Con coste, explicado, Algoritmo bien
-0'5

9'5

Grupo P29

Estudiantes:

- MARÍN GÓMEZ, NÉSTOR ANTONIO (DA42)
- PARRA GONZÁLEZ, JAVIER MARÍA (DA45)

ID envío	Usuario/a	Hora envío	Veredicto
80323	DA45	2023-10-25T11:57:35.658	AC
80306	DA45	2023-10-25T11:40:40.929	AC

Fichero autonomia.cpp

```
/*@ <answer>
 *
 * Nombre y Apellidos:Néstor Marín Gómez DA42 && Javier Parra González DA45
 *
 *@ </answer> */

#include <iostream>
#include <fstream>
#include "PriorityQueue.h"
#include "ConjuntosDisjuntos.h"

using namespace std;

#include "GrafoValorado.h"  // propios o los de las estructuras de datos de clase

/*@ <answer>

 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.

    Hemos añadido la clase ARM_kruska que, calcula el ARM de grafo g pasasdo por el
    contructor.
    A esta clase, hemos añadido varios atributos, uno para ver la cantidad de componentes
    conexas entre si, otro para guardar la automonía minima necesaria para
    cubrir dos ciudades cualequiera del grafo.

    Hemos añadido una operaciones observadora para el atributo numConexas mencionado
    anteriomente llamada numConexass();

    Ha sido modificada la clase ARM_kruskal para obtener el número de componentes conexas
    entre si y además, mientras calcula el ARM obtien el valor máximo que corresponderá
    a la la autonomía minima necesaria para que el coche recorra dos ciudades cualesquiera
    sin quedarse sin EV(carga).

    Coste O(A log A) y espacio adicional de O(A) siendo A el número de aristas del grafo
    Valorado.
```

¿Por qué vale?
Estamos interesados en la longitud del tramo mayor dentro de los tramos más cortos para ir entre ciudades

```
 @ </answer> */


 // ================================================================
 // Escribe el código completo de tu solución aquí debajo
 // ================================================================
 //@ <answer>

template <typename Valor>
class ARM_Kruskal {
private:
    std::vector<Arista<Valor>> _ARM;
    Valor coste;
    int numConexas;
    int maxAuto;

public:
    Valor costeARM() const { return coste; }
    std::vector<Arista<Valor>> const& ARM() const { return _ARM; }

    int numConexass() const {
        return this->numConexas;
    }

    ARM_Kruskal(GrafoValorado<Valor> const& g) : coste(0), numConexas(0), maxAuto(0){

        PriorityQueue<Arista<Valor>> pq(g.aristas()); //O(A) siendo A el num de aristas.
        ConjuntosDisjuntos cjtos(g.V()); //O(V) siendo v el numero de vertices del grafo


        while (!pq.empty()) {
            auto a = pq.top();
            pq.pop();  //O(logA)
            int v = a.uno(), w = a.otro(v); //O(1)

            if (!cjtos.unidos(v, w)) { //O(lg ^* V)
                cjtos.unir(v, w);

                _ARM.push_back(a); //cte

                coste = a.valor();
                maxAuto = max(maxAuto, coste);

                if (_ARM.size() == g.V() - 1) break;

            }
        }
```

Una ref más: no la necesitas, no la copies -0'5

```
        numConexas = (cjtos.num_cjtos()); //cte
    }
};

bool resuelveCaso() {

    // leer los datos de la entrada
    int n; cin >> n; int m; cin >> m;

    if (!std::cin)  // fin de la entrada
        return false;

    GrafoValorado<int> g(n);
    // resolver el caso posiblemente llamando a otras funciones
    for (int i = 0; i < m; i++) {
        int longi, origen, destino;
        cin >> origen >> destino >> longi;
        Arista<int> a(origen-1, destino-1, longi);
        g.ponArista(a);
    }

    ARM_Kruskal<int> sol(g);
    // escribir la solución
    if (sol.numConexass() > 1) cout << "Imposible\n";
    else  cout << sol.costeARM() << "\n";
    return true;
}

//@ </answer>
//  Lo que se escriba dejado de esta línea ya no forma parte de la solución.

int main() {
    // ajustes para que cin extraiga directamente de un fichero
#ifndef DOMJUDGE
    std::ifstream in("casos.txt");
    auto cinbuf = std::cin.rdbuf(in.rdbuf());
#endif

    while (resuelveCaso());

    // para dejar todo como estaba al principio
#ifndef DOMJUDGE
    std::cin.rdbuf(cinbuf);
    system("PAUSE");
#endif
    return 0;
}
```