

Algoritmos y Estructuras de Datos II

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

De los creadores de sacarCompu...

Trabajo práctico 2

Diseño - DCNet

Grupo 11

Integrante	LU	Correo electrónico
Frizzo, Franco	013/14	francofrizzo@gmail.com
Martínez, Manuela	160/14	martinez.manuela.22@gmail.com
Rabinowicz, Lucía	105/14	lu.rabinowicz@gmail.com
Weber, Andrés	923/13	herr.andyweber@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

1. Módulo Red

Notas preliminares

En todos los casos, al indicar las complejidades de los algoritmos, las variables que se utilizan corresponden a:

- n : Número de computadoras en la red.
- L : Longitud de nombre de computadora más largo de la red.
- I : Mayor cantidad de interfaces que tiene alguna computadora en la red en el momento.

Interfaz

se explica con: RED, ITERADOR UNIDIRECCIONAL(COMPU)

géneros: red, itRed

Operaciones básicas de Red

INICIARRED() $\rightarrow res : Red$

Pre $\equiv \{true\}$

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

Complejidad: $\Theta(1)$

Descripción: Genera una nueva red sin ninguna computadora.

AGREGARCOMPU(**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 $\equiv \{r =_{obs} agregarCompu(r_0, c)\}$

Complejidad: $\Theta(I)$

Descripción: Agrega una nueva pc a una red.

CONECTAR(**in/out** $r : Red$, **in** $c_0 : compu$, **in** $i_0 : interfaz$, **in** $c_1 : compu$, **in** $i_1 : interfaz$) $\rightarrow res : Red$

Pre $\equiv \{r =_{obs} r_0 \wedge c_1 \in computadoras(r) \wedge c_2 \in computadoras(r) \wedge ip(c_0) \neq ip(c_1) \wedge \neg conectadas?(r, c_0, c_1) \wedge \neg usaInterfaz?(r, c_0, i_0) \wedge \neg usaInterfaz?(r, c_1, i_1)\}$

Post $\equiv \{r =_{obs} conectar(r_0, c_0, i_0, c_1, i_1)\}$

Complejidad: $\Theta(n + I)$

Descripción: Conecta la pc c_0 con la pc c_1 a través de las interfaces i_0 y i_1 respectivamente.

COMPUTADORAS(**in** $r : Red$) $\rightarrow res : conj(compu)$

Pre $\equiv \{true\}$

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

Complejidad: $\Theta(n \times (n + I^2))$

Descripción: Devuelve todas las computadoras de la red.

CONECTADAS?(**in** $r : Red$, **in** $c_0 : compu$, **in** $c_1 : compu$) $\rightarrow res : bool$

Pre $\equiv \{c_0 \in computadoras(r) \wedge c_1 \in computadoras(r)\}$

Post $\equiv \{res =_{obs} conectadas?(r, c_0, c_1)\}$

Complejidad: $\Theta(n + I)$

Descripción: Devuelve true si y solo si la pc c_0 esta conectada a la pc c_1

INTERFAZUSADA(**in** $r : Red$, **in** $c_0 : compu$, **in** $c_1 : compu$) $\rightarrow res : interfaz$

Pre $\equiv \{c_0 \in computadoras(r) \wedge c_1 \in computadoras(r) \wedge_L conectadas?(r, c_0, c_1)\}$

Post $\equiv \{res =_{obs} interfazUsada(r, c_0, c_1)\}$

Complejidad: $\Theta(n + I)$

Descripción: Devuelve la interfaz usada por c_0 para conectarse a c_1

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: $\Theta(n + I^3)$

Descripción: Devuelve el conjunto de vecinos de la pc c

USAIINTERFAZ?(**in** $r : \text{Red}$, **in** $c : \text{compu}$, **in** $i : \text{interfaz}$) $\rightarrow res : \text{bool}$

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

Post $\equiv \{res =_{\text{obs}} \text{usaInterfaz?}(r, c, i)\}$

Complejidad: $\Theta(n + I)$

Descripción: Devuelve true si y solo si la pc c esta usando la interfaz i .

CAMINOSMÍNIMOS(**in** $r : \text{Red}$, **in** $c_0 : \text{compu}$, **in** $c_1 : \text{compu}$) $\rightarrow res : \text{conj}(\text{secu}(\text{compu}))$

Pre $\equiv \{c_0 \in \text{computadoras}(r) \wedge c_1 \in \text{computadoras}(r)\}$

Post $\equiv \{res =_{\text{obs}} \text{caminosMinimos}(r, c_0, c_1)\}$

Complejidad: $\Theta(1234)$

Descripción: Devuelve todos los caminos mínimos posibles entre c_0 y c_1 . De no haber ninguno, devuelve \emptyset .

HAYCAMINO?(**in** $r : \text{Red}$, **in** $c_0 : \text{compu}$, **in** $c_1 : \text{compu}$) $\rightarrow res : \text{bool}$

Pre $\equiv \{c_0 \in \text{computadoras}(r) \wedge c_1 \in \text{computadoras}(r)\}$

Post $\equiv \{res =_{\text{obs}} \text{hayCamino?}(r, c_0, c_1)\}$

Complejidad: $\Theta(1234)$

Descripción: Devuelve true si y solo si hay algún camino posible entre c_0 y c_1 .

CANTCOMPUS(**in** $r : \text{Red}$) $\rightarrow res : \text{nat}$

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

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

Complejidad: $\Theta(1)$

Descripción: Devuelve cuantas computadoras hay en la red

COPIAR(**in** $r : \text{Red}$) $\rightarrow res : \text{Red}$

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

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

Complejidad: $\Theta(n \times I)$

Descripción: Devuelve una copia de la red

Operaciones básicas del iterador de Red

CREARIT(**in** $r : \text{Red}$) $\rightarrow res : \text{itRed}$

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

Post $\equiv \{res =_{\text{obs}} \text{crearItUni}(r.\text{compus})\}$

Complejidad: $\Theta(1)$

Descripción: Crea un iterador de red. ????????

HAYSIGUIENTE?(**in** $it : \text{itRed}$) $\rightarrow res : \text{bool}$

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

Post $\equiv \{res =_{\text{obs}} \text{hayMas?}(it)\}$

Complejidad: $\Theta(1)$

Descripción: Devuelve true si y solo si it tiene siguiente.

SIGUIENTE(**in** $it : \text{itRed}$) $\rightarrow res : \text{compu}$

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

Post $\equiv \{res =_{\text{obs}} \text{actual}(it)\}$

Complejidad: $\Theta(I^2)???????$

Aliasing: res se devuelve por referencia

AVANZAR(**in/out** $it : \text{itRed}$)

Pre $\equiv \{it =_{\text{obs}} it_0 \wedge \text{haySiguiente?}(it)\}$

Post $\equiv \{it =_{\text{obs}} \text{avanzar}(it_0)\}$

Complejidad: $\Theta(1)$

Descripción: Avanza el iterador a la siguiente posición.

Representación

Estructura Red

Red se representa con `estrRed`

donde `estrRed` es `tupla(compus: lista(estrCompu) , cantidadCompus: nat)`

donde `estrCompu` es `tupla(IP: string , conexiones: lista(tupla(inter: interfaz, com: itLista(estrCompu))))`

`Rep : Red \rightarrow bool`

`Rep(e) \equiv ($\forall c: \text{compu}$)($c \in \text{ArmarComputadoras}(e.\text{compus}) \Rightarrow_L \neg \text{Pertenece?}(e.\text{compus}, c, c)$) \wedge
 $\# \text{ArmarComputadoras}(e.\text{compus}) = e.\text{cantidadCompus} \wedge$
 $(\forall c_1: \text{compu})(\forall c_2: \text{compu}) (c_1 \in \text{ArmarComputadoras}(e.\text{compus}) \wedge c_2 \in \text{ArmarComputadoras}(e.\text{compus})$
 $\Rightarrow_L \text{Pertenece?}(e.\text{compus}, c_1, c_2) \Leftrightarrow \text{Pertenece?}(e.\text{compus}, c_2, c_1)) \wedge$
 $(\forall c_1: \text{compu})(c_1 \in \text{ArmarComputadoras}(e.\text{compus}) \Rightarrow_L (\forall c_2: \text{compu}) (\text{Pertenece?}(e.\text{compus}, c_1, c_2) \Rightarrow c_2$
 $\in \text{ArmarComputadoras}(e.\text{compus}))) \wedge$
 $\text{sinRepetidos}(\text{ArmarSecuencia}(e.\text{compus}))$`

`Abs : estrRed e \rightarrow Red`

`{Rep(e)}`

`Abs(e) \equiv (r: Red | $\text{computadoras}(r) = \text{ArmarComputadoras}(e.\text{compus}) \wedge$
 $(\forall c_1: \text{compu})(\forall c_2: \text{compu}) \text{conectados?}(r, c_1, c_2) = \text{Pertenece?}(e.\text{compus}, c_1, c_2) \wedge$
 $\text{InterfazUsada}(r, c_1, c_2) = \text{DevolverInterfaz}(e.\text{compus}, c_1, c_2))$`

Estructura iterador de Red

itRed se representa con `itLista(estrCompu)`

`Rep : itRed \rightarrow bool`

`Rep(it) \equiv true`

`Abs : itRed itl \rightarrow itUni(estrCompu)`

`{Rep(itl)}`

`Abs(itl) \equiv itr: itUni(estrCompu) | $\text{siguientes}(itr) =_{\text{obs}} \text{armarCompus}(\text{siguiente}(itl))$`

`ArmarComputadoras : secu(tupla(string, secu(tupla(Interfaz, ItRed)))) \rightarrow conj(compu)`

`ArmarComputadoras(l) \equiv if vacia?(l) then`

`\emptyset`

`else`

`Ag($\langle \pi_1(\text{prim}(l))$, GenerarInterfaces($\pi_2(\text{prim}(l))$)`, ArmarComputadoras(fin(l)))

`fi`

`ArmarSecuencia : secu(tupla(string, secu(tupla(interfaz, itLista(compu))))) \rightarrow secu(string)`

`ArmarSecuencia(s) \equiv if vacia?(s) then $\langle \rangle$ else $(\pi_1(\text{prim}(s))) \bullet \text{ArmarSecuencia}(\text{fin}(s))$ fi`

`sinRepetidos : secu(string) \rightarrow bool`

`sinRepetidos(s) \equiv $\#(\text{pasarSecuAConj}(s) = \text{long}(s))$`

`pasarSecuAConj : secu(string) \rightarrow conj(string)`

$\text{pasarSecuAConj}(s) \equiv \text{if vacia?}(s) \text{ then } \emptyset \text{ else } \text{Ag}(\text{prim}(s), \text{pasarSecuAConj}(\text{fin}(s))) \text{ fi}$
 $\text{GenerarInterfaces} : \text{secu}(\text{tupla}(\text{Interfaz}, \text{ItLista}(\text{estrCompu}))) \longrightarrow \text{conj}(\text{Interfaz})$
 $\text{GenerarInterfaces}(l) \equiv \text{if vacia?}(l) \text{ then } \emptyset \text{ else } \text{Ag}(\pi_1(\text{prim}(l)), \text{GenerarInterfaces}(\text{fin}(l))) \text{ fi}$

$\text{Pertenece?} : \text{secu}(\text{tupla}(\text{string}, \text{secu}(\text{tupla}(\text{Interfaz}, \text{ItRed})))) \text{ } l \times \text{compu } c_1 \times \text{compu } c_2 \longrightarrow \text{bool}$
 $\text{Pertenece?}(l, c_1, c_2) \equiv \text{if } (\pi_1(\text{prim}(l)) = \pi_1(c_1)) \text{ then}$
 $\quad \pi_1(c_2) \in \text{GenerarCompus}(\pi_2(\text{prim}(l)))$
 $\quad \text{else}$
 $\quad \text{Pertenece?}(\text{fin}(l), c_1, c_2)$
 fi

$\text{GenerarCompus} : \text{secu}(\text{tupla}(\text{Interfaz} \times \text{ItLista}(\text{estrCompu}))) \longrightarrow \text{conj}(\text{string})$
 $\text{GenerarCompus}(l) \equiv \text{if vacia?}(l) \text{ then } \emptyset \text{ else } \text{Ag}(\pi_1(\text{siguiente}(\pi_2(\text{prim}(l)))), \text{GenerarCompus}(\text{fin}(l))) \text{ fi}$

