

# 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

<b>1. Módulo DCNet</b>	<b>3</b>
1.1. Interfaz	3
1.1.1. Operaciones básicas de mapa	3
1.2. Representación	3
1.2.1. Representación de dcnet	3
1.2.2. Invariante de Representación	3
1.2.3. Función de Abstracción	6
<b>2. Módulo Red</b>	<b>7</b>
2.1. Interfaz	7
2.2. Representación	8
2.2.1. Estructura	8
2.2.2. Invariante de Representación	8
2.2.3. Función de Abstracción	9
<b>3. Módulo Cola de mínima prioridad(<math>\alpha</math>)</b>	<b>10</b>
3.1. Especificación	10
3.2. Interfaz	11
3.2.1. Operaciones básicas de Cola de mínima prioridad	11
3.3. Representación	11
3.3.1. Representación de colaMinPrior	11
3.3.2. Invariante de Representación	11
3.3.3. Función de Abstracción	12
3.4. Algoritmos	12

# 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**  $\equiv \{res =_{obs} vacio()\}$

**Complejidad:**  $O(1)$

**Descripción:** crea un mapa nuevo

## 1.2. Representación

### 1.2.1. Representación de dcnet

dcnet se representa con estr

donde estr es tupla(*topología*: red,  
                           *vectorCompusDCNet*: vector(compuDCNet),  
                           *diccCompusDCNet*: dicc<sub>trie</sub>(puntero(compuDCNet)),  
                           *laQueMásEnvió*: puntero(compuDCNet))

donde compuDCNet es tupla(*pc*: puntero(compu),  
                               *conjPaquetes*: conj(paquete),  
                               *diccPaquetesDCNet*: dicc<sub>avl</sub>(nat, paqueteDCNet),  
                               *colaPaquetesDCNet*: colaPrioridad(nat, puntero(paqueteDCNet)),  
                               *paqueteAEnviar*: paqueteDCNet, *enviados*: nat)

donde paqueteDCNet es tupla(*it*: itConj(paquete), *recorrido*: lista(compu))

donde paquete es tupla(*id*: nat, *prioridad*: nat, *origen*: compu, *destino*: compu)

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

### 1.2.2. Invariante de Representación

- (I) Las compus de los elementos de vectorCompusDCNet son punteros a todas las compus de la topología
- (II) Las claves de diccCompusDCNet son todos los hostnames de la topología
- (III) Los significados de diccCompusDCNet son punteros que apuntan a las compuDCNet cuyo hostname equivale a su clave en vectorCompusDCNet
- (IV) laQueMásEnvió es un puntero a la compuDCNet en vectorCompusDCNet que más paquetes enviados tiene. Si no hay compus es NULL
- (V) Todos los paquetes en conjPaquetes de cada compuDCNet tienen id único y tanto su origen como destino existen en la topología
- (VI) El paquete en conjPaquetes tiene que tener en su recorrido a la compuDCNet en la que se encuentra y esta no puede ser igual al destino del recorrido

- (VII) Las claves de `diccPaquetesDCNet` son los id de los paquetes en `conjPaquetes`
- (VIII) Los significados de `diccPaquetesDCNet` contienen un `itConj` que apunta al paquete con el id equivalente a su clave y en recorrido, un camino mínimo válido para el origen del paquete y la compu en la que se encuentra
- (IX) La `colaPaquetesDCNet` es vacía si y sólo si `conjPaquetes` lo es, si no lo es, su próximo es un puntero a un paqueteDCNet de `diccPaquetesDCNet` que contiene un `itConj` cuyo siguiente es uno de los paquetes de `conjPaquetes` con mayor prioridad
- (X) La cantidad de enviados de una `compuDCNet` es igual o mayor a la cantidad de apariciones de esa compu en los caminos recorridos de paquetes en la red

