

Algoritmos y Estructuras de Datos II

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo Práctico II

Grupo: 12

Integrante	LU	Correo electrónico
Pondal, Iván	078/14	ivan.pondal@gmail.com
Paz, Maximiliano León	251/14	m4xileon@gmail.com
Mena, Manuel	313/14	manuelmena1993@gmail.com
Demartino, Francisco	348/14	demartino.francisco@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Índice

1. Módulo DCNet

1.1. Interfaz

se explica con: DCNET.

géneros: dcnet.

1.1.1. Operaciones básicas de DCNet

INICIARDCNET(**in** $r : \text{red}$) $\rightarrow res : \text{dcnet}$

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

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

Complejidad: $O(n * (n + L))$ donde n es la cantidad de computadoras y L es la longitud de nombre de computadora mas larga

Descripción: crea una DCNet nueva tomando una red

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

Pre $\equiv \{$

$dcn =_{\text{obs}} dcn_0 \wedge$

$\neg((\exists p' : \text{paquete}) (\text{paqueteEnTransito}(dcn, p') \wedge \text{id}(p) = \text{id}(p') \wedge \text{origen}(p) \in \text{computadoras}(\text{red}(dcn)) \wedge_L$

$\text{destino}(p) \in \text{computadoras}(\text{red}(dcn)) \wedge_L \text{hayCamino?}(\text{red}(dcn), \text{origen}(p), \text{destino}(p))))$

$\}$

Post $\equiv \{dcn =_{\text{obs}} \text{crearPaquete}(dcn_0)\}$

Complejidad: $O(L + \log(k))$ donde L es la longitud de nombre de computadora mas larga y k es la longitud de la cola de paquetes mas larga

Descripción: crea un nuevo paquete

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

Pre $\equiv \{dcn =_{\text{obs}} dcn_0\}$

Post $\equiv \{dcn =_{\text{obs}} \text{avanzarSegundo}(dcn_0)\}$

Complejidad: $O(n * (L + \log(k)))$ donde n es la cantidad de computadoras, L es la longitud de nombre de computadora mas larga y k es la longitud de la cola de paquetes mas larga

Descripción: envia los paquetes con mayor prioridad a la siguiente compu

RED(**in** $dcn : \text{dcnet}$) $\rightarrow res : \text{red}$

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

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

Complejidad: $O(1)$

Descripción: devuelve la red de una DCNet

Aliasing: res es una referencia no modificable

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

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

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

Complejidad: $O(n * \log(k))$ donde n es la cantidad de computadoras y k es la longitud de la cola de paquetes mas larga

Descripción: devuelve el camino recorrido por un paquete

Aliasing: res es una referencia no modificable

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

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

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

Complejidad: $O(L)$ donde L es la longitud de nombre de computadora mas larga

Descripción: devuelve la cantidad de paquetes enviados por una compu

ENESPERA(**in** *dcn*: *dcnet*, **in** *c*: *compu*) \rightarrow *res* : *conj*(*paquete*)

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

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

Complejidad: $O(L)$ donde L es la longitud de nombre de computadora mas larga

Descripción: devuelve el conjunto de paquetes encolados en una compu

Aliasing: res es una referencia no modificable

PAQUETEENTRANSITO(**in** *dcn*: *dcnet*, **in** *p*: *paquete*) \rightarrow *res* : *bool*

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

Post $\equiv \{res =_{\text{obs}} \text{paqueteEnTransito}(dcn, p)\}$

Complejidad: $O(n * \log(k))$ donde n es es la cantidad de computadoras y k es la longitud de la cola de paquetes mas larga

Descripción: indica si el paquete está en transito

LAQUEMASENVIO(**in** *dcn*: *dcnet*) \rightarrow *res* : *compu*

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

Post $\equiv \{\text{alias}(res =_{\text{obs}} \text{laQueMasEnvio}(dcn))\}$

Complejidad: $O(1)$

Descripción: devuelve la compu que mas paquetes envió

Aliasing: res es una referencia no modificable

1.2. Representación

1.2.1. Representación de *dcnet*

dcnet se representa con *estr*

donde *estr* es *tupla*(*topología*: *red*,
vectorCompusDCNet: *vector*(*compuDCNet*),
diccCompusDCNet: *dicc_{trie}*(*puntero*(*compuDCNet*)),
conjPaquetesDCNet: *conj*(*paqueteDCNet*),
laQueMásEnvio: *puntero*(*compuDCNet*))

donde *compuDCNet* es *tupla*(*pc*: *puntero*(*compu*),
conjPaquetes: *conj*(*paquete*),
diccPaquetesDCNet: *dicc_{avl}*(*nat*, *itConj*(*paqueteDCNet*)),
colaPaquetesDCNet: *colaPrioridad*(*nat*, *itConj*(*paqueteDCNet*)),
paqueteAEnviar: *itConj*(*paqueteDCNet*), *enviados*: *nat*)

donde *paqueteDCNet* es *tupla*(*it*: *itConj*(*paquete*), *recorrido*: *lista*(*compu*))

donde *paquete* es *tupla*(*id*: *nat*, *prioridad*: *nat*, *origen*: *compu*, *destino*: *compu*)

donde *compu* es *tupla*(*ip*: *string*, *interfaces*: *conj*(*nat*))

1.2.2. Invariante de Representación

(I) Las compus de los elementos de *vectorCompusDCNet* son punteros a todas las compus de la topología

- (II) Las claves de `diccCompusDCNet` son todos los hostnames de la topología
- (III) Los significados de `diccCompusDCNet` son punteros que apuntan a las `compuDCNet` cuyo hostname equivale a su clave en `vectorCompusDCNet`
- (IV) `laQueMásEnvio` es un puntero a la `compuDCNet` en `vectorCompusDCNet` que más paquetes enviados tiene. Si no hay compus es `NULL`
- (V) Todos los paquetes en `conjPaquetes` de cada `compuDCNet` tienen id único y tanto su origen como destino existen en la topología
- (VI) El paquete en `conjPaquetes` tiene que tener en su recorrido a la `compuDCNet` en la que se encuentra y esta no puede ser igual al destino del recorrido
- (VII) Las claves de `diccPaquetesDCNet` son los id de los paquetes en `conjPaquetes`
- (VIII) Los significados de `diccPaquetesDCNet` contienen un `itConj` que apunta al paquete con el id equivalente a su clave y en recorrido, un camino mínimo válido para el origen del paquete y la compu en la que se encuentra
- (IX) La `colaPaquetesDCNet` es vacía si y sólo si `conjPaquetes` lo es, si no lo es, su próximo es un puntero a un paqueteDCNet de `diccPaquetesDCNet` que contiene un `itConj` cuyo siguiente es uno de los paquetes de `conjPaquetes` con mayor prioridad
- (X) La cantidad de enviados de una `compuDCNet` es igual o mayor a la cantidad de apariciones de esa compu en los caminos recorridos de paquetes en la red

