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.
6. Asumimos a $|Nm|$ como la longitud más larga entre todos los nombres del campusSeguro, Na la cantidad de agentes y Ne la cantidad de estudiante en el momento donde será usado y Nh la cantidad de hippies, en el momento donde va a ser usado.
7. Por consigna, se desestiman los costos de eliminación de elementos, con lo cual se pueden ignorar en el cálculo de complejidades.

1. Diseño del Tipo Rastrillaje

1.1. Especificación

Se usa el Tad CampusSeguro especificado por la cátedra.

1.2. Aspectos de la interfaz

1.2.1. Interfaz

Se explica con especificación de CampusSeguro

Género `rastr`

Operaciones básicas de Rastrillaje

Campus(**in** r: $rastr$) $\longrightarrow res : campus$
Pre $\equiv$ { $true$ }
Post $\equiv$ { $res =_{\text{obs}} campus(r)$ }

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

Descripción: Devuelve el campus.

Estudiantes(**in** r: $rastr$) $\longrightarrow res : conj(nombre)$
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: $rastr$) $\longrightarrow res : conj(nombre)$
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: $rastr$) $\longrightarrow res : conj(agente)$
Pre $\equiv$ { $true$ }
Post $\equiv$ { $res =_{\text{obs}} agentes(r)$ }

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

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

PosEstudiantesYHippie(**in** r: $rastr$, **in** id: $nombre$) $\longrightarrow res : posicion$
Pre $\equiv$ { $id \in$ (estudiantes(r) $\cup$ hippies(cs))}
Post $\equiv$ { $res =_{\text{obs}} posEstudianteYHippie(id, r)$ }

Complejidad: $\mathcal{O}(|N_m|)$

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

PosAgente(**in** r: $rastr$, **in** a: $agente$) $\longrightarrow res : posicion$
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. La complejidad se da en el caso promedio.

CantSanciones(**in** r: $rastr$, **in** a: $agente$) $\longrightarrow res : nat$
Pre $\equiv$ { $a \in$ cantSanciones(a,r)}

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

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

Descripción: Devuelve la cantidad de sanciones recibidas por el agente pasado como parámetro. La complejidad se da en el caso promedio.

CantHippiesAtrapados(**in** r: $rastr$, **in** a: $agente$) $\longrightarrow res : nat$
Pre $\equiv$ { $a \in$ agentes(r)}
Post $\equiv$ { $res =_{obs} cantHippiesAtrapados(a, r)$ }

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

Descripción: Devuelve la cantidad de hippies atrapados por el agente pasado como parámetro. La complejidad se da en el caso promedio.

ComenzarRastrillaje(**in** c: $campus$, **in** d: $dicc(agente, posicion)$) $\longrightarrow res : rastr$
Pre $\equiv$ { $(\forall$ a : agente)(def?(a,d) $\Rightarrow_{\text{L}}$ (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 $\neq a_2$) $\Rightarrow_{\text{L}}$ obtener(a,d)$\neq$ obtener(a_2,d))}
Post $\equiv$ { $res =_{obs} comenzarRastrillaje(c, d)$ }

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

Descripción: Crea un Rastrillaje.

IngresarEstudiante(**in/out** r: $rastr$, **in** e: $nombre$, **in** p: $posicion$) $\longrightarrow$
Pre $\equiv$ { $r =$r$_0 \wedge$ e $\notin$ (estudiantes(r)$\cup$hippies(r)) $\wedge$ esIngreso?(p,campus(r)) $\wedge$ $\neg$estaOcupada?(p,r)}
Post $\equiv$ { $r =_{obs} ingresarEstudiante(e, p,r_0)$}

Complejidad: $\mathcal{O}(|\ Nm\ |)$

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

IngresarHippie(**in/out** r: $rastr$, **in** h: $nombre$, **in** p: $posicion$) $\longrightarrow$
Pre $\equiv$ { $r =$r$_0 \wedge$ h $\notin$ (estudiantes(r)$\cup$hippies(r)) $\wedge$ esIngreso?(p,campus(r)) $\wedge$ $\neg$estaOcupada?(p,r)}
Post $\equiv$ { $r =_{obs} ingresarHippie(h, p,r_0)$}

Complejidad: $\mathcal{O}(|Nm|)$

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

MoverEstudiante(**in/out** r: $rastr$, **in** e: $nombre$, **in** dir: $direccion$) $\longrightarrow$
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 $\equiv$ { $r =_{obs} moverEstudiante(e, d,r_0)$}

