注意事项

每个题目都有一个附件包，你可以从 CMS 系统或你的机器上取得这个附件包。

- 附件包中包括了评测工具的示例、实现的示例、作为例子的测试数据以及编译脚本。
- 你只能提交一个文件，并且最多只能提交 50 次。
- 你所提交的程序不允许读取标准输入，也不允许输出到标准输出或者与其他任何文件进行交互。但是，你的程序可以输出到标准错误流。
- 在题面的上方给出了你应当提交的文件名。它应当实现在题面中列出的函数，其名称和接口应与给出的实现示例一致。
- 你可以根据需要来实现其他的函数。
- 当使用评测工具示例来测试你的程序时，你的输入应当与题面中规定的格式和限制条件相匹配，否则有可能导致未知的结果。

约定

在题面描述中，用统一的数据类型代称bool，integer，int64，以及 `int[]`(数组)指定了函数名和函数接口。

对于所支持的每种编程语言，测试工具所用的相应数据类型和函数如下表所示：

语言	**bool**	**int**	**int64**	**int[]**	求数组**a**的长度
C++	`bool`	`int`	`long long`	`std::vector<int>`	`a.size()`
Pascal	`boolean`	`longint`	`int64`	`array of longint`	`Length(a)`
Java	`boolean`	`int`	`long`	`int[]`	`a.length`

限制

题目	时间限制	内存限制
prize	1秒	1024 MB
simurgh	3秒	1024 MB
books	2秒	1024 MB

大奖（The Big Prize）

“大奖”是一个家喻户晓的TV游戏节目。这次你很幸运地进入到最后一轮。已知编号从0到$n-1$的 n 个盒子从左到右排成一行，你就站在这排盒子的前面。每个盒子里面放有一个奖品，必须打开盒子才能看到是什么奖品。已知有$v \geq 2$种不同*类型*的奖品。这v种类型按照奖品价值的降序从1到v排列。

类型1的奖品是一块钻石，价值最高。所有盒子加起来刚好只有一块钻石。类型v的奖品是一块棒棒糖，价值最低。为使得游戏更加激动人心，相对便宜的奖品数量远比价值昂贵的奖品数量要多。更具体一点，对于满足$2 \leq t \leq v$的所有t，我们有： 如果类型$t-1$的奖品有k个，那么类型t的奖品将严格多于 k^2个。

你的目标是赢得那块钻石。在游戏结束时，你必须打开一个盒子并收取盒子内的奖品。在选择要打开的盒子之前，你可以向节目主持人Rambod提一些问题。在每一个问题中，你选择就某个i号盒子进行提问。Rambod将给你一个包含两个整数的数组 a作为回答。这两个整数的意义如下：

- 在i号盒子左面的盒子中，刚好有$a[0]$个盒子中的奖品，价值比i号盒子中的奖品价值要高。
- 在i号盒子右面的盒子中，刚好有$a[1]$个盒子中的奖品，价值比i号盒子中的奖品价值要高。

例如，假设$n = 8$，在你的问题中，你选择就$i = 2$号盒子进行提问。Rambod的回答是$a = [1, 2]$。这个回答的意义是：

- 0号盒子和1号盒子中恰好有一个盒子中的奖品比2号盒子中的奖品更贵。
- 在$3, 4, \ldots, 7$号盒子中恰好有2个盒子中的奖品比2号盒子中的奖品更贵。

你的任务就是通过问少量的问题找出包含钻石的盒子。

实现细节

你应当实现如下函数段:

```
int find_best(int n)
```

- 此函数只被评测程序调用仅一次。
- n: 盒子的数量.
- 你实现的这个函数应该返回装有钻石的盒子编号，即，唯一的整数d $(0 \leq d \leq n-1)$使得 d号盒子放有类型为1的奖品。

上述函数可以调用下列函数:

```
int[] ask(int i)
```

- i: 你在询问时选择的盒子编号。i的数值必须介于0 和 $n-1$之间（含）。
- 这个函数返回包含2个元素的数组a。其中，$a[0]$是在i号盒子的左面，奖品比i号盒子的奖品价值更高的盒子数目。而$a[1]$则是在i号盒子右面，奖品比i号盒子的奖品价值更高的盒子数目。

