Mediam Problem 题解

简单分析发现要同时维护每个 k 的答案并不容易，于是考虑对每个 k 单独计算。

对一个确定的 $b_1 = k$，某个 a_i 对 b_1 的影响只与 a_i 相对 k 的大小有关。我们可以将所有小于 k 的 a_i 都看作 1，大于的看作 2，等于的看作 0。先统计这些 $0, 1, 2$ 在树上分布的可行方案，再乘上相应的排列数即可。

设 $dp[u][i][0/1/2/3]$ 表示以结点 u 为根的子树中满足 $a_v < k$ 的节点 v 的个数为 i，且 b_u 等于 k、一定小于 k、一定大于 k、小于或大于 k 的方案数。这里“b_u 小于或大于 k”的状态是考虑到集合大小为偶数时中位数有两个，但对应的状态只有一个而设置的。

DFS 整棵树进行转移。考虑到计算 $dp[u][i][j]$ 时，我们需要的是集合 $S = \{a_u\} \cup \{b_v | \text{node } u \text{ is the parent of node } v\}$ 的中位数，这无法直接转移，因为我们需要知道 S 中的数小于、大于、等于 k 的个数。

于是考虑维护一个临时 DP 数组 $tp[x][y][z][0/1]$ 表示以 u 为根的子树中有 x 个 a_v 小于 k，S 中的数有 y 个一定小于 k，有 z 个一定大于 k，且里面不包含、包含 k 的方案数。但是这样转移的复杂度过高，需要简化。考虑到一个不等于 k 的状态如果不是“一定小于 k”，又不是“一定大于 k”，那么它就是“小于或大于 k”。于是可以分别计算“一定小于 k”、“一定大于 k” 的方案数，再用总方案数减去它们来得到“小于或大于 k” 的方案。

设 $tp_<[x][y][0/1]$ 表示以 u 为根的子树中有 x 个 a_v 小于 k，S 中的数有 y 个一定小于 k，且里面不包含、包含 k 的方案数，有转移方程：

	$dp[v][i][0]$	$dp[v][i][1]$	$dp[v][i][2]$	$dp[v][i][3]$
$tp_<[x][y][0]$	$tp_<[x+i][y][1]$	$tp_<[x+i][y+1][0]$	$tp_<[x+i][y][0]$	$tp_<[x+i][y][0]$
$tp_<[x][y][1]$	$-$	$tp_<[x+i][y+1][1]$	$ty_<[x+i][y][1]$	$tp_<[x+i][y][1]$

设 $tp_>[x][y][0/1]$ 表示以 u 为根的子树中有 x 个 a_v 小于 k，S 中的数有 y 个一定大于 k，且里面不包含、包含 k 的方案数，有转移方程：

	$dp[v][i][0]$	$dp[v][i][1]$	$dp[v][i][2]$	$dp[v][i][3]$
$tp_>[x][y][0]$	$tp_>[x+i][y][1]$	$tp_>[x+i][y][0]$	$tp_>[x+i][y+1][0]$	$tp_>[x+i][y][0]$
$tp_>[x][y][1]$	$-$	$tp_>[x+i][y][1]$	$ty_>[x+i][y+1][1]$	$tp_>[x+i][y][1]$

注意：这里的 tp 数组实际已经滚动掉了一维。需要按照 x, y, i 的顺序，降序枚举变量，并且需在读取 $tp[x][y][0/1]$ 后、枚举 i 进行转移前将其置零。

设 $sz = |S|$，有转移方程：

$$dp[u][i][1]=\sum_{y=\lfloor sz/2\rfloor+1}^{sz}tp_{<}[i][y][0]$$

$$dp[u][i][2]=\sum_{y=\lfloor sz/2\rfloor+1}^{sz}tp_{>}[i][y][0]$$

$$dp[u][i][3]=\sum_{y=0}^{sz}tp_{<}[i][y][0]-dp[u][i][1]-dp[u][i][2]$$

$$dp[u][i][0]=\sum_{y=0}^{sz}tp_{<}[i][y][1]-\sum_{y=\lfloor sz/2\rfloor+1}^{sz}tp_{<}[i][y][1]-\sum_{y=\lfloor sz/2\rfloor+1}^{sz}tp_{>}[i][y][1]$$