Rep : `estr` \longrightarrow `bool`

$$\begin{aligned}
\text{Rep}(e) \equiv & \text{true} \iff \\
& (\#(\text{computadoras}(e.\text{topologia})) = \text{long}(e.\text{vectorCompusDCNet}) = \#(\text{claves}(e.\text{diccCompusDCNet}))) \wedge_L \\
& (\forall c: \text{compu})(c \in \text{computadoras}(e.\text{topologia}) \Rightarrow \\
& \quad (\\
& \quad (\exists cd: \text{compuDCNet})(\text{está?}(cd, e.\text{vectorCompusDCNet}) \wedge cd.\text{pc} = \text{puntero}(c)) \wedge \\
& \quad (\exists s: \text{string})(\text{def?}(s, e.\text{diccCompusDCNet}) \wedge s = c.\text{ip}) \\
& \quad) \\
&) \wedge_L \\
& (\forall cd: \text{compuDCNet})(\text{está?}(cd, e.\text{vectorCompusDCNet}) \Rightarrow_L \\
& \quad (\exists s: \text{string})(\text{def?}(s, e.\text{diccCompusDCNet}) \wedge \\
& \quad s = cd.\text{pc} \rightarrow \text{ip} \wedge_L \text{obtener}(s, e.\text{diccCompusDCNet}) = \text{puntero}(cd)) \\
&) \wedge_L \\
& (\exists cd: \text{compuDCNet})(\text{está?}(cd, e.\text{vectorCompusDCNet}) \wedge_L \\
& * (cd.\text{pc}) = \text{compuQueMásEnvió}(e.\text{vectorCompusDCNet}) \wedge e.\text{laQueMásEnvió} = \text{puntero}(cd)) \wedge_L \\
& (\forall cd_1: \text{compuDCNet})(\text{está?}(cd_1, e.\text{vectorCompusDCNet}) \Rightarrow \\
& \quad (\forall p_1: \text{paquete})(p_1 \in cd_1.\text{conjPaquetes} \Rightarrow \\
& \quad (\forall cd_2: \text{compuDCNet})(\text{está?}(cd_2, e.\text{vectorCompusDCNet}) \wedge cd_1 \neq cd_2) \Rightarrow \\
& \quad (\forall p_2: \text{paquete})(p_2 \in cd_2.\text{conjPaquetes} \Rightarrow p_1.\text{id} \neq p_2.\text{id}) \\
& \quad) \\
&) \\
&) \wedge_L \\
& (\forall cd: \text{compuDCNet})(\text{está?}(cd, e.\text{vectorCompusDCNet}) \Rightarrow \\
& \quad (\\
& \quad (\#(cd.\text{conjPaquetes}) = \#(\text{claves}(cd.\text{diccPaquetesDCNet}))) \wedge_L \\
& \quad (\forall p: \text{paquete})(p \in cd.\text{conjPaquetes} \Rightarrow \\
& \quad \quad (\\
& \quad \quad ((p.\text{origen} \in \text{computadoras}(e.\text{topologia}) \wedge p.\text{destino} \in \text{computadoras}(e.\text{topologia}) \wedge \\
& \quad \quad p.\text{destino} \neq *(cd.\text{pc})) \wedge_L \\
& \quad \quad (\exists sc: \text{secu}(\text{compu}))(sc \in \text{caminosMinimos}(e.\text{topologia}, p.\text{origen}, p.\text{destino}) \wedge \text{está?}(*(cd.\text{pc}), sc))) \wedge \\
& \quad \quad (\exists n: \text{nat})((\text{def?}(n, cd.\text{diccPaquetesDCNet}) \wedge p.\text{id} = n) \wedge_L \\
& \quad \quad (\text{Siguiente}(\text{obtener}(n, e.\text{diccPaquetesDCNet}).\text{it}) = p \wedge \\
& \quad \quad ((p.\text{origen} = *(cd.\text{pc}) \wedge \text{obtener}(n, e.\text{diccPaquetesDCNet}).\text{recorrido} = *(cd.\text{pc}) \bullet <>) \vee \\
& \quad \quad (p.\text{origen} \neq *(cd.\text{pc}) \wedge \\
& \quad \quad \text{obtener}(n, e.\text{diccPaquetesDCNet}).\text{recorrido} \in \text{caminosMinimos}(e.\text{topologia}, p.\text{origen}, *(cd.\text{pc})))) \\
& \quad \quad) \\
& \quad) \wedge_L \\
& \quad (\emptyset?(cd.\text{conjPaquetes}) \Leftrightarrow \text{vacía?}(cd.\text{colaPaquetesDCNet})) \wedge \\
& \quad (\neg \text{vacía?}(cd.\text{colaPaquetesDCNet}) \Rightarrow_L \\
& \quad (\exists n: \text{nat})(\text{def?}(n, cd.\text{diccPaquetesDCNet}) \wedge_L \\
& \quad \quad (\\
& \quad \quad \text{Siguiente}(\text{obtener}(n, cd.\text{diccPaquetesDCNet}).\text{it}) = \text{paqueteMásPrioridad}(cd.\text{conjPaquetes}) \wedge \\
& \quad \quad \text{proximo}(cd.\text{colaPaquetesDCNet}) = \text{puntero}(\text{obtener}(n, cd.\text{diccPaquetesDCNet})) \\
& \quad \quad) \\
& \quad) \wedge \\
& \quad (cd.\text{enviados} \geq \text{enviadosCompu}(*(cd.\text{pc}), e.\text{vectorCompusDCNet})) \\
& \quad) \\
&)
\end{aligned}$$

$\text{compuQueMásEnvió} : \text{secu}(\text{compuDCNet}) \text{ } scd \longrightarrow \text{compu}$	$\{\neg \text{vacía?}(scd)\}$
$\text{maxEnviado} : \text{secu}(\text{compuDCNet}) \text{ } scd \longrightarrow \text{nat}$	$\{\neg \text{vacía?}(scd)\}$
$\text{enviaronK} : \text{secu}(\text{compuDCNet}) \times \text{nat} \longrightarrow \text{conj}(\text{compu})$	
$\text{paqueteMásPrioridad} : \text{conj}(\text{paquete}) \text{ } cp \longrightarrow \text{paquete}$	$\{\neg \emptyset?(cp)\}$
$\text{paquetesConPrioridadK} : \text{conj}(cp) \times \text{nat} \longrightarrow \text{conj}(\text{paquete})$	
$\text{altaPrioridad} : \text{conj}(\text{paquetes}) \text{ } cp \longrightarrow \text{nat}$	$\{\neg \emptyset?(cp)\}$
$\text{enviadosCompu} : \text{compu} \times \text{secu}(\text{compuDCNet}) \longrightarrow \text{nat}$	

$\text{aparicionesCompu} : \text{compu} \times \text{conj}(\text{nat}) \text{ } cn \times \text{dicc}(\text{nat} \times \text{paqueteDCNet}) \text{ } dp \longrightarrow \text{nat} \quad \{\text{claves}(dp) \subseteq cn\}$

$\text{compuQueMásEnvió}(scd) \equiv \text{dameUno}(\text{enviaronK}(scd, \text{maxEnviado}(scd)))$

