9'5

## Problema 29

OTEGUI MARÍN, JOSÉ (TAIS73)

| ID envio | Usuario/a | Hora envío | Veredicto |
|---|---|---|---|
| 60382 | TAIS73 | 2022-10-26 10:53 | AC |
| 60381 | TAIS73 | 2022-10-26 10:50 | RTE |
| 60380 | TAIS73 | 2022-10-26 10:50 | RTE |
| 60379 | TAIS73 | 2022-10-26 10:49 | RTE |
| 60369 | TAIS73 | 2022-10-26 10:42 | RTE |
| 60343 | TAIS73 | 2022-10-26 10:27 | RTE |
| 60338 | TAIS73 | 2022-10-26 10:25 | RTE |

Fichero prob-29.cpp

```
*
* Indicad el nombre completo y usuario del juez de quienes habéis hecho esta solución:
* Estudiante 1: Alberto Chaves // TAIS20
* Estudiante 2: Jose Otegui // TAIS73
*

Hemos creado un grafo valorado dirigido y a su vez hemos devuelto el inverso para ver
la distancia de la salida a cada raton desde el final.
Cuando esta distancia es menor que T sumamos un 1 al contador solucion que devolvera
el numero de ratones que es capaz de salir.

El coste de Dijstra es O(A log V) y al hacerlo N veces supone un coste de
O(A log v + V * Gs(V))

donde V * Gs(V) es el coste de hacer el inverso
```

```
template <typename Valor>
class Dijkstra {
public:
  Dijkstra(DigrafoValorado<Valor> const& g, int orig) : origen(orig),
    dist(g.V(), INF), ulti(g.V()), 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]; }
  list<AristaDirigida<Valor>> camino(int v) const {
    list<AristaDirigida<Valor>> cam;
    // recuperamos el camino retrocediendo
    AristaDirigida<Valor> a;
    for (a = ulti[v]; a.desde() != origen; a = ulti[a.desde()])
```

```
      cam.push_front(a);
    cam.push_front(a);
    return cam;
  }
private:
  const Valor INF = std::numeric_limits<Valor>::max();
  int origen;
  std::vector<Valor> dist;
  std::vector<AristaDirigida<Valor>> ulti;
  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(); ulti[w] = a;
      pq.update(w, dist[w]);
    }
  }
};


bool resuelveCaso() {

  // leemos la entrada
  int N, S, T, P;
  cin >> N >> S >> T >> P;

  if (!cin)
    return false;

  // leer el resto del caso y resolverlo
  DigrafoValorado<int> laberinto(N);
  int v1, v2, a;
  for (int i = 0; i < P; i++) {
    cin >> v1 >> v2 >> a;
    laberinto.ponArista({ v1 - 1, v2 - 1, a });
  }
  auto laberintoInverso = laberinto.inverso();
  Dijkstra<int> recorridos(laberintoInverso, S -1);
  int sol = 0;
  for (int i = 0; i < N; i++) {
    if (i != S - 1) {
      int distancia = recorridos.distancia(i);
      if (distancia <= T && recorridos.hayCamino(i))
        sol++;
    }
  }
  cout << sol << "\n";

  return true;
}
```