

# 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()  $\rightarrow res : \text{Red}$

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

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

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

**Descripción:** Genera una nueva red sin ninguna computadora.

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

**Pre**  $\equiv \{r =_{\text{obs}} r_0 \wedge (\forall c' : \text{compu})(c' \in \text{computadoras}(r) \rightarrow \text{ip}(c) \neq \text{ip}(c'))\}$

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

**Complejidad:**  $\Theta(I)$

**Descripción:** Agrega una nueva pc a una red.

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

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

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

**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**  $r : \text{Red}$ )  $\rightarrow res : \text{conj}(\text{compu})$

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

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

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

**Descripción:** Devuelve todas las computadoras de la red.

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

**Complejidad:**  $\Theta(n + I)$

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

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

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

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

**Complejidad:**  $\Theta(n + I)$

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

VECINOS(**in**  $r : \text{Red}$ , **in**  $c : \text{compu}$ )  $\rightarrow res : \text{conj}(\text{compu})$

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

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

**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 posicion.

## Representación

### Estructura Red

**Red se representa con estrRed**

donde **estrRed** es  $\text{tupla}(\text{compus: lista(estrCompu)}, \text{cantidadCompus: nat})$

donde **estrCompu** es  $\text{tupla}(\text{IP: string}, \text{conexiones: lista(tupla(inter: interfaz, com: itLista(estrCompu)))})$

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

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

$\text{Abs} : \text{estrRed } e \rightarrow \text{Red}$

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

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

### Estructura iterador de Red

**itRed se representa con itLista(estrCompu)**

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

$\text{Rep}(it) \equiv \text{true}$

$\text{Abs} : \text{itRed } itl \rightarrow \text{itUni(estrCompu)}$

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

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

$\text{ArmarComputadoras} : \text{secu}(\text{tupla}(\text{string}, \text{secu}(\text{tupla}(\text{Interfaz}, \text{ItRed})))) \rightarrow \text{conj}(\text{compu})$

$\text{ArmarComputadoras}(l) \equiv \text{if vacia?}(l) \text{ then}$

$\emptyset$

**else**

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

**fi**

$\text{ArmarSecuencia} : \text{secu}(\text{tupla}(\text{string}, \text{secu}(\text{tupla}(\text{interfaz}, \text{itLista}(\text{compu})))) \rightarrow \text{secu}(\text{string})$

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

$\text{sinRepetidos} : \text{secu}(\text{string}) \rightarrow \text{bool}$

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

$\text{pasarSecuAConj} : \text{secu}(\text{string}) \rightarrow \text{conj}(\text{string})$

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

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

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

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

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

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

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

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

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

## 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	$res \leftarrow vacio()$	$\triangleright \Theta(1)$
2	<b>if</b> pertenece?( $c_2$ , vecinos( $r$ , $c_1$ )) <b>then</b>	$\triangleright \Theta(I)$
3	agregar( $res$ , agregarAtras(agregarAtras( $<>$ , $c_1$ ), $c_2$ ))	$\triangleright \Theta(n + I)$
4	<b>else</b>	
5	$res \leftarrow dameMinimos(Caminos(r, c_1, c_2, agregarAtras(<>, c_1), pasarConjASecu(vecinos(r, c_1))))$	
6	<b>end</b>	

---

**Complejidad:**

---

DAMEMINIMOS(in $c$ : conj(lista(compu))) $\rightarrow res$ : conj(lista(compu))		
1	<b>if</b> esVacio?( $c$ ) <b>then</b>	$\triangleright \Theta(1)$
2	$res \leftarrow vacio()$	$\triangleright \Theta(1)$
3	<b>else</b>	
4	ItConj(lista(compu)) $it \leftarrow crearIt(c)$	$\triangleright \Theta(1)$
5	$res \leftarrow dameMinimosAux(c, minimaLong(c, long(siguiete(it))))$	$\triangleright \Theta(n \times n!)$
6	<b>end</b>	

---

**Complejidad:**

$\triangleright \Theta(n \times n!)$

7 **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))		
2	ItConj(lista(compu)) $it \leftarrow crearIt(c)$	$\triangleright \Theta(1)$
3	$res \leftarrow vacio()$	
4	<b>while</b> haySiguiete( $it$ ) <b>do</b>	$\triangleright \Theta(1)$
5	<b>if</b> long(siguiete( $it$ )) = $n$ <b>then</b>	$\triangleright \Theta(n!)$ iteraciones
6	agregar( $res$ , siguiete( $it$ ))	$\triangleright \Theta(1)$
7	avanzar( $it$ )	$\triangleright \Theta(n)$
8	<b>else</b>	$\triangleright \Theta(1)$
9	avanzar( $it$ )	$\triangleright \Theta(1)$
10	<b>end</b>	
11	<b>end</b>	