Rep : `estr`  $\rightarrow$  `bool`

Rep(*e*)  $\equiv$  `true`  $\iff$

$$\begin{aligned}
 & (\#(\text{computadoras}(e.\text{topologia})) = \text{long}(e.\text{vectorCompusDCNet}) = \#(\text{claves}(e.\text{diccCompusDCNet}))) \wedge_L \\
 & (\forall c: \text{compu})(c \in \text{computadoras}(e.\text{topologia}) \Rightarrow \\
 & \quad ( \\
 & \quad \quad (\exists cd: \text{compuDCNet})(\text{está?}(cd, e.\text{vectorCompusDCNet}) \wedge cd.pc = \text{puntero}(c)) \wedge \\
 & \quad \quad (\exists s: \text{string})(\text{def?}(s, e.\text{diccCompusDCNet}) \wedge s = c.ip) \\
 & \quad ) \\
 & ) \wedge_L \\
 & (\forall cd: \text{compuDCNet})(\text{está?}(cd, e.\text{vectorCompusDCNet}) \Rightarrow_L \\
 & \quad (\exists s: \text{string})(\text{def?}(s, e.\text{diccCompusDCNet}) \wedge \\
 & \quad \quad s = cd.pc \rightarrow ip \wedge_L \text{obtener}(s, e.\text{diccCompusDCNet}) = \text{puntero}(cd)) \\
 & ) \wedge_L \\
 & (\exists cd: \text{compuDCNet})(\text{está?}(cd, e.\text{vectorCompusDCNet}) \wedge_L \\
 & \quad * (cd.pc) = \text{compuQueMásEnvió}(e.\text{vectorCompusDCNet}) \wedge e.\text{laQueMásEnvió} = \text{puntero}(cd)) \wedge_L \\
 & (\forall cd_1: \text{compuDCNet})(\text{está?}(cd_1, e.\text{vectorCompusDCNet}) \Rightarrow \\
 & \quad (\forall p_1: \text{paquete})(p_1 \in cd_1.\text{conjPaquetes} \Rightarrow \\
 & \quad \quad (\forall cd_2: \text{compuDCNet})(\text{está?}(cd_2, e.\text{vectorCompusDCNet}) \wedge cd_1 \neq cd_2) \Rightarrow \\
 & \quad \quad (\forall p_2: \text{paquete})(p_2 \in cd_2.\text{conjPaquetes} \Rightarrow p_1.id \neq p_2.id) \\
 & \quad ) \\
 & ) \\
 & ) \wedge_L \\
 & (\forall cd: \text{compuDCNet})(\text{está?}(cd, e.\text{vectorCompusDCNet}) \Rightarrow \\
 & \quad ( \\
 & \quad \quad (\#(cd.\text{conjPaquetes}) = \#(\text{claves}(cd.\text{diccPaquetesDCNet}))) \wedge_L \\
 & \quad \quad (\forall p: \text{paquete})(p \in cd.\text{conjPaquetes} \Rightarrow \\
 & \quad \quad \quad ( \\
 & \quad \quad \quad \quad ((p.\text{origen} \in \text{computadoras}(e.\text{topologia}) \wedge p.\text{destino} \in \text{computadoras}(e.\text{topologia}) \wedge \\
 & \quad \quad \quad \quad \quad p.\text{destino} \neq *(cd.pc)) \wedge_L \\
 & \quad \quad \quad \quad (\exists sc: \text{secu}(\text{compu}))(sc \in \text{caminosMinimos}(e.\text{topologia}, p.\text{origen}, p.\text{destino}) \wedge \text{está?}(*(cd.pc), sc))) \wedge \\
 & \quad \quad \quad \quad (\exists n: \text{nat})((\text{def?}(n, cd.\text{diccPaquetesDCNet}) \wedge p.id = n) \wedge_L \\
 & \quad \quad \quad \quad \quad (\text{Siguiente}(\text{obtener}(n, e.\text{diccPaquetesDCNet}).it) = p \wedge \\
 & \quad \quad \quad \quad \quad ((p.\text{origen} = *(cd.pc) \wedge \text{obtener}(n, e.\text{diccPaquetesDCNet}).\text{recorrido} = *(cd.pc) \bullet \langle \rangle) \vee \\
 & \quad \quad \quad \quad \quad (p.\text{origen} \neq *(cd.pc) \wedge \\
 & \quad \quad \quad \quad \quad \quad \text{obtener}(n, e.\text{diccPaquetesDCNet}).\text{recorrido} \in \text{caminosMinimos}(e.\text{topologia}, p.\text{origen}, *(cd.pc)))) \\
 & \quad \quad \quad ) \\
 & \quad \quad ) \wedge_L \\
 & \quad \quad (\emptyset?(cd.\text{conjPaquetes}) \Leftrightarrow \text{vacía?}(cd.\text{colaPaquetesDCNet})) \wedge \\
 & \quad \quad (\neg \text{vacía?}(cd.\text{colaPaquetesDCNet}) \Rightarrow_L \\
 & \quad \quad \quad (\exists n: \text{nat})(\text{def?}(n, cd.\text{diccPaquetesDCNet}) \wedge_L \\
 & \quad \quad \quad \quad ( \\
 & \quad \quad \quad \quad \quad \text{Siguiente}(\text{obtener}(n, cd.\text{diccPaquetesDCNet}).it) = \text{paqueteMásPrioridad}(cd.\text{conjPaquetes}) \wedge \\
 & \quad \quad \quad \quad \quad \text{proximo}(cd.\text{colaPaquetesDCNet}) = \text{puntero}(\text{obtener}(n, cd.\text{diccPaquetesDCNet})) \\
 & \quad \quad \quad \quad ) \\
 & \quad \quad \quad ) \wedge \\
 & \quad \quad \quad (cd.\text{enviados} \geq \text{enviadosCompu}(*(cd.pc), e.\text{vectorCompusDCNet})) \\
 & \quad \quad ) \\
 & ) \\
 & )
 \end{aligned}$$

