# Leetcode151 题目详解

## 1 第一章线性表

此类题目考察线性表的操作，例如数组，单链表，双向链表

### 1.1 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 A = [1,1,2],

Your function should return length = 2, and A is now [1,2].

**分析**

无

代码：

```
public class Solution {
    public int removeDuplicates(int[] A) {
        if(A==null||A.length==0)
            return 0;
        int index=0;
        for(int i=1;i<A.length;i++){
            if(A[index]!=A[i]){
                A[++index]=A[i];
            }
        }
        return index+1;
    }
}
```

相关题：

Remove Duplicates from Sorted Array II 见 1.2

### 1.2 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 A = [1,1,1,2,2,3],

Your function should return length = 5, and A is now [1,1,2,2,3]

**分析**

可以加一个变量来记录重复元素的个数能很好的解决问题。由于此题已经是排序的了，如果是没有排序的，可以使用 hashmap 来求元素的个数。

**代码**

```
public class Solution {
    public int removeDuplicates(int[] A) {
```

```
            if(A==null||A.length==0)
                return 0;
            int index=0,cnt=0;
            for(int i=1;i<A.length;i++){
                if(A[index]!=A[i]){
                    A[++index]=A[i];
                    cnt=0;
                }else if(A[index]==A[i]&&cnt<1){
                    A[++index]=A[i];
                    cnt++;
                }
            }
            return index+1;
        }
}
```

相关题

扩展该题到可重复 n 次的场景，只需要将 cnt 的上限设为<n-1 即可。

### 1.3 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.

**分析**

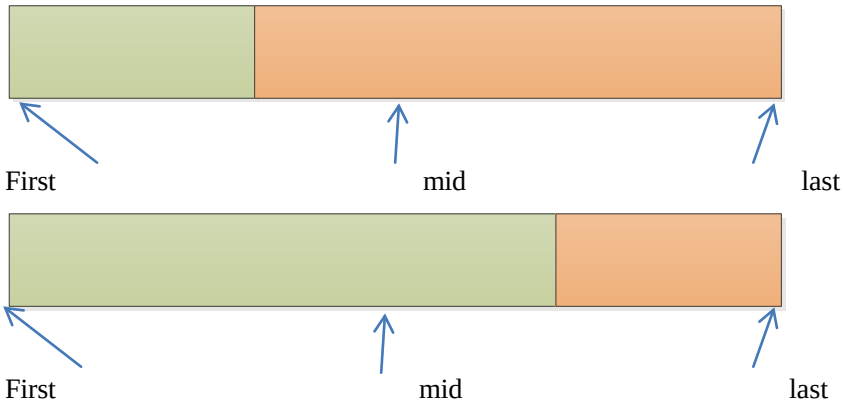
二分查找，此题要特别注意边界值。此题的边界比较多。

（1） mid 的位置判定，mid 可能在左边区域，也可能在右边区域，需求通过（A[mid]与 A[first]大小关系进行判定）

（2） 在判定边界时，要注意考虑大于，小于，等于三种情况，在写代码时，本题我开始忘记了一个等号，如代码中标红的地方。

（3） 二分的思想是一步一步将区域缩小，并且要充分考虑缩小的正确性，不能放松对边界的警惕（即要注意等于的情况）。

如图所示：




要注意 mid 落在哪个区域，通过 A[mid]与 A[first]大小比较可以确定，同时要注意边界条件的判断，当 A[mid]==A[first]应该是将其归类 A[mid]>A[first]的情况。

**代码**

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

**相关题目**

Search in Rotated Sorted Array II，见 §1.4

### 1.4 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?

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

**分析**

问题就出在边界上，即 A[mid]==A[first]的情况，如果这样，那么无法确定 mid 的位置属于上题的图一还是图二。

因此这时判断 A[first]==target,如果不成立则 first++缩小一个范围。

**代码**

```
public class Solution {
    public boolean search(int[] A, int target) {
        if(A==null||A.length==0)
            return false;
        int first=0,last=A.length-1;
        while(first<=last){
            int mid=(first+last)/2;
```

```
                if(A[mid]==target){
                    return true;
                }else if(A[mid]>A[first]){
                    if(target>=A[first]&&target<A[mid]){
                        last=mid-1;
                    }else{
                        first=mid+1;
                    }
                }else if(A[mid]<A[first]){
                    if(target>A[mid]&&target<=A[last]){
                        first=mid+1;
                    }else{
                        last=mid-1;
                    }
                }else{
                    if(A[first]==target)
                        return true;
                    else
                        first++;
                }
            }
            return false;
        }
}
```

**相关题目**

1.3 Search in Rotated Sorted Array

### 1.5 Median of Two Sorted Arrays

**描述**

There are two sorted arrays A and B 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)).