Complejidad: $\mathcal{O}(|Nm|)$

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

MoverHippie(**in/out** r: $rastr$, **in** h: $nombre$) $\longrightarrow$
Pre $\equiv$ { $r =$r$_0 \wedge$ h $\in$ hippies(r) $\wedge$ $\neg$todasOcupadas?(vecinos(posEstudianteYHippie(h,r),campus(r)),r) }
Post $\equiv$ { $r =_{obs} moverHippie(r,$r$_0)$}

Complejidad: $\mathcal{O}(|Nm|)$ + O(Ne)

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

MoverAgente(**in/out** r: $rastr$, **in** a: $agente$) $\longrightarrow$
Pre $\equiv$ { $r =$r$_0 \wedge$ a $\in$ agentes(r) $\wedge_{\text{L}}$ cantSanciones(a,r) $\leq$ 3 $\wedge$ $\neg$todasOcupadas?(vecinos(posAgente(a,r),campus(r)),r)}
Post $\equiv$ { $r =_{obs} moverAgente(a,$r$_0)$}

Complejidad: $\mathcal{O}(|Nm|)$ + O($logNa$) + O(Ne)

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

MasVigilante(**in** r: $rastr$) $\longrightarrow res : agente$
Pre $\equiv$ { $true$ }

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

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

Descripción: Devuelve el agente con mas capturas.

ConKSanciones(**in** r: $rastr$, **in** k: nat) $\longrightarrow res : conj(agente)$
Pre $\equiv$ { $true$ }
Post $\equiv$ { $res =_{obs} conKSanciones(k, r)$ }

Complejidad: $\mathcal{O}(Na)/\mathcal{O}(logNa)$

Descripción: Devuelve el agente con mas capturas. La primera vez que se llama será O(Na) luego mientras no haya sanciones, O($logNa$).

ConMismasSanciones(**in** r: $rastr$, **in** a: $agente$) $\longrightarrow res : conj(agente)$
Pre $\equiv$ { $a \in$ agentes(r)}
Post $\equiv$ { $res =_{obs} conMismasSanciones(a, r)$ }

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

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

1.3. Pautas de implementación

1.3.1. Estructura de representación

campus **se representa con** *estr*
donde *estr* **es**
tupla(
campo: ***campus*** ×
agentes: ***diccPromedio(agente ; datosAg)*** ×
posAgentesLog: ***arreglo(tupla(placa;posicion))*** ×
hippies: ***conjLineal(datosHoE)*** ×
estudiantes: ***conjLineal(datosHoE)*** ×
posCiviles: ***diccString(nombre;posicion)*** ×
posRapida: ***diccLineal(nombre;posicion)*** ×
quienOcupa: ***vector(vector(datosPos))*** ×
masVigilante: ***itConj(agente)*** ×
agregoEn1: ***lista(datosK))*** ×
hayNuevas: ***bool*** ×
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*** $\times$
grupoK: ***conjLineal(agente)***
)

1.3.2. Justificación

1.3.3. Invariante de Representación

Informal

1. Todos los agentes tienen distinta posicion.

2. La cantidad de sanciones se ve reflejada dos veces en la tupla DatosAg y debe ser la misma.

3. Si dos agentes tienen la misma cantidad de sanciones, pertenecen al mismo grupo. En caso contrario, sus grupos son disjuntos.

4. Todas las posiciones estan dentro del rango permitido en el campus.

5. El conjunto que contiene a todas las placas de posAgentesLog es igual al conjunto de claves de agentes.

6. Todas las posiciones de los agentes son los significados del diccionario .ªgentesƶ tambien se ven en "posAgentesLogƶ son las mismas.

7. La union de los gruposK pertenecientes a .ªgregoEn1.^{es} igual al conjunto de claves de agentes.

8.

Formal

Rep : estr $\longrightarrow$ *boolean*

($\forall$ e : estr)