$\text{compuQueMásEnvió} : \text{secu}(\text{compuDCNet}) \text{ } scd \longrightarrow \text{compu} \quad \{\neg \text{vacía?}(scd)\}$   
 $\text{maxEnviado} : \text{secu}(\text{compuDCNet}) \text{ } scd \longrightarrow \text{nat} \quad \{\neg \text{vacía?}(scd)\}$   
 $\text{enviaronK} : \text{secu}(\text{compuDCNet}) \times \text{nat} \longrightarrow \text{conj}(\text{compu})$   
 $\text{paqueteMásPrioridad} : \text{conj}(\text{paquete}) \text{ } cp \longrightarrow \text{paquete} \quad \{\neg \emptyset?(cp)\}$   
 $\text{paquetesConPrioridadK} : \text{conj}(cp) \times \text{nat} \longrightarrow \text{conj}(\text{paquete})$   
 $\text{altaPrioridad} : \text{conj}(\text{paquetes}) \text{ } cp \longrightarrow \text{nat} \quad \{\neg \emptyset?(cp)\}$   
 $\text{enviadosCompu} : \text{compu} \times \text{secu}(\text{compuDCNet}) \longrightarrow \text{nat}$   
 $\text{aparicionesCompu} : \text{compu} \times \text{conj}(\text{nat}) \text{ } cn \times \text{dicc}(\text{nat} \times \text{paqueteDCNet}) \text{ } dp \longrightarrow \text{nat} \quad \{\text{claves}(dp) \subseteq cn\}$

$\text{compuQueMásEnvió}(scd) \equiv \text{dameUno}(\text{enviaronK}(scd, \text{maxEnviado}(scd)))$   
 $\text{maxEnviado}(scd) \equiv \text{if } \text{vacía?}(\text{fin}(scd)) \text{ then } \text{prim}(scd).\text{enviados} \text{ else } \text{max}(\text{prim}(scd), \text{maxEnviado}(\text{fin}(scd))) \text{ fi}$   
 $\text{enviaronK}(scd, k) \equiv \text{if } \text{vacía?}(scd) \text{ then}$   
 $\quad \emptyset$   
 $\quad \text{else}$   
 $\quad \quad \text{if } \text{prim}(scd).\text{enviados} = k \text{ then}$   
 $\quad \quad \quad \text{Ag}(*(\text{prim}(scd).\text{pc}), \text{enviaronK}(\text{fin}(scd), k))$   
 $\quad \quad \text{else}$   
 $\quad \quad \quad \text{enviaronK}(\text{fin}(scd), k)$   
 $\quad \text{fi}$   
 $\text{fi}$   
 $\text{paqueteMásPrioridad}(dcn, cp) \equiv \text{dameUno}(\text{paquetesConPrioridadK}(cp, \text{altaPrioridad}(cp)))$   
 $\text{altaPrioridad}(cp) \equiv \text{if } \emptyset?(\text{sinUno}(cp)) \text{ then}$   
 $\quad \text{dameUno}(cp).\text{prioridad}$   
 $\quad \text{else}$   
 $\quad \quad \text{min}(\text{dameUno}(cp).\text{prioridad}, \text{altaPrioridad}(\text{sinUno}(cp)))$   
 $\quad \text{fi}$   
 $\text{fi}$   
 $\text{paquetesConPrioridadK}(cp, k) \equiv \text{if } \emptyset?(cp) \text{ then}$   
 $\quad \emptyset$   
 $\quad \text{else}$   
 $\quad \quad \text{if } \text{dameUno}(cp).\text{prioridad} = k \text{ then}$   
 $\quad \quad \quad \text{Ag}(\text{dameUno}(cp), \text{paquetesConPrioridadK}(\text{sinUno}(cp), k))$   
 $\quad \quad \text{else}$   
 $\quad \quad \quad \text{paquetesConPrioridadK}(\text{sinUno}(cp), k)$   
 $\quad \quad \text{fi}$   
 $\quad \text{fi}$   
 $\text{fi}$   
 $\text{enviadosCompu}(c, scd) \equiv \text{if } \text{vacía?}(scd) \text{ then}$   
 $\quad 0$   
 $\quad \text{else}$   
 $\quad \quad \text{if } \text{prim}(scd) = c \text{ then}$   
 $\quad \quad \quad \text{enviadosCompu}(c, \text{fin}(scd))$   
 $\quad \quad \text{else}$   
 $\quad \quad \quad \text{aparicionesCompu}(c, \text{claves}(\text{prim}(scd).\text{diccPaquetesDCNet}),$   
 $\quad \quad \quad \text{prim}(scd).\text{diccPaquetesDCNet}) + \text{enviadosCompu}(c, \text{fin}(scd))$   
 $\quad \quad \text{fi}$   
 $\quad \text{fi}$

```

aparicionesCompu(c, cn, dpc)  $\equiv$  if  $\emptyset?(cn)$  then
    0
else
    if está?(c, significado(dameUno(cn), dpc).recorrido) then
        1 + aparicionesCompu(c, sinUno(cn), dpc)
    else
        aparicionesCompu(c, sinUno(cn), dpc)
    fi
fi

