La Ultima Cena

Leonardo estuvo realmente ocupado cuando trabajó en La Ultima Cena, su pintura mural más famosa: una de sus primeras tareas diarias era la de decidir cuales colores de témpera utilizar durante el resto del día de trabajo. El necesitaba muchos colores, pero sólo podía tener un número limitado de ellos en su andamio. Su asistente, era el encargado entre otras cosas, de subir al andamio para entregar los colores a Leonardo y luego bajar para colocarlos nuevamente en el estante adecuado en el piso.

En esta tarea, usted va a necesitar escribir dos programas separados para ayudar al asistente. El primer programa recibirá las instrucciones de Leonardo (una secuencia de colores que Leonardo va a requerir durante el día), y crear un *corto* string de bits, llamado *ayuda*. Mientras se procesan las peticiones de Leonardo durante el día, el asistente no tendrá acceso a las peticiones futuras de Leonardo, solo la ayuda producida por su primer programa. El segundo programa recibirá la ayuda y luego recibirá y procesará las peticiones de Leonardo de forma secuencial (es decir, una a la vez). Este programa debe ser capaz de entender el significado de la ayuda y utilizarla para obtener decisiones óptimas. Todo será explicado con mayor detalle en las secciones siguientes.

Moviendo colores entre el estante y el andamio

Consideraremos un escenario simplificado. Suponga que existen N colores numerados desde 0 hasta N-1, y cada día Leonardo le solicita a su asistente un nuevo color exactamente N veces. Sea C la secuencia de los N colores requeridos por Leonardo. Podemos pensar en C como una secuencia de N números, cada uno de los cuales está entre 0 y N-1 inclusive. Note que algunos colores pueden no aparecer en C, y otros pueden aparecer varias veces.

El andamio está inicialmente lleno y contiene K de los N colores, donde $K < N$. Inicialmente, el andamio contiene los colores desde 0 hasta K-1, inclusive.

El asistente procesa las solicitudes de Leonardo una a la vez. Cuando el color requerido está *ubicado en el andamio*, el asistente puede descansar. De otra forma, es necesario que el asistente lleve el nuevo color desde el estante hacia el andamio. Por supuesto, no existe espacio en el andamio para el nuevo color, por ende, el asistente debe elegir uno de los colores en el andamio para llevarlo de nuevo al estante.

La estrategia óptima de Leonardo

El asistente desea descansar tantas veces como le sea posible. El número de solicitudes en las cuales el asistente puede descansar depende de sus elecciones durante el proceso. Más específicamente, cada vez que el asistente debe remover un color del andamio, diferentes elecciones pueden conducir a diferentes resultados en el futuro. Leonardo le ha explicado como lograr ese objetivo al

conocer C. Un color debe ser elegido entre aquellos que se encuentran en el andamio de acuerdo a las siguientes reglas:

- Si existe un color en el andamio que no será necesario en el futuro, el asistente deberá remover dicho color del andamio.
- De cualquier otra forma, el color removido del andamio debe ser uno que *sea el más tardío en ser requerido en el futuro*. (Esto es, para cada uno de los colores en el andamio debemos encontrar la próxima vez que será necesario en el futuro. El color devuelto al estante es aquel que será requerido de último de acuerdo al orden en que son solicitados).

Un color debe ser elegido entre aquellos que están en el andamio, de tal manera que, sea un color que no vuelva a ser solicitado en el futuro o sea solicitado luego de cualquier otro color que esté actualmente en el andamio.

Puede ser probado que cuando se utiliza la estrategia de Leonardo, el asistente puede descansar tantas veces como sea posible.

Ejemplo 1

Con N = 4, tenemos 4 colores (numerados desde 0 hasta 3) y 4 solicitudes. Sea la secuencia de solicitudes C = {2, 0, 3, 0}. También asuma que K = 2. Esto significa, Leonardo tiene un andamio capaz de tener 2 colores en cualquier momento. Como se mencionó anteriormente, el andamio inicialmente contiene los colores 0 y 1. Ahora escribiremos el contenido del andamio como sigue: [0, 1]. Una de las formas posibles para el asistente de manejar las solicitudes es como sigue.