$\text{DevolverInterfaz} : \text{secu}(\text{tupla}(\text{string} \times \text{secu}(\text{tupla}(\text{Interfaz} \times \text{ItRed})))) \text{ } l \times \text{compu } c_1 \times \text{compu } c_2 \longrightarrow \text{Interfaz}$
 $\quad \{\text{Pertenece?}(l, c_1, c_2)\}$

$\text{DevolverInterfaz}(l, c_1, c_2) \equiv \text{if } (\pi_1(\text{prim}(l)) = \pi_1(c_1)) \text{ then}$
 $\quad \text{DevolverInterfazAux}(\pi_2(\text{prim}(l), c_2))$
 $\quad \text{else}$
 $\quad \text{DevolverInterfaz}(\text{fin}(l, c_1, c_2))$
 fi

$\text{DevolverInterfazAux} : \text{secu}(\text{tupla}(\text{Interfaz} \times \text{ItRed})) \text{ } l \times \text{compu } c \longrightarrow \text{Interfaz}$
 $\text{DevolverInterfaz}(l, c) \equiv \text{if } (\pi_1(c_2) = \pi_1(\text{siguiente}(\pi_2(\text{prim}(l))))) \text{ then}$
 $\quad \pi_1(\text{prim}(l))$
 $\quad \text{else}$
 $\quad \text{DevolverInterfazAux}(\text{fin}(l, c))$
 fi

$\text{armarCompus} : \text{secu}(\text{estrCompu}) \text{ } l \longrightarrow \text{secu}(\text{compu})$
 $\text{armarCompus}(es) \equiv \text{if vacia}(es) \text{ then } \langle \rangle \text{ else } \text{armarCompus}(\text{prim}(es)) \bullet \text{armarCompus}(\text{fin}(es)) \text{ fi}$

$\text{armarCompu} : \text{estrCompu } e \longrightarrow \text{compu}$
 $\text{armarCompu}(e) \equiv \langle e.\text{IP}, \text{dame}\Pi_1(e.\text{conexiones}) \rangle$

$\text{dame}\Pi_1 : \text{secu}(\text{tupla}(\text{inter:interfaz} \times \text{itCompu:itLista}(\text{estrCompu}))) \text{ } l \longrightarrow \text{conj}(\text{interfaz})$
 $\text{dame}\Pi_1(l) \equiv \text{if vacia}(l) \text{ then } \emptyset \text{ else } \text{ag}(\text{Pi}_1(\text{prim}(l)), \text{dame}\Pi_1(\text{fin}(l))) \text{ fi}$

Algoritmos

Algoritmos de Red

$\text{INICIARRED}() \rightarrow res : \text{estrRed}$	
$1 \text{ } res \leftarrow \langle \langle \rangle, 0 \rangle$	$\triangleright \Theta(1)$

Complejidad: $\Theta(1)$

IAGREGARCOMPU(in/out r : estrRed, in c : compu)

```

1 agregarAtras(r.compus, ⟨c.IP, iArmarLista(c.interfaces)⟩)      ▷  $\Theta(I)$ 
2  $r.cantidadCompus \leftarrow r.cantidadCompus + 1$               ▷  $\Theta(1)$ 

```

Complejidad: $\Theta(I)$

IARMARLISTA(in c : conj(interfaz)) $\rightarrow res$: lista(⟨Interfaz, itLista(estrCompu)⟩)

```

1  $res \leftarrow vacia()$                                           ▷  $\Theta(1)$ 
2  $itConj(interfaz) \leftarrow crearIt(c)$                         ▷  $\Theta(1)$ 
3 while haySiguiente( $it$ ) do                                    ▷  $\Theta(I)$  iteraciones
4   |  $agregarAtras(res, \langle siguiente(it), NULL \rangle)$         ▷  $\Theta(1)$ 
5   |  $avanzar(it)$                                               ▷  $\Theta(1)$ 
6 end

```

Complejidad: $\Theta(I)$

Justificación: Dado un conjunto de interfaces, arma una lista, que representa las computadoras sin conexiones n tuplas de Interfaz e Iterador a NULL (ya que cuando una computadora se agrega a la red no está conectada a ninguna otra)

ICONECTAR(in/out r : estrRed), in c_1 : compu, in i_1 : interfaz, in c_2 : compu, in i_2 : interfaz)

```

1 itLista(estrCompu)  $it_1 \leftarrow crearIt(r.compus)$           ▷  $\Theta(1)$ 
2 itLista(estrCompu)  $it_2 \leftarrow crearIt(r.compus)$           ▷  $\Theta(1)$ 
3 while siguiente( $it_1$ ).IP  $\neq c_1$ .IP do                        ▷  $\Theta(n)$  iteraciones
4   |  $avanzar(it_1)$                                               ▷  $\Theta(1)$ 
5 end
6 while siguiente( $it_2$ ).IP  $\neq c_2$ .IP do                        ▷  $\Theta(n)$  iteraciones
7   |  $avanzar(it_2)$                                               ▷  $\Theta(1)$ 
8 end
9 itLista(tupla(interfaz, itLista(estrCompu)))  $it_3 \leftarrow crearIt(siguiente(it_1).conexiones)$   ▷  $\Theta(1)$ 
10 itLista(tupla(interfaz, itLista(estrCompu)))  $it_4 \leftarrow crearIt(siguiente(it_2).conexiones)$   ▷  $\Theta(1)$ 
11 while siguiente( $it_3$ ).inter  $\neq i_1$  do                      ▷  $\Theta(I)$  iteraciones
12   |  $avanzar(it_3)$                                               ▷  $\Theta(1)$ 
13 end
14 while siguiente( $it_4$ ).inter  $\neq i_2$  do                      ▷  $\Theta(I)$  iteraciones
15   |  $avanzar(it_4)$                                               ▷  $\Theta(1)$ 
16 end
17 siguiente( $it_3$ ).com  $\leftarrow it_2$                             ▷  $\Theta(1)$ 
18 siguiente( $it_4$ ).com  $\leftarrow it_1$                             ▷  $\Theta(1)$ 

```

Complejidad: $\Theta(n + I)$

ICONECTADAS?(in r : estrRed, in c_1 : compu, in c_2 : compu) $\rightarrow res$: bool

```

1 itLista(estrCompu)  $it_1 \leftarrow crearIt(r.compus)$           ▷  $\Theta(1)$ 
2 while siguiente( $it_1$ ).IP  $\neq c_1$ .IP do                        ▷  $\Theta(n)$  iteraciones
3   |  $avanzar(it_1)$                                               ▷  $\Theta(1)$ 
4 end
5 itLista(tupla(interfaz, itLista(estrCompu)))  $it_2 \leftarrow crearIt(siguiente(it_1).conexiones)$   ▷  $\Theta(1)$ 
6 while haySiguiente( $it_2$ )  $\wedge_L$  siguiente(siguiente( $it_2$ ).com)).IP  $\neq c_2$ .IP do  ▷  $\Theta(I)$  iteraciones
7   |  $avanzar(it_2)$                                               ▷  $\Theta(1)$ 
8 end
9  $res \leftarrow (siguiente(siguiente(it_2).com)).IP = c_2.IP$     ▷  $\Theta(1)$ 

```

Complejidad: $\Theta(n + I)$

INTERFAZUSADA(**in** r : **estrRed**, **in** c_1 : **compu**, **in** c_2 : **compu**) $\rightarrow res$: **interfaz**

```

1 itLista(estrCompu)  $it_1 \leftarrow$  crearIt(r.compus)  $\triangleright \Theta(1)$ 
2 while siguiente( $it_1$ ).IP  $\neq c_1$ .IP do  $\triangleright \Theta(n)$  iteraciones
3   | avanzar( $it_1$ )  $\triangleright \Theta(1)$ 
4 end
5 itLista(tupla(interfaz, itLista(estrCompu)))  $it_2 \leftarrow$  crearIt(siguiente( $it_1$ ).conexiones)  $\triangleright \Theta(1)$ 
6 while (siguiente(siguiente( $it_2$ ).com)).IP  $\neq c_2$ .IP do  $\triangleright \Theta(I)$  iteraciones
7   | avanzar( $it_1$ )  $\triangleright \Theta(1)$ 
8 end
9  $res \leftarrow$  siguiente( $it_2$ ).inter  $\triangleright \Theta(1)$ 

```

Complejidad: $\Theta(n + I)$

IVECINOS(**in** r : **estrRed**, **in** c : **compu**) $\rightarrow res$: **conj**(**compu**)

```

1  $res \leftarrow$  vacio()  $\triangleright \Theta(1)$ 
2 itLista(estrComp)  $it_1 \leftarrow$  crearIt(r.compus)  $\triangleright \Theta(1)$ 
3 while siguiente( $it_1$ ).IP  $\neq c$ .IP do  $\triangleright \Theta(n)$  iteraciones
4   | avanzar( $it_1$ )  $\triangleright \Theta(1)$ 
5 end
6 itLista(tupla(interfaz, itLista(estrCompu)))  $it_2 \leftarrow$  crearIt(siguiente( $it_1$ ).conexiones)  $\triangleright \Theta(1)$ 
7 while haySiguiente?( $it_2$ ) do  $\triangleright \Theta(n)$  iteraciones
8   | if haySiguiente?(siguiente( $it_2$ ).com) then  $\triangleright \Theta(1)$ 
9     | agregar( $res$ , (siguiente(siguiente( $it_2$ ).com).IP,
10      | crearConjunto(siguiente(siguiente( $it_2$ ).com).conexiones))))  $\triangleright \Theta(I^2)$ 
11   | end
12   | avanzar( $it_2$ )  $\triangleright \Theta(1)$ 
13 end

```

Complejidad: $\Theta(n + I^3)$

ICREARCONJUNTO(**in** l : **lista**(**tupla**(**inter** : **interfaz**, **com** : **itLista**(**estrCompu**)))) $\rightarrow res$: **conj**(**interfaz**)

```

1 nat  $n \leftarrow 0$   $\triangleright \Theta(1)$ 
2  $res \leftarrow$  vacio()  $\triangleright \Theta(1)$ 
3 while  $n < longitud(l)$  do  $\triangleright \Theta(I)$  iteraciones
4   | agregar( $res$ , ( $l[n]$ ).inter)  $\triangleright \Theta(I)$ 
5   |  $n \leftarrow n + 1$   $\triangleright \Theta(1)$ 
6 end

```

Complejidad: $\Theta(I^2)$

Justificación: Dada una lista de tupla de $\langle \text{Interfaz}, \text{Iterador} \rangle$ (que representa las conexiones de la computadora), devuelve el conjunto de todas las interfaces que se encuentran en ella.

