

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

5

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 $\equiv \{\}$

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

Complejidad: $\Theta(1)$

Descripción: Genera una nueva red sin ninguna computadora.

AGREGARCOMPU(**in/out** $r : Red$, **in** $c : compu$)

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

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

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

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

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 $\equiv \{\}$

Post $\equiv \{esAlias(res, computadoras(r))\}$

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

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

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

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** $r : \text{Red}$, **in** $c : \text{compu}$, **in** $i : \text{interfaz}$) $\rightarrow res : \text{bool}$

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

Post $\equiv \{res =_{\text{obs}} \text{usaInterfaz?}(r, c, i)\}$

Complejidad: $\Theta(n + I)$

Descripción: Devuelve *true* si y solo si la computadora c está 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: $\Theta(n^3 \times n! \times n! + I)$

Descripción: Devuelve el conjunto de todos los caminos mínimos posibles entre c_0 y c_1 . De no haber ninguno, devuelve \emptyset .

Aliasing: Devuelve el conjunto por copia.

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

Descripción: Devuelve *true* si y solo si hay al menos un camino posible entre c_0 y c_1 .

CANTCOMPUS(**in** $r : \text{Red}$) $\rightarrow res : \text{nat}$

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

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

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

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

Descripción: Devuelve una copia de la red.

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))*

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

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

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

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

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

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

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

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

$ArmarSecuencia(s) \equiv \text{if vacia?}(s) \text{ then } <> \text{ else } (\Pi_1(prim(s))) \bullet ArmarSecuencia(fin(s)) \text{ fi}$

$sinRepetidos : secu(string) \longrightarrow bool$

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

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

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

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

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

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

$Pertenece?(l, c_1, c_2) \equiv \text{if } (\Pi_1(prim(l)) = \Pi_1(c_1)) \text{ then}$
 $\quad \Pi_1(c_2) \in GenerarCompus(\Pi_2(prim(l)))$
 else
 $\quad Pertenece?(fin(l), c_1, c_2)$
 fi

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

$GenerarCompus(l) \equiv \text{if vacia?}(l) \text{ then } \emptyset \text{ else } Ag(\Pi_1(siguiente(\Pi_2(prim(l)))), GenerarCompus(fin(l))) \text{ fi}$

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

$DevolverInterfaz(l, c_1, c_2) \equiv \text{if } (\Pi_1(prim(l)) = \Pi_1(c_1)) \text{ then}$
 $\quad DevolverInterfaz_{aux}(\Pi_2(prim(l), c_2))$
 else
 $\quad DevolverInterfaz(fin(l), c_1, c_2)$
 fi

$DevolverInterfaz_{aux} : secu(tupla(Interfaz \times ItRed)) l \times compu c \longrightarrow Interfaz$

```

DevolverInterfaz( $l, c$ )  $\equiv$  if ( $\Pi_1(c_2) = \Pi_1(\text{siguiente}(\Pi_2(\text{prim}(l))))$ ) then
     $\Pi_1(\text{prim}(l))$ 
else
    DevolverInterfazaux(fin( $l, c$ ))
fi

```

Algoritmos

Algoritmo renovado

INICIARRED() $\rightarrow res : \text{estrRed}$	
1 $res \leftarrow \langle \text{Vacio}(), \text{Vacio}() \rangle$	$\triangleright \Theta(1)$

Complejidad: $\Theta(1)$

Algoritmo renovado

IAGREGARCOMPU(in/out $r : \text{estrRed}$, in $c : \text{compu}$)	
1 AgregarRapido($r.compus, c$)	$\triangleright \Theta(1)$
2 DefinirRapido($r.conexiones, c.IP, \text{Vacio}()$)	$\triangleright \Theta(1)$

