

Algoritmos y Estructuras de Datos II

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo Práctico I

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		

1. TADs Auxiliares

TAD pc **ES** nat

TAD paquete **ES** tupla(nat id, nat ipOrigen, nat ipDestino, nat prioridad)

TAD segmento **ES** tupla(nat ipOrigen, nat interfazOrigen, nat ipDestino, nat interfazDestino)

2. TAD DCNET

TAD DCNET

géneros dcnnet

igualdad observacional

$$(\forall d, d' : \text{dcnnet}) \left(d =_{\text{obs}} d' \iff \left(\begin{array}{l} (\text{topo}(d) =_{\text{obs}} \text{topo}(d')) \wedge_{\text{L}} (\\ (\forall p : \text{pc}) (p \in \text{compus}(\text{topo}(d))) \Rightarrow_{\text{L}} (\\ (\text{buffer}(d, p) =_{\text{obs}} \text{buffer}(d', p)) \wedge \\ (\# \text{enviados}(d, p) =_{\text{obs}} \# \text{enviados}(d', p)) \end{array} \right) \right)$$

generadores

CrearRed	: topo	→ dcnnet
Seg	: dcnnet	→ dcnnet
CrearPaquete	: dcnnet dcn × paquete p	→ dcnnet
	{(π ₂ (p) ∈ compus(topo(dcn)) ∧ π ₃ (p) ∈ compus(topo(dcn)) ∧ _L conectadas?(topo(dcn), π ₂ (p), π ₃ (p))}	

observadores básicos

topo	: dcnnet	→ topologia
#enviados	: dcnnet dcn × pc ip	→ nat {ip ∈ compus(topo(dcn))}
buffer	: dcnnet dcn × pc ip	→ conj(paquete) {ip ∈ compus(topo(dcn))}

otras operaciones

recorridoPaquete	: dcnnet dcn × nat id	→ secu(segmento) {paqueteEnTransito?(dcn, id)}
cortarRecHasta	: secu(segmento) × pc	→ secu(segmento)
buscarPaquete	: dcnnet dcn × conj(pc) pcs × nat id	→ pc {pcs ⊆ compus(topo(dcn)) ∧ paqueteEnTransito?(dcn, id)}
ids	: conj(paquete)	→ conj(nat)
paqueteEnTransito?	: dcnnet × nat	→ bool
darPaqueteEnviado	: conj(paquete) cp	→ paquete {¬∅?(cp)}
rutaPaqueteEnviado	: dcnnet dcn × pc compu	→ secu(segmento) {compu ∈ compus(topo(dcn))}
paquetesRecibidos	: dcnnet × conj(pc) vecinasPc × pc	→ conj(paquete) compu {compu ∈ compus(topo(dcn)) ∧ _L vecinasPc ⊆ vecinas(topo(dcn), compu)}
darPrioridad	: dcnnet dcn × nat id	→ nat {paqueteEnTransito?(dcn, id)}
buscarPrioridad	: nat × conj(paquetes) cp	→ nat {¬∅?(cp)}
maxPrioridad	: dcnnet × conj(pc)c	→ nat {¬∅?(cc)}
PaquetesConPrioridadK	: dcnnet × conj(pc) × nat	→ conj(paquete) {k > 0 ∧ ¬∅?(cc)}

paquetesEnLaRed	: dcnnetdcn	\longrightarrow conj(paquete) $\{\neg\emptyset?(\text{compus}(\text{topo}(\text{dcn})))\}$
buscarPaquetesEnLaRed	: dcnnet \times conj(pc))	\longrightarrow conj(paquete)
compuQueMasEnvio	: dcnnetdcn	\longrightarrow pc $\{\neg\emptyset?(\text{compus}(\text{topo}(\text{dcn})))\}$
maxEnviado	: dcnnet \times conj(pc)cc	\longrightarrow nat $\{\neg\emptyset?(cc)\}$
enviaronK	: dcnnet \times conj(pc) \times nat	\longrightarrow conj(pc)

