最后的晚餐

李奥纳多在创作他最著名的壁画“最后的晚餐”时非常活跃。他的一项日常工作是决定使用哪种色彩作画。他需要使用很多颜色，但是只能在他的画架上保持一定数量的色彩。他的助手的一项任务是爬上画架把所需要的颜色递给他，然后再把颜色放回到地面。

在画架和地面之间传递颜色

我们考虑一种经过简化的情景。假设有 N 种颜色，编号从 0 到 N-1，并且每一天李奥纳多要求助手送一种新颜色正好 N 次。令 C 为李奥纳多需要这 N 种颜色的次序。我们可以把 C 看成由 0 到 N-1 这 N 个数组成的序列。注意，有些颜色可能在 C 中不出现，有些颜色可能出现多次。

画架总是摆满了 N 种颜色中的 K 种，并且 K<N。刚开始时画架上颜色编号为从 0 到 K-1。

助手每次执行李奥纳多的一条命令。当所需的颜色已在画架上时，助手就可以休息，否则他必须从地面上找到所需的颜色，并且把它放到画架上。当然，画架上没有多余的空位置放置新的颜色，因此，助手必须从画架上挑选一种颜色并把它拿回到地面。

李奥纳多的最佳策略

助手希望尽量休息。他能够休息的次数取决于他干活时的选择。更为准确地说，每次助手必须从画架上取走一种颜色，不同的选择可能导致后面出现不同的结果。李奥纳多告诉他当知道 C 时如何实现他的目标：当所需的颜色不在画架上时，最好的选择可以通过检验当前在画架上的颜色以及 C 中其余的颜色请求来实现。被拿下的颜色应该是此后再也不需要，或者其需要的时间晚于任何其他颜色。

- 如果画架上有一种颜色在后面绝不会需要，助手将从画架上取走这种颜色。
- 否则，从画架上取走的颜色应当是，其下一次需要在未来最晚。

例 1

令 N=4，即我们有四种颜色（编号 0-3）以及 4 项请求。令请求序列为 C = (2, 0, 3, 0)。同时假设 K=2。即李奥纳多的画架在任何时候只能放两种颜色。如上所述，画架上的初始颜色编号为 0 和 1。我们把画架上的颜色表示为[0，1]。助手处理这些请求的一种可能的方法如下：

第一个请求的颜色编号为 2，不在画架上。助手把它放上画架并取下 1 号颜色，此时画架上颜色状态为[0，2]。

第二个请求的颜色编号为 0，已在画架上，因此助手可以休息。

第三个请求的颜色编号为 3，助手取下 0 号颜色，此时，画架上的颜色状态变为[3，2]。

最后请求的颜色编号为 0，助手决定取下 2 号颜色，因此，画架颜色状态变为[3，0]。

注意在上面的例子中，助手并没有遵循李奥纳多的最优策略。最优策略需要在第三步拿下

2 号颜色，这样，就能在最后一步休息。

当内存有限时助手的策略

早上，助手要求李奥纳多在纸上写出 C，以便他能够找出并遵循最优策略。然而，李敖纳多希望保持他工作技术的秘密，因此，拒绝让他的助手把纸拿走。他只允许助手看到 C，并且尽量记住它。

然而，助手的记性很差，他只能记住 M 个二进制位。一般说来，这可能会防止他重建整个序列 C。这样，助手必须找到聪明的方法来计算他需要记住的二进制位序列。我们把这一序列称为提示序列，并记为 A。

例 2

早上，助手可以拿到李奥纳多写在纸上的序列 C，阅读这一序列，做出所有可能的选择。他可以选择要做的一件事情是写下每一次请求之后的画架状态。例如，当使用例 1 中给出的次优策略时画架状态是 [0, 2], [0, 2], [3, 2], [3, 0]。（因为他知道画架的初始状态为 [0，1]，所以不需要把它写下来）。

我们假设 M=16，因此，助手只能记住 16 个二进制位的信息。由于 N=4，我们可以把每种颜色用两个二进制位表示。因此，16 bits 足够保存上述画架状态。这样，助手算出下列提示序列： A = (0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0)。

