

Problema de laboratorio (Semana 7) Repartiendo paquetes	AC Enunciado	RTE Todos	8 Nota
---	------------------------	---------------------	-----------

Evaluable1.cpp

```
/*
    El coste en tiempo total es:  $O((N + C) * \log(C))$  donde  $N$  son las casas,  $C$  son los
    caminos.
    Hacemos Dijkstra una vez desde el origen (caminos mínimos de la oficina a cada casa), y
    otra con el grafo invertido para saber los caminos mínimos de cada casa a la oficina.
    Hay que explicarlo algo más.
    El coste en espacio adicional es:  $O(2N)$  donde  $N$  es el numero de casas.
    Cada vez que Creamos una comarca creamos un espacio adicional de  $N$ .
*/

template <typename Valor>
class Comarca {
private:
    const Valor INF = std::numeric_limits<Valor>::max();
    int origen;
    std::vector<Valor> dist;
    IndexPQ<Valor> pq;
public:
    Comarca(DigrafoValorado<Valor> const& g, int orig) : origen(orig),
        dist(g.V(), INF), pq(g.V()) {
        dist[origen] = 0;
        pq.push(origen, 0);
        //  $O((N + C) * \log(N))$ 
        while (!pq.empty()) {
            int v = pq.top().elem; pq.pop();
            for (auto a : g.ady(v)) //  $O(C * \log(N))$ 
                relajar(a);
        }
    }
    //  $O(1)$ 
    bool hayCamino(int v) const { return dist[v] != INF; }

    //  $O(1)$ 
    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();
            pq.update(w, dist[w]);
        }
    }
}
```

```

    }

};

bool resuelveCaso() {

    // Leemos la entrada.
    int N, C;
    cin >> N >> C;
    if (!cin)
        return false;

    // Leer el resto del caso y resolverlo.
    DigrafoValorado<int> cordilleraCantabrica(N);
    for (int i = 0; i < C; i++) // O(C)
    {
        int v, w, valor;
        cin >> v >> w >> valor;
        AristaDirigida<int> ar(v - 1, w - 1, valor);
        cordilleraCantabrica.ponArista(ar);
    }
    int O, P;
    cin >> O >> P;
    long esfuerzoTotal = 0, i = 0;
    bool imposible = false;
    Comarca<int> comarca(cordilleraCantabrica, 0 - 1); // O((N+C) * log(N))
    Comarca<int> comarcaInvertida(cordilleraCantabrica.inverso(), 0 - 1); // O((N+C) *
    ↵ log(N))
    // Consultas.
    while (i < P && !imposible) // O(P) si el bucle se para antes de tiempo no se lee toda la
    ↵ entrada del caso de prueba, y todo se lía
    {
        int p;
        cin >> p;
        if (!comarca.hayCamino(p - 1) || !comarcaInvertida.hayCamino(0 - 1)) imposible =
    ↵ true;
        else {
            esfuerzoTotal += comarca.distancia(p - 1);
            esfuerzoTotal += comarcaInvertida.distancia(p - 1);
        }
        i++;
    }

    cout << (imposible ? "Imposible" : to_string(esfuerzoTotal)) << "\n";
}

return true;
}

```