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		

Índice

1. Módulo Red

Notas preliminares

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

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

Servicios usados: interfaz, tupla, nat, IP, lista

Interfaz

se explica con: RED, ITERADOR UNIDIRECCIONAL(COMPU)

géneros: `red`, `itRed`

Operaciones del TAD Red

INICIARRED() $\rightarrow$ *res* : `Red`
Pre $\equiv$ {true}
Post $\equiv$ {res $=_{\text{obs}}$ iniciarRed()}
Complejidad: $\Theta(1)$
Descripción: Genera una nueva red sin ninguna computadora.

AGREGARCOMPU(**in/out** r : `Red`, **in** c : `compu`)
Pre $\equiv$ {$r =_{\text{obs}} r_0 \wedge (\forall c'$: compu)($c' \in$ computadoras(r) $\rightarrow$ ip(c) $\neq$ ip(c'))}
Post $\equiv$ {$r =_{\text{obs}}$ agregarCompu(r_0, c)}
Complejidad: $\Theta(I)$
Descripción: Agrega una nueva computadora a la red.

CONECTAR(**in/out** r : `Red`, **in** c_0 : `compu`, **in** i_0 : `interfaz`, **in** c_1 : `compu`, **in** i_1 : `interfaz`) $\rightarrow$ *res* : `Red`
Pre $\equiv$ {$r =_{\text{obs}} r_0 \wedge c_1 \in$ computadoras(r) $\wedge$ $c_2 \in$ computadoras(r) $\wedge$ ip(c_0) $\neq$ ip(c_1) $\wedge$ $\neg$conectadas?(r, c_0 , c_1) $\wedge$ $\neg$usaInterfaz?(r, c_0, i_0) $\wedge$ $\neg$usaInterfaz?(r, c_1, i_1)}
Post $\equiv$ {$r =_{\text{obs}}$ conectar(r_0, c_0, i_0, c_1, i_1)}
Complejidad: $\Theta(n + I)$
Descripción: Conecta la computadora c_0 con la computadora c_1 a través de las interfaces i_0 y i_1 respectivamente.

COMPUTADORAS(**in** r : `Red`) $\rightarrow$ *res* : `conj(compu)`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ computadoras(r)}
Complejidad: $\Theta(1)$
Descripción: Devuelve el conjunto de todas las computadoras de la red.
Aliasing: El conjunto es devuelto por referencia.

CONECTADAS?(**in** r : `Red`, **in** c_0 : `compu`, **in** c_1 : `compu`) $\rightarrow$ *res* : `bool`
Pre $\equiv$ {$c_0 \in$ computadoras(r) $\wedge$ $c_1 \in$ computadoras(r)}
Post $\equiv$ {$res =_{\text{obs}}$ conectadas?(r, c_0, c_1)}
Complejidad: $\Theta(n + I)$
Descripción: Devuelve *true* si y solo si la computadora c_0 esta conectada a la computadora c_1

INTERFAZUSADA(**in** r : `Red`, **in** c_0 : `compu`, **in** c_1 : `compu`) $\rightarrow$ *res* : `interfaz`
Pre $\equiv$ {$c_0 \in$ computadoras(r) $\wedge$ $c_1 \in$ computadoras(r) $\wedge_{\text{L}}$ conectadas?(r, c_0, c_1)}
Post $\equiv$ {$res =_{\text{obs}}$ interfazUsada(r, c_0, c_1)}
Complejidad: $\Theta(n + I)$
Descripción: Devuelve la interfaz usada por c_0 para conectarse a c_1

VECINOS(**in** r : `Red`, **in** c : `compu`) $\rightarrow$ *res* : `conj(compu)`
Pre $\equiv$ {$c \in$ computadoras(r)}

Post $\equiv \{res =_{\text{obs}} \text{vecinos}(r, c)\}$
Complejidad: $\Theta(n + I^3)$
Descripción: Devuelve el conjunto de vecinos de la computadora c, es decir, las computadoras que tienen una conexión directa con c.
Aliasing: Devuelve el conjunto por copia.

