



DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2: Diseño

Primer cuatrimestre - 2015

Algoritmos y Estructuras de Datos II

Grupo 2

Integrante	LU	Correo electrónico
Benitez, Nelson	945/13	nelson.benitez92@gmail.com

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		



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) 4576-3300

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

Índice

1. CampusSeguro	2
1.1. Interfaz	2
1.2. Interfaz	2
1.3. Representación	4
1.4. Algoritmos	7
1.5. Algoritmos operaciones auxiliares	11
2. Pos es tupla(x:Nat, y:Nat)	15
3. Placa es Nat	15
4. Nombre es String	15
5. Diccionario Rapido	15
5.1. Interfaz	15
5.2. Representación	16
5.3. Algoritmos	16
5.4. Operaciones privadas	17
5.5. Representación del iterador	17
6. Diccionario por nombres	17
6.1. Interfaz	17
6.2. Representación	18
6.3. Algoritmos	19
6.4. Operaciones del iterador	20
6.5. Representación del iterador	22
6.6. Algoritmos del iterador	22
7. Campus	23
7.1. Interfaz	23
7.2. Representación	25
7.3. Algoritmos	26
7.4. Servicios Usados	29

1 CampusSeguro

1.1 Interfaz

se explica con AS

usa

géneros as

1.2 Interfaz

se explica con CAMPUSSEGURO

usa

géneros CampusSeguro

Operaciones

CAMPUS(**in** *cs* : campusSeguro) \longrightarrow *res* : campus

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

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

Descripción: Devuelve el campus del campusSeguro ingresado.

Complejidad: $O(1)$

ESTUDIANTES(**in** *cs* : campusSeguro) \longrightarrow *res* : conj(nombre)

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

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

Descripción: Devuelve un conjunto con los estudiantes del campusSeguro ingresado.

Complejidad: $O(1)$

HIPPIES(**in** *cs* : campusSeguro) \longrightarrow *res* : conj(nombre)

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

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

Descripción: Devuelve un conjunto con los hippies del campusSeguro ingresado.

Complejidad: $O(1)$

AGENTES(**in** *cs* : campusSeguro) \longrightarrow *res* : conj(agentes)

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

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

Descripción: Devuelve un conjunto con los agentes del campusSeguro ingresado.

Complejidad: $O(1)$

POSICIONESTUDIANTESYHIPPIES(**in** *id* : nombre, *cs* : campusSeguro) \longrightarrow *res* : posicion

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

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

Descripción: Devuelve la posicion del estudiante o hippie ingresado.

Complejidad: $O(|n_m|)$

POSICIONAGENTE(**in** *a* : agente, *cs* : campusSeguro) \longrightarrow *res* : posicion

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

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

Descripción: Devuelve la posicion del agente ingresado.

Complejidad: $O(1)$

CANTIDADSANCIONES(**in** $a : agente$, $cs : campusSeguro$) $\longrightarrow res : nat$

Pre $\equiv \{a \in agentes(cs)\}$

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

Descripción: Devuelve la cantidad de sanciones del agente ingresado.

Complejidad: $O(1)$

CANTIDADHIPPIESATRAPADOS(**in** $a : agente$, $cs : campusSeguro$) $\longrightarrow res : nat$

Pre $\equiv \{a \in agentes(cs)\}$

Post $\equiv \{res =_{obs} cantHippiesAtrapados(a, cs)\}$

Descripción: Devuelve la cantidad de hippies atrapados por el agente ingresado.

Complejidad: $O(1)$

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

Pre $\equiv \{(\forall a : agente)(def?(a, d) \Rightarrow_L (posValida?(obtener(a, d)) \wedge \neg ocupada?(obtener(a, d), c))) \wedge$
 $(\forall a, a2 : agente)((def?(a, d) \wedge def?(a2, d) \wedge a \neq a2) \Rightarrow_L obtener(a, d) \neq obtener(a2, d))\}$

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

Descripción: Crea un nuevo campusSeguro con campus y los agentes ingresados.

Complejidad: $O(1)$

INGRESARESTUDIANTE(**in** $e : nombre$, $p : posicion$, $in/out\ cs : campusSeguro$)

Pre $\equiv \{cs =_{obs} cs_0 \wedge e \notin (estudiantes(cs) \cup hippies(cs))esIngreso?(p, campus(cs)) \wedge$
 $\neg estaOcupada?(p, cs)\}$

Post $\equiv \{res =_{obs} ingresarEstudiante(e, p, cs_0)\}$

Descripción: Ingresa un nuevo estudiante al campus por una de las entradas.

Complejidad: $O(|n_m|)$

INGRESARHIPPIE(**in** $h : nombre$, $p : posicion$, $in/out\ cs : campusSeguro$)

Pre $\equiv \{(cs) \equiv (cs_0) \wedge h \notin (estudiantes(cs) \cup hippies(cs))esIngreso?(p, campus(cs)) \wedge$
 $\neg estaOcupada?(p, cs)\}$

Post $\equiv \{res =_{obs} ingresarHippie(e, p, cs_0)\}$

Descripción: Ingresa un nuevo hippie al campus por una de las entradas.

Complejidad: $O(|n_m|)$

MOVERESTUDIANTE(**in** $e : nombre$, $d : direccion$, $in/out\ cs : campusSeguro$)

Pre $\equiv \{(cs) \equiv (cs_0) \wedge e \in estudiantes(cs) \wedge (seRetira(e, d, cs) \vee$
 $(posValida?(proxPosicion(posEstudianteYHippie(e, cs), d, campus(cs)), campus(cs)) \wedge$
 $\neg estaOcupada?(proxPosicion(posEstudianteYHippie(e, cs), d, campus(cs)), cs))\}$

Post $\equiv \{res =_{obs} moverEstudiante(e, d, cs_0)\}$

Descripción: Mueve un estudiante en la direccion indicada.

Complejidad: $O(|n_m|)$

MOVERHIPPIE(**in** $h : nombre$, $in/out\ cs : campusSeguro$)

Pre $\equiv \{(cs) \equiv (cs_0) \wedge h \in hippies(cs) \wedge$
 $\neg todasOcupadas?(vecinos(posEstudianteYHippie(h, cs), campus(cs)), cs)\}$

Post $\equiv \{res =_{obs} moverHippie(h, cs_0)\}$

Descripción: Mueve un hippie hacia el estudiante más cercano.

Complejidad: $O(|n_m| + N_e)$

MOVERAGENTE(**in** $a : nombre$, $in/out\ cs : campusSeguro$)

Pre $\equiv \{(cs) \equiv (cs_0) \wedge a \in agentes(cs) \wedge_L cantSanciones(a, cs) \leq 3 \wedge$
 $\neg todasOcupadas?(vecinos(posAgente(a, cs), campus(cs)), cs)\}$

Post $\equiv \{res =_{obs} moverAgente(a, cs_0)\}$

Descripción: Mueve un agente hacia el hippie más cercano.

Complejidad: $O(|n_m| + \log N_a + N_h)$

CANTIDADHIPPIES(**in** $cs : campusSeguro$) $\longrightarrow res : nat$

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

Post $\equiv \{res =_{\text{obs}} \text{cantHippies}(cs)\}$

Descripción: Devuelve la cantidad de hippies en el campus.

Complejidad: $O(1)$

CANTIDADESTUDIANTES(**in** $cs : \text{campusSeguro}$) $\longrightarrow res : \text{nat}$

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

Post $\equiv \{res =_{\text{obs}} \text{cantEstudiantes}(cs)\}$

Descripción: Devuelve la cantidad de estudiantes en el campus.

Complejidad: $O(1)$

MÁSVIGILANTE(**in** $cs : \text{campusSeguro}$) $\longrightarrow res : \text{agente}$

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

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

Descripción: Devuelve al agente con más capturas realizadas del campus.

Complejidad: $O(1)$ **HAYALGO**(**in** $cs : \text{campusSeguro}, p : \text{pos}$) $\longrightarrow res : \text{bool}$

Pre $\equiv \{\text{PosValida}(cs.\text{campus}, p)\}$

Post $\equiv \{(cs.\text{campus}.[p.X][p.Y - 1].\text{hayHippie}) \vee (cs.\text{campus}.[p.X][p.Y - 1].\text{hayEst}) \vee (cs.\text{campus}.[p.X][p.Y -$

Descripción: Dice si la posición está ocupada.

Complejidad: $O(1)$

Aliasing:

Las complejidades están en función de las siguientes variables:

c : es una instancia del `campusSeguro`,

p : es una posición,

n : es el nombre de un estudiante/hippie y $|n_m|$ es la longitud más larga entre todos los nombres del `campusSeguro`,

d : es una dirección,

N_a : es la cantidad de agentes,

N_e : es la cantidad actual de estudiantes,

N_h : es la cantidad actual de hippies.

Las complejidades están en función de las siguientes variables:

n : la cantidad total de computadoras que hay en el sistema,

L : el hostname más largo de todas las computadoras,

k : la cola de paquetes más larga de todas las computadoras.