IUSAINTERFAZ?(**in** r : **estrRed**, **in** c : **compu**, **in** i : **interfaz**) $\rightarrow res$: **bool**

```

1 itLista(estrComp)  $it_1 \leftarrow$  crearIt(r.compus)  $\triangleright \Theta(1)$ 
2 while siguiente( $it_1$ ).IP  $\neq c$ .IP do  $\triangleright \Theta(n)$  iteraciones
3
4   | avanzar( $it_1$ )  $\triangleright \Theta(1)$ 
5 end
6 itLista(tupla(interfaz, itLista(estrCompu)))  $it_2 \leftarrow$  crearIt(siguiente( $it_1$ ).conexiones)  $\triangleright \Theta(1)$ 
7 while siguiente( $it_2$ ).inter  $\neq i$  do  $\triangleright \Theta(I)$  iteraciones
8   | avanzar( $it_2$ )  $\triangleright \Theta(1)$ 
9 end
10  $res \leftarrow$  haySiguiente(siguiente( $it_2$ ).com)  $\triangleright \Theta(1)$ 

```

Complejidad: $\Theta(n + I)$

ICAMINOSMINIMOS(in r : estrRed, in c_1 : compu, in c_2 : compu) $\rightarrow res$: conj(lista(compu))

```

1  res ← vacío()                                ▷  $\Theta(1)$ 
2  if pertenece?( $c_2$ , vecinos( $r$ ,  $c_1$ )) then      ▷  $\Theta(I)$ 
3  |  agregar(res, agregarAtras(agregarAtras(<>,  $c_1$ ),  $c_2$ ))  ▷  $\Theta(n + I)$ 
4  else
5  |  res ← dameMinimos(Caminos( $r$ ,  $c_1$ ,  $c_2$ , agregarAtras(<>,  $c_1$ ), pasarConjASecu(vecinos( $r$ ,  $c_1$ ))))
6  end

```

Complejidad:

DAMEMINIMOS(in c : conj(lista(compu))) $\rightarrow res$: conj(lista(compu))

```

1  if esVacio?( $c$ ) then                                ▷  $\Theta(1)$ 
2  |  res ← vacío()                                    ▷  $\Theta(1)$ 
3  else
4  |  ItConj(lista(compu))  $it$  ← crearIt( $c$ )          ▷  $\Theta(1)$ 
5  |  res ← dameMinimosAux( $c$ , minimaLong( $c$ , long(siguiente( $it$ ))))
6  end

```

Complejidad:

Justificación: Devuelve, del total de caminos posibles, solo los de longitud mínima

DAMEMINIMOSAUX(in c : conj(lista(compu)), in n : nat) $\rightarrow res$: conj(lista(compu))

```

1  ItConj(lista(compu))  $it$  ← crearIt( $c$ )                                ▷  $\Theta(1)$ 
2  res ← vacío()
3  while haySiguiente( $it$ ) do                                           ▷  $\Theta(1)$ 
4  |  if long(siguiente( $it$ )) =  $n$  then                                  ▷  $\Theta(1)$ 
5  |  |  agregar(res, siguiente( $it$ ))                                  ▷  $\Theta(n)$ 
6  |  |  avanzar( $it$ )                                                  ▷  $\Theta(1)$ 
7  |  else
8  |  |  avanzar( $it$ )                                                  ▷  $\Theta(1)$ 
9  |  end
10 end

```

Complejidad:

▷ $\Theta(n \times n!)$

MINIMALONG(in c : conj(lista(compu)), in n : nat) $\rightarrow res$: nat

```

2  nat  $i$  ←  $n$                                 ▷  $\Theta(1)$ 
3  ItConj(lista(compu))  $it$  ← crearIt( $c$ )          ▷  $\Theta(1)$ 
4  while haySiguiente( $it$ ) do
5  |  if long(siguiente( $it$ )) then
6  |  |   $i$  ← longitud(siguiente( $it$ ))
7  |  |  avanzar( $it$ )
8  |  else
9  |  |  avanzar( $it$ )
10 |  end
11 end
12 res ←  $i$ 

```

Complejidad:

Justificación: Devuelve la longitud de la secuencia más chica

	PASARCONJASECU(in c : conj(compu)) $\rightarrow res$: secu(compu)	
1	2 $res \leftarrow vacia()$	$\triangleright \Theta(1)$
	3 ItConj $it \leftarrow crearIt(c)$	$\triangleright \Theta(1)$
	4 While($\triangleright \Theta(n)$ iteraciones haySiguiente(it) agregarAtras(res , siguiente(it))	$\triangleright \Theta(I)$

5 **Complejidad:** $\Theta(n \times I)$

6 **Justificación:** Devuelve una secuencia que contiene a todos los elementos del conjunto pasado por parámetro

	IHAYCAMINO?(in r : estrRed, in c_1 : compu, in c_2 : compu) $\rightarrow res$: bool	
1	2 $res \leftarrow (\neg esVacio?(iCaminosMinimos(r, c_1, c_2)))$	

3 **Complejidad:**

	ICAMINOS(in r : estrRed, in c_1 : compu, in c_2 : compu, in l : lista(estrCompu), in vec : lista(estrCompu)) $\rightarrow res$: conj(lista(estrCompu))	
	2 if vacia?(vec) then	
	3 $res \leftarrow vacia()$	
	4 else	
	5 if /último(l) = c_1 then	
	6 $res \leftarrow agregar(l, vacia())$	
	7 else	
1	8 if está?(primero(vec , l)) then	
	9 $res \leftarrow unión(caminos(r, c_0, c_1, agregarAtras(l, primero(vec)), iVecinos(r, primeros(vec))),$ $caminos(r, c_0, c_1, l, fin(vec)))$	
	10 else	
	11 $res \leftarrow caminos(r, c_0, c_1, l, fin(vec))$	
	12 end	
	13 end	
	14 end	

15 **Complejidad:**

16 **Justificación:** Dada una red, dos compus, los vecinos de la primer compu, y una lista que usamos para guardar las computadoras por las que ya preguntamos, iteramos sobre todas las computadoras y devolvemos el conjunto de todos los caminos posibles desde la primer computadora hasta la segunda.

	IUNIÓN(in c_1 : conj(lista(compu)), in c_2 : conj(lista(compu))) $\rightarrow res$: conj(lista(compu))	
	2 $res \leftarrow vacio()$	$\triangleright \Theta(1)$
	3 if vacio?(c_1) then	$\triangleright \Theta(1)$
	4 $res \leftarrow c_2$	
	5 else	
1	6 itConj(lista(compu)) $it \leftarrow crearIt(c_1)$	$\triangleright \Theta(1)$
	7 while haySiguiente(it) do $\triangleright [$	
	8 f]* $\Theta(n)$	
	9 Ag(siguiente(it), c_2)	$\triangleright \Theta(n)$
	10 avanzar(it)	$\triangleright \Theta(1)$
	11 end	
	12 end	

13 **Complejidad:**

14 **Justificación:** Devuelve la unión de dos conjuntos.

	$\text{IESTA?}(\text{in } c : \text{compu}, \text{in } l : \text{lista}(\text{compu})) \rightarrow res : \text{bool}$	
	2 if vacia?(l) then	$\triangleright \Theta(1)$
	3	
	4 $res \leftarrow \text{false}$	$\triangleright \Theta(1)$
	5 else	
1	6 $\text{ItLista}(\text{compu}) \text{ it} \leftarrow \text{crearIt}(l)$	$\triangleright \Theta(1)$
	7 while haySiguiente(it) \wedge_L siguiente(it) $\neq c$ do	$\triangleright \Theta(n)$
	8	
	9 $\text{avanzar}(\text{it})$	$\triangleright \Theta(1)$
	10 end	
	11 end	
	12 $res \leftarrow (\text{haySiguiente}(\text{it}))$	$\triangleright \Theta(1)$

13 **Complejidad:** $\Theta(n)$

14 **Justificación:** Devuelve True si y solo si la compu c se encuentra en la lista l.

	$\text{ICOMPUTADORAS}(\text{in } r : \text{estrRed}) \rightarrow res : \text{conj}(\text{compu})$	
	2 $res \leftarrow \text{vacío}()$	$\triangleright \Theta(1)$
	3 $\text{itRed } it \leftarrow \text{crearItRed}()$	$\triangleright \Theta(1)$
1	4 while haySiguiente?(it) do	$\triangleright \Theta(n)$ iteraciones
	5 $\text{agregar}(res, \text{siguiente}(it))$	$\triangleright \Theta(n + I^2)$
	6 $\text{avanzar}(it)$	$\triangleright \Theta(1)$
	7 end	

8 **Complejidad:** $\Theta(n \times (n + I^2))$

	$\text{ICOPIAR}(\text{in } r : : \text{estrRed}) \rightarrow res : \text{Red}$	
1	2 $res \leftarrow \langle \text{copiar}(r.\text{compus}, r.\text{cantidadCompus}) \rangle$	$\triangleright \Theta(n \times I)$

3 **Complejidad:** $\Theta(n \times I)$

4 **Justificación:** Devuelve una copia de la Red

	$\text{ICANTCOMPUS}(\text{in } r : \text{Red}) \rightarrow res : \text{nat}$	
1	2 $res \leftarrow r.\text{cantCompus}$	$\triangleright \Theta(1)$

3 **Complejidad:** $\Theta(1)$

4 Algoritmos del iterador de Red

	$\text{IHAYSIGUIENTE?}(\text{in } it : \text{itLista}(\text{estrCompu})) \rightarrow res : \text{bool}$	
1	2 $res \leftarrow \text{haySiguiente?}(it)$	$\triangleright \Theta(1)$

3 **Complejidad:** $\Theta(1)$

	<hr/>	
	ISIGUIENTE(in <i>it</i> : itLista(estrCompu)) \rightarrow <i>res</i> : compu	
	<hr/>	
	2 estrCompu <i>e</i> \leftarrow siguiente(<i>it</i>)	$\triangleright \Theta(1)$
	3 <i>res</i> .IP \leftarrow <i>e</i> .IP	$\triangleright \Theta(1)$
	4 conj(interfaz) <i>interfaces</i> \leftarrow vacío()	$\triangleright \Theta(1)$
	5 itLista(tupla(<i>inter</i> : interfaz, <i>com</i> : itLista(estrCompu))) <i>itInterfaces</i> \leftarrow crearIt(<i>e</i> .conexiones)	$\triangleright \Theta(1)$
1	$\triangleright \Theta(1)$	
	6 while haySiguiente?(<i>itInterfaces</i>) do	$\triangleright \Theta(I)$ iteraciones
	7 agregar(<i>interfaces</i> , siguiente(<i>itInterfaces</i>).inter)	$\triangleright \Theta(I)$
	8 avanzar(<i>itInterfaces</i>)	$\triangleright \Theta(1)$
	9 end	
	10 <i>res</i> .Interfaces \leftarrow <i>e</i> .Interfaces	$\triangleright \Theta(I)$
	<hr/>	
11	Complejidad: $\Theta(I^2)$	
	<hr/>	
	ICREARIT(in <i>e</i> : estrRed)	
1	2 <i>res</i> \leftarrow crearIt(<i>e</i> .compus)	$\triangleright \Theta(1)$
	<hr/>	
	3 Complejidad: $\Theta(1)$	
	<hr/>	
	IAVANZAR(in/out <i>it</i> : itLista(estrCompu))	
1	2 <i>it</i> \leftarrow avanzar(<i>it</i>)	$\triangleright \Theta(1)$
	<hr/>	
	3 Complejidad: $\Theta(1)$	
	<hr/>	

2. Módulo DCNet

Notas preliminares

En todos los casos, al indicar las complejidades de los algoritmos, las variables que se utilizan corresponden a:

- n : Número de computadoras en la red.
- k : Longitud de la cola de paquetes más larga al momento.
- L : Longitud de nombre de computadora más largo de la red.
- i : Mayor cantidad de interfaces que tiene alguna computadora en la red en el momento.

Interfaz

se explica con: DCNET

géneros: dcnet

Operaciones básicas de lista

INICIARDCNET(**in** r : Red) $\rightarrow res$: DCNet

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

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

Complejidad: $\Theta(1234)$

Descripción: Genera un nuevo DCNet sin paquetes.

CREARPAQUETE(**in/out** D : DCNet, **in** p : paquete)

Pre $\equiv \{D =_{\text{obs}} D_0 \wedge ((\exists p': \text{paquete})(\text{paqueteEnTransito?}(D, p') \wedge \text{id}(p') = \text{id}(p)) \wedge \text{origen}(p) \in \text{computadoras}(\text{red}(D)) \wedge_L \text{destino}(p) \in \text{computadoras}(\text{red}(D)) \wedge_L \text{hayCamino?}(\text{red}(D), \text{origen}(p), \text{destino}(p)))\}$

Post $\equiv \{D =_{\text{obs}} \text{crearPaquete}(D_0, p)\}$

Complejidad: $\Theta(l + \log(k))$

Descripción: Crea un nuevo paquete que no existe en el DCNet anterior.

AVANZARSEGUNDO(**in/out** D : DCNet)

Pre $\equiv \{D =_{\text{obs}} D_0\}$

Post $\equiv \{D =_{\text{obs}} \text{avanzarSegundo}(D_0)\}$

Complejidad: $\Theta(n \cdot (L + \log(n) + \log(k)))$

Descripción: Avanza un segundo en el DCNet, moviendo todos los paquetes correspondientes.

RED(**in** D : DCNet) $\rightarrow res$: red

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

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

Complejidad: $\Theta(1234)$

Descripción: Devuelve la red donde esta funcionando el DCNet.

CAMINORECORIDO(**in** D : DCNet, **in** p : paquete) $\rightarrow res$: secu(compu)

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

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

Complejidad: $\Theta(n \cdot \log(\max(n, k)))$

Descripción: Devuelve la secuencia que contiene de forma ordenada todas las computadoras por las que fue pasando.

CANTIDADENVIADOS(**in** D : DCNet, **in** c : compu) $\rightarrow res$: nat

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

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

Complejidad: $\Theta(1234)$

Descripción: Devuelve la cantidad de paquetes que envió la computadora “c”.

