

# Algoritmos y Estructuras de Datos II

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

## Trabajo Práctico II

### Grupo: 12

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

### Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

# Índice

<b>1. Módulo DCNet</b>	<b>3</b>
1.1. Interfaz . . . . .	3
1.1.1. Operaciones básicas de mapa . . . . .	3
1.2. Representación . . . . .	3
1.2.1. Representación de dcnet . . . . .	3
1.2.2. Invariante de Representación . . . . .	3
<b>2. Módulo Red</b>	<b>7</b>
2.1. Interfaz . . . . .	7
2.2. Representación . . . . .	8
2.2.1. Estructura . . . . .	8
2.2.2. Invariante de Representación . . . . .	8
2.2.3. Función de Abstracción . . . . .	8
2.2.4. Función de Abstracción . . . . .	8
2.3. Algoritmos . . . . .	8

# 1. Módulo DCNet

## 1.1. Interfaz

se explica con: DCNET.

géneros: dcnet.

### 1.1.1. Operaciones básicas de mapa

CREAR()  $\rightarrow res : dcnet$

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

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

**Complejidad:**  $O(1)$

**Descripción:** crea un mapa nuevo

## 1.2. Representación

### 1.2.1. Representación de dcnet

dcnet se representa con estr

donde estr es tupla(*topología*: red,  
                           *vectorCompusDCNet*: vector(compuDCNet),  
                           *diccCompusDCNet*: dicc<sub>trie</sub>(puntero(compuDCNet)),  
                           *laQueMásEnvió*: puntero(compuDCNet))

donde compuDCNet es tupla(*pc*: puntero(compu),  
                               *conjPaquetes*: conj(paquete),  
                               *diccPaquetesDCNet*: dicc<sub>avl</sub>(nat, paqueteDCNet),  
                               *colaPaquetesDCNet*: colaPrioridad(nat, puntero(paqueteDCNet)),  
                               *paqueteAEnviar*: paqueteDCNet, *enviados*: nat)

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

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

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

### 1.2.2. Invariante de Representación

- (I) Las compus de los elementos de vectorCompusDCNet son punteros a todas las compus de la topología
- (II) Las claves de diccCompusDCNet son todos los hostnames de la topología
- (III) Los significados de diccCompusDCNet son punteros que apuntan a las compuDCNet cuyo hostname equivale a su clave en vectorCompusDCNet
- (IV) laQueMásEnvió es un puntero a la compuDCNet en vectorCompusDCNet que más paquetes enviados tiene. Si no hay compus es NULL
- (V) Todos los paquetes en conjPaquetes de cada compuDCNet tienen id único y tanto su origen como destino existen en la topología
- (VI) El paquete en conjPaquetes tiene que tener en su recorrido a la compuDCNet en la que se encuentra y esta no puede ser igual al destino del recorrido

- (VII) Las claves de `diccPaquetesDCNet` son los id de los paquetes en `conjPaquetes`
- (VIII) Los significados de `diccPaquetesDCNet` contienen un `itConj` que apunta al paquete con el id equivalente a su clave y en recorrido, un camino mínimo válido para el origen del paquete y la compu en la que se encuentra
- (IX) La `colaPaquetesDCNet` es vacía si y sólo si `conjPaquetes` lo es, si no lo es, su próximo es un puntero a un paqueteDCNet de `diccPaquetesDCNet` que contiene un `itConj` cuyo siguiente es uno de los paquetes de `conjPaquetes` con mayor prioridad
- (X) La cantidad de enviados de una `compuDCNet` es igual o mayor a la cantidad de apariciones de esa compu en los caminos recorridos de paquetes en la red

`Rep : estr  $\rightarrow$  bool`

`Rep(e)  $\equiv$  true  $\iff$`

```

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

