二维偏序

算法介绍

二维偏序是这样一类问题：已知点对的序列 $(a_1,b_1),(a_2,b_2),(a_3,b_3),\cdots$ 并在其上定义某种**偏序**关系 $\prec$ ，现有点 (a_i,b_i) ，求满足 $(a_j,b_j)\prec(a_i,b_i)$ 的 (a_j,b_j) 的数量。

什么叫偏序呢？数学中讲偏序是满足自反性、反对称性和传递性的序关系。但这听上去太抽象了，事实上，在二维的情形，我们分别对两个属性定义序关系，一定能得到一种偏序关系：

$$(a_j,b_j)\prec(a_i,b_i)\overset{def}{=\!=\!=}a_j\lesseqgtr a_i\text{ and }b_j\lesseqgtr b_i$$

处理此类问题，我们有两维元素，直接不太好做。

一般来说，我们先按第一位元素排序，再用树状数组等数据结构，以第二位的值为位置插入，查询。

在数值过大时注意要离散化。

POJ 2352

题目描述

给你一张 n 颗星星的星图，第 i 颗星星的位置为 (x_i, y_i)，定义星星的等级是既不在它上面也不在它右边的星星的数量，求每个等级的星星的数量。这其实就是定义了这样的偏序关系：

$(a_j, b_j) \prec (a_i, b_i) \xlongequal{def} a_j \le a_i \text{ and } b_j \le b_i$

$n \le 15000, x_i, y_i \le 32000$

思路

这就是二维数点的板子。

直接先按第一维排序，第一维就满足了限制条件。

然后求满足限制条件的，且排序后的位置在这个点之前的点的数量。

考虑用树状数组，以第二维的值为树状数组上的位置进行插入和询问。

此题已将数据以 y 坐标为第一关键字，x 坐标为第二关键字排好了序，然后利用树状数组按顺序以值为位置插入统计答案即可。

代码实现

```cpp
#include <iostream>
#include <cstdio>
const int N = 32e3 + 5;
int n, x, s[N], ans[N];
void Add(int x, int v) {for (; x < N; x += x & -x) s[x] += v;}
int Ask(int x) {
    int ans = 0;
    for (; x; x -= x & -x) ans += s[x];
    return ans;
}
signed main() {
    scanf("%d", &n);
    for (int i = 0, x; i < n; ++i) {
        scanf("%d%*d", &x);
        ++ ans[Ask(x + 1)];
        Add(x + 1, 1);
    }
    for (int i = 0; i < n; ++i) printf("%d\n", ans[i]);
    return 0;
}
```

洛谷 P4479 [BJWC2018]第k大斜率

题目描述

在平面直角坐标系上，有 n 个**不同**的点。任意两个不同的点确定了一条直线。请求出所有斜率存在的直线按斜率**从大到小**排序后，第 k 条直线的斜率为多少。

为了避免精度误差，请输出斜率向下取整后的结果。

令 M 为所有斜率存在的直线的数量 。

$n \le 1e5$，$1 \le k \le M, |x_i|, |y_i| \le 1e8$

思路

考虑求一个斜率 k 有多少大于它的斜率

$(y_j - y_i) > k(x_j - x_i)$

$y_j - y_i > kx_j - kx_i$

$y_j - kx_j > yi - kx_i$

我们记 $t_i = y_i - kx_i$

发现对于一个斜率 k 只需要统计 $x_j \geq x_i$ 且 $t_j > t_i$ 的(i, j)对数

直接二分答案 k 即可

对于统计(i, j)个数，可以用树状数组等进行二维数点，也可以模仿归并求逆序对的方法。

代码实现

```cpp
#include <bits/stdc++.h>
#define int long long
const int N = 1e5 + 5;
int n, k;
struct point {int x, y, t, id;} p[N];
int px[N];
namespace TreeArray {
    int s[N];
    void Clear() {std::fill(s + 1, s + n + 1, 0);}
    void Update(int x) {for (; x <= n; x += x & -x) ++ s[x];}
    int Query(int x) {
        int ans = 0;
        for (; x; x -= x & -x) ans += s[x];
        return ans;
    }
};
bool Check(int x) {
    for (int i = 1; i <= n; ++i)
        p[i].t = p[i].y - x * p[i].x;
    std::sort(p + 1, p + n + 1, [](point a, point b) {
        return a.t == b.t ? a.x < b.x : a.t < b.t;
    });
    TreeArray::Clear();
    int rankk = 0;
    for (int i = 1; i <= n; ++i) {
        rankk += TreeArray::Query(p[i].id - 1);
        TreeArray::Update(p[i].id);
    }
    return rankk >= k;
}
signed main() {
    scanf("%lld%lld", &n, &k);
    for (int i = 1; i <= n; ++i)
        scanf("%lld%lld", &p[i].x, &p[i].y), px[i] = p[i].x;
    std::sort(px + 1, px + n + 1);
    int m = std::unique(px + 1, px + n + 1) - px - 1;
    for (int i = 1; i <= n; ++i) p[i].id = std::lower_bound(px + 1, px + m + 1,
p[i].x) - px;
    int l = -2e8, r = 2e8, ans;
    while (l <= r) {
        int mid = (l + r) >> 1;
        if (Check(mid)) ans = mid, l = mid + 1;
        else r = mid - 1;
    } return printf("%lld\n", ans), 0;
}
```