**分析**

看到本题对算法的复杂度要求非常的高，是对数级别的，因此一定要运用二分查找的思想才行。本题更通用的问法是求第 k 个数。现在需要我们求的是中位数即第（m+n）/2 个数。但也不完全准确，如果数组的个数是奇数，那么应该是（m+n）/2,如果是偶数，那么是((m+n)/2+(m+n)/2+1)/2.

现在问题转换成了求第 K+1 个数。（此处的 K 从 1 开始计算），从 1 开始是为了方便。

思路是两个数组都查找弟 k/2 个数，如果

A[k/2]==B[k/2]　return A[k/2]

A[[k/2]>B[k/2]　递归找 k-k/2 个数，且 b 的 start 位置为 k/2+1

A[k/2]<B[k/2] 同样递归找 k-k/2,且 a 的 start 位置为 k/2+1

同时要注意如果两个数组长度悬殊太大，那么段数组可能不存在第 k/2 个元素，因此这是的 K 为 short.length。

此题的边界条件判断比较繁琐，当 k==1 时还需要单独判断，这个在考虑的时候没考虑到是在用例测试的时候发现的问题。

代码

```
public class Solution {
    public double findMedianSortedArrays(int A[], int B[]) {
        int lena=A.length;
        int lenb=B.length;
        int len=lena+lenb;
        if(len%2==0){
            return    (findMedianCore(A,B,0,lena-1,0,lenb-1,len/2)+
                       findMedianCore(A,B,0,lena-1,0,lenb-1,len/2+1))/2;
        }else{
             return findMedianCore(A,B,0,lena-1,0,lenb-1,len/2+1);
        }

    }
    public double findMedianCore(int[] A,int[] B,int astart,int aend,int bstart,int bend,int k){
         int lena=aend-astart+1;
         int lenb=bend-bstart+1;
// the length of a is always smaller than the length of b
         if(lena>lenb){
              return findMedianCore(B,A,bstart,bend,astart,aend,k);
         }
         if(lena<=0){
              return B[bstart+k-1];
         }
         if(k==1){
              return A[astart]>B[bstart]?B[bstart]:A[astart];
         }
         int pa=k/2>lena?lena:k/2;
         int pb=k-pa;
         if(A[astart+pa-1]==B[bstart+pb-1]){
              return A[astart+pa-1];
         }else if(A[astart+pa-1]>B[bstart+pb-1]){
              return findMedianCore(A,B,astart,aend,bstart+pb,bend,k-pb);
         }else{
              return findMedianCore(A,B,astart+pa,aend,bstart,bend,k-pa);
         }
    }
}
```

相关题目

无

### 1.6 Longest Consecutive Sequence

描述

Given an unsorted array of integers, find the length of the longest consecutive elements sequence. For example, Given [100, 4, 200, 1, 3, 2], The longest consecutive elements

sequence is [1,2, 3, 4]. Return its length: 4. Your algorithm should run in O(n) complexity.

**分析**

此题由于复杂度是 O(n)因此不能排序。因此想到用 hash。考虑数组当 hash 但是由于题目中说是整型，因此不能用。所以使用 HashSet 解决。将所有的数放入集合中（已经去重）。取数，前后探索，记录连续的个数更新 max。当 set 为空时返回的 max 即为最大值。

本题在实验室发现一个问题，即你在遍历集合的时候修改集合会报错，java.util.ConcurrentModificationException，因此我采用了 HashMap 来做，value 记录是否访问过。

