

## 1. Módulo Lista Enlazada( $\alpha$ )

### Interfaz

**parámetros formales**

**géneros**  $\alpha$

**función** COPIAR(**in**  $a : \alpha$ )  $\rightarrow res : \alpha$

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{res =_{\text{obs}} a\}$

**Complejidad:**  $\Theta(\text{copy}(a))$

**Descripción:** función de copia de  $\alpha$ 's

**se explica con:** SECUENCIA( $\alpha$ ), ITERADOR BIDIRECCIONAL( $\alpha$ ).

**géneros:** lista( $\alpha$ ), itLista( $\alpha$ ).

### Operaciones básicas de lista

VACÍA()  $\rightarrow res : \text{lista}(\alpha)$

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{res =_{\text{obs}} <>\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** genera una lista vacía.

AGREGARADELANTE(**in/out**  $l : \text{lista}(\alpha)$ , **in**  $a : \alpha$ )  $\rightarrow res : \text{itLista}(\alpha)$

**Pre**  $\equiv \{l =_{\text{obs}} l_0\}$

**Post**  $\equiv \{l =_{\text{obs}} a \bullet l_0 \wedge res = \text{CrearItBi}(<>, l) \wedge \text{alias}(\text{SecuSuby}(res) = l)\}$

**Complejidad:**  $\Theta(\text{copy}(a))$

**Descripción:** agrega el elemento  $a$  como primer elemento de la lista. Retorna un iterador a  $l$ , de forma tal que Siguiente devuelva  $a$ .

**Aliasing:** el elemento  $a$  agrega por copia. El iterador se invalida si y sólo si se elimina el elemento siguiente del iterador sin utilizar la función ELIMINARSIGUIENTE.

### Operaciones del iterador

CREARIT(**in**  $l : \text{lista}(\alpha)$ )  $\rightarrow res : \text{itLista}(\alpha)$

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{crearItBi}(<>, l) \wedge \text{alias}(\text{SecuSuby}(it) = l)\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** crea un iterador bidireccional de la lista, de forma tal que al pedir SIGUIENTE se obtenga el primer elemento de  $l$ .

**Aliasing:** el iterador se invalida si y sólo si se elimina el elemento siguiente del iterador sin utilizar la función ELIMINARSIGUIENTE.

CREARITULT(**in**  $l : \text{lista}(\alpha)$ )  $\rightarrow res : \text{itLista}(\alpha)$

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{crearItBi}(l, <>) \wedge \text{alias}(\text{SecuSuby}(it) = l)\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** crea un iterador bidireccional de la lista, de forma tal que al pedir ANTERIOR se obtenga el último elemento de  $l$ .

**Aliasing:** el iterador se invalida si y sólo si se elimina el elemento siguiente del iterador sin utilizar la función ELIMINARSIGUIENTE.

## Representación

### Representación de la lista

lista( $\alpha$ ) se representa con **lst**

donde **lst** es **tupla**(*primero*: puntero(nodo), *longitud*: nat)

donde **nodo** es  $\text{tupla}(\text{dato}: \alpha, \text{anterior}: \text{puntero}(\text{nodo}), \text{siguiente}: \text{puntero}(\text{nodo}))$

$\text{Rep} : \text{lst} \rightarrow \text{bool}$

$\text{Rep}(l) \equiv \text{true} \iff (l.\text{primero} = \text{NULL}) = (l.\text{longitud} = 0) \wedge_L (l.\text{longitud} \neq 0 \Rightarrow_L$   
 $\text{Nodo}(l, l.\text{longitud}) = l.\text{primero} \wedge$   
 $(\forall i: \text{nat})(\text{Nodo}(l, i) \rightarrow \text{siguiente} = \text{Nodo}(l, i + 1) \rightarrow \text{anterior}) \wedge$   
 $(\forall i: \text{nat})(1 \leq i < l.\text{longitud} \Rightarrow \text{Nodo}(l, i) \neq l.\text{primero})$

$\text{Nodo} : \text{lst } l \times \text{nat} \rightarrow \text{puntero}(\text{nodo})$

