临近面试，把刷过的题按照tag辅助Cracking code interview这本书来进行总结扫盲。第一部分：数组！

Array (数组)

数组作为一种最基本以及极为常见的数据结构，牵扯的算法琳琅满目，各种技巧性的运用令人叹为观止。因此，在这里总结leetcode上一些比较具有代表性的题目，然后给出AC代码，基本上来自于medium类型。

在Java中，数组作为对象而存在于对内存中，它不像字符串那样，具有可变性(Mutable)。对数组中某个元素重新复制，数组在堆中内存保持不变。

Problem 1: Rotate Array(Leetcode 189)

题目：
给定一个数组和一个正整数k，将数组往右移动k步。要求原地旋转。

测试样例：

> Input: [1,2,3,4,5,6,7], k = 3
> output: [5,6,7,1,2,3,4]

思路：

- 因为是往右移动k步，因此rotate之后的数组最前边k个元素是之前的后k个元素；
- 首先将前面length－k个元素rotate；
- 然后将后面k个元素rotate；
- 最后将整个数组rotate；

AC代码：

```java
public class Solution {
    public void rotate(int[] nums, int k) {
        int length = nums.length;
        if(k%length == 0) return;
        k = k%length;
        rotate(0, length-k-1, nums);
        rotate(length-k, length-1, nums);
        rotate(0, length-1, nums);
    }
    private void rotate(int start, int end, int[] nums) {
        for (int i = start; i < start + (end - start + 1)/2; i++) {
            int temp = nums[i];
            nums[i] = nums[start + end - i];
            nums[start + end - i] = temp;
        }
    }
}
```

Problem2: Merge Sorted Array:(Leetcode 88)

问题：给定两个按升序排好的数组a，b以及他们包含的元素的长度，将a，b合并到a数组，且仍然保持升序。假定a数组长度足够长到可以包含a，b所有元素。

思路：
从后往前排，这样就可以一次循环解决。

AC代码：

```java
public class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        int i = m - 1;
        int j = n - 1;
        int k = m + n - 1;
        while (i >= 0 && j >= 0) {
            if (nums1[i] > nums2[j]) {
                nums1[k] = nums1[i];
                k--;
                i--;
            } else {
                nums1[k] = nums2[j];
                k--;
                j--;
            }
        }
        while (j >= 0) {
            nums1[k] = nums2[j];
            k--;
            j--;
        }
    }
}
```

Problem3: Majority element(Leetcode 169)

问题：
给定一个数组，找出其中的绝大多数元素，其中绝大多数元素占$[n/2]$个数以上。

思路：

- 计数法
  设定一个计数器，该计数器有三种情况：
  - 为0，重新开始计数，并且设定计数元素为当前元素
  - 不为0，计数元素和当前元素相同，计数器加1；
  - 不为0，计数元素和当前元素不相同，计数器减1；
    最后输出计数元素即可。
- bit manipulation
  因为绝大多数元素占一半以上，因此对于它的二进制表示来说，如果当前位是1，那么整个数组在该位上的1的个数大于0的个数，根据这点来得出绝大多数元素。
  AC代码：只给出计数法

```java
public class Solution {
    public int majorityElement(int[] nums) {
        int result = 0;
        int count = 0;
        for(int i=0;i<nums.length;i++){
            if(count==0){
                result = nums[i];
                count=1;
            }
            else if(result==nums[i]){
                count++;
            }
            else{
                count--;
            }
        }
        return result;
    }
}
```

Unique path 1&2

简单的dp。

Two sum:

给定一个数组和一个整数k，求出这个数组中两个元素之和刚好是k，返回这两个元素的下标(基于1的下标)

思路：
利用hashmap，将值映射到下标＋1。

```java
public class Solution {
    public int[] twoSum(int[] nums, int target) {
        int[] ret = new int[2];
        HashMap<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            if (map.containsKey(target - nums[i])) {
                ret[1] = i + 1;
                ret[0] = map.get(target - nums[i]);
                return ret;
            }
            map.put(nums[i], i + 1);
        }
        return ret;
    }
}
```

Subsets

问题：
给定一个数组，返回其所有的subsets。

思路：

- backtracking
  经典backtracking，保留现场。
- bit manipulation
  根据数学知识，数组长度为k，那么其所有subsets个数为2^k个，且数组中每个元素只有两个状态：取或者不取。因此根据个数的bit 表示，可以在第几个subset中判定哪个元素是取，哪个元素不取。

