



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
Roizman, Violeta	273/11	violeroizman@gmail.com
Vázquez, Jérica	318/13	jesis_93@hotmail.com
Zavalla, Agustín	670/13	nkm747@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. DCNet	2
1.1. Interfaz	2
1.2. Representación	3
1.3. Algoritmos	6
1.4. Servicios Usados	9
2. Red	10
2.1. Interfaz	10
2.2. Representación	11
2.3. Invariante de representación	11
2.4. Función de abstracción	12
2.5. Algoritmos	12
2.5.1. Extensión módulo diccionario	15
3. ConjLog	16
3.1. Interfaz($\alpha, =_\alpha, <_\alpha$)	16
3.1.1. parámetros formales	16
3.2. Representación	17
3.3. Invariante de representación	18
3.4. Función de abstracción	18
3.5. Algoritmos	19
3.6. Auxiliares	23
3.7. Operaciones auxiliares de conj(α)	28
4. Diccionario por Prefijos	30
4.1. Interfaz	30
5. Paquete	32
5.1. Interfaz	32
5.2. Representación	33
6. PaquetePos	34
6.1. Interfaz	34
6.2. Representación	35
7. ColaP	36
7.1. Interfaz($\alpha, =_\alpha, <_\alpha$)	36
7.1.1. parámetros formales	36
7.2. Operaciones	36
7.3. Representación	36
7.3.1. Invariante de representación	37
7.4. Función de abstracción	37
7.5. Algoritmos	37

1 DCNet

Una DCNet es un sistema que tiene computadoras en red que reciben paquetes que envían a la computadora destino a cada segundo.

1.1 Interfaz

se explica con DCNET

usa Compu, Paquete, Red, dicePref, conjLog, conjLogP

géneros dcnet

Operaciones

CREARSISTEMA(**in** $r : \text{red}$) $\longrightarrow res : \text{dcnet}$

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

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

Descripción: Crea un sistema DCNet.

Complejidad: $O(L \times n^5)$

Aliasing: res.red es un puntero a la red que recibimos por parámetro

CREARPAQUETE(**in/out** $s : \text{dcnet}$, **in** $p : \text{paquete}$)

Pre $\equiv \{s =_{\text{obs}} s_0 \wedge (\forall p_0 : \text{paquete}, \text{paqueteEnTransito?}(p, s)) \neg(p_0 =_{\text{obs}} p) \wedge \text{destino}(p) \in \text{compus}(\text{red}) \wedge \text{origen}(p) \in \text{compus}(\text{red}) \wedge_L \text{haycamino?}(\text{destino}(p), \text{origen}(p), \text{red}(s))\}$

Post $\equiv \{s =_{\text{obs}} \text{crearPaquete}(s_0, p)\}$

Descripción: Crea un paquete y lo agrega a la computadora correspondiente.

Complejidad: $O(L + \log(k))$

AVANZARSEGUNDO(**in/out** $s : \text{dcnet}$)

Pre $\equiv \{s =_{\text{obs}} s_0\}$

Post $\equiv \{s =_{\text{obs}} \text{avanzarSegundo}(s_0)\}$

Descripción: Avanza un segundo el sistema. Todas las computadoras envían su respectivo paquete y en consecuencia se actualizan los paquetes en espera de cada una de ellas.

Complejidad: $O(n \times (L + \log(k)))$

Aliasing:

DAMERED(**in** $s : \text{dcnet}$) $\longrightarrow res : \text{puntero}(\text{red})$

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

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

Descripción: Devuelve la red de DCNet.

Complejidad: $O(1)$

Aliasing: Devuelve un puntero a la misma red que la que se pasó como parámetro para crear el sistema

CAMINORECORRIDO(**in** $s : \text{dcnet}$, **in** $p : \text{paquete}$) $\longrightarrow res : \text{secu}(\text{compu})$

Pre $\equiv \{\text{paqueteEnTransito?}(s, p)\}$

Post $\equiv \{res =_{\text{obs}} \text{caminoRecorrido}(s, p)\}$

Descripción: Devuelve el camino recorrido hasta el momento por un paquete.

Complejidad: $O(n \times \log(\max(n, k)))$

Aliasing: Devuelve puntero al camino recorrido que se encuentra en CaminosMinimos

CANTIDADENVIADOS(**in** $s : \text{dcnet}$, **in** $c : \text{compu}$) $\longrightarrow res : \text{nat}$

Pre $\equiv \{c \in \text{computadoras}(\text{red}(s))\}$

Post $\equiv \{res =_{\text{obs}} \text{cantidadEnviados}(s, c)\}$

Descripción: Devuelve la cantidad de paquetes enviados por una computadora.

Complejidad: $O(n)$

ENESPERA(**in** $s : \text{dcnet}$, **in** $c : \text{compu}$) $\longrightarrow res : \text{puntero}(\text{conjLogP}(\text{paquete}))$

Pre $\equiv \{c \in \text{computadoras}(\text{red}(s))\}$

Post $\equiv \{res =_{\text{obs}} \text{enEspera}(s, c)\}$

Descripción: Devuelve un puntero a los paquetes de la computadora.

Complejidad: $O(L)$

Aliasing: Hay aliasing entre res y el conjunto de paquetes de la computadora pasada por parámetro

LAQUEMASENVIO(**in** $s : \text{dcnet}$) $\longrightarrow res : \text{compu}$

Pre $\equiv \{true\}$

Post $\equiv \{res =_{\text{obs}} \text{laQueMasEnvio}(s, p)\}$

Descripción: Devuelve la computadora que más paquetes envió por referencia

Complejidad: $O(1)$

Aliasing: Hay aliasing entre res y la $compu$ en la estructura

PAQUETEENTRANSITO?(**in** $s : \text{dcnet}$, **in** $p : \text{paquete}$) $\longrightarrow res : \text{bool}$

Pre $\equiv \{true\}$

Post $\equiv \{res = \text{paqueteEnTransito?}(s, p)\}$

Descripción: Devuelve true si el paquete se encuentra en transito en el sistema

Complejidad: $O(n \times \log(k))$

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.

1.2 Representación

se representa con sistema

donde sistema es $\text{tupla}(\text{Compus} : \text{arreglo}(\text{tupla}(\text{IP} : \text{String}, \text{pN} : \text{puntero}(\text{conjLog}(\text{paquete})), \text{pN}' : \text{puntero}(\text{conjLog}(\text{paquetePos})), \text{\#PaquetesEnviados} : \text{nat}), \text{CompusPorPref} : \text{diccPref}(\text{compu}, \text{tupla}(\text{PorNom} : \text{conjLog}(\text{paquete}), \text{PorPrior} : \text{conjLog}(\text{paquete}), \text{PorNom}' : \text{conjLog}(\text{paquetePos}), \text{PorPrior}' : \text{conjLog}(\text{paquetePos})), \text{CaminosMinimos} : \text{arreglo}(\text{arreglo}(\text{arreglo}(\text{compu}))), \text{LaQMaseEnvio} : \text{nat}, \text{Red} : \text{red})$

Aclaración: cada vez que dice arreglo en esta estructura nos referimos a arreglo_dimensionable

Invariante de representación

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

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

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

Rep(s) \equiv

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

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

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

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

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

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

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

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

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

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

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

Función de abstracción

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

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

1.3 Algoritmos

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

$res \leftarrow s.comp[s.LaQMasEnvio].IP$	O(1)
	O(1)
IDAMERED (in $s : dcnet$) $\longrightarrow res : puntero(red)$	
$res \leftarrow \&(s.red)$	O(1)
	O(1)
IENESPERA (in $s : dcnet$, in $c : compu$) $\longrightarrow res : puntero(conjLogP(paquete))$	
$t : \langle nat, conjLog(paquete, <_{id}), conjLog(paquete, <_p),$ $conjLog(paquetePos, <_{id}), conjLog(paquetePos, <_p) \rangle$	
$t \leftarrow OBTENER(\pi_1(c), s.Comp[s.CompPorPref])$	O(L)
$res \leftarrow \&(\pi_3(t))$	O(1)
	O(L)
IAVANZARSEGUNDO (in/out $s : dcnet$)	
var $i : nat$	
$i \leftarrow 0$	O(1)
var $m : nat$	
$m \leftarrow s.Comp[s.LaQMasEnvio].\#PaquetesEnviados$	
while $i < LONGITUD(s.Comp)$ do	O(n)
var $paqYProxDes : arreglo_dimensionable$ de tupla de paquetePos y nat	
$paqYProxDes \leftarrow CREAMARREGLO(n)$	
var $IP : String$	
$IP \leftarrow s.Comp[i].IP$	
$t_1 : \langle nat, conjLog(paquete, <_{id}), conjLog(paquete, <_p),$ $conjLog(paquetePos, <_{id}), conjLog(paquetePos, <_p) \rangle$	
$t_1 \leftarrow OBTENER(IP, s.Comp[s.CompPorPref])$	O(L)
var $p : paquete$	
if $\neg VACIA?(\pi_5(t_1))$ then	
$p' \leftarrow MENOR(\pi_5(t_1))$	O(log(k))
BORRAR($\pi_2(t_1)$, PAQUETE(p'))	O(log(k))
BORRAR($\pi_3(t_1)$, PAQUETE(p'))	O(log(k))
BORRAR($\pi_4(t_1)$, p')	O(log(k))
BORRAR($\pi_5(t_1)$, p')	O(log(k))
$s.Comp[i].\#PaquetesEnviados \leftarrow s.comp[i].\#PaquetesEnviados + 1$	O(1)
$proxima \leftarrow s.CaminosMinimos[INDICEORIGEN(p')][INDICEDESTINO(p')][POSACTUAL(p') +$ 1]	O(L) (se copia)
if $s.Comp[i].\#PaquetesEnviados > max$ then	O(1)
$max \leftarrow i$	O(1)
end if	
$paqYProxDes[i] \leftarrow \langle p', proxima \rangle$	
else	
$paqYProxDes[i] \leftarrow 0$	
end if	
$i \leftarrow i + 1$	O(1)
end while	
$s.LaQMasEnvio \leftarrow max$	O(1)
$i \leftarrow 0$	O(1)
while $i < LONGITUD(s.Comp)$ do	O(n)
if $paqYProxDes[i] \neq 0$ then	O(1)
$p' \leftarrow \pi_1(paqYProxDes[i])$	O(1)