$\{l.\text{primero} \neq \text{NULL}\}$

$\text{Nodo}(l, i) \equiv \text{if } i = 0 \text{ then } l.\text{primero} \text{ else } \text{Nodo}(\text{FinLst}(l), i - 1) \text{ fi}$

$\text{FinLst} : \text{lst} \rightarrow \text{lst}$

$\text{FinLst}(l) \equiv \text{Lst}(l.\text{primero} \rightarrow \text{siguiente}, l.\text{longitud} - \min\{l.\text{longitud}, 1\})$

$\text{Lst} : \text{puntero}(\text{nodo}) \times \text{nat} \rightarrow \text{lst}$

$\text{Lst}(p, n) \equiv \langle p, n \rangle$

$\text{Abs} : \text{lst } l \rightarrow \text{secu}(\alpha)$

$\{\text{Rep}(l)\}$

$\text{Abs}(l) \equiv \text{if } l.\text{longitud} = 0 \text{ then } <> \text{ else } l.\text{primero} \rightarrow \text{dato} \bullet \text{Abs}(\text{FinLst}(l)) \text{ fi}$

## Representación del iterador

**itLista( $\alpha$ ) se representa con iter**

donde **iter** es  $\text{tupla}(\text{siguiente}: \text{puntero}(\text{nodo}), \text{lista}: \text{puntero}(\text{lst}))$

$\text{Rep} : \text{iter} \rightarrow \text{bool}$

$\text{Rep}(it) \equiv \text{true} \iff \text{Rep}(*(\text{it}.\text{lista})) \wedge_L (\text{it}.\text{siguiente} = \text{NULL} \vee_L (\exists i: \text{nat})(\text{Nodo}(*it.\text{lista}, i) = \text{it}.\text{siguiente}))$

$\text{Abs} : \text{iter } it \rightarrow \text{itBi}(\alpha)$

$\{\text{Rep}(it)\}$

$\text{Abs}(it) =_{\text{obs}} b: \text{itBi}(\alpha) \mid \text{Siguietes}(b) = \text{Abs}(\text{Sig}(it.\text{lista}, it.\text{siguiente})) \wedge$   
 $\text{Anteriores}(b) = \text{Abs}(\text{Ant}(it.\text{lista}, it.\text{siguiente}))$

$\text{Sig} : \text{puntero}(\text{lst}) l \times \text{puntero}(\text{nodo}) p \rightarrow \text{lst}$

$\{\text{Rep}(\langle l, p \rangle)\}$

$\text{Sig}(i, p) \equiv \text{Lst}(p, l \rightarrow \text{longitud} - \text{Pos}(*l, p))$

$\text{Ant} : \text{puntero}(\text{lst}) l \times \text{puntero}(\text{nodo}) p \rightarrow \text{lst}$

$\{\text{Rep}(\langle l, p \rangle)\}$

$\text{Ant}(i, p) \equiv \text{Lst}(\text{if } p = l \rightarrow \text{primero} \text{ then } \text{NULL} \text{ else } l \rightarrow \text{primero} \text{ fi}, \text{Pos}(*l, p))$

Nota: cuando  $p = \text{NULL}$ ,  $\text{Pos}$  devuelve la longitud de la lista, lo cual está bien, porque significa que el iterador no tiene siguiente.

$\text{Pos} : \text{lst } l \times \text{puntero}(\text{nodo}) p \rightarrow \text{puntero}(\text{nodo})$

$\{\text{Rep}(\langle l, p \rangle)\}$

$\text{Pos}(l, p) \equiv \text{if } l.\text{primero} = p \vee l.\text{longitud} = 0 \text{ then } 0 \text{ else } 1 + \text{Pos}(\text{FinLst}(l), p) \text{ fi}$

## 2. Módulo Agentes

### Interfaz

**se explica con:** AGENTES

**géneros:** agentes

### Operaciones básicas de agentes

**NUEVOAGENTES**(in  $as: \text{dicc}(\text{placa}, \text{posicion}) \rightarrow res: \text{agentes}$

**Pre**  $\equiv \{\neg \emptyset?(\text{claves}(as))\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{nuevoAgentes}(as)\}$

