

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 Diccionario AVL(κ, σ)

Interfaz

parámetros formales

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

Operaciones de diccionario

VACIO () $\rightarrow res : \text{diccAVL}(\kappa, \sigma)$ Pre $\equiv \{\text{true}\}$ Post $\equiv \{res =_{\text{obs}} \text{vacío}\}$
DEFINIR (in/out $d : \text{diccAVL}(\kappa, \sigma)$, in $k : \kappa$, in $s : \sigma$) Pre $\equiv \{d =_{\text{obs}} d_0\}$ Post $\equiv \{d =_{\text{obs}} \text{definir}(k, s, d_0)\}$
BORRAR (in/out $d : \text{diccAVL}(\kappa, \sigma)$, in $k : \kappa$) Pre $\equiv \{d =_{\text{obs}} d_0 \wedge \text{def?}(k, d)\}$ Post $\equiv \{d =_{\text{obs}} \text{borrar}(k, d_0)\}$
#CLAVES (in $d : \text{diccAVL}(\kappa, \sigma)$) $\rightarrow res : \text{nat}$ Pre $\equiv \{\text{true}\}$ Post $\equiv \{res =_{\text{obs}} \#(\text{claves}(d))\}$
DEFINIDO? (in $d : \text{diccAVL}(\kappa, \sigma)$, in $k : \kappa$) $\rightarrow res : \text{bool}$ Pre $\equiv \{\text{true}\}$ Post $\equiv \{res =_{\text{obs}} \text{def?}(k, d)\}$
OBTENER (in $d : \text{diccAVL}(\kappa, \sigma)$, in $k : \kappa$) $\rightarrow res : \sigma$ Pre $\equiv \{\text{def?}(k, d)\}$

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

Representación

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

donde `estrAVL` es `tupla(raiz: puntero(nodo), cantNodos: nat)`

donde `nodo` es `tupla`(*clave*: κ ,
significado: σ ,
padre: `puntero`(`nodo`),
izq: `puntero`(`nodo`),
der: `puntero`(`nodo`),
altSubarbol: `nat`)

$$\text{Rep} : \text{estrAVL} \longrightarrow \text{bool}$$
$$\begin{aligned} \text{Rep}(e) \equiv & \text{true} \iff e.\text{cantNodos} = \#(\text{nodos}(e)) \wedge \\ & \#(\text{claves}(e)) = \#(\text{nodos}(e)) \wedge \\ & (\forall n : \text{nodo}) (n \in \text{nodos}(e) \Rightarrow_{\text{L}} (\\ & n.\text{altSubarbol} = \text{altura}(\text{subárbol}(\&n)) \wedge \\ & \text{máx}(\text{altura}(\text{subárbol}(n.\text{izq})), \text{altura}(\text{subárbol}(n.\text{der}))) - \\ & \text{mín}(\text{altura}(\text{subárbol}(n.\text{izq})), \text{altura}(\text{subárbol}(n.\text{der}))) \leq 1 \wedge \\ & (\forall n' : \text{nodo}) (n' \in \text{nodos}(\text{subárbol}(n)) \Rightarrow n' \in \text{nodos}(\text{subárbol}(n.\text{der})) \Leftrightarrow \&n'.\text{clave} \leq n.\text{clave})) \wedge \\ & (* (n.\text{izq}) \neq * (n.\text{der}) \wedge ((n.\text{padre} = \text{NULL})) \Leftrightarrow \&n = e.\text{raiz}) \wedge_{\text{L}} \\ & ((\&n \neq e.\text{raiz}) \Rightarrow_{\text{L}} (\forall n' : \text{nodo}) (n.\text{padre} = \&n' \Leftrightarrow n'.\text{izq} = \&n \vee n'.\text{der} = \&n)))) \end{aligned}$$
$$\text{hijos} : \text{nodo} \longrightarrow \text{conj}(\text{nodo})$$
$$\begin{aligned} \text{hijos}(n) \equiv & \text{if } n.\text{izq} = \text{NULL} \text{ then } \emptyset \text{ else } \text{Ag}(*n.\text{izq}, \text{hijos}(*n.\text{izq})) \text{ fi} \\ & \cup \text{if } n.\text{der} = \text{NULL} \text{ then } \emptyset \text{ else } \text{Ag}(*n.\text{der}, \text{hijos}(*n.\text{der})) \text{ fi} \end{aligned}$$
$$\text{nodos} : \text{estrAVL} \longrightarrow \text{conj}(\text{nodo})$$
$$\text{nodos}(e) \equiv \text{if } e.\text{raiz} = \text{NULL} \text{ then } \emptyset \text{ else } \text{Ag}(*(\text{e.raiz}), \text{hijos}(*(\text{e.raiz}))) \text{ fi}$$
$$\text{subárbol} : \text{puntero}(\text{nodo}) \longrightarrow \text{estrAVL}$$
$$\text{subárbol}(p) \equiv \langle p, 1 + \#(\text{hijos}(*p)) \rangle$$
$$\text{claves} : \text{estrAVL} \longrightarrow \text{conj}(\kappa)$$

```

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

