DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2

Grupo Número 1

06 de Septiembre de 2015

Algoritmos y Estructuras de Datos II

Integrante	LU	Correo electrónico
Joel Esteban Camera	257/14	joel.e.camera@gmail.com
Manuel Mena	313/14	manuelmena1993@gmail.com
Kevin Frachtenberg Goldsmit	247/14	kevinfra94@gmail.com
Nicolás Bukovits	546/14	nicobuk@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - Pabellón I
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Argentina
Tel/Fax: (54 11) 4576-3359
http://exactas.uba.ar

Índice

1. Módulo CampusSeguro

1.1. Interfaz

se explica con: CampusSeguro.

géneros: `campusSeguro`.

1.1.1. Operaciones básicas de CampusSeguro

comenzarRastrillaje(**in** c: `campus`, **in** d: `dicc(placa,AS)`) $\rightarrow$ res : `campusSeguro`
Pre $\equiv$ {($\forall a$: agente) (def?(a,d) $\Rightarrow_L$ (posVálida(obtener(a,d)) $\wedge$ $\neg$ocupada?(obtener(a,d,c))) $\wedge$ ($\forall a, a2$: agente) ((def?(a,d) $\wedge$ def?($a2$,d) $\wedge a \neq a2$) $\Rightarrow_L$ obtener(a,d) $\neq$ obtener($a2$,d))}
Post $\equiv$ {res $=_{obs}$ comenzarRastrillaje(c,d)}
Complejidad: $O()$
Descripción: Crea un nuevo campusSeguro tomando un campus y un diccionario con agentes.

ingresarEstudiante(**in** e: `nombre`, **in** p: `posición`, **in/out** cs: `campusSeguro`)
Pre $\equiv$ {$cs =_{obs} cs_o \wedge e \notin$ (estudiantes(cs) $\bigcup$ hippies(cs)) $\wedge$ esIngreso?(p,campus(cs)) $\wedge$ $\neg$estaOcupada?(p,cs)}
Post $\equiv$ {$cs =_{obs}$ ingresarEstudiante(e,p,cs_o)}
Complejidad: $O(|n_m|) + O(log(N_a))$
Descripción: Ingresa un nuevo estudiante al campusSeguro

ingresarHippie(**in** h: `nombre`, **in** p: `posición`, **in/out** cs: `campusSeguro`)
Pre $\equiv$ {$cs =_{obs} cs_o \wedge h \notin$ (estudiantes(cs) $\bigcup$ hippies(cs)) $\wedge$ esIngreso?(p,campus(cs)) $\wedge$ $\neg$estaOcupada?(p,cs)}
Post $\equiv$ {$cs =_{obs}$ ingresarHippie(h,p,cs_o)}
Complejidad: $O(|n_m|) + O(log(N_a))$
Descripción: Ingresa un nuevo hippie el campusSeguro

moverEstudiante(**in** e: `nombre`, **in** dir: `dirección`, **in/out** cs: `campusSeguro`)
Pre $\equiv$ {$cs =_{obs} cs_o \wedge e \in$ estudiantes(cs) $\wedge$ (seRetira(e,dir,cs) $\vee$ (posValida(proxPosicion(posEstudianteYHippie(e,cs),dir,campus(cs)), campus(cs)) $\wedge$ $\neg$estaOcupada?(proxPosicion(posEstudianteYHippie(e,cs),dir,campus(cs)),cs)}
Post $\equiv$ {$cs =_{obs}$ moverEstudiante(e,dir,cs_o)}
Complejidad: $O(|n_m|) + O(log(N_a))$
Descripción: Mueve un estudiante dentro del campus o lo hace salir y se actualizan los atrapados, sanciones (si es que las hay) y hippies atrapados (si es que los hay).

moverHippie(**in** h: `nombre`, **in** d: `direccion`, **in/out** cs: `campusSeguro`)
Pre $\equiv$ {$cs =_{obs} cs_o \wedge h \in$ hippies(cs) $\wedge$ $\neg$todasOcupadas?(vecinos(posEstudianteYHippie(h,cs),campus(cs)), cs)}
Post $\equiv$ {$cs =_{obs}$ moverHippie(h,d,cs_o}
Complejidad: $O(|n_m|) + O(log(N_a)) + O(N_e)$
Descripción: Mueve un hippie dentro del campus y se actualizan los atrapados, sanciones(si es que las hay) y hippies atrapados (si es que los hay).

moverAgente(**in** a: `agente`, **in/out** cs: `campusSeguro`)
Pre $\equiv$ {$cs =_{obs} cs_o \wedge a \in$ agentes(cs) $\wedge_L$ cantSanciones(a,cs) $\leq 3 \wedge$ $\neg$todasOcupadas?(vecinos(posEstudianteYHippie(h,cs),campus(cs)),cs)}
Post $\equiv$ {$cs =_{obs}$ moverAgente(a,cs_o)}
Complejidad: $O(|n_m|) + O(log(N_a)) + O(N_h)$
Descripción: Mueve un agente dentro del campus y se actualizan los atrapados, sanciones (si es que las hay) y hippies atrapados (si es que los hay).

CAMPUS(**in** cs: `campusSeguro`) $\rightarrow$ res : `campus`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{obs}$ campus(cs)}
Complejidad: $O(1)$
Descripción: Devuelve el campus del campusSeguro.
Aliasing: res es una referencia no modificable.

ESTUDIANTES(**in** cs: `campusSeguro`) $\rightarrow$ res : `conj(nombre)`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{obs}$ estudiantes(cs)}
Complejidad: $O(1)$
Descripción: Devuelve el conjunto de los estudiantes que estan en el campus.
Aliasing: res es una referencia no modificable.

HIPPIES(**in** cs: `campusSeguro`) $\rightarrow$ res : `conj(nombre)`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{obs}$ hippies(cs)}
Complejidad: $O(1)$
Descripción: Devuelve el conjunto de los hippies que estan en el campus.
Aliasing: res es una referencia no modificable.

