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 $\equiv$ $\{res =_{\text{obs}} \text{vacio}()\}$
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: `dicc`$_{trie}$`(puntero(compuDCNet))`,
conjPaquetesDCNet: `conj(paqueteDCNet)`,
laQueMásEnvió: `puntero(compuDCNet)`)

donde `compuDCNet` es `tupla`(*pc*: `puntero(compu)`,
conjPaquetes: `conj(paquete)`,
diccPaquetesDCNet: `dicc`$_{avl}$`(nat, itConj(paqueteDCNet))`,
colaPaquetesDCNet: `colaPrioridad(nat, itConj(paqueteDCNet))`,
paqueteAEnviar: `itConj(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

$\text{Rep} : \text{estr} \longrightarrow \text{bool}$

Rep(e) $\equiv$ true $\Longleftrightarrow$
(#(computadoras(e.topologia)) = long(e.vectorCompusDCNet) = #(claves(e.diccCompusDCNet))) $\wedge_L$
($\forall c$: compu)($c \in$ computadoras(e.topologia) $\Rightarrow$
(
($\exists cd$: compuDCNet) (está?(cd, e.vectorCompusDCNet) $\wedge$ cd.pc = puntero(c)) $\wedge$
($\exists s$: string)(def?(s, e.diccCompusDCNet) $\wedge$ $s = c$.ip)
)
) $\wedge_L$
($\forall cd$: compuDCNet)(está?(cd, e.vectorCompusDCNet) $\Rightarrow_L$
($\exists s$: string) (def?(s, e.diccCompusDCNet) $\wedge$
$s = cd$.pc$\rightarrow$ip $\wedge_L$ obtener(s, e.diccCompusDCNet) = puntero(cd))
) $\wedge_L$
($\exists cd$: compuDCNet)(está?(cd, e.vectorCompusDCNet) $\wedge_L$
*(cd.pc) = compuQueMásEnvió(e.vectorCompusDCNet) $\wedge$ e.laQueMásEnvió = puntero(cd)) $\wedge_L$
($\forall cd_1$: 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)
)
)
) $\wedge_L$
($\forall cd$: compuDCNet)(está?(cd, e.vectorCompusDCNet) $\Rightarrow$
(
(#(cd.conjPaquetes) = #(claves(cd.diccPaquetesDCNet))) $\wedge_L$
($\forall p$: paquete)($p \in cd$.conjPaquetes $\Rightarrow$
(
((p.origen $\in$ computadoras(e.topologia) $\wedge$ p.destino $\in$ computadoras(e.topologia) $\wedge$
p.destino $\neq$ *(cd.pc)) $\wedge_L$
($\exists sc$: 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) $\wedge_L$
(Siguiente(obtener(n, e.diccPaquetesDCNet).it) = p $\wedge$
((p.origen = *(cd.pc) $\wedge$ obtener(n, e.diccPaquetesDCNet).recorrido = *(cd.pc) $\bullet$ <>) $\vee$
(p.origen $\neq$ *(cd.pc) $\wedge$
obtener(n, e.diccPaquetesDCNet).recorrido $\in$ caminosMinimos(e.topologia, p.origen, *(cd.pc))))
)
) $\wedge_L$
($\emptyset$?(cd.conjPaquetes) $\Leftrightarrow$ vacía?(cd.colaPaquetesDCNet)) $\wedge$
($\neg$vacía?(cd.colaPaquetesDCNet) $\Rightarrow_L$
($\exists n$: nat)(def?(n, cd.diccPaquetesDCNet) $\wedge_L$
(
Siguiente(obtener(n, cd.diccPaquetesDCNet).it) = paqueteMásPrioridad(cd.conjPaquetes) $\wedge$
proximo(cd.colaPaquetesDCNet) = puntero(obtener(n, cd.diccPaquetesDCNet))
))
) $\wedge$
(cd.enviados $\geq$ enviadosCompu(*(cd.pc), e.vectorCompusDCNet))
)
)