例子

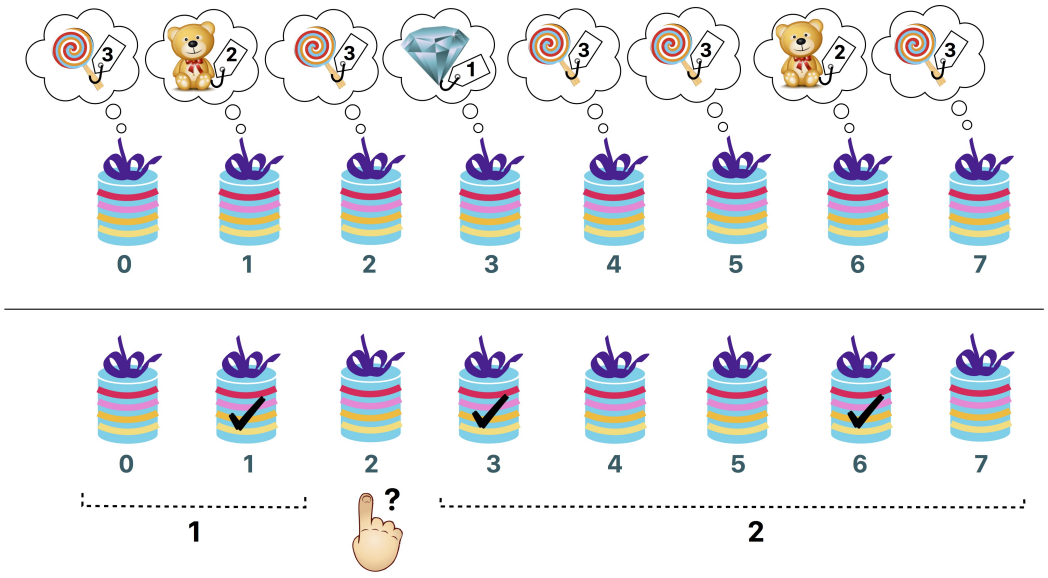
评测工具将做如下函数调用:

```
find_best(8)
```

这里有$n=8$个盒子。假定奖品类型为 $[3,2,3,1,3,3,2,3]$。对函数 `ask`的所有可能的调用以及相应的返回值列出如下：

- `ask(0)` 返回 $[0,3]$
- `ask(1)` 返回 $[0,1]$
- `ask(2)` 返回 $[1,2]$
- `ask(3)` 返回 $[0,0]$
- `ask(4)` 返回 $[2,1]$
- `ask(5)` 返回 $[2,1]$
- `ask(6)` 返回 $[1,0]$
- `ask(7)` 返回 $[3,0]$

在这个例子中, 钻石放在3号盒子里。所以函数 `find_best`应该返回3。



上图阐释了这个例子。图中上半部分给出了每个盒子中奖品的类型。图中的下半部分阐释了询问`ask(2)`。做了标记（打√）的盒子中装有比2号盒子的奖品价值更高的奖品。

限制

- $3 \leq n \leq 200\,000$.
- 每个盒子中奖品的类型介于 1 和 v之间（含）.
- 类型1的奖品恰有一个。
- 对于所有 $2 \leq t \leq v$，如果类型$t-1$的奖品有 k个，那么类型t的奖品将严格多于 k^2个。

子任务与评分

在某些测试数据中，评测工具的行为是自适应的。这意味着在这些测试数据中评测工具并没有一个固定的奖品序列。取而代之的是，由评测工具给出的答案可能依赖于你的程序问过的问题。评测工具的回答方式可以保证，在每次回答之后，至少有一个奖品序列与到目前为止给出的所有回答都不冲突。

1. (20 分) 恰好有 1 个钻石和 $n-1$ 个棒棒糖 (所以, $v=2$)。你可以调用函数 `ask` 最多 $10\,000$ 次。
2. (80 分) 没有附加限制。

在子任务2 中你可以获得部分分。令 q 是在这个子任务的所有测试数据上函数 `ask`被调用的最大次数，那么你在这个子任务上的得分将按照下表计算：

问题	得分
$10\,000 < q$	0 (在CMS中报告为 'Wrong Answer')
$6000 < q \leq 10\,000$	70
$5000 < q \leq 6000$	$80 - (q - 5000)/100$
$q \leq 5000$	80

评测工具示例

评测工具的这个示例不是自适应的。取而代之的是，它只是读取并使用一个固定的奖品类型的数组 p。 对于所有的$0 \leq b \leq n-1$，b号盒子中的奖品类型将由 $p[b]$给出。评测工具示例期望的输入格式如下：