AGENTES(**in** cs: `campusSeguro`) $\rightarrow$ res : `conj(agentes)`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{obs}$ agentes(cs)}
Complejidad: $O(1)$
Descripción: Devuelve el conjunto de los agentes que estan en el campus.
Aliasing: res es una referencia no modificable.

POSESTUDIANTEYHIPPIE(**in** id: `nombre`, **in** cs: `campusSeguro`) $\rightarrow$ res : `posición`
Pre $\equiv$ {$id \in$ (estudiantes(cs) $\bigcup$ hippies(cs)}
Post $\equiv$ {$res =_{obs}$ posEstudianteYHippie(id,cs)}
Complejidad: $O(|n_m|)$, donde $|m_n|$ es la longitud mas larga entre todos los nombres.
Descripción: Devuelve la posicion del estudiante o hippie.

POSAGENTE(**in** a: `agente`, **in** cs: `campusSeguro`) $\rightarrow$ res : `posición`
Pre $\equiv$ {$a \in$ agentes(cs)}
Post $\equiv$ {$res =_{obs}$ posAgente(a,cs)}
Complejidad: $O(1)$ en caso promedio.
Descripción: Devuelve la posicion del agente pasado como parametro.

CANTSANCIONES(**in** a: `agente`, **in** cs: `campusSeguro`) $\rightarrow$ res : `nat`
Pre $\equiv$ {$a \in$ agentes(cs)}
Post $\equiv$ {$res =_{obs}$ cantSanciones(a,cs)}
Complejidad: $O(1)$ en caso promedio.
Descripción: Devuelve la cantidad de sanciones que posee el agente pasado como parametro.

CANTHIPPIESATRAPADOS(**in** a: `agente`, **in** cs: `campusSeguro`) $\rightarrow$ res : `nat`
Pre $\equiv$ {$a \in$ agentes(cs)}
Post $\equiv$ {$res =_{obs}$ cantHippiesAtrapados(a,cs)}
Complejidad: $O(1)$ en caso promedio.
Descripción: Devuelve la cantidad de hippies que atrapo el agente pasado como parametro.

MÁSVIGILANTE(**in** cs : `campusSeguro`) $\rightarrow$ res : `agente`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{obs}$ másVigilante(cs)}
Complejidad: $O(1)$
Descripción: Devuelve la placa del agente que ha atrapado mas hippies.

CONMISMASSANCIONES(**in** a : `agente`, **in** cs : `campusSeguro`) $\rightarrow$ res : `conj(agentes)`
Pre $\equiv$ {$a \in$ agentes(cs)}
Post $\equiv$ {$res =_{obs}$ conMismasSanciones(a,cs)}
Complejidad: $O(1)$
Descripción: Devuelve el conjunto de los agentes que tienen el mismo numero de sanciones que el agente pasado como parametro.
Aliasing: res es una referencia no modificable.

CONKSANCIONES(**in** k : `nat`, **in** cs : `campusSeguro`) $\rightarrow$ res : `conj(agentes)`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{obs}$ conKSanciones(k,cs)}
Complejidad: $O(N_a)$ la primera vez que se la llama y $O(log(N_a))$ en futuras llamadas mientras no ocurran sanciones.
Descripción: Devuelve el conjuto de agenes que tienen k sanciones.
Aliasing: res es una referencia no modificable.

1.1.2. Representación de campusSeguro

`campusSeguro` **se representa con** `estr`

donde `estr` es `tupla`(*campus*: `campus`,
agentes: `AS`,
hippies: `conj(hippies)`,
estudiantes: `conj(estudiantes)`)

1.1.3. Invariante de Representación

(I)

Rep : estr $\longrightarrow$ bool

Rep(e) $\equiv$ true $\Longleftrightarrow$

($\forall c$: campusSeguro)()

