《积性函数》解题报告

题目大意

题面描述

给定正整数 n, P 和长为 n 的数列 $a_{1\cdots n}$，判断是否存在积性函数 $f(x)$ 同时满足：

- $f(x)$ 的定义域和陪域均为正整数集。
- $f(x)$ 在其定义域上非严格单调递增。
- $\forall x \in [1, n], f(x) \bmod P = a_x$。

输入格式

本题有多组测试数据。

第一行两个正整数 T, ID，分别表示数据组数和当前测试点所在的测试包编号。特别地，样例中 $ID = 0$。

接下来依次输入每组测试数据，对于每组测试数据：

第一行两个正整数 n, P，意义如题。

第二行 n 个非负整数，第 i 个表示 a_i，意义如题。

请注意，本题部分测试点读入量较大，故下发了附加文件中的 `fastread.cpp` 作为快速读入模板。

使用时只需将模板粘贴至代码中，调用 `read()` 会从标准输入流中读入一个整数并返回，**切勿将其与其它输入方式混用**。保证标准算法的读入部分使用此模板实现。

若你仍然不理解如何使用此模板，可以查看选手目录下的 `sample.cpp`，其中给出了一种示例实现，补全此代码即可。

输出格式

对于每组测试数据输出一行，若存在合法的 $f(x)$ 则输出 `YES`，否则输出 `NO`。

请注意需全部使用半角英文大写字母。

样例一

样例一输入

```
3 0
3 5
1 4 4
4 2023
1 2 2 2
5 2023
1 2 10 11 12
```

样例一输出

```
YES
NO
NO
```

样例一解释

本测试点包含三组测试数据。

对于第一组测试数据，取 $f(x)=x^2$，可以验证 $f(x)$ 是非严格单调递增的积性函数，符合题意。同时 $f(1) \bmod 5 = 1, f(2) \bmod 5 = 4, f(3) \bmod 5 = 9 \bmod 5 = 4$ 符合要求。

对于第二组测试数据，可以证明不存在合法的 $f(x)$。比如，若 $f(1)=1, f(2)=f(3)=f(4)=2$，则可以得到 $f(6)=f(12)=4$，故 $f(7)=f(10)=4, f(5)=2$，于是 $f(15)=f(3)f(5)=4>f(14)=f(2)f(7)=8$，这构成了矛盾。

对于第三组测试数据，同样可以证明不存在合法的 $f(x)$。比如，若 $f(1)=1, f(2)=2, f(3)=10, f(4)=11, f(5)=12$，则可以证明 $f(6)=20 \le f(7) \le f(10)=24$，而 $f(12)=110, f(14) \le 24 \times 2 = 48, f(14)<f(12)$，矛盾。