Complejidad: $\Theta(1)$

Algoritmo renovado

ICONECTAR(in/out $r : \text{estrRed}$), in $c_1 : \text{compu}$, in $i_1 : \text{interfaz}$, in $c_2 : \text{compu}$, in $i_2 : \text{interfaz}$)	
1 itDicc(IP, diccConexiones) $it_1 \leftarrow \text{Creadit}(r.conexiones)$	$\triangleright \Theta(1)$
2 itDicc(IP, diccConexiones) $it_2 \leftarrow \text{Creadit}(r.conexiones)$	$\triangleright \Theta(1)$
3 while SiguieteClave(it_1) $\neq c_1.IP$ do	$\triangleright \Theta(n)$ iteraciones
4 Avanzar(it_1)	$\triangleright \Theta(1)$
5 end while	
6 while SiguieteClave(it_2) $\neq c_1.IP$ do	$\triangleright \Theta(n)$ iteraciones
7 Avanzar(it_2)	$\triangleright \Theta(1)$
8 end while	
9 DefinirRapido(SiguieteSignificado(it_1), i_1 , Copiar(it_2))	$\triangleright \Theta(1)$
10 DefinirRapido(SiguieteSignificado(it_2), i_2 , Copiar(it_1))	$\triangleright \Theta(1)$

Complejidad: $\Theta(n)$

Justificación: El algoritmo tiene dos ciclos que se ejecutan $\Theta(n)$ veces, cada una con complejidad $\Theta(1)$. El resto de las operaciones tiene complejidad $\Theta(1)$.

Algoritmo renovado

ICONECTADAS?(in r : estrRed, in c_1 : compu, in c_2 : compu) $\rightarrow res$: bool

```

1 itDicc(IP, diccConexiones)  $it_1 \leftarrow \text{CrearIt}(r.conexiones)$   $\triangleright \Theta(1)$ 
2 while SiguienteClave( $it_1$ )  $\neq c_1.IP$  do  $\triangleright \Theta(n)$  iteraciones
3   | Avanzar( $it_1$ )  $\triangleright \Theta(1)$ 
4 end while
5 itDicc(interfaz, itDicc(IP, diccConexiones))  $it_2 \leftarrow \text{CrearIt}(\text{Significado}(\text{Siguiente}(it_1)))$   $\triangleright \Theta(1)$ 
6 while HaySiguiente?( $it_2$ )  $\wedge_L$  SiguienteClave(SiguienteSignificado( $it_2$ ))  $\neq c_2.IP$  do  $\triangleright \Theta(I)$  iteraciones
7   | Avanzar( $it_2$ )  $\triangleright \Theta(1)$ 
8 end while
9  $res \leftarrow \text{HaySiguiente}(it_2) \wedge_L \text{SiguienteClave}(\text{SiguienteSignificado}(it_2)) = c_2.IP$   $\triangleright \Theta(1)$ 

```

Complejidad: $\Theta(n + I)$

Justificación: El algoritmo tiene dos ciclos; uno de ellos se ejecuta $\Theta(n)$ veces, y el otro $\Theta(I)$ veces, todas ellas con complejidad $\Theta(1)$. El resto de las operaciones tiene complejidad $\Theta(1)$.

INTERFAZUSADA(in r : estrRed, in c_1 : compu, in c_2 : compu) $\rightarrow res$: interfaz

```

1 itLista(estrCompu)  $it_1 \leftarrow \text{crearIt}(r.compues)$   $\triangleright \Theta(1)$ 
2 while siguiente( $it_1$ ).IP  $\neq c_1.IP$  do  $\triangleright \Theta(n)$  iteraciones
3   | avanzar( $it_1$ )  $\triangleright \Theta(1)$ 
4 end while
5 itLista(tupla(interfaz, itLista(estrCompu)))  $it_2 \leftarrow \text{crearIt}(\text{siguiente}(it_1).conexiones)$   $\triangleright \Theta(1)$ 
6 while (siguiente(siguiente( $it_2$ ).com)).IP  $\neq c_2.IP$  do  $\triangleright \Theta(I)$  iteraciones
7   | avanzar( $it_1$ )  $\triangleright \Theta(1)$ 
8 end while
9  $res \leftarrow \text{siguiente}(it_2).inter$   $\triangleright \Theta(1)$ 

