



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	9
2. Diccionario Rapido	12
2.1. Interfaz	12
2.2. Representación	13
2.3. Algoritmos	13
2.4. Servicios Usados	14
3. Diccionario por nombres	14
3.1. Interfaz	14
3.2. Representación	15
3.3. Algoritmos	16
3.4. Operaciones del iterador	17
3.5. Representación del iterador	19
3.6. Algoritmos del iterador	19
4. Campus	20
4.1. Interfaz	20
4.2. Representación	21
4.3. Algoritmos	24
4.4. Servicios Usados	25

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

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: $O(1)$

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

Invariante de representación

1. En cada posicion de campus hay como máximo una entidad (agente, estudiante, hippie, obstaculo)
2. Si hayEst, hayHippie o hayAgente es true en alguna posición, entonces el iterador correspondiente debe tener siguiente y apuntar a un lugar en el contenedor correspondiente
3. No puede haber dos iteradores que apunten a lo mismo
4. La cantidad de agentes, hippies y estudiantes en campus debe ser igual al tamaño de su correspondiente contenedor
5. MasVigilante es el que mas hippiesCapturados tiene. En caso de empate, el que mayor nro de placa tiene
6. El conjunto de todos los agentes en porSanciones es igual a las claves del dicc de agentes
7. Si ocurrio sancion, el conjunto de agentes formado por la unión de los conjuntos en cada posicion de conKSanciones es igual a las claves del dicc de agentes
8. porSanciones está ordenado por #sanciones y en caso de empate por nro de placa
9. Si ocurrio sancion, entonces, conKSanciones es 'una copia' (sin iteradores y pasando de lista de agentes a conj) de la lista de porSanciones
10. conKSanciones esá ordenado por #sanciones y en caso de empate por nro de placa

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

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

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

Función de abstracción

$\text{Abs} : \widehat{\text{dcnet}} \ s \longrightarrow \widehat{\text{DCNet}} \qquad \{\text{Rep}(s)\}$
 $(\forall s : \widehat{\text{dcnet}})$
 $\text{Abs}(s) \equiv dc : \widehat{\text{DCNet}} \mid$
 $red(dc) =^*(s.red) \wedge (\forall c : compu, c \in compus(dc)) (enEspera(dc, c) =^*(enEspera(s, c)) \wedge$
 $cantidadEnviados(dc, c) = cantidadEnviados(s, c)) \wedge$
 $(\forall p : paquete, paqueteEnTransito?(dc, p)) caminoRecorrido(dc, p) =^*(caminoRecorrido(s, p))$