- 第1行: n
- 第2行: $p[0]$ $p[1]$ $\ldots$ $p[n-1]$

评测工具示例输出单独一行，其中包含 `find_best` 的返回值以及调用函数`ask`的次数。

西默夫（Simurgh）

根据沙纳玛（Shahnameh）中的古代波斯传说，Zal，传奇的波斯英雄，疯狂地爱上了Kabul王国的公主Rudaba。在Zal向Rudaba求婚时，Rudaba的父亲给他了一个挑战。

在波斯有n个城市，标记为从0到$n-1$，以及m条双向道路，标记为从0到$m-1$。 每条道路连接两个不同的城市。每一对城市至多会被一条道路连接。有些道路是*御道（royal roads）*，专用于皇室行驶，但这是保密的。Zal的任务是找出哪些道路是御道。

Zal有一张包括所有城市和所有道路的波斯地图。他不知道哪些道路是御道，但是他可以求救于Simurgh——好心的神鸟、Zal的保护者。然而，Simurgh并不想直接告诉他哪些道路是御道。作为替代，Simurgh告诉Zal，所有御道的集合是一个*黄金集合（golden set）*。一个道路的集合是黄金集合，当且仅当：

- 它*恰好*包含$n-1$条道路，而且
- 对于每一对城市，仅沿着这个集合中的道路即可从其中一个城市抵达另外一个城市。

此外，Zal可以问Simurgh一些问题。对于每个问题：

1. Zal选出道路的一个*黄金*集合，然后
2. Simurgh会告诉Zal，在所选择的黄金集合中有多少条道路是御道。

你的程序可以问Simurgh最多q个问题，以此帮助Zal找出御道的集合。评测工具将扮演Simurgh的角色。

实现细节

你需要实现下面的函数：

```
int[] find_roads(int n, int[] u, int[] v)
```

- n：城市的数量，
- u和v：均为长度为m的数组。对于所有$0 \le i \le m-1$，$u[i]$和$v[i]$是被道路i所连接的城市。
- 该函数需要返回一个长度为$n-1$的数组，其中包括了所有御道的标号（可以以任意的顺序给出）。

你的程序至多只能调用评测工具中的如下函数q次：

```
int count_common_roads(int[] r)
```

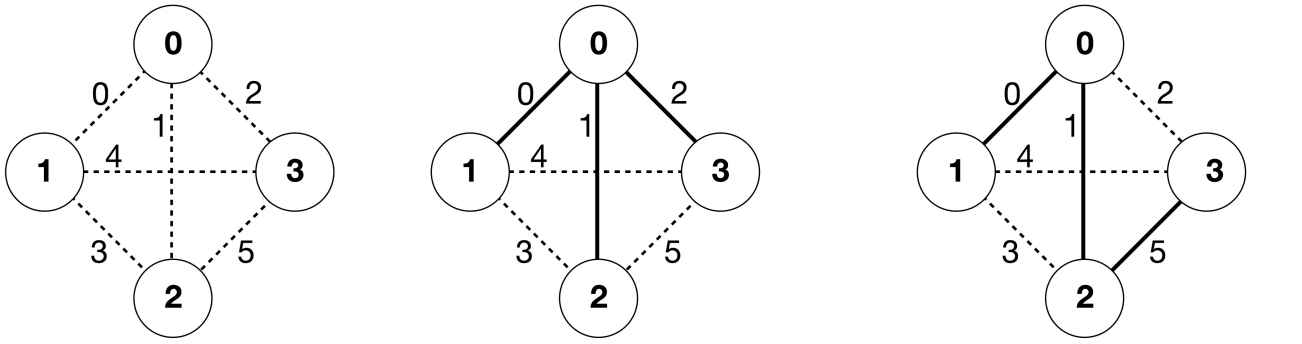
- r：长度为$n-1$的数组，其中包括了一个黄金集合中的道路标号（可以以任意的顺序给出）。

- 该函数将返回r中的御道数量。

例子

```
find_roads(4, [0, 0, 0, 1, 1, 2], [1, 2, 3, 2, 3, 3])
```

`find_roads(...)` `count_common_roads([0, 1, 2]) = 2` `count_common_roads([5, 1, 0]) = 3`



这个例子中有4个城市和6条道路。 我们将连接城市a和b的道路表示为(a, b)。这些道路按照下面的顺序被标为从0到5：$(0, 1)$，$(0, 2)$，$(0, 3)$，$(1, 2)$，$(1, 3)$和$(2, 3)$。每个黄金集合包含$n - 1 = 3$条道路。