Rep : estr $\longrightarrow$ bool

Rep(e) $\equiv$ true $\Longleftrightarrow$
($\forall c$: compu)($c \in$ computadoras(e.topologia) $\Leftrightarrow$
(
($\exists cd$: compuDCNet) (está?(cd, e.vectorCompusDCNet) $\wedge$ (cd.pc = puntero(c)) $\wedge$
($\exists s$: string)(def?(s, e.diccCompusDCNet) $\wedge$ ($s = c$.ip)))
)
) $\wedge_{\text{L}}$
($\forall cd$: compuDCNet)(está?(cd, e.vectorCompusDCNet) $\Leftrightarrow$
($\exists s$: string)(($s = cd$.pc$\rightarrow$ip) $\wedge$ def?(s, e.diccCompusDCNet) $\wedge_{\text{L}}$
obtener(s, e.diccCompusDCNet) = puntero(cd))
) $\wedge_{\text{L}}$
($\exists cd$: compuDCNet)(está?(cd, e.vectorCompusDCNet) $\wedge_{\text{L}}$
*(cd.pc) = compuQueMásEnvió(e.vectorCompusDCNet) $\wedge$ e.laQueMásEnvió = puntero(cd)) $\wedge_{\text{L}}$
($\forall cd_1$: compuDCNet)(está?(cd_1, e.vectorCompusDCNet) $\Rightarrow$
($\forall p_1$: paquete)($p_1 \in cd_1$.conjPaquetes $\Rightarrow$
($\forall cd_2$: compuDCNet)((está?(cd_2, e.vectorCompusDCNet) $\wedge$ $cd_1 \neq cd_2$) $\Rightarrow$
($\forall p_2$: paquete)($p_2 \in cd_2$.conjPaquetes $\Rightarrow$ p_1.id $\neq$ p_2.id)
)
)
) $\wedge_{\text{L}}$
($\forall cd$: compuDCNet)(está?(cd, e.vectorCompusDCNet) $\Rightarrow$
(
($\forall p$: paquete)($p \in cd$.conjPaquetes $\Leftrightarrow$
(
((p.origen $\in$ computadoras(e.topologia) $\wedge$ p.destino $\in$ computadoras(e.topologia) $\wedge$
p.destino $\neq$ *(cd.pc)) $\wedge_{\text{L}}$
($\exists sc$: secu(compu))($sc \in$ caminosMinimos(e.topologia, p.origen, p.destino) $\wedge$ está(*(cd.pc), sc))) $\wedge$
($\exists n$: nat) ((def?(n, cd.diccPaquetesDCNet) $\wedge$ p.id = n) $\wedge_{\text{L}}$
($\exists pdn$: paqueteDCNet)($pdn \in e$.conjPaquetesDCNet $\wedge$ Siguiente(pdn.it) = p $\wedge$
((p.origen = *(cd.pc) $\wedge$ pdn.recorrido = *(cd.pc) $\bullet$ <>) $\vee$
(p.origen $\neq$ *(cd.pc) $\wedge$ pdn.recorrido $\in$ caminosMinimos(e.topologia, p.origen, *(cd.pc)))) $\wedge$
Siguiente(obtener(n, cd.diccPaquetesDCNet)) = pdn
)
)
)
) $\wedge_{\text{L}}$
(¬vacía?(cd.colaPaquetesDCNet) $\Leftrightarrow$
($\exists p$: paquete)(($p \in cd$.conjPaquetes) $\wedge$ (p = paqueteMásPrioridad(cd.conjPaquetes)) $\wedge$
($\exists pdn$: paqueteDCNet)(($pdn \in e$.conjPaquetesDCNet) $\wedge$ (Siguiente(pdn.it) = p) $\wedge$
(Siguiente(proximo(cd.colaPaquetesDCNet)) = pdn))
)
) $\wedge_{\text{L}}$
(cd.enviados $\geq$ enviadosCompu(*(cd.pc), e.vectorCompusDCNet)) $\wedge$
(¬HaySiguiente?(cd.paqueteAEnviar)))
)

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

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

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

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

paquetesConPrioridadK : conj(paquete) × nat ⟶ conj(paquete)

altaPrioridad : conj(paquetes) *cp* ⟶ nat {¬∅?(*cp*)}

enviadosCompu : compu × secu(compuDCNet) ⟶ nat

aparicionesCompu : compu × conj(nat) *cn* × dicc(nat × itConj(paqueteDCNet)) *dp* ⟶ 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, Siguiente(obtener(dameUno(cn), dpd)).recorrido) then
                                       1 + aparicionesCompu(c, sinUno(cn), dpd)
                                   else
                                       aparicionesCompu(c, sinUno(cn), dpd)
                                   fi
                               fi
```

1.1.4. Función de Abstracción

Abs : estr e $\longrightarrow$ dcnet {Rep(e)}

Abs(e) $=_{obs}$ dcn: dcnet | red(dcn) = e.topología $\wedge$
($\forall cdn$: compuDCNet)(está?(cdn, e.vectorCompusDCNet) $\Rightarrow_L$
enEspera(dcn, *(cdn.pc)) = cdn.conjPaquetes $\wedge$
cantidadEnviados(dcn, *(cdn.pc)) = cdn.enviados $\wedge$
($\forall p$: paquete)($p \in cdn$.conjPaquetes $\Rightarrow_L$
caminoRecorrido(dcn, p) = Siguiente(obtener(p.id, cdn.diccPaquetesDCNet)).recorrido
)
)

1.2. Algoritmos

iIniciarDCNet (**in** *topo*: `red`) → res: estr

```
res.topologia ← Copiar(topo)                                          O(n! * n^6)
res.vectorCompusDCNet ← Vacia()                                       O(1)
res.diccCompusDCNet ← CrearDicc()                                     O(1)
res.laQueMasEnvio ← NULL                                              O(1)
res.conjPaquetesDCNet ← Vacio()                                       O(1)

itConj(compu): it ← CrearIt(Computadoras(topo))                       O(1)

if(HaySiguiente?(it)) then                                            O(1)
    res.laQueMasEnvio ← puntero(Siguiente(it))                        O(1)
end if

while HaySiguiente?(it) do                                            O(1)
    compuDCNet: compudcnet ← <puntero(Siguiente(it)), Vacio(), CrearDicc(),
        Vacia(), CrearIt(Vacio()), 0>                                 O(1)
    AgregarAtras(res.vectorCompusDCNet, compudcnet)                   O(n)
    Definir(res.diccCompusDCNet, Siguiente(it).ip, puntero(compudcnet))  O(L)
    Avanzar(it)                                                       O(1)
end while                                                             O(n * (n + L))
```

Complejidad : $O(n * (n + L))$

iCrearPaquete (**in/out** *dcn*: `dcnet`, **in** *p*: `paquete`)

```
puntero(compuDCNet): compudcnet ←
    Significado(dcn.diccCompusDCNet, p.origen.ip)                     O(L)
itConj(paquete): itPaq ← AgregarRapido(compudcnet→conjPaquetes, p)     O(1)
lista(compu): recorr ← AgregarAtras(Vacia(), p.origen)                O(1)
paqueteDCNet: paqDCNet ← <itPaq, recorr>                              O(1)

itConj(paqueteDCNet): itPaqDCNet ←
    AgregarRapido(dcn.conjPaquetesDCNet, paqDCNet)                    O(1)
Definir(compudcnet→diccPaquetesDCNet, p.id, itPaqDCNet)               O(log(k))
Encolar(compudcnet→colaPaquetesDCNet, p.prioridad, itPaqDCNet)        O(log(k))
```

Complejidad : $O(log(k) + L)$

iAvanzarSegundo (**in/out** *dcn*: dcnet)

