

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. TAD DCNET

TAD DCNET

géneros `dcnet`

igualdad observacional

$$(\forall d, d' : \text{dcnet}) \quad d =_{\text{obs}} d' \iff \left(\begin{array}{l} (topo(d) =_{\text{obs}} topo(d')) \wedge ((\forall p : pc)(p \in pcs(topo(d)) \wedge \\ p \in pcs(topo(d')) \Rightarrow_L (dcNetBuffer(d, p) =_{\text{obs}} \\ dcNetBuffer(d', p) \wedge paquetesMandados(d, p) =_{\text{obs}} \\ paquetesMandados(d', p)) \wedge ((\forall p : paquetes)((\exists c : \\ pc)(c \in pcs(topo(d')) \wedge c \in pcs(topo(d')) \wedge_L (p \in \\ dcNetBuffer(d, c) \wedge p \in dcNetBuffer(d', c))) \Rightarrow_L \\ (recorridoPaquete(d, p) =_{\text{obs}} recorridoPaquete(d', p))) \end{array} \right)$$

generadores

`crearRed` : `topo` \longrightarrow `dcnet`

`seg` : `dcnet` \longrightarrow `dcnet`

`paquetePendiente` : `dcnet dcn` \times `pc p1` \times `pc p2` \times `paquete` \longrightarrow `dcnet`
 $\{(p_1 \in pcs(topo(dcn)) \wedge p_2 \in pcs(topo(dcn))) \wedge_L conectadas?(topo(dcn), p_1, p_2)\}$

observadores básicos

`recorridoPaquete` : `dcnet dcn` \times `paquete p` \longrightarrow `secu((ip, interface))`
 $\{(\exists c : pc)(c \in pcs(topo(dcn)) \wedge_L (p \in dcNetBuffer(dcn, c)))\}$

`dcNetBuffer` : `dcnet dcn` \times `pc p` \longrightarrow `conj(paquete)`
 $\{p \in pcs(topo(dcn))\}$

`paquetesMandados` : `dcnet dcn` \times `pc p` \longrightarrow `nat` $\{p \in pcs(topo(dcn))\}$

`topo` : `dcnet` \longrightarrow `topologia`

otras operaciones

`paqueteEnTransito?` : `dcnet` \times `paquete` \longrightarrow `bool`

`perteneceBuffers?` : `paquete` \times `buffers` \longrightarrow `bool`

`maxPaquetesMandados` : `dcnet` \longrightarrow `pc`

`auxMaxPaquetes` : `dcnet` \times `conj(pc)` \longrightarrow `pc`

`pasoSeg` : `topo` \times `buffers` \times `buffers` \longrightarrow `buffers`

`regresion` : `topo` \times `buffers` \times `secu(buffers)` \longrightarrow `buffers`

`cronoPaquetes` : `dcnet` \times `diccionario(pc` \times \longrightarrow `secu(buffers)`
`conj(paquete))`

`auxDefinir` : `buffers` \times `pc` \times `conj(paquete)` \times \longrightarrow `buffers`
`conj(paquete)`

`auxBorrar` : `buffers` \times `pc` \times `conj(paquete)` \times \longrightarrow `buffers`
`conj(paquete)`

`transacion` : `topo` \times `buffers` \times `conj(pc)` \longrightarrow `buffers`

`envio` : `topo` \times `buffers` \times `buffer` \longrightarrow `buffers`

`nuevosPaquetes` : `buffers` \times `buffers` \longrightarrow `buffers`

`damePaquete` : `buffer` \longrightarrow `paquete`

`pasarA` : `topologia` \times `pc` \times `pc` \longrightarrow `pc`

axiomas $\forall p, p' : \text{paquete}, \forall c, c' : \text{pc}, \forall dcn : \text{dcnet}, \forall t : \text{topologia}$
 $topo(crearRed(t)) \equiv t$