**Complejidad:**  $O(Na)$  //Revisar al hacer algoritmo

**Descripción:** Crea un nuevo contenedor de Agentes con los agentes contenidos en *as*. *Na* es la cantidad de agentes definidos en *as*

AGENTES?(**in** *as*: agentes)  $\rightarrow$  *res* : conj(placa)

**Pre**  $\equiv$  {true}

**Post**  $\equiv$  {*res* =<sub>obs</sub> agentes?(*as*)}

**Complejidad:**  $\Theta(1)$

**Descripción:** Me devuelve un conjunto de todos los agentes definidos en *as*

AGREGARSANCION(**in** *a*: placa, **in/out** *as*: agentes)

**Pre**  $\equiv$  {estaAgente(*a*, *as*)  $\wedge$  *as* = *as*<sub>0</sub>}

**Post**  $\equiv$  {*as* =<sub>obs</sub> agregarSancion(*a*, *as*<sub>0</sub>)}

**Complejidad:**  $\Theta(1)$

**Descripción:** Agrega una sancion al agente *a*

CAMBIARPOSICION(**in** *a*: placa, **in** *p*: posicion, **in/out** *as*: agentes)

**Pre**  $\equiv$  {estaAgente(*a*, *as*)  $\wedge$  *as* = *as*<sub>0</sub>}

**Post**  $\equiv$  {*as* =<sub>obs</sub> cambiarPos(*a*, *p*, *as*<sub>0</sub>)}

**Complejidad:**  $\Theta(1)$

**Descripción:** Modifica la posicion del agente *a*, para que sea *p*

AGREGARCAPTURA(**in** *a*: placa, **in/out** *as*: agentes)

**Pre**  $\equiv$  {estaAgente(*a*, *as*)  $\wedge$  *as* = *as*<sub>0</sub>}

**Post**  $\equiv$  {*as* =<sub>obs</sub> agregarCaptura(*a*, *as*<sub>0</sub>)}

**Complejidad:**  $\Theta(1)$

**Descripción:** Agrega una captura al agente *a*

POSAGENTE(**in** *a*: placa, **in** *as*: agentes)  $\rightarrow$  *res* : posicion

**Pre**  $\equiv$  {estaAgente(*a*, *as*)}

**Post**  $\equiv$  {*res* =<sub>obs</sub> posicionAgente(*a*, *as*)}

**Complejidad:**  $\Theta(1)$

**Descripción:** Devuelve la posicion actual del agente *a*

SANCIONESAGENTE(**in** *a*: placa, **in** *as*: agentes)  $\rightarrow$  *res* : nat

**Pre**  $\equiv$  {estaAgente(*a*, *as*)}

**Post**  $\equiv$  {*res* =<sub>obs</sub> sancionesAgente(*a*, *as*)}

**Complejidad:**  $\Theta(1)$

**Descripción:** Devuelve las sanciones actuales del agente *a*

CAPTURASAGENTE(**in** *a*: placa, **in** *as*: agentes)  $\rightarrow$  *res* : nat

**Pre**  $\equiv$  {estaAgente(*a*, *as*)}

**Post**  $\equiv$  {*res* =<sub>obs</sub> capturasAgente(*a*, *as*)}

**Complejidad:**  $\Theta(1)$

**Descripción:** Devuelve la cantidad de capturas actuales del agente *a*

MASVIGILANTE(**in** *as*: agentes)  $\rightarrow$  *res* : placa

**Pre**  $\equiv$  {true}

**Post**  $\equiv$  {*res* =<sub>obs</sub> masVigilante(*as*)}

**Complejidad:**  $\Theta(1)$

**Descripción:** Devuelve el agente que mas capturas tiene en *as*. Si hubiera mas de uno, devuelve el de menor placa.

CONMISMASANCIONES(**in** *a*: placa, **in** *as*: agentes)  $\rightarrow$  *res* : puntero(conj(placa))

