Soluciones entrenos OIE

The Big Prize

Te encuentras parado delante de una fila de cajas, numeradas desde 0 hasta $n-1$, de izquierda a derecha. Cada caja contiene un premio que no es revelado hasta que no se abre la caja. Hay $v \geq 2$ *tipos* de premios diferentes. Los tipos se numeran desde 1 hasta v, en orden *decreciente* de valor.

El premio del tipo 1 es el más caro de todos: un diamante. Hay exactamente un diamante en las cajas. El premio del tipo v es el más barato de todos: un chupetín. Para que el juego sea más emocionante, hay muchos más premios entre los más baratos que entre los más caros. Más precisamente, para cada t tal que $2 \leq t \leq v$ sabemos lo siguiente: si hay k premios de tipo $t-1$, hay *estrictamente más* de k^2 premios de tipo t.

Tu objetivo es ganar el diamante. Al finalizar el juego, deberás abrir una caja y ganarás el premio que esta contenga. Antes de elegir qué caja abrir, tienes la oportunidad de hacerle algunas preguntas a Rambod, el anfitrión del programa. Para cada pregunta, deberás elegir una cierta caja i. En respuesta a la pregunta, rambod te dará un vector a de 2 posiciones. Su significado es el siguiente:

- Entre las cajas que se encuentran a la izquierda de la caja i, hay exactamente $a[0]$ cajas que contiene un premio más caro que el que hay en la caja i.
- Entre las cajas que se encuentran a la derecha de la caja i, hay exactamente $a[1]$ cajas que contiene un premio más caro que el que hay en la caja i.

Por ejemplo, supongamos que $n = 8$ y que decides preguntar por la caja $i = 2$. La respuesta de Rambod será $a = [1, 2]$. Esta respuesta significa que:

- Exactamente una de las cajas 0 y 1 contiene un premio más caro que el que hay en la caja 2.
- Exactamente dos de las cajas 3, 4, ..., 7 contienen un premio más caro que el que hay en la caja 2.

Tu tarea consiste en identificar la caa¡ja que contiene el diamante, realizando pocas preguntas en total.

Detalles de implementación

Debes implementar la siguiente función:

```
int find_best(int n)
```

Esta función es llamada exactamente una vez por el evaluador.

- n: cantidad de cajas

- Esta función debe retornar el número de la caja que contiene el diamante, es decir, el único entero d ($0 \leq d \leq n-1$) tal que la caja d contiene un premio de tipo 1.

La función anterior puede llamar a la siguiente:

```
vector <int> ask(int i)
```

- i: número de la caja por la cual se quiere preguntar. El valor de i debe estar entre 0 y $n-1$, inclusive.
- Esta función devuelve el vector a con 2 elementos de forma que $a[0]$ sea la cantidad de premios más caros en cajas a la izquierda de la caja i, y $a[1]$ sea la cantidad de premios más caros en cajas a la derecha de la caja i.

Constraints

- $3 \leq n \leq 200000$
- El tipo de premio que hay en cada caja se encuentra entre 1 y v, inclusive.
- Hay exactamente un premio de tipo 1.
- Para cada t tal que $2 \leq t \leq v$, si hay k premios de tipo $t-1$, hay *estrictamente más* de k^2 premios de tipo t.

Subtareas y puntuación

- (20 puntos) Hay exactamente 1 diamante y $n-1$ chupetines (o sea $v = 2$). Se puede llamar a la funcion ask como máximo 10000 veces.
- (80 puntos) Sin restricciones adicionales.

En la subtarea 2, es posible obtener puntuación parcial. En concreto, si q es la cantidad máxima de preguntas que hace el programa para todos los casos de prueba obtendréis:

- 0 puntos si $q > 10000$
- 70 puntos si $10000 \geq q > 6000$
- $80 - (q - 5000)/100$ puntos si $6000 \geq q > 5000$
- 80 puntos si $5000 \geq q$

Nota: No hace falta que vuestra programa tenga main.

Solución

El primer detalle importante (aplicable a muchos otros problemas interactivos) es que es interesante guardarnos las respuestas a cada pregunta por si nos son útiles más adelante y, sobretodo, para evitar repertir las mismas preguntas. Para ello, usaremos una matriz de tamaño $n \times 2$ que inicializaremos a -1. A la hora de hacer una pregunta, en vez de usar la función que nos da el enunciado, llamaremos a una función alternativa creada por nosotros (*myask* en el código). Dentro de esta función comprobaremos si ya

hemos preguntado por una posición y en caso afirmativo, devolveremos el valor que habíamos guardado. Y si no, guardaremos el valor y lo devolveremos. De esta manera la implementación es muy limpia a la vez que evitamos repetir preguntas. Ahora ya podemos empezar con la solución.

Llamemos interesantes a aquellas cajas que contegan un premio que no sea un chupetín. La primera observación importante es que el número de premios interesantes nunca será mayor que 473. Puesto que hay menos de $n \leq 200000$ chupetines y se tiene que cumplir que si hay k premios del tipo $t-1$, hay más de k^2 premios del tipo t, sabemos que hay como mucho $\lfloor\sqrt{200000}\rfloor = 447$ premios del tipo exactamente superior a los chupetines. Como mucho hay $\lfloor\sqrt{447}\rfloor = 21$ premios del tipo siguiente y de manera análoga llegamos a que hay como mucho $447 + 21 + 4 + 1 = 473$ premios interesantes.

