

Algoritmos y Estructuras de Datos II

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

Trabajo Práctico II

Grupo: 12

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

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Índice

1. Módulo DCNet	3
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
2. Módulo Red	7
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_{trie}(puntero(compuDCNet)),
 laQueMásEnvió: puntero(compuDCNet))

donde compuDCNet es tupla(*pc*: puntero(compu),
 conjPaquetes: conj(paquete),
 diccPaquetesDCNet: dicc_{avl}(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`

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

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

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

```
aparicionesCompu(c, cn, dpc)  $\equiv$  if  $\emptyset?(cn)$  then  
    0  
else  
    if está?(c, significado(dameUno(cn), dpc).recorrido) then  
        1 + aparicionesCompu(c, sinUno(cn), dpc)  
    else  
        aparicionesCompu(c, sinUno(cn), dpc)  
    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: $O(n)$

USAIINTERFAZ?(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 $\text{tupla}(\text{compus: conj}(\text{compu}),$
 $\text{dns: dicc}_{Trie}(\text{ip}, \text{nodoRed}))$

donde **nodoRed** es $\text{tupla}(c: \text{puntero}(\text{compu}),$
 $\text{caminos: dicc}_{Trie}(\text{ip}, \text{conj}(\text{lista}(\text{compu}))) ,$
 $\text{conexiones: dicc}_{Lineal}(\text{interfaz}, \text{compu}))$

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 $c.interfaces$)

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

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

2.2.3. Función de Abstracción

$\text{Abs} : \text{estr } e \rightarrow \text{red}$

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

$\text{Abs}(e) =_{\text{obs}} r : \text{red} \mid$

2.2.4. Función de Abstracción

2.3. Algoritmos


```

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

```

```

iAgregarComputadora (in/out r: red, in c: compu)
  AgregoCompuNuevaAlResto(r.dns, c)
  AgregarRapido(r.compus, c)
  Definir(r.dns, compu.ip, Tupla(&c, Vacio(), Vacio()) >)
  InicializarConjCaminos(r, c)
Complejidad :  $O(n * L)$ 

```

```

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

```

```

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

```

```

iConectar (in/out r: red, in c0: compu, in c1: compu, in i0: compu, in i1: compu)
  nr0:nodoRed ← Significado(r.dns, c0.ip)
  nr1:nodoRed ← Significado(r.dns, c1.ip)
  DefinirRapido(nr0.conexiones, i0, nr1)
  DefinirRapido(nr1.conexiones, i1, nr0)
  CrearCaminosDelDNS(r)
Complejidad :  $O(n! * (n^4))$ 

```

```

CrearCaminosDelDNS (in/out r: red)
  itCompus: itConj(compu) ← CrearIt(r.compus)
  while HaySiguiente?(itCompus) do
    nr:nodoRed ← Significado(d, Siguiente(itCompus).ip)
    AsignarCaminosMinimos(nr, r)
    Avanzar(itCompus)
  end while
Complejidad :  $O(n! * (n^4))$ 

```

```

AsignarCaminosMinimos (in/out nr: nodoRed)
  itCompus: itConj(compu) ← CrearIt(r.compus)
  while HaySiguiente?(itCompus) do
    Definir(nr.camino, Siguiente(itCompus).ip)
    CrearCaminosMinimos(nr, Siguiente(itCompus).ip)
    Avanzar(itCompus)
  end while
Complejidad :  $O(n! * (n^3))$ 

```

```

CrearCaminosMinimos (in desde: nodoRed, in hasta: compu) → res: conj(lista(compu))
  camMins: conj(lista(compu)) ← Vacio()
  itCamMins: itConj(conj(lista(compu))) ← CrearIt(camMins)
  pendientes: pila(nodoRed) ← Vacia()
  ApilarConj(pendientes, Conexas(desde.conexiones))
  visitados: conj(nodoRed) ← Vacio()
  cam: lista(compu) ← Vacia()
  AgregarAdelante(cam, *(desde.pc))
  while ¬EsVacía?(pendientes) do
    AgregarRapido(visitados, Siguiente(itNrs))
    AgregarAdelante(cam, *(Siguiente(itNrs).pc))
    if (Tope(pendientes).pc → ip = hasta.ip) then
      if (Vacío?(camMins) ∨ (Longitud(cam) = Longitud(Siguiente(itCamMins)))) then
        AgregarRapido(camMins, cam)
      else
        if (Longitud(cam) < Longitud(Siguiente(itCamMins))) then
          camMins ← Vacio()
          AgregarRapido(camMins, cam)
        endif
      endif
      Comienzo(cam)
    end if
    Desapilar(pendientes)
    nv: conj(nodoRed) ← NoVisitadas(Tope(pendientes), visitados)
    if (Vacía?(nv))
      Comienzo(cam)
    else
      ApilarConj(pendientes, nv)
    end if
  end while
Complejidad :  $O(n! * (n^2))$ 

```