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

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

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

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

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

altaPrioridad : conj(paquetes) cp $\longrightarrow$ nat {$\neg\emptyset$?(cp)}

enviadosCompu : compu $\times$ secu(compuDCNet) $\longrightarrow$ nat aparicionesCompu : compu × conj(nat) cn × dicc(nat × 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, significado(dameUno(cn), dpd).recorrido) **then**
    1 + aparicionesCompu(c, sinUno(cn), dpd)
  else
    aparicionesCompu(c, sinUno(cn), dpd)
  fi
fi

1.2.3. Función de Abstracción

Abs : estr e ⟶ dcnet {Rep(e)}

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

)
)

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 $\equiv$ $\{(r =_{\text{obs}} r_0) \wedge ((\forall\, c'\text{: compu})\ (c' \in \text{computadoras}(r) \Rightarrow \text{ip}(c) \neq \text{ip}(c')))\ \}$
Post $\equiv$ $\{r =_{\text{obs}} \text{agregarComputadora}(r_0, c))\ \}$
Complejidad: $O(L + n)$
Descripción: Agrega una computadora a la red
Aliasing: La compu se agrega por copia

CONECTAR(**in/out** r : `red`, **in** c : `compu`, **in** c' : `compu`, **in** i : `compu`, **in** i' : `compu`)
Pre $\equiv$ $\{(r =_{\text{obs}} r_0) \wedge (c \in \text{computadoras}(r)) \wedge (c' \in \text{computadoras(r)}) \wedge (\text{ip}(c) \neq \text{ip}(c'))$
$\wedge\ (\neg\text{conectadas?}(r, c, c')) \wedge (\neg\text{usaInterfaz?}(r, c, i) \wedge \neg\text{usaInterfaz?}(r, c', i'))\}$
Post $\equiv$ $\{r =_{\text{obs}} \text{conectar}(r_0, c, i, c', i'))\}$
Complejidad: $O(L)$?
Descripción: Conecta dos computadoras

COMPUTADORAS(**in** r : `red`) $\rightarrow$ *res* : `conj(compu)`
Pre $\equiv$ {true}
Post $\equiv$ $\{\text{alias}(res =_{\text{obs}} \text{computadoras}(r))\}$
Complejidad: $O(1)$
Descripción: Devuelve las computadoras de la red| [Devuelve una referancia no modificable]

CONECTADAS?(**in** r : `red`, **in** c : `compu`, **in** c' : `compu`) $\rightarrow$ *res* : `bool`
Pre $\equiv$ $\{(c \in \text{computadoras}(r)) \wedge (c' \in \text{computadoras}(r))\}$
Post $\equiv$ $\{res =_{\text{obs}} \text{conectadas?}(r, c, c')\}$
Complejidad: $O(1)$
Descripción: Indica si dos computadoras de la red estan conectadas

INTERFAZUSADA(**in** r : `red`, **in** c : `compu`, **in** c' : `compu`) $\rightarrow$ *res* : `interfaz`
Pre $\equiv$ $\{\text{conectadas?}(r, c, c')\}$
Post $\equiv$ $\{res =_{\text{obs}} \text{interfazUsada}(r, c, c')\}$
Complejidad: $O(L + n)$
Descripción: Devuelve la interfaz con la cual se conecta c con c'

VECINOS(**in** r : `red`, **in** c : `compu`) $\rightarrow$ *res* : `conj(compu)`
Pre $\equiv$ $\{c \in \text{computadoras}(r)\}$
Post $\equiv$ $\{res =_{\text{obs}} \text{vecinos}(r, c)\}$
Complejidad: $O(n)$
Descripción: Devuelve el conjunto de computadoras conectadas con c
Aliasing: Devuelve una copia de las computadoras conectadas a 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: $O(L+n)$
Descripción: Indica si la interfaz i es usada por la computadora c

CAMINOSMINIMOS(**in** r : `red`, **in** c : `compu`, **in** c' : `compu`) $\rightarrow res$: `conj(lista(compu))`
Pre $\equiv$ $\{(c \in$ computadoras$(r)) \wedge (c' \in$ computadoras(r)$)\}$
Post $\equiv$ $\{$alias$(res =_{\text{obs}}$ caminosMinimos$(r,\ c,\ i))\}$
Complejidad: $O(L)$
Descripción: Devuelve el conjunto de caminos minimos de c a c'
Aliasing: Devuelve una refencia no modificable