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

(1)(2)(3)(4) ($\forall$ a,a2: Agente)(a $\neq$ a2 $\wedge$ definido?(a,e.agentes) $\wedge$ definido?(a2,e.agentes)
$\wedge_{L}$ PosValida(e.campo,obtener(a,e.agentes).PosActual) $\wedge$ PosValida(e.campo,obtener(a2,e.agentes).PosActual))) $\Rightarrow_{L}$ obtener(a,e.agentes).PosActual $\neq$ obtener(a2,e.agentes).PosActual
$\wedge$ (obtener(a,e.agentes).Qsanciones=siguiente(obtener(a,e.agentes).verK).K
$\wedge$ obtener(a,e.agentes).grupoSanciones=siguiente(obtener(a,e.agentes).verK).grupoK
$\wedge$ (a2 $\in$ obtener(a,e.agentes).grupoSanciones) $\iff$ (obtener(a,e.agentes).Qsanciones = obtener(a2,e.agentes).Qsanciones)
$\wedge$ (5) TodasLasPlacas(e,e.posAgentesLog)= claves(e.agentes)
$\wedge$ (6) ($\forall$ a3:agente, t: tupla(agente,posicion))(t $\in$ e.posAgentesLog $\wedge$ a3= Π_1(t) $\wedge_{L}$ definido?(a3,e.agentes) $\Rightarrow_{L}$ obtener(a3,e.agentes)= Π_2(t))
$\wedge$ enOrden(e.posAgentesLog) $\wedge$ enOrden(e.buscoEnLog)
$\wedge$ (7) UnionConjuntos(e,e.agregoEn1)=claves(e.agentes)
$\wedge$ ($\forall$ h,h1:tupla(nombre,itDicc(nombre;posicion)))(h $\in$ e.hippies $\wedge$ h1 $\in$ e.hippies $\wedge$ Π_1(h) $\neq$ Π_1(h1)) $\Rightarrow_{L}$ (Π_2(h) $\neq$ Π_2(h1))
$\wedge$ e.posCiviles=e.posRapida $\wedge$ ($\forall$ hi:nombre,e:nombre)((definido?(hi,e.posCiviles) $\wedge$ definido?(e,e.posCiviles)) $\Rightarrow_{L}$ obtener(e,e.posCiviles) $\neq$ obtener(hi,e.posCiviles)
$\wedge$ ($\forall$ a:agente, civ:nombre)(definido?(a,e.agentes) $\wedge$ definido?(civ,e.posCiviles))
$\Rightarrow_{L}$ (obtener(a,e.agentes) $\neq$ obtener(civ,e.posCiviles)) $\wedge$ (e.hippies $\cap$ e.estudiantes)= $\emptyset$
$\wedge$ JuntarIDS(e.estudiantes)$\cup$JuntarIDS(e.hippies)=claves(e.posCiviles))
$\wedge$ ($\forall$ i:nat, j:nat)(i$\geq$0 $\wedge$ i<e.campo.filas $\wedge$ j$\geq$0 $\wedge$ j<e.campo.columnas) $\Rightarrow_{L}$ **if** Π_1(e.quienOcupa[i][j])=false
then Π_2(e.quienOcupa[i][j]) = "*nada*"
else **if** Π_2(e.quienOcupa[i][j]) = "hippie"$\vee$ Π_2(e.quienOcupa[i][j]) = "*estudiante*" **then**
Π_3(e.quienOcupa[i][j])=itvacio
else
Π_4(e.quienOcupa[i][j])=itvacio **fi**
fi
$\wedge$ ($\forall$ k:nat)(($\exists$ i:nat)(i$\geq$0 $\wedge$ i<longitud(e.agregoEn1) $\Rightarrow_{L}$ e.agregoEn1[i].K=k) $\iff$ ($\exists$ ag:agente)(definido?(ag,e.agentes) $\Rightarrow_{L}$ obtener(ag,e.agentes).Qsanciones=k $\wedge$ ag $\in$ e.agregoEn1[i].grupoK))
$\wedge$ ($\forall$ k:nat)(($\exists$ i:nat)(i$\geq$0 $\wedge$ i<longitud(e.buscoEnLog) $\Rightarrow_{L}$ e.buscoEnLog[i].K=k) $\iff$ ($\exists$ ag:agente)(definido?(ag,e.agentes) $\Rightarrow_{L}$ obtener(ag,e.agentes).Qsanciones=k $\wedge$ ag $\in$ e.buscoEnLog[i].grupoK)))

1.3.4. Función de Abstracción

Abs : estr $e \longrightarrow$ rastrillaje {Rep(e)}

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

$\Big($campus(r) = e.campo $\wedge$ estudiantes(r) = e.estudiantes $\wedge$ hippies(r)=e.hippies $\wedge$ agentes(r)=e.agentes

$\wedge$ ($\forall$ n:nombre)((definido?(n,posEstudianteYHippie(n,r)) $\Longleftrightarrow$ definido?(n,e.diccString))

$\Rightarrow_{L}$ obtener(n,e.diccString)=obtener(n,posEstudianteYHippie(n,r))) $\wedge$