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/>	
AGENTES(in <i>as</i> : as) \longrightarrow <i>res</i> : itDiccSuperRapido (agente)	O(1)
<i>res</i> \leftarrow <i>CrearItSuperRapido</i> (<i>as.agentes</i>)	O(1)
<hr/>	
ESTUDIANTES(in <i>as</i> : as) \longrightarrow <i>res</i> : itDPN (< estudiante : String , pos : pos >)	O(1)
<i>res</i> \leftarrow <i>CrearItDPN</i> (<i>as.estudiantes</i>)	O(1)
<hr/>	
HIPPIES(in <i>as</i> : as) \longrightarrow <i>res</i> : itDPN (< hippie : String , pos : pos >)	O(1)
<i>res</i> \leftarrow <i>CrearItDPN</i> (<i>as.estudiantes</i>)	O(1)
<hr/>	
POSESTUDIANTESYHIPPIES(in <i>as</i> : as , <i>in nombre</i> : string) \longrightarrow <i>res</i> : pos	
if <i>as.estudiantes.definido?</i> (<i>nombre</i>) then	O(long(nombre))
<i>return res</i> \leftarrow <i>as.estudiantes.obtener</i> (<i>nombre</i>)	O(long(nombre))
end if	
if <i>as.hippies.definido?</i> (<i>nombre</i>) then	O(long(nombre))
<i>return res</i> \leftarrow <i>as.hippies.obtener</i> (<i>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.dameS</i> (<i>placa</i>). <i>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.dameS</i> (<i>placa</i>). <i>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.dameS</i> (<i>placa</i>). <i>cantHippiesAtrapados</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</i> , <i>pos</i> , <i>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</i> , <i>pos</i>)	O(1)
<hr/>	
// Hippificar al estudiante	
if <i>as.estAHippie?</i> (<i>as</i> , <i>pos</i>) then	
<i>as.hippificar</i> (<i>as</i> , <i>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>	$O(\text{long}(\text{nombre}))$
if <i>as.campus[pos.x][pos.y].hayHippie?</i> then	
<i>capturarHippie(pos)</i>	$O(\text{long}(\text{nombre}))$
end if	
	<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.CampusEstatico</i> \leftarrow <i>ce</i>	$O(1)$
<i>ItAgentes</i> \leftarrow <i>CrearItAgentes(Agentes)</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>CrearDiccSuperRapido()</i>	$O(1)$
<i>MayorPlaca</i> \leftarrow 0	$O(1)$
while <i>ItAgente.HaySiguiente</i> do	$O(\text{Cantidad Agentes})$
<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].Pl</i> \leftarrow	
<i>Definir(as.Agentes, ItAgentes.Siguiente.Pl, Tupla(Pos</i> \leftarrow <i>ItAgentes.Siguiente.Pos, CantSanciones</i> \leftarrow	
0, <i>CantCapturas</i> \leftarrow 0)	$O(1)$
if <i>MayorPlaca</i> < <i>ItAgentes.Siguientes.Pl</i> then	
<i>MayorPlaca</i> \leftarrow <i>ItAgenda.Siguiente.Pl</i>	$O(1)$
<i>MayorAgente</i> \leftarrow <i>ItAgenda.Siguiente</i>	$O(1)$
end if	
<i>ItAgentes.Avanzar</i>	$O(1)$
end while	
<i>as.Estudiante</i> \leftarrow <i>CrearDiccPorNombre()</i>	$O(1)$
<i>as.Hippie</i> \leftarrow <i>CrearDiccPorNombre()</i>	$O(1)$
<i>as.MasVigilante</i> \leftarrow <i>MayorAgente</i>	$O(1)$
<i>as.PorSanciones</i> \leftarrow <i>CrearLista(Tupla</i> < <i>Agentes</i> \leftarrow \emptyset , <i>#Sanciones</i> \leftarrow 0 >	$O(1)$
<i>ItAgentesRapido</i> \leftarrow <i>CrearItAgentesRapido()</i>	$O(1)$
while <i>ItAgentesRapido.HaySiguiente</i> do	$O(\text{Cantidad Agentes})$
<i>ItAgentesRapido.HaySiguiente.MiHubicacion</i> \leftarrow	
<i>as.PorSancion.ObtenerUltimo.Agentes.Agregar(ItAgentesRapido, SiguienteClave)</i>	$O(1)$
<i>ItAgentesRapido.Avanzar</i>	$O(1)$
end while	
	<hr/>
	$O((\text{Ancho} * \text{Alto}) + N_a)$

```

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)

posVieja ← as.estudiantes.obtener(nombre) O(long(nombre))
if ¬(as.campusEstatico.seRetira(as.campus, dir, posVieja)) then
    // Mover el estudiante
    as.campus[posVieja.x][posVieja.y].hayEst? ← False
    as.campus[as.campusEstatico.proxPos(posVieja, dir).x][as.campusEstatico.proxPos(posVieja, dir).y].
True
    as.campus[as.campusEstatico.proxPos(posVieja, dir).x][as.campusEstatico.proxPos(posVieja, dir).y].
as.campus[posVieja.x][posVieja.y].estudiante O(1)
    as.campus[posVieja.x][posVieja.y].estudiante ← NULL O(1)

    // Sancionar agentes vecinos y a los que encierran a est vecinos
    sancionarAgentesENCerrandoEstVecinos(as, pos) O(1)
    sancionarAgentesVecinos(as, pos) O(1)
    // Convertir a estudiantes los hippies vecinos o capturarlos
    aplicarHippiesVecinos(as, pos) O(1)
else
    as.campus[posVieja.x][posVieja.y].hayEst? ← False O(1)
    as.campus[posVieja.x][posVieja.y].estudiante.eliminarSiguiente O(long(nombre))
end if