axiomas

topo(crearRed(t))	\equiv t
topo(seg(dcn))	\equiv topo(dcn)
topo(CrearPaquete(dcn, p))	\equiv topo(dcn)
#enviados(crearRed(t), ip)	\equiv 0
#enviados(seg(dcn), ip)	\equiv if $\emptyset?(\text{buffer}(\text{dcn}, \text{ip}))$ then #enviados(dcn, ip) + 1 else #enviados(dcn, ip) fi
#enviados(CrearPaquete(dcn, p), ip)	\equiv #enviados(dcn, ip)
buffer(CrearRed(t), c)	\equiv \emptyset
buffer(CrearPaquete(dcn, p), c)	\equiv if $\pi_2(p) = c$ then $\text{Ag}(p, \emptyset) \cup \text{buffer}(\text{dcn}, c)$ else $\text{buffer}(\text{dcn}, c)$ fi
buffer(segundo(dcn), c)	\equiv ($\text{buffer}(\text{dcn}, c) - \text{darPaqueteEnviado}(\text{buffer}(\text{dcn}, c))$) \cup $\text{paquetesRecibidos}(\text{dcn}, \text{vecinas}(c), c)$
recorridoPaquete(dcn, p)	\equiv cortarRecHasta(darCaminoMasCorto(topo(dcn), origen(p), destino(p)), buscarPaquete(compus(topo(dcn)), p))
cortarRecHasta(s, ip)	\equiv if vacia?(s) \vee_L ip = ipOrigen(prim(s)) then $\langle \rangle$ else $\text{prim}(s) \bullet \text{cortarRecHasta}(\text{fin}(s), \text{ip})$ fi
buscarPaquete(dcn, pcs, id)	\equiv if id \in ids(buffer(dcn, dameUno(pcs))) then dameUno(pcs) else buscarPaquete(dcn, sinUno(pcs), id) fi
ids(paquetes)	\equiv if $\emptyset?(paquetes)$ then \emptyset else $\text{Ag}(\pi_1(\text{dameUno}(paquetes)), \text{ids}(\text{sinUno}(paquetes)))$ fi
paqueteEnTransito?(dcn, id)	\equiv id \in ids(paquetesEnLaRed(dcn))
buscarPaquetesEnLaRed(dcn, cc)	\equiv if $\emptyset?(cc)$ then \emptyset else $\text{buffer}(\text{dcn}, \text{dameUno}(cc)) \cup \text{buscarPaquetesEnLaRed}(\text{dcn}, \text{sinUno}(cc))$ fi
paquetesEnLaRed(d)	\equiv buscarPaquetesEnLaRed(d, compus(topo(d)))

```

buscarPrioridad(idPaq, cs)      ≡ if idPaq =  $\pi_1$ (dameUno(cs)) then
                                 $\pi_4$ (dameUno(cs))
                                else
                                    buscarPrioridad(idPaq, sinUno(cs))
                                fi

darPrioridad(d, idPaq)          ≡ buscarPrioridad(idPaq, compus(topo(dcn)))

darPaqueteEnviado(dcn,cp)       ≡ dameUno(PaquetesConPrioridadK (dcn, cp, maxPrioridad(dcn, cp)))

rutaPaqueteEnviado(dcn, c)     ≡ darCaminoMasCorto(topo(dcn),
                                 $\pi_2$ (darPaqueteEnviado(dcn, buffer(dcn, c))),
                                 $\pi_3$ (darPaqueteEnviado(dcn, buffer(dcn, c))))

paquetesRecibidos(dcn, vecinasPc, c) ≡ if darSiguientePc(
                                rutaPaqueteEnviado(dcn, dameUno(vecinasPc)),
                                dameUno(vecinasPc)) = c then
                                    Ag(darPaqueteEnviado(dcn,
                                        buffer(dcn, dameUno(vecinasPc))),  $\emptyset$ )  $\cup$ 
                                        paquetesRecibidos(dcn, sinUno(vecinasPc), c)
                                    else
                                        paquetesRecibidos(dcn, sinUno(vecinasPc), c)
                                    fi
                                fi

maxPrioridad(dcn,cp)           ≡ if  $\emptyset?$ (sinUno(cp)) then
                                darPrioridad(dcn, dameUno(cp))
                                else
                                     $\max$ (darPrioridad(dcn, dameUno(cp),
                                        maxPrioridad(dcn, sinUno(cp)))
                                    fi

PaquetesConPrioridadK(dcn,cp,k) ≡ if  $\emptyset?$ (cp) then
                                 $\emptyset$ 
                                else
                                    if darPrioridad(dcn, dameUno(cp)) = k then
                                        Ag(dameUno(cp), PaquetesConPrioridadK
                                            (dcn, sinUno(cp), k))
                                        else
                                            PaquetesConPrioridadK(dcn, sinUno(cp), k)
                                        fi
                                    fi

compuQueMasEnvio(dcn)          ≡ dameUno(enviaronK(dcn, compus(topo(dcn))),
                                maxEnviado(dcn, compus(topo(dcn)))))

maxEnviado(dcn,cc)             ≡ if  $\emptyset?$ (sinUno(cc)) then
                                #enviados(dcn, dameUno(cc))
                                else
                                     $\max$ (#enviados(dcn, dameUno(cc),
                                        maxEnviado(dcn, sinUno(cc)))
                                    fi

enviaronK(dcn,cc,k)           ≡ if  $\emptyset?$ (cc) then
                                 $\emptyset$ 
                                else
                                    if #enviados(dcn, dameUno(cc)) = k then
                                        Ag(dameUno(cc), enviaronK (dcn, sinUno(cc), k))
                                        else
                                            enviaronK(dcn, sinUno(cc), k)
                                        fi
                                    fi

