香农先修班第三次课题解

bny 学妹人物传记

> 难度约为洛谷红题

根据逆序对定义暴力统计即可。

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll a[] = {2030, 201, 2030, 2030, 2050, 50, 2021, 50, 2030, 20, 2021, 11, 25}, n, cnt;
signed main()
{
    n = sizeof(a) / sizeof(ll);
    for (ll i = 0; i < n - 1; ++i)
    {
        for (ll j = i + 1; j < n; ++j)
        {
            if (a[i] > a[j])
            {
                ++cnt;
            }
        }
    }
    printf("%lld", cnt);
    return 0;
}
```

计数排序

> 难度约为洛谷橙题

模板题，具体思路参见课件。有几点需要注意的地方：

- 一个 10^7 长度的 int 数组大小为 $38.1\ MB$ ，$long\ long$ 为 $76.3\ MB$ ，所以最多只可以开 3 个 int 数组或 1 个 int 加 1 个 $long\ long$ 。
- 使用 int 时，取模公式注意强转 $long\ long$ ，否则 $a \times b$ 时会直接爆 int 溢出。
- 输入量很大，使用 cin 会炸；要关闭同步流或用 $scanf$ 或用快读。

  (`ios::sync_with_stdio(false)` 效果显著，再加上 `cout.tie(0)` 也能再加速 $100ms$)

参考代码：

```cpp
#include <iostream>
using namespace std;
#define mn 10000002
int a[mn], c[mn], n, m, ai, mod = 1e9 + 7;
long long ans;
```

```cpp
signed main()
{
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++i)
    {
        scanf("%d", &a[i]);
        ++c[a[i]];
    }
    for (int i = 1; i <= m; ++i)
    {
        for (int j = 1; j <= c[i]; ++j)
        {
            a[++ai] = i;
        }
    }
    for (int i = 1; i <= n; ++i)
    {
        (ans += 1LL * i * a[i]) %=mod;
    }
    printf("%lld\n", ans);
    return 0;
}
```

计数排序2

> 难度约为洛谷橙题

模板题，具体思路参见课件。离散化计数排序。需要注意的地方为：

- 空间只有 $8\ MB$ 可用，所以每次输入完要直接存入 map ， map 大小为 $m=100$ ，不会超空间

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef int ll;
#define mn 10000010
map<ll, ll> h;
ll n, m, mod = 1e9 + 7, v, k;
long long ans;
signed main()
{
    scanf("%d%d", &n, &m);
    for (ll i = 0; i < n; ++i)
    {
        scanf("%d", &v);
        ++h[v];
    }
    for (auto &i : h)
    {
        for (ll j = 0; j < i.second; ++j)
        {
            ans = (ans + 1LL * (++k) * i.first % mod) % mod;
        }
    }
    printf("%lld", ans);
```

```
    return 0;
}
```

上述做法时间复杂度为 $\mathrm{O}(n+n+m\log m)$ 。

也可以用数列优化，把内层循环优化掉。时间复杂度为 $\mathrm{O}(n+m\log m)$ 。用等差数列即可：

```
#include <bits/stdc++.h>
using namespace std;
typedef int ll;
#define mn 10000010
map<ll, ll> h;
ll n, m, mod = 1e9 + 7, v, c = 1, d, inv2 = 5e8 + 4;
long long ans;
signed main()
{
    scanf("%d%d", &n, &m);
    for (ll i = 0; i < n; ++i)
    {
        scanf("%d", &v);
        ++h[v];
    }
    for (auto &i : h)
    {
        d = i.second; //数列长
        ans = (ans + i.first * 1ll * d % mod * (2 * c + d - 1 + mod) % mod * inv2) % mod; //c是首项，c+d-1是末项
        c += d; //更新下一次的首项
    }
    printf("%lld", ans);
    return 0;
}
```