**代码**

```
import java.util.Map.Entry;
public class Solution {
    public int longestConsecutive(int[] num) {
        if(num==null)
            return 0;
        HashMap<Integer,Boolean> set=new HashMap<Integer,Boolean>();
        for(int i=0;i<num.length;i++){
            set.put(num[i],false);
        }
        int max=0;
        //traversal
        Iterator<Entry<Integer,Boolean>> it=set.entrySet().iterator();
        while(it.hasNext()){
            Entry<Integer,Boolean> tem=it.next();
            if(tem.getValue()==true)
                continue;
            int key=tem.getKey();
            int cnt=1;
            tem.setValue(true);
            while(set.get(--key)!=null){
                cnt++;
                set.put(key,true);
            }
            key=tem.getKey();
            while(set.get(++key)!=null){
                cnt++;
                set.put(key,true);
            }
            max=max>cnt?max:cnt;
        }
          return max;
    }
}
```

但是这个方法过于繁琐，我后来看了我最开始写的代码，发现我的处理方式是修改完以后每次遍历之前另外使用 it.iterator();重新获取遍历对象。

**代码 2**

```
public class Solution {
    public int longestConsecutive(int[] num) {
        if(num==null) return 0;
        HashSet<Integer> set=new HashSet<Integer>();
        for(int i=0;i<num.length;i++){
            set.add(num[i]);
        }
        Iterator<Integer> it=set.iterator();
        int count=0;
        int max=0;
        while(it.hasNext()){
            int a=it.next();
            count++;
            set.remove(a);
            int tem=a;
            while(set.contains(++tem)){
                count++;
                set.remove(tem);
            }
            tem=a;
            while(set.contains(--tem)){
                count++;
                set.remove(tem);
            }
            if(count>max)
                max=count;
            count=0;
             it=set.iterator();
        }
        return max;
    }
}
```

这个代码写的时候我又犯了错误，即两个循环一个向前查找一个向后查找，向前查找完以后要将 tem 值重新回到中间位置再向后查找。

### 1.7 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

**分析**

方法 1：暴力，每两个元素都考虑到，这样就有 $O(n^2)$复杂度

方法 2：哈希，很简单。

方法 3：先排序再夹逼，但是顺序乱了 index 也乱了，因此不好。

最终采用方法 2 的思路。

**代码**

```
public class Solution {
    public int[] twoSum(int[] numbers, int target) {
        //<number,index>
        HashMap<Integer,Integer> map=new HashMap<Integer,Integer>();
        for(int i=0;i<numbers.length;i++){
            map.put(numbers[i],i);
        }
        int index1=0,index2=0;
        for(int i=0;i<numbers.length;i++){
            index1=i;
            int find=target-numbers[index1];
            if(map.get(find)!=null&&map.get(find)!=index1){
                index2=map.get(find);
                break;
            }
        }
        int[] res=new int[2];
        res[0]=index1<index2?index1+1:index2+1;
        res[1]=index1>index2?index1+1:index2+1;
        return res;
    }
}
```

**相关题目**

3Sum，1.8

3Sum，1.9

4Sum，1.10

### 1.8 3Sum

**描述**

Given an array S of n integers, are there elements a,b,c in S such that a + b + c = 0? Find all unique triplets in the array which gives the sum of zero.

Note:

- Elements in a triplet (a,b,c) must be in non-descending order. (ie, a ≤ b ≤ c)
- The solution set must not contain duplicate triplets.

For example, given array S = {-1 0 1 2 -1 -4}.

A solution set is:

(-1, 0, 1)

(-1, -1, 2)

**分析**

利用 two sum 的夹逼方法，先对数组进行排序。然后两层循环，第一层遍历每个元素，然后找到后面数组中两个元素和等于该元素。

但是要考虑去重复的问题。在第二层循环里，碰到相同元素就继续向前。

第一层循环碰到相同元素也继续向前，这样可以去重复。

**代码**

```
public class Solution {
    public List<List<Integer>> threeSum(int[] num) {
        List<List<Integer>> res=new ArrayList<List<Integer>>();
        if(num==null||num.length==0)
            return res;
        Arrays.sort(num);//from small to big num
        for(int i=0;i<num.length;i++){
            ArrayList<Integer> sub=new ArrayList<Integer>();
            if(i>0&&num[i]==num[i-1])
                continue;
            int n1=num[i];
            int sum=-n1;
            int p=i+1,q=num.length-1;
            while(p<q){
                if(num[q]+num[p]==sum){
                    sub.add(n1);
                    sub.add(num[p]);
                    sub.add(num[q]);
                    res.add((ArrayList<Integer>)sub.clone());
                    sub=new ArrayList<Integer>();
                    while(p+1<q&&num[p+1]==num[p]){
                        p++;
                    }
                   p++;
                    while(q-1>p&&num[q-1]==num[q])
                        q--;
                   q--;
                }else if(num[q]+num[p]>sum){
                    q--;
                }else{
                    p++;
                }
            }
        }
         return res;
    }
}
```

相关题目

3Sum，1.8

3Sum，1.9

4Sum，1.10

## 1.9 3Sum Closest

描述

Given an array S of n integers, find three integers in S such that the sum is closest to a given number, target. Return the sum of the three integers. You may assume that each input would have exactly one solution.

