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 AS

usa

géneros `as`

1.2 Interfaz

se explica con CAMPUSSEGURO

usa

géneros `CampusSeguro`

Operaciones

CAMPUS(**in** cs : `campusSeguro`) $\longrightarrow$ res : `campus`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} campus(cs)\}$
Descripción: Devuelve el campus del campusSeguro ingresado.
Complejidad: O(1)

ESTUDIANTES(**in** cs : `campusSeguro`) $\longrightarrow$ res : `conj(nombre)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} estudiantes(cs)\}$
Descripción: Devuelve un conjunto con los estudiantes del campusSeguro ingresado.
Complejidad: O(1)

HIPPIES(**in** cs : `campusSeguro`) $\longrightarrow$ res : `conj(nombre)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} hippies(cs)\}$
Descripción: Devuelve un conjunto con los hippies del campusSeguro ingresado.
Complejidad: O(1)

AGENTES(**in** cs : `campusSeguro`) $\longrightarrow$ res : `conj(agentes)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} agentes(cs)\}$
Descripción: Devuelve un conjunto con los agentes del campusSeguro ingresado.
Complejidad: O(1)

POSICIONESTUDIANTESYHIPPIES(**in** id : `nombre`, cs : `campusSeguro`) $\longrightarrow$ res : `posicion`
Pre $\equiv$ $\{id \in (estudiantes(cs) \cup hippies(cs))\}$
Post $\equiv$ $\{res =_{\text{obs}} posEstudianteYHippie(id, cs)\}$
Descripción: Devuelve la posicion del estudiante o hippie ingresado.
Complejidad: O($|n_m|$)

POSICIONAGENTE(**in** a : `agente`, cs : `campusSeguro`) $\longrightarrow$ res : `posicion`
Pre $\equiv$ $\{a \in agentes(cs)\}$
Post $\equiv$ $\{res =_{\text{obs}} posAgente(a, cs)\}$
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 $\equiv \{res =_{\text{obs}} cantSanciones(a, cs)\}$
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 $\equiv \{res =_{\text{obs}} cantHippiesAtrapados(a, cs)\}$
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 $\equiv \{(\forall a : \text{agente})(def?(a, d) \Rightarrow_{\text{L}} (posValida?(obtener(a, d)) \wedge \neg ocupada?(obtener(a, d), c))) \wedge$
$(\forall a, a2 : \text{agente})((def?(a, d) \wedge def?(a2, d) \wedge a \neq a2) \Rightarrow_{\text{L}} obtener(a, d) \neq obtener(a2, d)\}$
Post $\equiv \{res =_{\text{obs}} comenzarRastrillaje(c, d)\}$
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 $\equiv \{res =_{\text{obs}} ingresarEstudiante(e, p, cs_0)\}$
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 $\equiv \{res =_{\text{obs}} ingresarHippie(e, p, cs_0)\}$
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 $\equiv \{res =_{\text{obs}} moverEstudiante(e, d, cs_0)\}$
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 $\equiv \{res =_{\text{obs}} moverHippie(h, cs_0)\}$
Descripción: Mueve un hippie hacia el estudiante más cercano.
Complejidad: O($|n_m| + N_e$)

moverAgente(**in** a : `nombre`, *in/out* cs : `campusSeguro`)
Pre $\equiv \{(cs) \equiv (cs_0) \wedge a \in agentes(cs) \wedge_{\text{L}} cantSanciones(a, cs) \leq 3 \wedge$
$\neg todasOcupadas?(vecinos(posAgente(a, cs), campus(cs)), cs)\}$
Post $\equiv \{res =_{\text{obs}} moverAgente(a, cs_0)\}$
Descripción: Mueve un agente hacia el hippie más cercano.
Complejidad: O($|n_m| + logN_a + N_h$)

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

Pre $\equiv$ {true}
Post $\equiv \{res =_{\text{obs}} cantHippies(cs)\}$
Descripción: Devuelve la cantidad de hippies en el campus.
Complejidad: O(1)

CantidadEstudiantes(**in** *cs* : `campusSeguro`) $\longrightarrow$ *res* : `nat`
Pre $\equiv$ {true}
Post $\equiv \{res =_{\text{obs}} cantEstudiantes(cs)\}$
Descripción: Devuelve la cantidad de estudiantes en el campus.
Complejidad: O(1)

MásVigilante(**in** *cs* : `campusSeguro`) $\longrightarrow$ *res* : `agente`
Pre $\equiv$ {true}
Post $\equiv \{res =_{\text{obs}} masVigilante(cs)\}$
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** `tupla`⟨CampusEstatico : `Campus`,
Campus : `arreglo(arreglo(tupla`⟨hayHippie : `bool`,))),
hayEst : `bool`,
hayAgente : `bool`,
hayObst : `bool`,
pl : `itLista(agente)`,
estudiante : `itDPN(tupla`⟨nombre : `string`,),
pos : `pos`⟩
hippie : `itDPN(tupla`⟨nombre : `String`,)⟩
pos : `pos`⟩
estudiantes : `DiccPorNombre(nombre:string, pos:pos)`,
hippies : `DiccPorNombre(nombre:string, pos:pos)`,
agentesPorPlaca : `arreglo(placa:placa, pos:pos)`,
agentes : `DiccSuperRapido(pl:nat,tupla`⟨pos : `pos`,),
cantSanc : `nat`,
cantCapturas : `nat`,
mismas : `itLista(conMismasBucket)`,
miUbicacion : `itLista(agente)`⟩
masVigilante : `agente:itDiccRapido`,
porSanciones : `Lista(conMismasBucket)`,
conKSanciones : `tupla`⟨ocurrioSancion : `bool`, ⟩
arreglo : `arreglo(tupla`⟨porKSanc : `conj(agente)`,)⟩
#Sanciones : `nat`⟩

donde `conMismasBucket` **es** `tupla`⟨agentes : `Conj(agente)`,
#Sanc : `nat`⟩

Invariante de representación

1. 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 it que apunta a los 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