```

O(long(nombre))

1.5 Algoritmos operaciones auxiliares

```

SANCIONARAGENTESVECINOS(in/out as : as, in pos : pos)
vecinos ← as.campusEstatico.vecinos(pos) O(1)
if as.atrapadoPorAgente?(pos) then
    while i < vecinos.tamano() do O(1)
        if as.campus[vecinos[i].x][vecinos[i].y].hayAgente? then
            as.sancionarAgente(vecinos[i].agente) O(1)
        end if
        i ++
    end while
end if

```

O(1)

```

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

```

```

SANCIONARAGENTE(in/out as : as, in/out agente : itDiccRapido)
  agente.siguiente.cantSanciones ← agente.siguiente.cantSanciones + 1
  agente.siguiente.miUbicacion.eliminarSiguiente()
  // 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.mismas.haySiguiente() ∧ agente.siguiente.mismas.siguiente.cantSanciones <
agente.siguiente.cantSanciones do
    agente.siguiente.mismas.avanzar()
  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 ¬(agente.siguiente.mismas.haySiguiente) ∨
(agente.siguiente.mismas.haySiguiente ∧
agente.siguiente.cantSanciones = agente.siguiente.mismas.cantSanciones) then
    nConMismasB ← nuevaTupla(CrearNuevoDiccLineal(), agente.siguiente.cantSanciones)
    agente.siguiente.mismas ← agente.siguiente.mismas.agregarComoSiguiente(nConMismasB)
    agente.siguiente.miUbicacion ← agente.siguiente.mismas.siguiente.agentes.agregarComoSiguiente(agente.siguiente.pl)
  else
    agente.siguiente.mismas.siguiente.agentes.agregarComoSiguiente(agente.siguiente.pl)
  end if

```

```

ATRAPADOPORAGENTE?(in as : as, in pos : pos) → res : bool
  vecinos ← as.campusEstatico.vecinos(pos)
  alMenos1Agente ← False
  i ← 0
  if ¬(encerrado?(pos, as.campusEstatico.vecinos(pos))) then
    return false
  end if
  // Veo si hay algun agente alrededor
  while i < vecinos.tamano() do
    if as.campus[vecinos[i].x][vecinos[i].y].hayAgente? then
      return true
    end if
    i ++
  end while

```

```

HIPPIFICAR(in 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 ← True

```

<i>as.campus[pos.x][pos.y].hippie.agregarComoSiguiente(nombre, pos)</i>	$O(\text{long}(\text{nombreEstudiante}))$
<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>	$O(\text{long}(\text{nombreEstudiante}))$
<hr/>	
	$O(\text{long}(\text{nombre}))$
ESTAHIPPIE? (<i>in as : as, in pos : pos, in vecinos : arreglo(pos)</i>) \rightarrow <i>res : bool</i>	
if $\neg(\text{encerrado?}(\text{pos}, \text{vecinos}))$ 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? (<i>in as : as, in pos : pos</i>) \rightarrow <i>res : bool</i>	
<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 $\neg(\text{as.campus}[\text{vecinos}[i].x][\text{vecinos}[i].y].\text{hayEst?})$ then	
<i>return False</i>	$O(1)$
end if	
end while	
<i>return True</i>	
<hr/>	
	$O(1)$
ENCERRADO? (<i>in as : as, in pos : pos</i>)	
<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 $\neg(\text{as.campus}[\text{vecinos}[i].x][\text{vecinos}[i].y].\text{hayAgente?} \vee$	
<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 (<i>in/out as : as, in pos : pos</i>)	
<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	$O(\text{long}(\text{nombre}))$