```

$$\text{altura} : \text{estrAVL} \longrightarrow \text{nat}$$

```

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

```

significado : $\text{estrAVL } e \times \kappa k \rightarrow \sigma$ $\{k \in \text{claves}(e)\}$

```

significado(e, k)  $\equiv$  if e.raiz  $\rightarrow$  clave = k then
    e.raiz  $\rightarrow$  significado
else
    if k  $\in$  claves(subárbol(e.raiz  $\rightarrow$  izq)) then
        significado(k, subárbol(e.raiz  $\rightarrow$  izq))
    else
        significado(k, subárbol(e.raiz  $\rightarrow$  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* : **estrAVL**, **in** *k* : κ , **in** *s* : σ)

```

1 puntero(nodo) padre  $\leftarrow$  NULL  $\triangleright \Theta(1)$ 
2 puntero(nodo) lugar  $\leftarrow$  Buscar(e, k, padre)  $\triangleright \Theta(\log(n) \times \text{order}(k))$ 
3 if lugar  $\neq$  NULL then
4   | (lugar  $\rightarrow$  significado)  $\leftarrow$  s  $\triangleright \Theta(\text{copy}(s))$ 
5 else
6   | puntero(nodo) nuevo  $\leftarrow$  & $\langle k, s, \text{NULL}, \text{NULL}, \text{NULL}, 1 \rangle$  // Reservamos memoria para el nuevo nodo
   |  $\triangleright \Theta(\text{copy}(k) + \text{copy}(s))$ 
7   | if k  $\leq$  (padre  $\rightarrow$  clave) then
8   |   | (padre  $\rightarrow$  izq)  $\leftarrow$  nuevo  $\triangleright \Theta(1)$ 
9   | else
10  |   | (padre  $\rightarrow$  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: $\Theta(\log(n) \times \text{order}(k) + \text{copy}(k) + \text{copy}(s))$

IOBTENER(**in** *e* : **estrAVL**, **in** *k* : κ) $\rightarrow res : \sigma$

```

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

```

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

#CLAVES(**in** *e* : **estrAVL**) $\rightarrow res : \text{nat}$

1 *res* \leftarrow *e.cantNodos* $\triangleright \Theta(1)$

Complejidad: $\Theta(1)$

DEFINIDO?(in e : <i>estrAVL</i> , in k : κ) \rightarrow res : <i>bool</i>		
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: $\Theta(\log(n) \times order(k))$		
IBORRAR(in/out e : <i>estrAVL</i> , in k : κ)		
1 puntero(nodo) $padre \leftarrow \text{NULL}$		$\triangleright \Theta(1)$
2 puntero(nodo) $lugar \leftarrow \text{Buscar}(e, k, padre)$		$\triangleright \Theta(\log(n) \times order(k))$
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)                                                            ▷  $\Theta(1)$ 

```

Complejidad:

IBUSCAR(in e : **estrAVL**, in k : κ , out $padre$: **puntero**(nodo)) $\rightarrow res$: **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  $res \leftarrow actual$   $\triangleright \Theta(1)$ 

