试题分析

Fudan University

September 22, 2019

A. Angle Beats

首先分两种情况考虑。

A. Angle Beats

首先分两种情况考虑。

如果 A 是直角顶点，那么就只需要对 n 个点关于 A 做一个极角排序，然后统计有多少个点对 (B, C)，使得 AB 垂直 AC，这个是很好求的。

A. Angle Beats

首先分两种情况考虑。

如果 A 是直角顶点，那么就只需要对 n 个点关于 A 做一个极角排序，然后统计有多少个点对 (B, C)，使得 AB 垂直 AC，这个是很好求的。

如果 A 不是直角顶点，考虑离线做法。先把所有的询问点读进来，然后枚举 n 个给定点，对于每个给定点 B，给其他 $n+q-1$ 个点关于 B 进行极角排序并按极角序扫描所有点，每扫到一个询问点就统计垂直方向上的给定点的个数并把答案累加到当前询问。

A. Angle Beats

首先分两种情况考虑。

如果 A 是直角顶点，那么就只需要对 n 个点关于 A 做一个极角排序，然后统计有多少个点对 (B, C)，使得 AB 垂直 AC，这个是很好求的。

如果 A 不是直角顶点，考虑离线做法。先把所有的询问点读进来，然后枚举 n 个给定点，对于每个给定点 B，给其他 $n+q-1$ 个点关于 B 进行极角排序并按极角序扫描所有点，每扫到一个询问点就统计垂直方向上的给定点的个数并把答案累加到当前询问。

时间复杂度：$O(qn\log n + n(n+q)\log(n+q))$。

B. The Tree of Haruhi Suzumiya

首先可以把 $V(A)$ 改写成 $\binom{|A|}{2}$ 减去 i 是 j 祖先且 $w_i \leq w_j$ 的点对数，再加上 A 中所有点的深度。

B. The Tree of Haruhi Suzumiya

首先可以把 $V(A)$ 改写成 $\binom{|A|}{2}$ 减去 i 是 j 祖先且 $w_i \leq w_j$ 的点对数，再加上 A 中所有点的深度。

考虑先解决 $V(B) = n$ 的情况，用树状数组在 $O(n \log n)$ 的时间求出每个点 x 有多少个它的祖先权值小于它的权值，不妨设为 a_x，则 $V(B) = n$ 时候的答案即为所有 a_x 值之和。

B. The Tree of Haruhi Suzumiya

首先可以把 $V(A)$ 改写成 $\binom{|A|}{2}$ 减去 i 是 j 祖先且 $w_i \le w_j$ 的点对数，再加上 A 中所有点的深度。

考虑先解决 $V(B) = n$ 的情况，用树状数组在 $O(n \log n)$ 的时间求出每个点 x 有多少个它的祖先权值小于它的权值，不妨设为 a_x，则 $V(B) = n$ 时候的答案即为所有 a_x 值之和。

先假设没有两点权值相等的情况，考虑从 B 中取出一个点 x 加入 A 对厌恶值带来的贡献：

B. The Tree of Haruhi Suzumiya

首先可以把 $V(A)$ 改写成 $\binom{|A|}{2}$ 减去 i 是 j 祖先且 $w_i \leq w_j$ 的点对数，再加上 A 中所有点的深度。

考虑先解决 $V(B) = n$ 的情况，用树状数组在 $O(n \log n)$ 的时间求出每个点 x 有多少个它的祖先权值小于它的权值，不妨设为 a_x，则 $V(B) = n$ 时候的答案即为所有 a_x 值之和。

先假设没有两点权值相等的情况，考虑从 B 中取出一个点 x 加入 A 对厌恶值带来的贡献：d_x

B. The Tree of Haruhi Suzumiya

首先可以把 $V(A)$ 改写成 $\binom{|A|}{2}$ 减去 i 是 j 祖先且 $w_i \le w_j$ 的点对数，再加上 A 中所有点的深度。

考虑先解决 $V(B) = n$ 的情况，用树状数组在 $O(n \log n)$ 的时间求出每个点 x 有多少个它的祖先权值小于它的权值，不妨设为 a_x，则 $V(B) = n$ 时候的答案即为所有 a_x 值之和。

先假设没有两点权值相等的情况，考虑从 B 中取出一个点 x 加入 A 对厌恶值带来的贡献：d_x 减去 $x, i (i \in A)$ 为祖孙关系且祖先点权值小于子孙点权值的点对数