洛谷 P3431

题目描述

给定一个 $n \times m$ 的网格，有 k 个点有权值，第 i 个点的权值为 a_i。

你从 $(1,1)$ 走到 (n,m) ，每次只能从 (x,y) 走到 $(x+1,y)$ 或 $(x,y+1)$。

求你经过的路径所获得的最大权值和。

$n,m,k \le 1e5$

思路

设 $f_{i,j}$ 为 (i,j) 时的答案最大值，转移肯定枚举一个最大的 $f_{k,l}(k \le i, l \le j)$ 再加上这个点的贡献。

考虑求的最大值是在最下脚的子矩阵，也就是满足

$(x_j, y_j) \prec (x_i, y_i) \overset{def}{=\!=\!=} x_j \le x_i \text{ and } y_j \le y_i$

w_i 最大的一个即为我们需要。

我们只需要对于上面的树状数组稍加改动，改为查询最大值。

那么我们就可以做到维护矩阵最大值，从而 `AC` 本题。

注:由于 x, y 过大，而点数很少，故而需要离散化。

代码实现

```cpp
#include <bits/stdc++.h>
#define int long long
static constexpr int N = 1e5 + 5;
int n, m, k, cnt, ans, tmp;
int b[N], s[N];
struct node {int x, y, num;} a[N];
int lowbit(int x) {return (x) & (-x);}
int add(int x, int v) {for (; x < N; x += x & -x) s[x] = std::max(s[x], v);}
int ask(int x) {
    int ans = 0;
    for (; x; x -= lowbit(x)) ans = std::max(ans, s[x]);
    return ans;
}
signed main() {
    scanf("%lld%lld%lld", &n, &m, &k);
    for (int i = 1; i <= k; ++i) {
        scanf("%lld%lld%lld", &a[i].x, &a[i].y, &a[i].num);
        b[i] = a[i].y;
    }
    std::stable_sort(a + 1, a + k + 1, [&](node a, node b) {return a.x != b.x ? a.x < b.x : a.y < b.y;});
    std::stable_sort(b + 1, b + k + 1);
    cnt = std::unique(b + 1, b + k + 1) - b - 1;
    for (int i = 1; i <= k; ++i) {
        a[i].y = std::lower_bound(b + 1, b + cnt + 1, a[i].y) - b;
        int res = ask(a[i].y) + a[i].num;
        ans = std::max(ans, res);
        add(a[i].y, res);
    }
    return printf("%lld\n", ans), 0;
}
```

GYM103687F

题目描述

给定一个排列 p 。

定义 $A[i]$ 为 $[1, i-1]$ 中 小于 $p[i]$ 的元素个数， $B[i]$ 为 $[i+1, n]$ 中小于 $p[i]$ 的元素个数。

定义整个排列的贡献为 $\sum_{i=1}^{n} min(A[i], B[i])$ 。

现在给出 m 次操作，每次操作，给出 x, y 交换排列中 $p[x], p[y]$ 。**每次操作独立。**(操作独立就是每次操作后会还原回去)

问每次操作后整个排列的贡献是多少。

$n \leq 1e5, m \leq 2e5$

思路

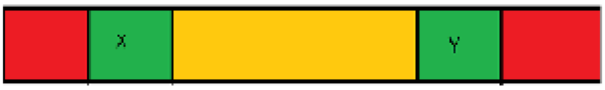
考虑计算原先的贡献。

$A[i]$ 实际上就是求顺序对数量。

由于排列的性质，所以 $A[i]+B[i]=p[i]-1$

那么可以轻松得出 $B[i]$

思考交换后的贡献



可以发现对于 x 左边和 y 右边是没有改变的。

那么也就是区间 $[x,y]$ 的贡献可能会变化。

分一下几种情况讨论一下:

1. $p_i \le \min(p[x],p[y])$

   此时由于两个都是大于它的，所以 A,B 数组均无贡献

2. $p_i \ge \max(p[x],p[y])$

   此时由于两个都是小于它的，所以调换后依旧均小于，无改变

3. $\min(p[x],p[y]) \le p_i \le \max(p[x],p[y])$

   那么我们先钦定 $p[x] \le p[y]$

   发现 $A[i]$ 减去了一

   而 $B[i]$ 加上了一

   那么哪些数的贡献产生了变化呢?

