2，数组

2018年4月27日 15:32

- 数组简介
- 面试题总体分析
  - 选题原则
  - 难度
  - 经典(常见)
  - 新颖
- 一些例题
  - 例1 局部最小值
  - 例2 第一个缺失的正整数
  - 例3 元素间的最大距离
  - 例4 只出现一次的数
  - 例5 众数问题
  - 例6 “前缀和”的应用
- 总结
- 数组简介
- 数组(array)
  - java : [], ArrayList
  - C++ : STL vector, []
  - C：只有[]
  - 理解:输入的数组通常理解为集合，我们自己可以排序，查找
  - 注意
    - C++ STL中vector的一种实现（可以规定容量，如果超过容量，就扩大该容量至两倍，重新开辟一块空间，将之前的复制过去）
    - 数组下标是一种特殊的hash…做计数（数组下标可以用来做排序or计数）

- 理解数组与map
- 给数组“顺序”

- 面试题总体分析
- 查找和排序
  - 二分查找
  - 元素交换
  - 排序，中位数
  - 归并
  - 位运算
  - 前缀和的应用
- 动态规划
- 排列组合
- 一些例题
  - 例1 局部最小值

```
/*
问题： 一个给定的不包含相同元素的整数数组，每个，局部极小值的定义是一个值比左右相邻的（如果存在）都小的值，求它的一个局部最小值


分析：首先，我们分析其是否存在?首先，全局最小值一定是一个局部极小值，因此一定存在。找最小值需整体扫一边，时间复杂度为o(n).


    能不能更少一点呢？我们规定数组下标为a[1...n],并定义a[0] = a[n+1] = 无穷大，因此，我们有a[1]<a[0],a[n]<a[n+1],
    则，我们得出结论，a[x,y],若a[x]<a[x-1],a[y]<a[y+1],则它包含一个局部极小值。
    为什么这么说呢，首先a[x,y]中一定有一个全局最小值，但如果在端点上，就需要结合上述条件限制，因此，只要保证有上述条件，局部最小一定存在


    因此，我们用二分法来确定局部最小值所在的区间，取mid = (l+r)/2,将数组分成a[l,mid]和a[mid+1,r]
    若a[mid]<a[mid+1],那么在[l,mid]满足条件，定存在一个局部极小值
    若a[mid+1]<a[mid],那么[mid+1,r]满足条件，一定存在一个局部最小值


    时复logn


    从这道题我们也了解到，二分不只可以超找有序的序列，其实是找逐步减小解空间的途径。


    思考题
        • 循环有序数组最小值、查找元素x （Leetcode 153,154）
        • 一个严格单增的数组，查找a[x] == x的位置
```

```
*/
#include<bits/stdc++.h>
using namespace std;
class Solution
{
public:
    int solve(vector<int> &s,int n)
    {
        if(n == 0) return -1;//三种特殊情况
        if(n == 1 || s[0]<s[1]) return s[0];
        if(s[n-1]<s[n-2]) return s[n-1];
        int l = 1,r = n-2;//去除端点的区间，这里经过上面的特殊情况，已知s[1]>
s[0],s[n-2]>s[n-1]
        while(l<r)
        {
            int mid = (l+r)/2;
            if(arr[mid-1]<arr[mid])
                r = mid-1;
            else if(arr[mid]>arr[mid+1])//不用考虑相等的情况，因为题目中说了没重复元
素
            {
                l = mid+1;
            }
            else//mid比两者都小
                return s[mid];
        }
        return s[l];
    }
};


int main()
{
    int a[7] = {5,4,7,8,5,2,3};
    vector<int> s(a,a+7);
    Solution so;
    cout<<so.solve(s,7)<<endl;
    return 0;
}
```

来自 <http://tool.oschina.net/highlight>

○ 例2 第一个缺失的正整数

```
/*
https://leetcode.com/problems/first-missing-positive/description/
问题：  给一个数组，找到从1开始，第一个不在里面的正整数。
    例如 [3,4,-1,1]输出2。  (Leetcode 41)
```

分析：将每个数字放至它正确的位置，比如我们找到5,将其和A[4]位置上的数交换。

最后，找出第一个数不对的位置，返回该位置+1,即可，时复o(n)

*/

```
class Solution {
public:
    int firstMissingPositive(vector<int>& nums) {
        int n = nums.size();
        for(int i = 0;i<n;i++)
        {
            while(nums[i]>0&&nums[i]<=n&&nums[nums[i]-1]!=nums[i])//需要交换要满足的条件，1，是1-n范围内的整数，2，该位置上放得不是对的数
                swap(nums[i],nums[nums[i]-1]);
        }

        for(int i = 0;i<n;i++)
        {
            if(nums[i] != i+1)
                return i+1;
        }
        return n+1;
    }
};
```

