



**DEPARTAMENTO  
DE COMPUTACION**

Facultad de Ciencias Exactas y Naturales - UBA

## Trabajo Práctico Número 2

---

Algoritmos y Estructuras de Datos II

**Grupo: 21**

Integrante	LU	Correo electrónico
Langberg, Andrés	249/14	andreslangberg@gmail.com
Walter, Nicolás	272/14	nicowalter25@gmail.com
Sticco, Patricio Bernardo	337/14	pbsticco@hotmail.com
Len, Julián	467/14	julianlen@gmail.com



**Facultad de Ciencias Exactas y Naturales**

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

<http://www.fcen.uba.ar>

1. **TAD POSICION ES TUPLA**(X:NAT, Y:NAT)
2. **TAD DIRECCION ES ENUM**{ IZQ,DER,ARRIBA,ABAJO}
3. **TAD AGENTE ES NAT**
4. **TAD NOMBRE ES STRING**
5. Suponemos que contamos con el TAD DiccionarioM, donde la funcion vacio() toma como parámetro un 'k', cuyo valor acota superiormente a la cantidad de claves.

# 1. Diseño del Tipo CAMPUS

## 1.1. Especificación

Se usa el TAD CAMPUS especificado por la cátedra.

## 1.2. Aspectos de la interfaz

### 1.2.1. Interfaz

Se explica con especificación de CAMPUS

Género *campus*

Operaciones básicas de **Campus**

**CREARCAMPUS**(*in c: nat, in f: nat*)  $\rightarrow res: campus$

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

**Post**  $\equiv \{ res=_{\text{obs}} crearCampus(c, f) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Crea un campus de c columnas y f filas.

**FILAS?**(*in c: campus*)  $\rightarrow res: nat$

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

**Post**  $\equiv \{ res=_{\text{obs}} filas(c) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve la cantidad de filas en el campus.

**COLUMNAS?**(*in c: campus*)  $\rightarrow res: nat$

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

**Post**  $\equiv \{ res=_{\text{obs}} columnas(c) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve la cantidad de columnas en el campus.

**OCUPADA?**(*in c: campus, in p: posicion*)  $\rightarrow res: bool$

**Pre**  $\equiv \{ posValida(p, c) \}$

**Post**  $\equiv \{ res=_{\text{obs}} ocupada?(p, c) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve *true* sii p esta ocupada por un obstaculo.

**AGREGAROBSTACULO**(*in/out c: campus, in p: posicion*)  $\rightarrow$

**Pre**  $\equiv \{ c=_{\text{obs}} c_0 \wedge posValida(p, c) \wedge_L \neg ocupada?(p, c) \}$

**Post**  $\equiv \{ c=_{\text{obs}} agregarObstaculo(p, c_0) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve *true* sii p esta ocupada por un obstaculo.

**POSVALIDA?**(*in c: campus, in p: posicion*)  $\rightarrow res: bool$

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

**Post**  $\equiv \{ res=_{\text{obs}} posValida?(p, c) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve *true* sii p es parte del mapa.

**ESINGRESO?**(*in c: campus, in p: posicion*)  $\rightarrow res: bool$

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

**Post**  $\equiv \{ res=_{\text{obs}} esIngreso?(p, c) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve *true* sii p es un ingreso.

VECINOS(**in**  $c$ : *campus*, **in**  $p$ : *posicion*)  $\rightarrow res$  : *conj(posicion)*

**Pre**  $\equiv \{ posValida(p, c) \}$

**Post**  $\equiv \{ res =_{obs} vecinos(p, c) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve el conjunto de posiciones vecinas a p.

VECINOSCOMUNES(**in**  $c$ : *campus*, **in**  $p$ : *posicion*, **in**  $p2$ : *posicion*)  $\rightarrow res$  : *conj(posicion)*

**Pre**  $\equiv \{ posValida(p, c) \wedge posValida(p2, c) \}$

**Post**  $\equiv \{ res =_{obs} vecinos(p, c) \cap vecinos(p2, c) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve el conjunto de vecinos comunes. La complejidad es  $\mathcal{O}(1)$  dado que los vecinos son a lo sumo 4, o sea, constantes

VECINOSCOMUNES(**in**  $c$ : *campus*, **in**  $p$ : *posicion*, **in**  $p2$ : *posicion*)  $\rightarrow res$  : *conj(posicion)*

**Pre**  $\equiv \{ posValida(p, c) \wedge posValida(p2, c) \}$

**Post**  $\equiv \{ res =_{obs} vecinos(p, c) \cap vecinos(p2, c) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve el conjunto de vecinos comunes entre dos posiciones. La complejidad es  $\mathcal{O}(1)$  dado que los vecinos son a lo sumo 4, o sea, constantes.

PROXPOSICION(**in**  $c$ : *campus*, **in**  $dir$ : *direccion*, **in**  $p$ : *posicion*)  $\rightarrow res$  : *posicion*

**Pre**  $\equiv \{ posValida(p, c) \}$

**Post**  $\equiv \{ res =_{obs} proxPosicion(p, d, c) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve la posicion vecina a p que esta en la direccion dir.

INGRESOSMASCERCANOS(**in**  $c$ : *campus*, **in**  $p$ : *posicion*)  $\rightarrow res$  : *conj(posicion)*

**Pre**  $\equiv \{ posValida(p, c) \}$

**Post**  $\equiv \{ res =_{obs} ingresosMasCercanos(p, c) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve el conjunto de ingresos mas cercanos a p.

### 1.3. Pautas de implementación

#### 1.3.1. Estructura de representación

*campus* se representa con *estr*

donde *estr* es

```
tupla(
  filas: nat ×
  columnas: nat ×
  mapa: vector(vector(bool))
)
```

#### 1.3.2. Justificación

#### 1.3.3. Invariante de Representación

**Informal**

1. El mapa debe tener tantas filas como indica la estructura, lo mismo con las columnas.

**Formal**

$Rep : estr \rightarrow boolean$

$(\forall e : estr)$

$\text{Rep}(e) \equiv (\text{true} \iff$   
 (1)  $e.\text{filas} = \text{longitud}(e.\text{mapa}) \wedge_L (\forall i : \text{nat})(i \leq e.\text{filas} \Rightarrow \text{longitud}(e.\text{mapa}[i]) = e.\text{columnas}))$

#### 1.3.4. Función de Abstracción

$\text{Abs} : \text{estr } e \longrightarrow \text{campus}$   $\{\text{Rep}(e)\}$   
 $(\forall e : \text{estr}) \text{ Abs}(e) =_{\text{obs}} c : \text{campus} /$   
 $\left( \text{filas}(c) = e.\text{filas} \wedge \text{columnas}(c) = e.\text{columnas} \wedge_L (\forall p : \text{posicion})(p.X \leq e.\text{filas} \wedge \right.$   
 $\left. p.Y \leq e.\text{columnas} \Rightarrow_L \text{ocupada?}(p, c) \Leftrightarrow (e.\text{mapa}[p.X])[p.Y]) \right)$

**1.3.5. Algoritmos**


---

```

1: function iCREARCAMPUS(in c: nat, in f: nat)  $\longrightarrow$  res: estr  $\triangleright \mathcal{O}(f^2 * c^2)$ 
2:   var vector(vector(bool)) mapa  $\leftarrow$  vacia(vacia())  $\triangleright \mathcal{O}(1)$ 
3:   var nat i  $\leftarrow$  0  $\triangleright \mathcal{O}(1)$ 
4:   while i  $\leq$  f do  $\triangleright \mathcal{O}(f)$ 
5:     var vector(bool) nuevo  $\leftarrow$  vacia()  $\triangleright \mathcal{O}(1)$ 
6:     var nat j  $\leftarrow$  0  $\triangleright \mathcal{O}(1)$ 
7:     while j  $\leq$  c do  $\triangleright \mathcal{O}(c)$ 
8:       AgregarAtras(nuevo, false)  $\triangleright \mathcal{O}(c)$ 
9:       j++  $\triangleright \mathcal{O}(1)$ 
10:    end while
11:    AgregarAtras(mapa, nuevo)  $\triangleright \mathcal{O}(f)$ 
12:    i++  $\triangleright \mathcal{O}(1)$ 
13:  end while
14:  res  $\leftarrow$   $\langle f, c, \text{mapa} \rangle$   $\triangleright \mathcal{O}(1)$ 
15: end function

```

---



---

```

1: function iAGREGAROBSTACULO(in/out e: estr, in p: posicion)  $\longrightarrow$  res: estr  $\triangleright \mathcal{O}(\text{longitud}(e.\text{mapa}[p.X]))$ 
2:   Agregar(e.mapa[p.X], p.Y, true)  $\triangleright \mathcal{O}(\text{longitud}(e.\text{mapa}[p.X]))$ 
3: end function

```

---



---

```

1: function iFILAS?(in e: estr)  $\longrightarrow$  res: nat  $\triangleright \mathcal{O}(1)$ 
2:   res  $\leftarrow$  e.filas  $\triangleright \mathcal{O}(1)$ 
3: end function

```

---



---

```

1: function iCOLUMNAS?(in e: estr)  $\longrightarrow$  res: nat  $\triangleright \mathcal{O}(1)$ 
2:   res  $\leftarrow$  e.columnas  $\triangleright \mathcal{O}(1)$ 
3: end function

```

---



---

```

1: function iOCUPADA?(in e: estr, in p: posicion)  $\longrightarrow$  res: bool  $\triangleright \mathcal{O}(1)$ 
2:   res  $\leftarrow$  (e.mapa[p.X])[p.Y]  $\triangleright \mathcal{O}(1)$ 
3: end function

```

---



---

```

1: function iPOSVALIDA?(in e: estr, in p: posicion)  $\longrightarrow$  res: bool  $\triangleright \mathcal{O}(1)$ 
2:   res  $\leftarrow$  ( $0 < p.X$ )  $\wedge$  ( $p.X \leq e.filas$ )  $\wedge$  ( $0 < p.Y$ )  $\wedge$  ( $p.Y \leq e.columnas$ )  $\triangleright \mathcal{O}(1)$ 
3: end function

```

---



---

```

1: function iESINGRESO?(in e: estr, in p: posicion)  $\longrightarrow$  res: bool  $\triangleright \mathcal{O}(1)$ 
2:   res  $\leftarrow$  (p.Y = 1)  $\vee$  (p.Y = e.filas)  $\triangleright \mathcal{O}(1)$ 
3: end function

```

---

---

```

1: function iVECINOS(in e: estr, in p: posicion)  $\rightarrow$  res : conj(posicion)  $\triangleright \mathcal{O}(1)$ 
2:   var conj(posicion) nuevo  $\leftarrow$  vacio()  $\triangleright \mathcal{O}(1)$ 
3:   Agregar(nuevo, (p.X+1,p.Y))
4:   Agregar(nuevo, (p.X-1,p.Y))
5:   Agregar(nuevo, (p.X,p.Y+1))
6:   Agregar(nuevo, (p.X,p.Y-1))
7:   var itConj(posicion) it  $\leftarrow$  crearIt(nuevo)
8:   while haySiguiente(it) do  $\triangleright \mathcal{O}(c)$ 
9:     if iPosValida?(e,siguiente(it)) then  $\triangleright \mathcal{O}(1)$ 
10:      avanzar(it)  $\triangleright \mathcal{O}(1)$ 
11:     else
12:       eliminarSiguiente(it)  $\triangleright \mathcal{O}(1)$ 
13:     end if
14:   end while
15:   res  $\leftarrow$  nuevo  $\triangleright \mathcal{O}(1)$ 
16: end function

