



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

1.1 Interfaz

se explica con `CAMPUSSEGURO`

usa

géneros `CampusSeguro`

Operaciones

`CAMPUS(in cs : campusSeguro) → res : campus`

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

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

Descripción: Devuelve el campus del campusSeguro ingresado.

Complejidad: $O(1)$

Aliasing: El conjunto se retorna por referencia, hay aliasing

`ESTUDIANTES(in cs : campusSeguro) → res : itDPN(String Pos)`

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

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

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

Complejidad: $O(1)$

Aliasing: Se devuelve un iterador al diccionario de estudiantes por nombre, hay aliasing

`HIPPIES(in cs : campusSeguro) → res : itDPN(String Pos)`

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

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

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

Complejidad: $O(1)$

Aliasing: Se devuelve un iterador al diccionario de hippies por nombre, hay aliasing

`AGENTES(in cs : campusSeguro) → res : itDiccR(Placa ConMismasBucket)`

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

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

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

Complejidad: $O(1)$

Aliasing: Se devuelve un iterador al diccionario de agentes por nombre, hay aliasing

`POSICIONESTUDIANTESYHIPPIES(in id : nombre, cs : campusSeguro) → res : pos`

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

`POSICIONAGENTE(in a : agente, cs : campusSeguro) → 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: $\theta(1)$

`CANTIDADSANCIONES(in a : agente, cs : campusSeguro) → res : nat`

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

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

Descripción: Devuelve la cantidad de sanciones del agente ingresado.

Complejidad: $\theta(1)$

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

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

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

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

Complejidad: $\theta(1)$

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

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

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

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

Complejidad: $O(1)$

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

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

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

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

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

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

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

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

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

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

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

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

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

Descripción: Mueve un estudiante en la direccion indicada.

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

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

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

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

Descripción: Mueve un hippie hacia el estudiante más cercano.

Complejidad: $O(\text{long}(\text{nombre}) + N_e)$

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

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

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

Descripción: Mueve un agente hacia el hippie más cercano.

Complejidad: $O(\text{long}(\text{nombre}) + \log N_a + N_h)$

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

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

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

Descripción: Devuelve la cantidad de hippies en el campus.

Complejidad: $O(1)$

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

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

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

Descripción: Devuelve la cantidad de estudiantes en el campus.

Complejidad: $O(1)$

$\text{MÁS VIGILANTE}(\text{in } cs : \text{campusSeguro}) \longrightarrow res : \text{agente}$

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

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

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

Complejidad: $O(1)$

Aliasing: El mas vigilante se retorna por copia

Las complejidades están en función de las siguientes variables:

c : es una instancia del `campusSeguro`,

p : es una posición,

nombre : representa la longitud más larga entre todos los nombres del `campusSeguro`, o el nombre depende el caso

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.

1.2 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(conMismasBucket)⟩
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 los it mismas en los significados del dicc de agentes, la contatenacion de los anteriores , el actual y los siguientes son iguales a porSanciones
14. Para todos los it mismas en los significados del dicc de agentes, la cantSanciones en el significado es igual a la cantSanciones en el siguiente del iterador
15. Los anteriores unidos con los siguientes del iterador MiUbicacion en cada bucket del DiccRapido de agentes es igual al conjunto de agentes al que apunta mismas
16. La union de todos los agentes en porSanciones es igual al conjunto de claves en el dicc de agentes
17. No hay interseccion entre los conjuntos de agentes en porSanciones
18. Si conKSanciones.ocurrioSancion, entonces, conKSanciones es una copia de la lista de porSanciones.
19. El mas vigilante apunta a un agente en el dicc de agentes
20. El mas vigilante es el agente que tiene mas cantCapturas y en caso de empate mayor nro de placa

Función de abstracción

$$\begin{aligned}
 \text{Abs} : \widehat{\text{sistema}} s &\longrightarrow \widehat{\text{CampusSeguro}} && \{\text{Rep}(s)\} \\
 (\forall s : \widehat{\text{sistema}}) & \\
 \text{Abs}(s) \equiv cs : \widehat{\text{CampusSeguro}} & \mid s.\text{campus} =_{\text{obs}} \text{campus}(cs) \wedge \\
 s.\text{estudiantes} =_{\text{obs}} \text{estudiantes}(cs) \wedge & \\
 s.\text{hippies} =_{\text{obs}} \text{hippies}(cs) \wedge & \\
 s.\text{agentes} =_{\text{obs}} \text{agentes}(cs) \wedge & \\
 ((\forall n : \text{nombre}) s.\text{hippies}.\text{definido}(n) \Rightarrow_{\text{L}} s.\text{hippies}.\text{obtener}(n) =_{\text{obs}} \text{posEstYHippie}(n, cs) \vee & \\
 (\forall n : \text{nombre}) s.\text{estudiantes}.\text{definido}(n) \Rightarrow_{\text{L}} s.\text{estudiantes}.\text{obtener}(n) =_{\text{obs}} \text{posEstYHippie}(n, cs)) & \\
 (\forall pl : \text{placa}) s.\text{agentes}.\text{definido}(pl) \Rightarrow_{\text{L}} s.\text{estudiantes}.\text{obtener}(pl).\text{pos} =_{\text{obs}} \text{posAgente}(pl, cs)) & \\
 (\forall pl : \text{placa}) s.\text{agentes}.\text{definido}(pl) \Rightarrow_{\text{L}} s.\text{estudiantes}.\text{obtener}(pl).\text{cantSanciones} =_{\text{obs}} \text{cantSanciones}(pl, cs)) & \\
 (\forall pl : \text{placa}) s.\text{agentes}.\text{definido}(pl) \Rightarrow_{\text{L}} s.\text{estudiantes}.\text{obtener}(pl).\text{cantCapturas} =_{\text{obs}} \text{cantCapturas}(pl, cs)) &
 \end{aligned}$$