```

$\text{compuQueMásEnvió} : \text{secu}(\text{compuDCNet}) \text{ } scd \longrightarrow \text{compu} \quad \{\neg \text{vacía?}(scd)\}$   
 $\text{maxEnviado} : \text{secu}(\text{compuDCNet}) \text{ } scd \longrightarrow \text{nat} \quad \{\neg \text{vacía?}(scd)\}$   
 $\text{enviaronK} : \text{secu}(\text{compuDCNet}) \times \text{nat} \longrightarrow \text{conj}(\text{compu})$   
 $\text{paqueteMásPrioridad} : \text{conj}(\text{paquete}) \text{ } cp \longrightarrow \text{paquete} \quad \{\neg \emptyset?(cp)\}$   
 $\text{paquetesConPrioridadK} : \text{conj}(cp) \times \text{nat} \longrightarrow \text{conj}(\text{paquete})$   
 $\text{altaPrioridad} : \text{conj}(\text{paquetes}) \text{ } cp \longrightarrow \text{nat} \quad \{\neg \emptyset?(cp)\}$   
 $\text{enviadosCompu} : \text{compu} \times \text{secu}(\text{compuDCNet}) \longrightarrow \text{nat}$   
 $\text{aparicionesCompu} : \text{compu} \times \text{conj}(\text{nat}) \text{ } cn \times \text{dicc}(\text{nat} \times \text{paqueteDCNet}) \text{ } dp \longrightarrow \text{nat} \quad \{\text{claves}(dp) \subseteq cn\}$

$\text{compuQueMásEnvió}(scd) \equiv \text{dameUno}(\text{enviaronK}(scd, \text{maxEnviado}(scd)))$   
 $\text{maxEnviado}(scd) \equiv \text{if } \text{vacía?}(\text{fin}(scd)) \text{ then } \text{prim}(scd).\text{enviados} \text{ else } \text{max}(\text{prim}(scd), \text{maxEnviado}(\text{fin}(scd))) \text{ fi}$   
 $\text{enviaronK}(scd, k) \equiv \text{if } \text{vacía?}(scd) \text{ then}$   
 $\quad \emptyset$   
 $\text{else}$   
 $\quad \text{if } \text{prim}(scd).\text{enviados} = k \text{ then}$   
 $\quad \quad \text{Ag}(*(\text{prim}(scd).\text{pc}), \text{enviaronK}(\text{fin}(scd), k))$   
 $\quad \text{else}$   
 $\quad \quad \text{enviaronK}(\text{fin}(scd), k)$   
 $\quad \text{fi}$   
 $\text{fi}$   
 $\text{paqueteMásPrioridad}(dcn, cp) \equiv \text{dameUno}(\text{paquetesConPrioridadK}(cp, \text{altaPrioridad}(cp)))$   
 $\text{altaPrioridad}(cp) \equiv \text{if } \emptyset?(\text{sinUno}(cp)) \text{ then}$   
 $\quad \text{dameUno}(cp).\text{prioridad}$   
 $\text{else}$   
 $\quad \text{min}(\text{dameUno}(cp).\text{prioridad}, \text{altaPrioridad}(\text{sinUno}(cp)))$   
 $\text{fi}$   
 $\text{paquetesConPrioridadK}(cp, k) \equiv \text{if } \emptyset?(cp) \text{ then}$   
 $\quad \emptyset$   
 $\text{else}$   
 $\quad \text{if } \text{dameUno}(cp).\text{prioridad} = k \text{ then}$   
 $\quad \quad \text{Ag}(\text{dameUno}(cp), \text{paquetesConPrioridadK}(\text{sinUno}(cp), k))$   
 $\quad \text{else}$   
 $\quad \quad \text{paquetesConPrioridadK}(\text{sinUno}(cp), k)$   
 $\quad \text{fi}$   
 $\text{fi}$   
 $\text{enviadosCompu}(c, scd) \equiv \text{if } \text{vacía?}(scd) \text{ then}$   
 $\quad 0$   
 $\text{else}$   
 $\quad \text{if } \text{prim}(scd) = c \text{ then}$   
 $\quad \quad \text{enviadosCompu}(c, \text{fin}(scd))$   
 $\quad \text{else}$   
 $\quad \quad \text{aparicionesCompu}(c, \text{claves}(\text{prim}(scd).\text{diccPaquetesDCNet}),$   
 $\quad \quad \text{prim}(scd).\text{diccPaquetesDCNet}) + \text{enviadosCompu}(c, \text{fin}(scd))$   
 $\quad \text{fi}$   
 $\text{fi}$

