氨基酸序列

简要题意：有一个定义在 L 元集合 S 上的数组 A，初始时值全为 0，有两种操作：

- 给定子集 $T \subseteq S$，令 $A[T] \leftarrow A[T] + v$。
- 给定子集 $T \subseteq U \subseteq S$，求 $\sum\limits_{T \subseteq I \subseteq U} A[I]$。

算法零

xpp。

期望得分 3 分。

算法一

对于 $mask$ 中不含 `*`，即 $T = S$ 的情形，不难发现整个问题没有用到集合的性质，因此可以把 2^L 个集合通过二进制编码看成 $0 \sim 2^L - 1$ 的普通下标，然后转化为裸的单点修改单点查询的问题，使用**数组**这一数据结构即可。

期望得分 9 分。

算法二

对于子任务 3, 4, 5，只需要暴力枚举满足 $T \subseteq I \subseteq U$ 集合 I，然后通过算法一将其相加，即可通过。

结合算法一，或者将暴力枚举改成更加优秀的**子集枚举**，期望得分 34 分。

算法三

对于 `Q` 在 `I` 之后的部分，相当于对一个固定的数组做询问，故我们可以对这个已知的数组进行一些预处理以得到更优秀的询问效果。

若在满足 `Q` 在 `I` 之后的条件下，又满足 $mask$ 中不含 1 (即 $T = \varnothing$)，不难发现每次是询问一个子集和。

于是可以使用子集和变换 (快速 Möbius 变换)。

那如果 $mask$ 中含 1，那么显然 3^n 的子集和就报废了，不过根据套路，我们可以想到对这些强制为 1 的位置进行容斥 (即用所有的减去是 0 的)。

这样就把问题转化为 2^c 个 $mask$ 中不含 1 的子问题了，其中 $c = |T|$ 是 $mask$ 中 1 的个数。

那万一 c 太大怎么办？注意到 0 和 1 的地位实际上是相同的，因此如果 1 的个数太多，那么 0 的个数一定不会太多，于是我们做一个 "超集和变换" (即反着的快速 Möbius 变换) 即可。

这样单次询问的复杂度不超过 $O\left(2^{L/2}\right)$，是可以通过的 (因为只有加法，没有取模，还是比较快的)。

当然，如果你做题百万，你能做到一个叫做 [JOI 2018 Final]毒蛇越狱 的题，使用那题的算法亦可，这样单次询问的复杂度就是 $O\left(2^{L/3}\right)$。~~(其实就是继续利用 * 的个数进一步阈值)~~。

可以通过第 7, 10 个子任务，期望得分 58 分。

算法四

对于只有 $mask$ 中不含 1 的部分，相当于动态维护子集和，而这是可以在 $O\left(q\cdot 2^{L/2}\right)$ 时间内完成的，具体做法如下：

具体地，我们将 S "分块"：分成大小均为 $\frac{L}{2}$ 的两部分 S_1, S_2。然后还是维护子集和，不过只对 S_2 部分求子集和。

这样，修改的时候，我们只需要对 S_2 部分进行枚举即可，查询的时候，再对 S_1 部分求个和即可。时间复杂度均为 $O\left(2^{L/2}\right)$。

可以通过第 8, 11 个子任务，期望得分 58 分。结合算法三，期望得分 82 分。

算法五 (std)

考虑结合算法三和算法四，将序列分为等长的五段：S_1, S_2, S_3, S_4, S_5。

然后对 S_3, S_4, S_5 部分求子集和与超集和。

考虑修改，只需要对 S_3, S_4, S_5 部分进行枚举，这部分复杂度为 $O\left(2^{3L/5}\right)$。

对于询问，先枚举 S_1, S_2，然后根据算法三，可知剩下的 $\frac{3}{5}L$ 至多只需要枚举其中的 $\frac{1}{3}$，即 $\frac{L}{5}$，故单次总复杂度也是 $O\left(2^{3L/5}\right)$。

这样总时间复杂度 $O\left(q\cdot 2^{3L/5}\right)$，可以通过所有子任务。

(ps: 经实测，按照 $7+11(4+4+3)$ 的分块是最优的)

算法六 (std 优化)

