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

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* $=_{\text{obs}}$ 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: $\texttt{dicc}_{trie}$`(puntero(compuDCNet))`,
laQueMásEnvió: `puntero(compuDCNet)`)

donde `compuDCNet` es `tupla`(*pc*: `puntero(compu)`,
conjPaquetes: `conj(paquete)`,
diccPaquetesDCNet: $\texttt{dicc}_{avl}$`(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

$$
\begin{aligned}
&\text{Rep} : \text{estr} \longrightarrow \text{bool}\\
&\text{Rep}(e) \equiv \text{true} \iff \\
&\quad (\#(\text{computadoras}(e.\text{topologia})) = \text{long}(e.\text{vectorCompusDCNet}) = \#(\text{claves}(e.\text{diccCompusDCNet}))) \wedge_{\text{L}}\\
&\quad (\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.\text{pc} = \text{puntero}(c)) \wedge\\
&\quad (\exists s\text{: string})(\text{def?}(s, e.\text{diccCompusDCNet}) \wedge s = c.\text{ip})\\
&\quad)\\
&\quad) \wedge_{\text{L}}\\
&\quad (\forall cd\text{: compuDCNet})(\text{está?}(cd, e.\text{vectorCompusDCNet}) \Rightarrow_{\text{L}}\\
&\quad (\exists s\text{: string})\ (\text{def?}(s, e.\text{diccCompusDCNet}) \wedge\\
&\quad s = cd.\text{pc}\rightarrow\text{ip} \wedge_{\text{L}} \text{obtener(s, } e.\text{diccCompusDCNet}) = \text{puntero}(cd))\\
&\quad) \wedge_{\text{L}}\\
&\quad (\exists cd\text{: compuDCNet})(\text{está?}(cd, e.\text{vectorCompusDCNet}) \wedge_{\text{L}}\\
&\quad *(cd.\text{pc}) = \text{compuQueMásEnvió}(e.\text{vectorCompusDCNet}) \wedge e.\text{laQueMásEnvió} = \text{puntero}(cd)) \wedge_{\text{L}}\\
&\quad (\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.\text{id} \neq p_2.\text{id})\\
&\quad)\\
&\quad)\\
&\quad) \wedge_{\text{L}}\\
&\quad (\forall cd\text{: compuDCNet})(\text{está?}(cd, e.\text{vectorCompusDCNet}) \Rightarrow\\
&\quad (\\
&\quad (\#(cd.\text{conjPaquetes}) = \#(\text{claves}(cd.\text{diccPaquetesDCNet}))) \wedge_{\text{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.\text{pc})) \wedge_{\text{L}}\\
&\quad (\exists sc\text{: secu(compu)})(sc \in \text{caminosMinimos}(e.\text{topologia}, p.\text{origen}, p.\text{destino}) \wedge \text{está}(*(cd.\text{pc}), sc))) \wedge\\
&\quad (\exists n\text{: nat})\ ((\text{def?}(n, cd.\text{diccPaquetesDCNet}) \wedge p.\text{id} = n) \wedge_{\text{L}}\\
&\quad (\text{Siguiente}(\text{obtener}(n, e.\text{diccPaquetesDCNet}).\text{it}) = p \wedge\\
&\quad ((p.\text{origen} = *(cd.\text{pc}) \wedge \text{obtener}(n, e.\text{diccPaquetesDCNet}).\text{recorrido} = *(cd.\text{pc}) \bullet <>) \vee\\
&\quad (p.\text{origen} \neq *(cd.\text{pc}) \wedge\\
&\quad \text{obtener}(n, e.\text{diccPaquetesDCNet}).\text{recorrido} \in \text{caminosMinimos}(e.\text{topologia}, p.\text{origen}, *(cd.\text{pc}))))\\
&\quad)\\
&\quad) \wedge_{\text{L}}\\
&\quad (\emptyset?(cd.\text{conjPaquetes}) \Leftrightarrow \text{vacía?}(cd.\text{colaPaquetesDCNet})) \wedge\\
&\quad (\neg\text{vacía?}(cd.\text{colaPaquetesDCNet}) \Rightarrow_{\text{L}}\\
&\quad (\exists n\text{: nat})(\text{def?}(n, cd.\text{diccPaquetesDCNet}) \wedge_{\text{L}}\\
&\quad (\\
&\quad \text{Siguiente}(\text{obtener}(n, cd.\text{diccPaquetesDCNet}).\text{it}) = \text{paqueteMásPrioridad}(cd.\text{conjPaquetes}) \wedge\\
&\quad \text{proximo}(cd.\text{colaPaquetesDCNet}) = \text{puntero}(\text{obtener}(n, cd.\text{diccPaquetesDCNet}))\\
&\quad))\\
&\quad) \wedge\\
&\quad (cd.\text{enviados} \geq \text{enviadosCompu}(*(cd.\text{pc}), e.\text{vectorCompusDCNet}))\\
&\quad)\\
&\quad)
\end{aligned}
$$

