CQD(陈丹琦)分治 & 整体二分——专题小结

[摘要：团体两分战CDQ分治 有一些题目良多时光皆坑正在斜率战凸壳上了么……感到斜率战凸壳种种弄没有懂 …… 团体两分 团体两分的材料似乎没有是良多，我正在网上找到了一篇没有错的材料： 团体]

整体二分和CDQ分治

有一些问题很多时间都坑在斜率和凸壳上了么……感觉斜率和凸壳各种搞不懂……

整体二分

整体二分的资料好像不是很多，我在网上找到了一篇不错的资料：

整体二分是个很神的东西，它可以把许多复杂的数据结构题化简。它的精髓在于巧妙地利用了离线的特点，把所有的修改、询问操作整体把握。

先说说第k大数吧，这种问题是整体二分的标志性题目，什么划分树啊，主席树啊，树套树啊见了整体二分都得自叹不如。首先对于一次询问来说我们可以二分答案，然后通过验证比答案大的数有多少个来不断地缩小答案范围直至得到一个准确的答案。而对于多个询问我们同样可以这么做，只不过对每一个询问我们都需要判定一下，以决定它被划分到哪一个答案的区间里。这个判定过程就是通过比较比二分的mid大的数的个数和k。同时我们看到，如果比二分的mid大的数的个数小于k了，我们是要去寻找小的答案，那么这些比mid大的数在以后的递归里始终会对答案有贡献，所以我们没必要去做重复的工作，只需要把这些数的个数累积到贡献里，以后递归的时候就不用考虑这些数了。具体地，我们把数列里的数也和询问一起递归，这样这些数也会被分到属于的答案区间里，并且只对相应区间里的询问有影响。如果有修改呢，我们把修改看成一个数就好了，一样可以随着递归不断地被划分下去。

我们看到，整体二分的过程实质上是个按照数值来划分操作序列的过程，于是我们的复杂度也就和操作序列的长度线性相关，那么我们在中间维护一些信息的时候，就一定不能有何数列长线性相关的东西，否则会破坏其时间复杂度，具体的复杂度证明请见2013年集训队XHR论文。

贴个典型性的代码吧，带修改区间第k小数

```
#include<iostream> #include<cstdio> #include<algorithm> #include<cmath> #include<cstring> #define maxn 220000 #define inf 1000000000  using namespace std; struct query {     int x,y,k,s,tp,cur; }q[maxn],q1[maxn],q2[maxn]; int a[maxn],ans[maxn],tmp[maxn],t[maxn]; int n,m,num,cnt;  void add(int x,int y) {     for (int i=x;i<=n;i+=(i&-i)) t[i]+=y; }  int ask(int x) {     int tmp=0;     for (int i=x;i>0;i-=(i&-i)) tmp+=t[i];     return tmp; }  void divide(int head,int tail,int l,int r) {     //cout<<head<<' '<<tail<<' '<<l<<' '<<r<<endl;     if (head>tail) return ;     if (l==r)     {         for (int i=head;i<=tail;i++)             if (q[i].tp==3) ans[q[i].s]=l;//,cout<<l<<endl;         return ;     }     int mid=(l+r)>>1;     for (int i=head;i<=tail;i++)     {         if (q[i].tp==1&&q[i].y<=mid) add(q[i].x,1);         else         if (q[i].tp==2&&q[i].y<=mid) add(q[i].x,-1);         else         if (q[i].tp==3) tmp[i]=ask(q[i].y)-ask(q[i].x-1);     }     for (int i=head;i<=tail;i++)     {         if (q[i].tp==1&&q[i].y<=mid) add(q[i].x,-1);         else         if (q[i].tp==2&&q[i].y<=mid) add(q[i].x,1);     }     int l1=0,l2=0;     for (int i=head;i<=tail;i++)         if (q[i].tp==3)  {             if (q[i].cur+tmp[i]>q[i].k-1)//q[i].cur+tmp[i]表示累积了多少个数                 q1[++l1]=q[i];             else             {                 q[i].cur+=tmp[i];                 q2[++l2]=q[i];             }         }         else         {             if (q[i].y<=mid) q1[++l1]=q[i];             else q2[++l2]=q[i];         }     for (int i=1;i<=l1;i++) q[head+i-1]=q1[i];     for (int i=1;i<=l2;i++) q[head+l1+i-1]=q2[i];  divide(head,head+l1-1,l,mid);     divide(head+l1,tail,mid+1,r); }      int main() {     //freopen("ranking.in","r",stdin);     //freopen("ranking.out","w",stdout);     scanf("%d%d",&n,&m);     for (int i=1;i<=n;i++)     {         scanf("%d",&a[i]);         q[++num].x=i; q[num].y=a[i];         q[num].tp=1; q[num].s=0;     }  char sign;     int x,y,z;     for (int i=1;i<=m;i++)     {         scanf("n%c",&sign);         if (sign=='Q')         {             scanf("%d%d%d",&x,&y,&z);             q[++num].x=x,q[num].y=y,q[num].k=z;             q[num].tp=3; q[num].s=++cnt;         }         else         {             scanf("%d%d",&x,&y);             q[++num].x=x; q[num].y=a[x];             q[num].tp=2; q[num].s=0;             q[++num].x=x; q[num].y=y;             q[num].tp=1; q[num].s=0;             a[x]=y;         }     }     divide(1,num,0,inf);     for (int i=1;i<=cnt;i++)         printf("%dn",ans[i]);     return 0; }
```

