DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2: Diseño

Primer cuatrimestre - 2015 | Algoritmos y Estructuras de Datos II

Grupo 2

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

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria – Pabellón I (Planta Baja)

Intendente Güiraldes 2160 – C1428EGA

Ciudad Autónoma de Buenos Aires – Rep. Argentina

Tel/Fax: (++54 +11) 4576-3300

http://www.exactas.uba.ar

Índice

1 CampusSeguro

1.1 Interfaz

se explica con CampusSeguro

usa

géneros `CampusSeguro`

Operaciones

campus(**in** cs : `campusSeguro`) $\longrightarrow$ 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.

1.2 Representación

se representa con `sistema`

donde `sistema` **es** `tupla`⟨Compus : `arreglo(arreglo(tupla`⟨hayHippie : `bool`, hayEst : `bool`, hayAgente : `bool`⟩ `, tupla`⟨pl : `nat`, nombre : `string`⟩`))`,
estudiantes : `DiccTrie(nombre:string, pos:pos)`,
hippies : `DiccTrie(nombre:string, pos:pos)`,
agentes : `DiccHash(pl:nat,tupla`⟨pos : `pos`, cantSanc : `nat`, cantCapturas : `nat`⟩`)`,
masVigilante : `placa:nat`,
conKSanciones : `tupla`⟨ocurrioSancion : `bool`, porKSanc : `arreglo(placa)`⟩⟩

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.
$(\forall c : \text{nat})\ 0 \leq c < Longitud(s.compus) \Rightarrow_L$
$Longitud(s.compus[c].pN) = Longitud(s.compus[c].pN') \wedge$
$(\forall p : \text{paquetePos})\text{esta?}(p, s.compus[c].pN') \Rightarrow_L$
$\text{esta}(\pi_1(p), s.compus[c].pN) \wedge 0 \leq indiceOrigen(p) < Longitud(s.compus)$
$\wedge\ 0 \leq indiceDestino(p) < Longitud(s.compus)$
$\wedge\ 0 \leq posActual(p) < Longitud(s.compus)$
$\wedge\ \neg(indiceDestino(p) = indiceOrigen(p))$
7. $(\forall c : \text{nat})\ 0 \leq c < Longitud(s.compus) \Rightarrow_L$
$(\forall p : \text{paquetePos})\ \text{pertenece}(s.compus[c].pN', p) \Rightarrow_L$
$\beta(\text{esta}(s.compus[c], caminoMinimo(s.red, s.compus[indiceOrigen(p)], s.compus[posActual(p)])))$
8. $\forall s, t$: String def?$(s, s.CompusPorPref) \wedge$ def?$(t, s.CompusPorPref) \wedge s \neq t \Rightarrow_L$
obtener$(s, s.CompusPorPref) \bigcap$ obtener$(t, s.CompusPorPref) = \emptyset$
9. $\forall s$: String def?$(s, s.CompusPorPref) \Rightarrow_L \pi_1(\text{obtener}(s, s.CompusPorPref)) =$
$\pi_2(\text{obtener}(s, s.CompusPorPref))$
10. $Longitud(s.compus) = Longitud(CaminosMinimos(s)) \wedge$
$(\forall i : \text{nat})\ 0 \leq i < Longitud(s.compus) \Rightarrow_L$
$Longitud(s.CaminosMinimos[i]) = Longitud(s.compus)$
11. $(\forall f, d : \text{nat})\ \neg(f = d) \wedge\ 0 \leq f, d < Longitud(s.compus) \Rightarrow_L$
$CaminosMinimos[f][d] =$
$caminoMinimo(s.red, ipACompu(s.red, \pi_1(s.compus[f])), ipACompu(s.red, \pi_1(s.compus[d])))$
12, 13, 14. $(\forall i, j : \text{nat}),\ 0 \leq i, j <$ longitud$(s.CaminosMinimos) \Rightarrow_L$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

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

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

1.3 Algoritmos

iCREARSISTEMA(**in** r : red) $\longrightarrow$ res : dcnet

$res.red \leftarrow r$

$n \leftarrow Longitud(\textsc{compus}(red))$ — O(1)

$i \leftarrow 0$

$j \leftarrow 0$ — O(1)

$res.Compus \leftarrow \textsc{CrearArreglo}(n)$ — O(n)

$res.CaminosMinimos \leftarrow \textsc{CrearArreglo}(n)$ — O(n)

var p : arreglo_dimensionable de puntero(conjLog(paquete))

while i<n **do** — $O(L * n^5)$

O(n)

  $res.CaminosMinimos[i] \leftarrow \textsc{CrearArreglo}(n)$ — O(n)

  s: $< nat, conjLog(paquete, <_{id}), conjLog(paquete, <_p),$
$conjLog(paquetePos, <_{id}), conjLog(paquetePos, <_p) >$

  $\pi_1(s) \leftarrow compu(r, i)$

  $\pi_2(s) \leftarrow \textsc{nuevo}()$

  $\pi_3(s) \leftarrow \textsc{nuevo}()$

  $\pi_4(s) \leftarrow \textsc{nuevo}()$

  $\pi_5(s) \leftarrow \textsc{nuevo}()$

  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)