ENESPERA(**in** D : DCNet, **in** c : compu) $\rightarrow res$: conj(paquete)

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

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

Complejidad: $\Theta(L)$

Descripción: Devuelve los paquetes que tiene en espera la compu “c”.

`PAQUETESENTRÁNSITO?(in D: DCNet, in p: paquete) → res : bool`

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

Post $\equiv \{\text{res} =_{\text{obs}} \text{paquetesEnTrancito?}(D, p)\}$

Complejidad: $\Theta(1234)$

Descripción: Devuelve “True” si y solo si el paquete esta en los paquetes en espera de alguna computadora.

`LAQUEMÁSENVIÓ(in D: DCNet) → res : compu`

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

Post $\equiv \{\text{res} =_{\text{obs}} \text{laQueMásEnvió}(D)\}$

Complejidad: $\Theta(1)$

Descripción: Devuelve una de las computadoras con mas paquetes enviados

Representación

dcnet se representa con `estrDCNet`

donde `estrDCNet` es `tupla(red: red , IDsCompusPorIP: dicc_trie(string, nat) , siguientesCompus: ad(ad(nat)) , paquetesEnEspera: ad(tupla(enConjunto: conj(paquete), porID: dicc_AVL(ID, tupla(iPaquete: itConj(paquete), codOrigen: nat, codDestino: nat), porPrioridad: heap(tupla(prioridad, itConjunto(paquete)))))) , #PaqEnviados: ad(nat) , laQueMásEnvió: itRed , IPsCompuPorID: ad(itRed))`

`Rep : e → bool`

$$\begin{aligned}
\text{Rep}(l) \equiv & \text{true} \iff (\forall e: \text{estr}) (\\
& (\text{tam}(\text{e.paquetesEnEspera}) = \text{tam}(\text{e.IPcompusXID}) = \text{tam}(\text{e.siguienteCompu}) = \text{tam} \\
& (\text{e.cantPaquetesEnviados}) = \#(\text{computadoras}(\text{e.estrRed})) \wedge (\forall n: \text{nat})(\text{definido?}(\text{e.siguienteCompu}, n) \Rightarrow_L \\
& \text{tam}(\text{e.siguienteCompu}[n]) = \#(\text{computadoras}(\text{e.estrRed}))) \wedge \\
& \text{Maximo}(\text{e.cantPaqEnviados}) = \text{e.cantidadEnviados}[\text{obtener}(\pi_1(\text{siguiente}(\text{e.laQueMásEnvió})), \text{e.IDcompusPorIP})] \\
& \wedge (\forall c: \text{compu})(c \in \text{computadoras}(\text{e.estrRed}) \Rightarrow_L \text{obtener}(\pi_1(c), \text{e.IDcompusPorIP}) < \#(\text{computado-} \\
& \text{ras}(\text{e.estrRed}))) \wedge ((\forall c_1, c_2: \text{compu})((c_1 \in \text{computadoras}(\text{e.estrRed}) \wedge (c_2 \in \text{computadoras}(\text{e.estrRed})) \wedge \\
& (c_1 \neq c_2)) \Rightarrow_L ((\text{obtener}(\pi_1(c_1), \text{e.IDcompusPorIP}) \neq \text{obtener}(\pi_1(c_2), \text{e.IDcompusPorIP})))) \wedge \\
& (\text{dameNombres}(\text{computadoras}(\text{e.estrRed})) = \text{claus}(\text{IDcompusPorIP})) \wedge \\
& (\forall L: \text{nat})(0 \leq L < \text{tam}(\text{e.paqEnEspera}) \Rightarrow_L (\\
& (\forall it_1: \text{ItConj}(\text{paquete})) it_1 \in \text{dame}\pi_1(\text{juntarSignificado}(\pi_2(\text{e.paquetesEnEspera}[L]))) \Rightarrow_L \text{haySiguiente?}(it_1) \\
& \wedge \\
& (\forall it_2: \text{ITconj}(\text{pauquete})) it_2 \in \text{dame}\pi_2(\text{juntarColaEnConjunto}(\pi_3(\text{e.paquetesEnEspera}[L]))) \Rightarrow_L \\
& \text{haySiguiente?}(it_2) \wedge_L \\
& (\forall i: \text{ItConj}(\text{paquete}))(\text{siguiente}(i) \in (\text{dameSiguientes}(\text{dame}\pi_1(\text{juntarSignificados}_1(\pi_2(\text{e.paquetesEnEspera}[L]))) \\
& \text{siguiente}(i) \in \pi_1(\text{e.paquetesEnEspera}[L]))) \wedge \\
& (\forall c: \text{ID})(c \in \text{claves}(\pi_2(\text{e.paquetesEnEspera}[L])) \Rightarrow_L \pi_1(\text{siguiente}(\pi_1(\text{obtener}(\pi_2(c), \\
& \text{e.paquetesEnEspera}[L]))) = c) \wedge \\
& (\forall it: \text{ItConj}(\text{paquete})) \text{siguiente}(it) \in \text{dameSiguientes}(\text{dame}\pi_2(\text{juntarColaPrioriEnConj}(\pi_3(\text{e.paquetesEnEspera}[L])) \\
& \Rightarrow_L \text{siguiente}(it) \in \pi_1(\text{e.paquetesEnEspera}[L])) \\
& (\forall t: \text{tupla} < \text{prioridad}, \text{ItConj}(\text{paquete}) >) t \in \text{juntarColaPrioriEnConj}(\pi_3(\text{e.paquetesEnEspera}[L])) \rightarrow \\
& \text{siguiente}(\pi_2(t)).\text{prioridad} = \pi_1(t) \\
& (\forall x, z: \text{nat})((0 \leq x < \text{tam}(\text{e.paquetesEnEspera}) \wedge 0 \leq z < \text{tam}(\text{e.paquetesEnEspera}) \wedge x \neq z) \Rightarrow_L \\
& (\pi_1(\text{e.paquetesEnEspera}[x]) \cap \pi_2(\text{e.paquetesEnEspera}[z])) = \emptyset) \wedge \\
& (\forall i: \text{nat})(0 \leq i < \#(\text{computadoras}(\text{e.estrRed})) \wedge \text{obtener}(\pi_1(\text{siguiente}(\text{e.IPcompusPorID}[i])), \\
& \text{e.IDcompusPorID}) = i) \wedge \\
& (\forall n, m: \text{nat})(0 \leq n < \#(\text{computadoras}(\text{e.estrRed})) \wedge 0 \leq m < \#(\text{computadoras}(\text{e.estrRed})) \Rightarrow_L \\
& (\exists x: (\text{secu}(\text{compu})) x \in \text{caminosminimos}(\text{e.estrRed}, \text{siguiente}(\text{e.IPcompusPorID}[n]), \text{siguien-} \\
& \text{te}(\text{e.IPcompusPorID}[m])) \wedge \text{prim}(x) = \text{e.siguienteCompu}[n][m]) \\
& (\forall i: \text{nat})(0 \leq i \leq \#(\text{computadoras}(\text{e.estrRed})) \Rightarrow_L \text{siguiente}(\text{e.IPcompusPorID}[i]) \in \text{compitadoras}(\text{e.estrRed})) \\
& \wedge \\
& (\forall x, y: \text{nat})((0 \leq x < \#(\text{computadoras}(\text{e.estrRed})) \wedge 0 \leq y < \#(\text{computadoras}(\text{e.estrRed})) \wedge x \neq y) \Rightarrow_L \\
& (\text{siguiente}(\text{e.IPcompusPorID}[x]) \neq \text{siguiente}(\text{e.IPcompusPorID}[y]))) \wedge \\
& \#(\pi_1(\text{e.paquetesEnEspera})) = \#(\text{claves}(\pi_2(\text{e.paquetesEnEspera}))) \wedge = \#(\text{juntarSecuenciasEnConj}(\text{juntarSignificad} \\
& \wedge \\
& (\forall it_1: \text{ItConj}(\text{paquete})) (\forall it_2: \text{ItConj}(\text{paquete})) it_1 \in \text{dame}\pi_2(\text{juntarColaPrioriEnConj}(\pi_3(\text{e.paquetesEnEspera}[L])) \\
& \wedge it_2 \in \text{dame}\pi_2(\text{juntarColaPrioriEnConj}(\pi_3(\text{e.paquetesEnEspera}[L]))) it_1 \neq it_2 \Rightarrow_L \text{siguiente}(it_1) \neq \\
& \text{siguiente}(it_2)
\end{aligned}$$

$$\text{Abs} : \text{estrDCNet } e \longrightarrow \text{DCNet}$$

$$\{\text{Rep}(e)\}$$

$Abs(e) \equiv red(d) = e.estrRed \wedge$

$(\forall p: paquete) \quad paqueteEnTránsito?(d, p) \Rightarrow_L caminoRecorrido(d, p) = caminoDelPaquete(e.siguienteCompu, e.IPsCompusPorID, e.PaquetesEnEspera, p) \wedge$

$(\forall c: compu) \quad c \in computadoras(red(d)) \Rightarrow_L cantidadEnviados(d, c) = e.\#PaqEnviados[obtener(c, e.IDsCompusPorIP)] \wedge$

$(\forall c: compu) \quad c \in computadoras(red(d)) \Rightarrow_L enEspera(d, c) = enConjunto(e.paquetesEnEspera[obtener(c, e.IDsCompusPorIP)])$

$caminoDelPaquete : ad(ad(nat)) \times ad(ItRed) \times ad(tupla(enConjunto:conj(paquete) \times porID:dicc_{AVL}(ID \text{ tupla}(iPaquete:itConj(paquete) \times codOrigen:nat \times codDestino:nat)))$

$caminoDelPaquete(t, CsxID, ps, p, Psr) \equiv \text{if } def?(ID(p), porID(ps[0])) \text{ then}$
 $caminoDelPaquete_{aux}(t, CsxID, ps, p, codOrigen(obtener(ID(p), porId(ps[0])), codDestino(obtener(ID(p), porId(ps[0]))))$
 else
 $caminoDelPaquete(t, CsxID, ps, p, finAd(psr))$
fi

$caminoDelPaquete_{aux} : ad(ad(nat)) \times ad(ItRed) \times ad(tupla(enConjunto:conj(paquete) \times porID:dicc_{AVL}(ID \text{ tupla}(iPaquete:itConj(paquete) \times codOrigen:nat \times codDestino:nat)))$

$caminoDelPaquete_{aux}(t, CsxID, ps, p, Psr, compuActual, d) \equiv \text{if } def?(ID(p), porID(ps[compuActual])) \text{ then}$
 $siguiente(CsxID[compuActual]) \bullet <>$
 else
 $siguiente(CsxID[compuActual]) \bullet$
 $caminoDelPaquete_{aux}(t, CsxID, ps, p, Psr, t[compuActual][d], d)$
fi

$finAd : ad(tupla(enConjunto:conj(paquete) \times porID:dicc_{AVL}(ID \text{ tupla}(iPaquete:itConj(paquete) \times codOrigen:nat \times codDestino:nat)))$

$finAd(a) \equiv \text{if } (tam(a) \leq 1) \text{ then } crearArreglo(0) \text{ else } finAd_{aux}(a, crearArreglo(tam(a)-1), tam(a)-2) \text{ fi}$

$finAd_{aux} : ad(tupla(enConjunto:conj(paquete) \times porID:dicc_{AVL}(ID \text{ tupla}(iPaquete:itConj(paquete) \times codOrigen:nat \times codDestino:nat)))$
 $\{tam(a) > 1 \wedge tam(b) > 0\}$

$finAd_{aux}(a, b, n) \equiv \text{if } (n=0) \text{ then } b[0] \leftarrow a[1] \text{ else } finAd_{aux}(a, b[n] \leftarrow a[n+1], n-1) \text{ fi}$

$juntarColaPriorEnConj : colaPrior(tupla(<prioridad, ItConj(paquete)>)) \longrightarrow conj(tupla(<prioridad, ItConj(paquete)>))$

$juntarColaPriorEnConj(c) \equiv \text{if } vacia?(c) \text{ then } \emptyset \text{ else } ag(proximo(c), juntarColaPriorEnConj(desencolar(c))) \text{ fi}$

$dame\pi_2 : conj(tupla(<prioridad \times ItConj(paquete)>)) \longrightarrow conj(ItConj(paquete))$