此后，助手可以解读这一提示序列，即使在事先不知道序列 C 时也可以用它来生成自己的选择。

（当然，当 M=16 时，助手也可以选择只使用 16bits 中的 8 位来记住整个序列 C。在这个例子中，我们只想说明还有其他的选择而没有给出任何好的方案）

声明

你必须用同一种编程语言写出两个独立的程序。这两个程序将顺序执行，并且在执行过程中不能互相通信。

第一个程序是助手早上使用的，这个程序读入序列 C，并计算出提示序列 A。

第二个程序是助手在工作时使用的。这个程序接收提示序列 A，然后处理李奥纳多的请求序列 C。注意，这个程序每次只能知道序列 C 中的一个请求，每个请求必须在获取下一个请求之前处理完毕。

更准确的说，在第一个程序中，你必须实现一个子程序 ComputeAdvice(C, N, K, M)，其中，C 是 N 个整数的数组，每个整数都在 0 到 N-1 之间，数字 K 是画架上颜色的数量，数字 M 是提示序列中的可用的 bits。这一程序必须计算出最多由 M 个 bits 组成的提示序列 A。程序必须调用下述函数将 A 中的每一个 bit 通知系统：

WriteAdvice(B)—— 这一函数将 bit B 加入到提示序列 A 的尾部。（你最多调用这个函数 M 次）

在第二个程序中，你必须实现函数 Assist(A, N, K, R)，这一函数的输入数据是提示序列 A，整数 N 和 K，其含义如上所述，以及提示序列 A 中 bit 的个数 R（R ≤ M）。这一函数，将使用为你提供的下列函数执行你为助手提供的策略：

GetRequest() — 返回李奥纳多所需要的下一种颜色。（不会提供任何关于以后请求的信息）

PutBack(T) — 从画架上取下颜色 T。只有当 T 在画架上时，才可调用此函数。

当程序执行时，你的函数 Assist 必须调用 GetRequest 正好 N 次，每次按顺序获取李奥纳多的一个请求。在每次调用 GetRequest 之后，如果返回的颜色不在画架上你必须使用你选择的 T 调用 PutBack(T)。否则，你不得调用 PutBack。不遵守上述要求，将被视为错误，并停止你的程序的运行。请注意，在开始时地面上包含 0 到 K-1（含）种颜色。

对于给定的测试用例，如果你的两个函数遵循上述所有的限制，并且调用 PutBack 的次数严格等于李奥纳多的最优策略，则认为成功解决了该问题。注意，如果有多个策略调用 PutBack 的次数相同，你的程序可以执行其中的任何一个（即：如果有同样好的策略，并不要求你一定要遵循李奥纳多的策略）

例 3

继续例 2，假设，在 ComputeAdvice 中，你计算出的 A = (0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0)。

为与系统通信，你需要执行下述调用序列。WriteAdvice(0), WriteAdvice(0), WriteAdvice(1), WriteAdvice(0), WriteAdvice(0), WriteAdvice(0), WriteAdvice(1), WriteAdvice(0), WriteAdvice(0), WriteAdvice(1), WriteAdvice(1), WriteAdvice(1), WriteAdvice(0), WriteAdvice(0), WriteAdvice(1), WriteAdvice(1), WriteAdvice(0), WriteAdvice(0)。

你的第二个函数 Assist 将被执行，其参数为上述序列 A，其他参数为 N = 4, K = 2，R = 16。函数 Assist 必须执行正好 N = 4 次对 GetRequest 的调用。同时，在处理了一些

请求之后，Assist 将需要使用适当的 T 调用 PutBack(T) 。

下述表格给出了例 1 中次优策略所对应的调用序列。横线表示没有调用 PutBack。

GetRequest()	Action
2	PutBack(1)
0	-
3	PutBack(0)
0	PutBack(2)

Subtask 1 [8 points]

N ≤ 5 000.

最多使用 M = 65 000 bits.

Subtask 2 [9 points]

N ≤ 100 000.

最多使用 M = 2 000 000 bits.

