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)`,
agentes : `DiccSuperRapido(pl:nat,tupla`⟨pos : `pos`,),
cantSanc : `nat`,
cantCapturas : `nat`,
mismas : `itLista(conMismasBucket)`,
miUbicacion : `itLista(agente)`⟩
masVigilante : `placa:nat`,
porSanciones : `Lista(conMismasBucket)`,
conKSanciones : `arreglo(tupla`⟨ocurrioSancion : `bool`,)⟩
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 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*

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

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

Función de abstracción

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

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

1.4 Algoritmos

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

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

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

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

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

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

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

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

POSESTUDIANTESYHIPPIES(**in** *as* : `as`, *in nombre* : `string`) $\longrightarrow$ *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`

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

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

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

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

$res \leftarrow as.agentes.dameS(placa).cantHippiesAtrapados$	$\theta(1)$
	$\theta(1)$

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)
// Hippificar al estudiante	
if $as.estAHippie?(as, pos, as.campusEstatico.vecinos(as, pos))$ **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))
if $as.campus[pos.x][pos.y].hayHippie?$ **then**	
$capturarHippie(pos)$	O(long(nombre))
end if	
	O(long(nombre))

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

$as.CampusEstatico \leftarrow ce$	O(1)
$ItAgentes \leftarrow CrearItAgentes(Agentes)$	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	
$as.Agentes \leftarrow CrearDiccSuperRapido()$	O(1)
$MayorPlaca \leftarrow 0$	O(1)
while $ItAgente.HaySiguiente$ **do**	O(Cantidad Agentes)
$as.Campus[ItAgentes.Siguiente.pos.X][ItAgentes.Siguiente.pos.Y].HayAgente \leftarrow True$	O(1)
$as.Campus[ItAgentes.Siguiente.pos.X][ItAgentes.Siguiente.pos.Y].Pl \leftarrow Definir(as.Agentes, ItAgentes.Siguiente.Pl, Tupla(Pos \leftarrow ItAgentes.Siguiente.Pos, CantSanciones \leftarrow 0, CantCapturas \leftarrow 0)$	O(1)
if $MayorPlaca < ItAgentes.Siguientes.Pl$ **then**	
$MayorPlaca \leftarrow ItAgenda.Siguiente.Pl$	O(1)
$MayorAgente \leftarrow ItAgenda.Siguiente$	O(1)
end if	
$ItAgentes.Avanzar$	O(1)
end while	
$as.Estudiante \leftarrow CrearDiccPorNombre()$	O(1)
$as.Hippie \leftarrow CrearDiccPorNombre()$	O(1)
$as.MasVigilante \leftarrow MayorAgente$	O(1)
$as.PorSanciones \leftarrow CrearLista(Tupla < Agentes \leftarrow \varnothing, \#Sanciones \leftarrow 0 >$	O(1)
$ItAgentesRapido \leftarrow CrearItAgentesRapido()$	O(1)
while $ItAgentesRapido.HaySiguiente$ **do**	O(Cantidad Agentes)
$ItAgentesRapido.HaySiguiente.MiHubicacion \leftarrow as.PorSancion.ObtenerUltimo.Agentes.Agregar(ItAgentesRapido, SiguienteClave)$	O(1)
$ItAgentesRapido.Avanzar$	O(1)
end while	
	O((Ancho*Alto) + N_a)

1.5 Algoritmos operaciones auxiliares

sancionarAgentesVecinos(**in/out** *as* : as, *in pos* : pos)

vecinos ← *as.campusEstatico.vecinos*(*pos*) O(1)
// La cantidad de vecinos es como maximo 4
if *as.atrapadoPorAgente?*(*pos*, *vecinos*) **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)

sancionarAgente(**in/out** *as* : as, *in/out agente* : itDiccRapido)

// Tengo como maximo 4 sanciones
agente.siguiente.cantSanciones ← *agente.siguiente.cantSanciones*+*cantEstAtrapados*(*agente.siguiente.po*
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*()∧*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 ¬(*agente.siguiente.mismas.haySiguiente*)∨(*agente.siguiente.mismas.haySiguiente*∧*agente.siguiente.ca*
agente.siguiente.mismas.cantSanciones) **then** O(1)
  nConMismasB ← *nuevaTupla*(*agentes* : *Nuevo*(), *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

atrapadoPorAgente?(**in** *as* : as, *in pos* : pos, *in vecinos* : arreglo(pos)) ⟶ *res* : bool

alMenos1Agente ← *False* O(1)
$i \leftarrow 0$
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)
```

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                         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))
```

