

关于多维偏序集系列问题的研究

徐润麒

[摘要]

使用 **CDQ 分治**,二分思想,分块思想解决多维偏序集问题的方案研究与对比.

参考资料:

- 1."catch the penguins"(4 维偏序问题) 杭州学军中学 张闻涛
- 2."浅谈分块思想在一类数据处理问题中的应用"(分块思想)北京市第 18 中学 罗剑桥
- 3.《从<Cash>谈一类分治算法的应用》(cdq 分治)陈丹琦<-CDQ

[引入]

“设 R 是集合 A 上的一个关系,如果 R 是自反的、反对称的和可传递的,则称 R 是集合 A 的偏序关系,简称偏序.对于 $(a, b) \in R$,就把它表示成 $a \leq b$ 。若在集合 A 上给定一个偏序关系,则称集合 A 按偏序关系 \leq 构成一个偏序集合,集合 A 和偏序 R 一起称为偏序集。”

----摘自深奥的百度百科

偏序集(Partially order set)定义:

集合 $A\{x_1, x_2, x_3 \dots x_n\}, B\{y_1, y_2, y_3 \dots y_n\}$

满足 $x_1 < y_1$ 且 $x_2 < y_2$ 且 $x_3 < y_3 \dots$ 且 $x_n < y_n$

注:也有可能是 $x_1 \leq y_1$ 且 $x_2 \leq y_2$... 根据实际情况而定

偏序集性质: (A, B, C 为集合)

1. 传递性 A 于 B 偏序(记作 $A \leq B$), $B \leq C$, 则 $A \leq C$

2. 反对称性 $A \leq B, B \leq A$, 则 $A = B$

3. 自反性 $A \leq A$

[Dilworth(反链)定理]

令 A 是一个有限偏序集, 并令 m 是反链的最大的大小。则 A 可以被划分成 m 个但不能再少的链。

证明(摘自网络)

设 p 为最少反链个数

1. 先证明 A 不能划分成小于 r 个反链。由于 r 是最大链 C 的大小, C 中任两个元素都可比, 因此 C 中任两个元素都不能属于同一反链。所以 $p \geq r$ 。

2. 设 $B_1 = A$, A_1 是 A 中的极小元的集合。从 B_1 中

删除 A_1 得到 B_2 。注意到对于 B_2 中任意元素 a_2 ，必存在 B_1 中的元素 a_1 ，使得 $a_1 \leq a_2$ 。令 A_2 是 B_2 中极小元的集合，从 B_2 中删除 A_2 得到 B_3最终，会有一个 B_k 非空而 $B_{(k+1)}$ 为空。于是 A_1, A_2, \dots, A_k 就是 A 的反链的划分，同时存在链 $a_1 \leq a_2 \leq \dots \leq a_k$ ，其中 a_i 在 A_i 内。由于 r 是最长链大小，因此 $r \geq k$ 。由于 A 被划分成了 k 个反链，因此 $r \geq k \geq p$ 。因此 $r = p$ ，定理得证。

[问题]

- 1 类问题:最优化问题-满足偏序关系意义下的最长子序列
- 2 类问题:计数问题-求与某元素构成偏序关系的元素数量

[解决]

通用题面(摘自“俄罗斯套娃”系列问题)

描述

来自 k 维世界的你有 n 个独立的俄罗斯套娃，编号 1 到 n 。因为处于 k 维世界，所以套娃 i 号有 k 个描述大小的属性 $a[i][1..k]$ 。如果你想将一个套娃 i 套在另一个套娃 j 外面，外部套娃 i 的全部大小属性必须全部大于 j 的，即 $a[i][1] > a[j][1]$ 并且 $a[i][2] > a[j][2]$并

且 $a[i][k] > a[j][k]$ 。现在你想知道最多可以将多少个套娃套在一起。

输入

n k

$n \times k$ 二维数组

输出

最多可以将多少个套娃套在一起

数据范围

$n \leq 200000, k \leq 100$

为什么 $k \leq 100$ 这个尴尬的数字?

因为 k 太大了输入太慢,数组开不下.....

暴力代码(n 维度,可用于自测速度差异)

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 long long n,k,a[10005][50],vst[10005],ans,flag,fflag;
4 void dfs(long long x,long long pre){
5     for(int i=1;i<=n;i++,fflag=0){
6         if(vst[i]==0){ flag=0;
7             for(int j=1;j<=k;j++) if(a[i][j]<=a[pre][j]) {flag=1; break; }
8             if(flag==0) vst[i]=1,dfs(x+1,i),vst[i]=0,fflag=1;
9         }
10    }
```

