10，o(N)时间复杂度解决面试中遇到的问题下

2018年4月27日 19:50

- 例题汇总
  - 例1 最大子数组（和、乘积）
  - 例2 循环移位
  - 例3 快排partition
  - 例4 众数问题
  - 例5 单调堆栈
  - 例6 单调队列
  - 例7 树
  - 例8 滑动窗口
  - 例9 其他
- 结束语
- 例题汇总
  - 例1 最大子数组（和、乘积）

```
/*最大子数组和

问题1：给一个数组，求最大的连续子数组和。 （动态规划实战例2， leetcode 53）
        https://leetcode.com/problems/maximum-subarray/description/
        • 方法1 记录最小前缀和——两个前缀和的差就是一段连续的子数组
        • 方法2 动态规划，记录以每个位置结束的最大子数组的和。

问题2：给一个数组，求最大的连续子数组乘积 （leetcode 152）
        https://leetcode.com/problems/maximum-product-subarray/description/
        类似例1.1
        考虑的问题：
            溢出——没有溢出
            到当前项乘积最大:之前乘积绝对值大(因此即要保存之前的最小乘积（考虑负数的情况），也要保存之前最大乘积）
*/
class Solution1 {
public:
    int maxSubArray(vector<int>& nums) {
        /*
        dp[i]为以第i个元素结尾的最大连续子序列和，那么
dp[i] = max(0,dp[i-1])+nums[i];
        即，如果i之前的subarray和大于0时，是有帮助的，此时加上。否则，是负数，还不如从头来过，此时，其最大子序列和为其本身
        return max(dp[i])即可
        */
```

```
        int dp[nums.size()];
        dp[0] = nums[0];
        int res = dp[0];
        for(int i = 1;i<nums.size();i++)
        {
            dp[i] = max(0,dp[i-1])+nums[i];
            res = max(res,dp[i]);
        }
        return res;
    }
};

class Solution2 {
public:
    int maxProduct(vector<int>& nums) {
        int n = nums.size();
        if(n == 0) return 1;//特殊情况
        int mini = nums[0],maxi = nums[0];//以i项结尾的最小or最大子数组积
        int ansmax=nums[0];//全局最大子数组积,
        for(int i = 1;i<n;i++)
        {
            int tempmin = min(nums[i],min(mini*nums[i],maxi*nums[i]));//此时mini,
maxi记录的上一项为止的最小or最大子数组乘积
            int tempmax = max(nums[i],max(mini*nums[i],maxi*nums[i]));
            mini = tempmin;
            maxi = tempmax;
            ansmax = max(maxi,ansmax);
            //ansmin = min(mini,ansmin);//若求最小
        }
        return ansmax;
    }
};
```

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

○ 例2 循环移位

```
/*
    循环移位
    问题1： 一个数组，比如{1,2,3,4,5}循环移动一位就是{2,3,4,5,1},再移动一位变为
{3,4,5,1,2}。
    分析： 长度为n, 把它移动m位,和移动m % n位是一样的。
        翻转前m位 (0,m-1)
        翻转后(n - m)位 (m,n-1)
        总体再翻转 (0,n-1)
        翻转可以O(n)做到：
            for (int i = from, j = to; i < j; swap(a[i++], a[j--]));
    问题2：单词翻转 (字符串高频面试题 例5 1_5.cpp)
    问题3：回文判断
        for (int i = from, j = to; i < j;)
```

```
            if(a[i++] != a[j--]) return false;


*/
```