Rep : $\widehat{\texttt{sistema}} \longrightarrow$ *boolean*

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

Rep(s) $\equiv$

Función de abstracción

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

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

1.4 Algoritmos

CAMPUS(**in** *as* : as) $\longrightarrow$ *res* : campus

$res \leftarrow as.campus$	O(1)
	O(1)

AGENTES(**in** *as* : as) $\longrightarrow$ *res* : itDiccRapido(agente)

$res \leftarrow CrearItRapido(as.agentes)$	O(1)
	O(1)

ESTUDIANTES(**in** *as* : as) $\longrightarrow$ *res* : itDPN(<estudiante:nombre,pos:pos>)

$res \leftarrow CrearItDPN(as.estudiantes)$	O(1)
	O(1)

HIPPIES(**in** *as* : as) $\longrightarrow$ *res* : itDPN(<hippie:nombre,pos:pos>)

$res \leftarrow CrearItDPN(as.hippies)$	O(1)
	O(1)

POSESTUDIANTESYHIPPIES(**in** *as* : as, *in nombre* : nombre) $\longrightarrow$ *res* : pos

if $as.estudiantes.definido?(nombre)$ **then**	O(long(nombre))
$return\ res \leftarrow as.estudiantes.obtener(nombre)$	O(long(nombre))
end if	
if $as.hippies.definido?(nombre)$ **then**	O(long(nombre))
$return\ res \leftarrow as.hippies.obtener(nombre)$	O(long(nombre))
end if	
	O(long(nombre))

POSAGENTE(**in** *as* : as, *in placa* : agente) $\longrightarrow$ *res* : pos

$res \leftarrow as.agentes.obtener(placa).pos$	$\theta(1)$
	$\theta(1)$

CANTSANCIONES(**in** *as* : as, *in placa* : agente) $\longrightarrow$ *res* : pos

$res \leftarrow as.agentes.obtener(placa).cantSanciones$	$\theta(1)$
	$\theta(1)$

CANTHIPPIESATRAPADOS(**in** *as* : as, *in placa* : agente) $\longrightarrow$ *res* : pos

$res \leftarrow as.agentes.obtener(placa).cantCapturas$	$\theta(1)$
	$\theta(1)$

INGRESARESTUDIANTE(**in/out** *as* : as, *in nombre* : string, *in pos* : pos)

$as.agregarEstudiante(as, pos, nombre)$	O(long(nombre))
// Sanciono a los agentes que rodean a los estudiantes atrapados al ingresar uno nuevo	
$as.sancionarAgentesVecinos(as, pos)$	O(1)
$as.sancionarAgentesEncerrandoEstVecinos(as, pos)$	O(1)
// Hippificar al estudiante	
if $as.estAHippie?(as, pos)$ **then**	
$as.hippificar(as, pos)$	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	

$as.aplicarHippiesVecinos(as, pos)$	O(long(nombre))
// Capturar hippie en pos actual	
if $as.campus[pos.x][pos.y].hayHippie?$ **then**	
$aplicarHippie(pos)$	O(long(nombre))
end if	
	O(long(nombre))

ComenzarRastrillaje(**in/out** *as* : `as`, *in ce* : `CampusEstatico`, *in Agentes* : `dicc(placa pos)`)