```

---



---

```

1: function iVECINOSCOMUNES(in e: estr, in p: posicion, in p2: posicion)  $\rightarrow$  res : conj(posicion)  $\triangleright \mathcal{O}(1)$ 
2:   var conj(posicion) v  $\leftarrow$  vecinos(e,p)  $\triangleright \mathcal{O}(1)$ 
3:   var conj(posicion) v2  $\leftarrow$  vecinos(e,p2)  $\triangleright \mathcal{O}(1)$ 
4:   var conj(posicion) nuevo  $\leftarrow$  vacio()  $\triangleright \mathcal{O}(1)$ 
5:   var itConj(posicion) it  $\leftarrow$  crearIt(v)  $\triangleright \mathcal{O}(1)$ 
6:   while haySiguiente(it) do  $\triangleright \mathcal{O}(1)$ 
7:     if Pertenece?(v2,Siguiente(it)) then  $\triangleright \mathcal{O}(1)$ 
8:       Agregar(nuevo, Siguiente(it))  $\triangleright \mathcal{O}(1)$ 
9:     end if
10:    Avanzar(it)  $\triangleright \mathcal{O}(1)$ 
11:  end while
12:  res  $\leftarrow$  nuevo  $\triangleright \mathcal{O}(1)$ 
13: end function

```

---



---

```

1: function iVECINOSVALIDOS(in e: estr, in ps: conj(posicion))  $\rightarrow$  res : conj(posicion)  $\triangleright \mathcal{O}(1)$ 
2:   var conj(posicion) nuevo  $\leftarrow$  vacio()  $\triangleright \mathcal{O}(1)$ 
3:   var itConj(posicion) it  $\leftarrow$  crearIt(ps)  $\triangleright \mathcal{O}(1)$ 
4:   while haySiguiente(it) do  $\triangleright \mathcal{O}(1)$ 
5:     if PosValida?(e,siguiente(it)) then  $\triangleright \mathcal{O}(1)$ 
6:       Agregar(nuevo, siguiente(it))  $\triangleright \mathcal{O}(1)$ 
7:     end if
8:     avanzar(it)  $\triangleright \mathcal{O}(1)$ 
9:   end while
10:  res  $\leftarrow$  nuevo  $\triangleright \mathcal{O}(1)$ 
11: end function

```

---



---

```

1: function iDISTANCIA(in e: estr, in p: posicion, in p2: posicion)  $\rightarrow$  res : nat  $\triangleright \mathcal{O}(1)$ 
2:   res  $\leftarrow$  |p.X - p2.X| + |p.Y - p2.Y|  $\triangleright \mathcal{O}(1)$ 
3: end function

```

---

---

```

1: function iPROXPOSICION(in e: estr, in d: direccion, in p: posicion) → res : posicion           ▷  $\mathcal{O}(1)$ 
2:   var posicion p2 ← p
3:   if d==izq then
4:     p2 ← <p2.X+1, p2.Y>                                           ▷  $\mathcal{O}(1)$ 
5:   else
6:     if d==der then
7:       p2 ← <p2.X, p2.Y>                                           ▷  $\mathcal{O}(1)$ 
8:     else
9:       if d==arriba then                                           ▷  $\mathcal{O}(1)$ 
10:        p2 ← <p2.X, p2.Y-1>                                         ▷  $\mathcal{O}(1)$ 
11:      else
12:        p2 ← <p2.X, p2.Y+1>                                         ▷  $\mathcal{O}(1)$ 
13:      end if
14:    end if
15:  end if
16:  res ← p2                                                         ▷  $\mathcal{O}(1)$ 
17: end function

```

---



---

```

1: function iINGRESOSMASCERCANOS(in e: estr, in p: posicion) → res : conj(posicion)           ▷  $\mathcal{O}(1)$ 
2:   var conj(posicion) nuevo ← Vacio()                             ▷  $\mathcal{O}(1)$ 
3:   if distancia(e, p, <p.x,1>) < distancia(e, p, <p.x,e.filas>) then   ▷  $\mathcal{O}(1)$ 
4:     Agregar(nuevo, <p.x,1>)                                         ▷  $\mathcal{O}(1)$ 
5:   else
6:     if distancia(e, p, <p.x,1>) > distancia(e, p, <p.x,filas(e)>) then   ▷  $\mathcal{O}(1)$ 
7:       Agregar(nuevo, <p.x,e.filas>)                                   ▷  $\mathcal{O}(1)$ 
8:     else
9:       Agregar(nuevo, <p.x,1>)                                         ▷  $\mathcal{O}(1)$ 
10:      Agregar(nuevo, <p.x,e.filas>)                                   ▷  $\mathcal{O}(1)$ 
11:    end if
12:  end if
13:  res ← nuevo                                                         ▷  $\mathcal{O}(1)$ 
14: end function

```

---



## 2. Diseño del Tipo RASTRILLAJE

### 2.1. Especificación

Se usa el TAD CAMPUSSEGURO especificado por la cátedra.

### 2.2. Aspectos de la interfaz

#### 2.2.1. Interfaz

Se explica con especificación de CAMPUSSEGURO

Género *rastr*

Operaciones básicas de Rastrillaje

**CAMPUS**(*in r: rastr*)  $\rightarrow res : campus$

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

**Post**  $\equiv \{ res =_{\text{obs}} campus(r) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve el campus.

**ESTUDIANTES**(*in r: rastr*)  $\rightarrow res : conj(nombre)$

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

**Post**  $\equiv \{ res =_{\text{obs}} estudiantes(r) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve el conjunto de estudiantes presentes en el campus.

**HIPPIES**(*in r: rastr*)  $\rightarrow res : conj(nombre)$

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

**Post**  $\equiv \{ res =_{\text{obs}} hippies(r) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve el conjunto de hippies presentes en el campus.

**AGENTES**(*in r: rastr*)  $\rightarrow res : conj(agente)$

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

**Post**  $\equiv \{ res =_{\text{obs}} agentes(r) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve el conjunto de agentes presentes en el campus.

**POSESTUDIANTEYHIPPIE**(*in r: rastr, in id: nombre*)  $\rightarrow res : posicion$

**Pre**  $\equiv \{ id \in (estudiantes(r) \cup hippies(cs)) \}$

**Post**  $\equiv \{ res =_{\text{obs}} posEstudianteYHippie(id, r) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve la posición del estudiante/hippie pasado como parámetro.

**POSAGENTE**(*in r: rastr, in a: agente*)  $\rightarrow res : posicion$

**Pre**  $\equiv \{ a \in posAgente(a, r) \}$

**Post**  $\equiv \{ res =_{\text{obs}} posAgente(a, r) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve la posición del agente pasado como parámetro.

CANTSANCIONES(**in**  $r: rastr$ , **in**  $a: agente$ )  $\rightarrow res: nat$

**Pre**  $\equiv \{ a \in cantSanciones(a, r) \}$

**Post**  $\equiv \{ res =_{\text{obs}} cantSanciones(a, r) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve la cantidad de sanciones recibidas por el agente pasado como parámetro.

CANTHIPPIESATRAPADOS(**in**  $r: rastr$ , **in**  $a: agente$ )  $\rightarrow res: nat$

**Pre**  $\equiv \{ a \in agentes(r) \}$

**Post**  $\equiv \{ res =_{\text{obs}} cantHippiesAtrapados(a, r) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve la cantidad de hippies atrapados por el agente pasado como parámetro.

COMENZARRASTRILLAJE(**in**  $c: campus$ , **in**  $d: dicc(agente, posicion)$ )  $\rightarrow res: rastr$

**Pre**  $\equiv \{ (\forall a: agente)(def?(a, d) \Rightarrow_L (posValida?(obtener(a, d))) \wedge \neg ocupada?(obtener(a, d), c)) \wedge (\forall a, a_2: agente)((def?(a, d) \wedge def?(a_2, d) \wedge a \neq a_2) \Rightarrow_L obtener(a, d) \neq obtener(a_2, d)) \}$

**Post**  $\equiv \{ res =_{\text{obs}} comenzarRastrillaje(c, d) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Crea un Rastrillaje.

INGRESARESTUDIANTE(**in/out**  $r: rastr$ , **in**  $e: nombre$ , **in**  $p: posicion$ )  $\rightarrow$

**Pre**  $\equiv \{ r = r_0 \wedge e \notin (estudiantes(r) \cup hippies(r)) \wedge esIngreso?(p, campus(r)) \wedge \neg estaOcupada?(p, r) \}$

**Post**  $\equiv \{ r =_{\text{obs}} ingresarEstudiante(e, p, r_0) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Modifica el rastrillaje, ingresando un estudiante al campus.

INGRESARHIPPIE(**in/out**  $r: rastr$ , **in**  $h: nombre$ , **in**  $p: posicion$ )  $\rightarrow$

**Pre**  $\equiv \{ r = r_0 \wedge h \notin (estudiantes(r) \cup hippies(r)) \wedge esIngreso?(p, campus(r)) \wedge \neg estaOcupada?(p, r) \}$

**Post**  $\equiv \{ r =_{\text{obs}} ingresarHippie(h, p, r_0) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Modifica el rastrillaje, ingresando un hippie al campus.

MOVERESTUDIANTE(**in/out**  $r: rastr$ , **in**  $e: nombre$ , **in**  $dir: direccion$ )  $\rightarrow$

**Pre**  $\equiv \{ r = r_0 \wedge e \in estudiantes(r) \wedge (seRetira(e, dir, r) \vee (posValida?(proxPosicion(posEstudianteYHippie(e, r), dir, campus(r)), campus(r)) \wedge \neg estaOcupada?(proxPosicion(posEstudianteYHippie(e, r), dir, campus(r)), r))) \}$

**Post**  $\equiv \{ r =_{\text{obs}} moverEstudiante(e, d, r_0) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Modifica el rastrillaje, al mover un estudiante del campus.

MOVERHIPPIE(**in/out**  $r: rastr$ , **in**  $h: nombre$ )  $\rightarrow$

**Pre**  $\equiv \{ r = r_0 \wedge h \in hippies(r) \wedge \neg todasOcupadas?(vecinos(posEstudianteYHippie(h, r), campus(r)), r) \}$

**Post**  $\equiv \{ r =_{\text{obs}} \text{moverHippie}(r, r_0) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Modifica el rastillaje, al mover un hippie del campus.

MOVERAGENTE(**in/out**  $r$ : *rastr*, **in**  $a$ : *agente*)  $\rightarrow$

**Pre**  $\equiv \{ r = r_0 \wedge a \in \text{agentes}(r) \wedge \text{cantSanciones}(a, r) \leq 3 \wedge \neg \text{todasOcupadas}(\text{vecinos}(\text{posAgente}(a, r), \text{campus}(r)), r) \}$

**Post**  $\equiv \{ r =_{\text{obs}} \text{moverAgente}(a, r_0) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Modifica el rastillaje, al mover un agente del campus.

MASVIGILANTE(**in**  $r$ : *rastr*)  $\rightarrow res$ : *agente*

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

**Post**  $\equiv \{ res =_{\text{obs}} \text{masVigilante}(r) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve el agente con mas capturas.

CONKSANCIONES(**in**  $r$ : *rastr*, **in**  $k$ : *nat*)  $\rightarrow res$ : *conj(agente)*

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

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

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve el agente con mas capturas.

CONMISMASANCIONES(**in**  $r$ : *rastr*, **in**  $a$ : *agente*)  $\rightarrow res$ : *conj(agente)*

**Pre**  $\equiv \{ a \in \text{agentes}(r) \}$

**Post**  $\equiv \{ res =_{\text{obs}} \text{conMismasSanciones}(a, r) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve el conjunto de agentes con la misma cantidad de sanciones que a.

