“算法竞赛导论”学期报告

竞赛中的CDQ分治算法

金杰

2016-1-11

1. 概述

CDQ 分治一般被认为是处理时序的利器。它与莫队算法相同，一出世便大杀四方，无数曾经的难题都被暴力算法套上CDQ而轻松秒杀，颇有「万物皆可CDQ」的气势。更出人意料的是，CDQ 分治本身十分简单，甚至不能被称为一种新算法，只是启发了用分治思想来解决某些问题。

多数关于 CDQ 分治的资料皆从时序入手。为了展示其一般性，本文将从更一般的角度入手，再向特例化的几类问题收拢，以避免只有在看到时序才想到 CDQ 的情况的出现。

本文将用简单的逆序对问题来介绍 CDQ 分治，再从二维偏序问题的分治解法，推广到三维偏序，再证明三维偏序模型与一类问题等价，再介绍 CDQ 分治的更多应用。

2. 从归并排序讲起

众所周知，求逆序对问题可以用归并排序处理，只需在合并的同时增加统计，其代码如下：

```
Sort(int left,int right){
    if (left==right) return;
    mid=left+right>>1;
    Sort(left,mid);
    Sort(mid+1,right);
    MergeAndCount(left,mid,right);
}
```

事实上 CDQ 分治并不陌生，以上就是一个例子。CDQ 分治思想的代码如下：

```
CDQ(int left,int right){
    if (left==right) return;
    mid=left+right>>1;
    CDQ(left,mid);
```

```
    work(left,mid,right);
    CDQ(mid+1,right);
}
```

注意 work()函数可以依题目需要出现在 CDQ()中的任何位置。

将一段区间切成前后两段，用前半段的信息更新后半段，然后递归处理前后两段内部的子问题。可见 CDQ 分治就是普通的分治算法，只是发现可以被用来解决更多问题。

3. 二维偏序

问题描述：

平面上有 N 个点，给出每个点坐标(x,y)，问对每个点 i，有多少点在其左下方(即 xj<=xi&&yj<=yi)。数据范围 N<=100000。

解题思路：

一般解法为先按 y 坐标排序，再按顺序依次将 x 坐标插入一个树状数组，插入时统计比当前点 x 坐标小的数有多少个。效率 O(N*logN)。

代码：

```
struct point{
    int x,y,p;
}a[120000];
int n,ans[120000],b[120000];
bool cmp(point a,point b){
    return a.y<b.y || (a.y==b.y && a.x<b.x);
}
void add(int x){
    for(;x<=n;x+=x&(-x)) b[x]++;
}
int ask(int x){
    int ret=0;
```

```
	for(;x;x-=x&(-x)) ret+=b[x];
	return ret;
}
void work(){
	sort(a+1,a+n+1,cmp);
	for(int i=1;i<=n;i++){
		ans[a[i].p]=ask(a[i].x);
		add(a[i].x);
	}
}
int main(){
	cin>>n;
	for(int i=1;i<=n;i++) {
	   cin>>a[i].x>>a[i].y;
	   a[i].p=i;
	}
	work();
	for(int i=1;i<=n;i++) cout<<ans[i]<<endl;
}
```

分治解法：

其实，二维偏序也可以用分治思想求解。先按 y 坐标排序，问题就等价为以 x 为关键字求次序在前面的数有多少比当前的小，即正序对数，用第 2 节的归并排序求解即可。

4. 三维偏序

问题描述：

空间中有 N 个点，给出每个点坐标(x,y,z)，问对每个点，有多少点其 xyz 坐标皆小于等于该点。保证没有两个点的 x 或 y 或 z 相同。数据范围 N<=100000。

解题思路：

现在又多了一维，怎么办呢？树套树？持久化线段树？都好麻烦。我们已经知道分治可以处理一维信息，只要先用分治算法把三维降为二维，就可以用第 3 节讲过的方法求解了。

首先按 z 坐标排序。定义 CDQ(l,r)为 l..r 范围内的该问题，我们的目标是处理 CDQ(1,n)。根据第 2 节的代码，我们只需要统计出 l..mid 对 mid+1..r 的影响，l..mid 和 mid+1..r 内部的统计可以递归处理。将 l..mid 看做点，mid+1..r 看做询问，由于前面的 z 坐标值一定比后面的小，我们只需考虑 xy 小的有多少点。每次 work() 暴力重新做，先按 y 坐标重新排序，再依次处理，如果是前面的点，就插入树状数组，如果是后面的询问点，就查询树状数组中比 x 值小的有多少个。

例如对最后一个第 N 个点来说，它一共做了 logN 次 work()操作，分别统计了 1~n/2 中有多少 xy 小于第 n 点的,n/2~3n/4 中的,3n/4~7n/8 中的,...