限于篇幅，无法于题面中对后两组测试数据中 $f(x)$ 的不存在性给出完整证明。

数据规模与约定

对于 100% 的数据，$1 \le T \le 20, 1 \le n < P \le 5 \times 10^6, 0 \le a_i < P$。

保证每个数据点中的 $\sum n, \sum P \le 10^7$。

本题使用捆绑测试，且评测时会按照各测试包特殊限制的逻辑关系绑定依赖。

各测试包的分值和特殊限制如下：

- 测试包 1：2 分，$n, P \le 1000$，$a_1 = 0$。
- 测试包 2：3 分，$n, P \le 1000$，$\forall i \in [1, n], a_i = 1$。
- 测试包 3：4 分，$n, P \le 1000$，保证 $n = 500$，且每个 a_i 在 $[0, P)$ 内独立地均匀随机生成。同时保证该测试包内恰有 3 个测试点。
- 测试包 4：12 分，保证如果存在合法的 $f(x)$，则存在合法 $f(x)$ 为严格单调递增的完全积性函数，且 $\forall x \le n, f(x) < P$，且 $n, P \le 10$，P 是质数。
- 测试包 5：13 分，保证如果存在合法的 $f(x)$，则存在合法 $f(x)$ 为严格单调递增的完全积性函数，且 $\forall x \le n, f(x) < P$，且 $n, P \le 1000$，P 是质数。
- 测试包 6：7 分，保证如果存在合法的 $f(x)$，则存在合法 $f(x)$ 为严格单调递增的完全积性函数，且 $n, P \le 1000$，P 是质数。
- 测试包 7：5 分，保证如果存在合法的 $f(x)$，则存在合法 $f(x)$ 为严格单调递增的完全积性函数，且 $n, P \le 3 \times 10^5, \sum n, \sum P \le 6 \times 10^5$，$P$ 是质数。
- 测试包 8：14 分，保证如果存在合法的 $f(x)$，则存在合法 $f(x)$ 严格单调递增，且 $n, P \le 1000$，P 是质数。
- 测试包 9：9 分，$n, P \le 3 \times 10^5, \sum n, \sum P \le 6 \times 10^5$，$n = P - 1$，$P$ 是质数。
- 测试包 10：8 分，$n, P \le 3 \times 10^5, \sum n, \sum P \le 6 \times 10^5$，$P$ 是奇数。
- 测试包 11：15 分，$n, P \le 3 \times 10^5, \sum n, \sum P \le 6 \times 10^5$。
- 测试包 12：2 分，$n, P \le 10^6, \sum n, \sum P \le 2 \times 10^6$。
- 测试包 13：6 分，无特殊限制。

时空限制

时间限制：2 秒。

空间限制：1 GiB。

解题过程

部分分

对于子任务一，由于 $f(x)$ 是积性函数，因此必然有 $f(1)=1$，这与题面的 $a_1=0$ 矛盾，因此输出 `NO` 即可。

对于子任务二，考虑取 $f(x)=1$，容易验证 $f(x)$ 符合条件，因此输出 `YES` 即可。

对于子任务三，可以猜测在随机情况下极大概率不合法，而且由于只有三个测试点，因此大概率此测试包中每组测试数据的答案都是 `NO`，实测直接输出 `NO` 可以通过本测试包。（这里的概率极小在推理出之后的关键结论后是显然的）

对于子任务四，考虑模拟推理过程，由 $a_{1\cdots n}$ 向后推理延伸一段较长的距离判断一下。当然也可能有其它不同的算法可以通过此测试包。

关键结论一

考虑本题中的一个部分分：保证 $f(x)$ 是完全积性函数。随便举一些例子，我们猜想：

- $f(x)$ 是非严格单调的完全积性函数当且仅当存在 $k\in\mathbb{N}$，使 $f(x)=x^k$。

  这么猜想是因为完全积性函数的限制实在太紧了，稍微偏离一点都会导出矛盾。

考虑证明，充分性是显然的。至于必要性，对 $x>1$ 设 $g(x)=\log_x f(x)$，证明所有有 $g(x)$ 相等。

由于 $f(x)$ 是完全积性的，所以证明变得十分容易。

具体地，若有 $g(n)<g(m)$，不妨设 $n<m$（易证 $g(m)=g(m^t),t\in\mathbb{N}_+$，故若 $n>m$ 可以取某个比 n 大的 m 的幂）。

我们考虑：

$$g(n)<g(m)\Rightarrow\frac{\log_m f(m)}{\log_n f(n)}>1\Rightarrow\frac{\ln f(m)}{\ln f(n)}>\frac{\ln m}{\ln n}>1$$

由有理数的稠密性可以找到 $\dfrac{\ln f(m)}{\ln f(n)}>\dfrac{u}{v}>\dfrac{\ln m}{\ln n},u,v\in\mathbb{N}_+$。

取 $A=n^u,B=m^v$，可知 $\ln A=u\ln n>v\ln m=\ln B$，而 $\ln f(A)=u\ln f(n)<v\ln f(m)=\ln f(B)$，得到矛盾，证毕。