---

12 **Complejidad:**

$\triangleright \Theta(n \times n!)$

---

MINIMALONG(in $c$ : conj(lista(compu)), in $n$ : nat) $\rightarrow res$ : nat		
2	nat $i \leftarrow n$	$\triangleright \Theta(1)$
3	ItConj(lista(compu)) $it \leftarrow crearIt(c)$	$\triangleright \Theta(1)$
4	<b>while</b> haySiguiete( $it$ ) <b>do</b>	$\triangleright \Theta(n!)$
5	<b>if</b> long(siguiete( $it$ )) <b>then</b>	$\triangleright \Theta(1)$
6	$i \leftarrow longitud(siguiete(it))$	$\triangleright \Theta(1)$
7	avanzar( $it$ )	$\triangleright \Theta(1)$
8	<b>else</b>	
9	avanzar( $it$ )	$\triangleright \Theta(1)$
10	<b>end</b>	
11	<b>end</b>	
12	$res \leftarrow i$	$\triangleright \Theta(1)$

---

13 **Complejidad:**

$\triangleright \Theta(n!)$

14 **Justificación:** Devuelve la longitud de la secuencia más chica

---

	PASARCONJASECU( <b>in</b> $c$ : 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?( <b>in</b> $r$ : estrRed, <b>in</b> $c_1$ : compu, <b>in</b> $c_2$ : compu) $\rightarrow res$ : bool	
1	2 $res \leftarrow (\neg esVacio?(iCaminosMinimos(r, c_1, c_2)))$	

---

3 **Complejidad:**

---

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

---

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

---

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

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

---

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

---

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

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

---

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

---

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

---

	<b>1</b> $\text{ICOPIAR}(\text{in } r : : \text{estrRed}) \rightarrow res : \text{Red}$	
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

---

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

---

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

#### 4 Algoritmos del iterador de Red

---

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

---

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

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

## 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}$$

Abs : estrDCNet  $e \rightarrow$  DCNet{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   $itRed\ it_0 \leftarrow \text{crearItRed}(res.\text{red})$   $\triangleright \Theta(1)$ 
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              $\triangleright \Theta(n)$ 
23              $itConj\ it_0 \leftarrow \text{crearIt}(\text{caminoMinimos}(res.\text{red}, \text{siguiente}(res.\text{IPsCompusPorID}[j]),$ 
24                  $\text{siguiente}(res.\text{IPsCompusPorID}[k])))$ 
25              $res.\text{siguientesCompus}[j][k] \leftarrow \text{prim}(\text{fin}(\text{siguiente}(it_1)))$ 
26         end
27          $k \leftarrow k + 1$ 
28     end
29      $j \leftarrow j + 1$ 
30 end

```

---

**Complejidad:**  $\triangleright \Theta()$

---

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

---