$proxima \leftarrow \pi_2(paqYProxDes[i])$	$O(L)$
if $\neg(\text{DESTINO}(\text{PAQUETE}(p')) = proxima)$ then	$O(L)$
$\text{ACTUALIZARINDICE}(p')$	$O(1)$
$t_2 :< nat, conjLog(paquete, <_{id}), conjLog(paquete, <_p),$ $conjLog(paquetePos, <_{id}), conjLog(paquetePos, <_p) >$	
$t_2 \leftarrow \text{OBTENER}(proxima, s.CompusPorPref)$	$O(L)$
$\text{INSERTAR}(\pi_2(t_2), \text{PAQUETE}(p'))$	$O(\log(k))$
$\text{INSERTAR}(\pi_3(t_2), \text{PAQUETE}(p'))$	$O(\log(k))$
$\text{INSERTAR}(\pi_4(t_2), p')$	$O(\log(k))$
$\text{INSERTAR}(\pi_5(t_2), p')$	$O(\log(k))$
end if	
end if	
end while	
<hr/>	
	$O(n \times (L + \log(k)))$
<hr/>	
$\text{ICANTIDADENVIADOS}(\text{in/out } s : \text{dcnet}, \text{in } c : \text{compu}) \longrightarrow res : \text{nat}$	
var $i : \text{nat}$	
$i \leftarrow 0$	$O(1)$
while $s.compus[i].IP \neq \pi_1(c)$ do	$O(n)$
$i \leftarrow i + 1$	$O(1)$
end while	
$res \leftarrow s.compus[i].\#PaquetesEnviados$	$O(1)$
<hr/>	
	$O(n)$
<hr/>	
$\text{IPAQUETEENTRANSITO?}(\text{in } s : \text{dcnet}, \text{in } p : \text{paquete}) \longrightarrow res : \text{bool}$	
var $i : \text{nat}$	
$i \leftarrow 0$	$O(1)$
var $b : \text{bool}$	
$b \leftarrow \neg(\text{PERTENECE?}(p, *(s.compus[i].pN)))$	$O(\log(k))$
while $b \wedge i < n$ do	$O(n)$
$i \leftarrow i + 1$	$O(1)$
$b \leftarrow \neg(\text{PERTENECE?}(p, *(s.compus[i].pN)))$	$O(\log(k))$
end while	
if $i = n \wedge b$ then	$O(1)$
$res \leftarrow false$	$O(1)$
else	
$res \leftarrow true$	$O(1)$
end if	
<hr/>	
	$O(n \times \log(k))$
<hr/>	
$\text{ICAMINORECORRIDO}(\text{in } s : \text{dcnet}, \text{in } p : \text{paquete}) \longrightarrow res : \text{secu}(\text{compu})$	
var $i : \text{nat}$	
$i \leftarrow 0$	$O(1)$
var $b : \text{bool}$	
$b \leftarrow \neg(\text{PERTENECE?}(p, *(s.compus[i].pN)))$	$O(\log(k))$
while b do	$O(n)$
$i \leftarrow i + 1$	$O(1)$
$b \leftarrow \neg(\text{PERTENECE?}(p, *(s.compus[i].pN)))$	$O(\log(k))$
end while	
$res \leftarrow s.CaminosMinimos[\text{INDICEORIGEN}(p')][i]$	$O(1)$
<hr/>	
	$O(n \times \log(k))$

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 Red

El módulo red permite crear una red de computadoras, agregar nuevas, conectarlas y averiguar el camino mínimo entre dos de ellas.

2.1 Interfaz

se explica con RED

géneros red

Operaciones

Aclaración: $n = \#(r.compus)$, $L = \text{Longitud de IP más larga}$

NUEVA() $\rightarrow res : \text{red}$

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

Post $\equiv \{res \equiv \text{iniciarRed}\}$

Descripción: Crea una red vacía

Complejidad: $O(1)$

Aliasing:

INTERFAZUSADA(**in** $r : \text{red}$, **in** $c : \text{compu}$, **in** $c1 : \text{compu}$) $\rightarrow res : \text{interfaz}$

Pre $\equiv \{\neg(c = c1) \wedge c \in \text{compus}(r) \wedge c1 \in \text{compus}(r)\}$

Post $\equiv \{res =_{\text{obs}} \text{InterfazUsada}(r, c, c1)\}$

Descripción: Retorna por copia la interfaz de la primer compu recibida por parámetro usada para conectarse con la segunda

Complejidad: $O(\#(\text{compus}(r)))$

Aliasing:

COMPUS(**in** $r : \text{red}$) $\rightarrow res : \text{conj}(\text{compu})$

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

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

Descripción: Devuelve el conjunto de compus

Complejidad: $O(1)$

Aliasing: Retorna el conjunto de computadoras por referencia

CONECTADAS?(**in** $r : \text{red}$, **in** $c : \text{compu}$, **in** $c1 : \text{compu}$) $\rightarrow res : \text{bool}$

Pre $\equiv \{\neg(c = c1) \wedge c \in \text{compus}(r) \wedge c1 \in \text{compus}(r)\}$

Post $\equiv \{res =_{\text{obs}} \text{conectadas?}(r)\}$

Descripción: Indica si dos compus estan conectadas

Complejidad: $O(\#(\text{compus}(r)))$

AGREGARCOMPU(**in/out** $r : \text{red}$, **in** $c : \text{compu}$)

Pre $\equiv \{r \equiv r_0 \wedge (\forall c1 : \text{compu})(c1 \in \text{computadoras}(r)) \Rightarrow \text{IP}(c) \neq \text{IP}(c1)\}$

Post $\equiv \{r \equiv \text{agregarComputadora}(r, c)\}$

Descripción: Agrega a la red r la computadora c

Complejidad: $O(n + \text{copy}(c))$

Aliasing:

CONECTAR(**in/out** $r : \text{red}$, **in** $c1 : \text{compu}$, **in** $i1 : \text{interfaz}$, **in** $c2 : \text{compu}$, **in** $i2 : \text{interfaz}$)

Pre $\equiv \{r \equiv r_0 \wedge c1 \in \text{computadoras}(r) \wedge c2 \in \text{computadoras}(r) \wedge \text{IP}(c1) \neq \text{IP}(c2)$

$\wedge \neg \text{conectadas}(r, c1, c2) \wedge \neg \text{UsaInterfaz?}(r, c1, i1) \wedge \neg \text{UsaInterfaz?}(r, c2, i2)\}$

Post $\equiv \{r \equiv \text{conectar}(r, c1, i1, c2, i2)\}$

Descripción: Conecta las computadoras c_1 y c_2

Complejidad: $O(n)$

Aliasing:

CAMINOMINIMO(**in** $r : \text{red}$, $in\ f : \text{compu}$, $in\ d : \text{compu}$) $\longrightarrow res : \text{secu}(\text{compu})$

Pre $\equiv \{c_1 \in \text{computadoras}(r) \wedge c_2 \in \text{computadoras}(r)\}$

Post $\equiv \{r \equiv \text{caminosMinimos}(r, c_1, c_2)\}$

Descripción: Devuelve los caminos mínimos entre dos computadoras

Complejidad: $O(L * n^3)$

Aliasing:

STRINGAINDICE(**in** $r : \text{red}$, **in** $c : \text{compu}$) $\longrightarrow res : \text{nat}$

Pre $\equiv \{c \in \text{computadoras}(r)\}$

Post $\equiv \{res \equiv \text{indice}(r, c)\}$

Descripción: Devuelve un nat distinto para cada IP, una vez creada la red siempre retorna el mismo para cada compu

Complejidad: $O(n)$

Aliasing:

USAINTERFAZ?(**in** $r : \text{red}$, **in** $c : \text{compu}$, **in** $i : \text{interfaz}$) $\longrightarrow res : \text{bool}$

Pre $\equiv \{c \in \text{computadoras}(r) \wedge i \in \text{interfaces}(c)\}$

Post $\equiv \{res \equiv \text{UsaInterfaz?}(r, c, i)\}$

Descripción: Devuelve TRUE si y solo si la interfaz está siendo utilizada

Complejidad: $O(n)$

Aliasing:

HAYCAMINO?(**in** $r : \text{red}$, **in** $c1 : \text{compu}$, **in** $c2 : \text{compu}$) $\longrightarrow res : \text{bool}$

Pre $\equiv \{c_1 \in \text{computadoras}(r) \wedge c_2 \in \text{computadoras}(r)\}$

Post $\equiv \{res \equiv \neg(\text{Vacía?}(\text{caminoMinimo}(r, c1, c2)))\}$

Descripción: Devuelve TRUE sí y solo sí hay un camino válido entre las dos computadoras

Complejidad: $O(L * n^3)$

Aliasing:

2.2 Representación

se representa con **redstr**

donde **redstr** es $\text{tupla}(\text{Compus} : \text{conj}(\text{compu}),$
 $\text{Conexiones} : \text{dicc}(\text{compu}, \text{dicc}(\text{interfaz}, \text{compu}))$)

donde **compu** es $\text{tupla}(\text{IP} : \text{string},$
 $\text{interfaces} : \text{conj}(\text{interfaz}))$

donde **interfaz** es **nat**

2.3 Invariante de representación

1. El conjunto de claves de conexiones está incluido o es igual al conjunto de compus
2. 'Simetría en las conexiones' : Cada una de las claves de conexiones está en los significados de las compus a las que está conectada
3. Las compus conectadas a c están en las claves de las conexiones
4. Las claves del significado de cada una de las compus ' c ' está incluida en el conjunto de interfaces de ' c '

Rep.

$$\begin{aligned}
& \text{claves}(\text{conexiones}(r)) \subseteq \text{compus}(r) \wedge \\
& ((\forall c : \text{compu}) c \in \text{claves}(r.\text{conexiones})) \Rightarrow_L \\
& \quad \text{claves}(\text{significado}(r.\text{conexiones}, c)) \subseteq c.\text{interfaces} \wedge \\
& ((\forall c_1 : \text{compu}) c_1 \in \text{significados}(\text{conexiones}(r), c)) \Rightarrow_L \\
& \quad c_1 \in \text{claves}(r.\text{conexiones}) \wedge_L c \in \text{significados}(\text{conexiones}(r), c_1) \wedge
\end{aligned}$$

2.4 Función de abstracción

$$\begin{aligned}
& \text{Abs} : \widehat{\text{redstr}} \text{ } rstr \longrightarrow \widehat{\text{red}} \quad \{\text{Rep}(rstr)\} \\
& (\forall rstr : \widehat{\text{redstr}}) \\
& \text{Abs}(rstr) \equiv r : \widehat{\text{red}} \mid \\
& \text{computadoras}(r) =_{\text{obs}} rstr.\text{compus} \wedge \\
& (\forall c_0, c_1 : \text{compu}) \text{conectadas?}(r, c_0, c_1) =_{\text{obs}} c_1 \in \text{significados}(rstr.\text{conexiones}, c_0) \wedge \\
& \text{interfazUsada}(r, c_0, c_1) =_{\text{obs}} \text{dameClave}(\text{significado}(rstr.\text{conexiones}, c_0), c_1)
\end{aligned}$$

Extensión TAD diccionario

$$\begin{aligned}
& \text{significados} : \text{dicc}(\text{compu}, \text{dicc}(\text{interfaz}, \text{compu})) \text{ } d \times \text{compu } c \rightarrow \text{conj}(\text{compu}) \\
& \text{significados}(d) = \text{obtenerSignificados}(\text{significado}(d, c), \text{claves}(\text{significado}(d, c))) \\
& \text{obtenerSignificados} : \text{dicc}(\text{interfaz}, \text{compu}) \text{ } d \times \text{conj}(\text{interfaz}) \text{ } ci \rightarrow \text{conj}(\text{compu}) \\
& \text{obtenerSignificados}(d, c) = \\
& \quad \text{if } (\#(c) = 0) \text{ then} \\
& \quad \quad \emptyset \\
& \quad \text{else} \\
& \quad \quad \text{significado}(d, \text{dameUno}(c)) \cup \text{obtenerSignificado}(d, \text{sinUno}(c)) \\
& \quad \text{fi}
\end{aligned}$$

2.5 Algoritmos

$$\begin{aligned}
& \text{INUEVA}() \longrightarrow res : \text{redstr} \\
& \quad res \leftarrow \text{CrearTupla}(\text{compus} : \text{conj}(\text{compu}), \text{conexiones} : \text{dicc}(\text{compu}, \text{dicc}(\text{interfaz}, \text{compu}))) \\
& \quad \quad \quad \text{O}(1) \\
& \quad res.\text{conexiones} \leftarrow \text{Vacio}() \quad \text{O}(1) \\
& \quad res.\text{compus} \leftarrow \text{Vacio}() \quad \text{O}(1) \\
& \quad \text{return } res \\
& \quad \quad \quad \hline \quad \quad \quad \text{O}(1) \\
& \text{IAGREGARCOMPU}(\text{in/out } r : \text{redstr}, \text{ in } c : \text{compu}) \\
& \quad \text{AgregarRapido}(r.\text{compus}, c) \quad \text{O}(1) \\
& \quad \quad \quad \hline \quad \quad \quad \text{O}(1) \\
& \text{ICONECTAR}(\text{in/out } r : \text{redstr}, \text{ in } c_0 : \text{compu}, \text{ in } c_1 : \text{compu}, \text{ in } i_0 : \text{interfaz}, \text{ in } i_1 : \text{interfaz}) \\
& \quad \text{if } \neg (\text{definido?}(r.\text{conexiones}, c_0)) \text{ then}
\end{aligned}$$

<i>definir</i> (<i>r.conexiones</i> , <i>c</i> ₀ , <i>Vacio</i> ())	$O(\#(r.compūs))$
end if	
if \neg (<i>definido?</i> (<i>r.conexiones</i> , <i>c</i> ₁)) then	
<i>definir</i> (<i>r.conexiones</i> , <i>c</i> ₁ , <i>Vacio</i> ())	$O(\#(r.compūs))$
end if	
<i>definir</i> (<i>significado</i> (<i>r.conexiones</i> , <i>c</i> ₀), <i>i</i> ₀ , <i>c</i> ₁)	$O(\#(r.compūs))$
<i>definir</i> (<i>significado</i> (<i>r.conexiones</i> , <i>c</i> ₁), <i>i</i> ₁ , <i>c</i> ₀)	$O(\#(r.compūs))$
<hr/>	
	$O(\#(r.compūs))$
ISTRINGAINDICE (in <i>r</i> : redstr , in <i>c</i> : compu) \rightarrow <i>res</i> : nat	
<i>itCompūs</i> \leftarrow <i>CrearIt</i> (<i>r.compūs</i>)	$O(1)$
// <i>L</i> = longitud de <i>IP</i> mas larga	
while <i>itCompūs.siguiente</i> () <i>IP</i> ! = <i>c.IP</i> do	$O(L)$
<i>res</i> ++	$O(1)$
<i>avanzar</i> (<i>itCompūs</i>)	$O(1)$
end while	
return <i>res</i>	$O(1)$
<hr/>	
	$O(\#(r.compūs) * L)$
IUSAINTERFAZ (in <i>r</i> : redstr , in <i>c</i> : compu , in <i>i</i> : interfaz) \rightarrow <i>res</i> : bool	
if <i>definido?</i> (<i>r.conexiones</i> , <i>c</i>) then	$O(\#(r.compūs))$
<i>res</i> \leftarrow <i>definido?</i> (<i>significado</i> (<i>r.conexiones</i> , <i>c</i>), <i>i</i>)	$O(\#(r.compūs))$
else	
<i>res</i> \leftarrow <i>false</i>	$O(1)$
end if	
return <i>res</i>	$O(1)$
<hr/>	
	$O(\#(r.compūs))$
ICONECTADAS? (in <i>r</i> : redstr , in <i>c</i> ₁ : compu , in <i>c</i> ₂ : compu) \rightarrow <i>res</i> : bool	
<i>res</i> \leftarrow <i>pertence</i> (<i>c</i> ₂ , <i>significados</i> (<i>significado</i> (<i>r.conexiones</i> , <i>c</i> ₁)))	$O(\#(r.compūs))$
return <i>res</i>	$O(1)$
<hr/>	
	$O(\#(r.compūs))$
IINTERFAZUSADA (in <i>r</i> : redstr , in <i>c</i> ₁ : compu , in <i>c</i> ₂ : compu) \rightarrow <i>res</i> : interfaz	
<i>res</i> \leftarrow <i>dameClave</i> (<i>significado</i> (<i>d</i> , <i>c</i> ₁), <i>c</i> ₂)	$O(\#(r.compūs))$
return <i>res</i>	$O(1)$
<hr/>	
	$O(\#(r.compūs))$
ICAMINOMINIMO (in <i>r</i> : red , in <i>f</i> : compu , in <i>d</i> : compu) \rightarrow <i>res</i> : Lista (compu)	
var <i>n</i> : nat \leftarrow <i>Longitud</i> (<i>r.compūs</i>)	$O(1)$
var <i>dist</i> : Arreglo (nat) \leftarrow <i>CrearArreglo</i> (<i>n</i>)	$O(n)$
var <i>prev</i> : Arreglo (nat) \leftarrow <i>CrearArreglo</i> (<i>n</i>)	$O(n)$
var <i>s</i> : nat <i>larr</i> <i>stringAIndice</i> (<i>r</i> , <i>f</i>)	$O(L * n)$
<i>dist</i> [<i>s</i>] \leftarrow 0	$O(1)$
<i>prev</i> [<i>s</i>] \leftarrow <i>NULL</i>	$O(1)$
var <i>haySiguiente</i> : bool \leftarrow \neg (<i>Longitud</i> (<i>r.compūs</i>) == 0)	$O(1)$
var <i>it</i> : itConj \leftarrow <i>CrearIt</i> (<i>r.compūs</i>)	$O(1)$
var <i>v</i> : nat \leftarrow 0	$O(1)$

```

vd : tupla(nat, nat) O(1)
// vd0, vd1 : Tupla(nat, nat)
// vd0 =α vd1 ⇔ π1(vd0) = π1(vd1)
// vd0 <α vd1 ⇔ (π2(vd0) = π2(vd1) ∧ π1(vd0) < π1(vd1)) ∨
// ((π2(vd0) < π2(vd1))

var cp : cp(tupla(nat, nat), <α) O(1)
// Inicializo las distancias y previos de cada compu O(1)
while haySiguiente do O(n * log(n))
    haySiguiente ← haySiguiente?(it) O(1)
    if ¬(v = s) then O(1)
        dist[v] ← infinito() O(1)
        prev[v] ← NULL O(1)
    end if
    vd ← CrearTupla(v, dist[v]) O(1)
    encolar(cp, vd) O(log(n))
    v ++ O(1)
    avanzar(it) O(1)
end while
// Calculo distancias
while ¬Vacia?(cp) do O(L * n3)

    var u : nat ← proximo(cp) O(1)
    desencolar(cp) O(log(n))
    // Vecinos de u que todavía están en cp
    var vecinos : Lista(compu) ← nuevosVecinos(r, u, cp) O(n)
    var itVecinos : itLista ← CrearIt(itVecinos) O(1)
    var haySiguiente : bool ← Vacia?(vecinos) O(1)
    while haySiguiente do O(L * n2)
        haySiguiente ← haySiguiente?(itVecinos) O(1)
        var vert : nat ← stringAIndice(anterior(siguiente(itVecinos))) O(L * n)

        var distAux : nat ← dist[π1(u)] + 1 O(1)
        if distAux < dist[vert] then O(1)
            dist[vert] ← distAux O(1)
            prev[vert] ← u O(1)
        end if
        avanzar(itVecinos) O(1)
    end while
end while
// Agrego desde el destino hacia la fuente las compus, recorriendolas sobre el arreglo prev y
// agregandolas por ref.
agAdelante(res, destino) O(1)
var cactual : nat ← stringAIndice(r, destino) O(L * n)
while ¬(cactual = s) do O(n)
    agAdelante(res, r.compus[prev[cactual]]) O(1)
    cactual ← prev[cactual] O(1)
end while
if stringAIndice(primero(res))! = s then O(n)
    res ← Vacia
end if
return res