```

11  if(fflag==0){ans=max(ans,x); return;}
12 }
13 int main(){
14     cin >> n >> k;//k:维度数量
15     for(int i=1;i<=n;i++)
16         for(int j=1;j<=k;j++)
17             cin >> a[i][j];
18     dfs(1,0),cout << ans-1;
19     return 0;
20 }

```

很有意思的 DFS 终止条件~
复杂度 $O((Nk)^k)$

N,K	时间(s)	空间(MB)
100,3	0.97	10.2
1000,3	10.2	77.6
10000,5	692.7	298.9
200000,100	-INF(stack overflow)-	-INF-



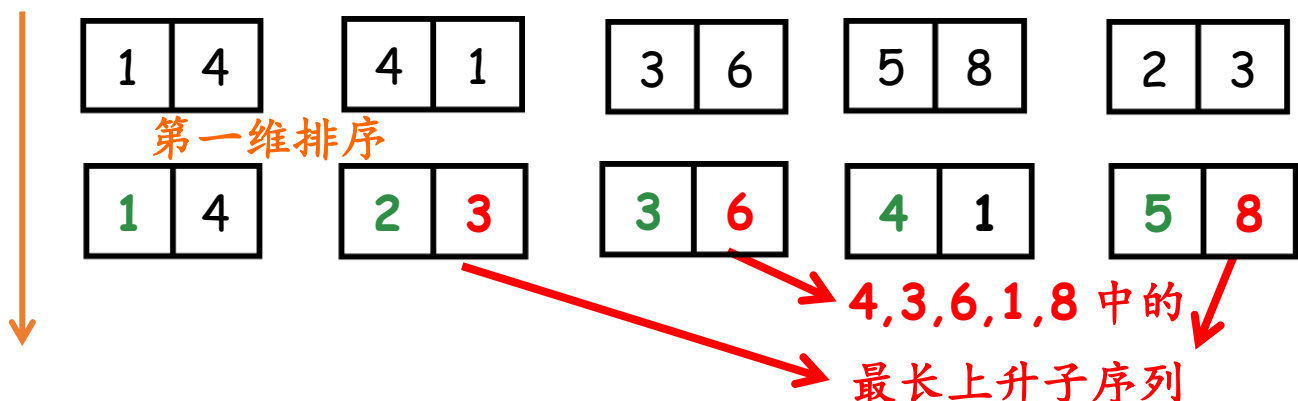
1 维偏序集: (So easy~)(时间复杂度 $O(n), O(n\log n)$, 空间 $O(n)$)

1 类问题: 排序后输出满足 $a[i] > a[i-1]$ 的个数

2 类问题: 排序后输出排在询问元素之前的元素数量(为了剔除相等的元素)

2 维偏序集: ($O(n\log n), O(n\log n)$)

1 类问题: 最长上升子序列的一个维度一定是递增的, 所以先按照一个维度排序.



在排完序之后使用一维的最长上升子序列算法(Dilworth 定理->贪心+二分查找,在另一维中求解.

时间复杂度: $O(n\log n)$

代码:


```
1 #include<bits/stdc++.h>
2 using namespace std;
3 long long n,ans=1,dp[200005],aa[200005],k;
4 struct info{long long a,b;}x[200005];
5 bool cmp(info x,info y){
6     if(x.a==y.a) return x.b>y.b;
7     return x.a<y.a;
8 }
9 int main(){
10     cin >> n;
11     for(int i=1;i<=n;i++) cin >> x[i].a;
12     for(int i=1;i<=n;i++) cin >> x[i].b;
13     sort(x+1,x+n+1,cmp);
14     for(int i=1;i<=n;i++) aa[i]=x[i].b;
```

```

15  dp[1]=x[1].b;
16  for(int i=2;i<=n;i++) {
17      if(dp[ans]<aa[i]) k=++ans;
18      else k=lower_bound(dp,dp+ans,aa[i])-dp;
19      dp[k]=aa[i];
20  }
21  cout << ans;
22  return 0;
23 }

```

一维上升子序列标准解



2 类问题:需统计偏序集的数量,可以维护一个前缀和数组,运用二维 BIT 作为数据结构可提升时间效率并可解决动态加点的该类问题(**这是解决下文多维偏序集的基础**)

代码:

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  long long n,a[200066],b[200066],bit[10066][566];
4  inline long long LSB(long long x){return x&(-x);}
5  void add(long long x,long long y){
6      for(int i=x;i<=H;i+=LSB(i))
7          for(int j=y;j<=W;j+=LSB(j))
8              bit[i][j]++;