<code>topo(seg(dcn))</code>	\equiv <code>topo(dcn)</code>
<code>topo(paquetePendiente(dcn,c,c',p))</code>	\equiv <code>topo(dcn)</code>
<code>paquetesMandados(crearRed(t),c)</code>	\equiv 0
<code>paquetesMandados(seg(dcn),c)</code>	\equiv <code>paquetesMandados(dcn)</code>
<code>paquetesMandados(paquetePendiente(dcn,o,d,p),c)</code>	\equiv if $c = o$ then \quad <code>paquetesMandados(dcn, c) + 1 else \quad <code>paquetesMandados(dcn, c)</code> fi</code>
<code>dcNetBuffer(dcn,c)</code>	\equiv <code>obtener(c,regresion(topo(dcn),vacio,cronoPaquetes(dcn,vacio)))</code>
<code>maxPaquetesMandados(dcn)</code>	\equiv <code>auxMaxPaquetes(dcn,pcs(topo(dcn)))</code>
<code>auxMaxPaquetes(dcn,cs)</code>	\equiv if $\emptyset?(sinUno(cs))$ then \quad <code>dameUno(cs)</code> else \quad if <code>paquetesMandados(dcn, dameUno(cs))</code> < <code>paquetesMandados(dcn, auxMaxPaquetes(dcn, sinUno(cs)))</code> then $\quad\quad$ <code>auxMaxPaquetes(dcn, sinUno(cs))</code> \quad else $\quad\quad$ <code>dameUno(cs)</code> \quad fi fi
<code>paqueteEnTransito?(dcn,p)</code>	\equiv <code>perteneceBuffers?(p,regresion(topo(dcn),vacio,cronoPaquetes(dcn,vacio)))</code>
<code>perteneceBuffers?(p,bs)</code>	\equiv if $\emptyset?(claves(bs))$ then \quad <code>false</code> else \quad if $p \in obtener(dameUno(claves(bs)), bs)$ then $\quad\quad$ <code>true</code> \quad else $\quad\quad$ <code>perteneceBuffers?(p, borrar(dameUno(claves(bs)), bs))</code> \quad fi fi
<code>cronoPaquetes(crearRed(t),bs)</code>	\equiv <code><bs></code>
<code>cronoPaquetes(seg(dcn),bs)</code>	\equiv <code>bs • cronoPaquetes(dcn,∅)</code>
<code>cronoPaquetes(paquetePendiente(dcn,o,d,p),bs)</code>	\equiv if $def?(c, bs)$ then \quad <code>cronoPaquetes(dcn, definir(c, n</code> \cup \quad <code>obtener(o, bs), bs))</code> else \quad <code>cronoPaquetes(dcn, definir(c, n))</code> fi
<code>auxBorrar(bs,c,b,p)</code>	\equiv if $\emptyset?(p - \{b\})$ then \quad <code>borrar(c, n)</code> else \quad <code>borrar(c, bs) definir(c, p - \{b\}, bs)</code> fi
<code>regresion(t,bs,cbs)</code>	\equiv if $vacia?(fin(cbs))$ then \quad <code>pasoSeg(bs, t, prim(cbs))</code> else \quad <code>regresion(t, pasoSeg(bs, t, prim(cbs)), fin(cbs))</code> fi
<code>pasoSeg(t,bs,nbs)</code>	\equiv <code>nuevosPaquetes(transacion(t,bs,claves(bs)) ,nbs)</code>

transacion(t,bs,cp)

```
≡ if ∅?(cp) then
    bs
  else
    transacion(t,envio(t,bs,dameUno(cp)),
              sinUno(cp))
  fi
```

pasarA(t,o,d)

```
≡ prim(caminoMin(t,o,d))
```

envio(t,bs,b)

```
≡ if ∅?(damePaquete(b)) then
    bs
  else
    if pasarA(t,Π1(b),dest(Π2(b))) =
       destino(damePaquete(b)) then
      envio(t,quitarPaquete(bs,Π1(b)),(Π1(b),Π2(b)−
       damePaquete(b)))
    else
      envio(t,quitarPaquete(pasarPaquete(bs,Π1(b),damePaquete(b)),
        (Π1(b),Π2(b) − damePaquete(b)))
    fi
  fi
```

nuevosPaquetes(bs,nbs)

```
≡ if ∅?(claves(nbs)) then
    bs
  else
    auxDefinir(bs,dameUno(claves(nbs)),obtener
      (dameUno(claves(nbs)),nbs),obtener(dameUno
      (claves(nbs),bs)))
    nuevosPaquetes(bs,sinUno(nbs))
  fi
```

TAD buffers es diccionario(pc,conj(paquete))

TAD buffer es tupla(pc,conj(paquete))

