预祝大家今天要的愉快

信息学

离线分治

数据结构大致分为2个部分

1 数据存储的结构

2 结构支持的操作

→ 利用数据的结构特点维护额外信息加速操作 → 多个操作被依次完成 **完美AC**

离线与在线算法

如果操作可以打乱顺序求解，再按序输出。 操作可**离线**

如果一个操作必须依次解决，那么我们必须**在线**。?

尝试

在线 -> 离线也可做?

动态问题->静态问题

如果你没看书，你可以看下面的截图

根据“查询”响应时间的不同，可以把解决上述数据结构问题的算法分为“在线”和“离线”两类。一个**在线算法**依次获得每项操作，并能够在每次“查询”时立即回答正确的结果，然后再继续执行下一次操作。一个**离线算法**需要预先知晓整个操作序列，经过一系列计算，最后再批量回答所有“查询”的结果。

根据两种操作在“时间”轴上分布的不同，可以把上述数据结构问题分成“动态”和“静态”两类。只包含“查询”型操作，或一切“查询”型操作都在一切“修改”型操作之后的问题称为**静态问题**，其余问题称为**动态问题**。

离线分治1

基于时间的分治算法-CDQ分治

回顾：求逆序对

逆序对问题，
当i<j时，求序列中有多少对<ai,aj> 满足ai > aj

也是一个**二维偏序问题**(可理解为满足两个维度大小关系)
- i < j
- ai > aj

可以使用**归并排序求逆序对**

回顾：归并排序求逆序对

A还记得怎么求的不？

1 Yes

2

B简单说一下，归并排序2个过程

1 二分区间。直到不能再分后

2 合并区间时统计逆序对（左侧区间对右侧区间产生贡献）

?

C回顾分治过程

左右区间已经完成计算

计算由左右区域**跨区域合并**时产生的**贡献**

当a[L] <b[R]时，ans += len(b_rest)

归并排序，其实就是一个**CDQ分治**　CDQ用于解决**多维偏序问题**

回顾：归并排序求逆序对

A还记得怎么求的不?

1 Yes

2 yes

B简单说一下，归并排序2个过程

1 二分区间。直到不能再分后

2 合并区间时统计逆序对（左侧区间对右侧区间产生贡献）

?