```

Complejidad: $\Theta(n + I)$

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

```

1  $res \leftarrow \text{vacío}()$   $\triangleright \Theta(1)$ 
2 itLista(estrComp)  $it_1 \leftarrow \text{crearIt}(r.compues)$   $\triangleright \Theta(1)$ 
3 while siguiente( $it_1$ ).IP  $\neq c.IP$  do  $\triangleright \Theta(n)$  iteraciones
4   | avanzar( $it_1$ )  $\triangleright \Theta(1)$ 
5 end while
6 itLista(tupla(interfaz, itLista(estrCompu)))  $it_2 \leftarrow \text{crearIt}(\text{siguiente}(it_1).conexiones)$   $\triangleright \Theta(1)$ 
7 while haySiguiente?( $it_2$ ) do  $\triangleright \Theta(n)$  iteraciones
8   | if haySiguiente?(siguiente( $it_2$ ).com) then  $\triangleright \Theta(1)$ 
9     | agregar( $res$ , (siguiente(siguiente( $it_2$ ).com).IP,
10       | crearConjunto(siguiente(siguiente( $it_2$ ).com).conexiones))))  $\triangleright \Theta(I^2)$ 
11   | end if
12   | avanzar( $it_2$ )  $\triangleright \Theta(1)$ 
13 end while

```

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

```

ICREARCONJUNTO(in l: lista(tupla(inter: interfaz, com: itLista(estrCompu)))) → res :
conj(interfaz)
1 nat n ← 0                                     ▷  $\Theta(1)$ 
2 res ← vacío()                                   ▷  $\Theta(1)$ 
3 while n < longitud(l) do                       ▷  $\Theta(I)$  iteraciones
4   | agregar(res, (l[n]).inter)                 ▷  $\Theta(I)$ 
5   | n ← n + 1                                  ▷  $\Theta(1)$ 
6 end while

```

Descripción: Dada una lista de tupla de $\langle \text{Interfaz}, \text{Iterador} \rangle$ (que representa las conexiones de la computadora), devuelve el conjunto de todas las interfaces que se encuentran en ella.

Complejidad: $\Theta(I^2)$

```

IUSAINTERFAZ?(in r: estrRed, in c: compu, in i: interfaz) → res : bool
1 itLista(estrComp) it1 ← crearIt(r.compus)                                     ▷  $\Theta(1)$ 
2 while siguiente(it1).IP ≠ c.IP do                                           ▷  $\Theta(n)$  iteraciones
3   | avanzar(it1)                                                            ▷  $\Theta(1)$ 
4 end while
5 itLista(tupla(interfaz, itLista(estrCompu))) it2 ← crearIt(siguiente(it1).conexiones) ▷  $\Theta(1)$ 
6 while siguiente(it2).inter ≠ i do                                           ▷  $\Theta(I)$  iteraciones
7   | avanzar(it2)                                                            ▷  $\Theta(1)$ 
8 end while
9 res ← haySiguiente(siguiente(it2).com)                                       ▷  $\Theta(1)$ 

```

Complejidad: $\Theta(n + I)$

```

ICAMINOSMINIMOS(in r: estrRed, in c1: compu, in c2: compu) → res : conj(lista(compu))
1 res ← vacío()                                                                    ▷  $\Theta(1)$ 
2 if pertenece?(c2, vecinos(r, c1)) then                                     ▷  $\Theta(I)$ 
3   | agregar(res, agregarAtras(agregarAtras(<>, c1), c2))                 ▷  $\Theta(n + I)$ 
4 else
5   | res ← dameMinimos(Caminos(r, c1, c2, agregarAtras(<>, c1), pasarConjASecu(vecinos(r, c1))))
6   | ▷  $\Theta(n^3 \times n! \times n!)$ 
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                                                           ▷  $\Theta(1)$ 
2   | res ← vacío()                                                              ▷  $\Theta(1)$ 
3 else
4   | itConj(lista(compu)) it ← crearIt(c)                                     ▷  $\Theta(1)$ 
5   | res ← dameMinimosAux(c, minimaLong(c, long(siguiente(it))))           ▷  $\Theta(n \times n!)$ 
6 end if

```

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

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

DAMEMINIMOSAUX(**in** $c : \text{conj}(\text{lista}(\text{compu}))$, **in** $n : \text{nat}$) $\rightarrow res : \text{conj}(\text{lista}(\text{compu}))$

```

1  itConj(lista(compu)) it ← crearIt(c)                                ▷  $\Theta(1)$ 
2  res ← vacío()                                                         ▷  $\Theta(1)$ 
3  while haySiguiente(it) do                                           ▷  $\Theta(n!)$  iteraciones
4    if long(siguiente(it)) = n then                                   ▷  $\Theta(1)$ 
5      | agregar(res, siguiente(it))                                  ▷  $\Theta(n)$ 
6      | avanzar(it)                                                 ▷  $\Theta(1)$ 
7    else
8      | avanzar(it)                                                 ▷  $\Theta(1)$ 
9    end if
10 end while

