1. Módulo Red

hostname es string
interfaz es nat
prioridad es nat
id es nat
paquete es tupla<id:Id, prioridad : prioridad, origen : compu, destino : compu>
compu es tupla<ip : ip, interfaces : conj(interfaz)>

Interfaz

se explica con: RED.

géneros:`red`.

Operaciones básicas de Red

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

AGREGARCOMPUTADORA(**in/out** r : `Red` , **in** c : `compu`)
Pre $\equiv$ {$r =_{\text{obs}} r_0$}
Post $\equiv$ {$r =_{\text{obs}} agregarComputadora(r_0, c)$}
Complejidad: O(L)
Descripción: Agrega una computadora a la red

CONECTAR(**in/out** r : `Red` , **in** $c1$: `compu` , **in** $i1$: `interfaz`, **in** $c2$: `compu`, **in** $i2$: `interfaz`)
Pre $\equiv$ {$r =_{\text{obs}} r_0 \wedge c1 \in computadoras(r_0) \wedge c2 \in computadoras(r_0) \wedge ip(c1) \neq ip(c2) \wedge \neg conectadas(r_0, c1, c2) \wedge \neg usaInterfaz(r_0, c1, i1) \wedge \neg usaInterfaz(r_0, c2, i2)$}
Post $\equiv$ {$r =_{\text{obs}} conectar(r_0, c1, i1, c2, i2) \wedge conectadas(r_0, c1, c2) \wedge usaInterfaz(r_0, c1, i1) \wedge usaInterfaz(r_0, c2, i2)$}
Complejidad: O((L x n + Cardinal(dU.interfaces))x n^3) donde dU es la computadora de la red con más interfaces.
Descripción: Modifica la red r conectando las computadoras c1 y c2 a través de las interfaces i1 e i2 respectivamente.

COMPUTADORAS(**in** r : `Red`) $\rightarrow$ *res* : `conj(compu)`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} computadoras(r)$}
Complejidad: O(1)
Descripción: Devuelve un conjunto con todas las computadoras de la red

CONECTADAS?(**in** r : `Red`, **in** $c1$: `compu`, **in** $c2$: `compu`) $\rightarrow$ *res* : `bool`
Pre $\equiv$ {$c1 \in computadoras(r) \wedge c2 \in computadoras(r)$}
Post $\equiv$ {$res =_{\text{obs}} conectadas(r, c1, c2)$}
Complejidad: O($\Sigma_{a' \in Significado(r.Conexiones, c1.hostname)} equal(c2.hostname, a')$))
Descripción: Devuelve true si c1 y c2 están conectadas en la red r.

INTERFAZUSADA(**in** r : `Red`, **in** $c1$: `compu`, **in** $c2$: `compu`) $\rightarrow$ *res* : `interfaz`
Pre $\equiv$ {$conectadas?(r, c1, c2)$}
Post $\equiv$ {$res =_{\text{obs}} interfazUsada(r, c1, c2)$}
Complejidad: O(L)
Descripción: Devuelve la interfaz de c1 que se conecta con c2 en la red r.

VECINOS(**in** r : `Red`, **in** c : `compu`) $\rightarrow$ *res* : `conj(compu)`
Pre $\equiv$ {$c \in computadoras(r)$}
Post $\equiv$ {$res =_{\text{obs}} vecinos(r, c)$}
Complejidad: O(L)
Descripción: Devuelve un conjunto de computadoras con las cuales está conectada la computadora c.

UsaInterfaz?(**in** r : `Red`, **in** c : `compu`, **in** i : `interfaz`) $\rightarrow$ res : `bool`
Pre $\equiv$ $\{c \in computadoras(r)\}$
Post $\equiv$ $\{res =_{\text{obs}} usaInterfaz?(r, c, i)\}$
Complejidad: O(n x L)
Descripción: Devuelve true si la interfaz i de la computadora c está siendo utilizada para conectarse con alguna computadora de la red r.