B. The Tree of Haruhi Suzumiya

首先可以把 $V(A)$ 改写成 $\binom{|A|}{2}$ 减去 i 是 j 祖先且 $w_i \le w_j$ 的点对数，再加上 A 中所有点的深度。

考虑先解决 $V(B) = n$ 的情况，用树状数组在 $O(n \log n)$ 的时间求出每个点 x 有多少个它的祖先权值小于它的权值，不妨设为 a_x，则 $V(B) = n$ 时候的答案即为所有 a_x 值之和。

先假设没有两点权值相等的情况，考虑从 B 中取出一个点 x 加入 A 对厌恶值带来的贡献：d_x 减去 $x, i (i \in A)$ 为祖孙关系且祖先点权值小于子孙点权值的点对数　再减去 $x, j (j \in B)$ 为祖孙关系且祖先点权值小于子孙点权值的点对数。

B. The Tree of Haruhi Suzumiya

首先可以把 $V(A)$ 改写成 $\binom{|A|}{2}$ 减去 i 是 j 祖先且 $w_i \le w_j$ 的点对数，再加上 A 中所有点的深度。

考虑先解决 $V(B) = n$ 的情况，用树状数组在 $O(n \log n)$ 的时间求出每个点 x 有多少个它的祖先权值小于它的权值，不妨设为 a_x，则 $V(B) = n$ 时候的答案即为所有 a_x 值之和。

先假设没有两点权值相等的情况，考虑从 B 中取出一个点 x 加入 A 对厌恶值带来的贡献：d_x 减去 $x, i (i \in A)$ 为祖孙关系且祖先点权值小于子孙点权值的点对数　再减去 $x, j (j \in B)$ 为祖孙关系且祖先点权值小于子孙点权值的点对数。由于 A, B 的并集就是全集，所以这个值是个常数，我们只要把所有点按照这个增量值从小到大排序然后每次贪心地从 B 中取一个加入 A 即可。

B. The Tree of Haruhi Suzumiya

现在考虑两点权值相等的情况，此时需要在上述做法的基础上减去 A 中权值相等且为祖孙关系的点对个数。

B. The Tree of Haruhi Suzumiya

现在考虑两点权值相等的情况，此时需要在上述做法的基础上减去 A 中权值相等且为祖孙关系的点对个数。

之前提到了每个点的厌恶增量值，也定义了 a_x 表示权值小于 w_x 且是 x 的祖先的点数，现在定义 b_x 表示权值大于 w_x 且是 x 的子孙的点数，之前的增量值就等于 $d_x - a_x - b_x$。

B. The Tree of Haruhi Suzumiya

现在考虑两点权值相等的情况，此时需要在上述做法的基础上减去 A 中权值相等且为祖孙关系的点对个数。

之前提到了每个点的厌恶增量值，也定义了 a_x 表示权值小于 w_x 且是 x 的祖先的点数，现在定义 b_x 表示权值大于 w_x 且是 x 的子孙的点数，之前的增量值就等于 $d_x - a_x - b_x$。

若 u 是 v 的祖先，且 $w_u = w_v$，那么先取 u 一定不会比先取 v 差，因为有 $a_v - a_u \leq d_v - d_u$，以及 $b_u \geq b_v$，整合起来即 $d_u - a_u - b_u \leq d_v - a_v - b_v$，而且把一个祖先节点先放到 A 能给更多的子孙节点的厌恶增量值带来 -1 的贡献，所以即使有权值相等的情况也是先选祖先更优。

B. The Tree of Haruhi Suzumiya

现在考虑两点权值相等的情况，此时需要在上述做法的基础上减去 A 中权值相等且为祖孙关系的点对个数。

之前提到了每个点的厌恶增量值，也定义了 a_x 表示权值小于 w_x 且是 x 的祖先的点数，现在定义 b_x 表示权值大于 w_x 且是 x 的子孙的点数，之前的增量值就等于 $d_x - a_x - b_x$。

若 u 是 v 的祖先，且 $w_u = w_v$，那么先取 u 一定不会比先取 v 差，因为有 $a_v - a_u \leq d_v - d_u$，以及 $b_u \geq b_v$，整合起来即 $d_u - a_u - b_u \leq d_v - a_v - b_v$，而且把一个祖先节点先放到 A 能给更多的子孙节点的厌恶增量值带来 -1 的贡献，所以即使有权值相等的情况也是先选祖先更优。