```

$$O(L * n^3)$$

INUEVOSVECINOS(in $r : \text{redstr}$, in $u : \text{tupla}$, in $cp : \text{cp}$, in $dist : \text{Arreglo}(\text{nat})$) $\longrightarrow res : \text{Lista}(\text{compu})$

```

var vecinos : Lista(compu)  $\leftarrow$  significados( $r.conexiones, r.comp[ \pi_1(u) ]$ )
                                                                    O(n)
var itVecinos : itLista  $\leftarrow$  CrearIt(vecinos)
                                                                    O(1)
while haySiguiente do
  haySiguiente  $\leftarrow$  haySiguiente?(itVecinos)
                                                                    O(L * n2)
  var v : nat  $\leftarrow$  stringAIndice(stringAIndice(vec))
                                                                    O(1)
  var vecino : tupla(nat,nat)  $\leftarrow$  CrearTupla(v, dist[v])
                                                                    O(L*n)
  // Si el vecino es mayor que el de mas prioridad,
  // entonces todavia esta en la cola (no fue desencholado)
  if cp.maxP < vecino then
                                                                    O(1)
    AgAdelante(res,r.comp[ $\pi_1$ (vecino)])
                                                                    O(1)
  end if
end while
return ret
                                                                    O(1)

```

$$O(L * n^2)$$

2.5.1 Extensión módulo diccionario

SIGNIFICADOS(in $d : \text{dicc}(\alpha_0 \text{ dicc}(\alpha_1 \alpha_0))$, in $c : \alpha_0$) $\longrightarrow res : \text{Lista}(\alpha_0)$

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

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

Descripción: Retorna todos los significados de la clave c

Complejidad: $O(\#claves(\text{significado}(d, c)))$

Aliasing: Hay aliasing en res

SIGNIFICADOS(in $d : \text{dicc}(\alpha_0 \text{ dicc}(\alpha_1 \alpha_0))$, in $c : \alpha_0$) $\longrightarrow res : \text{Lista}(\alpha_0)$

```

var itSignificados : itDicc  $\leftarrow$  CrearIt(significado(d, c))
                                                                    O(1)
var haySiguiente : bool  $\leftarrow$  #claves(significado(d, c))! = 0
                                                                    O(1)
while haySiguiente do
  haySiguiente  $\leftarrow$  haySiguiente?(itDicc)
                                                                    O(n)
  // Se agregan los elementos por referencia
  AgAtras(res, anteriorSignificado(siguienteSignificado(itDicc)))
                                                                    O(1)
  avanzar(itDicc)
                                                                    O(1)
end while
return res
                                                                    O(1)

```

$$O(n)$$

3 ConjLog

3.1 Interfaz($\alpha, =_\alpha, <_\alpha$)

3.1.1 parámetros formales

géneros α

operaciones

• $=_\alpha \bullet : \alpha \times \alpha \rightarrow bool$ Relación de equivalencia

• $<_\alpha \bullet : \alpha \times \alpha \rightarrow bool$ Relación de orden

se explica con $CONJ(\alpha)$

géneros $conjLog(\alpha)$

Operaciones

NUEVO($\rightarrow res : conjLog(\alpha)$)

Pre $\equiv \{true\}$

Post $\equiv \{res =_{obs} \emptyset\}$

Descripción: Crea un nuevo conjLog vacío

Complejidad: $O(1)$

VACÍO?(**in** $cl : conjLog(\alpha)$) $\rightarrow res : bool$

Pre $\equiv \{true\}$

Post $\equiv \{res = (\emptyset?(cl))\}$

Descripción: Indica si el conjunto es vacío

Complejidad: $O(1)$

PERTENECE(**in** $cl : conjLog(\alpha)$, **in** $e : \alpha$) $\rightarrow res : bool$

Pre $\equiv \{true\}$

Post $\equiv \{res = (e \in cl)\}$

Descripción: Retorna un booleano que indica si el elemento pertenece al conjunto

Complejidad: $O(\log(\#(cl)))$

BUSCAR(**in** $cl : conjLog(\alpha)$, **in** $e : \alpha$) $\rightarrow res : \alpha$

Pre $\equiv \{e \in cl\}$

Post $\equiv \{res == e\}$

Descripción: Devuelve el elemento que se está buscando

Complejidad: $O(\log(\#(cl)))$

Aliasing: El elemento se devuelve por referencia y es modificable

MENOR(**in** $cl : conjLog(\alpha)$, **in** $e : \alpha$) $\rightarrow res : \alpha$

Pre $\equiv \{e \in cl\}$

Post $\equiv \{res == \max(cl)\}$

Descripción: Devuelve el menor elemento del conjunto

Complejidad: $O(\log(\#(cl)))$

Aliasing: El elemento se devuelve por referencia y es modificable

INSERTAR(**in/out** $cl : cl(\alpha)$, **in** $e : \alpha$)

Pre $\equiv \{cl_0 =_{obs} cl \wedge \neg(e \in cl)\}$

Post $\equiv \{cl_0 =_{obs} Agregar(cl_0, e)\}$

Descripción: Inserta un nuevo elemento en el conjunto

Complejidad: $O(\log(\#(cl)))$

Aliasing: Hay aliasing entre el elemento recibido y el que se inserta en el conjunto

BORRAR(**in/out** $cl : cl(\alpha)$, *in* $e : \alpha$)

Pre $\equiv \{cl_0 =_{\text{obs}} cl \wedge (e \in cl)\}$

Post $\equiv \{cl =_{\text{obs}} (cl_0 - \{e\})\}$

Descripción: Elimina el elemento e del conjunto cl , los iteradores que apunten a este elemento se indefinen

Complejidad: $O(\log(\#(cl)))$

3.2 Representación

se representa con `clog`

donde `clog` es `raiz : puntero(nodo)`

donde `nodo` es `tupla⟨der : puntero(nodo),
izq : puntero(nodo),
valor : α ,
padre : puntero(nodo),
fdb : nat⟩`

3.3 Invariante de representación

1. Para todas las raíces, la altura del subárbol derecho menos la altura del subárbol izquierdo de esa raíz es igual al fdb.
2. El fdb de todas las raíces es 0, 1 o -1.
3. Si un nodo no es una hoja del árbol entonces los padres de los hijos derecho e izquierdo son iguales y es el nodo
4. Si un nodo es una hoja del árbol entonces los hijos derecho e izquierdo del árbol son NULL
5. Para todos los nodos n, todos los nodos del subárbol derecho son mayores que n
6. Para todos los nodos n, todos los nodos del subárbol izquierdo son menores que n
7. No hay nodos repetidos
8. El padre de la raíz es NULL