$as.agentesPorPlaca \leftarrow CrearVector(Agentes.tamanio())$	
$as.CampusEstatico \leftarrow ce$	O(1)
$i \leftarrow 0$	O(1)
$j \leftarrow 0$	O(1)
while $i < ce.Ancho$ **do**	O(Ancho)
while $j < ce.Alto$ **do**	O(Alto)
$as.Campus[i][j].HayHippie \leftarrow False$	O(1)
$as.Campus[i][j].HayEst \leftarrow False$	O(1)
$as.Campus[i][j].HayObst \leftarrow ce.Obstaculos[i][j]$	O(1)
$i \leftarrow i + 1$	O(1)
$j \leftarrow j + 1$	O(1)
end while	
end while	
$as.Agentes \leftarrow CrearDiccRapido(Agentes)$	O(1)
$ItAgentes \leftarrow CrearItAgentes(as.Agentes)$	O(1)
$MayorPlaca \leftarrow 0$	O(1)
$i \leftarrow 0$	
while $ItAgente.HaySiguiente$ **do**	O(Cantidad Agentes)
// Copio el iterador y lo asigno al lugar correspondiente	
$nuevoIt = ItAgente$	O(1)
$as.Campus[ItAgentes.Siguiente.pos.X][ItAgentes.Siguiente.pos.Y].HayAgente \leftarrow True$	O(1)
$as.Campus[ItAgentes.Siguiente.pos.X][ItAgentes.Siguiente.pos.Y].agente \leftarrow nuevoIt$	O(1)
if $MayorPlaca < ItAgentes.siguienteClave()$ **then**	
$MayorPlaca \leftarrow ItAgentes.siguienteClave()$	O(1)
$MayorAgente \leftarrow ItAgentes()$	O(1)
end if	
$as.agentesPorPlaca[i] \leftarrow ItAgentes.siguienteSignificado().pos$	O(1)
$ItAgentes.Avanzar$	O(1)
$i++$	O(1)
end while	
// Ordena el arreglo de agentes por placa	
$MergeSort(as.agentesPorPlaca)$	O(log(N_a)*N_a)
$as.Estudiante \leftarrow Vacio()$	O(1)
$as.Hippie \leftarrow Vacio()$	O(1)
$as.masVigilante \leftarrow MayorAgente$	O(1)
$as.porSanciones \leftarrow CrearLista(Tupla < agentes \leftarrow Vacio(), \#sanciones \leftarrow 0 >$	O(1)
$ItAgentesRapido \leftarrow dameIterador(as.agentes)$	O(1)
while $ItAgentesRapido.HaySiguiente$ **do**	O(Cantidad Agentes)
$ItAgentesRapido.siguiente.mismas \leftarrow CrearIt(as.porSanciones)$	O(1)

$ItAgentesRapido.siguiente.miUbicacion \leftarrow$
$as.porSanciones.obtenerUltimo.Agentes.Agregar(ItAgentesRapido.siguienteClave)$ O(1)

$ItAgentesRapido.Avanzar$ O(1)

end while

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

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)

$posVieja \leftarrow as.estudiantes.obtener(nombre)$ O(long(nombre))

if $\neg(as.campusEstatico.seRetira(as.campus, dir, posVieja))$ **then**

// Mover el estudiante

$proxPos \leftarrow as.campusEstatico.proxPos(posVieja, dir)$ O(1)

$as.campus[posVieja.x][posVieja.y].hayEst? \leftarrow False$

$as.campus[proxPos.x][proxPos.y].hayEst \leftarrow True$ O(1)

$as.campus[proxPos.x][proxPos.y].estudiante \leftarrow as.campus[posVieja.x][posVieja.y].estudiante$ O(1)

$as.campus[posVieja.x][posVieja.y].estudiante \leftarrow NULL$ 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

$as.campus[posVieja.x][posVieja.y].hayEst? \leftarrow False$ O(1)

$as.campus[posVieja.x][posVieja.y].estudiante.eliminarSiguiente()$ O(long(nombre))

end if

O(long(nombre))

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

if $\neg(encerrado?(as, as.hippies.obtener(nombre)))$ **then** O(long(nombre))

// Obtener pos siguiente y actualizar posicion de hippie

$posVieja \leftarrow as.hippies.obtener(nombre)$ O(long(nombre))

$posNueva \leftarrow proxPosHippie(as, nombre)$ O(long(nombre) + N_e)

$as.campus[posVieja.x][posVieja.y].hayHippie \leftarrow False$ O(1)

$as.campus[posNueva.x][posNueva.y].hayHippie \leftarrow True$ O(1)

$itHippie \leftarrow as.campus[posVieja.x][posVieja.y].hippie$ O(1)

$as.campus[posVieja.x][posVieja.y].hippie \leftarrow NULL$ O(1)

$as.campus[posNueva.x][posNueva.y].hippie \leftarrow itHippie$ O(1)

// Sancionar agentes que rodean a los estudiantes que encierro

sancionarAgentesEncerrandoEstVecinos(as,posNueva) O(1)

// Capturar hippies encerrados

aplicarHippiesVecinos(as,posNueva) O(long(nombre))

// Hippificar estudiantes

hippificarEstudiantesVecinos(as,posNueva) O(long(nombre))