$\text{maxEnviado}(scd) \equiv \text{if } \text{vacía?}(\text{fin}(scd)) \text{ then } \text{prim}(scd).\text{enviados} \text{ else } \text{max}(\text{prim}(scd), \text{maxEnviado}(\text{fin}(scd))) \text{ fi}$

$\text{enviaronK}(scd, k) \equiv \text{if } \text{vacía?}(scd) \text{ then}$
 $\quad \emptyset$
 $\quad \text{else}$
 $\quad \quad \text{if } \text{prim}(scd).\text{enviados} = k \text{ then}$
 $\quad \quad \quad \text{Ag}(*(\text{prim}(scd).\text{pc}), \text{enviaronK}(\text{fin}(scd), k))$
 $\quad \quad \text{else}$
 $\quad \quad \quad \text{enviaronK}(\text{fin}(scd), k)$
 $\quad \text{fi}$
 fi

$\text{paqueteMásPrioridad}(dcn, cp) \equiv \text{dameUno}(\text{paquetesConPrioridadK}(cp, \text{altaPrioridad}(cp)))$

$\text{altaPrioridad}(cp) \equiv \text{if } \emptyset?(\text{sinUno}(cp)) \text{ then}$
 $\quad \text{dameUno}(cp).\text{prioridad}$
 else
 $\quad \text{min}(\text{dameUno}(cp).\text{prioridad}, \text{altaPrioridad}(\text{sinUno}(cp)))$
 fi

$\text{paquetesConPrioridadK}(cp, k) \equiv \text{if } \emptyset?(cp) \text{ then}$
 $\quad \emptyset$
 else
 $\quad \text{if } \text{dameUno}(cp).\text{prioridad} = k \text{ then}$
 $\quad \quad \text{Ag}(\text{dameUno}(cp), \text{paquetesConPrioridadK}(\text{sinUno}(cp), k))$
 $\quad \text{else}$
 $\quad \quad \text{paquetesConPrioridadK}(\text{sinUno}(cp), k)$
 $\quad \text{fi}$
 fi

$\text{enviadosCompu}(c, scd) \equiv \text{if } \text{vacía?}(scd) \text{ then}$
 $\quad 0$
 else
 $\quad \text{if } \text{prim}(scd) = c \text{ then}$
 $\quad \quad \text{enviadosCompu}(c, \text{fin}(scd))$
 $\quad \text{else}$
 $\quad \quad \text{aparicionesCompu}(c, \text{claves}(\text{prim}(scd).\text{diccPaquetesDCNet}),$
 $\quad \quad \text{prim}(scd).\text{diccPaquetesDCNet}) + \text{enviadosCompu}(c, \text{fin}(scd))$
 $\quad \text{fi}$
 fi

$\text{aparicionesCompu}(c, cn, dpd) \equiv \text{if } \emptyset?(cn) \text{ then}$
 $\quad 0$
 else
 $\quad \text{if } \text{está?}(c, \text{significado}(\text{dameUno}(cn), dpd).\text{recorrido}) \text{ then}$
 $\quad \quad 1 + \text{aparicionesCompu}(c, \text{sinUno}(cn), dpd)$
 $\quad \text{else}$
 $\quad \quad \text{aparicionesCompu}(c, \text{sinUno}(cn), dpd)$
 $\quad \text{fi}$
 fi

1.2.3. Función de Abstracción

$\text{Abs} : \text{estr } e \longrightarrow \text{dcnet} \quad \{\text{Rep}(e)\}$

$\text{Abs}(e) =_{\text{obs}} \text{dcn} : \text{dcnet} \mid \text{red}(dcn) = e.\text{topología} \wedge$
 $(\forall cdn : \text{compuDCNet})(\text{está?}(cdn, e.\text{vectorCompusDCNet}) \Rightarrow_{\text{L}}$
 $\text{enEspera}(dcn, *(cdn.\text{pc})) = cdn.\text{conjPaquetes} \wedge$
 $\text{cantidadEnviados}(dcn, *(cdn.\text{pc})) = cdn.\text{enviados} \wedge$
 $(\forall p : \text{paquete})(p \in cdn.\text{conjPaquetes} \Rightarrow_{\text{L}}$
 $\text{caminoRecorrido}(dcn, p) = \text{obtener}(p.\text{id}, cdn.\text{diccPaquetesDCNet}).\text{recorrido}$

)
)

2. Módulo Red

2.1. Interfaz

se explica con: RED.

géneros: red.

INICIARRED() $\rightarrow res : red$
Pre $\equiv \{true\}$
Post $\equiv \{res =_{obs} iniciarRed\}$
Complejidad: $O(1)$
Descripción: Crea una red nueva

AGREGARCOMPUTADORA(**in/out** $r : red$, **in** $c : compu$)
Pre $\equiv \{(r =_{obs} r_0) \wedge ((\forall c' : compu) (c' \in computadoras(r) \Rightarrow ip(c) \neq ip(c')))\}$
Post $\equiv \{r =_{obs} agregarComputadora(r_0, c)\}$
Complejidad: $O((n * L))$
Descripción: Agrega una computadora a la red
Aliasing: La compu se agrega por copia

CONECTAR(**in/out** $r : red$, **in** $c : compu$, **in** $c' : compu$, **in** $i : compu$, **in** $i' : compu$)
Pre $\equiv \{(r =_{obs} r_0) \wedge (c \in computadoras(r)) \wedge (c' \in computadoras(r)) \wedge (ip(c) \neq ip(c')) \wedge (\neg conectadas?(r, c, c')) \wedge (\neg usaInterfaz?(r, c, i) \wedge \neg usaInterfaz?(r, c', i'))\}$
Post $\equiv \{r =_{obs} conectar(r_0, c, i, c', i')\}$
Complejidad: $O(n! * (n^4))$
Descripción: Conecta dos computadoras

COMPUTADORAS(**in** $r : red$) $\rightarrow res : conj(compu)$
Pre $\equiv \{true\}$
Post $\equiv \{alias(res =_{obs} computadoras(r))\}$
Complejidad: $O(1)$
Descripción: Devuelve las computadoras de la red [Devuelve una referencia no modificable]

CONECTADAS?(**in** $r : red$, **in** $c : compu$, **in** $c' : compu$) $\rightarrow res : bool$
Pre $\equiv \{(c \in computadoras(r)) \wedge (c' \in computadoras(r))\}$
Post $\equiv \{res =_{obs} conectadas?(r, c, c')\}$
Complejidad: $O(1)$
Descripción: Indica si dos computadoras de la red estan conectadas

INTERFAZUSADA(**in** $r : red$, **in** $c : compu$, **in** $c' : compu$) $\rightarrow res : interfaz$
Pre $\equiv \{conectadas?(r, c, c')\}$
Post $\equiv \{res =_{obs} interfazUsada(r, c, c')\}$
Complejidad: $O(L + n)$
Descripción: Devuelve la interfaz con la cual se conecta c con c'