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

## 2. Módulo Red

### 2.1. Interfaz

**se explica con:** RED.

**géneros:** red.

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

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

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

**COMPUTADORAS(in  $r : red$ )**  $\rightarrow res : conj(compu)$   
**Pre**  $\equiv \{true\}$   
**Post**  $\equiv \{res = computadoras(r)\}$   
**Complejidad:**  $O(1)$

**CONECTADAS?(in  $r : red$ , in  $c : compu$ , in  $c' : compu$ )**  $\rightarrow res : bool$   
**Pre**  $\equiv \{(c \in computadoras(r)) \wedge (c' \in computadoras(r))\}$   
**Post**  $\equiv \{res = conectadas?(r, c, c')\}$   
**Complejidad:**  $O(1)$

**INTERFAZUSADA(in  $r : red$ , in  $c : compu$ , in  $c' : compu$ )**  $\rightarrow res : interfaz$   
**Pre**  $\equiv \{conectadas?(r, c, c')\}$   
**Post**  $\equiv \{res = interfazUsada(r, c, c')\}$   
**Complejidad:**  $O(?)$

**VECINOS(in  $r : red$ , in  $c : compu$ )**  $\rightarrow res : conj(compu)$   
**Pre**  $\equiv \{c \in computadoras(r)\}$   
**Post**  $\equiv \{res = vecinos(r, c)\}$   
**Complejidad:**  $O(n)$

**USAINTERFAZ?(in  $r : red$ , in  $c : compu$ , in  $i : interfaz$ )**  $\rightarrow res : bool$   
**Pre**  $\equiv \{c \in computadoras(r)\}$   
**Post**  $\equiv \{res = usaInterfaz?(r, c, i)\}$   
**Complejidad:**  $O(?)$

CAMINOSMINIMOS(**in**  $r$ : red, **in**  $c$ : compu, **in**  $c'$ : compu)  $\rightarrow res$  : conj(secu(compu))  
**Pre**  $\equiv \{(c \in computadoras(r)) \wedge (c' \in computadoras(r))\}$   
**Post**  $\equiv \{res = caminosMinimos(r, c, i)\}$   
**Complejidad:**  $O(L)$

HAYCAMINO?(**in**  $r$ : red, **in**  $c$ : compu, **in**  $c'$ : compu)  $\rightarrow res$  : bool  
**Pre**  $\equiv \{(c \in computadoras(r)) \wedge (c' \in computadoras(r))\}$   
**Post**  $\equiv \{res = hayCamino?(r, c, i)\}$   
**Complejidad:**  $O(L)$

## 2.2. Representación

### 2.2.1. Estructura

red se representa con **estr**

donde **estr** es tupla( $compus$ : conj(compu) ,  
 $dns$ : dicc<sub>Trie</sub>(ip, nodoRed) )

donde **nodoRed** es tupla( $c$ : puntero(compu) ,  
 $caminos$ : dicc<sub>Trie</sub>(ip, conj(lista(compu))) ,  
 $conexiones$ : dicc<sub>Lineal</sub>(interfaz, compu) )

### 2.2.2. Invariante de Representación

- (I) Todas las compus deben tener IPs distintas.
- (II) Ninguna compu se conecta con si misma.
- (III) Ninguna compu se conecta a otra a traves de dos interfaces distintas.
- (IV) El trie **estr.dns** apunta a un **nodoRed** por cada elemento de *compus*.
- (V) En cada **nodoRed**,  $c$  tiene que apuntar a un elemento de **estr.compus**.
- (VI) Para cada **nodoRed**, *caminos* tiene como claves todas las IPs de las compus de la red, y los significados corresponden a todos los caminos mínimos desde la compu  $c$  hacia la compu cuya IP es clave.
- (VII) **nodoRed.conexiones** contiene como claves todas las **interfaz** usaconedadas de la compu  $c$  (que tienen que estar en  $c.interfaces$ )

