

Tarea 1. Monedas Pesadas

En el reino de lowland existió una muralla magnífica que durante siglos protegió su territorio de las invasiones. Sin embargo, exactamente hace un año una banda de ladrones decidió que la muralla era demasiado problemática y la incendió por completo. Debido a lo acontecido, la princesa Maria ha planeado contratar constructores expertos para restaurar la muralla a su antigua gloria. Para pagar a dichos constructores, fue a buscar el tesoro real donde encontró $N = 2^K$ monedas de las cuales, cada una posee un único número desde 0 hasta $N - 1$. El contador real, le informó que existen dos tipos de monedas – ligeras y pesadas, siendo todas las monedas pesadas del mismo peso y del mismo modo las monedas ligeras. También, de acuerdo con el contador, existe al menos una moneda pesada y una moneda ligera en el tesoro real.

Según los términos acordados con los constructores de la nueva muralla, la princesa Maria tiene que pagarles únicamente con monedas pesadas, y antes de que el trabajo pueda comenzar, debe pagar un anticipo de una moneda pesada. El problema es que el contador no sabe cuantas ni cuales de las monedas son pesadas. Por suerte, él cuenta con una balanza con la cual puede comparar el peso combinado de dos conjuntos de monedas de igual cantidad. El contador ahora debe asegurarse de que la cantidad de monedas pesadas existentes en el tesoro real sea suficiente para el pago además de encontrar la moneda con la que podrá pagar el anticipo. Sin embargo, esto va a tomarle mucho tiempo, y la princesa Maria quiere iniciar la construcción de la nueva muralla tan pronto como sea posible, por lo que ella acude hacia ti para pedir tu consejo.

Tarea

Escriba un programa `coins`, implementando la función `count_heavy`, la cual será compilada con el programa del juez y mediante su conexión con esta, encontrará el número total de monedas pesadas en el tesoro real así como el número de una de esas monedas.

Detalles de implementación

Tu función debe tener el siguiente formato:

```
pair<int, int> count_heavy(int k)
```

Recibirá como parámetro el número K (recuerde que el número de monedas es 2^K) y tiene que retornar un par de números enteros – la cantidad de monedas pesadas y el número único de una de estas monedas.

Para comunicarse con el programa del juez, se le brinda la siguiente función:

```
int weigh(const vector<int> &a, const vector<int> &b)
```

Recibe como parámetros 2 vectores los cuales contienen los números de las monedas para el pesaje. Los vectores deben tener el mismo número de elementos y no deben contener números repetidos. La función retornará un valor:

- ☐ 0, si los dos conjuntos de monedas tienen el mismo peso;
- ☐ -1, si el peso total de las monedas cuyos números están en el vector a es menor que el peso total de las monedas cuyos números están en el vector b ;
- ☐ +1, si el peso total de las monedas cuyos números están en el vector a es mayor que el peso total de las monedas cuyos números están en el vector b .

Note que la complejidad de la función `weigh` es proporcional con el número total de elementos en los vectores recibidos como parámetros.

Debes enviar un archivo `coins.cpp` al sistema el cual contenga la función `count_heavy`. El archivo puede tener cualquier otro código y funciones necesarios para tu trabajo, pero no debe tener una función `main`. Además, no debes leer desde la entrada estándar o imprimir en la salida estándar. Tu programa también debe incluir el archivo de cabecera `coins.h` como instrucción de preprocesamiento:

```
#include "coins.h"
```

Restricciones

$$1 \leq K \leq 20$$

Puntaje

Si, durante la ejecución, tu solución da una respuesta correcta a un test usando no más de 10000 llamadas a la función *weigh*, se calculara un coeficiente para ese test de acuerdo con la fórmula $coef = \min\left(1, \frac{11}{9} * \left(\frac{author+1}{yours+1}\right)\right)$, donde *author*, es la máxima cuenta de las llamadas a la función *weigh*, realizadas por una de las soluciones del autor (vease en las subtareas abajo), mientras que *yours* es la cuenta de las llamadas a la función *weigh*, realizadas por tu solución. En caso contrario, para ese test *coef* tendrá un valor de 0.

Cada subtask consiste de un cierto numero de grupos, cada grupo tiene exactamente tres tests. Cada uno de los grupos es evaluado **independientemente**, y el numero de puntos, que recibirás por cada grupo, es igual al producto del número de puntos previstos para este y el mínimo valor del coeficiente *coef* de entre los test de este grupo.

Subtareas

Subtarea	Puntos	<i>author</i>	Otras restricciones
1	5	≤ 40	$K \leq 5$
2	10	≤ 250	Habrà solamente o una moneda pesada o una ligera.
3	20	≤ 1000	No hay mas de K monedas pesadas o no mas de K monedas ligeras.
4	25	≤ 100	$K \leq 10$
5	40	≤ 500	No hay más restricciones.

Ejemplo de comunicación

Función del Contestant	Programa del juez
	Llamar count_heavy(2)
Llamar weigh({0, 1}, {2, 3})	Retornar valor -1
Llamar weigh({1, 2}, {0, 3})	Retornar valor +1
Llamar weigh({1}, {2})	Retornar valor 0
Retornar {3, 2}	

Probar localmente

Para probar de manera local se te brinda los archivos *coins.h* y *Lgrader.cpp*. Ubica tu archivo *coins.cpp* y los dos archivos brindados en la misma carpeta. Entonces compila los tres archivos juntos. De este modo, se obtendrá un programa para verificar la correctitud de tu función. El programa necesita la siguiente secuencia de datos:

- en la primera linea: un entero positivo indicando el valor de K .

- en la segunda línea: 2^K números con valores 1 o 0 dependiendo si la moneda correspondiente es pesada o ligera.

El programa devolverá de salida información sobre las llamadas de la función *weigh*, la respuesta que tu función *count_heavy* retorna y un mensaje indicando si la respuesta es correcta o no.