C回顾分治过程

左右区间已经完成计算

计算由左右区域**跨区域合并**时产生的**贡献**

D如果合并时还有**第三维关系?**

数据结构维护（树状数组）

归并排序，其实就是一个**CDQ分治**

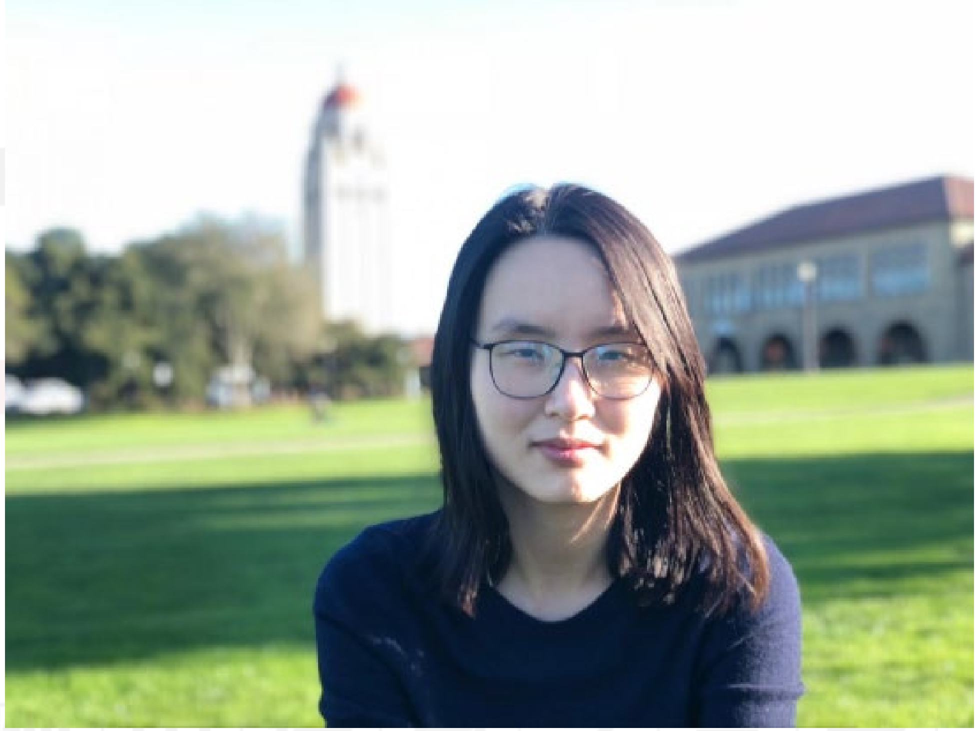
CDQ用于解决**多维偏序问题**

在具体介绍CDQ分治前

介绍一下CDQ

CDQ（陈丹琦）

IOI2008 金牌（左二）



斯坦福大学-计算机博士

目前是普林斯顿大学的助理教授
专注于人工智能领域的
自然语言处理（NLP）

她很强，所以我们今天开始学习**CDQ分治**

可以这样理解，在M个操作中：

每个查询的结果 = 原始数据结果 AND 之前所有修改影响

定义函数 solve(l,r) 为，当$\forall k \in [l,r], (\forall l,r \in [1,M], l < r)$时，若第k项操作为查询，则计算第l～k-1项操作中的修改，对查询结果的影响