1.3 Representación

se representa con sistema

```

donde sistema es tupla⟨CampusEstatico : Campus,
                        Campus : arreglo(arreglo(tupla⟨hayHippie : bool,
                                                    hayEst : bool,
                                                    hayAgente : bool,
                                                    hayObst : bool,
                                                    agente : itDiccR(agente),
                                                    estudiante : itDPN(tupla⟨nombre : string,
                                                                    pos : pos⟩
                                                    hippie : itDPN(tupla⟨nombre : String,
                                                                    pos : pos⟩
estudiantes : DiccPorNombre(nombre:string, pos:pos),
hippies : DiccPorNombre(nombre:string, pos:pos),
agentesPorPlaca : arreglo(placa:placa, pos:pos),
agentes : DiccSuperRapido(pl:nat,tupla⟨pos : pos,
                                     cantSanc : nat,
                                     cantCapturas : nat,
                                     mismas : itLista(conMismasBucket),
                                     miUbicacion : itLista(agente)⟩
masVigilante : agente:itDiccRapido,
porSanciones : Lista(conMismasBucket),
conKSanciones : tupla⟨ocurrioSancion : bool,
                      arreglo : arreglo(tupla⟨porKSanc : conj(agente),)⟩
                                     #Sanciones : nat⟩
donde conMismasBucket es  tupla⟨agentes : Conj(agente),
                              #Sanc : nat⟩

```

Invariante de representación

1. El campus estatico tiene las mismas dimensiones que campus
2. Los obstaculos de campusEstatico estan en la misma pos que en Campus
3. Hay a lo sumo un bool en true en cada posicion de Campus
4. Todos los iteradores de las posiciones donde hayHippie apuntan a una pos valida del dicc de hippies
5. Todos los iteradores de las posiciones donde hayEst apuntan a una pos valida del dicc de estudiantes
6. Todos los iteradores de las posiciones donde hayAgente apuntan a una pos valida del dicc de agentes
7. Todos los significados de hippies apuntan a una posicion valida dentro de Campus, hay un hippie en esa pos, el iterador apunta a la tupla (clave,significado) que estoy consultando
8. Todos los significados de estudiantes apuntan a una posicion valida dentro de Campus, hay un estudiante en esa pos, el iterador apunta a la tupla (clave,significado) que estoy consultando
9. Todos las pos de los significados de agentes apuntan a una posicion valida dentro de Campus, hay un agente en esa pos, el iterador apunta a la tupla (clave,significado) que estoy consultando
10. AgentesPorPlaca tiene el mismo tamaño que el dicc de agentes

11. El conjunto de tuplas (placa,pos), formado por placas y pos de los significados del siguiente de agente en cada posicion de campus donde hayAgente y las posiciones de AgentesPorPlaca, son iguales
12. AgentesPorPlaca esta ordenado por numero de placa
13. Para todos mismas en los significados del dicc de agentes, la contatenacion de los anteriores , el actual y los siguientes son iguales a porSanciones

Rep : $\widehat{\text{sistema}} \longrightarrow \text{boolean}$

($\forall s : \widehat{\text{sistema}}$)

Rep(s) \equiv

Función de abstracción

Abs : $\widehat{\text{dcnet}} \ s \longrightarrow \widehat{\text{DCNet}}$

{Rep(s)}

($\forall s : \widehat{\text{dcnet}}$)

Abs(s) $\equiv dc : \widehat{\text{DCNet}} \mid$

$red(dc) =^*(s.red) \wedge (\forall c : \text{compu}, c \in \text{compus}(dc))(enEspera(dc, c) =^*(enEspera(s, c)) \wedge$

$cantidadEnviados(dc, c) = cantidadEnviados(s, c)) \wedge$

$(\forall p : \text{paquete}, paqueteEnTransito?(dc, p))caminoRecorrido(dc, p) =^*(caminoRecorrido(s, p))$

1.4 Algoritmos

CAMPUS(in <i>as</i> : as) \longrightarrow <i>res</i> : campus <i>res</i> \leftarrow <i>as.campus</i>	O(1)
	<hr/>
	O(1)
AGENTES(in <i>as</i> : as) \longrightarrow <i>res</i> : itDiccRapido (agente) <i>res</i> \leftarrow <i>CrearItRapido</i> (<i>as.agentes</i>)	O(1)
	<hr/>
	O(1)
ESTUDIANTES(in <i>as</i> : as) \longrightarrow <i>res</i> : itDPN (< estudiante:nombre , pos:pos >) <i>res</i> \leftarrow <i>CrearItDPN</i> (<i>as.estudiantes</i>)	O(1)
	<hr/>
	O(1)
HIPPIES(in <i>as</i> : as) \longrightarrow <i>res</i> : itDPN (< hippie:nombre , pos:pos >) <i>res</i> \leftarrow <i>CrearItDPN</i> (<i>as.hippies</i>)	O(1)
	<hr/>
	O(1)
POSESTUDIANTESYHIPPIES(in <i>as</i> : as , <i>in nombre</i> : nombre) \longrightarrow <i>res</i> : pos if <i>as.estudiantes.definido?(nombre)</i> then O(long(nombre)) <i>return res</i> \leftarrow <i>as.estudiantes.obtener(nombre)</i> O(long(nombre)) end if if <i>as.hippies.definido?(nombre)</i> then O(long(nombre)) <i>return res</i> \leftarrow <i>as.hippies.obtener(nombre)</i> O(long(nombre)) end if	
	<hr/>
	O(long(nombre))
POSAGENTE(in <i>as</i> : as , <i>in placa</i> : agente) \longrightarrow <i>res</i> : pos <i>res</i> \leftarrow <i>as.agentes.obtener(placa).pos</i>	$\theta(1)$
	<hr/>
	$\theta(1)$
CANTSANCIONES(in <i>as</i> : as , <i>in placa</i> : agente) \longrightarrow <i>res</i> : pos <i>res</i> \leftarrow <i>as.agentes.obtener(placa).cantSanciones</i>	$\theta(1)$
	<hr/>
	$\theta(1)$
CANTHIPPIESATRAPADOS(in <i>as</i> : as , <i>in placa</i> : agente) \longrightarrow <i>res</i> : pos <i>res</i> \leftarrow <i>as.agentes.obtener(placa).cantCapturas</i>	$\theta(1)$
	<hr/>
	$\theta(1)$
INGRESARESTUDIANTE(in/out <i>as</i> : as , <i>in nombre</i> : string , <i>in pos</i> : pos) <i>as.agregarEstudiante</i> (<i>as,pos,nombre</i>) O(long(nombre)) // Sanciono a los agentes que rodean a los estudiantes atrapados al ingresar uno nuevo <i>as.sancionarAgentesVecinos</i> (<i>as,pos</i>) O(1) <i>as.sancionarAgentesEncerrandoEstVecinos</i> (<i>as,pos</i>) O(1) // Hippificar al estudiante if <i>as.estAHippie?(as,pos)</i> then <i>as.hippiificar</i> (<i>as,pos</i>) O(long(nombre)) end if // Convertir a los hippies vecinos que quedaron encerrados por 4 estudiantes o eliminar a los que quedaron atrapados por agentes	

<i>as.aplicarHippiesVecinos(as, pos)</i>	$O(\text{long}(\text{nombre}))$
<i>// Capturar hippie en pos actual</i>	
if <i>as.campus[pos.x][pos.y].hayHippie?</i> then	
<i>aplicarHippie(pos)</i>	$O(\text{long}(\text{nombre}))$
end if	
<hr/>	
INGRESARHIPPIE(in/out <i>as : as</i> , <i>in nombre : string</i> , <i>in pos : pos</i>)	
<i>as.agregarHippie(as, pos, nombre)</i>	$O(\text{long}(\text{nombre}))$
<i>as.sancionarAgentesEncerrandoEstVecinos(as, pos)</i>	$O(1)$
<i>as.aplicarHippie(as, pos)</i>	$O(\text{long}(\text{nombre}))$
<i>as.aplicarHippiesVecinos(as, pos)</i>	$O(\text{long}(\text{nombre}))$
<hr/>	
	$O(\text{long}(\text{nombre}))$
COMENZARRASTRILLAJE(in/out <i>as : as</i> , <i>in ce : CampusEstatico</i> , <i>in Agentes : dicc(placa pos)</i>)	
<i>as.agentesPorPlaca</i> \leftarrow <i>CrearVector(Agentes.tamano())</i>	
<i>as.CampusEstatico</i> \leftarrow <i>ce</i>	$O(1)$
<i>i</i> \leftarrow 0	$O(1)$
<i>j</i> \leftarrow 0	$O(1)$
while <i>i</i> < <i>ce.Ancho</i> do	$O(\text{Ancho})$
while <i>j</i> < <i>ce.Alto</i> do	$O(\text{Alto})$
<i>as.Campus[i][j].HayHippie</i> \leftarrow <i>False</i>	$O(1)$
<i>as.Campus[i][j].HayEst</i> \leftarrow <i>False</i>	$O(1)$
<i>as.Campus[i][j].HayObst</i> \leftarrow <i>ce.Obstaculos[i][j]</i>	$O(1)$
<i>i</i> \leftarrow <i>i</i> + 1	$O(1)$
<i>j</i> \leftarrow <i>j</i> + 1	$O(1)$
end while	
end while	
<i>as.Agentes</i> \leftarrow <i>CrearDiccRapido(Agentes)</i>	$O(1)$
<i>ItAgentes</i> \leftarrow <i>CrearItAgentes(as.Agentes)</i>	$O(1)$
<i>MayorPlaca</i> \leftarrow 0	$O(1)$
<i>i</i> \leftarrow 0	
while <i>ItAgente.HaySiguiente</i> do	$O(\text{Cantidad Agentes})$
<i>// Copio el iterador y lo asigno al lugar correspondiente</i>	
<i>nuevoIt</i> = <i>ItAgente</i>	$O(1)$
<i>as.Campus[ItAgentes.Siguiente.pos.X][ItAgentes.Siguiente.pos.Y].HayAgente</i> \leftarrow <i>True</i>	$O(1)$
<i>as.Campus[ItAgentes.Siguiente.pos.X][ItAgentes.Siguiente.pos.Y].agente</i> \leftarrow <i>nuevoIt</i>	$O(1)$
if <i>MayorPlaca</i> < <i>ItAgentes.siguienteClave()</i> then	
<i>MayorPlaca</i> \leftarrow <i>ItAgentes.siguienteClave()</i>	$O(1)$
<i>MayorAgente</i> \leftarrow <i>ItAgentes()</i>	$O(1)$
end if	
<i>as.agentesPorPlaca[i]</i> \leftarrow <i>ItAgentes.siguienteSignificado().pos</i>	$O(1)$
<i>ItAgentes.Avanzar</i>	$O(1)$
<i>i</i> ++	$O(1)$
end while	
<i>// Ordena el arreglo de agentes por placa</i>	
<i>MergeSort(as.agentesPorPlaca)</i>	$O(\log(N_a) * N_a)$
<i>as.Estudiante</i> \leftarrow <i>Vacio()</i>	$O(1)$