定义 c_x 表示权值等于 w_x 且是它祖先的点的个数，那么所有点按照 $d_x - a_x - b_x - c_x$ 从小到大排序后贪心取就可以了。

B. The Tree of Haruhi Suzumiya

现在考虑两点权值相等的情况，此时需要在上述做法的基础上减去 A 中权值相等且为祖孙关系的点对个数。

之前提到了每个点的厌恶增量值，也定义了 a_x 表示权值小于 w_x 且是 x 的祖先的点数，现在定义 b_x 表示权值大于 w_x 且是 x 的子孙的点数，之前的增量值就等于 $d_x - a_x - b_x$。

若 u 是 v 的祖先，且 $w_u = w_v$，那么先取 u 一定不会比先取 v 差，因为有 $a_v - a_u \leq d_v - d_u$，以及 $b_u \geq b_v$，整合起来即 $d_u - a_u - b_u \leq d_v - a_v - b_v$，而且把一个祖先节点先放到 A 能给更多的子孙节点的厌恶增量值带来 -1 的贡献，所以即使有权值相等的情况也是先选祖先更优。

定义 c_x 表示权值等于 w_x 且是它祖先的点的个数，那么所有点按照 $d_x - a_x - b_x - c_x$ 从小到大排序后贪心取就可以了。

时间复杂度：$O(n \log n)$。

C. Sakurada Reset

首先考虑如何判断一个字符串 s 是否是某个字符串 t 的子序列：贪心匹配，即首先找到 s 的第一个与 t_1 相等的字符 s_{p_1}，然后找到位置 $> p_1$ 的与 t_2 相等的字符 s_{p_2}，以此类推。如果 t 的所有字符都能被这样子匹配上，则 t 是 s 的一个子序列。

C. Sakurada Reset

首先考虑如何判断一个字符串 s 是否是某个字符串 t 的子序列：贪心匹配，即首先找到 s 的第一个与 t_1 相等的字符 s_{p_1}，然后找到位置 $> p_1$ 的与 t_2 相等的字符 s_{p_2}，以此类推。如果 t 的所有字符都能被这样子匹配上，则 t 是 s 的一个子序列。

考虑求一个串的本质不同的子序列个数：$dp[i]$ 表示贪心匹配最后会匹配到原串中的第 i 个位置的方案数，则有：

C. Sakurada Reset

首先考虑如何判断一个字符串 s 是否是某个字符串 t 的子序列：贪心匹配，即首先找到 s 的第一个与 t_1 相等的字符 s_{p_1}，然后找到位置 $> p_1$ 的与 t_2 相等的字符 s_{p_2}，以此类推。如果 t 的所有字符都能被这样子匹配上，则 t 是 s 的一个子序列。

考虑求一个串的本质不同的子序列个数：$dp[i]$ 表示贪心匹配最后会匹配到原串中的第 i 个位置的方案数，则有：

$$dp[i] = \sum_{j=last[i]}^{i-1} dp[j]$$

C. Sakurada Reset

首先考虑如何判断一个字符串 s 是否是某个字符串 t 的子序列：贪心匹配，即首先找到 s 的第一个与 t_1 相等的字符 s_{p_1}，然后找到位置 $> p_1$ 的与 t_2 相等的字符 s_{p_2}，以此类推。如果 t 的所有字符都能被这样子匹配上，则 t 是 s 的一个子序列。

考虑求一个串的本质不同的子序列个数：$dp[i]$ 表示贪心匹配最后会匹配到原串中的第 i 个位置的方案数，则有：

$$dp[i] = \sum_{j=last[i]}^{i-1} dp[j]$$

其中 $last[i]$ 表示上一个与第 i 个位置字符相同的位置，不存在则记为 0。前缀和优化可以使这部分时间复杂度为 $O(n)$。

C. Sakurada Reset

对于这道题，分两种情况考虑：

C. Sakurada Reset

对于这道题，分两种情况考虑：

- 第 1 种是选出的两个子序列长度不同，等价于对每个字符串统计给定长度的子序列个数，只要给之前的 dp 加一维长度就可以做到 $O(n^2 + m^2)$。

C. Sakurada Reset

对于这道题，分两种情况考虑：

- 第 1 种是选出的两个子序列长度不同，等价于对每个字符串统计给定长度的子序列个数，只要给之前的 dp 加一维长度就可以做到 $O(n^2 + m^2)$。
- 第 2 种情况是选出的两个子序列长度相同，那么考虑枚举两个子序列相同部分的前缀，$f[i][j]$ 表示在第 1 个串中匹配到了位置 i，在第 2 个串中匹配到了位置 j 的方案数，其中每一项都可以由一个矩形内的 f 值转移过来，二维前缀和优化后时间复杂度可以做到 $O(nm)$。