1.3.5. Algoritmos

```
1: function iCampus(in e: estr)⟶ res : campus                                   ▷ O(1)
2:     res ← e.campo
3: end function
```

```
1: function iEstudiantes(in e: estr)⟶ res : itConj(nombre)                      ▷ O(1)
2:     res ← crearIt (e.estudiantes)
3: end function
```

```
1: function iHippies(in e: estr)⟶ res : itConj(nombre)                          ▷ O(1)
2:     res ← crearIt (e.hippies)
3: end function
```

```
1: function iAgentes(in e: estr)⟶ res : itConj(agente)                          ▷ O(1)
2:     res ← claves(e.agentes)
3: end function
```

```
1: function iposEstudiantesYHippie(in e: estr, in n: nombre)⟶ res : posicion    ▷ O(|N_m|)
2:     res ← obtener(n,e.posCiviles)
3: end function
```

```
1: function iposAgente(in e: estr in a: agente)⟶ res : posicion                 ▷ O(1)(promedio)
2:     res ← obtener(a,e.agentes).posActual
3: end function
```

```
1: function icantSanciones(in e: estr, in a: agente)⟶ res : nat                 ▷ O(1)(promedio)
2:     res ← obtener(a,e.agentes).Qsanciones
3: end function
```

```
1: function icantHippiesAtrapados(in e: estr, in a: agente)⟶ res : nat          ▷ O(1)(promedio)
2:     res ← obtener(a,e.agentes).premios
3: end function
```

```
1: function imasVigilante(in e: estr)⟶ res : agente                             ▷ O(1)
2:     res ← siguiente(e.masVigilante)
3: end function
```

```
1: function iconMismasSanciones(in e: estr in a: agente)⟶ res : conj(agente)    ▷ O(1)
2:     res ← siguiente(obtener(e.agentes,a).grupoSanciones)
3: end function
```

```
1: function iConKSanciones(in e: estr in k: nat)⟶ res : conj(agente)   ▷ O(Na) la primera vez, luego mientras no haya sanciones O(logNa)
2:     if ¬e.hayNuevas then                                                ▷ O(1)
3:         var nat i ← BusquedaBin(e.buscoEnLog, k)                        ▷ O(logNa)
4:         res ← e.buscoEnLog[i].grupoK                                    ▷ O(1)
5:     else
6:         var itLista(datosK) itK ← crearIt(e.agregoEn1)                  ▷ O(1)
7:         while haySiguiente(it) do                                       ▷ O(1)
8:             buscoEnLog [i] ← siguiente(itK)                             ▷ O(1)
9:             avanzar(itK)
10:        end while
11:        var nat i ← BusquedaBin (e.buscoEnLog, k)                       ▷ O(logNa)
12:        res ← e.buscoEnLog[i].grupoK                                    ▷ O(1)
13:        e.hayNuevas ← false                                             ▷ O(1)
14:    end if
15: end function
```

```
1: function iComenzarRastrillaje(in c: campus in d: dicc(placa, posicion))⟶ res : estr   ▷ O(1)
2:     var diccPromedio(placa,datosAg) dprom ← vacio(#claves(d))                          ▷ O(#claves(d))
3:     var lista(datosK) Klista ← vacia()                                                 ▷ O(1)
4:     var vector(vector(datosPos)) map ← vacia()                                         ▷ O(1)
5:     for i=0 to filas?(c) do                                                            ▷ O(columnas?(c)^2 * filas?(c)^2)
6:     var vector(datosPos) filita ← vacia()                                              ▷ O(1)
7:     for j=0 to columnas?(c) do                                                         ▷ O(columnas?(c))
8:     if ocupada?(c,(j,i)) then                                                          ▷ O(1)
9:         AgAtras(filita,<true,"obstaculo",crearIt(),crearIt()>)                         ▷ O(columnas?(c))
10:    else
11:        AgAtras(filita,<false,"nada",crearIt(),crearIt()>)                             ▷ O(columnas?(c))
12:    end if
13:    EndFor
14:    AgAtras(map,filita)                                                                ▷ O(filas?(c))
15:    EndFor
16:    var Arreglo(<placa,posicion>) arr ← crearArreglo[#claves(d)]                       ▷ O(#claves(d))
17:    var itDicc(placa,posicion) iter ← crearIt(d)                                       ▷ O(1)
18:    var itLista(datosK) itk ← AgregarAtras(Klista,<0,vacio>)                           ▷ O(1)
19:    while haySiguiente(iter) do                                                        ▷ O(#claves(d))
20:        AgregarOrdenado(arr,<SiguienteClave(iter),SiguienteSignificado(iter)>)         ▷ O((#claves(d))^2)
21:        var datosAg datosN ← <0,0,SiguienteSignificado(iter),Agregar(Siguiente(itK).grupoK,SiguienteClave(iter)),itK>
22:
23:        map[SiguienteSignificado(iter.X)][SiguienteSignificado(iter.Y)] ← <true,"agente",definirRapido(dprom,
24:        SiguienteClave(iter),datosN),crearIt()>                                        ▷ O(1)
25:
26:        avanzar(iter)                                                                  ▷ O(1)
27:    end while
28:    var conj(datosHoE) hip ← vacio()                                                   ▷ O(1)
29:    var conj(datosHoE) est ← vacio()                                                   ▷ O(1)
30:    var diccString(nombre,posicion) diccS ← vacio()                                    ▷ O(1)
31:    var diccLineal(nombre,posicion) diccL ← vacio()                                    ▷ O(1)
32:    var itConj(placa) masV ← crearIt(dprom)                                            ▷ O(1)