考虑如果我们能维护出子集和，那么根据算法三，单次询问的复杂度不超过 $O\left(2^{L/2}\right)$。

注意到，如果 seq 中 1 的个数 $|T| \geq \dfrac{L}{2}$，那么暴力更新的复杂度是对的。

因此，如果所有的 seq 都满足这个性质，就可以得到一个较为优(bao)美(li)的 $O\left(q \cdot 2^{L/2}\right)$ 做法。

但是并不是所有 seq 的 1 的个数都不小于 $\dfrac{L}{2}$。

此时，我们可以把这些数的贡献分开来算，看成 (seq 中 1 的个数 $\geq \dfrac{L}{2}$ 的贡献和) 加上 (seq 中 1 的个数 $\leq \dfrac{L}{2}$ 的贡献和)。

对于后半部分的贡献和，由于 0 和 1 地位相同，改成超集和即可。

这样总时间复杂度 $O\left(q \cdot 2^{L/2}\right)$，可以通过所有子任务。参考提交。

算法七 (尚未实现)

对整个操作序列分块，设每 B 个为一块，然后每块过后按照算法三暴力重构子集和数组以及超集和数组。

对于一组询问，将最近的块的答案通过算法三得到，然后暴力枚举当前块的前缀询问对它的贡献。

考虑时间复杂度，每一组询问的复杂度为 $O\left(2^{L/3}+B\right)$。

重构复杂度 $O\left(2^L \cdot L\right)$，需要重构 $O\left(\dfrac{q}{B}\right)$ 次。

故总时间复杂度 $O\left(q \cdot \left(2^{L/3}+B\right)+2^L \cdot L \cdot \dfrac{q}{B}\right)$，不难发现当 $B=O\left(2^{L/2}\sqrt{L}\right)$ 时最优，此时时间复杂度 $O\left(q \cdot 2^{L/2}\sqrt{L}+q \cdot 2^{L/3}\right)$，常数可能有些大，但一般可以通过 ~~(有没有老哥来写的？)~~

Bonus (unsolved)

本题可以在低于 $O\left(q \cdot 2^{L/2}\right)$ 时间内解决吗？

反·易题

简要题意：给定一个 Hash 算法和它跑出的若干结果，请还原 $seed$ 的值及剩下的输出。

首先，不难发现如果我们知道了 $seed$，那么只需要模(mo)拟(gai)一(chk)遍就可以获得全部的输出，因此以下只考虑如何求 $seed$。

通法貌似也挺少，那么就直接讲面向数据的算法了。

测试点 1, 2

嗯哼？怎么全是 Yes？

参考原题，得知应该是 Hash 算法写挂了，事实上只要 $p = q$ 那么所有长度相等的串一定相等。

怎么让 $p = q$ 呢？$seed = 0$ 即可，此时 Xor-shift 生成出来的数全是 0。

测试点 3 ~ 8

这几个点的模数都不算太大，因此如果随机数生成器是均匀的话，我们可以直接考虑枚举 $seed$。

在期望 $O(m)$ 次枚举后即可找到答案。

事实上，我们可以通过写 SA/SAM ~~(换一个底数)~~ 找到所有使得冲突的询问 (这样的询问并不多)，来加快检验速度。

由于是提交答案，这个时间是可接受的。

如果你觉得 $7, 8$ 两个点还是太慢，可以参考后面对 $16 \sim 19$ 的做法。

测试点 11

如果你写过 SA/SAM ~~(换一个底数)~~，那么你会发现这个点没有冲突。

而这个点的模数又很大，因此由生日攻击知随机底数冲突的概率非常小，于是随机几次就可以了 (大多数情况下是随机一次就可以，比如 $seed = 1$ 就行)。

中场

接下来我们需要稍稍进一步分析题目的性质。

考虑两个串 $s_0 s_1 \cdots s_{k-1}, t_0 t_1 \cdots t_{k-1}$，什么时候它们会冲突。

我们不妨设 s_0, t_0 就是 p 或 q。若 $p \equiv q \pmod{m}$，那么显然永远冲突，下设 $p \not\equiv q \pmod{m}$。

由定义，有 $\sum_{i=0}^{k-1} s_i b^i \equiv \sum_{i=0}^{k-1} t_i b^i \pmod m$

即 $\sum_{i=0}^{k-1} (s_i - t_i) b^i \equiv 0 \pmod m$

即如果 $s_i = t_i$，我们不用理会它，如果 $s_i \neq t_i$，那么根据 s_i 是左括号还是右括号可知它要么等于 $p - q$，要么等于 $q - p$。