$dame\pi_2(c) \equiv \text{if } \emptyset?(c) \text{ then } \emptyset \text{ else } ag(\pi_2(dameuno(c)), dame\pi_2(sinuno(c))) \text{ fi}$

$dameSiguientes : conj(ItConj(paquete)) \longrightarrow conj(paquete) \quad \{restricciones\}$

dameSiguientes(c) \equiv **if** $\emptyset?(c)$ **then** \emptyset **else** ag(siguiente(dameuno(c)), dameSiguientes(sinuno(c))) **fi**

Algoritmos

INICIARDCNET(**in** $r : \text{Red}$) $\rightarrow res : \text{DCNet}$

```

1   $res \leftarrow \text{iCopiar}(r)$   $\triangleright \Theta(n \times I)$ 
2   $res.\#PaqEnviados \leftarrow \text{crearArreglo}(\text{cantCompus}(res.\text{red}))$   $\triangleright \Theta(n)$ 
3   $res.\text{IPsCompuPorID} \leftarrow \text{crearArreglo}(\text{cantCompus}(res.\text{red}))$   $\triangleright \Theta(n)$ 
4   $res.\text{siguientesCompus} \leftarrow \text{crearArreglo}(\text{cantCompus}(res.\text{red}))$   $\triangleright \Theta(n)$ 
5   $res.\text{paquetesEnEspera} \leftarrow \text{crearArreglo}(\text{cantCompus}(res.\text{red}))$   $\triangleright \Theta(n)$ 
6   $itRed \leftarrow \text{crearItRed}(res.\text{red})$   $\triangleright \Theta(1)$ 
7   $\text{nat } j \leftarrow 0$   $\triangleright \Theta(1)$ 
8  while  $j < \text{Cardinal}(\text{Computadoras}(res.\text{red}))$  do  $\triangleright \Theta(n)$ 
9       $res.\text{siguientesCompus}[j] \leftarrow \text{crearArreglo}(\text{cantCompus}(res.\text{red}))$   $\triangleright \Theta(n)$ 
10      $res.\#PaqEnviados[j] \leftarrow 0$   $\triangleright \Theta(1)$ 
11      $res.\text{paquetesEnEspera}[j] \leftarrow \langle \text{vacío}(), \text{vacío}(), \text{vacío}() \rangle$   $\triangleright \Theta(1)$ 
12      $\text{definir}(\text{siguiente}(it_0).\text{IP}, j, res.\text{IDsCompusPorIP})$   $\triangleright \Theta(L + I^2)$ 
13      $res.\text{IPsCompusPorID}[j] \leftarrow it_1$   $\triangleright \Theta(1)$ 
14      $j \leftarrow j + 1$   $\triangleright \Theta(1)$ 
15      $\text{avanzar}(it_1)$   $\triangleright \Theta(1)$ 
16 end
17  $\text{nat } k \leftarrow 0$   $\triangleright \Theta(1)$ 
18  $j \leftarrow 0$   $\triangleright \Theta(1)$ 
19 while  $j < \text{Cardinal}(\text{Computadoras}(res.\text{red}))$  do  $\triangleright \Theta(n)$ 
20     while  $k < \text{Cardinal}(\text{Computadoras}(res.\text{red}))$  do  $\triangleright \Theta(n)$ 
21         if  $\text{conectadas?}(res.\text{red}, \text{siguiente}(res.\text{IPsCompusPorID}[j]), \text{siguiente}(res.\text{IPsCompusPorID}[k]))$  then
22              $\triangleright \Theta(n + I)$ 
23              $itConj \leftarrow \text{crearIt}(\text{caminosMinimos}(res.\text{red}, \text{siguiente}(res.\text{IPsCompusPorID}[j]),$ 
24                  $\text{siguiente}(res.\text{IPsCompusPorID}[k])))$ 
25              $res.\text{siguientesCompus}[j][k] \leftarrow \text{prim}(\text{fin}(\text{siguiente}(it_1)))$   $\triangleright \Theta(1)$ 
26         end
27          $k \leftarrow k + 1$   $\triangleright \Theta(1)$ 
28     end
29      $j \leftarrow j + 1$   $\triangleright \Theta(1)$ 
30 end

```

Complejidad: $\triangleright \Theta()$

ICREARPAQUETE(**in/out** $d : \text{DCNet}$, **in** $p : \text{paquete}$)

```

2   $\text{nat } o \leftarrow \text{iObtener}(p.\text{origen}, d.\text{IDsCompusPorIP})$ 
3   $\text{nat } dest \leftarrow \text{iObtener}(p.\text{destino}, d.\text{IDsCompusPorIP})$ 
4   $it \leftarrow \text{CrearIt}((d.\text{paquetesEnEspera}[o]).\text{enConjunto})$ 
5   $it \leftarrow \text{iAgregar}((d.\text{paquetesEnEspera}[o]).\text{enConjunto}, p)$ 
6   $\text{iDefinir}(d.\text{paquetesEnEspera}[o].\text{porID}, p.\text{ID}, \langle it, o, dest, \rangle)$ 
7   $\text{iAgregarHeap}(d.\text{paquetesEnEspera}[o].\text{porPrioridad}, p.\text{prioridad}, it)$ 

```

IAVANZARSEGUNDO(in/out d : DCNet)

```

1  2 nat  $j \leftarrow 0$ 
    3 nat  $o$ 
    4 nat  $dest$ 
    5 paquete  $paq$ 
    6 while  $j < iCardinal(iComputadoras(d.red))$  do
    7   if !(iEsVacio?(( $d.paquetesEnEspera[j]$ ).enConjunto) then
    8      $paq \leftarrow iSiguiente(iDameElDeMayorPrioridad((d.paquetesEnEspera[j]).porPrioridad))$ 
    9      $iBorrarElDeMaxPrioridad(d.paquetesEnEspera[j].porPrioridad)$ 
   10     $o \leftarrow (iObtener((d.paquetesEnEspera[j]).porID, paq.ID)).codOrigen$ 
   11     $dest \leftarrow (iObtener((d.paquetesEnEspera[j]).porID, paq.ID)).codDestino$ 
   12     $iBorrar((d.paquetesEnEspera[j]).porID, paq.ID)$ 
   13     $d.\#paqEnviados[j]++$ 
   14    if !( $d.siguienteCompu[j][dest] = dest$ ) then
   15       $it \leftarrow crearIt(d.paquetesEnEspera[d.siguienteCompu[j][dest]])$ 
   16       $it \leftarrow iAgregar((d.paquetesEnEspera[d.siguienteCompu[j][dest]].enConjunto, p)$ 
1  17       $iDefinir(d.paquetesEnEspera[d.siguienteCompu[j][dest]].porID, p.ID, \langle it,$ 
       $d.siguienteCompu[j][dest], dest, \rangle)$ 
   18       $iAgregarHeap(d.paquetesEnEspera[d.siguienteCompu[j][dest]].porPrioridad, paq.prioridad, it)$ 
   19    end
   20  end
   21 end
   22 nat  $k \leftarrow 0$ 
   23 nat  $h \leftarrow 0$ 
   24 if  $iCardinal(iComputadoras(d.red)) > 0$  then
   25   while  $k < iCardinal(iComputadoras(d.red)) > 0$  do
   26     if  $d.\#paqEnviados[k] > d.\#paqEnviados[h]$  then
   27        $h \leftarrow k$ 
   28        $k++$ 
   29     end
   30   end
   31 end
   32  $d.laQueMÁSEnvío \leftarrow d.IPsCompusPorID[h]$ 

```

ICAMINORECORRIDO(in d : DCNet, in p : paquete) $\rightarrow res$: lista(compu)

```

1  2 nat  $j \leftarrow 0$ 
    3  $res \leftarrow iVacia()$ 
    4 while !(iDefinido?(( $d.paquetesEnEspera[j]$ ).porID),  $p.ID$ ) do
    5    $j++$ 
    6 end
    7 nat  $o$ 
1  8  $o \leftarrow (iObtener((d.paquetesEnEspera[j]).porID, p.ID)).codOrigen$ 
    9 nat  $dest$ 
   10  $dest \leftarrow (iObtener((d.paquetesEnEspera[o]).porID, p.ID)).codDestino$ 
   11 while !(iDefinido?(( $d.paquetesEnEspera[o]$ ).porID),  $p.ID$ ) do
   12    $iAgregarAtras(res, siguiente(d.IPsCompusPorID[o]))$ 
   13    $o \leftarrow d.siguientesCompus[o][dest]$ 
   14 end
   15  $iAgregarAtras(res, siguiente(d.IPsCompusPorID[o]))$ 

```

ICANTIDADENVIADOS(in d : DCNet, in c : compu) $\rightarrow res$: nat

```

1  2 nat  $i \leftarrow iObtener(d.IDsCompusPorIP, c.IP)$ 
    3  $res \leftarrow d.\#paqEnviados[i]$ 

```

IENESPERA(**in** d : DCNet, **in** c : compu) $\rightarrow res$: conj(Paquete)

¹ **2** nat $i \leftarrow$ iObtener(d .IDsCompusPorIP, c .IP)
 3 $res \leftarrow (d.paquetesEnEspera[i]).enConjunto$

3. Módulo Diccionario AVL

Notas preliminares

En todos los casos, al indicar las complejidades de los algoritmos, las variables que se utilizan corresponden a:

- n : Cantidad de claves definidas en el diccionario.

Interfaz

parámetros formales

géneros	κ, σ
función	$\bullet = \bullet(\text{in } k_1 : \kappa, \text{in } k_2 : \kappa) \rightarrow res : \text{bool}$ $\text{Pre} \equiv \{\text{true}\}$ $\text{Post} \equiv \{res =_{\text{obs}} (k_1 = k_2)\}$ Complejidad: $\Theta(\text{equal}(k_1, k_2))$ Descripción: función de igualdad de κ
función	$\bullet \leq \bullet(\text{in } k_1 : \kappa, \text{in } k_2 : \kappa) \rightarrow res : \text{bool}$ $\text{Pre} \equiv \{\text{true}\}$ $\text{Post} \equiv \{res =_{\text{obs}} (k_1 \leq k_2)\}$ Complejidad: $\Theta(\text{order}(k_1, k_2))$ Descripción: función de comparación por orden total estricto de κ
función	$\text{COPIAR}(\text{in } k : \kappa) \rightarrow res : \kappa$ $\text{Pre} \equiv \{\text{true}\}$ $\text{Post} \equiv \{res =_{\text{obs}} k\}$ Complejidad: $\Theta(\text{copy}(k))$ Descripción: función de copia de κ
función	$\text{COPIAR}(\text{in } s : \sigma) \rightarrow res : \sigma$ $\text{Pre} \equiv \{\text{true}\}$ $\text{Post} \equiv \{res =_{\text{obs}} s\}$ Complejidad: $\Theta(\text{copy}(s))$ Descripción: función de copia de σ
se explica con:	$\text{DICCIONARIO}(\kappa, \sigma)$
géneros:	$\text{diccAVL}(\kappa, \sigma)$

Operaciones de diccionario

$\text{VACIO}() \rightarrow res : \text{diccAVL}(\kappa, \sigma)$
 $\text{Pre} \equiv \{\text{true}\}$
 $\text{Post} \equiv \{res =_{\text{obs}} \text{vacío}\}$
Complejidad: $\Theta(1)$
Descripción: Crea y devuelve un diccionario AVL vacío.

$\text{DEFINIR}(\text{in/out } d : \text{diccAVL}(\kappa, \sigma), \text{in } k : \kappa, \text{in } s : \sigma)$
 $\text{Pre} \equiv \{d =_{\text{obs}} d_0\}$
 $\text{Post} \equiv \{d =_{\text{obs}} \text{definir}(k, s, d_0)\}$
Complejidad: $\Theta(\log(n) \times \text{order}(k) + \text{copy}(k) + \text{copy}(s))$
Descripción: Define en el diccionario la clave pasada por parámetro con el significado pasado por parámetro. En caso de que la clave ya esté definida, sobrescribe su significado con el nuevo.
Aliasing: Las claves y significados se almacenan por copia.

$\text{BORRAR}(\text{in/out } d : \text{diccAVL}(\kappa, \sigma), \text{in } k : \kappa)$
 $\text{Pre} \equiv \{d =_{\text{obs}} d_0 \wedge \text{def?}(k, d)\}$
 $\text{Post} \equiv \{d =_{\text{obs}} \text{borrar}(k, d_0)\}$
Complejidad: $\Theta(\log(n) \times \text{order}(k))$
Descripción: Elimina del diccionario la clave pasada por parámetro.