$dp[u][i][j]$ 的第二维 i 的转移本质上是一个树形背包，复杂度为 $O(n^2)$，第三维 j 的转移需要借助 $tp[x][y][0/1]$ 的第二维 y，这会使复杂度乘上 n，再乘上枚举 k 的复杂度 n，总时间复杂度为 $O(n^4)$。空间复杂度为 $O(n^2)$。

Kanade Doesn't Want to Learn CG 题解

首先排除一些乱七八糟的反弹进球的情况，比如撞到篮筐打铁进球（这种情况规定球直接飞出去），无限次与地板碰撞反弹（这里不考虑地面与球之间的作用）。

考虑方程 $ax^2+bx+c=y_0$ 的解，这决定球是否可以上升到篮筐高度，也就决定是否可能进。

显然，令 $\Delta=b^2-4a(c-y_0)$，若 $\Delta\le 0$，则不可能上升到篮筐高度，或只可能与篮筐相切。这些情况都无法进球。

只有 $\Delta>0$ 时可能进球，则方程 $ax^2+bx+c=y_0$ 必有两个解，设为 x_{left} 和 x_{right}，并且保证 $x_{\text{left}}\neq x_{\text{right}}$。不妨假设 $x_{\text{left}}<x_{\text{right}}$。

因为在上升阶段进球是无效的（判罚违例），因此在上升阶段达到 y_0 高度时，篮筐线段的左端点一定要在球位置的右边才可能进球。因此需要保证 $x_{\text{left}}<x_0$，否则一定输出 `No`。

在下降阶段进球有两种情况。第一种是直接进球，也就是 $x_0<x_{\text{right}}<x_1$，请注意两端均不能取等号，否则会导致球打在篮筐上（俗称打铁），不算进球。

第二种情况是打在篮板上，经过篮板反弹进球（俗称打板），根据球与篮板之间的碰撞是完全弹性碰撞，可知打板后球的运动轨迹与没有篮板时打板后球的运动轨迹关于篮板对称，根据对称性，可以做篮筐关于篮板的对称位置（折到篮板后面）。因此这种情况的进球条件是 $x_1<x_{\text{right}}<2x_1-x_0$，请注意两端均不能取等号，否则仍会打铁。这里的对称，在球处于上升阶段时与篮板碰撞时仍然适用（但是这种情况一定不会进球）。还需要注意球要打在篮板上才会反弹，否则就飞出场外了，因此还要保证 $y_0<y\,|_{x=x_1}\le y_2$。请注意右端可以取等号。

综上，能进球的条件为：

- $\Delta>0$
- $x_{\text{left}}<x_0$
- 以下两个条件满足一个：
  - $x_0<x_{\text{right}}<x_1$
  - $x_1<x_{\text{right}}<2x_1-x_0$ 且 $y_0<y\,|_{x=x_1}\le y_2$

这里

$$x_{\text{left}}=\frac{-b+\sqrt{\Delta}}{2a}$$

$$x_{\text{right}}=\frac{-b-\sqrt{\Delta}}{2a}$$

一些精度误差较小的写法会在不等式两边乘 $2a$，但是一定要注意 $a < 0$ 会导致在两边乘 $2a$ 时不等号方向改变。

虽然在不等号两边平方时需要注意 $y = x^2$ 不是单值函数，需要注意要平方的值的范围，再根据单调性比较，但在本题化简后由几何意义，可以不考虑这种单调性进行比较。

GCD on Tree 题解

本题考虑维护10000个森林，每个森林的节点编号都是$1 \sim n$，第i个森林中任何两个点有边当且仅当这两个点在原树中有边并且这两个点的权值都能够被i整除，这样在每个森林中会形成若干个连通块，我们将一个大小为$size$的连通块的权值大小记为$size * (size + 1)$，第i个森林的权值$h[i]$代表森林中的所有连通块权值之和。若将答案记作$ans[x]$，根据容斥原理那么不难发现$ans[x] = \sum_{x|y} \mu(\frac{y}{x}) \frac{h[y]}{2}$(其中$\mu(x)$为莫比乌斯函数)。