end if

```
moverAgente(in/out as : as,  in placa : placa)
  // Obtener pos siguiente y actualizar pos de agente
  posVieja ← busquedaBinariaPorPlaca(as.agentesPorPlaca, placa).pos
                                                              O(log(N_a))
  if ¬(encerrado?(as, posVieja))
  ∧
  as.campus[posVieja.x][posVieja.y].agente.siguienteSignificado().cantSanciones < 3 then
                                                              O(1)
    proxPos ← as.proxPosAgente(posVieja)                      O(N_h)
    as.campus[posVieja.x][posVieja.y].hayAgente ← False       O(1)
    as.campus[proxPos.x][proxPos.y].hayAgente ← True          O(1)
    itAgente ← as.campus[posVieja.x][posVieja.y].agente       O(1)
    as.campus[proxPos.x][proxPos.y].agente ← itAgente         O(1)
    as.campus[posVieja.x][posVieja.y].agente ← NULL           O(1)
    busquedaBinariaPorPlaca(as.agentesPorPlaca, placa).pos ← proxPos
                                                              O(log(N_a))
    sancionarAgentesEncerrandoEstVecinos(as, proxPos)         O(1)
    aplicarHippiesVecinos(as, proxPos)                        O(long(nombre))
  end if
                                                              ----------
                                                              O(N_h + log(N_a) + long(nombre))

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

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

	$O(N_a) \vee O(log(N_a))$
masVigilante(**in** *as* : as) $\longrightarrow$ *res* : placa	
$res \leftarrow as.masVigilante.siguienteClave()$	$O(1)$
	$O(1)$

1.5 Algoritmos operaciones auxiliares

sancionarAgentesVecinos(**in/out** *as* : as, *in pos* : pos)	
$vecinos \leftarrow as.campusEstatico.vecinos(pos)$	$O(1)$
if $as.atrapadoPorAgente?(pos)$ **then**	
while $i < vecinos.tamanio()$ **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)	
$vecinos \leftarrow as.campusEstatico.vecinos(pos)$	$O(1)$
$i \leftarrow 0$	
while $i < vecinos.tamanio$ **do**	$O(1)$
if $as.campus[vecinos[i].x][vecinos[i].y].hayEst \wedge atrapadoPorAgente?(as, pos)$ **then**	
	$O(1)$
$sancionarAgentesVecinos(as, pos)$	$O(1)$
end if	
$i++$	
end while	
	$O(1)$
sancionarAgente(**in/out** *as* : as, *in/out agente* : itDiccRapido)	
$as.conKSanciones.ocurrioSancion \leftarrow True$	$O(1)$
$agente.siguiente.cantSanciones + 1$	$O(1)$
$agente.siguiente.miUbicacion.eliminarSiguiente()$	$O(1)$
// 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 $agente.siguiente.mismas.haySiguiente() \wedge agente.siguiente.mismas.siguiente.cantSanciones <$ $agente.siguiente.cantSanciones$ **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 ← *nuevaTupla*(*CrearNuevoDiccLineal*(), *agente.siguiente.cantSanciones*)
agente.siguiente.mismas ← *agente.siguiente.mismas.agregarComoSiguiente*(*nConMismasB*)

O(1)

agente.siguiente.miUbicacion ← *agente.siguiente.mismas.siguiente.agentes.agregarComoSiguiente*(*ag*

O(1)

else
agente.siguiente.mismas.siguiente.agentes.agregarComoSiguiente(*agente.siguiente.pl*)

O(1)

end if

O(1)

ATRAPADOPORAGENTE?(**in** *as* : `as`, *in pos* : `pos`) ⟶ *res* : `bool`

vecinos ← *as.campusEstatico.vecinos*(*pos*)
*alMenos*1*Agente* ← *False* O(1)
i ← 0
if ¬(*encerrado*?(*pos*, *as.campusEstatico.vecinos*(*pos*))) **then**
return false
end if
// Veo si hay algun agente alrededor
while *i* < *vecinos.tamanio*() **do** O(1)
if *as.campus*[*vecinos*[*i*].*x*][*vecinos*[*i*].*y*].*hayAgente*? **then**
return true
end if
i + + O(1)
end while

O(1)

HIPPIFICARESTUDIANTESVECINOS(**in/out** *as* : `as`, *in pos* : `pos`)

vecinos ← *as.campusEstatico.vecinos*(*pos*) O(1)
i ← 0 O(1)
while *i* < *vecinos.tamanio*() **do** O(long(nombre))
if *estAHippie*?(*as*, *vecinos*[*i*]) **then**
hippificar(*as*, *vecinos*[*i*]) O(long(nombre))
end if
i + + O(1)
end while

O(long(nombre))

HIPPIFICAR(**in/out** *as* : `as`, *in pos* : `pos`)

// PRE: La posicion esta en el tablero y hay estudiante en la posicion
as.campus[*pos.x*][*pos.y*].*hayHippie* ← *True* O(1)
as.campus[*pos.x*][*pos.y*].*hippie.agregarComoSiguiente*(*nombre*, *pos*)

O(long(nombreEstudiante))

as.campus[*pos.x*][*pos.y*].*hayEst* ← *False* O(1)
as.campus[*pos.x*][*pos.y*].*estudiante.eliminarSiguiente*() O(long(nombreEstudiante))

O(long(nombre))

ESTAHIPPIE?(**in** *as* : `as`, *in pos* : `pos`) ⟶ *res* : `bool`

if ¬(*encerrado*?(*pos*, *vecinos*)) **then**
return false O(1)

```
  end if                                                          
  i ← 0                                                           O(1)
  cantHippies ← 0                                                 O(1)
  vecinos ← as.campusEstatico.vecinos(pos)                        O(1)
  while i < vecinos.tamanio() do                                  
      if campus[vecinos[i].x][vecinos[i].y].hayHippie then        
          cantHippies + +                                         O(1)
      end if                                                      
      i + +                                                       
  end while                                                       
  return cantHippies ≥ 2                                          O(1)
```

O(1)

```
hippieAEst?(in as : as, in pos : pos) ⟶ res : bool
  i ← 0                                                           O(1)
  vecinos ← as.campusEstatico.vecinos(pos)                        O(1)
  while i < vecinos.tamanio() do                                  O(1)
      if ¬(as.campus[vecinos[i].x][vecinos[i].y].hayEst?) then    
          return False                                            O(1)
      end if                                                      
  end while                                                       
  return True                                                     
```

O(1)