```



```

9 }
10 ll query(long long x,long long y){
11     ll sum=0;
12     for(int i=x;i-=LSB(i))
13         for(int j=y;j-=LSB(j))
14             sum+=bit[i][j];
15     return sum;
16 }
17 int main(){
18     cin >> n;
19     for(int i=1;i<=n;i++)cin >> a[i] >> b[i],a[i]++,b[i]++,add(a[i],b[i]);
20     for(int i=1;i<=n;i++) cout << query(a[i]-1,b[i]-1)-1 << " ";
21     return 0;
22 }

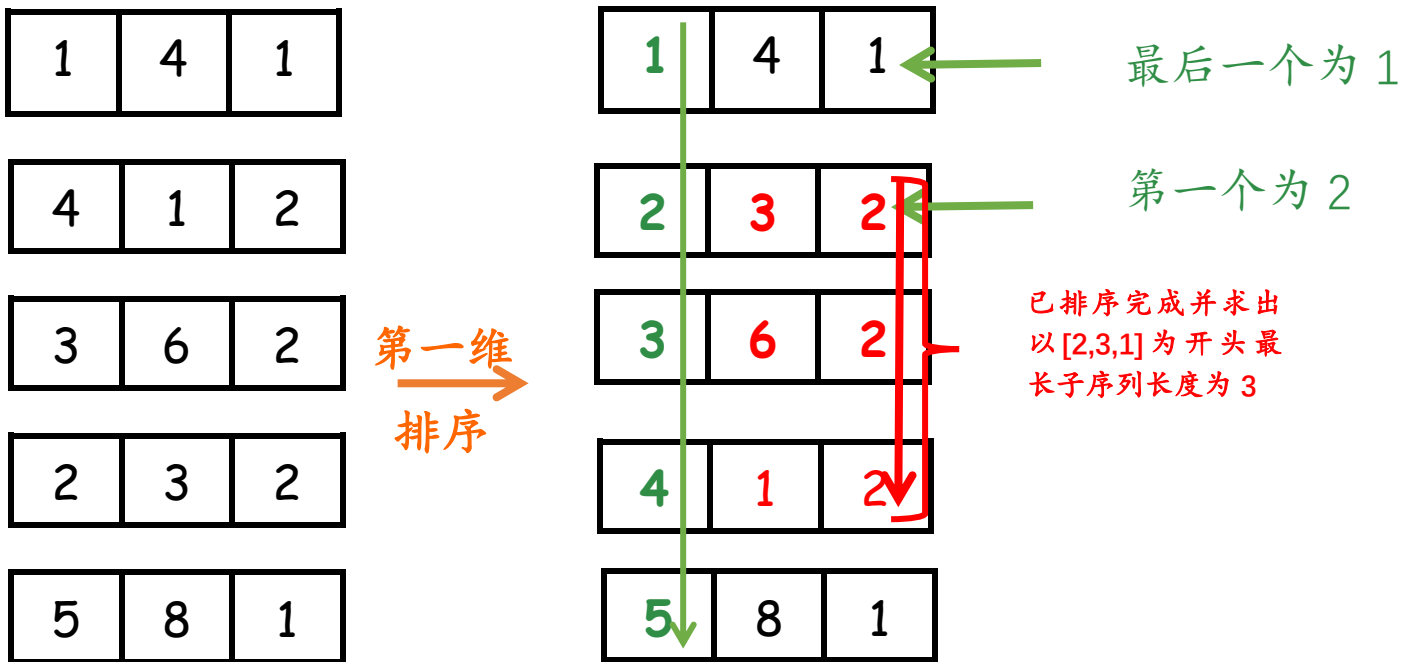
```

3 维偏序集($O(n\log n)$)

我们先考虑一种特殊情况:

第一种思路:如果第三维度只有两种值(1,2),那么第三维度两两之间的偏序关系已经绝对确定(仅 1 种),那么即可转换为二维偏序集问题求解并对第 3 维进行分类合并(因为在任意一个解中,第三维度的 1 总是排在 0 前面,所以可以枚举第三维度由最后一个数值为 1 到第一个数值为 2 的节点是哪一个),在所有的 n 种情况种取最大值(枚举分割点思想)

总复杂度 $O(n\log n+n)$



代码:

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 struct node{long long x,y,z;}a[100005];
4 long long
    p,l[100005],r[100005],f1[100005],f2[100005],n,ans1,ans2,ans;
5 bool cmp(node a,node b) {
6     if(a.x==b.x&&a.y==b.y) return a.z<=b.z;
7     if(a.x==b.x) return a.y<=b.y;
8     return a.x<=b.x;
9 }
10 int main() {

```