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 $\equiv$ $\{res =_{\text{obs}}$ hayCamino?$(r,\ c,\ i)\}$
Complejidad: $O(L)$
Descripción: Indica si existe algún camino entre c y c'

COPIAR(**in** r : `red`) $\rightarrow res$: `red`
Pre $\equiv$ $\{$true$\}$
Post $\equiv$ $\{res =_{\text{obs}} r\}$
Complejidad: $O(?)$
Descripción: Devuelve una copia la red

$\bullet = \bullet$(**in** r : `red`, **in** r' : `red`) $\rightarrow res$: `bool`
Pre $\equiv$ $\{$true$\}$
Post $\equiv$ $\{res =_{\text{obs}} (r =_{\text{obs}} r')\}$
Complejidad: $O(?)$
Descripción: Indica si r es igual a r'

2.2. Representación

2.2.1. Estructura

`red` se representa con `estr`

donde `estr` es `tupla`(*compus*: `conj(compu)` ,
dns: $\texttt{dicc}_{Trie}$`(nodoRed)`)

donde `nodoRed` es `tupla`(*pc*: `puntero(compu)` ,
caminos: $\texttt{dicc}_{Trie}$`(conj(lista(compu)))` ,
conexiones: $\texttt{dicc}_{Lineal}$`(nat, puntero(nodoRed))`)

donde `compu` es `tupla`(*ip*: `string`, *interfaces*: `conj(nat)`)

2.2.2. Invariante de Representación

(I) Todas los elementos de *compus* deben tener IPs distintas.

(II) Para cada compu, el trie *dns* define para la clave <IP de esa compu> un `nodoRed` cuyo *pc* es puntero a esa compu.

(III) `nodoRed`.*conexiones* contiene como claves todas las interfaces usadas de la compu *c* (que tienen que estar en *pc*.interfaces)

(IV) Ningun nodo se conecta con si mismo.

(V) Ningun nodo se conecta a otro a traves de dos interfaces distintas.

(VI) Para cada `nodoRed` en *dns*, *caminos* tiene como claves todas las IPs de las compus de la red (`estr`.*compus*), y los significados corresponden a todos los caminos mínimos desde la compu *pc* hacia la compu cuya IP es clave.

$$\text{Rep} : \text{estr} \longrightarrow \text{bool}$$

$$\text{Rep}(e) \equiv \text{true} \Longleftrightarrow ($$

$((\forall c1, c2\text{: compu})\ (c1 \neq c2 \wedge c1 \in e.\text{compus} \wedge c2 \in e.\text{compus}) \Rightarrow c1.\text{ip} \neq c2.\text{ip}) \wedge$

$((\forall c\text{: compu})(c \in e.\text{compus} \Rightarrow$
$(\ \text{def?}(c.\text{ip},\ e.\text{dns}) \wedge_L \text{obtener}(c.\text{ip},\ e.\text{dns}).\text{pc} = \text{puntero}(c)\ )$
$)) \wedge$

$((\forall i\text{: string},\ n\text{: nodoRed})\ ((\text{def?}(i,\ e.\text{dns}) \wedge_L n = \text{obtener}(i,\ e.\text{dns})) \Rightarrow$
$(\exists c\text{: compu})\ (c \in e.\text{compus} \wedge (n.\text{pc} = \text{puntero}(c)))$
$)) \wedge$

$((\forall i\text{: string},\ n\text{: nodoRed})\ ((\text{def?}(i,\ e.\text{dns}) \wedge_L n = \text{obtener}(i,\ e.\text{dns})) \Rightarrow$
$((\forall t\text{: nat})\ (\text{def?}(t,\ n.\text{conexiones}) \Rightarrow (t \in n.\text{pc}{\rightarrow}\text{interfaces})))$
$)) \wedge$