For example, given array S = {-1 2 1 -4}, and target = 1.

The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

分析

在 3Sum 的基础上改的代码，夹逼的思想不变。最后 return 的值是三个数的和而不是差距 ab 的值，这里要特别注意。

如果差距为零就直接返回，否则就一直夹逼到不能夹逼为止。

比较值取 target-n1

代码

```
public class Solution {
    public int threeSumClosest(int[] num, int target) {
        Arrays.sort(num);//from small to big num
        int res=Integer.MAX_VALUE;
        int res_res=Integer.MAX_VALUE;
        for(int i=0;i<num.length;i++){
            if(i>0&&num[i]==num[i-1])
                continue;
            int n1=num[i];
            int sum=target-n1;//near sum
            int p=i+1,q=num.length-1;
            int sub=Integer.MAX_VALUE;
            int res_sub=Integer.MAX_VALUE;
            while(p<q){
                if(num[q]+num[p]==sum){
                    return target;
                }else if(num[q]+num[p]>sum){
                    int ab=Math.abs(num[q]+num[p]+n1-target);
                if(sub>ab){
                        sub=ab;
                        res_sub=num[q]+num[p]+n1;
                    }
                    q--;

                }else{
                    int ab=Math.abs(num[q]+num[p]+n1-target);
                    if(sub>ab){
```

```
                                    sub=ab;
                                    res_sub=num[q]+num[p]+n1;
                                }
                                p++;
                            }
                        }
                        if(res>sub){
                            res=sub;
                            res_res=res_sub;
                        }
                    }
                    return res_res;
                }
            }
```

**相关题目**

3Sum，1.8

3Sum，1.9

4Sum，1.10

**1.10 4Sum**

**描述**

Given an array S of n integers, are there elements a,b,c, and d in S such that a+b+c+d = target? Find all unique quadruplets in the array which gives the sum of target.

Note:

- Elements in a quadruplet (a,b,c,d) must be in non-descending order. (ie, a ≤ b ≤ c ≤ d)
- The solution set must not contain duplicate quadruplets.

For example, given array S = {1 0 -1 0 -2 2}, and target = 0.

A solution set is:

(-1,0, 0, 1)

(-2, -1, 1, 2)

(-2,0, 0, 2)

**分析**

本题直接使用 3Sum 的思路，该 3Sum 的代码使得能处理任意 target 值（3Sum 原本只能处理 0 值）。

然后在外面加一层循环即可。

**代码**

```
public class Solution {
    public List<List<Integer>> threeSum(int[] num,int target,int start) {
            List<List<Integer>> res=new ArrayList<List<Integer>>();
            if(num==null||num.length==0)
                return res;
            for(int i=start;i<num.length;i++){
                ArrayList<Integer> sub=new ArrayList<Integer>();
                if(i>start&&num[i]==num[i-1])
                    continue;
```

```
                int n1=num[i];
                int sum=target-n1;
                int p=i+1,q=num.length-1;
                while(p<q){
                    if(num[q]+num[p]==sum){
                        sub.add(n1);
                        sub.add(num[p]);
                        sub.add(num[q]);
                        res.add((ArrayList<Integer>)sub.clone());
                        sub=new ArrayList<Integer>();
                        while(p+1<q&&num[p+1]==num[p]){
                            p++;
                        }
                       p++;
                        while(q-1>p&&num[q-1]==num[q])
                            q--;
                       q--;
                       }else if(num[q]+num[p]>sum){
                       q--;
                    }else{
                        p++;
                    }
                }
            }
             return res;
      }
    public List<List<Integer>> fourSum(int[] num, int target) {
          List<List<Integer>> res=new ArrayList<List<Integer>>();
           if(num==null||num.length==0)
                return res;
           Arrays.sort(num);//from small to big num
           for(int i=0;i<num.length;i++){
                if(i>0&&num[i]==num[i-1])
                    continue;
                int tem=num[i];
                List<List<Integer>> half=threeSum(num,target-tem,i+1);
                for(int j=0;j<half.size();j++){
                    half.get(j).add(0,tem);
                }
                res.addAll(half);
           }
           return res;
    }
}
```