CaminosMinimos(**in** r : `Red`, **in** $c1$: `compu`, **in** $c2$: `compu`) $\rightarrow$ res : `conj(secu(compu)`
Pre $\equiv$ $\{c1 \in computadoras(r) \land c2 \in computadoras(r)\}$
Post $\equiv$ $\{res =_{\text{obs}} caminosMinimos(r, c1, c2)\}$
Complejidad: O(L)
Descripción: Devuelve todos los caminos más cortos para llegar de c1 a c2.

HayCamino?(**in** r : `Red`, **in** $c1$: `compu`, **in** $c2$: `compu`) $\rightarrow$ res : `bool`
Pre $\equiv$ $\{c1 \in computadoras(r) \land c2 \in computadoras(r)\}$
Post $\equiv$ $\{res =_{\text{obs}} hayCamino?(r, c1, c2)\}$
Complejidad: O(L)
Descripción: Devuelve true si hay algún camino que conduzca de c1 a c2.

Funciones auxiliares:

ConstruirCaminosMinimos(in r : Red, in c1 : compu , in c2 : compu) $\rightarrow$ res : conj(lista(compu))
-Dado unas computadoras c1 y c2 de una red r, se debe contruir un conj que contenga los caminos minimos para llegar desde la computadora c1 a c2. c1 y c2 deben tener distinto hostname.

AuxCaminos(in r : Red, in c1 : compu , in c2 : compu, in recorrido : lista(compu), in candidatos : conj(compu)) $\rightarrow$ res : conj(lista(compu))
-Dado dos computadoras c1 y c2 de una red r, una lista de compu que estan conectadas cada una con la siguiente formando un camino que comienza con c1 y el conjunto de computadoras conectadas con el último valor de recorrido en la red r.

Representación

Representación de Red

`red` se representa con `rd`

donde `rd` es `tupla`(*Computadoras*: `conj(compu)` , *Conexiones*: `tried(hostname, conj(compu))` , *InterfacesQueConectan*: `tried(hostname,tried(hostname, interfaz))`, *caminosMinimos*: `tried(hostname,tried(hostname, conj(lista(compu))))`)

Rep : rd $\longrightarrow$ bool
Rep(r) $\equiv$ true $\iff$ ($\forall$ c : compu)(c $\in$ r.Computadoras $\Leftrightarrow$ (Definido?(r.Conexiones, c.hostname) $\land$ Definido?(r.InterfacesQueConectan, c.hostname) $\land$ Definido?(r.caminosMinimos, c.hostname) $\land_{\text{L}}$ ($\forall$ ca : compu)(ca $\in$ Significado(r.Conexiones, c.hostname) $\Leftrightarrow$ ca $\in$ r.Computadoras $\land$ (Definido?(Significado(r.InterfacesQueConectan, c.hostname), ca.hostname) $\land_{\text{L}}$ Significado(Significado(r.InterfacesQueConectan, c.hostname), ca.hostname) $\in$ c.interfaces) $\land$ (Definido?(Significado(r.InterfacesQueConectan, ca.hostname), c.hostname) $\land_{\text{L}}$ Significado(Significado(r.InterfacesQueConectan, ca.hostname), c.hostname) $\in$ ca.interfaces))))) $\land$ ($\forall$ c1, c2 : compu)(c1 $\in$ r.Computadoras $\land$ c2 $\in$ r.Computadoras $\Leftrightarrow$ Definido?(Significado(r.caminosMinimos, c1.hostname), c2.hostname) $\land$ Definido?(Significado(r.caminosMinimos, c2.hostname), c1.hostname) $\land_{\text{L}}$ Significado(Significado(r.caminosMinimos,c1.hostname), c2.hostname)=construirCaminosMinimos(r.Conexiones, c1, c2))

construirCaminosMinimos : diccionario(hostname $\times$ conj(compu)) $\times$ compu $\times$ compu $\longrightarrow$ conj(secu(compu))
construirCaminosMinimos(cr, c1, c2) $\equiv$ auxMinimos(caminos(cr, c1, c2))

