Grupo Grupo 18

Estudiantes:

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

Puntuación	Explicaciones (/2)	Coste (/2)	Algoritmo (/4)	Implementación (/2)
10	2	2	4	2

ID envío	Usuario/a	Hora envío	Veredicto
10943	DA10	2024-10-30T12:28:52.840	AC
10907	DA10	2024-10-30T12:03:22.390	AC

Fichero `FileName.cpp`

```
/*@ <answer>
 *
 * Nombre y Apellidos:
 *
 * D10 Jorge Bravo
 * D08 Pablo Blanco
 *
 *@ </answer> */

#include <iostream>
#include <fstream>

using namespace std;

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

/*@ <answer>

Utilizamos un grafo dirigido que vamos rellenando con cada celda origen, cela destino, y el
    tiempo que se tarda de una
a otra como valor, después generamos un dijkstra al que le pasamos como origen la salida, de
    esta manera sabremos
todos los caminos que hay desde la salida que nos dan hasta cada una de las celdas, para ello
     hay que pasarle también el
grafo dirigido inverso, así se generan los mismos caminos pero teniendo como origen el
    destino al que debe llegar cada ratón.
(Esto es indiferente porque se tarda lo mismo en llegar de u a v que de v a u).

Después utilizamos un while que recorre todos las celdas, dentro tenemos una condición que
    comprueba si existe camino
```

```
hasta la celda específica, cuánto tiempo se tarda y si es menor o igual al solicitado, y si
    la celda que comprobamos no
es la celda origen.
Si todo esto se cumple se incrementa el contador de los ratones en 1. Cuando acabe el bucle
    imprimimos el contador.

Coste:

Inicialización de vectores: O ( V )
Cola de prioridad: O ( V )
Pop: O ( V logV )
Update: O ( A logV )

Total:
    En tiempo -> O ( A logV )
    En espacio adicional -> O ( V )

 @ </answer> */


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

const int INF = 1000000000;

template <typename Valor>
class Dijkstra {
public:
    Dijkstra(DigrafoValorado<Valor> const& g, int orig) :
        dist(g.V(), INF), pq(g.V()) {
        dist[orig] = 0;
        pq.push(orig, 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:

    std::vector<Valor> dist;
    IndexPQ<Valor> pq;
    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();
            pq.update(w, dist[w]);
        }
    }
};

bool resuelveCaso() {

    int n, s, t, p;
    cin >> n >> s >> t >> p;

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

    DigrafoValorado<int> dg(n);
    int u, v, tiempo;

    for (int i = 0; i < p; i++) {

        cin >> u >> v >> tiempo;
        dg.ponArista({ u - 1, v - 1, tiempo });
    }

    Dijkstra<int> dijkstra(dg.inverso(), s - 1);

    int ratones = 0, dest = 0;
    while (dest < n) { // O ( V )
        if (dijkstra.hayCamino(dest) /* O ( 1 ) */ && dijkstra.distancia(dest) /* O ( 1 ) */
    <= t && dest != s - 1)
            ratones++;
        dest++;
    }

    cout << ratones << "\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;
}
```