La idea de la solución es buscar cada uno de los premios interesantes y preguntar por ellos uno a uno. Sabremos que hemos encontrado el diamante cuando no haya ningún premio de más valor ni a la izquierda ni a la derecha.

En primer lugar, buscaremos un chupetín. Puesto que hay, como mucho, 473 premios interesantes, sabemos que preguntando por los primeros 474 premios, obtendremos seguro un chupetín. Será el que la suma de los valores devueltos al preguntar por su posición sea mayor. Una vez localizado el chupetín en la posición *ind*, buscaremos ,con la ayuda de una función recursiva, todos los premios intersantes en el intervalo $[ind, n]$ (los que hay en $[0, ind-1]$ han sido encontrados cuando buscábamos el primer chupetín). Además, sabemos cuántos premios interesantes hay en este intervalo (los mismos que hay a la derecha del chupetín en la posición *ind*).

Nuestra función recursiva tomará tres valores l, r, q indicando que en el intervalo $[l, r]$ hay q premios interesantes y que además la posición l contiene un chupetín. Si $q = 0$, podemos parar la recursión puesto que no quedan más interesantes en el intervalo, si $q = r - l$, sabemos que todos los premios en $[l+1, r]$ son interesantes (porque l no lo es) y podemos preguntar uno a uno y parar la recursión. Y si no, buscamos un chupetín que esta más alla de la posición $mid = (l + r)/2$ (supongamos que está en la posición c) y llamamos recursivamente a la función con los intervalos $[l, mid - 1]$ y $[c, r]$ recalculando, gracias a las preguntas que hemos hecho y las que habíamos hecho antes, cuantos interesantes hay en cada intervalo. Si no hay ningún chupetín más allá de *mid*, solo hacemos la recursión con el intervalo de la izquierda.

La gracia es que solo preguntaremos por cada premio interesante una vez (como mucho haremos 473 preguntas para estos tipos de premios). Y como hay pocos premios interesantes, habrá segmentos consecutivos con muchos chupetines, así que podremos cortar bastantes trozos de la recursión pronto haciendo las mínimas preguntas posibles.

Solo implementando esta idea, se pueden obtener más de 99 puntos. Para obtener el último punto, hay que reducir unas cuantas preguntas. La idea es darse cuenta de que no siempre hacen falta las 473 preguntas iniciales para encontrar un chupetín. Si la respuesta para alguna pregunta es que hay más de 40 premios de más valor (sumando las cantidades a los 2 lados), automáticamente sabemos que se trata de un chupetín (de hecho esta cota se puede mejorar bastante y para demostrarlo solo hace falta usar un argumento similar al que nos ha llevado a acotar el número de premios interesantes).

Si nos cuesta mucho encontrar el primer chupetín, será porque hay muchos premios interesantes al principio y por lo tanto, la recursión irá más rápida (menos premios a encontrar). Si, por el contrario, hay muchas piruletas al principio, añadiendo la comprobación que acabamos de comentar y un break, podremos parar muy pronto y evitar la mayor parte de las 473 preguntas. El código que se muestra a continuación implementa esta solución de manera bastante concisa.

Código (C++)

```cpp
#include <bits/stdc++.h>

using namespace std;

vector <vector <int> >mem;
int ans, mx;

vector <int> myask(int i) { // evitamos repetir preguntas
        if (mem[i][0] == -1) mem[i] = ask(i);
        return mem[i];
}

void solve(int l, int r, int q) {
        if (q == 0) return; // no hay cajas interesantes
        if (r-l == q) { // todas las cajas en [l+1,r] son interesantes
                for (int i = l+1; i <= r; ++i) {
                        myask(i);
                        if (mem[i][0]+mem[i][1] == 0) ans = i;
                }
                return;
        }
        int mid = (l+r)/2, lollipop = -1;
        for (int i = mid; i <= r; ++i) {
                myask(i);
                if (mem[i][0]+mem[i][1] == 0) {
                        ans = i;
                        return;
                }
                if (mem[i][0]+mem[i][1] == mx) {
                        lollipop = i; // primer chupetin despues de mid
                        break;
                }
        }
        if (lollipop == -1) { // no hay chupetines entre mid y r
                solve(l, mid-1, q-(r-mid+1));
        }
        else { // hacemos la recursion en las 2 mitades recalculando cuantos
        ↪  chupetines hay en cada mitad
                int q2 = mem[lollipop][0]-mem[l][0]-(lollipop-mid);
                solve(l, mid-1, q2);
                solve(lollipop, r, q-mem[lollipop][0]+mem[l][0]);
        }
}

int find_best(int n) {
        mem = vector <vector<int> > (n, vector <int> (2, -1));
        mx = 0;
        int ind = 0;
        for (int i = 0; i < min(n, 473); ++i) { // buscamos un chupetin
                myask(i);
                if (mem[i][0]+mem[i][1] == 0) return i;
                if (mx < mem[i][0]+mem[i][1]) {
                        mx = mem[i][0]+mem[i][1];
                        ind = i;
                }
                if (mx >= 40) break; // ya hemos encontrado chupetin
        }
        solve(ind, n-1, mem[ind][1]);
        return ans;
}
```