C. Sakurada Reset

对于这道题，分两种情况考虑：

- 第 1 种是选出的两个子序列长度不同，等价于对每个字符串统计给定长度的子序列个数，只要给之前的 dp 加一维长度就可以做到 $O(n^2 + m^2)$。
- 第 2 种情况是选出的两个子序列长度相同，那么考虑枚举两个子序列相同部分的前缀，$f[i][j]$ 表示在第 1 个串中匹配到了位置 i，在第 2 个串中匹配到了位置 j 的方案数，其中每一项都可以由一个矩形内的 f 值转移过来，二维前缀和优化后时间复杂度可以做到 $O(nm)$。
  之后就是枚举第 1 个不同的位置，即枚举 i, j，且 $A_i > B_j$，则前面相等的前缀数同样也可以由 f 的一个矩阵转移。然后考虑在后面填上长度相等的后缀，这部分的做法和求 f 是类似的，区别在于要倒着算且不需要每一位都对应相等。

C. Sakurada Reset

对于这道题，分两种情况考虑：

- 第 1 种是选出的两个子序列长度不同，等价于对每个字符串统计给定长度的子序列个数，只要给之前的 dp 加一维长度就可以做到 $O(n^2 + m^2)$。
- 第 2 种情况是选出的两个子序列长度相同，那么考虑枚举两个子序列相同部分的前缀，$f[i][j]$ 表示在第 1 个串中匹配到了位置 i，在第 2 个串中匹配到了位置 j 的方案数，其中每一项都可以由一个矩形内的 f 值转移过来，二维前缀和优化后时间复杂度可以做到 $O(nm)$。
  之后就是枚举第 1 个不同的位置，即枚举 i, j，且 $A_i > B_j$，则前面相等的前缀数同样也可以由 f 的一个矩阵转移。然后考虑在后面填上长度相等的后缀，这部分的做法和求 f 是类似的，区别在于要倒着算且不需要每一位都对应相等。

时间复杂度：$O((n+m)^2)$。

D. Decimal

如果 n 只有 2 和 5 两种质因子，则 $\frac{1}{n}$ 不是无限小数。

D. Decimal

如果 n 只有 2 和 5 两种质因子，则 $\frac{1}{n}$ 不是无限小数。

时间复杂度：$O(T \log n)$。

E. Escape

每个格子的水平方向和竖直方向都只能被使用一次，因为两个机器人的路径不可能合并，也不可能迎面相撞。

E. Escape

每个格子的水平方向和竖直方向都只能被使用一次，因为两个机器人的路径不可能合并，也不可能迎面相撞。

如果一个格子没有放转弯装置，则可以被水平穿过一次，竖直穿过一次。如果一个格子放了转弯装置，则这个格子只能被一个机器人经过一次。

E. Escape

每个格子的水平方向和竖直方向都只能被使用一次，因为两个机器人的路径不可能合并，也不可能迎面相撞。

如果一个格子没有放转弯装置，则可以被水平穿过一次，竖直穿过一次。如果一个格子放了转弯装置，则这个格子只能被一个机器人经过一次。

所以对于所有非障碍格子，可以拆成水平点和竖直点，每个点限流 1，上下相邻的格子连竖直点（竖直直行），左右相邻的格子连水平点（水平直行），格子内部的水平点和竖直点互相相连（转弯），源连向起点的竖直点，出口的竖直点连向汇，跑最大流，如果最大流 = 机器人个数，则输出 Yes，否则输出 No。

E. Escape

每个格子的水平方向和竖直方向都只能被使用一次，因为两个机器人的路径不可能合并，也不可能迎面相撞。

如果一个格子没有放转弯装置，则可以被水平穿过一次，竖直穿过一次。如果一个格子放了转弯装置，则这个格子只能被一个机器人经过一次。

所以对于所有非障碍格子，可以拆成水平点和竖直点，每个点限流 1，上下相邻的格子连竖直点（竖直直行），左右相邻的格子连水平点（水平直行），格子内部的水平点和竖直点互相相连（转弯），源连向起点的竖直点，出口的竖直点连向汇，跑最大流，如果最大流 = 机器人个数，则输出 Yes，否则输出 No。