关键结论二

事实上我们还可以证明：

- 若 $f(x)$ 是非严格单调的积性函数，则 $f(x)$ 是完全积性函数。

先证明一个引理：对任意质数 p，有：

$$\inf_{x\perp p}\frac{f(x+p)}{f(x)}=1$$

其中 $\perp$ 表示两数互质，$\inf S$ 表示 S 的下确界，如 $\inf(0,1]=0$。

由于 $f(x)$ 不降，所以下确界 u 至少为 1。而另一方面取一个不等于 p 的质数 q，对任意正整数 k，取 $x \perp pq, x > kp$，可得：

$$u^k f(x) \le f(x+kp) \le f(2x) \le f(qx) = f(q)f(x)$$

即 $\forall k \in \mathbb{N}_+, u^k \le f(q)$，这意味着 $u \le 1$。由此 $u = 1$。

考虑证明原命题，只需证 $\forall p \in \mathbb{P}, k \in \mathbb{N}_+, f(p^{k+1}) = f(p)f(p^k)$。考虑取任意 $x \perp p$，都有：

$$(px-1)p^k \le xp^{k+1} \Rightarrow f(px-1)f(p^k) \le f(x)f(p^{k+1}) \Rightarrow \frac{f(p^{k+1})}{f(p^k)} \ge \frac{f(px-p^2)}{f(x)} \ge f(p) \times \frac{f(x-p)}{f(x)}$$

这意味着 $\forall x \perp p, \frac{f(p^{k+1})}{f(p^k)} \ge f(p) \times \frac{f(x-p)}{f(x)}$，由刚才的引理可知 $\frac{f(p^{k+1})}{f(p^k)} \ge f(p)$。

用 $(px+1)p^k \ge xp^{k+1}$ 可以类似地证明另一个方向，综合起来可以得到最初的结论。

证毕。

事实上，这在数学上称为 Erdos 积性函数定理，本文中的证明过程参考了 [1]。

由上面的结论可以把原问题转化为判断是否存在 $k \in \mathbb{N}$ 使得 $\forall x \in [1,n], x^k \bmod P = a_x$。

（由此可以发现，本题中保证严格单调的部分分只是比非严格单调少了 $f(x) = 1$ 一种情况）

算法一

暴力做法是枚举所有 $k \in [0, 2\varphi(P))$ 一一检验，由拓展欧拉定理易证枚举到 $2\varphi(P)$ 已经足够。

复杂度 $\mathcal{O}(n^2)$（此处 n 指单组数据中所有 n 的和，认为 n, P 同阶，下同），可以得到 55 分。

当然，$\mathcal{O}(n^2 \log n)$ 也是可以得到 55 分的。

算法二

如何优化？事实上，**我们可以把 P 质因数分解，解决每个 $P_i = p_i^{q_i}$ 的情况，最后通过某种方式把它们合并起来**。

一种思路是对每个 $p_i^{q_i}$ 在 $[1,n]$ 内找到原根。

第一部分

假如对每个 $p_i^{q_i}$ 都能在 $[1,n]$ 内找到原根，那么可以解出 $k \bmod \varphi(P)$ 的值，尝试至多两个 k 即可。

对于 P 是质数，$n = P-1$ 的部分分，$[1,n]$ 中必然存在原根，直接使用此算法即可通过。

第二部分

若上述算法失效，则 n 的大小一定不超过 $p_i^{q_i}$ 的最小原根。

- 若 $q_i = 1$，依熟知结论（[2]）一定有 $n = \mathcal{O}(P^{\frac{1}{4}+\epsilon})$，朴素地对单个 k 的判定可以使用线性筛做到 $\mathcal{O}(n)$，总复杂度 $\mathcal{O}(P^{\frac{5}{4}+\epsilon})$。
- 若 $q_i > 1$，我们接下来将证明 $p_i^{q_i}$ 的最小原根不超过 p_i。这样我们有 $n < p = \mathcal{O}(P^{\frac{1}{2}})$，类似 $q_i = 1$ 的部分，可以做到 $\mathcal{O}(P^{\frac{3}{2}})$。

但是，若 P 中含有若干个质因子 2，上面的算法将失效。

综上，此算法预计可以通过本题中 P 为奇数的部分分。

结论证明

为了方便，记 $p=p_i, m=q_i, P=p^m$。

我们先证明，对 $m\geq 2$，p^m 的所有小于 p^2 的原根组成的集合等于 p^2 的原根集合。

归纳证明之，$m=2$ 时显然成立，现证 $g<p^m$ 是 p^m 的原根当且仅当 g 是 p^{m+1} 的原根。

事实上，若 g 不是 p^m 的原根，即存在 $d<(p-1)p^{m-1}$ 使得 $g^d\equiv 1\pmod{p^m}$，

则 $(g+kp^m)^{pd}\equiv (wp^m+1)^p\equiv 1\pmod{p^{m+1}}, w,k\in\mathbb{N}$，因此 $\forall k\in\mathbb{N}, g+kp^m$ 也不是 p^{m+1} 的原根。