caminos : diccionario(hostname $\times$ conj(compu)) $\times$ compu $\times$ compu $\longrightarrow$ conj(secu(compu))
caminos(cr, c1, c2) $\equiv$ auxCaminos(cr, c1, c2, c1 $\bullet$ $<>$, Significado(cr, c1.hostname))

auxCaminos : diccionario(hostname $\times$ conj(compu)) $\times$ compu $\times$ compu $\times$ secu(compu) $\times$ conj(compu) $\longrightarrow$ conj(secu(c

auxCaminos(r, c1, c2, recorrido, candidatos) $\equiv$ **if** ϕ?(candidatos) **then**
$ag(<>, \phi)$
else
if ult(recorrido) = c2 **then**
ag(recorrido, ϕ)
else
if ¬esta?(dameUno(candidatos), recorrido) **then**
auxCaminos(r, c1 , c2 , recorrido ∘ dameUno(candidatos), Significado(r, dameUno(candidatos).hostname)) ∪ auxCaminos(r, c1 , c2 , recorrido, sinUno(candidatos))
else
auxCaminos(r, c1 , c2 , recorrido, sinUno(candidatos))
fi
fi
fi

auxMinimos : conj(secu(compu)) $\longrightarrow$ bool
auxMinimos(cc) $\equiv$ **if** ϕ?(cc) **then**
ϕ
else
if #(cc) = 1 **then**
ag(dameUno(c), ϕ)
else
if long(dameUno(cc)) <long(dameUno(auxMinimos(sinUno(cc)))) **then**
ag(dameUno(cc), ϕ)
else
if long(dameUno(cc)) = long(dameUno(auxMinimos(sinUno(cc)))) **then**
ag(dameUno(c), auxMinimos(sinUno(cc)))
else
auxMinimos(sinUno(cc))
fi
fi
fi
fi

Abs : rd r $\longrightarrow$ red {Rep(r)}
Abs(r) $=_{obs}$ e: red | Computadoras(e) = r.Computadora $\wedge_L$ ($\forall$ c1, c2 : compu)(conectadas?(e, c1, c2) = (c2 $\in$ Significado(r.Conexiones, c1.hostname) $\wedge$ c1 $\in$ Significado(r.Conexiones, c2.hostname)) $\wedge_L$ interfazUsada(e, c1, c2) = Significado(Significado(r.InterfacesQueConectan, c1.hostname), c2.hostname))
O(n x L + Cardinal(c.interfaces))

Algoritmos

---

iIniciarRed() $\rightarrow$ *res* : *red*

res $\leftarrow$ <Vacío(),Vacío(),Vacío(),Vacío()> ▷ O(1)

Complejidad: O(1)
Justificación:

---

iAgregarCompu(**in/out** r : `Red`, **in** c : `compu`)

Agregar(r.Computadoras, c) ▷ $O(\sum_{a' \in r.Computadoras} equal(c, a'))$
Definir(r.Conexiones, c.hostname, Vacío()) ▷ O(long(c.hostname) + copy(c.hostname)) = O(long(c.hostname)) = O(L)
Definir(r.InterfacesQueConectan, c.hostname, Vacío()) ▷ O(long(c.hostname) + copy(c.hostname)) = O(long(c.hostname)) = O(L)
Definir(r.caminosMinimos, c.hostname, Vacío()) ▷ O(long(c.hostname) + copy(c.hostname)) = O(long(c.hostname)) = O(L)
iteradorCompu : itConj(compu) ← crearIT(r.Computadoras) ▷ O(1)
while HaySiguiente(iteradorCompu) **do** ▷ O((n-1) x L) = O(n x L)
  if Siguiente(iteradorCompu).hostname ≠ c.hostname **then** ▷ O(3 x L + 4) = O(L)
    vacia1 : lista(compu) ← Vacía() ▷ O(1)
    vacia2 : lista(compu) ← Vacía() ▷ O(1)
    Agregar(vacia1, Vacía()) ▷ O(1) ya que no hay elementos en vacia1
    Agregar(vacia1, Vacía()) ▷ O(1) ya que no hay elementos en vacia2
    Definir(Significado(r.caminosMinimos, c.hostname) , Siguiente(iteradorCompu).hostname, vacia1) ▷ O(2 x L) = O(L)
    Definir(Significado(r.caminosMinimos, Siguiente(iteradorCompu).hostname) , c.hostname, vacia2) ▷ O(2 x L) = O(L)
  end if
  Avanzar(iteradorCompu) ▷ O(1)
end while