   - $A[i]=B[i]$ 贡献减一 $A[i] \to A[i]-1$
   - $A[i]+1=B[i]$ 贡献减一 $A[i] \to A[i]-1$
   - $A[i]=B[i]+1$ 贡献不变 $B[i] \to B[i]$
   - $A[i]+2 \le B[i]$ 贡献减一 $A[i] \to A[i]-1$
   - $A[i] \ge B[i]+2$ 贡献减一 $B[i] \to B[i]+1$

   所以只需要快乐地建立 5 个二维数点，就可以解决了。

注意前面我们并未包含 x,y 的贡献变化，所以我们直接减去原来的值 $A[x],B[x],A[y],B[y]$

然后再加上新的值，即 $A[x]$ 增加 $[x,y]$ 中小于 $p[x]$ 的个数，以此类推 $\cdots$

最后直接全程离线即可。

代码实现

```cpp
#include <bits/stdc++.h>
using pii = std::pair<int, int>;
static constexpr int N = 2e5 + 5;
struct tp {
    int n = 0, m = 0, s[N];
    void Update(int x, int v) {for (; x < N; x += x & -x) s[x] += v;}
    int Query(int x) {
        int ans = 0;
        for (; x; x -= x & -x) ans += s[x];
        return ans;
    }
    pii v[N];
    struct node{
        int x, y, id, type;
        friend bool operator < (node a, node b) {return a.x < b.x;}
    } q[N << 2];
    int cnt = 0;
    void add(int x, int y) {v[++ n] = pii(x, y);}
    void que(int x1, int y1, int x2, int y2, int i) {
        q[++cnt] = {x2, y2, i, 1};
        q[++cnt] = {x1 - 1, y2, i, -1};
        q[++cnt] = {x2, y1 - 1, i, -1};
        q[++cnt] = {x1 - 1, y1 - 1, i, 1};
        ++ m;
    }
    void get(int *ans) {
        std::stable_sort(v + 1, v + 1 + n);
        std::stable_sort(q + 1, q + 1 + cnt);
        int u = 1;
        for (int i = 1; i <= cnt; i++) {
            while (v[u].first <= q[i].x && u <= n) Update(v[u ++].second, 1);
            ans[q[i].id] += q[i].type * (Query(q[i].y));
        }
        for (int i = 0; i < N; i++) ans[i] = std::max(ans[i], 0);
    }
};
tp tree, t1, t2, t3, t4, t5, xx, yy;
int n, m, sum, p[N], a[N], b[N], x[N], y[N], ans[N];
int ans1[N], ans2[N], ans3[N], ans4[N], ans5[N], ansxx[N], ansyy[N];
int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i) scanf("%d", p + i);
    for (int i = 1; i <= n; ++i) {
        a[i] = tree.Query(p[i]);
        b[i] = p[i] - 1 - a[i];
        tree.Update(p[i], 1);
        sum += std::min(a[i], b[i]);
        xx.add(i, p[i]);
        yy.add(i, p[i]);
    }
```

```
    for (int i = 1; i <= n; ++i) {
        if (a[i] == b[i]) t1.add(i, p[i]);
        if (a[i] + 1 == b[i]) t2.add(i, p[i]);
        if (a[i] == b[i] + 1) t3.add(i, p[i]);
        if (a[i] + 2 <= b[i]) t4.add(i, p[i]);
        if (a[i] >= b[i] + 2) t5.add(i, p[i]);
    }
    scanf("%d", &m);
    for (int i = 1; i <= m; ++i) {
        scanf("%d%d", x + i, y + i);
        if (x[i] > y[i]) std::swap(x[i], y[i]);
        int l = x[i] + 1, r = y[i] - 1;
        int low = std::min(p[x[i]], p[y[i]]);
        int high = std::max(p[x[i]], p[y[i]]);
        t1.que(l, low, r, high, i);
        t2.que(l, low, r, high, i);
        t3.que(l, low, r, high, i);
        t4.que(l, low, r, high, i);
        t5.que(l, low, r, high, i);
        xx.que(x[i], 1, y[i], p[x[i]] - 1, i);
        yy.que(x[i], 1, y[i], p[y[i]] - 1, i);
    }
    t1.get(ans1); t2.get(ans2); t3.get(ans3); t4.get(ans4);
    t5.get(ans5); xx.get(ansxx); yy.get(ansyy);
    for (int i = 1; i <= m; i++) {
        ans[i] = sum - ans1[i];
        ans[i] -= (p[x[i]] < p[y[i]] ? ans2[i] : ans3[i]);
        ans[i] -= (p[x[i]] < p[y[i]] ? 1 : -1) * (ans4[i] - ans5[i]);
        ans[i] -= std::min(a[x[i]], b[x[i]]);
        ans[i] -= std::min(a[y[i]], b[y[i]]);
        ans[i] += std::min(a[x[i]] + ansxx[i], b[x[i]] - ansxx[i]);
        ans[i] += std::min(a[y[i]] - ansyy[i], b[y[i]] + ansyy[i]);
    }
    for (int i = 1; i <= m; i++) printf("%d\n", ans[i]);
    return 0;
}
```

CF1548E

题目描述

给定两个长度分别为 n, m 的序列 a, b 和一个参数 x

生成一个 $n \times m$ 的黑白矩阵，(i, j) 为黑当且仅当 $a_i + b_j \leq x$

求矩阵内黑色连通块数

$n, m, a_i, b_i \leq 2 \times 10^5$

思路

注意到我们把一个连通块的代表元放到这个连通块 $a_i + b_j$ 最小的一个点上，那么我们就只要求出代表元的数量。

对于 (i,j)，我们记 pre_i 表示左边第一个 a_j 不大于它的，suf_i 表示右边第一个 a_j 小于它的。那么 (i,j) 能作为代表元当且仅当它走不到 pre_i 和 suf_i 。

