10

Problema 29

PAREDES SÁNCHEZ, MICHAEL STEVEN (TAIS74)

ID envio	Usuario/a	Hora envío	Veredicto
60316	TAIS74	2022-10-26 10:12	AC
60277	TAIS74	2022-10-26 09:57	AC

Fichero Source.cpp

```
*
* Nombre y Apellidos:  MICHAEL PAREDES / TAIS74
*                      IVAN GONZALEZ / TAIS50
*

Hemos hecho el grafo inverso, ya que queremos saber los caminos mas cortos a la salida.
   Luego hemos pasado ese grafo
a un dijkstra modificado, poniendo tres variables nuevas: un contador cont (los ratones que
   salen), un tiempo t(para
llegar a la salida) y un vector de booleanos salidos(para comprobar si un raton ya ha salido
    o no).
Mientras hacemos el dijkstra, comprobamos si la distancia de la sala en la que estamos es
   menor o igual al tiempo, y si
no ha salido anteriormente, aumentamos el contador y ponemos a true su posicion en el vector
    salidos.
Para terminar, mostramos por pantalla el contador del dj.

El coste del dijkstra modificado es el mismo que el original, por lo tanto es 0(A log V): A
   siendo las aristas, y
V siendo vertices, aunque el nuestro ocupa un poco mas de espacio por el uso del vector
   salidos.
```

```
template <typename Valor>
class Dijkstra {
public:
    //Coste 0(V + A)
    Dijkstra(DigrafoValorado<Valor> const& g, int orig, int t) : origen(orig),
        dist(g.V(), INF), ulti(g.V()), pq(g.V()), tiempo(t) {
        cont = 0;
        salidos.assign(g.V(), false);
        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]; }
    int contador() { return cont; }

private:
    const Valor INF = std::numeric_limits<Valor>::max();
    int origen;
    int cont;
    int tiempo;
    std::vector<bool> salidos;
    std::vector<Valor> dist;
    std::vector<AristaDirigida<Valor>> ulti;
    IndexPQ<Valor> pq;

    //0(1)
    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(); ulti[w] = a;
            if (dist[w] <= tiempo && !salidos[w]) {
                cont++;
                salidos[w] = true;
            }
            pq.update(w, dist[w]);
        }
    }
};


bool resuelveCaso() {

    // leer los datos de la entrada
    int N, S, T, P;
    cin >> N >> S >> T >> P;

    if (!std::cin)  // fin de la entrada
        return false;
    int aux1, aux2, tiempo;

    DigrafoValorado<int> grafo(N);
    for (int i = 0; i < P; i++)
    {
        cin >> aux1 >> aux2 >> tiempo;
        AristaDirigida<int> arista(aux2 - 1, aux1 - 1, tiempo);
        grafo.ponArista(arista);
    }
    // resolver el caso posiblemente llamando a otras funciones
     Dijkstra<int>dj (grafo, S-1, T);
    // escribir la solución
     cout << dj.contador() << '\n';

    return true;
```

mejor contarlos cuando salen de la cola de prioridad (más fácil)

```
}
```