现在考虑如何计算并维护h数组，这里采用的方法是LCT动态开点维护。首先不难在LCT中维护出每个节点所在的连通块的大小以及在该节点去除的情况下该节点所有儿子所在的连通块的大小的权值之和。其次我们需要实现在一颗森林中加入某个点，这个操作可能会让多个不同的连通块合并在一起，我们需要计算这一操作对森林权值产生的增量，同样地，我们也要实现在森林中删除某个点，而这个操作可能会让一个连通块分裂成多个连通块，这一操作也会对森林的权值产生增量。为了快速完成上述操作，我们对每个连通块定根，一个连通块的根就是该连通块在原树中深度最小的节点，然后考虑为每个节点设置一个标记用于表示该节点是否存在，如果一个连通块的根节点是被标记为不存在的，那么这其实是一个连通块群，即由根节点的所有儿子所在的连通块组成，我们能够轻易根据根节点来判断连通块的权值之和。于是在森林中加入某个节点的时候我们只需要将它设置为存在并且将它与其父节点连边(无论父节点是否被标记为存在，当然父节点本身若未被创建则必须被创建，只是标记为不存在)，而删除节点就是断开其与父节点的边即可。

在以上加入节点以及删除节点过程中，我们采用lCT去维护整棵森林权值的变化量，维护的方式就是先算一遍连边之前关联的所有连通块的权值，再连完边后再算一遍，后者减前者即是变化量，由于有了虚拟节点的概念，我们不难维护出与虚拟节点相关联的所有连通块的权值之和。

有了以上转化后，在一开始我们先预处理h数组，通过h数组处理出ans数组，对于之后的每个修改操作`0 u c`我们都枚举c的所有因数，找到这个因数对应的森林，再森林中激活点u即可(当然在此之前需要删除u原来的权值产生的贡献)，由于10000以内的数的因数个数不会很多，最大只有64，因此只需要在最多64个森林中激活即可，而每次激活的复杂度是$O(log(n))$级别，因此这一部分复杂度为$O(64log(n))$，在修改每个森林的同时我们会得到关于这个森林的权值的增量，由于这个森林的权值的增量会对答案产生影响，因此还需要对c的所有因数对应的答案作出修改，最坏情况下需要修改900次，复杂度略小于$O(64log(n))$，这样一次修改操作的总复杂度约为$O(klog(n))$，k大概取值500。而询问可以直接$O(1)$回答，总复杂度约为$O(500Tqlog(n))$，可以通过本题。

Primality Test 题解

当 $x = 1$，$f(x) = 2$，$f(f(x)) = f(2) = 3$，$g(x) = \lfloor \frac{2 + 3}{2} \rfloor = 2$，是质数。

当 $x > 1$，$f(x)$ 和 $f(f(x))$ 都大于 2，且都是质数，因此二者都是奇数，从而 $\frac{f(x) + f(f(x))}{2}$ 是整数。显然 $f(x) < \frac{f(x) + f(f(x))}{2} < f(f(x))$，假设 $\frac{f(x) + f(f(x))}{2}$ 是质数，则必有 $f(f(x)) \le \frac{f(x) + f(f(x))}{2}$，矛盾。故 $\frac{f(x) + f(f(x))}{2}$ 是合数。

综上所述，当 $x = 1$，输出 `YES`，否则输出 `NO`。

通俗的解释：当 $x > 1$ 时，$\frac{f(x) + f(f(x))}{2}$ 必然在两个相邻质数之间，所以必然是合数。

Monopoly 题解

首先很容易想到，我们以 n 个数为一个周期考虑，令序列的前缀和为 s_i，原序列 n 个数的和为 S，可知 $s_{i+n} = s_i + S$。

首先讨论 $S > 0$ 的情况。对于询问的 x，我们需找到最小的 i 使得 $s_i = x$，令 $i = j + kn$（$1 \le j \le n$，$k \ge 0$），则 $s_i = s_{j+kn} = s_j + kS$，此时有 $s_i \equiv s_j \mod S$。因此对于询问的 x，我们需要找到原序列前缀和中的一个 s_j，使得两者对 S 取模同余，并且有 $s_j \le x$。如果有多个满足条件，我们则需要找到使得 $x = s_j + kS$ 中 k 最小的 s_j，不难看出这是满足条件的数中最大的那个。如果仍然有多个这样的数，则取位置最靠前的。