3.4 Función de abstracción

$$\text{Abs} : \widehat{\text{clog}(\alpha)} \text{ } cl \longrightarrow \widehat{\text{conj}(\alpha)} \quad \{\text{Rep}(cl)\}$$

$$(\forall cl : \widehat{\text{clog}(\alpha)})$$

$$\text{Abs}(cl) \equiv c : \widehat{\text{conj}(\alpha)} \mid$$

$$((\forall e : \alpha) e \in c \Rightarrow_L \text{esta}(cl, e)) \wedge \text{size}(cl) = \#(c)$$

3.5 Algoritmos

IVACÍO? (in $cl : \text{conjLog}(\alpha)$) $\longrightarrow res : \text{bool}$	
$res \leftarrow cl == \text{NULL}$	$O(1)$
<hr/>	
IBORRAR (in/out $cl : \text{conjLog}(\alpha)$, in $e : \alpha$)	$O(1)$
var $\text{variandoHijoDerecho?} : \text{bool} \leftarrow \text{true}$	
var $clactual : \text{conjLog}(\alpha) \leftarrow cl$	$O(1)$
var $aBorrar : \text{conjLog}(\alpha)$	
if $(\neg(cl.der == \text{NULL}) \wedge \neg(cl.izq == \text{NULL}))$ then	$O(1)$
$clactual \leftarrow \text{IENCONTRARPADRE}(clactual, e)$	$O(\log(\text{size}(cl)))$
if $clactual.der! = \text{NULL} \wedge_L clactual.der.valor == e$ then	
$aBorrar \leftarrow clactual.der$	$O(1)$
else	
$aBorrar \leftarrow clactual.izq$	$O(1)$
end if	
var $mm : \text{conjLog}(\alpha) \leftarrow \text{IDAMEMAYORMENORES}(clactual)$	$O(\log(\text{size}(cl)))$
if $mm.valor == e$ then	$O(1)$
if $mm.padre.der! = \text{NULL} \wedge_L mm.padre.der.valor == mm.valor$ then	
$\text{variandoHijoDerecho?} \leftarrow \text{true}$	$O(1)$
$mm.padre.der = \text{NULL}$	
$mm.padre.fdb --$	
else	
$\text{variandoHijoDerecho?} \leftarrow \text{false}$	$O(1)$
$mm.padre.izq = \text{NULL}$	
$mm.padre.fdb ++$	
end if	
else	
var $mmValor : \alpha \leftarrow mm.valor$	$O(1)$
$aBorrar.valor \leftarrow mmValor$	$O(1)$
if $mm.izq! = \text{NULL}$ then	$O(1)$
$mm.valor \leftarrow mm.izq.valor$	$O(1)$
$mm.izq \leftarrow \text{NULL}$	$O(1)$
$mm.fdb ++$	$O(1)$
if $mm.padre.valor == e$ then	
$\text{variandoHijoDerecho?} \leftarrow \text{false}$	
else	
$\text{variandoHijoDerecho?} \leftarrow \text{true}$	
end if	
else	

```

    if  $mm.padre.valor == e$  then
         $mm.padre.izq = NULL$ 
         $variandoHijoDerecho? \leftarrow false$ 
    else
         $mm.padre.der = NULL$ 
         $variandoHijoDerecho? \leftarrow true$ 
    end if
end if
end if
iREBYRECALCFDB( $mm.padre, variandoHijoDerecho?, estoyBorrando?$ )
                                          $O(\log(size(cl)))$ 
else

    if  $cl.der == NULL \wedge cl.izq == NULL$  then
         $cl \leftarrow NULL$ 
    else

        if  $cl.der == NULL$  then

            if  $cl.izq.valor == e$  then
                 $cl.izq \leftarrow NULL$ 
            else
                 $cl.valor \leftarrow cl.izq.valor$ 
                 $cl.izq \leftarrow NULL$ 
            end if
        else

            if  $cl.der.valor == e$  then
                 $cl.der \leftarrow NULL$ 
            else
                 $cl.valor \leftarrow cl.der.valor$ 
                 $cl.der \leftarrow NULL$ 
            end if
        end if
    end if
end if
end if

```

$O(\log(size(cl)))$

Justificación de complejidad

Por álgebra de órdenes

iINSERTAR(in/out $cl : conjLog(\alpha)$, in $e : \alpha$)

```

var  $clactual : conjLog(\alpha) \leftarrow cl$ 
                                          $O(1)$ 

if  $\neg(cl.der == NULL) \wedge \neg(cl.izq == NULL)$  then
     $clactual \leftarrow iENCONTRARPADRE(clactual, e)$ 
                                          $O(\log(size(cl)))$ 

    if  $clactual.valor < e$  then

```

<pre> <i>clactual.der</i> ← tupla(<i>der</i> : NULL, <i>izq</i> : NULL, <i>valor</i> : <i>e</i>, <i>padre</i> : <i>clactual</i>, <i>fdb</i> : 0) IREBYRECALCFDB(<i>clactual</i>, <i>true</i>, <i>false</i>) else <i>clactual.izq</i> ← tupla(<i>der</i> : NULL, <i>izq</i> : NULL, <i>valor</i> : <i>e</i>, <i>padre</i> : <i>clactual</i>, <i>fdb</i> : 0) IREBYRECALCFDB(<i>clactual</i>, <i>false</i>, <i>false</i>) end if else if <i>cl.der</i> == <i>NULL</i> ∧ <i>cl.izq</i> == <i>NULL</i> then <i>cl</i> ← tupla(<i>der</i> : NULL, <i>izq</i> : NULL, <i>valor</i> : <i>e</i>, <i>padre</i> : <i>clactual</i>, <i>fdb</i> : 0) else if <i>cl.der</i>! = <i>NULL</i> then <i>cl.izq</i> ← tupla(<i>der</i> : NULL, <i>izq</i> : NULL, <i>valor</i> : <i>e</i>, <i>padre</i> : <i>cl</i>, <i>fdb</i> : 0) else <i>cl.der</i> ← tupla(<i>der</i> : NULL, <i>izq</i> : NULL, <i>valor</i> : <i>e</i>, <i>padre</i> : <i>cl</i>, <i>fdb</i> : 0) end if end if end if </pre>	<p>$O(1)$</p> <p>$O(1)$</p> <p>$O(1)$</p> <p>$O(1)$</p> <p>$O(1)$</p> <p>$O(1)$</p> <p>$O(1)$</p> <p>$O(1)$</p> <p>$O(1)$</p>	<hr style="border: 0; border-top: 1px solid black; margin-bottom: 5px;"/> $O(\log(\text{size}(cl)))$
--	--	--

Justificación de complejidad

Por álgebra de órdenes

$\text{IPERTENECE}(\text{in/out } cl : \text{conjLog}(\alpha), \text{ in } e : \alpha) \longrightarrow res : \text{bool}$

<pre> var <i>encontrado?</i> : bool ← <i>false</i> </pre>	<p>$O(1)$</p>
<pre> var <i>clactual</i> : conjLog(α) ← <i>cl</i> </pre>	<p>$O(1)$</p>
<pre> while (<i>clactual</i>! = <i>NULL</i>) ∧ ¬(<i>encontrado?</i>) do </pre>	<p>$O(1)$</p>

if $e > clactual.valor$ then	$O(1)$
$clactual \leftarrow clactual.der$	$O(1)$
else	
if $ce < clactual.valor$ then	$O(1)$
$clactual \leftarrow clactual.izq$	$O(1)$
else	
$encontrado? \leftarrow true$	$O(1)$
end if	
end if	
end while	
$clactual \leftarrow NULL$	$O(1)$
$res \leftarrow encontrado?$	$O(1)$
<hr/>	
$O(\log(size(cl)))$	

Justificación de complejidad

El ciclo recorre a lo sumo una rama del árbol (el árbol está ordenado), teniendo ésta como máximo la longitud del árbol (sus alturas no difieren en más de una hoja) que es $\log(n)$, con $n = size(cl)$

$IMENOR(in\ cl : conjLog(\alpha)) \longrightarrow res : \alpha$

var $clactual : conjLog(\alpha) \leftarrow cl$	$O(1)$
$clactual \leftarrow iMenorNodo(clactual)$	$O(\log(size(cl)))$
$res \leftarrow clactual.valor$	$O(1)$
<hr/>	
$O(\log(size(cl)))$	

Justificación de complejidad

Por álgebra de órdenes

$IBUSCAR(in\ cl : conjLog(\alpha),\ e : \alpha) \longrightarrow res : \alpha$

var $padre : conjLog(\alpha) \leftarrow iEncontrarPadre(cl, e)$	$O(\log(size(cl)))$
if $padre.der \neq NULL \wedge_L padre.der.valor == e$ then	$O(1)$
$res \leftarrow padre.der.valor$	
else	
$res \leftarrow padre.izq.valor$	
end if	
<hr/>	
$O(\log(size(cl)))$	