1.3 Algoritmos

| | |
|--|--------------------------|
| CAMPUS(in <i>campus</i> : campusSeguro) \rightarrow <i>res</i> : campus | |
| <i>res</i> \leftarrow <i>campus.campusEstatico</i> | O(1) |
| | <hr/> |
| | O(1) |
| AGENTES(in <i>campus</i> : campusSeguro) \rightarrow <i>res</i> : itDiccRapido (<i>agente</i>) | |
| <i>res</i> \leftarrow <i>CrearItRapido</i> (<i>campus.agentes</i>) | O(1) |
| | <hr/> |
| | O(1) |
| ESTUDIANTES(in <i>campus</i> : campusSeguro) \rightarrow <i>res</i> : itDPN (< <i>estudiante</i> : <i>nombre</i> , <i>pos</i> : <i>pos</i> >) | |
| <i>res</i> \leftarrow <i>CrearItDPN</i> (<i>campus.estudiantes</i>) | O(1) |
| | <hr/> |
| | O(1) |
| HIPPIES(in <i>campus</i> : campusSeguro) \rightarrow <i>res</i> : itDPN (< <i>hippie</i> : <i>nombre</i> , <i>pos</i> : <i>pos</i> >) | |
| <i>res</i> \leftarrow <i>CrearItDPN</i> (<i>campus.hippies</i>) | O(1) |
| | <hr/> |
| | O(1) |
| POSESTUDIANTESYHIPPIES(in <i>campus</i> : campusSeguro , <i>in nombre</i> : nombre) \rightarrow <i>res</i> : pos | |
| if <i>campus.estudiantes.definido?</i> (<i>nombre</i>) then | O(long(<i>nombre</i>)) |
| <i>return res</i> \leftarrow <i>campus.estudiantes.obtener</i> (<i>nombre</i>) | O(long(<i>nombre</i>)) |
| end if | |
| if <i>campus.hippies.definido?</i> (<i>nombre</i>) then | O(long(<i>nombre</i>)) |
| <i>return res</i> \leftarrow <i>campus.hippies.obtener</i> (<i>nombre</i>) | O(long(<i>nombre</i>)) |
| end if | |
| | <hr/> |
| | O(long(<i>nombre</i>)) |
| POSAGENTE(in <i>campus</i> : campusSeguro , <i>in placa</i> : agente) \rightarrow <i>res</i> : pos | |
| <i>res</i> \leftarrow <i>campus.agentes.obtener</i> (<i>placa</i>). <i>pos</i> | $\theta(1)$ |
| | <hr/> |
| | $\theta(1)$ |
| CANTSANCIONES(in <i>campus</i> : campusSeguro , <i>in placa</i> : agente) \rightarrow <i>res</i> : pos | |
| <i>res</i> \leftarrow <i>campus.agentes.obtener</i> (<i>placa</i>). <i>cantSanciones</i> | $\theta(1)$ |
| | <hr/> |
| | $\theta(1)$ |
| CANTHIPPIESATRAPADOS(in <i>campus</i> : campusSeguro , <i>in placa</i> : agente) \rightarrow <i>res</i> : pos | |
| <i>res</i> \leftarrow <i>campus.agentes.obtener</i> (<i>placa</i>). <i>cantCapturas</i> | $\theta(1)$ |
| | <hr/> |
| | $\theta(1)$ |
| COMENZARRASTRILLAJE(in/out <i>campus</i> : campusSeguro , <i>in ce</i> : CampusEstatico , <i>in Agentes</i> : dicc (<i>placa pos</i>)) | |
| <i>campus.agentesPorPlaca</i> \leftarrow <i>CrearVector</i> (<i>Agentes.tamano</i> ()) | |
| <i>campus.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(<i>Ancho</i>) |
| while <i>j</i> < <i>ce.Alto</i> do | O(<i>Alto</i>) |
| <i>campus.Campus[i][j].HayHippie</i> \leftarrow <i>False</i> | O(1) |
| <i>campus.Campus[i][j].HayEst</i> \leftarrow <i>False</i> | O(1) |
| <i>campus.Campus[i][j].HayObst</i> \leftarrow <i>ce.Obstaculos[i][j]</i> | O(1) |

| | |
|---|--|
| $i \leftarrow i + 1$ | $O(1)$ |
| $j \leftarrow j + 1$ | $O(1)$ |
| end while | |
| end while | |
| $campus.Agentes \leftarrow CrearDiccRapido(Agentes)$ | $O(1)$ |
| $ItAgentes \leftarrow CrearItAgentes(campus.Agentes)$ | $O(1)$ |
| $MayorPlaca \leftarrow 0$ | $O(1)$ |
| $i \leftarrow 0$ | |
| while $ItAgente.HaySiguiente$ do | $O(\text{Cantidad Agentes} * \text{Cantidad Agentes})$ |
| // Copio el iterador y lo asigno al lugar correspondiente | |
| $nuevoIt = ItAgente$ | $O(1)$ |
| $campus.Campus[ItAgentes.Siguiente.pos.X][ItAgentes.Siguiente.pos.Y].HayAgente \leftarrow True$ | $O(1)$ |
| $campus.Campus[ItAgentes.Siguiente.pos.X][ItAgentes.Siguiente.pos.Y].agente \leftarrow nuevoIt$ | $O(1)$ |
| if $MayorPlaca < ItAgentes.siguienteClave()$ then | |
| $MayorPlaca \leftarrow ItAgentes.siguienteClave()$ | $O(1)$ |
| $MayorAgente \leftarrow ItAgentes()$ | $O(1)$ |
| end if | |
| $campus.agentesPorPlaca.insertarOrdenadoPorPlaca(< ItAgentes.siguienteClave(), ItAgentes.siguienteClave())$ | $O(1)$ |
| $ItAgentes.Avanzar$ | $O(1)$ |
| $i++$ | $O(1)$ |
| end while | |
| // Ordena el arreglo de agentes por placa | |
| $campus.Estudiante \leftarrow Vacio()$ | $O(1)$ |
| $campus.Hipie \leftarrow Vacio()$ | $O(1)$ |
| $campus.masVigilante \leftarrow MayorAgente$ | $O(1)$ |
| $campus.porSanciones \leftarrow CrearLista(Tupla < agentes \leftarrow Vacio(), \#sanciones \leftarrow 0 >)$ | $O(1)$ |
| $ItAgentesRapido \leftarrow dameIterador(campus.agentes)$ | $O(1)$ |
| while $ItAgentesRapido.HaySiguiente$ do | $O(\text{Cantidad Agentes})$ |
| $ItAgentesRapido.siguiente.mismas \leftarrow CrearIt(campus.porSanciones)$ | $O(1)$ |
| $ItAgentesRapido.siguiente.miUbicacion \leftarrow$ | |
| $campus.porSanciones.obtenerUltimo.Agentes.Agregar(ItAgentesRapido.siguienteClave())$ | $O(1)$ |
| $ItAgentesRapido.Avanzar$ | $O(1)$ |
| end while | |

