件的结点列表的列表。

代码：

```
public List<List<Integer>> pathSum(TreeNode root, int sum){
 List<List<Integer>>list=new ArrayList<List<Integer>>();
 int num=0;
 if (root == null)
  return list;
 List<Integer>subList=new ArrayList<Integer>();
 LinkedList<TreeNode>linkedList=new LinkedList<TreeNode>();
 Stack<TreeNode> st1 = new Stack<TreeNode>();
 linkedList.addLast(root);
 num+=root.val;
 TreeNode top = null;
 while (!linkedList.isEmpty()) {
  top=linkedList.getLast();
  while(top.left!=null)//
  {
   linkedList.addLast(top.left);
   num+=top.left.val;
   top=top.left;
  }
  while(!linkedList.isEmpty())
  {
   top=linkedList.getLast();
```

By mnmlist,2015-9-2 9:35:19

```
   if(top.right==null)//
   {
    if(top.left==null&&num==sum)
    {
     for(int i=0;i<linkedList.size();i++)
      subList.add(linkedList.get(i).val);
     list.add(subList);
     subList=new ArrayList<Integer>();
    }
    linkedList.removeLast();
    num-=top.val;
   }else
   {
    if(st1.empty()||(!st1.empty()&&st1.peek()!=top))//st1
    {
     st1.push(top);
     linkedList.addLast(top.right);
     num+=top.right.val;
     break;//
    }
    else//
    {
     st1.pop();
     linkedList.removeLast();
     num-=top.val;
    }
   }

  }
 }
 return list;
   }
```

结果：

leetcode_145_Binary Tree Postorder Traversal

描述：

By mnmlist,2015-9-2 9:35:19

Given a binary tree, return the postorder traversal of its nodes' values.

For example:
Given binary tree {1,#,2,3},

```
 1
  \
   2
  /
 3
```

return [3,2,1].

Note: Recursive solution is trivial, could you do it iteratively?

思路：

刚开始的时候由于刚做出前序和中序的遍历，落入这两种遍历的窠臼，总以为在中序遍历的基础上改改就可以了，但是，改了好久，费了了好大的劲，走了好多弯路，in vain！最后联想到后序遍历不就是在中序遍历的基础上，经由子树根节点访问右子树，然后再访问根节点，不是么？所以，想到再多用一个栈来标记是否是第一次访问子树根节点，第二次访问子树根节点时（当然不算第一次将所有左子树入栈的那一次）就存储根节点的值。bravo！It's done！

代码：

```
class TreeNode {
int val;
TreeNode left;
TreeNode right;

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

public class _145_PostOrderTraverse {
public List<Integer> postorderTraversal(TreeNode root) {
 List<Integer> list = new ArrayList<Integer>();
 if (root == null)
  return list;
 Stack<TreeNode> st = new Stack<TreeNode>();
 Stack<TreeNode> st1 = new Stack<TreeNode>();
 st.push(root);
 TreeNode top = null;
 while (!st.empty()) {
  top=st.peek();
  while(top.left!=null)//
  {
   st.push(top.left);
   top=top.left;
  }
```

By mnmlist,2015-9-2 9:35:19

```
  while(!st.empty())
  {
   top=st.peek();
   if(top.right==null)//
   {
    list.add(top.val);
    st.pop();
   }else
   {
    if(st1.empty()||(!st1.empty()&&st1.peek()!=top))//st1
    {
     st1.push(top);
     st.push(top.right);
     break;//
    }
    else//
    {
     list.add(top.val);
     st1.pop();
     st.pop();
    }
   }

  }
 }
 return list;
}
}
```

结果：

leetcode_114_Flatten Binary Tree to Linked List

描述：

Given a binary tree, flatten it to a linked list in-place.

By mnmlist,2015-9-2 9:35:19

For example,
Given

```
    1
   /\
  2  5
 /\  \
3  4  6
```

The flattened tree should look like:

```
1
 \
  2
   \
    3
     \
      4
       \
        5
         \
          6
```

click to show hints.

Hints:
If you notice carefully in the flattened tree, each node's right child points to the next node of a pre-order traversal.

思路：

按照中序遍历的方式将结点存储到一个list列表中，然后分别将各个节点的left赋值为空，将right指向下一个节点，最后将最后一个结点的right赋值为空即可生成题目要求的树结构。结果对了是对了，好像不太符合题目要求，题目要求 do it in-place，不知道我的算不算 in place。很显然，从时间上来看，我想到的思路也不能算好。刚开始以为是要生成一个链表，走了好多弯路，发现返回值就是一个void，题目要实现的就是改变下树是结构而已。

代码：

```
class TreeNode{
 int val;
 TreeNode left;
 TreeNode right;
```

By mnmlist,2015-9-2 9:35:19

```
 TreeNode(int x){val=x;}
}
   public void flatten(TreeNode root)
   {
    List<TreeNode>list=new ArrayList<TreeNode>();
     if(root==null)
   return ;
     Stack<TreeNode>st=new Stack<TreeNode>();
     st.push(root);
     TreeNode top=null;
     list.add(root);
     while(!st.empty())
     {
      top=st.peek();
      while(top.left!=null)
      {
       st.push(top.left);
       list.add(top.left);
       top=top.left;
      }
      while(top.right==null)
      {

       st.pop();
       if(!st.empty())
        top=st.peek();
       else
        break;
      }
      if(!st.empty())
      {
       st.pop();
       st.push(top.right);
       list.add(top.right);
      }

     }
     int len=list.size();
     TreeNode pre =list.get(0),cur=null;
     for(int i=1;i<len;i++)
     {
      cur=list.get(i);
      pre.right=cur;
      pre.left=null;
      pre=cur;
     }
     cur=list.get(len-1);
     cur.left=null;
     cur.right=null;

   }
```

By mnmlist,2015-9-2 9:35:19

结果：

leetcode_111_Minimum Depth of Binary Tree

描述：

Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

思路：

和求最大深度思路是类似的，只是题目要求的最小深度是到最近的叶子节点的最小深度，所以当一个节点的深度为0时，很显然是没有叶子节点的，应该取另外一个num并加1

代码：

```
public int minDepth(TreeNode root) {
      if(root==null)
       return 0;
      if(root.left==null&&root.right==null)
       return 1;
      int num1=minDepth(root.left),num2=minDepth(root.right);
      int num=0;
      if(num1==0)
       return num=num2+1;
      if(num2==0)
       return num =num1+1;
      num=(num1<=num2?num1:num2)+1;
      return num;
   }
```

By mnmlist,2015-9-2 9:35:19

结果：

leetcode_104_Maximum Depth of Binary Tree

描述：

Given a binary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

思路：

用递归，很简单，树为空时返回0，左子树为空且右子树为空时返回1，否则就返回左子树或右子树的最大深度再加1

代码：

```
class TreeNode{
 int val;
 TreeNode left;
 TreeNode right;
 TreeNode(int x){val=x;}
}
public int maxDepth(TreeNode root) {

       if(root==null)
        return 0;
```

By mnmlist,2015-9-2 9:35:19

```
        if(root.left==null&&root.right==null)
         return 1;
        int num1=maxDepth(root.left),num2=maxDepth(root.right);
        int num=(num1>=num2?num1:num2)+1;
        return num;
    }
```

结果：

leetcode_101_Symmetric Tree

描述：

Given a binary tree, check whether it is a mirror of itself (ie, symmetric around its center).
For example, this binary tree is symmetric:

```
  1
 /\
2  2
/\/\
3 4 4 3
```

But the following is not:

```
  1
 /\
2  2
 \  \
 3   3
```

Note:
Bonus points if you could solve it both recursively and iteratively.

confused what `"{1,#,2,3}"` means? > read more on how binary tree is serialized on OJ.