**Pre**  $\equiv$  {estaAgente(*a*, *as*)}

**Post**  $\equiv$  {*res* =<sub>obs</sub> conMismasSanciones(*a*, *as*)}

**Complejidad:**  $\Theta(1)$

**Descripción:** Devuelve la referencia a el conjunto de agentes que tienen la misma cantidad de sanciones que el agente *a*

**Aliasing:** *res* no es modificable. Cualquier referencia que se guarde queda invalidada si se agregan sanciones.

CONKSANCIONES(**in** *k*: nat, **in** *as*: agentes)  $\rightarrow$  *res* : conj(placa)

**Pre**  $\equiv$  {true}

**Post**  $\equiv \{res =_{\text{obs}} \text{conKSanciones}(k, as)\}$

**Complejidad:**  $O(Na)$  la primera vez,  $O(\log(Na))$  en siguientes llamadas mientras no ocurran sanciones.

**Descripción:** Devuelve el conjunto de agentes que tienen exactamente  $k$  sanciones.  $Na$  es la cantidad de agentes definidos en  $as$

**Aliasing:**  $res$  no es modificable. Cualquier referencia que se guarde queda invalidada si se agregan sanciones.

## Representación

### Representación de los Agentes

agentes se representa con **estr**

donde **estr** es `tupla(as: DiccRapido(nat, datos), claves: conj(nat) , masVig: nat, huboSanciones: bool, mismSanciones: lista(conj(nat)), kSanciones: Arreglo(tuplaK))`

donde **datos** es `tupla(pos: posicion, sanciones: nat, capturas: nat, conMismSanciones: itLista(conj(nat)))`

donde **tuplaK** es `tupla(sanciones: nat, placa: nat)`

donde **posicion** es `tupla(fila: nat, columna: nat)`

La idea de la lista enlazada `mismSanciones` es que guarde en cada posicion a todos aquellos agentes que comparten sanciones, con rapido acceso gracias al Iterador en los datos del agente. El arreglo `kSanciones` se utiliza para ordenar a los agentes por su cantidad de sanciones en tiempo  $O(N)$ , y poder buscar a alguno con  $K$  sanciones en  $O(\log(N))$  para acceder a aquellos que tienen la misma cantidad via `mismSanciones`

Invariante de representacion en castellano:

1. *claves* son las claves del diccionario *as*
2. *masVigilante* esta definido en agentes.
3. *masVigilante* es el agente con menor numero de placa entre aquellos que tienen mas capturas en el diccionario de agentes.
4. El arreglo *kSanciones* tiene almacenadas todas las placas de agentes.
5. Si no hubo sanciones ( $\neg \text{huboSanciones}$ ), entonces el arreglo *kSanciones* representa a los agentes en orden creciente de sanciones.
6. Los agentes de la lista *mismSanciones* no estan repetidos, y son exactamente los definidos en diccionario de agentes.
7. Para todo item de la lista *mismSanciones*, y para todo agente dentro del conjunto del item, la cantidad de sanciones es igual al resto del conjunto, y menor al de todos los agentes de items siguientes.

$\text{Rep} : \text{estr} \rightarrow \text{bool}$

$\text{Rep}(e) \equiv \text{true} \iff$

1.  $( (\forall a : \text{nat}) \text{def?}(a, e.as) \Rightarrow \text{En}(a, e.claves) ) \wedge$
2.  $\text{def?}(e.masVigilante, e.as) \wedge_L$
3.  $((\forall a : \text{nat}) \text{def?}(a, e.as) \Rightarrow_L (\text{obtener}(a, e.as).capturas = \text{obtener}(e.masVigilante, e.as).capturas \wedge a > e.masVigilante) \vee (\text{obtener}(a, e.as).capturas < \text{obtener}(e.masVigilante, e.as).capturas) \vee (a = e.masVigilante))) \wedge$
4.  $( (\forall a : \text{nat}) \text{def?}(a, e.as) \Rightarrow_L (\exists i : \text{nat}) e.kSanciones[i] = a \wedge$
5.  $\neg(e.huboSanciones) \Rightarrow \text{SancionesOrdenadas}(e) \wedge$
6.  $( ((\forall i : \text{nat}) i < \text{Longitud}(e.mismSanciones) \Rightarrow_L \text{MSDefinidosEnDicc}(e,i)) \wedge \text{DiccDefinidosEnMSY-NoRepetidos}(e) ) \wedge_L$
7.  $( (\forall i : \text{nat}) (i < \text{Longitud}(e.mismSanciones) \Rightarrow_L \text{MSTieneMismSanciones}(e, i)) \wedge (i < (\text{Longitud}(e.mismSanciones) - 1) \Rightarrow_L \text{MSCadaItemTieneDifSanciones}(e,i)) )$