```

Fin TAD

3. TAD TOPOLOGÍA

Este TAD modela cómo se conectan las computadoras. Las IP son únicas entre compus de la topología. Las compus tienen interfaces numeradas con los naturales de manera consecutiva (todas funcionan perfecto y todo eso, el DC las cuida y mantiene como corresponde).

TAD TOPOLOGÍA

géneros topologia

igualdad observacional

$$(\forall t, t' : \text{topo}) \left(t =_{\text{obs}} t' \iff \left(\begin{array}{l} (\text{compus}(t) =_{\text{obs}} \text{compus}(t')) \wedge_L \\ ((\forall p : \text{pc}) (p \in \text{compus}(t) \Rightarrow_L (\\ \quad (\text{cablesEn}(t, p) =_{\text{obs}} \text{cablesEn}(t', p)) \wedge \\ \quad (\# \text{interfaces}(t, p) =_{\text{obs}} \# \text{interfaces}(t', p)) \\ \end{array} \right) \right) \right)$$

generadores

NuevaTopo	:		\longrightarrow	topologia
Compu	:	topologia \times pc $ip \times$ nat	\longrightarrow	topologia $\{ \neg(ip \in \text{compus}(t)) \}$
Cable	:	topologia \times pc $ipA \times$ nat $ifA \times$ pc $ipB \times$ nat ifB	\longrightarrow	topologia $\left\{ \begin{array}{l} (ipA \in \text{compus}(t) \wedge ipB \in \text{compus}(t)) \wedge_L \\ (ifA < \# \text{interfaces}(t, ipA)) \wedge \\ (ifB < \# \text{interfaces}(t, ipB)) \wedge \\ \neg(ifA \in \text{interfacesOcupadasDe}(t, ipA)) \wedge \\ \neg(ifB \in \text{interfacesOcupadasDe}(t, ipB)) \wedge \\ \neg(ipA \in \text{vecinas}(t, ipB)) \end{array} \right\}$