通过上述分析可知，我们需要维护一种数据结构，对于询问的 x，我们可以在所有与其模 S 同余的数中，找到小于等于 x 的最大的数。如果说 S 不大，我们可以对 0 到 $S-1$ 的余数各自维护一个有序的数据结构（如 vector+sort、set、map等），对于询问的 x 在对应的数据结构中二分查找即可找到找到小于等于 x 的最大的数，同时得到其最靠前的位置。此题 S 可以很大，但原序列的数并不多，可以通过 map 或哈希解决。此题标程使用了 map 套 map，外层 map 是一种余数对应一个内层 map，而内层 map 则是每个数对应其最靠前的位置，时间复杂度 $O(m \log^2 n)$。若外层采用哈希，则时间复杂度为 $O(m \log n)$。

然后是 $S < 0$ 的情况，此时如果我们把原序列所有数取相反数，同时对询问的 x 也取相反数，则可转化为 $S > 0$ 的情况。

最后是 $S = 0$ 的情况，此时如果对 S 取模会导致运行错误，所以必须特判。此时序列的前缀和只是原封不动地进行周期重复，所以只需要在原序列前缀和中找是否有 x 即可。

Nuh Heh Heh Aaaaaaaaaaa 题解

考虑 dp。将 NuhHehHeh 与后面的 a 拆开，设 $f(i, j)$ 表示前 i 位中匹配到第 j 位的方案数，之后对于每个 $f(i, 9)$ 乘上后面 a 的个数贡献就行。

Occupying Grids 题解

*注：此题假了，出题人已立正接受挨打（详情见知乎回答）。

Subpermutation 题解

答案分两部分，落在一个排列内的和落在两个排列边界的

排列内的直接推导计算：

在长度为n的这个排列中选择连续的x个数

$(n-x+1)x!(n-x)!$

边界处的：

考虑什么样的排列和他的下一个排列的交界处会产生答案

性质：当前序列最长下降后缀的前一位会和该后缀中第一个大于他的数字交换，并重新排序该后缀形成下一个后缀

如：13542最长下降后缀为542，前一位为3，交换3和4得到14532，重新排序后缀532得到14235，即为下一个后缀。

假设当前排列为p，下一个排列为q

1. p首尾元素必须<=x，且>x的元素都在中间

如果不满足此条件：如果能匹配成功则q的首部一定会发生变化，考虑存在<x的元素在中间部分，故最长递 减后缀的首位一定不会在首部的下一位，所以首部不发生变化。

2. q满足首部元素种类和p相同(显然)

先不考虑下一个排列，只满足条件1的排列数可以同上一种情况的计算方式计算：$(x-1)x!(n-x)!$

之后要减去不满足条件2的情况

假设当前排列p长这个样子

A	B	C
首部 <=x	中间元素 >x	尾部 <=x

根据性质，只有当p的BC部分递减时，q的A部分的元素会发生变化，且无法左右移动答案区间（左移加入了p的B中元素，右移或区间不动一定包含了被交换位的元素，该元素>x），故不满足条件。

情况二的答案为$x!(x-1)(n-x)!-\sum_{i=1}^{x-1}C_x^i(x-i)!$

综合两种情况

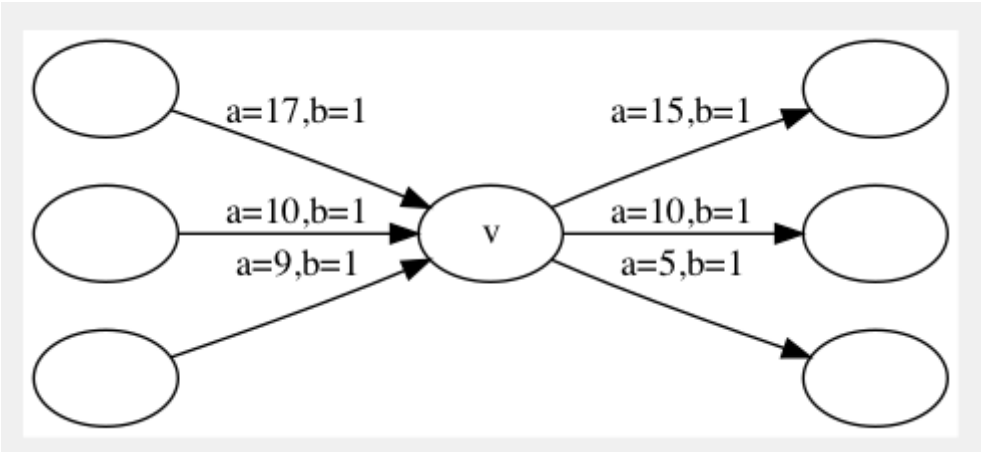
$ans=x!((x-1)(n-x)!-\sum_{i=1}^{x-1}\frac{1}{i!}+(n-x+1)(n-x)!)$