上面我们讨论了行的情况，列的情况也同理。

我们记 $max_a[l,r]$ 表示第 l 行到第 r 行的 a_i 最大值，max_b 同理。

为了方便起见，我们令 $na = \min(max_b[preb_j, j], max_b[j, sufb_j])$，$nb$ 同理

那么我们要求的就转化为了

$a_i + b_j \leq x$

$a_i + nb_j > x$

$b_j + na_i > x$

对于固定的 i 求满足的 j 的个数，这不是妥妥的二维偏序，扫描线和树状数组就可以了。

具体的，我们先用单调栈求出 $prea, preb, sufa, sufb$。

然后我们构造出一对 $(na_i - a_i, a_i)$ $(nb_i - b_i, b_i)$ 表示坐标。

我们可以构造两颗二叉搜索树 Ta, Tb，将点对递减排序。

如果现在是 a_i 的对，那么加入处于 $[na_i - a_i, a_i]$ 的答案，并将 a_i 加入 T_a

对于 b_i 的对差不多，只是将查询答案改成插入。

可是这样太麻烦了有木有。

注意到 $a_i, b_i \leq 2 \times 10^5$

我们可以将点对全部塞进 `vector` 里面，以其中的一维作为下标。

然后由于点对的递减排序后考虑，我们直接按照值域从大到小枚举。

如果是 a_i 那就加上答案。

如果是 b_i 那就插入树状数组。

时间复杂度 $O(n \log n)$

代码实现

```cpp
#include <bits/stdc++.h>
#define int long long
static constexpr int N = 2e5 + 5;
int n, m, x, top, ans, a[N], b[N], s[N], st[N], max[N];
int prea[N], sufa[N], preb[N], sufb[N], na[N], nb[N];
std::vector<int> qa[N], qb[N];
auto add(int x, int v) -> void {for (; x < N; x += x & -x) s[x] += v;}
auto ask(int x) -> int {int ans = 0; for (; x; x -= x & -x) ans += s[x]; return ans;}
auto Calc(int nn, int *a, int *b, int type) -> void {
    top = 1;
    for (int i = type ? nn : 1; type ? i : i <= nn; type ? --i : ++i) {
        while (a[i] - type < a[st[top - 1]]) {
            max[top - 2] = std::max(max[top - 2], max[top - 1]);
            -- top;
        }
        b[i] = std::max(max[top - 1], a[i]);
        st[top] = i, max[top ++] = a[i];
    }
}
auto main() -> decltype(0) {
    scanf("%lld%lld%lld", &n, &m, &x);
    for (int i = 1; i <= n; ++i) scanf("%lld", a + i);
    for (int i = 1; i <= m; ++i) scanf("%lld", b + i);
    max[0] = N - 1, Calc(n, a, prea, 0), Calc(m, b, preb, 0);
    st[0] = n + 1, Calc(n, a, sufa, 1);
    st[0] = m + 1, Calc(m, b, sufb, 1);
    for (int i = 1; i <= n; ++i) na[i] = std::min(prea[i], sufa[i]);
    for (int i = 1; i <= m; ++i) nb[i] = std::min(preb[i], sufb[i]);
    for (int i = 1; i <= n; ++i) if (a[i] <= x) qa[x - a[i]].push_back(i);
    for (int i = 1; i <= m; ++i) qb[nb[i]].push_back(i);
    for (int i = N - 1; i; --i) {
        for (int j : qa[i]) ans += ask(x - a[j]) - ask(std::max(x - na[j], 0ll));
        for (int j : qb[i]) add(b[j], 1);
    } return printf("%lld\n", ans), 0;
}
```

三维偏序

算法介绍

三维偏序就是在二维偏序的基础上加上一维。

给你一个长度为 n 的序列，每个元素都有 a，b，c 三种属性，让你求具有某些关系的点对 (i,j) 个数。一般思路是先确定第一维的顺序，分治维护第二维，再结合上边的二维偏序中的树状数组维护第三维。

具体的分为以下几步：

1. 找到区间中点 mid
2. 将所有点对划分为3类
    - $1 \le i,j \le mid$
    - $1 \le i \le mid < j \le n$
    - $mid+1 \le i,j \le n$

   可以发现对于 $1,3$ 两类都可以再分治的递归中解决，也就是我们在单个分治中只要解决第二类。
3. 通过双指针 + 树状数组等骚操作解决第二类。

至于如何操作可以具体看题目，一般性的有维护和，最小最大值等。

其实还有一种树套树写法，不过相对于 `cdq分治` 代码量要大，但比较无脑，对于强制在线的题，我们不能使用 `cdq分治` 时，一般套用 `树套树`,而不是用 `KDT` （因为复杂度较大，为 $O(n^{5/3})$ ）