<i>as.Hippee</i> \leftarrow <i>Vacio</i> ()	$O(1)$
<i>as.masVigilante</i> \leftarrow <i>MayorAgente</i>	$O(1)$
<i>as.porSanciones</i> \leftarrow <i>CrearLista</i> (<i>Tupla</i> $<$ <i>agentes</i> \leftarrow <i>Vacio</i> () , # <i>sanciones</i> \leftarrow 0 $>$	$O(1)$
<i>ItAgentesRapido</i> \leftarrow <i>dameIterador</i> (<i>as.agentes</i>)	$O(1)$
while <i>ItAgentesRapido.HaySiguiente</i> do	$O(\text{Cantidad Agentes})$
<i>ItAgentesRapido.siguiente.mismas</i> \leftarrow <i>CrearIt</i> (<i>as.porSanciones</i>)	$O(1)$
<i>ItAgentesRapido.siguiente.miUbicacion</i> \leftarrow	
<i>as.porSanciones.obtenerUltimo.Agentes.Agregar</i> (<i>ItAgentesRapido.siguienteClave</i>)	$O(1)$
<i>ItAgentesRapido.Avanzar</i>	$O(1)$
end while	

 $O((\text{Ancho} * \text{Alto}) + N_a)$

MOVERESTUDIANTE(**in/out** *as* : **as**, *in nombre* : **String**, *in dir* : **direccion**)

// PRE: El nombre es una clave del dicc de estudiantes, se retira o (La prox posicion es valida y no esta ocupada)

<i>posVieja</i> \leftarrow <i>as.estudiantes.obtener</i> (<i>nombre</i>)	$O(\text{long}(\text{nombre}))$
if $\neg(\text{as.campusEstatico.seRetira}(\text{as.campus}, \text{dir}, \text{posVieja}))$ then	
// Mover el estudiante	
<i>proxPos</i> \leftarrow <i>as.campusEstatico.proxPos</i> (<i>posVieja</i> , <i>dir</i>)	$O(1)$
<i>as.campus[posVieja.x][posVieja.y].hayEst?</i> \leftarrow <i>False</i>	
<i>as.campus[proxPos.x][proxPos.y].hayEst</i> \leftarrow <i>True</i>	$O(1)$
<i>as.campus[proxPos.x][proxPos.y].estudiante</i> \leftarrow <i>as.campus[posVieja.x][posVieja.y].estudiante</i>	$O(1)$
<i>as.campus[posVieja.x][posVieja.y].estudiante</i> \leftarrow <i>NULL</i>	$O(1)$
// Sancionar agentes vecinos y a los que encierran a est vecinos	
<i>sancionarAgentesEncerrandoEstVecinos</i> (<i>as</i> , <i>pos</i>)	$O(1)$
<i>sancionarAgentesVecinos</i> (<i>as</i> , <i>pos</i>)	$O(1)$
// Convertir a estudiantes los hippies vecinos o capturarlos	
<i>aplicarHippiesVecinos</i> (<i>as</i> , <i>pos</i>)	$O(1)$
else	
<i>as.campus[posVieja.x][posVieja.y].hayEst?</i> \leftarrow <i>False</i>	$O(1)$
<i>as.campus[posVieja.x][posVieja.y].estudiante.eliminarSiguiente</i> ()	$O(\text{long}(\text{nombre}))$
end if	

 $O(\text{long}(\text{nombre}))$

MOVERHIPPIE(**in/out** *as* : **as**, *in nombre* : **string**)

if $\neg(\text{encerrado?}(\text{as}, \text{as.hippies.obtener}(\text{nombre})))$ then	$O(\text{long}(\text{nombre}))$
// Obtener pos siguiente y actualizar posicion de hippie	
<i>posVieja</i> \leftarrow <i>as.hippies.obtener</i> (<i>nombre</i>)	$O(\text{long}(\text{nombre}))$
<i>posNueva</i> \leftarrow <i>proxPosHippie</i> (<i>as</i> , <i>nombre</i>)	$O(\text{long}(\text{nombre}) + N_e)$
<i>as.campus[posVieja.x][posVieja.y].hayHippie</i> \leftarrow <i>False</i>	$O(1)$
<i>as.campus[posNueva.x][posNueva.y].hayHippie</i> \leftarrow <i>True</i>	$O(1)$
<i>itHippie</i> \leftarrow <i>as.campus[posVieja.x][posVieja.y].hippie</i>	$O(1)$
<i>as.campus[posVieja.x][posVieja.y].hippie</i> \leftarrow <i>NULL</i>	$O(1)$
<i>as.campus[posNueva.x][posNueva.y].hippie</i> \leftarrow <i>itHippie</i>	$O(1)$

```

// Sancionar agentes que rodean a los estudiantes que encierro
sancionarAgentesEncerrandoEstVecinos(as,posNueva)          O(1)
// Capturar hippies encerrados
aplicarHippiesVecinos(as,posNueva)                          O(long(nombre))
// Hippificar estudiantes
hippificarEstudiantesVecinos(as,posNueva)                   O(long(nombre))
end if

```

```

MOVERAGENTE(in/out as : as, in placa : placa)
// Obtener pos siguiente y actualizar pos de agente
posVieja ← busquedaBinariaPorPlaca(as.agentesPorPlaca,placa).pos
                                                    O(log(Na))

if ¬(encerrado?(as,posVieja))
  ∧
  as.campus[posVieja.x][posVieja.y].agente.siguienteSignificado().cantSanciones < 3 then
                                                    O(1)
    proxPos ← as.proxPosAgente(posVieja)                O(Nh)
    as.campus[posVieja.x][posVieja.y].hayAgente ← False O(1)
    as.campus[proxPos.x][proxPos.y].hayAgente ← True   O(1)
    itAgente ← as.campus[posVieja.x][posVieja.y].agente O(1)
    as.campus[proxPos.x][proxPos.y].agente ← itAgente O(1)
    as.campus[posVieja.x][posVieja.y].agente ← NULL   O(1)
    busquedaBinariaPorPlaca(as.agentesPorPlaca,placa).pos ← proxPos
                                                    O(log(Na))
    sancionarAgentesEncerrandoEstVecinos(as,proxPos)   O(1)
    aplicarHippiesVecinos(as,proxPos)                   O(long(nombre))
  end if
                                                    O(Nh + log(Na) + long(nombre))

```

```

CONMISMASANCIONES(in as : as, in placa : placa) → res : Conj(agente)
posAgente ← as.agentes.obtener(placa)                O(θ(1))
res ← as.campus[posAgente.x][posAgente.y].agente.siguienteSignificado().mismas.agentes
                                                    O(1)
                                                    O(θ(1))

```

```

CONKSANCIONES(in as : as, in k : nat) → res : Conj(agente)
res ← ∅                                                O(1)
if as.conKSanciones.ocurrioSancion then
  // 'Copio' la lista de porSanciones a un vector, asi luego puedo hacer bus binaria sobre el
  as.conKSanciones ← CrearArreglo(as.porSanciones.tamano())
                                                    O(1)
  it ← CrearItLista(as.porSanciones)                 O(1)
  i ← 0                                                O(1)
  while it.haySiguiente do                             O(Na)
    conKSanciones.arreglo[i].cantSanciones ← it.siguienteSignificado().cantSanciones
                                                    O(1)
    // Por referencia
    conKSanciones.arreglo[i].conKSanciones ← it.siguienteSignificado().agentes
                                                    O(1)
    if it.siguienteSignificado().cantSanciones = k then
      res ← it.siguienteSignificado().agentes         O(1)
    end if
  end while

```

end if	
<i>i</i> ++	O(1)
<i>it.avanzar()</i>	O(1)
end while	
<i>return res</i>	O(1)
else	
<i>return res</i> \leftarrow <i>busquedaBinariaPorSanciones</i> (<i>conKSanciones.arreglo</i> , <i>k</i>). <i>conKSanciones</i>	$O(\log(N_a))$
end if	
<hr/>	
	$O(N_a) \vee O(\log(N_a))$
MASVIGILANTE(in <i>as</i> : as) \longrightarrow <i>res</i> : placa	
<i>res</i> \leftarrow <i>as.masVigilante.siguienteClave()</i>	O(1)
	<hr/>
	O(1)