$O((Ancho * Alto) + N_a * N_a)$

INGRESARESTUDIANTE(**in/out** $campus : campusSeguro$, in $nombre : string$, in $pos : pos$)

 // pos es un ingreso

$campus.agregarEstudiante(campus, pos, nombre)$ $O(\text{long}(\text{nombre}))$

 // Sanciono a los agentes que rodean a los estudiantes atrapados al ingresar uno nuevo

$campus.sancionarAgentesVecinos(campus, pos)$ $O(1)$

$campus.sancionarAgentesEncerrandoEstVecinos(campus, pos)$ $O(1)$

 // Hippificar al estudiante

if $campus.estAHipie?(campus, pos)$ **then**

$campus.hippiificar(campus, pos)$ $O(\text{long}(\text{nombre}))$

end if

```

// Convertir a los hippies vecinos que quedaron encerrados por 4 estudiantes o eliminar a los
que quedaron atrapados por agentes
campus.aplicarHippiesVecinos(campus, pos) O(long(nombre))
// Capturar hippie en pos actual
if campus.campus[pos.x][pos.y].hayHippie? then
    aplicarHippie(pos) O(long(nombre))
end if


---


O(long(nombre))

INGRESARHIPPIE(in/out campus : campusSeguro, in nombre : string, in pos : pos)
campus.agregarHippie(campus, pos, nombre) O(long(nombre))
campus.sancionarAgentesEncerrandoEstVecinos(campus, pos) O(1)
campus.aplicarHippie(campus, pos) O(long(nombre))
campus.aplicarHippiesVecinos(campus, pos) O(long(nombre))


---


O(long(nombre))

MOVERESTUDIANTE(in/out campus : campusSeguro, 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)

posVieja ← campus.estudiantes.obtener(nombre) O(long(nombre))
if ¬(campus.campusEstatico.seRetira(campus.campus, dir, posVieja)) then
    // Mover el estudiante
    proxPos ← campus.campusEstatico.proxPos(posVieja, dir) O(1)
    campus.campus[posVieja.x][posVieja.y].hayEst? ← False
    campus.campus[proxPos.x][proxPos.y].hayEst ← True O(1)
    campus.campus[proxPos.x][proxPos.y].estudiante ←
campus.campus[posVieja.x][posVieja.y].estudiante O(1)
campus.campus[posVieja.x][posVieja.y].estudiante ← NULL O(1)
    // Sancionar agentes vecinos y a los que encierran a est vecinos
    sancionarAgentesEncerrandoEstVecinos(campus, pos) O(1)
    sancionarAgentesVecinos(campus, pos) O(1)
    // Convertir a estudiantes los hippies vecinos o capturarlos
    aplicarHippiesVecinos(campus, pos) O(1)
else
    campus.campus[posVieja.x][posVieja.y].hayEst? ← False O(1)
    campus.campus[posVieja.x][posVieja.y].estudiante.eliminarSiguiente() O(long(nombre))
end if


---


O(long(nombre))

MOVERHIPPIE(in/out campus : campusSeguro, in nombre : string)
if ¬(encerrado?(campus, campus.hippies.obtener(nombre))) then
    // Obtener pos siguiente y actualizar posicion de hippie
    posVieja ← campus.hippies.obtener(nombre) O(long(nombre))
    posNueva ← proxPosHippie(campus, nombre) O(long(nombre) + Ne)
    campus.campus[posVieja.x][posVieja.y].hayHippie ← False O(1)