```

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

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

```

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

```

Complejidad: $\Theta(1)$

IFBD(in n : **puntero**(nodo)) $\rightarrow res$: **int**

```

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

```

Complejidad: $\Theta(1)$

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

```

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( $n \rightarrow padre$ )  $\triangleright \Theta(1)$ 

```

Complejidad: $\Theta(1)$

IROTARADERECHA(in n : puntero(nodo))	
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($n \rightarrow padre$) $\triangleright \Theta(1)$

Complejidad: $\Theta(1)$

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

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

2. Módulo Heap(α)

Interfaz

parámetros formales

géneros α

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 α .

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 α

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 $\text{vector}(\alpha)$

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

$\text{Rep}(hp) \equiv \text{true} \iff$

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

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

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

Algoritmos

$\text{IVACÍO}() \rightarrow res : \text{heap}(\alpha)$

1 $res \leftarrow \text{vector}(\alpha)$

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

```

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
```

$\text{IVACÍO?}(\text{in } hp : \text{heap}(\alpha)) \rightarrow res : \text{bool}$

1 $res \leftarrow (hp = \text{vacío}())$

$\text{IPROXIMO}(\text{in } hp : \text{heap}(\alpha)) \rightarrow res : \alpha$

1 $res \leftarrow hp[0]$

IDSENCOLAR(**in/out** hp : heap(α))

```

1 nat  $i \leftarrow hp.longitud - 1$ 
2 iSwap( $hp[0]$ ,  $hp[i]$ )
3  $hp.pop\_back()$ 
4 nat  $hijo_0 \leftarrow 1$ 
5 nat  $hijo_1 \leftarrow 2$ 
6 while  $\neg estaOrdenado$  do
7   if  $hijo_0 < hp.longitud$  then
8     if  $hijo_1 < hp.longitud$  then
9       if  $hp[hijo_0] \geq hp[hijo_1]$  then
10        if  $hp[hijo_0] > hp[i]$  then
11          iSwap( $hp[hijo_0]$ ,  $hp[i]$ )
12           $i \leftarrow hijo_0$ 
13        else
14           $estaOrdenado \leftarrow \text{True}$ 
15        end
16      else
17        if  $hp[hijo_1] > hp[i]$  then
18          iSwap( $hp[hijo_1]$ ,  $hp[i]$ )
19           $i \leftarrow hijo_1$ 
20        else
21           $estaOrdenado \leftarrow \text{True}$ 
22        end
23      end
24    else
25      if  $hp[hijo_0] > hp[i]$  then
26        iSwap( $hp[hijo_0]$ ,  $hp[i]$ )
27         $i \leftarrow hijo_0$ 
28      else
29         $estaOrdenado \leftarrow \text{True}$ 
30      end
31    end
32  else
33     $estaOrdenado \leftarrow \text{True}$ 
34  end
35   $hijo_0 \leftarrow 2i+1$ 
36   $hijo_1 \leftarrow 2i+2$ 
37 end

```

IDSENCOLAR2(**in/out** $hp: \text{heap}(\alpha)$) $\rightarrow res : \alpha$

```

1 nat  $i \leftarrow hp.\text{longitud} - 1$ 
2 iSwap( $hp[0]$ ,  $hp[i]$ )
3  $res \leftarrow hp.\text{pop\_back}()$ 
4 nat  $hijo_0 \leftarrow 1$ 
5 nat  $hijo_1 \leftarrow 2$ 
6 while  $\neg \text{estaOrdenado}$  do
7   if  $hijo_0 < hp.\text{longitud}$  then
8     if  $hijo_1 < hp.\text{longitud}$  then
9       if  $hp[hijo_0] \geq hp[hijo_1]$  then
10        if  $hp[hijo_0] > hp[i]$  then
11          iSwap( $hp[hijo_0]$ ,  $hp[i]$ )
12           $i \leftarrow hijo_0$ 
13        else
14           $\text{estaOrdenado} \leftarrow \text{True}$ 
15        end
16      else
17        if  $hp[hijo_1] > hp[i]$  then
18          iSwap( $hp[hijo_1]$ ,  $hp[i]$ )
19           $i \leftarrow hijo_1$ 
20        else
21           $\text{estaOrdenado} \leftarrow \text{True}$ 
22        end
23      end
24    else
25      if  $hp[hijo_0] > hp[i]$  then
26        iSwap( $hp[hijo_0]$ ,  $hp[i]$ )
27         $i \leftarrow hijo_0$ 
28      else
29         $\text{estaOrdenado} \leftarrow \text{True}$ 
30      end
31    end
32  else
33     $\text{estaOrdenado} \leftarrow \text{True}$ 
34  end
35   $hijo_0 \leftarrow 2i+1$ 
36   $hijo_1 \leftarrow 2i+2$ 
37 end