El primer color requerido (número 2) no está en el andamio. El asistente coloca dicho color en el andamio y remueve el color 1 del mismo. El andamio actual es el siguiente [0, 2].

- El siguiente color solicitado (número 0) se encuentra actualmente en el andamio, por lo tanto el asistente puede descansar.

- Para la tercera petición (número 3), el asistente remueve el color 0, cambiando el andamio al estado siguiente [3, 2].

- Finalmente, el último color requerido (número 0) debe ser tomado desde el estante al andamio. El asistente decide remover el color 2, y el andamio queda de la siguiente manera [3, 0].

Note que en el ejemplo anterior el asistente no siguió la estrategia óptima de Leonardo. La estrategia óptima sería remover el color 2 en el tercer paso, con lo cual el asistente puede descansar nuevamente en el paso final.

Estrategia del Asistente cuando su memoria está limitada

En la mañana, el asistente le solicita a Leonardo escribir C en un pedazo de papel, tal que él pueda hallar la estrategia óptima para seguirla. Sin embargo, Leonardo está obsesionado con mantener sus técnicas de trabajo en secreto, por lo cual, se niega a permitir que el asistente tenga el papel. El sólo le ha permitido al asistente leer C y tratar de recordarlo.

Desafortunadamente, la memoria del asistente es realmente mala. El sólo es capaz de recordar hasta

M bits. En general, esto puede impedir que él sea capaz de reconstruir la secuencia C completa. Es por ello, que el asistente ha desarrollado una manera inteligente de calcular una secuencia de bits que pueda recordar. Dicha secuencia la llamaremos *secuencia de ayuda" y será denotada con la letra A*.

Ejemplo 2

En la mañana, el asistente puede llevar el papel de Leonardo con la secuencia C, leer la secuencia, y luego tomar todas las decisiones necesarias. Una cosa que él puede elegir hacer, es examinar el estado actual del andamio luego de cada una de las solicitudes. Por ejemplo, cuando se utiliza una estrategia (sub-óptima) como la del ejemplo 1, la secuencia de los estados del andamio sería la siguiente [0, 2], [0, 2], [3, 2], [3, 0]. (Tenga en cuenta que él sabe que el estado inicial del andamio es [0, 1].)

Ahora asuma que tiene M = 16, lo cual, indica que el asistente es capaz de recordar hasta 16 bits de información. Como N = 4, podemos almacenar cada color utilizando 2 bits. Por lo tanto 16 bits son suficientes para almacenar la secuencia anterior de estados del andamio. El asistente puede calcular la siguiente secuencia de ayuda: A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`).

Luego más adelante en el día, el asistente puede decodificar la secuencia de ayuda y utilizarla para tomar sus decisiones.

(Por supuesto, con M = 16 el asistente puede además elegir recordar la secuencia C completa, utilizando sólo 8 de los 16 bits disponibles. En este ejemplo sólo hemos querido demostrar que pueden existir otras opciones, sin descartar ninguna buena solución.)

Problema

Usted deberá escribir *dos programas separados* en el mismo lenguaje de programación. Dichos programas serán ejecutados secuencialmente, sin ser capaces de comunicarse entre ellos durante su ejecución.

El primer programa será utilizado por el asistente en la mañana. Dicho programa recibirá la secuencia C, y debe calcular una secuencia de ayuda A.

El segundo programa debe ser utilizado por el asistente durante el día. Dicho programa debe recibir una secuencia de ayuda A, y luego deberá procesar la secuencia C de las peticiones de Leonardo. Note que la secuencia C sólo es revelada a este programa solicitud por solicitud, y cada solicitud debe ser procesada antes de recibir la siguiente.

