从第 K 元素看数据结构

这篇文章讨论的是序列中第K大或第K小元素，由于第K大元素可以转化为求第N-K+1小元素（N为序列的长度），所以，本文专注于讨论第K小元素。

本文讨论的几个问题：

1. 对给定整数序列，求该序列中第K小的元素。
2. 对某一整数序列，允许动态更改序列中的数。动态查询序列中第K小元素。
3. 给定一个整数序列和若干个区间，回答该区间内第K小元素。
4. 对某一整数序列，允许动态更改序列中的数。动态查询序列中的第K小元素。

【关键字】
第K小元素　树状数组　线段树　平衡二叉树　归并树　划分树　单调队列　堆　块状表

【问题一】

<u>问题描述</u>：
给出一个乱序整数序列 a[1…n] ，求该序列中的第K小元素。（1<=K<=N）。

<u>算法分析</u>：
用基于快速排序的分治算法，期望复杂度为 O(N)。
<u>代码</u>：

```
int qs(int *a , int l , int r , int k){
    if(l == r)   return a[l] ;
    int i = l , j = r , x = a[(l+r)>>1] , temp ;
    do{
        while(a[i] < x) ++ i ;
        while(a[j] > x) -- j ;
        if(i <= j){
            temp = a[i] ;   a[i] = a[j] ,   a[j] = temp ;
            i++ ;   j-- ;
        }
    }while(i<=j) ;
    if(k <= j)   return qs(a , l , j , k);
    if(k >= i)   return qs(a , i , r , k);
    return x ;
}
```

练习

RQNOJ 350　这题数据量比较小 1≤N≤10000,1≤M≤2000 。所以计算量不会超过 10^7。当然用到后面的归并树或划分树，能将复杂度降低。

【问题二】

问题描述：

给出一个乱序整数序列 a[1...n] ，有 3 种操作：

操作一：ADD　NUM　　往序列添加一个数 NUM。

操作二：DEL　NUM　　从序列中删除一个数 NUM（若有多个，只删除一个）。

操作三：QUERY　K　　询问当前序列中第 K 小的数。

输出每次询问的数。假设操作的次数为 M。

算法分析：

这题实际上就是一边动态增删点，一边查询第 K 小数。这类题有两种思维方法：一是二分答案，对当前测试值 mid，查询 mid 在当前序列中的排名 rank ， 然后根据 rank 决定向左边还是右边继续二分。另一种是直接求第 K 小元素。

这个题可以用各种类型的数据结构解决，其时间复杂度和编程复杂度稍有区别：

线段树：运用第一种思维，当添加（删除）一个数 x 时，相当于往线段树上添加（删除）一条(x , maxlen)（注意是闭区间）长度的线段。这样询问时，覆盖[mid , mid]区间的线段数就是比 mid 小的数，加上 1 就是 rank。二分次数为 log(maxlen) ，查一次 mid 的 rank ， 复杂度为 O(logN) 。所以总复杂度上界为 O(M*logN*logN) 。为方便比较，这里认为 log(maxlen)等于 logN。

树状数组：

第一种思维：这个相对简单，因为树状数组求比 mid 小的数就一个 getsum(mid-1)就搞定。复杂度同线段树一样。只是常数很小，代码量也很小。

第二种思维：我只能说很巧妙。回顾树状数组求和时的操作：

代码

```
int getsum(int x ){
    int res = 0 ;
    for( ; x>0 ; x-=lowbit(x) )    res += arr[x] ;
    return res ;
}
```

对二进制数 10100010 是依次累加 arr[10100010] , arr[10100000] , arr[10000000] 。从而得到小于 x 的数的个数。当反过来看的时候，就有了这种方法：从高位到低位依次确定答案的当前为是 0 还是 1，首先假设是 1，判断累计结果是否会超过 K，超过 K 则假设不成立，应为 0，否则继续确定下一位。看程序就明白了。

代码：

```
int getkth(int k){
        int ans = 0 , cnt = 0 , i ;
        for(i = 20 ; i>=0 ; --i){
            ans += 1<<i ;
            if(ans>=maxn||cnt+c[ans]>=k) ans-=1<<i ;
            else cnt +=c[ans] ;
        }
        return ans+1 ;
    }
```

复杂度：自然就比第一种少了一个阶。O(M*logN)。

平衡二叉树：

各种平衡二叉树都可以解决这个问题：Size Balance Tree ，Spaly ，Treap ，红黑树等等。不得不说，Size Balance Tree 解这个问题是比较方便的。因为 SBT 本身就有一个 Select 操作，直接调用一下，就出来了。

代码

复杂度：用的是第二种思维。O(M*logN)。

总结：

其实，综合起来，各种数据结构，各种算法 ，各种纠结。平衡树太复杂，线段树又高了点（一般情况不会有问题的）。最好的方法还是**树状数组的二进制算法**，时间复杂度和编程复杂度达到双赢。

