逆反心理

- 求答案补集
- **时间倒流**...可以将难以维护的操作变为较容易的操作（插入和删除，分裂和合并）

数据结构更多利用的是**值域**的性质，下列方法更多利用的是**时间**的性质

众所周知，**前面的修改**会对**后面的询问**造成影响...

CDQ

动态问题，修改+询问，且不容易用数据结构同时维护，**离线！！**

如果静态版本容易（修改全部在询问前面），并且静态版本询问的复杂度只和询问本身有关，可以一试

要求修改能够分批对询问贡献！！

具体操作，考虑一个序列(C,Q,Q,C,C,Q,C,Q)，可以通过分治变成两个序列：

$(C,Q,Q,C)|(C,Q,C,Q)$

我们可以递归地解决两边子问题，以及左区间$(C,.,.,.,C)$对右区间$(.,Q,.,.,Q)$的贡献（静态）

复杂度：$O(logn)\times 查询+总大小为O(nlogn)的构建$

可以发现，CDQ实际上是在一棵时间的线段树上递归，因此遍历方式是有讲究的

常见使用场景：

- 偏序问题降维，比如模板题...陌上花开 （CDQ+数据结构）or（CDQ套CDQ）
- 必须按照顺序求解的（涉及偏序的动态规划）：中序遍历
- 需要进行归并的：后序遍历

注意事项：

- 重(chong)点
- 每一层的数据结构维护方式（清空、撤回、删除、归并）
- 遍历顺序（中序、后序）

[HEOI2016/TJOI2016] 序列 例题

长度为n的序列a_n，m个形如(x,y)的修改表示将$a[x]$修改为y。

找到最长的一个子序列，使得该子序列在任意**一种**修改下都是不降的（也可以不修改），求出该子序列的长度。

$1\le n,m,a_i,x\le 10^5$

题解

对每个位置x预处理出$min[x]$和$max[x]$

设$dp[x]$表示以位置x结尾的最长上升子序列长度

$dp[x]=max(dp[y])+1$，其中$y<x,max[y]\leq a[x],a[y]\leq min[x]$，发现是一个三维偏序问题

这就可以用树套树/CDQ解决了

`CDQ`具体做法如下：

`CDQ(l,r)`表示求完了$[l,r]$内的dp值，如何求呢

先调用`CDQ(l,mid)`，然后考虑左半部分对右半部分的贡献

将左半按照max排序，右半按照a排序，然后双指针扫，第三维用数据结构维护

然后清空数据结构、恢复右半部分为按照下标排序，并调用`CDQ(mid+1,r)`

注意清空不能使用`memset`，不然复杂度不对

$T(n)=2T(\frac{n}{2})+O(n\log n)=O(n\log^2 n)$

LuoguP3769 TATT 习题

n个点，每个点有四维坐标(a_i,b_i,c_i,d_i)，要求选出最多的点，使得他们能构成一条**任意一维都单调不降**的路径。

$1\leq n\leq 5\times 10^4$

题解

首先对于重合的点，需要将它们合并

$dp[x]=w_x+max\{dp[y]\}(a[y]\leq a[x],b[y]\leq b[x],c[y]\leq c[x],d[y]\leq d[x])$