```
nat: maxEnviados ← 0
nat: i ← 0                                                                  O(1)
while i < Longitud(dcn.vectorCompusDCNet) do                                O(1)
    if (¬EsVacia?(dcn.vectorCompusDCNet[i].colaPaquetesDCNet)) then
        dcn.vectorCompusDCNet[i].paqueteAEnviar ←
            Desencolar(dcn.vectorCompusDCNet[i].colaPaquetesDCNet)          O(log(k))
    end if
    i++                                                                     O(1)
end while                                                                   O(n * log(k))

i ← 0                                                                       O(1)
while i < Longitud(dcn.vectorCompusDCNet) do                                O(1)
    if (HaySiguiente?(dcn.vectorCompusDCNet[i].paqueteAEnviar)) then        O(1)

        dcn.vectorCompusDCNet[i].enviados++                                 O(1)
        if(dcn.vectorCompusDCNet[i].enviados > maxEnviados) then            O(1)
            dcn.laQueMasEnvio ← puntero(dcn.vectorCompusDCNet[i])           O(1)
        end if

        paquete: pAEnviar ←
            Siguiente(Siguiente(dcn.vectorCompusDCNet[i].paqueteAEnviar).it)  O(1)
        itConj(lista(compu)): itercaminos ←
            CrearIt(CaminosMinimos(dcn.topologia,
            *(dcn.vectorCompusDCNet[i].pc), pAEnviar.destino))              O(1)
        compu: siguientecompu ← Siguiente(itercaminos)[1]                   O(1)

        if(pAEnviar.destino ≠ siguientecompu) then                          O(1)

            compuDCNet: siguientecompudcnet ←
                *(Obtener(dcn.diccCompusDCNet, siguientecompu.ip))          O(L)

            itConj(paquete): itpaquete ←
                AgregarRapido(siguientecompudcnet.conjPaquetes, pAEnviar)   O(1)

            itConj(paqueteDCNet): paqAEnviar ←
                Obtener(dcn.vectorCompusDCNet[i].diccPaquetesDCNet,
                pAEnviar.id)                                                O(log(k))

            AgregarAtras(Siguiente(paqAEnviar).recorrido, siguientecompu)   O(1)

            Encolar(siguientecompudcnet.colaPaquetesDCNet,
                pAEnviar.prioridad, paqAEnviar)                             O(log(k))
            Definir(siguientecompudcnet.diccPaquetesDCNet,
                pAEnviar.id, paqAEnviar)                                    O(log(k))
        end if

        Borrar(dcn.vectorCompusDCNet[i].diccPaquetesDCNet,
            Siguiente(dcn.vectorCompusDCNet[i].paqueteAEnviar→it).id)       O(log(k))
        EliminarSiguiente(Siguiente(dcn.vectorCompusDCNet[i].paqueteAEnviar).it)
                                                                            O(1)
        EliminarSiguiente(dcn.vectorCompusDCNet[i].paqueteAEnviar)          O(1)

        dcn.vectorCompusDCNet[i].paqueteAEnviar ← CrearIt(Vacio())          O(1)

    end if
    i++                                                                     O(1)
```

```
    end while                                                    O(n * (L + log(k)))
```

Complejidad : $O(n * (L + log(k)))$

Red (**in** *dcn*: `dcnet`) → res: red

```
    res ← dcn.topologia                                          O(1)
```

Complejidad : $O(1)$

CaminoRecorrido (**in** *dcn*: `dcnet`, **in** *p*: `paquete`) → res: lista(compu)

```
    nat: i ← 0                                                   O(1)
    while i < Longitud(dcn.vectorCompusDCNet) do                 O(1)
        if Definido?(dcn.vectorCompusDCNet[i].diccPaquetesDCNet, p.id) then   O(log(k))
            res ← Siguiente(Obtener(dcn.vectorCompusDCNet[i].diccPaquetesDCNet,
                p.id)).recorrido                                 O(log(k))
        end if
        i++                                                      O(1)
    end while                                                    O(n * log(k))
```

Complejidad : $O(n * log(k))$

CantidadEnviados (**in** *dcn*: `dcnet`, **in** *c*: `compu`) → res: nat

```
    res ← Obtener(dcn.diccCompusDCNet, c.ip)→enviados            O(L)
```

Complejidad : $O(L)$

EnEspera (**in** *dcn*: `dcnet`, **in** *c*: `compu`) → res: nat

```
    res ← Obtener(dcn.diccCompusDCNet, c.ip)→conjPaquetes        O(L)
```

Complejidad : $O(L)$

PaqueteEnTransito (**in** *dcn*: `dcnet`, **in** *p*: `paquete`) → res: bool

```
    res ← false
    nat: i ← 0                                                   O(1)
    while i < Longitud(dcn.vectorCompusDCNet) do                 O(1)
        if Definido?(dcn.vectorCompusDCNet[i].diccPaquetesDCNet, p.id) then   O(log(k))
            res ← true                                           O(1)
        end if
        i++                                                      O(1)
    end while                                                    O(n * log(k))
```

Complejidad : $O(n * log(k))$

LaQueMasEnvio (**in** *dcn* : `dcnet`) → res: compu

```
res ← *(dcn.laQueMasEnvio→pc)                                   O(1)
```

Complejidad : $O(1)$

• $=_i$ • (**in** dcn_1 : `dcnet`, **in** dcn_2 : `dcnet`) → res: bool

```
bool: boolTopo ← dcn1.topologia = dcn2.topologia                O(n + L^2)
bool: boolVec ← dcn1.vectorCompusDCNet = dcn2.vectorCompusDCNet  O(n * k * (k + n))
bool: boolConj ← dcn1.conjPaquetesDCNet = dcn2.conjPaquetesDCNet O(k^3 * (k + n))
bool: boolMasEnvio ← *(dcn1.laQueMasEnvio) = *(dcn2.laQueMasEnvio) O(1)

res ← boolTopo ∧ boolVec ∧ boolTrie ∧ boolConj ∧ boolMasEnvio    O(1)
```

Complejidad : $O(n * k^3 * (k + n))$

• $=_{compudcn}$ • (**in** c_1 : `compuDCNet`, **in** c_2 : `compuDCNet`) → res: bool