逆序数

> 难度约为洛谷黄题

模版题，具体思路参见课件。直接用归并排序来实现即可。

注意最后结果逆序数可能是 $long\ long$ 的。

```
#include <bits/stdc++.h>
using namespace std;
#define mn 100010
typedef long long ll;
ll n, a[mn], b[mn], ans;
void mergesort(ll lf, ll rf)
{
    if (lf < rf)
    {
        ll cf = (lf + rf) >> 1;
        mergesort(lf, cf);
        mergesort(cf + 1, rf);
        ll i = lf, j = cf + 1, je = rf, ie = cf, k = 0;
        while (i <= ie && j <= je)
```

```
        {
            if (a[i] <= a[j])
            {
                b[k++] = a[i++];
            }
            else
            {
                ans += ie - i + 1;
                b[k++] = a[j++];
            }
        }
        while (i <= ie)
        {
            b[k++] = a[i++];
        }
        while (j <= je)
        {
            b[k++] = a[j++];
        }
        for (ll h = 0; h < k; ++h)
        {
            a[lf + h] = b[h];
        }
    }
}
signed main()
{
    scanf("%lld", &n);
    for (ll i = 1; i <= n; ++i)
    {
        scanf("%lld", &a[i]);
    }
    mergesort(1, n);
    printf("%lld", ans);
    return 0;
}
```

Easy String Problem

> 2020 SCNUCPC校赛真题 E题 赛时只有五支队伍过题（难度约为洛谷绿题）

可以设排序后字符串数组 B 各个字符的下标是 $0,1,2,\cdots,n-1$ ，那么依字符串划分，就得到了每个字符各个下标组成的数组，即对每个字符，第一、第二......各次出现的下标是多少；对排序前字符串数组 A ，对每个字符，第几次出现该字符它在排序后的下标就是多少。

例如：对样例 $A=$ `CABC` ， $B=$ `CCAB` ，得下标数组 `A` 为 (2) ， `B` 为 (3) , `C` 为 $(0,1)$ ，代入 A 得 A 的下标数组 $s=(0,2,3,1)$ ，问题转化为把数组 $(0,2,3,1)$ 转化为 B 的下标数组即 $(0,1,2,3)$ 需要相邻对换多少次。

这个问题也就是把无序数组排序所需的交换次数，即求无序数组 s 的逆序数。这时候使用归并排序求逆序数模板即可。总时间复杂度 $\mathrm{O}(n\log n)$ ，空间复杂度 $\mathrm{O}(n)$ ，可以过题。

这里的一个关键是对相同的字符，在对 A 求下标数组的时候一定是顺着标记下标的，例如 s 不应是 $(1,2,3,0)$ ，即对相同字符，在结果里是相对升序排序的，那么在原本也是相对升序排序的话，可以保证相同字符间不会进行多余的交换。否则，在求逆序数的时候，会让相同字符间进行交换。

> 由于交换的等效性，求 A 的各字符下标数组再推 B 的 s 数组也是可以的。即从 A 转为 B 和从 B 逆推 A 的交换次数完全一致。

参考代码：

```cpp
#include <bits/stdc++.h>
using namespace std;
#define mn 1000010
#define ascii 130
typedef long long ll;
ll n, nowi[ascii], s[mn], t[mn], ans;
char a[mn], b[mn];
vector<ll> v[ascii];
void merges(ll lf, ll rf)
{
    if (lf == rf)
    {
        return;
    }
    ll cf = lf + rf >> 1, i = lf, j = cf + 1, k = lf;
    merges(lf, cf);
    merges(cf + 1, rf);
    while (i <= cf && j <= rf)
    {
        if (s[i] <= s[j])
        {
            t[k++] = s[i++];
        }
        else
        {
            t[k++] = s[j++];
            ans += cf - i + 1;
        }
    }
    while (i <= cf)
    {
        t[k++] = s[i++];
    }
    while (j <= rf)
    {
        t[k++] = s[j++];
    }
    for (ll h = lf; h <= rf; ++h)
    {
        s[h] = t[h];
    }
}
signed main()
{
    scanf("%lld%s%s", &n, a, b);
    for (ll i = 0; i < n; ++i)
    {
        v[b[i]].emplace_back(i);
    }
    for (ll i = 0; i < n; ++i)
    {
        s[i] = v[a[i]][nowi[a[i]]++];
```

```
    }
    merges(0, n - 1);
    printf("%lld", ans);
    return 0;
}
```