CQD分治

来看一道例题：

[Poi2011] Meteors

有n个国家和m个空间站，每个空间站都属于一个国家，一个国家可以有多个空间站，所有空间站按照顺序形成一个环，也就是说，m号空间站和1号空间站相邻。

现在，将会有k场流星雨降临，每一场流星雨都会给区间[li,ri]内的每个空间站带来ai单位的陨石，每个国家都有一个收集陨石的目标pi，即第i个国家需要收集pi单位的陨石。

询问：每个国家最早完成陨石收集目标是在第几场流星雨过后。

数据范围：1<=n,m,k<=300000

对于单个查询（假设为第i个国家），我们可以二分k，每次对于一个区间[l,r]，手动模拟一下在第mid场流星雨过后，第i个国家一共收集到了多少单位的陨石，如果比pi大，那么答案在[l,mid]范围内，否则答案在[mid+1,r]范围内。

对于多组查询，我们也可以这么做。首先，我们需要用一个列表id[]记录所有查询的编号，刚开始的时候，id[]自然是递增的.同时，我们用一个数组cur[i]记录下，第i个国家在l-1场流星雨过后，收集到的陨石的数目。

主过程为void solve(int head,int tail,int l,int r)，表示对于id[head]到id[tail]的所有询问，在[l,r]范围内查询答案，通过上一层的操作，我们保证id[head]到id[tail]的所有询问的答案都在[l,r]范围内。

首先，我们先模拟[l,mid]这么多次操作（在询问重新划分之后，必须要再次模拟，将数组清空），用树状数组或者是线段树计算出在[l,mid]场流星雨之后，每个空间站收集到的陨石的数目。

然后我们查询，每个国家收集到的陨石的数目，要注意的是，我们需要用链表储存每个国家对应的空间站，并且一一枚举，用tmp[id[i]]表示国家id[i]收集到的陨石的数目。

那么从[1,mid]这么多次操作之后，国家id[i]收集到的陨石数目就是tmp[id[i]]+cur[id[i]]，如果tmp[id[i]]+cur[id[i]]>p[id[i]]，那么表明对于国家id[i]，其答案在[l,mid]这个范围内，否则其答案在[mid+1,r]范围内，并将tmp[id[i]]累加到cur[id[i]]上。

还有一个坑点是，tmp[id[i]]可能很大，会爆掉long long，所以如果枚举一个国家的所有空间站的时候，发现tmp[id[i]]已经大于p[id[i]]了，那么就break好了，不然会出错。

因为可能会出现怎么也无法满足的情况，所以我们需要多增加一场流星雨，这场流星雨的数量为infi，保证能够让所有国家都满足要求，那么最后，对于所有答案为k+1的询问，输出NIE就行了。

总的来说，整体二分就是将所有询问一起二分，然后获得每个询问的答案。

CDQ相比整体二分略有不同，整体二分是对答案进行二分，而CDQ分治则是对于所有操作进行二分。

一开始的时候，一般都需要先排序一下，将所有的操作按照一定的性质排起来（Cash一题是按-a[i]/b[i]从大到小排序，Mokia一题是按照x坐标从小到大排序），否则的话和直接暴力没有区别。

接着就是对所有操作进行二分，对于一个区间[l,r]，我们首先要对[l,mid]范围内的操作进行查询，所以需要将[l,r]范围内的操作重排，保证[l,mid]范围内的操作的id都<=mid，然后再对[l,mid]范围内的操作进行递归求解。

递归完之后，[l,mid]范围内的所有操作都已经按照先后的顺序排序好了，但是[mid+1,r]范围内的操作还是按照某个关键字排序。我们根据[l,mid]范围内的操作更新[mid+1,r]范围内的操作（在Cash一题中是更新它们的f[]数组，在Mokia中则要和整体二分差不多，先用[l,mid]范围内的操作模拟，再更新[mid+1,r]范围内所有查询的答案）。