```
bool: boolPC ← *(c1.pc) = *(c2.pc)                                          O(1)
bool: boolConj ← c1.conjPaquetes = c1.conjPaquetes                          O(k^2)
bool: boolAVL ← true                                                        O(1)
bool: boolCola ← true                                                       O(1)
bool: boolPaq ← Siguiente(c1.paqueteAEnviar) =paqdcn Siguiente(c2.paqueteAEnviar)
                                                                            O(n)
bool: boolEnviados ← c1.enviados = c2.enviados                              O(1)

if boolConj then                                                            O(1)
    itConj: itconj1 ← CrearIt(c1.conjPaquetes)                              O(1)
    while HaySiguiente?(itconj1) do                                         O(1)
        if Definido?(c2.diccPaquetesDCNet, Siguiente(itconj1).id then       O(log(n))
            if ¬(Siguiente(Obtener(c1.diccPaquetesDCNet, Siguiente(itconj1).id))
                =paqdcn
                Siguiente(Obtener(c1.diccPaquetesDCNet, Siguiente(itconj1).id)))
                then                                                        O(n)
                boolAVL ← false                                             O(1)
            end if
        else
            boolAVL ← false                                                 O(1)
        end if
        Avanzar(itconj1)                                                    O(1)
    end while                                                               O(n * k)
end if

if EsVacia(c1.colaPrioridad) then                                           O(1)
    if ¬EsVacia(c2.colaPrioridad) then                                      O(1)
        boolCola ← false                                                    O(1)
    end if
else
    if EsVacia(c1.colaPrioridad) then                                       O(1)
        boolCola ← false                                                    O(1)
    else
        if ¬(Siguiente(Proximo(c1.colaPrioridad)) =paqdcn
            Siguiente(Proximo(c2.colaPrioridad))) then                      O(n)
```

```
                boolCola ← false                                                    O(1)
            end if
        end if
    end if

    res ← boolPC ∧ boolConj ∧ boolAVL ∧ boolCola ∧ boolPaq ∧ boolEnviados           O(1)
```

Complejidad : $O(k^2 + n * k) = O(k * (k + n))$

• $=_{paqdcn}$ • (**in** p_1 : `paqueteDCNet`, **in** p_2 : `paqueteDCNet`,) → res: bool

```
    bool: boolPaq ← Siguiente(p1.it) = Siguiente(p2.it)                             O(1)
    bool: boolRecorrido ← p1.recorrido = p2.recorrido                               O(n)

