Última Cena

Leonardo era muy activo cuando trabajaba en la Última Cena, su pintura mural más famosa: una de sus primeras tareas diarias era la de decidir qué colores de óleos usar durante el resto de la jornada de trabajo. Necesitaba muchos colores pero sólo podía mantener un número limitado de ellos en su andamio. Su ayudante era el encargado, entre otras cosas, de subir al andamio para ofrecerle colores y luego bajar para ponerlos de nuevo en el estante adecuado en el suelo.

En esta tarea, deberás escribir dos programas para ayudar al asistente. El primer program recibirá las instrucciones de Leonardo (una secuencia de colores que Leonardo necesitará durante el día),y crear una *corta* cadena de bits, llamada *advice*. Mientras procesas el pedido de Leonardo durante el día, el asistente no tendrá aceso a los requerimientos futuros, sólo al advice producido por su primer programa. El segundo programa recibirá el advice, y entonces recibe y procesa los pedidos de Leonardo en un modo online (i.e., uno por vez). Este programa debe ser capaz de entender lo que el advice significa y usarlo para efectuar elecciones óptimas. Todo los aspectos son descriptos abajo con mayor detalle.

Moviendo entre el estante y el andamio

Vamos a considerar un escenario simplificado. Supongamos que hay N colores numerados de 0 a N - 1, y que cada día Leonardo le pide al asistente exactamente N veces un nuevo color. Sea C la secuencia de los N colores pedidos por Leonardo. Entonces, podemos pensar en C, como una secuencia de N números, cada uno estará entre 0 y N - 1, ambos inclusive. Tenga en cuenta que algunos colores podrían no presentarse nunca en C, y otros pueden aparecer varias veces.

El andamio está siempre lleno y contiene K de los N colores, con K <N. Inicialmente, el andamio contiene los colores de 0 a K - 1, ambos inclusive.

El asistente procesa los pedidos de Leonardo uno por vez. Cada vez que el color solicitado está *ya en el andamio*, el asistente puede descansar. De lo contrario, tiene que agarrar el color solicitado de la estantería y moverlo al andamio. Por supuesto, no hay lugar en el andamio para el nuevo color, por lo que el asistente debe elegir uno de los colores en el andamio y lo toma de la parte posterior de andamio para ponerlo en el estante adecuado en el suelo.

La estrategia optima de Leonardo

El asistente quiere descansar tanto como sea posible. El número de pedidos en los cuales puede descansar depende de como elija los colores durante el proceso. Más precisamente, cada vez que el asistente tenga que remover un color del andamio, diferentes elecciones pueden traducirse en diferentes resultados en el futuro. Leonardo le explica como puede lograr su objetivo conociendo C. La mejor elección para el color a ser removido del andamio se obtiene examinando los colores

actualmente en el andamio y los restantes requerimientos de colores en C. Un color debería ser elegido de entre los que están en el andamio de acuerdo a las siguientes reglas:

- Si hay un color en el andamio que nunca será usado en el futuro, el asistente remueve ese color del andamio.
- De otro modo, el color removido en el andamio será aquel que *será requerido más alejado en el futuro*. (Eso es, por cada color en el andamio buscamos su próxima ocurrencia. El color retornado al estante es aquél que será requerido último.)

Puede ser demostrado que usando la estrategia de Leonardo, el asistente descansará tantas veces como sea posible.

Ejemplo 1

Sea N = 4, entonces tenemos 4 colores (numerados de 0 to 3) y 4 requeridos. Sea la secuencia de pedidos C = (2, 0, 3, 0). Asumiendo ademas que, K = 2. Es decir, Leonardo tiene un andamio que puede tener dos colores al mismo tiempo. Como fue descripto anteriormente, el andamio tiene inicialmente los colores 0 and 1. Escribiremos el contenido del andamio de la siguiente manera [0, 1]. Una forma posible en que el asistente puede manejar los pedidos es la qque sigue.

- El primer color pedido (número 2) no está en el andamio. El asistente lo pone en el andamio y decide retirar el color (número 1) . El estado del andamio queda [0, 2].

- El siguiente color pedido es (número 0), este color está en el andamio, luego el asistente descansa.

- Para el tercer color pedido (número 3), el asistente retira el color (número 0), cambiando el estado del andamio a [3, 2].

- Finalmente, el último pedido color (número 0) debe ser llevado del estante al andamio . El asistente decide retirar el (color 2), y el andamio ahora queda [3, 0].

Note que en el ejemplo anterior, el asistente no siguio la estrategia óptima de Leonardo. La estrategia óptima hubiera retirado el color número 2 en el tercer paso, asi el asistente podría descansar nuevamente en el paso final..

Estrategia del Asistente cuando su memoria está limitada

En la mañana, el asistente le pidió a Leonardo que le escriba C en una hoja de papel, para de esa manera poder encontrar y seguir la estrategiaa óptima. Sin embargo Leonardo es obsesivo en mantener sus trabajos técnicos en secreto , por lo que se negó a darle a su asistente el papel. El sólo le permitió leer C y tratar de recordarlo.