VECINOS(**in** $r : red$, **in** $c : compu$) $\rightarrow res : conj(compu)$
Pre $\equiv \{c \in computadoras(r)\}$
Post $\equiv \{res =_{obs} vecinos(r, c)\}$
Complejidad: $O(n^2)$
Descripción: Devuelve el conjunto de computadoras conectadas con c
Aliasing: Devuelve una copia de las computadoras conectadas a c

$$\text{USAINTERFAZ?}(\text{in } r : \text{red}, \text{in } c : \text{compu}, \text{in } i : \text{interfaz}) \rightarrow \text{res} : \text{bool}$$
$$\mathbf{Pre} \equiv \{c \in \text{computadoras}(r)\}$$
$$\mathbf{Post} \equiv \{res =_{\text{obs}} \text{usaInterfaz?}(r, c, i)\}$$

Complejidad: $O(L + n)$

Descripción: Indica si la interfaz i es usada por la computadora c

$$\text{CAMINOSMINIMOS}(\text{in } r : \text{red}, \text{in } c : \text{compu}, \text{in } c' : \text{compu}) \rightarrow res : \text{conj}(\text{lista}(\text{compu}))$$
$$\mathbf{Pre} \equiv \{(c \in \text{computadoras}(r)) \wedge (c' \in \text{computadoras}(r))\}$$
$$\mathbf{Post} \equiv \{\text{alias}(res =_{\text{obs}} \text{caminosMinimos}(r, c, i))\}$$

Complejidad: $O(L)$

Descripción: Devuelve el conjunto de caminos minimos de c a c'

Aliasing: Devuelve una referencia no modificable

$$\text{HAYCAMINO?}(\text{in } r : \text{red}, \text{in } c : \text{compu}, \text{in } c' : \text{compu}) \rightarrow \text{res} : \text{bool}$$
$$\mathbf{Pre} \equiv \{(c \in \text{computadoras}(r)) \wedge (c' \in \text{computadoras}(r))\}$$
$$\mathbf{Post} \equiv \{res =_{\text{obs}} \text{hayCamino?}(r, c, i)\}$$

Complejidad: $O(L)$

Descripción: Indica si existe algún camino entre c y c'

$$\text{COPIAR}(\text{in } r : \text{red}) \rightarrow res : \text{red}$$
$$\mathbf{Pre} \equiv \{\text{true}\}$$
$$\mathbf{Post} \equiv \{res =_{\text{obs}} r\}$$

Complejidad: $O(n! * (n^6))$

Descripción: Devuelve una copia la red

$$\bullet = \bullet(\text{in } r : \text{red}, \text{in } r' : \text{red}) \rightarrow \text{res} : \text{bool}$$
$$\mathbf{Pre} \equiv \{\text{true}\}$$
$$\mathbf{Post} \equiv \{res =_{\text{obs}} (r =_{\text{obs}} r')\}$$

Complejidad: $O(n + L^2)$

Descripción: Indica si r es igual a r'

2.2. Representación

2.2.1. Estructura

red se representa con estr

donde **estr** es $\text{tupla}(\text{compus: conj}(\text{compu}) ,$
 $\text{dns: dicc}_{Trie}(\text{nodoRed}))$

donde `nodoRed` es `tupla(pc: puntero(compu) ,`
`caminos: diccTrie(conj(lista(compu))) ,`
`conexiones: diccLineal(nat, puntero(nodoRed)))`

donde `compu` es `tupla(ip: string, interfaces: conj(nat))`

2.2.2. Invariante de Representación

- (I) Todas los elementos de *compus* deben tener IPs distintas.

- (II) Para cada compu, el trie *dns* define para la clave <IP de esa compu> un **nodoRed** cuyo *pc* es puntero a esa compu.
- (III) **nodoRed.conexiones** contiene como claves todas las interfaces usadas de la compu *c* (que tienen que estar en *pc.interfaces*)
- (IV) Ningun nodo se conecta con si mismo.
- (V) Ningun nodo se conecta a otro a traves de dos interfaces distintas.
- (VI) Para cada **nodoRed** en *dns*, *caminos* tiene como claves todas las IPs de las compus de la red (**estr.compuls**), y los significados corresponden a todos los caminos mínimos desde la compu *pc* hacia la compu cuya IP es clave.

Rep : estr \rightarrow bool

Rep(e) \equiv true \iff (

(($\forall c1, c2$: compu) ($c1 \neq c2 \wedge c1 \in e.compus \wedge c2 \in e.compus$) \Rightarrow $c1.ip \neq c2.ip$) \wedge

(($\forall c$: compu) ($c \in e.compus \Rightarrow$
($\text{def?}(c.ip, e.dns) \wedge_L \text{obtener}(c.ip, e.dns).pc = \text{puntero}(c)$)
)) \wedge

(($\forall i$: string, n : nodoRed) (($\text{def?}(i, e.dns) \wedge_L n = \text{obtener}(i, e.dns)$) \Rightarrow
($\exists c$: compu) ($c \in e.compus \wedge (n.pc = \text{puntero}(c))$)
)) \wedge

(($\forall i$: string, n : nodoRed) (($\text{def?}(i, e.dns) \wedge_L n = \text{obtener}(i, e.dns)$) \Rightarrow
($(\forall t$: nat) ($\text{def?}(t, n.conexiones) \Rightarrow (t \in n.pc \rightarrow \text{interfaces}))$)
)) \wedge

(($\forall i$: string, n : nodoRed) (($\text{def?}(i, e.dns) \wedge_L n = \text{obtener}(i, e.dns)$) \Rightarrow
($(\forall t$: nat) ($\text{def?}(t, n.conexiones) \Rightarrow_L (\text{obtener}(t, n.conexiones) \neq \text{puntero}(n))$)
)) \wedge

(($\forall i$: string, n : nodoRed) (($\text{def?}(i, e.dns) \wedge_L n = \text{obtener}(i, e.dns)$) \Rightarrow
($(\forall t1, t2$: nat) ($(t1 \neq t2 \wedge \text{def?}(t1, n.conexiones) \wedge \text{def?}(t2, n.conexiones)) \Rightarrow_L$
($\text{obtener}(t1, n.conexiones) \neq \text{obtener}(t2, n.conexiones)$)
))
)) \wedge

(($\forall i1, i2$: string, $n1, n2$: nodoRed) ((
($\text{def?}(i1, e.dns) \wedge_L n1 = \text{obtener}(i1, e.dns)$) \wedge
($\text{def?}(i2, e.dns) \wedge_L n2 = \text{obtener}(i2, e.dns)$)
) \Rightarrow ($\text{def?}(i2, n1.camino) \wedge_L \text{obtener}(i2, n1.camino) = \text{darCaminosMinimos}(n1, n2)$)
))