aplicarHippie(as, pos)	O(long(nombre))
end while	
<hr/>	
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].hippie \leftarrow CrearIt(as.hippies)	
	O(1)
as.campus[pos.x][pos.y].hippie.agregarComoSiguiente(as.campus[pos.x][pos.y].estudiante.nombre)	O(long(nombre))
as.campus[pos.x][pos.y].estudiante.eliminarSiguiente()	
	O(long(nombre))
else	
if as.campus[pos.x][pos.y].hayHippie? \wedge atrapadoPorAgente(pos) then	
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	
<hr/>	
	O(long(nombre))

2 Diccionario Rapido

Es un diccionario que dado un numero de placa como clave, nos da su significado en promedio O(1)

2.1 Interfaz

parámetros formales

géneros Nat, α

se explica con DICCIONARIO(NAT, CONJ(α))

géneros diccR(Nat, conj(α))

usa Bool, Nat, Conjunto(α)

Operaciones

CREAR(**in** n : nat) \longrightarrow res : diccR(Nat, α)

Pre \equiv {true}

Post \equiv {#Claves(res) =_{obs} n}

Descripción: Crea un diccionario rapido.

Complejidad: $O(n)$

Aliasing: Completar Aliasing

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

Pre $\equiv \{v =_{\text{obs}} v_0 \wedge \text{Definido?}(p, v)\}$

Post $\equiv \{\text{Definir}(p, \text{Ag}(\text{Obtener}(p, v_0), s), v)\}$

Descripción: Agrega el valor de s , al significado actual, para la clave dada

Complejidad: $O(1)$

Aliasing: Completar Aliasing

DAMES(**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: Completar Aliasing

Las complejidades están en función de las siguientes variables:

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

2.2 Representación

se representa con acceso

donde acceso es $\text{claves} : \text{arreglo}(\text{contenido})$

donde contenido es $\text{conj1}(\alpha)$

Aclaración: cada vez que dice arreglo en esta estructura nos referimos a arreglo_estatico y conj1 es conjunto lineal

Invariante de representación

1. Todos los índices del arreglo, pertenecen al conjunto de claves del diccionario sin repetidos.
2. Para todos los índices i del arreglo, contenido es igual al significado del diccionario para ese i .

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

$(\forall a : \widehat{\text{acceso}})$

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

1. $\forall p : \text{Nat} \text{ Definido?}(a, p) = \text{obtener}(\pi_1(c), s, \text{CompusPorPref})$

Función de abstracción

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

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

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

$\text{Abs}(s) \equiv dc : \widehat{\text{DCNet}} \mid$
 $\text{red}(dc) =^*(s.\text{red}) \wedge (\forall c : \text{compu}, c \in \text{compus}(dc))(\text{enEspera}(dc, c) =^*(\text{enEspera}(s, c)) \wedge$
 $\text{cantidadEnviados}(dc, c) = \text{cantidadEnviados}(s, c)) \wedge$
 $(\forall p : \text{paquete}, \text{paqueteEnTransito?}(dc, p)) \text{caminoRecorrido}(dc, p) =^*(\text{caminoRecorrido}(s, p))$

2.3 Algoritmos

ICREAR (in $r : \text{Nat}$) $\longrightarrow res : \text{diccR}()$	
$i \leftarrow 0$	$O(1)$
$p \leftarrow \text{CrearArreglo}(n)$	$O(n)$
while $i < n$ do	$O(n)$
$p[i] \leftarrow \text{vacio}()$	$O(1)$
$i++$	$O(1)$
end while	
$res \leftarrow p$	$O(1)$
	<hr/>
	$O(n)$
IASIGNAR (in/out $a : \text{acceso}$, <i>in</i> $p : \text{Nat}$, <i>in</i> $s : \alpha$)	
$a[\text{FhashPlaca}(p, a)] = \text{AgregarRapido}(a[\text{FhashPlaca}(p, a)], s)$	$O(1)$
	<hr/>
	$O(1)$
IDAMES (in/out $a : \text{acceso}$, <i>in</i> $p : \text{Nat}$) $\longrightarrow res : \text{contenido}$	
$res = a[\text{FhashPlaca}(p, a)]$	$O(1)$
	<hr/>
	$O(1)$