Rep : **estr**  $\rightarrow$  bool

Rep( $e$ )  $\equiv$  true  $\iff$

### 2.2.3. Función de Abstracción

Abs : **estr**  $e \rightarrow$  red

{Rep( $e$ )}

Abs( $e$ ) =<sub>obs</sub>  $r$ : red |

### 2.2.4. Función de Abstracción

## 2.3. Algoritmos



```
iIniciarRed () → res: red
  res.compus ← Vacio ()
  res.dns ← Vacio ()
Complejidad :  $O(?)$ 
```

```
iAgregarComputadora (in/out r: red, in c: compu) → res: itConj
  Definir (r.dns, compu.ip, Tupla(&c, Vacio(), Vacio()))
  res ← AgregarRapido(r.compus, c)
Complejidad :  $O(?)$ 
```

```
iConectar (in/out r: red, in c0: compu, in c1: compu, in i0: compu, in i1: compu)
  pnr0: puntero(nodoRed) ← Significado(r.dns, c0.ip)
  pnr1: puntero(nodoRed) ← Significado(r.dns, c1.ip)
  DefinirRapido(pnr0→conexiones, i0, *(pnr1→c))
  DefinirRapido(pnr1→conexiones, i1, *(pnr0→c))
  CrearCaminosDelDNS(r)
Complejidad :  $O(?)$ 
```

```

CrearCaminosDelDNS (in/out r: red)
  itCompus: it Conj (compu)  $\leftarrow$  CrearIt (r.compus)
  while HaySiguiente?(itCompus) do
    nr: nodoRed  $\leftarrow$  Significado(d, Siguiente(itCompus).ip)
    AsignarCaminosMinimos(nr, r)
    Avanzar(itCompus)
  end while
Complejidad :  $O(?)$ 

```

```

AsignarCaminosMinimos (in/out nr: nodoRed, , in r: red)
  itCompus: it Conj (compu)  $\leftarrow$  CrearIt (r.compus)
  while HaySiguiente?(itCompus) do
    Definir(nr.caminos, Siguiente(itCompus).ip, r)
    CrearCaminosMinimos(nr, Siguiente(itCompus).ip))
    Avanzar(itCompus)
  end while
Complejidad :  $O(?)$ 

```

```

CrearCaminosMinimos (in desde: nodoRed, in r: red, in hasta: compu)  $\rightarrow$  res: conj(lista(compu))
  camMins: conj ( lista (compu) )  $\leftarrow$  Vacio ()
  itCamMins: it Conj (conj ( lista (compu) ) )  $\leftarrow$  CrearIt (camMins)
  pendientes: conj (nodoRed)  $\leftarrow$  Vecinos(r.desde.ip)
  visitados: conj (nodoRed)  $\leftarrow$  Vacio ()
  itNrs: it Conj (nodoRed)  $\leftarrow$  CrearIt (pendientes)
  cam: lista (compu)  $\leftarrow$  Vacia ()
  AgregarAdelante(cam, *(desde.pc))
  while HaySiguiente?(itNrs) do
    AgregarRapido(visitados, Siguiente(itNrs))
    AgregarAdelante(cam, *(Siguiente(itNrs).pc))
    if (Siguiente(itNrs).pc  $\rightarrow$  ip = hasta.ip) then
      if (Vacio?(camMins)  $\vee$  (Longitud(cam) = Longitud(Siguiente(itCamMins)))) then
        AgregarRapido(camMins, cam)
      else
        if (Longitud(cam) < Longitud(Siguiente(itCamMins))) then
          camMins  $\leftarrow$  Vacio ()
          AgregarRapido(camMins, cam)
        endif
      endif
      Comienzo(cam)
    else
      if (Vacía?(NoVisitadas(pendientes, Siguiente(itNrs), visitados, r)))
        Comienzo(cam)
      else
        UnirConjs(pendientes, NoVisitadas(pendientes,
          Siguiente(itNrs), visitados, r))
      end if
    end if
    Avanzar(itNrs)
  end while
Complejidad :  $O(?)$ 