)

vecinas	: nodoRed	\rightarrow conj(nodoRed)
auxVecinas	: nodoRed \times dicc(nat \times puntero(nodoRed))	\rightarrow conj(nodoRed)
secusDeLongK	: conj(secu(α)) \times nat	\rightarrow conj(secu(α))
longMenorSec	: conj(secu(α)) <i>secus</i>	\rightarrow nat $\{ \neg \emptyset?(\text{secus}) \}$
darRutas	: nodoRed $nA \times$ nodoRed $nB \times$ conj(pc) \times secu(nodoRed)	\rightarrow conj(secu(nodoRed))
darRutasVecinas	: conj(pc) <i>vec</i> \times nodoRed $n \times$ conj(pc) \times secu(nodoRed)	\rightarrow conj(secu(nodoRed))
darCaminosMinimos	: nodoRed $n1 \times$ nodoRed $n1$	\rightarrow conj(secu(compu))

vecinas(n) \equiv auxVecinas($n, n.conexiones$)

auxVecinas(n, cs) \equiv **if** $\emptyset?(cs)$ **then**
 \emptyset
else
 Ag($\text{obtener}(\text{dameUno}(\text{claves}(cs)), cs)$, auxVecinas($n, \text{sinUno}(cs)$))
fi

```

secusDeLongK(secus, k)      ≡ if ∅?(secus) then
                               ∅
                             else
                               if long(dameUno(secus)) = k then
                                 dameUno(secus) ∪ secusDeLongK(sinUno(secus), k)
                               else
                                 secusDeLongK(sinUno(secus), k)
                               fi
                             fi
longMenorSec(secus)        ≡ if ∅?(sinUno(secus)) then
                               long(dameUno(secus))
                             else
                               min(long(dameUno(secus)),
                                   longMenorSec(sinUno(secus)))
                             fi
darRutas(nA, nB, rec, ruta) ≡ if nB ∈ vecinas(nA) then
                               Ag(ruta ∘ nB, ∅)
                             else
                               if ∅?(vecinas(nA) - rec) then
                                 ∅
                               else
                                 darRutas(dameUno(vecinas(nA) - rec),
                                           nB, Ag(nA, rec),
                                           ruta ∘ dameUno(vecinas(nA) - rec)) ∪
                                 darRutasVecinas(sinUno(vecinas(nA) - rec),
                                                  nB, Ag(nA, rec),
                                                  ruta ∘ dameUno(vecinas(nA) - rec))
                               fi
                             fi
darRutasVecinas(vecinas, n, rec, ruta) ≡ if ∅?(vecinas) then
                                           ∅
                                         else
                                           darRutas(dameUno(vecinas), n, rec, ruta) ∪
                                           darRutasVecinas(sinUno(vecinas), n, rec, ruta)
                                         fi
darCaminosMinimos(nA, nB) ≡ secusDeLongK(darRutas(nA, nB, ∅, <>),
                                           longMenorSec(darRutas(nA, nB, ∅, <>)))

```

2.2.3. Función de Abstracción

Abs : estr e \longrightarrow red $\{Rep(e)\}$
 Abs(e) =_{obs} r: red |

2.2.4. Función de Abstracción

2.3. Algoritmos

```

iIniciarRed () → res: red
  res.compuls ← Vacio ()
  res.dns ← Vacio ()
Complejidad : O(1)

```

```

iAgregarComputadora (in/out r: red, in c: compu)
  AgregoCompuNuevaAlResto (r.dns, c)

```

O(n*L)

AgregarRapido(r.compbus, c)	O(1)
Definir(r.dns, compu.ip, Tupla(&c, Vacio(), Vacio())>)	O(L)
InicializarConjCaminos(r, c)	O(n*L)
Complejidad : $O(n * L)$	

AgregoCompuNuevaAlResto (in/out r: red, in c: compu)	
itCompbus:itConj(compu) ← CrearIt(r.compbus)	O(1)
while HaySiguiente?(itCompbus) do	O(1)
nr:nodoRed ← Significado(r.dns, Siguiente(itCompbus).ip)	O(L)
Definir(nr.caminos, c.ip, Vacio())	O(L)
Avanzar(itCompbus)	O(1)
end while	O(n*L)
Complejidad : $O(n * L)$	

InicializarConjCaminos (in/out r: red, in c: compu)	
itCompbus:itConj(compu) ← CrearIt(r.compbus)	O(1)
cams:diccTrie(ip, conj(lista(compu))) ←	
Significado(r.dns, c.ip).caminos	O(L)
while HaySiguiente?(itCompbus) do	O(1)
Definir(cams, Siguiente(itCompbus).ip, Vacio())	O(L)
Avanzar(itCompbus)	O(1)
end while	O(n*L)
Complejidad : $O(n * L)$	

iConectar (in/out r: red, in c ₀ : compu, in c ₁ : compu, in i ₀ : compu, in i ₁ : compu)	
nr0:nodoRed ← Significado(r.dns, c ₀ .ip)	O(L)
nr1:nodoRed ← Significado(r.dns, c ₁ .ip)	O(L)
DefinirRapido(nr0.conexiones, i ₀ , nr1)	O(1)
DefinirRapido(nr1.conexiones, i ₁ , nr0)	O(1)
CrearTodosLosCaminos(r)	O(n!*(n ⁴))
Complejidad : $O(n! * (n^4))$	

CrearTodosLosCaminos (in/out r: red)	
itCompbus:itConj(compu) ← CrearIt(r.compbus)	O(1)
while HaySiguiente?(itCompbus) do	O(1)
nr:nodoRed ← Significado(d, Siguiente(itCompbus).ip)	O(L)
AsignarCaminosMinimos(nr, r)	O(n!*(n ³))
Avanzar(itCompbus)	O(1)
end while	O(n!*(n ⁴))
Complejidad : $O(n! * (n^4))$	

AsignarCaminosMinimos (in/out nr: nodoRed, in r: red)	
itCompbus:itConj(compu) ← CrearIt(r.compbus)	O(1)
while HaySiguiente?(itCompbus) do	O(1)
Definir(nr.caminos, Siguiente(itCompbus).ip)	O(L)
CrearCaminosMinimos(nr, Siguiente(itCompbus).ip)	O(n!*(n ²))
Avanzar(itCompbus)	O(1)
end while	O(n!*(n ³))
Complejidad : $O(n! * (n^3))$	

```

CrearCaminosMinimos (in desde: nodoRed, in hasta: compu) → res: conj(lista(compu))
  camMins: conj( lista (compu)) ← Vacio()                                O(1)
  itCamMins: itConj( conj( lista (compu))) ← CrearIt (camMins)          O(1)
  pendientes: pila(nodoRed) ← Vacia()                                   O(1)
  ApilarConj( pendientes , NodosVecinos( desde . conexiones ))          O(n)
  visitados: conj(nodoRed) ← Vacio()                                    O(1)
  cam: lista (compu) ← Vacia()                                          O(1)
  AgregarAdelante(cam, *(desde.pc))                                    O(1)
  while ¬EsVacía?(pendientes) do                                       O(1)
    AgregarRapido( visitados ,Tope(pendientes))                        O(1)
    AgregarAdelante(cam, *(Tope(pendientes).pc))                      O(1)
    if (Tope(pendientes).pc→ip = hasta.ip) then                        O(L)
      if (Vacío?(camMins) ∨ (Longitud(cam) = Longitud(Siguiente
        (itCamMins)))) then                                           O(n)
        AgregarRapido(camMins , cam)                                   O(1)
      else
        if (Longitud(cam) < Longitud(Siguiente(itCamMins))) then     O(n)
          camMins ← Vacio()                                           O(1)
          AgregarRapido(camMins , cam)                                   O(1)
        endif                                                         O(n)
      endif                                                            O(n)
      Comienzo(cam)                                                    O(1)
    end if                                                            O(n)
    Desapilar( pendientes )                                           O(1)
    nv: conj(nodoRed) ← VecinasNoVisitadas(Tope(pendientes), visitados) O(n2)
    if (Vacía?(nv))                                                    O(1)
      Comienzo(cam)                                                    O(1)
    else
      ApilarConj( pendientes , nv)                                     O(n)
    end if                                                            O(n)
  end while                                                            O(n! * n2)
Complejidad : O(n! * (n2))