其实就是一个裸的偏序问题，可以用CDQ套CDQ的方式解决

先按照a排序，发现求左区间对右区间贡献时并不能简单的看作降成了三维，因为如果再按照b排序，将无法保证a的相对顺序

有一个小技巧，我们把左半区间变成$(0,b,c,d)$，右半区间变成$(1,b,c,d)$，我们按照b排序后只求0对1的贡献即可这样就可以把它看作是三维的问题了

$T(n)=2T(\frac{n}{2})+O(n\log^2 n)=O(n\log^3 n)$

[CEOI2017] Building Bridges 例题

有 n 根柱子依次排列，每根柱子都有一个高度。第 i 根柱子的高度为 h_i。

现在想要建造若干座桥，如果一座桥架在第 i 根柱子和第 j 根柱子之间，那么需要 $(h_i-h_j)^2$ 的代价。

在造桥前，所有用不到的柱子都会被拆除，因为他们会干扰造桥进程。第 i 根柱子被拆除的代价为 w_i，注意 w_i 不一定非负，因为可能政府希望拆除某些柱子。

现在政府想要知道，通过桥梁把第 1 根柱子和第 n 根柱子连接的最小代价。注意桥梁不能在端点以外的任何地方相交。

$2 \le n \le 10^5; 0 \le h_i, |w_i| \le 10^6$

题解

设f_i表示只考虑前i根柱子，并且1和i连接的最小代价

$f_i = max_{j<i}\{f_j + (h_i - h_j)^2 + sw_{i-1} - sw_j\}$，展开得到

$f_i = -2h_ih_j + (f_j + h_j^2 - sw_j) + (h_i^2 + sw_{i-1})$

发现转移时，$(h_i^2 + sw_{i-1})$是常数项，因此可以只考虑前两项的影响

李超线段树做法

对每个点，看做是一个$k = -2h_i, b = f_i + h_i^2 - sw_i$的直线，将$y = kx + b$插入到李超线段树中即可

时间复杂度$O(n \log n)$

斜率优化做法

设$Y(j) = f_j + h_j^2 - sw_j, X(j) = h_j$

那么若$j_1 < j_2$且j_2更优，有式子$Y(j_2) - 2h_iX(j_2) \le Y(j_1) - 2h_iX(j_1)$

化简得$\frac{Y(j_2)-Y(j_1)}{X(j_2)-X(j_1)} \le 2h_i$，可以发现最优点一定在下凸壳上面

但是横坐标和斜率都不单调...因此需要平衡树维护凸壳，查询时在凸壳上面二分，时间复杂度$O(nlogn)$

也可以强行有单调性，考虑分治

发现右半部分对左半部分是没有贡献的，因此可以先递归解决左半部分

然后计算左半部分对右半部分的贡献，我们希望左半部分能有坐标单调、右半部分能有斜率单调的性质

因此这就要求左右区间的h_i都是单调的，如果每次分治的时候按照h_i排序，那么总时间复杂度会是两个log...

可以使用一个小trick，把每个点看作一个二元组(h_i,$0/1$)，其中$0/1$表示点属于左/右区间

我们在分治之前就将所有点按照h_i排序，在递归时再去计算他的$0/1$，这样在递归每一层就可以$O(n)$排序了

排好序之后单调队列求出对右半部分的贡献，再递归求解右半部分

最后将两个部分按照h_i归并即可，$T(n) = 2T(\frac{n}{2}) + O(n)$

总时间复杂度为$O(n \log n)$

通常CDQ分治优化1D/1D复杂度为$O(n \log^2 n)$，或者时空都$O(n \log n)$，但这题情况特殊

```
#include <cstdio>
#include <algorithm>

using namespace std;

typedef long long ll;
const int MAXN = 100010;

```

```
int n;
ll h[MAXN], s[MAXN], f[MAXN];
inline ll pw2(ll x) {return x*x;}
inline ll X(int i) {return h[i];}
inline ll Y(int i) {return pw2(h[i])+f[i]-s[i];}
struct node{
    ll x, y;
    int ind;
    node(ll X=0,ll Y=0,int I=0):x(X),y(Y),ind(I){}
    bool operator<(const node&o)const{return x ^ o.x ? x < o.x : y < o.y;}
}a[MAXN], tmp[MAXN];

int q[MAXN], he, ta;

void cdq(int l, int r)
{
    if(l == r)
    {
        a[l].y = Y(l);
        return ;
    }
    int mid = (l+r)>>1;
    int x = l, y = mid+1;
    for(int i=l;i<=r;i++)
        if(a[i].ind <= mid) tmp[x++] = a[i];
        else tmp[y++] = a[i];
    for(int i=l;i<=r;i++)
        a[i] = tmp[i];
    cdq(l, mid);
    he = 1; ta = 0;
    for(int i=l;i<=mid;i++)
    {
        if(i > l && a[i].x == a[i-1].x) continue;
        while(he < ta && (a[i].y-a[q[ta]].y)*(a[q[ta]].x-a[q[ta-1]].x) <=
(a[q[ta]].y-a[q[ta-1]].y)*(a[i].x-a[q[ta]].x))
            --ta;
        q[++ta] = i;
    }
    for(int i=mid+1;i<=r;i++)
    {
        while(he < ta && a[q[he+1]].y-a[q[he]].y <= 2*a[i].x*(a[q[he+1]].x-
a[q[he]].x)) ++he;
        int u = a[i].ind, v = a[q[he]].ind;
        f[u] = min(f[u], f[v] + s[u-1] - s[v] + pw2(h[u] - h[v]));
    }
    cdq(mid+1, r);
    x = l; y = mid+1;
    int now = l;
    while(x <= mid && y <= r)
    {
        if(a[x].x < a[y].x || (a[x].x == a[y].x && a[x].y < a[y].y)) tmp[now++]
= a[x++];
        else tmp[now++] = a[y++];
    }
    while(x <= mid) tmp[now++] = a[x++];
    while(y <= r) tmp[now++] = a[y++];
    for(int i=l;i<=r;i++) a[i] = tmp[i];
}
```

```
int main()
{
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
        scanf("%lld",h+i), a[a[i].ind = i].x = h[i];
    for(int i=1;i<=n;i++)
        scanf("%lld",s+i), s[i] += s[i-1];
    for(int i=2;i<=n;i++)
        f[i] = 0x3f3f3f3f3f3f3f3f;
    sort(a+1, a+n+1);
    cdq(1, n);
    printf("%lld\n",f[n]);
    return 0;
}
```