33:    var bool hayNuevasS ← true                                                         ▷ O(1)
34:    var vector(datosK) paraLog ← vacia()                                               ▷ O(1)
35:    res ← <c,dprom,arr,hip,est,diccS,diccL,map,masV,kLista,paraLog,hayNuevasS>         ▷ O(1)
36: end function
```

```
1: function iIngresarEstudiante(in/out e: estr, in n: nombre, in p: posicion)                    ▷ O(| Nm |)
2:     if esHippizable(e,p) then                                                                  ▷ O(1)
3:         if esCapturable(e,p) then                                                              ▷ O(1)
4:             var conj(posicion) v ← vecinos(e.campus, p)                                        ▷ O(1)
5:             var itConj(posicion) it ← crearIt(v)                                               ▷ O(1)
6:             while haySiguiente(it) do
7:                 if e.quienOcupa[siguiente(it).X] [siguiente(it).Y].queHay == "agente" then     ▷ O(1)
8:                     recompensar(e, siguiente(it))                                              ▷ O(1)
9:                 end if
10:                avanzar(it)                                                                    ▷ O(1)
11:            end while
12:        else
13:            definir(e.posCiviles, n, p)                                                        ▷ O(| Nm |)
14:            var itDicc(nombre, posicion) iterPos ← definirRapido(e.posRapida,n,p)              ▷ O(1)
15:            e.quienOcupa[p.X] [p.Y] ← < true,"hippie",crearIt(), agregarRapido(e.hippies,<n,iterPos>)>   ▷ O(1)
16:            var conj(posicion) Ps ← vecinos(e.campus, p)                                       ▷ O(1)
17:            var itConj(posicion) it ← crearIt(Ps)                                              ▷ O(1)
18:            while haySiguiente(it) do
19:                if esEstudiante(e,siguiente(it)) ∧ esHippizable(e,siguiente(it)) then          ▷ O(1)
20:                    Hippizar(e, siguiente(it))                                                 ▷ O(1)
21:                    if esCapturable(e,siguiente(it)) then                                      ▷ O(1)
22:                        capturarHippie(e,siguiente(it))                                        ▷ O(| Nm |)
23:                    end if
24:                else
25:                    if esEstudiante(e,siguiente(it)) ∧ esCapturable(e, siguiente(it)) then     ▷ O(1)
26:                        var itConj(posicion) itAg ← vecinos(e.campus, siguiente(it))           ▷ O(1)
27:                        while haySiguiente(itAg) do                                            ▷ O(1)
28:                            if esAgente(e, siguiente(itAg)) then                               ▷ O(1)
29:                                sancionar(e,siguiente(itAg))                                   ▷ O(1)
30:                            end if
31:                            avanzar(itAg)                                                      ▷ O(1)
32:                        end while
33:                    else
34:                        if esHippie(e, siguiente(it)) ∧ esCapturable(e,siguiente(it)) then     ▷ O(1)
35:                            capturarHippie(e,siguiente(it))                                    ▷ O(| Nm |)
36:                        end if
37:                    end if
38:                end if
39:                avanzar(it)                                                                    ▷ O(1)
40:            end while
41:        end if
42:    else
43:        definir(e.posCiviles, n, p)                                                            ▷ O(| Nm |)
44:        var itDicc(nombre, posicion) iterPos ← definirRapido(e.posRapida, n, p)                ▷ O(1)
45:        e.quienOcupa[p.X] [p.Y] ← < true,"estudiante",crearIt(), agregarRapido(e.estudiantes,<n,iterPos>)>
46:        var conj(posicion) Ps ← vecinos(e.campus, p)                                           ▷ O(1)
47:        var itConj(posicion) it ← crearIt(ps)                                                  ▷ O(1)
48:
```

```
1: while haySiguiente(it) do
2:     if esHippie(e,siguiente(it)) ∧ esEstudiantizable(e,siguiente(it)) then                ▷ O(1)
3:         Estudiantizar(e,siguiente(it))                                                    ▷ O(1)
4:     else
5:         if esEstudiante(e,siguiente(it)) ∧ esCapturable(e,siguiente(it)) then             ▷ O(1)
6:             var itConj(posicion) itAg ← vecinos(e.campus, siguiente(it))                  ▷ O(1)
7:             while haySiguiente(itAg) do
8:                 if esAgente(e,siguiente(itAg)) then                                       ▷ O(1)
9:                     Sancionar(e,siguiente(itAg))                                          ▷ O(1)
10:                end if
11:                avanzar(itAg)                                                             ▷ O(1)
12:            end while
13:        else
14:            if esHippie(e,siguiente(it) ∧ esCapturable(e,siguiente(it)) then              ▷ O(1)
15:                capturarHippie(e,siguiente(it))                                           ▷ O(| Nm |)
16:            end if
17:        end if
18:    end if
19: end while
20: endFunction =0
```