## 2.3. Pautas de implementación

### 2.3.1. Estructura de representación

*campus* se representa con *estr*

donde *estr* es

*tupla*(  
 campo: *campus*  $\times$   
 agentes: *diccPromedio(agente ; datosAg)*  $\times$   
 posAgentesLog: *arreglo(tupla(placa;posicion))*  $\times$   
 hippies: *conjLineal(datosHoE)*  $\times$   
 estudiantes: *conjLineal(datosHoE)*  $\times$   
 posCiviles: *diccString(nombre;posicion)*  $\times$   
 posRapida: *diccLineal(nombre;posicion)*  $\times$   
 quienOcupa: *vector(vector(datosPos))*  $\times$   
 masVigilante: *itConj(agente)*  $\times$   
 agregoEn1: *lista(datosK)*  $\times$   
 buscoEnLog: *vector(datosK)*  
 )

**donde** *datosAg* es

*tupla*(  
  QSanciones: *nat* ×  
  premios: *nat* ×  
  posActual: *posicion* ×  
  grupoSanciones: *itConj(agente)* ×  
  verK: *itLista(datosK)*  
)

**donde** *datosHoE* es

*tupla*(  
  ID: *nombre* ×  
  posActual: *itDicc(nombre;posicion)*  
)

**donde** *datosPos* es

*tupla*(  
  ocupada?: *bool* ×  
  queHay: *clases* ×  
  hayCana: *itDicc(agente)* ×  
  hayHoE: *itConj(nombre)*  
)

**donde** *clases* es enum{“agente”, “estudiante”, “hippie”, “obstaculo”, “nada”}

**donde** *datosK* es

*tupla*(  
  K: *nat* ×  
  grupoK: *conjLineal(agente)*  
)

### 2.3.2. Justificación

### 2.3.3. Invariante de Representación

#### Informal

1. El mapa debe tener tantas filas como indica la estructura, lo mismo con las columnas.

#### Formal

$\text{Rep} : \text{estr} \longrightarrow \text{boolean}$

$(\forall e : \text{estr})$

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

$(1)(2)(3)(4) (\forall a, a2 : \text{Agente})(a \neq a2 \wedge \text{definido?}(a, e.\text{agentes}) \wedge \text{definido?}(a2, e.\text{agentes})$

$\wedge_{\text{L}} \text{PosValida}(e.\text{campo}, \text{obtener}(a, e.\text{agentes}).\text{PosActual}) \wedge \text{PosValida}(e.\text{campo}, \text{obtener}(a2, e.\text{agentes}).\text{PosActual})) \Rightarrow_{\text{L}}$

$\text{obtener}(a, e.\text{agentes}).\text{PosActual} \neq \text{obtener}(a2, e.\text{agentes}).\text{PosActual}$

$\wedge (\text{obtener}(a, e.\text{agentes}).\text{Qsanciones} = \text{siguiente}(\text{obtener}(a, e.\text{agentes}).\text{verK}).\text{K}$

$\wedge \text{obtener}(a, e.\text{agentes}).\text{grupoSanciones} = \text{siguiente}(\text{obtener}(a, e.\text{agentes}).\text{verK}).\text{grupoK}$

$\wedge (a2 \in \text{obtener}(a, e.\text{agentes}).\text{grupoSanciones}) \iff (\text{obtener}(a, e.\text{agentes}).\text{Qsanciones} = \text{obtener}(a2, e.\text{agentes}).\text{Qsanciones})$

$\wedge (5) \text{TodasLasPlacas}(e, e.\text{posAgentesLog}) = \text{claves}(e.\text{agentes})$

$\wedge (6) (\forall a3 : \text{agente}, t : \text{tupla}(\text{agente}, \text{posicion}))(t \in e.\text{posAgentesLog} \wedge a3 = \Pi_1(t) \wedge_{\text{L}} \text{definido?}(a3, e.\text{agentes}) \Rightarrow_{\text{L}} \text{obte-}$

$\text{ner}(a3, e.\text{agentes}) = \Pi_2(t))$

$\wedge \text{enOrden}(e.\text{posAgentesLog}) \wedge \text{enOrden}(e.\text{buscoEnLog})$

$\wedge (7) \text{UnionConjuntos}(e, e.\text{lista}) = \text{claves}(e.\text{agentes})$

$\wedge (\forall h, h1 : \text{tupla}(\text{nombre}, \text{itDicc}(\text{nombre}, \text{posicion}))) (h \in e.\text{hippies} \wedge h1 \in e.\text{hippies} \wedge \Pi_1(h) \neq \Pi_1(h1)) \Rightarrow_{\text{L}} (\Pi_2(h) \neq \Pi_2(h1))$

$\wedge e.\text{posCiviles} = e.\text{posRapida} \wedge (\forall hi : \text{nombre}, e : \text{nombre}) ((\text{definido?}(hi, e.\text{posCiviles}) \wedge \text{definido?}(e, e.\text{posCiviles})) \Rightarrow_{\text{L}} \text{obte-}$

$\text{ner}(e, e.\text{posCiviles}) \neq \text{obtener}(hi, e.\text{posCiviles}))$

$\wedge (\forall a : \text{agente}, civ : \text{nombre}) (\text{definido?}(a, e.\text{agentes}) \wedge \text{definido?}(civ, e.\text{posCiviles}))$

$\Rightarrow_{\text{L}} (\text{obtener}(a, e.\text{agentes}) \neq \text{obtener}(civ, e.\text{posCiviles})) \wedge (e.\text{hippies} \cap e.\text{estudiantes}) = \emptyset$

$\wedge \text{JuntarIDS}(e.\text{estudiantes}) \cup \text{JuntarIDS}(e.\text{hippies}) = \text{claves}(e.\text{posCiviles}))$

### 2.3.4. Función de Abstracción

$\text{Abs} : \text{estr } e \longrightarrow \text{campus}$

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

$(\forall e : \text{estr}) \text{Abs}(e) =_{\text{obs}} c : \text{campus} /$

$(\text{filas}(c) = e.\text{filas} \wedge \text{columnas}(c) = e.\text{columnas} \wedge_{\text{L}} (\forall p : \text{posicion})(p.X \leq e.\text{filas} \wedge$

$p.Y \leq e.\text{columnas} \Rightarrow_{\text{L}} \text{ocupada?}(p, c) \iff (e.\text{mapa}[f])[c]))$

**2.3.5. Algoritmos**


---

```

1: function iCAMPUS(in e: estr)  $\longrightarrow$  res : campus  $\triangleright \mathcal{O}(1)$ 
2:   res  $\leftarrow$  e.campo
3: end function

```

---



---

```

1: function iESTUDIANTES(in e: estr)  $\longrightarrow$  res : itConj(nombre)  $\triangleright \mathcal{O}(1)$ 
2:   res  $\leftarrow$  crearIt (e.estudiantes)
3: end function

```

---



---

```

1: function iHIPPIES(in e: estr)  $\longrightarrow$  res : itConj(nombre)  $\triangleright \mathcal{O}(1)$ 
2:   res  $\leftarrow$  crearIt (e.hippies)
3: end function

```

---



---

```

1: function iAGENTES(in e: estr)  $\longrightarrow$  res : itConj(agente)  $\triangleright \mathcal{O}(1)$ 
2:   res  $\leftarrow$  claves(e.agentes)
3: end function

```

---



---

```

1: function iPOSESTUDIANTESYHIPPIE(in n: nombre in e: estr)  $\longrightarrow$  res : posicion  $\triangleright \mathcal{O}(n_m)$ 
2:   res  $\leftarrow$  obtener(n,e.posCiviles)
3: end function

```

---



---

```

1: function iPOSAGENTE(in a: agente in e: estr)  $\longrightarrow$  res : posicion  $\triangleright \mathcal{O}(1)(promedio)$ 
2:   res  $\leftarrow$  obtener(a,e.agentes).posActual
3: end function

```

---



---

```

1: function iCANTSANCIONES(in a: agente in e: estr)  $\longrightarrow$  res : nat  $\triangleright \mathcal{O}(1)(promedio)$ 
2:   res  $\leftarrow$  obtener(a,e.agentes).Qsanciones
3: end function

```

---



---

```

1: function iCANTHIPPIESATRAPADOS(in a: agente in e: estr)  $\longrightarrow$  res : nat  $\triangleright \mathcal{O}(1)(promedio)$ 
2:   res  $\leftarrow$  obtener(a,e.agentes).premios
3: end function

```

---



---

```

1: function iMASVIGILANTE(in r: rastr)  $\longrightarrow$  res : agente  $\triangleright \mathcal{O}(1)$ 
2:   res  $\leftarrow$  siguiente(e.masVigilante)
3: end function

```

---



---

```

1: function iCONMISMASANCIONES(in r: rastr in a: agente)  $\longrightarrow$  res : conj(agente)  $\triangleright \mathcal{O}(1)$ 
2:   res  $\leftarrow$  siguiente(obtener(e.agentes,a).grupoSanciones)
3: end function

```

---

---

```

1: function iCONKSANCIONES(in r: rastr in k: nat)  $\rightarrow$  res : conj(agente)     $\triangleright \mathcal{O}(n)$  la primera vez, luego mientras no
   haya sanciones  $\mathcal{O}(\log n)$ 
2:   if  $\neg$ e.huboNuevas then                                                     $\triangleright \mathcal{O}(1)$ 
3:     var nat i  $\leftarrow$  BusquedaBin(e.buscoEnLog, k)                           $\triangleright \mathcal{O}(\log n)$ 
4:     res  $\leftarrow$  e.buscoEnLog[i].grupoK                                        $\triangleright \mathcal{O}(1)$ 
5:   else
6:     var itLista(datosK) itK  $\leftarrow$  crearIt(e.agregoEn1)                     $\triangleright \mathcal{O}(1)$ 
7:     while haySiguiente(it) do                                               $\triangleright \mathcal{O}(1)$ 
8:       buscoEnLog[i]  $\leftarrow$  siguiente(itK)                                   $\triangleright \mathcal{O}(1)$ 
9:       avanzar(itK)
10:    end while
11:    var nat i  $\leftarrow$  BusquedaBin (e.buscoEnLog, k)                           $\triangleright \mathcal{O}(\log n)$ 
12:    res  $\leftarrow$  e.buscoEnLog[i].grupoK                                        $\triangleright \mathcal{O}(1)$ 
13:    e.hayNuevas  $\leftarrow$  false                                               $\triangleright \mathcal{O}(1)$ 
14:  end if
15: end function

```

---



---

```

1: function iINGRESARHIPPIE(in/out r: rastr in p: posicion in h: nombre: )  $\triangleright \mathcal{O}(|Nm|)$ 
2:   definir(e.posCiviles, h,p)                                               $\triangleright \mathcal{O}(|Nm|)$ 
3:   var itDicc(nombre,posicion) iterPos  $\leftarrow$  definirRapido(e.posRapida,h,p)  $\triangleright \mathcal{O}(1)$ 
4:   e.quienOcupa[p.X] [p.Y]  $\leftarrow$  < true, "hippie", crearIt(), agregarRapido(e.hippies,<h,iterPos>)>  $\triangleright \mathcal{O}(1)$ 
5:   var conj(posicion) Ps  $\leftarrow$  vecinos(e.campus,p)                          $\triangleright \mathcal{O}(1)$ 
6:   var itConj(posicion)  $\leftarrow$  crearIt(ps)                                    $\triangleright \mathcal{O}(1)$ 
7:   if esCapturable(e,p) then                                               $\triangleright \mathcal{O}(1)$ 
8:     capturarHippie(e,p)                                                   $\triangleright \mathcal{O}(|Nm|)$ 
9:   else
10:    while haySiguiente(it) do                                               $\triangleright \mathcal{O}(1)$ 
11:      if ocupada(e.campus, siguiente(it))  $\vee \neg$ e.quienOcupa[siguiente(it).X] [siguiente(it).Y].ocupada? then  $\triangleright \mathcal{O}(1)$ 
12:        avanzar(it)                                                        $\triangleright \mathcal{O}(1)$ 
13:      else
14:        if esEstudiante(e,siguiente(it))  $\wedge$  esHippizable(e,siguiente(it)) then  $\triangleright \mathcal{O}(1)$ 
15:          Hippizar(e, siguiente(it))                                        $\triangleright \mathcal{O}(1)$ 
16:          if esCapturable(e,siguiente(it)) then                           $\triangleright \mathcal{O}(1)$ 
17:            capturarHippie(e,siguiente(it))                                $\triangleright \mathcal{O}(|Nm|)$ 
18:          end if
19:        else
20:          if esEstudiante(e,siguiente(it))  $\wedge$  esCapturable(e,siguiente(it)) then  $\triangleright \mathcal{O}(1)$ 
21:            var itConj(posicion) itAg  $\leftarrow$  vecinos(e.campus, siguiente(it))  $\triangleright \mathcal{O}(1)$ 
22:            while haySiguiente(itAg) do  $\triangleright \mathcal{O}(1)$ 
23:              if esAgente(siguiente(itAg)) then  $\triangleright \mathcal{O}(1)$ 
24:                sancionar(e,siguiente(itAg))  $\triangleright \mathcal{O}(1)$ 
25:              end if
26:              avanzar(itAg)
27:            end while
28:          end if
29:        end if
30:      end if
31:      avanzar(it)
32:    end while
33:  end if
34: end function