AC代码：

```java
public class Solution {
    public List<List<Integer>> subsets(int[] nums) {
        //1.Using backtracking.
        //2.Using bitmanipulation
        List<List<Integer>> list = new LinkedList<List<Integer>>();
        if (nums == null) return list;
        Arrays.sort(nums);
        //List<Integer> cur = new LinkedList<Integer>();
        //helper(nums, list, cur, 0);
        return list = bitWay(nums);
    }
    public void helper(int[] nums, List<List<Integer>> list, List<Integer> cur, int start) {
        list.add(new LinkedList(cur));

        for (int i = start; i<nums.length;i++) {
            cur.add(nums[i]);
            helper(nums, list, cur, i+1);
            cur.remove(cur.size()-1);
        }
    }
    public List<List<Integer>> bitWay(int[] nums) {
        List<List<Integer>> list = new LinkedList<List<Integer>>();
        int num = (int)Math.pow(2,nums.length);
        for (int i=0;i<num;i++) {
            List<Integer> cur = new LinkedList<Integer>();
            for (int j=0;j<nums.length;j++) {
                if (((i >> j) & 1) == 1) {
                    cur.add(nums[j]);
                }
            }
            list.add(cur);
        }
        return list;
    }
}
```

Sort Color

问题：
给定一个数组，这个数组只包含0，1，2. 给数组排序，时间复杂度$O(n)$。

思路：
two-pointer。给定两个指针，一个标记0的位置，一个标记1的位置。
有两种情况：

- 如果当前元素是0，指针0和指针1都往后移；
- 如果当前元素是1，只有指针1往后移；

AC代码：

```java
public class Solution {
    public void sortColors(int[] nums) {
    if(nums==null || nums.length==0)
    return;
    int idx0 = 0;
    int idx1 = 0;
    for(int i=0;i<nums.length;i++)
    {
        if(nums[i]==0)
        {
            nums[i] = 2;
            nums[idx1++] = 1;
            nums[idx0++] = 0;
        }
        else if(nums[i]==1)
        {
            nums[i] = 2;
            nums[idx1++] = 1;
        }
    }
    }
}
```

Search a 2D Matrix:

给定一个二维数组，这个数组是按行和列升序摆放。给定一个整数，在这个二维数组中找到这个数。
思路：
二分查找。首先将行列定为到第一行的最后一列，然后不断缩小范围。

AC 代码：

```java
public class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        if (matrix == null) return false;
        int m = matrix.length;
        int n = matrix[0].length;
        if (m == 0 || n == 0) return false;
        int row = 0;
        int column = n-1;
        while(row < m && column >= 0) {
            if(matrix[row][column] == target) return true;
            else if(matrix[row][column] > target) column--;
            else row++;
        }
        return false;
    }
}
```

80. Remove Duplicates from Sorted Array II

问题：
给定一个含有重复元素的增序数组，要求返回数组中只能包含的元素最多只能重复两次。返回结果是数组的长度。要求$O(n)$

思路：
双指针法。保证每个元素出现的次数不超过两次即可。

AC 代码：

```java
public class Solution {
    public int removeDuplicates(int[] nums) {
        //The main idea is to use two poiters.
        if (nums == null || nums.length <= 0) return 0;
        int n = nums.length;
        if(n <= 2) return n;
        int index = 2;
        for(int i = 2; i < n; i++) {
            if(nums[i] != nums[index-2]) {
                nums[index++] = nums[i];
            }
        }
        return index;
    }
}
```

238. Product of Array Except Self

题目：
给定一个数组，返回一个新的数组，新数组中元素的值是原数组中除当前位置所有元素的乘机。

思路：
假设给定数组$A=[a1,a2,a3,a4]$,构造两个新的数组

$$B=[1,a1,a1*a2,a1*a2*a3]$$

和

$$C=[a2*a3*a4,a3*a4,a4,1]$$

然后将两个数组相乘即可。

```java
public class Solution {
    public int[] productExceptSelf(int[] nums) {
        int[] result = new int[nums.length];
        int p = 1;
        result[0] = 1;
        for(int i=1;i<nums.length;i++) {
            p = p * nums[i-1];
            result[i] = p;
        }
        p = 1;
        for(int i = nums.length-2;i>=0;i--) {
            p = p * nums[i+1];
            result[i] = result[i] * p;
        }
        return result;
    }
}
```