```

```

NoVisitadas (in pendientes: conj(nodoRed), in nr: nodoRed, in visitados: conj(nodoRed), in r: red) → res:
conj(nodoRed)
  news: conj(nodoRed) ← Vecinos(r, nr.pc → ip)
  itVecinos: itConj(nodoRed) ← CrearIt(news)
  res ← Vacio()
  while HaySiguiente?(itVecinos) do
    if (¬Pertenece?(visitados, Siguiente(itVecinos))) then
      AgregarRapido(res, Siguiente(itVecinos))
    end if
    Avanzar(itVecinos)
  end while
Complejidad :  $O(?)$ 

```

```

iComputadoras (in r: red) → res: conj(compu)
  res ← r.computadoras Complejidad :  $O(1)$ 
Complejidad :  $O(1)$ 

```

```

iConectadas? (in r: red, in c0: compu, in c1: compu) → res: bool
  pnr0: puntero(nodoRed) ← Significado(r.dns, c0.ip)  $O(L)$ 
  it : itDicc(interfaz, puntero(nodoRed)) ← CrearIt(pnr0.conexiones)  $O(1)$ 
  res ← false  $O(1)$ 
  while HaySiguiente?(it) do  $O(L)$ 
    if c1.ip = SiguienteSignificado(it).ip then  $O(?)$ 
      res ← true  $O(1)$ 
    end if
    Avanzar(it)  $O(1)$ 
  end while
Complejidad :  $O(L*?)$ 

```

```

iInterfazUsada (in r: red, in c0: compu, in c1: compu) → res: interfaz
  pnr0: puntero(nodoRed) ← Significado(r.dns, c0.ip)  $O(L)$ 
  it : itDicc(itDicc(interfaz, puntero(nodoRed))
    ← CrearIt(pnr0.conexiones)  $O(1)$ 
  while HaySiguiente?(it) do  $O(L)$ 
    if c1.ip = SiguienteSignificado(it).ip then  $O(?)$ 
      res ← SiguienteClave(it)  $O(1)$ 
    end if
    Avanzar(it)  $O(1)$ 
  end while
Complejidad :  $O(1)$ 

```

```

iVecinos (in r: red, in c: compu) → res: conj(compu)
  pnr: puntero(nodoRed) ← Significado(r.dns, c.ip)  $O(1)$ 
  res ← pnr → conexiones  $O(1)$ 
Complejidad :  $O(1)$ 

```

```

iUsaInterfaz? (in r: red, in c: compu, in i: interfaz) → res: bool
  pnr: puntero(nodoRed) ← Significado(r.dns, c.ip)  $O(1)$ 
  res ← Definido?(pnr → conexiones, i)  $O(n)$ 

```

**Complejidad :**  $O(n)$

```
iCaminosMinimos (in r: red, in c0: compu, in c1: compu) → res: conj(secu(compu))  
  pnr: puntero(nodoRed) ← Significado(r.dns, c0.ip) O(1)  
  res ← Significado(pnr→caminosMinimos, c1.ip) O(1)  
Complejidad :  $O(1)$ 
```

```
HayCamino? (in r: red, in c0: compu, in c1: compu) → res: bool  
  pnr: puntero(nodoRed) ← Significado(r.dns, c0.ip) O(1)  
  res ← EsVacio?(Significado(pnr→caminos, c1.ip))  
Complejidad :  $O(1)$ 
```

```
copiar (in r: red) → res: red  
  auxR: red ← iIniciarRed  
  auxR.dns ← r.dns  
  res ← auxR  
Complejidad :  $O(1)$ 
```

```
• = • (in r0: red, in r1: red) → res: bool  
  eq: bool ← (r0.compus = r1.compus)  
  eq ← (r0.dns = r1.dns)  
  res ← eq  
Complejidad :  $O(1)$ 
```