○ 例3 快排partition

```
/*
快排partition
    问题1：荷兰国旗问题 （ leetcode 75 ）https://leetcode.com/problems/sort-colors/description/
    给定一个带有红色，白色或蓝色的n个对象的数组，将它们就地排序，以便相同颜色的对象相邻，颜色顺序为红色，白色和蓝色。
    输入：[2,0,2,1,1,0]
    输出：[0,0,1,1,2,2]
    分析：类似快排的思路，维持两个索引，zero和second，用来记录0的最右位置和2的最左位置


    问题2：奇偶数分开，  正负数分开
    分析:利用partition，可以n时间复杂度，关键是维持一个分界点，然后从左往右扫，不符合的就交换，并更新分界点


    问题3：01交换排序 （字符串高频面试题 例1，1_1.cpp）
    分析：明显，排好序之后0在左边，1在右边，因此左边的0和右边的1都可以不考虑，
     维持两个指针，分别从左到右扫第一个1的位置，和从右往左扫第1个0的位置，交换即可



    问题4：交换星号 （字符串高频面试题 例3,1_3.cpp）
    分析：既有partition，又有倒着复制的思想
    (1)如果只有partition，那么数字的相对位置会发生变化,这个思想如下：设[0,i-1]都是*,[i,j-1]都是数字，[j,n-1]为未探测
    for(int i = 0,j = 0;j<n;j++)
        if(s[j] == '*') swap(s[i++],s[j]);//j在j到n的区间遍历，如果其是数字，不管，是*,就和i处交换，i同时++。
        i其实是*和数字的分界点
    (2)再加上倒着复制的思想，
    int j = n-1;//新索引
    for(int i = n-1;i>=0;i--)
        if(isdigit(s[i])) s[j--] = s[i];//倒着来，如果是数字，不管，直接放即可。这样结束，后面全是数字了
    //下面再将前面全赋值为*即可
    for(;j>=0;j--)
        s[j] = '*';


    问题5：第一个缺失的整数 （数组高频面试题 例2 leetcode 41 2_2.cpp）
    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)


    问题6：中位数、第k大（小）的数、最小的k个数
        （1）找第k小的数关键
            5数取中做分区元素，
            Partition分三段（小于，等于，大于）（分两段有相同数会退化）


        （2）找到最小的k个数
```

```
            基于partition的方法找到的数是无序的
            如果有序建议用最大堆nlogk

*/
class Solution1 {
public:
    //类似快排的思路，维持两个索引，zero和second，用来记录0的最右位置和2的最左位置
    //进行两次快排，第一次从左到右遇到2，如果在second之前，则与second处交换，并将second往左移
    //第二次从左到右遇到0，且在zero之后，就和zero处交换，并zero++;
    void sortColors(vector<int>& nums) {
        int zero = 0,second = nums.size()-1;
        for(int i = 0;i<=second;i++)
        {
            while(nums[i] == 2 && i<second) swap(nums[i],nums[second--]);
            while(nums[i] == 0 && i>zero) swap(nums[i],nums[zero++]);
        }

    }
};
```

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

例4 众数问题

```
/*
众数问题
    问题1：找出出现次数超过一半的数 （数组高频面试题例5 2_5.cpp）
    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?

    问题2：推广找出出现次数大于1/k的数，用(k - 1)个map (hash table)，复杂度O(k * n)（hash)，注意k是常数的时候就是O(n)

*/
```

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

例5 单调堆栈

```
/*
    单调堆栈
    问题1：最大直方图 （栈和队列例5 3_5.cpp）
        https://leetcode.com/problems/largest-rectangle-in-histogram/description/
        问题： 给出一个直方图，求最大面积矩形 （Leetcode 84）

        分析：用堆栈分析每一块板能延伸到的左右边界（要求，这个延伸范围内的板高都大于等于当前），为让所有元素出栈，再人为的在最后加一块高为0的板

        每块板i的高度h(i)入栈，高度比栈顶大入栈，该板的左边界定为之前栈顶元素的编号

        高度比栈顶小，栈顶元素出栈，确定右边界为当前板号i。

        左边界确定为当前板的下边板的编号（如果栈为空，则-1），有板比当前板小出栈，右边界确定为当前让其出栈的板号

        时间复杂度为O(n)


*/
```

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

例6 单调队列

```
/*
    单调队列

    问题1：滑动窗口最值 （栈和队列 例6 3_6.cpp）

    滑动窗口最大值：给定一个数组a [0..n],还有一个值k，计算数组b [i] = max(a[i - (k-1),i-(k-2).. i-(k-k)])
    若i<k-1,认为负数下标对应值是无穷小

    分析：即分析以i之前k个元素范围到i 这个范围的最大值，作为b[i]处的值，

    方法1：用一个最大堆存放这k个数，时间复杂度为n*logk,因为n个元素，然后每出入一次堆都为logk的时间复杂度
    如果计算好了b[i-1]，那么如何计算b[i]呢？
    首先， a[i-k]（即a[i-1 - (k-1)出堆
    然后a[i]入堆
    然后b[i] = 堆顶即可

    但这个时间复杂度还不是最理想的，我们可以做到线性
    方法2：我们可以使用一个双端队列，
    如果队尾的数<=将要入队的数a[i],则扔掉队尾的数，将a[i]放入队尾
    否则，如果入队元素小于队尾，不仍队尾的，直接诶放入队尾即可
    这样一来，队列的元素从队头到队尾是单减的，队头永远是窗口最大值
    队头何时过期呢？i>队头元素索引+k-1 比如队头为a[0],k=3,i= 3时，就过期了
    时间复杂度O(n)
```