Complejidad: O(n x L + Cardinal(c.interfaces))
Justificación: O(n x L + 3 x L + $\sum_{a' \in r.Computadoras} equal(c, a')$ +1) = O(n x L + $\sum_{a' \in r.Computadoras} equal(c, a')$). El costo de ver si dos computadoras son iguales es O(Cardinal(c.interfaces) + L), por las interfaces y el hostname. Nos queda O(n x L + Cardinal(c.interfaces) + L) = O(n x L + Cardinal(c.interfaces))

iConectar(**in/out** r : `Red`, **in** $c1$: `compu`, **in** $c2$: `compu`, **in** $i1$: `interfaz`, **in** $i2$: `interfaz`)

AgregarRapido(Significado(r.Conexiones, c1.hostname), c2) ▷ O(copy(c2)) = O(Cardinal(c2.interfaces) + long(c2.hostname)) = O(Cardinal(c2.interfaces) + L)
AgregarRapido(Significado(r.Conexiones, c2.hostname), c1) ▷ O(copy(c1)) = O(Cardinal(c1.interfaces) + long(c1.hostname)) = O(Cardinal(c1.interfaces) + L)
if ¬Definido?(r.InterfacesQueConectan, c1.hostname) **then** ▷ O(logn(c1.hostname)) = O(L)
  Definir(r.InterfacesQueConectan, c1.hostname, Vacío()) ▷ O(logn(c1.hostname)) = O(L)
end if
if ¬Definido?(r.InterfacesQueConectan, c2.hostname) **then** ▷ O(logn(c2.hostname)) = O(L)
  Definir(r.InterfacesQueConectan, c2.hostname, Vacío()) ▷ O(logn(c2.hostname)) = O(L)
end if
Definir(Significado(r.InterfacesQueConectan,c1.hostname), c2.hostname, i2) ▷ O(logn(c2.hostname) + logn(c1.hostname)) = O(L + L) = O(L)
Definir(Significado(r.InterfacesQueConectan,c2.hostname), c1.hostname, i1) ▷ O(logn(c2.hostname) + logn(c1.hostname)) = O(L + L) = O(L)
iteradorCompu1 : itConj(compu) ← crearIT(r.Computadoras) ▷ O(1)
while HayCamino(iteradorCompu1) **do** ▷ =O((L x n + Cardinal(dU.interfaces))x n^3 + n) = O((L x n + Cardinal(dU.interfaces))x n^3)
  iteradorCompu2 : itConj(compu) ← crearIT(r.Computadoras) ▷ O(1)
  while HayCamino(iteradorCompu2) **do** ▷ O(2 x (L x n + Cardinal(dU.interfaces))x n^2 + n) = O((L x n + Cardinal(dU.interfaces))x n^2)
    if Siguiente(iteradorCompu1).hostname ≠ Siguiente(iteradorCompu2).hostname **then** ▷ O(L + (L x n + Cardinal(dU.interfaces))x n) = O((L x n + Cardinal(dU.interfaces))x n)
      Definir(Significado(r.caminosMinimos, Siguiente(iteradorCompu1).hostname), Siguiente(iteradorCompu2).hostname, ConstruirCaminosMinimos(r, Siguiente(iteradorCompu1), Siguiente(iteradorCompu2))) ▷ O(2 x L + (L x n + Cardinal(dU.interfaces))x n)=O((L x n + Cardinal(dU.interfaces))x n) donde dU es la computadora entre los candidatos con mayor cantidad de interfaces conectadas.
      Definir(Significado(r.caminosMinimos, Siguiente(iteradorCompu2).hostname), Siguiente(iteradorCompu1).hostname, ConstruirCaminosMinimos(r, Siguiente(iteradorCompu2), Siguiente(iteradorCompu1))) ▷ O(2 x L + (L x n + Cardinal(dU.interfaces))x n)=O((L x n + Cardinal(dU.interfaces))x n) donde dU es la computadora entre los candidatos con mayor cantidad de interfaces conectadas.
    end if
    Avanzar(iteradorCompu2) ▷ O(1)
  end while
  Avanzar(iteradorCompu1) ▷ O(1)
end while