```

| | |
|---|---|
| <i>campus.campus</i> [<i>posNueva.x</i>][<i>posNueva.y</i>]. <i>hayHippie</i> \leftarrow <i>True</i> | $O(1)$ |
| <i>itHippie</i> \leftarrow <i>campus.campus</i> [<i>posVieja.x</i>][<i>posVieja.y</i>]. <i>hippie</i> | $O(1)$ |
| <i>campus.campus</i> [<i>posVieja.x</i>][<i>posVieja.y</i>]. <i>hippie</i> \leftarrow <i>NULL</i> | $O(1)$ |
| <i>campus.campus</i> [<i>posNueva.x</i>][<i>posNueva.y</i>]. <i>hippie</i> \leftarrow <i>itHippie</i> | $O(1)$ |
| // Sancionar agentes que rodean a los estudiantes que encierro <i>sancionarAgentesEncerrandoEstVecinos</i> (<i>campus</i> , <i>posNueva</i>) | $O(1)$ |
| // Capturar hippies encerrados <i>aplicarHippiesVecinos</i> (<i>campus</i> , <i>posNueva</i>) | $O(\text{long}(\text{nombre}))$ |
| // Hippificar estudiantes <i>hippificarEstudiantesVecinos</i> (<i>campus</i> , <i>posNueva</i>) | $O(\text{long}(\text{nombre}))$ |
| end if | $O(\text{long}(\text{nombre}) + N_e)$ |
| MOVERAGENTE (in/out <i>campus</i> : campusSeguro , <i>in placa</i> : placa) | |
| // Obtener pos siguiente y actualizar pos de agente <i>posVieja</i> \leftarrow <i>busquedaBinariaPorPlaca</i> (<i>campus.agentesPorPlaca</i> , <i>placa</i>). <i>pos</i> | $O(\log(N_a))$ |
| if $\neg(\text{encerrado?}(\text{campus}, \text{posVieja}))$ \wedge <i>campus.campus</i> [<i>posVieja.x</i>][<i>posVieja.y</i>]. <i>agente.siguienteSignificado</i> (). <i>cantSanciones</i> < 3 then | $O(1)$ |
| <i>proxPos</i> \leftarrow <i>campus.proxPosAgente</i> (<i>posVieja</i>) | $O(N_h)$ |
| <i>campus.campus</i> [<i>posVieja.x</i>][<i>posVieja.y</i>]. <i>hayAgente</i> \leftarrow <i>False</i> | $O(1)$ |
| <i>campus.campus</i> [<i>proxPos.x</i>][<i>proxPos.y</i>]. <i>hayAgente</i> \leftarrow <i>True</i> | $O(1)$ |
| <i>itAgente</i> \leftarrow <i>campus.campus</i> [<i>posVieja.x</i>][<i>posVieja.y</i>]. <i>agente</i> | $O(1)$ |
| <i>campus.campus</i> [<i>proxPos.x</i>][<i>proxPos.y</i>]. <i>agente</i> \leftarrow <i>itAgente</i> | $O(1)$ |
| <i>campus.campus</i> [<i>posVieja.x</i>][<i>posVieja.y</i>]. <i>agente</i> \leftarrow <i>NULL</i> | $O(1)$ |
| $\pi_2(\text{busquedaBinaria}(\text{campus.agentesPorPlaca}, \text{placa})).\text{pos}$ \leftarrow <i>proxPos</i> | $O(\log(N_a))$ |
| <i>sancionarAgentesEncerrandoEstVecinos</i> (<i>campus</i> , <i>proxPos</i>) | $O(1)$ |
| <i>aplicarHippiesVecinos</i> (<i>campus</i> , <i>proxPos</i>) | $O(\text{long}(\text{nombre}))$ |
| end if | $O(N_h + \log(N_a) + \text{long}(\text{nombre}))$ |
| CONMISMASANCIONES (in <i>campus</i> : campusSeguro , <i>in placa</i> : placa) \longrightarrow <i>res</i> : Conj (agente) | |
| <i>posAgente</i> \leftarrow <i>campus.agentes.obtener</i> (<i>placa</i>) | $O(\theta(1))$ |
| <i>res</i> \leftarrow <i>campus.campus</i> [<i>posAgente.x</i>][<i>posAgente.y</i>]. <i>agente.siguienteSignificado</i> (). <i>mismcampus.agentes</i> | $O(1)$ |
| $O(\theta(1))$ | |
| CONKSANCIONES (in <i>campus</i> : campusSeguro , <i>in k</i> : nat) \longrightarrow <i>res</i> : Conj (agente) | |

| | |
|---|----------------------------|
| $res \leftarrow \emptyset$ | $O(1)$ |
| if $campus.conKSanciones.ocurrioSancion$ then | |
| // 'Copio' la lista de porSanciones a un vector, así luego puedo hacer bus binaria sobre el | |
| $campus.conKSanciones \leftarrow CrearArreglo(campus.porSanciones.tamano())$ | $O(1)$ |
| $it \leftarrow CrearItLista(campus.porSanciones)$ | $O(1)$ |
| $i \leftarrow 0$ | $O(1)$ |
| while $it.haySiguiente$ do | $O(N_a)$ |
| $conKSanciones.arreglo[i].cantSanciones \leftarrow it.siguienteSignificado().cantSanciones$ | $O(1)$ |
| // Por referencia | |
| $conKSanciones.arreglo[i].conKSanciones \leftarrow it.siguienteSignificado().agentes$ | $O(1)$ |
| if $it.siguienteSignificado().cantSanciones = k$ then | |
| $res \leftarrow it.siguienteSignificado().agentes$ | $O(1)$ |
| end if | |
| $i++$ | $O(1)$ |
| $it.avanzar()$ | $O(1)$ |
| end while | |
| $return res$ | $O(1)$ |
| else | |
| $bb \leftarrow busquedaBinaria(conKSanciones.arreglo, k)$ | $O(\log(N_a))$ |
| if $\pi_1(bb)$ then | |
| $return res \leftarrow \pi_2(bb)$ | $O(1)$ |
| else | |
| $return res \leftarrow \emptyset$ | $O(1)$ |
| end if | |
| end if | |
| <hr/> | |
| | $O(N_a) \vee O(\log(N_a))$ |
| $MASVIGILANTE(\text{in } campus : campusSeguro) \longrightarrow res : placa$ | |
| $res \leftarrow campus.masVigilante.siguienteClave()$ | $O(1)$ |
| <hr/> | |
| | $O(1)$ |

1.4 Algoritmos operaciones auxiliares

| | |
|---|---------------------------------|
| $AGREGARESTUDIANTE(\text{in/out } campus : campusSeguro, \text{ in } pos : pos, \text{ in } nombre : nombre)$ | |
| $campus.campus[pos.x][pos.y].hayEst \leftarrow True$ | $O(1)$ |
| $campus.campus[pos.x][pos.y].estudiante \leftarrow definir(campus.estudiantes, nombre, pos)$ | $O(\text{long}(\text{nombre}))$ |
| <hr/> | |
| | $O(\text{long}(\text{nombre}))$ |
| $AGREGARHIPPIE(\text{in/out } campus : campusSeguro, \text{ in } pos : pos, \text{ in } nombre : nombre)$ | |
| $campus.campus[pos.x][pos.y].hayHippie \leftarrow True$ | $O(1)$ |
| $campus.campus[pos.x][pos.y].hippie \leftarrow definir(campus.hippies, nombre, pos)$ | $O(\text{long}(\text{nombre}))$ |
| <hr/> | |
| | $O(\text{long}(\text{nombre}))$ |
| $SANCIONARAGENTESVECINOS(\text{in/out } campus : campusSeguro, \text{ in } pos : pos)$ | |
| $vecinos \leftarrow campus.campusEstatico.vecinos(pos)$ | $O(1)$ |

```

if campus.atrapadoPorAgente?(pos) then
    while i < vecinos.tamano() do O(1)
        if campus.campus[vecinos[i].x][vecinos[i].y].hayAgente? then
            campus.sancionarAgente(vecinos[i].agente) O(1)
        end if
        i++
    end while
end if
O(1)


---


SANCIONARAGENTESENCERRANDOESTVECINOS(in/out campus : campusSeguro, in pos : pos)
    vecinos  $\leftarrow$  campus.campusEstatico.vecinos(pos) O(1)
    i  $\leftarrow$  0
    while i < vecinos.tamano do O(1)
        if campus.campus[vecinos[i].x][vecinos[i].y].hayEst  $\wedge$  atrapadoPorAgente?(campus, pos)
then O(1)
            sancionarAgentesVecinos(campus, pos) O(1)
        end if
        i++
    end while
O(1)


---


SANCIONARAGENTE(in/out campus : campusSeguro, in/out agente : itDiccRapido)
    campus.conKSanciones.ocurrioSancion  $\leftarrow$  True O(1)
    agente.siguiente.cantSanciones  $+$  1 O(1)
    agente.siguiente.miUbicacion.eliminarSiguiente() 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 agente.siguiente.mismacampus.haySiguiente()  $\wedge$  agente.siguiente.mismas.siguiente.cantSanciones <
agente.siguiente.cantSanciones do
        agente.siguiente.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.siguiente.mismas.haySiguiente)  $\vee$ 
(agente.siguiente.mismas.haySiguiente  $\wedge$ 
agente.siguiente.cantSanciones = agente.siguiente.mismas.cantSanciones) then
        O(1)
        nConMismasB  $\leftarrow$  nuevaTupla(CrearNuevoDiccLineal(), agente.siguiente.cantSanciones)
        agente.siguiente.mismas  $\leftarrow$  agente.siguiente.mismas.agregarComoSiguiente(nConMismasB)
        O(1)
        agente.siguiente.miUbicacion  $\leftarrow$  agente.siguiente.mismas.siguiente.agentes.agregarComoSiguiente(agente.siguiente.pl)
        O(1)
    else
        agente.siguiente.mismas.siguiente.agentes.agregarComoSiguiente(agente.siguiente.pl)
        O(1)
    end if
O(1)


---


O(1)