更新完之后，我们再递归求解[mid+1,r]。

因为大神CDQ的论文已经讲得很详细了，网上关于CDQ分治的资料也挺多的，就不再讲了，感觉这些神奇的算法还是挺好用的，尤其是在自己懒的时候（许多高级数据结构题的代码量真是醉了……）。

三份代码见下面：

http://victorwonder.is-programmer.com/posts/70210.html?utm_source=tuicool

Meteors

```
#include <cstdio> #include <algorithm> #include <iostream> #include <cstring> #include <cstdlib> #include <utility> #include <bitset> #include <vector> #include <string> #include <stack> #include <queue> #include <ctime> #include <cmath> #include <list> #include <map> #include <set> using namespace std; typedef long long ll; typedef double D; typedef pair<int,int> pr; const int inf i=1000000010; const int N=500010; const int M=2000100; struct node{int x,y,z;}p[N]; int n,m,k,c[N],id[N],ans[N]; int g[N],to[N],nxt[N],tot; int tol[N],tor[N]; ll h[N],tmp[N],cur[N]; void read(int &a) {  char ch; while (!((ch=getchar())>='0'&&ch<='9'));  a=ch-'0'; while ((ch=getchar())>='0'&&ch<='9') (a*=10)+=ch-'0'; } void add(int pos,int x) {while (pos<=m) h[pos]+=x,pos+=(pos&-pos);} void adddt(int x,int y,int z) {add(x,z); add(y+1,-z);} ll sum(int pos) {ll t=0; while (pos>0) t+=h[pos],pos-=(pos&-pos);return t;} void addop(int x,int y,int z,int i) {p[i].x=x; p[i].y=y; p[i].z=z;} void addpt(int x,int y) {to[++tot]=y; nxt[tot]=g[x]; g[x]=tot;} void solve(int head,int tail,int l,int r) {  if (head>tail) return;  int i,k,x,mid=(l+r)>>1,lnum=0,rnum=0;  if (l==r)  {  for (i=head;i<=tail;i++) ans[id[i]]=l;  return;  }  for (i=l;i<=mid;i++)  {  if (p[i].x<=p[i].y) adddt(p[i].x,p[i].y,p[i].z);  else adddt(p[i].x,m,p[i].z),adddt(1,p[i].y,p[i].z);  }  for (i=head;i<=tail;i++) {  tmp[id[i]]=0;  for (k=g[id[i]];k;k=nxt[k])  {  tmp[id[i]]+=sum(to[k]);  if (tmp[id[i]]+cur[id[i]]>c[id[i]]) break;  }  if (cur[id[i]]+tmp[id[i]]>=c[id[i]]) tol[++lnum]=id[i];  else tor[++rnum]=id[i],cur[id[i]]+=tmp[id[i]];  }  for (i=l;i<=mid;i++)  {  if (p[i].x<=p[i].y) adddt(p[i].x,p[i].y,-p[i].z);  else adddt(p[i].x,m,-p[i].z),adddt(1,p[i].y,-p[i].z);  }  for (i=0;i<lnum;i++) id[head+i]=tol[i+1];  for (i=0;i<rnum;i++) id[head+lnum+i]=tor[i+1];  solve(head,head+lnum-1,l,mid);  solve(head+lnum,tail,mid+1,r); } int main() {  int i,x,y,z;  scanf("%d%d",&n,&m);  for (i=1;i<=m;i++) {  scanf("%d",&x);  addpt(x,i);  }  for (i=1;i<=n;i++)  {  scanf("%d",&c[i]);  id[i]=i;  }  scanf("%d",&k);  for (i=1;i<=k;i++)  {  scanf("%d%d%d",&x,&y,&z);  addop(x,y,z,i);  }  addop(1,m,infi,++k);  solve(1,n,1,k);  for (i=1;i<=n;i++)  if (ans[i]!=k) printf("%dn",ans[i]);  else  puts("NIE");  return 0; }
```