问题2：给定一个数组A和整数K,问有多少对下标i <= j满足
max(A[i..j]) - min(A[i..j]) <= K
分析：如果(i,j)满足条件，则子窗口(i + 1,j) (i + 2, j)…都满足条件。这是因为，窗口越小，最大值肯定越小，最小值肯定越大，因此差值定小于k
对每个i，找到第一个不满足条件的j（此时这个窗口已经是从i开始的最大的窗口了）
那么在计算i+1开始的窗口时，j是无需往左，只需从当前位置继续往右
那么，如何求窗口[i..j]的最大最小值？
用两个单调队列，一个队头维持窗口最大值，一个队头维持窗口最小值
如果最大值-最小值<=k,j继续往右，直到不满足，那么有j-i对下标是满足条件的。

*/

```
#include<bits/stdc++.h>
using namespace std;
class Solution2
{
public:
    int solution(int k,vector<int> A)
    {
        deque<int> qmin,qmax;//存放窗口[i,j]中最小和最大值索引
        int answer = 0;
        for(int i = 0,j = 0;i<A.size();i++)
        {
            while(j<A.size())
            {
                while(!qmin.empty()&&(A[j]<=A[qmin.back()]))//如果队尾元素比a[j]要大or相等，没必要留
                    qmin.pop_back();
                qmin.push_back(j);
                while(!qmax.empty()&&A[j]>=A[qmax.back()])//如果队尾元素比a[j]要小or等于，没必要留
                    qmax.pop_back();
                qmax.push_back(j);
                if(A[qmax.front()]-A[qmin.front()]<=k)
                    ++j;
                else
                    break;
            }
            if(qmin.front() == i)//若是i， 下一次循环变成了i+1,i这个位置上的元素应该过期掉
                qmin.pop_front();
            if(qmax.front() == i)
                qmax.pop_front();
            answer+=(j-i-1);//注意这里j多加了一个位置
        }
        return answer;
    }
};

int main()
{
    int a[6] = {3,4,6,7,9,8};
    int k = 3;
    vector<int> A(a,a+6);
```

```
    Solution2 s;
    cout<<s.solution(k,A)<<endl;
    return 0;
}
```

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

-

- **例7 树**

```
/*
    树 5_2.cpp
    问题1 树的高度（最大深度）leetcode104
https://leetcode.com/problems/maximum-depth-of-binary-tree/description/

    问题2 二叉树对称判断
    leetcode101
https://leetcode.com/problems/symmetric-tree/description/

    问题3 二叉树平衡判断
    a binary tree in which the depth of the two subtrees of
every node never differ by more than 1.），leetcode110
https://leetcode.com/problems/balanced-binary-tree/description/

    问题4 二叉树的最小深度
    （从根到叶子最小经过的节点数）(Leetcode  111) 注意空子树
https://leetcode.com/problems/minimum-depth-of-binary-tree/description/

    问题5 指定和的路径
    (Leetcode 124) 二叉树每个节点有一个整数，返回和最大的路径。
https://leetcode.com/problems/binary-tree-maximum-path-sum/description/

分析：这个最大路径有三种可能：
    左子树延伸下去的最大路径，右子树延伸下去的最大路径，和通过根节点的路径

    问题6 二叉树双向链表转换 5-3.cpp

    问题7 前中后序遍历 5-1.cpp

    问题8 给定一个树（无向无环图），求距离最远的两个点(定是叶子节点）（树的直径）
    ○ 简单、巧妙地贪心
        • 以任意一点为根，找到距离它最远的节点x（dfs一次）
        • 以x为根找到距离x最远的点y（再dfs一次）
        • (x,y)就是一条直径
    ○ 如何找最远的点？ dfs求深度



*/
```