compuQueMásEnvió : secu(compuDCNet) *scd* $\longrightarrow$ compu {¬vacía?(*scd*)}

maxEnviado : secu(compuDCNet) *scd* $\longrightarrow$ nat {¬vacía?(*scd*)}

enviaronK : secu(compuDCNet) × nat $\longrightarrow$ conj(compu)

paqueteMásPrioridad : conj(paquete) *cp* $\longrightarrow$ paquete {¬∅?(*cp*)}

paquetesConPrioridadK : conj(cp) × nat $\longrightarrow$ conj(paquete)

altaPrioridad : conj(paquetes) *cp* $\longrightarrow$ nat {¬∅?(*cp*)}

enviadosCompu : compu × secu(compuDCNet) $\longrightarrow$ nat

aparicionesCompu : compu × conj(nat) *cn* × dicc(nat × paqueteDCNet) *dp* $\longrightarrow$ nat {claves(*dp*) ⊆ *cn*}

compuQueMásEnvió(*scd*) ≡ dameUno(enviaronK(*scd*, maxEnviado(*scd*)))

maxEnviado(*scd*) ≡ **if** vacía?(fin(*scd*)) **then** prim(*scd*).enviados **else** max(prim(*scd*), maxEnviado(fin(*scd*))) **fi**

```
enviaronK(scd, k) ≡ if vacía?(scd) then
                        ∅
                    else
                        if prim(scd).enviados = k then
                            Ag(*(prim(scd).pc), enviaronK(fin(scd), k))
                        else
                            enviaronK(fin(scd), k)
                        fi
                    fi
```

paqueteMásPrioridad(*dcn*, *cp*) ≡ dameUno(paquetesConPrioridadK(*cp*, altaPrioridad(*cp*)))

```
altaPrioridad(cp) ≡ if ∅?(sinUno(cp)) then
                        dameUno(cp).prioridad
                    else
                        min(dameUno(cp).prioridad, altaPrioridad(sinUno(cp)))
                    fi
```

```
paquetesConPrioridadK(cp, k) ≡ if ∅?(cp) then
                                   ∅
                               else
                                   if dameUno(cp).prioridad = k then
                                       Ag(dameUno(cp), paquetesConPrioridadK(sinUno(cp), k))
                                   else
                                       paquetesConPrioridadK(sinUno(cp), k)
                                   fi
                               fi
```

```
enviadosCompu(c, scd) ≡ if vacía?(scd) then
                            0
                        else
                            if prim(scd) = c then
                                enviadosCompu(c, fin(scd))
                            else
                                aparicionesCompu(c, claves(prim(scd).diccPaquetesDCNet),
                                prim(scd).diccPaquetesDCNet) + enviadosCompu(c, fin(scd))
                            fi
                        fi
```

```
aparicionesCompu(c, cn, dpd) ≡ if ∅?(cn) then
                                  0
                                else
                                  if está?(c, significado(dameUno(cn), dpd).recorrido) then
                                    1 + aparicionesCompu(c, sinUno(cn), dpd)
                                  else
                                    aparicionesCompu(c, 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* $=_{\text{obs}}$ 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 $\equiv$ {$r =_{\text{obs}}$ 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 $\equiv$ {$r =_{\text{obs}}$ 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_{Trie}`(ip, nodoRed)`)

donde `nodoRed` es `tupla`(c: `puntero(compu)` ,
caminos: dicc_{Trie}`(ip, conj(lista(compu)))` ,
conexiones: dicc_{Lineal}`(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` usaconedas de la compu c (que tienen que estar en $c.interfaces$)

Rep : estr $\longrightarrow$ bool
Rep(e) $\equiv$ true $\Longleftrightarrow$

2.2.3. Función de Abstracción

Abs : estr e $\longrightarrow$ red {Rep(e)}
Abs(e) $=_{\text{obs}}$ r: red |

2.2.4. Función de Abstracción

2.3. Algoritmos

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

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