2.4 Servicios Usados

Del modulo ConjLineal

3 Diccionario por nombres

3.1 Interfaz

se explica con DICC

usa

géneros **dpn**

Operaciones

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

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

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

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* $e : \alpha$)

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

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

Descripción: Se define e en el diccionario

Complejidad: No hay aliasing, se inserta por copia

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

ELIMINAR(**in/out** $d : \text{dpn}$, *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: $O(\text{long}(c))$

Aliasing: No hay aliasing

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

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

Pre $\equiv \{true\}$

Post $\equiv \{ALista(res) =_{\text{obs}} tuplasClaveDiccionario(d)\}$

Descripción: Retorna tuplas ¡clave,significado! del diccionario

Complejidad: $O(1)$

Aliasing: Retorna por referencia, hay aliasing

3.2 Representación

se representa con *estr*

donde *estr* es *tupla*(*buckets* : *Vector*(*puntero*(*nodo*)),
 enLista : *Lista*(<*clave*:*String*,
 significado : α >))

donde *Nodo* es *tupla*(*hayS* : *bool*,
 s : α ,
 enLista : *itLista*(<*clave*:*String*,
 significado : α >),
 hijos : *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 raiz 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

3.3 Algoritmos

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

$\text{O}(1)$

$\text{IDEFINIR}(\text{in/out } d : \text{dpn}, \text{ in clave : String}, \text{ in } e : \alpha) \longrightarrow res : \text{dpn}$
 $nodoClave : \text{puntero}(\text{nodoClave}) \leftarrow \text{nuevoNodoClave}(\text{clave}, d.\text{claves}, \text{NULL})$
 $\text{O}(\text{long}(\text{clave}))$

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

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

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

<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.eliminarSiguiete()</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 <i>d</i> : dpn , <i>in 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	
<i>// Por ref</i>	
<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 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))

3.4 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** $it : \text{itDPN}$) $\longrightarrow res : \text{bool}$
Pre $\equiv \{true\}$
Post $\equiv \{res =_{\text{obs}} \text{haySiguiente}(it)\}$
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

ANTERIORSIGNIFICADO(**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 $\equiv \{anteriores(it_0) \bullet primero(siguients(it_0)) =_{\text{obs}} anteriores(it) \wedge fin(siguients(it_0)) =_{\text{obs}} siguients(it)\}$
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 $\equiv \{comienzo(anteriores(it_0)) =_{\text{obs}} anteriores(it) \wedge ultimo(anteriores(it_0) \bullet siguients(it_0)) =_{\text{obs}} siguients(it)\}$

Descripción: Modifica el iterador, haciendolo retroceder una posicion

Complejidad: $O(1)$

3.5 Representación del iterador

se explica con ITERADOR DICCIONARIO

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

3.6 Algoritmos del iterador

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

$res \leftarrow NuevoItLista(d.ALista())$

$O(1)$

$O(1)$

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

4 Campus

4.1 Interfaz

se explica con CAMPUS

usa

géneros campus

Operaciones

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

Pre $\equiv \{true\}$

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

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

Descripción: Agrega un obstáculo al campus

Complejidad: O(1)

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

Pre $\equiv \{true\}$

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

Descripción: Indica la cantidad de filas de c

Complejidad: O(1)

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

Pre $\equiv \{true\}$

Post $\equiv \{res \equiv ancho(c)\}$

Descripción: Indica la cantidad de columnas de c

Complejidad: O(1)

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

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

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

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

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

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

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

Descripción: devuelve el conjunto de vecinos de una posición.