OJ's Binary Tree Serialization:
The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.

By mnmlist,2015-9-2 9:35:19

Here's an example:

```
  1
 /\
2  3
  /
 4
  \
   5
```

The above binary tree is serialized as `"{1,2,3,#,#,4,#,#,5}"`.

代码：

```
class TreeNode{
 int val;
 TreeNode left;
 TreeNode right;
 TreeNode(int x){val=x;}
}
public boolean isSymmetric(TreeNode root) {
 if(root==null)
  return true;
       if(treeIsSymmetric(root.left, root.right))
        return true;
 return false;
   }
public boolean treeIsSymmetric(TreeNode p,TreeNode q)
{
 if(p==null&&q==null)//
  return true;
 if(p==null||q==null)//
  return false;
 //
 if(p.val==q.val&&treeIsSymmetric(p.left, q.right)&&treeIsSymmetric(p.right, q.left))
  return true;
 return false;
}
```

结果：

By mnmlist,2015-9-2 9:35:19

leetcode_100_Same Tree

描述：

Given two binary trees, write a function to check if they are equal or not.

Two binary trees are considered equal if they are structurally identical and the nodes have the same value.

思路：

由于这题标识为easy，判断两棵树是否相等，想到了用递归。有这几种可能，两棵树相等的充要条件为root的值相等&&左子树相等&&右子树相等，两棵树都为空时也相等，其中一棵为空时就不相等

代码：

```
class TreeNode{
 int val;
 TreeNode left;
 TreeNode right;
 TreeNode(int x){val=x;}
}
public boolean isSameTree(TreeNode p, TreeNode q) {
 if(p==null&&q==null)//
  return true;
 if(p==null||q==null)//
  return false;
 //root&&&&
       if(p.val==q.val&&isSameTree(p.left, q.left)&&isSameTree(p.right, q.right))
  return true;
       return false;
   }
```

结果：

By mnmlist,2015-9-2 9:35:19

leetcode_23_Merge k Sorted Lists

描述：

Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

思路：

本来想模仿合并两个链表的方法，查找所有的lists链表一遍找出一个最小值，并将最小值链接到结果链表中，并将最小结点删除，将最小结点的next结点替代之，直至为null，将链表删去。
但是当lists.size()比较大时，时间复杂度就上来了，而且每次只能选一个最小结点，O(N*N)的节奏。。。
所以，想到不就是排序么，将所有的结点集合到一起，然后对其进行快速排序不就可以了么，啊哈，真的是这样，bravo！
很显然，这种方法肯定不是最优方法，但是时间复杂度也只是O(N*logN)，也是可以接受的。

代码：

```
public class Solution implements Comparator<ListNode>{
public int compare(ListNode t1,ListNode t2)
{
 return t1.val-t2.val;
}
public ListNode mergeKLists(List<ListNode> lists)
{
 if(lists==null||lists.size()==0)
   return null;
  int len=0;
  int i=0;
  while(i<lists.size())//delete the duplicated linkedlist
  {
   if(lists.get(i)==null)
   {
    lists.remove(i);
    i=0;
    continue;
   }
   i++;
  }
  if(lists.size()==0)//has no node
   return null;
  else if(lists.size()==1)//just have one linkedlist
```

By mnmlist,2015-9-2 9:35:19

```
      return lists.get(0);
  List<ListNode>returnList=new ArrayList<ListNode>();
  ListNode cur=null;
  for(i=0;i<lists.size();i++)
  {
   cur=lists.get(i);
   while(cur!=null)
   {
    returnList.add(cur);
    cur=cur.next;
   }
  }
  Collections.sort(returnList, new Solution());
  ListNode pre=returnList.get(0);
  cur=null;
  len=returnList.size();
  for(i=1;i<len;i++)
  {
   cur=returnList.get(i);
   pre.next=cur;
   pre=cur;
  }
  returnList.get(len-1).next=null;
  return returnList.get(0);
}
}
```

结果：

leetcode_103_Binary Tree Zigzag Level Order Traversal

描述：

Given a binary tree, return the zigzag level order traversal of its nodes' values. (ie, from left to right, then right to left for the next level and alternate between).

For example:
Given binary tree {3,9,20,#,#,15,7},

```
  3
 /\
```

By mnmlist,2015-9-2 9:35:19

```
 9 20
  / \
 15  7
```

return its zigzag level order traversal as:

```
[
 [3],
 [20,9],
 [15,7]
]
```

confused what `"{1,#,2,3}"` means? > read more on how binary tree is serialized on OJ.

OJ's Binary Tree Serialization:
The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.

Here's an example:

```
  1
 / \
2  3
  /
 4
  \
   5
```

The above binary tree is serialized as `"{1,2,3,#,#,4,#,#,5}"`.

思路：

具体的思路和层序遍历一致，只是需要将偶数层的结点值颠倒一下即可，时间略长，可见本方法并非好的方法。思路详见层序遍历：
http://blog.csdn.net/mnmlist/article/details/44490975

代码：

```
public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
 List<List<Integer>> list = new ArrayList<List<Integer>>();//
 if (root == null)
  return list;
 Queue<TreeNode> q1 = new LinkedList<TreeNode>();//
 Queue<TreeNode> q2 = new LinkedList<TreeNode>();
 Queue<TreeNode> temp = null;
 List<Integer> subList = null;//
 q1.add(root);
 TreeNode top = null;
 int flag=0;
```

By mnmlist,2015-9-2 9:35:19

```
 while (!q1.isEmpty()) {
  subList = new ArrayList<Integer>();
  while (!q1.isEmpty())//
  {
   top = q1.peek();
   q1.poll();
   if (top.left != null)
    q2.add(top.left);
   if (top.right != null)
    q2.add(top.right);
   subList.add(top.val);
  }
  flag++;
  if((flag&1)==0)
   Collections.reverse(subList);
  list.add(subList);
  temp = q2;// q1
  q2 = q1;
  q1 = temp;
 }
 return list;
}
```

结果：

leetcode_144_Binary Tree Preorder Traversal

描述：

Given a binary tree, return the preorder traversal of its nodes' values.

For example:
Given binary tree {1,#,2,3},

```
 1
 \
  2
 /
 3
```

return [1,2,3].