Fin TAD

2. 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

generadores

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

observadores básicos

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

otras operaciones

vecinas	:	topologia $t \times$ nat ip	\longrightarrow conj(nat) $\{ ip \in compus(t) \}$
interfacesOcupadasDe	:	topologia $t \times$ nat ip	\longrightarrow conj(nat) $\{ ip \in compus(t) \}$
conectados?	:	topologia $t \times$ nat $ipA \times$ nat ipB	\longrightarrow bool $\{ ipA \in compus(t) \wedge ipB \in compus(t) \}$
darCaminoMasCorto	:	topologia $t \times$ nat $a \times$ nat b	\longrightarrow secu(nat) $\{ conectados?(t, a, b) \}$
darRutas	:	topologia \times nat \times nat \times conj(nat) \times secu(nat)	\longrightarrow conj(secu(nat))
darRutasVecinas	:	topologia \times conj(nat) \times nat \times conj(nat) \times secu(nat)	\longrightarrow conj(secu(nat))
longMenorSec	:	conj(secu(α))	\longrightarrow nat
secusDeLongK	:	conj(secu(α)) \times nat	\longrightarrow conj(secu(α))
mapII ₁	:	conj(tupla(nat \times nat))	\longrightarrow conj(nat)
mapII ₂	:	conj(tupla(nat \times nat))	\longrightarrow conj(nat)

axiomas $\forall t$: topologia, $\forall ipNueva, ip, ipA, ipB, ifA, ifB, cantInterfaces, k$: nat, $\forall conjDuplas$: conj(tupla(nat, nat)), $\forall cs, rec, vecinas$: conj(nat), $\forall secus$: conj(secu(α)), $\forall sc$: conj(secu(α)), $\forall ruta$: secu(nat)

compus(NuevaTopo)	$\equiv \emptyset$
compus(Compu($t, ipNueva, cantInterfaces$))	$\equiv Ag(ipNueva, compus(t))$
compus(Cable(t, ipA, ifA, ipB, ifB))	$\equiv compus(t)$
cablesEn(NuevaTopo, ip)	$\equiv \emptyset$
cablesEn(Compu($t, ipNueva, cantInterfaces$), ip)	$\equiv cablesEn(t, ip)$

```

cablesEn(Cable(t, ipA, ifA, ipB, ifB), ip)    ≡ if ip = ipA then Ag(⟨ ifA, ipB ⟩, ∅) else ∅ fi ∪
                                     if ip = ipB then Ag(⟨ ifB, ipA ⟩, ∅) else ∅ fi ∪
                                     cablesEn(t, ip)

#interfaces(NuevaTopo, ip)                    ≡ 0

#interfaces(Compu(t, ipNueva, cantInterfaces), ip) ≡ if ip = ipNueva then
                                     cantInterfaces
                                     else
                                     #interfaces(t)
                                     fi

#interfaces(Cable(t, ipA, ifA, ipB, ifB), ip) ≡ #interfaces(t)

interfacesOcupadasDe(t, ip)                  ≡ mapΠ1(cablesEn(t, ip))

vecinas(t, ip)                              ≡ mapΠ2(cablesEn(t, ip))

conectados?(t, ipA, ipB)                    ≡ ¬ ∅?(darRutas(t, ipA, ipB, ∅, <>))

darRutas(t, ipA, ipB, rec, ruta)            ≡ if ipB ∈ vecinas(t, ipA) then
                                     Ag(ruta & (ipA • ipB • <>), ∅)
                                     else
                                     if ∅?(vecinas(t, ipA) - rec) then
                                     ∅
                                     else
                                     darRutas(t, dameUno(vecinas(t, ipA) - rec), ipB,
                                     Ag(ipA, rec), ruta ∘ ipA) ∪
                                     darRutasVecinas(t, sinUno(vecinas(t, ipA) - rec), ipB,
                                     Ag(ipA, rec), ruta ∘ ipA)
                                     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 ∅?(secus) then
                                     0
                                     else
                                     min(long(dameUno(secus)),
                                     longMenorSec(sinUno(secus)))
                                     fi

mapΠ1(conjDuplas)                          ≡ if ∅?(conjDuplas) then
                                     ∅
                                     else
                                     Ag(Π1(dameUno(conjDuplas)),
                                     mapΠ1(sinUno(conjDuplas)))
                                     fi

```

```
mapΠ2(conjDuplas)      ≡ if ∅?(conjDuplas) then  
                        ∅  
                        else  
                        Ag(Π2(dameUno(conjDuplas)),  
                          mapΠ2(sinUno(conjDuplas)))  
                        fi
```

Fin TAD