```

```

VecinasNoVisitadas (in nr: nodoRed, in visitados: conj(nodoRed)) → res: conj(nodoRed)
  res ← Vacio()                                                        O(1)
  itVecinos: itConj(nodoRed) ← CrearIt(NodosVecinos(nr.conexiones))    O(n)
  while HaySiguiente?(itVecinos) do                                    O(1)
    if (¬Pertenece?(visitados , Siguiente(itVecinos))) then          O(n)
      AgregarRapido(res , Siguiente(itVecinos))                      O(1)
    end if                                                            O(n)
    Avanzar(itVecinos)                                                 O(1)
  end while                                                            O(n2)
Complejidad : O(n2)

```

```

NodosVecinos (in conex: dicc(interfaz, puntero(nodoRed))) → res: conj(nodoRed)
  res ← Vacio()                                                        O(1)
  itVecinos : itDicc( interfaz , puntero(nodoRed))) ← CrearIt (conex)    O(1)
  while HaySiguiente?(itVecinos) do                                    O(1)
    AgregarRapido(res , *SiguienteSignificado(itVecinos))            O(1)
    Avanzar(itVecinos)                                                 O(1)
  end while                                                            O(n)
Complejidad : O(n)

```

ApilarConj (in/out pnr: pila(nodoRed), in news: conj(nodoRed)) it: itConj(nodoRed) \leftarrow CrearIt(news) while HaySiguiente?(it) do Apilar(pnr, Siguiente(it)) Avanzar(it) end while Complejidad : $O(n)$	$O(1)$ $O(1)$ $O(1)$ $O(1)$ $O(n)$
--	--

iComputadoras (in r: red) \rightarrow res: conj(compu) res \leftarrow r.compus Complejidad : $O(1)$	$O(1)$
--	--------

iConectadas? (in r: red, in c ₀ : compu, in c ₁ : compu) \rightarrow res: bool nr0: nodoRed \leftarrow Significado(r.dns, c ₀ .ip) it : itDicc(interfaz, puntero(nodoRed)) \leftarrow CrearIt(nr0.conexiones) res \leftarrow false while HaySiguiente?(it) do if c ₁ .ip = SiguienteSignificado(it) \rightarrow pc \rightarrow ip then res \leftarrow true end if Avanzar(it) end while Complejidad : $O(L + n)$	$O(L)$ $O(1)$ $O(1)$ $O(1)$ $O(1)$ $O(1)$ $O(1)$ $O(n)$
---	--

iInterfazUsada (in r: red, in c ₀ : compu, in c ₁ : compu) \rightarrow res: interfaz nr0: nodoRed \leftarrow Significado(r.dns, c ₀ .ip) it : itDicc(interfaz, puntero(nodoRed)) \leftarrow CrearIt(nr0.conexiones) while HaySiguiente?(it) do if c ₁ .ip = SiguienteSignificado(it) \rightarrow pc \rightarrow ip then res \leftarrow SiguienteClave(it) end if Avanzar(it) end while Complejidad : $O(L + n)$	$O(L)$ $O(1)$ $O(1)$ $O(1)$ $O(1)$ $O(1)$ $O(1)$ $O(n)$
--	--

iVecinos (in r: red, in c: compu) \rightarrow res: conj(compu) nr: nodoRed \leftarrow Significado(r.dns, c.ip) res: conj(compu) \leftarrow Vacio() it : itDicc(interfaz, puntero(nodoRed)) \leftarrow CrearIt(nr.conexiones) while HaySiguiente?(it) do AgregarRapido(res, *(SiguienteSignificado(it) \rightarrow pc)) Avanzar(it) end while Complejidad : $O(L + n)$	$O(L)$ $O(1)$ $O(1)$ $O(1)$ $O(1)$ $O(1)$ $O(1)$ $O(n)$
--	--

iUsaInterfaz? (in r: red, in c: compu, in i: interfaz) \rightarrow res: bool nr: nodoRed \leftarrow Significado(r.dns, c.ip) res \leftarrow Definido?(pnr.conexiones, i)	$O(L)$ $O(n)$
--	------------------

Complejidad : $O(L + n)$

iCaminosMinimos (**in** r : red, **in** c_0 : compu, **in** c_1 : compu) \rightarrow res: conj(secu(compu))

nr:nodoRed \leftarrow Significado($r.dns$, $c_0.ip$)

$O(L)$

res \leftarrow Significado($pnr.caminos$, $c_1.ip$)

$O(L)$

Complejidad : $O(L)$

HayCamino? (**in** r : red, **in** c_0 : compu, **in** c_1 : compu) \rightarrow res: bool

nr:nodoRed \leftarrow Significado($r.dns$, $c_0.ip$)

$O(L)$

res \leftarrow \neg EsVacio?(Significado($pnr.caminos$, $c_1.ip$))

$O(L)$

Complejidad : $O(L)$

Copiar (**in** r : red) \rightarrow res: red

res:red \leftarrow iIniciarRed

$O(1)$

itCompus:itConj(compu) \leftarrow CrearIt($r.compus$)

$O(1)$

while HaySiguiente?(itCompus) do

$O(1)$

iAgregarComputadora(res, Siguiente(itCompus))

$O(n*L)$

Avanzar(itCompus)

$O(1)$

end while

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

itCompus CrearIt($r.compus$)

$O(1)$

while HaySiguiente?(itCompus) do

$O(1)$

nr:nodoRed \leftarrow Significado($r.dns$, Siguiente(itCompus).ip) $O(L)$ itVecinos :itDicc(interfaz, puntero(nodoRed))) \leftarrow CrearIt(conex) \ ote {1} while HaySiguiente?(itVecinos) do \ ote {1} iConectar(res, Siguiente(itCompus), *SiguienteSignificado(itVecinos)) \ ote {n!*(n^4)} end while \ ote {n!*(n^5)} Avanzar(itCompus) \ ote {1} end while \ ote {n!*(n^6)} \ ofi {O(n!*(n^6))}

$\bullet = \bullet$ (**in** r_0 : red, **in** r_1 : red) \rightarrow res: bool

res \leftarrow ($r_0.compus = r_1.compus$) \wedge ($r_0.dns = r_1.dns$)

$O(n + L^2)$

Complejidad : $O(n + L^2)$

3. Módulo Cola de mínima prioridad(α)

El módulo cola de mínima prioridad consiste en una cola de prioridad de elementos del tipo α cuya prioridad está determinada por un *nat* de forma tal que el elemento que se ingrese con el menor *nat* será el de mayor prioridad.