但是，总结起来，发现线段树或者树状数组所消耗的空间跟数据的范围有关，当序列元素是浮点数或者范围很大时，就有点力不从心了（当然，离线的情况可以离散化，在线的某些情况可以离散化），而用平衡二叉树就不存在这样的问题。原来平衡二叉树才是王道。

练习：

最近发现，基于这种思想的题目还真是不少啊~～

[NOI2004]郁闷的出纳员　工资反过来看，职员工资不变，而是工资下界在变而已。当降低工资下界时，为了知道哪些职员走人了，我还用了个二叉堆。。。。

POJ 2761　Feed the dogs　首先该题用接下来**问题 3** 的一个特例。但其特殊性在于任意两个区间不包含，导致把区间按左端点（不会存在相同左端点滴，否则必包含）排序之后，依次扫描每个区间，当前区间和前一个区间相交的部分不动，前一区间有而当前区间没有的部分删除，前一区间没有而当前区间有的部分添加。这能保证每个元素正好添删各一次。

POJ 2823　Sliding Window　太特殊了，最大值就是第 K 小 ，　最小值就是第 1 小（这是一个很重要的启示：**树状数组也可以动态求最值**）。**单调队列**可以弄成线性算法。

HUNNU 10571 Counting Girls　**第四届华中南邀请赛** 但数据与题目稍有不符 但可以确定每个数大于等于 0 且不超过 200000 。关键是求第 X th 到第 Y th 个 MM 的 rating 和要注意下。

KiKi's K-Number 这题就没什么好说的了。求[a+1,MaxnInt]的第 K 小的数，先求出[0,a]有多少个数，设为 cnt，只要求第 K+cnt 小的数就可以了。哦，题目说是求第 K 大的，实际是求第 K 小的，这有点意思~

【问题三】

<u>问题描述</u>

给定一个序列 a[1...n] ， 有 m 个询问 ， 每次询问 a[i...j]之间第 K 小的数。
（引用一句英文：You can assume that n<100001 and m<50001）

<u>算法分析</u>

<u>块状表</u>

如果这道题能够想到用块状表的话，思维复杂度和编程复杂度都不高~，考虑这样操作：
首先预处理，将序列划分成 sqrt(N)个小段，每段长[sqrt(N)] ，划分时，将每小段排好序。
然后就是查询了，对区间[i,j]的查询，同样采用二分求比测试值 mid 小的个数。i 和 j 所在的零散的两小段直接枚举求，中间完整的小段则二分查找求。这样一次查询时间复杂度为

$$O(\log MaxInt * \sqrt{N} * \log \sqrt{N})$$

于是总复杂度为：

$$O(M * \log MaxInt * \sqrt{N} * \log \sqrt{N})$$

当然，这里计算量是比较大的，实际写了个程序也是超时的。但当 M 比较小时，也未尝不是一种好的选择（或者当开阔思路吧，但对<u>问题四</u>却正好打个擦边球）。

<u>划分树</u>

划分树应该是解决这道题复杂度最低的方法，复杂度为

$$O(N * \log N + M * \log N)$$

思想其实很简答，用线段树把整个序列分成若干区间。建树时，对区间[l,r]划分：选择该区间的中位数 value（注意：可以先用快速排序对原来序列排个序，于是可以速度得到中位数），将小于等于 value 的不超过 mid-l+1 个数划分到[l,mid]区间，其余划分到[mid+1,r]区间，用一个数组把每层划分后的序列保存起来。

然后，查找的时候，Find(x , l , r , k)表示超找 x 节点内区间[l,r]第 K 小的数。将该节点区间分成三个区间[seg_left , l-1] ， [l , r] , [r+1 , seg_right]来讨论问题，他们在划分过程中分到[l,mid]区间的个数依次为 ls , ms , rs 。若 ms<=K 自然查左边区间, Find(2*x , l+ls , l+ls+ms-1,K)。否则自然查右边，计算下标很烦啊。有代码在~

代码：

```
void build(lld d ,lld l , lld r ){
        if(l == r)    return    ;
        lld i , mid = (l+r)>>1 , j=l , k=mid+1    ;
        for(i = l ; i <= r ; ++ i){
                s[d][i] = s[d][i-1] ;
                if(tr[d][i] <= mid){
                      s[d][i]++ ;
                      tr[d+1][j++] = tr[d][i];
                }else{
                      tr[d+1][k++] = tr[d][i];
                }
        }
        build(d+1 ,l , mid);
        build(d+1 , mid+1 , r);
}

lld getkth(lld d ,lld lp ,lld rp , lld l , lld r , lld k){
      if(lp == rp )    return tr[d][lp] ;
      lld mid = (lp + rp)>>1 ;
      if(k<=s[d][r]-s[d][l-1])
            return getkth(d+1 ,lp , mid , lp+s[d][l-1]-s[d][lp-1] , lp+s[d][r]-s[d][lp-1]-1 , k );
      else
            return    getkth(d+1    ,mid+1    ,    rp    ,    mid+1+(l-lp)-(s[d][l-1]-s[d][lp-1])    ,
mid+(r-lp+1)-(s[d][r]-s[d][lp-1]) , k-(s[d][r]-s[d][l-1]) );
}
```