```

---

---

```

1: function iMOVERESTUDIANTE(in/out r: rastr, in d: direccion, in s: estudiante) ▷  $\mathcal{O}(|Nm|)$ 
2:   var posicion actual ← obtener(e.posCiviles,s) ▷  $\mathcal{O}(1)$ 
3:   var posicion prx ← proxPosicion(e.campus, d, actual) ▷  $\mathcal{O}(1)$ 
4:   if seFue?(e.campus,actual, prx) then
5:     borrar(e.posCiviles, s) ▷  $\mathcal{O}(|Nm|)$ 
6:     var itConj(datosHoE) dat ← copia(e.quienOcupa[actual.X] [actual.Y].hayHoE) ▷  $\mathcal{O}(1)$ 
7:     eliminarSiguiente(dat, posActual) ▷  $\mathcal{O}(1)$ 
8:     eliminarSiguiente(dat) ▷  $\mathcal{O}(1)$ 
9:     e.quienOcupa[actual.X] [actual.Y] ← < false, "nada", crearIt(), crearIt() > ▷  $\mathcal{O}(1)$ 
10:  else
11:    var itConj(datosHoE) iterAHOI ← copia(e.quienOcupa[actual.X] [actual.Y].hayHoE) ▷  $\mathcal{O}(1)$ 
12:    eliminarSiguiente(siguiente(iterAHOI).posActual) ▷  $\mathcal{O}(1)$ 
13:    siguiente(iterAHOI).posActual ← definirRapido(e.posRapida,s,prx) ▷  $\mathcal{O}(1)$ 
14:    e.quienOcupa[prx.X] [prx.Y] ← <true, "estudiante", crearIt(), iterAHOI> ▷  $\mathcal{O}(1)$ 
15:    e.quienOcupa[actual.X] [actual.Y] ← <false, "nada", crearIt(), crearIt()> ▷  $\mathcal{O}(1)$ 
16:    definir(e.posCiviles, s, prx)  $\mathcal{O}(|Nm|)$ 
17:    var conj(posicion) vc ← vecinos(e.campus, prx) ▷  $\mathcal{O}(1)$ 
18:    var itConj(posicion) it ← crearIt(vc) ▷  $\mathcal{O}(1)$ 
19:    if esHippizable(e,prx) then ▷  $\mathcal{O}(1)$ 
20:      hippizar(e, prx) ▷  $\mathcal{O}(1)$ 
21:      while haySiguiente(it) do ▷  $\mathcal{O}(1)$ 
22:        if ocupada(e.campus, siguiente(it)) ∨ ¬e.quienOcupa[siguiente(it).X] [siguiente(it).Y].ocupada? then
23:          avanzar(it)
24:        else
25:          if esEstudiante(e,siguiente(it)) ∧ esHippizable(e,siguiente(it)) then ▷  $\mathcal{O}(1)$ 
26:            Hippizar(e, siguiente(it)) ▷  $\mathcal{O}(1)$ 
27:            if esCapturable(e, siguiente(it)) then ▷  $\mathcal{O}(1)$ 
28:              capturarHippie(e,siguiente(it)) ▷  $\mathcal{O}(|Nm|)$ 
29:            end if
30:          else
31:            if esEstudiante(e,siguiente(it)) ∧ esCapturable(e,siguiente(it)) then ▷  $\mathcal{O}(1)$ 
32:              var itConj(posicion) itAg ← vecinos(e.campus, siguiente(it)) ▷  $\mathcal{O}(1)$ 
33:              while haySiguiente(itAg) do ▷  $\mathcal{O}(1)$ 
34:                if esAgente(e,siguiente(itAg)) then ▷  $\mathcal{O}(1)$ 
35:                  sancionar(e,siguiente(itAg)) ▷  $\mathcal{O}(1)$ 
36:                end if
37:                avanzar(itAg)
38:              end while
39:            else
40:              if esHippie(e,siguiente(it)) ∧ esCapturable(s,siguiente(it)) then ▷  $\mathcal{O}(1)$ 
41:                capturarHippie(e,siguiente(it)) ▷  $\mathcal{O}(|Nm|)$ 
42:              end if
43:            end if
44:          end if
45:        end if
46:      end while
47:    else
48:      while haySiguiente(it) do
49:        if ocupada(e.campus, siguiente(it)) ∨ ¬e.quienOcupa[siguiente(it).X] [siguiente(it).Y].ocupada? then ▷
50:          avanzar(it)
51:        else
52:          if esHippie(e, siguiente(it)) ∧ esEstudiantizable(e,siguiente(it)) then ▷  $\mathcal{O}(1)$ 
53:            Estudiantizar(e, siguiente(it)) ▷  $\mathcal{O}(1)$ 
54:          else
55:            if esEstudiante(e,siguiente(it)) ∧ esCapturable(e, siguiente(it)) then ▷  $\mathcal{O}(1)$ 
56:              var itConj(posicion) itAg2 ← vecinos(e.campus, siguiente(it)) ▷  $\mathcal{O}(1)$ 
57:              while haySiguiente(itAg2) do ▷  $\mathcal{O}(1)$ 
58:                if esAgente(e,siguiente(itAg2)) then ▷  $\mathcal{O}(1)$ 
59:                  sancionar(e,siguiente(itAg2)) ▷  $\mathcal{O}(1)$ 
60:                end if
61:                avanzar(itAg2)
62:              end while 16/??
63:            else
64:              if esHippie(e, siguiente(it)) ∧ esCapturable(e,siguiente(it)) then ▷  $\mathcal{O}(1)$ 

```

---



---

```

1: function iMOVERAGENTE(in/out r: rastr in a: agente)  $\triangleright \mathcal{O}(|Nm|) + \mathcal{O}(\log Na) + \mathcal{O}(Ne)$ 
2:   var nat j  $\leftarrow$  BusquedaBin(e.AgentesLog,a)  $\triangleright \mathcal{O}(\log n)$ 
3:   var posicion actual  $\leftarrow$  e.AgentesLog[j]  $\triangleright \mathcal{O}(1)$ 
4:   var direccion d  $\leftarrow$  proxPosicionA(e,a)  $\triangleright \mathcal{O}(Ne)$ 
5:   var posicion prx  $\leftarrow$  proxPosicion(e.campus, d, actual)  $\triangleright \mathcal{O}(1)$ 
6:   var datosAg datAux  $\leftarrow$  obtener(e.agentes, a)  $\triangleright \mathcal{O}(1)$ 
7:   datAux.posActual  $\leftarrow$  prx  $\triangleright \mathcal{O}(1)$ 
8:   var itDicc(placa,datosAg) itA  $\leftarrow$  copia(e.quienOcupa[actual.X] [actual.Y].hayCana)  $\triangleright \mathcal{O}(1)$ 
9:   e.quienOcupa[actual.X] [actual.Y]  $\leftarrow$  <false, "nadie", crearIt(), crearIt(>  $\triangleright \mathcal{O}(1)$ 
10:  e.quienOcupa[prx.X] [prx.Y]  $\leftarrow$  <true, "agente", itA, crearIt(>  $\triangleright \mathcal{O}(1)$ 
11:  var itConj(posicion)  $\leftarrow$  crearIt(vecinos(e.campus, prx))  $\triangleright \mathcal{O}(1)$ 
12:  while haySiguiente(it) do  $\triangleright \mathcal{O}(1)$ 
13:    if ocupada(e.campus, siguiente(it))  $\vee$   $\neg$ e.quienOcupa[siguiente(it).X] [siguiente(it).Y].ocupada? then
14:      avanzar(it)
15:    else
16:      if esEstudiante(e, siguiente(it))  $\wedge$  esCapturable(e, siguiente(it)) then  $\triangleright \mathcal{O}(1)$ 
17:        var itConj(posicion) itAg  $\leftarrow$  vecinos(e.campus, siguiente(it))  $\triangleright \mathcal{O}(1)$ 
18:        while haySiguiente(itAg) do  $\triangleright \mathcal{O}(1)$ 
19:          if esAgente(e,siguiente(itAg)) then  $\triangleright \mathcal{O}(1)$ 
20:            sancionar(e, siguiente(itAg))  $\triangleright \mathcal{O}(1)$ 
21:          end if
22:          avanzar(itAg)
23:        end while
24:      else
25:        if esHippie(e,siguiente(it))  $\wedge$  esCapturable(e,siguiente(it)) then  $\triangleright \mathcal{O}(1)$ 
26:          capturarHippie(e, siguiente(it))  $\triangleright \mathcal{O}(|Nm|)$ 
27:        end if
28:      end if
29:    end if
30:    avanzar(it)
31:  end while
32: end function=0

```

---



---

```

1: function iESESTUDIANTE(in r: rastr in p: posicion)  $\rightarrow$  res : bool  $\triangleright \mathcal{O}(1)$ 
2:   res  $\leftarrow$  e.quienOcupa[p.X] [p.Y] == ".estudiante"
3: end function

```

---



---

```

1: function iESHIPPIE(in r: rastr in p: posicion)  $\rightarrow$  res : bool  $\triangleright \mathcal{O}(1)$ 
2:   res  $\leftarrow$  e.quienOcupa[p.X] [p.Y] == "hippie"
3: end function

```

---



---

```

1: function iESAGENTE(in r: rastr in p: posicion)  $\rightarrow$  res : bool  $\triangleright \mathcal{O}(1)$ 
2:   res  $\leftarrow$  e.quienOcupa[p.X] [p.Y] == ".agente"
3: end function

```

---



---

```

1: function iESTUDIANTIZAR(in/out r: rastr in p: posicion)  $\triangleright \mathcal{O}(1)$ 
2:   var datosHoE dat  $\leftarrow$  <Siguiente(e.quienOcupa[p.X] [p.Y].hayHoE).ID, Siguiente(e.quienOcupa[p.X] [p.Y].hayHoe).posActual>  $\triangleright \mathcal{O}(1)$ 
3:   eliminarSiguiente(e.quienOcupa[p.X] [p.Y].hayHoe)  $\triangleright \mathcal{O}(1)$ 
4:   var itConj(nombre) it  $\leftarrow$  agregarRapido(e.estudiantes, dat)  $\triangleright \mathcal{O}(1)$ 
5:   e.quienOcupa[p.X] [p.Y]  $\leftarrow$  <true, ".estudiante", crearIt(), it>  $\triangleright \mathcal{O}(1)$ 
6: end function

```

---

---

```

1: function iHIPPIZAR(in/out r: rastr in p: posicion) ▷  $\mathcal{O}(1)$ 
2:   var datosHoE dat ← <Siguiente(e.quienOcupa[p.X] [p.Y].hayHoE).ID, Siguiente(e.quienOcupa[p.X]
   [p.Y].hayHoe).posActual> ▷  $\mathcal{O}(1)$ 
3:   eliminarSiguiente(e.quienOcupa[p.X] [p.Y].hayHoe) ▷  $\mathcal{O}(1)$ 
4:   var itConj(nombre) it ← agregarRapido(e.hippies, dat) ▷  $\mathcal{O}(1)$ 
5:   e.quienOcupa[p.X] [p.Y] ← <true, "hippie", crearIt(), it> ▷  $\mathcal{O}(1)$ 
6: end function

