最後晚餐

Leonardo 在創作他最著名的壁畫"最後的晚餐"時非常活躍。他的一項日常工作是決定使用哪種色彩作畫。他需要使用很多顏色，但是只能在他的工作架上保持一定數量的顏色。他的助手的一項任務是爬上工作架把所需要的顏色遞給他，然後再把顏色放回到地面。

在此題中，你必須編寫兩個分開的程式來幫助那位助理。第一個程式將會收到 Leonardo 的指示(Leonardo 當天會用到的的顏色的序列)，並*產生一個簡短*的位元字串，稱之為*提示*。當天助理處理 Leonardo 的要求的當時，將不會同時收到 Leonardo 後面的要求，而只會收到你第一個程式*產*生的提示。第二個程式將收到提示，然後接收到 Leonardo 線上模式的要求(也就是一次一個要求)。這個程式必須了解題式的含意，並用來進行最佳選擇。所有細節將在後面詳述。

在工作架和儲存架之間搬運顏料

讓我們考慮一種簡化的情景。假設有 N 種顏色，編號從 0 到 N-1，並且每一天 Leonardo 要求助手送一種新顏色正好 N 次。假設 C 為李奧納多需要這 N 種顏色的序列。我們可以把 C 看成由 0 到 N-1 這 N 個數組成的序列。注意，有些顏色可能在 C 中不出現，有些顏色可能出現多次。

棚架總是擺滿了 N 種顏色中的 K 種，並且 K<N。剛開始時棚架上顏色編號為從 0 到 K-1。

助手每次執行 Leonardo 的一條命令。當所需的顏色*已在畫架上*時，助手就可以休息。否則，他必須從架上找到所需的顏色，並且把它放到棚架上去。當然，棚架上沒有多餘的空位置放置新的顏色，此時，助手必須從棚架上挑選一種顏色並把它拿回到架上。

Leonardo 的最佳策略

助手希望盡量可以休息。他能夠休息的次數取決於他從棚架上取走顏色時的選擇。Leonardo 告訴他，當知道 C 時如何實現他的目標：當所需的顏色不在畫架上時，最好的選擇是儘量推遲，直至這一情況再次發生。這可以通過檢驗當前在畫架上的顏色以及 C 中其餘的顏色請求來實現。選擇從棚架取走顏色時應根據以下一些規則:

- 如果一隻在棚架上的顏色在將來再不會被使用, 則助手應該把這隻顏色取走。
- 否則，從工作架上移除的顏色應該是*最遠的未來才會再用到的*。(也就是說，比工作架上的其它顏色都更晚再被用到)。

已被證明的是，當使用 Leonardo 的策略時，助理將能休息最多次。

樣例 1

假設 N = 4，我們有 4 隻顏色 (編號 0 至 3) 和 4 個要求。假設要求的序列為 C = (2, 0, 3, 0)，並假設 K = 2，也就是工作架上能容納兩個顏色。如上所述，工作架上一開始的顏色是 0 和 1。我們把工作架上的顏色寫成: [0, 1]。助理處理要求的一個可能方式如下。

- 第一個要求的顏色(編號2)不在工作架上，助理把它拿上去且決定從工作架拿下顏色1。現在工作架是 [0, 2]。

下一個要求的顏色(編號0)已在工作架上，所以助理可以休息。

- 第三個要求(編號3)，助理拿下顏色0，工作架變成 [3, 2]。
- 最後，要求的顏色(編號0)要從儲存架搬到工作架上。助理決定拿下顏色2，現在工作架變成 [3, 0]。

注意上面的範例裡，助理沒有聽從 Leonardon 的最佳策略。最佳策略在第三步應該會拿下顏色2，這樣助理在最後一步就可以再休息了。

助理在記憶有限時的策略

早上助理請 Leonrado 把 C 寫在一張紙上，這樣他才能找出並遵循最佳策略。然而李奧納多很小心保守他工作技巧的秘密，因此他拒絕把這張紙交給助理，他只允許助理看到 C 並試著記住它。

不巧的是，助理的記性很差，他最多只能記住 M 位元。通常這會導致他無法回想出整個 C 的序列。因此助理必須想出個聰明的辦法來算出他要記住的位元。他將這個序列稱為"提示序列" (advice sequence)，且我們用 A 來代表它。