1.5 Algoritmos operaciones auxiliares

SANCIONARAGENTESVECINOS(in/out <i>as</i> : as , <i>in pos</i> : pos)	
<i>vecinos</i> \leftarrow <i>as.campusEstatico.vecinos</i> (<i>pos</i>)	O(1)
if <i>as.atrapadoPorAgente?</i> (<i>pos</i>) then	
while <i>i</i> < <i>vecinos.tamano()</i> do	O(1)
if <i>as.campus</i> [<i>vecinos</i> [<i>i</i>]. <i>x</i>][<i>vecinos</i> [<i>i</i>]. <i>y</i>]. <i>hayAgente?</i> then	
<i>as.sancionarAgente</i> (<i>vecinos</i> [<i>i</i>]. <i>agente</i>)	O(1)
end if	
<i>i</i> ++	
end while	
end if	
	<hr/>
	O(1)
SANCIONARAGENTESENCERRANDOESTVECINOS(in/out <i>as</i> : as , <i>in pos</i> : pos)	
<i>vecinos</i> \leftarrow <i>as.campusEstatico.vecinos</i> (<i>pos</i>)	O(1)
<i>i</i> \leftarrow 0	
while <i>i</i> < <i>vecinos.tamano</i> do	O(1)
if <i>as.campus</i> [<i>vecinos</i> [<i>i</i>]. <i>x</i>][<i>vecinos</i> [<i>i</i>]. <i>y</i>]. <i>hayEst</i> \wedge <i>atrapadoPorAgente?</i> (<i>as</i> , <i>pos</i>) then	
<i>sancionarAgentesVecinos</i> (<i>as</i> , <i>pos</i>)	O(1)
end if	O(1)
<i>i</i> ++	
end while	
	<hr/>
	O(1)
SANCIONARAGENTE(in/out <i>as</i> : as , <i>in/out agente</i> : itDiccRapido)	
<i>as.conKSanciones.ocurrioSancion</i> \leftarrow <i>True</i>	O(1)
<i>agente.siguiente.cantSanciones</i> + 1	O(1)
<i>agente.siguiente.miUbicacion.eliminarSiguiente()</i>	O(1)
// El iterador mismas apunta a la posicion correspondiente del agente dentro de la lista ordenada por cantSanciones	
// Como la lista en el peor caso puede contener a todos los agentes con igual cant de sanciones	
// la mayor cantidad posible de iteraciones del ciclo es 4	
while <i>agente.siguiente.mismas.haySiguiente()</i> \wedge <i>agente.siguiente.mismas.siguiente.cantSanciones</i> < <i>agente.siguiente.cantSanciones</i> do	

$agente.siguiete.mismas.avanzar()$	$O(1)$
end while	
// Si no hay siguiente o si la cantidad de sanciones del siguiente es menor que la del agente, entonces,	
// creo un conMismasBucket, lo inserto como siguiente y me guardo el iterador en miUbicacion	
// Sino, agrego el agente al conj de agentes del siguiente y me guardo el iterador en miUbicacion	
if $\neg(agente.siguiete.mismas.haySiguiete) \vee$	
$(agente.siguiete.mismas.haySiguiete \wedge$	
$agente.siguiete.cantSanciones = agente.siguiete.mismas.cantSanciones)$ then	
	$O(1)$
$nConMismasB \leftarrow nuevaTupla(CrearNuevoDiccLineal(), agente.siguiete.cantSanciones)$	
$agente.siguiete.mismas \leftarrow agente.siguiete.mismas.agregarComoSiguiete(nConMismasB)$	
	$O(1)$
$agente.siguiete.miUbicacion \leftarrow agente.siguiete.mismas.siguiete.agente.agregarComoSiguiete(agente.siguiete.pl)$	
	$O(1)$
else	
$agente.siguiete.mismas.siguiete.agente.agregarComoSiguiete(agente.siguiete.pl)$	
	$O(1)$
end if	
	<hr style="width: 100%; border: 0.5px solid black; margin-top: 5px;"/> $O(1)$
ATRAPADOPORAGENTE? (in $as : as$, in $pos : pos$) $\rightarrow res : bool$	
$vecinos \leftarrow as.campusEstatico.vecinos(pos)$	
$alMenos1Agente \leftarrow False$	
	$O(1)$
$i \leftarrow 0$	
if $\neg(encerrado?(pos, as.campusEstatico.vecinos(pos)))$ then	
$return false$	
end if	
// Veo si hay algun agente alrededor	
while $i < vecinos.tamano()$ do	
	$O(1)$
if $as.campus[vecinos[i].x][vecinos[i].y].hayAgente?$ then	
$return true$	
end if	
$i++$	
	$O(1)$
end while	
	<hr style="width: 100%; border: 0.5px solid black; margin-top: 5px;"/> $O(1)$
HIPPIFICARESTUDIANTESVECINOS (in/out $as : as$, in $pos : pos$)	
$vecinos \leftarrow as.campusEstatico.vecinos(pos)$	
	$O(1)$
$i \leftarrow 0$	
	$O(1)$
while $i < vecinos.tamano()$ do	
	$O(long(nombre))$
if $estAHippie?(as, vecinos[i])$ then	
$hippificar(as, vecinos[i])$	
	$O(long(nombre))$
end if	
$i++$	
	$O(1)$
end while	
	<hr style="width: 100%; border: 0.5px solid black; margin-top: 5px;"/> $O(long(nombre))$
HIPPIFICAR (in/out $as : as$, in $pos : pos$)	
// PRE: La posicion esta en el tablero y hay estudiante en la posicion	
$as.campus[pos.x][pos.y].hayHippie \leftarrow True$	
	$O(1)$
$as.campus[pos.x][pos.y].hippie.agregarComoSiguiete(nombre, pos)$	
	$O(long(nombreEstudiante))$

<i>as.campus[pos.x][pos.y].hayEst</i> \leftarrow <i>False</i>	O(1)
<i>as.campus[pos.x][pos.y].estudiante.eliminarSiguiente()</i>	O(long(nombreEstudiante))
<hr/>	
ESTAHIPPIE?(in <i>as</i> : as , <i>in pos</i> : pos) \longrightarrow <i>res</i> : bool	
if \neg (<i>encerrado?(pos, vecinos)</i>) then	
<i>return false</i>	O(1)
end if	
<i>i</i> \leftarrow 0	O(1)
<i>cantHippies</i> \leftarrow 0	O(1)
<i>vecinos</i> \leftarrow <i>as.campusEstatico.vecinos(pos)</i>	O(1)
while <i>i</i> < <i>vecinos.tamano()</i> do	
if <i>campus[vecinos[i].x][vecinos[i].y].hayHippie</i> then	
<i>cantHippies</i> ++	O(1)
end if	
<i>i</i> ++	
end while	
<i>return cantHippies</i> \geq 2	O(1)
<hr/>	
	O(1)
HIPPIEAEST?(in <i>as</i> : as , <i>in pos</i> : pos) \longrightarrow <i>res</i> : bool	
<i>i</i> \leftarrow 0	O(1)
<i>vecinos</i> \leftarrow <i>as.campusEstatico.vecinos(pos)</i>	O(1)
while <i>i</i> < <i>vecinos.tamano()</i> do	O(1)
if \neg (<i>as.campus[vecinos[i].x][vecinos[i].y].hayEst?</i>) then	
<i>return False</i>	O(1)
end if	
end while	
<i>return True</i>	
<hr/>	
	O(1)
ENCERRADO?(in <i>as</i> : as , <i>in pos</i> : pos)	
<i>vecinos</i> \leftarrow <i>vecinos(as.campusEstatico, pos)</i>	O(1)
<i>i</i> \leftarrow <i>vecinos.tamano()</i>	O(1)
while <i>i</i> < <i>vecinos.tamano()</i> do	O(1)
if \neg (<i>as.campus[vecinos[i].x][vecinos[i].y].hayAgente?</i> \vee	
<i>as.campus[vecinos[i].x][vecinos[i].y].hayEst?</i> \vee	
<i>as.campus[vecinos[i].x][vecinos[i].y].hayHippie?</i> \vee	
<i>as.campus[vecinos[i].x][vecinos[i].y].hayObst?</i>) then	O(1)
<i>return false</i>	O(1)
end if	
<i>i</i> ++	O(1)
end while	
<i>return true</i>	
<hr/>	
	O(1)
APLICARHIPPIESVECINOS(in/out <i>as</i> : as , <i>in pos</i> : pos)	
<i>vecinos</i> \leftarrow <i>as.campusEstatico.vecinos(pos)</i>	O(1)
<i>i</i> \leftarrow 0	O(1)
while <i>i</i> < <i>vecinos.tamano()</i> do	O(long(nombre))
<i>aplicarHippie(as, pos)</i>	O(long(nombre))
end while	

```

APLICARHIPPIE(in/out as : as, in pos : pos)
  // PRE: pos valida y hayHippie en as.campus[pos.x][pos.y]
  if as.campus[pos.x][pos.y].hayHippie then
    if as.hippieAEst(pos) then                                O(1)
      as.campus[pos.x][pos.y].hayHippie  $\leftarrow$  False          O(1)
      as.campus[pos.x][pos.y].hayEst  $\leftarrow$  True             O(1)
      as.campus[pos.x][pos.y].estudiante  $\leftarrow$  CrearIt(as.hippies)
                                                                O(1)
      as.campus[pos.x][pos.y].estudiante.agregarComoSiguiente(as.campus[pos.x][pos.y].estudiante.nombre)
                                                                O(long(nombre))
      as.campus[pos.x][pos.y].hippie.eliminarSiguiente()        O(long(nombre))
    else
      if as.campus[pos.x][pos.y].hayHippie?  $\wedge$  atrapadoPorAgente(pos) then
        vecinos  $\leftarrow$  as.campusSeguro.vecinos(pos)          O(1)
        i  $\leftarrow$  0                                           O(1)
        while i < vecinos.tamano() do                        O(1)
          posAct  $\leftarrow$  vecinos[pos.x][pos.y]                O(1)
          info  $\leftarrow$  as.campus[vecinos[i].x][vecinos[i].y]    O(1)
          if posAct.hayAgente then
            info.agente.siguiente.cantCapturas ++            O(1)
            // Actualizar mas vigilante
            if as.masVigilante.siguienteSignificado().cantCapturas < info.agente.siguienteSignificado()
              as.masVigilante  $\leftarrow$  info.agente            O(1)
            else
              if as.masVigilante.siguienteSignificado().cantCapturas = info.agente.siguienteSignificado()
                 $\wedge$  as.masVigilante.siguienteClave() < info.agente.siguienteClave() then
                  as.masVigilante  $\leftarrow$  info.agente        O(1)
                end if
              end if
            end if
            i ++
          end while
          as.campus[pos.x][pos.y].hayHippie? = False          O(1)
          as.campus[pos.x][pos.y].hippie.eliminarSiguiente()    O(long(nombre))
        end if
      end if
    end if
  end if