Complejidad: $O(1)$

Las complejidades están en función de las siguientes variables:

al : cantidad de filas del campus,

an : cantidad de columnas del campus,

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

4.2 Representación

se representa con **estr**

donde **estr** es $\text{tupla}(\text{ancho} : \text{nat},$

$\text{alto} : \text{nat},$

$\text{grilla} : \text{arreglo}(\text{arreglo}(\text{tupla}(\text{Ocupado} : \text{bool}, , \text{tupla}(\text{pl} : \text{nat}, , \text{EsObst} : \text{bool}, \text{nombre} : \text{string})$

$\text{EsAgente} : \text{bool})$

Invariante de representación

1. Todos los IP de *compus* pertenecen al conjunto de claves de *CompusPorPref* y la longitud de dicho arreglo es igual al cardinal de las claves del diccionario.
2. Los pN de las tuplas que tiene el arreglo *compus* apuntan al conjunto de paquetes(PorNom) de un significado en *CompusPorPref* cuya clave es igual al IP de esa posición en el arreglo.
3. Los pN' apuntan al conjunto de paquetes(porNom') de un significado en *CompusPorPref* cuya clave es igual al IP de esa posición en el arreglo
4. Los paquetes del significado pN' son iguales a los paquetes de pN
5. El origen de pN' es distinto al destino de pN' y ambos son posiciones válidas del arreglo *compus*
6. PosActual de pN' es una posicion válida del arreglo *compus*
7. La *#PaquetesEnviados* de cada compu es mayor o igual a la actual cantidad total de paquetes que pasaron por esa compu
8. Todos los conjuntos de los significados de *CompusPorPref* son disjuntos dos a dos.
9. Los conjuntos de los campos de la tupla PorNom, PorPrior son iguales.
10. La matriz de caminosMinimos es cuadrada de lado n, con n igual al tamaño del arreglo de *compus*.
11. Para cualquier compu en el sistema f,d caminosMinimos[f][d] se corresponde con caminoMinimo(red,f,d)
12. La longitud de *CaminosMinimos* es igual a la longitud del arreglo que tiene *CaminosMinimos* en cada posición.
13. La longitud del arreglo, que tiene un arreglo de *CaminosMinimos* es menor o igual a la longitud de *CaminosMinimos*.
14. Los elementos del arreglo anteriormente mencionado son IPs del diccionario *CompusPorPref* y no tiene repetidos.
15. La computadora que más paquetes envió es aquella cuyo índice es igual a *LaQMasEnvio*

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

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

Rep(s) \equiv

1. $\forall s : \text{String } \text{def?}(s, s.\text{CompusPorPref}), (\exists c : \text{compu}), \text{esta?}(c, s.\text{Compus}) \wedge \pi_1(c) = s \wedge \text{longitud}(s.\text{Compus}) = \#\text{CLAVES}(s.\text{CompusPorPref})$
2. $\forall c : \text{compu } \text{esta?}(c, s.\text{Compus}), * \pi_2(c) = \text{obtener}(\pi_1(c), s.\text{CompusPorPref})$
3. $\forall c : \text{compu } \text{esta?}(c, s.\text{Compus}), * \pi_3(c) = \text{obtener}(\pi_3(c), s.\text{CompusPorPref})$
- 4, 5, 6.