相关题目

3Sum，1.8

3Sum，1.9

4Sum，1.10

**1.11 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.

分析

无

代码

```
public class Solution {
    public int removeElement(int[] A, int elem) {
        if(A==null||A.length==0)
            return 0;
        int p=0;
        for(int q=0;q<A.length;q++)
            if(A[q]!=elem)
                A[p++]=A[q];
        return p;
    }
}
```

相关题目

无

**1.12 Next Permutation**

描述

Implement next permutation, which rearranges numbers into the lexicographically next greater permutation of numbers.

If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).

The replacement must be in-place, do not allocate extra memory.

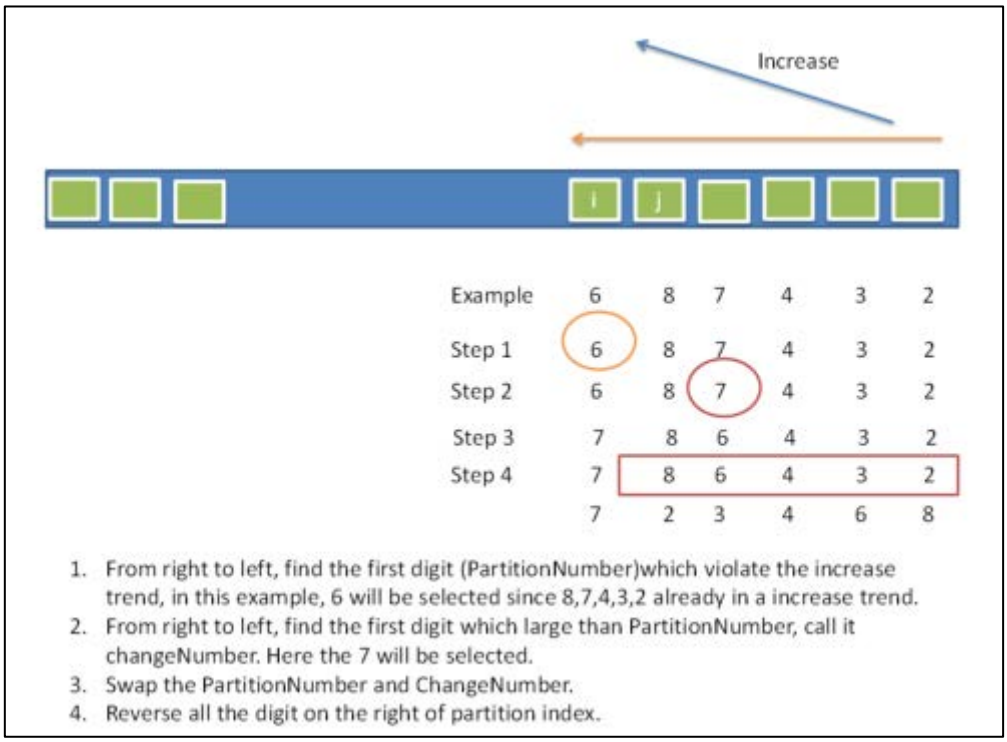
Here are some examples. Inputs are in the left-hand column and its corresponding outputs are in the right-hand column.

1,2,3 → 1,3,2

3,2,1 → 1,2,3

1,1,5 → 1,5,1

分析




下一个排列算法流程

**代码**

```
public class Solution {
    public void nextPermutation(int[] num) {
        if(num==null||num.length==0)
            return;
        //find the first small
        int i;
        for(i=num.length-2;i>=0;i--){
            if(num[i]<num[i+1]){
                break;
            }
        }
        if(i<0){
            Arrays.sort(num);
            return;
        }
        // find sec
        int j;
        for( j=num.length-1;j>i;j--){
            if(num[j]>num[i]){
                break;
            }
        }
        int tem=num[i];
        num[i]=num[j];
```

```
            num[j]=tem;
            Arrays.sort(num,i+1,num.length);
        }
}
```

**相关题目**

- Permutation Sequence, 见 §2.1.13
- Permutations, 见 §8.3
- Permutations II, 见 §8.4
- Combinations, 见 §8.5

**1.13 Permutation Sequence**

**描述**

The set [1,2,3,…,n] contains a total of n! unique permutations.By listing and labeling all of the permutations in order, We get the following sequence (ie, for n = 3):

"123"
"132"
"213"
"231"
"312"
"321"

Given n and k, return the kth permutation sequence.