```

---



---

```

1: function iESCAPTURABLE(in r: rastr in p: posicion) → res : bool ▷  $\mathcal{O}(1)$ 
2:   var itConj(posicion) it ← crearIt(vecinos(e.campus, p)) ▷  $\mathcal{O}(1)$ 
3:   var nat Contador ← 0 ▷  $\mathcal{O}(1)$ 
4:   bool hayGuardia ← false ▷  $\mathcal{O}(1)$ 
5:   while haySiguiente(it) do
6:     if e.quienOcupa[p.X] [p.Y].ocupada? then ▷  $\mathcal{O}(1)$ 
7:       contador+ + ▷  $\mathcal{O}(1)$ 
8:     end if
9:     if e.quienOcupa[p.X] [p.Y].quienOcupa == ".agente" then ▷  $\mathcal{O}(1)$ 
10:      hayGuardia ← true ▷  $\mathcal{O}(1)$ 
11:      avanzar(it)
12:
13:     res ← contador == 4 ∧ hayGuardia
14:

```

---



---

```

1: function iHIPPIZAR(in/out r: rastr in p: posicion) → res : bool ▷  $\mathcal{O}(1)$ 
2:   var itConj(posicion) it ← crearIt(vecinos(e.campus, p)) ▷  $\mathcal{O}(1)$ 
3:   var nat Contador ← 0 ▷  $\mathcal{O}(1)$ 
4:   while haySiguiente(it) do ▷  $\mathcal{O}(1)$ 
5:     if e.quienOcupa[siguiente(it).X] [siguiente(it).Y].quienOcupa == "hippie" then ▷  $\mathcal{O}(1)$ 
6:       contador + + ▷  $\mathcal{O}(1)$ 
7:     end if
8:     avanzar(it)
9:   end while
10:   res ← contador ≥ 2 ▷  $\mathcal{O}(1)$ 
11: end function=0

```

---



---

```

1: function iCAPTURARHIPPIE(in/out r: rastr in p: posicion) ▷  $\mathcal{O}(|Nm|)$ 
2:   var nombre n ← siguiente(e.quienOcupa[p.X] [p.Y].hayHoE).ID ▷  $\mathcal{O}(1)$ 
3:   eliminarSiguiente(siguiente(e.quienOcupa[p.X] [p.Y].hayHoE).posActual) ▷  $\mathcal{O}(1)$ 
4:   eliminarSiguiente(siguiente(e.quienOcupa[p.X] [p.Y].hayHoE)) ▷  $\mathcal{O}(1)$ 
5:   borrar(n, e.posCiviles) ▷  $\mathcal{O}(|Nm|)$ 
6:   e.quienOcupa[p.X] [p.Y] ← <false, nadie, crearIt(), crearIt() > ▷  $\mathcal{O}(1)$ 
7:   while haySiguiente(it) do ▷  $\mathcal{O}(1)$ 
8:     if e.quienOcupa[siguiente(it).X] [siguiente(it).Y].quienOcupa == ".agente" then ▷  $\mathcal{O}(1)$ 
9:       recompensar(e, siguiente(it)) ▷  $\mathcal{O}(1)$ 
10:    end if
11:    avanzar(it)
12:  end while
13: end function

```

---

---

1: <b>function</b> <i>i</i> TODASOCUPADAS( <b>in</b> <i>r</i> : <i>rastr</i> , <b>in</b> <i>p</i> : <i>conj</i> ( <i>posicion</i> )) res:bool	▷ $\mathcal{O}(1)$
2:   var itConj( <i>posicion</i> ) it ← crearIt( <i>p</i> )	▷ $\mathcal{O}(1)$
3:   var contador ← 0	▷ $\mathcal{O}(1)$
4: <b>while</b> haySiguiente(it) <b>do</b>	
5: <b>if</b> e.quienOcupa[ <i>siguiente</i> (it). <i>X</i> ] [ <i>siguiente</i> (it). <i>Y</i> ].ocupada? <b>then</b>	▷ $\mathcal{O}(1)$
6:       contador + +	▷ $\mathcal{O}(1)$
7: <b>end if</b>	
8: <b>end while</b>	
9:   res ← contador == 4	
10: <b>end function</b>	

---

---

```

1: function ¡INGRESARESTUDIANTE(in/out r: rastr, in n: nombre, in p: posicion)  $\triangleright \mathcal{O}(|Nm|)$ 
2:   if esHippizable(e,p) then  $\triangleright \mathcal{O}(1)$ 
3:     if esCapturable(e,p) then  $\triangleright \mathcal{O}(1)$ 
4:       var conj(posicion) v  $\leftarrow$  vecinos(e.campus, p)  $\triangleright \mathcal{O}(1)$ 
5:       var itConj(posicion) it  $\leftarrow$  crearIt(v)  $\triangleright \mathcal{O}(1)$ 
6:       while haySiguiente(it) do
7:         if e.quienOcupa[siguiente(it).X] [siguiente(it).Y].queHay == .agente then  $\triangleright \mathcal{O}(1)$ 
8:           recompensar(e, siguiente(it))  $\triangleright \mathcal{O}(1)$ 
9:         end if
10:        avanzar(it)
11:      end while
12:     else
13:       definir(e.posCiviles, n, p)  $\triangleright \mathcal{O}(|Nm|)$ 
14:       var itDicc(nombre, posicion) iterPos  $\leftarrow$  definirRapido(e.posRapida,n,p)  $\triangleright \mathcal{O}(1)$ 
15:       e.quienOcupa[p.X] [p.Y]  $\leftarrow$  < true,"hippie",crearIt(), agregarRapido(e.hippies,<n,iterPos>)>  $\triangleright \mathcal{O}(1)$ 
16:       var conj(posicion) Ps  $\leftarrow$  vecinos(e.campus, p)  $\triangleright \mathcal{O}(1)$ 
17:       var itConj(posicion) it  $\leftarrow$  crearIt(Ps)  $\triangleright \mathcal{O}(1)$ 
18:       while haySiguiente(it) do
19:         if esEstudiante(e,siguiente(it))  $\wedge$  esHippizable(e,siguiente(it)) then  $\triangleright \mathcal{O}(1)$ 
20:           Hippizar(e, siguiente(it))  $\triangleright \mathcal{O}(1)$ 
21:           if esCapturable(e,siguiente(it)) then  $\triangleright \mathcal{O}(1)$ 
22:             capturarHippie(e,siguiente(it))  $\triangleright \mathcal{O}(|Nm|)$ 
23:           end if
24:         else
25:           if esEstudiante(e,siguiente(it))  $\wedge$  esCapturable(e, siguiente(it)) then  $\triangleright \mathcal{O}(1)$ 
26:             var itConj(posicion) itAg  $\leftarrow$  vecinos(e.campus, siguiente(it))  $\triangleright \mathcal{O}(1)$ 
27:             while haySiguiente(itAg) do  $\triangleright \mathcal{O}(1)$ 
28:               if esAgente(e, siguiente(itAg)) then  $\triangleright \mathcal{O}(1)$ 
29:                 sancionar(e,siguiente(itAg))  $\triangleright \mathcal{O}(1)$ 
30:               end if
31:               avanzar(itAg)
32:             end while
33:           else
34:             if esHippie(e, siguiente(it))  $\wedge$  esCapturable(e,siguiente(it)) then  $\triangleright \mathcal{O}(1)$ 
35:               capturarHippie(e,siguiente(it))  $\triangleright \mathcal{O}(|Nm|)$ 
36:             end if
37:           end if
38:         end if
39:         avanzar(it)
40:       end while
41:     end if
42:   else
43:     definir(e.posCiviles, n, p)  $\triangleright \mathcal{O}(|Nm|)$ 
44:     var itDicc(nombre, posicion) iterPos  $\leftarrow$  definirRapido(e.posRapida, n, p)  $\triangleright \mathcal{O}(1)$ 
45:     e.quienOcupa[p.X] [p.Y]  $\leftarrow$  < true,"estudiante",crearIt(), agregarRapido(e.estudiantes,<n,iterPos>)>  $\triangleright \mathcal{O}(1)$ 
46:     var conj(posicion) Ps  $\leftarrow$  vecinos(e.campus, p)  $\triangleright \mathcal{O}(1)$ 
47:     var itConj(posicion) it  $\leftarrow$  crearIt(ps)  $\triangleright \mathcal{O}(1)$ 
48:     while haySiguiente(it) do
49:       if esHippie(e,siguiente(it))  $\wedge$  esEstudiantizable(e,siguiente(it)) then  $\triangleright \mathcal{O}(1)$ 
50:         Estudiantizar(e,siguiente(it))  $\triangleright \mathcal{O}(1)$ 
51:       else
52:         if esEstudiante(e,siguiente(it))  $\wedge$  esCapturable(e,siguiente(it)) then  $\triangleright \mathcal{O}(1)$ 
53:           var itConj(posicion) itAg  $\leftarrow$  vecinos(e.campus, siguiente(it))  $\triangleright \mathcal{O}(1)$ 
54:           while haySiguiente(itAg) do
55:             if esAgente(e,siguiente(itAg)) then  $\triangleright \mathcal{O}(1)$ 
56:               Sancionar(e,siguiente(itAg))  $\triangleright \mathcal{O}(1)$ 
57:             end if
58:             avanzar(itAg)
59:           end while
60:         else
61:           if esHippie(e,siguiente(it)  $\wedge$  esCapturable(e,siguiente(it)) then  $\triangleright \mathcal{O}(1)$ 
62:             capturarHippie(e,siguiente(it))  $\triangleright \mathcal{O}(|Nm|)$ 
63:           end if
64:         end if
65:       end if

```

**function**

(  
)

---

**function** *iRECOMPENSAR*(**in/out** *r: rastr*, **in** *a: posicion*)  $\triangleright \mathcal{O}(1)$   
 var placa p  $\leftarrow$  siguienteClave(quienOcupa<sub>[a.X]</sub> [a.Y].hayCana  $\triangleright \mathcal{O}(1)$   
 var datosAgente dat  $\leftarrow$  obtener(e.agentes,p)  $\triangleright \mathcal{O}(1)$   
 dat.premios  $\leftarrow$  dat.premios+1  $\triangleright \mathcal{O}(1)$   
**if** dat.premios  $\geq$  obtener(e.agentes, siguienteClave(e.masVigilante)).premios **then**  $\triangleright \mathcal{O}(1)$   
   e.masVigilante  $\leftarrow$  quienOcupa<sub>[a.X]</sub> [a.Y].hayCana  $\triangleright \mathcal{O}(1)$   
**else**  
   **if** dat.premios == obtener(e.agentes, siguienteClave(e.masVigilante)).premios **then**  $\triangleright \mathcal{O}(1)$   
     **if** p  $\neq$  siguienteClave(e.masVigilante) **then**  $\triangleright \mathcal{O}(1)$   
       e.masVigilante  $\leftarrow$  quienOcupa<sub>[a.X]</sub> [a.Y].hayCana  $\triangleright \mathcal{O}(1)$   
     **end if**  
   **end if**  
**end if**  
**end function**

---

=0

---

**function** *iSANCIONAR*(**in/out** *r: rastr*, **in** *a: posicion*)  $\triangleright \mathcal{O}(1)$   
 var placa p  $\leftarrow$  siguienteClave(quienOcupa<sub>[a.X]</sub> [a.Y].hayCana  $\triangleright \mathcal{O}(1)$   
 var datosAgente dat  $\leftarrow$  obtener(e.agentes,p)  $\triangleright \mathcal{O}(1)$   
 dat.Qsanciones  $\leftarrow$  dat.Qsanciones+1  $\triangleright \mathcal{O}(1)$   
 eliminarSiguiente(dat.grupoSanciones)  $\triangleright \mathcal{O}(1)$   
 avanzar(dat.verK)  $\triangleright \mathcal{O}(1)$   
 e.hayNuevas  $\leftarrow$  true  $\triangleright \mathcal{O}(1)$   
**if** Siguiente(dat.verK).K == dat.Qsanciones **then**  $\triangleright \mathcal{O}(1)$   
   dat.grupoSanciones  $\leftarrow$  Agregar(siguiente(dat.verK).grupoK, p)  $\triangleright \mathcal{O}(1)$   
**else**  
   dat.grupoSanciones  $\leftarrow$  AgregarComoAnterior(dat.verK,  $\downarrow$ dat.Qsanciones, Agregar(Vacio(),p);)  $\triangleright \mathcal{O}(1)$   
**end if**  
**end function**