($\forall c : \text{nat}$) $0 \leq c < \text{Longitud}(s.\text{compus}) \Rightarrow_L$

$\text{Longitud}(s.\text{compus}[c].\text{pN}) = \text{Longitud}(s.\text{compus}[c].\text{pN}') \wedge$

($\forall p : \text{paquetePos}$) $\text{esta?}(p, s.\text{compus}[c].\text{pN}') \Rightarrow_L$

$\text{esta}(\pi_1(p), s.\text{compus}[c].\text{pN}) \wedge 0 \leq \text{indiceOrigen}(p) < \text{Longitud}(s.\text{compus})$

$\wedge 0 \leq \text{indiceDestino}(p) < \text{Longitud}(s.\text{compus})$

$\wedge 0 \leq \text{posActual}(p) < \text{Longitud}(s.\text{compus})$

$\wedge \neg(\text{indiceDestino}(p) = \text{indiceOrigen}(p))$

7. ($\forall c : \text{nat}$) $0 \leq c < \text{Longitud}(s.\text{compus}) \Rightarrow_L$

($\forall p : \text{paquetePos}$) $\text{pertenece}(s.\text{compus}[c].\text{pN}', p) \Rightarrow_L$

- $\beta(\text{esta}(s.\text{compus}[c], \text{caminoMinimo}(s.\text{red}, s.\text{compus}[\text{indiceOrigen}(p)], s.\text{compus}[\text{posActual}(p)])))$
8. $\forall s, t : \text{String} \text{ def?}(s, s.\text{CompusPorPref}) \wedge \text{def?}(t, s.\text{CompusPorPref}) \wedge s \neq t \Rightarrow_L$
 $\text{obtener}(s, s.\text{CompusPorPref}) \cap \text{obtener}(t, s.\text{CompusPorPref}) = \emptyset$
 9. $\forall s : \text{String} \text{ def?}(s, s.\text{CompusPorPref}) \Rightarrow_L \pi_1(\text{obtener}(s, s.\text{CompusPorPref})) =$
 $\pi_2(\text{obtener}(s, s.\text{CompusPorPref}))$
 10. $\text{Longitud}(s.\text{compus}) = \text{Longitud}(\text{CaminosMinimos}(s)) \wedge$
 $(\forall i : \text{nat}) 0 \leq i < \text{Longitud}(s.\text{compus}) \Rightarrow_L$
 $\text{Longitud}(s.\text{CaminosMinimos}[i]) = \text{Longitud}(s.\text{compus})$
 11. $(\forall f, d : \text{nat}) \neg(f = d) \wedge 0 \leq f, d < \text{Longitud}(s.\text{compus}) \Rightarrow_L$
 $\text{CaminosMinimos}[f][d] =$
 $\text{caminoMinimo}(s.\text{red}, \text{ipACompu}(s.\text{red}, \pi_1(s.\text{compus}[f])), \text{ipACompu}(s.\text{red}, \pi_1(s.\text{compus}[d])))$
 - 12, 13, 14. $(\forall i, j : \text{nat}), 0 \leq i, j < \text{longitud}(s.\text{CaminosMinimos}) \Rightarrow_L \text{longitud}(s.\text{CaminosMinimos}) =$
 $\text{longitud}(s.\text{CaminosMinimos}[i]) \wedge \text{longitud}(s.\text{CaminosMinimos}[i][j]) < \text{longitud}(s.\text{CaminosMinimos}) \wedge$
 $(\forall e : \text{nat}), \text{esta?}(e, s.\text{CaminosMinimos}[i][j]) \Rightarrow \text{pertenece}(e, s.\text{CompusPorPref})$
 15. $\forall c : \text{compu} \text{ esta?}(c, s.\text{Compus}) \Rightarrow_L \pi_3(c) \leq \pi_3(s.\text{Compus}[s.\text{LaQMasEnvio}])$

Función de abstracción

$\text{Abs} : \widehat{\text{dcnet}} s \longrightarrow \widehat{\text{DCNet}} \quad \{\text{Rep}(s)\}$

$(\forall s : \widehat{\text{dcnet}})$
 $\text{Abs}(s) \equiv dc : \widehat{\text{DCNet}} \mid$
 $\text{red}(dc) =^*(s.\text{red}) \wedge (\forall c : \text{compu}, c \in \text{compus}(dc))(\text{enEspera}(dc, c) =^*(\text{enEspera}(s, c)) \wedge$
 $\text{cantidadEnviados}(dc, c) = \text{cantidadEnviados}(s, c)) \wedge$
 $(\forall p : \text{paquete}, \text{paqueteEnTransito?}(dc, p)) \text{caminoRecorrido}(dc, p) =^*(\text{caminoRecorrido}(s, p))$