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

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

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

```
1: function iMOVERHIPPIE(in/out e: estr in h: nombre)                                   ▷ O(|Nm|) + O(Ne)
2:     var posicion actual ← obtener(e.posCiviles, h)                                   ▷ O(|Nm|)
3:     var direccion d ← proxPosicionH(e,h)                                             ▷ O(Ne)
4:     var posicion prx ← proxPosicion(e.campus, d, obtener(e.posCiviles,h))            ▷ 1
5:     definir(e.posCiviles, h, prx)                                                    ▷ O(|Nm|)
6:     var itConj(nombre) itR ← e.quienOcupa[actual.X] [actual.Y].hayHoe                ▷ O(1)
7:     eliminarSiguiente(siguiente(itR).posActual)                                      ▷ O(1)
8:     siguiente(itR).posActual ← definirRapido(e.posRapida, h, prx)                    ▷ O(1)
9:     e.quienOcupa[prx.X] [prx.Y] ← <true, "hippie", crearIt(), itR>                   ▷ O(1)
10:    e.quienOcupa[actual.X] [actual.Y] ← <false, "nadie", crearIt(), crearIt()>       ▷ O(1)
11:    while haySiguiente(it) do
12:        if ocupada(e.campus, siguiente(it)) ∨ ¬e.quienOcupa[siguiente(it).X] [siguiente(it).Y].ocupada? then   ▷ O(1)
13:            avanzar(it)                                                              ▷ O(1)
14:        else
15:            if esEstudiante(e,siguiente(it)) ∧ esHippizable(e, siguiente(it)) then   ▷ O(1)
16:                hippizar(e, siguiente(it))                                           ▷ O(1)
17:                if esCapturable(e,siguiente(it) then
18:                    capturarHippie(e,siguiente(it))                                  ▷ O(|Nm|)
19:                end if
20:            else
21:                if esEstudiante(e, siguiente(it)) ∧ esCapturable(e, siguiente(it)) then   ▷ O(1)
22:                    var itConj(posicion) itAg ← vecinos(e.campus, siguiente(it))     ▷ O(1)
23:                    while haySiguiente(itAg) do
24:                        if esAgente(e,siguiente(itAg)) then
25:                            sancionar(e, siguiente(itAg))                            ▷ O(1)
26:                        end if
27:                        avanzar(itAg)                                                ▷ O(1)
28:                    end while
29:                else
30:                    if esHippie(e,siguiente(it)) ∧ esCapturable(e,siguiente(it)) then   ▷ O(1)
31:                        capturarHippie(e,siguiente(it))                              ▷ O(|Nm|)
32:                    end if
33:                end if
34:            end if
35:        end if
36:        avanzar(it)                                                                  ▷ O(1)
37:    end while
38: end function
```

```
1: function iESESTUDIANTE(in e: estr in p: posicion)⟶ res : bool                        ▷ O(1)
2:     res ← e.quienOcupa[p.X] [p.Y].queHay == "estudiante"
3: end function
```

```
1: function iESHIPPIE(in e: estr in p: posicion)⟶ res : bool                            ▷ O(1)
2:     res ← e.quienOcupa[p.X] [p.Y].queHay == "hippie"
3: end function
```

```
1: function iESAGENTE(in e: estr in p: posicion)⟶ res : bool                            ▷ O(1)
2:     res ← e.quienOcupa[p.X] [p.Y].queHay == "agente"
3: end function
```

```
1: function iEstudiantizar(in/out e: estr in p: posicion)                                   ▷ O(1)
2:     var datosHoE dat ← <Siguiente(e.quienOcupa[p.X] [p.Y].hayHoE).ID, Siguiente(e.quienOcupa[p.X]
   [p.Y].hayHoe).posActual>                                                                 ▷ O(1)
3:     eliminarSiguiente(e.quienOcupa[p.X] [p.Y].hayHoe)                                    ▷ O(1)
4:     var itConj(nombre) it ← agregarRapido(e.estudiantes, dat)                            ▷ O(1)
5:     e.quienOcupa[p.X] [p.Y] ← <true, "estudiante", crearIt(), it>                        ▷ O(1)
6: end function
```