树套树的话，对于外层的树每个节点为一颗内层的树，代表的是第二维的一个区间，内层的树，代表的是第三维的一个区间。

其实内外两棵树写法相似，所以思维难度相对较低。

CQD分治解决三维偏序

我们假设现在每个元素有3维，分别为 a_i,b_i,c_i。

先仿照二维偏序的思维，对第一维 a_i 排序。

此时一定保证对于任意的 $i \le j$ 都有 $a_i \le a_j$。

PS: 对于排序的时候我们考虑到 a_i 相等的细节

1. 若偏序关系中没有等号，则需要对第 2/3 维降序排序。因为 a_i 相等的这数是互不构成偏序关系的，所以要钦定 $b_i \ge b_j$ 这样的。
2. 若有等号，则需要对第 2/3 维升序排序。道理同上。
3. `cdq分治` 的过程中, 我们保证了左边的 b 是升序的且右边的 b 也是升序的。所以可以利用扫描线 + 树状数组或者继续套 `cdq`。对左边的 c 插入,右边查询 c 来计算贡献。最后记得**撤销**树状数组的贡献。

对于计算时间复杂度，我们考虑设 $T(n)=2T(\frac{n}{2})+n\log n$

应用主定理，发现 $f(n)=\Theta(n\log n)=\Theta(n^{\log_b a}\log^k n)$ with $k \ge 0$

那么复杂度就为 $T(n)=\Theta(n^{log_b a}\log^{k+1} n)=\Theta(n\log^2 n)$

```cpp
#include <bits/stdc++.h>
const int N = 2e5 + 5;
int n, d;
int f[N], s[N], b[N], idx[N];
struct node {
    int x, y, z, id;
    node(int x = 0, int y = 0, int z = 0, int id = 0):
        x(x), y(y), z(z), id(id) {}
} a[N];
void add(int x, int v) {
    for (; x <= d; x += x & -x) s[x] += v;
}
int ask(int x) {
    int ans = 0; for (; x; x -= x & -x) ans += s[x]; return ans;
}
void cdq(int l, int r) {
    if (l == r) return;
    int mid = l + r >> 1;
    cdq(l, mid);
    cdq(mid + 1, r);
    std::sort(a + l, a + r + 1, [&](node a, node b) {
        return a.y != b.y ? (a.y < b.y) : (a.z != b.z ? a.z < b.z : a.x < b.x);
    });
    // 在[l,r]内按y为第一维关键字，z为第二关键字，x为第三关键字排序。
    // 此时保证第二维有序，我们只需要对于x有序时对z统计贡献即可
    for (int i = l; i <= r; ++i) {
        if (a[i].x <= mid)
            add(a[i].z, 1);
        // 对 x <= mid 时插入 保证了 x 的偏序关系
        else
            b[a[i].id] += ask(a[i].z);
        // 对 x > mid 时查询
    }
    for (int i = l; i <= r; ++i)
        if (a[i].x <= mid)
            add(a[i].z, -1);
    // 撤销树状数组的贡献
}
signed main() {
    std::cin >> n >> d;
    for (int i = 1; i <= n; ++i) {
        std::cin >> a[i].x >> a[i].y >> a[i].z;
        a[i].id = i;
    }
    std::sort(a + 1, a + 1 + n, [&](node a, node b) {
        return a.x != b.x ? a.x < b.x : (a.y != b.y ? a.y < b.y : a.z < b.z);
    });
    for (int i = 1, j; i <= n;) {
        j = i + 1;
        while (j <= n && a[i].x == a[j].x && a[i].y == a[j].y && a[i].z == a[j].z)
            ++ j;
        while (i < j) idx[a[i].id] = a[j - 1].id, ++i;
    }
```

```cpp
    // 考虑到三元组相等的情况，我们直接离散掉。
    for (int i = 1; i <= n; ++i) a[i].x = i;
    cdq(1, n);
    for (int i = 1; i <= n; ++i) ++ f[b[idx[a[i].id]]];
    for (int i = 0; i < n; ++i) std::cout << f[i] << std::endl;
    return 0;
}
```