usaInterfaz?(**in** r : `Red`, **in** c : `compu`, **in** i : `interfaz`) $\rightarrow$ res : `bool`
Pre $\equiv \{c \in \text{computadoras}(r)\}$
Post $\equiv \{res =_{\text{obs}} \text{usaInterfaz?}(r, c, i)\}$
Complejidad: $\Theta(n + I)$
Descripción: Devuelve *true* si y solo si la computadora c está usando la interfaz i.

caminosMinimos(**in** r : `Red`, **in** c_0 : `compu`, **in** c_1 : `compu`) $\rightarrow$ res : `conj(secu(compu))`
Pre $\equiv \{c_0 \in \text{computadoras}(r) \land c_1 \in \text{computadoras}(r)\}$
Post $\equiv \{res =_{\text{obs}} \text{caminosMinimos}(r, c_0, c_1)\}$
Complejidad: $\Theta(n^3 \times n! \times n! + I)$
Descripción: Devuelve el conjunto de todos los caminos mńimos posibles entre c_0 y c_1. De no haber ninguno, devuelve $\emptyset$.
Aliasing: Devuelve el conjunto por copia.

hayCamino?(**in** r : `Red`, **in** c_0 : `compu`, **in** c_1 : `compu`) $\rightarrow$ res : `bool`
Pre $\equiv \{c_0 \in \text{computadoras}(r) \land c_1 \in \text{computadoras}(r)\}$
Post $\equiv \{res =_{\text{obs}} \text{hayCamino?}(r, c_0, c_1)\}$
Complejidad: $\Theta(n^2 \times n!)$
Descripción: Devuelve *true* si y solo si hay al menos un camino posible entre c_0 y c_1.

cantCompus(**in** r : `Red`) $\rightarrow$ res : `nat`
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \#(\text{computadoras(r)})\}$
Complejidad: $\Theta(1)$
Descripción: Devuelve cuántas computadoras hay en la red.

copiar(**in** r : `Red`) $\rightarrow$ res : `Red`
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} r\}$
Complejidad: $\Theta(n \times I)$
Descripción: Devuelve una copia de la red.

Representación

`red` se representa con `estrRed`

donde `estrRed` es `tupla`(*compus*: `conjunto(compu)` , *conexiones*: `dicc(IP, diccConexiones)`)

donde `diccConexiones` es `dicc(interfaz, itDicc(IP, diccConexiones))`

Rep : Red $\longrightarrow$ bool

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

Abs : estrRed e $\longrightarrow$ Red {Rep(e)}

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

Estructura iterador de Red

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

Rep : itRed $\longrightarrow$ bool

Rep(it) $\equiv$ true

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

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

ArmarComputadoras : secu(tupla(string,secu(tupla(Interfaz,ItRed)))) $\longrightarrow$ conj(compu)

ArmarComputadoras(l) $\equiv$ **if** vacia?(l) **then**
$\emptyset$
else
Ag($\langle \Pi_1(\text{prim}(l)), \text{GenerarInterfaces}(\Pi_2(\text{prim}(l)))\rangle$, ArmarComputadoras(fin($l$)))
fi

ArmarSecuencia : secu(tupla(string,secu(tupla(interfaz,itLista(compu))))) $\longrightarrow$ secu(string)

ArmarSecuencia(s) $\equiv$ **if** vacia?(s) **then** <> **else** (Π_1(prim(s))) $\bullet$ ArmarSecuencia(fin(s)) **fi**

sinRepetidos : secu(string) $\longrightarrow$ bool

sinRepetidos(s) $\equiv$ #(pasarSecuAConj(s) = long(s)

pasarSecuAConj : secu(string) $\longrightarrow$ conj(string)

pasarSecuAConj(s) $\equiv$ **if** vacia?(s) **then** $\emptyset$ **else** Ag(prim(s), pasarSecuAConj(fin(s))) **fi**

GenerarInterfaces : secu(tupla(Interfaz,ItLista(estrCompu))) $\longrightarrow$ conj(Interfaz)

