A - An Easy Geometry Problem

令 $B_i = 2A_i - ir$，则有 $A_{i+r} - A_{i-r} = kr + b$ 等价于 $B_{i+r} - B_{i-r} = 2b$。

可以线段树维护哈希，开两棵树分别维护 A_i 正过来的哈希和 $A_i - 2b$ 倒过来的哈希，每次线段树上二分就好了。

由于值域比较大，可能需要双模/较大模数。

B - Counting Multisets

考虑 $p(S)$ 为奇数意味着什么。

对于可重集 S，$p(S)$ 是容易计算的。记 occ_i 表示 i 在可重集 S 中的出现次数，显然 $p(S) = \binom{|S|}{occ}$。

由 Lucas 定理的经典推论，$[\binom{n}{m} \bmod 2 = 1] = [m \subseteq n]$，其中 $m \subseteq n$ 表示 m 在二进制下为 n 的子集。由于 $\sum occ_i = |S|$，$[p(S) \bmod 2 = 1]$ 实际上可以表示为 occ_i 在二进制表示下两两无交。

现在考虑如何求 $f(n, x, y)$。直接枚举每个数出现了多少次是无法接受的。考虑进行一些转化。由于 $|S| = n$ 为定值，考虑对 n 在二进制下的每一位 i，若 n 的第 i 位为 1，则选择一个 $a_i \subseteq y$，表示 occ_{a_i} 的第 i 位为 1，并对 x 造成 $a_i \times 2^i$ 的贡献。不难发现合法的序列 $\{a_i\}$ 与可重集 S 一一对应。

考虑如何统计合法的序列 $\{a_i\}$ 的数量。注意到，对于每一位 i 选择一个 $a_i \subseteq y$，相当于对于 y 的每一位 j，若 y 的第 j 位为 1 则 a_i 的第 j 位可以取到 1，并最终对 x 造成 2^{i+j} 的贡献。这样一来，求 $f(n, x, y)$ 的事情就转化为，统计大小为 $\log_2 n \times \log_2 y$ 的 $0/1$ 矩阵的数量，使得：

- 对于 n 为 0 的每一位 i，第 i 行均为 0；
- 对于 y 为 0 的每一位 j，第 j 行均为 0；
- 对于 y 为 1 的每一位 j，第 j 行有 1 出现；
- 若位置 (i, j) 为 1，则对和有 2^{i+j} 的贡献。矩阵中每个位置的贡献之和为 x。

考虑按照 $i + j$ 从小到大的顺序进行数位 DP。记 $dp_{s,d}$ 表示每一行是否有 1 出现的状态为 s，且和的 $< i + j$ 的二进制位恰好为 x 的前 $i + j$ 位，同时 $\geq i + j$ 的二进制位为 d 的方案数。转移时，枚举矩阵的 (i, j) 这一位是否为 1。容易做到 $O(2^{\mathrm{pcnt}(y_{\max})} \log^2 y \log n)$ 并通过此题。

C - Counting Strings

考虑建出 SAM，令节点 u 对应的子串长度在 $L(u), R(u)$ 间，子树内所有 endpos 组成集合 E，则对答案的贡献为

$$\sum_{i=L(u)-1}^{R(u)-1} [\exists x \in E, \gcd(i, x) = 1] i$$

考虑全局地枚举每个 i，计算其对答案的贡献次数。

对于每个 $\gcd(i, x) = 1$，若 x 对应的 endpos 在 t 处，则对于 t 到根路径上的每个点 u，只要 $i \in [L(u) - 1, R(u) - 1]$，就能产生贡献。但这样的 x 可能很多，直接枚举无法接受。

考虑预处理出 $1 \sim n$ 的所有素数，然后爆搜 i 各质因子构成的集合 S，同时维护所有 t 构成的集合 T，以及所有 i 的贡献次数 c_i。每当加入素数 p 后，$\forall p|x$，记 x 对应的 endpos 在 t_0 处，则将 t_0 剔除出 T。对于每个 S，枚举所有质因子集合 $= S$ 的 i，更新答案即可。

具体来说，当 t_0 将被剔除出 T 时，若对 c 有影响，会存在一条从 t_0 向上的链，使得链上所有的 u 由合法变为不合法。由于 parent tree 上各点的 (L, R) 是相对连续的，对 c 的影响始终是区间减。考虑差分，将一次区间减看做两次单点修改，将一次单点查询看做一次前缀查询和，通过根号平衡，将前者复杂度控制在 $O(1)$，后者 $O(\sqrt{n})$。

最后的问题在于，如何求出 t_0 对应的链的顶端。根据经典结论，我们只关心 T 中 t_0 在 DFS 序上的前驱和后继，借助 $O(n \log n) - O(1)$ LCA 可以确定链的顶端。至于如何求前驱和后继，注意到只有删除和撤销操作，链表可以 $O(1)$ 维护。