3.1. Especificación

TAD COLA DE MÍNIMA PRIORIDAD(α)

igualdad observacional

$$(\forall c, c' : \text{colaMinPrior}(\alpha)) \left(c =_{\text{obs}} c' \iff \left(\begin{array}{l} \text{vacía?}(c) =_{\text{obs}} \text{vacía?}(c') \wedge_L \\ (\neg \text{vacía?}(c) \Rightarrow_L (\text{próximo}(c) =_{\text{obs}} \text{próximo}(c') \wedge \\ \text{desencolar}(c) =_{\text{obs}} \text{desencolar}(c'))) \end{array} \right) \right)$$

parámetros formales

géneros α

operaciones $\bullet < \bullet$: $\alpha \times \alpha \rightarrow \text{bool}$

Relación de orden total estricto¹

géneros $\text{colaMinPrior}(\alpha)$

exporta $\text{colaMinPrior}(\alpha)$, generadores, observadores

usa **BOOL**

observadores básicos

$\text{vacía?} : \text{colaMinPrior}(\alpha) \rightarrow \text{bool}$

$\text{próximo} : \text{colaMinPrior}(\alpha) \rightarrow \alpha$ $\{\neg \text{vacía?}(c)\}$

$\text{desencolar} : \text{colaMinPrior}(\alpha) \rightarrow \text{colaMinPrior}(\alpha)$ $\{\neg \text{vacía?}(c)\}$

generadores

$\text{vacía} : \rightarrow \text{colaMinPrior}(\alpha)$

$\text{encolar} : \alpha \times \text{colaMinPrior}(\alpha) \rightarrow \text{colaMinPrior}(\alpha)$

otras operaciones

$\text{tamaño} : \text{colaMinPrior}(\alpha) \rightarrow \text{nat}$

axiomas $\forall c : \text{colaMinPrior}(\alpha), \forall e : \alpha$

$\text{vacía?}(\text{vacía}) \equiv \text{true}$

$\text{vacía?}(\text{encolar}(e, c)) \equiv \text{false}$

$\text{próximo}(\text{encolar}(e, c)) \equiv \text{if } \text{vacía?}(c) \vee_L \text{próximo}(c) > e \text{ then } e \text{ else } \text{próximo}(c) \text{ fi}$

$\text{desencolar}(\text{encolar}(e, c)) \equiv \text{if } \text{vacía?}(c) \vee_L \text{próximo}(c) > e \text{ then } c \text{ else } \text{encolar}(e, \text{desencolar}(c)) \text{ fi}$

Fin TAD

¹Una relación es un orden total estricto cuando se cumple:

Antirreflexividad: $\neg a < a$ para todo $a : \alpha$

Antisimetría: $(a < b \Rightarrow \neg b < a)$ para todo $a, b : \alpha, a \neq b$

Transitividad: $((a < b \wedge b < c) \Rightarrow a < c)$ para todo $a, b, c : \alpha$

Totalidad: $(a < b \vee b < a)$ para todo $a, b : \alpha$

3.2. Interfaz

parámetros formales

géneros α

se explica con: COLA DE MÍNIMA PRIORIDAD(NAT).

géneros: colaMinPrior(α).

3.2.1. Operaciones básicas de Cola de mínima prioridad

VACÍA() $\rightarrow res : \text{colaMinPrior}(\alpha)$

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

Post $\equiv \{res =_{\text{obs}} \text{vacía}\}$

Complejidad: O(1)

Descripción: Crea una cola de prioridad vacía

VACÍA?(in $c : \text{colaMinPrior}(\alpha)$) $\rightarrow res : \text{bool}$

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

Post $\equiv \{res =_{\text{obs}} \text{vacía?}(c)\}$

Complejidad: O(1)

Descripción: Devuelve true si y sólo si la cola está vacía

DESENCOLAR(in/out $c : \text{colaMinPrior}(\alpha)$) $\rightarrow res : \alpha$

Pre $\equiv \{\neg \text{vacía?}(c) \wedge c =_{\text{obs}} c_0\}$

Post $\equiv \{res =_{\text{obs}} \text{proximo}(c_0) \wedge c =_{\text{obs}} \text{desencolar}(c_0)\}$

Complejidad: O(log(tamaño(c)))

Descripción: Quita el elemento más prioritario

Aliasing: Se devuelve el elemento por copia

ENCOLAR(in/out $c : \text{colaMinPrior}(\alpha)$, in $p : \text{nat}$, in $a : \alpha$)

Pre $\equiv \{c =_{\text{obs}} c_0\}$

Post $\equiv \{c =_{\text{obs}} \text{encolar}(p, c_0)\}$

Complejidad: O(log(tamaño(c)))

Descripción: Agrega al elemento α con prioridad p a la cola

Aliasing: Se agrega el elemento por copia

3.3. Representación

3.3.1. Representación de colaMinPrior

colaMinPrior(α) se representa con estr

donde estr es $\text{dicc}_{\text{avl}}(\text{nat}, \text{nodoEncolados})$

donde nodoEncolados es $\text{tupla}(\text{encolados} : \text{cola}(\alpha), \text{prioridad} : \text{nat})$

3.3.2. Invariante de Representación

(I) Todos los significados del diccionario tienen como clave el valor de *prioridad*

(II) Todos los significados del diccionario no pueden tener una cola vacía

Rep $\quad \quad \quad : \text{estr} \quad \quad \quad \rightarrow \text{bool}$

$$\begin{aligned} \text{Rep}(e) &\equiv \text{true} \iff \\ &(\forall n : \text{nat}) \text{ def?}(n, e) \Rightarrow_{\text{L}} ((\text{obtener}(n, e).\text{prioridad} = n) \wedge \\ &\neg \text{vacía?}(\text{obtener}(n, e).\text{encolados})) \end{aligned}$$

3.3.3. Función de Abstracción

$$\begin{aligned} \text{Abs} &: \text{estr } e \longrightarrow \text{colaMinPrior } \{\text{Rep}(e)\} \\ \text{Abs}(e) =_{\text{obs}} \text{cmp: colaMinPrior} \mid &(\text{vacía?}(\text{cmp}) \Leftrightarrow (\# \text{claves}(e) = 0)) \wedge \\ &\neg \text{vacía?}(\text{cmp}) \Rightarrow_{\text{L}} \\ &((\text{próximo}(\text{cmp}) = \text{próximo}(\text{mínimo}(e).\text{encolados})) \wedge \\ &(\text{desencolar}(\text{cmp}) = \text{desencolar}(\text{mínimo}(e).\text{encolados}))) \end{aligned}$$

