

```

*****
*****                                Interfaz                                *****
*****

```

Se explica con: Ciudad
 Géneros: Ciudad

```

*****
*****                                Operaciones                                *****
*****

```

```

Crear(in m : Mapa) → res : Ciudad
Pre ≡ {true}
Post ≡ {res =obs Crear(m)} actualizar con la copia de mapa.
Complejidad:  $O(\#estaciones(m) * |e_m|)$ 
Descripcion: Crea una nueva ciudad a partir de un mapa.
Aliasing: Se realiza una copia del mapa.

Entrar(in ts : conjRapidoString, in e : estacion, in/out c : Ciudad)
Pre ≡ {e ∈ estaciones(mapa(c)) ∧ c =obs c₀}
Post ≡ {c =obs entrar(ts, e, c₀)}
Complejidad:  $O(\text{long}(e) + S \cdot R + N_{total})$ 
Descripcion: Agrega un conjunto de características nuevo a la ciudad, creando un nuevo robot.

Mover(in rur : nat, in e : estacion, in/out c : Ciudad)
Pre ≡ {u ∈ robots(c) ∧ e ∈ estaciones(c) ∧
  conectadas?(estacion(u, c), e, mapa(c)) ∧ c₀ =obs c}
Post ≡ {c =obs mover(rur, e, c₀)}
Complejidad:  $O(\text{long}(e) + \text{long}(\text{estacion}(u, c)) + \log(\#robotsEn(\text{estacion}(u, c), c)) + \log(\#robotsEn(e, c)))$ 
Descripcion: Mueve a un robot de una estacion a otra.

Inspeccion(in e : estacion, in/out c : Ciudad)
Pre ≡ {e ∈ estaciones(c)}
Post ≡ {c =obs inspección(e, c)}
Complejidad:  $O(\text{long}(e) + \log(\#robotsEn(e, c)))$ 
Descripcion: Remueve al robot mas infractor en la estacion e de la ciudad.

ProximoRUR(in c : ciudad) → res : nat
Pre ≡ {true}
Post ≡ {res =obs ProximoRUR(c)}
Complejidad:  $O(1)$ 
Descripcion: Devuelve el proximo rur disponible.

Mapa(in c : ciudad) → res : Mapa
Pre ≡ {true}
Post ≡ {res =obs Mapa(c)}
Complejidad:  $O(\text{Copiar}(c.mapa))$ 
Descripcion: Devuelve el mapa de la ciudad.
Aliasing: res es una copia de mapa

Robots(in c : ciudad) → res : ItVectorPointer(robot)
Pre ≡ {true}
Post ≡ {alias(res =obs Iterador Unidireccional(Puntero(robot)))}
Complejidad:  $O(1)$ 
Descripcion: Devuelve un iterador de los robots que hay en la ciudad.
Aliasing: res no es modificable.

Estacion(in c : ciudad, in u : nat) → res : estacion
Pre ≡ {u ∈ robots(c)}
Post ≡ {res =obs estacion(u, c)}
Complejidad:  $O(1)$ 
Descripcion: Devuelve la estacion donde esta el robot con rur u.

Tags(in c : ciudad, in u : nat) → res : itConj(string)
Pre ≡ {u ∈ robots(c)}
Post ≡ {alias(esPermutacion?(SecuSuby(res), tags(u, c)) ∧ vacia?(Anteriores(res)))}
Complejidad:  $O(1)$ 
Descripcion: Devuelve un iterador a los tags del robot u.

```

Aliasing: res no es modificable.

```
#Infracciones(in c : ciudad, in u : nat) → res : nat
Pre ≡ {u ∈ robots(c)}
Post ≡ {res =obs #infracciones(u, c)}
Complejidad: O(1)
Descripcion: Devuelve la cantidad de infracciones del robot con rur u.
```

```
*****
*****
***** Representación *****
*****
```

```
Ciudad se representa con city,
  donde city es: tupla(
    robots      : VectorPointer(robot),
    mapa        : Mapa,
    robotsEnEstacion : DiccString(colaPrioridad(robot))
  donde robot es: tupla(
    rur          : nat,
    infracciones : nat,
    tags         : Puntero(conjRapidoString),
    estacion     : string,
    infringe_restriccion : Vector(Bool),
    mi_estacion  : itCola(robot)
  )
```

```
Pre ≡ {true}
Post ≡ {res =obs (r1 < r2)}
Complejidad: O(1)
Descripción: función de comparación de robots
• < • (in r1 : robot, r2 : robot) → res : bool
  if r1.infracciones < r2.infracciones then
    res ← true
  else
    if r1.infracciones == r2.infracciones then
      res ← r1.rur < r2.rur
    else
      res ← false
    end if
  end if
end function
```

Escribimos el rep de ciudad informalmente:

- Las claves del diccionario e.robotsEnEstacion todas pertenecen a las estaciones definidas en e.mapa.
- Todos los robots pertenecientes a las colas de los significados de e.robotsEnEstacion tienen la clausula estación puesta al significado de la cola a la que pertenecen.
- Los elementos de e.robots que no sean NULL cumplen que
 - El rur es igual a su indice en e.robots.
 - El robot pertenece a la cola de prioridad en la entrada estacion de e.robotsEnEstacion (obviamente, la entrada debe estar también definida).
 - mi_estacion se corresponde con la posición del robot en la cola de prioridad asociada a la estación en la que se encuentra.
 - infringe_restriccion se corresponde con los caminos que hay en el mapa, y aparte se corresponde con si el robot verifica o no la restricción entre las sendas.
 - Los elementos en tags tienen una longitud menor o igual a 64.

```
Abs: ^ (city) c → Ciudad {Rep(c)}
(∀ e : ^ (city)) Abs(e) =obs c /
  convertir(e.robots) =obs robots(c) ∧
  long(e.robots) =obs ProximoRUR(c) ∧
  e.mapa =obs mapa(c) ∧
  (∀ u : rur)
    (u < Longitud(e.robots) ∧ e.robots[u] != NULL) ⇒1
      ((e.robots[u].estacion =obs estacion(u, c)) ∧
       (e.robots[u].tags =obs tags(u, c)) ∧
       (e.robots[u].infracciones =obs #infracciones(u, c)))
```