最后分析时间复杂度。设 X 为在 T 中删去元素的次数，则复杂度为 $O(X + n\sqrt{n})$。

那 X 是多少呢？写个程序算一下，发现 X 在 $n = 10^5$ 时只有 19071573。至此，我们解决了本题。

如果你的 LCA 是 $O(n \log n) - O(\log)$ 的，或者求前驱、后继带了 $\log$，或者你没有根号平衡，会多一个 $\log$，可能无法通过。

D - Deep Intervals

考虑如何解决 1 询问，建出猫树，记录 $dp_{i,0/1,j}$ 表示向一边延伸，第一个是左括号/右括号，选出 j 个括号的答案。做完前后缀和后左右可以 $O(k)$ 合并，故总复杂度为 $O(nk \log n + qk)$。

对于 2 询问，考虑对于每个子序列计算贡献。设这个子序列的最左边的括号位置是 L，最右边 R，则它的贡献为 $(L - l + 1) \times (r - R + 1)$。将式子拆开，处理出 dp 后记录选出子序列并乘上左端点位置或乘上右端点位置后的前后缀和，同样可以 $O(k)$ 合并，注意整个选出子序列在 $[l, mid]$ 和 $[mid + 1, r]$ 的情况。总复杂度 $O(nk \log n + qk)$。

E - Dominating Point

正解

找到第一个支配点

引理：竞赛图中出度最大的点是支配点。

> 设该点为 u，u 连向的点集为 S，连向 u 的点集为 T。u 一步显然能到达 S 中所有点，下证 u 两步可以到达 T 中所有点。
>
> 考虑反证，若存在不满足条件的点，设为 $x \in T$，则有对于任意 $y \in S$，x 连向 y，即存在边 $(x \to y)$。注意到 u 的出度是 $|S|$，但是现在 x 不仅连向 S 中所有点，还连向 u，故 x 的出度大于等于 $|S| + 1$，与 u 出度最大矛盾。
>
> 故引理得证。

由引理，我们可以立即得出一个支配点。引理同时能得出结论：**对于每个竞赛图，都有至少一个支配点。**

找到第二个支配点

设第一个支配点为 a，a 连向的点集为 S，连向 a 的点集为 T，找出 T 内点组成子图的支配点 b，则 b 也是整张图的一个支配点。因为 b 可以连向 a，紧接着可以在第二步连向 S 中所有点，且它还能在两步之内走到 T 中所有点，故 b 是支配点。

这里需要特判一点：如果 T 是空集，则显然支配点只有一个，输出 `NOT FOUND`。即为结论：**如果一个竞赛图不存在一个连向其余所有点的点，那么它有至少三个支配点，否则仅有一个。**为了方便，我们称这样的点是图的中心。

找到第三个支配点

如果 b 不是 T 的中心，那么 T 其实有三个支配点，我们只需要递归下去找到这三个点就能找到所求答案了。下面假设 b 是 T 的中心，令 $T_0 = T \setminus \{b\}$。

设 S 的所有点中，连向 b 的点集为 S_1，b 连向的点集为 S_2。若 S_1 非空，则取出 S_1 的一个支配点 c。它可以通过以下方式达到剩余所有点：$c \to b \to (a, T_0, S_2)$。这样我们就找到了三个支配点。

若 S_1 为空，即 b 能连向 S_2, T_0, a，并且因为 S_1 是空集，$S_2 \cup T_0 \cup \{a\}$ 其实就是除了 b 以外的所有点，与 a 出度最大矛盾。

如果没有发现这个引理

同样先特判 `NOT FOUND`。

任取点 a，设连向 a 的点集为 S_1，a 连向的点集为 S_2。若 S_1 不存在中心，递归求出 S_1 中三个支配点即为答案。否则设 S_1 的中心为 b，同时令 $S_3 = S_1 \setminus \{b\}$。令 S_2 中连向 b 的点集为 S_4，b 连向的点集为 S_5。

若 S_4 是空集，那 b 就是整张图的中心，已经被特判掉，故 S_4 非空。若 S_4 不存在中心，递归求 S_4 三个支配点即为答案。否则设 S_4 中心为 c，同时令 $S_6 = S_4 \setminus \{c\}$。此时 b, c 均为支配点，只需再找到一个即可。

