多维偏序集系列问题研究

徐润麒

概念

“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）∈R，就把它表示成a≤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≤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{x1,x2,x3...xn},B{y1,y2,y3...yn}

满足x1<y1且x2<y2且x3<y3...且xn<yn

->也有可能是x1<=y1且x2<=y2...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<=B),B<=C,则A<=C

2.反对称性 A<=B,B<=A,则A=B

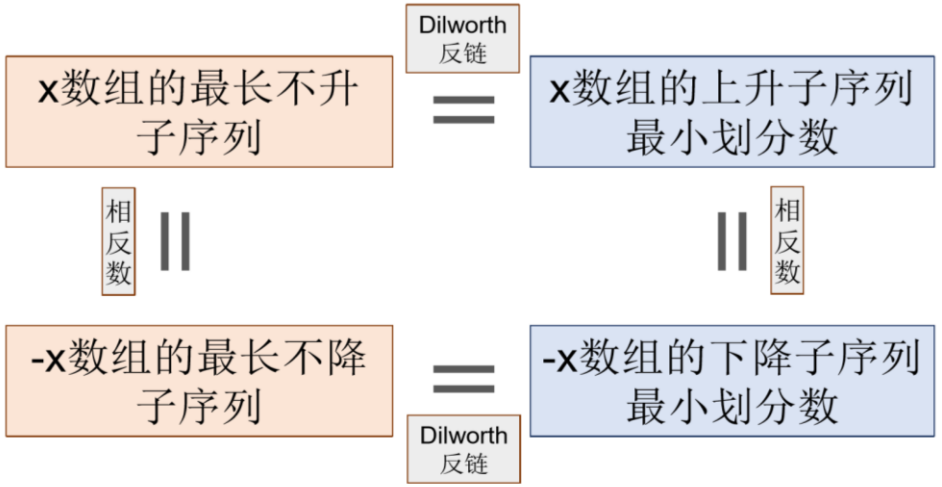
3.自反性 A <=A

[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>=r

2.设B1＝A，A1是A中的极小元的集合。从B1中删除A1得到B2。

对于B2中任意元素a2，必存在B1中的元素a1，使得a1<=a2。

令A2是B2中极小元的集合，从B2中删除A2得到B3……

最终，会有一个Bk非空而B(k+1)为空。

于是A1,A2,…,Ak就是A的反链的划分，

同时存在链a1<=a2<=…<=ak，其中ai在Ai内。

由于r是最长链大小，因此r>=k。

由于A被划分成了k个反链，因此r>=k>=p。因此r=p，

定理得证。

[题目]

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个描述体积的属性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k维世界的你想知道最多可以将多少个套娃套在一起。

[输入]

n k

n*k的二维数组

[输出]

最多可以将多少个套娃套在一起

[数据范围]

n<=200000,k<=100

逐步解答

很明显,K是关键

开始分类讨论~

->k=1.. So easy! 排序后输出满足a[i]>a[i-1]的个数

->k=2.. So easy! 二维LIS 复杂度O(nlogn)

->k=3.4 只能暴力了... 享受等待的快乐~(15min+)

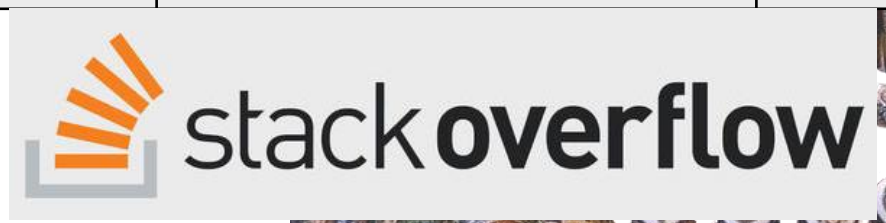
复杂度$O((Nk)^k)$


```
void dfs(long long x,long long pre){
  for(int i=1;i<=n;i++ fflag=0) {
    if(vst[i]==0){ flag=0;
      for(int j=1;j<=k;j++) if(a[i][j]<=a[pre][j]) {flag=1; break;}
      if(flag==0) vst[i]=1,dfs(x+1,i),vst[i]=0,fflag=1;
    }
  }
  if(fflag==0) {ans=max(ans,x); return;}
}
```