```

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

```

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

```

3. Módulo Red

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: $\Theta(1234)$

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: $\Theta(1324)$

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: $\Theta(1234)$

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: $\Theta(1234)$

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: $\Theta(1234)$

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

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

Complejidad: $\Theta(1234)$

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: $\Theta(1234)$

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: $\Theta(1234)$

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

CAMINOSMINIMOS(**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 .

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: $\Theta(1)???????$

Aliasing: 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: $\Theta(1)???????$

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

Red se representa con **estrRed**

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

donde **estrCompu** es **tupla**(**IP**: **string** , **conexiones**: **lista**(**tupla**(**interfaz**, **itLista**(**estrCompu**)))))

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

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

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

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

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

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

$Rep : itRed \longrightarrow bool$

$Rep(it) \equiv true$

$Abs : itRed \ itl \longrightarrow itUni(estrCompu) \qquad \{Rep(itl)\}$

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

$ArmarComputadoras : lista(tupla<string \times lista(tupla<Interfaz \times ItRed>)>) \longrightarrow conj(compu)$

$ArmarComputadoras(l) \equiv \text{if vacia?}(l) \text{ then } \emptyset$
 else
 $\quad Ag(<\pi_1(prim(l)), GenerarInterfaces(\pi_2(prim(l)))>, ArmarComputadoras(fin(l)))$
 fi

$GenerarInterfaces : lista(tupla<Interfaz \times ItLista(estrCompu)>) \longrightarrow conj(Interfaz)$

$GenerarInterfaces(l) \equiv \text{if vacia?}(l) \text{ then } \emptyset \text{ else } Ag(\pi_1(prim(l)), GenerarInterfaces(fin(l))) \text{ fi}$

$Pertenece? : lista(tupla<string \times lista(tupla<Interfaz \times ItRed>)>) \ l \times compu \ c_1 \times compu \ c_2 \longrightarrow bool$

$Pertenece?(l, c_1, c_2) \equiv \text{if } (\pi_1(prim(l)) = \pi_1(c_1)) \text{ then}$
 $\quad \pi_1(c_2) \in GenerarCompus(\pi_2(prim(l)))$
 else
 $\quad Pertenece?(fin(l), c_1, c_2)$
 fi

$GenerarCompus : lista(tupla<Interfaz \times ItLista(estrCompu)>) \longrightarrow conj(string)$

$GenerarCompus(l) \equiv \text{if vacia?}(l) \text{ then } \emptyset \text{ else } Ag(\pi_1(siguiente(\pi_2(prim(l)))), GenerarCompus(fin(l))) \text{ fi}$

$DevolverInterfaz : lista(tupla<string \times lista(tupla<Interfaz \times ItRed>)>) \ l \times compu \ c_1 \times compu \ c_2 \longrightarrow Interfaz$

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

$DevolverInterfazAux : lista(tupla<Interfaz \times ItRed>) \ l \times compu \ c \longrightarrow Interfaz$

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

$armarCompus : lista(estrCompu) \ l \longrightarrow secu(compu)$

$armarCompus(es) \equiv \text{if vacia}(es) \text{ then } <> \text{ else } armarCompus(prim(es)) \bullet armarCompus(fin(es)) \text{ fi}$