Más especificamente, en el primer programa usted debe implementar una función `ComputeAdvice`(C, N, K, M), la cual, reciba como parámetros el arreglo de entrada C de N enteros (cada uno en el rango 0, ..., N - 1), el número K de colores sobre el andamio, y el número M de bits disponibles para la ayuda. Dicho programa debe calcular una secuencia de ayuda A que consista en a lo sumo M bits. La función debe enviar la secuencia A al sistema mediante una llamada, por cada bit en el orden deseado al siguiente procedimiento:

- `WriteAdvice`(B) — la cual agrega el bit B a la secuencia actual de ayuda A. (Usted puede llamar a este procedimiento a lo sumo M veces.)

En el segundo programa usted debe implementar una función `Assist`(A, N, K, R). La cual, recibirá como parámetros la secuencia de ayuda A, los enteros N y K definidos anteriormente, y la longitud actual R de la secuencia A en bits ($R \leq M$). Dicha rutina debe ejecutar la estrategia propuesta por usted para el asistente, utilizando las siguientes rutinas provistas para usted:

- `GetRequest`() — retorna el siguiente color requerido por Leonardo. (Ninguna información de las siguientes peticiones es revelada.)

- `PutBack`(T) — coloca el color T desde el andamio de nuevo en el estante. Usted puede llamar a esta rutina solo con T siendo uno de los colores que actualmente se encuentran en el andamio.

Cuando es ejecutada, su función `Assist` debe llamar a la función `GetRequest` exactamente N veces, y en cada llamada se recibe una de las solicitudes de Leonardo, en orden. Luego de cada llamada a `GetRequest`, si el color retornado es *not* en el andamio, usted *debe* llamar también a la función `PutBack`(T) con su elección de T. De otro modo, usted *no debe* llamar a `PutBack`(T). Fallar en este procedimiento es considerado un error y conllevará la terminación de su programa. Por favor tenga en mente que el andamio contiene los colores desde 0 a K - 1, inclusive.

Un caso de prueba particular será considerado como resuelto si sus dos funciones cumplen todas las restricciones impuestas, y el total de las llamadas a `PutBack`(T) es *exactamente igual* a la estrategia óptima de Leonardo. Note que si existen múltiples estrategias que permiten alcanzar el mismo número de llamadas a `PutBack`(T), su función tiene permitido elegir cualquiera de ellas. (Es decir, no es requerido seguir la estrategia de Leonardo, si existe una estrategia igualmente buena.)

Ejemplo 3

Continuando con el Ejemplo 2, asuma que en `ComputeAdvice` usted calculó la secuencia de ayuda A = (`0`, `0`, `1`, `0`, `0`, `0`, `1`, `0`, `1`, `1`, `1`, `0`, `1`, `1`, `0`, `0`). Con el objetivo de enviar esta información al sistema, usted debe hacerlo mediante 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` ahora será ejecutada, recibiendo la secuencia anterior A, y los valores N = 4, K = 2, and R = 16. La función `Assist` luego debe realizar exactamente N = 4 llamadas a `GetRequest`. También, luego de algunas peticiones, `Assist` deberá llamar a `PutBack`(T) con un valor apropiado para T.

La tabla siguiente muestra una secuencia de llamadas que corresponde a la estrategia (sub-óptima) del Ejemplo 1. El guión indica que no hubo llamada a la función `PutBack`.

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

Sub-tarea 1 [8 puntos]

- $N \leq 5\,000$.
- Usted puede utilizar a lo sumo M = 65 000 bits.

Sub-tarea 2 [9 puntos]

- $N \leq 100\,000$.
- Usted puede utilizar a lo sumo M = 2 000 000 bits.

Sub-tarea 3 [9 puntos]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Usted puede utilizar a lo sumo M = 1 500 000 bits.

Sub-tarea 4 [35 puntos]

- $N \leq 5\,000$.
- Usted puede utilizar a lo sumo M = 10 000 bits.

Sub-tarea 5 [hasta 39 puntos]

- $N \leq 100\,000$.
- $K \leq 25\,000$.
- Usted puede utilizar a lo sumo M = 1 800 000 bits.