Lamentablememnte, la memoria del asistente es muy mala. El sólo puede recordar hasta M bits. En general, esto puede evitar que el pueda reconstruir completamente la secuencia C. Entonces el asistente se ha inventado una ingeniosa manera de computar la secuencia de bits que el puede recordar. Llamaremos a esta secuencia, "secuencia de ayuda" y la denominaremos A.

Ejemplo 2

En la mañana, el asistente puede ver los papeles de Leonardo con la secuencia C, leer la secuencia

C, y elegir los colores necesarios. Una cosa que el puede hacer es escribir el estado del andamio después de cada uno de los cambios requeridos. Por ejemplo, cuando usa el la estrategia (sub-optimal) dada en el Ejemplo 1, la secuencia de los estados del andamio debería ser [0, 2], [0, 2], [3, 2], [3, 0]. (Recuerda que el estado inicial del andamio es [0, 1]. No es necesario escribir esto)

Ahora asuma que tenemos M = 16, entonces el asistente podrá recordar hasta 16 bits de informacion. Como N = 4, podemos guardar cada color usando 2 bits. por lo que 16 bits son suficientes para guarda la secuencia de estado del andamio. Por lo que el asistente computa la secuencia de ayuda A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`).

Mas tarde en el día, el asistente puede decodificar la secuencia de ayuda y usarla para tomar sus decisiones.

(Por supuesto que con M=16 el asistente puede tambien elegir recordar toda la secuencia C usando solo 8 de los 16 bits. En este ejemplo nosotros quríamos ilustrar que el puede tener otras opciones sin descubrir la buena solución).

Enunciado

Usted debe escribir "dos programas separados" en el mismo lenguaje de prograamación. Estos programas se ejecutarán secuencialmente sin poder comunicarse entre ellos durante la ejcución.

El primer programa será el usado por el asistente por la mañana. A este programa le será dada la secuencia C, y deberá computar la secuencia de ayuda A.

El segundo programa será usado por el asisitente durante el día. Este programa recibirá la secuencia de ayuda A, y deberá procesar la secuencia de pedidos C de Leonardo. Note que la secuencia C sólo será revelada a este programa de a un pedido por vez y cada pedido deberá ser procesado antes de recibir el siguiente.

Mas precisamente, el primer problema debe implementar una rutina simple `ComputeAdvice`(C, N, K, M) teniendo como entrada el arreglo C de N enteros, cada uno entre (0 .... N-1), el numero K de cantidad de colores en el andamio y el número M del numero de bits disponibles para la secuencia de ayuda A que consiste en hasta M bits. El programa debe entonces comunicar esta secuencia A al sistema, llamando para cada bit de A, en orden, la siguiente rutina.

- `WriteAdvice`(B) — añada el bit B a la secuencia de ayuda actual A. (puede llamar a esta rutina no más de M veces.)

En el segundo programa usted debe implementar una rutina `Assist`(A, N, K, R). La entrada de esta rutina, es la secuencia de ayuda A. Los enteros N y K definidos anteriormente, y la actual longitud R de la secuencia A en bits (R ≤ M). Esta rutina debe ejecutar su propia estrategia para el asistente usando las siguientes rutinas que le fueron provistas.

- `GetRequest`() — retorna el siguiente color pedido por Leonardo. . (No hay información sobre los futuros pedidos.)

- `PutBack`(T) — pone el color T del andamio nuevamente en el estante . Usted puede llamar a esta rutina con T, sólo si T es un color que está en el andamio.

Cuando se ejecuta la rutina `Assist` debe llamar `GetRequest` exactamente N veces, cada vez recibirá un pedido de Leonardo . Después de cada llamada a `GetRequest`, si el color no está en el andamio, usted debe llamar a `PutBack`(T) con su elección de T. Caso contrario, usted no debe llamar a `PutBack`. No cumplir con esto será considerado un error y dará por terminado su programa. Por favor recuerde que al principio el andamio los colores de 0 to K - 1, inclusive.

Un particular caso de test será considerado resuelto si sus dos rutinas siguen las restricciones descriptas y en total el número de llamadas a `PutBack` es exactamente igual al número de llamadas de la estrategia óptima de Leonardo. Note que si hay mas de una estrategia que logre el mismo número de llamadas a `PutBack`, su programa puede realizar cualquiera de ellas (es decir que no es obligatorio seguir la estrategia de Leonardo si hay otra estrategia igual de buena.)

Ejemplo 3

Continuando el Ejemplo 2, asuma en `ComputeAdvice` usted computó A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`). Para comunicarse con el sistema, usted debería hacer la siguiente secuencia de llamadas: `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)`.

Su segunda rutina `Assist` sería luego ejecutada, recibiendo la anterior secuencia A, y los valores N = 4, K = 2, y R = 16. La rutina `Assist` tiene que hacer exactamente N = 4 llamadas a `GetRequest`. Ademas después de estos pedidos, `Assist` deberállamar a `PutBack`(T) con un T apropiado.