$armarCompu : estrCompu \ e \longrightarrow compu$

$armarCompus(e) \equiv \langle e.IP, dame\Pi_1(e.conecciones) \rangle$

dameII₁ : secu(tupla(inter:interfaz × itCompu:itLista(estrCompu)) l → conj(interfaz)
dameII₁(l) ≡ if vacía(l) then ∅ else ag(Pi₁(prim(l)), dameII₁(fin(l))) fi

Algoritmos

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, itCompu: itLista(estrCompu))))

IVACIA() → res : estrRed

1 res ← ⟨<>, 0⟩ ▷ Θ(1)

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

1 agregarAtras(r.compus, ⟨c.IP, iArmarLista(c.interfaces)⟩)
2 r.cantidadCompus ← r.cantidadCompus + 1 ▷ Θ(1)

IARMARLISTA(in ci: conj(interfaz)) → res : lista(⟨Interfaz, itLista(estrComp)⟩)

1 res ← vacía()
2 itConj(interfaz) it ← crearIt(ci)
3 while haySiguiente(it) do
4 | agregarAtras(res, ⟨siguiente(it), NULL⟩)
5 | avanzar(it)
6 end

ICONECTAR(in/out r: estrRed), in c₀: estrCompu, in i₀: interfaz, in c₁: estrCompu, in i₁: interfaz)

1 itLista(estrComp) it₀ ← crearIt(r.compus)
2 itLista(estrComp) it₁ ← crearIt(r.compus)
3 while siguiente(it₀.IP ≠ c₀.IP) do
4 | avanzar(it₀)
5 end
6 while siguiente(it₁.IP ≠ c₁.IP) do
7 | avanzar(it₁)
8 end
9 itLista(tupla(interfaz, itLista(estrCompu))) it₂ ← crearIt(siguiente(it₀.conexiones))
10 itLista(tupla(interfaz, itLista(estrCompu))) it₃ ← crearIt(siguiente(it₁.conexiones))
11 while siguiente(it₂.inter ≠ i₀) do
12 | avanzar(it₂)
13 end
14 while siguiente(it₃.inter ≠ i₁) do
15 | avanzar(it₃)
16 end
17 siguiente(it₂).itCompu ← i₀ siguiente(it₃).itCompu ← i₁

```
ICREARCONJUNTODEINTERFAZES(in  $l$ : lista(tupla(interfaz, itLista(estrCompu))))  $\rightarrow res$  :
```

```
1 nat  $n \leftarrow 0$ 
2  $res \leftarrow \text{vacío}()$ 
3 while  $n < \text{longitud}(l)$  do
4   |  $\text{agregar}(res, \Pi_1(l[n]))$ 
5   |  $n \leftarrow n + 1$ 
6 end
```

```
ICONECTADAS?(in  $r$ : estrRed, in  $c_0$ : estrCompu, in  $c_1$ : estrCompu)  $\rightarrow res$  : bool
```

```
1 itLista(estrComp)  $it_0 \leftarrow \text{crearIt}(r.\text{compus})$ 
2 while siguiente( $it_0$ ).IP  $\neq c_0$ .IP do
3   |  $\text{avanzar}(it_0)$ 
4 end
5 itLista(tupla(interfaz, itLista(estrCompu)))  $it_1 \leftarrow \text{crearIt}(\text{siguiente}(it_0.\text{conexiones}))$ 
6 while haySiguiente( $it_1$ )  $\wedge_L$  siguiente(siguiente( $it_1$ ).itCompu).IP  $\neq c_1$ .IP do
7   |  $\text{avanzar}(it_1)$ 
8 end
9  $res \leftarrow (\text{siguiente}(\text{siguiente}(it_1).\text{itCompu}).IP = c_1.IP)$ 
```

```
IINTERFAZUSADA(in  $r$ : estrRed, in  $c_0$ : estrCompu, in  $c_1$ : estrCompu)  $\rightarrow res$  : interfaz
```

```
1 itLista(estrComp)  $it_0 \leftarrow \text{crearIt}(r.\text{compus})$ 
2 while siguiente( $it_0$ ).IP  $\neq c_0$ .IP do
3   |  $\text{avanzar}(it_0)$ 
4 end
5 itLista(tupla(interfaz, itLista(estrCompu)))  $it_1 \leftarrow \text{crearIt}(\text{siguiente}(it_0.\text{conexiones}))$ 
6 while haySiguiente( $it_1$ )  $\wedge_L$  siguiente(siguiente( $it_1$ ).itCompu).IP  $\neq c_1$ .IP do
7   |  $\text{avanzar}(it_1)$ 
8 end
9  $res \leftarrow \text{siguiente}(it_1).\text{inter}$ 
```