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

```
  if ¬(encerrado?(pos, vecinos)) then
    return false                                                   O(1)
  end if
  i ← 0                                                            O(1)
  cantHippies ← 0
  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)
```

ENCERRADO?(**in** *as* : `as`, *in pos* : `pos`, *in vecinos* : `conj(pos)`)

```
  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
```

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

```
  vecinos ← as.campusEstatico.vecinos(pos)                         O(1)
  while vecinos.haySiguiente do
    if as.campus[vecinos.siguiente.x][vecinos.siguiente.y].hayHippie then
      if as.hippieAEst(vecinos.siguiente) then                     O(1)
        as.campus[vecinos.siguiente.x][vecinos.siguiente.y].hayHippie ← False
                                                                   O(1)
        as.campus[vecinos.siguiente.x][vecinos.siguiente.y].hayEst ← True
                                                                   O(1)
        as.campus[vecinos.siguiente.x][vecinos.siguiente.y].hippie ← CrearIt(as.hippies)
                                                                   O(1)
```

$as.campus[vecinos.siguiente.x][vecinos.siguiente.y].hippie.agregarComoSiguiente(as.campus[ve$
O(long(nombre))
$as.campus[vecinos.siguiente.x][vecinos.siguiente.y].estudiante.eliminarSiguiente()$
O(long(nombre))

end if
end if
end while

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 $\equiv$ $\{\#Claves(res) =_{\text{obs}} n\}$
Descripción: Crea un diccionario rapido.
Complejidad: O(n)
Aliasing: Completar Aliasing

ASIGNAR(**in/out** v : `diccR(Nat;conj(`α`))`, **in** p : `nat`, **in** s : α)
Pre $\equiv$ $\{v =_{\text{obs}} v_0 \wedge$ Definido?(p,v)$\}$
Post $\equiv$ $\{Definir(p, Ag(Obtener(p, v_0), s), v)\}$
Descripción: Agrega el valor de s, al significado actual, para la clave dada
Complejidad: O(1)
Aliasing: Completar Aliasing

DAMES(**in/out** v : `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: 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** `claves : arreglo(contenido)`

donde `contenido` **es** `conjl(`α

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

Invariante de representación

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

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

$(\forall a : \widehat{\texttt{acceso}})$
Rep$(a) \equiv$

1. 1. $\forall p$: Nat Definido?(a,p) = obtener(π_1(c),s.CompusPorPref)

Función de abstracciÃ³n

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

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

2.3 Algoritmos

ICREAR(**in** r : `Nat`) $\longrightarrow$ res : `diccR()`

$i \leftarrow 0$	O(1)
$p \leftarrow CrearArreglo(n)$	O(n)
while $i < n$ **do**	O(n)
$p[i] \leftarrow vacio()$	O(1)
$i++$	O(1)
end while	
$res \leftarrow p$	O(1)
	O(n)

IASIGNAR(**in/out** a : `acceso`, $in\ p$: `Nat`, $in\ s : \alpha$)

a[FhashPlaca(p,a)] = AgregarRapido(a[FhashPlaca(p,a)],s)	O(1)
	O(1)

IDAMES(**in/out** a : `acceso`, *in* p : `Nat`) $\longrightarrow$ *res* : `contenido`

res = a[FhashPlaca(p,a)]	O(1)
	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* : `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* $e : \alpha$)
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: $O(long(c))$

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 : α
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 : α
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

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

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

3.3 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))

3.4 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: $O(1)$

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 : `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)