小朋友排队

> 难度约为洛谷绿题

交换是相互的，每次交换会同时积累两个小朋友的交换次数。且每个小朋友不高兴程度的和是交换次数求等差数列前 n 项和。即若第 i 个小朋友共交换了 x 次，则不高兴程度为 $\dfrac{x(x+1)}{2}$ ，所求为各小朋友的不高兴程度之和。

考虑把相互交换拆分为两个过程，即对两个小朋友 x,y ，若满足身高关系 $h_x>h_y$ ，那么 y 积累一次交换次数；若 $h_x<h_y$ ，那么 x 积累一次交换次数。

考虑归并排序中滑动窗口合并两有序数组的过程：

- 若左大于右，那么左数组从当前开始往后(含当前)的 $ie-i+1$ 个值都比右 j 大，也就是第 j 个小朋友需要跟 $ie-i+1$ 个元素交换才能归位，即第 j 个小朋友积累 $ie-i+1$ 次交换
- 否则，此时左是无需交换的，直接归位，但是右边已经有 $j-(cf+1)$ 个元素(即不含 j 的之前所有元素)已经归位，这些元素都跟第 i 个小朋友交换过，所以第 i 个小朋友积累 $j-cf-1$ 次交换

特别地，滑动窗口合并后：

- 如果左数组有剩余，事实上右数组共计 $rf-(cf+1)+1=rf-cf$ 个元素都跟剩余的这些元素交换了，所以所有剩余的所有第 i 个小朋友都积累 $rf-cf$ 次交换
- 如果右数组有剩余，不需要交换直接归位。不需要额外积累。

> 实在不理解上述过程的话建议动手画一下合并的过程

左右两个有序数组内部已经有序，它们内部的交换在之前的递归中积累过了交换次数，所以不需要再额外统计。

因此改写归并排序求逆序数模板，把上述结论转化为代码即可。

由于每个小朋友不仅有身高值，还有交换次数。排序对换时需要一并对换，所以使用结构体排序。

时间复杂度 $\mathrm{O}(n\log n)$ ，空间复杂度 $\mathrm{O}(n)$ 。

参考代码：

```
#include <bits/stdc++.h>
#define mn 100002
typedef long long ll;
struct pupil
{
    ll h, m; //h:身高; m:交换次数
} a[mn], b[mn];
ll ans, n;
void merges(ll lf, ll rf)
{
    if (lf == rf)
    {
        return;
    }
```

```
    ll cf = (lf + rf) >> 1;
    merges(lf, cf);
    merges(cf + 1, rf);
    ll p = lf, q = cf + 1, t = lf;
    while (p <= cf && q <= rf)
    {
        if (a[p].h > a[q].h)
        {
            a[q].m += cf + 1 - p;
            b[t++] = a[q++];
        }
        else
        {
            a[p].m += q - 1 - cf;
            b[t++] = a[p++];
        }
    }
    while (q <= rf)
    {
        b[t++] = a[q++];
    }
    while (p <= cf)
    {
        a[p].m += rf - cf;
        b[t++] = a[p++];
    }
    for (int i = lf; i <= rf; ++i)
    {
        a[i] = b[i];
    }
}
signed main()
{
    scanf("%lld", &n);
    for (int i = 0; i < n; ++i)
    {
        scanf("%lld", &a[i].h);
    }
    merges(0, n - 1);
    for (int i = 0; i < n; ++i)
    {
        ans += (a[i].m + 1) * a[i].m / 2; //x,x+1一奇一偶得偶，偶/2必然为整数
    }
    printf("%lld", ans);
    return 0;
}
```

后缀排序(BF实现)

> BF: Brute Force ， 即暴力解法

比较简单的结构体排序应用题。第 i 个字符串为从 i 开始的子串，直接使用字符串的 `substr(x)` 方法即可取从下标 x (这里是从 0 开始数的)开始的子串。取所有子串复杂度为 $\mathrm{O}(|t|^2)$ 。定义结构体，存下标和子串。以字典序为依据排序。第 i 小是 $sa[i]$ ，所以升序排序。排序后取当前结构体下标即为 $sa[i]$ 。字符串比较复杂度为 $\mathrm{O}(|t|)$ ，所以结构体快排时间复杂度为 $\mathrm{O}(|t|^2\log|t|)$ 。