最小mex生成树（此题仅为普通分治）习题

给定 n 个点 m 条边的无向连通图，边权w_i，现在你要求出一个这个图的生成树，使得其边权集合的 mex 尽可能小。

$1 \le n \le 10^6, 1 \le m \le 2 \times 10^6, 0 \le w \le 10^5$

题解

首先，最大生成树是不对的，构造一个环，1条边为1，其余边为0...

有个naive的做法，考虑枚举mex的值，只要判断不加入权值为mex的边是否能连成连通块即可，但这样是$O(wn)$的

对于这种删掉一部分边的问题，我们可以考虑用分治去优化这个过程，设$solve(l,r)$表示权值在$[l,r]$内的边都**没有**加入。在递归$solve(l,mid)$时，将权值在$[mid+1,r]$的边加入，递归$solve(mid+1)$时，将权值在$[l,mid]$的边加入。发现使用可撤销并查集维护连通性可以做到每层$O(m\log n)$

总时间复杂度$O(m\log n\log w)$

[HNOI2010] 城市建设 习题

n个点m条边的图，q次操作，每次修改一条边的边权，回答经过前i次修改后，最小生成树的边权和

$1 \le n \le 2 \times 10^4, 1 \le m, q \le 5 \times 10^4, 1 \le \text{边权} \le 5 \times 10^7$

题解

一句话题意：给定一张图支持动态的修改边权，要求在每次修改边权之后输出这张图的最小生成树的最小代价和。

事实上，有一个线段树分治套 lct 的做法可以解决这个问题，但是这个实现方式的常数过大，可能需要精妙的卡常技巧才可以通过本题，因此不妨考虑 CDQ 分治来解决这个问题。

和一般的 CDQ 分治解决的问题不同，此时使用 CDQ 分治的时候并没有修改和询问的关系来让我们进行分治，因为无法单独考虑「修改一个边对整张图的最小生成树有什么贡献」。传统的 CDQ 分治思路似乎不是很好使。

通过刚才的例题可以发现，一般的 CDQ 分治和线段树有着特殊的联系：我们在 CDQ 分治的过程中其实隐式地建了一棵线段树出来（因为 CDQ 分治的递归树就是一颗线段树）。通常的 CDQ 是考虑线段树左右儿子之间的联系。而对于这道题，我们需要考虑的是父亲和孩子之间的关系；换句话来讲，我们在$solve(l,r)$这段区间的时候，如果可以想办法使图的规模变成和区间长度相关的一个变量的话，就可以解决这个问题了。

那么具体来讲如何设计算法呢?

假设我们正在构造(l,r)这段区间的最小生成树边集，并且我们已知它父亲最小生成树的边集。我们将在(l,r)这段区间中发生变化的边分别赋与$+\infty$和$-\infty$的边权，并各跑一边 kruskal，求出在最小生成树里的那些边。

对于一条边来讲：

- 如果最小生成树里所有被修改的边权都被赋成了$+\infty$，而它未出现在树中，则证明它不可能出现在(l,r)这些询问的最小生成树当中。所以我们仅仅在的(l,r)边集中加入最小生成树的树边。
- 如果最小生成树里所有被修改的边权都被赋成了$-\infty$，而它未出现在树中，则证明它一定会出现这(l,r)段的区间的最小生成树当中。这样的话我们就可以使用并查集将这些边对应的点缩起来，并且将答案加上这些边的边权。

这样我们就将(l,r)这段区间的边集构造出来了。用这些边求出来的最小生成树和直接求原图的最小生成树等价。

那么为什么我们的复杂度是对的呢?