时间复杂度：$O(T\text{maxflow}(n, n))$。

F. Forest Program

要将仙人掌变成树（或者森林），只需要保证对于仙人掌中的每个环，至少有一条边被删去即可。

F. Forest Program

要将仙人掌变成树（或者森林），只需要保证对于仙人掌中的每个环，至少有一条边被删去即可。

设图中环的大小分别为 c_1，c_2，...，c_k，不属于任何一个环的边数为 b，则答案为：

F. Forest Program

要将仙人掌变成树（或者森林），只需要保证对于仙人掌中的每个环，至少有一条边被删去即可。

设图中环的大小分别为 c_1，c_2，...，c_k，不属于任何一个环的边数为 b，则答案为：

$$2^b \prod_{i=1}^{k} (2^{c_i} - 1)$$

F. Forest Program

要将仙人掌变成树（或者森林），只需要保证对于仙人掌中的每个环，至少有一条边被删去即可。

设图中环的大小分别为 c_1，c_2，...，c_k，不属于任何一个环的边数为 b，则答案为：

$$2^b \prod_{i=1}^{k} (2^{c_i} - 1)$$

时间复杂度：$O(n+m)$。

G. Game on Chessboard

对于只拿一个的拿法，因为这是最差的拿法，所以我们完全可以留到不得不拿（左下角没有别的棋子，且卡住了最优方案中要拿的某组黑白棋子对）的时候再去拿。

G. Game on Chessboard

对于只拿一个的拿法，因为这是最差的拿法，所以我们完全可以留到不得不拿（左下角没有别的棋子，且卡住了最优方案中要拿的某组黑白棋子对）的时候再去拿。

很显然任何时候的状态都可以用一条从左上到右下的一条只向右或者向下的分界线表示，所以可以状压 dp，有 $\binom{2n}{n}$ 种状态，每种状态有 n^2 种后继状态，所以基础复杂度就达到了 $\binom{2n}{n}n^2$，有 10^7 的数量级，所以状态访问和状态转移分别都得做到 $O(1)$。

G. Game on Chessboard

对于只拿一个的拿法，因为这是最差的拿法，所以我们完全可以留到不得不拿（左下角没有别的棋子，且卡住了最优方案中要拿的某组黑白棋子对）的时候再去拿。

很显然任何时候的状态都可以用一条从左上到右下的一条只向右或者向下的分界线表示，所以可以状压 dp，有 $\binom{2n}{n}$ 种状态，每种状态有 n^2 种后继状态，所以基础复杂度就达到了 $\binom{2n}{n}n^2$，有 10^7 的数量级，所以状态访问和状态转移分别都得做到 $O(1)$。

题解的做法是用 01 串表示这样的分界线（0 表示向下，1 表示向右）并用一个 int 来表示，这样状态空间最多是 2^{2n}，开个数组就行，那么状态访问就是 $O(1)$ 的。然后每个 “01” 对应一个合法的格子，去掉某个格子只需要把 “01” 变成 “10”，状态转移也可以做到 $O(1)$。

G. Game on Chessboard

对于只拿一个的拿法，因为这是最差的拿法，所以我们完全可以留到不得不拿（左下角没有别的棋子，且卡住了最优方案中要拿的某组黑白棋子对）的时候再去拿。

很显然任何时候的状态都可以用一条从左上到右下的一条只向右或者向下的分界线表示，所以可以状压 dp，有 $\binom{2n}{n}$ 种状态，每种状态有 n^2 种后继状态，所以基础复杂度就达到了 $\binom{2n}{n}n^2$，有 10^7 的数量级，所以状态访问和状态转移分别都得做到 $O(1)$。

题解的做法是用 01 串表示这样的分界线（0 表示向下，1 表示向右）并用一个 int 来表示，这样状态空间最多是 2^{2n}，开个数组就行，那么状态访问就是 $O(1)$ 的。然后每个 “01” 对应一个合法的格子，去掉某个格子只需要把 “01” 变成 “10”，状态转移也可以做到 $O(1)$。