El resultado de esta subtarea depende de la longitud R de la ayuda que su programa genere. Más específicamente, si R_{max} es el máximo (sobre todos los casos de prueba) de la longitud de la cadena de ayuda producida por su rutina `ComputeAdvice`, su puntaje será:

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

El primer archivo es llamado `advisor.c`, `advisor.cpp` o `advisor.pas`. Este archivo debe implementar la función `ComputeAdvice` como fue descrita anteriormente y puede llamar a la función `WriteAdvice`. El segundo archivo es denominado `assistant.c`, `assistant.cpp` o `assistant.pas`. Dicho archivo debe implementar la función `Assist` de la forma descrita anteriormente y puede llamar a las funciones `GetRequest` y `PutBack`.

Los encabezados de cada una de las funciones es a continuación.

Programas en C/C++

```
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();
```

Programas en Pascal

```
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;
```

Dichos programas deben comportarse en la manera descrita anteriormente. Por supuesto, usted es libre de implementar otras funciones para su uso interno. Para los programas en C/C++, sus funciones internas deben ser declaradas como estáticas, así pueden ser enlazadas sus funciones con el evaluador ejemplo. Adicionalmente, evite tener dos funciones (en cada programa) con el mismo nombre. Sus envíos no deben interactuar en ninguna forma con entrada/salida estándar, ni con ningún archivo.

Cuando programe su solución, usted además debe tener cuidado de las instrucciones siguientes (las plantillas que encontrará en su ambiente de competencia cumplen los requerimientos listados a continuación).

Programas C/C++

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

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

o

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

Los dos archivos `advisor.h` y `assistant.h` le serán provistos a usted en un directorio dentro de su ambiente de competición y adicionalmente estará disponible a través de la interfaz Web. Usted adicionalmente será provisto (a través de los mismos canales) con códigos y scripts necesarios para compilar y probar su solución. Especificamente, despues de copiar su solución en el directorio con dichos scripts, usted deberá ejecutar `compile_c.sh` o `compile_cpp.sh` (dependiendo del lenguaje de su código).

Programas en Pascal

Usted deberá utilizar las unidades `advisorlib` y `assistantlib`, respectivamente, en el advisor y en el asistente. Esto se logra al incluir en su código fuente la línea:

```
uses advisorlib;
```

o

```
uses assistantlib;
```

Los dos archivos `advisorlib.pas` y `assistantlib.pas` serán provistos para usted en un directorio dentro de su ambiente de competición y adicionalmente estarán disponibles para los competidores a través de la interfaz Web. Usted adicionalmente será provisto (a través de los mismos canales) con los códigos y scripts necesarios para compilar y probar su solución. Específicamente, luego de copiar su solución en el directorio con esos scripts, usted deberá ejecutar `compile_pas.sh`.

Evaluador Ejemplo

El evaluador ejemplo aceptará la entrada con el formato siguiente:

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

El evaluador primero ejecutará la rutina `ComputeAdvice`. Esto generará un archivo `advice.txt`, el cual, contendrá los bits individuales de la secuencia de ayuda, separados por espacios y terminados por un 2.

Entonces se procederá a ejecutar su función `Assist`, y se generará una salida en la cual, cada línea es de la forma "`R [number]`" o de la forma "`P [number]`". Las líneas del primer tipo indican llamadas a la función `GetRequest()` y las respuestas recibidas. Líneas del segundo tipo representan llamadas a `PutBack`() y los colores elegidos para ser guardados. La salida es terminada por una línea de la forma "`E`".

Por favor note que en el evaluador oficial, el tiempo de ejecución de su programa puede diferir ligeramente al que tiene en su máquina local. Esta diferencia no debería ser significante. Sin embargo, se le extiende la invitación a probar la interfaz de prueba para verificar cuando su solución está dentro del límite de tiempo para su ejecución.

Límites de Memoria y Tiempo

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