方法如下 mid = (l+r)>>1

1. solve(l, mid)
2. solve(mid+1, r)
3. 计算l～ mid 操作中的修改，对mid+1 ~r 操作中的查询的影响

和归并排序的过程类似，递归初始状态为solve(1,M),边界条件为r==l

可以这样理解，在M个操作中：

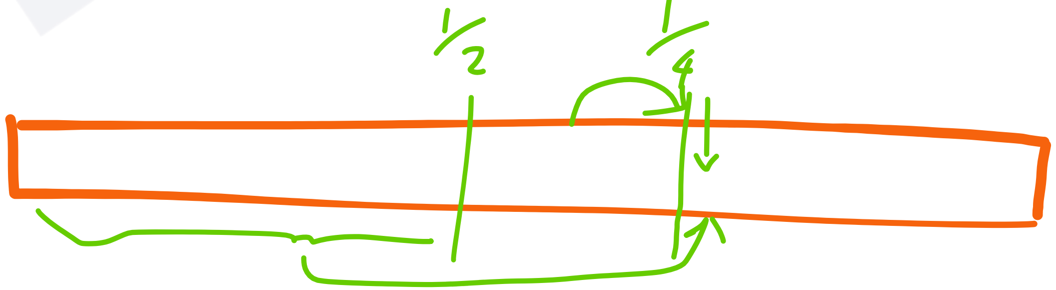
每个查询的结果 = 原始数据结果 AND 之前所有修改影响

单独注意第3个操作：

3. 计算1 ~ mid 操作中的修改，对mid+1 ~r 操作中的查询的影响

这样操作带来的实质性变化：

将M个操作的动态问题，通过分治转化为logM个静态问题



不论前面的操作组合顺序如何其对后的影响结果一致

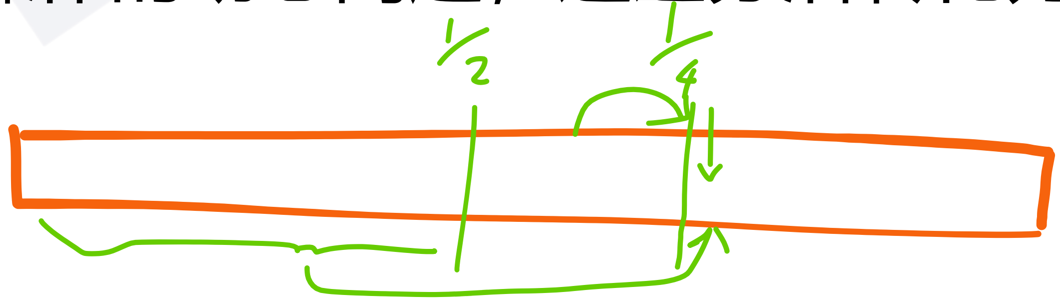
可以这样理解，在M个操作中：

每个查询的结果 = 原始数据结果 AND 之前所有修改影响

单独注意第3个操作：　3. 计算l ~ mid 操作中的修改，对mid+1 ~r 操作中的查询的影响

这样操作带来的实质性变化：

将M个操作的动态问题，通过分治转化为logM个静态问题



将操作序列的时间分治

话回：归并排序

- 归并排序可以处理逆序对
- 树状数组也可以处理逆序对。
- 因为本质上树状数组（值域线段树）也是维护的偏序关系。

数据结构维护偏序关系

优点：动态维护

代价：常数大、代码长、空间多

分治维护偏序关系

正好反过来

启发：不要无脑数据结构

例 三维偏序（陌上花开）

太空中n个星星，每个都有3个坐标xi,yi,zi，表示它们的立体位置
现在问你对于每个星星i，有多少个星星j满足xj<=xi，yj<=yi，zj<=zi
n<=1e5

口胡：借鉴归并排序的思路：
1维sort
2维归并
3维树状数组

sort x后，在快排y的过程中
保证前一段x小于后一段x的同时
用树状数组维护z

例 三维偏序（陌上花开）

sort x后，在快排y的过程中
保证前一段x小于后一段x的同时
用树状数组维护z

求跨越左右的偏序时，我们可以不用管第一维的影响，因为第一维左边肯定小于右边。
然后对第二维做归并排序。
1 每次左边被归并时，保存第三维的数据在树状数组中；
2 右边被归并时，在树状数组中查询小于自己第三维的数据有多少个。
录入答案贡献。

例 三维偏序（陌上花开）

sort x后，在快排y的过程中
保证前一段x小于后一段x的同时
用树状数组维护z

1 在归并合并y的时候，会破坏x的有序性。
所以归并应先从左区间开始归并
即：

```
f():
    f(l,mid)
    f(mid+1,r)
    merge()
```

Ok不?

2 当归并完左区间时
为保证归并正确性
先sort_y(mid+1,r)
再用2个指针p1,p2分别
扫ya(l,mid) yb(mid+1,r)
扫的过程中，用树状数组
统计ya[p1]<=yb[p2]时
区间(p1,mid)中z维小于z[p2]的数据。
完事后，记得复原刚刚sort
sort_x(mid+1,r)

例 三维偏序（陌上花开）code


```
#include<bits/stdc++.h>
#define low_bit(x) x&(-x)
using namespace std;
const int N=100002;
struct Flower{
    int s,c,m,cnt,sum;
}a[N],q[N];
int n,k,tot;
inline int cmp(Flower x,Flower y){
    if(x.s==y.s&&x.c==y.c) return x.m<y.m;
    if(x.s==y.s) return x.c<y.c;
    return x.s<y.s;
}
inline int cmp2(Flower x,Flower y){
    if(x.c==y.c) return x.m<y.m;
    return x.c<y.c;
}
int ans[N],tree[N*4];
inline void add(int x,int c){
    for(;x<=k;x+=low_bit(x)) tree[x]+=c;
}
inline int ask(int x){
    int res=0;
    for(;x;x-=low_bit(x)) res+=tree[x];
    return res;
}
```

```
inline void solve(int l,int r){
    if(l==r) return;
    int mid=(l+r)>>1;
    solve(l,mid);solve(mid+1,r);
    sort(q+l,q+mid+1,cmp2);sort(q+mid+1,q+r+1,cmp2);
    int j=l;
    for(int i=mid+1;i<=r;i++){
        while(j<=mid&&q[j].c<=q[i].c){
            add(q[j].m,q[j].cnt);
            j++;
        }
        q[i].sum+=ask(q[i].m);
    }
    for(int i=l;i<j;i++) add(q[i].m,-q[i].cnt);
}
int main(){
    scanf("%d %d",&n,&k);
    for(int i=1;i<=n;i++) scanf("%d %d %d",&a[i].s,&a[i].c,&a[i].m);
    sort(a+1,a+n+1,cmp);
    q[1]=a[1];
    q[1].cnt=1;
    tot=1;
    for(int i=2;i<=n;i++){
        if(q[tot].s==a[i].s&&q[tot].c==a[i].c&&q[tot].m==a[i].m) {
                                        q[tot].cnt++;
        }else{
            q[++tot]=a[i];
            q[tot].cnt=1;
        }
    }
    solve(1,tot);
    for(int i=1;i<=tot;i++) ans[q[i].sum+q[i].cnt-1]+=q[i].cnt;
    for(int i=0;i<n;i++) printf("%d\n",ans[i]);
    return 0;
}
```