来自 <http://tool.oschina.net/highlight>

○ 例3 元素间的最大距离

```
/*
https://leetcode.com/problems/maximum-gap/description/
问题：给定一个整数数组(n > 1)，求把这些整数表示在数轴上，相邻两个数差的最大值。（Leetcode 164）

分析：首先，最简单的思路是排序，那么有没有更好的思路呢？
    1 首先，我们找到最小值x 最大值y，若两者相等，返回0
    2 然后把数放进n+1个桶，每个桶的大小为double d = (x-y)/n+1
    每个桶区间为[x+i*d,x+(i+1)*d),i取0-n，是左闭右开区间，最后一个桶是双闭区间
    最小值在0号桶，最大值在n号桶，
    如，数r，放进((r-x)/d),这里进行下取整。注意r == y时，答案取n（即最大值特殊处理下）
    3 我们容易知道n个数，n+1个桶，定有一个桶非空，明显非0,n号桶，那么其一定在中间，
    中间有空桶，那么最大值的间隔是一定大于d的，这说明最大间隙的两个数定不出现在一个桶内。
    而是出现在一个非空桶的最大值和下一个非空桶的最小值之间。
    因此，我们记录每个桶的最大值和最小值即可。
    时间复杂度为O(n).
*/
#include<bits/stdc++.h>
using namespace std;
```

```
class Solution {
public:
    int maximumGap(vector<int>& nums) {
        int n = nums.size();
        if(n ==0 || n==1) return 0;//特殊情况，0个值or1个值无法求gap

        int x = nums[0],y = nums[0];//遍历，求得数组最小x最大值y
        for(auto num:nums)
        {
            x = min(x,num);
            y = max(y,num);
        }

        double d = double(y-x)/(n+1);//求桶距d
        vector<int> bucketsMin(n+1,INT_MAX);//存贮每号桶的最小值，初始化size,并且设置初始值
        vector<int> bucketsMax(n+1,INT_MIN);

        for(auto num:nums)//统计每号桶的最大值和最小值
        {
            int idx = n;//桶号，最大值的特殊情况为n
            if(num != y)//非最大值的情况
                idx = (num-x)/d;//正常向下取整
            bucketsMin[idx] = min(num,bucketsMin[idx]);
            bucketsMax[idx] = max(num,bucketsMax[idx]);
        }

        int maxGap = INT_MIN;
        int pre = x;//前一个桶的最大值，先随便初始化为数组最小值
        for(int i = 0;i<=n;i++)
        {
            if(bucketsMin[i] == INT_MAX && bucketsMax[i] == INT_MIN)//桶空
                continue;
            maxGap = max(maxGap,bucketsMin[i]-pre);
            pre = bucketsMax[i];
        }
        return maxGap;

    }
};
```

来自 <http://tool.oschina.net/highlight>

○ 例4 只出现一次的数

```
/*
问题：  一个数组，所有元素都出现了两次，只有两个数只出现了一次，求这两个数。
```

```
分析：所有数做亦或，得到的结果为这俩数亦或的结果。然后我们找到这个结果中二进制表示第一个非0的位，

    然后将所有数根据该位是否为1分开，分别做亦或，得到的结果即为这两个数。

*/
#include<bits/stdc++.h>
using namespace std;


class Solution
{
public:
    void solve(vector<int> a)
    {
        int res = 0;
        for(int i = 0;i<a.size();i++)
        {
            res ^= a[i];
        }


        int temp = res;
        int flag = 0;//哪一位非0


        for(int i = 0;i<32;i++)
        {
            if((temp>>i) & 1 == 1)
            {
                flag = i;
                break;
            }
        }
        int res1 = 0;
        for(int i = 0;i<a.size();i++)
        {
            if((a[i]>>flag)&1 == 1)//只处理flag位为0的值
                res1^=a[i];
        }
        res = temp^res1;
        cout<<res<<" "<<res1<<endl;

    }
};


int main()
{
    int arr[6] = {1,2,4,5,2,1};
    vector<int> a(arr,arr+6);
    Solution so;
    so.solve(a);
    return 0;
```

```
}
```