```

NoVisitadas (in nr: nodoRed, in visitados: conj(nodoRed)) → res: conj(nodoRed)
  res ← Vacio()
  itVecinos: itConj(nodoRed) ← CrearIt(Conexas(nr.conexiones))
  while HaySiguiente?(itVecinos) do
    if (¬Pertenece?(visitados, Siguiente(itVecinos))) then
      AgregarRapido(res, Siguiente(itVecinos))
    end if
    Avanzar(itVecinos)
  end while
Complejidad :  $O(n^2)$ 

```

```

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

```

```

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

```

```

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

```

```

iConectadas? (in r: red, in c0: compu, in c1: compu) → res: bool
  nr0:nodoRed ← Significado(r.dns , c0.ip)
  it :itDicc(interfaz , puntero(nodoRed))) ← CrearIt(nr0.conexiones)
  res ← false
  while HaySiguiente?(it) do
    if c1.ip = SiguienteSignificado(it)→pc→ip then
      res ← true
    end if
    Avanzar(it)
  end while
Complejidad :  $O(L + n)$ 

```

```

iInterfazUsada (in r: red, in c0: compu, in c1: compu) → res: interfaz
  nr0:nodoRed ← Significado(r.dns , c0.ip)
  it :itDicc(itDicc(interfaz , puntero(nodoRed)))
  ← CrearIt(nr0.conexiones)
  while HaySiguiente?(it) do
    if c1.ip = SiguienteSignificado(it)→pc→ip then
      res ← SiguienteClave(it)
    end if
    Avanzar(it)
  end while
Complejidad :  $O(L + n)$ 

```

```

iVecinos (in r: red, in c: compu) → res: conj(compu)
  nr:nodoRed ← Significado(r.dns , c.ip)
  res: conj(compu) ← Vacio()
  it :itDicc(itDicc(interfaz , puntero(nodoRed)))
  ← CrearIt(nr.conexiones)

```

while HaySiguiente?(it) do	O(1)
AgregarRapido(res, *(SiguienteSignificado(it) -> pc))	O(1)
Avanzar(it)	O(1)
end while	O(n)
Complejidad : $O(L + n)$	

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

iCaminosMinimos (in r: red, in c ₀ : compu, in c ₁ : compu) → res: conj(secu(compu))	
nr:nodoRed ← Significado(r.dns, c ₀ .ip)	O(L)
res ← Significado(pnr.caminos, c ₁ .ip)	O(L)
Complejidad : $O(L)$	

HayCamino? (in r: red, in c ₀ : compu, in c ₁ : compu) → res: bool	
nr:nodoRed ← Significado(r.dns, c ₀ .ip)	O(L)
res ← ¬EsVacio?(Significado(pnr.caminos, c ₁ .ip))	O(L)
Complejidad : $O(L)$	

Copiar (in r: red) → res: red	
res:red ← iIniciarRed	O(1)
itCompus:itConj(compu) ← CrearIt(r.compus)	O(1)
while HaySiguiente?(itCompus) do	O(1)
iAgregarComputadora(res, Siguiente(itCompus))	O(n*L)
iConectar(res, Siguiente(itCompus))	O(n!(n ⁴))
end while	O(n!(n ⁵))
Complejidad : $O(n! * (n^5))$	

• = • (in r ₀ : red, in r ₁ : red) → res: bool	
res ← (r ₀ .compus = r ₁ .compus) ∧ (r ₀ .dns = r ₁ .dns)	O(n ²)
Complejidad : $O(n^2)$	