注意
去重

实际上，如何维护第三维关系，是一个很灵活的东西

例如CDQ1 CDQ2大显神威，其实也可以
https://blog.csdn.net/qq_41510496/article/details/82914046

但是一般使用数状数组维护第三维数据

例2：天使玩偶

- 蓝书讲解还行。

- 配合使用更佳：

1. https://blog.nowcoder.net/n/3ac34da57bd14f718c82aa883b098738
2. https://blog.csdn.net/sslz_fsy/article/details/86485208

CDQ分治小结

- 用于维护三维偏序问题
- 常规套路：一维：sort，一维：CDQ分治，一维：BIT

离线分治2

基于空间的分治算法

上回说

- 基于操作时间的分治。CDQ分治

1. 定义操作，solve（1，m）表示求解1到m个操作后所产生的答案。
2. 将操作序列一分为二，分割到l==r后，归并地用前面的修改去影响后面的答案。

- 除了按**时间分治**外，我们还可以**值域分治**。

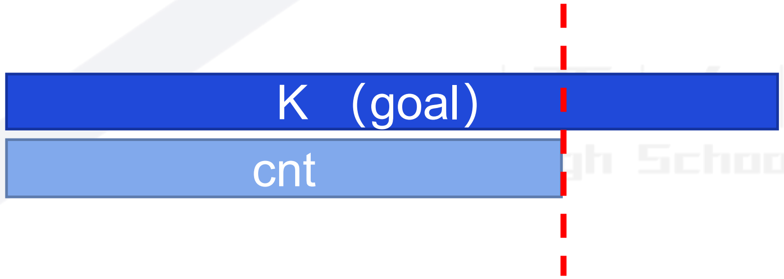
- 信息学竞赛，就是把**已有信息**玩出花的竞赛。

问题提出

- **求N个数中的第K小。（1次询问）**
- 1. sort 复杂度 O(NlogN)
- 2. 归并排一半复杂度 O（N）

问题提出

- 3. 二分答案
  - 根据输入数据范围[l,r]假定mid为答案，测试序列中有多少个数<=mid,记为cnt
    - If k<=cnt，则说明答案属于[l,mid]，继续在左半区域二分。
    - If k>cnt 则说明答案属于[mid+1,r]，继续在右半区域二分，注意，k-=cnt
    - 复杂度为O(NlogS),S为值域的大小



变成求在
区间[mid+1,r]第k-cnt小

问题提出

- 如果询问M次?
- 时间复杂度会达到O (MNlogS)