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

4.2 Representación

se representa con `estr`

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

Invariante de representación

1. 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.
$(\forall c : \text{nat})\ 0 \leq c < Longitud(s.compus) \ \Rightarrow_{\text{L}}$
$Longitud(s.compus[c].pN) = Longitud(s.compus[c].pN') \wedge$
$(\forall p : \text{paquetePos})\text{esta?}(p, s.compus[c].pN') \ \Rightarrow_{\text{L}}$
$\text{esta}(\pi_1(p), s.compus[c].pN) \wedge 0 \leq indiceOrigen(p) < Longitud(s.compus)$
$\wedge\ 0 \leq indiceDestino(p) < Longitud(s.compus)$
$\wedge \ \ 0 \leq posActual(p) < Longitud(s.compus)$
$\wedge \ \neg(indiceDestino(p) = indiceOrigen(p))$
7. $(\forall c : \text{nat})\ 0 \leq c < Longitud(s.compus) \ \Rightarrow_{\text{L}}$
$(\forall p : \text{paquetePos})\ \text{pertenece}(s.compus[c].pN', p) \ \Rightarrow_{\text{L}}$
$\beta(\text{esta}(s.compus[c], caminoMinimo(s.red, s.compus[indiceOrigen(p)], s.compus[posActual(p)])))$
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. $(\forall f, d : \text{nat})\ \neg(f = d) \wedge\ 0 \leq f, d < Longitud(s.compus) \ \Rightarrow_{\text{L}}$
$CaminosMinimos[f][d] =$
$caminoMinimo(s.red, ipACompu(s.red, \pi_1(s.compus[f])), ipACompu(s.red, \pi_1(s.compus[d])))$

$12, 13, 14.$ $(\forall i, j : \text{nat}),\ 0 \leq i, j < \text{longitud}(s.CaminosMinimos) \ \Rightarrow_{\text{L}} \text{longitud}(s.CaminosMinimos) = \text{longitud}(s.CaminosMinimos[i]) \wedge \text{longitud}(s.CaminosMinimos[i][j]) < \text{longitud}(s.CaminosMinimos) \wedge (\forall e : \text{nat}), \text{esta?}(e, s.CaminosMinimos[i][j]) \Rightarrow \text{pertenece}(e, s.CompusPorPref)$
$15. \forall c : \text{compu esta?}(c, s.Compus) \ \Rightarrow_{\text{L}} \ \pi_3(c) \leq \pi_3(s.Compus[s.LaQMasEnvio])$

Función de abstracción

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

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

4.3 Algoritmos

iArmarCampus(**in** *ancho* : nat, *in alto* : nat) $\longrightarrow$ *res* : campus	
$res.ancho \leftarrow ancho$	O(1)
$res.alto \leftarrow alto$	O(1)
$res.grilla \leftarrow CrearArreglo(ancho)$	O(an)
$i \leftarrow 0$	O(1)
	O(1)
while $i < ancho$ **do**	O($al * an$)
	O(an)
$res.grilla[i] \leftarrow$ CrearArreglo($alto$)	O(al)
$j \leftarrow 0$	O(1)
while $j < alto$ **do**	O (al)
$res.grilla[i][j] \leftarrow < False, False, False, < 0, "" >>$	O(1)
end while	
end while	
	O($an * al$)
iAgregarObs(**in/out** c : campus, **in** p : pos)	
$\pi_1(c.grilla[X.p][Y.p]) \leftarrow True$	O(1)
$\pi_2(c.grilla[X.p][Y.p]) \leftarrow True$	O(1)
	O(1)
iAlto(**in** c : campus) $\longrightarrow$ *res* : nat	
$res \leftarrow c.alto$	O(1)
	O(1)
iAncho(**in** c : campus) $\longrightarrow$ *res* : nat	
$res \leftarrow c.ancho$	O(1)
	O(1)
iOcupada(**in** c : campus, *in* p : pos) $\longrightarrow$ *res* : bool	
$res \leftarrow c.grilla[X.p][Y.p].Ocupado$	O(1)
	O(1)
iPosValida(**in** c : campus, *in* p : pos) $\longrightarrow$ *res* : bool	
$res \leftarrow X.p < c.ancho \wedge Y.p < c.alto$	O(1)
	O(1)
iEsIngreso(**in** c : campus, *in* p : pos) $\longrightarrow$ *res* : bool	
$res \leftarrow Y.p == 0 \vee Y.p == c.alto - 1$	O(1)
	O(1)
iIngresoSup(**in** c : campus, *in* p : pos) $\longrightarrow$ *res* : bool	
$res \leftarrow Y.p == 0$	O(1)
	O(1)
iIngresoInf(**in** c : campus, *in* p : pos) $\longrightarrow$ *res* : bool	
$res \leftarrow Y.p == c.alto - 1$	O(1)
	O(1)
iDistancia(**in** c : campus, *in p1* : pos,	
()	
	O(1)

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).