#CLAVES(**in** $d: \text{diccAVL}(\kappa, \sigma) \rightarrow res: \text{nat}$

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

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

Complejidad: $\Theta(1)$

Descripción: Devuelve la cantidad de claves del diccionario.

DEFINIDO?(**in** $d: \text{diccAVL}(\kappa, \sigma)$, **in** $k: \kappa \rightarrow res: \text{bool}$

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

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

Complejidad: $\Theta(\log(n) \times \text{order}(k))$

Descripción: Devuelve true si y solo si la clave pasada por parámetro está definida en el diccionario.

OBTENER(**in** $d: \text{diccAVL}(\kappa, \sigma)$, **in** $k: \kappa \rightarrow res: \sigma$

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

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

Complejidad: $\Theta(\log(n) \times \text{order}(k))$

Descripción: Devuelve el significado con el que la clave pasada por parámetro está definida en el diccionario.

Aliasing: El significado se pasa por referencia. Modificarlo implica cambiarlo en la estructura interna del diccionario.

Representación

$\text{diccAVL}(\kappa, \sigma)$ se representa con estrAVL

donde estrAVL es $\text{tupla}(\text{raiz: puntero(nodo)},$
 $\text{cantNodos: nat})$

donde nodo es $\text{tupla}(\text{clave: } \kappa,$
 $\text{significado: } \sigma,$
 $\text{padre: puntero(nodo)},$
 $\text{izq: puntero(nodo)},$
 $\text{der: puntero(nodo)},$
 $\text{altSubarbol: nat})$

$\text{Rep} : \text{estrAVL} \rightarrow \text{bool}$

$\text{Rep}(e) \equiv \text{true} \iff e.\text{cantNodos} = \#(\text{nodos}(e)) \wedge$
 $\#(\text{claves}(e)) = \#(\text{nodos}(e)) \wedge$
 $(\forall n : \text{nodo}) (n \in \text{nodos}(e) \Rightarrow_{\text{L}} ($
 $n.\text{altSubarbol} = \text{altura}(\text{subárbol}(\&n)) \wedge$
 $\text{máx}(\text{altura}(\text{subárbol}(n.\text{izq})), \text{altura}(\text{subárbol}(n.\text{der}))) -$
 $\text{mín}(\text{altura}(\text{subárbol}(n.\text{izq})), \text{altura}(\text{subárbol}(n.\text{der}))) \leq 1 \wedge$
 $(\forall n' : \text{nodo}) ((n' \in \text{nodos}(\text{subárbol}(n))) \Rightarrow (n' \in \text{nodos}(\text{subárbol}(n.\text{der})) \Leftrightarrow \neg(n'.\text{clave} \leq n.\text{clave}))) \wedge$
 $*(n.\text{izq}) \neq *(n.\text{der}) \wedge (n.\text{padre} = \text{NULL} \Leftrightarrow \&n = e.\text{raiz}) \wedge_{\text{L}}$
 $((\&n \neq e.\text{raiz}) \Rightarrow_{\text{L}} (\forall n' : \text{nodo}) (n.\text{padre} = \&n' \Leftrightarrow n'.\text{izq} = \&n \vee n'.\text{der} = \&n))))$

$\text{Abs} : \text{estrAVL } e \rightarrow \text{dicc}(\kappa, \sigma) \quad \{\text{Rep}(e)\}$

$\text{Abs}(e) =_{\text{obs}} d: \text{dicc}(\kappa, \sigma) \mid (\forall \kappa : \kappa) ((\text{def?}(k, d)) =_{\text{obs}} (k \in \text{claves}(e)) \wedge_{\text{L}} (\text{def?}(k, d) \Rightarrow_{\text{L}} \text{obtener}(k, d) =_{\text{obs}} \text{significado}(k, e)))$

$\text{hijos} : \text{nodo} \rightarrow \text{conj}(\text{nodo})$

$\text{hijos}(n) \equiv \text{if } n.\text{izq} = \text{NULL} \text{ then } \emptyset \text{ else } \text{Ag}(*(n.\text{izq}), \text{hijos}(*(n.\text{izq}))) \text{ fi}$
 $\cup \text{if } n.\text{der} = \text{NULL} \text{ then } \emptyset \text{ else } \text{Ag}(*(n.\text{der}), \text{hijos}(*(n.\text{der}))) \text{ fi}$

$\text{nodos} : \text{estrAVL} \rightarrow \text{conj}(\text{nodo})$

$\text{nodos}(e) \equiv \text{if } e.\text{raiz} = \text{NULL} \text{ then } \emptyset \text{ else } \text{Ag}(*(e.\text{raiz}), \text{hijos}(*(e.\text{raiz}))) \text{ fi}$

subárbol : puntero(nodo) \rightarrow estrAVL

subárbol(p) $\equiv \langle p, 1 + \#(\text{hijos}(*p)) \rangle$

claves : estrAVL \rightarrow conj(κ)

claves(e) \equiv **if** $e.raiz = \text{NULL}$ **then**
 \emptyset
else
 $\text{Ag}(e.raiz \rightarrow \text{clave}, \text{claves}(\text{subárbol}(e.raiz \rightarrow \text{izq})) \cup \text{claves}(\text{subárbol}(e.raiz \rightarrow \text{der})))$
fi

altura : estrAVL \rightarrow nat

altura(e) \equiv **if** $e.raiz = \text{NULL}$ **then**
 0
else
 $1 + \text{máx}(\text{altura}(\text{subárbol}(e.raiz \rightarrow \text{izq})), \text{altura}(\text{subárbol}(e.raiz \rightarrow \text{der})))$
fi

significado : estrAVL $e \times \kappa k \rightarrow \sigma$

$\{k \in \text{claves}(e)\}$

significado(e, k) \equiv **if** $e.raiz \rightarrow \text{clave} = k$ **then**
 $e.raiz \rightarrow \text{significado}$
else
if $k \in \text{claves}(\text{subárbol}(e.raiz \rightarrow \text{izq}))$ **then**
 $\text{significado}(k, \text{subárbol}(e.raiz \rightarrow \text{izq}))$
else
 $\text{significado}(k, \text{subárbol}(e.raiz \rightarrow \text{der}))$
fi
fi

Algoritmos

iVACIO() $\rightarrow res : \text{estrAVL}$		
1	$res \leftarrow \langle \text{NULL}, 0 \rangle$	$\triangleright \Theta(1)$
Complejidad: $\Theta(1)$		
iDEFINIR(in/out $e : \text{estrAVL}$, in $k : \kappa$, in $s : \sigma$)		
1	puntero(nodo) $padre \leftarrow \text{NULL}$	$\triangleright \Theta(1)$
2	puntero(nodo) $lugar \leftarrow \text{Buscar}(e, k, padre)$	$\triangleright \Theta(\log(n) \times \text{order}(k))$
3	if $lugar \neq \text{NULL}$ then	$\triangleright \Theta(1)$
4	$(lugar \rightarrow \text{significado}) \leftarrow \text{Copiar}(s)$	$\triangleright \Theta(\text{copy}(s))$
5	else	
6	puntero(nodo) $nuevo \leftarrow \&\langle \text{Copiar}(k), \text{Copiar}(s), \text{NULL}, \text{NULL}, \text{NULL}, 1 \rangle$ // Reservamos memoria para el nuevo nodo	$\triangleright \Theta(\text{copy}(k) + \text{copy}(s))$
7	if $k \leq (padre \rightarrow \text{clave})$ then	$\triangleright \Theta(\text{order}(k))$
8	$(padre \rightarrow \text{izq}) \leftarrow nuevo$	$\triangleright \Theta(1)$
9	else	
10	$(padre \rightarrow \text{der}) \leftarrow nuevo$	$\triangleright \Theta(1)$
11	end	
12	$(nuevo \rightarrow padre) \leftarrow padre$	$\triangleright \Theta(1)$
13	RebalancearArbol ($padre$)	$\triangleright \Theta(\log(n))$
14	$e.cantNodos++$	$\triangleright \Theta(1)$
15	end	
Complejidad: $\Theta(\log(n) \times \text{order}(k) + \text{copy}(k) + \text{copy}(s))$		

Justificación: La función tiene llamadas a funciones con complejidad $\Theta(\log(n) \times order(k))$ y $\Theta(copy(k) + copy(s))$.

IOBTENER (in $e : \text{estrAVL}$, in $k : \kappa$) $\rightarrow res : \sigma$		
1 puntero(nodo) $padre \leftarrow \text{NULL}$		$\triangleright \Theta(1)$
2 puntero(nodo) $lugar \leftarrow \text{Buscar}(e, k, padre)$		$\triangleright \Theta(\log(n) \times order(k))$
3 $res \leftarrow (lugar \rightarrow significado)$		$\triangleright \Theta(1)$

Complejidad: $\Theta(\log(n) \times order(k))$

Justificación: La función tiene llamadas a funciones con complejidad $\Theta(\log(n) \times order(k))$ y $\Theta(copy(k) + copy(s))$.

I#CLAVES (in $e : \text{estrAVL}$) $\rightarrow res : \text{nat}$		
1 $res \leftarrow e.cantNodos$		$\triangleright \Theta(1)$

Complejidad: $\Theta(1)$

IDEFINIDO? (in $e : \text{estrAVL}$, in $k : \kappa$) $\rightarrow res : \text{bool}$		
1 puntero(nodo) $padre \leftarrow \text{NULL}$		$\triangleright \Theta(1)$
2 puntero(nodo) $lugar \leftarrow \text{Buscar}(e, k, padre)$		$\triangleright \Theta(\log(n) \times order(k))$
3 $res \leftarrow (lugar \neq \text{NULL})$		$\triangleright \Theta(1)$

Complejidad: $\Theta(\log(n) \times order(k))$

Justificación: $\Theta(1) + \Theta(\log(n) \times order(k)) + \Theta(1) = \Theta(\log(n) \times order(k)) + \Theta(1)$

IBORRAR(in/out e : **estrAVL, in k : κ)**

1	puntero(nodo) $padre \leftarrow \text{NULL}$	$\triangleright \Theta(1)$
2	puntero(nodo) $lugar \leftarrow \text{Buscar}(e, k, padre)$	$\triangleright \Theta(\log(n) \times \text{order}(k))$
3	if $lugar \rightarrow izq = \text{NULL} \wedge lugar \rightarrow der = \text{NULL}$ then	$\triangleright \Theta(1)$
4	if $padre \neq \text{NULL}$ then	$\triangleright \Theta(1)$
5	if $padre \rightarrow izq = lugar$ then	$\triangleright \Theta(1)$
6	$(padre \rightarrow izq) \leftarrow \text{NULL}$	$\triangleright \Theta(1)$
7	else	
8	$(padre \rightarrow der) \leftarrow \text{NULL}$	$\triangleright \Theta(1)$
9	end	
10	RebalancearArbol($padre$)	$\triangleright \Theta(\log(n))$
11	else	
12	$e.raiz = \text{NULL}$	$\triangleright \Theta(1)$
13	end	
14	else if $lugar \rightarrow der = \text{NULL}$ then	$\triangleright \Theta(1)$
15	$(lugar \rightarrow izq \rightarrow padre) \leftarrow padre$	$\triangleright \Theta(1)$
16	if $padre \neq \text{NULL}$ then	$\triangleright \Theta(1)$
17	if $padre \rightarrow izq = lugar$ then	$\triangleright \Theta(1)$
18	$(padre \rightarrow izq) \leftarrow (lugar \rightarrow izq)$	$\triangleright \Theta(1)$
19	else	
20	$(padre \rightarrow der) \leftarrow (lugar \rightarrow izq)$	$\triangleright \Theta(1)$
21	end	
22	RebalancearArbol($padre$)	$\triangleright \Theta(\log(n))$
23	else	
24	$e.raiz \leftarrow lugar \rightarrow izq$	$\triangleright \Theta(1)$
25	end	
26	else if $lugar \rightarrow izq = \text{NULL}$ then	$\triangleright \Theta(1)$
27	$(lugar \rightarrow der \rightarrow padre) \leftarrow padre$	$\triangleright \Theta(1)$
28	if $padre \neq \text{NULL}$ then	$\triangleright \Theta(1)$
29	if $padre \rightarrow izq = lugar$ then	$\triangleright \Theta(1)$
30	$(padre \rightarrow izq) \leftarrow (lugar \rightarrow der)$	$\triangleright \Theta(1)$
31	else	
32	$(padre \rightarrow der) \leftarrow (lugar \rightarrow der)$	$\triangleright \Theta(1)$
33	end	
34	RebalancearArbol($padre$)	$\triangleright \Theta(\log(n))$
35	else	
36	$e.raiz \leftarrow lugar \rightarrow izq$	$\triangleright \Theta(1)$
37	end	