GenerarInterfaces(l) $\equiv$ **if** vacia?(l) **then** $\emptyset$ **else** Ag(Π_1(prim(l)), GenerarInterfaces(fin(l))) **fi**

Pertenece? : secu(tupla(string,secu(tupla(Interfaz,ItRed)))) l $\times$ compu c_1 $\times$ compu c_2 $\longrightarrow$ bool

Pertenece?(l, c_1, c_2) $\equiv$ **if** (Π_1(prim(l) = Π_1(c_1))) **then**
$\Pi_1(c_2) \in$ GenerarCompus(Π_2(prim(l)))
else
Pertenece?(fin(l), c_1, c_2)
fi

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

GenerarCompus(l) $\equiv$ **if** vacia?(l) **then** $\emptyset$ **else** Ag(Π_1(siguiente(Π_2(prim(l)))), GenerarCompus(fin(l))) **fi**

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

DevolverInterfaz(l, c_1, c_2) $\equiv$ **if** (Π_1(prim(l)) = Π_1(c_1)) **then**
DevolverInterfazAux(Π_2(prim(l), c_2))
else
DevolverInterfaz(fin(l, c_1, c_2))
fi

DevolverInterfazAux : secu(tupla(Interfaz × ItRed)) l × compu c $\longrightarrow$ Interfaz

DevolverInterfaz(l, c) $\equiv$ **if** ($\Pi_1(c_2) = \Pi_1$(siguiente(Π_2(prim(l))))) **then**
Π_1(prim(l))
else
DevolverInterfazAux(fin(l, c))
fi

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

armarCompus(es) $\equiv$ **if** vacía(es) **then** $<>$ **else** armarCompu(prim(es)) • armarCompus(fin(es)) **fi**

armarCompu : estrCompu e $\longrightarrow$ compu

armarCompu(e) $\equiv$ $\langle e$.IP, dame$\Pi_1(e$.conexiones)$\rangle$

dameΠ_1 : secu(tupla(*inter*:interfaz × *itCompu*:itLista(estrCompu))) l $\longrightarrow$ conj(interfaz)

dame$\Pi_1(l)$ $\equiv$ **if** vacía(l) **then** $\emptyset$ **else** ag(Pi_1(prim(l)), dameΠ_1(fin(l))) **fi**

Algoritmos

Algoritmos de Red

iIniciarRed() → *res* : `estrRed`	
1 *res* ← $\langle <>, 0 \rangle$	▷ $\Theta(1)$

Complejidad: $\Theta(1)$

iAgregarCompu(**in/out** *r* : `estrRed`, **in** *c* : `compu`)	
1 agregarAtras(r.compus, ⟨c.IP, ArmarLista(c.interfaces)⟩)	▷ $\Theta(I)$
2 $r.cantidadCompus \leftarrow r.cantidadCompus + 1$	▷ $\Theta(1)$

Complejidad: $\Theta(I)$

iArmarLista(**in** *c* : `conj(interfaz)`) → *res* : `lista(⟨Interfaz, itLista(estrCompu)⟩)`	
1 *res* ← vacia()	▷ $\Theta(1)$
2 `itConj(interfaz)` *it* ← crearIt(c)	▷ $\Theta(1)$
3 **while** haySiguiente(*it*) **do**	▷ $\Theta(I)$ iteraciones
4 agregarAtras(*res*, ⟨siguiente(*it*), NULL⟩)	▷ $\Theta(1)$
5 avanzar(*it*)	▷ $\Theta(1)$
6 **end while**	

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

Complejidad: $\Theta(I)$

iConectar(**in/out** r : `estrRed`), **in** c_1 : `compu`, **in** i_1 : `interfaz`, **in** c_2 : `compu`, **in** i_2 : `interfaz`)