```

```

PROXPOSHIPPIE(in/out as : as, in nombre : string)  $\longrightarrow$  res : pos
  // PRE: El nombre es un hippie y el hippie no esta encerrado
  posHippie  $\leftarrow$  as.hippies.obtener(nombre)                O(long(nombre))
  if as.estudiantes.tamano() > 0 then
    // Retorna de las posiciones mas cercanas, la que esta mas cerca del (0,0)
    proxPos  $\leftarrow$  aPosMasCercana(as.campusEstatico, posHippie, as.estudiantes.significados)
                                                                O( $N_e$ )
  else

```

// Retorna el ingreso mas cercan, en caso de empate, el de abajo	
$proxPos \leftarrow aIngresoMasCercano(as.campusEstatico, posHippie)$	$O(1)$
end if	
$res \leftarrow proxPos$	$O(1)$
	<hr/>
	$O(N_e)$
PROXPOSAGENTE(in/out $as : as$, in $posAgente : pos$) $\longrightarrow res : pos$	
// PRE: En la posicion hay un agente que se puede mover	
if $as.hippies.tamano() > 0$ then	
// Retorna de las posiciones mas cercanas, la que esta mas cerca del (0,0)	
$proxPos \leftarrow aPosMasCercana(as.campusEstatico, posAgente, as.hippies.significados)$	$O(N_h)$
else	
// Retorna el ingreso mas cercano, en caso de empate, el de abajo	
$proxPos \leftarrow aIngresoMasCercano(as.campusEstatico, posAgente)$	$O(1)$
end if	
$res \leftarrow proxPos$	$O(1)$
	<hr/>
	$O(N_h)$

2 Pos es tupla(x:Nat, y:Nat)

3 Placa es Nat

4 Nombre es String

5 Diccionario Rapido

Es un diccionario que dado una clave nat distribuida uniformemente, nos da su significado en promedio $O(1)$

5.1 Interfaz

parámetros formales

géneros Nat, β

se explica con DICCIONARIO(NAT, CONJ(β)), ITERADOR BIDIRECCIONAL

géneros diccR(Nat, conj(β))

usa Bool, Nat, Conjunto(β)

Operaciones

CREAR(in $dicc : Dicc(c:nat s:\beta)$) $\longrightarrow res : diccSR(Nat, \beta)$

Pre $\equiv \{true\}$

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

Descripción: Crea un diccionario rapido.

Complejidad: $O(n)$

Aliasing: No hay aliasing

OBTENER(**in/out** $v : \text{diccR}(\text{Nat}; \text{conj}(\alpha))$, **in** $p : \text{nat}$) $\longrightarrow res : \text{conj}(\alpha)$

Pre $\equiv \{\text{Definido?}(p, v)\}$

Post $\equiv \{\text{Obtener}(p, v)\}$

Descripción: Retorna el significado actual, para la clave dada.

Complejidad: $O(1)$

Aliasing: El retorno se hace por referencia, hay aliasing entre el objeto devuelto y el del contenedor

Las complejidades están en función de las siguientes variables:

n : la cantidad total de claves definidas en el diccionario.

5.2 Representación

se representa con estr

donde estr es $\text{tupla}(\text{accesoRapido} : \text{vector}(\text{acceso} : \text{Dicc}(\text{nat}, \text{itDicc}(\text{nat}, \beta))),$
 $\text{contenedor} : \text{Dicc}(\text{c} : \text{nat}, \text{s} : \beta))$

El contenedor esta implementado sobre Dicc lineal

Cada posicion del vector esta implementado sobre Dicc lineal

Invariante de representación

1. El diccionario que resulta de Definir ordenadamente los siguientes de cada (clave, sign) del dicc de cada posicion del vector, es igual al contenedor
2. El tamaño del vector es igual al tamaño del contenedor
3. La suma de todos los tamanios de los dicc en vector es igual al tamaño del contenedor

5.3 Algoritmos

INUEVO(**in** $\text{diccACopiar} : \text{dicc}(\text{c} : \text{Nat}, \text{s} : \beta)$) $\longrightarrow res : \text{diccR}()$

$it \leftarrow \text{CrearIt}(\text{diccACopiar})$ $O(1)$

// Inicializo el vector

while $it.\text{haySiguiente}$ **do**

$res.\text{accesoRapido}.\text{agregarAtras}(\text{NULL})$ $O(1)$

$it.\text{avanzar}()$ $O(1)$

end while

$it \leftarrow \text{CrearIt}(\text{diccACopiar})$ $O(1)$

// El iterador vuelve a apuntar al primer elemento, defino todos los elementos del vector en el hash

while $it.\text{haySiguiente}()$ **do** $O(\text{diccACopiar.tamano}())$

$res.\text{accesoRapido}[\text{fHash}(it.\text{siguienteClave}())].$

<i>Definir</i> (<i>c</i> , <i>res</i> . <i>contenedor</i> . <i>Definir</i> (<i>it</i> . <i>siguienteClave</i> (), <i>it</i> . <i>siguienteSignificado</i> ()))	$\theta(1)$
<i>it</i> . <i>avanzar</i> ()	$O(1)$
end while	<hr/>
	$\theta(n)$
IDAMEITERADOR (in <i>a</i> : <i>diccR</i>) \longrightarrow <i>res</i> : <i>itDiccR</i>	
<i>res</i> \leftarrow <i>CrearIt</i> (<i>a</i> . <i>contenedor</i>)	$O(1)$
	<hr/>
IOBTENER (in/out <i>a</i> : <i>diccR</i> , <i>in</i> <i>c</i> : <i>Nat</i>) \longrightarrow <i>res</i> : β	
<i>res</i> \leftarrow <i>a</i> . <i>accesoRapido</i> [<i>a</i> . <i>fHash</i> (<i>c</i>)]. <i>obtener</i> (<i>c</i>). <i>siguiente</i> ()	$\theta(1)$
	<hr/>
	$\theta(1)$

5.4 Operaciones privadas

FHASH (in <i>a</i> : <i>diccR</i> , <i>in</i> <i>c</i> : <i>nat</i>) \longrightarrow <i>res</i> : <i>Nat</i>	
<i>res</i> \leftarrow (<i>c</i> % <i>a</i> . <i>tamano</i> ())	$O(1)$
	<hr/>

5.5 Representación del iterador

se representa con *itDiccR* *itDicc*(*clave*:*nat*, *significado*: β)

6 Diccionario por nombres

6.1 Interfaz

se explica con *DICC*

usa

géneros *dpn*

Operaciones

VACIO() \longrightarrow *res* : *dpn*

Pre \equiv {*true*}

Post \equiv {*dpn* =_{obs} *vacía*()}

Descripción: Crea un nuevo diccionario

Complejidad:

Aliasing: $O(1)$

DEFINIDO?(**in/out** *d* : *dpn*, *in* *c* : *String*) \longrightarrow *res* : *bool*

Pre \equiv {*true*}

Post \equiv {*res* =_{obs} *def?*(*d*₀, *e*)}

Descripción: Indica si la clave tiene un significado

Complejidad:

Aliasing: $O(\text{long}(c))$

DEFINIR(**in/out** $d : \text{dpn}$, $\text{in } c : \text{String}$, $\text{in } e : \alpha$)
Pre $\equiv \{d = d_0\}$
Post $\equiv \{d =_{\text{obs}} \text{Definir}(d_0, e)\}$
Descripción: Se define e en el diccionario
Complejidad: No hay aliasing, se inserta por copia
Aliasing: $O(\text{long}(c))$

ELIMINAR(**in/out** $d : \text{dpn}$, $\text{in } c : \text{String}$)
Pre $\equiv \{d =_{\text{obs}} d_0 \wedge \text{definido?}(d, c)\}$
Post $\equiv \{d =_{\text{obs}} \text{eliminar}(d_0, c)\}$
Descripción:
Complejidad: $O(\text{long}(c))$
Aliasing: No hay aliasing