4,3,6,1,8 最长上升子

很有意思的 DFS 终止条件~

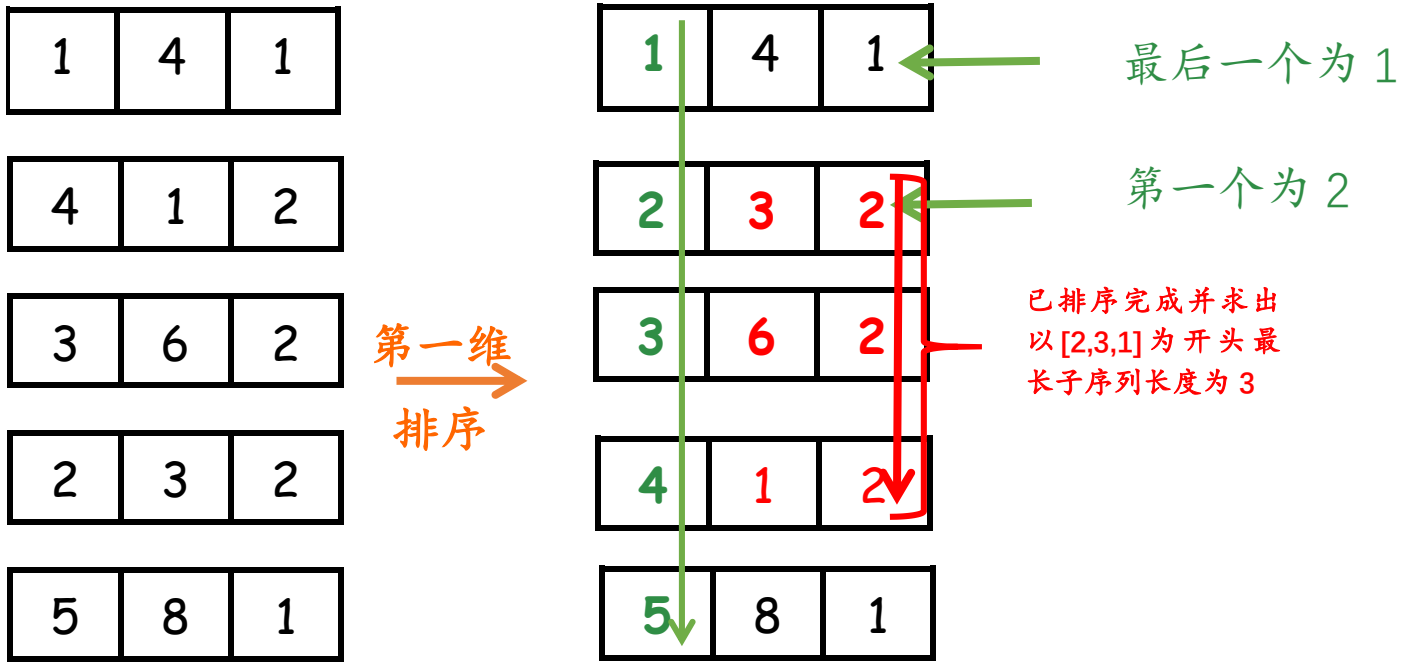
N,K	时间(s)	空间(MB)
100,3	0.97	10.2
1000,3	10.2	77.6
10000,5	692.7	298.9
200000,100	-INF(stack overflow)-	-INF-



特殊情况

->如果第三维只有两种值能怎么解决?

可以前两维度先做二维LIS然后第三维度合并(枚举分割点)