Subtask 3 [9 points]

N ≤ 100 000.
K ≤ 25 000.

最多使用 M = 1 500 000 bits.

Subtask 4 [35 points]

N ≤ 5 000.

最多使用 M = 10 000 bits.

Subtask 5 [up to 39 points]

N ≤ 100 000.
K ≤ 25 000.

最多使用 M = 1 800 000 bits.

这一子任务的得分取决于你的程序生成的提示序列长度 R。更准确地说，如果 Rmax 是你的函数 ComputeAdvice 所生成的提示序列中的最大长度（超过所有的测试用例），你的得分如下：

39 points if Rmax ≤ 200 000;
39 (1 800 000 - Rmax) / 1 600 000 points if 200 000 < Rmax < 1 800 000;
0 points if Rmax ≥ 1 800 000.

实现细节

你必须提供用同一种编程语言写的两个文件

第一个文件名为 advisor.c, advisor.cpp 或 advisor.pas。这个文件必须实现函数 ComputeAdvice 并且能够调用函数 WriteAdvice。第二个文件名为 assistant.c, assistant.cpp 或 assistant.pas。这个文件必须实现函数 Assist，并且可以调用函数 GetRequest 和 PutBack。

所有函数的原型如下：

C/C++ programs

```
void ComputeAdvice(int *C, int N, int K, int M);
void WriteAdvice(unsigned char a);

void Assist(unsigned char *A, int N, int K, int R);
void PutBack(int T);
int GetRequest();
```

Pascal programs

```
procedure ComputeAdvice(var C : array of LongInt; N, K, M : LongInt);
procedure WriteAdvice(a : Boolean);

procedure Assist(var A : array of Byte; N, K, R : LongInt);
```

```
procedure PutBack(T : LongInt);
function GetRequest : LongInt;
```

这些函数的功能如前所述。当然你可以选择实现在这些函数内部使用的其它函数。对于 C/C++ 程序你的函数必须声明为 static，因为样例评分器将把它们连接在一起。你的程序不得以任何方式与标准输入/输出或任何其他文件交互。

当编程时，你也必须遵循下述要求（在你竞赛环境中的模板已经满足下列的要求）

C/C++ 程序

在你的 advisor 和 assistant 程序的开始必须分别包含文件 advisor.h 和 assistant.h，这需要在你的程序中包含：

```
#include "advisor.h"
```

或

```
#include "assistant.h"
```

文件 advisor.h 和 assistant.h 在你竞赛环境的目录中提供，同时也通过 Web 界面提供。通过相同方式提供的还有测试你程序的代码和脚本。特别是在将你的程序拷入这些脚本所在的目录之后，你必须运行 compile_c.sh 或 compile_cpp.sh（取决于你代码所使用的语言）。

Pascal 程序

你必须在 advisor 和 assistant 中分别使用单元 advisorlib 和 assistantlib：这需要在你的程序中包含：

```
uses advisorlib;
```

或

```
uses assistantlib;
```

文件 advisorlib.pas 和 assistantlib.pas 在你竞赛环境的目录中提供，同时也通过 Web 界面提供。通过相同方式提供的还有测试你程序的代码和脚本。特别是在将你的程序拷入这些脚本所在的目录之后，你必须运行 compile_pas.sh。

样例评分器

样例评分器接受如下格式的输入：

```
line 1: N, K, M;
lines 2, ..., N + 1: C[i].
```

评分器首先运行函数 ComputeAdvice。这将生成文件 advice.txt, 其中包含提示序列中的各个由空格分隔的二进制位，结尾是一个 2。

然后它将执行你的 Assist 函数，并生成输出数据，其中，每一行格式或者是 "R [number]", 或者是"P [number]"。第一种格式，表示对 GetRequest() 的调用并返回所收到的内容。第二种格式表示调用 PutBack()，把所选择的颜色放回去。输出由"E"结束。

正式的评分器工作方式不同。特别是，执行时间会显著不同。你最好使用测试界面使你的程序在正式评分器下运行。

时间和内存限制

时间限制: 7 秒

内存限制: 256 MB.