By mnmlist,2015-9-2 9:35:19

Note: Recursive solution is trivial, could you do it iteratively?

思路：

具体思路和中序遍历是一致的，只是访问结点的值的时机不同罢了，具体思路参见：
http://blog.csdn.net/mnmlist/article/details/44312315

代码：

```
/**
* Definition for binary tree
* public class TreeNode {
*     int val;
*     TreeNode left;
*     TreeNode right;
*     TreeNode(int x) { val = x; }
* }
*/
public class Solution {
   public List<Integer> preorderTraversal(TreeNode root) {
     List<Integer>list=new ArrayList<Integer>();
     if(root==null)
   return list;
     Stack<TreeNode>st=new Stack<TreeNode>();
     st.push(root);
     TreeNode top=null;
     list.add(root.val);
     while(!st.empty())
     {
      top=st.peek();
      while(top.left!=null)
      {
       st.push(top.left);
       list.add(top.left.val);
       top=top.left;
      }
      while(top.right==null)
      {

       st.pop();
       if(!st.empty())
        top=st.peek();
       else
        break;
      }
      if(!st.empty())
      {
       st.pop();
       st.push(top.right);
       list.add(top.right.val);
      }
```

```
By mnmlist,2015-9-2 9:35:19
```

```
        }
        return list;
    }
}
```

结果：

leetcode_107_Binary Tree Level Order Traversal II

描述：

Given a binary tree, return the bottom-up level order traversal of its nodes' values. (ie, from left to right, level by level from leaf to root).

For example:
Given binary tree {3,9,20,#,#,15,7},

```
  3
 / \
9  20
  / \
 15  7
```

return its bottom-up level order traversal as:

```
[
 [15,7],
 [9,20],
 [3]
]
```

confused what "{1,#,2,3}" means? > read more on how binary tree is serialized on OJ.

OJ's Binary Tree Serialization:
The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.

Here's an example:

By mnmlist,2015-9-2 9:35:19

```
  1
 /\
2  3
  /
 4
  \
   5
```

The above binary tree is serialized as `"{1,2,3,#,#,4,#,#,5}"`.

思路：

一般的层序遍历直接打印出结果，用队列即可，但是此次的要求尼是按层次打印结果，所以考虑到用两个队列来交替存储，遍历上一层次的同时将下一层的结点存储到另一个队列中，并在将上面一层的遍历完成后交换两个队列的值。最后，将结果列表调换下顺序即可。

代码：

```
/**
* Definition for binary tree
* public class TreeNode {
*     int val;
*     TreeNode left;
*     TreeNode right;
*     TreeNode(int x) { val = x; }
* }
*/
public class Solution {
   public List<List<Integer>> levelOrderBottom(TreeNode root) {
      List<List<Integer>>list=new ArrayList<List<Integer>>();//
      if(root==null)
       return list;
      Queue<TreeNode>q1=new LinkedList<TreeNode>();//
      Queue<TreeNode>q2=new LinkedList<TreeNode>();
      Queue<TreeNode>temp=null;
      List<Integer>subList=null;//
      q1.add(root);
      TreeNode top=null;
      while(!q1.isEmpty())
      {
       subList=new ArrayList<Integer>();
       while(!q1.isEmpty())//
       {
        top=q1.peek();
        q1.poll();
        if(top.left!=null)
         q2.add(top.left);
        if(top.right!=null)
         q2.add(top.right);
        subList.add(top.val);
       }
       list.add(subList);
```

By mnmlist,2015-9-2 9:35:19

```
        temp=q2;//q1
        q2=q1;
        q1=temp;
       }
       int len=list.size();
       int num=len-1;
       len/=2;
       for(int i=0;i<len;i++)//
       {
        subList=list.get(i);
        list.set(i,list.get(num-i));
        list.set(num-i,subList);
       }
       return list;
}
}
```

结果：

leetcode_102_Binary Tree Level Order Traversal

1.描述

Given a binary tree, return the level order traversal of its nodes' values. (ie, from left to right, level by level).

For example:
Given binary tree {3,9,20,#,#,15,7},

```
  3
 / \
9  20
  / \
 15  7
```

return its level order traversal as:

```
[
 [3],
```

By mnmlist,2015-9-2 9:35:19

```
 [9,20],
 [15,7]
]
```

confused what `"{1,#,2,3}"` means? > read more on how binary tree is serialized on OJ.

OJ's Binary Tree Serialization:
The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.

Here's an example:

```
  1
 / \
2  3
  /
 4
  \
   5
```

The above binary tree is serialized as `"{1,2,3,#,#,4,#,#,5}"`.

2.思路

一般的层序遍历直接打印出结果，用队列即可，但是此次的要求尼是按层次打印结果，所以考虑到用两个队列来交替存储，遍历上一层次的同时将下一层的结点存储到另一个队列中，并在将上面一层的遍历完成后交换两个队列的值。

3.代码

```
/**
* Definition for binary tree
* public class TreeNode {
*     int val;
*     TreeNode left;
*     TreeNode right;
*     TreeNode(int x) { val = x; }
* }
*/
public class Solution {
   public List<List<Integer>> levelOrder(TreeNode root) {
      List<List<Integer>>list=new ArrayList<List<Integer>>();//
      if(root==null)
       return list;
      Queue<TreeNode>q1=new LinkedList<TreeNode>();//
      Queue<TreeNode>q2=new LinkedList<TreeNode>();
      Queue<TreeNode>temp=null;
```

By mnmlist,2015-9-2 9:35:19

```
        List<Integer>subList=null;//
        q1.add(root);
        TreeNode top=null;
        while(!q1.isEmpty())
        {
         subList=new ArrayList<Integer>();
         while(!q1.isEmpty())//
         {
          top=q1.peek();
          q1.poll();
          if(top.left!=null)
           q2.add(top.left);
          if(top.right!=null)
           q2.add(top.right);
          subList.add(top.val);
         }
         list.add(subList);
         temp=q2;//q1
         q2=q1;
         q1=temp;
        }
        return list;
}
}
```

4.结果

leetcode_94_Binary Tree Inorder Traversal

1.描述

Given a binary tree, return the inorder traversal of its nodes' values.

For example:
Given binary tree {1,#,2,3},

```
1
 \
  2
```

By mnmlist,2015-9-2 9:35:19

```
 /
3
```

return [1,3,2].

Note: Recursive solution is trivial, could you do it iteratively?