首先，修改过的边一定会加进我们的边集，这些边的数目是$O(len)$级别的。

接下来我们需要证明边集当中不会有过多的未被修改的边。我们只会加入所有边权取$+\infty$最小生成树的树边，因此我们加入的边数目不会超过当前图的点数。

现在我们只需证明每递归一层图的点数是$O(len)$级别的，就可以说明图的边数是$O(len)$级别的了。

证明点数是$O(len)$几倍就变得十分简单了。我们每次向下递归的时候缩掉的边是在$-\infty$生成树中出现的未被修改边，反过来想就是，我们割掉了出现在$-\infty$生成树当中的所有的被修改边。显然我们最多割掉len条边，整张图最多分裂成$O(len)$个连通块，这样的话新图点数就是$O(len)$级别的了。所以我们就证明了每次我们用来跑 kruskal 的图都是$O(len)$级别的了，从而每一层的时间复杂度都是$O(n\log n)$了。

时间复杂度是$T(n)=T(\lfloor\frac{n}{2}\rfloor)+T(\lceil\frac{n}{2}\rceil)+O(n\log n)=O(n\log^2 n)$。

代码实现上可能会有一些难度。需要注意的是并查集不能使用路径压缩，否则就不支持回退操作了。执行缩点操作的时候也没有必要真的执行，而是每一层的 kruskal 都在上一层的并查集里直接做就可以了。

```
#include <algorithm>
#include <cstdio>
#include <stack>
#include <vector>
using namespace std;
typedef long long ll;
int n;
int m;
int ask;

struct bcj {
  int fa[20010];
  int size[20010];

  struct opt {
```

```
    int u;
    int v;
  };

  stack<opt> st;

  void ih() {
    for (int i = 1; i <= n; i++) fa[i] = i, size[i] = 1;
  }

  int f(int x) { return (fa[x] == x) ? x : f(fa[x]); }

  void u(int x, int y) {  // 带撤回
    int u = f(x);
    int v = f(y);
    if (u == v) return;
    if (size[u] < size[v]) swap(u, v);
    size[u] += size[v];
    fa[v] = u;
    opt o;
    o.u = u;
    o.v = v;
    st.push(o);
  }

  void undo() {
    opt o = st.top();
    st.pop();
    fa[o.v] = o.v;
    size[o.u] -= size[o.v];
  }

  void clear(int tim) {
    while (st.size() > tim) {
      undo();
    }
  }
} s, s1;

struct edge  // 静态边
{
  int u;
  int v;
  ll val;
  int mrk;

  friend bool operator<(edge a, edge b) { return a.val < b.val; }
} e[50010];

struct moved {
  int u;
  int v;
};  // 动态边

struct query {
  int num;
  ll val;
  ll ans;
```

```
} q[50010];

bool book[50010];  // 询问
vector<edge> ve[30];
vector<moved> vq;
vector<edge> tr;
ll res[30];
int tim[30];

void pushdown(int dep)  // 缩边
{
  tr.clear();  // 这里要复制一份，以免无法回撤操作
  for (int i = 0; i < ve[dep].size(); i++) {
    tr.push_back(ve[dep][i]);
  }
  sort(tr.begin(), tr.end());
  for (int i = 0; i < tr.size(); i++) {  // 无用边
    if (s1.f(tr[i].u) == s1.f(tr[i].v)) {
      tr[i].mrk = -1;
      continue;
    }
    s1.u(tr[i].u, tr[i].v);
  }
  s1.clear(0);
  res[dep + 1] = res[dep];
  for (int i = 0; i < vq.size(); i++) {
    s1.u(vq[i].u, vq[i].v);
  }
  vq.clear();
  for (int i = 0; i < tr.size(); i++) {  // 必须边
    if (tr[i].mrk == -1 || s1.f(tr[i].u) == s1.f(tr[i].v)) continue;
    tr[i].mrk = 1;
    s1.u(tr[i].u, tr[i].v);
    s.u(tr[i].u, tr[i].v);
    res[dep + 1] += tr[i].val;
  }
  s1.clear(0);
  ve[dep + 1].clear();
  for (int i = 0; i < tr.size(); i++) {  // 缩边
    if (tr[i].mrk != 0) continue;
    edge p;
    p.u = s.f(tr[i].u);
    p.v = s.f(tr[i].v);
    if (p.u == p.v) continue;