```

| | |
|---|---------------------------|
| ATRAPADOPORAGENTE? (in <i>campus</i> : campusSeguro , <i>in pos</i> : pos) \longrightarrow <i>res</i> : bool | |
| <i>vecinos</i> \leftarrow <i>campus.campusEstatico.vecinos(pos)</i> | |
| <i>alMenos1Agente</i> \leftarrow <i>False</i> | O(1) |
| <i>i</i> \leftarrow 0 | |
| if \neg (<i>encerrado?(pos, campus.campusEstatico.vecinos(pos))</i>) then | |
| <i>return false</i> | |
| end if | |
| // Veo si hay algun agente alrededor | |
| while <i>i</i> < <i>vecinos.tamano()</i> do | O(1) |
| if <i>as.campus[vecinos[i].x][vecinos[i].y].hayAgente?</i> then | |
| <i>return true</i> | |
| end if | |
| <i>i</i> ++ | O(1) |
| end while | |
| | <hr/> O(1) |
| HIPPIFICAR ESTUDIANTES VECINOS (in/out <i>campus</i> : campusSeguro , <i>in pos</i> : pos) | |
| <i>vecinos</i> \leftarrow <i>campus.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)) |
| if <i>estAHippie?(campus, vecinos[i])</i> then | |
| <i>hippificar(campus, vecinos[i])</i> | O(long(nombre)) |
| end if | |
| <i>i</i> ++ | O(1) |
| end while | |
| | <hr/> O(long(nombre)) |
| HIPPIFICAR (in/out <i>campus</i> : campusSeguro , <i>in pos</i> : pos) | |
| // PRE: La posicion esta en el tablero y hay estudiante en la posicion | |
| <i>as.campus[pos.x][pos.y].hayHippie</i> \leftarrow <i>True</i> | O(1) |
| <i>as.campus[pos.x][pos.y].hippie.agregarComoSiguiente(nombre, pos)</i> | 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/> O(long(nombre)) |
| ESTAHIPPIE? (in <i>campus</i> : campusSeguro , <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>campus.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>campus</i> : campusSeguro , <i>in pos</i> : pos) \longrightarrow <i>res</i> : bool | |

| | |
|---|-----------------|
| $i \leftarrow 0$ | O(1) |
| $vecinos \leftarrow \text{campus.campusEstatico.vecinos}(pos)$ | O(1) |
| while $i < vecinos.tamano()$ do | O(1) |
| if $\neg(as.campus[vecinos[i].x][vecinos[i].y].hayEst?)$ then | |
| $return\ False$ | O(1) |
| end if | |
| end while | |
| $return\ True$ | |
| <hr/> | |
| | O(1) |
| ENCERRADO?(in $campus : \text{campusSeguro}$, in $pos : pos$) | |
| $vecinos \leftarrow vecinos(as.campusEstatico, pos)$ | O(1) |
| $i \leftarrow vecinos.tamano()$ | O(1) |
| while $i < vecinos.tamano()$ do | O(1) |
| if $\neg(campus.campus[vecinos[i].x][vecinos[i].y].hayAgente? \vee$ | |
| $campus.campus[vecinos[i].x][vecinos[i].y].hayEst? \vee$ | |
| $campus.campus[vecinos[i].x][vecinos[i].y].hayHippie? \vee$ | |
| $campus.campus[vecinos[i].x][vecinos[i].y].hayObst?)$ then | O(1) |
| $return\ false$ | O(1) |
| end if | |
| $i++$ | O(1) |
| end while | |
| $return true$ | |
| <hr/> | |
| | O(1) |
| APLICARHIPPIESVECINOS(in/out $campus : \text{campusSeguro}$, in $pos : pos$) | |
| $vecinos \leftarrow campus.campusEstatico.vecinos(pos)$ | O(1) |
| $i \leftarrow 0$ | O(1) |
| while $i < vecinos.tamano()$ do | O(long(nombre)) |
| $aplicarHippie(campus, pos)$ | O(long(nombre)) |
| end while | |
| <hr/> | |
| | O(long(nombre)) |
| APLICARHIPPIE(in/out $campus : \text{campusSeguro}$, in $pos : pos$) | |
| // PRE: pos valida y hayHippie en campus.campus[pos.x][pos.y] | |
| if $campus.campus[pos.x][pos.y].hayHippie$ then | |
| if $as.hippieAEst(pos)$ then | O(1) |
| $campus : campusSeguro.campus[pos.x][pos.y].hayHippie \leftarrow False$ | O(1) |
| $campus : campusSeguro.campus[pos.x][pos.y].hayEst \leftarrow True$ | O(1) |
| $as.campus[pos.x][pos.y].estudiante \leftarrow CrearIt(campus.hippies)$ | O(1) |
| $campus.campus[pos.x][pos.y].estudiante.agregarComoSiguiente(campus.campus[pos.x][pos.y].estudiante)$ | O(long(nombre)) |
| $campus.campus[pos.x][pos.y].hippie.eliminarSiguiente()$ | O(long(nombre)) |
| else | |
| if $campus.campus[pos.x][pos.y].hayHippie? \wedge atrapadoPorAgente(pos)$ then | |
| $vecinos \leftarrow campus.campusSeguro.vecinos(pos)$ | O(1) |
| $i \leftarrow 0$ | O(1) |
| while $i < vecinos.tamano()$ do | O(1) |

```

    posAct ← vecinos[pos.x][pos.y]                                O(1)
    info ← campus.campus[vecinos[i].x][vecinos[i].y]            O(1)

    if posAct.hayAgente then
        info.agente.siguiente.cantCapturas ++                  O(1)
        // Actualizar mas vigilante
        if campus.masVigilante.siguienteSignificado().cantCapturas <
info.agente.siguienteSignificado().cantCapturas then
            campus.masVigilante ← info.agente                  O(1)
        else
            if campus.masVigilante.siguienteSignificado().cantCapturas =
info.agente.siguienteSignificado().cantCapturas
            ∧ campus.masVigilante.siguienteClave() < info.agente.siguienteClave() then
                campus.masVigilante ← info.agente              O(1)
            end if
        end if
    end if
    i ++
end while
campus.campus[pos.x][pos.y].hayHippie? = False
campus.campus[pos.x][pos.y].hippie.eliminarSiguiente()          O(1)
campus.campus[pos.x][pos.y].hippie.eliminarSiguiente()          O(long(nombre))
end if
end if
end if