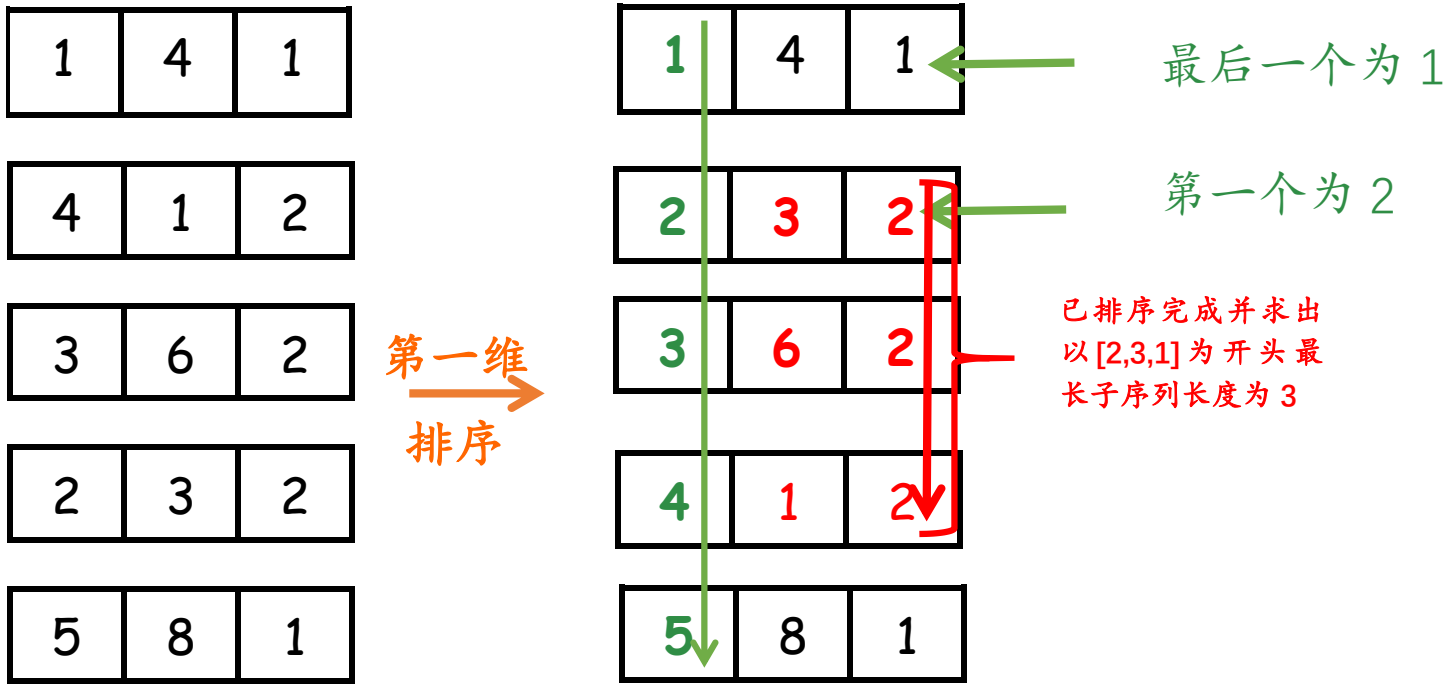
```
for(int i=1;i<=n;i++)
  if(a[i].z==1) if(a[i].y>=f1[ans1]) f1[++ans1]=a[i].y,l[i]=ans1;
    else p=upper_bound(f1+1,f1+1+ans1,a[i].y)-f1,l[i]=p,f1[p]=a[i].y;
  else l[i]=upper_bound(f1+1,f1+1+ans1,a[i].y)-f1;
fill(f2,f2+100004,987654321);
for(int i=n;i>=1;i--)
  if(a[i].z==2) if(a[i].y<=f2[ans2])f2[++ans2]=a[i].y,r[i]=ans2;
      else p=upper_bound(f2+1,f2+1+ans2,a[i].y,greater<int>())-f2,r[i]=p,f2[p]=a[i].y;
  else r[i]=upper_bound(f2+1,f2+1+ans2,a[i].y,greater<int>())-f2;
for(int i=1;i<=n;i++) ans=max(ans,l[i]+r[i]-1);
```

做第三维为1的个二维LIS存在l数组中

做第三维为2的二维LIS存在r数组中同时统计以该元素作为答案中第一个第三维为2时最大能接上的l数组中的元素编号

枚举分割点合并

l,r的涵义为以第i号元素作为分割点第三维为1,为2分别有最多有多少个元素



正式三维偏序集

正解:CDQ点分治,树套树,K-D树

->树套树:

方法1(树状数组套平衡树):时间复杂度O(t(较小)*nlogn*q)

方法2(线段树套线段树) 时间复杂度O(t(>=16)*q*nlogn)大常数预警~

方法3(线段树套树状数组):方法2的优化版,减小常数.

->K-D树:

K-D树是在k维欧几里德空间组织点的数据结构

用于计算K维欧几里德空间内不同点的位置关系

在此不再赘述

*注意事项:K-D树有较高的复杂度,所以在搜到最大值的时候一定要及时退出(急流勇退)

正式三维偏序集

正解:CDQ点分治,树套树,K-D树

CDQ分治 ->最大限度利用现有信息.

对于暴力方法,如何减少计算量?

CDQ分治步骤: 分治->合并->算左边对右边的贡献->增量式统计结果

时间复杂度O(nlogn)

先排序(直接解决一维)<-偏序集系列问题初级手段

然后把整个序列分成两个子序列,在子序列中进行排序(按照另一维).