Note: Given n will be between 1 and 9 inclusive.

**分析**

两种方法

1. 利用上题的 next_permutation 方法，但是这种超时了，因为每次都要排序一次，而我们只需要弟 k 个，所以做了很多无用功。
2. 数学计算
   第 i 位（i 从 0 开始）
   value=(k-1)/(n-i-1)!+1
   K=k-(num[i]-1)*(n-i-1)!
   Value 从 vis 数组中查找，是第 value 个没有访问过的数。

**代码一**

```
public class Solution {
    public void nextPermutation(int[] num) {
            if(num==null||num.length==0)
                return;
            //find the first small
            int i;
            for(i=num.length-2;i>=0;i--){
                if(num[i]<num[i+1]){
                    break;
                }
            }
            if(i<0){
                Arrays.sort(num);
```

```
                return;
            }
            // find sec
            int j;
            for( j=num.length-1;j>i;j--){
                if(num[j]>num[i]){
                    break;
                }
            }
            int tem=num[i];
            num[i]=num[j];
            num[j]=tem;
            Arrays.sort(num,i+1,num.length);
        }
    public String getPermutation(int n, int k) {
        // define the array
        int[] num=new int[n];
        for(int i=0;i<n;i++){
            num[i]=i+1;
        }
        while(k!=1){
            nextPermutation(num);
            k--;
        }
        String res="";
        for(int i=0;i<num.length;i++)
            res+=num[i];
        return res;

    }
}
```

代码二

```
public class Solution {
    public String getPermutation(int n, int k) {
        String res="";
        int[] vis=new int[n];//mark the visit status
        int[] factorial=new int[n];
        factorial[0]=1;
        for(int i=1;i<n;i++){
            factorial[i]=factorial[i-1]*(i+1);
        }
        for(int i=0;i<n-1;i++){
            int value=(k-1)/factorial[n-i-2]+1;
            k=k-(value-1)*factorial[n-i-2];
```

```
                int p=0;
                while(value!=0){
                    if(vis[p%n]==0){
                        value--;
                    }
                    p++;
                }
                p--;
                vis[p%n]=1;
                res+=(p+1);
            }
            for(int i=0;i<n;i++){
                if(vis[i]==0){
                    res+=(i+1);
                    break;
                }
            }
            return res;
        }
}
```

**1.14 Valid Sudoku**

**描述**

Determine if a Sudoku is valid, according to: Sudoku Puzzles. The Rules http://sudoku.com.au/TheRules.aspx .

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

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 5 | 3 | | | 7 | | | | |
| 6 | | | 1 | 9 | 5 | | | |
| | 9 | 8 | | | | | 6 | |
| 8 | | | | 6 | | | | 3 |
| 4 | | | 8 | | 3 | | | 1 |
| 7 | | | | 2 | | | | 6 |
| | 6 | | | | | 2 | 8 | |
| | | | 4 | 1 | 9 | | | 5 |
| | | | | 8 | | | 7 | 9 |

图 2-2　A partially filled sudoku which is valid

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

**分析**

数独的规则行不能重复，列不能重复，每个小方块（3*3）不能重复。这里只需要判断已填数的部分是否 validate 即可。

**代码**

```
public class Solution {
```

```
        public boolean isValidSudoku(char[][] board) {
                if(board==null||board.length==0)
                    return false;
                int[] vis=new int[10];
                for(int i=0;i<9;i++){
                    vis=new int[10];
                    for(int j=0;j<9;j++){
                        if(!checkValide(vis,board[i][j]))
                            return false;
                    }
                    //check col
                    vis=new int[10];
                    for(int j=0;j<9;j++){
                        if(!checkValide(vis,board[j][i]))
                            return false;
                    }
                }
                //check the block
                int x=0,y=0;
                for(int i=0;i<9;i++){
                    x=(i/3)*3;//line
                    y=(i%3)*3;//col
                    vis=new int[10];
                    for(int a=0;a<3;a++){
                        for(int b=0;b<3;b++){
                            if(!checkValide(vis,board[x+a][y+b]))
                                return false;
                        }
                    }
                }
return true;
        }
        public boolean checkValide(int[] vis,char c){
            switch(c){
            case '.':return true;
            default:
                int num=c-'0';
                if(vis[num]==1)
                    return false;
                else
                    vis[num]=1;
            }
            return true;
        }
```

```
}
```