又 $p \neq q$ 且 m 是素数，故我们可以两端同时除以 $p - q$，即将 $p - q$ 看成 1，$q - p$ 看成 -1。

也就是说，对于所有 s 为 `(`，t 为 `)` 的位置，在这一位记 1；所有 s 为 `)`，t 为 `(` 的位置，在这一位记 -1；其它位记 0。

于是我们就得到了一个关于 b 的 $k-1$ 次多项式 $P(x)$，其中每一项的系数为 $-1, 0, 1$。

b 是这个多项式在模意义下的根。

反之，如果 b 满足 $P(b) = 0$，那么 b 也一定会造成这一组数据的冲突。

那么接下来我们就忽略 p, q，直接讨论 b。

在解出 b 之后，我们注意到题目要求的是 $seed$ 不是 b，于是我们希望找到一个 $seed$ 来生成所需的 b。

观察 Xor-shift 生成器，不难注意到这个过程是可逆的，且它的逆函数也容易求得。

比如，对于 `x ^= x >> 12`，我们按位考虑，不难发现它其实是做一个每 12 个二进制位为一组的**差分**，于是逆运算只需要做**部分和**即可。

一个较简单的实现是利用倍增：`x ^= x >> 12, x ^= x >> 24, x ^= x >> 48`。

于是我们可以求出函数 Next_int 的逆，于是先假设一个 b 的时候的 $seed$，然后倒着做 11 次即可得到 $seed$。因此，**下面默认我们只需要求 b**。

(ps: 提交答案题的一个好处是你可以使用各种手段，比如拉代码/轮子，而无人知道，这会对后面几个点有大量帮助。~~std 生非异也，善假于物也~~)

测试点 12

不难发现多项式 $P(x) = x^2 + 1$，于是只需要解方程 $b^2 + 1 \equiv 0 \pmod m$。

这是一个裸的二次剩余问题，于是使用 Cipolla 等算法求解即可。

测试点 13

不难发现多项式 $P(x) = \sum_{i=0}^{1037} x^i$。经尝试，发现当 $b = 1$ 时不满足条件，因此两边乘以 $1 - x$ 不影响结果。

于是我们只需要求解 $(1-b)P(b)=1-b^{1038}\equiv 0$，类比于单位根可知，$b$ 是一个 1038 次单位根，求出 m 的原根后快速幂即可。

注意这里我们要分解 $m-1$ (显然有 $1038\mid m-1$)，这里可以使用 Linux bash 自带的 `factor` 工具。

测试点 9, 10

经过观察，可以发现这两组数据的串为 parity 串是一种用来卡自然溢出 Hash 算法的串。

然而这道题目并没有出现合数模 (自然溢出为合数模)，因此我们需要分析这个串卡掉自然溢出的本质。

通过上述多项式的角度去分析，不难发现并证明，当串的长度为 2^k 时，$P(b)=(1-b)\left(1-b^2\right)\left(1-b^4\right)\cdots\left(1-b^{2^{k-1}}\right)$。

而一个数在 $(\bmod\ 2^n)$ 条件下的阶也不过是 2^{n-2}，因此取 $k=O(\sqrt{n})$ 即可 Hack 掉。

那么，在这两个测试点中，我们找到符合要求的长度为 2^k 的串，如果发现它冲突了，说明这个底数的阶不超过 2^{k-1}；反之，如果没有冲突，说明底数的阶超过 2^{k-1}。

于是，只需要对每个 k 找到这样一个例子，然后确定答案模 m 的阶 $\delta_m(b)$，发现这个阶并不是很大，在阶为 $\delta_m(b)$ 的元素中枚举即可 (好像随机一个就有很大概率没有其它冲突)。

找到所有阶为 $\delta_m(b)$ 的元素可以通过测试点 13 所述的找原根的方法。

测试点 14, 15

不难发现只有一组冲突，且 $P(x)$ 的次数分别为 $8,83$。

于是我们需要完成模意义下解 100 次以内方程的功能。

测试点 16 ~ 19

发现方程的次数很大，直接解方程似乎非常慢。不过，这里有两组冲突，分别记多项式为 $P(x),Q(x)$。

注意到 $P(b)\equiv Q(b)\equiv 0\ (\bmod\ m)$，这说明 b 是 $P(x)$ 和 $Q(x)$ 的公共根 —— 换句话说，b 是多项式 $\gcd(P(x),Q(x))$ 的一个根。