洛谷 P3157 [CQOI2011]动态逆序对

题目描述

对于序列 a，它的逆序对数定义为集合 $\{(i,j)|i<j \wedge a_i > a_j\}$ 中的元素个数。

现在给 $1 \sim n$ 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。

$n \leq 1e5, m \leq 5e4$

思路

考虑多少点 j 对 i 有贡献 ？

那一定是满足 $T_i < T_j, a_i < a_j, pos_i > pos_j$。

或者 $T_i < T_j, a_i > a_j, pos_i < pos_j$ 。

也就是对于你普通求逆序对的二维数点多加上了时间一维。

注意到偏序关系中的 $pos_i < pos_j$ 之类的，与前面的符号是相反的。

剩下的就是裸的三维数点了，直接上板子（树套树）。

代码实现

```cpp
#include <bits/stdc++.h>

using ll = long long;

const int N = 1e5 + 5, M = 20;

int n, m, x, cnt;
ll ans;
int a[N], pos[N];
int ls[N * 400], rs[N * 400], tr[N * 400];

namespace tree {

    void modify(int &rt, int x, int l, int r, int w) {
        if (!rt) rt = ++ cnt;
        tr[rt] += w;
        if (l == r) return;
        int mid = l + r >> 1;
        if (x <= mid) modify(ls[rt], x, l, mid, w);
        else modify(rs[rt], x, mid + 1, r, w);
    }

    int query(int &rt, int L, int R, int l, int r) {
        if (!rt) return 0;
        if (l <= L && R <= r) return tr[rt];
        int mid = L + R >> 1, ans = 0;
        if (l <= mid) ans += query(ls[rt], L, mid, l, r);
        if (r > mid) ans += query(rs[rt], mid + 1, R, l, r);
        return ans;
    }

    void update(int x, int p, int w) {
        for (; x <= n; x += x & -x) modify(x, p, 1, n, w);
    }

    int ask(int l, int r, int L, int R) {
        int ans = 0;
        if (l > r || L > R) return 0; -- l;
        for (; r; r -= r & -r) ans += query(r, 1, n, L, R);
        for (; l; l -= l & -l) ans -= query(l, 1, n, L, R);
        return ans;
    }
}

signed main(int argc, char *argv[]) {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr); std::cout.tie(nullptr);

    std::cin >> n >> m; cnt = n;
    for (int i = 1; i <= n; ++i) {
```

```
        std::cin >> a[i]; pos[a[i]] = i;
        tree::update(i, a[i], 1);
        ans += tree::ask(1, i - 1, a[i] + 1, n);
    }
    for (int i = 1; i <= m; ++i) {
        std::cin >> x;
        std::cout << ans << std::endl;
        if (i == m) return 0;
        x = pos[x];
        tree::update(x, a[x], -1);
        ans -= tree::ask(1, x - 1, a[x] + 1, n);
        ans -= tree::ask(x + 1, n, 1, a[x] - 1);
    } return 0;
}
```

BZOJ 3489

题目描述

给出一个长度为 n 的序列，给出 m 个询问：

在 $[l, r]$ 之间找到一个在这个区间间只出现过一次的数，并且要求找的这个数尽可能大。

如果找不到这样的数，则直接输出 0。强制在线。

$1 \leq n, a_i \leq 10^5, 1 \leq m \leq 2 \times 10^5$

思路

一个区间 $[l,r]$ 一个数仅出现一次，当且仅当 $pre_i < l, r < next_i$

那么我们要求的就是 $pre_i < l, next_i > r, l < i < r$ 的最大 a_i 了。

这不妥妥的二维数点吗。

注意到强制在线，求的还是最大值，所以直接上 `可持久化树套树`。

不用担心码量太大，因为没有修改等操作，也就是两遍主席树稍加修改。

具体的，我们先按 pre_i 进行排序，然后外围一颗主席树维护 $next_i$ 的区间，每个节点是一颗内层的主席树维护 i 的区间。

每次查询，我们需要二分出最大的 p 使得 $pre_p \leq l$。

那么我们查询 p 版本的外围主席树，然后查询 $[r+1, n+1]$ 区间，囊括 $n+1$ 是为了考虑后继没有得情况，同样还有前缀没有赋为 0 的情况。

时空复杂度均为 $O(n\log^2 n)$

代码实现

```cpp
#include <bits/stdc++.h>
 static constexpr int N = 1e5 + 5, M = 2e6 + 5, K = 4e7 + 5;
 int n, m, ans, cnt, tot;
 int last[N], root1[N], son1[M][2], root2[M], son2[K][2], tr[K];
struct Node {
    int w, pre, nxt, id;
    friend bool operator < (const Node &a, const Node &b) {return a.pre < b.pre;}
    friend bool operator < (const Node &a, const int &b) {return a.pre < b;}
} a[N];
void Insert2(int &rt, int old, int l, int r, int x, int w) {
    if (!rt) rt = ++ tot;
    tr[rt] = std::max(tr[old], w);
    if (l == r) return;
    int mid = (l + r) >> 1;
    if (x <= mid) son2[rt][1] = son2[old][1], Insert2(son2[rt][0], son2[old][0], l, mid, x, w);
    else son2[rt][0] = son2[old][0], Insert2(son2[rt][1], son2[old][1], mid + 1, r, x, w);
}
void Insert1(int &rt, int old, int l, int r, int nxt, int x, int w) {
    if (!rt) rt = ++ cnt;
    Insert2(root2[rt], root2[old], 0, n + 1, x, w);
    if (l == r) return;
    int mid = (l + r) >> 1;
    if (nxt <= mid) son1[rt][1] = son1[old][1], Insert1(son1[rt][0], son1[old][0], l, mid, nxt, x, w);
    else son1[rt][0] = son1[old][0], Insert1(son1[rt][1], son1[old][1], mid + 1, r, nxt, x, w);
}
int Query2(int rt, int l, int r, int ql, int qr) {
    if (!rt) return 0;
    if (ql <= l && r <= qr) return tr[rt];
    int mid = (l + r) >> 1, ans = 0;
    if (ql <= mid) ans = std::max(ans, Query2(son2[rt][0], l, mid, ql, qr));
    if (qr > mid) ans = std::max(ans, Query2(son2[rt][1], mid + 1, r, ql, qr));
    return ans;
}
int Query1(int rt, int l, int r, int ql, int qr, int L, int R) {
    if (!rt) return 0;
    if (ql <= l && r <= qr) return Query2(root2[rt], 0, n + 1, L, R);
    int mid = (l + r) >> 1, ans = 0;
    if (ql <= mid) ans = std::max(ans, Query1(son1[rt][0], l, mid, ql, qr, L, R));
    if (qr > mid) ans = std::max(ans, Query1(son1[rt][1], mid + 1, r, ql, qr, L, R));
    return ans;
}
signed main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &a[i].w);
```

```
        a[i].pre = last[a[i].w]; a[i].nxt = n + 1;
        a[a[i].pre].nxt = i; last[a[i].w] = i; a[i].id = i;
    } std::stable_sort(a + 1, a + n + 1);
    for (int i = 1; i <= n; ++i) Insert1(root1[i], root1[i - 1], 0, n + 1, a[i].nxt, a[i].id, a[i].w);
    for (int i = 1, x, y, l, r, p; i <= m; ++i) {
        scanf("%d%d", &x, &y);
        l = std::min((x + ans) % n + 1, (y + ans) % n + 1);
        r = std::max((x + ans) % n + 1, (y + ans) % n + 1);
        p = std::lower_bound(a + 1, a + n + 1, l) - a - 1;
        ans = Query1(root1[p], 0, n + 1, r + 1, n + 1, l, r);
        printf("%d\n", ans);
    } return 0;
}
```