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

Complejidad: $\Theta(n + I)$

iConectadas?(**in** r : `estrRed`, **in** c_1 : `compu`, **in** c_2 : `compu`) → res : `bool`

```
1  itLista(estrCompu) it_1 ← crearIt(r.compus)                                         ▷ Θ(1)
2  while siguiente(it_1).IP ≠ c_1.IP do                                                ▷ Θ(n) iteraciones
3  |  avanzar(it_1)                                                                   ▷ Θ(1)
4  end while
5  itLista(tupla(interfaz, itLista(estrCompu))) it_2 ← crearIt(siguiente(it_1).conexiones)   ▷ Θ(1)
6  while haySiguiente(it_2) ∧_L siguiente(siguiente(it_2).com)).IP ≠ c_2.IP do         ▷ Θ(I) iteraciones
7  |  avanzar(it_2)                                                                   ▷ Θ(1)
8  end while
9  res ← (siguiente(siguiente(it_2).com)).IP = c_2.IP)                                 ▷ Θ(1)
```

Complejidad: $\Theta(n + I)$

iInterfazUsada(**in** r : `estrRed`, **in** c_1 : `compu`, **in** c_2 : `compu`) → res : `interfaz`

```
1  itLista(estrCompu) it_1 ← crearIt(r.compus)                                         ▷ Θ(1)
2  while siguiente(it_1).IP ≠ c_1.IP do                                                ▷ Θ(n) iteraciones
3  |  avanzar(it_1)                                                                   ▷ Θ(1)
4  end while
5  itLista(tupla(interfaz, itLista(estrCompu))) it_2 ← crearIt(siguiente(it_1).conexiones)   ▷ Θ(1)
6  while (siguiente(siguiente(it_2).com)).IP ≠ c_2.IP do                               ▷ Θ(I) iteraciones
7  |  avanzar(it_1)                                                                   ▷ Θ(1)
8  end while
9  res ← siguiente(it_2).inter                                                         ▷ Θ(1)
```

Complejidad: $\Theta(n + I)$

iVecinos(**in** r : `estrRed`, **in** c : `compu`) → res : `conj(compu)`

```
1  res ← vacio()                                                                  ▷ Θ(1)
2  itLista(estrComp) it1 ← crearIt(r.compus)                                      ▷ Θ(1)
3  while siguiente(it1).IP ≠ c.IP do                                              ▷ Θ(n) iteraciones
4  |  avanzar(it1)                                                               ▷ Θ(1)
5  end while
6  itLista(tupla(interfaz, itLista(estrCompu))) it2 ← crearIt(siguiente(it1).conexiones)   ▷ Θ(1)
7  while haySiguiente?(it2) do                                                    ▷ Θ(n) iteraciones
8  |  if haySiguiente?(siguiente(it2).com) then                                   ▷ Θ(1)
9  |  |  agregar(res, ⟨siguiente(siguiente(it2).com).IP,
   |  |  crearConjunto(siguiente(siguiente(it2).com).conexiones)⟩)               ▷ Θ(I²)
10 |  end if
11 |  avanzar(it2)                                                              ▷ Θ(1)
12 end while
```

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

iCrearConjunto(**in** l: `lista(tupla(`$inter$`: interfaz,` com`: itLista(estrCompu)))`) → res : `conj(interfaz)`

```
1  nat n ← 0                                                                      ▷ Θ(1)
2  res ← vacio()                                                                  ▷ Θ(1)
3  while n <longitud(l) do                                                        ▷ Θ(I) iteraciones
4  |  agregar(res, (l[n]).inter)                                                 ▷ Θ(I)
5  |  n ← n + 1                                                                  ▷ Θ(1)
6  end while
```

Descripción: Dada una lista de tupla de ⟨Interfaz,Iterador⟩ (que representa las conexiones de la computadora), devuelve el conjunto de todas las interfaces que se encuentran en ella.