observadores básicos

compus	:	topologia	\longrightarrow	conj(pc)
cablesEn	:	topologia $t \times$ pc ip	\longrightarrow	conj(tupla(pc, nat)) $\{ ip \in \text{compus}(t) \}$
#interfaces	:	topologia $t \times$ pc ip	\longrightarrow	nat $\{ ip \in \text{compus}(t) \}$

otras operaciones

vecinas	:	topologia $t \times$ pc ip	\longrightarrow	conj(pc) $\{ ip \in \text{compus}(t) \}$
interfacesOcupadasDe	:	topologia $t \times$ pc ip	\longrightarrow	conj(nat) $\{ ip \in \text{compus}(t) \}$
conectados?	:	topologia $t \times$ pc $ipA \times$ pc ipB	\longrightarrow	bool $\{ ipA \in \text{compus}(t) \wedge ipB \in \text{compus}(t) \}$
darInterfazConectada	:	conj(tupla(pc, nat)) $cablesA \times$ pc ipB	\longrightarrow	nat $\{ ipB \in \pi_2 \text{Conj}(cablesA) \}$
darSegmento	:	topologia $t \times$ pc $ipA \times$ pc ipB	\longrightarrow	segmento $\{ ipA \in \text{compus}(t) \wedge_L ipB \in \text{vecinas}(t, ipA) \}$
estáEnRuta?	:	secu(segmento) $ruta \times$ pc ip	\longrightarrow	bool
darSiguientePc	:	secu(segmento) $ruta \times$ pc ip	\longrightarrow	pc $\{ estáEnRuta?(ruta, ip) \}$
darCaminoMasCorto	:	topologia $t \times$ pc $ipA \times$ pc ipB	\longrightarrow	secu(segmento) $\{ ipA \in \text{compus}(t) \wedge ipB \in \text{compus}(t) \wedge_L \text{conectados?}(t, ipA, ipB) \}$
darRutas	:	topologia \times pc $ipA \times$ pc $ipB \times$ conj(nat) \times secu(segmento)	\longrightarrow	conj(secu(segmento)) $\{ ipA \in \text{compus}(t) \wedge ipB \in \text{compus}(t) \}$

darRutasVecinas	: topologia \times conj(pc) \times pc $ip \times$ conj(pc) \times se- cu(segmento)	\longrightarrow conj(secu(segmento)) $\{ip \in compus(t)\}$
longMenorSec	: conj(secu(α)) secus	\longrightarrow nat $\{-\emptyset?(secus)\}$
secusDeLongK	: conj(secu(α)) \times nat	\longrightarrow conj(secu(α))
π_1 Conj	: conj(tupla(pc, nat))	\longrightarrow conj(pc)
π_2 Conj	: conj(tupla(pc, nat))	\longrightarrow conj(nat)