分治过程一共有 logN 层，每一层都有 logN 的排序和树状数组，但是是加关系，所以总时间复杂度为 O(Nlog^2N)。

代码：

```
struct point{
	int x,y,z,p;
}a[120000],b[120000];
int n,nn,ans[120000],c[120000];
//a 为点集，b 为 CDQ 后从 a 中取出的复制，ans 为答案，c 为树状数组

bool cmpY(point a,point b){
	return a.y<b.y;
}
bool cmpZ(point a,point b){
	return a.z<b.z;
}
//树状数组操作
void add(int x){
```

```
	for(;x<=nn;x+=x&(-x)) c[x]++;
}
int ask(int x){
	int ret=0;
	for(;x;x-=x&(-x)) ret+=c[x];
	return ret;
}
//每次重新用树状数组统计答案
void work(int l,int mid,int r){
	nn=r-l+1;
	int midZ=a[mid].z;
	for(int i=1;i<=nn;i++) b[i]=a[l+i-1],c[i]=0;
	sort(b+1,b+nn+1,cmpY);
	for(int i=1;i<=nn;i++){
		if (b[i].z>midZ) ans[b[i].p]+=ask(b[i].x);
		else add(b[i].x);
	}
}
//CDQ 分治，多数情况下只需修改 work 函数
void CDQ(int l,int r){
	if(l==r) return;
	int mid=l+r>>1;
	CDQ(l,mid); CDQ(mid+1,r);
	work(l,mid,r);
}

int main(){
	cin>>n;
	for(int i=1;i<=n;i++) {
	     cin>>a[i].x>>a[i].y>>a[i].z;
```

```
        a[i].p=i;
    }
    sort(a+1,a+n+1,cmpZ);
    CDQ(1,n);
    for(int i=1;i<=n;i++) cout<<ans[i]<<endl;
}
```

补充：

如果是四维偏序怎么办呢？外面再套一层 CDQ 分治呗！复杂度 O(Nlog^3N)。

5. 动态逆序对

由第 4 节，我们知道 CDQ 分治可以处理 k 维偏序问题，其中以三维偏序问题的变种最为常见。

问题描述：

给 1 到 n 的一个排列，按照某种顺序依次删除 m 个元素，你的任务是在每次删除一个元素之前统计整个序列的逆序对数。

输入：第一行包含两个整数 n 和 m，即初始元素的个数和删除的元素个数。以下 n 行每行包含一个 1 到 n 之间的正整数，即初始排列。以下 m 行每行一个正整数，依次为每次删除的元素。

输出：包含 m 行，依次为删除每个元素之前，逆序对的个数。

解题思路：

倒过来考虑，原题变成从空序列开始每加一个点统计当前逆序对数。我们发现该问题可以转化为三维偏序问题。每个点被表示成(x,y,z)，x 为原序列中的位置，y 为数值，z 为加入时间。每对逆序对由后加入的人来统计。则每个点(xi,yi,zi)贡献的逆序对数为满足(xj<xi&&yj>yi&&zj<zi)或(xj>xi&&yj<yi&&zj<zi)的点的数量，分别统计。虽然有大于小于号，只要对坐标做一次翻转就能变成三维偏序问题。