根据定义，第 i 小是后缀 $sa[i]$ ，根据定义，即 $rk[sa[i]]$ 排在第 i 名，即有性质： $rk[sa[i]]=i$。所以输出 rk 的时间复杂度为 $\mathrm{O}(|t|)$ 。

> 有 $\mathrm{O}(|t|\log|t|)$ 的做法，使用倍增思想+基数排序+计数排序。感兴趣的前往洛谷 `P3809` 查看详细题解。

参考代码：

```cpp
#include <bits/stdc++.h>
using namespace std;
#define mn 1010
typedef long long ll;
ll n, rk[mn];
string t;
struct node
{
    string s;
    ll i;
    bool operator<(const node &x) const
    {
        return s < x.s;
    }
} s[mn];
signed main()
{
    cin >> t;
    n = t.size();
    for (ll i = 1; i <= n; ++i)
    {
        s[i].i = i;
        s[i].s = t.substr(i - 1);
    }
    sort(s + 1, s + 1 + n);
    for (ll i = 1; i <= n; ++i)
    {
        printf("%lld ", s[i].i); //即sa[i]
        rk[s[i].i] = i;
    }
    printf("\n");
    for (ll i = 1; i <= n; ++i)
    {
        printf("%lld ", rk[i]);
    }
    return 0;
}
```

Left 4 Dead 2

> 2020 SCNUCPC校赛真题 J题 赛时有七个人过题 难度约为洛谷黄题

小模拟题。按题意翻译成代码即可。有一些需要注意的地方：

- $n = 1$ 时，不是 `No winner`，而是胜者为 1；因为根据定义，`No winner` 需要多个玩家有同分，不符合多个。
- 判定 `No winner` 的依据是排序后第一名和第二名成绩完全一样(且 $n > 1$)
- `sum of hp` 在题中定义不是击杀 `hp` 之和，而是每场比赛己方击杀减去对方击杀之和
- 每次询问前要清零数组，防止上次存储的值影响当前的值

在实现的时候，可以：

- 设结构体数组 `p[i]`，有成员 n, s, i，依次代表胜场数，击杀差之和，下标编号
- 设二维数组 `q[i][j]` 存每场比赛 `i` 对 `j` 时 `i` 的击杀 `hp` 和，读入时把四种怪物的击杀求和存在这里
- 遍历 `q` 计算 `p`，结构体快排 `p`，输出答案

时间复杂度为 $\mathrm{O}(tn^2 + tn \log n)$。

参考代码：

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll t, n, q[105][105]; //每局HP总和
struct node
{
    ll n, s, i; //number kill; sum hp; index
    bool operator<(const node &r) const
    {
        return n != r.n ? n > r.n : s > r.s;
    }
} p[105];
char temp[66];
signed main()
{
    for (scanf("%lld", &t); t; --t)
    {
        scanf("%lld", &n);
        memset(p, 0, sizeof p), memset(q, 0, sizeof q);
        for (ll i = 1; i <= n; ++i)
        {
            p[i].i = i;
        }
        for (ll i = 1; i <= n; ++i)
        {
            for (ll j = 1, a1, a2, a3, a4; j <= n; ++j)
            {
                scanf("%lld%lld%lld%lld", &a1, &a2, &a3, &a4);
                q[i][j] = a1 * 98000 + a2 * 9800 + a3 * 980 + a4 * 98;
                if (j < n)
                {
                    scanf("%s", temp); //读走|
                }
            }
        }
        for (ll i = 2; i <= n; ++i)
        {
            for (ll j = 1; j < i; ++j)
            {
```