Justificación de complejidad

Por álgebra de órdenes

3.6 Auxiliares

IREBYRECALCFDB(**in/out** $cl : \text{conjLog}(\alpha)$, *in variandoHijoDerecho?* : **bool**, *in estoyBorrando?* : **bool**)

var $clactual : \text{conjLog}(\alpha) \leftarrow cl$ $O(1)$

var $termino? : \text{bool} \leftarrow false$ $O(1)$

if *estoyBorrando?* **then**

while $clactual! = NULL \wedge \neg(termino?)$ **do** $O(1)$

if *variandoHijoDerecho?* **then**

if $clactual.fdb == -1$ **then** $O(1)$

var $fdbIzq : \text{nat}$ $O(1)$

if $cl.izq! = NULL$ **then**
 $fdbIzq \leftarrow cl.izq$ $O(1)$
end if

if $cl.izq! = NULL \wedge_L cl.izq.fdb == 1$ **then** $O(1)$
iROTARLR($cl.izq$) $O(1)$
end if
iROTARLL(cl) $O(1)$

if $cl.izq! = NULL \wedge_L fdbIzq == 0$ **then** $O(1)$
 $termino? \leftarrow true$ $O(1)$
end if

else

if $cl.fdb == +1$ **then** $O(1)$
 $cl.fdb \leftarrow 0$ $O(1)$
 $termino? \leftarrow true$ $O(1)$

else
 $cl.fdb \leftarrow -1$ $O(1)$

end if

end if

else

if $clactual.fdb == -1$ **then** $O(1)$

var $fdbDer : \text{nat}$ $O(1)$

if $cl.der! = NULL$ **then** $O(1)$
 $fdbDer \leftarrow cl.der.fdb$ $O(1)$
end if

if $cl.der! = NULL \wedge_L fdbDer == 1$ **then** $O(1)$
iROTARRL($cl.der$) $O(1)$


```

    end if
    iROTARRR(cl) O(1)

    if fdbDer == 0 then O(1)
        termino? ← true O(1)
    end if
else
    if cl.fdb == -1 then O(1)
        cl.fdb ← 0 O(1)
        termino? ← true O(1)
    else
        cl.fdb ← +1 O(1)
    end if
end if
end if
variandoHijoDerecho ← (cl.padre! = NULL ∧L cl.padre.der.valor == cl.valor) O(1)
clactual ← clactual.padre O(1)
end while
else // No hubo borrado, entonces hubo una inserción

while clactual! = NULL ∧ ¬(termino?) do O(1)

    if variandoHijoDerecho? then

        if clactual.fdb == +1 then O(1)

            var fdbDer : nat O(1)

            if cl.der! = NULL then O(1)
                fdbDer ← cl.der.fdb O(1)
            end if

            if cl.der! = NULL ∧L fdbDer == -1 then O(1)
                iROTARRL(cl.der) O(1)
            end if
            iROTARRR(cl) O(1)
            termino? ← true
        else

            if clactual.fdb == -1 then O(1)
                clactual.fdb ← 0 O(1)
                termino? ← true O(1)
            else
                clactual.fdb ← 1 O(1)
            end if
        end if
    end if
end if
else

    if clactual.fdb == -1 then O(1)
        fdbIzq : nat O(1)

```

if $cl.izq! = NULL$ then	$O(1)$
$fdbIzq \leftarrow cl.izq.fdb$	
end if	
if $cl.izq! = NULL \wedge_L fdbIzq == +1$ then	$O(1)$
$iROTARLR(cl.izq)$	$O(1)$
end if	
$iROTARLL(cl)$	$O(1)$
$termino? \leftarrow true$	$O(1)$
else	
if $clactual.fdb == +1$ then	$O(1)$
$clactual.fdb \leftarrow 0$	$O(1)$
$termino? \leftarrow true$	$O(1)$
else	
$clactual.fdb \leftarrow -1$	$O(1)$
end if	
end if	
end if	
$variandoHijoDerecho \leftarrow (cl.padre! = NULL \wedge_L cl.padre.der.valor == cl.valor)$	$O(1)$
$clactual \leftarrow clactual.padre$	$O(1)$
end while	
end if	
	$O(\log(size(cl)))$

Justificación de complejidad

En éste ciclo, tanto para el borrado y para la inserción se tiene un nodo interno que fue modificado y a partir de éste se comienza a subir hasta llegar como máximo al nodo raíz del árbol, recorriendo como mucho la altura del árbol

$iROTARRR(\text{in/out } cl : \text{conjLog}(\alpha))$

var $nietoRR : \text{conjLog}(\alpha) \leftarrow cl.der.der$	$O(1)$
var $hijoDer : \text{conjLog}(\alpha) \leftarrow cl.der$	$O(1)$
var $hijoIzq : \text{conjLog}(\alpha) \leftarrow cl.izq$	$O(1)$
$cl.der \leftarrow NULL$	$O(1)$
$cl.izq = \text{tupla}(\text{der} : \text{hijoDer.der},$	$O(1)$
$izq : cl.izq,$	
$valor : cl.valor,$	
$padre : cl,$	
$fdb : 0)$	
$cl.izq.izq.padre \leftarrow cl.izq$	$O(1)$
$cl.izq.der.padre \leftarrow cl.izq$	$O(1)$
$cl.valor = hijoDer.valor$	$O(1)$
$cl.der = nietoRR$	$O(1)$
$cl.der.padre \leftarrow cl$	$O(1)$
	$O(1)$

Justificación de complejidad

Son operaciones sobre α y punteros

IROTARRL(**in/out** $cl : \text{conjLog}(\alpha)$)

var <i>nietoRR</i> : $\text{conjLog}(\alpha) \leftarrow cl.der.der$	$O(1)$
var <i>nietoRL</i> : $\text{conjLog}(\alpha) \leftarrow cl.der.izq$	$O(1)$
var <i>valorDer</i> : $\alpha \leftarrow cl.der.valor$	$O(1)$
<i>cl.der.valor</i> \leftarrow <i>nietoRL.valor</i>	$O(1)$
<i>nietoRR.izq</i> \leftarrow <i>nietoRL.der</i>	$O(1)$
<i>cl.der.der</i> \leftarrow <i>nietoRR</i>	$O(1)$
<i>cl.der.izq</i> \leftarrow <i>nietoRL.izq</i>	$O(1)$
<i>cl.der.der.padre</i> \leftarrow <i>cl.der</i>	$O(1)$
<i>cl.der.izq.padre</i> \leftarrow <i>cl.der</i>	$O(1)$
<i>cl.izq.fdb</i> \leftarrow +1	$O(1)$
<hr/>	
	$O(1)$

Justificación de complejidad

Son operaciones sobre α y punteros

IROTARLL(**in/out** $cl : \text{conjLog}(\alpha)$)

var <i>nietoLL</i> : $\text{conjLog}(\alpha) \leftarrow cl.izq.izq$	$O(1)$
var <i>hijoIzq</i> : $\text{conjLog}(\alpha) \leftarrow cl.izq$	$O(1)$
var <i>hijoDer</i> : $\text{conjLog}(\alpha) \leftarrow cl.der$	$O(1)$
<i>cl.izq</i> \leftarrow <i>NULL</i>	$O(1)$
<i>cl.der</i> = <i>tupla</i> (<i>der</i> : <i>cl.der</i> , <i>izq</i> : <i>hijoIzq.der</i> , <i>valor</i> : <i>cl.valor</i> , <i>padre</i> : <i>cl</i> , <i>fdb</i> : 0)	$O(1)$
<i>cl.der.izq.padre</i> \leftarrow <i>cl.der</i>	$O(1)$
<i>cl.der.der.padre</i> \leftarrow <i>cl.der</i>	$O(1)$
<i>cl.valor</i> = <i>hijoIzq.valor</i>	$O(1)$
<i>cl.izq</i> = <i>nietoLL</i>	$O(1)$
<i>cl.izq.padre</i> \leftarrow <i>cl</i>	$O(1)$
<hr/>	
	$O(1)$

Justificación de complejidad

Son operaciones sobre α y punteros

IROTARLR(**in/out** $cl : \text{conjLog}(\alpha)$)

var <i>nietoLL</i> : $\text{conjLog}(\alpha) \leftarrow cl.izq.izq$	$O(1)$
--	--------

var <i>nietoLR</i> : $\text{conjLog}(\alpha) \leftarrow \text{cl.izq.der}$	$O(1)$
var <i>valorIzq</i> : $\alpha \leftarrow \text{cl.izq.valor}$	$O(1)$
<i>cl.izq.valor</i> $\leftarrow \text{nietoRL.valor}$	$O(1)$
<i>nietoLL.izq</i> $\leftarrow \text{nietoLR.izq}$	$O(1)$
<i>cl.izq.izq</i> $\leftarrow \text{nietoLL}$	$O(1)$
<i>cl.izq.der</i> $\leftarrow \text{nietoLR.der}$	$O(1)$
<i>cl.izq.izq.padre</i> $\leftarrow \text{cl.izq}$	$O(1)$
<i>cl.izq.der.padre</i> $\leftarrow \text{cl.izq}$	$O(1)$
<i>cl.izq.fdb</i> $\leftarrow -1$	$O(1)$
	<hr/>
	$O(1)$