Complejidad: O((L x n + Cardinal(dU.interfaces))x n^3) donde dU es la computadora de la red con más interfaces.
Justificación: O(Cardinal(c1.interfaces) + Cardinal(c2.interfaces) + 8 x L + 1 + (L x n + Cardinal(dU.interfaces))x n^3) = O((L x n + Cardinal(dU.interfaces))x n^3) . Podemos usar AgregarRapido ya que la precondición de conectar nos dice que las computadoras c1 y c2 están en la red y no estan conectadas entre si. Como Cardinal(dU.interfaces) es la cantidad de interfaces de la compu con más interfaces esta le gana a los cardinales de c1.interfaces y c2.interfaces

---

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

res ← r.Computadoras ▷ O(1)

Complejidad: O(1)
Justificación:

iConectadas?(**in** r : `Red`, **in** $c1$: `compu`, **in** $c2$: `compu`) $\rightarrow$ res : $bool$

res $\leftarrow$ Pertenece?(Significado(r.Conexiones, c1.hostname), c2) ▷ $O(\Sigma_{a' \in Significado(r.Conexiones, c1.hostname)} equal(c2.hostname, a'))$ + long(c1.hostname))

Complejidad: $O(\Sigma_{a' \in Significado(r.Conexiones, c1.hostname)} equal(c2.hostname, a')))$
Justificación: $O(\Sigma_{a' \in Significado(r.Conexiones, c1.hostname)} equal(c2.hostname, a'))$ + long(c1.hostname)) = $O(\Sigma_{a' \in Significado(r.Conexiones, c1.hostname)} equal(c2.hostname, a')) + L) = O(\Sigma_{a' \in Significado(r.Conexiones, c1.hostname)} equal(c2.ho$
Como comparar las computadoras es hacerlo con sus hostname e interfaces entonces puedo decir que L pierde contra la comparación de todas las computadoras que puede tener el conjunto.

---

iInterfazUsada(**in** r : `Red`, **in** $c1$: `compu`, **in** $c2$: `compu`) $\rightarrow$ res : $interfaz$

res $\leftarrow$ Significado(Significado(r.InterfacesQueConectan, c1.hostname), c2.hostname) ▷ O(long(c1.hostname) + long(c2.hostname))

Complejidad: O(L)
Justificación: O(long(c1.hostname) + long(c2.hostname)) = O(L + L) = O(L)

---

iVecinos(**in** r : `Red`, **in** c : `compu`, **in** i : `interfaz`) $\rightarrow$ res : $conj(compu)$

res $\leftarrow$ Significado(r.Conexiones, c.hostname) ▷ O(long(c.hostname))

Complejidad: O(L)
Justificación: O(long(c.hostname)) = O(L)

---

iUsaInterfaz?(**in** r : `Red`, **in** c : `compu`, **in** i : `interfaz`) $\rightarrow$ res : $bool$

iteradorConexion : itConj(hostname)$\leftarrow$ crearIT(Claves(Significado(r.InterfacesQueConectan, c1.hostname))) ▷ O(long(c1.hostname)) = O(L)
encontre : bool $\leftarrow$ false ▷ O(1)
while HaySiguiente(iteradorConexion) $\wedge$ $\neg$encontre **do** ▷ O(n x L)
  if Significado(Significado(r.InterfacesQueConectan, c1.hostname), Siguiente(iteradorConexion)) $\neq$ i **then** ▷ O(1 + long(c1.hostname) + long(Siguiente(iteradorConexion)) = O(1 + L + L) = O(L)
    encontre $\leftarrow$ true ▷ O(1)
  end if
  Avanzar(iteradorConexion) ▷ O(1)
end while
res $\leftarrow$ HaySiguiente(iteradorConexion) ▷ O(1)