```
                p[i].s += q[i][j] - q[j][i];
                p[j].s += q[j][i] - q[i][j];
                if (q[i][j] > q[j][i])
                {
                    ++p[i].n;
                }
                else if (q[i][j] < q[j][i])
                {
                    ++p[j].n;
                }
            }
        }
        sort(p + 1, p + 1 + n);
        if (n == 1)
        {
            printf("1\n");
        }
        else if (p[1].n == p[2].n && p[1].s == p[2].s)
        {
            printf("No winner\n");
        }
        else
        {
            printf("%lld\n", p[1].i);
        }
    }
    return 0;
}
```

排兵布阵

> 2019 SCNU蓝桥杯热身赛 E题； 难度约为洛谷黄/绿题

可以发现一个事实， x 坐标与 y 坐标互不干扰。无论一方取何值都不影响另一方的取值。所以可以把问题拆分为两个子问题：把所有 y 坐标排相同和所有 x 坐标排相邻，在这两个子问题里都让移动次数最小。

先考虑相同，设排完后坐标为 y' ，为使移动次数 s_1 最小，即：

$$s_1 = |y_1 - y'| + |y_2 - y'| + \cdots + |y_n - y'|$$

转化为有自变量为 x 的一元函数 $f(x)$ 和若干常数 $x_1, x_2, \cdots, x_n$ ，求 $f(x)$ 最小值。其中 $f(x)$ 为：

$$f(x) = |x_1 - x| + |x_2 - x| + \cdots + |x_n - x|$$

不妨设数组有序，即 $x_1 \le x_2 \le \cdots \le x_n$。

1. 若 $n = 1$ ，显然取 $x = x_1$, 有 $f(x) = 0$， $f(x)$ 最小。
2. 若 $n = 2$ ，若 $x_1 = x_2$ ，显然取 $x = x_1$ 即可使最小 $f(x) = 0$ 。若 $x_1 \ne x_2$， 假设 $x < x_1$ 或 $x > x_2$ ，作图数形结合可知，x 越往两端走， $f(x)$ 越大。结合函数图像不难发现，选取 $x \in [x_1, x_2]$ 的结果必然优于选取 x 在其他区间的值。 而 $x \in [x_1, x_2]$ 时， 有：

$$f(x) = |x_1 - x| + |x_2 - x| = x - x_1 + x_2 - x = x_2 - x_1$$

因此对 $n = 2$ 时， 任取 $x \in [x_1, x_2]$ ，均可获得最小值 $f(x) = x_2 - x_1$

3. 若 $n=3$，类似分析可得， $x\in[x_1,x_3]$ 必然优于 x 取其他值的情况，因为在两端时距 x_2 的距离也必然加大。在这个区间时，有：
$$f(x)=|x_1-x|+|x_2-x|+|x_3-x|=x_3-x_1+|x_2-x|$$
为使得 $f(x)$ 最小，可以令 $x=x_2$， 得 $f(x)=x_3-x_1$
4. 若 $n=4$，同理， $x\in[x_1,x_4]$，有：
$$f(x)=x_4-x_1+|x_2-x|+|x_3-x|$$
其中 $|x_2-x|+|x_3-x|$ 的最小值可以看做忽略点 x_1,x_4 后， $n'=2$ 的一个子问题，根据上面第二点可知， $x\in[x_2,x_3]$ 时，该子问题的最优解是 x_3-x_2，而 $[x_2,x_3]\subset[x_1,x_4]$，故答案为 $f(x)=x_4-x_1+x_3-x_2$。
5. 若 $n=5$，同理用子问题的思想，得 $x=x_3$ 时， $f(x)=x_5-x_1+x_4-x_2$。 若 $n\geq 6$，可以每次用子问题的思想分析 $n'=n-2$ 时的子问题，得到答案。综上所述， $x\in[x_{\lfloor\frac{n}{2}\rfloor},x_{\lfloor\frac{n+1}{2}\rfloor}]$ 时， $f(x)=x_{n-1}-x_1+x_{n-2}-x_2+\cdots$。($n$ 为偶数时 x 取值区间有长度； n 为奇数时，取值区间是一个点)

总结为：无论奇偶，使得函数最小时的 x 取值范围必然包含点 $x_{\lfloor\frac{n}{2}\rfloor}$，所以即求 $f(x_{\lfloor\frac{n}{2}\rfloor})$ 即为答案。

对于后一个问题：使得 x 坐标相邻，**排序后**的原数列相邻的特点是 $|x_i-x_{i-1}|=1$，即构成公差为 1 的等差数列，如果第 i 个点坐标值减去 i，即可把排序后的原数列转化为常数列，问题等效转化为上述求最小移动使得坐标值相同的问题。注意在这个减 i 过程中，没有真实的移动发生，所以转化时不贡献移动次数。(对这个子问题需要**两次排序**，第一次排序后等差数列转常数列，此时原等差数列中位数可能会发生改变，然后第二次排序求中位数)

使用快排，时间复杂度为 $\mathrm{O}(n\log n)$。

参考代码如下：

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define mn 10010
ll n, x[mn], y[mn], ans;
void cal(ll *a) //求f(x)的最小值
{
    sort(a + 1, a + 1 + n);
    for (ll i = 1; i <= n; ++i) //求f(xmid)
    {
        ans += abs(a[i] - a[(n + 1) / 2]);
    }
}
signed main()
{
    scanf("%lld", &n);
    for (ll i = 1; i <= n; ++i)
    {
        scanf("%lld%lld", &x[i], &y[i]);
    }
    sort(x + 1, x + 1 + n);
    for (ll i = 1; i <= n; ++i)
    {
        x[i] -= i;
    }
    cal(x), cal(y);
    printf("%lld", ans);
    return 0;
```

```
}
```