```

2   $\text{nat } o \leftarrow \text{iObtener}(p.\text{origen}, d.\text{IDsCompusPorIP})$   $\triangleright \Theta(L)$ 
3   $\text{nat } dest \leftarrow \text{iObtener}(p.\text{destino}, d.\text{IDsCompusPorIP})$   $\triangleright \Theta(L)$ 
4   $it \leftarrow \text{CrearIt}((d.\text{paquetesEnEspera}[o]).\text{enConjunto})$   $\triangleright \Theta(1)$ 
5   $it \leftarrow \text{Agregar}((d.\text{paquetesEnEspera}[o]).\text{enConjunto}, p)$   $\triangleright \Theta(k)$ 
6   $\text{Definir}(d.\text{paquetesEnEspera}[o].\text{porID}, p.\text{ID}, \langle it, o, dest, \rangle)$   $\triangleright \Theta(\log(k))$ 
7   $\text{encolar}(d.\text{paquetesEnEspera}[o].\text{porPrioridad}, p.\text{prioridad}, it)$   $\triangleright \Theta(\log(k))$ 

```

---

**s Complejidad:**  $\Theta(L + \log(k))$

IAVANZARSEGUNDO(in/out d: DCNet)		
2	nat $j \leftarrow 0$	$\triangleright \Theta(1)$
3	nat $o$	$\triangleright \Theta(1)$
4	nat $dest$	$\triangleright \Theta(1)$
5	paquete $paq$	$\triangleright \Theta(1)$
6	while $j < \text{Cardinal}(\text{Computadoras}(d.\text{red}))$ do	$\triangleright \Theta(n)$
7	if $!(\text{EsVacio?}(d.\text{paquetesEnEspera}[j]).\text{enConjunto})$ then	$\triangleright \Theta(1)$
8	$paq \leftarrow \text{Siguiente}(\text{desencolar}((d.\text{paquetesEnEspera}[j]).\text{porPrioridad}))$	$\triangleright \Theta(\log(k))$
9	$o \leftarrow (\text{Obtener}((d.\text{paquetesEnEspera}[j]).\text{porID}, paq.\text{ID})).\text{codOrigen}$	$\triangleright \Theta(\log(k))$
10	$dest \leftarrow (\text{Obtener}((d.\text{paquetesEnEspera}[j]).\text{porID}, paq.\text{ID})).\text{codDestino}$	$\triangleright \Theta(\log(k))$
11	$i \leftarrow (\text{Obtener}((d.\text{paquetesEnEspera}[j]).\text{porID}, paq.\text{ID})).i\text{Paquete}$	$\triangleright \Theta(\log(k))$
12	$\text{Borrar}((d.\text{paquetesEnEspera}[j]).\text{porID}, paq.\text{ID})$	$\triangleright \Theta(\log(k))$
13	$\text{EliminarSiguiente}(i)$	$\triangleright \Theta(\log(1))$
14	$d.\#paqEnviados[j]++$	$\triangleright \Theta(1)$
15	if $!(d.\text{siguienteCompu}[j][dest] = dest)$ then	$\triangleright \Theta(1)$
16	$it \leftarrow \text{crearIt}(d.\text{paquetesEnEspera}[d.\text{siguienteCompu}[j][dest])$	$\triangleright \Theta(1)$
17	$it \leftarrow \text{AgregarRapido}((d.\text{paquetesEnEspera}[d.\text{siguienteCompu}[j][dest]).\text{enConjunto}, p)$	$\triangleright \Theta(I)$ lo
18	tomamos como $\Theta(1)$	
19	$\text{Definir}(d.\text{paquetesEnEspera}[d.\text{siguienteCompu}[j][dest].\text{porID}, p.\text{ID}, \langle it, d.\text{siguienteCompu}[j][dest], dest, \rangle)$	$\triangleright \Theta(\log(k))$
20	$\text{encolar}(d.\text{paquetesEnEspera}[d.\text{siguienteCompu}[j][dest].\text{porPrioridad}, paq.\text{prioridad}, it)$	$\triangleright \Theta(\log(k))$
21	end	
22	end	
23	nat $k \leftarrow 0$	$\triangleright \Theta(1)$
24	nat $h \leftarrow 0$	$\triangleright \Theta(1)$
25	if $\text{Cardinal}(\text{Computadoras}(d.\text{red})) > 0$ then	$\triangleright \Theta(1)$
26	while $k < \text{Cardinal}(\text{Computadoras}(d.\text{red}))$ do	$\triangleright \Theta(n)$ iteraciones
27	if $d.\#paqEnviados[k] > d.\#paqEnviados[h]$ then	$\triangleright \Theta(1)$
28	$h \leftarrow k$	$\triangleright \Theta(1)$
29	$k++$	$\triangleright \Theta(1)$
30	end	
31	end	
32	end	
33	$d.\text{laQueMásEnvío} \leftarrow d.\text{IPsCompusPorID}[h]$	$\triangleright \Theta(1)$
34	Complejidad:	$\triangleright \Theta(n \times \log(k))$ esta incluido en $\triangleright \Theta(n \times (L + \log(k) + \log(n)))$
35		

ICAMINORECORRIDO(in d: DCNet, in p: paquete) $\rightarrow res : \text{lista}(\text{compu})$		
2	nat $j \leftarrow 0$	$\triangleright \Theta(1)$
3	$res \leftarrow i\text{Vacía}()$	$\triangleright \Theta(1)$
4	while $!(i\text{Definido?}((d.\text{paquetesEnEspera}[j]).\text{porID}), p.\text{ID})$ do	$\triangleright \Theta(\log(k))$ iteraciones
5	$j++$	$\triangleright \Theta(1)$