故我们需要求两个多项式的最大公因式。由于数据较为随机，因此最大公因式的次数往往非常低，或者就是一次，对于前者套用 14, 15 的算法，后者就直接得到解了。

测试点 20

需要单独解一个 1000 次方程，可以尝试对 $14, 15$ 部分的算法进行卡常。

如果你还是解不出来，可以上网找相应的解方程工具

总结

总之，这个题还是蛮有意思的，考察了选手对字符串 Hash、位运算、数论、多项式的多方面的掌握，是一道不失有趣的提交答案题。

小 ω 的魔方

简要题意：定义一个 $n \times n \times n$ 的正方体为**合法的魔方**，当且仅当它的六个表面的 $6n^2$ 个单位正方形中包含各种颜色的贴纸各 n^2 个。

现在有若干种贴纸，每种贴纸都有无限张。每种贴纸有一个**颜色**和**权值** (权值范围为 $-1, 0, 1$)。

定义魔方的权值为 $6n^2$ 个贴纸的权值之和，求对于每个 $k \in \left[-6n^2, 6n^2\right]$，有多少种不同的魔方权值为 k。

两个魔方相同当且仅当可以再三维空间中旋转，使得每个位置上的贴纸种类相同。

算法一

为了方便起见，先来考虑数据类型为 S 的情形。

此时，所有魔方的权值均为 0，因此答案只有一个数 —— 即不同的魔方种数。

对于 $n = 1$ 的情形，只需注意到我们对于每种颜色均选择一种贴纸，均有 30 种 (见样例一) 方案组成不同的魔方。

故答案就等于 $30Y_0 \cdot W_0 \cdot R_0 \cdot O_0 \cdot B_0 \cdot G_0$，期望得分 4 分。

算法二

注意到上面那份代码在 $n = 2$ 的情形中，借助样例三的结果 (135277941853080) 却挂了，这是为什么呢？

~~当然不是 std 挂子~~，是因为问题没有那么简单。

我们先来试着手算一下，这个样例三答案是如何得出来的。

首先，不考虑旋转的情况下，我们相当于需要求对一个长度为 24 的序列进行 $6-$染色，每种颜色恰好染 4 个位置的方案数。

这个答案显然等于多项式系数 $\binom{24}{4,4,4,4,4,4} = \dfrac{24!}{(4!)^6} = 3246670537110000$。

那这样显然会算重复。那一种情况会算重复多少次呢？

我们按照样例一的角度来思考：

本来应该是 6 的全排列 $6! = 720$，倒头来只有 30 种方案，说明一个方案会被重复算 24 次！事实上，对于一个一阶魔方，通过合适的组合计数也可得知一个魔方可以旋转得到 24 个不**全等**的魔方 (有 6 种方法选择底面，固定底面后有 4 个方向可以作为前面)。

于是这个系数理应是 24。然而当你计算 $3246670537110000/24$ 的时候发现它等于 135277939046250，比答案少了那么一丁点，这是为什么呢？

~~当然也不是 std 挂子~~，这是因为，在二阶魔方中，有些方案计算的并不足 24 次，如下图：

注意上图中的 "魔方" (用展开图表示)，在整个魔方沿竖轴转体 180° 时保持不变，而不存在满足如上性质的一阶魔方。

这说明，存在某些 "魔方"，在给定的操作下具有保持不变的特性。

这引导我们去使用 Pólya 定理。

在使用 Pólya 定理之前，我们先要摸清楚使整个魔方转动构成的操作群，下简称**立方体转动群**。

(ps: 对群论比较熟悉的同学马上就可以看出来这个群同构于 4 元对称群 S_4，这里略去不提)

我们来分析一下这个群中有哪些成员 ~~(不要想歪子)~~。

1. 恒等变换，共一个。
2. 考虑固定上下底面，将魔方转体 $\pm 90^\circ$，共两个变换，又由于魔方有三对底面，故有 6 个这样的变换。
3. 考虑固定上下底面，将魔方转体 180°。同样注意到魔方有三对底面，于是这一类变换有 3 个。
4. 考虑固定一对顶点 (要求这对顶点的连线为*体对角线*)，将魔方沿着这条体对角线进行~~糖葫芦式的~~转体，可以旋转 $\pm 120^\circ$，故有 2 种这样的变换。同样，由于立方体有四对体对角线，故第 4 类变换一共有 8 个。
5. 考虑固定一条棱，将这条棱连接的两个面 (顶点) 交换。对于每一种这样的交换，其实恰好有 (立方体的) 两条平行的对棱满足这个性质，而里立方体一共有 6 对棱，故这样的变换一共有 6 个。