Ares, Toilet Ares

> 其实这是一道复习题。是一道不错的逆元题目，所以拿来当练习题了。难度约为洛谷黄/绿题

题意理解翻译为中文、省略掉干扰信息，即：能做对 a 道题，且对于题号为 `K` 的这道题，能得到 k 个代码片段，第 i 个片段有 $\frac{y_i}{z_i}$ 的概率失效。能做对这道题当且仅当所有**长度不为零**的片段均不失效。总概率为：

$$\prod_{i=1}^{n} \frac{z_i - y_i}{z_i}$$

因此，数学期望为：

$$E = a + \prod_{i=1}^{n} \frac{z_i - y_i}{z_i}$$

注意：

$$(a + \frac{b}{c}) \bmod p \neq (a + \frac{b \bmod p}{c \bmod p}) \bmod p$$

所以需要转化为：

$$(a + \frac{b}{c}) \bmod p = \frac{ac + b}{c} \bmod p$$

即先求分子乘积 b 和分母乘积 c，再代入上述等式可得答案。

时间复杂度为 $\mathrm{O}(k + \log p)$。

参考代码：

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define sc(x) scanf("%lld", &x)
ll n, m, k, a, l, x, y, z, tfz = 1, tfm = 1, mod = 4933;
ll qpow(ll a, ll b = mod - 2)
{
    ll res = 1;
    for (; b; b >>= 1, (a *= a) %= mod)
    {
        if (b & 1)
        {
            (res *= a) %= mod;
        }
    }
    return res;
}
signed main()
{
    sc(n), sc(m), sc(k), sc(a), sc(l);
    for (ll i = 0; i < k; ++i)
    {
```

```
        sc(x), sc(y), sc(z);
        if (!x)
        {
            continue;
        }
        (tfz *= (z - y)) %= mod;
        (tfm *= z) %= mod;
    }
    printf("%lld", (tfm * (a % mod) % mod + tfz) % mod * qpow(tfm) % mod);
    return 0;
}
```

次小值

> 难度约为洛谷橙题

有序的维护的简单应用。

由于求的是第一个比 v 小的不同的整数，容易想到需要去重，所以使用 `set` 。用 `lower_bound` 找第一个大于等于 v 位置的迭代器，其前一个位置即所求。无论数组里有无 v 均可如此求。

`not exist` 的充要条件是数组为空或找到的迭代器是首个迭代器。

时间复杂度为 $\mathrm{O}(n\log n)$ 。参考代码：

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
ll n, c, v;
set<ll> s;
signed main()
{
    scanf("%lld", &n);
    while (n--)
    {
        scanf("%lld%lld", &c, &v);
        if (c == 1)
        {
            s.insert(v);
        }
        else
        {
            auto p = s.lower_bound(v);
            if (!s.size() || p == s.begin())
            {
                printf("not exist\n");
            }
            else
            {
                printf("%lld\n", *--p);
            }
        }
    }
    return 0;
}
```