可得所有 p^{m+1} 的原根 g 都满足 $g \bmod p^m$ 是 p^m 的原根。（此结论对 $m\geq 1$ 都有效，接下来将重复利用之）

而依熟知结论（[3]），p^m 的原根个数是 $\varphi(\varphi(p^m))=\varphi(p-1)(p-1)p^{m-2}$，而 p^{m+1} 的原根个数是 $\varphi(p-1)(p-1)p^{m-1}$。

考虑 $[0,p^{m+1})$ 中所有数 g 满足 $g \bmod p^m$ 是 p^m 的原根，这样的 g 共有 $\varphi(p-1)(p-1)p^{m-1}$ 个，恰好等于 p^{m+1} 的原根个数。

而其所有原根都必然出自这个集合，可知这个集合中的每个 g 都是 p^{m+1} 的原根。证毕。

再证明 p^2 的最小原根小于 p。

同样由 [3]，p 的原根个数是 $\varphi(p-1)$，p^2 的原根个数是 $\varphi(p-1)(p-1)$。

设 S 为所有 $g<p^2$ 满足 $g \bmod p$ 是 p 的原根的数组成的集合，由上可知 $|S|=\varphi(p-1)p$ 且所有 p^2 的原根出自此集合。

假如 S 中所有不超过 p 的数（共有 $\varphi(p-1)$ 个）都不是 p^2 的原根，那么由原根个数可得 S 中所有大于 p 的数都是 p^2 的原根。

考虑 p 的一个原根 g，设 g 在模 p^2 意义下的逆元为 g^{-1}，由逆元的对称性易证 $g^{-1} \bmod p$ 是 p 的原根，但 g^{-1} 不是 p^2 的原根。

这意味着 $g^{-1}<p$，这样 $1<gg^{-1}<p^2$，但这与 $gg^{-1}\equiv 1\pmod{p^2}$ 矛盾。证毕。

（此部分证明参考了 [4]）

事实上，由 [5] 中的结果，p^2 的最小原根小于 $p^{0.99}$；由 [6] 中的结果，最小原根小于 $p^{0.74}$ 。

由此可以优化上面的复杂度分析结果，但是不重要。

算法三

结论一览

下面将引用一些熟知结论，列举如下：

- $\max_{k=1}^{n}\omega(k)=\Theta(\frac{\log n}{\log\log n})$;
- $\max_{k=1}^{n}\sigma(k)=\Theta(n\log\log n)$;
- $\varphi(n)=\Omega(\frac{n}{\log\log n})$;

这些结论的证明可见 [7]。

算法过程

考虑一般情况下的解。

依然采用刚才的思路，把 P 质因数分解后解决每个 $P_i=p_i^{q_i}$ 的情况。

方便起见，下面记当前枚举的 P_i,p_i,q_i 分别为 P,p,q，全局的 P 为 P_0。

每次我们可以通过 $\mathcal{O}(n)$ 的时间扫一遍判断是否有 $a_i\equiv a_{i-P}\pmod P$，然后把 n 缩到 P 的级别以保证复杂度。

这部分的复杂度是 $\mathcal{O}(n\omega(n))=\mathcal{O}(\frac{n\log n}{\log\log n})$。

考虑所有 $p\mid x$ 的位置，若有 a_x 不是 0，那么就可以确定出**全局（而不是单独的一个 $P_i=p_i^{q_i}$）中** k 的精确值，判断即可。

否则对每个 $x\perp p$ 的数求出其阶 $v(x)$ 表示最小的 $x^{v(x)}\bmod P=1$ 的正整数，这可以 $\mathcal{O}(n\log^2 n)$ 地完成。

具体地，显然 $v(x)\mid\varphi(P)$，我们把 $\varphi(P)$ 质因数分解，设分解出来是 $y_1,y_2,\cdots,y_m$（彼此可能相同），比如 12 的结果是 $\{2,2,3\}$。