---

---

1: <b>function</b> <i>iPROXPOSICIONH</i> ( <b>in/out</b> <i>e: rastr in h: nombre</i> )	$\triangleright \mathcal{O}(N_e)$
2:   var <i>itConj</i> ( <i>datosHoE</i> ) <i>it</i> ← <i>crearIt</i> ( <i>e.estudiantes</i> )	$\triangleright \mathcal{O}(1)$
3:   var <i>posicion menorD</i> ← <i>obtener</i> ( <i>e.posRapida</i> , <i>h</i> )	$\triangleright \mathcal{O}(1)$
4:   var <i>direccion direcc</i>	$\triangleright \mathcal{O}(1)$
5: <b>if</b> ( $\neg$ <i>haySiguiente</i> ( <i>it</i> )) <b>then</b>	$\triangleright \mathcal{O}(1)$
6: <b>if</b> ( <i>menorD.Y</i> ≤ <i>e.campus.filas</i> /2) <b>then</b>	$\triangleright \mathcal{O}(1)$
7: <b>if</b> ( $\neg$ <i>ocupadaD</i> ( <i>e,p,abajo</i> )) <b>then</b>	$\triangleright \mathcal{O}(1)$
8: <i>res</i> ← <i>Abajo</i>	
9: <b>else</b>	$\triangleright \mathcal{O}(1)$
10: <b>if</b> ( $\neg$ <i>ocupadaD</i> ( <i>e,p,derecha</i> )) <b>then</b>	$\triangleright \mathcal{O}(1)$
11: <i>res</i> ← <i>Derecha</i>	
12: <b>else</b>	$\triangleright \mathcal{O}(1)$
13: <b>if</b> ( $\neg$ <i>ocupadaD</i> ( <i>e,p,izquierda</i> )) <b>then</b>	$\triangleright \mathcal{O}(1)$
14: <i>res</i> ← <i>izquierda</i>	
15: <b>else</b>	$\triangleright \mathcal{O}(1)$
16: <i>res</i> ← <i>arriba</i>	$\triangleright \mathcal{O}(1)$
17: <b>end if</b>	
18: <b>end if</b>	
19: <b>end if</b>	
20: <b>else</b>	
21: <b>if</b> ( $\neg$ <i>ocupadaD</i> ( <i>e,p,Arriba</i> )) <b>then</b>	$\triangleright \mathcal{O}(1)$
22: <i>res</i> ← <i>Arriba</i>	
23: <b>else</b>	$\triangleright \mathcal{O}(1)$
24: <b>if</b> ( $\neg$ <i>ocupadaD</i> ( <i>e,p,derecha</i> )) <b>then</b>	$\triangleright \mathcal{O}(1)$
25: <i>res</i> ← <i>Derecha</i>	
26: <b>else</b>	$\triangleright \mathcal{O}(1)$
27: <b>if</b> ( $\neg$ <i>ocupadaD</i> ( <i>e,p,izquierda</i> )) <b>then</b>	$\triangleright \mathcal{O}(1)$
28: <i>res</i> ← <i>izquierda</i>	
29: <b>else</b>	$\triangleright \mathcal{O}(1)$
30: <i>res</i> ← <i>Abajo</i>	$\triangleright \mathcal{O}(1)$
31: <b>end if</b>	
32: <b>end if</b>	
33: <b>end if</b>	
34: <b>end if</b>	
35: <b>else</b>	
36: <i>menorD</i> ← <i>SiguienteSignificado</i> ( <i>siguiente</i> ( <i>it</i> ). <i>posActual</i> )	$\triangleright \mathcal{O}(1)$
37:     var <i>posicion otraPos</i>	$\triangleright \mathcal{O}(1)$
38: <b>while</b> <i>haySiguiente</i> ( <i>it</i> ) <b>do</b>	$\triangleright \mathcal{O}(N_e)$
39: <i>otraPos</i> ← <i>SiguienteSignificado</i> ( <i>siguiente</i> ( <i>it</i> ). <i>posActual</i> )	$\triangleright \mathcal{O}(1)$
40: <b>if</b> ( <i>distancia</i> ( <i>e,p,otraPos</i> ) < <i>distancia</i> ( <i>e,p,menorD</i> )) <b>then</b>	$\triangleright \mathcal{O}(1)$
41: <i>menorD</i> ← <i>otraPos</i>	$\triangleright \mathcal{O}(1)$
42: <b>end if</b>	
43: <b>end while</b>	
44: <i>res</i> ← <i>VecinoMasCercanoA</i> ( <i>e,p,menorD</i> )	$\triangleright \mathcal{O}(1)$
45: <b>end if</b>	
46: <b>end function</b>	

---

---

1: <b>function</b> <i>iPROXPOSICIONA</i> ( <b>in/out</b> <i>e: rastr in a: placa</i> )	$\triangleright \mathcal{O}(N_h)$
2:   var itConj(datosHoe) it $\leftarrow$ crearIt(e.hippies)	$\triangleright \mathcal{O}(1)$
3:   var posicion menorD $\leftarrow$ obtener(e.posRapida,a)	$\triangleright \mathcal{O}(1)$
4:   var direccion direcc	$\triangleright \mathcal{O}(1)$
5: <b>if</b> ( $\neg$ haySiguiente(it)) <b>then</b>	$\triangleright \mathcal{O}(1)$
6: <b>if</b> ( $\text{menorD.Y} \leq \text{e.campus.filas}/2$ ) <b>then</b>	$\triangleright \mathcal{O}(1)$
7: <b>if</b> ( $\neg$ ocupadaD(e,p,abajo)) <b>then</b>	$\triangleright \mathcal{O}(1)$
8:          res $\leftarrow$ Abajo	
9: <b>else</b>	$\triangleright \mathcal{O}(1)$
10: <b>if</b> ( $\neg$ ocupadaD(e,p,derecha)) <b>then</b>	$\triangleright \mathcal{O}(1)$
11:           res $\leftarrow$ Derecha	
12: <b>else</b>	$\triangleright \mathcal{O}(1)$
13: <b>if</b> ( $\neg$ ocupadaD(e,p,izquierda)) <b>then</b>	$\triangleright \mathcal{O}(1)$
14:            res $\leftarrow$ izquierda	
15: <b>else</b>	$\triangleright \mathcal{O}(1)$
16:            res $\leftarrow$ arriba	$\triangleright \mathcal{O}(1)$
17: <b>end if</b>	
18: <b>end if</b>	
19: <b>end if</b>	
20: <b>else</b>	
21: <b>if</b> ( $\neg$ ocupadaD(e,p,Arriba)) <b>then</b>	$\triangleright \mathcal{O}(1)$
22:       res $\leftarrow$ Arriba	
23: <b>else</b>	$\triangleright \mathcal{O}(1)$
24: <b>if</b> ( $\neg$ ocupadaD(e,p,derecha)) <b>then</b>	$\triangleright \mathcal{O}(1)$
25:          res $\leftarrow$ Derecha	
26: <b>else</b>	$\triangleright \mathcal{O}(1)$
27: <b>if</b> ( $\neg$ ocupadaD(e,p,izquierda)) <b>then</b>	$\triangleright \mathcal{O}(1)$
28:           res $\leftarrow$ izquierda	
29: <b>else</b>	$\triangleright \mathcal{O}(1)$
30:           res $\leftarrow$ Abajo	$\triangleright \mathcal{O}(1)$
31: <b>end if</b>	
32: <b>end if</b>	
33: <b>end if</b>	
34: <b>end if</b>	
35: <b>else</b>	
36:     menorD $\leftarrow$ siguiente(it)	$\triangleright \mathcal{O}(1)$
37:     var posicion otraPos	$\triangleright \mathcal{O}(1)$
38: <b>while</b> haySiguiente(it) <b>do</b>	$\triangleright \mathcal{O}(N_h)$
39:       otraPos $\leftarrow$ SiguienteSignificado(siguiente(it).posActual)	$\triangleright \mathcal{O}(1)$
40: <b>if</b> ( $\text{distancia}(\text{e,p,otraPos}) < \text{distancia}(\text{e,p,menorD})$ ) <b>then</b>	$\triangleright \mathcal{O}(1)$
41:          menorD $\leftarrow$ otraPos	$\triangleright \mathcal{O}(1)$
42: <b>end if</b>	
43: <b>end while</b>	
44:     res $\leftarrow$ VecinoMasCercanoA(e,p,menorD)	$\triangleright \mathcal{O}(1)$
45: <b>end if</b>	
46: <b>end function</b>	

---

---

```

1: function iVECINOMASCERCANO(in e: rastr in p: posicion in p2: posicion )  $\rightarrow$  res : direccion  $\triangleright \mathcal{O}(1)$ 
2:   var conj(posicion) Ps  $\leftarrow$  vecinos(e.campus, p)  $\triangleright \mathcal{O}(1)$ 
3:   var itConj(posicion) it  $\leftarrow$  crearIt(Ps)  $\triangleright \mathcal{O}(1)$ 
4:   var posicion destino  $\leftarrow$  siguiente(it)  $\triangleright \mathcal{O}(1)$ 
5:   while haySiguiente(it) do  $\triangleright \mathcal{O}(1)$ 
6:     if ( $\neg$ e.quienOcupa[siguiente(it).X][siguiente(it).Y].ocupada?) then  $\triangleright \mathcal{O}(1)$ 
7:       if (distancia(e,p,siguiente(it))<distancia(e,p,destino)) then  $\triangleright \mathcal{O}(1)$ 
8:         destino  $\leftarrow$  siguiente(it)  $\triangleright \mathcal{O}(1)$ 
9:       end if
10:    end if
11:    avanzar(it)  $\triangleright \mathcal{O}(1)$ 
12:  end while
13:  if (destino.X  $\neq$  p.X) then  $\triangleright \mathcal{O}(1)$ 
14:    if (destino.y > p.Y) then  $\triangleright \mathcal{O}(1)$ 
15:      res  $\leftarrow$  Arriba
16:    else  $\triangleright \mathcal{O}(1)$ 
17:      res  $\leftarrow$  Abajo  $\triangleright \mathcal{O}(1)$ 
18:    end if
19:  else
20:    if (destino.x > p.x) then  $\triangleright \mathcal{O}(1)$ 
21:      res  $\leftarrow$  Derecha
22:    else  $\triangleright \mathcal{O}(1)$ 
23:      res  $\leftarrow$  Izquierda  $\triangleright \mathcal{O}(1)$ 
24:    end if
25:  end if
26: end function

```

---



---

```

1: function iSEFUE(in e: rastr in p: posicion in destino: posicion )  $\rightarrow$  res : bool  $\triangleright \mathcal{O}(1)$ 
2:   res  $\leftarrow$  (p.Y == e.campus.alto - 1  $\wedge$  destino.y == e.campus.alto)  $\vee$  (p.Y == 0 destino.y == -1)  $\triangleright \mathcal{O}(1)$ 
3: end function

```

---



---

```

1: function iBUSQUEDABIN(in v: vector(datosK) in obj: nat)  $\rightarrow$  res : nat  $\triangleright \mathcal{O}(\log_2(\text{longitud}(v)))$ 
2:   var int i  $\leftarrow$  0  $\triangleright \mathcal{O}(1)$ 
3:   var int d  $\leftarrow$  longitud(v)-1  $\triangleright \mathcal{O}(1)$ 
4:   while (i + 1 < d) do  $\triangleright \mathcal{O}(\log_2(\text{longitud}(v)))$ 
5:     var int m  $\leftarrow$  (i+d)/2  $\triangleright \mathcal{O}(1)$ 
6:     if (v[m].k < obj) then  $\triangleright \mathcal{O}(1)$ 
7:       i  $\leftarrow$  m
8:     else  $\triangleright \mathcal{O}(1)$ 
9:       d  $\leftarrow$  m  $\triangleright \mathcal{O}(1)$ 
10:    end if
11:  end while
12:  if (v[i].k == obj) then  $\triangleright \mathcal{O}(1)$ 
13:    res  $\leftarrow$  i
14:  else  $\triangleright \mathcal{O}(1)$ 
15:    res  $\leftarrow$  d  $\triangleright \mathcal{O}(1)$ 
16:  end if
17: end function

```

---



### 3. Diseño del Tipo DICCIONARIOSTRING( $\sigma$ )

#### 3.1. Especificación

SE USA EL TAD DICCIONARIO( $\kappa, \sigma$ ) ESPECIFICADO EN EL APUNTE DE TADS BÁSICOS.