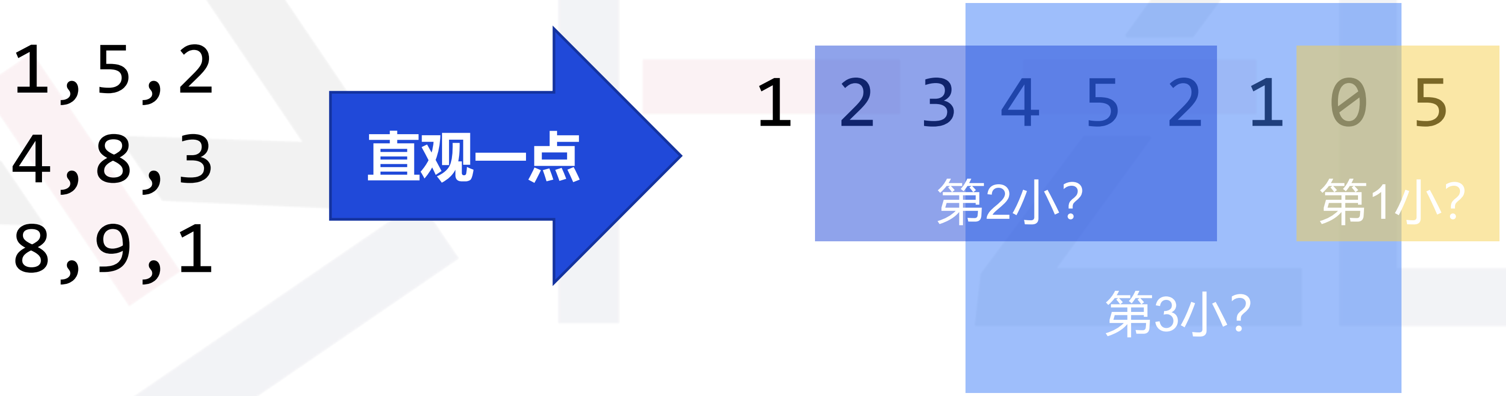
- 我们可以结合CDQ分治的思想

- **在处理值域二分的同时，分治操作序列**
- （即，在值域二分的同时，分治M次询问构成的序列）

怎么做?
分析二分答案过程

例K-th Number

给定一个长度为 N 的整数序列 A，执行 M 次操作，其中第 i 次操作给出三个整数 l_i, r_i, k_i，求 $A[l_i], A[l_i+1], \cdots, A[r_i]$（即 A 的下标区间 $[l_i, r_i]$）中第 k_i 小的数是多少。$N \le 10^5, M \le 10^4, |A[i]| \le 10^9$。



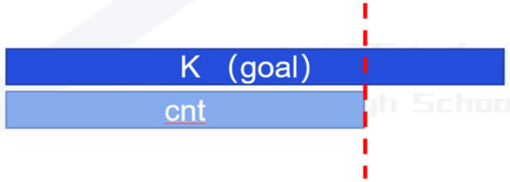
如果还是按照之前的做法，连续做M次二分答案
复杂度达到O(MNlogS)
假定M与N同阶，N方复杂度跑不掉了不能接受

信息的记录与复用

尝试把多个二分答案“打包处理”

回顾二分答案找第k小过程

- 3. 二分答案
  - 根据输入数据范围[l,r]假定mid为答案，测试序列中有多少个数<=mid,记为cnt
    - If k<=cnt，则说明答案属于[l,mid]，继续在左半区域二分。
    - If k>cnt 则说明答案属于[mid+1,r]，继续在右半区域二分，注意，k-=cnt
    - 复杂度为O(NlogS),S为值域的大小