SIGNIFICADO(**in/out** $d : \text{dpn}$, $\text{in } c : \text{String}$) $\longrightarrow res : \alpha$
Pre $\equiv \{\text{def?}(d, c)\}$
Post $\equiv \{res =_{\text{obs}} \text{significado}(d, c)\}$
Descripción: Se retornan los significados
Complejidad: $O(\text{long}(c))$
Aliasing: Hay aliasing entre el objeto devuelto y el almacenado

ALISTA(**in/out** $d : \text{dpn}$, $\text{in } c : \text{String}$) $\longrightarrow res : \alpha$
Pre $\equiv \{\text{true}\}$
Post $\equiv \{ALista(res) =_{\text{obs}} \text{tuplasClaveDiccionario}(d)\}$
Descripción: Retorna tuplas ¡clave,significado¡del diccionario
Complejidad: $O(1)$
Aliasing: Retorna por referencia, hay aliasing

6.2 Representación

se representa con **estr**

donde **estr** es $\text{tupla}(\text{buckets} : \text{Vector}(\text{puntero}(\text{nodo})),$
 $\text{enLista} : \text{Lista}(\text{clave} : \text{String},$
 $\text{significado} : \alpha))$
 donde **Nodo** es $\text{tupla}(\text{hayS} : \text{bool},$
 $s : \alpha,$
 $\text{enLista} : \text{itLista}(\text{clave} : \text{String},$
 $\text{significado} : \alpha),$
 $\text{hijos} : \text{estr})$

Invariante de representación

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

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

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

1. El tamaño de buckets de **estr** es 256
2. El conjunto de claves de **estr** es igual al conjunto formado por cada prefijo obtenido al ir desde la raíz hasta un nodo con **hayS**=true

$\text{Abs} : \widehat{\text{estr}} e \longrightarrow \widehat{\text{dicc}} \quad \{\text{Rep}(e)\}$

$(\forall e : \widehat{\text{estr}})$
 $\text{Abs}(e) \equiv d : \widehat{\text{dicc}} \mid (\forall s : \text{String}) s \in e.\text{claves} =_{\text{obs}} \text{def?}(d, s) \wedge$
 $((\forall s : \text{String}) \text{Definido?}(d, s)) \Rightarrow_L \text{Definido?}(e, s) \wedge_L (\text{obtener}(d, s) =_{\text{obs}} \text{Significado}(e, s))$

Auxiliares

6.3 Algoritmos

VACIO() $\longrightarrow res : \text{dpn}$
 $res \leftarrow \text{CrearTupla}(\text{InicializarVector}(), \text{NULL})$

 $\text{O}(1)$

IDEFINIR(in/out d : dpn, in clave : String, in e : α) $\longrightarrow res : \text{dpn}$
 $nodoClave : \text{puntero}(\text{nodoClave}) \leftarrow \text{nuevoNodoClave}(\text{clave}, d.\text{claves}, \text{NULL})$
 $\text{O}(\text{long}(\text{clave}))$
 $nodo : \text{puntero}(\text{Nodo}) \leftarrow \text{NULL}$
 $i : \text{nat} \leftarrow 0$
// Por ref
 $\text{caracteres} \leftarrow d.\text{buckets}$ $\text{O}(1)$
if $\text{caracteres.esVacia}()$ **then** $\text{O}(1)$
 $\text{caracteres} = \text{CrearHijos}()$ $\text{O}(1)$
 $d.\text{bucket} \leftarrow \text{caracteres}$ $\text{O}(1)$
end if
while $i \leq \text{Longitud}(\text{clave})$ **do** $\text{O}(\text{long}(\text{clave}))$
 $nodo \leftarrow \text{caracteres}[\text{ord}(\text{clave}[i])]$ $\text{O}(1)$
// Por ref
 $\text{caracteres} \leftarrow nodo.\text{hijos}$ $\text{O}(1)$
if $\text{caracteres.esVacia}()$ **then** $\text{O}(1)$
 $\text{caracteres} = \text{CrearHijos}()$ $\text{O}(1)$
 $nodo.\text{hijos} \leftarrow \text{caracteres}$ $\text{O}(1)$
end if
 $i++$ $\text{O}(1)$
end while
 $nodo.\text{hayS} \leftarrow \text{True}$ $\text{O}(1)$
 $nodo.\text{significado} \leftarrow e$ $\text{O}(1)$
// Almaceno el iterador de lista al agregar atras la clave a la lista de claves del trie, por interfaz de listaEnlazada
 $nodo.\text{enLista} \leftarrow d.\text{claves.agAtras}(< \text{clave}, e >)$ $\text{O}(\text{long}(\text{clave}))$

 $\text{O}(\text{long}(\text{clave}))$

IELIMINAR(in/out d : dpn, in clave : String) $\longrightarrow res : \text{dpn}$
 $nodo : \text{puntero}(\text{Nodo}) \leftarrow \text{NULL}$
 $i : \text{nat} \leftarrow 0$
// Por ref
 $\text{caracteres} \leftarrow d.\text{buckets}$ $\text{O}(1)$
while $i \leq \text{Longitud}(\text{clave})$ **do** $\text{O}(\text{long}(\text{clave}))$
 $nodo \leftarrow \text{caracteres}[\text{ord}(\text{clave}[i])]$ $\text{O}(1)$
// Por ref
 $\text{caracteres} \leftarrow nodo.\text{hijos}$ $\text{O}(1)$
 $i++$ $\text{O}(1)$
end while

<i>nodo.hayS</i> \leftarrow <i>False</i>	O(1)
<i>nodo.enLista.eliminarSiguiente()</i>	O(1)
if <i>nodo.hijos</i> = <i>NULL</i> then	
// Elimina un puntero	
<i>borrar(nodo)</i>	O(1)
end if	
<hr/>	
	O(long(clave))
ISIGNIFICADO (in/out <i>d</i> : dpn , <i>in</i> <i>clave</i> : String) \longrightarrow <i>res</i> : α	
<i>nodo</i> : <i>puntero(Nodo)</i> \leftarrow <i>NULL</i>	O(1)
<i>buckets</i> : <i>puntero(Nodo)</i> \leftarrow <i>d.buckets</i>	O(1)
<i>i</i> \leftarrow 0	O(1)
while <i>i</i> \leq <i>Longitud(clave)</i> do	
<i>nodo</i> \leftarrow <i>buckets[ord(clave[i])]</i>	O(1)
<i>i</i> ++	
end while	
// Por ref	
<i>res</i> \leftarrow <i>nodo.significado</i>	O(1)
<hr/>	
	O(long(clave))
ICLAVES (in/out <i>d</i> : dpn) \longrightarrow <i>res</i> : Lista(String)	
<i>res</i> \leftarrow <i>d.claves</i>	O(1)
<hr/>	
	O(1)
IDEFINIDO? (in/out <i>d</i> : dpn , <i>in</i> <i>clave</i> : String) \longrightarrow <i>res</i> : bool	
<i>nodo</i> : <i>puntero(Nodo)</i> \leftarrow <i>NULL</i>	O(1)
<i>buckets</i> : <i>puntero(Nodo)</i> \leftarrow <i>d.buckets</i>	O(1)
<i>i</i> \leftarrow 0	O(1)
while <i>i</i> \leq <i>Longitud(clave)</i> do	O(long(clave))
<i>nodo</i> \leftarrow <i>buckets[ord(clave[i])]</i>	O(1)
if <i>nodo</i> = <i>NULL</i> then	O(1)
<i>return False</i>	O(1)
end if	
<i>i</i> ++	O(1)
end while	
<i>res</i> \leftarrow <i>nodo.hayS</i>	O(1)
<hr/>	
	O(long(clave))

6.4 Operaciones del iterador

CREARITERADOR(**in** *d* : **dpn**) \longrightarrow *res* : **itDPN**

Pre \equiv {*true*}

Post \equiv {*tuplasClaveSignificado(d)* =_{obs} *siguientes(res)* \wedge_L *aliasing(tuplasClaveSignificado(d), siguientes(res))*}

Descripción: Crea un iterador del diccionario por nombres

Complejidad: O(1)

Aliasing: Existe aliasing entre todas las tuplas ¡Clave, Significado! del dicc y siguientes del iterador

HAYSIGUIENTE(**in** *it* : **itDPN**) \longrightarrow *res* : **bool**

Pre \equiv {*true*}

Post \equiv {*res* =_{obs} *haySiguiente(it)*}

Descripción: Indica si hay siguiente

Complejidad: $O(1)$

HAYANTERIOR(**in** $it : \text{itDPN}$) $\longrightarrow res : \text{bool}$

Pre $\equiv \{true\}$

Post $\equiv \{res =_{\text{obs}} \text{hayAnterior}(it)\}$

Descripción: Indica si hay anterior

Complejidad: $O(1)$

SIGUIENTE(**in** $it : \text{itDPN}$) $\longrightarrow res : \langle \text{clave:String}, \text{significado}:\alpha \rangle$

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

Post $\equiv \{res =_{\text{obs}} \text{siguiente}(it)\}$

Descripción: Retorna el siguiente

Complejidad: $O(1)$

Aliasing: Hay aliasing

ANTERIOR(**in** $it : \text{itDPN}$) $\longrightarrow res : \langle \text{clave:String}, \text{significado}:\alpha \rangle$