```
encerrado?(in as : as, in pos : pos)
  vecinos ← vecinos(as.campusEstatico, pos)                       O(1)
  i ← vecinos.tamanio()                                           O(1)
  while i < vecinos.tamanio() do                                  O(1)
      if ¬(as.campus[vecinos[i].x][vecinos[i].y].hayAgente? ∨     
  as.campus[vecinos[i].x][vecinos[i].y].hayEst? ∨                 
  as.campus[vecinos[i].x][vecinos[i].y].hayHippie? ∨              
  as.campus[vecinos[i].x][vecinos[i].y].hayObst?) then            O(1)
          return false                                            O(1)
      end if                                                      
      i + +                                                       O(1)
  end while                                                       
  returntrue                                                      
```

O(1)

```
aplicarHippiesVecinos(in/out as : as, in pos : pos)
  vecinos ← as.campusEstatico.vecinos(pos)                        O(1)
  i ← 0                                                           O(1)
  while i < vecinos.tamanio() do                                  O(long(nombre))
      aplicarHippie(as, pos)                                      O(long(nombre))
  end while                                                       
```

O(long(nombre))

```
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 ← False               O(1)
```

```
        as.campus[pos.x][pos.y].hayEst ← True                                   O(1)
        as.campus[pos.x][pos.y].estudiante ← CrearIt(as.hippies)
                                                                                O(1)
        as.campus[pos.x][pos.y].estudiante.agregarComoSiguiente(as.campus[pos.x][pos.y].estudiante.nomb
                                                                                O(long(nombre))
        as.campus[pos.x][pos.y].hippie.eliminarSiguiente()                      O(long(nombre))
    else
        if as.campus[pos.x][pos.y].hayHippie? ∧ atrapadoPorAgente(pos) then
            vecinos ← as.campusSeguro.vecinos(pos)                              O(1)
            i ← 0                                                               O(1)
            while i < vecinos.tamanio() do                                      O(1)
                posAct ← vecinos[pos.x][pos.y]                                  O(1)
                info ← as.campus[vecinos[i].x][vecinos[i].y]                    O(1)
                if posAct.hayAgente then
                    info.agente.siguiente.cantCapturas + +                      O(1)
                    // Actualizar mas vigilante
                    if as.masVigilante.siguienteSignificado().cantCapturas < info.agente.siguienteSignific
then
                        as.masVigilante ← info.agente                           O(1)
                    else
                        if as.masVigilante.siguienteSignificado().cantCapturas = info.agente.siguienteSign
∧as.masVigilante.siguienteClave() < info.agente.siguienteClave() then
                                                                                O(1)
                            as.masVigilante ← info.agente                       O(1)
                        end if
                    end if
                end if
                i + +
            end while
            as.campus[pos.x][pos.y].hayHippie? = False                          O(1)
            as.campus[pos.x][pos.y].hippie.eliminarSiguiente()
                                                                                O(long(nombre))
        end if
    end if
end if
                                                                                ---------------
                                                                                O(long(nombre))
```

proxPosHippie(**in/out** *as* : as, *in nombre* : string) ⟶ *res* : pos