$((\forall i\text{: string},\ n\text{: nodoRed})\ ((\text{def?}(i,\ e.\text{dns}) \wedge_L n = \text{obtener}(i,\ e.\text{dns})) \Rightarrow$
$((\forall t\text{: nat})\ (\text{def?}(t,\ n.\text{conexiones}) \Rightarrow_L (\text{obtener}(t,\ n.\text{conexiones}) \neq \text{puntero}(n)))\ )$
$)) \wedge$

$((\forall i\text{: string},\ n\text{: nodoRed})\ ((\text{def?}(i,\ e.\text{dns}) \wedge_L n = \text{obtener}(i,\ e.\text{dns})) \Rightarrow$
$((\forall t1, t2\text{: nat})\ ((t1 \neq t2 \wedge \text{def?}(t1,\ n.\text{conexiones}) \wedge \text{def?}(t2,\ n.\text{conexiones})) \Rightarrow_L$
$(\text{obtener}(t1,\ n.\text{conexiones}) \neq \text{obtener}(t2,\ n.\text{conexiones}))$
$))$
$)) \wedge$

$((\forall i1, i2\text{: string},\ n1, n2\text{: nodoRed})\ (($
$(\text{def?}(i1,\ e.\text{dns}) \wedge_L n1 = \text{obtener}(i1,\ e.\text{dns})) \wedge$
$(\text{def?}(i2,\ e.\text{dns}) \wedge_L n2 = \text{obtener}(i2,\ e.\text{dns}))$
$) \Rightarrow (\text{def?}(i2,\ n1.\text{caminos}) \wedge_L \text{obtener}(i2,\ n1.\text{caminos}) = \text{darCaminosMinimos}(n1,\ n2))$
$))$

$)$

vecinas	: nodoRed	$\longrightarrow$ conj(nodoRed)	
auxVecinas	: nodoRed $\times$ dicc(nat $\times$ puntero(nodoRed))	$\longrightarrow$ conj(nodoRed)	
secusDeLongK	: conj(secu(α)) $\times$ nat	$\longrightarrow$ conj(secu(α))	
longMenorSec	: conj(secu(α)) *secus*	$\longrightarrow$ nat	$\{\neg\emptyset?(secus)\}$
darRutas	: nodoRed nA $\times$ nodoRed nB $\times$ conj(pc) $\times$ secu(nodoRed)	$\longrightarrow$ conj(secu(nodoRed))	
darRutasVecinas	: conj(pc) vec $\times$ nodoRed n $\times$ conj(pc) $\times$ secu(nodoRed)	$\longrightarrow$ conj(secu(nodoRed))	
darCaminosMinimos	: nodoRed $n1$ $\times$ nodoRed $n1$	$\longrightarrow$ conj(secu(compu))	

vecinas(n) $\equiv$ auxVecinas(n, n.conexiones)

auxVecinas(n, cs) $\equiv$ **if** $\emptyset?(cs)$ **then**
$\emptyset$
else
Ag(obtener(dameUno(claves(cs)), cs), auxVecinas(n, sinUno(cs)))
fi

secusDeLongK($secus$, k) $\equiv$ **if** $\emptyset$?($secus$) **then**
$\emptyset$
else
if long(dameUno($secus$)) $= k$ **then**
dameUno($secus$) $\cup$ secusDeLongK(sinUno($secus$), k)
else
secusDeLongK(sinUno($secus$), k)
fi
fi

longMenorSec($secus$) $\equiv$ **if** $\emptyset$?(sinUno($secus$)) **then**
long(dameUno($secus$))
else
min(long(dameUno($secus$)),
longMenorSec(sinUno($secus$)))
fi

darRutas(nA, nB, rec, $ruta$) $\equiv$ **if** $nB \in$ vecinas(nA) **then**
Ag($ruta \circ nB$, $\emptyset$)
else
if $\emptyset$?(vecinas(nA) - rec) **then**
$\emptyset$
else
darRutas(dameUno(vecinas(nA) - rec),
nB, Ag(nA, rec),
$ruta \circ$ dameUno(vecinas(nA) - rec)) $\cup$
darRutasVecinas(sinUno(vecinas(nA) - rec),
nB, Ag(nA, rec),
$ruta \circ$ dameUno(vecinas(nA) - rec))
fi
fi

