Grupo DA34

Estudiantes:

- PARRA GONZÁLEZ, JAVIER MARÍA (DA34)

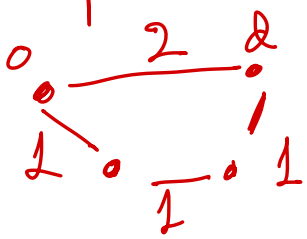
Puntuación	Coste (/0.5)	Explicaciones (/0.5)	Algoritmo (/1.5)	Implementación (/1)
1'25	0	0	0'5	0'75

ID envío	Usuario/a	Hora envío	Veredicto
34570	DA34	2024-01-11T10:24:16.115	WA
34561	DA34	2024-01-11T10:22:19.608	CE

Fichero `ej1.cpp`

```
/**
 * DA 2023-24. Plantilla Ejercicio 1
 * Escribe tu nombre y respuestas en las zonas indicadas
 *
 * @ <authors>
 * Nombre y apellidos: Javier Parra Gonzalez DA34
 * Usuario del juez de exámenes: DA34
 *@ </authors>
 */

#include <iostream>
#include <fstream>
#include <algorithm>

#include "IndexPQ.h"
#include "DigrafoValorado.h"


using namespace std;


/*@ <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.

 @ </answer> */

 // ================================================================
 // Escribe el código completo de tu solución debajo de la etiqueta <answer>
 // ================================================================
 //@ <answer>
```

Se pide la ruta que minimice la capacidad de aguante de la respiración. ¡Ojo!



Si

La ruta es la de 1's!

```
template <typename Valor>
class Dijkstra {
public:
private:
    const Valor INF = std::numeric_limits<Valor>::max();
    int origen;
    std::vector<Valor> dist;
    int max_resp ;
    int dest;
    IndexPQ<Valor> pq;
public:

    int resp() const{
        return max_resp;
    }

    Dijkstra(DigrafoValorado<Valor> const& g, int orig, int dest) : origen(orig), max_resp(0)
    ,dest(dest),
        dist(g.V(), INF), pq(g.V()) {
        dist[origen] = 0;
        pq.push(origen, 0);
        while (!pq.empty()) {
            int v = pq.top().elem; pq.pop();
            for (auto a : g.ady(v))
                relajar(a);
        }
    }
    bool hayCamino(int v) const { return dist[v] != INF; }
    Valor distancia(int v) const { return dist[v]; }

private:

    void relajar(AristaDirigida<Valor> a) {
        int v = a.desde(), w = a.hasta();

        if (dist[w] > dist[v] + a.valor()) {
            dist[w] = dist[v] + a.valor();
            max_resp = max(max_resp, a.valor());
            if ( dest != w)
                max_resp = 0;
            pq.update(w, dist[w]);
        }


    }
};


bool resuelveCaso() {

    // leer los datos de la entrada
```

Y el problema viene aquí

veo por donde vas pero si haces esto te quedas con la última

```
    int n; cin >> n;

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

    DigrafoValorado<int> g(n);
    for (int i = 0; i < m; i++) {
        int ini, fin, c;
        cin >> ini >> fin >> c;
        AristaDirigida<int> a(ini -1, fin-1, c);
        AristaDirigida<int> noAa(fin - 1, ini - 1, c);
        g.ponArista(a);
        g.ponArista(noAa);


    }


    int o; cin >> o;
    o--;
    int t; cin >> t;
    t--;

    Dijkstra<int> sol(g, o,t);

    if (!sol.hayCamino(t)) {
        cout << "Imposible" << "\n";
    }
    else
        cout << sol.resp() << "\n";



    return true;
}

//@ </answer>
//  Lo que se escriba debajo 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("casos1.txt");
    auto cinbuf = std::cin.rdbuf(in.rdbuf());
#endif

    // Resolvemos
    while (resuelveCaso());

    // para dejar todo como estaba al principio
```

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