樣例 2

每天早上助理向 Leonrado 拿寫有 C 的紙張，讀取序列，並做好必要的選擇。他可以選擇做的一件事，是檢視每次要求之後工作架的狀態。例如，當使用範例一的(非最佳)策略時，工作架的狀態是[0, 2], [0, 2], [3, 2], [3, 0]。(要記得他知道工作架的起始狀態是[0, 1])

現在假設 M = 16，所以助理最多只能記住 16 位元的資訊。由於 N = 4, 我們用 2 位元儲存每個顏色。因此 16 位元足以儲存上述工作架狀態的序列。因此助理算出下列的提示序列: A = (0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0)

之後，助理就能將這段提示序列解碼，並用來作出選擇。

(當然，當 M = 16 時，助理也可以選擇記下整段 C 序列，且僅用到 16 位元中的 8 位元。我們只是想藉此範例展示他有其它的方案，而不至於洩漏解答。)

說明

你必須用相同的程式語言寫*兩個分開的程式*。這兩個程式將會依序被執行，且於執行時不能彼此溝通。

第一個程式是助理早上用的。這個程式將會收到 C 序列，並且算出提示序列 A。

第二個程式是助理當天後來用的。這個程式將會收到提示序列 A，然後它必須處理李奧納多要求的序列 C。請注意這個程式一次只會看到序列 C 的一筆要求，且每筆要求必須先處理完，才會收到下一筆要求。

具體而言，在第一個程式中你必須實作一個程序 ComputeAdvice(C, N, K, M)，它的輸入陣列 C 由 N 個整數構成(每個落在0, …, N - 1)，工作架上有 K 個顏色，以及可用做提示的 M 位元。這個程式必須算出提示序列 A，且長度最多為 M 位元。這個程式然後必須依照 A 每個位元的順序逐一呼叫下列程序，將提示序列 A 傳送給系統:

- WriteAdvice(B) — 將位元 B 加到現有提示序列 A 的尾端。(你最多可呼叫此程序 M 次)

在第二個程式你必須實作一個程序 `Assist`(A, N, K, R)。程序的輸入是提示序列 A、整數 N 和 K 定義同上、以及提示序列 A 的實際長度 R 位元 (R ≤ M)。這個程序應該執行你為助理提出的策略，利用下列提供給你的程序:

- `GetRequest`() — 回傳下一個 L要求的顏色。(不會透漏未來的要求)

`PutBack`(T) — 把顏色 T 從工作架放回儲存架。呼叫此程序時，顏色 T 必須是目前工作架上有的顏色。

在運行時， 你的程式 `Assist` 一定要乎叫 `GetRequest` 正好 N 次, 每一次順序接收 Leonardo 的一個要求。在每次呼叫完 `GetRequest`, 若該返回的顏色並*不*是在工作架上, 你就*必須*同時再以你選定的 T 呼叫 `PutBack`(T)。否則你*必不要*呼 叫 `PutBack`。未能做到這一點的會被當為程式錯誤, 你的程式將被停止。請記著在開始時, 工作架上是有顏色 0 至 K-1 的。

當你的兩個程式跟足所有的條件, 並且呼叫 `PutBack` 的次數是*正好等於* Leonardo 的最佳策略時， 你的程式就算是通過了該測試數據了。請注意當有多種方法可以實現相同呼叫 `PutBack` 數目時, 你可以採用其中任何一種。(即當若有其他相同好的策略時, 你是不一定要跟從 Leonardo 策略的)。

樣例 3

讓我們繼續例 2, 假設在 `ComputeAdvice` 中你計算 A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`)。 為了要將它和系統溝通, 你必要作出以下一系列的呼叫: `WriteAdvice`(`0`), `WriteAdvice`(`0`) , `WriteAdvice`(`1`) , `WriteAdvice`(`0`) , `WriteAdvice`(`0`), `WriteAdvice`(`0`) , `WriteAdvice`(`1`) , `WriteAdvice`(`0`) , `WriteAdvice`(`1`), `WriteAdvice`(`1`) , `WriteAdvice`(`1`) , `WriteAdvice`(`0`) , `WriteAdvice`(`1`), `WriteAdvice`(`1`), `WriteAdvice`(`0`), `WriteAdvice`(`0`).

你的第二個子程式 `Assist` 會跟著執行, 它接收了上述的序列 A, 及相關值 N = 4, K = 2, 和 R = 16。該 `Assist` 必要執行正好 N=4 次呼叫子程式 `GetRequest`。同時, 經若干次呼叫後, `Assist` 必要以合適的 T 來呼叫 `PutBack`(T)。

下表呈現對應於範例一的 (非最佳) 選擇的一序列的呼叫。連字符號 (-) 代表沒有呼叫

`PutBack` 。

`GetRequest()`	動作
2	`PutBack(1)`
0	-
3	`PutBack(0)`
0	`PutBack(2)`

子任務 1 [8 分]

- $N \leq 5\,000$.
- 你最多可以用 M = 65 000 bits.