时间复杂度：$O(\binom{2n}{n}n^2)$。

H. Houraisan Kaguya

首先定义小于质数 p 的正整数 a 关于 p 的阶定义为使得 $a^l \equiv 1(\mathrm{mod}\ p)$ 的最小的正整数 l，记作 $ord(a)$，那么可以证明：

H. Houraisan Kaguya

首先定义小于质数 p 的正整数 a 关于 p 的阶定义为使得 $a^l \equiv 1(\mathrm{mod}\ p)$ 的最小的正整数 l，记作 $ord(a)$，那么可以证明：

$$f(a,b) = \frac{ord(a)}{\gcd(ord(a), ord(b))}$$

H. Houraisan Kaguya

首先定义小于质数 p 的正整数 a 关于 p 的阶定义为使得 $a^l \equiv 1(\mathrm{mod}\ p)$ 的最小的正整数 l，记作 $ord(a)$，那么可以证明：

$$f(a,b) = \frac{ord(a)}{\gcd(ord(a), ord(b))}$$

考虑如何求一个数的阶。首先模 p 意义下的阶一定是 $p-1$ 的约数，所以我们可以对 $p-1$ 质因数分解，然后求阶在关于 $p-1$ 的每个质因子的幂次。具体就是枚举 $p-1$ 的每个质因子 q_i，不妨设其对应的指数为 u_i，然后从小到大枚举 q_i 的指数 t_i，如果到了某个 t_i 满足 $a^{\frac{p-1}{q_i^{u_i}} \times q_i^{t_i}} \equiv 1(\mathrm{mod}\ p)$，则表明 a 的阶在 q_i 下的指数就是 t_i。

H. Houraisan Kaguya

记 $c(x)$ 为 x 的不同质因子个数，那么求一次阶要枚举 $c(p-1)$ 个质因子，对于每个质因子 q_i，都要用 $O(\log p)$ 的时间求出 $a^{\frac{q-1}{{q_i}^{u_i}}}$ 的值，而每次枚举指数可以 $O(\log q_i)$ 转移，所以转移的总复杂度是 $O(\log {q_i}^{u_i})$ 的，总之这样暴力求一次阶的总复杂度是 $O(c(p-1)\log p)$ 的。但预处理 a 的幂次的时候可以用分治来优化，复杂度可以降到 $O(\log c(p-1)\log p)$。

H. Houraisan Kaguya

记 $c(x)$ 为 x 的不同质因子个数，那么求一次阶要枚举 $c(p-1)$ 个质因子，对于每个质因子 q_i，都要用 $O(\log p)$ 的时间求出 $a^{\frac{q-1}{{q_i}^{u_i}}}$ 的值，而每次枚举指数可以 $O(\log q_i)$ 转移，所以转移的总复杂度是 $O(\log {q_i}^{u_i})$ 的，总之这样暴力求一次阶的总复杂度是 $O(c(p-1)\log p)$ 的。但预处理 a 的幂次的时候可以用分治来优化，复杂度可以降到 $O(\log c(p-1)\log p)$。

现在把所有的阶都求出来了，考虑用以下的式子算答案：

$$\sum_{d}\sum_{i=1}^{n}\sum_{j=1}^{n}[\gcd(ord(a),ord(b))=d]\frac{ord(a)ord(b)}{d^2}$$

H. Houraisan Kaguya

如果把 gcd 用 $p-1$ 的质因子的幂次来表示，那么这个过程实际上是一个“按位取 min”，和按位与本质是一样的，所以可以稍加改动把与卷积改成“按位 min”卷积，就可以求出上式中每一项的贡献了。

H. Houraisan Kaguya

如果把 gcd 用 $p-1$ 的质因子的幂次来表示，那么这个过程实际上是一个 “按位取 min”，和按位与本质是一样的，所以可以稍加改动把与卷积改成 “按位 min” 卷积，就可以求出上式中每一项的贡献了。

时间复杂度：$O(p^{\frac{1}{4}} + n \log c(p-1) \log p + d(p-1)c(p-1))$。

其中 $c(x)$ 为 x 的不同质因子个数，$d(x)$ 为 x 的约数个数，可以验证在 10^{18} 范围内的质数 p 的 $c(p-1)$ 不超过 15，$d(p-1)$ 不超过 10^5。