    p.val = tr[i].val;
    p.mrk = 0;
    ve[dep + 1].push_back(p);
  }
  return;
}

void solve(int l, int r, int dep) {
  tim[dep] = s.st.size();
  int mid = (l + r) / 2;
  if (r - l == 1) {  // 终止条件
    edge p;
    p.u = s.f(e[q[r].num].u);
    p.v = s.f(e[q[r].num].v);
```

```
    p.val = q[r].val;
    e[q[r].num].val = q[r].val;
    p.mrk = 0;
    ve[dep].push_back(p);
    pushdown(dep);
    q[r].ans = res[dep + 1];
    s.clear(tim[dep - 1]);
    return;
  }
  for (int i = l + 1; i <= mid; i++) {
    book[q[i].num] = true;
  }
  for (int i = mid + 1; i <= r; i++) {  // 动转静
    if (book[q[i].num]) continue;
    edge p;
    p.u = s.f(e[q[i].num].u);
    p.v = s.f(e[q[i].num].v);
    p.val = e[q[i].num].val;
    p.mrk = 0;
    ve[dep].push_back(p);
  }
  for (int i = l + 1; i <= mid; i++) {  // 询问转动态
    moved p;
    p.u = s.f(e[q[i].num].u);
    p.v = s.f(e[q[i].num].v);
    vq.push_back(p);
  }
  pushdown(dep);  // 下面的是回撤
  for (int i = mid + 1; i <= r; i++) {
    if (book[q[i].num]) continue;
    ve[dep].pop_back();
  }
  for (int i = l + 1; i <= mid; i++) {
    book[q[i].num] = false;
  }
  solve(l, mid, dep + 1);
  for (int i = 0; i < ve[dep].size(); i++) {
    ve[dep][i].mrk = 0;
  }
  for (int i = mid + 1; i <= r; i++) {
    book[q[i].num] = true;
  }
  for (int i = l + 1; i <= mid; i++) {  // 动转静
    if (book[q[i].num]) continue;
    edge p;
    p.u = s.f(e[q[i].num].u);
    p.v = s.f(e[q[i].num].v);
    p.val = e[q[i].num].val;
    p.mrk = 0;
    ve[dep].push_back(p);
  }
  for (int i = mid + 1; i <= r; i++) {  // 询问转动
    book[q[i].num] = false;
    moved p;
    p.u = s.f(e[q[i].num].u);
    p.v = s.f(e[q[i].num].v);
    vq.push_back(p);
  }
```

```
    pushdown(dep);
    solve(mid, r, dep + 1);
    s.clear(tim[dep - 1]);
    return;  // 时间倒流至上一层
  }

  int main() {
    scanf("%d%d%d", &n, &m, &ask);
    s.ih();
    s1.ih();
    for (int i = 1; i <= m; i++) {
      scanf("%d%d%lld", &e[i].u, &e[i].v, &e[i].val);
    }
    for (int i = 1; i <= ask; i++) {
      scanf("%d%lld", &q[i].num, &q[i].val);
    }
    for (int i = 1; i <= ask; i++) {  // 初始动态边
      book[q[i].num] = true;
      moved p;
      p.u = e[q[i].num].u;
      p.v = e[q[i].num].v;
      vq.push_back(p);
    }
    for (int i = 1; i <= m; i++) {
      if (book[i]) continue;
      ve[1].push_back(e[i]);
    }  // 初始静态
    for (int i = 1; i <= ask; i++) {
      book[q[i].num] = false;
    }
    solve(0, ask, 1);
    for (int i = 1; i <= ask; i++) {
      printf("%lld\n", q[i].ans);
    }
    return 0;
  }
```

线段树分治

奇怪的动态问题，有查询，插入，删除三种操作

如果没有删除操作我们会做，可以考虑使用这个方法

先考虑如何消除掉删除操作：将每个元素存在的时间段算出来，然后对每个时间开个桶，然后把元素丢到每个时间的桶里...对每个桶都做一遍里面的插入，这样既可以得到对应时间的正确状态