6	end	
7	nat $o$	$\triangleright \Theta(1)$
8	$o \leftarrow (\text{Obtener}((d.\text{paquetesEnEspera}[j]).\text{porID}, p.\text{ID})).\text{codOrigen}$	$\triangleright \Theta(\log(k))$
9	nat $dest$	$\triangleright \Theta(1)$
10	$dest \leftarrow (\text{Obtener}((d.\text{paquetesEnEspera}[o]).\text{porID}, p.\text{ID})).\text{codDestino}$	$\triangleright \Theta(\log(k))$
11	while $!(\text{Definido?}((d.\text{paquetesEnEspera}[o]).\text{porID}), p.\text{ID})$ do	$\triangleright \Theta(n \times \log(k))$
12	$i\text{AgregarAtras}(res, \text{siguiente}(d.\text{IPsCompusPorID}[0]))$	$\triangleright \Theta(I)$ lo tomamos como $\Theta(1)$
13	$o \leftarrow d.\text{siguientesCompus}[o][dest]$	$\triangleright \Theta(1)$
14	end	
15	$\text{AgregarAtras}(res, \text{siguiente}(d.\text{IPsCompusPorID}[o]))$	$\triangleright \Theta(I)$ lo tomamos como $\Theta(1)$

---

	$\text{ICANTIDADENVIADOS}(\text{in } d : \text{DCNet}, \text{in } c : \text{compu}) \rightarrow res : \text{nat}$	
$^1$	$\mathbf{2} \text{ nat } i \leftarrow \text{Obtener}(d.\text{IDsCompusPorIP}, c.\text{IP})$ $\mathbf{3} \text{ res} \leftarrow d.\text{\#paqEnviados}[i]$	$\triangleright \Theta(L)$ $\triangleright \Theta(1)$

---

### 4 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

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

#### Operaciones de diccionario

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

**DEFINIR**(**in/out**  $d : \text{diccAVL}(\kappa, \sigma)$ , **in**  $k : \kappa$ , **in**  $s : \sigma$ )  
**Pre**  $\equiv \{d =_{\text{obs}} d_0\}$   
**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.

**BORRAR**(**in/out**  $d : \text{diccAVL}(\kappa, \sigma)$ , **in**  $k : \kappa$ )  
**Pre**  $\equiv \{d =_{\text{obs}} d_0 \wedge \text{def?}(k, d)\}$   
**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  $\text{estrAVL}$

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

donde  $\text{nodo}$  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( $\kappa$ )

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() $\rightarrow res$ : estrAVL		
1	$res \leftarrow \langle \text{NULL}, 0 \rangle$	$\triangleright \Theta(1)$
<b>Complejidad:</b> $\Theta(1)$		
iDEFINIR(in/out $e$ : 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	<b>if</b> $lugar \neq \text{NULL}$ <b>then</b>	$\triangleright \Theta(1)$
4	$(lugar \rightarrow \text{significado}) \leftarrow \text{Copiar}(s)$	$\triangleright \Theta(\text{copy}(s))$
5	<b>else</b>	
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	<b>if</b> $k \leq (padre \rightarrow \text{clave})$ <b>then</b>	$\triangleright \Theta(\text{order}(k))$
8	$(padre \rightarrow \text{izq}) \leftarrow nuevo$	$\triangleright \Theta(1)$
9	<b>else</b>	
10	$(padre \rightarrow \text{der}) \leftarrow nuevo$	$\triangleright \Theta(1)$
11	<b>end</b>	
12	$(nuevo \rightarrow padre) \leftarrow padre$	$\triangleright \Theta(1)$
13	RebalancearArbol ( $padre$ )	$\triangleright \Theta(\log(n))$
14	$e.cantNodos++$	$\triangleright \Theta(1)$
15	<b>end</b>	
<b>Complejidad:</b> $\Theta(\log(n) \times \text{order}(k) + \text{copy}(k) + \text{copy}(s))$		

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

---

<b>IOBTENER</b> ( <b>in</b> $e : \text{estrAVL}$ , <b>in</b> $k : \kappa$ ) $\rightarrow res : \sigma$		
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  $\Theta(copy(k) + copy(s))$ .

---

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

---

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

---

<b>IDEFINIDO?</b> ( <b>in</b> $e : \text{estrAVL}$ , <b>in</b> $k : \kappa$ ) $\rightarrow res : \text{bool}$		
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  $k$ :  $\kappa$ )**


---

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

---



## 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  $padre : \text{puntero(nodo)}$ )  $\rightarrow res : \text{puntero(nodo)}$ 


---