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

```
1  $res \leftarrow \text{vacío}()$ 
2 itLista(estrComp)  $it_0 \leftarrow \text{crearIt}(r.\text{compus})$ 
3 while siguiente( $it_0$ ).IP  $\neq c$ .IP do
4   |  $\text{avanzar}(it_0)$ 
5 end
6 itLista(tupla(interfaz, itLista(estrCompu)))  $it_1 \leftarrow \text{crearIt}(\text{siguiente}(it_0.\text{conexiones}))$ 
7 while haySiguiente?( $it_1$ ) do
8   | if haySiguiente?(siguiente( $it_1$ ).itCompu) then
9     |  $\text{agregar}(res, \text{siguiente}(\text{siguiente}(it_1).\text{itCompu}))$ 
10  | end
11 |  $\text{avanzar}(it_1)$ 
12 end
```

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

```

1 itLista(estrComp)  $it_0 \leftarrow$  crearIt(r.compus)
2 while siguiente( $it_0$ ).IP  $\neq c$ .IP do
3   | avanzar( $it_0$ )
4 end
5 itLista(tupla(interfaz, itLista(estrCompu)))  $it_1 \leftarrow$  crearIt(siguiente( $it_0$ .conexiones))
6 while siguiente( $it_1$ ).inter  $\neq i$  do
7   | avanzar( $it_1$ )
8 end
9  $res \leftarrow$  haySiguiente?(siguiente( $it_1$ ).itCompu)
```

ICAMINOSMINIMOS(in r : estrRed, in c_0 : estrCompu, in c_1 : estrCompu) $\rightarrow res$: conj(lista(estrCompu))

```

1  $res \leftarrow$  vacio()
2 if pertenece( $c_1$ , vecinos( $r$ ,  $c_1$ )) then
3   | agregar( $res$ , agregarAtras(agregarAtras( $\langle \rangle$ ,  $c_0$ ),  $c_1$ ))
4 else
5   |  $res \leftarrow$  dameMinimos(iCaminos( $r$ ,  $c_0$ ,  $c_1$ , agregarAtras( $\langle \rangle$ ,  $c_0$ ), pasarConjASecu(vecinos( $r$ ,  $c_0$ ))))
6 end
```

IHAYCAMINOS?(in r : estrRed, in c_0 : estrCompu, in c_1 : estrCompu) $\rightarrow res$: bool

```

1  $res \leftarrow$  esVacio?(iCaminosMinimos( $r$ ,  $c_0$ ,  $c_1$ ))
```

ICAMINOS(in r : estrRed, in c_0 : estrCompu, in c_1 : estrCompu, in l : lista(estrCompu), in vec : lista(estrCompu)) $\rightarrow res$: conj(lista(estrCompu))

```

1 if vacia?( $vec$ ) then
2   |  $res \leftarrow$  vacia()
3 else
4   | if /último( $l$ ) =  $c_1$  then
5     |  $res \leftarrow$  agregar( $l$ , vacia())
6   | else
7     | if está?(primero( $vec$ ,  $l$ )) then
8       |  $res \leftarrow$  unión(caminos( $r$ ,  $c_0$ ,  $c_1$ , agregarAtras( $l$ , primero( $vec$ )), iVecinos( $r$ , primeros( $vec$ ))),
9         | 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
```