发现每个元素会多做$O(n)$次...，可以用线段树优化往桶里面丢的过程

将每个元素存在的时间拆成log段，往线段树上对应节点丢

在遍历线段树的时候，儿子继承父亲的状态，回溯的时候撤销儿子的影响即可

时间复杂度：$O(n\log T)\times O(\text{插入}+\text{撤销})+q\times O(\text{查询})$

空间复杂度：$O(n\log T)$

P5787 二分图 /【模板】线段树分治 习题

给出一个n个点m条边的无向图，总时长为k，每条边有一个存在时间区间，问每个时刻整个图是否是二分图

$n,k\le 10^5,m\le 10^6$

题解

如果没有删除可以直接用并查集解决，但是删除不好处理

边的存在时间为区间，考虑线段树分治

可撤回并查集维护图的连通性即可

$O(m\log k\log n)$

CF1140F Extending Set of Points

转化题意之后和模板题类似

FJOI2015 火星商店问题 例题

n个商店，某个商店某个时刻会买入权值为v的商品。若干个顾客，各有一个参数x，询问在$[l,r]$商店购买进货时间在$[tl,tr]$的商品中，$v\ \mathbf{xor}\ x$的最大值，顾客和进货数量总数为m

$n,m,v,x\le 10^5$

题解

将买商品和询问的时间区间都拆成log段，然后挂到对应的节点

问题就转化为了：查询编号在$[tl,tr]$的商店所有商品中，$v\ \mathbf{xor}\ x$的最大值，可持久化Trie维护即可

$O(nlog^2n)$

CF576E Painting Edges 习题

n个点m条边的无向图，一共k种颜色，一开始所有边都没有颜色。

定义合法状态为仅保留任意一种颜色的边，图都是二分图

q次操作，每次将一条边重新染色，如果该操作执行后仍然合法才会保留该操作

判断每次操作是否合法

$n,m,q\le 5\times 10^5,k\le 50$

题解

每次染色只会对染出的颜色造成影响，颜色总数很少，用k个并查集维护即可

考虑一条边相邻两次染色操作i和j，操作i影响的范围是$[i, j-1]$，但是操作可能是不合法的

那么我们可以看作操作i只对i时刻生效，$[i+1, j-1]$时刻的颜色需要根据i时刻是否合法来判断：

1.染上去了，颜色为i时刻的颜色

2.没染上去，颜色是上一次的颜色

因为我们是按照时间顺序进行处理的，那么只需要在叶子的时候判断即可

时间复杂度$O(n\log^2 n)$，空间复杂度$O(n\log n)$

整体二分

离线算法

某些奇怪的问题，具有答案单调性，可以二分回答询问

但是每次回答mid需要的状态不同，暴力维护状态代价**过高**

这时候可以试试整体二分，可以带修！！

贡献需要满足交换、结合、可加，修改的贡献和二分的值无关且相互独立

设操作数为n，值域为w

在二分的整个过程中，所有的操作（插入、删除、查询）都会执行$O(\log w)$次

时间复杂度：$O(\log w) \times (O(\text{插入}) + O(\text{删除}) + O(\text{查询}))$

注意事项：

1.不带修：可以维护一个指针T，表示$[min_w, T]$的状态都加入到了数据结构里，通过移动指针来维护数据结构（可以减少常数）

2.带修：修改对询问的贡献容易合并（类似CDQ）或减去，就可以将修改带着一起整体二分，可能需要将修改拆为删除+插入。注意在带修整体二分时，操作的时间顺序仍然是保持的