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

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

```
1: function iesHippizable(in/out e: estr in p: posicion)⟶ res : bool                       ▷ O(1)
2:     var itConj(posicion) it ← crearIt(vecinos(e.campus, p))                              ▷ O(1)
3:     var nat Contador ← 0                                                                 ▷ O(1)
4:     while haySiguiente(it) do                                                            ▷ O(1)
5:         if e.quienOcupa[siguiente(it).X] [siguiente(it).Y].quienOcupa == "hippie" then   ▷ O(1)
6:             contador + +                                                                 ▷ O(1)
7:         end if
8:         avanzar(it)                                                                      ▷ O(1)
9:     end while
10:    res ← contador ⩾ 2                                                                   ▷ O(1)
11: end function
```

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

```
1: function iTODASOCUPADAS(in e: estr, in p: conj(posicion)) res:bool                       ▷ O(1)
2:     var itConj(posicion) it ← crearIt(p)                                                 ▷ O(1)
3:     var contador ← 0                                                                     ▷ O(1)
4:     while haySiguiente(it) do
5:         if e.quienOcupa[siguiente(it).X] [siguiente(it).Y].ocupada? then                 ▷ O(1)
6:             contador + +                                                                 ▷ O(1)
7:         end if
8:     end while
9:     res ← contador == 4                                                                  ▷ O(1)
10: end function
```

```
function iRECOMPENSAR(in/out e: estr, in a: posicion)                                       ▷ O(1)
    var placa p ← siguienteClave(quienOcupa[a.X] [a.Y].hayCana                              ▷ O(1)
    var datosAgente dat ← obtener(e.agentes,p)                                              ▷ O(1)
    dat.premios ← dat.premios+1                                                             ▷ O(1)
    if dat.premios > obtener(e.agentes, siguienteClave(e.masVigilante)).premios then        ▷ O(1)
        e.masVigilante ← quienOcupa[a.X] [a.Y].hayCana                                      ▷ O(1)
    else
        if dat.premios == obtener(e.agentes, siuienteClave(e.masVigilante)).premios then    ▷ O(1)
            if p < siguienteClave(e.masVigilante)) then                                     ▷ O(1)
                e.masVigilante ← quienOcupa[a.X] [a.Y].hayCana                              ▷ O(1)
            end if
        end if
    end if
end function
```

```
function iSANCIONAR(in/out e: estr, in a: posicion)                                         ▷ O(1)
    var placa p ← siguienteClave(quienOcupa[a.X] [a.Y].hayCana                              ▷ O(1)
    var datosAgente dat ← obtener(e.agentes,p)                                              ▷ O(1)
    dat.Qsanciones ← dat.Qsanciones+1                                                       ▷ O(1)
    eliminarSiguiente(dat.grupoSanciones)                                                   ▷ O(1)
    avanzar(dat.verK)                                                                       ▷ O(1)
    e.hayNuevas ← true                                                                      ▷ O(1)
    if Siguiente(dat.verK).K == dat.Qsanciones then                                         ▷ O(1)
        dat.grupoSanciones ← Agregar(siguiente(dat.verK).grupoK, p)                         ▷ O(1)
    else
        dat.grupoSanciones ← AgregarComoAnterior(dat.verK, <dat.Qsanciones,Agregar(Vacio(),p)>)  ▷ O(1)
    end if
end function
```

