D. 座位安排 (seatingplan)

时间限制: 4 秒
空间限制: 1024 MiB

EGOI 闭幕式马上就要到了，将有 N 位重要宾客出席。 他们需要按照非常严格的外交协议，排成一排坐好。 Noemi 为了确定正确的座位顺序，熬了两个通宵。

Veronica 是这次闭幕式的负责人，她得确保前排的座位名牌是正确的。 但有个小麻烦：Noemi 没有告诉她正确的座位顺序，而且人也找不到了。 好在摄影师 Dorka 有个能派上用场的 App。

Dorka 需要调整相机，以便给前排的宾客拍特定的照片。为了调试设备，她需要知道每张照片会覆盖多大的范围，所以 Noemi 给她写了个 App，能快速输出她需要的信息。 Veronica 现在想用这个 App 来找出正确的座位安排。

这 N 位重要宾客编号从 0 到 $N-1$。 前排的座位也从左到右编号为 0 到 $N-1$。 对于每个 I $(0 \le I \le N-1)$，g_I 表示坐在座位 I 上的宾客编号，s_I 表示宾客 I 应该坐的座位编号。

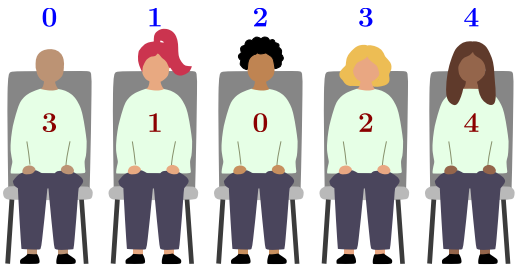

图 1　一排五个宾客。对于这一排，$g = [3, 1, 0, 2, 4]$ 且 $s = [2, 1, 3, 0, 4]$。

这个 App 的工作方式如下：

- Dorka 输入三个不同宾客的编号 I、J、K。
- App 会告诉她，如果非要把这三位宾客都拍进照片里，照片里至少会出现多少位宾客。形式化地说，App 会显示 $\max(s_I, s_J, s_K) - \min(s_I, s_J, s_K) + 1$ 的值。

举个例子，看看 图 1 ：

- 宾客 $I = 0$、$J = 2$ 和 $K = 4$ 坐在位置 $s_I = 2$、$s_J = 3$ 和 $s_K = 4$ 上。如果 Dorka 选择了他们，App 会显示 $\max(2, 3, 4) - \min(2, 3, 4) + 1 = 3$。

  换句话说，包含宾客 0、2 和 4 的最窄照片里，刚好只有这三位宾客。

- 宾客 $I = 0$、$J = 4$ 和 $K = 3$ 坐在位置 $s_I = 2$、$s_J = 4$ 和 $s_K = 0$ 上。如果 Dorka 选择了他们，App 会显示 $\max(2, 4, 0) - \min(2, 4, 0) + 1 = 5$。

  换句话说，包含这三位宾客的照片必须覆盖所有 5 位宾客。

帮 Veronica 用 Dorka 的 App 找出正确的座位顺序。 具体来说，你的程序需要确定并输出序列 $g_0, g_1, \ldots, g_{N-1}$。 总共有两个正确的答案（一个是另一个的逆序），你可以输出其中任意一个。 你的得分取决于你向 App 查询的次数。

实现详情

⇨ 这是一个交互题。你的程序将通过标准输入和输出与评测程序进行交互，格式如下所述。

你的程序应该首先输入一行，包含一个正整数 T，表示测试数据的数量。

对于每个测试数据，你的程序应该先输入一行，包含一个正整数 N，即座位的数量（也是宾客的数量）。

如果要进行查询，你的程序应该输出一行 "? I J K"，其中 $0 \le I, J, K \le N-1$ 是三个**不同**的数字。

查询后，你的程序应该输入一行，包含一个正整数，即查询的答案。

如果要输出正确的座位顺序，你的程序应该输出一行 "! g_0 ⋯ g_{N-1}"。

处理完所有 T 个测试数据后，程序应正常终止。

请注意，CMS 中使用的官方评测机可能是 **自适应的（adaptive）**。 这意味着，对于某些测试用例，嘉宾的排列顺序并不是提前定好的。 相反，评测机可能会根据你的程序已经问出的查询，来决定使用剩余排列中的哪一种。

刷新缓冲区。如果你没有使用提供的模板，请确保在输出每一行后刷新标准输出，否则你的程序可能会被判定为 *Not correct*。在 Python 中，如果你使用 `input()` 输入行，这会自动完成，你可以使用 `print(..., flush=True)` 来强制刷新。在 C++ 中，`cout << endl;` 在输出换行符之外还会进行刷新；如果使用 `printf`，请使用 `fflush(stdout)`。