假设御道是标号为0，1和5的道路，即$(0, 1)$，$(0, 2)$和$(2, 3)$。这样的话：

- `count_common_roads([0, 1, 2])`返回2。该询问涉及到标号为$0, 1$和2的道路，即$(0, 1)$，$(0, 2)$和$(0, 3)$。其中有两条道路是御道。
- `count_common_roads([5, 1, 0])`返回3。该询问涉及到所有的御道。

函数`find_roads`需要返回`[5, 1, 0]`或任意其他包含这三个元素且长度为3的数组。

注意，下面列出的调用是不允许的：

- `count_common_roads([0, 1])`：这里r的长度不是3。
- `count_common_roads([0, 1, 3])`：这里r不是一个黄金集合，因为无法仅沿道路$(0, 1)$，$(0, 2)$，$(1, 2)$就从城市0走到城市3。

限制条件

- $2 \le n \le 500$
- $n - 1 \le m \le n(n - 1)/2$
- $0 \le u[i], v[i] \le n - 1$（对于所有$0 \le i \le m - 1$）
- 对于所有$0 \le i \le m - 1$，道路i连接两个不同的城市（即$u[i] \neq v[i]$）。
- 每对城市之间至多连有一条道路。
- 经由这些道路，可以在任意一对城市之间来往。
- 所有的御道组成一个黄金集合。
- `find_roads`可以调用`count_common_roads`最多q次。在每次调用中，由r所给出的道路必须

是一个黄金集合。

子任务

1. （13分）$n \leq 7$，$q = 30\,000$
2. （17分）$n \leq 50$，$q = 30\,000$
3. （21分）$n \leq 240$，$q = 30\,000$
4. （19分）$q = 12\,000$，在任意两个城市之间都连有一条道路
5. （30分） $q = 8000$

评测工具示例

评测工具示例将读入下述格式的输入数据：

- 第1行： n m
- 第$2 + i$行（对于所有$0 \leq i \leq m - 1$）： $u[i]$ $v[i]$
- 第$2 + m$行： $s[0]$ $s[1]$ $\ldots$ $s[n - 2]$

这里$s[0], s[1], \ldots, s[n - 2]$是所有御道的标号。

如果 `find_roads`最多调用`count_common_roads`了$30\,000$次，而且正确地返回了御道的集合，评测工具示例将会输出`YES`。 否则评测工具示例将会输出`NO`。

需要明确的是，评测工具示例中的函数`count_common_roads`不会检查r是否满足一个黄金集合的所有条件。替代性地，它会对数组r中的御道进行计数，并且返回。然而，在你提交的程序调用`count_common_roads`时，如果传给它的不是对应某个黄金集合的标号集合，评测结果将会是'Wrong Answer'.

技术提示

出于效率方面的考虑，面向C++和Pascal的函数`count_common_roads`使用了*传引用调用（call by reference）*的方式。你可以与平常一样调用这个函数。评测工具确保不会改变r中的值。

古书（Ancient Books）

德黑兰市是伊朗国家图书馆的所在地。这个图书馆的镇馆之宝位于一个长长的大厅内，大厅里有排成一行的n张桌子，从左到右依次编号为从0到$n-1$。每张桌子上都陈列着一本手写的古书。这些古书是根据其历史年份进行排序的，这使得访客们难以根据书名来查找它们。所以，图书馆主管决定按照书名的字母序来重新排列它们。

图书管理员Aryan要完成这项工作。他创建了一个长度为n的列表p，其中包括由0到$n-1$的不同整数。这个列表描述了按字母序来重排古书所要做的改变：对于$0 \leq i < n$，目前在桌子i上的古书应该被移到桌子$p[i]$上。

Aryan从桌子s开始重排这些古书。他希望在做完重排工作之后再回到同一张桌子上。由于这些古书非常珍贵，在任何时间，他手持的古书都不能超过一本。在重排古书的过程中，Aryan将会做一系列的操作。每个操作只能是以下其中之一：

- 如果他手上没有书，而他所在的桌子上恰好有一本书时，他可以拿起这本书。
- 如果他手上有一本书，而他所在的桌子上恰好有另一本书时，他可以把手上的书和桌子上的书进行交换。
- 如果他手上有一本书，而他所在的桌子上没有书时，他可以把手上的书放到这个桌子上。
- 他可以走到任何一张桌子前。当他进行这个操作时，他手上可以拿一本书。