来自 <http://tool.oschina.net/highlight>

- 思考题
  - Leetcode 137 除一个外，所有数出现了3次，求那个数*（难）
  - 1-100，缺少了两个数，求这两个数？ 位运算？解方程？

  - 例5 众数问题

```
/*
问题：找出超过一半的数

分析：众数出现的次数大于所有其余数出现次数之和。因此，当你每次删除两个不同的数，众数不变
    为什么呢？如果扔掉一个众数一个非众，不变。扔掉两个非众，仍不变。

    整体的思想就是，维护一个x，再来一个y，不同，则都丢掉。相同，则记录x出现的次数
    int count = 0,x;
    for(int i = 0;i<n;i++)//士兵守阵地的思想
    {
        if(count == 0) { x = a[i],count = 1;}//count为0了，更新x
        else if(x == a[i]) ++count;//相同，则计数++
        else --count;//不相同，则count--,表示丢掉了一个x，同时a[i]也没存贮
    }//最后x就是我们想要的

    注意，有些题目要数一下x的出现次数是否真的大于1半，

    拓展题：如何找到出现次数严格大于1/k的数？（众数是1/2）
        提示：保留k-1个数，来一个数，和这k-1个数比较，相同的话，对应count+1,不同，每个数出现的次数减1.
        如何维持这k-1个数，用hash or map?

*/
#include<bits/stdc++.h>
using namespace std;
class Solution
{
public:
    int solution(vector<int> a)
    {
        int res,count = 0;
        for(int i = 0;i<a.size();i++)
        {
            if(count == 0)
            {
                res = a[i];
                count = 1;
            }
```

```
            else if(a[i] == res)
                count++;
            else
                count--;
        }
        return res;
    }
};


int main()
{
    int arr[6] = {1,2,2,1,1,3};
    vector<int> a(arr,arr+6);
    Solution so;
    cout<<so.solution(a)<<endl;
    return 0;
}
```

来自 <http://tool.oschina.net/highlight>

○ 例6 “前缀和”的应用

```
/*
问题：给定浮点数组a,求一个数组
b, b[i] = a[0] * a[1] *…*a[i - 1] * a[i + 1] * …*a[n - 1],跳过a[i]
    不能使用除法，不允许再开数组

分析：先求后缀积，从a[i]****a[n-1]
    for(int i = n-1;i>=0;i--)
        b[i] = a[i]*((i==n-1)?1:b[i+1])//注意，这里用了迭代的思想，乘的是b[i+1]
     顺便求前缀积，
    顺便求前缀积
    for(int i = 0,j =1;i<n;j*=a[i++])//j初始化为1,是因为累乘
        b[i]= j*((i==n-1)?1:b[i+1]);//注意，这里是先执行这个，然后才j*=a[i++]的，因此此时
b[i] 就是b[i+1]*(前缀和a[i-1]，*a[i]还没来得及算

拓展：a[i] + a[i + 1] + … + a[j] = sum[j] - sum[i - 1]，易知，sum为前缀和

思考题：给定数组，求连续子数组和，绝对值最小，即和越接近0.
    结合上面，只需两两求前缀和的差，排个序

思考题2:    把一个数组从中间p位置分开，使得a[0] + ..+ a[p - 1]与
a[p] + a[p + 1] + …+ a[n - 1]差值最小?
        提示：前缀和，与总和减去该前缀和（后面部分），的差最小，枚举

*/
#include<bits/stdc++.h>
using namespace std;
class Solution
{
public:
```

```
    vector<int> solution(vector<int> a)
    {

        vector<int> b(4,0);//这里一定要初始化
        int n = a.size();

        for(int i = n-1;i>=0;i--)//求后缀和
        {
            if(i == n-1)
                b[i] = a[i];
            else
                b[i] = a[i]*b[i+1];
            //b[i] = a[i]*((i == n-1)?1:b[i+1]);
        }

        //更新b
        for(int i = 0,j=1;i<n;j*=a[i++])
        {
            b[i] = j*((i == n-1)?1:b[i+1]);
        }
        return b;
    }
};


int main()
{
    int arr[4] = {1,2,3,4};
    vector<int> a(arr,arr+4);
    Solution so;
    vector<int> b;
    b = so.solution(a);
    for(int i = 0;i<4;i++)
        cout<<b[i]<<endl;
    return 0;

}
```

来自 <http://tool.oschina.net/highlight>

- 总结
- 利用序
  - 理解二分查找
- 利用前缀和
  - 查找、计算、排序
- 理解数组
  - map

- 用数组实现高级数据结构
  - 一般树：存每个节点的父亲 （并查集）
  - 二叉树：下标从1开始a[i]的儿子是a[i * 2]和a[i * 2 + 1] (堆)
- 抓住简单题
  - 分治法求逆序对数
  - 有序数组归并 leetcode做过
  - 两个有序数组的中位数 leetcode做过
  - 两头扫的方法 （2-SUM, 3-SUM)