```
// PRE: El nombre es un hippie y el hippie no esta encerrado
posHippie ← as.hippies.obtener(nombre)                                          O(long(nombre))
if as.estudiantes.tamanio() > 0 then
    // Retorna de las posiciones mas cercanas, la que esta mas cerca del (0,0)
    proxPos ← aPosMasCercana(as.campusEstatico, posHippie, as.estudiantes.significados)
                                                                                O(N_e)
else
    // Retorna el ingreso mas cercan, en caso de empate, el de abajo
    proxPos ← aIngresoMasCercano(as.campusEstatico, posHippie)
                                                                                O(1)
end if
res ← proxPos                                                                   O(1)
                                                                                ---------------
                                                                                O(N_e)
```

proxPosAgente(**in/out** *as* : as, *in posAgente* : pos) ⟶ *res* : pos

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

O(N_h)

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

3 Placa es Nat

4 Nombre es String

5 Diccionario Rapido

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

5.1 Interfaz

parámetros formales

géneros Nat, β

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

géneros `diccR(Nat, conj(`β`))`

usa Bool, Nat, Conjunto(β)

Operaciones

CREAR(**in** *dicc* : `Dicc(c:nat s:`β`)`) $\longrightarrow$ *res* : `diccSR(Nat,`β`)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} Nuevo()\}$
Descripción: Crea un diccionario rapido.
Complejidad: O(n)
Aliasing: No hay aliasing

OBTENER(**in/out** v : `diccR(Nat;conj(`α`))`, **in** p : `nat`) $\longrightarrow$ *res* : `conj(`α`)`
Pre $\equiv$ {Definido?(p,v)}
Post $\equiv$ $\{Obtener(p, v)\}$
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** `tupla`⟨accesoRapido : `vector(acceso:Dicc(nat,itDicc(nat,`β`)))`,
contenedor : `Dicc(c:nat,s:`β`)`⟩

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** *diccACopiar* : `dicc(c:Nat,` $s : \beta$`)` $\longrightarrow$ *res* : `diccR()`

$it \leftarrow CrearIt(diccACopiar)$	O(1)
// Inicializo el vector	
while $it.haySiguiente$ **do**	
$res.accesoRapido.agregarAtras(NULL)$	O(1)
$it.avanzar()$	O(1)
end while	
$it \leftarrow CrearIt(diccACopiar)$	O(1)
// El iterador vuelve a apuntar al primer elemento, defino todos los elementos del vector en el hash	
while $it.haySiguiente()$ **do**	O(diccACopiar.tamanio())
$res.accesoRapido[fHash(it.siguienteClave())].$	
$Definir(c, res.contenedor.Definir(it.siguienteClave(), it.siguienteSignificado()))$	$\theta(1)$
$it.avanzar()$	O(1)
end while	
	θ(n)

iDameIterador(**in** a : `diccR`) $\longrightarrow$ *res* : `itDiccR`

$res \leftarrow CrearIt(a.contenedor)$	O(1)

iObtener(**in/out** a : diccR, *in* c : Nat) $\longrightarrow$ *res* : β	
$res \leftarrow a.accesoRapido[a.fHash(c)].obtener(c).siguiente()$	$\theta(1)$
	$\theta(1)$

5.4 Operaciones privadas

fHash(**in** a : diccR, *in* c : nat) $\longrightarrow$ *res* : Nat	
$res \leftarrow (c \% a.tamanio())$	O(1)

5.5 Representación del iterador

se representa con itDiccRIt**itDicc(¡clave:nat, significado:**β**¿)**

6 Diccionario por nombres

6.1 Interfaz

se explica con Dicc

usa

géneros dpn

Operaciones

Vacio() $\longrightarrow$ *res* : dpn
Pre $\equiv$ {true}
Post $\equiv$ $\{dpn =_{\text{obs}} vacia()\}$
Descripción: Crea un nuevo diccionario
Complejidad:
Aliasing: $O(1)$

Definido?(**in/out** d : dpn, *in* c : String) $\longrightarrow$ *res* : bool
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} def?(d_0, e)\}$
Descripción: Indica si la clave tiene un significado
Complejidad:
Aliasing: $O(long(c))$

Definir(**in/out** d : dpn, *in* c : String, *in* e : α)
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: $O(long(c))$

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

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

ALISTA(**in/out** d : `dpn`, *in* c : `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

6.2 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

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

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

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

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

Auxiliares

6.3 Algoritmos

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

$res \leftarrow CrearTupla(InicializarVector(), NULL)$	
	O(1)

IDEFINIR(**in/out** d : `dpn`, *in clave* : `String`, *in* $e : \alpha$) $\longrightarrow$ *res* : `dpn`

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

IELIMINAR(**in/out** d : `dpn`, *in clave* : `String`) $\longrightarrow$ *res* : `dpn`

$nodo : puntero(Nodo) \leftarrow NULL$	
$i : nat \leftarrow 0$	
// Por ref	
$caracteres \leftarrow d.buckets$	O(1)
while $i \leq Longitud(clave)$ **do**	O(long(clave))