ICOMPUTADORAS(in r : estrRed) $\rightarrow res$: conj(estrCompu)

```

1  $res \leftarrow$  vacio()
2 itRed  $it \leftarrow$  crearItRed()
3 while haySiguiente?( $it$ ) do
4   | agregar( $res$ , siguiente( $it$ ))
5   | avanzar( $it$ )
6 end
```

IHAYSIGUIENTE?(in it : itLista(estrCompu)) $\rightarrow res$: bool

```

1  $res \leftarrow$  haySiguiente?( $it$ )
```

ISIGUIENTE(**in** *it*: itLista(estrCompu)) → *res* : compu

```
1  estrCompu e ← siguiente(it)
2  res.IP ← e.IP
3  conoj(interfaz) interfaces ← vacío()
4  itLista(tupla( inter: interfaz, itCompu: itLista(estrCompu))) itInterfaces ←
   crearIt(e.conexiones)
5  while haySiguiente?(itInterfaces) do
6    |  agregar(interfaces, siguiente(itInterfaces).inter)
7    |  avanzar(itInterfaces)
8  end
9  res.Interfaces ← e.Interfaces
```

ICREARIT(**in** *e*: estrRed)

```
1  res ← crearIt(e.compus)
```

IAVANZAR(**in/out** *it*: itLista(estrCompu))

```
1  it ← avanzar(it)
```

4. Módulo DCNet

Interfaz

se explica con: DCNet

géneros: dcnet

Operaciones básicas de lista

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

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

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

Complejidad: $\Theta(1234)$

Descripción: Genera un nuevo DCNet sin paquetes. “vacía”

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

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** $D : \text{DCNet}$)

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** $D : \text{DCNet}$) $\rightarrow res : \text{red}$

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

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

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** $D : \text{DCNet}$, **in** $c : \text{compu}$) $\rightarrow res : \text{nat}$

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

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** $D : \text{DCNet}$, **in** $p : \text{paquete}$) $\rightarrow res : \text{bool}$

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

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

Complejidad: $\Theta(1234)$

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

LAQUEMÁSENVIÓ(**in** $D : \text{DCNet}$) $\rightarrow res : \text{compu}$

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_tribe(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 \rightarrow bool`