    res ← boolPaq ∧ boolRecorrido                                                   O(1)
```

Complejidad : $O(n)$

2. Módulo Diccionario String(α)

2.1. Interfaz

se explica con: DICCIONARIO(STRING,α).

géneros: `diccString`(α).

Se representa mediante un árbol n-ario con invariante de trie

2.1.1. Operaciones básicas de Diccionario String(α)

CREARDICCIONARIO(**in** c: `campus`, **in** d: `diccString`(α)) $\rightarrow$ res : `true`
Pre $\equiv$ {true}
Post $\equiv$ {res $=_{obs}$ vacío()}
Complejidad: $O(1)$
Descripción: Crea un diccionario vacío.

DEFINIDO?(**in** d: `diccString`(α), **in** c: `string`)) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {res $=_{\text{obs}}$ def?(d, c)}
Complejidad: $O(L)$
Descripción: Devuelve true si la clave está definida en el diccionario y false en caso contrario.

DEFINIR(**in** d: `diccString`(α), **in** c: `string`, **in** s: α)
Pre $\equiv$ {d $=_{\text{obs}}$ d_0}
Post $\equiv$ {d $=_{\text{obs}}$ definir(c, s, d_0)}
Complejidad: $O(L)$
Descripción: Define la clave c con el significado s
Aliasing: Almacena una copia de s.

OBTENER(**in** d: `diccString`(α), **in** c: `string`) $\rightarrow$ res : α
Pre $\equiv$ {def?(c, d)}
Post $\equiv$ {alias(res $=_{\text{obs}}$ obtener(c, d))}
Complejidad: $O(L)$
Descripción: Devuelve el significado correspondiente a la clave c.
Aliasing: Devuelve el significado almacenado en el diccionario, por lo que res es modificable si y sólo si d lo es.

$\bullet = \bullet$(**in/out** d: `diccString`(α), **in/out** d': `diccString`(α)) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {res $=_{\text{obs}}$ (d $=_{\text{obs}}$ d')}
Complejidad: $O(L * n * (\alpha =_{\text{obs}} \alpha'))$
Descripción: Indica si d es igual d'

COPIAR(**in** $dicc$: `diccString`(α)) $\rightarrow$ res : `diccString`(α)
Pre $\equiv$ {true}
Post $\equiv$ {res $=_{\text{obs}}$ dicc}
Complejidad: $O(n * L * copy(\alpha))$
Descripción: Devuelve una copia del diccionario

2.1.2. Representación de Diccionario String(α)

`Diccionario String(`α`)` **se representa con** `estr`

donde `estr` es `tupla`(*raiz*: `arreglo(puntero(Nodo))`, *listaIterable*: `listaEnlazada(posicion)`))

donde `Nodo` es `tupla`(*arbolTrie*: `arreglo(puntero(Nodo))`,
info: `posicion`,
posEnLista: `iterador(listaEnlazada)`)

2.1.3. Invariante de Representación

(I) Raiz es la raiz del arbol con invariante de trie y es un arreglo de 27 posiciones (que representan las letras). Cada posicion contiene un puntero a Nodo.

(II) listaIterable es una lista de posicion que provee un iterador para poder recorrer todo el arbol en O(n).

(III) Cada una de las posiciones de la lista tiene que corresonder con una posicion de un elemento del trie.

(IV) Nodo es una tupla que contiene un arreglo de 27 posiciones con un puntero a otro Nodo en cada posicion ,un elemento info que es la posicion que contiene esa clave del arbol, y un elemento iterador que es un puntero a un nodo de la lista enlazada.

(V) El iterador a la lista enlazada de cada nodo tiene que apuntar al elemento de la lista que contiene la misma posicion que el nodo.

(VI) No hay posiciones repetidas en la listaEnlazada ya que no puede haber mas de un elemento en cada posicion.

(VII) El arbolTrie tampoco tiene posiciones repetidas.

Rep : estr $\longrightarrow$ bool

Rep(e) $\equiv$ true $\Longleftrightarrow$

($\forall c$: campusSeguro)()

Rep : estr $\longrightarrow$ bool

Rep(e) $\equiv$ true $\Longleftrightarrow$
($\forall c$: compu)($c \in$ computadoras(e.topologia) $\Leftrightarrow$
(
($\exists cd$: compuDCNet) (está?(cd, e.vectorCompusDCNet) $\wedge$ (cd.pc = puntero(c)) $\wedge$
($\exists s$: string)(def?(s, e.diccCompusDCNet) $\wedge$ ($s = c$.ip)))
)
) $\wedge_{\text{L}}$
($\forall cd$: compuDCNet)(está?(cd, e.vectorCompusDCNet) $\Leftrightarrow$
($\exists s$: string)(($s = cd$.pc$\rightarrow$ip) $\wedge$ def?(s, e.diccCompusDCNet) $\wedge_{\text{L}}$
obtener(s, e.diccCompusDCNet) = puntero(cd))
) $\wedge_{\text{L}}$
($\exists cd$: compuDCNet)(está?(cd, e.vectorCompusDCNet) $\wedge_{\text{L}}$
*(cd.pc) = compuQueMásEnvió(e.vectorCompusDCNet) $\wedge$ e.laQueMásEnvió = puntero(cd)) $\wedge_{\text{L}}$
($\forall cd_1$: compuDCNet)(está?(cd_1, e.vectorCompusDCNet) $\Rightarrow$
($\forall p_1$: paquete)($p_1 \in cd_1$.conjPaquetes $\Rightarrow$
($\forall cd_2$: compuDCNet)((está?(cd_2, e.vectorCompusDCNet) $\wedge$ $cd_1 \neq cd_2$) $\Rightarrow$
($\forall p_2$: paquete)($p_2 \in cd_2$.conjPaquetes $\Rightarrow$ p_1.id $\neq$ p_2.id)
)
)
) $\wedge_{\text{L}}$
($\forall cd$: compuDCNet)(está?(cd, e.vectorCompusDCNet) $\Rightarrow$
(
($\forall p$: paquete)($p \in cd$.conjPaquetes $\Leftrightarrow$
(
((p.origen $\in$ computadoras(e.topologia) $\wedge$ p.destino $\in$ computadoras(e.topologia) $\wedge$
p.destino $\neq$ *(cd.pc)) $\wedge_{\text{L}}$
($\exists sc$: secu(compu))($sc \in$ caminosMinimos(e.topologia, p.origen, p.destino) $\wedge$ está(*(cd.pc), sc))) $\wedge$
($\exists n$: nat) ((def?(n, cd.diccPaquetesDCNet) $\wedge$ p.id = n) $\wedge_{\text{L}}$
($\exists pdn$: paqueteDCNet)($pdn \in e$.conjPaquetesDCNet $\wedge$ Siguiente(pdn.it) = p $\wedge$
((p.origen = *(cd.pc) $\wedge$ pdn.recorrido = *(cd.pc) $\bullet$ <>) $\vee$
(p.origen $\neq$ *(cd.pc) $\wedge$ pdn.recorrido $\in$ caminosMinimos(e.topologia, p.origen, *(cd.pc)))) $\wedge$
Siguiente(obtener(n, cd.diccPaquetesDCNet)) = pdn
)
)
)
) $\wedge_{\text{L}}$
(¬vacía?(cd.colaPaquetesDCNet) $\Leftrightarrow$
($\exists p$: paquete)(($p \in cd$.conjPaquetes) $\wedge$ (p = paqueteMásPrioridad(cd.conjPaquetes)) $\wedge$
($\exists pdn$: paqueteDCNet)(($pdn \in e$.conjPaquetesDCNet) $\wedge$ (Siguiente(pdn.it) = p) $\wedge$
(Siguiente(proximo(cd.colaPaquetesDCNet)) = pdn))
)
) $\wedge_{\text{L}}$
(cd.enviados $\geq$ enviadosCompu(*(cd.pc), e.vectorCompusDCNet)) $\wedge$
(¬HaySiguiente?(cd.paqueteAEnviar)))
)

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

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

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

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

paquetesConPrioridadK : conj(paquete) × nat ⟶ conj(paquete)

altaPrioridad : conj(paquetes) *cp* ⟶ nat {¬∅?(*cp*)}

enviadosCompu : compu × secu(compuDCNet) ⟶ nat

aparicionesCompu : compu × conj(nat) *cn* × dicc(nat × itConj(paqueteDCNet)) *dp* ⟶ 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, Siguiente(obtener(dameUno(cn), dpd)).recorrido) then
                                       1 + aparicionesCompu(c, sinUno(cn), dpd)
                                   else
                                       aparicionesCompu(c, sinUno(cn), dpd)
                                   fi
                               fi
```