若 S_3 为空，则 a 也是支配点，答案为 a, b, c。否则设 S_3 中连向 c 的点集为 S_7，c 连向的点集为 S_8。

若 S_7 为空，a 同样是支配点，答案为 a, b, c。否则取 S_7 的支配点 d，d 是原图的支配点，答案为 b, c, d。

F - An Easy Counting Problem

这个东西是显然的 Lucas 定理的形式，可以按照类似数位 DP 的方式解决。令 $dp_{i,j}$ 表示填好了前 i 位，现在的组合数值为 j 的方案数，转移式子为 $dp_{i,j} = \sum\limits_{xy \bmod p = j} dp_{i-1,x} cnt_y$，其中 cnt_y 为 $k = 1$ 时的答案数组（可以暴力预处理），容易做到 $O(kp^2)$。观察到转移满足结合律，可以用快速幂的思想优化到 $O(p^2 \log k)$。再观察到对下标取离散对数后转移就是比较一般的卷积的形式，可以使用多项式优化到 $O(p \log p \log k)$。

G - An Easy Math Problem

注意到 $\gcd(p, q) = d \geq 2$ 时，把 p, q 同时除以 d 仍然满足条件。因此只需要考虑 p, q 互质的情况。

先不考虑 $p \leq q$ 的条件。将 n 质因数分解为 $\prod \alpha_i^{\beta_i}$。对于每一个 α_i，由于 p, q 互质，不能同时是 α_i 的倍数，因此有 $2\beta_i + 1$ 种情况，总数为 $\prod (2\beta_i + 1)$。

注意到除了 $p=q=1$，其余全是两两对应的，所以答案是：

$$\frac{\prod(2\beta_i+1)+1}{2}$$

H - Elimination Series Once More

淘汰赛的进程类似一棵满二叉树的结构，每个叶子代表一个选手。

考虑计算选手 x 最多能赢多少次。枚举 t，钦定 x 能赢 t 次，这意味着 x 必须是 u 子树内所有叶子中能力最强的，其中 u 表示 x 向上跳 t 步到达的点。

考虑如何利用魔法。贪心地，我们将子树内所有比 x 强的人换出去，把子树外比 x 弱的人换进来。

问题转化为，求一个子树中比 x 弱的选手数量，若暴力统计，时间复杂度 $O(4^n)$。

考虑从弱到强枚举选手，维护每个子树中已加入的选手数量，每次只会更新 $O(n)$ 个点的信息。

时间复杂度 $O(2^n n)$，可以通过本题。

I - Max GCD

考虑枚举答案，再按在序列中的顺序枚举它的倍数。对于可能对答案有贡献的三元组(i,j,k)，一定满足 i 与 j 相邻，且 k 是离 j 最近的合法的标号。显然这样的三元组个数最多只有 $O(nd)$ 个。

接下来考虑怎么求每个三元组 (i,j,k)。一种写法是，因为对于一个枚举到的答案，随着 i 的增加， k 是没有单调性的，所以想求出所有三元组只能二分答案。但是如果我们从右往左枚举 i，用双指针维护 k，虽然没有办法求出所有三元组，但对于每个错过的三元组 (i,j,k)，一定存在一个三元组 (i',j',k') 使得 $i'\le i$ 且 $k\le k'$。所以我们错过的三元组一定不优。于是我们就可以在 $O(nd)$ 的时间内求出所有合法三元组。

另一种常数更小的写法是：我们不直接枚举答案，而是从左往右扫，对每个数 a_i 枚举它的因数 x。设上一次出现 x 的倍数的位置是 lst_x，这样所有在 $2\times i-lst_x$ 后面的数 k 都能组成 (lst_x,i,k) 且答案为 x 的三元组。于是我们把 x 挂在 $2\times i-lst_x$ 上就可以方便地求出所有三元组 (i,j,k)。

接下来，我们离线询问，将三元组 (i,j,k) 与对应的答案挂在 i 或 k 上，就只需要做一个二维数点。由于 $\sqrt{n}$ 与 d 同阶，且要做 $O(nd)$ 次修改和 $O(q)$ 次查询，我们用 $O(1)$ 修改 $O(\sqrt{n})$ 查询的分块即可做到 $O(nd+n\sqrt{n})$ 的复杂度。

J - Graph Changing

考虑 $k>3$。注意到操作完一次一个点不孤立当且仅当 u 能到最左边或最右边。

考虑 u,v 之间的距离，有三种情况：

- $|u-v|\ge k$，距离为 1。
- u,v 均可到达最左边或最右边，距离为 2。
- u,v 分别能到最左/最右，距离为 3。
- 其他情况，无法到达。

于是，经过至多两次操作图为空。

$k=3$ 时：

- $n \le 3$：经过一次操作图为空。
- $n = 4$：一次操作后为 $1 - 4$。
- $n = 5$：一次操作后为 $4 - 1 - 5 - 2$，两次操作后位 $2 - 4$。
- $n = 6$：操作一次同 $k > 3$ 情况，操作两次 $3 - 5 - 2 - 4$，操作三次 $3 - 4$。
- $n = 7$：操作一次同 $k > 3$ 情况，操作两次 $3 - 5$。
- $n \ge 8$：容易发现操作一次后不可能出现距离为 3 的情况。