darRutasVecinas($vecinas$, n, rec, $ruta$) $\equiv$ **if** $\emptyset$?($vecinas$) **then**
$\emptyset$
else
darRutas(dameUno($vecinas$), n, rec, $ruta$) $\cup$
darRutasVecinas(sinUno($vecinas$), n, rec, $ruta$)
fi

darCaminosMinimos(nA, nB) $\equiv$ secusDeLongK(darRutas(nA, nB, $\emptyset$, $<>$),
longMenorSec(darRutas(nA, nB, $\emptyset$, $<>$))

2.2.3. Función de Abstracción

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

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

3. Módulo Cola de mínima prioridad(α)

El módulo cola de mínima prioridad consiste en una cola de prioridad de elementos del tipo α cuya prioridad está determinada por un *nat* de forma tal que el elemento que se ingrese con el menor *nat* será el de mayor prioridad.

3.1. Especificación

TAD COLA DE MÍNIMA PRIORIDAD(α)

igualdad observacional

$$(\forall c, c' : \text{colaMinPrior}(\alpha)) \left(c =_{\text{obs}} c' \iff \begin{pmatrix} \text{vacía?}(c) =_{\text{obs}} \text{vacía?}(c') \land_{\text{L}} \\ (\neg\text{vacía?}(c) \Rightarrow_{\text{L}} (\text{próximo}(c) =_{\text{obs}} \text{próximo}(c') \land \\ \text{desencolar}(c) \quad =_{\text{obs}} \\ \text{desencolar}(c')) \end{pmatrix} \right)$$

parámetros formales

géneros α

operaciones $\bullet < \bullet$: $\alpha \times \alpha \longrightarrow$ bool

Relación de orden total estricto[1]

géneros colaMinPrior(α)

exporta colaMinPrior(α), generadores, observadores

usa BOOL

observadores básicos

vacía?	: colaMinPrior(α)	$\longrightarrow$ bool	
próximo	: colaMinPrior(α) c	$\longrightarrow$ α	$\{\neg$ vacía?$(c)\}$
desencolar	: colaMinPrior(α) c	$\longrightarrow$ colaMinPrior(α)	$\{\neg$ vacía?$(c)\}$

generadores

vacía	:	$\longrightarrow$ colaMinPrior(α)
encolar	: $\alpha \times$ colaMinPrior(α)	$\longrightarrow$ colaMinPrior(α)

otras operaciones

tamaño	: colaMinPrior(α)	$\longrightarrow$ nat

axiomas $\forall$ c: colaMinPrior(α), $\forall$ e: α

vacía?(vacía)	$\equiv$	true
vacía?(encolar(e, c))	$\equiv$	false
próximo(encolar(e, c))	$\equiv$	**if** vacía?(c) $\lor_{\text{L}}$ proximo$(c) > e$ **then** e **else** próximo(c) **fi**
desencolar(encolar(e, c))	$\equiv$	**if** vacía?(c) $\lor_{\text{L}}$ proximo$(c) > e$ **then** c **else** encolar(e, desencolar(c)) **fi**

Fin TAD

[1] Una relación es un orden total estricto cuando se cumple:

Antirreflexividad: $\neg\, a < a$ para todo $a : \alpha$

Antisimetría: $(a < b \Rightarrow \neg\, b < a)$ para todo $a, b : \alpha$, $a \neq b$

Transitividad: $((a < b \land b < c) \Rightarrow a < c)$ para todo $a, b, c : \alpha$

Totalidad: $(a < b \lor b < a)$ para todo $a, b : \alpha$

3.2. Interfaz

parámetros formales
géneros α

se explica con: COLA DE MÍNIMA PRIORIDAD(NAT).

géneros: `colaMinPrior`(α).