此时,两个子序列间形成了**二维偏序关系(分治+排序导致)**,

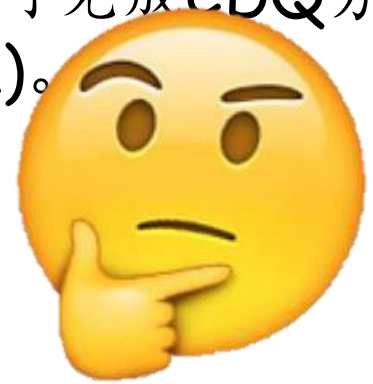
即原来第一维,第二维在左边的值小于等于右边的值.

目前子序列有只有左边的能对右边的产生贡献的特性.

所以可以花较小的代价统计增加的贡献.

继续在序列中递归操作,即可完成CDQ分治.

时间复杂度O(nlogn×logk)。



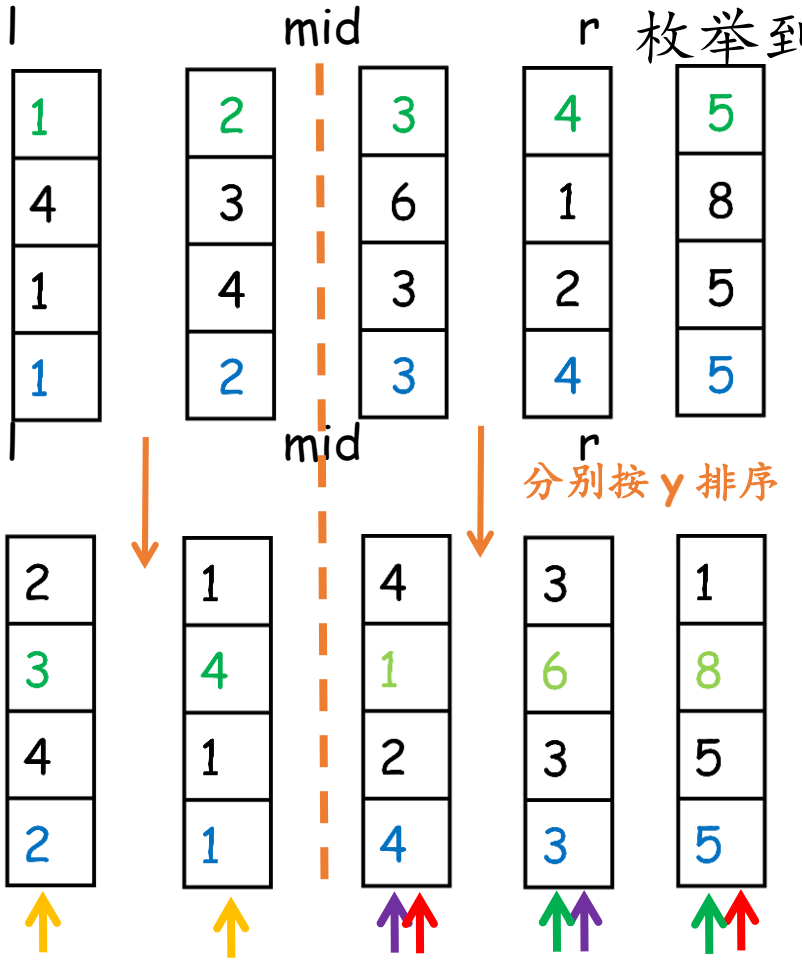
就这么完了

当然没这么简单.

如何计算贡献呢?

计算贡献

维护左序列游标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<=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]).
枚举到的每个j号元素,在更新了所有a[i].y<=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长度是
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]~dp[n]中的最大值).