Reemplazos sintacticos:

$\text{CorrespondenSanciones}(e) \equiv (\forall i : \text{nat}) i < \text{Tam}(e.kSanciones) \Rightarrow_L e.kSanciones[i].sanciones = \text{Obtener}(e.kSanciones[i].placa, e.as).sanciones$

$$\begin{aligned}
\text{SancionesOrdenadas}(e) &\equiv (\forall i : \text{nat}) \ i < (\text{Tam}(\text{e.kSanciones}) - 1) \Rightarrow_{\text{L}} \text{Obtener}(\text{e.kSanciones}[i], \text{e.as}).\text{sanciones} \leq \\
&\quad \text{Obtener}(\text{e.kSanciones}[i + 1], \text{e.as}).\text{sanciones} \\
\text{MSDefinidosEnDicc}(e, i) &\equiv (\forall a : \text{nat}) \ \text{En}(a, \text{e.mismSanciones}[i]) \Rightarrow \text{Def?}(a, \text{e.as}) \\
\text{DiccDefinidosEnMSYNoRepetidos}(e) &\equiv (\forall a : \text{nat}) \ \text{Def?}(a, \text{e.as}) \Rightarrow ((\exists! i : \text{nat}) \ i < \text{Longitud}(\text{e.mismSanciones}) \\
&\quad \Rightarrow_{\text{L}} \text{En}(a, \text{e.mismSanciones}[i])) \\
\text{MSTieneMismSanciones}(e, i) &\equiv (\forall a, a' : \text{nat}) \ (\text{En}(a, \text{e.mismSanciones}[i]) \wedge \text{En}(a', \text{e.mismSanciones}[i]) \wedge \neg(a = a')) \\
&\quad \Rightarrow_{\text{L}} \text{Obtener}(a, \text{e.as}) = \text{Obtener}(a', \text{e.as}) \\
\text{MSCadaItemTieneDifSanciones}(e, i) &\equiv (\forall a, a' : \text{nat}) \ (\text{En}(a, \text{e.mismSanciones}[i]) \wedge \text{En}(a', \text{e.mismSanciones}[i + 1])) \\
&\quad \Rightarrow_{\text{L}} \text{Obtener}(a, \text{e.as}) < \text{Obtener}(a', \text{e.as}) \\
\text{Abs} : \text{estr } e &\longrightarrow \text{agentes} && \{\text{Rep}(e)\} \\
\text{Abs}(e) &\equiv as : \text{agentes} / \\
&\quad (\forall a : \text{placa}) \ \text{Def?}(a, \text{e.as}) \iff a \in \text{agentes?}(as) \wedge_{\text{L}} \\
&\quad (\forall a : \text{placa}) \ \text{Def?}(a, \text{e.as}) \Rightarrow_{\text{L}} \\
&\quad (\text{Obtener}(a, \text{e.as}).\text{sanciones} =_{\text{obs}} \text{sancionesAgente}(a, as) \wedge \\
&\quad \text{Obtener}(a, \text{e.as}).\text{capturas} =_{\text{obs}} \text{capturasAgente}(a, as) \wedge \\
&\quad \text{Obtener}(a, \text{e.as}).\text{pos} =_{\text{obs}} \text{posicionAgente}(a, as) \wedge )
\end{aligned}$$