I. Invoker

记 $f[i][6]$ 为祈唤出第 i 个技能之后，身上三个法球的先后顺序为 $0 \sim 5$ 的状态的最少按键数。

I. Invoker

记 $f[i][6]$ 为祈唤出第 i 个技能之后，身上三个法球的先后顺序为 $0 \sim 5$ 的状态的最少按键数。

转移就暴力枚举上一个技能的结尾状态，然后算一下有几个法球是可以重复使用的，取个最优值就行了。

I. Invoker

记 $f[i][6]$ 为祈唤出第 i 个技能之后，身上三个法球的先后顺序为 $0 \sim 5$ 的状态的最少按键数。

转移就暴力枚举上一个技能的结尾状态，然后算一下有几个法球是可以重复使用的，取个最优值就行了。

时间复杂度：$O(n)$。

J. MUV LUV EXTRA

枚举循环节已出现的长度 p，最优的循环节就是最后 p 个字符构成的字符串的最短周期。

J. MUV LUV EXTRA

枚举循环节已出现的长度 p，最优的循环节就是最后 p 个字符构成的字符串的最短周期。

考虑把字符串倒过来，使用 kmp 可以求出每个前缀的最短周期，即求出了原串每个后缀的最短周期。

J. MUV LUV EXTRA

枚举循环节已出现的长度 p，最优的循环节就是最后 p 个字符构成的字符串的最短周期。

考虑把字符串倒过来，使用 kmp 可以求出每个前缀的最短周期，即求出了原串每个后缀的最短周期。

时间复杂度：$O(|s|)$。

K. MUV LUV UNLIMITED

如果存在一个叶子节点 x，且它的父亲节点的出度大于 1，那么先手一定必胜。考虑当先手只取 x 这一个节点后：

K. MUV LUV UNLIMITED

如果存在一个叶子节点 x，且它的父亲节点的出度大于 1，那么先手一定必胜。考虑当先手只取 x 这一个节点后：

- 变成先手必败态，那么当前先手自然是必胜的

K. MUV LUV UNLIMITED

如果存在一个叶子节点 x，且它的父亲节点的出度大于 1，那么先手一定必胜。考虑当先手只取 x 这一个节点后：

- 变成先手必败态，那么当前先手自然是必胜的
- 变成先手必胜态，那么存在一个方案使得去掉 x 以及其他某个叶子节点集合 S 后，能够到达一个先手必败态。而由于删除 x 后没有产生其他叶子节点，即 S 中所有点在删除 x 前就已经是叶子节点了，所以先手可以直接删除 $\{x\} \bigcup S$ 从而转移到先手必败态

K. MUV LUV UNLIMITED

接下来考虑所有叶子节点的父亲的出度都等于 1 的情况。

K. MUV LUV UNLIMITED

接下来考虑所有叶子节点的父亲的出度都等于 1 的情况。

求出每个叶子节点的链长（即到达第一个出度不为 1 的祖先需要经过多少条边）。

K. MUV LUV UNLIMITED

接下来考虑所有叶子节点的父亲的出度都等于 1 的情况。

求出每个叶子节点的链长（即到达第一个出度不为 1 的祖先需要经过多少条边）。

如果所有链长均为偶数则先手必败，否则先手必胜。

K. MUV LUV UNLIMITED

接下来考虑所有叶子节点的父亲的出度都等于 1 的情况。

求出每个叶子节点的链长（即到达第一个出度不为 1 的祖先需要经过多少条边）。

如果所有链长均为偶数则先手必败，否则先手必胜。

其中必胜的策略为将所有链长为奇数的叶子删去使得他们链长变为偶数。

K. MUV LUV UNLIMITED

接下来考虑所有叶子节点的父亲的出度都等于 1 的情况。

求出每个叶子节点的链长（即到达第一个出度不为 1 的祖先需要经过多少条边）。

如果所有链长均为偶数则先手必败，否则先手必胜。

其中必胜的策略为将所有链长为奇数的叶子删去使得他们链长变为偶数。

时间复杂度：$O(\sum n)$。

L. MUV LUV ALTERNATIVE

首先每个士兵可以看做一个二元组 (L_i, R_i)，其中 L_i, R_i 分别表示从左通道以及右通道逃脱的时刻。如果士兵只能从左通道逃脱则令 $R_i = +\infty$，只能从右通道逃脱则令 $L_i = +\infty$。

L. MUV LUV ALTERNATIVE

首先每个士兵可以看做一个二元组 (L_i, R_i)，其中 L_i, R_i 分别表示从左通道以及右通道逃脱的时刻。如果士兵只能从左通道逃脱则令 $R_i = +\infty$，只能从右通道逃脱则令 $L_i = +\infty$。