LOJ 3030 「JOISC 2019 Day1」考试

题目描述

有 N 名学生参加了一个考试，考试包括数学和信息学两个科目。第 i 名学生在数学中获得了 S_i 分，在信息学中获得了 T_i 分。`T` 教授和 `I` 教授将根据每名学生的得分决定是否挂他们的科。

- `T` 教授认为两门科目的得分都很重要，他认为只有数学考了至少 A 分，且信息学考了至少B 分的学生可以通过考试。
- `I` 教授认为只有总得分才是重要的，他认为两门科目的总得分至少为 C 分的学生可以通过考试。
- 只有按两名教授的标准都通过的学生才真的不会挂科。

现在给出 Q 组询问，每组询问的内容是，给定一个三元组 (X_j, Y_j, Z_j)，当 $A = X_j, B = Y_j, C = Z_j$ 时有多少学生能不挂科，你需要对每一组询问分别进行回答。

$n, q \le 1e5, 0 \le S_i, T_i, X_i, Y_i \le 1e9, 0 \le Z_i \le 2e9$

思路

考虑数学至少 A 分，信息学至少 B 分，总分至少 C 分，那么我们只需要记 $C_i = S_i + T_i$，然后对于三元组 (Si, Ti, Ci) ，我们需要满足 $S_i \geq X, T_i \geq Y, C_i \geq z$

那么这就是一个经典的三维数点的模型，只要先对数值进行离散化后跑 `cdq` 分治即可。

具体的，我们运用 `cqd分治`，将询问和学生得分离线下来，按照上述思路

1. 找到区间中点 mid
2. 将所有点对划分为3类
   - $1 \leq i, j \leq mid$
   - $1 \leq i \leq mid$ < $j \leq n$
   - $mid + 1 \leq i, j \leq n$

   可以发现对于 $1, 3$ 两类都可以再分治的递归中解决，也就是我们在单个分治中只要解决第二类。
3. 通过双指针 + 树状数组解决第二类。

就可以快乐地 `AC` 本题啦。

这题主要是考察了给定两维的信息，要维护成三维。

这题的合并相对简单且明了，其他的题会有更为隐式的此类转化。

代码实现

```cpp
#include <bits/stdc++.h>
static constexpr int N = 2e5 + 5;
struct Node {int a, b, c, id;} f[N], g[N];
int n, m, q, ans[N], s[N], c[N];
void Add(int x, int v) {for (; x; x -= x & -x) s[x] += v;}
int Ask(int x) {int ans = 0; for (; x <= m; x += x & -x) ans += s[x]; return ans;}
void Cdq(int l, int r) {
    if (l == r) return;
    int mid = (l + r) >> 1;
    Cdq(l, mid); Cdq(mid + 1, r);
    int j = l, k = l - 1;
    for (int i = mid + 1; i <= r; ++i) {
        while (j <= mid && f[j].b >= f[i].b) {
            if (!f[j].id) Add(f[j].c, 1);
            g[++ k] = f[j ++];
        }
        if (f[i].id) ans[f[i].id] += Ask(f[i].c);
        g[++ k] = f[i];
    }
    for (int i = l; i < j; ++i) if (!f[i].id) Add(f[i].c, -1);
    for (int i = j; i <= mid; ++i) g[++ k] = f[i];
    for (int i = l; i <= r; ++i) f[i] = g[i];
}
bool Compare(Node a, Node b) {
    if (a.a != b.a) return a.a > b.a;
    if (a.b != b.b) return a.b > b.b;
    if (a.c != b.c) return a.c > b.c;
    return a.id < b.id;
}
signed main() {
    scanf("%d%d", &n, &q);
    for (int i = 1; i <= n; ++i)
        scanf("%d%d", &f[i].a, &f[i].b), f[i].c = f[i].a + f[i].b;
    for (int i = 1; i <= q; ++i)
        scanf("%d%d%d", &f[n + i].a, &f[n + i].b, &f[n + i].c), f[n + i].id = i;
    n = n + q;
    for (int i = 1; i <= n; ++i) c[i] = f[i].c;
    std::stable_sort(c + 1, c + n + 1);
    m = std::unique(c + 1, c + n + 1) - c - 1;
    for (int i = 1; i <= n; ++i)
        f[i].c = std::lower_bound(c + 1, c + m + 1, f[i].c) - c;
    std::stable_sort(f + 1, f + n + 1, Compare);
    Cdq(1, n);
    for (int i = 1; i <= q; ++i) printf("%d\n", ans[i]);
    return 0;
}
```