$t_1 : < nat, conjLog(paquete, <_{id}), conjLog(paquete, <_p),$
$conjLog(paquetePos, <_{id}), conjLog(paquetePos, <_p) >$

$t_1 \leftarrow \textsc{Obtener}(origen(p),\ s.CompusPorPref)$ — $O(L)$

$t_2 : < nat, conjLog(paquete, <_{id}), conjLog(paquete, <_p),$
$conjLog(paquetePos, <_{id}), conjLog(paquetePos, <_p) >$

$t_2 \leftarrow \textsc{Obtener}(destino(p),\ s.CompusPorPref)$ — $O(L)$

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($\pi_5(t), p$') — $O(log(k))$

$O(L + log(k))$

1.4 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).

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 $\equiv$ $\{v =_{\text{obs}} v_0 \land$ 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 : $\widehat{\texttt{dcnet}}\ s \longrightarrow \widehat{\texttt{DCNet}}$ $\{\text{Rep}(s)\}$

$(\forall s : \widehat{\texttt{dcnet}})$
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)
$\quad p[i] \leftarrow vacio()$	O(1)
$\quad i++$	O(1)
end while	
$res \leftarrow p$	O(1)
	O(n)

IASIGNAR(**in/out** a : `acceso`, *in* p : `Nat`, *in* s : α)

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 : 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* : 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 $\equiv \{AList a(res) =_{\text{obs}} tuplasClaveDiccionario(d)\}$
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) $\equiv d : \widehat{\texttt{dicc}} \mid (\forall s : \text{String}) s \in e.claves =_{\text{obs}} def?(d, s) \land$
$((\forall s : \text{String}) Definido?(d, s)) \Rightarrow_{\text{L}} Definido?(e, s) \land_{\text{L}} (obtener(d, s) =_{\text{obs}} Significado(e, s))$