代码：

```
struct point{
	int x,y,z,p;
}a[120000],b[120000],ha[120000];
int n,m,nn,k,ans[120000],c[120000],fy[120000],aans[120000];
//a 为点集，b 为 CDQ 后从 a 中取出的复制，ans 为答案，c 为树状数组

bool cmpY(point a,point b){
	return a.y<b.y;
}
bool cmpZ(point a,point b){
	return a.z<b.z;
}
//树状数组操作
void add(int x){
	for(;x<=nn;x+=x&(-x)) c[x]++;
}
int ask(int x){
	int ret=0;
	for(;x;x-=x&(-x)) ret+=c[x];
	return ret;
}
void Hash(int nn){
	rep(i,1,nn) ha[i].x=i,ha[i].y=b[i].x;
	sort(ha+1,ha+nn+1,cmpY);
	rep(i,1,nn) b[ha[i].x].x=i;
}
//每次重新用树状数组统计答案
void work(int l,int mid,int r){
	nn=r-l+1;
```

```
    int midZ=a[mid].z;
    for(int i=1;i<=nn;i++) b[i]=a[l+i-1],c[i]=0;
    sort(b+1,b+nn+1,cmpY);
    Hash(nn); //必须把 b 的 x 坐标离散化，否则时间复杂度无法保证
    for(int i=1;i<=nn;i++){
        if (b[i].z>midZ) ans[b[i].p]+=ask(b[i].x);
        else add(b[i].x);
    }
}
//CDQ 分治，多数情况下只需修改 work 函数
void CDQ(int l,int r){
    if(l==r) return;
    int mid=(l+r)>>1;
    CDQ(l,mid); CDQ(mid+1,r);
    work(l,mid,r);
}

int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        a[i].x=i;
        cin>>a[i].y;
        fy[a[i].y]=i;
        a[i].p=i;
    }
    for(int i=1;i<=m;i++){
        cin>>k;
        a[fy[k]].z=n-i+1;
    }
    k=1;
```

```
    for(int i=1;i<=n;i++){
        if (a[i].z==0) a[i].z=k++;
    }
    //以上为将读入数据处理为三维点集
    sort(a+1,a+n+1,cmpZ);
    for(int i=1;i<=n;i++) a[i].y=-a[i].y;
    CDQ(1,n); //处理左上方
    for(int i=1;i<=n;i++) a[i].y=-a[i].y,a[i].x=n+1-a[i].x;
    CDQ(1,n); //处理右下方
    for(int i=1;i<=n;i++) aans[i]=aans[i-1]+ans[a[i].p];
    for(int i=1;i<=m;i++) cout<<aans[n-i+1]<<endl;
}
```

补充：

与此类似的题还有很多，也未必是三维偏序，只要注意增删改查时间、序号次序等都可以看成是一维信息，而这一维信息都可以用 CDQ 分治来维护即可，去掉这一维信息后就可以爱咋咋地。比如有些在线的题，把它变成时间序然后用 CDQ 去做，就可以用一个 log 的时间强行离线了。还有要维护 DP 决策凸线的，因为有时序关系，也可以把时序交给 CDQ 来做。

6. 整体二分

整体二分是将多个询问并行分治，通过 log 次共同查询分别确定区间，与 CDQ 有异曲同工之妙，代码也几乎完全一致。

问题描述：

BIU 有 N 个成员国。现在它发现了一颗新的星球，这颗星球的轨道被分为 M 份（第 M 份和第 1 份相邻），第 i 份上有第 Ai 个国家的太空站。

这个星球经常会下陨石雨。BIU 已经预测了接下来 K 场陨石雨的情况。BIU 的第 i 个成员国希望能够收集 Pi 单位的陨石样本。你的任务是判断对于每个国家，它

需要在第几次陨石雨之后，才能收集足够的陨石。

输入： 第一行是两个数 N,M。 第二行有 M 个数，第 i 个数 Oi 表示第 i 段轨道上有第 Oi 个国家的太空站。 第三行有 N 个数，第 i 个数 Pi 表示第 i 个国家希望收集的陨石数量。 第四行有一个数 K，表示 BIU 预测了接下来的 K 场陨石雨。接下来 K 行，每行有三个数 Li,Ri,Ai，表示第 K 场陨石雨的发生地点在从 Li 顺时针到 Ri 的区间中（如果 Li<=Ri，就是 Li,Li+1,…,Ri，否则就是 Ri,Ri+1,…,m-1,m,1,…,Li），向区间中的每个太空站提供 Ai 单位的陨石样本。

输出：N 行。第 i 行的数 Wi 表示第 i 个国家在第 Wi 波陨石雨之后能够收集到足够的陨石样本。如果到第 K 波结束后仍然收集不到，输出 NIE。

数据范围： 1<=n,m,k<=3*10^5 1<=Pi<=10^9 1<=Ai<10^9

解题思路：

首先，下 K 次流星雨到 m 个空间站上这件事可以用线段树维护，每个国家有哪些空间站用 vector 保存。

如果只有一个国家，当然可以直接模拟，也可以二分答案后从 1 模拟到 mid 判断是否有足够陨石，虽然代价稍大，不过我们马上将介绍 n 个国家一起二分的方法。

S 为一个国家集合，用 CDQ(l,r,S)表示 S 中的所有国家的答案都落在 l~r 范围内，初始为 CDQ(1,K,{1..N})。mid=(l+r)/2，我们先模拟下了 mid 场雨后线段树的状况，然后统计 S 中每个国家在下了 mid 场雨后有多少陨石（枚举 vector 中所有空间站在线段树中查询再求和），判断是否达标，达标的放到前面做新集合 S1，未达标新集合 S2，然后递归 CDQ(l,mid,S1),CDQ(mid+1,r,S2)，log 层后每个国家都被分到了只有一个值的区间，就是答案。如此便用分治思想解决了该题，只留下了一个与分治无关的小问题。