2.思路

a.先遍历左子树直至左孩子为空

b.然后看当前节点的右子树是否为空，为空，则访问当前节点，弹出当前节点，重新判断；不为空，访问当前节点，然后将当前节点弹出，并将当前节点的右孩子进栈，返回a

c.这中间需要注意的是，在弹出当前节点后，在获得栈顶的当前节点前先判断下栈是否为空

3.代码

```
public List<Integer> inorderTraversal(TreeNode root) {
     List<Integer>list=new ArrayList<Integer>();
     if(root==null)
   return list;
     Stack<TreeNode>st=new Stack<TreeNode>();
     st.push(root);
     TreeNode top=null;
     while(!st.empty())
     {
      top=st.peek();
      while(top.left!=null)
      {
       st.push(top.left);
       top=top.left;
      }
      while(top.right==null)
      {
       list.add(top.val);
       st.pop();
       if(!st.empty())
        top=st.peek();
       else
        break;
      }
      if(!st.empty())
      {
```

By mnmlist,2015-9-2 9:35:19

```
        list.add(top.val);
        st.pop();
        st.push(top.right);
       }

      }
      return list;
 }
```

4.结果

leetcode_160_Intersection of Two Linked Lists

描述：

Write a program to find the node at which the intersection of two singly linked lists begins.

For example, the following two linked lists:

```
A:          a1 → a2
                  ↘
                    c1 → c2 → c3
                  ↗
B:    b1 → b2 → b3
```

begin to intersect at node c1.

Notes:

- If the two linked lists have no intersection at all, return null.
- The linked lists must retain their original structure after the function returns.
- You may assume there are no cycles anywhere in the entire linked structure.
- Your code should preferably run in O(n) time and use only O(1) memory.

Credits:
Special thanks to @stellari for adding this problem and creating all test cases.

By mnmlist,2015-9-2 9:35:19

思路：

暴力解法，将第一个链表的所有结点放进HashSet，然后看第二个链表从头开始的第一个存在HashSet中的元素就是两个链表相交的地方。
方法不够好，题目说的空间复杂度最好为O(1)

代码：

```
public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
 if(headA==null||headB==null)
  return null;
       ListNode tempListNode=null;
       HashSet<ListNode>set=new HashSet<ListNode>();
       ListNode pListNode=headA;
       while(pListNode!=null)
       {
        set.add(pListNode);
        pListNode=pListNode.next;
       }
       pListNode=headB;
       while(pListNode!=null)
       {
        if(set.contains(pListNode))
        {
         tempListNode=pListNode;
         break;
        }else
        {
         pListNode=pListNode.next;
        }
       }
       return tempListNode;
   }
```

结果：

leetcode_143_Reorder List

By mnmlist,2015-9-2 9:35:19

描述：

Given a singly linked list L: L0→L1→…→Ln-1→Ln,
reorder it to: L0→Ln→L1→Ln-1→L2→Ln-2→…

You must do this in-place without altering the nodes' values.

For example,
Given {1,2,3,4}, reorder it to {1,4,2,3}.

思路：

大概思路就是将后面的一半结点以倒序的方式依次插到前面一半的每一个结点的后面，考虑到后面一半的结点要倒序插入所以会用到栈。
1.求出链表的长度
2.将后半部分结点依次入栈
3.将栈里的元素依次插入到前面的结点后面

代码：

```
public void reorderList(ListNode head) {
  if(head==null||head.next==null)
   return;
  ListNode pListNode=head,tempListNode=null;
  Stack<ListNode>stack=new Stack<ListNode>();
  int count=0,i=0;
  while(pListNode!=null)
  {
   count++;
   pListNode=pListNode.next;
  }
  int subLen=count/2+1;
  pListNode=head;
  for(i=1;i<subLen;i++)
   pListNode=pListNode.next;
  tempListNode=pListNode.next;
  pListNode.next=null;
  while(tempListNode!=null)
  {
   stack.push(tempListNode);
   tempListNode=tempListNode.next;
  }
  pListNode=head;
  while(!stack.empty())
  {
   tempListNode=stack.peek();
   stack.pop();
   tempListNode.next=pListNode.next;
   pListNode.next=tempListNode;
   pListNode=tempListNode.next;
```

```
By mnmlist,2015-9-2 9:35:19
```

```
  }
 }
```

结果：

leetcode_25_Reverse Nodes in k-Group

描述：

Given a linked list, reverse the nodes of a linked list k at a time and return its modified list.

If the number of nodes is not a multiple of k then left-out nodes in the end should remain as it is.

You may not alter the values in the nodes, only nodes itself may be changed.

Only constant memory is allowed.

For example,
Given this linked list: `1->2->3->4->5`

For k = 2, you should return: `2->1->4->3->5`

For k = 3, you should return: `3->2->1->4->5`

思路：

大概思路就是找出K个结点的起始位置和并将这K 个结点采用头插法的方式依次插入到这K个结点开始位置的前面一个位置之后，就可以了。
思路倒是很简单，但是指针所指的位置的捉摸是有点麻烦的，还有就是我竟然没有把创建的头节点和整个链表给链接起来。anyway,还是把这道题目给做出来了。

代码：

```
public ListNode reverseKGroup(ListNode head, int k) {
 if(head==null||head.next==null||k==1)
  return head;
 ListNode start=new ListNode(0);
 start.next=head;
 ListNode pre=start,p=head,q=head,temp=null;
 int count=1;
 int i=0;
 while(p!=null)
 {
```

By mnmlist,2015-9-2 9:35:19

```
  count=1;
  for(;count<k&&q!=null;count++)
   q=q.next;
  if(q==null)
   break;
  for(i=1;i<k;i++)
  {
   //delete the node
   temp=p.next;
   p.next=temp.next;
   //insert the node
   temp.next=pre.next;
   pre.next=temp;
  }
  pre=p;
  p=p.next;
  q=p;
 }
 head=start.next;
 return head;
   }
```

结果：

leetcode_86_Partition List

描述：

Given a linked list and a value x, partition it such that all nodes less than x come before nodes greater than or equal to x.

You should preserve the original relative order of the nodes in each of the two partitions.

For example,
Given `1->4->3->2->5->2` and x = 3,
return `1->2->2->4->3->5`.

思路：

刚开始试着把所有小于x的结点依次插到前面去，但是因为第一个和最后结点的问题真的把我搞得焦头烂额，后来想想，用我媳妇想到的方法可能更清晰一点，用两个链表分别连接小于和大于等于x 的结点，然后再把两个结点链接到一起，就可以了。在实施的时候稍

By mnmlist,2015-9-2 9:35:19

微偷点懒，首先创建两个头节点，哎，现在终于明白头节点的巨大作用了，其实，按我的那个思路，先搞个头节点，然后再用两个引用pre和cur就可以轻松搞定本题了。
做完本题感觉收获好大，头节点的出现真的让我可以很轻松地搞定许多前面我费了好大的劲才搞定的题目，尤其是涉及到在链表的开始进行插入的题目。

代码：

```
public ListNode partition(ListNode head, int x) {
 if(head==null||head.next==null)
  return head;
 ListNode little_start=new ListNode(0),little_end=little_start;
 ListNode big_start=new ListNode(0),big_end=big_start;
 ListNode pListNode=head;
 while(pListNode!=null)
 {
  if(pListNode.val<x)
  {
   little_end.next=pListNode;
   little_end=pListNode;
  }else
  {
   big_end.next=pListNode;
   big_end=pListNode;
  }
  pListNode=pListNode.next;
 }
 little_start=little_start.next;
 big_start=big_start.next;
 if(little_start!=null)
 {
  big_end.next=null;
  little_end.next=big_start;
  return little_start;
 }
 else
  return big_start;
   }
```

结果：

leetcode_92_Reverse Linked List II

By mnmlist,2015-9-2 9:35:19

描述：

Reverse a linked list from position m to n. Do it in-place and in one-pass.