## Algoritmos

### Algoritmos de Agentes

## Lista de algoritmos

1.	NuevoAgentes . . . . .	6
2.	Agentes? . . . . .	6
3.	AgregarSancion . . . . .	7
4.	CambiarPosicion . . . . .	7
5.	AgregarCaptura . . . . .	8
6.	PosAgente . . . . .	8
7.	SancionesAgente . . . . .	8
8.	CapturasAgente . . . . .	8
9.	masVigilante . . . . .	8
10.	ConMismasSanciones . . . . .	9
11.	ConKSanciones . . . . .	9

```

iNuevoAgentes(in d: DiccRapido(placa,datos)) → res: estr
begin
  var
    itAMismSanciones : itConj
    itClavesD : itConj
    i : nat
    i ← 0 // O(1)
    res.as ← DiccRapidoVacio() // O(1)
    res.claves ← Claves(d) // O(N)
    res.masVig ← 0 // O(1)
    res.mismSanciones ← Vacia() // O(1)
    AgregarAtras(res.mismSanciones, res.claves) // O(N)
    itAMismSanciones ← CrearIt(res.mismSanciones) // O(1)
    res.kSanciones ← CrearArreglo(Longitud(res.claves)) // O(N)
    itClavesD ← CrearIt(res.claves) // O(1)
    while HayMas(itClavesD) do
      if res.masVig == 0 then // O(1)
        | res.masVig ← Actual(itClavesD) // O(1)
      end
      if Actual(itClavesD) < res.masVig then // O(1)
        | res.masVig ← Actual(itClavesD) // O(1)
      end
      Definir ( Actual(itClavesD), ( Obtener(Actual(itClavesD), d), 0, 0, 0, itAMismSanciones ), res.as )
      // O(1)
      res.kSanciones[i] ← Actual(itClavesD) // O(1)
      i ← i+1 // O(1)
      Avanzar(itClavesD) // O(1)
    end
    // while es O(N)
    res.huboSanciones ← false // O(1)
end
Complejidad: O(N), con N la cantidad de agentes

```

---

**Algoritmo 1:** NuevoAgentes

---

```

iAgentes?(in as: estr) → res: itConj
begin
  | res ← CrearIt(as.claves) // O(1)
end
Complejidad: O(1)

```

---

**Algoritmo 2:** Agentes?

---

*i*AgregarSancion( in  $a$  : nat, in/out  $as$  : estr )

```

begin
  var
    iteradorLista : itLista
    //Primero, le agrego la sancion directamente sobre el significado (DiccRapido hace aliasing en la operacion Obtener)
    Obtener(a, as.as).sanciones  $\leftarrow$  (Obtener(a, as.as).sanciones + 1) //O(1)
    as.huboSanciones  $\leftarrow$  true //O(1)
    //Ahora tengo que modificar mismSanciones para que refleje el cambio
    iteradorLista  $\leftarrow$  Obtener(a, as.as).conMismSanciones //O(1)
    //Borro a  $a$  del conjunto, porque ya no comparte sanciones con nadie del mismo
    Borrar(a, Siguiente(iteradorLista)) //O(N), pero se desestima
    //Me muevo al lugar que le corresponde ahora
    Avanzar(iteradorLista) //O(1)
    if  $\neg$  HaySiguiente(iteradorLista) then //O(1)
      //Si no hay nada, creo un nuevo elemento que solo me tiene a mi
      AgregarComoSiguiente(iteradorLista, Vacio()) //O(1)
      Siguiente(iteradorLista).Agregar(a) //O(1)
    else
      if Obtener(DameUno(Siguiente(iteradorLista)), as).sanciones > Obtener(a, as.as).sanciones then
        //O(1)
        //Si el que esta en el lugar al que iba tiene mas sanciones que yo, me agrego antes para mantener el orden
        creciente
        AgregarComoAnterior(iteradorLista, Vacio()) //O(1)
        Anterior(iteradorLista).Agregar(a) //O(1)
        Retroceder(iteradorLista) //O(1)
      else
        //Si no, debe tener las mismas (tiene mas que las que yo tenia antes, y yo sume una sancion, a lo sumo tiene
        la misma cantidad)
        Siguiente(iteradorLista).Agregar(a) //O(1)
      end
    end
    Obtener(a, as.as).conMismSanciones  $\leftarrow$  iteradorLista //O(1)
end