convertir: secu(puntero(α)) \rightarrow conj(α)

```
convertir(xs)  $\equiv$ 
  if vacia?(xs) then  $\phi$ 
  else
    if prim(xs) =obs NULL then convertir(fin(xs))
    else Ag(prim(xs), convertir(fin(xs))) fi
  fi
```

```
*****
*****                                Algoritmos                                *****
*****
```

```
iCrear(in m : Mapa)  $\rightarrow$  res : city
  var robots_en_estacion: DiccString(ColaPrioridad(robot))  $\leftarrow$  Crear()
  var it: itConj(string)  $\leftarrow$  Estaciones(m)

  while HaySiguiente(it) do
    Definir(robots_en_estacion, Siguiente(it), Crear())
    Avanzar(it)
  end while

  Definir(robots_en_estacion, *it, Crear())

  res  $\leftarrow$  {robots: Vacia(), mapa: Copiar(m), robotsEnEstacion: robots_en_estacion}
end function
```

Recorrer todas las estaciones en el mapa es $O(E)$. Definir cada entrada en robots_en_estacion es $O(|e_m|)$, siendo e_m el nombre de estacion mas largo, y hay que hacerlo E veces. Por esto, la complejidad es $O(|e_m| * E)$.

```
iEntrar(in ts : conjRapidoString, in e : estacion, in/out c : city)
  var rob : robot  $\leftarrow$  {
    tags: &ts,
    infracciones: 0,
    rur: ProximoRUR(city),
    infringe_restriccion: Vacia(),
    estacion: e,
    mi_estacion: Encolar(obtener(robotsEnEstacion, e), rob)}

  var it : ItVectorPointer(Restriccion)  $\leftarrow$  Sendas(c.mapa)

  while HayMas?(it) do
    AgregarAtras(rob.infringe_restriccion,  $\neg$ Verifica?(ts, *Actual(it)))
    Avanzar(it)
  end while

  AgregarAtras(c.robots, &rob)
end function
```

La operacion obtener(robotsEnEstacion, e) tiene complejidad $O(|e|)$. La operacion Encolar de cola de prioridad es $O(\text{Log de la cantidad de elementos de la cola})$. Como se evaluan todas las sendas para saber si un robot infringe o no, el recorrido lineal del iterador es $O(S)$, siendo S la cantidad de sendas que hay en el mapa. Evaluar si un robot infringe o no cada senda (Verifica?) es $O(R)$. Finalmente, agregar un robot a robots de la ciudad es $O(N)$ en el peor caso, siendo N la cantidad de robots en la ciudad, como esta explicado en agregar atras de vector. Este $O(N)$ por algebra de ordenes acota superiormente al costo logaritmico de encolar. Luego, la complejidad queda en $|e| + S*R + N_{\text{total}}$.

```
iMover(in rur : nat, in e : estacion, in/out c : city)
  var rob : puntero(robot)  $\leftarrow$  c.robots[rur]

  Borrar(Obtener(c.robotsEnEstacion, rob $\rightarrow$ estacion), rob $\rightarrow$ mi_estacion)

  var infringe : nat  $\leftarrow$  rob $\rightarrow$ infracciones
  var id_senda : nat  $\leftarrow$  idSenda(c.mapa, rob $\rightarrow$ estacion, e)

  if rob $\rightarrow$ infringe_restriccion[id_senda] then
    infringe++
```

```

end if

rob→infracciones ← infringe
rob→estacion ← e
rob→mi_estacion ← Encolar(obtener(c.robotsEnEstacion, e), *rob)
end function

```

La complejidad de Obtener es $O(|e|)$, siendo el rob→estacion. Borrar de la cola de prioridad teniendo el iterador al elemento es $O(\text{Log}(\#cola))$, siendo #cola la cantidad de elementos de la cola. Todo el calculo de si infringe o no es $O(1)$ porque ya esta precalculado. Luego, insertarlo en otra cola es $O(|e|)$ para encontrarla en el diccionario y $O(\text{Log}(\#cola))$ para insertarlo. Esto da como resultado $O(|e| + |e| + \text{Log}(\#cola1) + \text{Log}(\#cola2))$.

```

iInspeccion(in e : estacion, in/out c : city)
  var cola : colaPrioridad(robot) ← Obtener(c.robotsEnEstacion, e)

```

```

  if tamaño(cola) > 0 then
    var rob : robot ← Desencolar(cola)
    c.robots[rob.rur] ← NULL
  end if
end function

```

```

iProximoRUR(in c : city) → res : nat
  res ← Longitud(c.robots)
end function

```

```

iMapa(in c : city) → res : Mapa
  res ← Copiar(c.mapa)
end function

```

```

iRobots(in c : city) → res : ItVectorPointer(robot)
  res ← CrearIt(c.robots)
end function

```

```

iEstacion(in c : city, in u : nat) → res : estacion
  res ← (c.robots[u])→estacion
end function

```

```

iTags(in c : city, in u : nat) → res : itConj(string)
  res ← CrearIt((c.robots[u])→tags)
end function

```

```

i#Infracciones(in c : city, in u : nat) → res : nat
  res ← (c.robots[u])→infracciones
end function

```