不难证明，这 5 种变换不重不漏的描述了立方体的所有旋转变换 (不重复比较显然，不遗漏只需注意到 $1+6+3+8+6=24$，而一阶魔方 (理应是最丰富的) 也仅仅只有 24 种变换，因此这 24 种变换恰好构成了我们所要的**立方体转动群**。

现在我们已经把**立方体转动群**的结构分析清楚了，也明白了这个系数为啥是 24。~~(其实就是经典的“一六三八六”号群)~~

接下来我们就需要使用 Pólya 定理 (Burnside 引理) 了。注意到每种颜色的使用系数有限制，我们使用标准版 (生成函数版) Pólya 定理：

对于恒等变换，不难列出它的生成函数其实就是
$(1+Y_0y+W_0w+R_0r+O_0o+B_0b+G_0g)^{24}$

最终取 $(ywrobg)^4$ 项系数，易知它就等于 $\binom{24}{4,4,4,4,4,4}\cdot(Y_0W_0R_0O_0B_0G_0)^4$

。

(番外：也就是说，我们一开始的算法其实只是计算了 Pólya 定理中其中一项)

考虑其它的项，在考虑之前，我们先要将**立方体转动群**嵌入这些面 (贴纸) 的置换群 (应该是一个 24 元置换群)。

对于第 2 类置换，则这个置换的循环指标为 t_4^6，故由 Pólya 定理，这部分的生成函数应为 $\left(1+Y_0y^4+W_0w^4+R_0r^4+O_0o^4+B_0b^4+G_0g^4\right)^6$

最终取 $(ywrobg)^4$ 项系数，别忘记这种置换由 6 个，故最终对答案的贡献为
$6\cdot\binom{6}{1,1,1,1,1,1}\cdot Y_0W_0R_0O_0B_0G_0$。

对于第 3 类置换，循环指标为 t_2^{12}，生成函数为
$\left(1+Y_0y^2+W_0w^2+\cdots+G_0g^2\right)^{12}$，贡献为
$3\cdot\binom{12}{2,2,2,2,2,2}\cdot(Y_0W_0R_0O_0B_0G_0)^2$。

对于第 4 类置换，循环指标为 t_3^8，生成函数为 $\left(1+Y_0y^3+W_0w^3+\cdots+G_0g^3\right)^8$。显然，不存在 $(ywrobg)^4$ 项，故贡献为 0。

对于第 5 类置换，循环指标为 t_2^{12}，生成函数为
$\left(1+Y_0y^2+W_0w^2+\cdots+G_0g^2\right)^{12}$，贡献为
$6\cdot\binom{12}{2,2,2,2,2,2}\cdot(Y_0W_0R_0O_0B_0G_0)^2$。

好了，最后别忘记除以群的阶数 (大小) $|G|=24$，所以最终答案的表达式就是 (设 $n=Y_0W_0R_0O_0B_0G_0$)：

$$\frac{1}{24}\cdot\left(\frac{24!}{(4!)^6}\cdot n^4+6\cdot\frac{6!}{(1!)^6}\cdot n+3\cdot\frac{12!}{(2!)^6}\cdot n^2+6\cdot\frac{12!}{(2!)^6}\cdot n^2\right)$$

结合算法一，期望得分 7 分。

算法三

考虑将算法三推广到一般阶魔方中。

此时，还是考虑每一类置换的贡献：

1. 第一类置换，循环指标为 $t_1^{6n^2}$，生成函数为 $(1+Y_0y+W_0w+\cdots+G_0g)^{6n^2}$，取 $(ywrobg)^{n^2}$ 项系数，贡献为 $\binom{6n^2}{n^2,n^2,n^2,n^2,n^2,n^2}\cdot(Y_0W_0R_0O_0B_0G_0)^{n^2}$。
2. 第四类置换，循环指标为 $t_3^{2n^2}$，生成函数为 $\left(1+Y_0y^3+W_0w^3+\cdots+G_0g^3\right)^{2n^2}$。不难发现，当且仅当 $3\mid n$ 时 $(ywrobg)^{n^2}$ 项系数非零，此时贡献为 $8\cdot\binom{2n^2}{n^2/3,n^2/3,n^2/3,n^2/3,n^2/3,n^2/3}\cdot(Y_0W_0R_0O_0B_0G_0)^{n^2/3}$。
3. 第五类置换，循环指标为 $t_2^{3n^2}$，生成函数为 $\left(1+Y_0y^2+W_0w^2+\cdots+G_0g^2\right)^{3n^2}$。不难发现，当且仅当 $2\mid n$ 时 $(ywrobg)^{n^2}$ 项系数非零，此时贡献为 $6\cdot\binom{3n^2}{n^2/2,n^2/2,n^2/2,n^2/2,n^2/2,n^2/2}\cdot(Y_0W_0R_0O_0B_0G_0)^{n^2/2}$。