31. Next Permutation

问题：
给定一个数组，找出由这个数组组成的下一个排列。

思路：

- 从数组最后开始，找出第一个破坏降序元素的下标j;
- 然后从这个下标开始，找除从元素最后开始的第一个比它大的元素的下标i，也就是找出j之后的比它最大的最小大那个元素的下标；
- 交换这两个元素的位置，然后reverse j下标之后的数组

时间复杂度$O(n)$
AC代码：

```cpp
void nextPermutation(vector<int> &num) {
    if(num.size()<=1) return;
    int i=num.size()-1,j;
    for(j=num.size()-2; j>=0; j--){
        if(num[j]<num[j+1]) break;
    }
    if(j>=0){
        while(num[i]<=num[j]) i--;
        swap(num[i], num[j]);
    }
    reverse(num.begin()+j+1, num.end());
}

```

268. Missing Number

给定一个数组，这个数组只含有数组长度以内的数字且互不相同。加入数组长度为n,那么数组的元素只能从$[0,n]$中取得。请找出数组中丢失的那个数字。

思路：
因为数组长度为n，但是从0到n，inc有n+1个数字，因此数组中必定存在丢失数字。因为时间复杂度是$O(n)$,所以不能对数组进行排序。因此思路就是，将数组的下标和数组元素全部xor上，将最后得到的数字和数组的长度做xor，得到的那个数就是丢失的数。
原理就是，尽管下标和元素不一定一一对应，但如果元素总是在数组中，总会xor上，而那个没有没有被xor上的数字就是丢失的数字。

AC代码：

```java
public class Solution {
    public int missingNumber(int[] nums) {
        // int result = (nums.length) * (nums.length + 1) / 2;
        // for (int i = 0; i < nums.length; i++) result -= nums[i];
        //Bit method
        int ret = 0;
        int i = 0;
        for (i = 0; i < nums.length; i++) {
            ret = ret ^ i ^ nums[i];
        }
        return ret ^ i;
    }
}
```

209. Minimum Size Subarray Sum

问题：给定一个数组和一个数字，找出数组中的一个子数组，满足子数组元素之和大于等于数字且子数组要是满足条件中子数组长度最短那个。如果不存在，返回0.

思路：
双指针法。用双指针限定一个滑动窗口；

- 首先left和right指针，满足left和right之间元素大于等于目标；
- 然后将left右移一直到不满足条件；
- 一旦left右移造成不满足条件，继续将right右移，直至满足条件，返回第二步；
- 在整个过程中，保留一个变量用来记录所有满足条件的数组的长度的最小值，最后返回这个值即可；

AC代码：

```java
public class Solution {
    public int minSubArrayLen(int s, int[] nums) {
        if (nums == null || nums.length == 0) return 0;
        int left = 0;
        int right = 0;
        int sum = 0;
        int result = nums.length + 1;
        int length = 0;
        while (right < nums.length) {
            while (sum < s && right < nums.length) {
                sum += nums[right++];
            }
            while (sum >= s) {
                result = Math.min(result, right - left);
                sum -= nums[left++];
            }
        }
        return result == nums.length + 1? 0:result;
    }
}
```

53. Maximum Subarray

题目：给定一个数组，找出其中和最大的子数组并且输出最大和。

思路：动态规划典型题目。
在这里保留两个变量：一个是局部最优解，一个是全局最优解。局部最优解必须包含当前元素值，而全局最优解不一定含有当前元素。
刚开始初始化，局部最优解和全局最优解相同，都是第一个元素。
然后每当一个新的元素进来，局部最优解和全局最优解都会相应的发生变化。因此我们可以分别求出局部最优解和全局最优解的状态转移方程，这也是dp的核心所在，有了状态转移方程，一切好办。
对于局部最优解的状态转移方程是：

$$local=(local+A[i])>local?(local+A[i]):A[i]$$

如果local是负数，那么就丢弃；对于全局最优解：

$$global=global>local?global:local$$

如果局部最优大于之前的全局最优，更新全局最优。

AC代码：

```java
public int maxSubArray(int[] A) {
    if(A==null || A.length==0)
        return 0;
    int global = A[0];
    int local = A[0];
    for(int i=1;i<A.length;i++)
    {
        local = Math.max(A[i],local+A[i]);
        global = Math.max(local,global);
    }
    return global;
}
```

这道题代码虽然较为简单，但是思想却很重要。后面还有几道dp使用到这种思想，比如Jump Game和Sell Stock and Buy Stock 1.