3.4. Algoritmos

iVacía () → res: colaMinPrior(α)

res ← Vacío () O(1)

Complejidad : O(1)

iVacía? (in c: colaMinPrior(α)) → res: bool

res ← (#Claves(c) = 0) O(1)

Complejidad : O(1)

iDesencolar (in/out c: colaMinPrior(α)) → res: α

res ← Copiar(Proximo(Minimo(c).encolados)) O(copy(α))
 Desencolar(Minimo(c).encolados) O(log(tamaño(c)))
 if EsVacía?(Minimo(c).encolados) then O(1)
 Borrar(c, Minimo(c).prioridad) O(log(tamaño(c)))
 end if

Complejidad : O(log(tamaño(c)) + O(copy(α)))

iEncolar (in/out c: colaMinPrior(α), in p: nat, in a: α)

if Definido?(p) then O(log(tamaño(c)))
 Encolar(Significado(c, p).encolados, a) O(log(tamaño(c)) + copy(α))
 else
 nodoEncolados nuevoNodoEncolados O(1)
 nuevoNodoEncolados.encolados ← Vacía() O(1)
 nuevoNodoEncolados.prioridad ← p O(1)
 Encolar(nuevoNodoEncolados.encolados, a) O(copy(a))
 Definir(c, p, nuevoNodoEncolados) O(log(tamaño(c)) + copy(nodoEncolados))
 end if

Complejidad : O(log(tamaño(c)) + O(copy(α)))

4. Módulo $\text{dicc}_{avl}(\alpha)$

4.1. Interfaz

se explica con: $\text{DICCIONARIO}(\text{NAT}, \alpha)$.

géneros: $\text{dicc}_{avl}(\alpha)$.

4.1.1. Operaciones básicas de $\text{dicc}_{avl}(\alpha)$

$\text{CREARDICC}() \rightarrow res : \text{dicc}_{avl}(\alpha)$

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

Post $\equiv \{res =_{\text{obs}} \text{vacío}\}$

Complejidad: $O(1)$

Descripción: Crea un diccionario vacío

$\text{DEFINIDO?}(\text{in } c : \text{nat}, \text{in } d : \text{dicc}_{avl}(\alpha)) \rightarrow res : \text{bool}$

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

Post $\equiv \{res =_{\text{obs}} \text{def?}(c, d)\}$

Complejidad: $O(\log(\#claves(d)))$

Descripción: Devuelve **true** si y sólo si la clave fue previamente definida en el diccionario

$\text{DEFINIR}(\text{in } c : \text{nat}, \text{in } s : \alpha, \text{in/out } d : \text{dicc}_{avl}(\alpha))$

Pre $\equiv \{d =_{\text{obs}} d_0\}$

Post $\equiv \{d =_{\text{obs}} \text{definir}(c, s, d_0)\}$

Complejidad: $O(\log(\#claves(d)) + \text{copy}(s))$

Descripción: Define la clave c con el significado s en d

$\text{OBTENER}(\text{in } c : \text{string}, \text{in/out } d : \text{dicc}_{avl}(\alpha)) \rightarrow res : \alpha$

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

Post $\equiv \{\text{alias}(res =_{\text{obs}} \text{obtener}(c, d))\}$

Complejidad: $O(\log(\#claves(d)))$

Descripción: Devuelve el significado correspondiente a la clave en el diccionario

Aliasing: res es modificable si y sólo si d es modificable

$\text{MÍNIMO}(\text{in/out } d : \text{dicc}_{avl}(\alpha)) \rightarrow res : \alpha$

Pre $\equiv \{\#claves(d) > 0\}$

Post $\equiv \{\text{alias}(res =_{\text{obs}} \text{obtener}(\text{claveMínima}(d), d))\}$

Complejidad: $O(\log(\#claves(d)))$

Descripción: Devuelve el significado correspondiente a la clave de mínimo valor en el diccionario

Aliasing: res es modificable si y sólo si d es modificable

4.1.2. Operaciones auxiliares del TAD

$\text{claveMínima} : \text{dicc}(\text{nat} \times \alpha) \rightarrow \text{nat} \quad \{\#claves(d) > 0\}$

$\text{darClaveMínima} : \text{dicc}(\text{nat} \times \alpha) \times \text{conj}(\text{nat}) \rightarrow \text{nat} \quad \{(\#claves(d) > 0) \wedge (c \subseteq \text{claves}(d))\}$

$\text{claveMínima}(d) \equiv \text{darClaveMínima}(d, \text{claves}(d))$

$\text{darClaveMínima}(d, c) \equiv \text{if } \emptyset?(\text{sinUno}(c)) \text{ then } \text{dameUno}(c) \text{ else } \text{min}(\text{dameUno}(c), \text{darClaveMínima}(d, \text{sinUno}(c))) \text{ fi}$

4.2. Representación

4.2.1. Representación de $\text{dicc}_{avl}(\alpha)$

$\text{dicc}_{avl}(\alpha)$ se representa con **estr**

donde **estr** es **puntero(nodoAvl)**

donde **nodoAvl** es **tupla(*data*: α , *balance*: int, *nodos*: arreglo[2] de puntero(nodoAvl))**

5. Módulo Trie(α)

5.1. Interfaz

se explica con: $\text{DICCIONARIO}(\text{STRING}, \alpha)$. **géneros:** $\text{dicc}_{\text{Trie}}(\alpha)$.

CREARDICC($\rightarrow res : \text{dicc}_{\text{Trie}}(\alpha)$)

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

Post $\equiv \{res =_{\text{obs}} \text{vacío}\}$

Complejidad: $O(1)$

Descripción: Crea un diccionario vacío.

DEFINIDO?(**in** $c : \text{string}$, **in** $d : \text{dicc}_{\text{Trie}}(\alpha)$) $\rightarrow res : \text{bool}$

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

Post $\equiv \{res =_{\text{obs}} \text{def?}(c, d)\}$

Complejidad: $O(L)$

Descripción: Devuelve true si la clave está definida en el diccionario y false en caso contrario.

DEFINIR(**in** $c : \text{string}$, **in** $s : \alpha$, **in/out** $d : \text{dicc}_{\text{Trie}}(\alpha)$)

Pre $\equiv \{d =_{\text{obs}} d_0\}$

Post $\equiv \{d =_{\text{obs}} \text{definir}(c, s, d_0)\}$

Complejidad: $O(L)$

Descripción: Define la clave c con el significado s

Aliasing: Almacena una copia de s .

OBTENER(**in** $c : \text{string}$, **in** $d : \text{dicc}_{\text{Trie}}(\alpha)$) $\rightarrow res : \alpha$

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

Post $\equiv \{\text{alias}(res =_{\text{obs}} \text{obtener}(c, d))\}$

Complejidad: $O(L)$

Descripción: Devuelve el significado correspondiente a la clave c .

Aliasing: Devuelve el significado almacenado en el diccionario, por lo que res es modificable si y sólo si d lo es.