我们可以维护一个答案 ans，初始 $ans=\varphi(P)$，枚举 $i\in[1,m]$，判断是否有 $x^{\frac{ans}{y_i}}\equiv 1\pmod P$。

- 若满足则一定有 $v(x)\mid\frac{ans}{y_i}$，我们令 $ans\leftarrow\frac{ans}{y_i}$；
- 否则 $v(x)\mid ans$ 但 $v(x)\nmid\frac{ans}{y_i}$，也就是说 $v(x)$ 一定有 y_i 这个因子。这种情况下我们不改变 ans 的值。

显然 $m=\mathcal{O}(\log P)$，而一次快速幂是 $\mathcal{O}(\log P)$ 的，复杂度 $\mathcal{O}(n\log^2 n)$。（这里是当前整题的复杂度瓶颈）

然后我们对于每个数 u，暴力枚举 $u^x\bmod P=a_u$ 的解可以 $\mathcal{O}(v(u))$ 地将其解出，若 $v(u)$ 的值之前已经出现过了就不枚举了。

我们得到的解的形式是 $k\bmod v(u)=x$，可以 $\mathcal{O}(\frac{\varphi(P)}{v(u)})$ 地标记哪些位置才可能合法。

这部分的复杂度是 $\mathcal{O}(\sigma(\varphi(P)))=\mathcal{O}(n\log\log n)$。

只要上面过程中解出来解，任意取其中的一个，给它加上 $\varphi(P)$ 检验，易证当且仅当检验成功时有解。

这样我们解决了 $P_i=p_i^{q_i}$ 的情况。假如我们要合并答案，又该怎么办呢？

由于 k 可以取得尽量大，所以只需要关心同余性质。

注意到由上面的标记过程，对于每个 $p_i^{q_i}$，我们对每个 $x\in[0,\varphi(p_i^{q_i}))$ 都判断出了 $k\bmod\varphi(p_i^{q_i})=x$ 时是否合法。

这样直接枚举每个 $k\in[0,\varphi(P_0))$ 判断是否在每个 $p_i^{q_i}$ 中都合法即可。假如在每个 $p_i^{q_i}$ 中都合法，由中国剩余定理，对于全局的 P_0 也合法。

这部分的复杂度是 $\mathcal{O}(n\omega(n))=\mathcal{O}(\frac{n\log n}{\log\log n})$。

总结

我们给出了本题的第一个通用解法，总复杂度是 $\mathcal{O}(n\log^2 n)$。

这个做法可以通过 $n\le 3\times 10^5,\sum n\le 6\times 10^5$ 的数据，控制一下实现时的常数因子可能已经足以通过 $n\le 10^6,\sum n\le 2\times 10^6$ 的数据。

算法四

考虑继续优化。在本题复杂度瓶颈的一部分中，我们需要对 $[1,P)$ 中的每个数求出其阶。

算法过程

相较于上面的 $\mathcal{O}(n\log^2 n)$ 做法，一种复杂度更优秀的做法是：

- 设 $\varphi(P)=\prod_{i=1}^m c_i^{d_i}$，对于每个质因子 c_i，我们可以**二分**出 $v(x)$ 中 c_i 的幂次。
- 设 $v(x)$ 中 c_i 的幂次为 w_i，那么 $v(x)=\prod_{i=1}^m c_i^{w_i}$。

复杂度分析

此算法的复杂度较难分析。

显然其最劣复杂度有平凡下界为 $\Omega(n\omega(n)\log n)=\Omega\left(\dfrac{n\log^2 n}{\log\log n}\right)$。

考虑分析其上界，显然 $\sum d_i=\mathcal{O}(\log n), m=\omega(n)=\Theta(\frac{\log n}{\log\log n})$，

由琴生不等式（[8]）和对数函数的凸性，二分的次数是
$\sum_{i=1}^m \log_2(d_i+1)\le m\log_2\left(\dfrac{\sum d_i}{m}+1\right)=\mathcal{O}\left(\dfrac{\log n\log\log\log n}{\log\log n}\right)$。

我们记二分的次数为 $F(n)$，目前的总复杂度就是 $\mathcal{O}(nF(n)\log n)=\mathcal{O}\left(\dfrac{n\log^2 n\log\log\log n}{\log\log n}\right)$。