For example:
Given `1->2->3->4->5->NULL`, m = 2 and n = 4,

return `1->4->3->2->5->NULL`.

Note:
Given m, n satisfy the following condition:
1 ≤ m ≤ n ≤ length of list.

思路：

这种题目，举个例子能让思路更加清晰，通过在草纸上演算可知，题目要分两种情况，m==1和m>1的情况，然后就是围绕这两种情况展开讨论，删除后面的结点，然后将后面的结点添加到前面，一次搞定，bravo！

代码：

```
public ListNode reverseBetween(ListNode head, int m, int n) {
 if (head==null) {
  return null;
 }
       ListNode p =head,q=head,temp=null;
       int i=0;
       for(i=0;i<m-1;i++)
        q=q.next;
       for(i=0;i<m-2;i++)
        p=p.next;
       if(m==1)
       {
        for(i=0;i<n-m;i++)
           {
            //delete the node in the list
            temp=q.next;
            q.next=temp.next;
            //insert the node in the list
            temp.next=p;
            p=temp;
           }
        head=p;
       }else
       {
        for(i=0;i<n-m;i++)
           {
            //delete the node in the list
            temp=q.next;
```

By mnmlist,2015-9-2 9:35:19

```
            q.next=temp.next;
            //insert the node in the list
            temp.next=p.next;
            p.next=temp;
           }
 }
 return head;
   }
```

结果：

leetcode_142_Linked List Cycle II

描述：

Given a linked list, return the node where the cycle begins. If there is no cycle, return null.

Follow up:
Can you solve it without using extra space?

思路：

从头开始遍历链表并将结点的引用存储在HashSet中，出现重复的地方就是出现环的地方。

代码：

```
public ListNode detectCycle(ListNode head) {
 if(head==null)
  return null;
       HashSet<ListNode>set=new HashSet<ListNode>();
       ListNode pListNode=head;
       while(pListNode!=null)
```

```
By mnmlist,2015-9-2 9:35:19
```

```
       {
        if(set.contains(pListNode))
         return pListNode;
        else
        {
         set.add(pListNode);
         pListNode=pListNode.next;
        }

       }
 return  null;
   }
```

结果：

leetcode_141_Linked List Cycle

描述：

Given a linked list, determine if it has a cycle in it.

Follow up:
Can you solve it without using extra space?

思路：

一想到唯一就直接联想到了hashSet，至于在不用额外存储空间的情况下把题目搞出来，这个，确实还没有想到，to be continued.....

代码：

```
public boolean hasCycle(ListNode head) {
 if(head==null)
  return false;
```

By mnmlist,2015-9-2 9:35:19

```
       HashSet<ListNode>set=new HashSet<ListNode>();
       ListNode pListNode=head;
       while(pListNode!=null)
       {
        if(set.contains(pListNode))
         return true;
        else
        {
         set.add(pListNode);
         pListNode=pListNode.next;
        }

       }
 return  false;
   }
```

结果：

leetcode_21_Merge Two Sorted Lists

描述：

Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

思路：

好像是数据结构上面的原题，就不多说了，通过比较把两个链表一起就可以了。需要注意的就是两个链表的head谁当新表的head问题，当然谁小谁当head 了，先比较一下即可。

代码：

```
public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
 if(l1==null)
```

By mnmlist,2015-9-2 9:35:19

```
 return l2;
if(l2==null)
 return l1;
ListNode listNew=null;
if(l1.val<l2.val)
{
 listNew=l1;
 l1=l1.next;
}
else
{
 listNew=l2;
 l2=l2.next;
}
ListNode pListNode=listNew;
while(l1!=null&&l2!=null)
{
 if(l1.val<l2.val)
 {
  pListNode.next=l1;
  pListNode=l1;
  l1=l1.next;
 }
 else
 {
  pListNode.next=l2;
  pListNode=l2;
  l2=l2.next;
 }
}
if(l1!=null)
 pListNode.next=l1;
if(l2!=null)
 pListNode.next=l2;
return listNew;
  }
```

结果：

leetcode_82_Remove Duplicates from

By mnmlist,2015-9-2 9:35:19

Sorted List II

描述：

Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list.

For example,
Given `1->2->3->3->4->4->5`, return `1->2->5`.
Given `1->1->1->2->3`, return `2->3`.

思路：

大致思路就是，遍历链表找出重复元素的子列并删除重复元素子列，当然，第一个元素开始有重复元素的话比较特种，需要特殊考虑。删除子列的过程稍微有点绕，题目倒是不难理解。

代码：

```
public ListNode deleteDuplicates(ListNode head) {
 if(head==null)
  return head;
 ListNode p=head,q,t=head;
 boolean flag=false;
 if(p.next!=null&&p.val==p.next.val)
 {
  flag=true;
  t=head;
 }
 while (p.next!=null)
 {
  q=p;
  if(q.val==p.next.val)
  {
   while(p.next!=null&&q.val==p.next.val)
    p=p.next;
   t.next=p.next;
   if(t.next!=null)
    p=t.next;
   else
    break;
  }
  else {
   t=p;
```

By mnmlist,2015-9-2 9:35:19

```
   p=p.next;
  }

 }
 if(flag&&head!=null)
  head=head.next;
 return head;
   }
```

结果：

leetcode_83_Remove Duplicates from Sorted List

描述：

Given a sorted linked list, delete all duplicates such that each element appear only once.

For example,
Given `1->1->2`, return `1->2`.
Given `1->1->2->3->3`, return `1->2->3`.

思路：

simple，从头到尾遍历结点，如果下一个结点不为空且当前结点和下一个节点相同，删除下一个结点，否则，遍历下一个结点。

代码：

By mnmlist,2015-9-2 9:35:19

```
public ListNode deleteDuplicates(ListNode head) {
 if(head==null)
  return head;
 ListNode p=head,q;
 while (p.next!=null)
 {
  q=p;
  if(q.val==p.next.val)
  {
   while(p.next!=null&&q.val==p.next.val)
    p=p.next;
   q.next=p.next;
   p=q.next;
   if(p==null)
    break;
  }
  else
   p=p.next;
 }
 return head;
   }
```

结果：

leetcode_19_Remove Nth Node From End of List

描述：

Given a linked list, remove the nth node from the end of list and return its head.
For example,

> Given linked list: 1->2->3->4->5, and n = 2.

By mnmlist,2015-9-2 9:35:19

> After removing the second node from the end, the linked list becomes 1->2->3->5.

Note:
Given n will always be valid.
Try to do this in one pass.