Cash

```
#include <cstdio> #include <algorithm> #include <iostream> #include <cstring> #include <cstdlib> #include <utility> #include <bitset> #include <vector> #include <string> #include <stack> #include <queue> #include <ctime> #include <cmath> #include <list> #include <map> #include <set> using namespace std; typedef long long ll; typedef double D; typedef pair<int,int> pr; const int inf=2147483647; const int N=100010; struct node{D a,b,r,k;}p[N]; struct func{D x,y;}f[N],v[N]; D ans[N],S; int n,id[N],s[N]; bool operator < (func a,func b) {return a.x<b.x||a.x==b.x&&a.y<b.y;} bool cmp(const int &a,const int &b) {return p[a].k>p[b].k;} D cross(func a,func b,func c) {return (b.x-a.x)*(c.y-b.y)-(b.y-a.y)*(c.x-b.x);} D calc(func t,int i) {return t.x*p[i].a+t.y*p[i].b;} void solve(int l,int r,D maxnum) { if (l==r) { ans[l]=max(ans[l],maxnum); f[l].y=ans[l]/(p[l].a*p[l].r+p[l].b); f[l].x=f[l].y*p[l].r; return; } int i,mid=(l+r)>>1,t1=l,t2=mid+1,t=0,h=0; for (i=l;i<=r;i++) if (id[i]<=mid) s[t1++]=id[i]; else s[t2++]=id[i]; memcpy(id+l,s+l,sizeof(int)*(r-l+1)); solve(l,mid,maxnum); for (i=l;i<=mid;v[t++]=f[i++]) while (t&&cross(v[t-2],v[t-1],f[i])>=0) t--; for (i=mid+1;i<=r;i++) { while (h<t-1&&calc(v[h],id[i])<calc(v[h+1],id[i])) h++; ans[id[i]]=max(ans[id[i]],calc(v[h],id[i])); } solve(mid+1,r,ans[mid]); merge(f+l,f+mid+1,f+mid+1,f+r+1,v); memcpy(f+l,v,sizeof(func)*(r-l+1)); } int main() { int i; scanf("%d%lf",&n,&S); for (i=0;i<n;i++) { scanf("%lf%lf%lf",&p[i].a,&p[i].b,&p[i].r); p[i].k=-p[i].a/p[i].b; id[i]=i; } sort(id,id+n,cmp); solve(0,n-1,S); printf("%.3lf",ans[n-1]); return 0; }
```

Mokia

```
#include <cstdio> #include <algorithm> #include <iostream> #include <cstring> #include <cstdlib> #include <utility> #include <bitset> #include <vector> #include <string> #include <stack> #include <queue> #include <ctime> #include <cmath> #include <list> #include <map> #include <set> using namespace std; typedef long long ll; typedef double D; typedef pair<int,int> pr; const int inf=2147483647; const int N=500010; const int M=2000100; struct node{int op,x,y,z,t,id;}p[N],s[N]; int S,n,num,tot,id[N]; int h[M],ans[N]; bool cmp(const node &a,const node &b) {return a.x<b.x;} void addquery(int i,int op,int x,int y,int z,int id) { p[i].id=i; p[i].op=op; p[i].x=x; p[i].y=y; p[i].z=z; p[i].t=id; } void add(int pos,int x) {while (pos<=n) h[pos]+=x,pos+=(pos&(-pos));} ll sum(int pos) {ll t=0; while (pos>0) t+=h[pos],pos-=(pos&(-pos));return t;} void solve(int l,int r) { if (l==r) return; int mid=(l+r)>>1,i,t1=l-1,t2=mid,t=l; for (i=l;i<=r;i++) if (p[i].id<=mid) s[++t1]=p[i]; else s[++t2]=p[i]; memcpy(p+l,s+l,sizeof(node)*(r-l+1)); for (i=mid+1;i<=r;i++) if (p[i].op==2) { for (;t<=mid&&p[t].x<=p[i].x;t++) if (p[t].op==1) add(p[t].y,p[t].z); ans[p[i].t]+=sum(p[i].y)*p[i].z; } for (i=l;i<t;i++) if (p[i].op==1) add(p[i].y,-p[i].z); solve(l,mid); solve(mid+1,r); } int main() { int i,k,x,y,z,w; scanf("%d%d",&S,&n); while (~scanf("%d",&k)) { if (k==3) brea
```

```
k;          if (k==1)          {          scanf("%d%d%d",&x,&y,&z);          addquery(++t
ot,1,x,y,z,0);          } else          {          scanf("%d%d%d%d",&x,&y,&z,&w); num++;
        addquery(++tot,2,z,w,1,num);          addquery(++tot,2,x-1,y-1,1,num);          ad
dquery(++tot,2,x-1,w,-1,num);          addquery(++tot,2,z,y-1,-1,num);          }      }      so
rt(p+1,p+1+tot,cmp);      solve(1,tot);      for (i=1;i<=num;i++) printf("%dn",ans[i]);      return
 0; }
```

CDQ分治理解

cdq分治是一种特别的分治方法，它由cdq神牛于09国家集训队作业中首次提出，因此得名。

首先，cdq分治属于分治的一种。它一般只能处理非强制在线的问题，除此之外这个算法作为某些复杂算法的替代品几乎是没有缺点的。
考虑这样一个问题，有若干询问和若干修改，要求在O(nlogn)时间复杂度内回答所有的询问。（下把询问与修改统称为操作）
我们把[l,r]代表当前处理的操作的区间，即处理第l个到第r个操作，先找到区间的中间m=(l+r)/2
(1)然后对于前一半[l,m]我们先递归解决。
(2)对于所有在[l,m]内的修改操作，枚举处理它对于[m,r]内的所有操作『影响』。
(3)之后递归处理[m,r]这一区间。