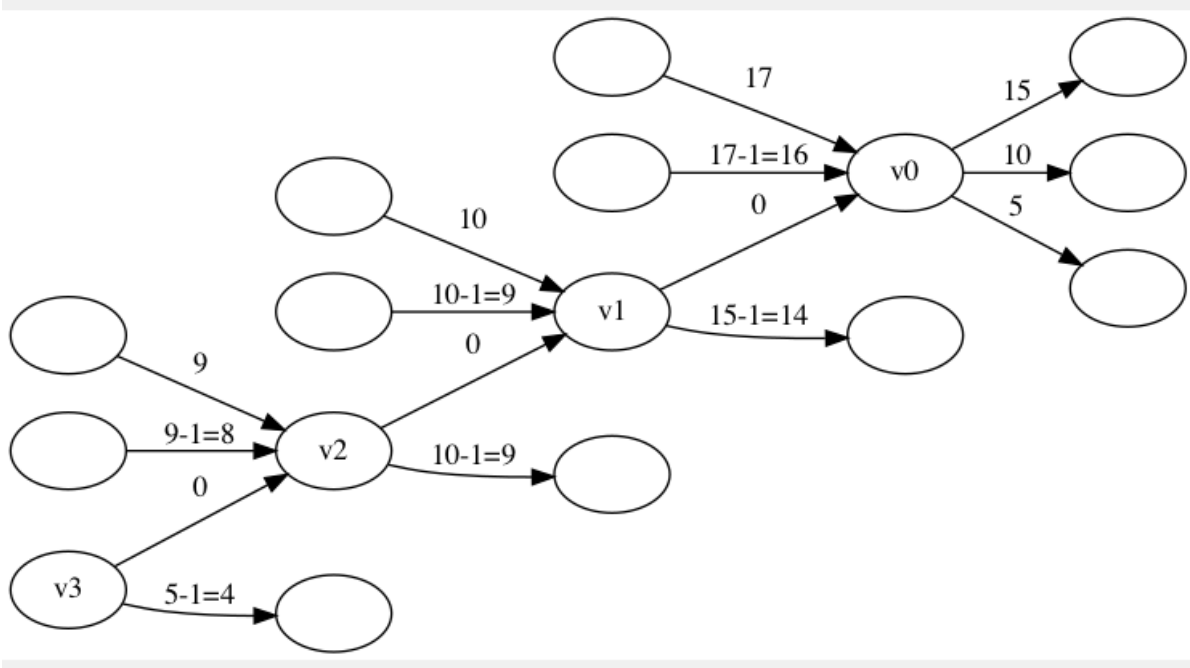
其中$\sum_{i=1}^{x-1}\frac{1}{i!}$预处理即可

Public Transport System 题解

原图中的一条边 e 有两个边权 a_e, b_e，不方便我们处理。于是考虑将其拆成边权为 a_e 和 $a_e - b_e$ 的两条（只有一个边权的）边。在任何情况下，我们都可以走边权为 a_e 的边，但走边权为 $a_e - b_e$ 的边则需满足题目所述的条件。直接的想法是为每条入边连接所有可行的出边，这样做的复杂度是无法接受的，但我们发现一条入边对应的可行的出边的边权 a_e 总是在一个区间中，于是有了优化的方法。

具体的，如下图所示，我们将结点 v 拆成 $d+1$ 个结点 $v_0, v_1, v_2, \ldots, v_d$，其中 d 是 v 的出度。将所有边权为 $a_e - b_e$ 的出边按 a_e 从大到小排序，分别以结点 $v_1, v_2, \ldots, v_d$ 作为其起点。所有边权为 a_e 的出边则以 v_0 作为起点。然后对所有的 $1 \le i \le d$，从 v_i 连一条边权为 0 的边到 v_{i-1}。这样对所有的入边，我们只要找到第一个对应的出边的边权 a_e 大于该入边的边权 a_{e-1} 的结点作为其终点连上即可。