$k = 2$ 时：

- $n \le 2$：经过一次操作图为空。
- $n = 3$：一次操作后 $1 - 3$，两次操作后图为空。
- $n \ge 4$：操作后会在 $1 - 2 - 3 - \cdots - n$ 和其补图之间切换。

$k = 1$ 时：经过任意次操作均为完全图。

于是，复杂度为 $O(q)$。

也可以对于 $k \le 2, n \le 7$ 的情况暴力预处理。

K - Penguins in Refrigerator

记 a_i 表示原序列翻转后，第 i 个位置的企鹅的宽度。

考虑整个序列，记 $p = \operatorname{argmax} a_i$，则对于 $a_i + a_p \le W$，则 i 可以被交换到序列的任意位置。对于剩下的 i，它们和 p 的相对位置不会改变。

那么考虑删去这些可以任意交换的 i。对于剩下的 i，可以分为 p 左侧和右侧的区间递归求解。

考虑上述操作的本质，实际上就是对每个点 i，找到其在笛卡尔树上深度最小的祖先 p 使得 $a_i + a_p \le W$。若 p 存在，则 i 可以在 p 支配的区间内随意移动。

考虑如何对每个 i 求出 p。一种方法是直接在祖先上二分，做到 $O(n \log n)$。但实际上可以做到线性。按照 a_i 从大到小的顺序遍历每个 i，此时合法的 p 越来越多。对于所有合法的 a_p，将 p 与 p 在笛卡尔树上的两个儿子用并查集合并，查询 i 所在的并查集中深度最小的点即可。然而并查集常数较大，实测跑得和 $O(n \log n)$ 差不多快。

在问题 (1) 中，直接对于可以在区间内随意移动的 i，在 p 处统计并乘上一个排列数，全部乘起来即为答案。

在问题 (2) 中，记 $[l_p, r_p]$ 表示 p 支配的区间。从左往右扫描序列，维护一个 set 表示哪些数可以放在当前位置。对于可以随意移动的 i，在 l_p 处加入，r_p 处删除。对于不能随意移动的 i，在 i 处加入，i 处删除。需要删除 i 时，若 i 已经从 set 中被删除则跳过。否则，将 set 中比 i 编号更小的依次弹出，再弹出 i 即可。

L - Prism Palace

求出多边形的每个内角，若一条边相邻两个内角 α, β 满足 $\alpha + \beta < \pi$，则这条边有 $\frac{\pi - \alpha - \beta}{\pi}$ 的概率成为集合中最大的值且等于剩下值的和。直接将这些概率求和即可。

M - Random Variables

考虑容斥，计算 $\max i \le k$ 的方案数。

问题转化为求 n 个不同的球放进 m 个不同的箱子，每个箱子不超过 k 个球的方案数。

记 $f_{i,j}$ 表示 $n=i,m=j$ 时上述问题的解。有转移 $f_{i,j}=j\times f_{i-1,j}-j\times\binom{i-1}{k}\times f_{i-k-1,j-1}$，其中前半部分表示随意转移的方案数，后半部分表示钦定第 i 个球所在盒子超出限制的方案数。容易发现第二维的范围是 $O(\frac{n}{k})$ 的。因此这个 DP 的复杂度为 $O(\frac{n^2}{k})$。因此直接枚举 k 并进行 DP 即可做到 $O(n^2\ln n)$。

N - Python Program

模拟外层循环，内层循环构成一个等差数列，可以直接使用通项公式计算。

一些使得代码简单的方式：

- 使用 `scanf` 的 `%[set]` 语法读入（参考 cpp-reference）；
- 使用指针，将内层使用外层变量的部分指向同一个变量，便于计算内层贡献。

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
int I,n,*a[3],*b[3],*c[3],ans;
void proc(auto a,auto A) {
    if (!isalpha(*a)) A[n]=new int(stoi(a));
    else A[n]=&I;
}
signed main() {
    string s;
    getline(cin,s);
    for (int _:{0,1}) {
        getline(cin,s);
        char a[9],b[9],c[9]{};
        sscanf(s.c_str(),"%*[\tf]or %*c in range(%[^,)],%[^,)],%[^,)]):",a,b,c);
        if (!*c) *c='1';
        proc(a,::a);proc(b,::b);proc(c,::c);
        ++n;
    }
    int F{**c>0?1:-1};
    for (I=**a;I*F<**b*F;I+=**c) {
        int A{*a[1]},B{*b[1]},C{*c[1]},f{1};
        if (C<0) A*=-1,B*=-1,C*=-1,f*=-1;
        int D{(B-A+C-1)/C};
        if (B>A) ans+=(A*2+(D-1)*C)*D/2*f;
    }
    cout<<ans<<endl;
    return 0;
}
```