```

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

---

```

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


---

```

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

```

---

**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  $\triangleright \Theta(1)$ 
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( $\alpha$ )

### Interfaz

parámetros formales

**géneros**       $\alpha$

**función**       $\bullet < \bullet (\text{in } a_1 : \alpha, \text{in } a_2 : \alpha) \rightarrow res : \text{bool}$   
**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 de menor de  $\alpha$ .

**función**       $\text{COPIAR}(\text{in } a : \alpha) \rightarrow res : \alpha$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} a\}$   
**Complejidad:**  $\Theta(\text{copy}(a))$   
**Descripción:** función de copia de  $\alpha$

**se explica con:**       $\text{COLAPRIOR}(\alpha)$

**géneros:**               $\text{heap}(\alpha)$

### Operaciones del heap

$\text{VACÍO}() \rightarrow res : \text{heap}(\alpha)$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{vacío}\}$   
**Complejidad:**  $\Theta(1)$   
**Descripción:** Constructor por defecto de  $\text{heap}(\alpha)$

$\text{ENCOLAR}(\text{in/out } h : \text{heap}(\alpha), \text{in } a : \alpha)$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{\}$   
**Complejidad:**  $h =_{\text{obs}} h_0 \ h =_{\text{obs}} \text{encolar}(a, h_0) \ [\Theta(\log(h.\text{longitud}))]$  [Agrega un elemento a la cola de prioridades]

$\text{VACÍO?}(\text{in } h : \text{heap}(\alpha)) \rightarrow res : \text{bool}$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{h =_{\text{obs}} \text{vacío}()\}$   
**Complejidad:**  $\Theta(1)$   
**Descripción:** Devuelve true si y solo si h es un heap vacío

$\text{PRÓXIMO}(\text{in } h : \text{heap}(\alpha)) \rightarrow res : \alpha$   
**Pre**  $\equiv \{\neg \text{vacío?}(h)\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{próximo}(h)\}$   
**Complejidad:**  $\Theta(1)$   
**Descripción:** Devuelve el próximo elemento en el heap

$\text{DESENCOLAR}(\text{in/out } h : \text{heap}(\alpha))$   
**Pre**  $\equiv \{\neg \text{vacío?}(h)\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{desencolar}(h)\}$   
**Complejidad:**  $\Theta(\log(h.\text{longitud}))$   
**Descripción:** Elimina el próximo elemento en el heap

$\text{DESENCOLAR2}(\text{in/out } h : \text{heap}(\alpha)) \rightarrow res : \alpha$   
**Pre**  $\equiv \{h =_{\text{obs}} h_0 \wedge \neg \text{vacío?}(h)\}$   
**Post**  $\equiv \{res =_{\text{obs}} \text{próximo}(h_0) \wedge h = \text{desencolar}(h_0)\}$   
**Complejidad:**  $\Theta(\log(h.\text{longitud}))$

**Descripción:** Elimina el próximo elemento en el heap

## Representación

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

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

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

$\text{Abs} : \text{vector}(\alpha) \rightarrow \text{colaPrior}(\alpha)$

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

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

## Algoritmos

---

$\text{IVACIO}() \rightarrow \text{res} : \text{vector}(\alpha)$

---

1  $\text{res} \leftarrow \text{Vaca}()$

---



---

$\text{IENCOLAR}(\text{in/out } hp : \text{heap}(\alpha), \text{in } a : \alpha)$

---

1  $hp.\text{push\_back}(a)$   
 2  $\text{nat } i \leftarrow hp.\text{longitud} - 1$   
 3  $\text{nat } p \leftarrow (i/2) - ((i+1) \% 2)$   
 4 **while**  $hp[p] < hp[i]$  **do**  
 5      $\text{iSwap}(hp[p], hp[i])$   
 6      $i \leftarrow p$   
 7      $p \leftarrow (i/2) - ((i+1) \% 2)$   
 8 **end**

---



---

$\text{IVACÍO?}(\text{in } v : \text{vector}(\alpha)) \rightarrow \text{res} : \text{bool}$

---

1  $\text{res} \leftarrow \text{esVacío?}(v)$

---

---

IDSENCOLAR(**in/out**  $v : \text{vector}(\alpha)$ )  $\rightarrow res : \alpha$ 


---