```

O(long(nombre))

```

PROXPOSHIPPIE(in/out campus : campusSeguro, in nombre : string) → res : pos
// PRE: El nombre es un hippie y el hippie no esta encerrado
posHippie ← campus.hippies.obtener(nombre)                      O(long(nombre))
if campus.estudiantes.tamano() > 0 then
    // Retorna de las posiciones mas cercanas, la que esta mas cerca del (0,0)
    proxPos ← aPosMasCercana(campus.campusEstatico, posHippie, campus.estudiantes.significados)
    O(Ne)
else
    // Retorna el ingreso mas cercan, en caso de empate, el de abajo
    proxPos ← aIngresoMasCercano(campus.campusEstatico, posHippie)
    O(1)
end if
res ← proxPos                                                    O(1)

```

O(N_e)

```

PROXPOSAGENTE(in/out campus : campusSeguro, in posAgente : pos) → res : pos
// PRE: En la posicion hay un agente que se puede mover
if campus.hippies.tamano() > 0 then
    // Retorna de las posiciones mas cercanas, la que esta mas cerca del (0,0)
    proxPos ← aPosMasCercana(campus.campusEstatico, posAgente, campus.hippies.significados)
    O(Nh)
else

```



```

// Retorna el ingreso mas cercano, en caso de empate, el de abajo
proxPos ← aIngresoMasCercano(campus.campusEstatico, posAgente)
                                                    O(1)

end if
res ← proxPos
                                                    O(1)

```

```

AINGRESOMASCERCANO(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
  else
    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

```

```

IBUSQUEDABINARIAPORSANCIONES(in ar : arreglo(val:nat otr: α>), in sanc : nat) → res
: < α, bool >
  res.π2 ← false
  min ← 0
  max ← |ar|
  while max - min > 1 do
    med ← (max + min)/2
    if ar[med].val ≤ sanc then
      min ← med
    else
      max ← med
    end if
  end while

```

O(1)

O(log(|ar|))

```

if  $ar[min].val = sanc$  then
   $res \leftarrow \langle ar[min].otr, true \rangle$ 
end if

```

 $O(\log(|ar|))$

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** $dicc : \text{Dicc}(c:\text{nat } s:\beta) \longrightarrow res : \text{diccSR}(\text{Nat}, \beta)$)

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

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

Descripción: Crea un diccionario rapido.

Complejidad: $O(n)$

Aliasing: No hay aliasing

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

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

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 <i>diccACopiar</i> : <i>dicc</i> (<i>c</i> : <i>Nat</i> , <i>s</i> : β)) \longrightarrow <i>res</i> : <i>diccR</i> () | |
| <i>it</i> \leftarrow <i>CrearIt</i> (<i>diccACopiar</i>) | O(1) |
| // Inicializo el vector | |
| while <i>it.haySiguiente</i> do | |
| <i>res.accesoRapido.agregarAtras</i> (<i>NULL</i>) | O(1) |
| <i>it.avanzar</i> () | O(1) |
| end while | |
| <i>it</i> \leftarrow <i>CrearIt</i> (<i>diccACopiar</i>) | O(1) |
| // El iterador vuelve a apuntar al primer elemento, defino todos los elementos del vector en el hash | |
| while <i>it.haySiguiente</i> () do | O(<i>diccACopiar.tamano</i> ()) |
| <i>res.accesoRapido[fHash(it.siguienteClave())].</i> | |
| <i>Definir</i> (<i>c</i> , <i>res.contenedor.Definir</i> (<i>it.siguienteClave</i> (), <i>it.siguienteSignificado</i> ())) | $\theta(1)$ |
| <i>it.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.contenedor</i>) | <hr/> O(1) |
| IOBTENER (in/out <i>a</i> : <i>diccR</i> , in <i>c</i> : <i>Nat</i>) \longrightarrow <i>res</i> : β | |
| <i>res</i> \leftarrow <i>a.accesoRapido[a.fHash(c)].obtener</i> (<i>c</i>). <i>siguiente</i> () | <hr/> $\theta(1)$ |
| | $\theta(1)$ |

5.4 Operaciones privadas

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

5.5 Representación del iterador

se representa con $\text{itDiccRItitDicc}(\text{clave: nat, significado: } \beta_i)$