$nodo \leftarrow caracteres[ord(clave[i])]$	O(1)
// Por ref	
$caracteres \leftarrow nodo.hijos$	O(1)
$i++$	O(1)
end while	
$nodo.hayS \leftarrow False$	O(1)
$nodo.enLista.eliminarSiguiente()$	O(1)
if $nodo.hijos = NULL$ **then**	
// Elimina un puntero	
$borrar(nodo)$	O(1)

end if

	O(long(clave))

iSignificado(**in/out** d : dpn, *in clave* : String) $\longrightarrow$ *res* : α

$nodo : puntero(Nodo) \leftarrow NULL$	O(1)
$buckets : puntero(Nodo) \leftarrow d.buckets$	O(1)
$i \leftarrow 0$	O(1)
while $i \leq Longitud(clave)$ **do**	
$nodo \leftarrow buckets[ord(clave[i])]$	O(1)
$i++$	
end while	
// Por ref	
$res \leftarrow nodo.significado$	O(1)
	O(long(clave))

iClaves(**in/out** d : dpn) $\longrightarrow$ *res* : Lista(String)

$res \leftarrow d.claves$	O(1)
	O(1)

iDefinido?(**in/out** d : dpn, *in clave* : String) $\longrightarrow$ *res* : bool

$nodo : puntero(Nodo) \leftarrow NULL$	O(1)
$buckets : puntero(Nodo) \leftarrow d.buckets$	O(1)
$i \leftarrow 0$	O(1)
while $i \leq Longitud(clave)$ **do**	O(long(clave))
$nodo \leftarrow buckets[ord(clave[i])]$	O(1)
if $nodo = NULL$ **then**	O(1)
$return\ False$	O(1)
end if	
$i++$	O(1)
end while	
$res \leftarrow nodo.hayS$	O(1)
	O(long(clave))

6.4 Operaciones del iterador

CrearIterador(**in** d : dpn) $\longrightarrow$ *res* : itDPN
Pre $\equiv \{true\}$
Post $\equiv \{tuplasClaveSignificado(d) =_{\text{obs}} siguientes(res) \wedge_{\text{L}} aliasing(tuplasClaveSignificado(d), siguientes($
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 $\equiv \{res =_{\text{obs}} haySiguiente(it)\}$
Descripción: Indica si hay siguiente
Complejidad: $O(1)$

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

Descripción: Indica si hay anterior
Complejidad: $O(1)$

SIGUIENTE(**in** *it* : `itDPN`) $\longrightarrow$ *res* : `<clave:String,significado:`α`>`
Pre $\equiv \{HaySiguiente(it)\}$
Post $\equiv \{res =_{\text{obs}} siguiente(it)\}$
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 $\equiv \{res =_{\text{obs}} anterior(it)\}$
Descripción: Retorna el anterior
Complejidad: $O(1)$
Aliasing: Hay aliasing

SIGUIENTECLAVE(**in** *it* : `itDPN`) $\longrightarrow$ *res* : `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* : `itDPN`) $\longrightarrow$ *res* : `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* : `itDPN`) $\longrightarrow$ *res* : α
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* : `itDPN`) $\longrightarrow$ *res* : α
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* : `itDPN`)
Pre $\equiv \{HaySiguiente(it) \wedge it =_{\text{obs}} it_0\}$
Post $\equiv \{anteriores(it_0) \bullet primero(siguientes(it_0)) =_{\text{obs}} anteriores(it) \wedge fin(siguientes(it_0)) =_{\text{obs}} siguientes($
Descripción: Modifica el iterador, haciendolo avanzar una posicion
Complejidad: $O(1)$

RETROCEDER(**in/out** *it* : `itDPN`)
Pre $\equiv \{Hayanterior(it) \wedge it =_{\text{obs}} it_0\}$
Post $\equiv \{comienzo(anteriores(it_0)) =_{\text{obs}} anteriores(it) \wedge ultimo(anteriores(it_0) \bullet siguientes(it_0) =_{\text{obs}} siguien$
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 : `dpn`) $\longrightarrow$ res : `itDPN`
$res \leftarrow NuevoItLista(d.ALista())$ — O(1)
O(1)

HaySiguiente(**in** it : `itDPN`) $\longrightarrow$ res : `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 : α
$res \leftarrow it.Siguiente().significado$ — O(1)
O(1)

AnteriorSignificado(**in** it : `itDPN`) $\longrightarrow$ res : α
$res \leftarrow it.Anterior().significado$ — 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 $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} crearCampus(ancho, alto)\}$
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 $\equiv$ $\{c =_{\text{obs}} agregarObstaculo(p, c_0)\}$
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 $\equiv$ $\{res \equiv alto(c)\}$
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* : `campus`, *p* : `pos`) $\longrightarrow$ *res* : `bool`
Pre $\equiv$ {true}
Post $\equiv$ $\{res \iff (\pi_1(p) < ancho(c) \land \pi_2(p) < alto(c))\}$
Descripción: Comprueba que una posición exista dentro del campus.
Complejidad: O(1)

ESINGRESO(**in** *c* : `campus`, *p* : `pos`) $\longrightarrow$ *res* : `bool`
Pre $\equiv$ {PosValida(c,p)}
Post $\equiv$ $\{res \iff (\pi_2(p) = alto(c) - 1 \lor \pi_2(p) = 0)\}$
Descripción: Comprueba si una posición es un ingreso al campus.
Complejidad: O(1)

INGRESOSUP(**in** *c* : `campus`, *p* : `pos`) $\longrightarrow$ *res* : `bool`
Pre $\equiv$ {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 $\equiv$ {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 : `campus`, $p1$: `pos`, $p2$: `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 : `campus`, p : `pos`) $\longrightarrow$ res : `conj(pos)`
Pre $\equiv \{PosValida(c, p)\}$
Post $\equiv \{res \equiv vecinos(p, c)\}$