```

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

MINIMALONG(**in** $c : \text{conj}(\text{lista}(\text{compu}))$, **in** $n : \text{nat}$) $\rightarrow res : \text{nat}$

```

1  nat i ← n                                                           ▷  $\Theta(1)$ 
2  itConj(lista(compu)) it ← crearIt(c)                               ▷  $\Theta(1)$ 
3  while haySiguiente(it) do
4    if long(siguiente(it)) then
5      | i ← longitud(siguiente(it))                                  ▷  $\Theta(1)$ 
6      | avanzar(it)                                                 ▷  $\Theta(1)$ 
7    else
8      | avanzar(it)                                                 ▷  $\Theta(1)$ 
9    end if
10 end while
11 res ← i                                                            ▷  $\Theta(1)$ 

```

Complejidad: $\Theta(n!)$

Justificación: Devuelve la longitud de la secuencia más chica

PASARCONJASECU(**in** $c : \text{conj}(\text{compu})$) $\rightarrow res : \text{secu}(\text{compu})$

```

1  res ← vacía()                                                       ▷  $\Theta(1)$ 
2  ItConj it ← crearIt(c)                                             ▷  $\Theta(1)$ 
3  while haySiguiente(it) do                                           ▷  $\Theta(n)$  iteraciones
4    | agregarAtras(res, siguiente(it))                              ▷  $\Theta(I)$ 
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** $r : \text{estrRed}$, **in** $c_1 : \text{compu}$, **in** $c_2 : \text{compu}$) $\rightarrow res : \text{bool}$

```

1  res ← (¬esVacio?(iCaminosMinimos(r, c1, c2)))                  ▷  $\Theta(n^2 \times n!)$ 

```

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))
 $\rightarrow res$: 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))),
9         | caminos(r, c0, c1, l, fin(vec)))
10    | else
11      | res ← caminos(r, c0, c1, l, fin(vec))
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))) $\rightarrow res$: conj(lista(compu))

```

1 res ← vacio()                                ▷  $\Theta(1)$ 
2 if vacio?(c1) then                            ▷  $\Theta(1)$ 
3   | res ← c2                                ▷  $\Theta(I \times n \times n!)$ 
4 else
5   | itConj(lista(compu)) it ← crearIt(c1)    ▷  $\Theta(1)$ 
6   | while haySiguiente(it) do                ▷  $\Theta(n)$ 
7     | Ag(siguiente(it), c2)                 ▷  $\Theta(n)$ 
8     | avanzar(it)                           ▷  $\Theta(1)$ 
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)) $\rightarrow res$: bool

```

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

```

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

Complejidad: $\Theta(n)$

Justificación: .

ICOMPUTADORAS(**in** $r : \text{estrRed}$) $\rightarrow res : \text{conj}(\text{compu})$

```

1  $res \leftarrow \text{vacío}()$   $\triangleright \Theta(1)$ 
2 itRed  $it \leftarrow \text{crearItRed}()$   $\triangleright \Theta(1)$ 
3 while haySiguiente?( $it$ ) do  $\triangleright \Theta(n)$  iteraciones
4   | agregar( $res$ , siguiente( $it$ ))  $\triangleright \Theta(n + I^2)$ 
5   | avanzar( $it$ )  $\triangleright \Theta(1)$ 
6 end while
```

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

ICOPIAR(**in** $r : : \text{estrRed}$) $\rightarrow res : \text{Red}$

```

1  $res \leftarrow \langle \text{copiar}(r.\text{compus}, r.\text{cantidadCompus}) \rangle$   $\triangleright \Theta(n \times I)$ 
```

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

Justificación: Devuelve una copia de la Red

ICANTCOMPUS(**in** $r : \text{Red}$) $\rightarrow res : \text{nat}$

```

1  $res \leftarrow r.\text{cantCompus}$   $\triangleright \Theta(1)$ 
```

Complejidad: $\Theta(1)$