对于所有$0 \leq i, j \leq n-1$，桌子i和桌子j之间的距离正好是$|j-i|$米。你的任务是，计算出Aryan重排好所有古书所走过的总距离的最小值。

实现细节

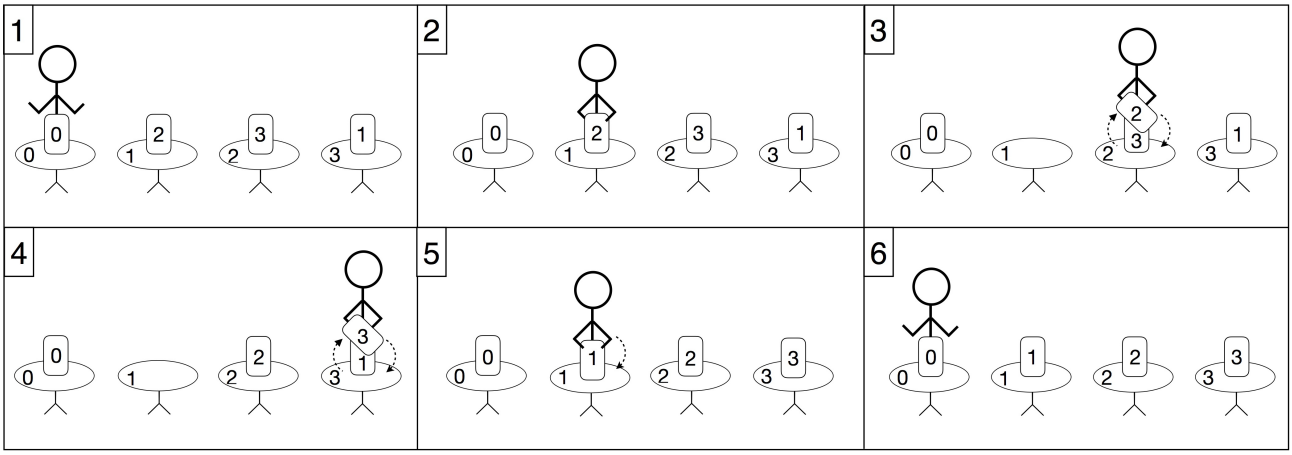
你需要实现下面的函数：

```
int64 minimum_walk(int[] p, int s)
```

- p 是一个长度为 n的数组。初始阶段在桌子i上的古书需要被Aryan移到桌子$p[i]$上（对于所有$0 \leq i < n$）。
- s是初始阶段Aryan所在桌子的编号，同时也是重排好所有古书之后他应该在的位置。
- 该函数要返回Aryan重排好所有古书所需走过的总距离的最小值（以米为单位）。

例子

```
minimum_walk([0, 2, 3, 1], 0)
```



在这个例子中，$n = 4$，在初始阶段Aryan位于桌子0处。他按照如下步骤进行重排：

- 走到桌子1处并且拿起桌上的书。这本书应该要被放到桌子2上。
- 然后，他走到桌子2处，并且把他手上的书和桌子上的书进行交换。现在他新拿到手上的书应该被放到桌子3上。
- 然后，他走到桌子3处，并且把他手上的书和桌子上的书进行交换。现在他新拿到手上的书应该被放到桌子1上。
- 然后，他走到桌子1处，并且把他手上的书放到桌子上。
- 最后，他回到桌子0处。

注意，桌子0上的书已经在正确的位置，即桌子0上，因此Aryan不需要把它拿起来。在这个方案中，他的总行走距离是6米。这是一个最优解；因此，函数应该返回6。

限制

- $1 \leq n \leq 1\,000\,000$
- $0 \leq s \leq n - 1$
- 数组p包含n个从0到$n - 1$（含）的不同整数。

子任务

1. （12分）$n \leq 4$且$s = 0$
2. （10分）$n \leq 1000$且$s = 0$
3. （28分）$s = 0$
4. （20分）$n \leq 1000$
5. （30分）没有附加任何限制

评测工具示例

评测工具示例将会读取如下格式的输入数据：

- 第1行： n s
- 第2行： $p[0]$ $p[1]$ $\ldots$ $p[n-1]$

评测工具示例将输出一行，其中包括`minimum_walk`的返回值。