6 Diccionario por nombres

6.1 Interfaz

se explica con `DICC`

usa

géneros `dpn`

Operaciones

`VACIO()` $\longrightarrow res : \text{dpn}$

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

Post $\equiv \{dpn =_{\text{obs}} vacia()\}$

Descripción: Crea un nuevo diccionario

Complejidad:

Aliasing: $O(1)$

`DEFINIDO?(in/out $d : \text{dpn}$, in $c : \text{String}$)` $\longrightarrow res : \text{bool}$

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

Post $\equiv \{res =_{\text{obs}} def?(d_0, e)\}$

Descripción: Indica si la clave tiene un significado

Complejidad:

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

`DEFINIR(in/out $d : \text{dpn}$, in $c : \text{String}$, in $e : \alpha$)` $\longrightarrow res : \text{itDPN}$

Pre $\equiv \{d = d_0\}$

Post $\equiv \{d =_{\text{obs}} Definir(d_0, e)\}$

Descripción: Se define e en el diccionario

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

Aliasing: El est se inserta por copia, pero retorna un iterador, hay aliasing

`ELIMINAR(in/out $d : \text{dpn}$, in $c : \text{String}$)`

Pre $\equiv \{d =_{\text{obs}} d_0 \wedge definido?(d, c)\}$

Post $\equiv \{d =_{\text{obs}} eliminar(d_0, c)\}$

Descripción:

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

`SIGNIFICADO(in/out $d : \text{dpn}$, in $c : \text{String}$)` $\longrightarrow res : \alpha$

Pre $\equiv \{def?(d, c)\}$

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

Descripción: Se retornan los significados

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

Aliasing: Hay aliasing entre el objeto devuelto y el almacenado

6.2 Representación

se representa con *estr*

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

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

Función de abstracción

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

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

6.3 Algoritmos

$\text{VACIO}() \longrightarrow \text{res} : \text{dpn}$

$\text{res} \leftarrow \text{CrearTupla}(\text{InicializarVector}(), \text{NULL})$

$O(1)$

$\text{IDEFINIR}(\text{in/out } d : \text{dpn}, \text{ in } \text{clave} : \text{String}, \text{ in } e : \alpha) \longrightarrow \text{res} : \text{itDPN}$

$\text{nodoClave} : \text{puntero}(\text{nodoClave}) \leftarrow \text{nuevoNodoClave}(\text{clave}, d.\text{claves}, \text{NULL})$

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

$\text{nodo} : \text{puntero}(\text{Nodo}) \leftarrow \text{NULL}$

$i : \text{nat} \leftarrow 0$

// Por ref

$\text{caracteres} \leftarrow d.\text{buckets}$

$O(1)$

if $\text{caracteres.esVacia}()$ **then**

$O(1)$

$\text{caracteres} = \text{CrearHijos}()$

$O(1)$

$d.\text{bucket} \leftarrow \text{caracteres}$

$O(1)$

end if

while $i \leq \text{Longitud}(\text{clave})$ **do**

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

$\text{nodo} \leftarrow \text{caracteres}[\text{ord}(\text{clave}[i])]$

$O(1)$

| | |
|--|----------------|
| <i>// Por ref</i> | |
| <i>caracteres</i> \leftarrow <i>nodo.hijos</i> | O(1) |
| if <i>caracteres.esVacia()</i> then | O(1) |
| <i>caracteres</i> = <i>CrearHijos()</i> | O(1) |
| <i>nodo.hijos</i> \leftarrow <i>caracteres</i> | O(1) |
| end if | |
| <i>i</i> ++ | O(1) |
| end while | |
| <i>nodo.hayS</i> \leftarrow <i>True</i> | O(1) |
| <i>nodo.significado</i> \leftarrow <i>e</i> | O(1) |
| <i>// Almaceno el iterador de lista al agregar atras la clave a la lista de claves del trie, por interfaz de listaEnlazada</i> | |
| <i>nodo.enLista</i> \leftarrow <i>d.claves.agAtras(< clave, e >)</i> | O(long(clave)) |
| <i>res</i> \leftarrow <i>nodo.enLista</i> | O(long(clave)) |
| <hr/> | |
| | O(long(clave)) |
| IELIMINAR(in/out d : dpn, in clave : String) | |
| <i>nodo : puntero(Nodo)</i> \leftarrow <i>NULL</i> | |
| <i>i : nat</i> \leftarrow 0 | |
| <i>// Por ref</i> | |
| <i>caracteres</i> \leftarrow <i>d.buckets</i> | O(1) |
| while <i>i</i> \leq <i>Longitud(clave)</i> do | O(long(clave)) |
| <i>nodo</i> \leftarrow <i>caracteres[ord(clave[i])]</i> | O(1) |
| <i>// Por ref</i> | |
| <i>caracteres</i> \leftarrow <i>nodo.hijos</i> | O(1) |
| <i>i</i> ++ | 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 | |
| <i>// Elimina un puntero</i> | |
| <i>borrar(nodo)</i> | O(1) |
| end if | |
| <hr/> | |
| | O(long(clave)) |
| ISIGNIFICADO(in/out d : dpn, in clave : String) \rightarrow res : α | |
| <i>nodo : puntero(Nodo)</i> \leftarrow <i>NULL</i> | O(1) |
| <i>buckets : 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 | |
| <i>// Por ref</i> | |
| <i>res</i> \leftarrow <i>nodo.significado</i> | O(1) |
| <hr/> | |
| | O(long(clave)) |
| ICLAVES(in/out d : dpn) \rightarrow res : Lista(String) | |
| <i>res</i> \leftarrow <i>d.claves</i> | O(1) |
| <hr/> | |
| | O(1) |
| IDEFINIDO?(in/out d : dpn, in clave : String) \rightarrow res : bool | |

| | |
|---|--------------------------------|
| <i>nodo</i> : <i>puntero</i> (<i>Nodo</i>) \leftarrow <i>NULL</i> | $O(1)$ |
| <i>buckets</i> : <i>puntero</i> (<i>Nodo</i>) \leftarrow <i>d.buckets</i> | $O(1)$ |
| <i>i</i> \leftarrow 0 | $O(1)$ |
| while <i>i</i> \leq <i>Longitud</i> (<i>clave</i>) do | $O(\text{long}(\text{clave}))$ |
| <i>nodo</i> \leftarrow <i>buckets</i> [<i>ord</i> (<i>clave</i> [<i>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/> | |
| | $O(\text{long}(\text{clave}))$ |
| IALISTA (in/out <i>d</i> : <i>dpn</i> , <i>in clave</i> : <i>String</i>) \longrightarrow <i>res</i> : <i>Lista</i> < <i>clave significado</i> > | |
| <i>res</i> \leftarrow <i>d.enLista</i> | $O(1)$ |
| <hr/> | |