话说为什么先不讨论 2, 3 两类置换呢？因为这两类置换的循环指标和 n 的奇偶性有关系。因次下面需要先对 n 的奇偶性做一个分类讨论。

- n 为偶数。
  1. 第二类置换，循环指标为 $t_4^{3n^2/2}$，生成函数为 $\left(1+Y_0y^4+W_0w^4+\cdots+G_0g^4\right)^{3n^2/2}$。贡献为 $6\cdot\binom{3n^2/2}{n^2/4,n^2/4,n^2/4,n^2/4,n^2/4,n^2/4}\cdot(Y_0W_0R_0O_0B_0G_0)^{n^2/4}$。
  2. 第三类置换，循环指标为 $t_2^{3n^2}$，生成函数为 $\left(1+Y_0y^2+W_0w^2+\cdots+G_0g^2\right)^{3n^2}$。贡献为 $3\cdot\binom{3n^2}{n^2/2,n^2/2,n^2/2,n^2/2,n^2/2,n^2/2}\cdot(Y_0W_0R_0O_0B_0G_0)^{n^2/2}$。
- n 为奇数。

1. 第二类置换，循环指标为 $t_1^2 t_4^{(3n^2-1)/2}$，生成函数为 $(1+Y_0y+W_0w+\cdots+G_0g)^2\cdot\left(1+Y_0y^4+W_0w^4+\cdots+G_0g^4\right)^{(3n^2-1)/2}$。不难证明，贡献为 0。
2. 第三类置换，循环指标为 $t_1^2 t_2^{3n^2-1}$，生成函数为 $(1+Y_0y+W_0w+\cdots+G_0g)^2\cdot\left(1+Y_0y^2+W_0w^2+\cdots+G_0g^2\right)^{3n^2-1}$。不难证明，贡献为 0。

你会发现，真巧，奇数时贡献全是 0

于是，根据 n 的奇偶性将上面这些粉色式子相加 (最后除以 24) 即得答案，期望得分 34 分。

当然，在实际实现中不难发现第三类置换的贡献永远等于第五类置换贡献的一半，故实际计算只需要算四次即可。

ps: 到这里也可知，设立 $n\equiv\pm1\pmod 6$ 的部分分的意义也明朗了：即在想要使用 Pólya 定理而不了解整个群的结构，只考虑了恒等置换的情况下可以通过 $n\equiv\pm1\pmod 6$ 的点 —— 因为在这个点下非恒等置换的贡献全是 0。

算法四

接下来考虑数据类型非 S 的情形。

首先，对于 $n=1$ ~~(或者 $n\equiv\pm1\pmod 6$)~~，仍无需考虑 Pólya 定理，因为非恒等置换不产生贡献。

这个时候，(从 DP 的角度来讲) 就相当于增加了一维 —— 每种贴纸的权值，那么从生成函数的角度来讲就是多增加一个未知数，下面设其为 x。

也就是说，考虑原来每一项的系数，比如 Y_0，现在就变成了一个单项式 $Y_{-1}x^{-1}+Y_0+Y_1x$。

于是，我们只需要将数乘改成多项式的乘法，最终各项系数即为所求，**注意输出的方式**。这里输出格式如此~~晦涩难懂~~的原因是为了输出美观，即权值为 0 的答案位于正中央。

期望得分 10 分 ($n=1$) ~~或 30 分 ($n\equiv\pm1\pmod 6$)~~。

算法五

考虑将 Pólya 定理加入其中，那么对于一个大小为 i 的循环，所产生的贡献就从 Y_0y^i 变成了 $Y_{-1}x^{-i}+Y_0+Y_1x^i$。

那么，在考虑最终式子时，如果最开始将 $Y_{-1}x^{-1}+Y_0+Y_1x$ 等多项式乘起来的话，最后要进行一次 "缩放"：即 $F(x)\to F\left(x^i\right)$。