归并树

归并树思想就跟简单了，说白了就是线段树每个区间[l,r]内的数都排好序然后保存起来，从两个儿子到父亲节点，其实就是两个有序序列归并成一个有序序列，所以就称归并树了。

用前面说的二分答案，对测试值 mid 求 rank。查找的时候，将查找区间划分成线段树中若干子区间之并，很明显各个子区间小于 mid 的个数加起来，就是该区间小于 mid 的个数。而每个子区间又是有序的，所有二分可以很快找到小区间小于 mid 的个数。

总结起来，有三次二分：

1.二分答案；

2.查找区间[a,b]划分成不超过 log(b - a)个小区间；

3.对每个子区间，二分查找小于 mid 的个数；

于是，整个算法复杂度为：

$$O(N*\log N+M*\log MaxInt*\log^2 N)$$

总结：

对本问题，提供了三种算法，其中基于快排的划分算法是最快的；块状表思维和编程都比较简单，但是复杂度比较高；归并树算法思想很好，二分答案，求 rank ， 后面问题四的算法就是根据这种思维设计出来的。

练习

POJ 2761 Feed the dogs

POJ 2104 K-th Number

NOI 2010 超级刚琴   这题还真有点难度。

思路：划分树+堆+单调队列

对每个区间，起点为 i+1,终点在区间[i+L,i+R]。计 s[i]=a[0]+a[1]+...+a[i] (规定 a[0]=0), 设 opt[i,k]表示第 k 大的 s[j] (i+L<=j<=i+R)，即 opt[i,k] = Max_k { s[j] | i+L<=j<=i+R} （Max_k 表示第 k 大）。

先进行两个预处理工作：

1.将 s[1] , s[2] , s[3] , .. s[n] 建成一颗静态划分树。

2.用单调队列预处理出所有 opt[i,1]的值，并将所有 opt[i,1]-s[i]用一个堆维护起来。当然也可以直接通过查询[i+L,i+R]区间第 1 大的值来预处理 opt[i,1]。

下面开始取值，并更新：依次从堆中取出最大值，设是 opt[i,j]-s[i]，把它累加至答案，再查询划分树中[i+L , i+R]区间第 j+1 大的值 opt[i,j+1] , 将 opt[i,j+1]-s[i]后入堆。直到累加次数为 K 停止。

【问题四】

问题描述

给定一个原始序列 a[1...n] , 有两种操作：

操作一： QUERY   i   j   k   询问当前序列中，a[i...j]之间第 k 小的数是多少

操作二： CHANG   I   T   将 a[i]改为 T ；

输出每次询问的结果。N<=50000 , 操作次数 M<=10000 。

算法分析

块状表

将 a[1...n]分成 sqrt(N)段，每段长[sqrt(N)],为方便二分查找每段，将每段排好序，对操作二，先删去 a[i] , 在插入 T , 维护该块得有序性，复杂度为 O(sqrt(N))。

对操作一，二分答案，设当前测试值为 mid , 先统计两端零散块，O(2*sqrt(N)) 。对中间完整块，每块二分查找，总复杂度为：

$$O(M * \log MaxInt * \sqrt{N} * \log \sqrt{N})$$

线段树+平衡二叉树

序列被线段树划分为区间节点，而每个节点又是一颗平衡二叉树，平衡二叉树放的是该区间段的所有数。

对操作二，依次更新从跟区间到叶子节点区间的平衡树即可（先删除 a[i],在插入 T），考虑复杂度，第一层规模为 N，第二层规模为 N/2，第 k 层规模为 N/2^k 。进行依次操作二的复杂度为：

$$O(\log N+\log\frac{N}{2}+\log\frac{N}{2^2}+\ldots+\log\frac{N}{2^k})=O(\log\frac{N^{k+1}}{2^{\frac{k(k+1)}{2}}})=O(\frac{1}{2}k(k+1))=O(\log^2 N)$$

这里 $k=[\log N]$

对操作一：询问区间被划分为不超过[logN]个线段树节点之并，每次区间查找上界为 logN 。所以，对每个测试值 mid，耗时(logN)^2，故查询一次的复杂度为

$$O(\log^2 N*\log MaxInt)$$

总复杂度为

$$O(M*\log^2 N*\log MaxInt)$$

练习

ZOJ 2112 Dynamic Rankings