复杂度分析：分治共有log(n)层，那么要求每一层都在线性的时间复杂度内完成，才能保证总时间复杂度是O(nlogn)
对于不同的题目，修改和询问是不一样的。在某些修改之间互相独立的题目下，还可以调换(2)(3)的顺序。
当然cdq分治也可以想其他的分治方法一样，巧妙利用归并排序，来实现每层O(logn)

就拿斜率优化dp来说，并且假设要用BST维护决策下凸性，那么就不得不写一个BST。想在考场上调出一个这样的程序，不说很难，也是很耗时耗力的。
如果运用cdq分治，事情就简单得多了：(1)(3)步不说，在(2)中将[l,m]所有的已经统计完的决策点排序，然后可以很方便地获得这个下凸壳，直接两个指针扫一下就可以在O(n)时间复杂度里统计出他们对[m,r]内的状态的影响了。这样做代码很短，容易在考试时候调试出来，节省时间。

好吧，我还是很不擅长描述算法...
没看懂的可以到cdq分治相关看下这篇，写得很不错

例题：

T1:BZOJ 3110: [Zjoi2013]K大数查询

网上看到各种Trie套Trie做法，显然是在坑，因为这题理论分析发现Trie套Trie无论在时间复杂度还是空间复杂度都是挂的，至于最后数据出的弱是另一回事。这题的正解应该是cdq分治。把答案当轴进行分治，对答案的操作，统计出当答案为m=(l+r)时候，每个询问的区间里>=它的数有多少个，这个用从头到尾扫一遍，然后线段树维护的方法做。

如果询问i区间[li,ri]里>=m的数<ci个那么显然，它的ans应该<m，归入0类;反之，则应该大于等于m，归入1类。

如果修改i，增加的数的值>=m那么归入1类，否则归入0类。然后我们发现，归入0类的询问i，在之后的分治中被检验到的ans必然是<=m的，那么对于每个被归入1类的修改，它所有的在区间[l[i],r[i]]内的数都会被统计进入『>=验证到的ans』的数的个数中，那么直接在ci上减去就可以了。

总时间复杂度O(nlog^2n)

cdq分治：http://ideone.com/QmgPc6

Trie套Trie：http://ideone.com/v1VeSN

T2:BZOJ 1492: [NOI2007]货币兑换Cash

斜率优化dp，相当强大的应用。这个题不同于一般的斜率优化dp，需要用bst维护凸壳。用了cdq分治之后，询问和决策就都变成离线了，直接排个序，当成普通的斜率优化dp来做，相当犀利。这里有一个特殊情况，就是对[m+1,r]也是要进行某个关键字的排序的，所以必须事先排序好并记录。

注意，这个时候时间复杂度是O(nlogn) 空间复杂度是O(nlogn)的，如果题目卡内存就不要用了，不过也没有这么sxbk的出题人吧。

http://ideone.com/ISBnao

T3:BZOJ 2527: [Poi2011]Meteors

这题不得不说有点神。大体思路和T1一样，然后就是这题最神的地方，必须要开unsigned long long才能通过本题 ....

如果程序调不出错，或许可以试试修改掉这个玩意儿。我的程序运行速度极慢= =

http://ideone.com/M99tLh

T4: Codeforces #232 div.1 C - On Changing Tree

昨天考试遇到的题目。当时写了树链剖分，然后爆栈了呵呵，没心情写手工栈导致爆零。今天突然意识到cdq分治可做，复杂度O(nlogn)，观察了一下时限3s，于是各种乱搞水过

http://ideone.com/KdMvZ0

CDQ分治题目小结

CDQ分治属于比较特殊的一类分治，许多问题转化为这类分治的时候，时空方面都会有很大节省，而且写起来没有这么麻烦。

这类分治的特殊性在于分治的左右两部分的合并，作用两部分在合并的时候作用是不同的，比如，通过左半部分的影响来更新右半部分，所以分治开始前都要按照某一个关键字排序，然后利用这个顺序，考虑一个区间[l, r]的两部分间的影响。感觉说的太多，还是不如具体题目分析，而且题目也不尽相同，记住几句话是没什么用的。

练习地址：
http://vjudge.net/contest/view.action?cid=55322#overview

Problem A HYSBZ 3110 K大数查询
这个是一个很经典的树套树题目，看别人博客发现CDQ分治能够很好处理。
题意：有n个位置编号1~n,m个操作，每个操作两种类型：1 a b c 表示将第a~b个位置之间的每个位置插入一个数c;2 a b c 查询第a~b个位置之间的所有数中，第c大的数。
范围：
N,M<=50000,N,M<=50000
a<=b<=N
1 操作中abs(c)<=N
2 操作中abs(c)<=Maxlongint
分析：
按照CDQ分治的做法，是答案当做关键字来分治，由于答案最终在-n~n之间，这里首先需要一个转化，将区间第c大变成第c小，只需要将每个数变成n-c+1。