Descripción: devuekve 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.

7.2 Representación

se representa con `estr`

donde `estr` **es** `tupla`⟨ancho : `nat`,
alto : `nat`,
grilla : `arreglo(arreglo(tupla`⟨Ocupado : `bool`, `, tupla`⟨pl : `nat`, `))`⟩
EsObst : `bool`, nombre : `string`⟩
EsAgente : `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 conjuno 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 : $\widehat{\texttt{sistema}} \longrightarrow$ *boolean*

$(\forall s : \widehat{\texttt{sistema}})$
Rep$(s) \equiv$
1. $\forall s$: String $def?(s, s.CompusPorPref), (\exists c : compu), esta?(c, s.Compus) \ \wedge \ \pi_1(c) = s \ \wedge$ longitud$(s.Compus) = \#\text{CLAVES}(s.CompusPorPref)$
2. $\forall c$: compu esta?$(c, s.Compus), *\pi_2(c) =$ obtener$(\pi_1(c), s.CompusPorPref)$
3. $\forall c$: compu esta?$(c, s.Compus), *\pi_3(c) =$ obtener$(\pi_3(c), s.CompusPorPref)$
4, 5, 6.
$(\forall c : \text{nat}) \ 0 \leq c < Longitud(s.compus) \ \Rightarrow_{\text{L}}$
$Longitud(s.compus[c].pN) = Longitud(s.compus[c].pN') \wedge$
$(\forall p : \text{paquetePos})\text{esta?}(p, s.compus[c].pN') \ \Rightarrow_{\text{L}}$
esta$(\pi_1(p), s.compus[c].pN) \wedge 0 \leq indiceOrigen(p) < Longitud(s.compus)$
$\wedge \ 0 \leq indiceDestino(p) < Longitud(s.compus)$
$\wedge \ \ 0 \leq posActual(p) < Longitud(s.compus)$
$\wedge \ \neg(indiceDestino(p) = indiceOrigen(p))$
7. $(\forall c : \text{nat}) \ 0 \leq c < Longitud(s.compus) \ \Rightarrow_{\text{L}}$
$(\forall p : \text{paquetePos})$ pertenece$(s.compus[c].pN', p) \ \Rightarrow_{\text{L}}$
$\beta(\text{esta}(s.compus[c], caminoMinimo(s.red, s.compus[indiceOrigen(p)], s.compus[posActual(p)])))$
8. $\forall s, t : \text{String def?}(s, s.CompusPorPref) \ \wedge \ \text{def?}(t, s.CompusPorPref) \ \wedge \ s \neq t \ \Rightarrow_{\text{L}}$
obtener$(s, s.CompusPorPref) \bigcap$ obtener$(t, s.CompusPorPref) = \emptyset$
9. $\forall s : \text{String def?}(s, s.CompusPorPref) \ \Rightarrow_{\text{L}} \ \pi_1(\text{obtener}(s, s.CompusPorPref)) =$
$\pi_2(\text{obtener}(s, s.CompusPorPref))$
10. $Longitud(s.compus) = Longitud(CaminosMinimos(s)) \wedge$
$(\forall i : \text{nat}) \ 0 \leq i < Longitud(s.compus) \ \Rightarrow_{\text{L}}$
$Longitud(s.CaminosMinimos[i]) = Longitud(s.compus)$
11. $(\forall f, d : \text{nat}) \ \neg(f = d) \wedge 0 \leq f, d < Longitud(s.compus) \ \Rightarrow_{\text{L}}$
$CaminosMinimos[f][d] =$
$caminoMinimo(s.red, ipACompu(s.red, \pi_1(s.compus[f])), ipACompu(s.red, \pi_1(s.compus[d])))$
12, 13, 14. $(\forall i, j : \text{nat}), \ 0 \leq i, j <$ longitud$(s.CaminosMinimos) \ \Rightarrow_{\text{L}}$longitud$(s.CaminosMinimos) =$
longitud$(s.CaminosMinimos[i]) \wedge$ longitud$(s.CaminosMinimos[i][j]) <$ longitud$(s.CaminosMinimos) \wedge$

$(\forall e : \text{nat}), \text{esta?}(e, s.CaminosMinimos[i][j]) \Rightarrow \text{pertenece}(e, s.CompusPorPref)$
$15.\forall c : \text{compu esta?}(c, s.Compus) \quad \Rightarrow_{\text{L}} \quad \pi_3(c) \leq \pi_3(s.Compus[s.LaQMasEnvio])$

Función de abstracción

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

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

7.3 Algoritmos

icrearSistema(**in** r : `red`) $\longrightarrow$ res : `dcnet`	
$res.red \leftarrow r$	
$n \leftarrow Longitud(\text{compus}(red))$	O(1)
$i \leftarrow 0$	
$j \leftarrow 0$	O(1)
$res.Compus \leftarrow$ CrearArreglo(n)	O(n)
$res.CaminosMinimos \leftarrow$ CrearArreglo(n)	O(n)
var p : `arreglo_dimensionable de puntero(conjLog(paquete))`	
while i<n **do**	$O(L * n^5)$
	O(n)
$res.CaminosMinimos[i] \leftarrow$ CrearArreglo(n)	O(n)
s: $< nat, conjLog(paquete, <_{id}), conjLog(paquete, <_p),$ $conjLog(paquetePos, <_{id}), conjLog(paquetePos, <_p) >$	
$\pi_1(s) \leftarrow compu(r, i)$	
$\pi_2(s) \leftarrow$ nuevo()	
$\pi_3(s) \leftarrow$ nuevo()	
$\pi_4(s) \leftarrow$ nuevo()	
$\pi_5(s) \leftarrow$ nuevo()	
definir(res.CompusPorPref,compu(r,i),s)	O(L)
$p[i] \leftarrow \pi_3(s)$	
$p'[i] \leftarrow \pi_5(s)$	
$res.Compus[i] \leftarrow < compu(r, i), p[i], p'[i], 0 >$	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)
	$O(L \times n^5)$

icrearPaquete(**in/out** s : `dcnet`, **in/out** p : `paquete`)	
$t_1 : < nat, conjLog(paquete, <_{id}), conjLog(paquete, <_p),$ $conjLog(paquetePos, <_{id}), conjLog(paquetePos, <_p) >$	
$t_1 \leftarrow$ Obtener($origen(p),\ s.CompusPorPref$)	O(L)
$t_2 : < nat, conjLog(paquete, <_{id}), conjLog(paquete, <_p),$ $conjLog(paquetePos, <_{id}), conjLog(paquetePos, <_p) >$	
$t_2 \leftarrow$ Obtener($destino(p),\ s.CompusPorPref$)	O(L)
p': $paquetePos$	
indiceOrigen(p') $\leftarrow \pi_1(t_1)$	O(1)
indiceDestino(p') $\leftarrow \pi_1(t_2)$	O(1)
posActual(p') $\leftarrow 0$	
insertar($\pi_2(t), p$)	$O(log(k))$
insertar($\pi_3(t), p$)	$O(log(k))$
insertar($\pi_4(t)$, p')	$O(log(k))$
insertar($\pi_5(t)$, p')	$O(log(k))$
	$O(L + log(k))$

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).