Complejidad: O(n x L)
Justificación: O(n x L + 1 + L) = O(n x L)

---

iCaminosMinimos(**in** r : `Red`, **in** $c1$: `compu`, **in** $c2$: `compu`) $\rightarrow$ res : $bool$

res $\leftarrow$ Significado(Significado(r.caminosMinimos, c1.hostname), c2.hostname) ▷ O(long(c1.hostname) + long(c2.hostname))

Complejidad: O(L)
Justificación: O(long(c1.hostname) + long(c2.hostname)) = O(L + L) = O(L)

---

iHayCamino?(**in** r : `Red`, **in** $c1$: `compu`, **in** $c2$: `compu`) → res : $bool$

res ← ¬EsVacía?(Siguiente(crearIT(Significado(Significado(r.caminosMinimos, c1.hostname), c2.hostname), Vacía()))) ▷ O(long(c1.hostname) + long(c2.hostname) + 2)

Complejidad: O(L)
Justificación: O(long(c1.hostname) + long(c2.hostname) + 2) = O(L + L) = O(L)

---

iConstruirCaminosMinimos(**in** r : `Red`, **in** $c1$: `compu` , **in** $c2$: `compu`) → res : $conj(lista(compu))$

recorrido ← AgregarAdelante(Vacía(), c1) ▷ O(1)
candidatos ← Significado(r.Conexiones, c1.hostname) ▷ O(long(c1.hostname)) = O(L)
camino ← AuxCaminos(r, c1, c2, recorrido, candidatos) ▷ O((L x n + Cardinal(dU.interfaces))x n) donde dU es la computadora entre los candidatos con mayor cantidad de interfaces conectadas.
iterador : itConj(lista(compu)) ← crearIT(camino) ▷ O(1)
minimoActual : lista(compu) ← Siguiente(iterador) ▷ O(1)
Avanzar(iterador) ▷ O(1)
while HaySiguiente(iterador) **do** ▷ O(3 x n)
  if Longitud(minimoActual) >Longitud(Siguiente(iterador)) **then** ▷ O(2)
    minimoActual ← Siguiente(iterador) ▷ O(1)
  end if
  Avanzar(iterador) ▷ O(1)
end while
iterador2 : itConj(lista(compu)) ← crearIT(camino) ▷ O(1)
while HaySiguiente(iterador2) **do** ▷ O(3 x n)
  if Longitud(minimoActual) <Longitud(Siguiente(iterador2)) **then** ▷ O(2)
    EliminarSiguiente(iterador2) ▷ O(1)
  else
    Avanzar(iterador) ▷ O(1)
  end if
end while
res ← camino ▷ O(1)

Complejidad: O((L x n + Cardinal(dU.interfaces))x n) donde dU es la computadora entre los candidatos con mayor cantidad de interfaces conectadas.
Justificación: O(6 x n + 6 + L + (L x n + Cardinal(dU.interfaces))x n) = O((L x n + Cardinal(dU.interfaces))x n) donde dU es la computadora entre los candidatos con mayor cantidad de interfaces conectadas.