```
11  cin >> n;
12  for(int i=1;i<=n;i++) cin >> a[i].x >> a[i].y >> a[i].z;
13  sort(a+1,a+1+n,cmp);
14  for(int i=1;i<=n;i++)
15      if(a[i].z==1)
16          if(a[i].y>=f1[ans1]) f1[++ans1]=a[i].y,l[i]=ans1;
17          else p=upper_bound(f1+1,f1+1+ans1,a[i].y)-
              f1,l[i]=p,f1[p]=a[i].y;
18  else l[i]=upper_bound(f1+1,f1+1+ans1,a[i].y)-f1;
19  fill(f2,f2+100004,987654321);
20  for(int i=n;i>=1;i--)
21      if(a[i].z==2)
22          if(a[i].y<=f2[ans2])f2[++ans2]=a[i].y,r[i]=ans2;
23  else p=upper_bound(f2+1,f2+1+ans2,a[i].y,greater<int>())-
      f2,r[i]=p,f2[p]=a[i].y;
24 else r[i]=upper_bound(f2+1,f2+1+ans2,a[i].y,greater<int>())-f2;
25  for(int i=1;i<=n;i++) ans=max(ans,l[i]+r[i]-1);
26  cout << ans;
27  return 0;
28 }
```

另一种思路:

进行排序后去重,并从小到大逐个用二维偏序集中的第 2 类问题求满足以自己为结尾的偏序集数量,再对第三维度进行判断并合并存入 `f` 数组中(该方法为下文正式的三位偏序集问题的简化版,使用 `if` 语句判断第三维度的情况)


代码:

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 struct node{long long x,y,z;}a[100005];
4 long long n,f[100005], c[2][100005], pp[100005],cnt;
5 bool cmp(node a,node b) {
6     if(a.y==b.y&& a.z==b.z) return a.x<b.x;
7     if(a.y==b.y) return a.z<b.z;
8     return a.y<b.y;
9 }
10 void add(long long p,long long x,long long w) {
11     for(;p<=n;p+=p&-p) c[w][p] = max(c[w][p], x);
12 }
13 long long query(long long p,long long w) {
14     long long res = 0;
15     for(;p;p-=p&-p) res=max(res,c[w][p]);
16     return res;
```

```

17 }
18 int main() {
19     cin >> n;
20     for(int i=1; i<=n; i++) cin >> a[i].x >> a[i].y >> a[i].z;
21     sort(a+1, a+1+n, cmp);
22     for(int i=1; i<=n; i++) pp[++cnt]=a[i].x;
23     sort(pp+1, pp+cnt+1);
24     cnt=unique(pp+1, pp+1+cnt)-pp-1;
25     for(int i=1; i<=n; i++)
26         a[i].x=lower_bound(pp+1, pp+1+cnt, a[i].x)-pp;
27     for(int i=1; i<=n; i++)
28         if(a[i].z == 1) f[i]=query(a[i].x, 0)+1, add(a[i].x, f[i], 0);
29     else f[i]=max(query(a[i].x, 0),
30         query(a[i].x, 1))+1, add(a[i].x, f[i], 1);
31     cout << *max_element(f+1, f+n+1);
32     return 0;
33 }

```

 unique:对数组进行去重

现在我们考虑一般情况.

→暴力为什么这么慢? 因为计算了不需要二次计算的内容. 设一次枚举中已经确定 A, B, C 三个集合之间存在 A 偏序于 (用 \leq 号连接两个集合表示) $B, B \leq C$ 的关系, 那么在下一次枚举中如果出现 C 在 B 前面

的情况可以直接否决该情况,所以我们要剔除冗余计算

先按第一维的坐标排好序,然后需要在第一维排好序的点中找剩下两维的二维偏序.因为第一维度已经进行了排序,所以剩下的二维偏序不能再次降维进行,可以使用的方法:

1. 二维树状结构维护最值(树套树),大家可以自己实现(👉👉👉)

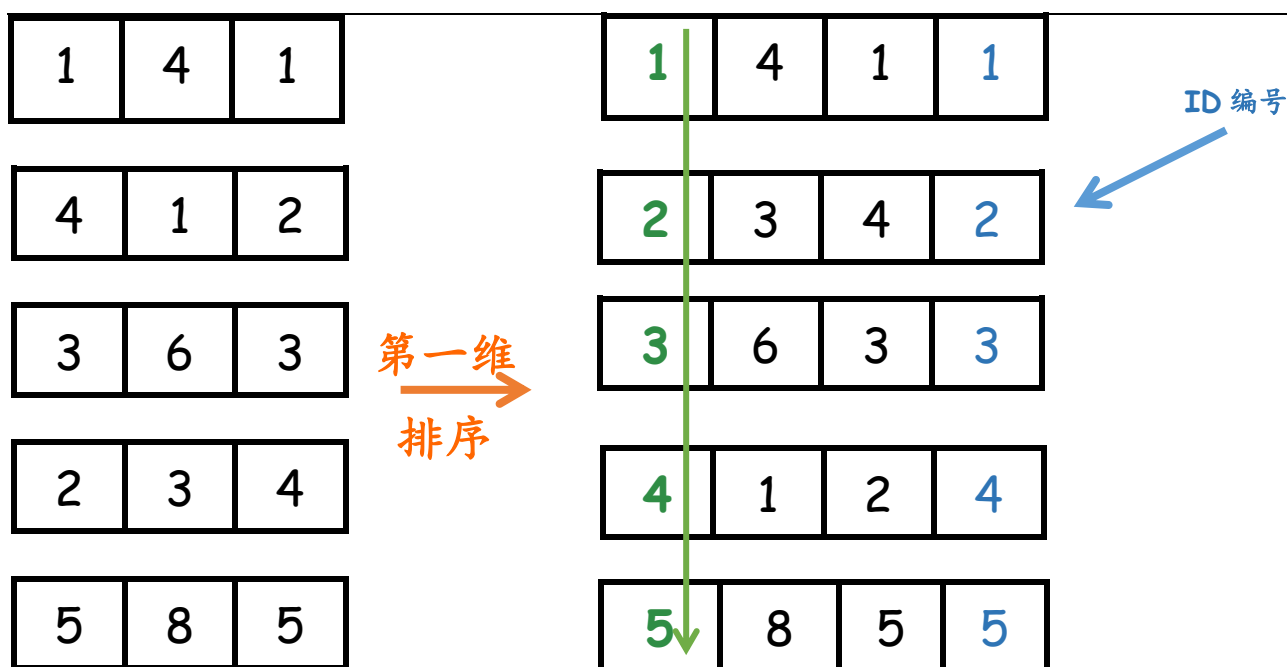
方法 1(树状数组套平衡树):我们要求的是与询问元素构成偏序关系的元素个数,换句话说也就是询问元素在原序列中的排名.关于排名的维护,可以使用平衡树(splay,treap...)但是因为是有 3 维,所以只维护 rank(排名)是不够的.于是开一个树状数组,每个节点为一个平衡树维护第 3 维的 rank 值,树状数组则在平衡树的查询下维护第 2 维的前缀数(有几个排在当前元素前面)即可.