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

- 例8 滑动窗口

```
/*
滑动窗口
    问题1：Leetcode 209 给定一个数组，里面全是正整数，再给一个正整数s，求数组里面最少多少个连续的数，满足总和不小于s
        核心，大窗口不满足条件，它的任意小窗口也不满足条件
        窗口[i..j]
            过小———++j
            过大———--i


    问题2 子串变位词 （字符串高频面试题精讲例4， 1-4.cpp）
    ○ 思考题1 最短子串包含全部字母 Leetcode 76 和问题2是一个问题
    ○ 思考题2 无重复字符的最长子串 leetcode 3 类似问题1,这里是小窗口不满足，则大窗口也不满足，
    若太大了，则将左边界往右移。若窗口太小了（窗口元素都只出现1次），则扩大窗口。


*/
class Solution1 {
public:
    //核心：大窗口不满足条件，它的任意子窗口也不满足。窗口[i,j]，过小，则++j，过大，则减小它，i往右移
    int minSubArrayLen(int s, vector<int>& nums) {
        int n = nums.size();


        int answer = INT_MAX;
        for(int i = 0,j= 0,sum = 0;j<n;)//[i,..,j-1]
        {
            while(sum<s && j<n)//结束时，j为对的j位置+1
                sum+=nums[j++];
            if(sum>=s)//往右增大i，从而缩小窗口大小。
            {
                for(;sum>=s;sum-=nums[i++]);//结束后，i为对的位置+1
                answer= min(answer,j-i+1);
            }
        }
        return (answer==INT_MAX)?0:answer;//为了处理[1,1] 3 这种特殊情况。
    }
};
```

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

- 思考题1 最短子串包含全部字母 Leetcode 76
- 思考题2 无重复字符的最长子串
- 例9 链表

```
/*
链表
    问题1： （k个一组）反转leetcode 206,92,25 4_2.cpp
    问题2：插入 leetcode147 4_1.cpp
```

问题3：删除 leetcode203,82,83 4_1.cpp
问题4：复制 leetcode138 4_5.cpp
问题5：求交 leetcode160 4_4.cpp
问题6：找环 leetcode141,142, 4_3.cpp
问题7：倒数第k个数 leetcode19 4_6.cpp

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

例10 其他

```
/*
问题1：2-sum
    分析：1,排序，然后经典两头扫
        2,用hash查找，对于x，找s-x是否存在

问题2：给定一个1-n的排列，每次只能把一个数放到序列末尾，至少几次能排好顺序？
        为什么要移动1？其他都排好了，1自然就好了
        如果要移动x，则之后我们必须把(x + 1), (x + 2).. n都移动到末尾。
        因此，从1-(x-1)必须有序的
        因此我们的目的是找到尽可能大的一个x，让其前面的1-x-1的数都是有序的。

问题3：给定一个1-n的排列，每次可以把一个数放到序列开头，也可以放到结尾，至少几次能排好序？
        分析
        我们可以把1..y移动到开头
        然后把x..n移动到末尾
        但要求[y + 1.. x — 1] 必须按顺序出现，因此，我们仍需统计一下多少数是按顺序出现的，这里只不过不用从1开始
        dp[x]表示从x开始在原数组中往后按顺序出现的最长长度
            即dp[x]的值表示：x, x + 1, …x + dp[x] - 1按顺序出现
            倒着循环i,  dp[a[i]] = dp[a[i] + 1] + 1

*/


class Solution1
{
public:
    int solution(vector<int> a)
    {
        int n = a.size(),want =1 ;//want为x的值,初始化为1
        for(int i = 0;i<n;i++)
        {
            if(a[i] == want)//这里want很精妙，相当于在数组范围内找到了所有有序的部分，
            //比如这里，找到了1,就该接着看1后面能不能找到2,一直看找到多少有序的数
                want++;
        }
        return n-want+1;//want,...,n-1都是要被移动的, 相当于n-(want-1)
    }
};
```

```
class Solution2
{
public:
    int solution(vector<int> a)
    int n = a.size(),m = 0;//m为最长有序的长度
    vector<int> dp(n+2,0);//使用1,...,n+1,因为数是从1-n的，我们统计的是以数x开头往后的有序长度
    for(int i = n-1;i>=0;i--)
    {
        m = max(m,dp[a[i]] = dp[a[i]+1]+1);//这个dp方程很奇妙，它计算每个以x开头的最大有序长度时，等于以x+1开头的最大有序长度+1。
        //当然这些有序长度最开始均初始化为0的，以 1 3 4 2为例，该方法算得dp2 = dp3 + 1 = 1,dp4 = dp5+1 = 1,dp3 = dp4+1 = 2,dp1 = dp2+1 = 2
        //至于这里为什么以倒序的顺序计算，是因为这样，能在计算dpx时，dpx+1的值是有效的
    }
    return n-m;
};
```

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

- O(n)很神奇
- 多思考，勤练习
- 多写代码，多实践
- 结束语