2.1.4. Función de Abstracción

Abs : estr e $\longrightarrow$ dcnet {Rep(e)}

Abs(e) $=_{obs}$ dcn: dcnet | red(dcn) = e.topología $\wedge$
($\forall cdn$: compuDCNet)(está?(cdn, e.vectorCompusDCNet) $\Rightarrow_L$
enEspera(dcn, *(cdn.pc)) = cdn.conjPaquetes $\wedge$
cantidadEnviados(dcn, *(cdn.pc)) = cdn.enviados $\wedge$
($\forall p$: paquete)($p \in cdn$.conjPaquetes $\Rightarrow_L$
caminoRecorrido(dcn, p) = Siguiente(obtener(p.id, cdn.diccPaquetesDCNet)).recorrido
)
)

2.2. Algoritmos

iIniciarDCNet (**in** *topo*: `red`) → res: estr

```
res.topologia ← Copiar(topo)                                          O(n! * n^6)
res.vectorCompusDCNet ← Vacia()                                       O(1)
res.diccCompusDCNet ← CrearDicc()                                     O(1)
res.laQueMasEnvio ← NULL                                              O(1)
res.conjPaquetesDCNet ← Vacio()                                       O(1)

itConj(compu): it ← CrearIt(Computadoras(topo))                       O(1)

if(HaySiguiente?(it)) then                                            O(1)
    res.laQueMasEnvio ← puntero(Siguiente(it))                        O(1)
end if

while HaySiguiente?(it) do                                            O(1)
    compuDCNet: compudcnet ← <puntero(Siguiente(it)), Vacio(), CrearDicc(),
        Vacia(), CrearIt(Vacio()), 0>                                 O(1)
    AgregarAtras(res.vectorCompusDCNet, compudcnet)                   O(n)
    Definir(res.diccCompusDCNet, Siguiente(it).ip, puntero(compudcnet))  O(L)
    Avanzar(it)                                                       O(1)
end while                                                             O(n * (n + L))
```

Complejidad : $O(n*(n+L))$

iCrearPaquete (**in/out** *dcn*: `dcnet`, **in** *p*: `paquete`)

```
puntero(compuDCNet): compudcnet ←
    Significado(dcn.diccCompusDCNet, p.origen.ip)                     O(L)
itConj(paquete): itPaq ← AgregarRapido(compudcnet→conjPaquetes, p)     O(1)
lista(compu): recorr ← AgregarAtras(Vacia(), p.origen)                O(1)
paqueteDCNet: paqDCNet ← <itPaq, recorr>                              O(1)

itConj(paqueteDCNet): itPaqDCNet ←
    AgregarRapido(dcn.conjPaquetesDCNet, paqDCNet)                    O(1)
Definir(compudcnet→diccPaquetesDCNet, p.id, itPaqDCNet)               O(log(k))
Encolar(compudcnet→colaPaquetesDCNet, p.prioridad, itPaqDCNet)        O(log(k))
```

Complejidad : $O(log(k)+L)$

iAvanzarSegundo (**in**/**out** *dcn*: dcnet)

```
nat: maxEnviados ← 0
nat: i ← 0                                                                  O(1)
while i < Longitud(dcn.vectorCompusDCNet) do                                O(1)
    if(¬EsVacia?(dcn.vectorCompusDCNet[i].colaPaquetesDCNet)) then
        dcn.vectorCompusDCNet[i].paqueteAEnviar ←
            Desencolar(dcn.vectorCompusDCNet[i].colaPaquetesDCNet)          O(log(k))
    end if
    i++                                                                     O(1)
end while                                                                   O(n * log(k))

i ← 0                                                                       O(1)
while i < Longitud(dcn.vectorCompusDCNet) do                                O(1)
    if(HaySiguiente?(dcn.vectorCompusDCNet[i].paqueteAEnviar)) then         O(1)

        dcn.vectorCompusDCNet[i].enviados++                                 O(1)
        if(dcn.vectorCompusDCNet[i].enviados > maxEnviados) then            O(1)
            dcn.laQueMasEnvio ← puntero(dcn.vectorCompusDCNet[i])           O(1)
        end if

        paquete: pAEnviar ←
            Siguiente(Siguiente(dcn.vectorCompusDCNet[i].paqueteAEnviar).it)  O(1)
        itConj(lista(compu)): itercaminos ←
            CrearIt(CaminosMinimos(dcn.topologia,
            *(dcn.vectorCompusDCNet[i].pc), pAEnviar.destino))              O(1)
        compu: siguientecompu ← Siguiente(itercaminos)[1]                   O(1)

        if(pAEnviar.destino ≠ siguientecompu) then                          O(1)

            compuDCNet: siguientecompudcnet ←
                *(Obtener(dcn.diccCompusDCNet, siguientecompu.ip))          O(L)

            itConj(paquete): itpaquete ←
                AgregarRapido(siguientecompudcnet.conjPaquetes, pAEnviar)   O(1)

            itConj(paqueteDCNet): paqAEnviar ←
                Obtener(dcn.vectorCompusDCNet[i].diccPaquetesDCNet,
                pAEnviar.id)                                                O(log(k))

            AgregarAtras(Siguiente(paqAEnviar).recorrido, siguientecompu)   O(1)

            Encolar(siguientecompudcnet.colaPaquetesDCNet,
                pAEnviar.prioridad, paqAEnviar)                             O(log(k))
            Definir(siguientecompudcnet.diccPaquetesDCNet,
                pAEnviar.id, paqAEnviar)                                    O(log(k))
        end if

        Borrar(dcn.vectorCompusDCNet[i].diccPaquetesDCNet,
            Siguiente(dcn.vectorCompusDCNet[i].paqueteAEnviar→it).id)       O(log(k))
        EliminarSiguiente(Siguiente(dcn.vectorCompusDCNet[i].paqueteAEnviar).it)
                                                                            O(1)
        EliminarSiguiente(dcn.vectorCompusDCNet[i].paqueteAEnviar)          O(1)

        dcn.vectorCompusDCNet[i].paqueteAEnviar ← CrearIt(Vacio())          O(1)

    end if
    i++                                                                     O(1)
```

```
    end while                                                    O(n * (L + log(k)))
```