IBORRAR (cont.)

```

38 else
39     puntero(nodo) reemplazo ← (lugar → der)                                ▷  $\Theta(1)$ 
40     if (reemplazo → izq = NULL) then                                       ▷  $\Theta(1)$ 
41         if padre ≠ NULL then                                              ▷  $\Theta(1)$ 
42             if (padre → izq) = lugar then                                  ▷  $\Theta(1)$ 
43                 (padre → izq) ← reemplazo                                ▷  $\Theta(1)$ 
44             else
45                 (padre → der) ← reemplazo                                ▷  $\Theta(1)$ 
46             end
47         else
48             e.raiz ← reemplazo                                           ▷  $\Theta(1)$ 
49         end
50         (reemplazo → padre) ← padre                                       ▷  $\Theta(1)$ 
51         (reemplazo → izq) ← lugar → izq                                  ▷  $\Theta(1)$ 
52         (lugar → izq → padre) ← reemplazo                                ▷  $\Theta(1)$ 
53         RebalancearArbol(reemplazo)                                       ▷  $\Theta(\log(n))$ 
54     else
55         while (reemplazo → izq) ≠ NULL do                                ▷  $\Theta(\log(n))$  iteraciones
56             reemplazo ← (reemplazo → izq)                                ▷  $\Theta(1)$ 
57         end
58         puntero(nodo) padreReemplazo ← (reemplazo → padre)               ▷  $\Theta(1)$ 
59         if padre ≠ NULL then                                              ▷  $\Theta(1)$ 
60             if padre → izq = lugar then                                  ▷  $\Theta(1)$ 
61                 (padre → izq) ← reemplazo                                ▷  $\Theta(1)$ 
62             else
63                 (padre → der) ← reemplazo                                ▷  $\Theta(1)$ 
64             end
65         else
66             e.raiz ← reemplazo                                           ▷  $\Theta(1)$ 
67         end
68         (reemplazo → padre) ← padre                                       ▷  $\Theta(1)$ 
69         (reemplazo → izq) ← lugar → izq                                  ▷  $\Theta(1)$ 
70         (lugar → izq → padre) ← reemplazo                                ▷  $\Theta(1)$ 
71         (padreReemplazo → izq) ← (reemplazo → der)                      ▷  $\Theta(1)$ 
72         if (reemplazo → der) ≠ NULL then                                  ▷  $\Theta(1)$ 
73             (reemplazo → der → padre) ← padreReemplazo                 ▷  $\Theta(1)$ 
74         end
75         (reemplazo → der) ← (lugar → der)                                ▷  $\Theta(1)$ 
76         (lugar → der → padre) ← reemplazo                                ▷  $\Theta(1)$ 
77         RebalancearArbol(reemplazo)                                       ▷  $\Theta(\log(n))$ 
78     end
79 end
80 delete(lugar) // Liberamos la memoria ocupada por el nodo eliminado.    ▷  $\Theta(1)$ 

```

Complejidad: $\Theta(\log(n) \times order(k))$

Justificación: El algoritmo tiene una llamada a función con complejidad $\Theta(\log(n) \times order(k))$, y luego presenta varios casos, pero en todos ellos las funciones llamadas son $O(\log(n))$.

IBUSCAR(in $e : \text{estrAVL}$, in $k : \kappa$, out $padre : \text{puntero}(\text{nodo})$) $\rightarrow res : \text{puntero}(\text{nodo})$

```

1  $padre \leftarrow \text{NULL}$   $\triangleright \Theta(1)$ 
2  $actual \leftarrow e.raiz$   $\triangleright \Theta(1)$ 
3 while  $actual \neq \text{NULL} \wedge_L (actual \rightarrow clave \neq k)$  do  $\triangleright \Theta(\log(n))$  iteraciones
4    $padre \leftarrow actual$   $\triangleright \Theta(1)$ 
5   if  $k \leq (padre \rightarrow clave)$  then  $\triangleright \Theta(\text{order}(k))$ 
6      $actual \leftarrow (actual \rightarrow izq)$   $\triangleright \Theta(1)$ 
7   else
8      $actual \leftarrow (actual \rightarrow der)$   $\triangleright \Theta(1)$ 
9   end
10 end
11  $res \leftarrow actual$   $\triangleright \Theta(1)$ 

```

Descripción: Esta operación privada recibe la estructura de representación interna del diccionario y una clave por parámetro. Si la clave está definida, devuelve un puntero al nodo que la contiene y coloca en el parámetro de out *padre* un puntero al padre de dicho nodo. En caso contrario, devuelve NULL y coloca en el parámetro de out *padre* un puntero a la hoja del árbol que debería ser padre del nodo buscado, si la clave estuviera definida.

Complejidad: $\Theta(\log(n) \times \text{order}(k))$

Justificación: El algoritmo presenta un ciclo que se repite $\Theta(\log(n))$ veces, y en cada una de ellas se realiza una llamada a función con complejidad $\Theta(\text{order}(k))$.

IRECALCULARALTURA(in $n : \text{puntero}(\text{nodo})$)

```

1 if  $n \rightarrow izq \neq \text{NULL} \wedge n \rightarrow der \neq \text{NULL}$  then  $\triangleright \Theta(1)$ 
2    $(n \rightarrow altSubarbol) \leftarrow 1 + \max(n \rightarrow izq \rightarrow altSubarbol, n \rightarrow der \rightarrow altSubarbol)$   $\triangleright \Theta(1)$ 
3 else if  $n \rightarrow izq \neq \text{NULL}$  then  $\triangleright \Theta(1)$ 
4    $(n \rightarrow altSubarbol) \leftarrow 1 + (n \rightarrow izq \rightarrow altSubarbol)$   $\triangleright \Theta(1)$ 
5 else if  $n \rightarrow der \neq \text{NULL}$  then  $\triangleright \Theta(1)$ 
6    $(n \rightarrow altSubarbol) \leftarrow 1 + (n \rightarrow der \rightarrow altSubarbol)$   $\triangleright \Theta(1)$ 
7 else  $\triangleright \Theta(1)$ 
8    $(n \rightarrow altSubarbol) \leftarrow 1$   $\triangleright \Theta(1)$ 
9 end

```

Descripción: Esta operación privada recibe un puntero a un nodo del árbol y recalcula el valor de su campo *altSubarbol* en función a los datos que sus nodos hijos poseen en este campo.

Complejidad: $\Theta(1)$

Justificación: El algoritmo presenta varios casos, y todos ellos realizan operaciones con complejidad $\Theta(1)$.

IFBD(in $n : \text{puntero}(\text{nodo})$) $\rightarrow res : \text{int}$

```

1  $\text{int } altIzq \leftarrow n \rightarrow izq = \text{NULL} ? 0 : n \rightarrow izq \rightarrow altSubarbol$   $\triangleright \Theta(1)$ 
2  $\text{int } altDer \leftarrow n \rightarrow der = \text{NULL} ? 0 : n \rightarrow der \rightarrow altSubarbol$   $\triangleright \Theta(1)$ 
3  $res \leftarrow altDer - altIzq$   $\triangleright \Theta(1)$ 

```

Descripción: Esta operación privada recibe un puntero a un nodo del árbol y calcula su factor de balanceo.

Complejidad: $\Theta(1)$

Justificación: $\Theta(1) + \Theta(1) + \Theta(1) = \Theta(1)$

iROTARAIZQUIERDA(in n : puntero(nodo))	
<hr/>	
1 if $n.padre \neq \text{NULL}$ then	$\triangleright \Theta(1)$
2 if $n.padre \rightarrow izq = n$ then	$\triangleright \Theta(1)$
3 $(n \rightarrow padre \rightarrow izq) \leftarrow n \rightarrow der$	$\triangleright \Theta(1)$
4 else	
5 $(n \rightarrow padre \rightarrow der) \leftarrow n \rightarrow der$	$\triangleright \Theta(1)$
6 end	
7 end	
8 $(n \rightarrow der \rightarrow padre) \leftarrow n \rightarrow padre$	$\triangleright \Theta(1)$
9 $n \rightarrow padre \leftarrow n \rightarrow der$	$\triangleright \Theta(1)$
10 $n \rightarrow der \leftarrow (n \rightarrow der \rightarrow izq)$	$\triangleright \Theta(1)$
11 if $n \rightarrow der \neq \text{NULL}$ then	$\triangleright \Theta(1)$
12 $(n \rightarrow der \rightarrow padre) \leftarrow n$	$\triangleright \Theta(1)$
13 end	
14 $(n \rightarrow padre \rightarrow izq) \leftarrow n$	$\triangleright \Theta(1)$
15 RecalcularAltura(n)	$\triangleright \Theta(1)$
16 RecalcularAltura($n \rightarrow padre$)	$\triangleright \Theta(1)$

Descripción: Esta operación privada recibe un puntero a un nodo del árbol y realiza una rotación a izquierda de dicho nodo. ¡Ojo, rompe el invariante de representación! (Los campos *altSubarbol* de los nodos superiores quedan inconsistentes).

Complejidad: $\Theta(1)$

Justificación: Todas las operaciones que realiza el algoritmo tienen complejidad $\Theta(1)$.

iROTARADERECHA(in n : puntero(nodo))	
<hr/>	
1 if $n \rightarrow padre \neq \text{NULL}$ then	$\triangleright \Theta(1)$
2 if $n \rightarrow padre \rightarrow izq = n$ then	$\triangleright \Theta(1)$
3 $(n \rightarrow padre \rightarrow izq) \leftarrow n \rightarrow izq$	$\triangleright \Theta(1)$
4 else	
5 $(n \rightarrow padre \rightarrow der) \leftarrow n \rightarrow izq$	$\triangleright \Theta(1)$
6 end	
7 end	
8 $(n \rightarrow izq \rightarrow padre) \leftarrow n \rightarrow padre$	$\triangleright \Theta(1)$
9 $n \rightarrow padre \leftarrow n \rightarrow izq$	$\triangleright \Theta(1)$
10 $n \rightarrow izq \leftarrow (n \rightarrow izq \rightarrow der)$	$\triangleright \Theta(1)$
11 if $n \rightarrow izq \neq \text{NULL}$ then	$\triangleright \Theta(1)$
12 $(n \rightarrow izq \rightarrow padre) \leftarrow n$	$\triangleright \Theta(1)$
13 end	
14 $(n \rightarrow padre \rightarrow der) \leftarrow n$	$\triangleright \Theta(1)$
15 RecalcularAltura(n)	$\triangleright \Theta(1)$
16 RecalcularAltura($n \rightarrow padre$)	$\triangleright \Theta(1)$

Descripción: Esta operación privada recibe un puntero a un nodo del árbol y realiza una rotación a derecha de dicho nodo. ¡Ojo, rompe el invariante de representación! (Los campos *altSubarbol* de los nodos superiores quedan inconsistentes).

Complejidad: $\Theta(1)$

Justificación: Todas las operaciones que realiza el algoritmo tienen complejidad $\Theta(1)$.