子任務 2 [9 分]

- $N \leq 100\,000$.
- 你可以使用最多 M = 2 000 000 bits.

子任務 3 [9 分]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- 你可以使用最多 M = 1 500 000 bits.

子任務 4 [35 分]

- $N \leq 5\,000$.
- 你最多可以使用 M = 10 000 bits.

子任務 5 [最多有 39 分]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- 你最多可以用 M = 1 800 000 bits.

這個子任務的得分與你的程式所傳遞提示的長度 R 有關。更仔細來說，如果 R_{max} 是你的 `ComputeAdvice` 程序所產生提示序列長度的最大值 (於全部測資裡)， 你的得分將是 :

- 可得 39 分若 $R_{max} \leq 200\,000$;

- 可得 39 (1 800 000 - R_{max}) / 1 600 000 分若 200 000 < R_{max} < 1 800 000;
- 0 分若 $R_{max} \geq 1\ 800\ 000$.

編程實現細則

你應該要上傳正好兩個檔案, 並且兩個檔案必定要再同一種程式語言。

第一個檔案名稱叫 `advisor.c`, `advisor.cpp` 或 `advisor.pas`。 這個檔案必須實現如前所述的 `ComputeAdvice` 而且可以呼叫子程式 `WriteAdvice`。第二個檔案名叫 `assistant.c`, `assistant.cpp` 或 `assistant.pas` 這個檔案必須實現如前所述的副程式 `Assist` 而且可以呼叫副程式 `GetRequest` 以及 `PutBack`.

所有函式原型如下。

C/C++ 程

```
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 程式

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

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

這些程序必須表現得如前所述。當然你可以實作內部使用的程序。若是 C/C++ 的程式, 你的內部程序應該宣告為 `static`, 讓範例評分系統把他們連結在一起。或者避免在兩個程式中使用同名的程序。你上傳的程式不能與標準輸入/輸出或其他檔案進行互動。

當你寫程式的時候, 你必須注意下面的指示 (在你的比賽環境中的 templates 已經滿足下列的要求)

C/C++ 程式

在你的解答程式一開始, advisor 以及 assistant 必須分別包含 檔 案`advisor.h` 與 `assistant.h` 。這可以藉由在你的程式碼檔案中加入這一行來完成 :

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

或

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

系統會在你的目錄及網頁介面中提供兩個檔案 advisor.h 與 assistant.h。經由同樣途徑可以取得程式碼以及 script 來編譯及測試你的程式。具體而言，當你將你的答案抄到那些含有 script 的目錄後，你必須執行 compile_c.sh 或compile_cpp.sh 編譯程式 (按照你所選定的語言)

Pascal 程式

你必須分別在 advisor 及 assistant 中使用單元 advisorlib 以及 assistantlib。你可以藉由在原始碼中加入以下這數行程式:

```
uses advisorlib;
```

或

```
uses assistantlib;
```

系統會在你的目錄及網頁介面中提供 advisorlib.pas 以及 assistantlib.pas 這兩個檔案。同時亦會經由同樣途徑為你提供程式碼以及 script 來編譯及測試你的程式。具體而言，當你將你的答案抄到那些含有 script 的目錄後, 你必須要執行 compile_pas.sh。

樣例 grader

樣例 grader 將會接受以下格式的輸入：

- 第 1 行: N, K, M;
- 第 2, …, N + 1 行: C[i].

grader 會先執行 ComputeAdvice。它會生成一個檔案 advice.txt, 其內包括.....

之後系統會執行你的Assist程序並產生結果，結果的每一行是下面兩種格式之一 ： "R [number]" 或是 "P [number]"。第一種類的行代表呼叫到 GetRequest() 及其收到的回復。第二種類的行代表呼吸 PutBack()及其要放回的顏色選擇。產出結果會由一行 "E" 結束。

請注意正式的評分系統執行你程式的時間可能會和在你電腦上的時間有些微的不同。但這差異不會太大，你可以使用測試介面確認你的程式是否在時間限制內完成。

時間及記憶體限制

- 時間限制: 7 秒.
- 記憶體限制: 256 MiB.