axiomas

compus(NuevaTopo)	$\equiv \emptyset$
compus(Compu($t, ipNueva, cantIfaces$))	$\equiv Ag(ipNueva, compus(t))$
compus(Cable(t, ipA, ifA, ipB, ifB))	$\equiv compus(t)$
cablesEn(NuevaTopo, ip)	$\equiv \emptyset$
cablesEn(Compu($t, ipNueva, cantIfaces$), ip)	$\equiv cablesEn(t, ip)$
cablesEn(Cable(t, ipA, ifA, ipB, ifB), ip)	\equiv if $ip = ipA$ then $Ag(\langle ifA, ipB \rangle, \emptyset)$ else \emptyset fi \cup if $ip = ipB$ then $Ag(\langle ifB, ipA \rangle, \emptyset)$ else \emptyset fi \cup $cablesEn(t, ip)$
#interfaces(NuevaTopo, ip)	$\equiv 0$
#interfaces(Compu($t, ipNueva, cantIfaces$), ip)	\equiv if $ip = ipNueva$ then $cantIfaces$ else $\#interfaces(t, ip)$ fi
#interfaces(Cable(t, ipA, ifA, ipB, ifB), ip)	$\equiv \#interfaces(t, ip)$
interfacesOcupadasDe(t, ip)	$\equiv \pi_1 Conj(cablesEn(t, ip))$
vecinas(t, ip)	$\equiv \pi_2 Conj(cablesEn(t, ip))$
conectados?(t, ipA, ipB)	$\equiv \neg \emptyset?(darRutas(t, ipA, ipB, \emptyset, <>))$
darInterfazConectada($conjCablesIpA, ipB$)	\equiv if $ipB = \pi_2(dameUno(conjCablesIpA))$ then $\pi_1(dameUno(conjCablesIpA))$ else $darInterfazConectada(sinUno(conjCablesIpA), ipB)$ fi
darSegmento(t, ipA, ipB)	$\equiv \langle ipA, darInterfazConectada(cablesEn(t, ipA), ipB),$ $ipB, darInterfazConectada(cablesEn(t, ipB), ipA) \rangle$
estáEnRuta?($ruta, ip$)	\equiv if vacía?($ruta$) then $false$ else if $\pi_1(prim(ruta))=ip$ then $true$ else $estáEnRuta?(fin(rutas), ip)$ fi
darSiguientePc($ruta, ip$)	\equiv if $\pi_1(prim(ruta))=ip$ then $\pi_3(prim(ruta))$ else $darSiguientePc(fin(rutas), ip)$ fi
darCaminoMasCorto(t, ipA, ipB)	$\equiv dameUno(secusDeLongK(darRutas(t, ipA, ipB, \emptyset, <>),$ $longMenorSec(darRutas(t, ipA, ipB, \emptyset, <>)))$

```

darRutas(t, ipA, ipB, rec, ruta)  ≡  if ipB ∈ vecinas(t, ipA) then
    Ag(ruta ∘ darSegmento(t, ipA, ipB) , ∅)
else
    if ∅?(vecinas(t, ipA) - rec) then
        ∅
    else
        darRutas(t, dameUno(vecinas(t, ipA) - rec),
            ipB, Ag(ipA, rec),
            ruta ∘ darSegmento(t, ipA, dameUno(vecinas(t, ipA) - rec))) ∪
            darRutasVecinas(t, sinUno(vecinas(t, ipA) - rec),
            ipB, Ag(ipA, rec),
            ruta ∘ darSegmento(t, ipA, dameUno(vecinas(t, ipA) - rec)))
    fi
fi

darRutasVecinas(t, vecinas, ipB, rec, ruta)  ≡  if ∅?(vecinas) then
    ∅
else
    darRutas(t, dameUno(vecinas), ipB, rec, ruta) ∪
    darRutasVecinas(t, sinUno(vecinas), ipB, rec, ruta)
fi

darCaminoMasCorto(t, ipA, ipB)  ≡  dameUno(secusDeLongK(darRutas(t, ipA, ipB, ∅, <>),
    longMenorSec(darRutas(t, ipA, ipB, ∅, <>)))

secusDeLongK(secus, k)  ≡  if ∅?(secus) then
    ∅
else
    if long(dameUno(secus)) = k then
        dameUno(secus) ∪ secusDeLongK(sinUno(secus), k)
    else
        secusDeLongK(sinUno(secus), k)
    fi
fi

longMenorSec(secus)  ≡  if ∅?(sinUno(secus)) then
    long(dameUno(secus))
else
    min(long(dameUno(secus)),
        longMenorSec(sinUno(secus)))
fi

π1Conj(conjDuplas)  ≡  if ∅?(conjDuplas) then
    ∅
else
    Ag(π1(dameUno(conjDuplas)),
        π1Conj(sinUno(conjDuplas)))
fi

π2Conj(conjDuplas)  ≡  if ∅?(conjDuplas) then
    ∅
else
    Ag(π2(dameUno(conjDuplas)),
        π2Conj(sinUno(conjDuplas)))
fi

```

Fin TAD