方法 2(线段树套线段树):先按照第一维度排序,然后再做 2 维偏序集,用动态开点+四分的思想,将区间分为左上,左下,右上,右下四块,分别进行递归求解,最后统计.时间复杂度 $O(16 * q * n \log n)$

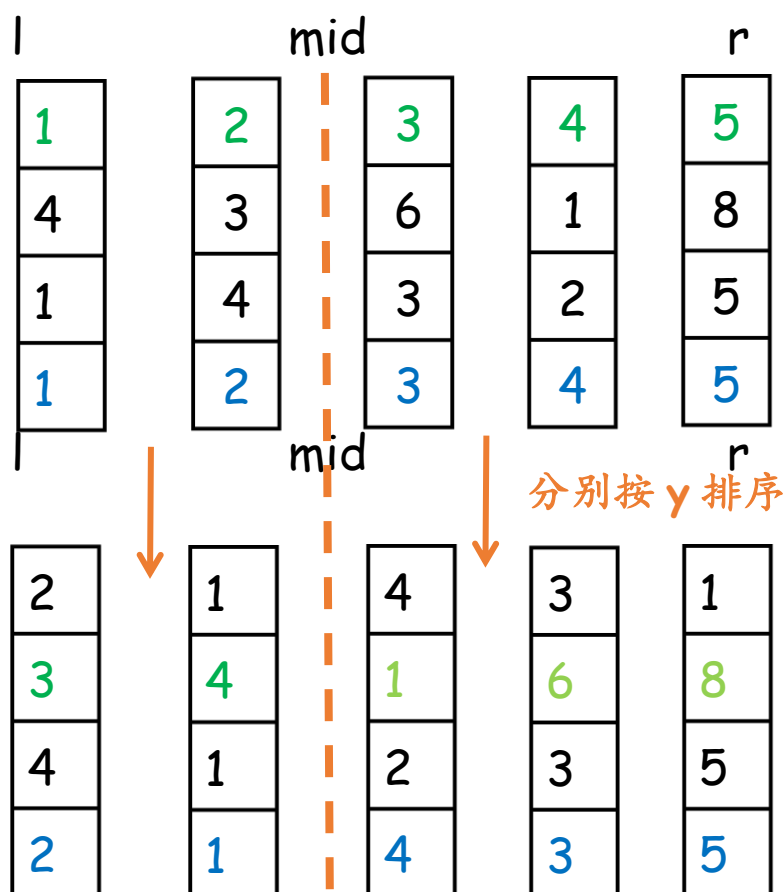
大常数预警~

方法 3(线段树套树状数组):是方法 2 的优化版.把方法 2 中的外层线段树换成树状数组维护,可以减小常数.

[CDQ 分治演示]



对分区间



树状数组操作列表

↑	update(a[2].id=1)
	query(1,a[1].z=1)
↑	update(a[3].id=4)
↑	update(a[3].id=4)
↑	update(a[4].id=3)
	query(1,a[2].z=4)

2.CDQ 分治(降维思想,计算贡献)

CDQ 分治的思想就是降维.我们先对第一维(x)排好序,然后将

区间 $[l,r]$ 内的集合对分成两个区间 $[l,mid]$ 和 $[mid+1,r]$ ($mid=(l+r)/2$).

将二分的区间分别再按第二维度(y)排序(因为第一维之前排过序,可以保证 $[mid+1,r]$ 的所有的 $x \geq *max_element(l,mid)$ ($[l,mid]$ 中所有的 x 的最大值)即前半边小于后半边. 因为我们需要更新的是左区间对右区间的影响, 所以就不用考虑区间内部信息. 做完之后第二维也是有序的. 然后枚举 $[mid+1,r]$ 每个元素, 并且维护左区间游标 i , 用 j 枚举 n 个元素的 y (第二维度值), 并用树状数组来维护第 3 维的值. 在枚举过程中当左区间出现 $a[i].y \leq a[j].y$ 的情况时将 $a[i].z$ 在树状数组中对应的位置的值 $bit[a[i].id]$ 更新为 $\max(dp[i], bit[a[i].id])$.

