## Grupo Grupo 18

**Estudiantes:**

- BLANCO CARRASCO, PABLO JESÚS (DA08)
- BRAVO MATEOS, JORGE (DA10)

| Puntuación | Explicaciones (/2) | Coste (/2) | Algoritmo (/4) |
|---|---|---|---|
| 8'5 | 2 | 0 | 6'5 |

| ID envío | Usuario/a | Hora envío | Veredicto |
|---|---|---|---|
| 4726 | DA10 | 2024-10-02T12:58:44.710 | WA |
| 4713 | DA10 | 2024-10-02T12:52:27.887 | WA |
| 4692 | DA10 | 2024-10-02T12:43:01.511 | RTE |
| 4690 | DA10 | 2024-10-02T12:42:37.748 | RTE |
| 4669 | DA10 | 2024-10-02T12:36:49.932 | RTE |
| 4667 | DA10 | 2024-10-02T12:36:28.137 | RTE |

Fichero `FileName.cpp`

```
/**
 * Nombre y usuario de cada miembro de la pareja
 *
 * D10 Jorge Bravo
 * D08 Pablo Blanco
 *
 * Explica tu código:
 *
 * Utilizamos una cola de prioridad con elemento el numero del canal y su prioridad el numero
     de espectadores
 * (cuanto mas espectadores, mayor prioridad), usamos también una estructura con un mapa, que
     contiene el numero del canal
 * como clave y un par de enteros (espectadores y minutos de mayor audiencia acumulados) como
     valor y un entero
 * que va guardando el canal con más espectadores en ese momento.
 *
 * Empezamos con un for que rellena la cola de prioridad y la estructura de datos y seguimos
    con un while que recorre
 * todas las actualizaciones que se hacen a lo largo de la entrada, guardamos los minutos
    actuales y entramos en un for
 * que recorre los canales que se actualizan, este for termina cuando lee -1.
 * Actualizamos la prioridad del canal con mas espectadores y guardamis los minutos en el que
     ha sido lider.
 *
 * Terminamos con la salida, que es un for que recorre cada clave del mapa y muestra los
    minutos acumulados, siempre
 * y cuando sean mayor que 0.
 *
```

Siempre debemos indicar los costes en tiempo y espacio

- Creación de la cola $O(C * \log C)$
- Consultas $O(N * \log C)$
- Salida ordenada $O(C * \log C)$

```
 */

#include <fstream>
#include <iostream>
#include <unordered_map>
#include <utility>
#include <queue>
#include <algorithm>

using namespace std;
#include "IndexPQ.h"

struct Telemaco {
    unordered_map<int, pair<int, int>> mins; // guardamos los canales con sus espectadores y
    mins
    int pico; // guardamos el canal que va ganando
};

bool resuelveCaso() {
    int D, C, N; // Duración del prime time, número de canales y número de actualizaciones
    cin >> D >> C >> N;
    if (!cin)
        return false;

    IndexPQ<int, greater<int>> canales(C+1);
    Telemaco tel;

    int espec, maxMin = 0;
    for (int i = 0; i < C; i++) {
        cin >> espec;
        canales.push(i+1, espec);
        tel.mins[i] = { espec, 0 };

        if (maxMin < tel.mins[i].first) {
            maxMin = tel.mins[i].first;
            tel.pico = i;
        }
    }

    int tiempoAnterior = 0, minsActual = 0;

    int j = 0;
    while (j < N) {
        cin >> minsActual;

        tel.pico = canales.top().elem;
        tel.mins[tel.pico - 1].second += minsActual - tiempoAnterior;

        int canal, espec;
        bool fin = false;
        for (int i = 0; i < C + 1 && !fin; i++) {
```

```
            cin >> canal;
            if (canal == -1)
                fin = true;
            else {
                cin >> espec;

                canales.update(canal, espec);
                tel.mins[canal - 1].first = espec;
            }
        }

        tiempoAnterior = minsActual;
        j++;

    }

    tel.mins[canales.top().elem - 1].second += D - minsActual;


    for (int i = 0; i < C; i++) {
        auto a = tel.mins[canales.top().elem-1].second;
        auto c = canales.top().elem;

        if (a > 0)
            cout << c << "␣" << a << "\n";
        canales.pop();
    }

    cout << "---\n";

    return true;
}

int main() {
#ifndef DOMJUDGE
    std::ifstream in("casos.txt");
    auto cinbuf = std::cin.rdbuf(in.rdbuf());
#endif

    while (resuelveCaso());

#ifndef DOMJUDGE
    std::cin.rdbuf(cinbuf);
#endif
    return 0;
}
```

No tenéis en cuenta el orden que pide el ejercicio. 1º por nº de minuto