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 : α) $\longrightarrow$ *res* : dpn
$nodoClave : puntero(nodoClave) \leftarrow nuevoNodoClave(clave, d.claves, NULL)$
O(long(clave))

```
nodo : puntero(Nodo) ← NULL
i : nat ← 0
// Por ref
caracteres ← d.buckets                                    O(1)
if caracteres.esVacia() then                              O(1)
    caracteres = CrearHijos()                             O(1)
    d.bucket ← caracteres                                 O(1)
end if
while i ≤ Longitud(clave) do                              O(long(clave))
    nodo ← caracteres[ord(clave[i])]                      O(1)
    // Por ref
    caracteres ← nodo.hijos                               O(1)
    if caracteres.esVacia() then                          O(1)
        caracteres = CrearHijos()                         O(1)
        nodo.hijos ← caracteres                           O(1)
    end if
    i++                                                   O(1)
end while
nodo.hayS ← True                                          O(1)
nodo.significado ← 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 ← d.claves.agAtras(< clave, e >)             O(long(clave))
                                                          ----------------
                                                          O(long(clave))
```

```
iEliminar(in/out d : dpn, in clave : String) ⟶ res : dpn
nodo : puntero(Nodo) ← NULL
i : nat ← 0
// Por ref
caracteres ← d.buckets                                    O(1)
while i ≤ Longitud(clave) do                              O(long(clave))
    nodo ← caracteres[ord(clave[i])]                      O(1)
    // Por ref
    caracteres ← nodo.hijos                               O(1)
    i++                                                   O(1)
end while
nodo.hayS ← 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) ⟶ res : α
nodo : puntero(Nodo) ← NULL                               O(1)
buckets : puntero(Nodo) ← d.buckets                       O(1)
i ← 0                                                     O(1)
while i ≤ Longitud(clave) do
    nodo ← 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 $\equiv \{tuplasClaveSignificado(d) =_{\mathrm{obs}} siguientes(res) \wedge_{\mathrm{L}} aliasing(tuplasClaveSignificado(d), siguientes($
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 $\equiv \{res =_{\mathrm{obs}} haySiguiente(it)\}$
Descripción: Indica si hay siguiente
Complejidad: $O(1)$

HayAnterior(**in** it : `itDPN`) $\longrightarrow$ res : `bool`
Pre $\equiv \{true\}$
Post $\equiv \{res =_{\mathrm{obs}} hayAnterior(it)\}$
Descripción: Indica si hay anterior
Complejidad: $O(1)$

Siguiente(**in** it : `itDPN`) $\longrightarrow$ res : `<clave:String,significado:`α`>`
Pre $\equiv \{HaySiguiente(it)\}$
Post $\equiv \{res =_{\mathrm{obs}} siguiente(it)\}$
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 $\equiv$ $\{res \iff (\pi_1(p) < ancho(c) \wedge \pi_2(p) < alto(c))\}$
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 $\equiv$ $\{res \iff (\pi_2(p) = alto(c) - 1 \vee \pi_2(p) = 0)\}$
Descripción: Comprueba si una posición es un ingreso al campus.
Complejidad: O(1)

IngresoSup(**in** *c* : `campus`, *p* : `pos`) $\longrightarrow$ *res* : `bool`
Pre $\equiv$ {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`⟩ `,` `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 $def?(s, s.CompusPorPref), (\exists c : compu), esta?(c, s.Compus) \land \pi_1(c) = s \land$ 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.
$(\forall c : \text{nat})\ 0 \leq c < Longitud(s.compus) \Rightarrow_{\text{L}}$
$Longitud(s.compus[c].pN) = Longitud(s.compus[c].pN') \land$
$(\forall p : \text{paquetePos})\text{esta?}(p, s.compus[c].pN') \Rightarrow_{\text{L}}$
$\text{esta}(\pi_1(p), s.compus[c].pN) \land 0 \leq indiceOrigen(p) < Longitud(s.compus)$
$\land\ 0 \leq indiceDestino(p) < Longitud(s.compus)$
$\land\ 0 \leq posActual(p) < Longitud(s.compus)$
$\land\ \neg(indiceDestino(p) = indiceOrigen(p))$
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. $\forall s, t$: String def?$(s, s.CompusPorPref) \land$ def?$(t, s.CompusPorPref) \land s \neq t \Rightarrow_{\text{L}}$
obtener$(s, s.CompusPorPref) \bigcap$ obtener$(t, s.CompusPorPref) = \emptyset$
9. $\forall s$: String def?$(s, s.CompusPorPref) \Rightarrow_{\text{L}} \pi_1(\text{obtener}(s, s.CompusPorPref)) =$
$\pi_2(\text{obtener}(s, s.CompusPorPref))$
10. $Longitud(s.compus) = Longitud(CaminosMinimos(s)) \land$
$(\forall i : \text{nat})\ 0 \leq i < Longitud(s.compus) \Rightarrow_{\text{L}}$
$Longitud(s.CaminosMinimos[i]) = Longitud(s.compus)$
11. $(\forall f, d : \text{nat})\ \neg(f = d) \land 0 \leq f, d < Longitud(s.compus) \Rightarrow_{\text{L}}$
$CaminosMinimos[f][d] =$
$caminoMinimo(s.red, ipACompu(s.red, \pi_1(s.compus[f])), ipACompu(s.red, \pi_1(s.compus[d])))$
12, 13, 14. $(\forall i, j : \text{nat}),\ 0 \leq i, j <$ longitud$(s.CaminosMinimos) \Rightarrow_{\text{L}}$longitud$(s.CaminosMinimos) =$ longitud$(s.CaminosMinimos[i]) \land$ longitud$(s.CaminosMinimos[i][j]) <$ longitud$(s.CaminosMinimos) \land$
$(\forall e : \text{nat}), \text{esta?}(e, s.CaminosMinimos[i][j]) \Rightarrow$ pertenece$(e, s.CompusPorPref)$
15.$\forall c$: compu esta?$(c, s.Compus) \Rightarrow_{\text{L}} \pi_3(c) \leq \pi_3(s.Compus[s.LaQMasEnvio])$

Función de abstracción

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

$(\forall s : \widehat{\texttt{dcnet}})$
Abs$(s) \equiv dc : \widehat{\texttt{DCNet}}$ |
$red(dc) =^*(s.red) \land (\forall c : compu, c \in compus(dc))(enEspera(dc, c) =^*(enEspera(s, c)) \land$

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

4.3 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)
$res.Compus \leftarrow \text{CREARARREGLO}(n)$	O(n)
$res.CaminosMinimos \leftarrow \text{CREARARREGLO}(n)$	O(n)
var p : `arreglo_dimensionable de puntero(conjLog(paquete))`	
while i<n **do**	$O(L * n^5)$
	O(n)
$res.CaminosMinimos[i] \leftarrow \text{CREARARREGLO}(n)$	O(n)
s: $< nat, conjLog(paquete, <_{id}), conjLog(paquete, <_p),$ $conjLog(paquetePos, <_{id}), conjLog(paquetePos, <_p) >$	
$\pi_1(s) \leftarrow compu(r, i)$	
$\pi_2(s) \leftarrow \text{NUEVO}()$	
$\pi_3(s) \leftarrow \text{NUEVO}()$	
$\pi_4(s) \leftarrow \text{NUEVO}()$	
$\pi_5(s) \leftarrow \text{NUEVO}()$	
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`)

$t_1 : < nat, conjLog(paquete, <_{id}), conjLog(paquete, <_p),$ $conjLog(paquetePos, <_{id}), conjLog(paquetePos, <_p) >$	
$t_1 \leftarrow \text{OBTENER}(origen(p),\ s.CompusPorPref)$	$O(L)$
$t_2 : < nat, conjLog(paquete, <_{id}), conjLog(paquete, <_p),$ $conjLog(paquetePos, <_{id}), conjLog(paquetePos, <_p) >$	
$t_2 \leftarrow \text{OBTENER}(destino(p),\ s.CompusPorPref)$	$O(L)$
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($\pi_5(t), p$')	$O(log(k))$
	$O(L + log(k))$

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).