普通平衡树

> 套 pb_ds 难度约为洛谷绿题；手写为紫题

模板题。直接套 `pb_ds` 的红黑树即可。具体请参考课件。没有什么特别挖坑的地方，数据也是洛谷的某个原测试点。

如果不懂异或的可以查一下各搜索引擎，异或是一种比较重要的位运算。

时间复杂度为 $O(n \log n)$ 。

使用 `pb_ds` 实现的参考代码：

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
__gnu_pbds::tree<ll, __gnu_pbds::null_type, less<ll>, __gnu_pbds::rb_tree_tag, __gnu_pbds::tree_order_statistics_node_update> tr;
ll n, m, lastans, dig, ans;
signed main()
{
    scanf("%lld%lld", &n, &m);
    dig = n + m;
    for (ll i = 1, a; i <= n; ++i)
    {
        scanf("%lld", &a);
        tr.insert(a * dig + i);
    }
    for (ll i = n + 1, opt, x; i <= m + n; ++i)
    {
        scanf("%lld%lld", &opt, &x);
        x ^= lastans;
        if (opt == 1)
        {
            tr.insert((x * dig) + i);
        }
        else if (opt == 2)
        {
            tr.erase(tr.lower_bound(x * dig));
        }
        else
        {
            if (opt == 3)
            {
                lastans = tr.order_of_key(x * dig) + 1;
            }
            else if (opt == 4)
            {
                lastans = (*tr.find_by_order(x - 1)) / dig;
            }
            else if (opt == 5)
            {
                lastans = (*--tr.lower_bound(x * dig)) / dig;
            }
```

