I 君的商店（shop）

这是一道交互题。

【题目背景】

V 君、I 君和 Y 君是好朋友。

I 君最近开了一家商店，商店里准备了 N 种物品（编号为 $0 \sim N-1$ 中的整数），每种物品均有无限个可供出售，每种物品的单价是 0 或者 1。

V 君想知道每个物品的价格，他已经通过某种超自然力量知道，这 N 个物品里，价格是 1 的物品恰好有奇数/偶数个，且**至少存在一个物品的价格是 1**。

然而，V 君并不想自己去问 I 君，他选择了这样一种方法：他准备了 $+\infty$ 的钱给 Y 君。然后让 Y 君帮他跑腿：每一次，他会给 Y 君指定两个非空物品**集合** S, T（同一个集合内的物品必须两两不同，即每种物品在每个集合类最多包含一个），Y 君会跑到商店，分别买下这两个集合的物品，把他们送回来，并告诉 V 君哪个集合的物品价格之和更高。但是，当**两集合价格之和相等**的时候，Y 君会按照 I 君的指示来回答 V 君。

带着很多物品跑腿是一个很累的事情，因此，我们定义一次跑腿的体力消耗是 $\mathsf{S}+\mathsf{T}$。其中，S 表示集合 S 包含的物品个数。

你的任务是：写一个程序，帮助 V 君决定如何合理地让 Y 君跑腿，从而推算出每种物品的价值。Y 君的体力有限，你当然不能让他过于劳 累，也即，你不能让他的总体力消耗超过某个预设的阈值 M。

【实现细节】

你不需要，也不应该实现主函数，你只需要实现下列函数：

- `find_price(task_id, N, K, ans)`
- 其中 task_id 表示子任务编号（见限制与约定）。N 表示物品个数，K 的意义为：
  - 若 $K=0$，表示有偶数个物品价值为 1 .
  - 若 $K=1$，表示有奇数个物品价值为 1 .
- 你需要将计算出的物品价格放在数组 ans[] 中，其中 ans[i] 表示编号为 i 的物品的价格。

你可以通过调用如下函数来向交互库发出询问：

- `query(S, nS, T, nT)`
- 这里 $\mathrm{nS}=\mathsf{S}, \mathrm{nT}=\mathsf{T}$, 数组 $S[0...(\mathrm{nS}-1)]$ 和数组 $T[0...(\mathrm{nT}-1)]$ 分别描述两个集合，你需要保证：
  - $\mathrm{nS}, \mathrm{nT} > 0$.
  - $\forall 0 \le i < \mathrm{nS}$, $0 \le S[i] < N$ ；
  - $\forall 0 \le i < \mathrm{nT}$, $0 \le T[i] < N$ ；

  - $\forall 0 \le i < j < \mathrm{nS}$, $S[i] \neq S[j]$ ；
  - $\forall 0 \le i < j < \mathrm{nT}$, $T[i] \neq T[j]$ 。

  $\forall$ 的意思是：“对于任意的”。例如：$\forall 0 \le i < \mathrm{nS}$, $0 \le S[i] < N$ 的意思是：“对于任意的在 $[0, nS)$ 内的 i ， $S[i]$ 在 $[0, N)$ 内”。
- 调用此函数一次的时间复杂度为 $\Theta(\mathrm{nS} + \mathrm{nT})$ 。它的返回值为 0 或 1，返回值的意义为：
  - 若集合 S 的物品价格和更大，返回 0 ;
  - 若集合 T 的物品价格和更大，返回 1 ;
  - 否则，按照某种未知规则返回 0 或 1 .
- 如题面所述，我们定义这样一次调用的代价为 $\mathrm{nS} + \mathrm{nT}$ 。

评测时，交互库可能会调用 `find_price` 多次（不超过 10 次），每次调用代表一次新的猜价格游戏，所有的物品的价格都会被重新设定。

【实现方法】

选手工作目录下已经提供了一个 `sample_shop.cpp/c/pas` ，我们推荐你在其基础上进行答题，但这不是必要的。

对 C++ / C 语言选手

你需要在你的程序开头加上 `#include "shop.h"` 。

你需要实现的函数 `find_price` ：

```
void find_price(int task_id, int N, int K, int ans[]);
```

函数 `query` 的接口如下：

```
int query(int S[], int nS, int T[], int nT);
```

对 Pascal 语言选手

你需要包含单元 `graderhelperlib` 。

你需要实现的函数 `find_price`：

```
procedure find_price(task_id, N, K: longint; var ans: array[0..N-1] of longint);
```

函数 `query` 的接口如下：

```
function query(var S: array of longint; nS: longint; var T: array of longint; nT: longint): longint;
```

【如何评测你的程序】

对 C/C++ 语言的选手：

你需要在本题目录下使用如下命令编译得到可执行程序：

对于 C 语言：

```
gcc grader.c shop.c -o shop -O2
```

对于 C++ 语言：

```
g++ grader.cpp shop.cpp -o shop -O2
```

对于 Pascal 语言的选手：
你需要在本题目录下使用如下命令编译得到可执行程序：

```
fpc grader.pas -o"shop" -O2
```

可执行文件将从**标准输入**读入以下格式的数据：
第一行一个整数 T（$T \le 100$），表示数据组数。对每组数据：

- 第一行包含两个整数 $task\_id, N$；
- 接下来一行一个长度为 N 的 01 串 s，其中 s_i 表示物品 i 的价格。你需要保证至少有一个物品价格为 1 。

读入完成之后，交互库将调用 T 次 `find_price` 函数。
接下来交互库会判断你的函数每次计算是否正确，若全部正确则会输出"Correct"，否则会输出相应的错误信息。

【评分标准】

最终评测时只会收取 shop.cpp/c/pas，修改选手目录中的其他文件对评测无效。
题目首先会受到和非交互式程序题相同的限制。例如编译错误会导致整道题目得 0 分，运行时错误、超过时间限制、超过空间限制等会导致相应测试点得 0 分等。你只能访问自己定义的和交互库给出的变量及其对应的内存空间，尝试访问其他空间将可能导致编译错误或运行错误。
在上述条件基础上，在一个测试点中，你得到满分，当且仅当：

- **每一次** `find_price` 的调用中：
- 你的每一个 `query` 的调用符合规则，且调用代价之和不超过 M；
- 你的函数正确计算了 ans[] 数组。

在一个测试点中，如果你没有获得满分，那么你获得 0 分。

【子任务】

我们令代价之和的上界为 M，记答案数组为 ans[]：

- 子任务 1：$N \le 5$，$M = 100$。
- 子任务 2：$N \le 10^3$，$M = 10^6$。
- 子任务 3：$N \le 10^5$，$M = 100$，保证 $\forall i < j < k$，若 $\text{ans}[i] = \text{ans}[k]$ 则必有 $\text{ans}[j] = \text{ans}[i]$。

- 子任务 4：$N \le 10^4$，$M = 2 \times 10^5$ 。
- 子任务 5：$N \le 5 \times 10^4$，$M = 350100$。
- 子任务 6：$N \le 10^5$，$M = 500100$ 。

【提示】

I 君可能并不愿意让 V 君知道每件物品的价格，在物品价格相等时，他会按照他自己的某种方式来回答问题。

【子任务分值】

在本场比赛中，测试点（或子任务）的分值分布与你是否为集训队选手有关。本题的分值设置如下：

子任务编号	1	2	3	4	5	6
集训队分值	20	11	9	12	17	31
非集训队分值	31	21	13	9	11	15