CF848C

题目描述

给定长度为 n 的序列, 定义数字 X 在 $[l,r]$内的贡献为数字 X 在 $[l,r]$ 内最后一次出现位置的下标减去第一次出现位置的下标

给定 m 次询问, 每次询问有三个整数 (a,b,c) 询问规则如下:

当 $a=1$ 时, 将数组内第 b 个元素更改为 c。

当 $a=2$ 时, 输出区间 $[b,c]$ 所有数字的贡献的和。

$n,m \le 1e5, 1 \le a_i \le n$

思路

考虑一个基本转换。

最后一次出现位置的下标减去第一次出现位置的下标等于区间内（除了开头）的值为 X 的数其下标减去前驱下标之和。

那么我们要求的就是 $\sum_{i=l}^{r} i - prev_i[prev_i \geq l]$。

由于小于 l 的部分 $prev$ 一定小于 l 就可以将式子化作 $\sum_{i=1}^{r} i - prev_i[prev_i \geq l]$。

那么这个式子也就是要求 1 ~ n 区间内 $i \leq r$ 且 $prev_i \geq l$ 的 $i - prev_i$ 之和。

再加上一个操作的时间轴 t 。

那我们就是要满足 $t_i \geq t_j, i \leq r_j, prev_i \geq l_j$。

那么就是三维偏序的板子题了。

代码实现

这边实现方法很多，诸如 `KDT` `cdq分治+平衡树维护链表` 等，这边直接就是 `树套树+set维护操作`。

```
#include <bits/stdc++.h>
#define int long long
static constexpr int N = 1e5 + 6;
int n, m, rt[N], a[N], fst[N];
std::set<int> s[N];
class SegmentTree {
    private: int cnt;
    private: class TreeNode {public: int lc, rc, w;} tr[N * 100];
    #define ls (tr[rt].lc)
    #define rs (tr[rt].rc)
    public: void Update(int &rt, int l, int r, int x, int w) {
        if (!rt) rt = ++ cnt;
        tr[rt].w += w;
        if (l == r) return;
        int mid = (l + r) >> 1;
        if (x <= mid) Update(ls, l, mid, x, w);
        else Update(rs, mid + 1, r, x, w);
    }
    public: int Query(int rt, int l, int r, int x) {
        if (!rt) return 0;
        if (l == r) return tr[rt].w;
        int mid = (l + r) >> 1;
        if (x <= mid) return Query(ls, l, mid, x) + tr[rs].w;
        else return Query(rs, mid + 1, r, x);
    }
    #undef ls
    #undef rs
} Tree;
class TreeArray {
    public: void Add(int x, int y, int w) {
        for (; x <= n; x += x & -x) Tree.Update(rt[x], 1, n, y, w);
    }
    public: int Ask(int x, int y) {
        int ans = 0;
        for (; y; y -= y & -y) ans += Tree.Query(rt[y], 1, n, x);
        return ans;
    }
} Bit;
void Update(int x, int y) {
    if (fst[x]) Bit.Add(x, fst[x], fst[x] - x);
    if (y) Bit.Add(x, y, x - y);
    fst[x] = y;
}
signed main() {
    scanf("%lld%lld", &n, &m);
    for (int i = 1; i <= n; ++i) scanf("%lld", a + i);
    for (int i = 1; i <= n; ++i) {
        if (s[a[i]].size())
```

```
            fst[i] = *s[a[i]].rbegin(),
            Bit.Add(i, fst[i], i - fst[i]);
        s[a[i]].insert(i);
    }
    for (int i = 1, opt, x, y; i <= m; ++i) {
        scanf("%lld%lld%lld", &opt, &x, &y);
        if (opt & 1) {
            auto it = s[a[x]].find(x);
            if (std::next(it) != s[a[x]].end())
                Update(*(std::next(it)), (it == s[a[x]].begin() ? 0 :
*std::prev(it)));
            s[a[x]].erase(x);
            a[x] = y;
            s[y].insert(x);
            it = s[y].find(x);
            Update(x, (it == s[y].begin() ? 0 : *std::prev(it)));
            if (std::next(it) != s[y].end())
                Update(*(std::next(it)), x);
        } else printf("%lld\n", Bit.Ask(x, y));
    } return 0;
}
```

其他偏序问题 (难度偏高，需要其他技巧 (tarjan、bitset、ac自动机))

$CF878C$

$CF1214G$

$CF590E$