Justificación de complejidad

Son operaciones sobre α y punteros

$\text{IENCONTRARPADRE}(\text{in } cl : \text{conjLog}(\alpha), e : \alpha) \longrightarrow res : \text{conjLog}(\alpha)$

var <i>clactual</i> : $\text{conjLog}(\alpha)$	$O(1)$
var <i>encontrado?</i> : $\text{bool} \leftarrow (\text{clactual.der!} = \text{NULL} \wedge_L \text{clactual.der.valor} == e) \vee (\text{clactual.izq!} = \text{NULL} \wedge_L \text{clactual.izq.valor} == e)$	
while $\neg \text{encontrado?}$ do	
if $e > \text{clactual.valor}$ then	
<i>clactual</i> $\leftarrow \text{clactual.der}$	$O(1)$
else	
<i>clactual</i> $\leftarrow \text{clactual.izq}$	$O(1)$
end if	
<i>encontrado?</i> $\leftarrow (\text{clactual.der!} = \text{NULL} \wedge_L \text{clactual.der.valor} == e) \vee (\text{clactual.izq!} = \text{NULL} \wedge_L \text{clactual.izq.valor} == e)$	
end while	
<i>res</i> $\leftarrow \text{clactual}$	$O(1)$
	<hr/>
	$O(\text{size}(cl))$

Justificación de complejidad

El ciclo recorre a lo sumo una rama del árbol (el árbol está ordenado), teniendo ésta como máximo la altura del árbol (sus alturas no difieren en más de una hoja) que es $\log(n)$, con $n = \text{size}(cl)$

$\text{IDAMEMAYORMENORES}(\text{in } cl : \text{conjLog}(\alpha), e : \alpha) \longrightarrow res : \text{conjLog}(\alpha)$

var <i>clactual</i> : $\text{conjLog}(\alpha) \leftarrow cl$	$O(1)$
if $\text{clactual.izq!} = \text{NULL}$ then	$O(1)$
<i>clactual</i> $\leftarrow \text{iMayorNodo}(\text{clactual})$	$O(\log(\text{size}(cl)))$
end if	
<i>res</i> $\leftarrow \text{clactual}$	$O(1)$
	<hr/>

Justificación de complejidad

Por álgebra de órdenes

$\text{IMENORNODO}(\text{in } cl : \text{conjLog}(\alpha)) \longrightarrow res : \text{conjLog}(\alpha)$

var $clactual : \text{conjLog}(\alpha) \leftarrow cl$ $O(1)$

while $clactual.izq! = NULL$ **do**

$clactual \leftarrow clactual.izq$

end while

$res \leftarrow clactual$

$O(1)$

$O(\log(\text{size}(cl)))$

Justificación de complejidad

Para encontrar el menor nodo se recorre el árbol siempre a la izquierda, se alcanza a recorrer una única rama, es decir la altura del árbol

$\text{IMAYORNODO}(\text{in } cl : \text{conjLog}(\alpha)) \longrightarrow res : \text{conjLog}(\alpha)$

var $clactual : \text{conjLog}(\alpha) \leftarrow cl$ $O(1)$

while $clactual.der! = NULL$ **do**

$clactual \leftarrow clactual.der$

end while

$res \leftarrow clactual$

$O(1)$

$O(\log(\text{size}(cl)))$

Justificación de complejidad

Para encontrar el mayor nodo se recorre el árbol siempre a la derecha, se alcanza a recorrer una única rama, es decir la altura del árbol

$\text{SIZE}(\text{in } cl : \text{conjLog}(\alpha)) \longrightarrow res : \text{nat}$

if $cl == NULL$ **then**

$res \leftarrow 0$

else

$res \leftarrow 1 + iSize(cl.der) + iSize(cl.izq)$

end if

3.7 Operaciones auxiliares de $\text{conj}(\alpha)$

$menor : \text{conj}(\alpha) \rightarrow \alpha \quad \{\#(c) > 0\}$

$menor(c) =$

if $(\#(c) = 1)$ **then**

$dameUno(c)$

else

if $(dameUno(c) < menor(sinUno(c)))$ **then**

```
        dameUno(c)
    else
        menor(sinUno(c))
    fi
fi
```

4 Diccionario por Prefijos

El módulo Diccionario por prefijos provee un diccionario en el que las claves son secuencias no acotadas de caracteres. Con el se puede definir una clave, obtener un significado y eliminar una clave. Estas tres operaciones están definidas en tiempo $O(L)$ con L la máxima longitud del conjunto de las claves introducidas (y cuando se está definiendo una clave se incluye en el conjunto la clave a introducir).

4.1 Interfaz

parámetros formales

géneros β

se explica con $\text{DICCIONARIO}(\text{SECU}(\text{CHAR}), \beta)$

géneros $\text{diccPref}(\text{secu}(\text{char}), \beta)$

Operaciones

$\text{NUEVO}() \longrightarrow res : \text{diccPref}(\text{secu}(\text{char}), \beta)$

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

Post $\equiv \{(\forall p:\text{secu}(\text{char})) \neg(\text{def?}(p, res))\}$

Descripción: Crea un nuevo diccionario vacío

Complejidad: $O(1)$

$\text{DEF?}(\text{in } dp : \text{diccPref}(\text{secu}(\text{char}), \beta), \text{ in } p : \text{secu}(\text{char})) \longrightarrow res : \text{bool}$

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

Post $\equiv \{res = p \in \text{claves}(dp)\}$

Descripción: Devuelve true o false según si la clave está o no definida

Complejidad: $O(L)$

$\text{CLAVES}(\text{in } dp : \text{diccPref}(\text{secu}(\text{char}), \beta)) \longrightarrow res : \text{conj}(\text{secu}(\text{char}))$

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

Post $\equiv \{(\forall c:\text{secu}(\text{char})) c \in \text{claves}(dp) \iff \text{def?}(c, dp)\}$

Descripción: Devuelve un conjunto de las claves del diccionario

Complejidad: $O(n)$

$\text{DEFINIR}(\text{in/out } dp : \text{diccPref}(\text{secu}(\text{char}), \beta), \text{ in } p : \text{secu}(\text{char}), \text{ in } s : \beta)$

Pre $\equiv \{dp = dp_0 \wedge \neg \text{def?}(p, dp)\}$

Post $\equiv \{\text{def?}(p, dp) \wedge \text{obtener}(p, dp) =_{\text{obs}} s \wedge (\forall c \in \text{claves}(dp_0)) \text{def?}(c, dp)\}$

Descripción: Inserta una nueva clave con su significado en el diccionario

Complejidad: $O(L)$

$\text{OBTENER}(\text{in } dp : \text{diccPref}(\text{secu}(\text{char}), \beta), \text{ in } p : \text{secu}(\text{char})) \longrightarrow res : \beta$

Pre $\equiv \{\text{def?}(p, dp)\}$

Post $\equiv \{res = \text{obtener}(p, dp)\}$

Descripción: Retorna el significado de la clave pedida

Complejidad: $O(L)$

Aliasing: Devuelve res por referencia

$\text{ELIMINAR}(\text{in/out } dp : \text{diccPref}(\text{secu}(\text{char}), \beta), \text{ in } p : \text{secu}(\text{char}))$

Pre $\equiv \{dp = dp_0 \wedge \text{def?}(p, dp)\}$

Post $\equiv \{\neg \text{def?}(p, dp) \wedge (\forall c \in \text{claves}(dp_0), c \neq p) \text{def?}(c, dp)\}$

Descripción: Elimina del diccionario la clave deseada
Complejidad: $O(L)$

5 Paquete

Un Paquete representa a un paquete a partir de una tupla que contiene el id del paquete, la prioridad, el origen el destino y un indicador de en que parte de su camino está.

5.1 Interfaz

se explica con **PAQUETE**

géneros **paquete**

Operaciones

CREARPAQUETE(**in** $id : \text{nat}$, **in** $o : \text{compu}$, **in** $d : \text{compu}$, **in** $pr : \text{nat}$) $\longrightarrow res : \text{paquete}$

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

Post $\equiv \{pi_1(res) = id \wedge pi_2(res) = pr \wedge pi_3(res) = o \wedge pi_4(res) = d\}$

Descripción: Crea un paquete

Complejidad: $O(1)$

• $<_p$ •(**in** $p_1 : \text{paquete}$, **in** $p_2 : \text{paquete}$) $\longrightarrow res : \text{bool}$

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

Post $\equiv \{res = \text{true} \iff (\pi_2(p_1) = \pi_2(p_2) \wedge pi_1(p_1) < pi_1(p_2) \vee (\pi_1(p_1) < \pi_1(p_2)))\}$

Descripción: Define un orden en paquete según la prioridad

Complejidad: $O(1)$

• $<_{id}$ •(**in** $p_1 : \text{paquete}$, **in** $p_2 : \text{paquete}$) $\longrightarrow res : \text{bool}$