对于这类操作类的题目 ，CDQ分治的做法首先要保证的是操作的顺序，接下来以答案为关键字，例如询问结果在L~R之间的操作2，分成两部分递归L~m，m+1~R处理，#11对于操作1如果添加的数<=m,则加入到相应的位置区间;#12否则说明操作1影响答案在右半区间m+1~R的操作2。然后对于每个操作2查询当前位置区间有多少个数，表示该区间<=m已经有多少个数（#21），如果（#22）数目tmp > c (查询数目），说明答案应该在m+1~R,否则在L~m。然后将操作1中影

响答案在左半部分的（编号#11）和操作2中答案在左半部分的（#21）集中在一起左半部分，剩下的集中在右半部分。然后递归处理答案在左半部分和右半部分的。每次进行子区间的递归时都将操作分成了2部分，表示不同区间被对应不同的操作。

具体成段增加一个值和查询某一段的和用到了树状数组，也可以用线段树，不过我觉得树状数组解法简洁有力，orz，上一下原文树状数组链接

http://www.cnblogs.com/lazycal/archive/2013/08/05/3239304.html

代码：

```
/*Time 2014 08 31 , 19:26 */ #include <bits/stdc++.h> #define in freopen("solve_in.txt", "r", stdin); #define bug(x) printf("Line %d : >>>>>>>n", (x)); using namespace std; typedef long long LL; const int maxn = 50000 + 100; LL x1[maxn][2], x2[maxn][2], ans[maxn]; int cnt; struct Node { int l, r, type; LL c; int id; } q[maxn]; int rk[maxn], t1[maxn], t2[maxn]; int n, m; LL query(LL a[][2], int x) { LL res = 0; for(; x > 0; x -= (x&(-x))) { if(a[x][0] == cnt) res += a[x][1]; } return res; } LL query(int l, int r) { return query(x1, l)*(r-l+1)+ (r+1)*(query(x1, r)-query(x1, l)) - (query(x2, r)-query(x2, l)); } void add(LL a[][2], int x, LL c) { for(; x <= n; x += ((-x)&x)) { if(a[x][0] == cnt) a[x][1] += c; else a[x][0] = cnt, a[x][1] = c; } } void add(int l, int r, int c) { add(x1, l, c); add(x2, l, (LL)l*c); add(x1, r+1, -c); add(x2, r+1, (LL)(r+1)*(-c)); } void solve(int ll, int rr, int l, int r) { if(l > r) return; if(ll == rr) { for(int i = l; i <= r; i++) if(q[rk[i]].type == 2) { ans[rk[i]] = ll; } return; } int m1 = (ll+rr)>>1, m2 = (l+r)>>1; cnt++; t1[0] = t2[0] = 0; for(int i = l; i <= r; i++) { if(q[rk[i]].type == 1) { if(q[rk[i]].c <= m1) { add(q[rk[i]].l, q[rk[i]].r, 1); t1[++t1[0]] = rk[i]; } else { t2[++t2[0]] = rk[i]; } } else { LL xx = query(q[rk[i]].l, q[rk[i]].r); if(xx < (LL)q[rk[i]].c) { q[rk[i]].c -= xx; t2[++t2[0]] = rk[i]; } else { t1[++t1[0]] = rk[i]; } } } m2 = l+t1[0]-1; for(int i = l; i <= r; i++) { if(i <= m2) { rk[i] = t1[i-l+1]; } else { rk[i] = t2[i-m2]; } } solve(ll, m1, l, m2); solve(m1+1, rr, m2+1, r); } int main() { scanf("%d%d", &n, &m); for(int i = 1; i <= m; i++) { rk[i] = i; scanf("%d%d%d%lld", &q[i].type, &q[i].l, &q[i].r, &q[i].c); if(q[i].type == 1) q[i].c = (LL)n-q[i].c+1; q[i].id = i; } solve(1, 2*n+1, 1, m); for(int i = 1; i <= m; i++) { if(q[i].type == 2) { printf("%dn", n-ans[i]+1); } } return 0; }
```

Problem B HYSBZ 1492 货币兑换Cash

题意：一开始有S元现金，接下来共有N天，每天两种货币的价格分别为a[i],b[i]，以及卖入时，ab货币的比列为r[i]，问N天结束时最多能有多少现金。

分析：

最后一天结束时一定时将货币全部换成现金，那么第i天货币数目x[i], y[i],第i天最多持有的现金f[i] = max{x[j]*a[i]+y[j]*b[i]|(j < i)}，

y[j] = f[j]/(a[j]*r[j]+b[j]), x[j] = y[j]*r[j].