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

Post $\equiv \{res =_{\text{obs}} \text{anterior}(it)\}$

Descripción: Retorna el anterior

Complejidad: $O(1)$

Aliasing: Hay aliasing

SIGUIENTECLAVE(**in** $it : \text{itDPN}$) $\longrightarrow res : \text{String}$

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

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

Descripción: Retorna la siguiente clave

Complejidad: $O(1)$

Aliasing: Hay aliasing

ANTERIORCLAVE(**in** $it : \text{itDPN}$) $\longrightarrow res : \text{String}$

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

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

Descripción: Retorna la clave anterior

Complejidad: $O(1)$

Aliasing: Hay aliasing

SIGUIENTESIGNIFICADO(**in** $it : \text{itDPN}$) $\longrightarrow res : \alpha$

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

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

Descripción: Retorna el siguiente significado

Complejidad: $O(1)$

Aliasing: Hay aliasing

ANTERIORESIGNIFICADO(**in** $it : \text{itDPN}$) $\longrightarrow res : \alpha$

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

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

Descripción: Retorna el significado anterior

Complejidad: $O(1)$

Aliasing: Hay aliasing

AVANZAR(**in/out** $it : \text{itDPN}$)

Pre $\equiv \{\text{HaySiguiente}(it) \wedge it =_{\text{obs}} it_0\}$

Post $\equiv \{\text{anteriores}(it_0) \bullet \text{primero}(\text{siguientes}(it_0)) =_{\text{obs}} \text{anteriores}(it) \wedge \text{fin}(\text{siguientes}(it_0)) =_{\text{obs}} \text{siguientes}(it)\}$

Descripción: Modifica el iterador, haciendolo avanzar una posicion

Complejidad: $O(1)$

RETROCEDER(**in/out** $it : \text{itDPN}$)

Pre $\equiv \{ \text{Hayanterior}(it) \wedge it =_{\text{obs}} it_0 \}$

Post $\equiv \{ \text{comienzo}(\text{anteriores}(it_0)) =_{\text{obs}} \text{anteriores}(it) \wedge \text{ultimo}(\text{anteriores}(it_0)) \bullet \text{siguientes}(it_0) =_{\text{obs}} \text{siguientes}(it) \}$

Descripción: Modifica el iterador, haciendolo retroceder una posicion

Complejidad: $O(1)$

6.5 Representación del iterador

se explica con ITERADOR DICCIONARIO

se representa con `itLista(<clave:String, significado:α>)`

6.6 Algoritmos del iterador

CREARITERADOR(**in** $d : \text{dpn}$) $\longrightarrow res : \text{itDPN}$

$res \leftarrow \text{NuevoItLista}(d.ALista())$ $O(1)$

$O(1)$

HAYSIGUIENTE(**in** $it : \text{itDPN}$) $\longrightarrow res : \text{bool}$

$res \leftarrow it.haySiguiente()$ $O(1)$

$O(1)$

HAYANTERIOR(**in** $it : \text{itDPN}$) $\longrightarrow res : \text{bool}$

$res \leftarrow it.hayAnterior()$ $O(1)$

$O(1)$

SIGUIENTE(**in** $it : \text{itDPN}$) $\longrightarrow res : \text{bool}$

$res \leftarrow it.Siguiente()$ $O(1)$

$O(1)$

ANTERIOR(**in** $it : \text{itDPN}$) $\longrightarrow res : \text{bool}$

$res \leftarrow it.Anterior()$ $O(1)$

$O(1)$

SIGUIENTECLAVE(**in** $it : \text{itDPN}$) $\longrightarrow res : \text{String}$

$res \leftarrow it.Siguiente().clave$ $O(1)$

$O(1)$

ANTERIORCLAVE(**in** $it : \text{itDPN}$) $\longrightarrow res : \text{String}$

$res \leftarrow it.Anterior().clave$ $O(1)$

$O(1)$

SIGUIENTESIGNIFICADO(**in** $it : \text{itDPN}$) $\longrightarrow res : \alpha$

$res \leftarrow it.Siguiente().significado$ $O(1)$

$O(1)$

ANTERIORSIGNIFICADO(**in** $it : \text{itDPN}$) $\longrightarrow res : \alpha$

$res \leftarrow it.Anterior().significado$ $O(1)$

AVANZAR(in/out <i>it</i> : itDPN)	O(1)
<i>it.avanzar()</i>	O(1)
RETROCEDER(in/out <i>it</i> : itDPN)	O(1)
<i>it.retroceder()</i>	O(1)
	O(1)

7 Campus

7.1 Interfaz

se explica con CAMPUS

usa

géneros campus

Operaciones

ARMARCAMPUS(**in** *ancho* : nat, *alto* : nat) \longrightarrow *res* : campus

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

Post $\equiv \{res =_{\text{obs}} \text{crearCampus}(ancho, alto)\}$

Descripción: Crea el campus, sin obstáculos

Complejidad: O(ancho x alto)

AGREGAROBS(**in/out** *c* : campus, *in* *p* : pos)

Pre $\equiv \{(c) \equiv (c_0)\}$

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

Descripción: Agrega un obstáculo al campus

Complejidad: O(1)

ALTO(**in** *c* : campus) \longrightarrow *res* : nat

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

Post $\equiv \{res \equiv \text{alto}(c)\}$

Descripción: Indica la cantidad de filas de c

Complejidad: O(1)

ANCHO(**in** *c* : campus) \longrightarrow *res* : nat

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

Post $\equiv \{res \equiv \text{alto}(c)\}$

Descripción: Indica la cantidad de columnas de c

Complejidad: O(1)

OCUPADA(**in** *c* : campus, *p* : pos) \longrightarrow *res* : bool

Pre $\equiv \{\text{PosValida}(c, p)\}$

Post $\equiv \{res \iff \pi_1(\text{grilla}(c)[\pi_1(p)][\pi_2(p)])\}$

Descripción: Comprueba si una posición está ocupada

Complejidad: O(1)

POSVALIDA(**in** $c : \text{campus}$, $p : \text{pos}$) $\longrightarrow res : \text{bool}$

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

Post $\equiv \{res \iff (\pi_1(p) < ancho(c) \wedge \pi_2(p) < alto(c))\}$

Descripción: Comprueba que una posición exista dentro del campus.

Complejidad: $O(1)$

ESINGRESO(**in** $c : \text{campus}$, $p : \text{pos}$) $\longrightarrow res : \text{bool}$

Pre $\equiv \{\text{PosValida}(c,p)\}$

Post $\equiv \{res \iff (\pi_2(p) = alto(c) - 1 \vee \pi_2(p) = 0)\}$

Descripción: Comprueba si una posición es un ingreso al campus.

Complejidad: $O(1)$

INGRESOSUP(**in** $c : \text{campus}$, $p : \text{pos}$) $\longrightarrow res : \text{bool}$

Pre $\equiv \{\text{PosValida}(c,p)\}$

Post $\equiv \{res \iff \pi_2(p) = 0\}$

Descripción: Comprueba si una posición es un ingreso superior al campus.

Complejidad: $O(1)$

INGRESOINF(**in** $c : \text{campus}$, $p : \text{pos}$) $\longrightarrow res : \text{bool}$

Pre $\equiv \{\text{PosValida}(c,p)\}$

Post $\equiv \{res \iff \pi_2(p) = alto(c) - 1\}$

Descripción: Comprueba si una posición es un ingreso superior al campus.

Complejidad: $O(1)$

DISTANCIA(**in** $c : \text{campus}$, $in\ p1 : \text{pos}$, $in\ p2 : \text{pos}$) $\longrightarrow res : \text{nat}$

Pre $\equiv \{\text{PosValida}(c,p1) \wedge \text{PosValida}(c,p2)\}$

Post $\equiv \{res \equiv distancia(p1,p2,c)\}$

Descripción: Comprueba si una posición es un ingreso inferior al campus.

Complejidad: $O(1)$

VECINOS(**in** $c : \text{campus}$, $in\ p : \text{pos}$) $\longrightarrow res : \text{conj}(\text{pos})$

Pre $\equiv \{\text{PosValida}(c,p)\}$

Post $\equiv \{res \equiv vecinos(p,c)\}$

Descripción: Devuelve el conjunto de vecinos de una posición.

Complejidad: $O(1)$ **APOSMA SCERCANA**(**in** $c : \text{campus}$, $in\ p : \text{pos}$, $in\ con : \text{conj}(\text{pos})$) $\longrightarrow res : \text{pos}$

Pre $\equiv \{\text{PosValida}(c,p) \wedge \#con > 0\}$

Post $\equiv \{(\forall p_1) \text{PosValida}(c,p_1) \Rightarrow_L Distancia(c,p) \leq Distancia(c,p_1)\}$

Descripción: Dado un conjunto de posiciones y una posiciones, da la más cercana a la posición dada.

Complejidad: $O(\#con)$ **AIINGRESOMAS CERCANO**(**in** $cs : \text{campusSeguro}$, $in\ p : \text{pos}$) $\longrightarrow res : \text{pos}$

Pre $\equiv \{\text{PosValida}(c,p)\}$

Post $\equiv \{\text{Pertenece}(res, Vecinos(cs.campus)) \wedge (\exists pi : \text{pos})(\forall pn : \text{pos})(EsIngreso(cs.campus, pi) \wedge EsIngreso(pn, cs.campus) \Rightarrow res = pi)\}$

Descripción: Devuelve la posición vecina

Complejidad: $O(1)$

Las complejidades están en función de las siguientes variables:

al : cantidad de filas del campus,

an : cantidad de columnas del campus.