可能可以分析出更紧的界，但囿于笔者水平，无法给出进一步的分析。

这可以通过 $n\le 10^6,\sum n\le 2\times 10^6$ 的数据。

算法五

如何进一步优化这个算法？注意到求阶依然是现在的复杂度瓶颈，考虑继续优化这部分的复杂度。

按照 $P=p^q$ 中 p 的奇偶性分类讨论。

奇质数

我们发现，对于 $P=p^q$，若 $p\ge 3$ 则 P 一定存在原根。

我们找到一个原根 g，对 $x\equiv g^k\pmod P$，可以证明 $v(x)=\dfrac{\varphi(P)}{\gcd(\varphi(P),k)}$，由此可以轻松做到 $\mathcal{O}(n\log n)$ 的复杂度。

由于原根有 $\varphi(\varphi(P))$ 个，密度较大，所以随机生成一个数判断其是否是原根的期望随机次数是 $\mathcal{O}(\log^2\log n)$，可以接受。

由算法二中的结论，直接从小到大枚举复杂度也是对的。

此部分的瓶颈在于求 $\gcd$，这可以 $\mathcal{O}(n)$ 地线性筛出，这部分被做到了线性。

偶质数

否则 $P=2^q$，我们考虑递推求出阶。设 $\delta_m(x)$ 为 x 模 m 意义下的阶，如 $\delta_2(1)=1$。

假设我们已经对所有 $2\nmid x, x\in[0,2^{k-1})$ 求出 $\delta_{2^{k-1}}(x)$ 的值，考虑对每个上述的 x 求出 $\delta_{2^k}(x)$ 和 $\delta_{2^k}(x+2^{k-1})$ 的值。

设 $d=\delta_{2^{k-1}}(x)$，显然 $d\mid\delta_{2^k}(x), d\mid\delta_{2^k}(x+2^{k-1})$。

而且我们发现 $x^{2d} \equiv (x+2^{k-1})^{2d} \equiv 1 \pmod{2^k}$，这样只需要 $\mathcal{O}(1)$ 次判断就可以求出上面两项的值。

这样我们做到了 $T(n) = T(n/2) + O(n\log n)$，即 $T(n) = \mathcal{O}(n\log n)$ 的复杂度。

要做到 $\mathcal{O}(n)$ 需要分析一下性质：

- 若 $x = 1$，显然 $\delta_{2^k}(x) = 1, \delta_{2^k}(x+2^{k-1}) = 2$；
- 同理若 $x = 2^{k-1}-1$，则 $\delta_{2^k}(x) = \delta_{2^k}(x+2^{k-1}) = 2$；
- 可以猜想，除了上面两种情况以外都有 $\delta_{2^k}(x) = \delta_{2^k}(x+2^{k-1}) = 2d$。我们之后将证明此结论。

根据上面三种情况分类讨论，我们可以线性地由 2^k 的结果递推出 2^{k+1} 的结果。

这部分被我们优化到了 $T(n) = T(n/2) + O(n)$，即 $T(n) = \mathcal{O}(n)$。

结论证明

先证明 $\delta_{2^k}(x) = \delta_{2^k}(x+2^{k-1})$。由上面的过程容易看出，所有 $\delta_{2^k}(x)$ 都是 2^u 形式的数。

若 $x \neq 1$，则 $u \geq 1, 2 \mid 2^u$。注意到 $x^2 \equiv (x+2^{k-1})^2 \pmod{2^k}$，由此
$x^{\delta_{2^k}(x)} = (x+2^{k-1})^{\delta_{2^k}(x)}$，易证上述结论成立。

再证明 $\delta_{2^k}(x) = 2\delta_{2^{k-1}}(x)$，这只需归纳证明
$\forall x \in [3, 2^k-3] \wedge 2 \nmid x, x^{\delta_{2^k}(x)} \equiv 2^k+1 \pmod{2^{k+1}}$ 即可。