考虑二分答案，即能否在不超过 x 的时间内全部逃脱。将二元组按照 R 从大到小排序，依次贪心的让其走左通道，如果左通道超出 x 的限制，则让其走右通道，看最后是否所有人都能在 x 的时间内逃脱。

L. MUV LUV ALTERNATIVE

首先每个士兵可以看做一个二元组 (L_i, R_i)，其中 L_i, R_i 分别表示从左通道以及右通道逃脱的时刻。如果士兵只能从左通道逃脱则令 $R_i = +\infty$，只能从右通道逃脱则令 $L_i = +\infty$。

考虑二分答案，即能否在不超过 x 的时间内全部逃脱。将二元组按照 R 从大到小排序，依次贪心的让其走左通道，如果左通道超出 x 的限制，则让其走右通道，看最后是否所有人都能在 x 的时间内逃脱。

注意每个通道每个时刻只能有一名士兵通过，所以得用并查集或者 set 来快速求出每个时刻之后的第一个空闲时刻。

L. MUV LUV ALTERNATIVE

首先每个士兵可以看做一个二元组 (L_i, R_i)，其中 L_i, R_i 分别表示从左通道以及右通道逃脱的时刻。如果士兵只能从左通道逃脱则令 $R_i = +\infty$，只能从右通道逃脱则令 $L_i = +\infty$。

考虑二分答案，即能否在不超过 x 的时间内全部逃脱。将二元组按照 R 从大到小排序，依次贪心的让其走左通道，如果左通道超出 x 的限制，则让其走右通道，看最后是否所有人都能在 x 的时间内逃脱。

注意每个通道每个时刻只能有一名士兵通过，所以得用并查集或者 set 来快速求出每个时刻之后的第一个空闲时刻。

时间复杂度：$O(k\alpha(k)\log n)$ 或者 $O(k\log k\log n)$。

L. MUV LUV ALTERNATIVE

正确性说明：

L. MUV LUV ALTERNATIVE

正确性说明：

假设按 R 从大到小排序后存在一种方案使得所有士兵在不超过 x 时间全部逃脱，假设选择的序列是 $A_1, A_2, \cdots, A_n$。其中 $A_i = L$ or R。

L. MUV LUV ALTERNATIVE

正确性说明：

假设按 R 从大到小排序后存在一种方案使得所有士兵在不超过 x 时间全部逃脱，假设选择的序列是 $A_1, A_2, \cdots, A_n$。其中 $A_i = L$ or R。

找到最小的 i 满足 $A_i = R$ 且在只考虑 $A_1 \sim A_{i-1}$ 这 $i-1$ 个士兵时，第 i 个士兵是可以选择 L 的。

L. MUV LUV ALTERNATIVE

正确性说明：

假设按 R 从大到小排序后存在一种方案使得所有士兵在不超过 x 时间全部逃脱，假设选择的序列是 $A_1, A_2, \cdots, A_n$。其中 $A_i = L$ or R。

找到最小的 i 满足 $A_i = R$ 且在只考虑 $A_1 \sim A_{i-1}$ 这 $i-1$ 个士兵时，第 i 个士兵是可以选择 L 的。

此时将 A_i 替换成 L，那么在接下来的 A 中，最多只需要将一个 $A_j = L$ 改成 $A_j = R$，其中 $i < j \leq n$，且由于 $R_j \leq R_i$，右通道所需要花费的时间不会变多。

L. MUV LUV ALTERNATIVE

正确性说明：

假设按 R 从大到小排序后存在一种方案使得所有士兵在不超过 x 时间全部逃脱，假设选择的序列是 $A_1, A_2, \cdots, A_n$。其中 $A_i = L$ or R。

找到最小的 i 满足 $A_i = R$ 且在只考虑 $A_1 \sim A_{i-1}$ 这 $i-1$ 个士兵时，第 i 个士兵是可以选择 L 的。

此时将 A_i 替换成 L，那么在接下来的 A 中，最多只需要将一个 $A_j = L$ 改成 $A_j = R$，其中 $i < j \leq n$，且由于 $R_j \leq R_i$，右通道所需要花费的时间不会变多。

即如果当前二分的 x 存在一个合法方案，那么是可以将该合法方案通过以上操作逐步转变到题解算法所求的方案的。

谢谢大家！