7.2 Representación

se representa con `estr`

donde `estr` es `tupla⟨ancho : nat,
alto : nat,
grilla : arreglo(arreglo(tupla⟨Ocupado : bool,
EsObst : bool,
EsAgente : bool,
HiippieoEst : tupla⟨pl : nat,
nombre : string⟩`

Invariante de representación

1. El tamaño de todos los arreglos internos de la grilla es el mismo e igual al alto del campus
2. El tamaño del arreglo principal de la grilla es igual al ancho del campus

$\text{Rep} : \widehat{\text{campus}} \longrightarrow \text{boolean}$

$(\forall : \widehat{\text{campus}})$

$\text{Rep}() \equiv$

$|c.\text{grilla}| = c.\text{ancho} \wedge (\forall n : \text{nat}, n \in [0, c.\text{ancho})) |c.\text{grilla}[n]| = c.\text{alto}$

Función de abstracción

$\text{Abs} : \widehat{\text{campus}} \ c \longrightarrow \widehat{\text{Campus}} \quad \{\text{Rep}(c)\}$

$(\forall c : \widehat{\text{campus}})$

$\text{Abs}(c) \equiv cE : \widehat{\text{Campus}} \mid$

$\text{columnas}(cE) = c.\text{ancho} \wedge \text{filas}(cE) = c.\text{alto} \wedge (\forall p : \text{pos}, \text{PosValida?}(cE, p)) \text{Ocupada?}(cE, p) = \text{Ocupada?}(c, p)$

7.3 Algoritmos

IARMARCAMPUS (in <i>ancho</i> : nat, <i>in</i> <i>alto</i> : nat) \longrightarrow <i>res</i> : campus	
<i>res.ancho</i> \leftarrow <i>ancho</i>	O(1)
<i>res.alto</i> \leftarrow <i>alto</i>	O(1)
<i>res.grilla</i> \leftarrow <i>CrearArreglo</i> (<i>ancho</i>)	O(an)
<i>i</i> \leftarrow 0	O(1)
while <i>i</i> < <i>ancho</i> do	O(1)
<i>res.grilla</i> [<i>i</i>] \leftarrow <i>CREARARREGLO</i> (<i>alto</i>)	O(al * an)
<i>j</i> \leftarrow 0	O(an)
while <i>j</i> < <i>alto</i> do	O(al)
<i>res.grilla</i> [<i>i</i>][<i>j</i>] \leftarrow < <i>False</i> , <i>False</i> , <i>False</i> , < 0, "" > >	O(1)
end while	O(al)
end while	O(1)
<hr/>	
O(an * al)	
IAGREGAROBS (in/out <i>c</i> : campus, in <i>p</i> : pos)	
$\pi_1(c.grilla[p.X][p.Y]) \leftarrow True$	O(1)
$\pi_2(c.grilla[p.X][p.Y]) \leftarrow True$	O(1)
<hr/>	
O(1)	
IALTO (in <i>c</i> : campus) \longrightarrow <i>res</i> : nat	
<i>res</i> \leftarrow <i>c.alto</i>	O(1)
<hr/>	
O(1)	
IANCHO (in <i>c</i> : campus) \longrightarrow <i>res</i> : nat	
<i>res</i> \leftarrow <i>c.ancho</i>	O(1)
<hr/>	
O(1)	
IOcupADA (in <i>c</i> : campus, <i>in</i> <i>p</i> : pos) \longrightarrow <i>res</i> : bool	
<i>res</i> \leftarrow <i>c.grilla</i> [<i>p.X</i>][<i>p.Y</i>]. <i>Ocupado</i>	O(1)
<hr/>	
O(1)	
IPOSVALIDA (in <i>c</i> : campus, <i>in</i> <i>p</i> : pos) \longrightarrow <i>res</i> : bool	
<i>res</i> \leftarrow <i>p.X</i> < <i>c.ancho</i> \wedge <i>p.Y</i> < <i>c.alto</i>	O(1)
<hr/>	
O(1)	
IESINGRESO (in <i>c</i> : campus, <i>in</i> <i>p</i> : pos) \longrightarrow <i>res</i> : bool	
<i>res</i> \leftarrow <i>Y.p</i> = 0 \vee <i>Y.p</i> = <i>c.alto</i> - 1	O(1)
<hr/>	
O(1)	
INGRESOSUP (in <i>c</i> : campus, <i>in</i> <i>p</i> : pos) \longrightarrow <i>res</i> : bool	
<i>res</i> \leftarrow <i>Y.p</i> = 0	O(1)
<hr/>	
O(1)	
INGRESOINF (in <i>c</i> : campus, <i>in</i> <i>p</i> : pos) \longrightarrow <i>res</i> : bool	
<i>res</i> \leftarrow <i>Y.p</i> = <i>c.alto</i> - 1	O(1)
<hr/>	
O(1)	
IDISTANCIA (in <i>c</i> : campus, <i>in</i> <i>p1</i> : pos, <i>in</i> <i>p2</i> : pos) \longrightarrow <i>res</i> : nat	

```

resX ← 0
resY ← 0
if p1.X < p2.X then
    resX ← p2.X − p1.X
else
    resX ← p1.X − p2.X
end if
if p1.Y < p2.Y then
    resY ← p2.Y − p1.Y
else
    resY ← p1.Y − p2.Y
end if
res ← resX + resY

```

O(1)

IVECINOS(**in** *c* : campus, *in p* : pos) → *res* : conj(pos)

```

res ← Vacio()
pn ← < p.X, p.Y + 1 >
if PosValida(c, pn) then
    agregar(res, pn)
end if
pn ← < p.X, p.Y − 1 >
if PosValida(c, pn) then
    agregar(res, pn)
end if
pn ← < p.X + 1, p.Y >
if PosValida(c, pn) then
    agregar(res, pn)
end if
pn ← < p.X − 1, p.Y >
if PosValida(c, pn) then
    agregar(res, pn)
end if

```

O(1)

IAPOSMA SCERCANA(**in** *c* : campus, *in p* : pos, *in con* : conj(pos)) → *res* : pos

```

it ← CrearIt(con)
res ← it.siguiente()
while HaySiguiente?(it) do
    if Distancia(c, p, res) > Distancia(c, it.siguiente(), res) then
        res ← it.siguiente()
        while HayAnterior() do
            EliminarAnterior(it)
        end while
        Avanzar(it)
    else
        if Distancia(c, p, res) < Distancia(c, it.siguiente(), res) then
            EliminarSiguiente(it)
        else
            Avanzar(it)
        end if
    end if

```

```

end while
it ← CrearIt(con)
res ← it.siguiente()
while HaySiguiente?(it) do
  if Distancia(c, < 0, 0 >, p) > Distancia(c, it.siguiente(), < 0, 0 >) then
    res ← it.siguiente()
    while HayAnterior() do
      EliminarAnterior(it)
    end while
    Avanzar(it)
  else
    if Distancia(c, p, < 0, 0 >) < Distancia(c, it.siguiente(), < 0, 0 >) then
      EliminarSiguiente(it)
    else
      Avanzar(it)
    end if
  end if
end while
it ← CrearIt(con)
res ← it.siguiente()
while HaySiguiente?(it) do
  if p.Y < it.siguiente().Y then
    res ← it.siguiente()
    while HayAnterior() do
      EliminarAnterior(it)
    end while
    Avanzar(it)
  else
    if p.Y > it.siguiente().Y then
      EliminarSiguiente(it)
    else
      Avanzar(it)
    end if
  end if
end while

```

$O(\#con)$

```

INGRESOMASCERCANO(in p : pos, cs : campusSeguro) → res : pos
if p.Y ≤ c.alto/2 then
  if PosValida(cs.campus, < p.X, p.Y - 1 >) ∧ ¬HayAlgo(cs, < p.X, p.Y - 1 >) then
    res ← < p.X, p.Y - 1 >
  else
    if PosValida/c, < p.X + 1, p.Y > ∧ ¬HayAlgo(c, < p.X + 1, p.Y > then
      res ← < p.X + 1, p.Y >
    else
      if PosValida/c, < p.X - 1, p.Y > ∧ ¬HayAlgo(c, < p.X - 1, p.Y > then
        res ← < p.X - 1, p.Y >
      else
        res ← < p.X, p.Y + 1 >
      end if
    end if
  end if
end if
else

```

```

if  $PosValida(cs.campus, \langle p.X, p.Y + 1 \rangle) \wedge \neg HayAlgo(cs, \langle p.X, p.Y + 1 \rangle)$  then
     $res \leftarrow \langle p.X, p.Y - 1 \rangle$ 
else
    if  $PosValida/c, \langle p.X + 1, p.Y \rangle \wedge \neg HayAlgo(c, \langle p.X + 1, p.Y \rangle)$  then
         $res \leftarrow \langle p.X + 1, p.Y \rangle$ 
    else
        if  $PosValida/c, \langle p.X - 1, p.Y \rangle \wedge \neg HayAlgo(c, \langle p.X - 1, p.Y \rangle)$  then
             $res \leftarrow \langle p.X - 1, p.Y \rangle$ 
        else
             $res \leftarrow \langle p.X, p.Y - 1 \rangle$ 
        end if
    end if
end if
end if

```

$O(1)$

7.4 Servicios Usados

Del modulo ConjLog requerimos pertenece, buscar, menor, insertar y borrar en $O(\log(k))$.

Del modulo Diccionario Por Prefijos requerimos Def?, obtener en $O(L)$.