约束条件

- $1 \le T \le 10$。
- N 为 5（只有样例符合）、8、40 或 2000。
- 对于每个测试数据，你最多可以进行 10 000 次查询。

评分方式

你的程序将在分成若干子任务的测试数据上进行测试。要获得某个子任务的分数，你必须正确解出该子任务中所有的测试数据。

- **子任务 0 [0 分]**: 样例 ($N = 5$)。
- **子任务 1 [9 分]**: $N = 8$。
- **子任务 2 [11 分]**: $N = 2000$，且宾客 0 和 1 坐在一起。
- **子任务 3 [15 分]**: $N = 40$。
- **子任务 4 [65 分]**: $N = 2000$。

对于子任务 1 和 2，任何能正确解决所有测试数据的解法都将获得全部分数。

对于子任务 3 和 4，你的解法必须正确解决所有测试数据才能得分，且你的得分取决于 Q_s，即你解决一个测试数据所需的最大查询次数。设 $X_s = \max(1, Q_s/N)$。子任务 3 和 4 的得分计算如下：

$$\text{score}_3 = \min\left(15, 3 + \frac{19}{X_s^{1.5}}\right), \qquad \text{score}_4 = \min\left(65, 3 + \frac{91}{X_s^{1.5}}\right)$$

对于每个子任务，score_s 的值四舍五入到最接近的整数，总分是各子任务得分之和。 为了获得满分，你需要在子任务 3 中最多使用 55 次查询，在子任务 4 中最多使用 2597 次查询。 子任务 3 和 4 的 Q_s 样例值及对应得分如下表所示。

Q_s	55	56	60	70	80	100	150	10000
score_3	15	14	13	11	10	8	6	3

Q_s	2597	2800	3000	4000	5000	6000	8000	10000
score_4	65	58	53	35	26	21	14	11

样例

评测程序	解法
1 5	
	? 0 2 4
3	
	? 3 0 1
3	
	? 0 4 3
5	
	! 3 1 0 2 4

样例解释

样例输入包含一个测试数据 ($T = 1$), 其中 $N = 5$ 位宾客。此测试数据中的隐藏宾客配置对应于 图 1。

程序进行的第一次查询是 0, 2, 4。 该查询的答案 3 告诉我们，这些宾客以某种未知顺序坐在三个相邻的座位上。

第二个查询的答案 3 告诉我们关于宾客 3、0 和 1 也是如此。

我们现在可以推断出宾客 0 一定坐在中间，宾客 2 和 4 在一侧，宾客 1 和 3 在另一侧。

第三次查询后，我们已经确定宾客一定以 $[3, 1, 0, 2, 4]$ 的顺序或反向顺序 $[4, 2, 0, 1, 3]$ 就座。 我们可以输出这两种顺序中的任意一种。

CMS 中的代码模板和评测详情

我们强烈建议使用提供的 C++ 和 Python 代码模板。这些模板会检查与评测程序的交互是否成功, 并在交互失败时优雅地终止程序。

与你程序交互的评测程序在遇到第一个错误时会进行报错, 然后终止。如果你不使用提供的模板, 这可能会导致你的程序崩溃或一直等待响应。如果发生这种情况, 你在 CMS 中通常会收到类似 “Execution timed out (wall clock limit exceeded)” 这样无意义的评测结果。

我们也建议使用测试工具（见下文）在提交前进行本地测试。测试工具会检查你程序的输出并报告协议违规情况（protocol violations）。

测试工具

为了方便你测试程序，我们提供了一个可以从 CMS 下载的简单工具。 该工具是可选的。注意，CMS 上的官方评测程序与测试工具不同。

要使用该工具，你需要一个输入文件。你可以使用提供的样例输入 `seatingplan.input0.txt` 或者自己创建一个。输入文件应以包含测试数据数量 T 的行开头, 然后每个测试数据占用两行: 一行包含 N, 另一行包含 $g_0, g_1, \ldots, g_{N-1}$。

对于 Python 程序，假设为 seatingplan.py（通常以 pypy3 seatingplan.py 运行），按如下方式运行测试工具：

```
python3 testing_tool.py pypy3 seatingplan.py < seatingplan.input0.txt
```

对于 C++ 程序，先编译你的程序：

```
g++ -DEVAL -std=gnu++20 -O2 -pipe -static -s -o seatingplan seatingplan.cpp
```

然后运行测试工具：

```
python3 testing_tool.py ./seatingplan < seatingplan.input0.txt
```