化简后f[i]/b[i] - x[j]*a[i]/b[i] = y[j]，发现最优解便是使得f[i]/b[i]最大，也就是斜率为-a[i]/b[i]的斜率，截距最大。对于点(x[j], y[j])能够影响到之后的f[i]， i >j,f[i]最优解一定落在前i-1天行成的凸壳上，那么怎么高效维护这个凸壳是问题的核心，与普通斜率优化不同的是这题的斜率与x均不会单调，所以事先将斜率排序，然后按照斜率递减的顺序来在凸壳上找最优解是可行的。因为斜率递减的话，切凸壳上点得到的截距会越来越大。然后就是维护以个凸壳，最终这个凸壳相邻两点斜率也要递减。那么每次递归结束时按照x[i]排序，方便下次维护生成凸壳。

代码：

http://vjudge.net/contest/viewSource.action?id=2724881

Problem C CodeForces 396C On Changing Tree

题解见这里

http://www.cnblogs.com/rootial/p/3948478.html

关键在于两个操作1的合并， 将树的叶子结点编号形成连续区间然后当做线段树做！每次查询时只需将结点变成对应的叶子结点区间在线段树上查询就可以了。

代码：

代码贴不上来。上链接好了。

http://vjudge.net/contest/viewSource.action?id=2726148

Problem D HDU 3698 Let the light guide us

题意：

n*m的两个矩阵，每个格子有两个值，一个是花费cost[i][j]，一个是魔力值magic[i][j]，(n<=100, m<=5000)要求每行选一个格子且格子对应的花费总和最小，任意响铃两行的格子魔力值满足条件|j-k|<=f[i, j]+f[i-1, k]。

分析：

CDQ分治做法还没想出来，之后在更新吧，看大家博客基本都是线段树做法..

dp[i][j]表示第i行选第j个格子的最小的花费。

分析一下， 对于任意相邻两行[i, j]和[i-1, k]的格子，[i-1, k]的花费dp[i-1][k]能够影响下一行k-magic[i-1, k]~k+magic[i-1, k]范围内格子的花费， [i, j]能够受上一行j-magic[i,j]~j+magic[i, j]

格子的花费的影响。这样用上一行花费dp[i-1][k]更新k-magic[i-1, k]~k+magic[i-1, k]最小值，到求dp[i, j]时， 查询j-magic[i, j]~j+magic[i,j]最小值min即可， dp[i][j] = min+cost[i][j] .

代码：

```
//Time 2014 09 01 , 10:22 #include <cstring> #include <algorithm> #include <cstdio> #include <iostream> #define in freopen("solve_in.txt", "r", stdin); #define bug(x) printf("Line %d : >>>>>>> n", (x)); #define lson rt<<1, l, m #define rson rt<<1|1, m+1, r #define inf 0x0f0f0f0f #define pb push_back using namespace std; typedef long long LL;  const int maxn = 5555; const int maxm = 111; int dp[maxn]; int n, m; int a[maxm][maxn], b[maxm][maxn], cover[maxn<<2], mi[maxn<<2]; void PushDown(int rt) {     cover[rt<<1] = min(cover[rt<<1], cover[rt]);     cover[rt<<1|1] = min(cover[rt<<1|1], cover[rt]);     mi[rt<<1] = min(mi[rt<<1], cover[rt<<1]);     mi[rt<<1|1] = min(mi[rt<<1|1], cover[rt<<1|1]);     cover[rt] = inf; } void update(int rt, int l, int r, int L, int R, int c) {     if(L <= l && R >= r)     {         cover[rt] = min(cover[rt], c);         mi[rt] = min(mi[rt], cover[rt]);         return;     }     int m = (l+r)>>1;     PushDown(rt);     if(L <= m)         update(lson, L, R, c);     if(R > m)         update(rson, L, R, c);     mi[rt] = min(mi[rt<<1], mi[rt<<1|1]); } int query(int rt, int l, int r, int L, int R) {     if(L <= l && R >= r)     {         return mi[rt];     }     int m = (l+r)>>1;     int ans = inf;     PushDown(rt);     if(L<=m)         ans = min(ans, query(lson, L, R));     if(R > m)         ans = min(ans, query(rson, L, R));     return ans; } void build(int rt, int l, int r){     cover[rt] = mi[rt] = inf;     if(l == r)     {         return;     }     int m = (l+r)>>1;     build(lson);     build(rson); } int main() {     while(scanf("%d%d", &n, &m), n||m)     {         for(int i = 1; i <= n; i++)             for(int j = 1; j <= m; j++)             {                 scanf("%d", &a[i][j]);                 if(i == 1)                     dp[j] = a[i][j];             }         for(int i = 1; i <= n; i++)             for(int j = 1; j <= m; j++)             {                 scanf("%d", &b[i][j]);             }         for(int i = 2; i <= n; i++)         {             build(1, 1, m);             for(int j= 1; j <= m; j++)             {                 int L = max(1, j-b[i-1][j]);                 int R = min(m, j+b[i-1][j]);                 update(1, 1, m, L, R, dp[j]);             }             for(int j = 1; j <= m; j++)             {                 int L = max(1, j-b[i][j]);                 int R = min(m, j+b[i][j]);                 dp[j] = query(1, 1, m, L, R)+a[i][j];             }         }         cout<<*min_element(dp+1, dp+m+1)<<endl;     }     return 0; }
```