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

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

```
1: function iVecinoMasCercano(in e: estr in p: posicion in p2: posicion ) ⟶ res : direccion        ▷ O(1)
2:     var conj(posicion) Ps ← vecinos(e.campus, p)                                               ▷ O(1)
3:     var itConj(posicion) it ← crearIt(Ps)                                                      ▷ O(1)
4:     var posicion destino← siguiente(it)                                                        ▷ O(1)
5:     while haySiguiente(it) do                                                                  ▷ O(1)
6:         if (¬e.quienOcupa[siguiente(it).X][siguiente(it).Y].ocupada?) then                     ▷ O(1)
7:             if (distancia(e,p,siguiente(it))<distancia(e,p,destino)) then                      ▷ O(1)
8:                 destino← siguiente(it)                                                         ▷ O(1)
9:             end if
10:        end if
11:        avanzar(it)                                                                            ▷ O(1)
12:    end while
13:    if (destino.X≠p.X) then                                                                    ▷ O(1)
14:        if (destino.y>p.Y) then                                                                ▷ O(1)
15:            res← Arriba
16:        else                                                                                   ▷ O(1)
17:            res← Abajo                                                                         ▷ O(1)
18:        end if
19:    else
20:        if (destino.x>p.x) then                                                                ▷ O(1)
21:            res← Derecha
22:        else                                                                                   ▷ O(1)
23:            res← Izquierda                                                                     ▷ O(1)
24:        end if
25:    end if
26: end function
```

```
1: function iSeFue(in e: estr in p: posicion in destino: posicion ) ⟶ res : bool                 ▷ O(1)
2:     res ← (p.Y==e.campus.alto −1 ∧ destino.y==e.campus.alto) ∨(p.Y==0  destino.y== −1)         ▷ O(1)
3: end function
```

```
1: function iBusquedaBin(in v: vector(datosK) in obj: nat) ⟶ res : nat                  ▷ O(log_2(longitud(v)))
2:     var int i← 0                                                                              ▷ O(1)
3:     var int d← longitud(v)-1                                                                  ▷ O(1)
4:     while (i +1 < d) do                                                                       ▷ O(log_2(longitud(v)))
5:         var int m← (i+d)/2                                                                    ▷ O(1)
6:         if (v[m].k<obj) then                                                                  ▷ O(1)
7:             i← m
8:         else                                                                                  ▷ O(1)
9:             d← m                                                                              ▷ O(1)
10:        end if
11:    end while
12:    if (v[i].k==obj) then                                                                     ▷ O(1)
13:        res← i
14:    else                                                                                      ▷ O(1)
15:        res← d                                                                                ▷ O(1)
16:    end if
17: end function
```

```
1: function iOCUPADAD(in e: rastr in p: posicion in dir: direccion) ⟶ res : bool          ▷ O(1)
2:     if dir== " Arriba" then                                                          ▷ O(1)
3:         res←e.quienOcupa[p.X][p.Y +1].ocupada?
4:     else                                                                             ▷ O(1)
5:         if dir== " Abajo" then                                                       ▷ O(1)
6:             res←e.quienOcupa[p.X][p.Y −1].ocupada?
7:         else                                                                         ▷ O(1)
8:             if dir== " izquierda" then                                               ▷ O(1)
9:                 res←e.quienOcupa[p.X−1][p.Y].ocupada?
10:            else                                                                     ▷ O(1)
11:                res←e.quienOcupa[p.X+1][p.Y].ocupada?                                ▷ O(1)
12:            end if
13:        end if
14:    end if
15: end function
```

2. Diseño del Tipo Iterador sobre Lista Extendido(α)

2.1. Aspectos de la interfaz

2.1.1. Interfaz

Se extiende la interfaz del Iterador sobre Lista dada en el apunte de módulos básicos, el cual recorrerá, una lista de tuplas, por lo que las operaciones Siguiente y Anterior, devuelven el primer elemento.

Operaciones básicas del Iterador Extendido

CrearIt(**in** l: $lista(\alpha)$) $\longrightarrow res : itListaE(\alpha)$

Pre $\equiv$ { $true$ }

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

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

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.

Siguiente(**in** it: $itListaE(\alpha)$) $\longrightarrow res : \alpha$

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

Post $\equiv$ { $alias(res =_{\text{obs}} \Pi_1(\text{Siguiente(it)}))$}

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

Descripción: Devuelve el elemento siguiente a la posición del iterador.

Aliasing: Res es modificable si y sólo si it es modificable.

Anterior(**in** it: $itListaE(\alpha)$) $\longrightarrow res : \alpha$

Pre $\equiv$ { $HayAnterior?(it)$ }

Post $\equiv$ { $alias(res =_{\text{obs}} \Pi_1(\text{Anterior(it)}))$}

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

Descripción: Devuelve el elemento siguiente a la posición del iterador.

Aliasing: Res es modificable si y sólo si it es modificable.

2.1.2. Algoritmos

```
1: function iCrearItE(in l: lista(α))⟶ res: itListaE                ▷ O(1)
2:     res ← <l.primero, l>                                          ▷ O(1)
3: end function
```

```
1: function iSiguiente(in it: itListaE(α))⟶ res: α                  ▷ O(1)
2:     res ← Π1(it.siguiente→ dato)                                  ▷ O(1)
3: end function
```

```
1: function iAnterior(in it: itListaE(α))⟶ res: α                   ▷ O(1)
2:     res ← Π1(SiguienteReal(it)→ anterior → dato)                  ▷ O(1)
3: end function
```