这样拆点后的总点数为 $n+m$，总边数为 $3m$，使用二叉堆优化的 Dijkstra 算法时间复杂度为 $O(n+m\log m)$。

Bigraph Extension 题解

首先假定 $m=0$，考虑到题目中的限制条件——任意二分图两侧两点都连通，则连边数量最小的情况是将 $2n$ 个点连成一棵树，加边数为 $2n-1$ 。由于树上两点路径唯一，考虑树上的一条边上的两点，最长路长度为 1 。不符合最长路长度严格大于 n 的条件，可知加边数下界为 $2n$ 。

下面给出边数为 $2n$ 的构造方式：将 $2n$ 个点连成一个环，由于环上任意两点间的路径有两条，设距离分别为 a 和 b ，有 $a+b=2n$ 。同时因为所选定两点分别属于二分图的两侧，a、b 只能是奇数。又因为 n 是偶数，不会存在 $a=b=n$ 的情况，即 a 和 b 不同。则任选两点均有 $min(max(a,b))\geq n+1$ 成立。

下面考虑 $m\neq 0$ 的情况。由于题中的给定边保证任意两条边无公共点，则输入不会存在构成子环的情况。同时题目中给定的加边操作都可以看做将边两端的点合成一个新的点，这样一定可以通过加边构成上面提到的 $2n$ 环，不会多出额外的边。最小加边数为 $2n-m$ 。

最后考虑字典序最小的限制。我们先从小到大枚举 A 中的点，再从小到大枚举 B 中的点，通过维护并查集和度数数组贪心判断两点能否连成链。这样保证了前 $2n-m-1$ 条边的字典序最小。此时只剩最后一条边，我们遍历度数数组找到 A、B 中度数为 1 的点连起来，放在第 $2n-m$ 条边的位置即可。（这条边中 A 集合度数为 1 的点一定是 n 号点，显然字典序比其他边更大，可用反证法证明）

由于在加边过程中， B 集合中最小的一部分点的度数已经变成 2 了，不会参与后续的加边过程。可以参考 $dinic$ 算法的当前弧优化，记录 B 中度数小于 2 的编号最小的点的编号 top 。每次从 top 开始枚举 B 集合的点，每次加边后维护 top 。可以将复杂度降为 $O(n)$ 。

Jumping Monkey 题解

考虑点权最大的那个结点（设为 u），显然不管从其他哪个结点开始跳一次，都可以到达 u，且不能越过 u，到达 u 后也不能再跳向其他结点。于是结点 u 一定是最优方案中最后一个到达的点。我们将结点 u 从树上去掉（相连的边也去掉），对剩下的每个连通块都递归进行上述过程即可得到答案。

但要实现上述递归过程并不容易，我们转而考虑反向进行这个过程：按点权从小到大枚举结点 u，并将 u 作为所有与 u 相连的（已经枚举过的）连通块的根。从而建立一棵新树，每个结点的深度（根的深度为 1）即是答案。

时间复杂度 $O(n \log n)$。

Contest Remake 题解

对最大的因子，我们讨论其与一常数 T 的关系：

1. 如果小于等于 T，我们直接暴力求出小于等于 T 的所有数的乘积方案；
2. 如果大于 T，我们可以用 $dp_{i,j}$ 表示当前最大因子是 i，乘积是 j 的方案数，其中 j 取值范围在 $[1, \frac{C}{T}]$。整个 DP 过程的复杂度是一个调和级数。剩下的最大因子的取值是一个范围，就变成我们有很多个询问 $[L, R]$ 范围内合法数的个数。

这个过程有几个优化点：

- 暴力求小于等于 T 的答案时候，可以拆成两部分 $[1, A]$ 与 $[A+1, T]$ 分别计算， 然后类似折半搜索计算方案数，这样 T 可以取得更大。
- DP 的时候可以把 $i \le T$ 的部分的 DP 值直接用第一部分得到的结果填进去，这样这个 DP 过程的时间复杂度基本接近线性。
- 询问 $[L, R]$ 时，大部分的 L, R 都小于等于 10^6。这部分可以预处理前缀和做到 $O(1)$ 的时间复杂度，只有几百个询问需要 $\log$ 级别的查询。

取 $T = 50$，这样极限数据 $n = 10^5, C = 10^9$ 的时候，可以在 0.4s 左右跑出一组数据。