最後的晚餐

李奧納多在創作他最有名的壁畫「最後的晚餐」的當時是非常活躍的: 他每天上工時第一件事，是決定當天接下來要用的顏料顏色。他要用到很多顏色，但是他的工作架只能容納有限個顏色。他的助理負責的工作之一，就是爬上工作架將顏料拿給他，以及爬下來把顏料放回地上的儲存架。

此題中，你必須寫兩個分開的程式來幫助助理。第一個程式將會收到李奧納多的指示(李奧納多當天會用到的一序列的顏色)，並產生一個"簡短"的位元字串，稱之為"提示"。當天助理處理李奧納多的要求的當時，將不會同時收到李奧納多後面的要求，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，然後接收到李奧納多線上模式的要求(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

我們先考慮一個簡化的情形。假設有 N 個顏色，編號 0 到 N-1，且每天 李奧納多 跟助理要 N 次新的顏料。讓 C 表示 李奧納多 要求那 N 次顏料的序列，我們可以把 C 想成 N 個數字的序列，每個數字介於 0 到 N-1。請注意有些顏色可能不會出現在 C 裡面，而有些顏色可能出現許多次。

工作架上永遠是滿的，且包含 N 個顏色中的 K 個顏色，且$K < N$。一開始時，工作架上的顏色是 0 至 K-1。

助理一次處理 李奧納多 的一項要求。當所要求的顏色已經在工作架上時，助理就可以休息，否則他就要從儲存架上拿起要求的顏色，再搬到工作架上。此時工作架上並沒有位置可以放新的顏色，因此助理必須選出工作架上的一個顏色，並把它放回儲存架。

李奧納多的最佳策略

助理希望能盡量多休息。他能休息幾次和他搬運過程中所選的顏色有關。更明確的說，每次助理必須從工作架移除一個顏色時，不同的選擇會導致未來不同的結果。李奧納多向助理解說，在知道 C 的情況下如何達到助理的目的: 從工作架上移除顏色的最佳選擇，可以藉由檢視工作架現有的顏色，和 C 裡面剩下的顏色要求來得知。工作架上的顏色應由下列規則選出一個顏色:

- 如果工作架上有個顏色是未來不再需要的，助理應該從工作架上移除這個顏色。
- 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，當使用李奧納多的策略時，助理將能休息最多次。

"範例一"

假設 N = 4，所以我們有 4 個顏色 (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，助理沒有聽從李奧納多的最佳策略。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。

"助理記憶有限時的策略"

早上助理請李奧納多把 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。

"範例二"

早上助理向李奧納多拿寫有 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，並用來做選擇。

(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。)

聲明

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`() — 回傳下一個李奧納多要求的顏色。(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。

執行時，你的 `Assist` 程序必須呼叫 `GetRequest` 剛好 N 次，每次依序接收李奧納多的一個要求。

如果你的兩個程序遵守所有的限制，且呼叫 `PutBack` 的總次數和李奧納多的最佳策略*剛好相等*，則這筆測資可視為已解，。請注意如果有好幾個策略可以達到相同的呼叫 `PutBack` 次數，你的程式可以採用任何一個。(也就是: 如果有一樣好的策略，你不必一定要依循李奧納多的策略。)

"範例三"

延續範例二，假設在`ComputeAdvice` 你算出 A = (`0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0`). 為了把它傳給系統，你會做下列的呼叫: `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)`.

你的第二個程序 `Assist` 會接著執行，接收上述的序列 A，以及數值 N = 4, K = 2, 和 R = 16。 程序 `Assist` 接著必須呼叫 `GetRequest` 剛好 N = 4 次。 還有，在幾個要求之後， `Assist` 必須用適當的 T 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()	行動
2	PutBack(1)
0	-
3	PutBack(0)
0	PutBack(2)

子任務一 [8 分]

- $N \leq 5{,}000$.
- 你最多可使用 M = 65,000 位元。

子任務二 [9 分]

- $N \leq 100{,}000$.
- 你最多可使用 M = 2,000,000 位元。

子任務三 [9 分]

- $N \leq 100{,}000$.
- $K \leq 25{,}000$.
- 你最多可使用 M = 1,500,000 位元。

子任務四 [35 分]

- $N \leq 5{,}000$.
- 你最多可使用 M = 10,000 位元。

子任務五 [39 分]

- $N \leq 100{,}000$.
- $K \leq 25{,}000$.
- 你最多可使用 M = 1,800,000 位元。

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:

- 39 分 if $R_{max} \leq 200{,}000$;
- 39 $(1{,}800{,}000 - R_{max}) / 1{,}600{,}000$ 分 if $200{,}000 < R_{max} < 1{,}800{,}000$;
- 0 分 if $R_{max} \geq 1{,}800{,}000$.

實作細節

你應該要上傳*同一個程式語言*的兩個檔案

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

所有函式原型如下.

C/C++ 程式

```
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 已經滿足下列的須求).

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"
```

兩個檔案advisor.h 與 assistant.h 在比賽的環境的一個目錄會提供給你，你也可以由比賽的網頁中獲得。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，在使用 script 將你的解答拷貝到這個目錄之後，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

在 advisor 以及 assistant 你必須分別使用程式單元 advisorlib 和 assistantlib, 這可藉由在原始碼中 include 這行:

```
uses advisorlib;
```

或

```
uses assistantlib;
```

兩個檔案 advisorlib.pas 以及assistantlib.pas 會在比賽環境中的一個目錄中提供，你也可以從比賽的網頁中下取用。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，在使用 script 將你的解答拷貝到這個目錄之後，你必須執行compile_pas.sh編譯程式 (按照你所選定的語言)

範例評分系統

範例評分系統從符合下面格式的資料進行輸入

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

評分系統會先執行程序 ComputeAdvice，產生一個 advice.txt 檔案，此檔案包含提示序列的每個位元，用空白隔開並以 2 結尾。

之後會執行你寫的 Assist 程序產生結果，結果的每一行是下面兩種格式之一: "R [number]" 或 "P [number]"。第一種類型的格式表示呼叫 GetRequest() 程序且收到回覆。第二種類型的格式表示呼叫 PutBack() 程序，並且被放回的顏色。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。

時間和記憶體限制

- 時間限制: 7 秒.
- 記憶體限制: 256 MiB.