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

Post $\equiv \{res = \text{true} \iff id(p_1) < id(p_2)\}$

Descripción: Define un orden en paquete según el id

Complejidad: $O(1)$

ID(**in** $p : \text{paquete}$) $\longrightarrow res : \text{nat}$

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

Post $\equiv \{res = \pi_1(paquete)\}$

Descripción: *Getterdeid*

Complejidad: $O(1)$

PRIORIDAD(**in** $p : \text{paquete}$) $\longrightarrow res : \text{nat}$

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

Post $\equiv \{res = \pi_2(paquete)\}$

Descripción: *Getterdeprioridad*

Complejidad: $O(1)$

ORIGEN(**in** $p : \text{paquete}$) $\longrightarrow res : \text{Ip}$

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

Post $\equiv \{res = \pi_3(paquete)\}$

Descripción: *Getterdeorigen*

Complejidad: $O(1)$

DESTINO(**in** $p : \text{paquete}$) $\longrightarrow res : \text{Ip}$

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

Post $\equiv \{res = \pi_4(paquete)\}$

Descripción: *Getterdedestino*

Complejidad: $O(1)$

5.2 Representación

se representa con paquete

donde paquete es tupla \langle id : nat,
 origen : Ip,
 destino : Ip,
 prioridad : nat \rangle

6 PaquetePos

Un PaquetePos es

6.1 Interfaz

se explica con $\text{tupla} \langle : \text{Paquete},$
 $\quad \quad \quad : \text{nat},$
 $\quad \quad \quad : \text{nat},$
 $\quad \quad \quad : \text{nat} \rangle$

géneros $\quad \quad \quad \text{paquetePos}$

Operaciones

$\text{CREARPAQUETE}(\text{in } id : \text{nat}, \text{in } o : \text{compu}, \text{in } d : \text{compu}, \text{in } pr : \text{nat}) \longrightarrow res : \langle \text{paquete}, \text{nat}, \text{nat}, \text{nat} \rangle$

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

Post $\equiv \{pi_1(pi_1(res)) = id \wedge pi_2(pi_1(res)) = pr \wedge pi_3(pi_1(res)) = o \wedge pi_4(pi_1(res)) = d\}$

Descripción: Crea un paquete

Complejidad: $O(1)$

$\bullet \langle_p \bullet (\text{in } p_1 : \langle \text{paquete}, \text{nat}, \text{nat}, \text{nat} \rangle, \text{in } p_2 : \langle \text{paquete}, \text{nat}, \text{nat}, \text{nat} \rangle) \longrightarrow res : \text{bool}$

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

Post $\equiv \{res = \text{true} \iff (\pi_2(\pi_1(p_1)) = \pi_2(\pi_1(p_2)) \wedge \pi_1(\pi_1(p_1)) < \pi_1(\pi_1(p_2)) \vee (\pi_1(\pi_1(p_1)) < \pi_1(\pi_1(p_2))) \vee (\pi_1(\pi_1(p_2)) < \pi_1(\pi_1(p_1))))\}$

Descripción: Define un orden en paquete según la prioridad

Complejidad: $O(1)$

$\bullet \langle_{id} \bullet (\text{in } p_1 : \langle \text{paquete}, \text{nat}, \text{nat}, \text{nat} \rangle, \text{in } p_2 : \langle \text{paquete}, \text{nat}, \text{nat}, \text{nat} \rangle) \longrightarrow res : \text{bool}$

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

Post $\equiv \{res = \text{true} \iff (pi_1(pi_1(p_1)) < (pi_1(pi_1(p_2))))\}$

Descripción: Define un orden en paquete según el id

Complejidad: $O(1)$

$\text{GETPAQUETE}(\text{in } ppos : \langle \text{paquete}, \text{nat}, \text{nat}, \text{nat} \rangle) \longrightarrow res : \text{paquete}$

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

Post $\equiv \{res =_{\text{obs}} \pi_1(\text{paquete})\}$

Descripción: Getter de paquete

Complejidad: $O(1)$

$\text{INDICEORIGEN}(\text{in } p : \langle \text{paquete}, \text{nat}, \text{nat}, \text{nat} \rangle) \longrightarrow res : \text{nat}$

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

Post $\equiv \{res = \pi_2(p)\}$

Descripción: Getter de indiceOrigen

Complejidad: $O(1)$

$\text{INDICEDESTINO}(\text{in } p : \langle \text{paquete}, \text{nat}, \text{nat}, \text{nat} \rangle) \longrightarrow res : \text{nat}$

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

Post $\equiv \{res = \pi_3(p)\}$

Descripción: Getter de indiceDestino

Complejidad: $O(1)$

$\text{POSACTUAL}(\text{in } p : \langle \text{paquete}, \text{nat}, \text{nat}, \text{nat} \rangle) \longrightarrow res : \text{nat}$

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

Post $\equiv \{res = \pi_4(p)\}$

Descripción: Getter de posActual

Complejidad: $O(1)$

ACTUALIZARPOSACTUAL(**in/out** $p : < paquete, nat, nat, nat >$)

Pre $\equiv \{p_1 =_{\text{obs}} p\}$

Post $\equiv \{posActual(p) = posActual(p_1) + 1\}$

Descripción: Aumentar la posición actual

Complejidad: $O(1)$

6.2 Representación

se representa con paqPos

donde paqPos es tupla($paquete : paquete,$
 $indiceOrigen : nat,$
 $indiceDestino : nat,$
 $posActual : nat$)

Justificación de estructura

Las restricciones de complejidad del tipo dcnet para caminoRecorrido y enEspera nos obligaron a tener representadas dos estructuras distintas para almacenar los paquetes. En una de ellas tenemos los paquetes a retornar por la operación enEspera que devuelve los paquetes en la cola de cierta computadora y en la otra devolvemos una estructura similar a paquete (paquetePos) con información adicional acerca de la posición en el arreglo de compus en que se encuentra el paquete que nos permite encontrar el caminoRecorrido en el tiempo solicitado.

7 ColaP

7.1 Interfaz($\alpha, =_\alpha, <_\alpha$)

7.1.1 parámetros formales

géneros α

operaciones

• $=_\alpha$ • : $\alpha \times \alpha \rightarrow bool$ Relación de equivalencia

• $<_\alpha$ • : $\alpha \times \alpha \rightarrow bool$ Relación de orden

se explica con COLA(α)

géneros $cp(\alpha)$

7.2 Operaciones

NUEVO() $\rightarrow res : cp(\alpha)$

Pre $\equiv \{true\}$

Post $\equiv \{res =_{obs} vacia\}$

Descripción: Crea una nueva cp

Complejidad: O(1)

ENCOLAR(**in/out** $cp : cp(\alpha)$, *in* $e : \alpha$)

Pre $\equiv \{(cp_0 = cp)\}$

Post $\equiv \{cp =_{obs} encolar(cp_0, e)\}$

Descripción: Se inserta el elemento e en la cola, si el elemento ya existe, no hace nada

Complejidad: O(log(n)), n : $\#(cp.colalog)$

Aliasing: El elemento se inserta por referencia

PROXIMO(**in** $cp : cp(\alpha)$) $\rightarrow res : \alpha$

Pre $\equiv \{\neg(Vacia?(cp))\}$

Post $\equiv \{res =_{obs} proximo(cp)\}$

Descripción: Retorna el proximo elemento

Complejidad: O(1)

Aliasing: El elemento se devuelve por referencia, hay aliasing

DESENCOLAR(**in/out** $cp : cp(\alpha)$)

Pre $\equiv \{(cp_0 =_{obs} cp) \wedge \neg(Vacia?(cp))\}$

Post $\equiv \{cp =_{obs} desencolar(cp_0)\}$

Descripción: Quita el elemento de mas prioridad de la cola

Complejidad: O(log(n)), n = $\#(cp.colalog)$

7.3 Representación

se representa con cpstr

donde cpstr es maxP : α , colaLog : $conjLog(\alpha)$

7.3.1 Invariante de representación

1. $\neg(\text{Vacío?}(cp.colalog)) \Rightarrow_L cp.maxP = menor(cp.colalog)$

7.4 Función de abstracción

$Abs : \widehat{cp(\alpha)} \rightarrow \widehat{cola(\alpha)} \quad \{\text{Rep}(cp)\}$
 $(\forall cp : \widehat{cp(\alpha)})$
 $Abs(cp) \equiv c : \widehat{cola(\alpha)} \mid$
 $Vacia?(c) =_{\text{obs}} Vacio?(cp.colalog) \wedge proximo(cp) =_{\text{obs}} proximo(c) \wedge desencolar(c) =_{\text{obs}} desencolar(cp)$

7.5 Algoritmos

n: cant de elementos de la cola

NUEVA() $\rightarrow res : cp(\alpha)$	
$res \leftarrow CrearTupla(\alpha,)$	
$res.maxP \leftarrow NULL$	O(1)
$res.colalog \leftarrow nuevo()$	O(1)
return res	O(1)
<hr/>	
	O(1)
ENCOLAR(in/out cp : cp(α), in e : α)	
if $\neg(Vacia?(cp))$ then	O(1)
$cp.maxP \leftarrow e$	O(1)
insertar(cp.colalog, e)	O(1)
else	
if $cp.maxP > e$ then	O(1)
$cp.maxP \leftarrow e$	O(1)
end if	
insertar(cp.colalog, e)	O(log(n))
end if	
<hr/>	
	O(log(n))
DESENCOLAR(in/out cp : cp(α))	
Borrar(cp.colalog, menor(cp.colalog))	O(log(n))
$cp.maxP \leftarrow menor(cp.colalog)$	O(1)
<hr/>	
	O(log(n))
PROXIMO(in cp : cp(α)) $\rightarrow res : \alpha$	
$res \leftarrow cp.maxP$	O(1)
return res	O(1)
<hr/>	
	O(1)