[国家集训队] 矩阵乘法 例题

$n \times n$的矩阵，q次询问，每次询问一个子矩阵的k小

$1 \le n \le 500, 1 \le q \le 6 \times 10^4, 0 \le a_{i,j} \le 10^9$

题解

答案一定是出现过的数值，因此不需要对值域二分，只需要将值排序之后二分下标即可

$solve(l, r, L, R)$表示：$[l, r]$是答案的区间，$[L, R]$是询问的区间

每次令$mid=(l+r)/2$，然后将$[l,mid]$的所有值插入到矩阵中，每个询问查询对应矩阵中是否存在至少k个

如果存在至少k个，说明答案大了，将询问放到左边区间，否则放到右边区间，可以用二维树状数组维护

然后递归调用$solve(l,mid,L,M)$和$solve(mid+1,r,M+1,R)$即可

注意树状数组的清空，不要用`memset`，可以用时间戳/删除操作，也可以用一个指针T表示$[1,T]$的数字都插入到了树状数组，可以减小常数

分析复杂度：

每个数会被插入删除$O(\log n)$次，每次$O(\log^2 n)$，总共$O(n^2\log^3 n)$

每个询问会查询$O(\log n)$次，每次$O(\log^2 n)$，总共$O(q\log^3 n)$

总时间复杂度$O((n^2+q)\log^3 n)$

[POI2011]MET-Meteors 流星 带修整体二分（做过了）

每次操作对序列区间加正数，回答每个位置达到$a[i]$所需最小操作数

题解

对操作时间序列进行二分

注意不能使用前缀和去维护，否则复杂度不对，需要使用和操作、询问长度相关的复杂度

可以采用右区间减去左区间贡献的方式来递归

LuoguP5163 WD与地图 习题

套路缝合题...难度3000+

「CF102354」Yet Another Convolution 习题

数学（μ反演）+整体二分，网上题解较少，2800起步吧

二进制分组

动态问题，修改+询问，且不容易用数据结构同时维护，**在线！！**

如果静态版本容易（修改全部在询问前面），并且静态版本询问的复杂度只和询问本身有关，可以一试

要求修改能够分批对询问贡献！！

复杂度：$O(\log n)\times 查询 + 总大小为O(n\log n)的构建$

注意事项：

其实要求和CDQ是一样哒~但是可以解决在线问题噜~

String Set Queries 例题

维护一个字符串集合D，支持三种操作，一共m次：

- 给出s，将s加入
- 给出s，将s删除
- 给出s，回答D中所有字符串在s中出现次数和

题目**强制在线**

$m \le 3 \times 10^5, \sum |s_i| \le 3 \times 10^5$

题解

发现如果询问都在修改后面其实很好做，AC自动机即可，可惜并不能

如果每次询问时都暴力将前面的修改暴力重构AC自动机，复杂度难以承受

又观察到其实修改对询问的贡献是独立的，没必要将所有修改都放到一个AC自动机上

可以考虑使用二进制分组，方法如下：

- 假设当前已经有了22个修改操作，$22 = 16 + 4 + 2$，也就是有3个AC自动机，上面分别有$16, 4, 2$个串，在查询时，在3个AC自动机上都做一次查询，结果加起来即可
- 考虑又加入了一个串，$23 = 16 + 4 + 2 + 1$，此时我们将新加入的串单独开一个AC自动机即可...查询时在4个自动机上查询
- 此时再加入一个串，$24 = 16 + 8$，怎么办呢？可以将$4, 2, 1$三个AC自动机都删掉，然后$(4 + 2 + 1 + 1) = 8$个串暴力重构一个AC自动机
- 也就是说，每次新加入一个串之后，我们根据新得到的二进制，将消失的二进制位对应AC自动机删除，然后暴力构造出新出现的二进制位对应AC自动机
- 整个过程类似小游戏2048，初始有一个空序列，每次在序列尾部生成一个1，生成后一直向左合并直到不变
- 时间复杂度？先考虑查询，可以发现AC自动机的数量为$\log m$个，单次查询$O(|s| \log m)$，总时间复杂度为$O(\sum |s| \log m)$；再考虑插入，可以发现每个串最多被重构$\log m$次，因为每次重构组内的个数都翻倍了，因此一个串的贡献为$O(|s| \log m)$，总时间复杂度同样是$O(\sum |s| \log m)$，有个26的常数

[SDOI2014] 向量集 习题

两个操作：1.push_back一个向量(a, b)；2.询问(l, r, x, y)，回答区间$[l, r]$内的向量和(x, y)的点积最大值

强制在线

$n \le 5 \times 10^4$

题解

$ax + by = y(a \times \frac{x}{y} + b)$

不妨设$y > 0$，对于小于0的取反处理即可

那么就是求$max\{k \times a + b\}$，因此需要维护上凸壳，然后在凸壳上二分

强制在线，用二进制分组即可，直接用线段树来实现，当一个区间满了之后就建出凸壳

卡常，需要归并凸壳

$O(n \log^2 n)$

UOJ 191. 【集训队互测2016】Unknown 习题吧

[SDOI2014] 向量集 的加强版 需要进行末尾删除

如果采用旧的策略，一次删除可能破坏一大块，复杂度不对

解决办法： 在这个区间刚满的时候不build这个区间，等到同层的下一个区间也填满的时候再build这个区间。

这样，显然每层只有至多 2 个未build的区间，查询复杂度仍然可以保证。

删除时，要越过后一个整个区间才能破坏前面已经被build的区间，复杂度仍然均摊正确。