这里的 i 可以取 $1,2,3,4$，因为算法三中提到过，存在大小为 $1,2,3,4$ 的循环。

于是只需要实现一个多项式乘法，考虑朴素的实现 (最终的次数为 $12n^2+1$)，可以通过 $n\leq 35$ 的测试点，期望得分 48 分。

算法六

最后的问题就是，如何快速求出 $\left(Y_{-1}x^{-1}+Y_0+Y_1x\right)^M\left(W_{-1}x^{-1}+W_0+W_1x\right)^M\cdots\left(G_{-1}x^{-1}+G_0+G_1x\right)^M$ (**注意这是一个一元多项式**)，其中 M 是一个比较大的数 (10^6 级别)

由于模数的原因，不考虑 FFT 系列内容。我们的目标就是对这个进行快速计算。

对于数据类型为 D 的部分，可以使用二项式定理将其展开，最后化成一个一般的卷积。

对于 M 的部分，相当于 $\left(1+x+x^2\right)^{6M}$，可以化成 $\left(\dfrac{1-x^3}{1-x}\right)^{6M}$ 后用卷积，或者去 ~~OEIS 找奇怪的公式计算~~。

不过这终究无法避免卷积~~或其它奇奇怪怪东西~~的命运，而众所周知 FFT 的常数是比较大的，尤其是三模，而且还是 $10^6\sim 10^7$ 级别的，套上 Pólya 定理的四次计算，稳 TLE 无疑。

所以，我们需要一个求多项式的快速方法。

首先，求出形如 $\left(Y_{-1}x^{-1}+Y_0+Y_1x\right)^M$ 的每一项后再求乘积一看就没什么前途，于是我们使用**幂的乘方法则**，将其化为如下形式：

$$F\left(x\right)=\left[\left(Y_{-1}x^{-1}+Y_0+Y_1x\right)\left(W_{-1}x^{-1}+W_0+W_1x\right)\cdots\left(G_{-1}x^{-1}+G_0+G_1x\right)\right]^M$$

(ps: 我们可以预先给这些多项式乘一个 x，化为非负次数的多项式)

首先内层的乘积怎么也不会超过 12 次，于是很容易算出来。

因此问题就转化为一个低次多项式的幂，~~而这是有如下套路的~~：

首先，如果能 FFT 的话，那么复杂度显然是 $O\left(m\log m\right)$ 的。当然，并不是所有情况下你都是能 FFT 上的，因此你还需要考虑对 FFT 不友好的情况。
如果使用暴力卷积 + 快速幂，那么复杂度是 $O\left(m^2\log n\right)$，表现也不是很好。下面将介绍一个简单易懂又好写的 $O\left(K\cdot m\right)$ 做法。
设给定的多项式为 $f\left(x\right)$，你需要求的多项式为 $g\left(x\right)=f^k\left(x\right)$，两边求导，得 $g'\left(x\right)=kf^{k-1}\left(x\right)f'\left(x\right)$，此即 $g'\left(x\right)=k\cdot\dfrac{g\left(x\right)}{f\left(x\right)}\cdot f'\left(x\right)$，整理得 $f\left(x\right)g'\left(x\right)=k\cdot f'\left(x\right)g\left(x\right)$。

和往常一样，两边取 x^{n-1} 项系数 (此处不妨假设 $f\left(x\right)$ 的常数项不为 0，否则简

单平移一下即可)，就可以得到一个关于 a_n 的长度不超过 K 的递推式，因此求 $g(x)$ 的一项系数可以在不超过 $O(K)$ 时间完成，因此总时间复杂度就自然是 $O(K \cdot m)$ 啦。

于是单次求幂只需要 $O\left(12n^2\right)$ 次运算，这是可以接受的。

如果你还是 TLE，这里简单介绍一个卡常技巧：注意到一次卷积只有不超过 12 项，因此可以使用 `unsigned long long` 存储最后一次取模，而不是每算一项取一次模 (这样要取 12 次模)。

这样就可以通过所有子任务了，期望得分 100 分。

(ps: std 写的有点丑，具体 `solve` 中只需要看 `#ifdef FAST` 这一部分即可，std 最慢的点不会超过 $850\,\mathrm{ms}$，因此一般不会有卡常风险。