4.3 Algoritmos

ICREARSISTEMA (in $r : \text{red}$) $\longrightarrow res : \text{dcnet}$	
$res.red \leftarrow r$	
$n \leftarrow \text{Longitud}(\text{COMPUS}(red))$	$O(1)$
$i \leftarrow 0$	
$j \leftarrow 0$	$O(1)$
$res.Compūs \leftarrow \text{CREARARREGLO}(n)$	$O(n)$
$res.CaminosMinimos \leftarrow \text{CREARARREGLO}(n)$	$O(n)$
var $p : \text{arreglo_dimensionable de puntero}(\text{conjLog}(\text{paquete}))$	
while $i < n$ do	$O(L * n^5)$
	$O(n)$
$res.CaminosMinimos[i] \leftarrow \text{CREARARREGLO}(n)$	$O(n)$
$s : < nat, conjLog(paquete, <_{id}), conjLog(paquete, <_p),$	
$conjLog(paquetePos, <_{id}), conjLog(paquetePos, <_p) >$	
$\pi_1(s) \leftarrow compu(r, i)$	
$\pi_2(s) \leftarrow \text{NUEVO}()$	
$\pi_3(s) \leftarrow \text{NUEVO}()$	
$\pi_4(s) \leftarrow \text{NUEVO}()$	
$\pi_5(s) \leftarrow \text{NUEVO}()$	
$\text{DEFINIR}(res.CompūsPorPref, compu(r, i), s)$	$O(L)$
$p[i] \leftarrow \pi_3(s)$	
$p'[i] \leftarrow \pi_5(s)$	
$res.Compūs[i] \leftarrow < compu(r, i), p[i], p'[i], 0 >$	$O(1)$
while $j < n$ do	$O(L * n^4)$
	$O(n)$
$res.CaminosMinimos[i][j] \leftarrow caminoMinimo(compu(r, i), compu(r, j), r)$	$O(L * n^3)$
$j++$	
end while	
$i++$	
end while	
$res.LaQMasEnvio \leftarrow 0$	$O(1)$
<hr/>	
	$O(L \times n^5)$
ICREARPAQUETE (in/out $s : \text{dcnet}$, in/out $p : \text{paquete}$)	
$t_1 : < nat, conjLog(paquete, <_{id}), conjLog(paquete, <_p),$	
$conjLog(paquetePos, <_{id}), conjLog(paquetePos, <_p) >$	
$t_1 \leftarrow \text{OBTENER}(\text{origen}(p), s.CompūsPorPref)$	$O(L)$
$t_2 : < nat, conjLog(paquete, <_{id}), conjLog(paquete, <_p),$	
$conjLog(paquetePos, <_{id}), conjLog(paquetePos, <_p) >$	
$t_2 \leftarrow \text{OBTENER}(\text{destino}(p), s.CompūsPorPref)$	$O(L)$
$p' : paquetePos$	
$\text{INDICEORIGEN}(p') \leftarrow \pi_1(t_1)$	$O(1)$
$\text{INDICEDESTINO}(p') \leftarrow \pi_1(t_2)$	$O(1)$
$\text{POSACTUAL}(p') \leftarrow 0$	
$\text{INSERTAR}(\pi_2(t), p)$	$O(\log(k))$
$\text{INSERTAR}(\pi_3(t), p)$	$O(\log(k))$
$\text{INSERTAR}(\pi_4(t), p')$	$O(\log(k))$
$\text{INSERTAR}(\pi_5(t), p')$	$O(\log(k))$
<hr/>	
	$O(L + \log(k))$

4.4 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 $O(L)$.