思路：

先让p指向head，p往后移动n个节点，然后让q指向head，p和q一起后移直至p指向最后一个结点，则q指向的结点即是倒数第n个结点。当然，倒数第len（链表的长度）个结点是一个特殊情况，直接head=head.next即可。

代码：

```
public ListNode removeNthFromEnd(ListNode head, int n) {
       if(head==null)
        return head;
       ListNode pListNode=head,qListNode=head;
       int len=0;
       while(pListNode!=null)
       {
        len++;
        pListNode=pListNode.next;
       }
       if(n==len)
       {
        head=head.next;
        return head;
       }
       pListNode=head;
       for(int i=0;i<n;i++)
       {
        qListNode=qListNode.next;
        if (qListNode==null)
         break;
       }

       while(qListNode!=null&&qListNode.next!=null)
       {
        pListNode=pListNode.next;
        qListNode=qListNode.next;
       }
       qListNode=pListNode.next;
       pListNode.next=qListNode.next;
       return head;
   }
```

By mnmlist,2015-9-2 9:35:19

结果：

leetcode_61_Rotate List

描述：

Given a list, rotate the list to the right by k places, where k is non-negative.

For example:
Given `1->2->3->4->5->NULL` and k = `2`,
return `4->5->1->2->3->NULL`.

思路：

本文的题意就是循环将后面的n个结点移动到前面去，所以，n有可能是大于链表的长度的，这是一个小小的陷阱。然后就是很简单的细节了，有点脑残，提交了好多次。

代码：

```
public ListNode rotateRight(ListNode head, int n) {
       if(n<=0||head==null)
        return head;
       int len=1,index=0;
       ListNode pListNode=head,qListNode=null,tailListNode=null;
       while(pListNode.next!=null)
       {
        len++;
           pListNode=pListNode.next;
       }
```

By mnmlist,2015-9-2 9:35:19

```
        tailListNode=pListNode;
        n=n%len;
        if(n==0)
         return head;
        index=len-n;
        index--;
        pListNode=head;
        for(int i=0;i<index;i++)
        {
         pListNode=pListNode.next;
        }
        qListNode=pListNode.next;
        pListNode.next=null;
        pListNode=head;
        head=qListNode;
        tailListNode.next=pListNode;
        return head;
    }
```

结果：

leetcode_24_Swap Nodes in Pairs

描述：

Given a linked list, swap every two adjacent nodes and return its head.

For example,
Given `1->2->3->4`, you should return the list as `2->1->4->3`.

Your algorithm should use only constant space. You may not modify the values in the list, only nodes itself can be changed.

思路：

By mnmlist,2015-9-2 9:35:19

简单粗暴，首先让p指向第一个节点，若果p.next不为空，q指向p.next，交换p和q的值即可，如果p.next为空，退出。交换值后，让q.next赋值给p，如果q.next为空的话，退出循环。

代码：

```
public ListNode swapPairs(ListNode head) {
       if(head==null||head.next==null)
        return head;
       ListNode pListNode=head,qListNode=null;
       int temp=0;
       while(pListNode.next!=null)
       {
        qListNode=pListNode.next;
        temp=pListNode.val;
        pListNode.val=qListNode.val;
        qListNode.val=temp;
        if(qListNode.next==null)
         break;
        else
         pListNode=qListNode.next;
       }
 return head;
   }
```

结果：

leetcode_9_Palindrome Number

描述：

Determine whether an integer is a palindrome. Do this without extra space.

By mnmlist,2015-9-2 9:35:19

click to show spoilers.

Some hints:
Could negative integers be palindromes? (ie, -1)

If you are thinking of converting the integer to string, note the restriction of using extra space.

You could also try reversing an integer. However, if you have solved the problem "Reverse Integer", you know that the reversed integer might overflow. How would you handle such case?

There is a more generic way of solving this problem.

思路：

很简单粗暴的思路就是将整数转换为StringBuilder，然后前面和后面的字符分别相比，直至前面和后面的下标相遇程序终止，任何一次比较不想等就不成立。但上面说without extra space，为啥我的程序能通过？好吧，程序不算真的通过，我用到了额外的存储空间。

代码：

```
public static boolean isPalindrome(int x) {
        if(x<0)
         return false;
        boolean flag=true;
        StringBuilder sBuilder=new StringBuilder(Integer.toString(x));
        int len=sBuilder.length();
        int i=0,j=len-1;
        while(i<j)
        {
         if(sBuilder.charAt(i)!=sBuilder.charAt(j))
         {
          flag=false;
          break;
         }
         else
         {
          i++;
          j--;
         }
        }
        return flag;
    }
```

By mnmlist,2015-9-2 9:35:19

结果：

leetcode_8_String to Integer (atoi)

描述：

Implement *atoi* to convert a string to an integer.

Hint: Carefully consider all possible input cases. If you want a challenge, please do not see below and ask yourself what are the possible input cases.

Notes: It is intended for this problem to be specified vaguely (ie, no given input specs). You are responsible to gather all the input requirements up front.

spoilers alert... click to show requirements for atoi.

Requirements for atoi:
The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes an optional initial plus or minus sign followed by as many numerical digits as possible, and interprets them as a numerical value.

The string can contain additional characters after those that form the integral number, which are ignored and have no effect on the behavior of this function.

If the first sequence of non-whitespace characters in str is not a valid integral number, or if no such sequence exists because either str is empty or it contains only whitespace characters, no conversion is performed.

If no valid conversion could be performed, a zero value is returned. If the correct value is out of the range of representable values, INT_MAX (2147483647) or INT_MIN (-2147483648) is returned.

思路：

思路是很简洁的，但正负号，空格包括中间和字符串的开始和结尾的字符，字符串溢出，整数表示的最大数和负数能表示的最小数。。。。

编程让人变的更加严谨，目测还有很大的提升空间。

By mnmlist,2015-9-2 9:35:19

代码：

```
public static int atoi(String str) {
       BigInteger integer=new BigInteger("0");
       if(str==null)
        return 0;
       str=str.trim();
       if(str.equals(""))
        return 0;
       int len=str.length();
       int flag=0;
       if(str.charAt(0)=='-')
        flag=-1;
       else if(str.charAt(0)=='+')
        flag=1;
       int i=0;
       if(flag!=0)
           i=1;
 if(str.charAt(i)>'9'||str.charAt(i)<'0')
  flag=-2;
       for(int j=i;j<len;j++)
       {
        if(str.charAt(j)>'9'||str.charAt(j)<'0')
         break;
        integer=integer.multiply(BigInteger.valueOf(10)).add(BigInteger.valueOf(str.charA
       }
       if(flag==-2)return 0;
       if(flag==-1)
       {
        integer=integer.multiply(BigInteger.valueOf(-1));
        if(integer.compareTo(BigInteger.valueOf(-2147483648))<0)
         return -2147483648;
       }else if(integer.compareTo(BigInteger.valueOf(2147483647))>0)
       {
        return 2147483647;
       }
       return integer.intValue();
   }
```

结果：

附教父里面的两句台词：

By mnmlist,2015-9-2 9:35:19

女人和小孩可以不细心，但男人不可以！
把意外当成是对个人尊严侮辱的人永远不会遭遇意外。

leetcode_7_Reverse Integer

描述:

Reverse digits of an integer.