IREBALANCEARÁRBOL(**in** n : puntero(nodo))

```

1 puntero(nodo)  $p \leftarrow n$   $\triangleright \Theta(1)$ 
2 while  $p \neq \text{NULL}$  do  $\triangleright \Theta(\log(n))$  iteraciones
3   RecalcularAltura( $p$ )  $\triangleright \Theta(1)$ 
4   int  $fdb1 \leftarrow \text{FDB}(p)$   $\triangleright \Theta(1)$ 
5   if  $fdb1 = 2$  then  $\triangleright \Theta(1)$ 
6     puntero(nodo)  $q \leftarrow (p \rightarrow \text{der})$   $\triangleright \Theta(1)$ 
7     int  $fdb2 \leftarrow \text{FDB}(q)$   $\triangleright \Theta(1)$ 
8     if  $fdb2 = 1 \vee fdb2 = 0$  then  $\triangleright \Theta(1)$ 
9       RotarAlzquierda( $p$ )  $\triangleright \Theta(1)$ 
10       $p \leftarrow q$   $\triangleright \Theta(1)$ 
11    else if  $fdb2 = -1$  then  $\triangleright \Theta(1)$ 
12      RotarADerecha( $q$ )  $\triangleright \Theta(1)$ 
13      RotarAlzquierda( $p$ )  $\triangleright \Theta(1)$ 
14       $p \leftarrow (q \rightarrow \text{padre})$   $\triangleright \Theta(1)$ 
15    end
16  else if  $fdb1 = -2$  then
17    puntero(nodo)  $q \leftarrow (p \rightarrow \text{izq})$   $\triangleright \Theta(1)$ 
18    int  $fdb2 \leftarrow \text{FDB}(q)$   $\triangleright \Theta(1)$ 
19    if  $fdb2 = -1 \vee fdb2 = 0$  then  $\triangleright \Theta(1)$ 
20      RotarADerecha( $p$ )  $\triangleright \Theta(1)$ 
21       $p \leftarrow q$   $\triangleright \Theta(1)$ 
22    else if  $fdb2 = 1$  then  $\triangleright \Theta(1)$ 
23      RotarAlzquierda( $q$ )  $\triangleright \Theta(1)$ 
24      RotarADerecha( $p$ )  $\triangleright \Theta(1)$ 
25       $p \leftarrow (q \rightarrow \text{padre})$   $\triangleright \Theta(1)$ 
26    end
27  end
28   $p \leftarrow (p \rightarrow \text{padre})$   $\triangleright \Theta(1)$ 
29 end

```

Descripción: Esta operación privada recibe un puntero a un nodo del árbol y restaura el invariante de representación en la rama ascendente a partir de dicho nodo, realizando las rotaciones necesarias para rebalancear el árbol.

Complejidad: $\Theta(\log(n))$

Justificación: El algoritmo presenta un ciclo que se ejecuta $\Theta(\log(n))$ veces, y en cada una de ellas se realizan operaciones con complejidad $\Theta(1)$.

4. Módulo Heap

Notas preliminares

En todos los casos, al indicar las complejidades de los algoritmos, las variables que se utilizan corresponden a:

- n : Número de elementos presentes en la cola de prioridad.

Interfaz

parámetros formales

géneros	α
función	$\bullet \leq \bullet (\text{in } a_1 : \alpha, \text{in } a_2 : \alpha) \rightarrow res : \text{bool}$ Pre $\equiv \{\text{true}\}$ Post $\equiv \{res =_{\text{obs}} (a_1 \leq a_2)\}$ Complejidad: $\Theta(\text{compare}(a_1, a_2))$ Descripción: función de comparación por orden total estricto de α
función	COPIAR (in $a : \alpha$) $\rightarrow res : \alpha$ Pre $\equiv \{\text{true}\}$ Post $\equiv \{res =_{\text{obs}} a\}$ Complejidad: $\Theta(\text{copy}(a))$ Descripción: función de copia de α

se explica con: COLA DE PRIORIDAD(α)

géneros: $\text{heap}(\alpha)$

Operaciones de cola de prioridad

VACÍO () $\rightarrow res : \text{heap}(\alpha)$ Pre $\equiv \{\text{true}\}$ Post $\equiv \{res =_{\text{obs}} \text{vacío}\}$ Complejidad: $\Theta(1)$ Descripción: Devuelve un heap vacío.
ENCOLAR (in/out $h : \text{heap}(\alpha)$, in $a : \alpha$) Pre $\equiv \{\text{true}\}$ Post $\equiv \{\}$ Complejidad: $h =_{\text{obs}} h_0 \ h =_{\text{obs}} \text{encolar}(a, h_0) [\Theta(\log(n))] [\text{Agrega un elemento al heap.}]$
VACÍO? (in $h : \text{heap}(\alpha)$) $\rightarrow res : \text{bool}$ Pre $\equiv \{\text{true}\}$ Post $\equiv \{h =_{\text{obs}} \text{vacío}()\}$ Complejidad: $\Theta(1)$ Descripción: Devuelve true si y solo si el heap no tiene elementos.
DESENCOLAR (in/out $h : \text{heap}(\alpha)$) Pre $\equiv \{\neg(\text{vacío?}(h))\}$ Post $\equiv \{res =_{\text{obs}} \text{desencolar}(h)\}$ Complejidad: $\Theta(\log(n))$ Descripción: Devuelve uno de los elementos de máxima prioridad del heap, y lo elimina de la estructura.

Representación

$\text{heap}(\alpha)$ se representa con $\text{vector}(\alpha)$

$\text{Rep} : \text{vector}(\alpha) \rightarrow \text{bool}$

$$\text{Rep}(v) \equiv \text{true} \iff (\forall \text{ nat } i : \text{ i } ((2i + 1 < \text{long}(v)) \Rightarrow_{\text{L}} \text{iésimo}(2i + 1, v) \leq \text{iésimo}(i, v)) \wedge ((2i + 2 < \text{long}(v)) \Rightarrow_{\text{L}} \text{iésimo}(2i + 2, v) \leq \text{iésimo}(i, v)))$$

$$\text{iésimo} : \text{nat } i \times \text{secu}(\alpha) s \longrightarrow \alpha \quad \{i \leq \text{long}(\alpha)\}$$

$$\text{iésimo}(i, s) \equiv \text{if } i = 0 \text{ then } \text{prim}(s) \text{ else } \text{iésimo}(i - 1, \text{fin}(s)) \text{ fi}$$

$$\text{Abs} : \text{vector}(\alpha) v \longrightarrow \text{colaPrior}(\alpha) \quad \{\text{Rep}(v)\}$$

$$\text{Abs}(v) \equiv \text{if } \text{long}(v) = 0 \text{ then } \text{vacía}() \text{ else } \text{encolar}(\text{prim}(v), \text{Abs}(\text{fin}(v))) \text{ fi}$$

Algoritmos

IVACÍO() $\rightarrow res : \text{vector}(\alpha)$	
1 $res \leftarrow \text{Vaca}()$	$\triangleright \Theta(1)$

Complejidad: $\Theta(1)$

IENCOLAR(in/out $v : \text{vector}(\alpha)$, in $a : \alpha$)	
1 $\text{agregarAtras}(v, a)$	$\triangleright \Theta(1)$
2 $\text{nat } i \leftarrow \text{long}(v)$	$\triangleright \Theta(1)$
3 while $i \neq 0 \wedge v[i] < v[i \text{ div } 2 - (i + 1) \% 2]$ do	
4 $\text{swap}(v, i, i \text{ div } 2 - (i + 1) \% 2)$	$\triangleright \Theta(\text{copy}(\alpha))$
5 $i \leftarrow i \text{ div } 2 - (i + 1) \% 2$	$\triangleright \Theta(1)$
6 end	

Complejidad: $\Theta(\log(n)) \times \Theta(\text{copy}(\alpha))$

Justificación: El while se repite como máximo $\log(n)$ veces y cada repetición tiene $\Theta(\text{copy}(\alpha))$.

IVACÍO?(in $v : \text{vector}(\alpha)$) $\rightarrow res : \text{bool}$	
1 $res \leftarrow \text{esVacio?}(v)$	$\triangleright \Theta(1)$

Complejidad: $\Theta(1)$

IDENCOLAR(in/out $v : \text{vector}(\alpha)$) $\rightarrow res : \alpha$	
1 $res \leftarrow v[0]$	$\triangleright \Theta(1)$
2 $\text{nat } i \leftarrow 0$	$\triangleright \Theta(1)$
3 while $((2i + 1 \leq \text{long}(v)) \wedge v[i] < v[2i + 1]) \vee (2i + 2 \leq \text{long}(v)) \wedge v[i] < v[2i + 2]$ do	
4 if $2i + 2 \leq \text{long}(v)$ then	
5 if $v[i] < v[2i + 2]$ then	
6 $\text{swap}(v, i, 2i + 2)$	$\triangleright \Theta(\text{copy}(\alpha))$
7 $i \leftarrow 2i + 2$	$\triangleright \Theta(1)$
8 else	
9 $\text{swap}(v, i, 2i + 1)$	$\triangleright \Theta(\text{copy}(\alpha))$
10 $i \leftarrow 2i + 1$	$\triangleright \Theta(1)$
11 end	
12 else	
13 $\text{swap}(v, i, 2i + 1)$	$\triangleright \Theta(\text{copy}(\alpha))$
14 $i \leftarrow 2i + 1$	$\triangleright \Theta(1)$
15 end	
16 end	

Complejidad: $\Theta(\log(n)) \times \Theta(\text{copy}(\alpha))$

Justificación: El while se repite como máximo $\log(n)$ veces y cada repetición tiene $\Theta(\text{copy}(\alpha))$.

ISWAP(**in/out** $v : \text{vector}(\alpha)$, **in** $i : \text{nat}$, **in** $j : \text{nat}$)

1 $\alpha \text{ aux} \leftarrow v[i]$	$\triangleright \Theta(\text{copy}(\alpha))$
2 $v[i] \leftarrow v[j]$	$\triangleright \Theta(\text{copy}(\alpha))$
3 $v[j] \leftarrow \text{aux}$	$\triangleright \Theta(\text{copy}(\alpha))$

Pre $\equiv \{i < \text{long}(v) \wedge j < \text{long}(v)\}$

Complejidad: $\Theta(\text{copy}(\alpha))$

Justificación: $3 \times \Theta(\text{copy}(\alpha)) = \Theta(\text{copy}(\alpha))$

5. Módulo Diccionario Trie

Notas preliminares

En todos los casos, al indicar las complejidades de los algoritmos, las variables que se utilizan corresponden a:

- n : Cantidad de claves definidas en el diccionario.

Interfaz

parámetros formales

géneros	α
función	$\bullet = \bullet(\text{in } a_0 : \alpha, \text{in } a_1 : \alpha) \rightarrow res : \text{bool}$ Pre $\equiv \{\text{true}\}$ Post $\equiv \{res =_{\text{obs}} \text{equal}(a_1, a_2)\}$ Complejidad: $\Theta(\text{equal}(a_1, a_2))$ Descripción: función de igualdad de α
función	COPIAR (in $a : \alpha$) $\rightarrow res : \alpha$ Pre $\equiv \{\text{true}\}$ Post $\equiv \{res =_{\text{obs}} a\}$ Complejidad: $\Theta(\text{copy}(a))$ Descripción: función de copia de α

se explica con: DICCIONARIO(SECUENCIA(CHAR), α)

géneros: $\text{diccTrie}(\alpha)$

Operaciones de diccionario

VACÍO() $\rightarrow res : \text{diccTrie}(\alpha)$
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \text{vacío}\}$
Complejidad: $\Theta(1)$
Descripción: Devuelve un diccionario vacío.

DEFINIR(**in/out** $d : \text{diccTrie}(\alpha)$, **in** $k : \text{string}$, **in** $s : \alpha$) $\rightarrow res : \text{diccTrie}(\alpha)$
Pre $\equiv \{d =_{\text{obs}} d_0\}$
Post $\equiv \{d =_{\text{obs}} \text{definir}(k, s, d_0)\}$
Complejidad: $\Theta(L)$
Descripción: Define una palabra en el diccionario.

BORRAR(**in/out** $d : \text{diccTrie}(\alpha)$, **in** $k : \text{string}$) $\rightarrow res : \text{diccTrie}(\alpha)$
Pre $\equiv \{d =_{\text{obs}} d_0 \wedge \text{def?}(d, k)\}$
Post $\equiv \{d =_{\text{obs}} \text{borrar}(k, s, d_0)\}$
Complejidad: $\Theta(L)$
Descripción: Borra una palabra del diccionario.

DEF?(**in** $d : \text{diccTrie}(\alpha)$, **in** $k : \text{string}$) $\rightarrow res : \text{bool}$
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \text{def?}(d, k)\}$
Complejidad: $\Theta(L)$
Descripción: Devuelve true si y solo si la clave k está definida en el diccionario.

OBTENER(**in** $d : \text{diccTrie}(\alpha)$, **in** $k : \text{string}$) $\rightarrow res : \alpha$
Pre $\equiv \{\text{def?}(d, k)\}$
Post $\equiv \{res =_{\text{obs}} \text{obtener}(d, k)\}$
Complejidad: $\Theta(L)$
Descripción: Devuelve el significado asociado a la clave k .