iAuxCaminos(**in** r : `Red`, **in** $c1$: `compu` , **in** $c2$: `compu`, **in** *recorrido* : `lista(compu)`, **in** *candidatos* : `conj(compu)`) → *res* : *conj*(*lista*(*compu*))

if EsVacío?(candidatos) **then** ▷ O(2)
res ← AgregarRapido(Vacío(), Vacía()) ▷ O(copy(Vacía())) = O(1)
else if Ultimo(recorrido).hostname = c2.hostname **then** ▷ O(n x (Cardinal(recorrido[j].interfaces)+L)) donde j es el indice de la compu con mayor cantidad de interfaces conectadas.
res ← AgregarRapido(Vacío(), recorrido) ▷ O(copy(recorrido))=O(Copiar(recorrido))=O($\sum_{i=1}^{long(recorrido)} copy(recorrido[i])$)=O($\sum_{i=1}^{long(recorrido)}$ ($Cardinal(recorrido[i].interfaces) + long(recorrido[i].hostname)$))=O($\sum_{i=1}^{long(recorrido)}$ ($Cardinal(recorrido[i].interfaces) + L$)))=O(n x (Cardinal(recorrido[j].interfaces)+L))donde j es el indice de la compu con mayor cantidad de interfaces conectadas.
else ▷ O(3 + n x L + $Cardinal(dUno.interfaces) + L$ + 2 x (AuxCaminos(r, c1, c2, recorrido, candidatos) - 1) + 1 + n) = O(n x L + $Cardinal(dUno.interfaces) + L$ + (AuxCaminos(r, c1, c2, recorrido, candidatos) - 1))
iterador : itLista(compu) ← crearIT(recorrido) ▷ O(1)
dameUno : compu ← Siguiente(crearIT(candidatos)) ▷ O(1)
loEncontre : bool ← false ▷ O(1)
while HaySiguiente(iterador) ∧ ¬loEncontre **do** ▷ O(L x n)
if dameUno.hostname = Siguiente(iterador).hostname **then** ▷ O(MAX(long(Siguiente(iterador).hostname), long(dameUno.hostname)) + 1) = O(L)
loEncontre ← true ▷ O(1)
end if
Avanzar(iterador) ▷ O(1)
end while
if ¬loEncontre **then** ▷ O($Cardinal(dUno.interfaces) + L$ + 2 x (AuxCaminos(r, c1, c2, recorrido, candidatos) - 1) + 1 + n) donde dU es la computadora entre los candidatos con mayor cantidad de interfaces conectadas.
camino1 : conj(lista(compu)) ← AuxCaminos(r, c1 , c2 , AgregarAtras(recorrido, dameUno), Significado(r, dameUno.hostname)) ▷ O(long(dameUno.hostname) + (AuxCaminos(r, c1, c2, recorrido, candidatos) - 1))
camino2 : conj(lista(compu)) ← AuxCaminos(r, c1 , c2 , recorrido, Eliminar(candidatos, dameUno)) ▷ O($\sum_{a' \in candidatos} equal(dameUno, a')$ + (AuxCaminos(r, c1, c2, recorrido, candidatos) - 1)) = O($Cardinal(dUno.interfaces) + L$ + (AuxCaminos(r, c1, c2, recorrido, candidatos) - 1)) donde dU es la computadora entre los candidatos con mayor cantidad de interfaces conectadas.
iteradorCamino : itConj(lista(compu)) ← crearIT(camino2) ▷ O(1)
while HayCamino(iteradorCamino) **do** ▷ O((1 + 1)x n)=O(2 x n) = O(n)
AgregarRapido(camino1, Siguiente(iteradorCamino)) ▷ O(copy(Siguiente(iteradorCamino))) ≤ O(1), pues O(siguiente(iteradorCamino))=O(1).
Avanzar(iteradorCamino) ▷ O(1)
end while
else
res ← AuxCaminos(r, c1 , c2 , recorrido, Eliminar(candidatos, dameUno)) ▷ O((AuxCaminos(r, c1, c2, recorrido, candidatos) - 1))
end if
end if

Complejidad: O((L x n + 1+ Cardinal(dU.interfaces)+ L)x n) = O((L x n + Cardinal(dU.interfaces))x n) donde dU es la computadora entre los candidatos con mayor cantidad de interfaces conectadas.
Justificación: Como cada vez que llamamos recursivamente a la funcion AuxCaminos estamos continuando con la contrucción del mismo a travez de recorrido, podemos decir que la función demora a lo sumo n en cada recurción. Esto se debe a que el camino no tiene repetidos.