#### 3.2. Aspectos de la interfaz

##### 3.2.1. Interfaz

parámetros formales

género  $\kappa, \sigma$

función  $\bullet = \bullet(\text{in } a_1: \kappa, \text{in } a_2: \kappa) \rightarrow res: bool$

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

**Post**  $\equiv \{ res =_{\text{obs}} (a_1 = a_2) \}$

**Complejidad:**  $\Theta(equals(a_1, a_2))$

**Descripción:** función de igualdad de  $\kappa$ 's

función **COPIAR**(in  $k: \kappa$ )  $\rightarrow res: \kappa$

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

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

**Complejidad:**  $\Theta(copy(k))$

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

función **COPIAR**(in  $s: \sigma$ )  $\rightarrow res: \sigma$

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

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

**Complejidad:**  $\Theta(copy(s))$

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

Se explica con especificación de DICCIONARIO( $\kappa, \sigma$ ), ITERADOR BIDIRECCIONAL(TUPLA( $\kappa, \sigma$ ))

Género  $\text{diccString}(\kappa, \sigma)$

Operaciones básicas de diccionario

DEFINIDO?(in  $d: \text{diccString}(\kappa, \sigma)$ , in  $k: \kappa$ )  $\rightarrow res: bool$

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

**Post**  $\equiv \{ res =_{\text{obs}} \text{def?}(d, k) \}$

**Complejidad:**  $\mathcal{O}(|k|)$   $|k|$  es la longitud de la clave.

**Descripción:** DEVUELVE TRUE SI Y SÓLO SI  $k$  ESTÁ DEFINIDO EN EL DICCIONARIO.

OBTENER(in  $d: \text{diccString}(\kappa, \sigma)$ , in  $k: \kappa$ )  $\rightarrow res: \sigma$

**Pre**  $\equiv \{ \text{def?}(d, k) \}$

**Post**  $\equiv \{ \text{alias}(res =_{\text{obs}} \text{obtener}(d, k)) \}$

**Complejidad:**  $\mathcal{O}(|k|)$   $|k|$  es la longitud de la clave.

**Descripción:** DEVUELVE EL SIGNIFICADO DE LA CLAVE  $k$  EN  $d$ .

**Aliasing:** RES NO ES MODIFICABLE.

VACIO()  $\rightarrow res: \text{diccString}(\kappa, \sigma)$

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

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

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** GENERA UN DICCIONARIO VACÍO.

**DEFINIR**(**in/out**  $d: \text{diccString}(\kappa, \sigma)$ , **in**  $k: \kappa$ , **in**  $s: \sigma$ )

**Pre**  $\equiv \{ d =_{\text{obs}} d_0 \}$

**Post**  $\equiv \{ d =_{\text{obs}} \text{definir}(k, s, d_0) \}$

**Complejidad:**  $\mathcal{O}(|k|)$   $|k|$  es la longitud de la clave.

**Descripción:** DEFINE LA CLAVE  $k$  CON EL SIGNIFICADO  $s$  EN EL DICCIONARIO.

**BORRAR**(**in/out**  $d: \text{diccString}(\kappa, \sigma)$ , **in**  $k: \kappa$ )  $\rightarrow res: \text{bool}$

**Pre**  $\equiv \{ d =_{D_0} \wedge \text{def?}(k, d) \}$

**Post**  $\equiv \{ d =_{\text{obs}} \text{borrar}(k, D_0) \}$

**Complejidad:**  $\mathcal{O}(|k|)$   $|k|$  es la longitud de la clave.

**Descripción:** ELIMINA LA CLAVE  $k$  DEL DICCIONARIO.

### Operaciones básicas del iterador

**CREARIT**(**in**  $d: \text{diccString}(\kappa, \sigma)$ )  $\rightarrow res: \text{itdiccString}(\kappa, \sigma)$

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

**Post**  $\equiv \{ \text{alias}(\text{esPermutacion}(\text{SecuSuby}(res), d)) \wedge \text{vacía?}(\text{Anteriores}(res)) \}$

**Complejidad:**  $\mathcal{O}(n)$   $n$  es la cantidad de claves.

**Descripción:** CREA UN ITERADOR DEL DICCIONARIO DE FORMA TAL QUE SE PUEDAN RECORRER SUS ELEMENTOS APLICANDO ITERATIVAMENTE SIGUIENTE( NO PONEMOS LA OPERACION SIGUIENTE EN LA INTERFAZ PUES NO LA USAMOS).

**HAYSIGUIENTE**(**in**  $it: \text{itdiccString}(\kappa, \sigma)$ )  $\rightarrow res: \text{bool}$

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

**Post**  $\equiv \{ res =_{\text{obs}} \text{HaySiguiente?}(it) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** DEVUELVE TRUE SI Y SOLO SI EN EL ITERADOR QUEDAN ELEMENTOS PARA AVANZAR.

**SIGUIENTESIGNIFICADO**(**in**  $it: \text{itdiccString}(\kappa, \sigma)$ )  $\rightarrow res: \sigma$

**Pre**  $\equiv \{ \text{HaySiguiente?}(it) \}$

**Post**  $\equiv \{ \text{alias}(res =_{\text{obs}} \text{Siguiente}(it).significado) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** DEVUELVE EL SIGNIFICADO DEL ELEMENTO SIGUIENTE DEL ITERADOR.

**Aliasing:** RES NO ES MODIFICABLE.

**AVANZAR**(**in/out**  $it: \text{itdiccString}(\kappa, \sigma)$ )

**Pre**  $\equiv \{ it =_{\text{obs}} it_0 \wedge \text{HAYSIGUIENTE?}(it) \}$

**Post**  $\equiv \{ it =_{\text{obs}} \text{Avanzar}(it_0) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** AVANZA A LA POSICION SIGUIENTE DEL ITERADOR.

## 3.3. Pautas de implementación

### 3.3.1. Estructura de representación

$\text{diccString}(\kappa, \sigma)$  se representa con *puntero(nodo)*

**donde *nodo* es**

***tupla***(  
 SIGNIFICADO: ***Puntero***( $\sigma$ )  $\times$   
 CARACTERES: ***arreglo***[256] de ***puntero***(*nodo*)  $\times$   
 PADRE: ***Puntero***(*nodo*)  
 )

### 3.3.2. Justificación

### 3.3.3. Invariante de Representación

#### Informal

- TODAS LAS POSICIONES DEL ARREGLO DE CARACTERES ESTÁN DEFINIDAS.
- NO HAY CLAVES DE 0 CARACTERES. EL SIGNIFICADO DE LA RAÍZ ES NULL.
- NO HAY CICLOS EN LA ESTRUCTURA. ES DECIR, EXISTE UNA COTA SUPERIOR SOBRE LA CANTIDAD DE NIVELES POSIBLES DEL ÁRBOL.
- DADO UN NODO CUALQUIERA DEL TRIE, EXISTE UN ÚNICO CAMINO DESDE LA RAÍZ HASTA EL NODO.

#### Formal

REP : ESTR  $\longrightarrow$  *boolean*

( $\forall e : \text{ESTR}$ )

REP(*e*)  $\equiv$  (*true*  $\iff$

(1)( $\forall i : \text{nat}$ )( $i < 256 \Rightarrow \text{DEFINIDO?}(E \rightarrow \text{CARACTERES}, i)$ )  $\wedge_L$

(2)( $E \rightarrow \text{significado} = \text{NULL}$ )  $\wedge_L$

(2)( $\exists N : \text{NAT}$ )(FINALIZA(*E*, *N*))  $\wedge_L$

(3)( $\forall P, Q : \text{PUNTERO}(\text{NODO})$ )( $P \in \text{PUNTEROS}(E) \wedge q \in (\text{punteros}(e) - \{p\}) \Rightarrow P \neq q$ )  $\wedge_L$

)

### 3.3.4. Función de Abstracción

ABS : ROSETREE(ESTRDATO) *r*  $\longrightarrow$  DICC\_TRIE( $\sigma$ )

{REP(*r*)}

( $\forall r : \text{ROSETREE}(\text{ESTRDATO})$ ) ABS(*r*) =<sub>obs</sub> *d* : DICC\_TRIE( $\sigma$ ) /

( $\forall k : \text{SECU}(\text{letra})$ )(DEF?(*K*, *D*) =<sub>obs</sub> ESTA?(*K*, *R*))  $\wedge$  (DEF?(*C*, *D*)  $\Rightarrow$  (OBTENER(*K*, *D*) =<sub>obs</sub> BUSCAR(*K*, *R*)))

#### Funciones Auxiliares

## 3.3.5. Algoritmos

---

```

1: function iVACIO( )  $\rightarrow$  res : estr  $\triangleright \mathcal{O}(1)$ 
2:   var arreglo(puntero(nodo)) letras  $\leftarrow$  crearArreglo[256]
3:   for i  $\leftarrow$  0 to 255 do  $\triangleright \mathcal{O}(1)$ 
4:     letras[i]  $\leftarrow$  NULL  $\triangleright \mathcal{O}(1)$ 
5:   end for
6:   var nodo nuevo  $\leftarrow$  <NULL,letras,NULL>  $\triangleright \mathcal{O}(1)$ 
7:   res  $\leftarrow$  &nuevo  $\triangleright \mathcal{O}(1)$ 
8: end function

```

---



---

```

1: function iDEFINIR(in/out d: estr, in k: string, in s:  $\sigma$ )
2:   nat i  $\leftarrow$  0  $\triangleright \mathcal{O}(1)$ 
3:   puntero(nodo) actual  $\leftarrow$  d  $\triangleright \mathcal{O}(1)$ 
4:   while (i < |k|) do  $\triangleright \mathcal{O}(|k|)$ 
5:     if actual  $\rightarrow$  caracteres[ord(k[i])] = NULL then  $\triangleright \mathcal{O}(1)$ 
6:       puntero(nodo) anterior  $\leftarrow$  actual
7:       actual  $\rightarrow$  caracteres[ord(k[i])]  $\leftarrow$  iVacio()  $\triangleright \mathcal{O}(1)$ 
8:       actual  $\rightarrow$  padre  $\leftarrow$  anterior
9:     else
10:      actual  $\leftarrow$  (actual  $\rightarrow$  caracteres[ord(k[i])])  $\triangleright \mathcal{O}(1)$ 
11:    end if
12:    i  $\leftarrow$  i + 1  $\triangleright \mathcal{O}(1)$ 
13:  end while
14:  actual  $\rightarrow$  significado  $\leftarrow$  &copiar(s)  $\triangleright \mathcal{O}(1)$ 
15: end function

```

---



---

```

1: function iOBTENER(in d: estr, in k: string)  $\rightarrow$  res :  $\sigma$ 
2:   nat i  $\leftarrow$  0  $\triangleright \mathcal{O}(1)$ 
3:   puntero actual  $\leftarrow$  d  $\triangleright \mathcal{O}(1)$ 
4:   while i < |k| do  $\triangleright \mathcal{O}(|k|)$ 
5:     actual  $\leftarrow$  (actual  $\rightarrow$  caracteres[ord(k[i])])  $\triangleright \mathcal{O}(1)$ 
6:     i  $\leftarrow$  i + 1
7:   end while
8:   res  $\leftarrow$  *(actual  $\rightarrow$  significado)
9: end function