```

**Complejidad:** O(N) | Desestimando el borrado: O(1)

---

### Algoritmo 3: AgregarSancion

---

*i*CambiarPosicion(in  $a$  : nat, in  $p$  : posicion, in/out  $as$  : estr)

```

begin
  | Obtener(a, as.as).posicion  $\leftarrow$  p //O(1)
end

```

**Complejidad:** O(1)

---

### Algoritmo 4: CambiarPosicion

---

*i*AgregarCaptura(**in**  $a : \text{nat}$ , **in/out**  $as : \text{estr}$ )

```
begin
  Obtener( $a$ ,  $as.as$ ).capturas  $\leftarrow$  (Obtener( $a$ ,  $as.as$ ).capturas + 1) //O(1)
  //Ademas de sumar captura, hago el mantenimiento de masVigilante
  if Obtener( $a$ ,  $as.as$ ).capturas > Obtener( $as.masVigilante$ ,  $as$ ).capturas then //O(1)
    |  $as.masVigilante \leftarrow a$  //O(1)
  else
    | if Obtener( $a$ ,  $as.as$ ).capturas == Obtener( $as.masVigilante$ ,  $as$ ).capturas then //O(1)
      | if  $a < as.masVigilante$  then //O(1)
        |  $as.masVigilante \leftarrow a$  //O(1)
      | end
    | end
  end
```

**end**

**Complejidad:** O(1)

---

**Algoritmo 5:** AgregarCaptura

---

*i*PosAgente(**in**  $a : \text{nat}$ , **in**  $as : \text{estr}$ )  $\rightarrow$  res: posicion

```
begin
  | res  $\leftarrow$  Obtener( $a$ ,  $as.as$ ).posicion //O(1)
end
```

**Complejidad:** O(1)

---

**Algoritmo 6:** PosAgente

---

*i*SancionesAgente(**in**  $a : \text{nat}$ , **in**  $as : \text{estr}$ )  $\rightarrow$  res: nat

```
begin
  | res  $\leftarrow$  Obtener( $a$ ,  $as.as$ ).sanciones //O(1)
end
```

**Complejidad:** O(1)

---

**Algoritmo 7:** SancionesAgente

---

*i*CapturasAgente(**in**  $a : \text{nat}$ , **in**  $as : \text{estr}$ )  $\rightarrow$  res: nat

```
begin
  | res  $\leftarrow$  Obtener( $a$ ,  $as.as$ ).capturas //O(1)
end
```

**Complejidad:** O(1)

---

**Algoritmo 8:** CapturasAgente

---

*i*masVigilante(**in**  $as : \text{estr}$ )  $\rightarrow$  res: nat

```
begin
  | res  $\leftarrow as.masVigilante$  //O(1)
end
```

**Complejidad:** O(1)

---

**Algoritmo 9:** masVigilante

---



```

iConMismasSanciones(in a: nat, in as: estr) → res: puntero(conj(nat))
begin
  | res ← &(Siguiente(Obtener(a, as.as).conMismSanciones)) // O(1)
end
Complejidad: O(1)

```

---

**Algoritmo 10:** ConMismasSanciones

---

```

iConKSanciones(in k: nat, in as: estr) → res: puntero(conj(nat))
begin
  var
    i: nat
  if as.huboSanciones == true then // O(1)
    | CountingSortSanciones(as.kSanciones) // O(N+r)
  end
  //En el arreglo kSanciones tengo, en orden creciente por sancion, a los agentes. Busco a uno con k sanciones
  //Busqueda binaria me devuelve el i que cumple que las sanciones del agente en posicion i del arreglo son k
  i ← BusquedaBinariaSanciones(k, as.kSanciones) // O(log(N))
  res ← &(Siguiente(Obtener(as.kSanciones[i], as.as).conMismSanciones)) // O(1)
end
Complejidad:  $O(N + r + \log(N)) = O(N)$  donde r es el rango para CountingSort, si hubo sanciones.
               O(log(N)) en llamadas siguientes.

```

---

**Algoritmo 11:** ConKSanciones

---