`Rep(l) \equiv true \iff ($\forall e$: estr)(`
`(tam(e.paquetesEnEspera) = tam(e.IPcompusXID) = tam(e.siguienteCompu) = tam`
`(e.cantPaquetesEnviados) = #(computadoras(e.estrRed))) \wedge ($\forall n$: nat)(definido?(e.siguienteCompu,n) \Rightarrow_L`
`tam(e.siguienteCompu[n]) = #(computadoras(e.estrRed))) \wedge`
`Maximo(e.cantPaqEnviados) = e.cantidadEnviados[obtener(π_1 (siguiente(e.laQueMásEnvió),e.IDcompusPorIP)]`
 `\wedge ($\forall c$: compu)(c \in computadoras(e.estrRed) \Rightarrow_L obtener(π_1 (c),e.IDcompusPorIP) $<_{\#}$ (computado-`
`ras(e.estrRed))) \wedge (($\forall c_1, c_2$: compu)(($c_1 \in$ computadoras(e.estrRed) \wedge ($c_2 \in$ computadoras(e.estrRed)) \wedge`
`($c_1 \neq c_2$) \Rightarrow_L ((obtener(π_1 (c_1), e.IDcompusPorIP) \neq obtener(π_1 (c_2), e.IDcompusPorIP)))))) \wedge`
`(dameNombres(computadoras(e.estrRed)) = claus(IDcompusPorIP)) \wedge`
`($\forall L$: nat)(0 $\leq L <$ tam(e.paqEnEspera) \Rightarrow_L (`
`($\forall it_1$: ItConj(paquete)) $it \in$ dame π_1 (juntarSignificado(π_2 (e.paquetesEnEspera[L]))) \Rightarrow_L haySiguiente?(it_1)`
 `\wedge ($\forall it_2$: ITconj(pauquete)) $it_2 \in$ dame π_2 (juntarColaEnConjunto(π_3 (e.paquetesEnEspera[L]))) \Rightarrow_L`
`haySiguiente?(it_2) \wedge_L`
`($\forall i$: ItConj(paquete))(siguiente(i) \in (dameSiguietes(dame π_1 (juntarSignificados(π_2 (e.paquetesEnEspera[L])))`
`siguiente(i) \in π_1 (e.paquetesEnEspera[L]))) \wedge ($\forall c$: ID)(c \in claves(π_2 (e.paquetesEnEspera[L])))`
 `\Rightarrow_L π_1 (siguiente(π_1 (obtener(π_2 (c), e.paquetesEnEspera[L]))) = c) \wedge ($\forall it$: ItConj(paquete))`
`siguiente(it) \in dameSiguietes(dame π_2 (juntarColaPrioriEnConj(π_3 (e.paquetesEnEspera[L])))`
 `\Rightarrow_L siguiente(it) \in π_1 (e.paquetesEnEspera[L]) ($\forall t$: tupla $<$ prioridad,ItConj(paquete) $>$) $t \in$`
`juntarColaPrioriEnConj(π_3 (e.paquetesEnEspera[L])) \rightarrow siguiente(π_2 (t)).prioridad = π_1 (t)`
`($\forall x, z$: nat)((0 $\leq x <$ tam(e.paquetesEnEspera) \wedge 0 $\leq z <$ tam(e.paquetesEnEspera) \wedge $x \neq z$) \Rightarrow_L`
`(π_1 (e.paquetesEnEspera[x]) \cap π_2 (e.paquetesEnEspera[z])) = \emptyset) \wedge`
`($\forall i$: nat)(0 $\leq i <_{\#}$ (computadoras(e.estrRed)) \wedge obtener(π_1 (siguiente(e.IPcompusPorID[i])),`
`e.IDcompusPorID) = i) \wedge`
`($\forall n, m$: nat)(0 $\leq n <_{\#}$ (computadoras(e.estrRed)) \wedge 0 $\leq m <_{\#}$ (computadoras(e.estrRed))`
 `\Rightarrow_L ($\exists x$: (secu(compu))) $x \in$ caminosminimos(e.estrRed, siguiente(e.IPcompusPorID[n]), siguien-`
`te(e.IPcompusPorID[m])) \wedge prim(x) = e.siguienteCompu[n][m]`
`($\forall i$: nat)(0 $\leq i \leq$ #computadoras(e.estrRed) \Rightarrow_L siguiente(e.IPcompusPorID[i]) \in compitadoras(e.estrRed))`
 `\wedge`
`($\forall x, y$: nat)((0 $\leq x <_{\#}$ computadoras(e.estrRed) \wedge 0 $\leq y <_{\#}$ computadoras(e.estrRed) \wedge $x \neq y$) \Rightarrow_L`
`(siguiente(e.IPcompusPorID[x]) \neq siguiente(e.IPcompusPorID[y]))) \wedge`
`#(π_1 (e.paquetesEnEspera)) = #claves(π_2 (e.paquetesEnEspera)) \wedge = #(juntarSecuenciasEnConj(juntarSignificad`
 `\wedge`
`($\forall it_1$: ItConj(paquete)) ($\forall it_2$: ItConj(paquete)) $it_1 \in$ dame π_2 (juntarColaPrioriEnConj(π_3 (e.paquetesEnEspera[L]))`
 `\wedge $it_2 \in$ dame π_2 (juntarColaPrioriEnConj(π_3 (e.paquetesEnEspera[L]))) $it_1 \neq it_2 \Rightarrow_L$ siguiente(it_1) \neq`
`siguiente(it_2)`

`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

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

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

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

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

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

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

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

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

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

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

```

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

```

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

```

IAVANZARSEGUNDO(in/out *d*: DCNet)

```

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

```

ICAMINORECORRIDO(in *d*: DCNet, in *p*: paquete) → *res*: lista(compu)

```

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

```

ICANTIDADENVIADOS(in *d*: DCNet, in *c*: compu) → *res*: nat

```

1 nat i ← iObtener(d.IDsCompusPorIP, c.IP)
2 res ← d.#paqEnviados[i]

```

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

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