```

---



---

```

1: function iBORRAR(in/out d: estr, in k: string)
2:   puntero(nodo) actual  $\leftarrow$  d
3:   for i  $\leftarrow$  0 to |k|  $\triangleright \mathcal{O}(1)$ 
4:     actual  $\leftarrow$  (actual  $\rightarrow$  caracteres[ord(k[i])])  $\triangleright \mathcal{O}(1)$ 
5:   end for
6:   (actual  $\rightarrow$  significado)  $\leftarrow$  NULL var puntero(nodo) camino  $\leftarrow$  NULL
7:   while (actual  $\rightarrow$  significado = NULL) or todosNULL(actual  $\rightarrow$  caracteres) do  $\triangleright \mathcal{O}(|k|)$ 
8:     camino  $\leftarrow$  actual  $\triangleright \mathcal{O}(1)$ 
9:     actual  $\leftarrow$  (actual  $\rightarrow$  padre)
10:    delete camino
11:  end while

```

---

---

```

1: function IDEFINIDO?(in d: estr, in k: string)  $\longrightarrow$  res : bool
2:   nat i  $\leftarrow$  0  $\triangleright \mathcal{O}(1)$ 
3:   puntero actual  $\leftarrow$  d  $\triangleright \mathcal{O}(1)$ 
4:   bool def  $\leftarrow$  true  $\triangleright \mathcal{O}(1)$ 
5:   while (i < |k| and def) do  $\triangleright \mathcal{O}(|k|)$ 
6:     if actual  $\longrightarrow$  caracteres[ord(k[i])] = NULL then  $\triangleright \mathcal{O}(1)$ 
7:       def  $\leftarrow$  false  $\triangleright \mathcal{O}(1)$ 
8:     else
9:       actual  $\leftarrow$  actual  $\longrightarrow$  caracteres[ord(k[i])]  $\triangleright \mathcal{O}(1)$ 
10:      i  $\leftarrow$  i + 1  $\triangleright \mathcal{O}(1)$ 
11:    end if
12:  end while
13:  res  $\leftarrow$  def  $\wedge$   $\neg$ (actual  $\longrightarrow$  significado(NULL))  $\triangleright \mathcal{O}(1)$ 
14: end function=0

```

---

### 3.4. Servicios Usados

#### Requerimientos sobre el Tipo

- LA FUNCIÓN  $|x|$  DEBE TENER COMPLEJIDAD  $\mathcal{O}(1)$  EN EL CASO PEOR.
- LA FUNCIÓN  $|x|$  DEBE TENER COMPLEJIDAD  $\mathcal{O}(1)$  EN EL CASO PEOR.
- LAS OPERACIONES DEBEN REALIZARSE POR REFERENCIA.
- DEBE PROVEER UNA OPERACIÓN **Copia** QUE DEVUELVE UNA NUEVA INSTANCIA DE LA SECUENCIA PERO

QUE ES

INDEPENDIENTE DE LA ACTUAL, CON COMPLEJIDAD  $\mathcal{O}(n)$  EN EL CASO PEOR.

- DEBE PROVEER UN **iterador** PARA AVANZAR QUE COMIENZA EN EL PRIMERO ELEMENTO DE LA SECUENCIA.
- DEBE PROVEER UN **iterador** PARA RETROCEDER QUE COMIENZA EN EL ÚLTIMO ELEMENTO DE LA SE-

CUENCIA.

- LAS OPERACIONES **CrearIt**, **Siguiente**, **Anterior**, **TieneSiguiente**, **TieneAnterior** DEBEN TENER COM-

PLEJIDAD

$\mathcal{O}(1)$  EN EL CASO PEOR.

DONDE  $n$  ES LA LONGITUD DE LA PALABRA.

## 4. Diseño del Tipo DICCIONARIO<sub>PROM</sub>

### 4.1. Especificación

SE USA EL TAD DICCIONARIO<sub>M</sub> (NOTA AL CORRECTOR: LEER OBSERVACIONES).

### 4.2. Aspectos de la interfaz

#### 4.2.1. Interfaz

Se explica con especificación de DICCIONARIO<sub>M</sub>( $\kappa, \sigma$ )

Género  $\text{diccProm}(\kappa, \sigma)$

Operaciones básicas de diccionario

DEFINIDO?(**in**  $d: \text{diccProm}(\kappa, \sigma)$ , **in**  $k: \kappa$ )  $\longrightarrow res: \text{bool}$

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

**Post**  $\equiv \{ res =_{\text{obs}} \text{def?}(d, k) \}$

**Complejidad:**  $\mathcal{O}(n)$   $n$  es la cantidad de claves.

**Descripción:** DEVUELVE TRUE SI Y SÓLO SI  $k$  ESTÁ DEFINIDO EN EL DICCIONARIO.

OBTENER(**in**  $d: \text{diccProm}(\kappa, \sigma)$ , **in**  $k: \kappa$ )  $\longrightarrow res: \sigma$

**Pre**  $\equiv \{ \text{def?}(d, k) \}$

**Post**  $\equiv \{ alias(res =_{\text{obs}} \text{obtener}(d, k)) \}$

**Complejidad:**  $\mathcal{O}(n)$   $n$  es la cantidad de claves.

**Descripción:** DEVUELVE EL SIGNIFICADO DE LA CLAVE  $k$  EN  $d$ .

**Aliasing:** SE DEVUELVE UNA REFERENCIA AL SIGNIFICADO DE LA CLAVE.

CLAVES(**in**  $d: \text{diccProm}(\kappa, \sigma)$ )  $\longrightarrow res: \text{itConj}(\kappa)$

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

**Post**  $\equiv \{ res =_{\text{obs}} \text{claves}(d) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve el conjunto con las claves definidas en  $d$ .

VACIO(**in**  $n: \text{nat}$ )  $\longrightarrow res: \text{diccProm}(\kappa, \sigma)$

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

**Post**  $\equiv \{ res =_{\text{obs}} \text{vacio}(n) \}$

**Complejidad:**  $\mathcal{O}(n)$

**Descripción:** Genera un diccionario vacío, donde  $n$  acota superiormente a la cantidad de claves.

DEFINIR(**in/out**  $d: \text{diccProm}(\kappa, \sigma)$ , **in**  $k: \kappa$ , **in**  $s: \sigma$ )

**Pre**  $\equiv \{ d =_{\text{obs}} d_0 \}$

**Post**  $\equiv \{ d =_{\text{obs}} \text{definir}(k, s, d_0) \}$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Define la clave  $k$  con el significado  $s$  en el diccionario.

### 4.3. Pautas de implementación

#### 4.3.1. Estructura de representación

$\text{diccProm}(\kappa, \sigma)$  se representa con *estr*

**donde** *estr* es

*tupla*(  
*Cclaves*: *conjLineal*( $\kappa$ )  $\times$   
*clavesMax*: *nat*  $\times$   
*tabla*: *arreglo* de *lista*(*datos*)  
 )

**donde** *datos* es

*tupla*(  
*clave*:  $\kappa$   $\times$   
*significado*:  $\sigma$   
 )

#### 4.3.2. Justificación

#### 4.3.3. Invariante de Representación

**Informal**

- *clavesMax* es mayor que cero.
- La longitud del arreglo es igual a *clavesMax*.
- Todas las posiciones del arreglo estan definidas.
- Todos los elementos de *Cclaves* estan definidos en la tabla y viceversa.
- Todas las claves de la tabla estan definidos en *Cclaves*.

**Formal**

*Rep* : *estr*  $\longrightarrow$  boolean

( $\forall e : estr$ )

*Rep*(*e*)  $\equiv$  (*true*  $\iff$

(1) *e.clavesMax*  $> 0 \wedge_L$

(2) *longitud*(*e.tabla*)  $== e.clavesMax \wedge$

(3) ( $\forall i : nat$ )( $i \leq e.clavesMax \Rightarrow_L \text{definido?}(e.tabla, i)$ )  $\wedge$

(3) ( $\forall k : \kappa$ )( $k \in e.Cclaves \Rightarrow (\exists j : nat)(\text{estaEn?}(e.tabla[j], k))$ )  $\wedge$

(4) ( $\forall i : nat$ )( $\forall k : \kappa$ )( $i \leq e.clavesMax \wedge_L \text{estaEn?}(e.tabla[i], k) \Rightarrow k \in e.Ccclaves$ ) )

**Funciones Auxiliares**

*estaEn?* : *lista*(*datos*)  $\times \kappa \longrightarrow bool$

*estaEn?*(*l*, *k*)  $\equiv (\exists i : nat)(i < longitud(l) \Rightarrow_L l[i].clave == k)$

#### 4.3.4. Función de Abstracción

*Abs* : *estr* *e*  $\longrightarrow$  *DiccionarioProm*( $\kappa$ ,  $\sigma$ )

( $\forall e : estr$ ) *Abs*(*e*)  $=_{\text{obs}}$  *d* : *DiccionarioProm*( $\kappa$ ,  $\sigma$ ) /  
*cla*

{*Rep*(*e*)}

**Funciones Auxiliares**



## 4.3.5. Algoritmos

---

```

1: function IVACIO(in  $n: nat$ )  $\longrightarrow$   $res: estr$   $\triangleright \mathcal{O}(clavesMax)$ 
2:    $var$  arreglo(lista(datos))  $tabla \leftarrow$  crearArreglo[n]  $\triangleright \mathcal{O}(clavesMax)$ 
3:   for  $i \leftarrow 0$  to  $n$  do  $\triangleright \mathcal{O}(clavesMax)$ 
4:      $tabla[i] \leftarrow$  Vacía()  $\triangleright \mathcal{O}(1)$ 
5:   end for
6:    $res \leftarrow \langle n, tabla \rangle$   $\triangleright \mathcal{O}(1)$ 
7: end function

```

---



---

```

1: function IDEFINIR(in/out  $d: estr$ , in  $k: nat$ , in  $s: \sigma$ )  $\triangleright \mathcal{O}(1)$ 
2:    $nat\ i \leftarrow$  fHash( $k, e.clavesMax$ )  $\triangleright \mathcal{O}(1)$ 
3:    $e.tabla[i] \leftarrow$  AgregarAtras( $e.tabla[i], \langle k, s \rangle$ )  $\triangleright \mathcal{O}(1)$ 
4: end function

```

---



---

```

1: function IOBTENER(in  $d: estr$ , in  $k: nat$ )  $\longrightarrow$   $res: \sigma$   $\triangleright \mathcal{O}(longitud(tabla[i]))$ 
2:    $nat\ i \leftarrow$  fHash( $k, e.clavesMax$ )  $\triangleright \mathcal{O}(1)$ 
3:    $var\ itLista(datos)\ it \leftarrow$  crearIt( $tabla[i]$ )
4:   while haySiguiente(it) do
5:     if siguiente(it).clave =  $k$  then
6:        $res \leftarrow$  siguiente(it).significado
7:     end if
8:   end while
9: end function

```

---



---

```

1: function IDEFINIDO?(in  $d: estr$ , in  $k: nat$ )  $\longrightarrow$   $res: bool$   $\triangleright \mathcal{O}(longitud(tabla[i]))$ 
2:    $nat\ i \leftarrow$  fHash( $k, e.clavesMax$ )  $\triangleright \mathcal{O}(1)$ 
3:    $var\ itLista(datos)\ it \leftarrow$  crearIt( $tabla[i]$ )
4:    $bool\ aux \leftarrow false$ 
5:   while haySiguiente(it) do
6:     if siguiente(it).clave =  $k$  then
7:        $aux \leftarrow true$ 
8:     end if
9:   end while
10:   $res \leftarrow aux$ 
11: end function

```

---



---

```

1: function FHASH(in  $k: nat$ , in  $clavesMax: nat$ )  $\longrightarrow$   $res: nat$   $\triangleright \mathcal{O}(1)$ 
2:    $res \leftarrow k \bmod clavesMax$   $\triangleright \mathcal{O}(1)$ 
3: end function

```

---



---

```

1: function ICLAVES(in  $d: estr$ )  $\longrightarrow$   $res: itConj(\kappa)$   $\triangleright \mathcal{O}(clavesMax)$ 
2:    $res \leftarrow$  crearIt( $e.Cclaves$ )
3: end function

```

---

#### 4.4. **Servicios Usados**