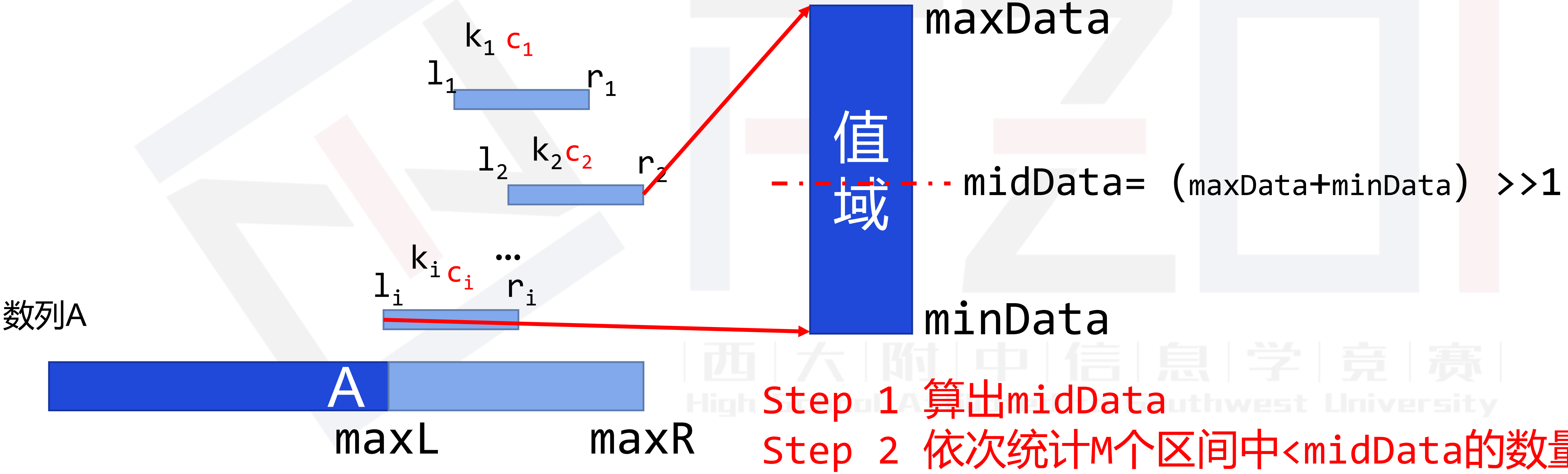
变成求在
区间[mid+1,r]第k-cnt小

对于M个询问区间，我们可以在值域 [minData,maxData]上二分答案

课后看，找到minData 和maxData具体做法：先找到maxL,minR 在其中打擂台求max min

回顾二分答案找第k小过程

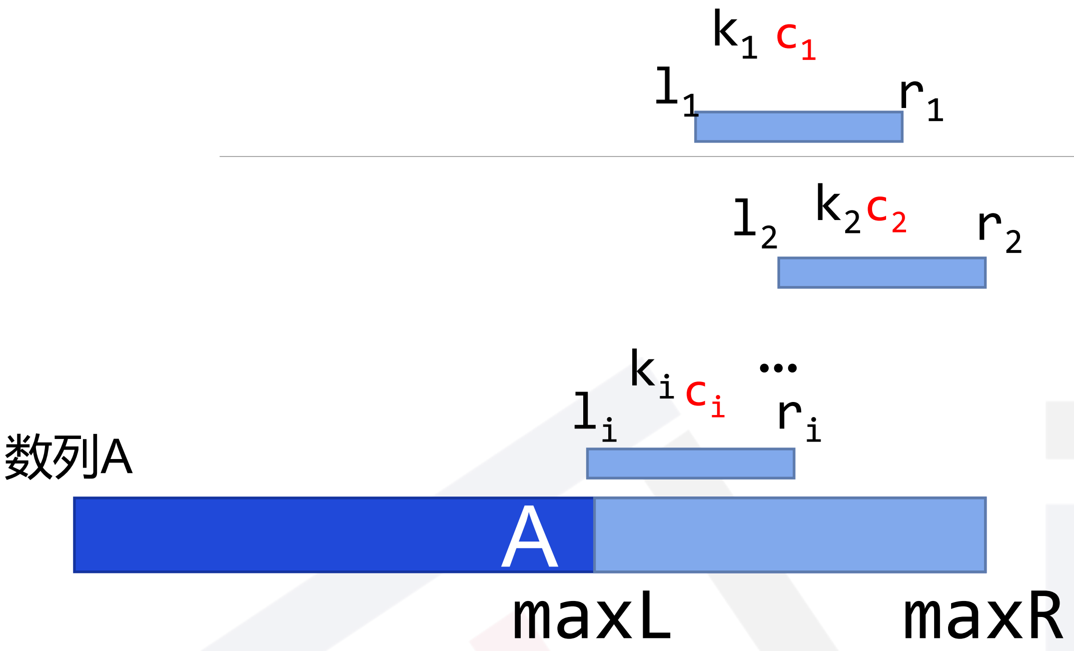
对于M个询问区间，我们可以在值域 [minData,maxData]上二分答案

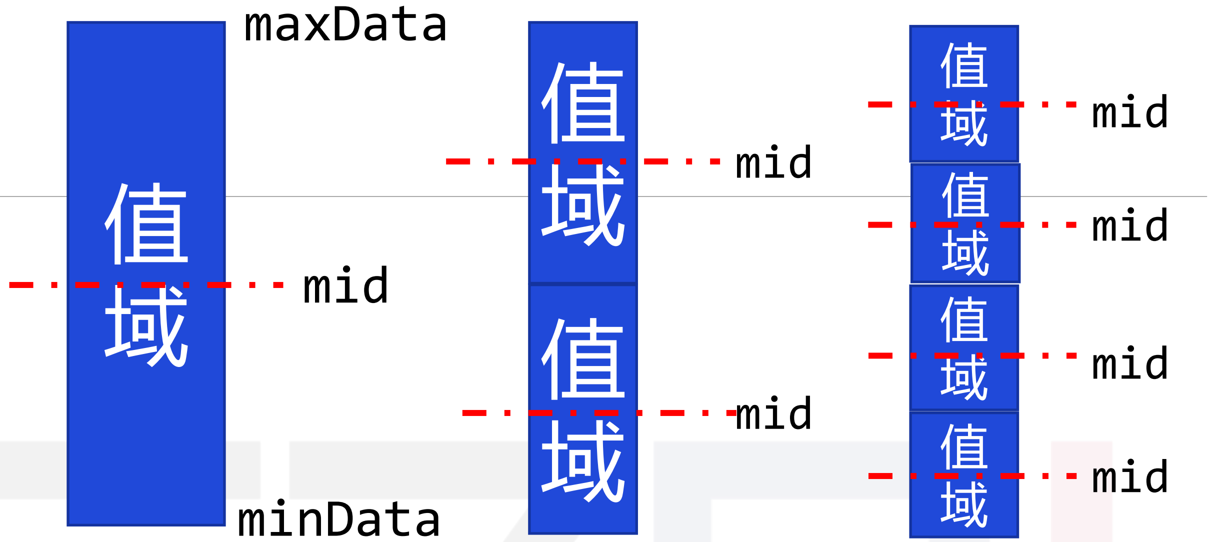


Step 1 算出midData

Step 2 依次统计M个区间中<midData的数量c_i

Step 3 比较k_i与c_i，根据结果，第i个序列向左or右二分





但是对于整个值域而言，这是分治的分！

值域分割

如果 ki<=ci

说明第i个询问的答案在值域[minData,mid]中

如果 ki>ci

说明第i个询问的答案在值域[mid+1,maxData]中

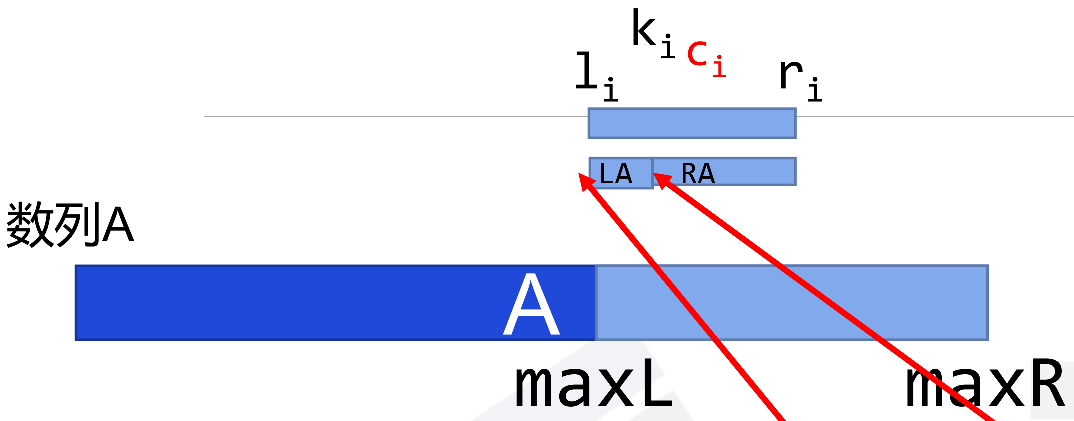
同时，记得ki=ki-ci

更新区域

重算mid

统计ci

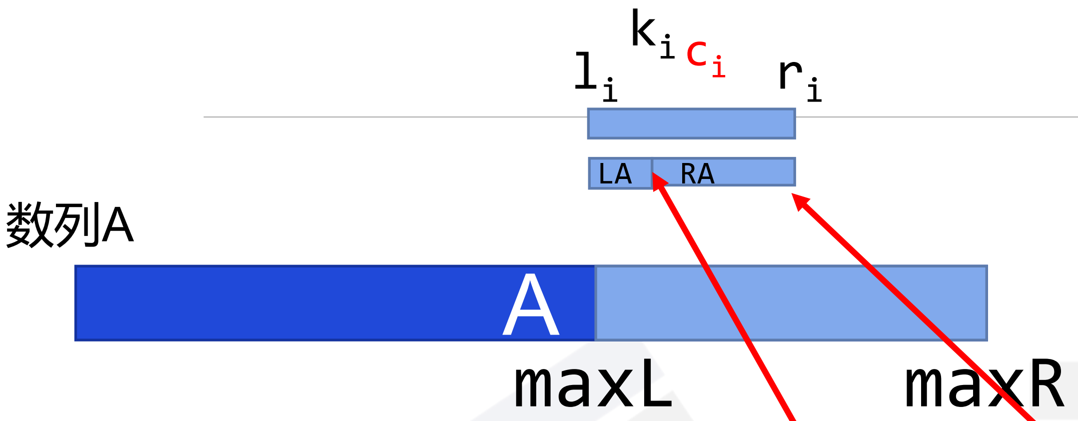
根据mid值，将区间中$[l_i,r_i]$ 划分为<mid 的LA序列，>mid的RA序列