Complejidad : $O(n * (L + log(k)))$

Red (**in** *dcn*: `dcnet`) → res: red

```
    res ← dcn.topologia                                          O(1)
```

Complejidad : $O(1)$

CaminoRecorrido (**in** *dcn*: `dcnet`, **in** *p*: `paquete`) → res: lista(compu)

```
    nat: i ← 0                                                   O(1)
    while i < Longitud(dcn.vectorCompusDCNet) do                 O(1)
        if Definido?(dcn.vectorCompusDCNet[i].diccPaquetesDCNet, p.id) then   O(log(k))
            res ← Siguiente(Obtener(dcn.vectorCompusDCNet[i].diccPaquetesDCNet,
                p.id)).recorrido                                 O(log(k))
        end if
        i++                                                      O(1)
    end while                                                    O(n * log(k))
```

Complejidad : $O(n * log(k))$

CantidadEnviados (**in** *dcn*: `dcnet`, **in** *c*: `compu`) → res: nat

```
    res ← Obtener(dcn.diccCompusDCNet, c.ip)→enviados            O(L)
```

Complejidad : $O(L)$

EnEspera (**in** *dcn*: `dcnet`, **in** *c*: `compu`) → res: nat

```
    res ← Obtener(dcn.diccCompusDCNet, c.ip)→conjPaquetes        O(L)
```

Complejidad : $O(L)$

PaqueteEnTransito (**in** *dcn*: `dcnet`, **in** *p*: `paquete`) → res: bool

```
    res ← false
    nat: i ← 0                                                   O(1)
    while i < Longitud(dcn.vectorCompusDCNet) do                 O(1)
        if Definido?(dcn.vectorCompusDCNet[i].diccPaquetesDCNet, p.id) then   O(log(k))
            res ← true                                           O(1)
        end if
        i++                                                      O(1)
    end while                                                    O(n * log(k))
```

Complejidad : $O(n * log(k))$

LaQueMasEnvio (**in** *dcn* : `dcnet`) → res: compu

```
res ← *(dcn.laQueMasEnvio→pc)                                   O(1)
```

Complejidad : $O(1)$

• $=_i$ • (**in** dcn_1 : `dcnet`, **in** dcn_2 : `dcnet`) → res: bool

```
bool: boolTopo ← dcn1.topologia = dcn2.topologia                          O(n + L^2)
bool: boolVec ← dcn1.vectorCompusDCNet = dcn2.vectorCompusDCNet            O(n * k * (k + n))
bool: boolConj ← dcn1.conjPaquetesDCNet = dcn2.conjPaquetesDCNet           O(k^3 * (k + n))
bool: boolMasEnvio ← *(dcn1.laQueMasEnvio) = *(dcn2.laQueMasEnvio)        O(1)

res ← boolTopo ∧ boolVec ∧ boolTrie ∧ boolConj ∧ boolMasEnvio              O(1)
```

Complejidad : $O(n * k^3 * (k + n))$

• $=_{compudcn}$ • (**in** c_1 : `compuDCNet`, **in** c_2 : `compuDCNet`) → res: bool

```
bool: boolPC ← *(c1.pc) = *(c2.pc)                                        O(1)
bool: boolConj ← c1.conjPaquetes = c1.conjPaquetes                        O(k^2)
bool: boolAVL ← true                                                      O(1)
bool: boolCola ← true                                                     O(1)
bool: boolPaq ← Siguiente(c1.paqueteAEnviar) =paqdcn Siguiente(c2.paqueteAEnviar)
                                                                          O(n)
bool: boolEnviados ← c1.enviados = c2.enviados                            O(1)

if boolConj then                                                          O(1)
    itConj: itconj1 ← CrearIt(c1.conjPaquetes)                            O(1)
    while HaySiguiente?(itconj1) do                                       O(1)
        if Definido?(c2.diccPaquetesDCNet, Siguiente(itconj1).id then     O(log(n))
            if ¬(Siguiente(Obtener(c1.diccPaquetesDCNet, Siguiente(itconj1).id))
                =paqdcn
                Siguiente(Obtener(c1.diccPaquetesDCNet, Siguiente(itconj1).id)))
                then                                                      O(n)
                boolAVL ← false                                           O(1)
            end if
        else
            boolAVL ← false                                               O(1)
        end if
        Avanzar(itconj1)                                                  O(1)
    end while                                                             O(n * k)
end if

if EsVacia(c1.colaPrioridad) then                                         O(1)
    if ¬EsVacia(c2.colaPrioridad) then                                    O(1)
        boolCola ← false                                                  O(1)
    end if
else
    if EsVacia(c1.colaPrioridad) then                                     O(1)
        boolCola ← false                                                  O(1)
    else
        if ¬(Siguiente(Proximo(c1.colaPrioridad)) =paqdcn
            Siguiente(Proximo(c2.colaPrioridad))) then                    O(n)
```

```
                boolCola ← false                                                    O(1)
            end if
        end if
    end if

    res ← boolPC ∧ boolConj ∧ boolAVL ∧ boolCola ∧ boolPaq ∧ boolEnviados           O(1)
```

Complejidad : $O(k^2 + n * k) = O(k * (k + n))$

• $=_{paqdcn}$ • (**in** p_1 : `paqueteDCNet`, **in** p_2 : `paqueteDCNet`,) → res: bool

```
    bool: boolPaq ← Siguiente(p1.it) = Siguiente(p2.it)                             O(1)
    bool: boolRecorrido ← p1.recorrido = p2.recorrido                               O(n)

    res ← boolPaq ∧ boolRecorrido                                                   O(1)
```

Complejidad : $O(n)$