**相关题目**

- Sudoku Solver, 见 §10.10

### 1.15 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.


图 2-3 Trapping Rain Water

**分析**

有点动态规划的思想，用 pre 和 aft 数组分别记录前面的最大墙和后面的最大墙。这样中间的面积等于 min(pre[i],aft[i])-A[i],当然如果得数是负数则忽略。

**代码**

```
public class Solution {
    public int trap(int[] A) {
        if(A==null||A.length==0)
            return 0;
        int[] pre=new int[A.length];
        int[] aft=new int[A.length];
        for(int i=1;i<A.length;i++){
            if(A[i-1]>pre[i-1])
                pre[i]=A[i-1];
            else
                pre[i]=pre[i-1];
        }
        for(int i=A.length-2;i>=0;i--){
            if(A[i+1]>aft[i+1])
                aft[i]=A[i+1];
            else
                aft[i]=aft[i+1];
        }
        int res=0;
        for(int i=0;i<A.length;i++){
            int min=pre[i]<aft[i]?pre[i]:aft[i];
            if(A[i]<min)
```

```
                    res+=min-A[i];
            }
            return res;
        }
}
```

相关题目

- Container With Most Water, 见 §12.6
- Largest Rectangle in Histogram, 见 §4.1.3

**1.16 Rotate Image**

**描述**

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

Rotate the image by 90 degrees (clockwise).

Follow up: Could you do this **in-place**?

**分析**

首先想到，纯模拟，从外到内一圈一圈的转，但这个方法太慢。如下图，首先沿着对角线翻转一次，然后沿着竖直中线翻转一次。

**代码**

```
public class Solution {
    public void rotate(int[][] matrix) {
        if(matrix==null||matrix.length==0)
            return;
        int m=matrix.length;
        int n=matrix[0].length;
        // matrix[i][j]<->matrix[j][i]
        for(int i=0;i<m;i++){
            for(int j=0;j<n;j++){
                if(i>j){
                    int tem=matrix[i][j];
                    matrix[i][j]=matrix[j][i];
                    matrix[j][i]=tem;
                }
            }
        }
        //matrix[i][j]<->matrix[m-1-i][j]
        for(int i=0;i<n/2;i++){
            for(int j=0;j<m;j++){
                int tem=matrix[j][i];
                matrix[j][i]=matrix[j][n-1-i];
                matrix[j][n-1-i]=tem;
            }

        }
    }
}
```

**1.17 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.

**分析**

高精度的加法

注意可能加出来的位数比原来位数多 1 位

**代码**

```
public class Solution {
    public int[] plusOne(int[] digits) {
        // the result len may be larger than digits len
        if(digits==null||digits.length==0)
            return digits;
        int len=digits.length;
        int ans=1;
        for(int i=len-1;i>=0;i--){
            int tem=digits[i]+ans;
            digits[i]=tem%10;
            ans=tem/10;
            if(ans==0)
                break;
        }
        int[] res=null;
        int start=0;
        if(ans!=0){
            res=new int[digits.length+1];
            res[0]=1;
            start=1;
        }else{
            res=new int[digits.length];
        }
        for(int i=0;i<digits.length;i++){
            res[i+start]=digits[i];
        }
        return res;
    }
}
```

**1.18 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?

**分析**

最基本的动态规划问题，f(n)=f(n-1)+f(n-2),f(1)=1,f(2)=2;

**代码**

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

相关题目

- Decode Ways，见 §13.10

**1.19 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

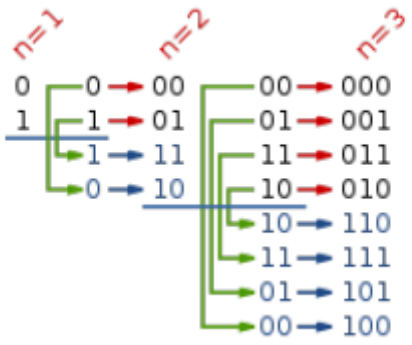
10 - 2

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.