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* : *itDPN*) \longrightarrow *res* : *bool*

Pre \equiv {*true*}

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

Descripción: Indica si hay anterior

Complejidad: $O(1)$

SIGUIENTE(**in** *it* : *itDPN*) \longrightarrow *res* : <*clave:String, significado:* α >

Pre \equiv {*HaySiguiente*(*it*)}

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

Descripción: Retorna el siguiente

Complejidad: $O(1)$

Aliasing: Hay aliasing

ANTERIOR(**in** *it* : *itDPN*) \longrightarrow *res* : <*clave:String, significado:* α >

Pre \equiv {*HayAnterior*(*it*)}

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

Descripción: Retorna el anterior

Complejidad: $O(1)$

Aliasing: Hay aliasing

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

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

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

Descripción: Retorna la siguiente clave

Complejidad: $O(1)$

Aliasing: Hay aliasing

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

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

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

Descripción: Retorna la clave anterior

Complejidad: $O(1)$

Aliasing: Hay aliasing

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

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

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

Descripción: Retorna el siguiente significado

Complejidad: $O(1)$

Aliasing: Hay aliasing

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

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

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

Descripción: Retorna el significado anterior

Complejidad: $O(1)$

Aliasing: Hay aliasing

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

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

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

Descripción: Modifica el iterador, haciendolo avanzar una posicion

Complejidad: $O(1)$

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

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

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

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 $it : itDPN$) $\longrightarrow res : bool$ | |
| $res \leftarrow it.hayAnterior()$ | O(1) |
| | O(1) |
| SIGUIENTE(in $it : itDPN$) $\longrightarrow res : bool$ | |
| $res \leftarrow it.Siguiente()$ | O(1) |
| | O(1) |
| ANTERIOR(in $it : itDPN$) $\longrightarrow res : bool$ | |
| $res \leftarrow it.Anterior()$ | O(1) |
| | O(1) |
| SIGUIENTECLAVE(in $it : itDPN$) $\longrightarrow res : String$ | |
| $res \leftarrow it.Siguiente().clave$ | O(1) |
| | O(1) |
| ANTERIORCLAVE(in $it : itDPN$) $\longrightarrow res : String$ | |
| $res \leftarrow it.Anterior().clave$ | O(1) |
| | O(1) |
| SIGUIENTESIGNIFICADO(in $it : itDPN$) $\longrightarrow res : \alpha$ | |
| $res \leftarrow it.Siguiente().significado$ | O(1) |
| | O(1) |
| ANTERIORSIGNIFICADO(in $it : itDPN$) $\longrightarrow res : \alpha$ | |
| $res \leftarrow it.Anterior().significado$ | O(1) |
| | O(1) |
| AVANZAR(in/out $it : itDPN$) | |
| $it.avanzar()$ | O(1) |
| | O(1) |
| RETROCEDER(in/out $it : itDPN$) | |
| $it.retroceder()$ | 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 $\equiv \{res =_{\text{obs}} \text{crearCampus}(\text{ancho}, \text{alto})\}$

Descripción: Crea el campus, sin obstáculos

Complejidad: $O(\text{ancho} \times \text{alto})$

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 : campus, p : pos) \longrightarrow res : bool

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

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

Descripción: Comprueba que una posición exista dentro del campus.

Complejidad: $O(1)$

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

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

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

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

Complejidad: $O(1)$

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

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

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

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

Complejidad: $O(1)$

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

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

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

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

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

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

Complejidad: $O(\#con)$

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 $\langle ancho : \text{nat},$
 $\quad\quad\quad alto : \text{nat},$
 $\quad\quad\quad grilla : \text{arreglo}(\text{arreglo}(\text{bool})) \rangle$

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

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

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

Rep() \equiv

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

Función de abstracción

Abs : $\widehat{\text{campus}} \ c \longrightarrow \widehat{\text{Campus}}$

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

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

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

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

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> | O(al) |
| end while | O(1) |
| end while | |
| <hr/> | |
| O(an * al) | |
| IAGREGAROBS (in/out <i>c</i> : campus, in <i>p</i> : pos) | |
| <i>c.grilla</i> [<i>p.X</i>][<i>p.Y</i>] \leftarrow <i>True</i> | O(1) |
| <i>c.grilla</i> [<i>p.X</i>][<i>p.Y</i>] \leftarrow <i>True</i> | 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>] | 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
 $it \leftarrow CrearIt(con)$ 
 $res \leftarrow it.siguiete()$ 
while HaySiguiete?(it) do
    if  $Distancia(c, < 0, 0 >, p) > Distancia(c, it.siguiete(), < 0, 0 >)$  then
         $res \leftarrow it.siguiete()$ 
        while HayAnterior() do
             $EliminarAnterior(it)$ 
        end while
        Avanzar(it)
    else
        if  $Distancia(c, p, < 0, 0 >) < Distancia(c, it.siguiete(), < 0, 0 >)$  then
             $EliminarSiguiete(it)$ 
        else
            Avanzar(it)
        end if
    end if
end while
 $it \leftarrow CrearIt(con)$ 
 $res \leftarrow it.siguiete()$ 
while HaySiguiete?(it) do
    if  $p.Y < it.siguiete().Y$  then
         $res \leftarrow it.siguiete()$ 
        while HayAnterior() do
             $EliminarAnterior(it)$ 
        end while
        Avanzar(it)
    else
        if  $p.Y > it.siguiete().Y$  then
             $EliminarSiguiete(it)$ 
        else
            Avanzar(it)
        end if
    end if
end while

```

 $O(\#con)$

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 $O(L)$.