UPDATE:

问了一下CLQ终于知道CDQ是怎么搞的了。

分析一下，CDQ分治也是在转移的时候发挥作用，在需要求dp[i][j]的时候需要用上一行中dp[i-1][k]最小的来更新，且满足|j-k|<=magic[i][j]+magic[i-1][k]这个不等式， 按照CDQ分治的做法，需要将其变形为下面两个形式：

1. k > j 时， k-magic[i-1][k] <= magic[i][j]+j;

2. j > k 时， -magic[i-1][k]-k <= magic[i][j]-j;

分治的时候只需要分别考虑k > j时， dp[i-1][k]的最小值，及 j >k时， dp[i-1][k]的最小值， 然后更新就可以了。以k >j 为例，分治区间[l, r]表示相应格子的列的范围， 然后利用列坐标 >=mid+1的dp[i-1][j]去更新列坐标 j <=m的dp[i][j], 具体可以这样先将i-1行每个格子按k-magic[i-1][k]增序排列，第i行格子按照magic[i][j]+j增序排列，有了单调性对于每个dp[i][j]只需要从队首往后扫， 并记录最小值， 可以证明这样对于magic[i][j]+j排在后面的dp[i][j]值的更新也是满足最优的，找到最优值就直接更新。

对于j > k的情况类似，这样一想中间过程又和前面几个题目类似了。

代码：

```
#include <iostream> #include <cstdio> #include <algorithm>  #define pb push_back #define inf 0x0f0f0f0f #define bug(x) printf("line %d: >>>>>>>>>>>>>>>n", (x)); #define in freopen("solve_in.txt", "r", stdin); #define SZ(x) ((int)x.size()) using namespace std; typedef pair<int, int> PII; const int maxn = 5555; int dp[111][maxn], magic[maxn][maxn], f[maxn][maxn]; PII t[4][maxn], t1[maxn];  void solve(int pos, int l, int r) {     if(l == r)     {         t[0][l] = PII(l-magic[pos-1][l], l);         t[1][l] = PII(magic[pos][l]+l, l);         t[2][l] = PII(-l-magic[pos-1][l], l);         t[3][l] = PII(magic[pos][l]-l, l);         return ;     }     int mid = (l+r)>>1;     solve(pos, l, mid);     solve(pos, mid+1, r);     int res = inf;     for(int i = l, j = mid+1; i <= mid; i++)     {         while(j <= r && t[0][j].first <= t[1][i].first)         {             res = min(res, dp[pos-1][t[0][j].second]);             j++;         }         dp[pos][t[1][i].second] = min(dp[pos][t[1][i].second], res+f[pos][t[1][i].second]);     }     res = inf;     for(int i = mid+1, j = l; i <= r; i++)     {         while(j <= mid && t[2][j].first <= t[3][i].first)         {             res = min(res, dp[pos-1][t[2][j].second]);             j++;         }         dp[pos][t[3][i].second] = min(dp[pos][t[3][i].second], res+f[pos][t[3][i].second]);     }     for(int k = 0; k < 4; k++)     {         int l1 = l, l2 = mid+1;         for(int i = l; i <= r; i++)             if(l1 <= mid && (l2 > r || t[k][l1].first < t[k][l2].first))                 t1[i] = t[k][l1++];             else t1[i] = t[k][l2++];         for(int i = l; i <= r; i++) t[k][i] = t1[i];     } } int main() {     int n, m;     while(scanf("%d%d", &n, &m), n||m)     {         for(int i = 1; i <= n; i++)             for(int j = 1; j <= m; j++)                 scanf("%d", &f[i][j]);         for(int i = 1; i <= n; i++)             for(int j = 1; j <= m; j++)                 scanf("%d", &magic[i][j]);         for(int i = 1; i <= n; i++)         {             for(int j = 1; j <= m; j++)                 if(i == 1)                     dp[i][j] = f[i][j];                 else                 {                     dp[i][j] = dp[i-1][j]+f[i][j];                 }                 if(i != 1)                     solve(i, 1, m);         }         cout<<*min_element(dp[n]+1, dp[n]+m+1)<<endl;     }     return 0; }
```