对于枚举到的每一个 j 号元素, 在更新了所有 $a[i].y \leq a[j].y$ 的 i 的信息之后我们就可以得出以 j 号元素为开头的最长 3 维 LIS 长度为 $dp[j] = query([1, j.z])$ (这一段区间的最大 dp 值) + 1;

最后, 答案即为 $*max_element(dp+1, dp+n+1)(dp[1] \sim dp[n])$ 中的最大值).

注意: 由于多次排序会造成前后的元素编号不统一, 所以我们要给每个元素一个确定的 id 编号以辨识.

[时间复杂度]

由于我们保证了 y 的单调性(已经排过序了), 所以时间复杂度大致是 $O(\text{区间长} * \text{更新单次的时间})$, 平摊之后是 $O(n \log 2n)$

代码:

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  struct node {long long x,y,z,id;} a[100005],tl[100005],tr[100005];
4  long long
5  dp[100005],n,m,d[100005],ans[100005],bit[100005],mid,t1,t2;
   bool cmpx(node aa,node bb) {
6      if(aa.x==bb.x&&aa.y==bb.y)return aa.z<bb.z;
7      if(aa.x==bb.x)return aa.y<bb.y;
8      return aa.x<bb.x;
9  }
10 bool cmpy(node aa,node bb) {
11     if(aa.y==bb.y)return aa.z<bb.z;
12     return aa.y<bb.y;
13 }
14 inline void clear(long long x) {
15     for(;x<=m;x+=x&(-x)) bit[x]=0;
16 }
17 void add(long long x,long long k) {
18     for(;x<=m;x+=x&(-x)) bit[x]=max(k,bit[x]);
```

```
19 }
20 long long query(long long x) {
21     long long ans=0;
22     for(;x;x-=x&(-x)) ans=max(ans,bit[x]);
23     return ans;
24 }
25 void calc(int l,int r) {
26     if(l>=r)return;
27     mid=(l+r)>>1,calc(l,mid),t1=0,t2=0;
28     for(int i=l; i<=mid; i++)tl[++t1]=a[i];
29     for(int i=mid+1; i<=r; i++)tr[++t2]=a[i];
30     sort(tl+1,tl+t1+1,cmpy),sort(tr+1,tr+t2+1,cmpy);
31     for(int i=1,j=1; j<=t2; j++) {
32         for(;tl[i].y<=tr[j].y&& i<=t1;i++) add(tl[i].z,dp[tl[i].id]);
33         long long sum=query(tr[j].z)+1;
34         if(dp[tr[j].id]<sum) dp[tr[j].id]=sum;
35     }
36     for(int i=l; i<=mid; i++)clear(a[i].z);
37     calc(mid+1,r);
38 }
39 int main() {
```

```

40  cin >> n;
41  for(int i=1; i<=n; i++) a[i].id=i,dp[i]=1,cin >> a[i].x >> a[i].y >>
    a[i].z,d[i]=a[i].z;
42  sort(d+1,d+n+1),m=unique(d+1,d+n+1)-d-1;
    for(int i=1; i<=n; i++)a[i].z=lower_bound(d+1,d+m+1,a[i].z)-d;
43  sort(a+1,a+n+1,cmpx),calc(1,n);
44  cout << *max_element(dp+1,dp+n+1);
45  return 0;
46  }

```

3.K-D tree(K 维偏序同样通用)

K-D 树是在 **k 维欧几里德空间** 组织点的数据结构。用 K-D 树解决三维偏序问题的大致思想和之前的 CDQ 分治差不多.先按照第一维 x 排序,然后再 2,3 两维(y,z)上分别建两棵 K-D 树维护当前元素前/后有多少个 y 和 z 符合条件(均小于当前位置的 y 和 z),查询时找出当前 K-D 树中在矩形 $\{(-inf,-inf),(y[i],z[i])\}$ 中的点权最大值+1 即是 $f[i]$,再把 $(y[i],z[i])$ 带上点权 $f[i]$ 插入 K-D 树中即可

注意:在 K-D 树种查询时遇到整个子树中的最大值小于等于当前搜到的最大值时就可以退出了,否则超时!

三维偏序集各种方法对比表

n=	树状数组 +平衡树	线段树+ 线段树	线段树+ 树状数组	CDQ 点分治	K-D 树
100	4ms 656.00K	9ms 6.75M	6ms 2.11M	5ms 788.00K	4ms 652.00K
5000	83ms 4.56M	69ms 17.12M	50ms 6.89M	44ms 1.77M	38ms 1.14M
10000	81ms 5.89M	775ms 129.84M	360ms 60.72M	92ms 2.53M	132ms 2.06M
50000	428ms 12.77M	1.76s 179.63M	763ms 118.88M	132ms 4.41M	406ms 4.66M
100000	641ms 19.01M	4.28s 347.83M	1.09s 239.52M	148ms 5.09M	766ms 7.66M
编码行数	119	81	78	46	114

总结:

1.CDQ 绝对是首选方案,一路完胜

2.线段树的空间,时间复杂度都要小心谨慎

3.能不用线段树就别用,代码量太大

那么如何解决维数更加高的偏序集问题呢?

CDQ 不断嵌套(写 if 语句判断是个好办法~)复杂度 $k * n \log n$

然而...有更好的方法

现在,偷偷的告诉你这个秘密~(赶紧看看有没有泄密的可能性)

->本论文中所有题目用暴力分块的方法都可以拿到 80 分以上
所以.....

对于 k 维偏序集问题,我们可以使用分块思想.分块时一种比较通用的方法,对于各种查询,最值,统计问题都可以减少操作次数.我们把套娃分成 \sqrt{n} 块,每个维度分开来单独处理,按照大小排序,并记录前缀和(到某个排序后的位置,每个数的出现情况)在此,用 ll,rr 维护每个块中的情况.

查询有哪些元素于待查元素构成偏序关系的时候二分查找即可。
看一下时间复杂度: $O(n*k+m*k*\sqrt{n})$ 当 $n=100000, k=50$ 时 4.76s
怎么办!!不够快!!

->使用 `bitset`(常数优化神器)->优化后 753ms

代码:

```
1 #include<bits/stdc++.h>
2 using namespace std;
3 struct node {
4     long long val,id;
5     const bool operator < (const node& o) const { return
        val < o.val;}
6 } a[105][100005];
7 long long
8 len,T,n,m,siz,cnt,l[505],r[505],bb[100005],ans,gg;
```

```
bitset<100005> mp[105][505],tmp,tmp1;
9 int main() {
10     cin >> n >> len,siz=sqrt(n),cnt=(n-1)/siz+1;
11     for(int j=1; j<=cnt; j++) {
12         l[j]=r[j-1]+1;
13         if(j==cnt) r[j]=n;
14         else r[j]=siz*j;
15         for (int k=l[j]; k<=r[j]; k++) bb[k]=j;
16     }
17     for(int i=1; i<=len; i++)
18         for(int j=1; j<=n; j++)
19             cin >> a[i][j].val,a[i][j].id=i;
20     for (int i=1; i<=len; i++) {
21         sort(a[i]+1,a[i]+n+1);
22         for (int j=1; j<=cnt; j++) mp[i][j]=0;
23     }
24     for(int i=1; i<=len; i++)
25         for(int j=1; j<=cnt; j++) {
26             for (int k=l[j]; k<=r[j]; k++) mp[i][j][a[i][k].id]=1;
27             mp[i][j]|=mp[i][j-1];
28         }
```

```
29  cin >> m,ans=0;
30  node x= {0,n+1};
31  for(int j=1; j<=m; j++) {
32      tmp.set(),gg=0;
33      for(long long t,i=1; i<=len; i++) {
34          cin >> x.val; if(gg)continue;
35          t=upper_bound(a[i]+1,a[i]+1+n,x)-a[i]-1;
36          if (t<1) {tmp.reset(),gg = 1;continue;}
37          tmp1=mp[i][bb[t]-1];
38          for (int j=l[bb[t]]; j<=t; j++) tmp1[a[i][j].id]=1;
39          tmp&=tmp1;
40      }
41      cout << tmp.count() << endl;
42  }
43  return 0;
44 }
```

[三维偏序衍生问题]

1.动态逆序对问题

给出 $1\sim n$ 的一个排列，按照某种顺序依次删除 m 个元素，每次删除元素前统计整个序列的逆序对数量。

$$1\leq n\leq 100000, 1\leq m\leq 50000$$

[思路]

1. 继续 CDQ 分治

删除操作可以变成倒着插入。设 $t[i]$ 表示第 i 个插入的时间为, 那么 $t[1]=n$, 表示最后一个插入。然后为了方便, 我们把未被删除的结点的 $t[i]$ 从左往右设为 $1, 2, 3 \dots$

考虑问题转换成了求对于 $(t[0], x[0], y[0])$ 满足 $t < t[0], x < x[0], y > y[0]$ 的 (t, x, y) 的个数, 这样就变成了三维偏序集问题了, 解法见 p14-p20。细节上有必要再说一下:

分治之前按 t 排序, 保证 t 已经有序, 在每次分治中, 先按 x 排序, 然后扫描整个区间. 对于 $[mid+1, r]$ 的区间就在树状数组上查询大于他的 y 的值的数量; 倒着扫, 对于 $[l, mid]$ 的区间就在树状数组上查询小于他的 y 的值的数量。因为左边的所有元素对于右边的所有元素而言, 是可以肯定 t 要小一些的, 所以分治就可以确保每次二分后所取的结果满足 $t < t_1, x < x_1, y < y_1$ 的条件, 如此就解决了三维偏序的问题, 之后再以同样的做法递归处理左边右边就可以了。

时间复杂度: $O(n \log^2 n)$ 空间复杂度: $O(n)$

核心部分代码(省略 CDQ):