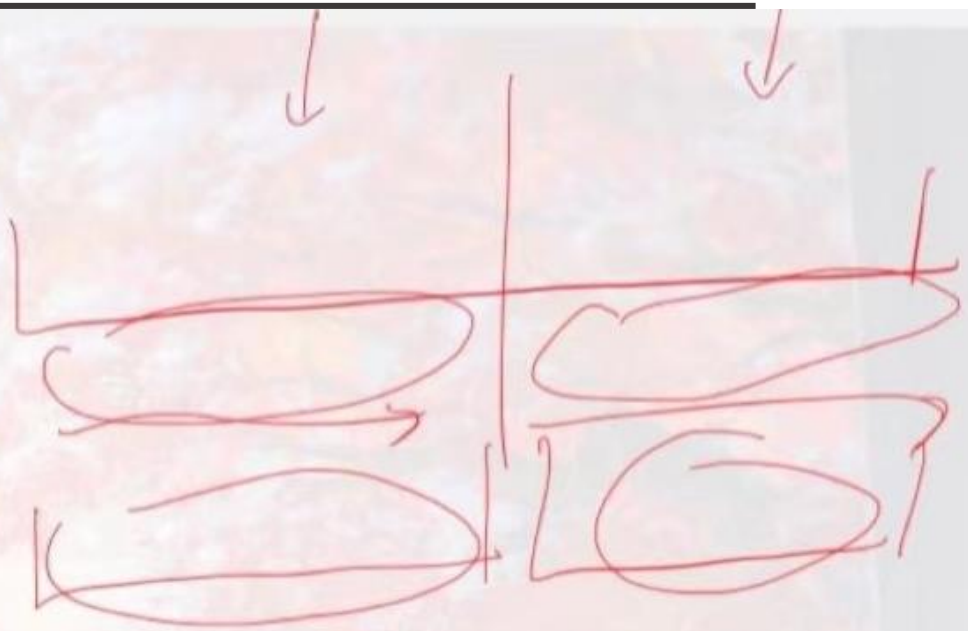
树状数组操作列表

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)

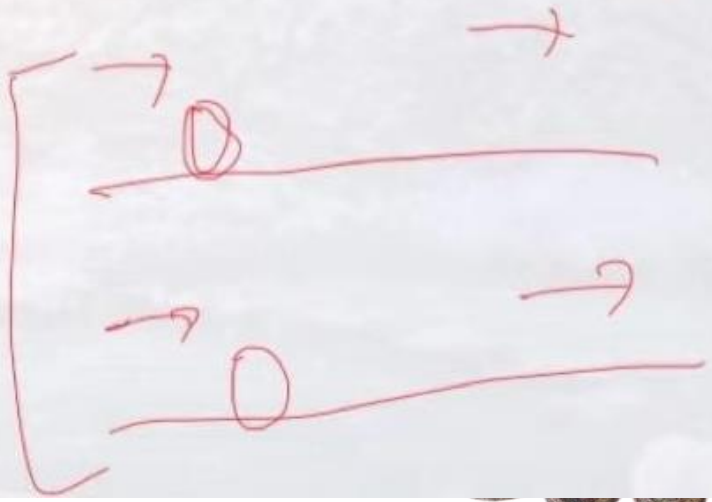
cdq 分治

第一维排序

第二维分治，考虑 (l, mid) $(mid+1, r)$ 都分别按照第二维排序



CDQ分治精髓写意图
分治法的完美诠释



性能测试

三维偏序五大高阶方法大比拼

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*nlogn

->难道就这样结束了吗?
->难道多维偏序一定要写100+行吗?

NO NO NO

还有一个很方便的方法~

**一定要用BitSet常数优化,
否则到10000*50就会超时

分块

把套娃分成√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.
查询有哪些元素与待查元素构成偏序关系数量:
在每个块中二分查找构成数量后累加
优化:有的块可以判断无贡献后直接跳过
时间复杂度:O(n*k+m*k*√n)

```
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 len,T,n,m,siz,cnt,l[505],r[505],bb[100005],ans,gg;
8   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++)   l[j]=r[j-1]+1;
12      if(j==cnt) r[j]=n; else r[j]=siz*j;
13      for (int k=l[j]; k<=r[j]; k++) bb[k]=j;
14    }
15    for(int i=1; i<=len; i++)  for(int j=1; j<=n; j++) cin >> a[i][j].val,a[i][j].id=i;
16    for (int i=1; i<=len; i++) {
17     sort(a[i]+1,a[i]+n+1); for (int j=1; j<=cnt; j++) mp[i][j]=0;
18    }
19    for(int i=1; i<=len; i++)
20     for(int j=1; j<=cnt; j++) {
21      for (int k=l[j]; k<=r[j]; k++) mp[i][j][a[i][k].id]=1;
22       mp[i][j]|=mp[i][j-1];
23     }
24    cin >> m,ans=0;
25    node x={0,n+1};
26    for(int j=1; j<=m; j++) {
27     tmp.set(),gg=0;
28     for(long long t,i=1; i<=len; i++) {
29      cin >> x.val; if(gg)continue;
30      t=upper_bound(a[i]+1,a[i]+1+n,x)-a[i]-1;
31      if (t<1) {tmp.reset(),gg = 1,continue;}
32      tmp1=mp[i][bb[t]-1];
33      for (int j=l[bb[t]]; j<=t; j++) tmp1[a[i][j].id]=1;
34      tmp&=tmp1;
35     }
36     cout << tmp.count() << endl;
```