[minData,mid]
在区间[li,ri]上的第k小
与
LA序列上的第k小等价
且LA是[li,ri]的子问题

如果 ki<=ci
说明第i个询问的答案在值域[minData,mid]中
如果 ki>ci
说明第i个询问的答案在值域[mid+1,maxData]中
同时，记得ki=ki-ci

根据mid值，将区间中$[l_i,r_i]$ 划分为<mid 的LA序列，>mid的RA序列



这也是一个子问题
且 ki=ki-ci后
子问题独立

如果 ki<=ci

说明第i个询问的答案在值域[minData,mid]中

如果 ki>ci

说明第i个询问的答案在值域[mid+1,maxData]中

同时，记得ki=ki-ci

这样，我们拥有了值域分治的全部理论基础与过程

我们就可以在分治过程中，逐片求解区域中的区域

综上所述，我们得到一个分治算法。设 solve(L,R,a,q) 表示有一个值域为 $[L,R]$ 的整数序列 a，对询问序列 q（q 中存储若干个三元组 l_i,r_i,k_i，表示求 a 的下标区间 $[l_i,r_i]$ 中的第 k_i 小的数）当中的每个询问作出回答。

1. 设 $mid=(L+R)\gg 1$。

2. 利用树状数组，对于 q 中的每个询问，统计在序列 a 的下标区间 $[l_i,r_i]$ 中不大于 mid 的数有多少个，记为 c_i。