Complejidad: $\Theta(I^2)$

iUsaInterfaz?(**in** r : `estrRed`, **in** c : `compu`, **in** i : `interfaz`) → res : `bool`

```
1  itLista(estrComp) it1 ← crearIt(r.compus)                                      ▷ Θ(1)
2  while siguiente(it1).IP ≠ c.IP do                                              ▷ Θ(n) iteraciones
3  |  avanzar(it1)                                                               ▷ Θ(1)
4  end while
5  itLista(tupla(interfaz, itLista(estrCompu))) it2 ← crearIt(siguiente(it1).conexiones)   ▷ Θ(1)
6  while siguiente(it2).inter ≠ i do                                              ▷ Θ(I) iteraciones
7  |  avanzar(it2)                                                               ▷ Θ(1)
8  end while
9  res ← haySiguiente(siguiente(it2).com)                                         ▷ Θ(1)
```

Complejidad: $\Theta(n + I)$

iCaminosMinimos(**in** r : `estrRed`, **in** c_1 : `compu`, **in** c_2 : `compu`) → res : `conj(lista(compu))`

```
1  res ← vacio()                                                                  ▷ Θ(1)
2  if pertenece?(c2, vecinos(r, c1)) then                                         ▷ Θ(I)
3  |  agregar(res, agregarAtras(agregarAtras(<>,c1), c2))                         ▷ Θ(n + I)
4  else
5  |  res ← dameMinimos(Caminos(r, c1, c2, agregarAtras(<>,c1), pasarConjASecu(vecinos(r, c1))))
   |  ▷ Θ(n³ × n! × n!)
6  end if
```

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

```
DAMEMINIMOS(in c: conj(lista(compu))) → res : conj(lista(compu))
1  if esVacio?(c) then                                                    ▷ Θ(1)
2  |  res ← vacio()                                                      ▷ Θ(1)
3  else
4  |  itConj(lista(compu)) it ← crearIt(c)                               ▷ Θ(1)
5  |  res ← dameMinimosAux(c, minimaLong(c, long(siguiente(it))))        ▷ Θ(n × n!)
6  end if
```

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

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

```
DAMEMINIMOSAUX(in c: conj(lista(compu)), in n: nat) → res : conj(lista(compu))
1  itConj(lista(compu)) it ← crearIt(c)                                   ▷ Θ(1)
2  res ← vacio()                                                          ▷ Θ(1)
3  while haySiguiente(it) do                                              ▷ Θ(n!) iteraciones
4  |  if long(siguiente(it))= n then                                      ▷ Θ(1)
5  |  |  agregar(res, siguiente(it))                                      ▷ Θ(n)
6  |  |  avanzar(it)                                                      ▷ Θ(1)
7  |  else
8  |  |  avanzar(it)                                                      ▷ Θ(1)
9  |  end if
10 end while
```

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

```
MINIMALONG(in c: conj(lista(compu)), in n: nat) → res : nat
1  nat i ← n                                                              ▷ Θ(1)
2  itConj(lista(compu)) it ← crearIt(c)                                   ▷ Θ(1)
3  while haySiguiente(it) do
4  |  if long(siguiente(it)) then
5  |  |  i ← longitud(siguiente(it))                                      ▷ Θ(1)
6  |  |  avanzar(it)                                                      ▷ Θ(1)
7  |  else
8  |  |  avanzar(it)                                                      ▷ Θ(1)
9  |  end if
10 end while
11 res ← i                                                                ▷ Θ(1)
```

Complejidad: $\Theta(n!)$

Justificación: Devuelve la longitud de la secuencia más chica

```
PASARCONJASECU(in c: conj(compu)) → res : secu(compu)
1  res ← vacia()                                                          ▷ Θ(1)
2  ItConj it ← crearIt(c)                                                 ▷ Θ(1)
3  while haySiguiente(it) do                                              ▷ Θ(n) iteraciones
4  |  agregarAtras(res, siguiente(it))                                    ▷ Θ(I)
5  end while
```

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

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

iHayCamino?(**in** r : `estrRed`, **in** c_1 : `compu`, **in** c_2 : `compu`) → res : `bool`

```
1 res ← (¬esVacio?(iCaminosMinimos(r, c1, c2)))                    ▷ Θ(n² × n!)
```