Example1: *x = 123, return 321*
Example2: *x = -123, return -321*

click to show spoilers.

Have you thought about this?
Here are some good questions to ask before coding. Bonus points for you if you have already thought through this!

If the integer's last digit is 0, what should the output be? ie, cases such as 10, 100.

Did you notice that the reversed integer might overflow? Assume the input is a 32-bit integer, then the reverse of 1000000003 overflows. How should you handle such cases?

For the purpose of this problem, assume that your function returns 0 when the reversed integer overflows.

Update (2014-11-10):
Test cases had been added to test the overflow behavior.

思路：

思路倒是简洁明了，将整数的非符号部分的各位存储到一个数组里面，然后将各位组装起来就成为一个新的整数。当然，溢出，符号都是需要考虑的。

代码：

```
int reverse(int x) {
       int i=0;
       int num[20]={0};
       long sum=0;
       int flag=0;
```

By mnmlist,2015-9-2 9:35:19

```
        if(x<0)
            {x=-x;flag=1;}
        while(x!=0)
        {
            num[i++]=x%10;
            x/=10;
        }
        for(int j=0;j<i;j++)
        {
            sum=sum*10+num[j];
        }
        if(flag)
        sum=-sum;
        return sum;

    }
```

结果：

leetcode_6_ZigZag Conversion

描述:

The string `"PAYPALISHIRING"` is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility)

```
P   A   H   N
A P L S I I G
Y   I   R
```

And then read line by line: `"PAHNAPLSIIGYIR"`

Write the code that will take a string and make this conversion given a number of rows:

```
```

By mnmlist,2015-9-2 9:35:19

```
string convert(string text, int nRows);
```

`convert("PAYPALISHIRING", 3)` should return `"PAHNAPLSIIGYIR"`.

思路：

想了好久，思维总是局限在二维数组，找字符串的长度和二维数组的行列数之间的某种联系，想了好久，没有思路。

然后，然后就上网看了一下，有一种思路说是用字符串数组即可，就想到了StringBuilder，直接Append多好，这得比二维数组高级多少啊！然后就用StringBuilder做这道题了。

代码：

```
public String convert(String s, int nRows) {
        if(s==null)
         return null;
        else if (s.equals(""))
         return "";
        int len=s.length();
        StringBuilder resultBuilder=new StringBuilder();
        StringBuilder []sBuilder=new StringBuilder[nRows];
        for(int i=0;i<nRows;i++)
         sBuilder[i]=new StringBuilder();
        int i=0,j=0;
        while(i<len)
        {
         for(j=0;j<nRows;j++)
         {
          if(i==len)
           break;
          sBuilder[j].append(s.charAt(i));
          i++;
         }
         for(j=nRows-2;j>=1;j--)
         {
          if(i==len)
           break;
          sBuilder[j].append(s.charAt(i));
          i++;
         }
        }
        for(i=0;i<nRows;i++)
         resultBuilder.append(sBuilder[i]);
        return resultBuilder.toString();
```

```
By mnmlist,2015-9-2 9:35:19
   }
```

结果：

leetcode_5_Longest Palindromic Substring

描述：

Given a string S, find the longest palindromic substring in S. You may assume that the maximum length of S is 1000, and there exists one unique longest palindromic substring.

思路：

刚开始非常天真，以为stringBuilder.reverse() 和stingBuilder通过错位滑行就可以比较了，当然不行！因为最长子串很有可能不是从头开始的。但也可以从头到尾取子串错位滑行比较，但是时间代价太高，pass。
想了好久，终于想到了，可以从每个字符开始，分别将字符前面和后面的字符进行比较，并将统计的结果给存储下来。但是也有可能最长回文子串的字符个数是偶数，这也是一种情况。将以上两种情况求得的结果进行比较，最长的回文子串就给求出来了。

代码：

```
public String longestPalindrome(String s) {
 String str;
 if(s==null)
  return null;
```

By mnmlist,2015-9-2 9:35:19

```
if(s.length()<=1)
 return s;
int len=s.length();
int endIndex=len-1;
int i=0;
int arr1[]=new int[len];
int arr2[]=new int[len];
Arrays.fill(arr1, 0);
Arrays.fill(arr2, 0);
int start=0,end=0,count=0;
for(i=0;i<endIndex;i++)
{
 start=i;
 end=i+1;
 count=0;
 while(s.charAt(start)==s.charAt(end))
 {
  count+=2;
  start--;
  end++;
  if(start<0||end>endIndex)
   break;
 }
 arr1[i]=count;
}
for(i=1;i<endIndex;i++)
{
 start=i-1;
 end=i+1;
 count=1;
 while(s.charAt(start)==s.charAt(end))
 {
  count+=2;
  start--;
  end++;
  if(start<0||end>endIndex)
   break;
 }
 arr2[i]=count;
}
int max1=arr1[0],max1Index=0;
int max2=arr2[0],max2Index=0;
for(i=1;i<len;i++)
{
 if(max1<arr1[i])
 {
  max1=arr1[i];
  max1Index=i;
 }
 if(max2<arr2[i])
 {
  max2=arr2[i];
  max2Index=i;
 }
}
if(max1>max2)
```

By mnmlist,2015-9-2 9:35:19

```
  str=s.substring(max1Index-max1/2+1, max1Index+max1/2+1);
 else if(max1<max2)
  str=s.substring(max2Index-max2/2, max2Index+max2/2+1);
 else
 {
  if(max1Index<max2Index)
   str=s.substring(max1Index-max1/2+1, max1Index+max1/2+1);
  else
   str=s.substring(max2Index-max2/2, max2Index+max2/2+1);
 }
 return str;
}
```

结果：

leetcode_3_Longest Substring Without Repeating Characters

描述：

Given a string, find the length of the longest substring without repeating characters. For example, the longest substring without repeating letters for "abcabcbb" is "abc", which the length is 3. For "bbbbb" the longest substring is "b", with the length of 1.

思路：

本题要求的是字符串的非重复最长子串，为了在最长的时间内判断子串是否有重复，很自然地就想到了HashSet（当然还有更好的方法，我女朋友就用数组进行判重），从头开始遍历直到出现重复的字符为止，将非重复的字符数目存储起来，并查出从开始处到重复字符第一次出现的地方，删去HashSet中从开始到重复字符第一次出现的地方的所有元素，并将count减去从开始处至重复字符第一次出现的地方的个数，然后从出现重复字符的下一个字符开始统计。将字符串遍历一遍，然后统计出最长的子串所含有的字符个

By mnmlist,2015-9-2 9:35:19

数即可。

这题最容易想的就是从每个字符开始统计最长非重复连续子串的字符个数，但是很显然，时间复杂度太高。然后想到了从出现重复字符的地方开始，很显然是不对的，这题想了好久，脑子都要糊了有木有^-^!。最后费了好大的劲，才想清楚，应该从重复字符第一次出现的地方开始，这又涉及到HashSet中元素的删除，count减去从开始处到重复字符第一次出现的地方的字符个数，很多。结果，我女朋友生生地用数组就解决了，很简单的那种！晚上回去也没写程序，但心里有谱了，第二天早上到实验室一会写出来了，尽管还是很复杂！Anyway,It's done!