```

1 nat i ← hp.longitud - 1
2 iSwap(hp[0], hp[i])
3 res ← hp.pop_back()
4 nat hijo0 ← 1
5 nat hijo1 ← 2
6 bool estaOrdenado ← false
7 while estaOrdenado do
8   if hijo0 < hp.longitud then
9     if hijo1 < hp.longitud then
10      if hp[hijo0] ≥ hp[hijo1] then
11        if hp[hijo0] > hp[i] then
12          iSwap(hp[hijo0], hp[i])
13          i ← hijo0
14        else
15          estaOrdenado ← True
16        end
17      else
18        if hp[hijo1] > hp[i] then
19          iSwap(hp[hijo1], hp[i])
20          i ← hijo1
21        else
22          estaOrdenado ← True
23        end
24      end
25    else
26      if hp[hijo0] > hp[i] then
27        iSwap(hp[hijo0], hp[i])
28        i ← hijo0
29      else
30        estaOrdenado ← True
31      end
32    end
33  else
34    estaOrdenado ← True
35  end
36  hijo0 ← 2xi+1
37  hijo1 ← 2xi+2
38 end

```

---



---

ISWAP(**in/out**  $a : \alpha$ , **in/out**  $b : \alpha$ )

---

```

1  $\alpha$  c
2  $c \leftarrow a$ 
3  $a \leftarrow b$ 
4  $b \leftarrow c$ 

```

---

## 5. Módulo Diccionario Trie

### Interfaz

#### parámetros formales

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

**se explica con:**  $\text{DICC}(\text{SECU}(\text{CHAR}), \alpha)$

**géneros:**  $\text{diccTrie}(\alpha)$

### Operaciones de diccionario

$\text{VACÍO}() \rightarrow res : \text{diccTrie}(\alpha)$ <b>Pre</b> $\equiv \{\text{true}\}$ <b>Post</b> $\equiv \{res =_{\text{obs}} \text{vacío}\}$ <b>Complejidad:</b> $\Theta(1)$ <b>Descripción:</b> Constructor por defecto de $\text{diccTrie}(\alpha)$
$\text{DEFINIR}(\text{in/out } d : \text{diccTrie}(\alpha), \text{in } k : \text{string}, \text{in } s : \alpha) \rightarrow res : \text{diccTrie}(\alpha)$ <b>Pre</b> $\equiv \{d =_{\text{obs}} d_0\}$ <b>Post</b> $\equiv \{d =_{\text{obs}} \text{definir}(k, s, d_0)\}$ <b>Complejidad:</b> $\Theta(L)$ <b>Descripción:</b> Define una palabra en el $\text{diccTrie}(\alpha)$
$\text{BORRAR}(\text{in/out } d : \text{diccTrie}(\alpha), \text{in } k : \text{string}) \rightarrow res : \text{diccTrie}(\alpha)$ <b>Pre</b> $\equiv \{d =_{\text{obs}} d_0 \wedge \text{def?}(d, k)\}$ <b>Post</b> $\equiv \{d =_{\text{obs}} \text{borrar}(k, s, d_0)\}$ <b>Complejidad:</b> $\Theta(L)$ <b>Descripción:</b> Borra una definicion en el $\text{diccTrie}(\alpha)$
$\text{DEF?}(\text{in } d : \text{diccTrie}(\alpha), \text{in } k : \text{string}) \rightarrow res : \text{bool}$ <b>Pre</b> $\equiv \{\text{true}\}$ <b>Post</b> $\equiv \{res =_{\text{obs}} \text{def?}(d, k)\}$ <b>Complejidad:</b> $\Theta(L)$ <b>Descripción:</b> Pregunta si la palabra $k$ esta definida en el $\text{diccTrie}(\alpha)$
$\text{OBTENER}(\text{in } d : \text{diccTrie}(\alpha), \text{in } k : \text{string}) \rightarrow res : \alpha$ <b>Pre</b> $\equiv \{\text{def?}(d, k)\}$ <b>Post</b> $\equiv \{res =_{\text{obs}} \text{obtener}(d, k)\}$ <b>Complejidad:</b> $\Theta(L)$ <b>Descripción:</b> Devuelve el significado de la palabra $k$