Complejidad: $\Theta(n^2 \times n!)$

iCaminos(**in** r : `estrRed`, **in** c_1 : `compu`, **in** c_2 : `compu`, **in** l : `lista(estrCompu)`, **in** vec : `lista(estrCompu)`) → res : `conj(lista(estrCompu))`

```
1  if vacia?(vec) then
2  |  res ← vacia()
3  else
4  |  if último(l) = c1 then
5  |  |  res ← agregar(l, vacia())
6  |  else
7  |  |  if ¬está?(primero(vec, l)) then
8  |  |  |  res ← unión(caminos(r, c0, c1, agregarAtras(l, primero(vec)), Vecinos(r, primeros(vec))),
   |  |  |  caminos(r, c0, c1, l, fin(vec)))
9  |  |  else
10 |  |  |  res ← caminos(r, c0, c1, l, fin(vec))
11 |  |  end if
12 |  end if
13 end if
```

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

Complejidad: $\Theta(n^2 \times n!)$

iUnión(**in** c_1 : `conj(lista(compu))`, **in** c_2 : `conj(lista(compu))`) → res : `conj(lista(compu))`

```
1  res ← vacio()                                                   ▷ Θ(1)
2  if vacio?(c1) then                                              ▷ Θ(1)
3  |  res ← c2                                                     ▷ Θ(I × n × n!)
4  else
5  |  itConj(lista(compu)) it ← crearIt(c1)                        ▷ Θ(1)
6  |  while haySiguiente(it) do                                    ▷ Θ(n)
7  |  |  Ag(siguiente(it), c2)                                     ▷ Θ(n)
8  |  |  avanzar(it)                                               ▷ Θ(1)
9  |  end while
10 end if
```

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

Justificación: Devuelve la unión de dos conjuntos.

iEsta?(**in** c : `compu`, **in** l : `lista(compu)`) → res : `bool`

```
1  if vacia?(l) then                                               ▷ Θ(1)
2  |  res ← false                                                  ▷ Θ(1)
3  else
4  |  ItLista(compu) it ← crearIt(l)                               ▷ Θ(1)
5  |  while haySiguiente(it) ∧L siguiente(it) ≠ c do               ▷ Θ(n)
6  |
7  |  |  avanzar(it)                                               ▷ Θ(1)
8  |  end while
9  end if
10 res ← (haySiguiente(it))                                        ▷ Θ(1)
```

Descripción: Devuelve True si y solo si la compu c se encuentra en la lista l

Complejidad: $\Theta(n)$

Justificación: .

---

iComputadoras(**in** r : `estrRed`) → res : `conj(compu)`

```
1 res ← vacio()                                                    ▷ Θ(1)
2 itRed it ← crearItRed()                                          ▷ Θ(1)
3 while haySiguiente?(it) do                                       ▷ Θ(n) iteraciones
4 |   agregar(res, siguiente(it))                                  ▷ Θ(n + I²)
5 |   avanzar(it)                                                  ▷ Θ(1)
6 end while
```

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

---

iCopiar(**in** r : : estrRed) → res : `Red`

```
1 res ← ⟨copiar(r.compus, r.cantidadCompus⟩                        ▷ Θ(n × I)
```

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

Justificación: Devuelve una copia de la Red

---

iCantCompus(**in** r : `Red`) → res : `nat`

```
1 res ← r.cantCompus                                               ▷ Θ(1)
```

Complejidad: $\Theta(1)$

Algoritmos del iterador de Red

---

iHaySiguiente?(**in** it : `itLista(estrCompu)`) → res : `bool`

```
1 res ← haySiguiente?(it)                                          ▷ Θ(1)
```

Complejidad: $\Theta(1)$

---

iSiguiente(**in** it : `itLista(estrCompu)`) → res : `compu`

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

Complejidad: $\Theta(I^2)$

---

iCrearIt(**in** e : `estrRed`)

```
1 res ← crearIt(e.compus)                                          ▷ Θ(1)
```

Complejidad: $\Theta(1)$

iAvanzar(**in**/**out** *it* : `itLista(estrCompu)`)	
1 *it* ← avanzar(*it*)	▷ Θ(1)

Complejidad: Θ(1)