```
            else if (opt == 6)
            {
                lastans = (*tr.upper_bound(x * dig + dig)) / dig;
            }
            ans ^= lastans;
        }
    }
    printf("%lld\n", ans);
    return 0;
}
```

寄蒜几盒

> 套`pb_ds`难度约为洛谷绿题；手写为紫题

平衡树应用题。一道较难的题目，考察细节很多，稍有不慎就WA。

解决如何确定斜率的问题。问避免精度损失和对无穷的判定，斜率可以用一个分数 $\frac{y}{x}=\frac{\Delta y}{\Delta x}$ 来表示。比较斜率大小为：

$$\frac{y_a}{x_a}<\frac{y_b}{x_b}\Rightarrow y_a\times x_b<y_b\times x_a$$

由此避免了小数运算，避开了精度损失。

特别地，思考这个例子：

$$\begin{cases}\frac{-1}{1}<\frac{1}{2} & \Rightarrow(-1)\times 2<1\times 1\\ \frac{1}{-1}<\frac{1}{2} & \Rightarrow 1\times 2<1\times(-1)\end{cases}$$

可以发现分母为负数时会导致错误，所以约定分母非负数。

特别地，当 $x=0$ 或 $y=0$ 时，需要把另一方设为 1 。否则容易发现，在该情况下，多个零斜率或无穷斜率与其他斜率的比较会出现偏差。

> 可以化简为最简分式，也可以不化简，不影响结果。

自此，解决了斜率问题。可以设结构体，有三个成员 x,y,i ，代表斜率和编号。比较依据为先看斜率大小，如果斜率相同看编号大小。以此来维护操作 3 和其他操作所要求的功能。

接下来考虑如何用 `pb_ds` 平衡树维护操作：

1. 操作 1 ： 直接插入即可，记分子分母是 $\Delta y,\Delta x$ ；注意不能各自加绝对值，否则会造成诸如 $k=-1$ 与 $k=1$ 相等。当发现分母小于零时，分子分母同乘 -1 。特判 $\Delta x,\Delta y$ 一方为零的情况。设一个变量表示当前编号，每次插入使其自增。
2. 操作 2 ： 先用对应斜率和负无穷编号(如 0)来 `lower_bound` (`lower` 或 `upper` 均可，因为编号不同不可能取等，下同)查找第一个大于该斜率的点，如果找到，即迭代器不为 `.end()` 且迭代器对应斜率为输入斜率，那么就删除，否则不操作。
3. 操作 3 ： 用上述方式查找，如果找到就输出迭代器对应编号，否则输出 `-1` 。
4. 操作 4 ： 先用上述方式找第一个同斜率点，如果找不到则输出 0 ；如果找到了，再用对应斜率和可以代表无穷编号(如 $n+1$)找第二个迭代器，如果找到了，依次计算这两个迭代器的排名(用 `.order_of_key(*迭代器)`)，作差即可；如果找不到(为 `.end()` ，这里不用判断斜率，因为这个找到的必然是下一个斜率)，取当前平衡树的长度与第一个迭代器排名作差。(直接作差是区间长度计数模板，对区间 $[x,y)$ ，长度 $=y-1-x+1=y-x$)

5. 操作 5： 用编号 $n+1$ 找迭代器，如果找到了(不为 `.end()`)，用平衡树长度与迭代器排名直接作差；如果找不到输出 0 。
6. 操作 6： 用编号 0 找迭代器，如果找到了，那么它是右边界，对 $[0, y)$ 作差即得 y ，所以输出迭代器排名即可。如果找不到输出平衡树长度。

时间复杂度为 $\mathrm{O}(n \log n)$ 。

参考代码:

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
typedef long long ll;
using namespace std;
struct line
{
    ll x, y, i; //k=y/x
    bool operator<(const line &r) const
    { // y/x < r.y/r.x
        if (x * r.y != r.x * y)
        {
            return y * r.x < r.y * x;
        }
        return i < r.i;
    }
};
__gnu_pbds::tree<line, __gnu_pbds::null_type, less<line>, __gnu_pbds::rb_tree_tag, __gnu_pbds::tree_order_statistics_node_update> tr;
ll n, c, ax, ay, bx, by, dx, dy, g, cnt;
signed main()
{
    scanf("%lld", &n);
    for (ll h = 0; h < n; ++h)
    {
        scanf("%lld%lld%lld%lld%lld", &c, &ax, &ay, &bx, &by);
        dx = ax - bx, dy = ay - by;
        if (dy < 0)
        {
            dx *= -1, dy *= -1;
        }
        g = max(1ll, __gcd(dx, dy)); //化简最简分式，也可以不做
        //__gcd函数，同负可能出错(加abs(__gcd)则无错)，同正或一正一负得正，有零得另一方
        if (dx == 0)
        {
            dy = 1;
        }
        else if (dy == 0)
        {
            dx = 1;
        }
        else
        {
            dx /= g, dy /= g;
        }

        if (c == 1)
```

```
        {
            tr.insert({dx, dy, ++cnt});
        }
        else if (c == 2)
        {
            auto pos = tr.lower_bound({dx, dy, 0});
            if (pos != tr.end() && pos->x * dy == pos->y * dx)
            {
                tr.erase(pos);
            }
        }
        else if (c == 3)
        {
            auto pos = tr.lower_bound({dx, dy, 0});
            if (pos == tr.end() || pos->x * dy != pos->y * dx)
            {
                printf("-1\n");
            }
            else
            {
                printf("%lld\n", pos->i);
            }
        }
        else if (c == 4)
        {
            auto p1 = tr.lower_bound({dx, dy, 0});
            auto p2 = tr.upper_bound({dx, dy, n + 1});
            if (p1 == tr.end() || p1->x * dy != p1->y * dx)
            {
                printf("0\n");
                continue;
            }
            ll m1 = tr.order_of_key(*p1);
            ll m2 = tr.order_of_key(*p2);
            if (p2 == tr.end())
            {
                m2 = tr.size();
            }
            printf("%lld\n", m2 - m1);
        }
        else if (c == 5)
        {
            auto p = tr.upper_bound({dx, dy, n + 1});
            if (p == tr.end())
            {
                printf("0\n");
                continue;
            }
            printf("%lld\n", tr.size() - tr.order_of_key(*p));
        }
        else if (c == 6)
        {
            auto p = tr.lower_bound({dx, dy, 0});
            if (p == tr.end())
            {
                printf("%lld\n", tr.size());
                continue;
            }
```

```
            printf("%lld\n", tr.order_of_key(*p));
        }
    }
    return 0;
}
```