La siguiente tabla muestra una secuencia de llamadas correspondientes a las elecciones (sub-optimas) del Ejemplo 1. El guión representa que no hubo llamadas a `PutBack`.

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

Subtarea 1 [8 puntos]

- $N \leq 5\,000$.
- Usted puede usar como máximo M = 65 000 bits.

Subtarea 2 [9 puntos]

- $N \leq 100\,000$.
- Usted puede usar como máximo M = 2 000 000 bits.

Subtarea 3 [9 puntos]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Usted puede usar como máximo M = 1 500 000 bits.

Subtarea 4 [35 puntos]

- $N \leq 5\,000$.
- Usted puede usar como máximo M = 10 000 bits.

Subtarea 5 [hasta up 39 puntos]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Usted puesde usar como máximo M = 1 800 000 bits.

El puntaje para esta subtarea, depende de la longitud R de la secuencia de ayuda que su programa comunica, mas precisamente si R_{max} es el máximo (sobre todos los casos de prueba) de la longitud de la secuencia de pruueba que provee la rutina `ComputeAdvice`, su puntaje será:

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

Detalles de Implementación

Usted debe enviar exactamente dos archivos en el mismo lenguaje de programación. viar dos

El primer archivo `advisor.c`, `advisor.cpp` o `advisor.pas`. Este archivo debe implementar la rutina `ComputeAdvice` tal como está descripto más arriba y puede llamar la rutina. `WriteAdvice`. El segundo archivo es llamado `assistant.c`, `assistant.cpp` o `assistant.pas`. Este archivo debe implementar la rutina `Assist` tal como está descripta arriba y puede llamar las rutinas`GetRequest` y `PutBack`.

Los encabezados para todas las rutinas son los siguientes

C/C++ programas

```
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 programas

```
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;
```

Las rutinas deben comportarse tal como fue descripto arriba. Por supuesto eres libre de implementar otras rutinas para uso interno. Para programs en C/C++, sus rutinas internas deben ser declaradas `static`, ya que el evaluador de muestra los vinculará entre sí. Alternativamente, evite tener dos rutinas (uno en cada programa) con el mismo nombre. Su envío no debe interactuar de ningún modo con el input/output estándar, ni con ningún otro archivo.

Cuando programe su solución, también deberá tener cuidado de seguir las instrucciones (las plantillas que se encuentran en su ambiente, ya satisfacen los siguientes requerimientos.)

Programas C/C++

Al comienzo de su solución, debes incluir el archivo `advisor.h` y `assistant.h`, respectivamente, en el advisor y en el asistente. Esto se logra incluyendo en su archivo fuente la línea:

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

o

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

Los dos archivos `advisor.h` y `assistant.h` le serán provistos en un directorio dentro de su ambiente de competencia y serán también ofrecidos por la interfaz Web de la competencia. Le será provisto también (a través de los mismos canales) código y scripts para compilar y verificar su solución. Especificamente, después de copiar su solución en el directorio con estos scripts, deberás correr `compile_c.sh` o `compile_cpp.sh` (dependiendo del lenguaje de su código). `compile_c.sh` o `compile_cpp.sh` (dependiendo del lenguaje de su código).

Programas Pascal

Usted debe usar la unidad `advisorlib` y `assistantlib`, respectivamente en el advisor y en el assistant. Esto lo logras incluyendo en su fuente la línea:

```
uses advisorlib;
```

or

```
uses assistantlib;
```

Los dos archivos `advisorlib.pas` y `assistantlib.pas` le serán provistos en un directorio dentro de su ambiente de competencia y serán también ofrecidos por la interfaz Web de la competencia. Le será provisto también (a través de los mismos canales) código y scripts para compilar y verificar su solución. Especificamente, después de copiar su solución en el directorio con estos scripts, deberás correr `compile_pas.sh`.

Calificador de muestra

El calificador de muestra, aceptará entradas en el siguiente formato:

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

El calificador, primero ejecutará la rutina `ComputeAdvice`. Esto generará un archivo `advice.txt`, conteniendo los bits individuales de la secuencia advice, separados por espacio y terminados por un 2.

Entonces procederá a ejecutar su rutina `Assist`, y generará salida en la cual cada línea será ya sea dela forma "`R [number]`", ya sea de la forma "`P [number]`". Líneas del primer tipo indican llamadas a `GetRequest()` y las respuestas recibidas. Líneas del segundo tipo representan llamadas a `PutBack`() y los colores elegidos para reponer. La salida es terminada por una línea de la forma "`E`".

Por favor note que en el calificador oficial, el tiempo de ejecución puede diferir ligeramente del tiempo en su computadora local. Esta diferencia no debiera ser significativa. Hasta, le invitamos a usar la interfaz de prueba a fin de verificar si su solución corre dentro de los límites de tiempo.

Límites de tiempo y de memoria

- Tiempo limite: 7 segundos.
- Memoria limite: 256 MiB.