考虑所有 mid 的总和，一共 K 个 mid，mid 的平均数是 K/2，下雨的总效率为 O(K^2logM)，这是不能接受的。我们只需保存当前线段树已经下了 1~l-1 场雨后的状态，然后用+[l,r]模拟下了 l~r 场雨，-[l,r]模拟撤销。CDQ 函数这样写：

```
CDQ(l,r,S){
    +[l,mid]
```

判断，分割 S

-[l,mid]

CDQ(l,mid,S1)

CDQ(mid+1,r,S2)

}

这样每次就可以不[1,mid]这么长，只模拟当前区间的一半就行，效率变为 O(KlogKlogM)，加上统计效率 O(MlogKlogM)，至此解决该问题。

代码：

```
int n,m,x,K,T;
int l[300005],r[300005],val[300005];
int p[300005],id[300005],tmp[300005],ans[300005];
vector<int> a[300005];
bool mark[300005];
ll t[300005];
void add(int x,int val){
    for(;x<=m;x+=x&(-x)) t[x]+=val;
}
ll ask(int x){
    ll ret=0;
    for(;x;x-=x&(-x))ret+=t[x];
    return ret;
}
//下雨，+-[l,r]
void opera(int k,int f){
    if(l[k]<=r[k]){
        add(l[k],f*val[k]);
        add(r[k]+1,f*(-val[k]));
    }
    else{
```

```
            add(1,f*val[k]);
            add(r[k]+1,f*(-val[k]));
            add(l[k],f*val[k]);
        }
}
void CDQ(int l,int r,int L,int R){
        if(l>r) return;
        if(L==R){
            for(int i=l;i<=r;i++) ans[id[i]]=L;
            return;
        }
        int mid=(L+R)>>1;
        while(T<=mid) opera(++T,1);
        while(T>mid) opera(T--,-1);
        int cnt=0,now;ll tot;
        for(int i=l;i<=r;i++){
            tot=0;now=id[i];
            for(int j=0;j<a[now].size();j++){
                tot+=ask(a[now][j]);
                if(tot>=p[now]) break;
            }
            if(tot>=p[now]) mark[now]=1,cnt++;
            else mark[now]=0;
        }
        int l1=l,l2=l+cnt;
        for(int i=l;i<=r;i++)
            if(mark[id[i]])tmp[l1++]=id[i];
            else tmp[l2++]=id[i];
        for(int i=l;i<=r;i++)id[i]=tmp[i];
        CDQ(l,l1-1,L,mid);
```

```
    CDQ(l1,l2-1,mid+1,R);
}
int main(){
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        cin>>x;
        a[x].push_back(i);
    }
    for(int i=1;i<=n;i++) cin>>p[i];
    cin>>K;
    for(int i=1;i<=K;i++) cin>>l[i]>>r[i]>>val[i];
    l[++K]=1;r[K]=m;val[K]=inf;
    for(int i=1;i<=n;i++) id[i]=i;
    CDQ(1,n,1,K);
    for(int i=1;i<=n;i++)
        if(ans[i]!=K) cout<<ans[i]<<endl;
        else cout<<"NIE"<<endl;
}
```

补充：

也可以使用并行分治求解。用线段树维护下雨，对于每一个国家来说，经过 logK 次二分之后一定可以出解。那么我们可以维护每个国家的可能答案区间（一开始是[1,K]），然后做 logK 次二分操作。每次我们先求出每个国家的区间中点 midX，然后给 midX 排序，之后用线段树顺序模拟 K 次流星雨，到第 midX 时统计答案。如果到 midX 时达到了 Pi，那么区间改成[l,midX]，否则[mid+1,r]。每次模拟下雨的效率为 KlogM，统计效率 M，再加上排序效率 NlogN，一共 logK 次，所以总效率为 logK*(KlogM+NlogN)，与整体二分的效率相同。

7. 总结

CDQ 分治可以离线处理一维信息，等于直接把问题扔掉一维，使问题变得更简单。

这一维可以是坐标轴上的一轴，也可以是时间序，也可以是序号次序。

因此，许多树套树和持久化数据结构的问题，都可以用 CDQ 来轻松代替。在线要求增删改查的题，也可以强行离线。维护 DP 决策凸线也有时间序，同样可以处理。

整体二分同样利用了分治思想，可以看做是 CDQ 分治的一种。