```
iConectar (in/out r: red, in c0: compu, in c1: compu, in i0: compu, in i1: compu)
    nr0:nodoRed ← Significado(r.dns, c0.ip)                       O(?)
    nr1:nodoRed ← Significado(r.dns, c1.ip)                       O(?)
    DefinirRapido(rn0.conexiones, i0, nr1)                        O(?)
    DefinirRapido(nr1.conexiones, i1, nr0)                        O(?)
    CrearCaminosDelDNS(r)
```
Complejidad : $O(?)$

```
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 : $O(?)$

```
AsignarCaminosMinimos (in/out nr: nodoRed)
  itCompus: itConj(compu) ← CrearIt(r.compus)
  while HaySiguiente?(itCompus) do
    Definir(nr.caminos, Siguiente(itCompus).ip
      CrearCaminosMinimos(nr, Siguiente(itCompus).ip))
    Avanzar(itCompus)
  end while
```
Complejidad : $O(?)$

```
CrearCaminosMinimos (in desde: nodoRed, in hasta: compu) → res: conj(lista(compu))
  camMins: conj(lista(compu)) ← Vacio()
  itCamMins: itConj(conj(lista(compu))) ← CrearIt(camMins)
  pendientes: conj(nodoRed) ← Conexas(desde.conexiones)
  visitados: conj(nodoRed) ← Vacio()
  itNrs: itConj(nodoRed) ← CrearIt(pendientes)
  cam: lista(compu) ← Vacia()
  AgregarAdelante(cam, *(desde.pc))
  while HaySiguiente?(itNrs) do
    AgregarRapido(visitados, Siguiente(itNrs))
    AgregarAdelante(cam, *(Siguiente(itNrs).pc))
    if (Siguiente(itNrs).pc→ip = hasta.ip) then
      if (Vacio?(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)
    else
      if(Vacia?(NoVisitadas(pendientes, Siguiente(itNrs), visitados, r)))
        Comienzo(cam)
      else
        UnirConjs(pendientes, NoVisitadas(Siguiente(itNrs), visitados))
      end if
    end if
    Avanzar(itNrs)
  end while
```
Complejidad : $O(?)$

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

```
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 : $O(?)$

```
UnirConjs (in/out cnr: conj(nodoRed), in news: conj(nodoRed))
  it: itConj(nodoRed) ← CrearIt(news)
  while HaySiguiente?(it) do
    AgregarRapido(res, Siguiente(it))
    Avanzar(it)
  end while
```
Complejidad : $O(?)$

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

```
iConectadas? (in r: red, in c0: compu, in c1: compu) → res: bool
  nr0: nodoRed ← Significado(r.dns, c0.ip)                                  O(L)
  it :itDicc(interfaz, puntero(nodoRed))) ← CrearIt(nr0.conexiones)         O(1)
  res ← false                                                               O(1)
  while HaySiguiente?(it) do                                                  O(L)
    if c1.ip = SiguienteSignificado(it)->pc->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
  nr0: nodoRed ← Significado(r.dns, c0.ip)                                  O(L)
  it :itDicc(itDicc(interfaz, puntero(nodoRed))
    ← CrearIt(nr0.conexiones)                                               O(1)
  while HaySiguiente?(it) do                                                  O(L)
```

```
    if c1.ip = SiguienteSignificado(it)->pc->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)
  nr:nodoRed ← Significado(r.dns, c.ip)                             O(1)
  res:conj(compu) ← Vacio()
  it :itDicc(itDicc(interfaz, puntero(nodoRed))
    ← CrearIt(nr.conexiones)
  while HaySiguiente?(it) do                                        O(L)
    AgregarRapido(res,*(SiguienteSignificado(it)->pc))
    Avanzar(it)                                                     O(1)
  end while
```
Complejidad : $O(1)$

```
iUsaInterfaz? (in r: red, in c: compu, in i: interfaz) → res: bool
  nr: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))
  nr:nodoRed ← Significado(r.dns, c0.ip)                            O(1)
  res ← Significado(pnr.caminos, c1.ip)                             O(1)
```
Complejidad : $O(1)$

```
HayCamino? (in r: red, in c0: compu, in c1: compu) → res: bool
  nr: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
  auxR.compus ← r.compus
  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 : $O(1)$