$k = 3$ 可以直接验证。现在设 $k = m$ 时成立，对某个 x 分类讨论：

- 若 $x^{\delta_{2^k}(x)} \equiv 2^k+1 \pmod{2^{k+2}}$，则有 $x^{\delta_{2^{k+1}}(x)} \equiv (x^{\delta_{2^k}(x)})^2 \equiv 2^{k+1}+1 \pmod{2^{k+2}}$ 成立。
- 若 $x^{\delta_{2^k}(x)} \equiv 2^{k+1}+2^k+1 \pmod{2^{k+2}}$，依然有 $x^{\delta_{2^{k+1}}(x)} \equiv 2^{k+1}+1 \pmod{2^{k+2}}$ 成立。

证毕。

由上可知，**给定 $P = p^q$，对 $i \in [1, P), i \perp P$ 的所有 i 求 $\delta_P(i)$ 可以做到 $\mathcal{O}(P)$ 复杂度。**

总结

这样这部分被我们优化掉了，瓶颈落到了整体框架上，整题的时间复杂度变为
$\mathcal{O}(n\omega(n)) = \mathcal{O}\left(\dfrac{n\log n}{\log\log n}\right)$。

空间复杂度视实现为 $\mathcal{O}\left(\dfrac{n\log n}{\log\log n}\right)$ 或 $\mathcal{O}(n)$。

这已经可以通过 $n \leq 5\times 10^6, \sum n \leq 10^7$ 的数据。

算法六

考虑将复杂度优化至 $\mathcal{O}(n)$。我们考虑上述算法中时间复杂度不为 $\mathcal{O}(n)$ 的部分，逐一优化之。

判同余部分

首先，对于每个 $P_i = p_i^{q_i}$，需要执行上面提到的：

> 每次通过 $\mathcal{O}(n)$ 的时间扫一遍判断是否有 $a_x \equiv a_{x-P_i} \pmod{P_i}$，然后把 n 缩到 P_i 的级别以保证复杂度。

由于 x^k 一定是完全积性函数，所以合法的一个必要条件是 $\forall x, y \in \mathbb{N}_+, a_{xy} \equiv a_x a_y \pmod{P_0}$。

也就是说，给定的 $a_{1\cdots n}$ 一定在 $\bmod P_0$ 意义下完全积性，我们可以使用线性筛判断之。

具体地，我们忽略所有合数 x 位置上 a_x 的值，只使用质数位置上的 a_x 线性筛出满足完全积性的 $a'_{1\cdots n}$，与 $a_{1\cdots n}$ 逐位比较。

而若 $a_{1\cdots n}$ 完全积性，则若存在一个 k 使得所有质数 x 位置上有 $a_x \equiv x^k \pmod{P_i}$，就可以推出 $\forall x \in [1,n], x^k \equiv a_x \pmod{P_0}$。

也就是说，对于每个 $P_i = p_i^{q_i}$ 判同余时只需检查是否对于所有不超过 n 的质数 x 都有 $a_x \equiv a_{x \bmod P_i} \pmod{P_i}$ 即可。

由质数定理（[9]），n 以内的质数个数是 $\Theta(\frac{n}{\log n})$ 级别的，这部分的总复杂度为 $\Theta(\frac{n}{\log n} \times \omega(n)) = \mathcal{O}(n)$，达到了线性。

解方程部分

考虑算法中解方程的部分，即上文中的：

> 然后我们对于每个数 u，暴力枚举 $u^x \bmod P = a_u$ 的解可以 $\mathcal{O}(v(u))$ 地将其解出，若 $v(u)$ 的值之前已经出现过了就不枚举了。

注意到我们从前向后枚举时，得到的解的形式一定为 $x \bmod A = B$ 的形式。

若当前的 $v(u) \mid A$，则得到的解 $x \bmod v(u) = y$ 只有两种可能：

- $y = (B \bmod v(u))$，这种情况下不会改变方程的解；
- $y \neq (B \bmod v(u))$，这种情况下一定无解，而无解情况可以在最后"任意取其中的一个，给它加上 $\varphi(P)$ 检验"这个检验过程中得到矛盾。

也就是说，无论如何这个解都不会改变情况，我们可以直接忽略 u。

否则 $v(u) \nmid A$，这样易证方程的新解 $x \bmod A' = B'$ 中一定有 $A' = \operatorname{lcm}(A, v(u)) > A$，故 $\varphi(P) \geq A' \geq 2A$。