分√n块

gg判断是否还有继续查更多块的必要

二分查询

注意:分块必备技能:bitset加速

不用bitset: n=100000,k=50 4.76s

使用bitset: n=200000,k=100 753ms

衍生问题研究

1.动态逆序对问题

[描述]

给出1~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。

[输入]

n m
a1 a2 a3 … an
q1 q2 q3… Qn

[输出]

ans1 ans2 ans3 … ansn

[数据范围]

1≤n≤100000，1≤m≤50000

良心数据

动态逆序对解法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...

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

这样动态逆序对问题就变成了三维偏序集问题了.细节见论文.

测试点信息

#1 AC 4ms/660.00KB	#2 AC 709ms/13.69MB	#3 AC 8ms/816.00KB	#4 AC 116ms/3.09MB	#5 AC 143ms/4.53MB
#6 AC 249ms/5.87MB	#7 AC 262ms/7.18MB	#8 AC 388ms/8.49MB	#9 AC 366ms/8.48MB	#10 AC 566ms/13.64MB

动态逆序对解法2.高境界暴力-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√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,

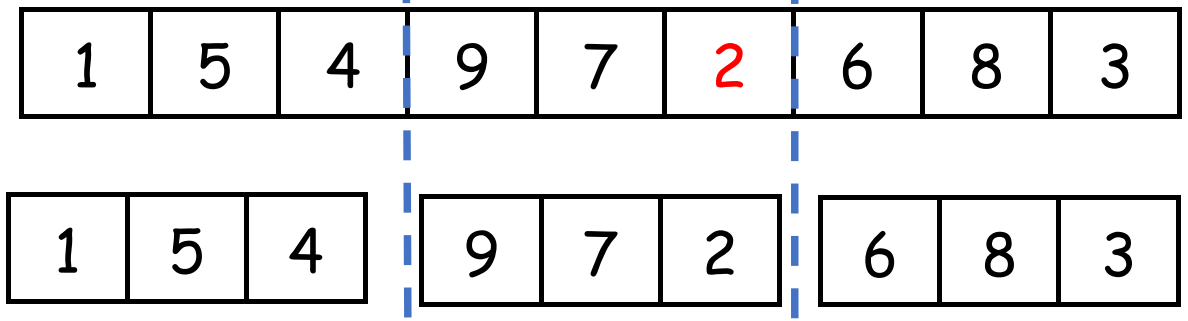
然后在k所在块之前的所有块中用二分找到第一个大于k的数(与k能构成逆序对)计算出当前大于k的数的数量b,并在k所在块之后的所有块中用二分找到最后一个小于k的数(与k能构成逆序对)计算出当前小于k的数的数量c,

a+b+c则是删去后减少的数量.

->时间复杂度O(n√n)

首先计算出全序列的逆序对数量:15 个

树状数组



动态逆序对解法2.高境界暴力-继续分块.

查询:删去第 6 个元素"2"之后逆序对数量

分块

9	7	2

中 2 所产生的逆序对维 2 个(9,2)和(7,2)

1	4	5

2	7	9

3	6	8

测试点信息

暴力分块

#1 AC 4ms/800.00KB	#2 AC 972ms/5.43MB	#3 AC 4ms/828.00KB	#4 AC 84ms/1.67MB	#5 AC 103ms/2.27MB
#6 AC 263ms/2.55MB	#7 AC 270ms/3.02MB	#8 AC 467ms/3.52MB	#9 AC 571ms/3.52MB	#10 AC 848ms/5.43MB

分块虽慢、写起来快

CDQ分治

4ms/660.00KB	#2 AC 709ms/13.69MB	#3 AC 8ms/816.00KB	#4 AC 116ms/3.09MB	#5 AC 143ms/4.53MB
#6 AC 249ms/5.87MB	#7 AC 262ms/7.18MB	#8 AC 388ms/8.49MB	#9 AC 366ms/8.48MB	#10 AC 566ms/13.64MB

THANKS