```
1 cin >> n >> m;
```

```
2 for (int i=1; i<=n; ++i)
```

```
    cin >> a[i], pos[a[i]]=i, b[++tot]=(node){1, a[i], i, 0, tot};
```

```
3 for (int i=1, x; i<=m; ++i) cin >> x, b[++tot]=(node){-1, x, pos[x], i, tot};
```



```

4 cdq(1,tot);
5 for (int i=1;k=m;++i) ans[i]+=ans[i-1];
6 for (int i=0;i<m;++i) cout << ans[i] << endl;

```

2.继续分块

每当删掉一个数,就相当于减去(该数前面比它大数的个数+该数后面比它小的个数).于是我们把整个序列分成 \sqrt{n} 块,并在每个块中进行排序.

首先用树状数组统计整体逆序对数量.

当要删除 k 时,先在 k 所在的块中用树状数组计算逆序对数量 a ,然后在 k 所在块之前的所有块中用二分找到第一个大于 k 的数(与 k 能构成逆序对)计算出当前大于 k 的数的数量 b ,并在 k 所在块之后的所有块中用二分找到最后一个小于 k 的数(与 k 能构成逆序对)计算出当前小于 k 的数的数量 c , $a+b+c$ 则是删去后减少的数量.

时间复杂度 $O(n\sqrt{n})$

[演示]

首先计算出全序列的逆序对数量:15 个



查询:删去第 6 个元素"2"之后逆序对数量

9	7	2
---	---	---

 中 2 所产生的逆序对维 2 个:(9,2)和(7,2)



1	4	5
---	---	---



处于前部块,找到 4 是第一个大于 2 的数,所以该块中有 2 个逆序对(4,2),(5,2)在删除"2"后消失.

3	6	8
---	---	---



处于后部块,没有找到小于 2 的数,所以没有逆序对在删除"2"后消失.

将三次得到的答案相加,2+2=4 即是删除后消失的逆序对数量.

所以存留逆序对数量 15-4=11 个,别忘了 ans 的更新是永久的~

[代码]

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 long long
  ans,n,m,x,y,k,ll,rr,rec,num,block,a[500005],b[500005],
  bit[500005],tmp[500005],belong[500005],l[500005],
  r[500005],s[500005],id[500005];
4 bool f[500005];
5 inline long long lowbit(long long x){return x&-x;}
6 void add(long long x,long long v){
7   for(;x<=n;x+=lowbit(x))bit[x]+=v;
8 }
```

```
9 long long query(long long x){
10     long long sum=0;
11     for(;x;x-=lowbit(x)) sum+=bit[x];
12     return sum;
13 }
14 long long finder(long long pos,long long val){
15     ll=l[pos],rr=r[pos],rec=r[pos];
16     while(ll<=rr){
17         int mid=(ll+rr)/2;
18         if(b[mid]<=val) rec=mid,ll=mid+1;
19         else rr=mid-1;
20     }
21     if(b[rec]>val) return l[pos]-1;
22     else return rec;
23 }
24 int main(){
25     cin >> n >> m;
26     block=sqrt(n),num=n/block+1;
27     for(int i=1; i<=num; i++) l[i]=(i-1)*block+1,r[i]=i*block;
28     r[num]=n;
29     for(int i=1; i<=n; i++) belong[i]=(i-1)/block+1;
```

```

    for(int i=1; i<=n; i++) cin >> a[i],tmp[i]=a[i],id[a[i]]=i,
30      add(a[i],1),ans+=i-query(a[i]),b[i]=a[i],f[i]=1;
31  for(int i=1; i<=num; i++) sort(b+l[i],b+r[i]+1);
32  while(m--){
33    cout << ans << endl, cin >> y;
34    x=id[y],k=belong[x];
35    for(int i=1; i<k; i++) ans-=(r[i]-finder(i,a[x]));
36    for(int i=k+1;i<=num;i++) ans-=finder(i,a[x])-l[i]+1-s[i];
37    for(int i=l[k]; i<x; i++) if(f[a[i]]&&a[i]>a[x]) ans--;
38    for(int i=x+1;i<=r[k];i++)if(f[a[i]]&&a[i]<a[x]) ans--;
39    b[finder(k,a[x])]=0,s[k]++,f[a[x]]=0;
40    sort(b+l[k],b+r[k]+1);
41  }
42  return 0;
43}

```

4 个 for 循环..很像莫队~

[结语]

多维偏序问题在算法领域的许多方面都有应用,欢迎大家继续探索效率更高的算法.