写程序这种事情，脑子不清楚的时候，最后不要做，否则你会后悔的，欲罢不能有做不出来！！！

代码：

```
public int lengthOfLongestSubstring(String s) {
 if(s.equals(""))
  return 0;
   int len=s.length();
      int max=0;
      int arr[]=new int[len];
      Arrays.fill(arr, 0);
      HashSet<Character>set=new HashSet<Character>();
      int count=0,j=0;
      char ch;
      int temp=0;
      for(int i=0;i<len;i++)
      {
       j=i;
       while(j<len)
       {
        ch=s.charAt(j);
        if(!set.contains(ch))
           {
            set.add(ch);
            count++;
            j++;
           }
           else
           {
            arr[i]=count;
            int end=s.indexOf(ch, temp);
            for(int k=temp;k<end;k++)
             set.remove(s.charAt(k));
            count-=(end-temp);
            i=j;
```

By mnmlist,2015-9-2 9:35:19

```
            temp=end+1;
            break;
         }
        }
        if(j==len)
        {
         arr[i]=count;
         break;
        }
       }
       max=arr[0];
       for(int i=1;i<len;i++)
        if(max<arr[i])
         max=arr[i];
       return max;
   }
```

结果：

leetcode_2_Add Two Numbers

描述：

You are given two linked lists representing two non-negative numbers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)
Output: 7 -> 0 -> 8

思路：

By mnmlist,2015-9-2 9:35:19

两个链表分别从第一个结点开始分别相加，大于10的进位和下两个结点的值一起相加直至两个结点都到链表的结尾，如果到链表结尾仍然有进位，创建一个新的节点放在较长的那个链表的后面。

代码：

```
/**
* Definition for singly-linked list.
* public class ListNode {
*     int val;
*     ListNode next;
*     ListNode(int x) {
*         val = x;
*         next = null;
*     }
* }
*/
public  ListNode addTwoNumbers(ListNode l1, ListNode l2) {
        int len1=0,len2=0;
        int sum=0,flag=0;
        ListNode pListNode=null;
        for(pListNode=l1;pListNode!=null;pListNode=pListNode.next)
        {
         if(pListNode!=null)
          len1++;
        }
        for(pListNode=l2;pListNode!=null;pListNode=pListNode.next)
        {
         if(pListNode!=null)
          len2++;
        }
        if(len1>len2)
         pListNode=l1;
        else
         pListNode=l2;
        ListNode listNode=pListNode;
        while(l1!=null&&l2!=null)
        {
         sum=l1.val+l2.val+flag;
         pListNode.val=sum%10;
         pListNode=pListNode.next;
         flag=sum/10;
         l1=l1.next;
         l2=l2.next;
        }
        while(pListNode!=null)
        {
         sum=pListNode.val+flag;
```

By mnmlist,2015-9-2 9:35:19

```
         pListNode.val=sum%10;
         flag=sum/10;
         if(flag==0)
          break;
         pListNode=pListNode.next;
        }
        if(flag!=0)
        {
         ListNode li =listNode;
         while(li.next!=null)
          li=li.next;
         ListNode temp=new ListNode(flag);
         li.next=temp;
        }
     return listNode;
    }
```

结果：

leetcode_1_Two Sum

描述：

Given an array of integers, find two numbers such that they add up to a specific target number.

The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2. Please note that your returned answers (both index1 and index2) are not zero-based.

You may assume that each input would have exactly one solution.

Input: numbers={2, 7, 11, 15}, target=9
Output: index1=1, index2=2

By mnmlist,2015-9-2 9:35:19

思路：

思路很简单，将所有的元素放入HashSet中，如果target-Numbers[i]在hashSet中，且*Numbers[i]target-Numbers[i]Numbers[i]target-Numbers[i]index*

代码：

```
public int[] twoSum(int[] numbers, int target) {
       int []result=new int[2];
       HashSet<Integer>hp=new HashSet<Integer>();
       int i=0,len=numbers.length-1;
       for(int num:numbers)
        hp.add(num);
       boolean flag=false;
       while(i<len)
       {
        flag=hp.contains(target-numbers[i]);
        if(flag)
        {
         result[0]=i+1;
               for(int j=i;j<=len;j++)
                if(target-numbers[i]==numbers[j])
                 result[1]=j+1;
               if(result[0]==result[1])
               {
                flag=false;
                i++;
                continue;
               }
               else
    break;
        }
        i++;
       }
       return result;
   }
```

结果：

By mnmlist,2015-9-2 9:35:19

leetcode_186_Reverse Words in a String

//由于考虑不够周全，犯了许多低级错误。还是有许多收获的，字符串的字符删除，字符串倒置，多余空格字符的删除。。。题目不算太难，但脑子奈何就是转不动，哎哎哎。。。思考，思考，思考，连这事都做不好，还好意思说自己什么什么么

Given an input string, reverse the string word by word.

For example,
Given s = "the sky is blue",
return "blue is sky the".

click to show clarification.

Clarification:

- What constitutes a word?
  A sequence of non-space characters constitutes a word.
- Could the input string contain leading or trailing spaces?
  Yes. However, your reversed string should not contain leading or trailing spaces.
- How about multiple spaces between two words?
  Reduce them to a single space in the reversed string.

```
class Solution {
public:
    void reverseWords(string &s) {
        int pos=0,index1=0,index2=0;
     char temp;//switch the char
     while(s.length()>0&&s[0]==' ')s.erase(0,1);//delete the blank char at the beginning c
     while(s.length()>0&&s[s.length()-1]==' ')s.erase(s.length()-1,1);//delete the blank c
     pos=0;
     while(pos<s.length())//to reverse the word of the string
     {
      while((pos<=s.length()-1)&&s[pos]==' ')
       pos++;
      index1=pos;
      while((pos<=s.length()-1)&&s[pos]!=' ')
       pos++;
      index2=pos-1;
      while(index1<index2)
      {
       temp=s[index1];
       s[index1]=s[index2];
       s[index2]=temp;
       index1++;
       index2--;
      }
     }
     index1=0;
     index2=s.length()-1;
```

By mnmlist,2015-9-2 9:35:19

```
    while(index1<index2)//to reverse the whole string
    {
     temp=s[index1];
     s[index1]=s[index2];
     s[index2]=temp;
     index1++;
     index2--;
    }
    pos=1;//to deal with the blank string
    while(pos<s.length())//delete the unnecessary blank char
    {
     if(s[pos]==' '&&s[pos-1]==' ')
      s.erase(pos,1);
     else
      pos++;

    }
   }
};
```