3.2.1. Operaciones básicas de Cola de mínima prioridad

VACÍA() $\to$ *res* : `colaMinPrior`(α)
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ vacía}
Complejidad: O(1)
Descripción: Crea una cola de prioridad vacía

VACÍA?(**in** c: `colaMinPrior`(α)) $\to$ *res* : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}}$ vacía?(c)}
Complejidad: O(1)
Descripción: Devuelve `true` si y sólo si la cola está vacía

DESENCOLAR(**in/out** c: `colaMinPrior`(α)) $\to$ *res* : α
Pre $\equiv$ {$\neg$vacía?(c) $\wedge$ $c =_{\text{obs}} c_0$}
Post $\equiv$ {$res =_{\text{obs}}$ proximo(c_0) $\wedge$ $c =_{\text{obs}}$ desencolar(c_0)}
Complejidad: O(log(tamaño(c)))
Descripción: Quita el elemento más prioritario
Aliasing: Se devuelve el elemento por copia

ENCOLAR(**in/out** c: `colaMinPrior`(α), **in** p: `nat`, **in** a: α)
Pre $\equiv$ {$c =_{\text{obs}} c_0$}
Post $\equiv$ {$c =_{\text{obs}}$ encolar(p,c_0)}
Complejidad: O(log(tamaño(c)))
Descripción: Agrega al elemento α con prioridad p a la cola
Aliasing: Se agrega el elemento por copia

3.3. Representación

3.3.1. Representación de colaMinPrior

`colaMinPrior`(α) **se representa con** `estr`

donde `estr` es `dicc`$_{avl}$`(nat, nodoEncolados)`

donde `nodoEncolados` es `tupla`(*encolados*: `cola`(α), *prioridad*: `nat`)

3.3.2. Invariante de Representación

(I) Todos los significados del diccionario tienen como clave el valor de *prioridad*

(II) Todos los significados del diccionario no pueden tener una cola vacía

Rep : estr $\longrightarrow$ bool

Rep(e) $\equiv$ true $\Longleftrightarrow$
($\forall n$: nat) def?(n, e) $\Rightarrow_L$ ((obtener(n, e).prioridad $=$ n) $\wedge$
$\neg$vacía?(obtener(n, e).encolados))

3.3.3. Función de Abstracción

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

Abs(e) $=_{\text{obs}}$ cmp: colaMinPrior | (vacía?(cmp) $\Leftrightarrow$ (#claves(e) $=$ 0)) $\wedge$
$\neg$vacía?(cmp) $\Rightarrow_L$
((próximo(cmp) $=$ próximo(mínimo(e).encolados)) $\wedge$
(desencolar(cmp) $=$ desencolar(mínimo(e).encolados)))

3.4. Algoritmos

iVacía () $\rightarrow$ res: colaMinPrior(α)

```
res ← Vacio()                                                    O(1)
```

Complejidad : $O(1)$

iVacía? (**in** c: `colaMinPrior`(α)) $\rightarrow$ res: bool

```
res ← (#Claves(c) = 0)                                           O(1)
```

Complejidad : $O(1)$

iDesencolar (**in**/**out** c: `colaMinPrior`(α)) $\rightarrow$ res: α

```
res ← Copiar(Proximo(Minimo(c).encolados))                       O(copy(α))
Desencolar(Minimo(c).encolados)                                  O(log(tamaño(c)))
if EsVacia?(Minimo(c).encolados) then                            O(1)
    Borrar(c, Minimo(c).prioridad)                               O(log(tamaño(c)))
end if
```

Complejidad : $O(log(tamano(c)) + O(copy(\alpha))$

iEncolar (**in**/**out** c: `colaMinPrior`(α), **in** p: `nat`, **in** a : α)

```
if Definido?(p) then                                             O(log(tamaño(c)))
    Encolar(Significado(c, p).encolados, a)                      O(log(tamaño(c)) + copy(α))
else
    nodoEncolados nuevoNodoEncolados                             O(1)
    nuevoNodoEncolados.encolados ← Vacia()                       O(1)
    nuevoNodoEncolados.prioridad ← p                             O(1)
    Encolar(nuevoNodoEncolados.encolados, a)                     O(copy(a))
    Definir(c, p, nuevoNodoEncolados)                            O(log(tamaño(c)) + copy(nodoEncolados))
end if
```

Complejidad : $O(log(tamano(c)) + O(copy(\alpha))$