3. 若 $k_i \le c_i$，则把该询问加入到序列 lq 中。否则，令 $k_i -= c_i$，将其加入到序列 rq 中。

这样，我们拥有了值域分治的全部理论基础与过程

4. 令序列 a 中 $\le mid$ 的数构成序列 la，$> mid$ 的数构成序列 ra。
5. 递归求解 $\text{solve}(L, mid, la, lq)$ 和 $\text{solve}(mid+1, R, ra, rq)$。

递归的边界为：当 q 为空时，直接返回。当 $L = R$ 时，直接把 L 作为 q 中每个询问的答案。读者可以利用样例数据在纸上模拟，理解上述过程。

与 CDQ 分治一样，还应该在 solve 的第 5 步之前撤销对树状数组的所有修改，不能每次建立一个新的树状数组。另外，在程序实现中，为了避免空间浪费，可以直接把 la, ra, lq, rq 拷贝回原来的数组 a, q，在 solve 函数的参数中用两个下标表示有效范围即可。还可以把初始序列 A 中的 N 个整数看作 N 次“赋值”操作，合并 a 和 q 两个序列，进一步简化代码。因为分治至多进行 $\log SIZE$ 层（$SIZE$ 为值域的大小），当操作序列 q 为空时会立即返回，再算上树状数组的开销，所以整个算法的时间复杂度为 $O((N+M)\log SIZE \log N)$。如果加上离散化，可以做到 $O((N+M)\log^2 N)$。

蓝书P250页
祝大家耍的愉快