26 · La autonomía de los coches

26/10 09:44 (80484) **AC**

MARP48 | JIANUO WEN HU

Nota:

Autonomia_coches.cpp

```
/*
 Escribe aquí un comentario general sobre la solución, explicando cómo
 se resuelve el problema y cuál es el coste de la solución, en función
 del tamaño del problema.

 Utilizamos la clase ARM_kruskal.
 ARM_kruskal se utiliza para recorrer las carreteras. ARM_kruskal encuentra el camino con
 ↪ menor coste que una las ciudades, por lo que guardamos
 en una variable la autonomía mínima que el coche necesita para recorrer todas las carreteras.
 También se comprueba si todas las ciudades están conectadas.

 Complejidad de A log A donde A es el número de carreteras (aristas)
*/

bool resuelveCaso() {

    // leemos la entrada
    int N, M;
    cin >> N >> M;

    if (!cin)
        return false;

    GrafoValorado<int> gv(N);
    // leer el resto del caso y resolverlo
    int v, w, val;
    for (int i = 0; i < M; ++i) {
        cin >> v >> w >> val;
        --v; --w;
        Arista<int> aux(v, w, val);
        gv.ponArista(aux);
    }
    ARM_Kruskal<int> krus(gv);

    if (!krus.conexo()) {
        cout << "Imposible\n";
    }
    else {
        cout << krus.costeMax() << "\n";
    }


    return true;
}
```

ARM_Kruskal.h

```
#pragma once
#include <queue>
#include "GrafoValorado.h"
#include "ConjuntosDisjuntos.h"
```

```
using namespace std;

template <typename Valor>
class ARM_Kruskal {
private:
        std::vector<Arista<Valor>> _ARM;
        Valor coste;
        Valor max;
        bool unidos;
public:
        Valor costeARM() const {
                return coste;
        }

        std::vector<Arista<Valor>> const& ARM() const {
                return _ARM;
        }
        ARM_Kruskal(GrafoValorado<Valor> const& g) : coste(0), max(0), unidos(true) {
                priority_queue<Arista<Valor>, vector<Arista<Valor>>, greater<Arista<Valor>>>
                ↪  pq;
                for (auto e : g.aristas()) { //A
                        pq.push(e);
                }
                ConjuntosDisjuntos cjtos(g.V());
                while (!pq.empty()) { //A
                        auto a = pq.top(); pq.pop(); //log A
                        int v = a.uno(), w = a.otro(v);
                        if (!cjtos.unidos(v, w)) { //1, se ejecuta A veces
                                cjtos.unir(v, w); //1, se ejecuta V - 1 veces
                                _ARM.push_back(a); coste += a.valor();
                                if (a.valor() > max) max = a.valor();
                                if (_ARM.size() == g.V() - 1) break;
                        }
                }
                unidos = cjtos.num_cjtos() < 2;
        }

        Valor costeMax() { return max; }

        bool conexo() { return unidos; }
```