**分析**

格雷码的生成方法如下图所示：




按照这种方式构造，十进制即每次加上 ans*2，ans 初始为 2

**代码**

```
public class Solution {
    public List<Integer> grayCode(int n) {
        List<Integer> res=new ArrayList<Integer>();
        if(n<0)
            return res;
        if(n==0)
        {
            res.add(0);
            return res;
        }
        res.add(0);res.add(1);
        int ans=2;
        while(n>1){
            n--;
            ArrayList<Integer> sub=new ArrayList<Integer>();
            for(int i=res.size()-1;i>=0;i--){
                sub.add(res.get(i)+ans);
            }
            ans*=2;
            res.addAll(sub);
        }
        return res;
    }
}
```

**1.20 Set Matrix Zeroes**

**描述**

Given a m × n matrix, if an element is 0, set its entire row and column to 0. **Do it in place**.

Follow up: Did you use extra space?

A straight forward solution using O(mn) space is probably a bad idea.

A simple improvement uses O(m + n) space, but still not the best solution.

Could you devise a constant space solution?

**分析**

此题挺繁琐的，我第一次做很多次都没通过。

代码 1：M+n 的空间复杂度，可以通过测试

代码 2：空间复杂度为 1。不使用单独的数组记录行列的 0 情况，而是使用第一行和第一列分别记录行列的情况。虽然代码二写法上复杂了些，但是思路其实跟代码 1 是一样的。

**代码 1**

```
public class Solution {
    public void setZeroes(int[][] matrix) {
        if(matrix==null||matrix.length==0)
            return ;
        int m=matrix.length;
        int n=matrix[0].length;
        boolean[] lineflag=new boolean[m];
```

```
        boolean[] colflag=new boolean[n];
        for(int i=0;i<m;i++){
            for(int j=0;j<n;j++){
                if(matrix[i][j]==0){
                    lineflag[i]=true;
                    colflag[j]=true;
                }
            }
        }
        for(int i=0;i<lineflag.length;i++)
            if(lineflag[i]==true)
                setZero(true,i,matrix);
        for(int i=0;i<colflag.length;i++)
            if(colflag[i]==true)
                setZero(false,i,matrix);
    }
    public void setZero(boolean flag,int k,int[][] matrix){
            //flag==true,line
            //flag==false,col
            if(flag==true){
                for(int j=0;j<matrix[0].length;j++){
                    matrix[k][j]=0;
                }
            }else{
                for(int i=0;i<matrix.length;i++){
                    matrix[i][k]=0;
                }
            }
        }
    }
```

**代码 2**

```
public class Solution {
    public void setZeroes(int[][] matrix) {
        if(matrix==null||matrix.length==0)
                return ;
        int m=matrix.length;
        int n=matrix[0].length;
        boolean first_line_flag=false;
        boolean first_col_flag=false;
        //check the first line
        for(int j=0;j<n;j++){
            if(matrix[0][j]==0){
                first_line_flag=true;
                break;
```

```
                }
            }
            //check the first col
            for(int i=0;i<m;i++){
                if(matrix[i][0]==0){
                    first_col_flag=true;
                    break;
                }
            }
            for(int i=1;i<m;i++){
                for(int j=1;j<n;j++){
                    if(matrix[i][j]==0){
                        matrix[0][j]=0;
                        matrix[i][0]=0;
                    }
                }
            }
            for(int i=1;i<m;i++){
                if(matrix[i][0]==0){
                    setZero(true,i,matrix);
                }
            }
            for(int j=1;j<n;j++){
                if(matrix[0][j]==0){
                    setZero(false,j,matrix);
                }
            }
            if(first_line_flag)
                setZero(true,0,matrix);
            if(first_col_flag)
                setZero(false,0,matrix);

    }
    public void setZero(boolean flag,int k,int[][] matrix){
        //flag==true,line
        //flag==false,col
        if(flag==true){
            for(int j=0;j<matrix[0].length;j++){
                matrix[k][j]=0;
            }
        }else{
            for(int i=0;i<matrix.length;i++){
                matrix[i][k]=0;
            }
```

```
        }
    }
}
```