因此，这样的情况一定不会超过 $\log_2 \varphi(P)$ 次。

我们使用快速幂求出 u^A, u^B 的值，然后从小到大枚举 $u^B, u^{A+B}, u^{2A+B}, \cdots$ 直到找到一个同余于 a_u 的数 $kA + B$ 作为 B'。若一直找到 $kA + B \geq \operatorname{lcm}(A, v(u))$ 都没有找到解则返回无解。

由于每次枚举至少会使 $kA + B$ 的值增加一，单次复杂度是 $\mathcal{O}(1)$ 的，所以枚举的总复杂度是 $\mathcal{O}(n)$。

而求快速幂、求 $\operatorname{lcm}(A, v(u))$ 的次数都是 $\mathcal{O}(\log n)$ 的，这部分的复杂度显然是 $\mathcal{O}(n)$ 的。

合并方程部分

对于上面我们提到的：

> 我们得到的解的形式是 $k \bmod v(u) = x$，可以 $\mathcal{O}(\frac{\varphi(P)}{v(u)})$ 地标记哪些位置才可能合法。

因为上文的“解方程部分”已经对一个 P_i 给出了完整的解，这部分合并已经不再需要。

对于合并答案时的：

> 注意到由上面的标记过程，对于每个 p^q，我们对每个 $x \in [0, \varphi(p^q))$ 都判断出了 $k \bmod \varphi(p^q) = x$ 时是否合法。
>
> 这样直接枚举每个 $k \in [0, \varphi(P))$ 判断即可，复杂度 $\mathcal{O}(n\omega(n)) = \mathcal{O}(\frac{n \log n}{\log \log n})$。

观察到对于每个 p^q 我们解出的结果相当于判断了当且仅当 $k \bmod A_i = B_i$ 时对此 p^q 合法，相当于一个同余方程。

我们可以使用 exCRT 求解。由于只有 $\omega(P)$ 个方程，复杂度依然是 $\mathcal{O}(n)$ 的。

总结

综上所述，我们优化掉了所有复杂度不为线性的部分，把整题复杂度做到了 $\mathcal{O}(n)$ 的复杂度，这已经是本题最优的复杂度。

如果有更简洁的做法或者对本文中的算法分析出了更紧的界欢迎与笔者交流。

本题数据较难构造，笔者水平有限，可能有放过了一些朴素算法，在此致歉。

致谢

感谢中国计算机学会提供学习和交流的平台。

感谢楼宇辰、张扬梓同学的验题、审稿工作和翟锦程同学提出的建议。

感谢国家集训队杨耀良教练的指导。

感谢陈合力、董烨华、翁天东老师的栽培与指导。

感谢父母的关心和支持。

参考文献

[1] kokic，Erdo˝s 单调乘性函数定理，https://kokic.github.io/article/erdos-mmf-thm/.

[2] OI-Wiki，原根：最小原根的范围估计， https://oi-wiki.org/math/number-theory/primitive-root/#%E6%9C%80%E5%B0%8F%E5%8E%9F%E6%A0%B9%E7%9A%84%E8%8C%83%E5%9B%B4%E4%BC%B0%E8%AE%A1.

[3] OI-Wiki，原根：原根个数，https://oi-wiki.org/math/number-theory/primitive-root/#%E5%8E%9F%E6%A0%B9%E4%B8%AA%E6%95%B0.

[4] Math Overflow，The smallest primitive root modulo powers of prime，https://mathoverflow.net/questions/368556/the-smallest-primitive-root-modulo-powers-of-prime.

[5] B. Kerr, *et al.*，The least primitive root modulo p^2，https://arxiv.org/abs/1908.11497.

[6] B. Chen，Explicit upper bound on the least primitive root modulo p^2，International Journal of Number Theory Vol. 18, No. 02, pp. 389-395 (2022).

[7] liqingyang，约数和的数量级， https://www.luogu.com.cn/blog/llqyy/sigma.

[8] Wikipedia，Jensen's inequality，https://en.wikipedia.org/wiki/Jensen%27s_inequality.

[9] Wikipedia，Prime number theorem，https://en.wikipedia.org/wiki/Prime_number_theorem.