```

### 1.2.3. Función de Abstracción

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

Abs(*e*) =<sub>obs</sub> dcn: dcnet | red(*dcn*) = *e*.topología  $\wedge$   
 $(\forall cdn: \text{compuDCNet})(\text{está?}(cdn, e.\text{vectorCompusDCNet}) \Rightarrow_{\mathcal{L}}$   
enEspera(*dcn*, \*(*cdn*.pc)) = *cdn*.conjPaquetes  $\wedge$   
cantidadEnviados(*dcn*, \*(*cdn*.pc)) = *cdn*.enviados  $\wedge$   
 $(\forall p: \text{paquete})(p \in cdn.\text{conjPaquetes} \Rightarrow_{\mathcal{L}}$   
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**  $\equiv \{res =_{obs} iniciarRed\}$   
**Complejidad:**  $O(1)$   
**Descripción:** Crea una red nueva

AGREGARCOMPUTADORA(**in/out**  $r : red$ , **in**  $c : compu$ )  
**Pre**  $\equiv \{(r =_{obs} r_0) \wedge ((\forall c' : compu) (c' \in computadoras(r) \Rightarrow ip(c) \neq ip(c')))\}$   
**Post**  $\equiv \{r =_{obs} agregarComputadora(r_0, c)\}$   
**Complejidad:**  $O(L + n)$   
**Descripción:** Agrega un computadora a la red

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

COMPUTADORAS(**in**  $r : red$ )  $\rightarrow res : conj(compu)$   
**Pre**  $\equiv \{true\}$   
**Post**  $\equiv \{res =_{obs} computadoras(r)\}$   
**Complejidad:**  $O(1)$   
**Descripción:** Devuelve las computadoras de la red  
**Aliasing:** Devuelve una referencia no modificable

CONECTADAS?(**in**  $r : red$ , **in**  $c : compu$ , **in**  $c' : compu$ )  $\rightarrow res : bool$   
**Pre**  $\equiv \{(c \in computadoras(r)) \wedge (c' \in computadoras(r))\}$   
**Post**  $\equiv \{res =_{obs} conectadas?(r, c, c')\}$   
**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**  $\equiv \{conectadas?(r, c, c')\}$   
**Post**  $\equiv \{res =_{obs} interfazUsada(r, c, c')\}$   
**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**  $\equiv \{c \in computadoras(r)\}$   
**Post**  $\equiv \{res =_{obs} vecinos(r, c)\}$   
**Complejidad:**  $O(n)$   
**Descripción:** Devuelve el conjunto de computadoras conectadas con c  
**Aliasing:** Devuelve una copia de las computadoras conectadas a c

USAIINTERFAZ?(**in**  $r$  : red, **in**  $c$  : compu, **in**  $i$  : interfaz)  $\rightarrow res$  : bool

**Pre**  $\equiv \{c \in computadoras(r)\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{usaInterfaz?}(r, c, i)\}$

**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(secu(compu))

**Pre**  $\equiv \{(c \in computadoras(r)) \wedge (c' \in computadoras(r))\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{caminosMinimos}(r, c, i)\}$

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

**Descripción:** Devuelve el conjunto de caminos minimos de  $c$  a  $c'$

**Aliasing:** Devuelve una referencia 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**  $\equiv \{res =_{\text{obs}} \text{hayCamino?}(r, c, i)\}$

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

**Descripción:** Indica si existe un/os camino/s  $c$  de  $a$  a  $c'$

COPIAR(**in**  $r$  : red)  $\rightarrow res$  : red

**Pre**  $\equiv \{\text{true}\}$

**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**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{res =_{\text{obs}} (r =_{\text{obs}} r')\}$

**Complejidad:**  $O(?)$

**Descripción:** Indica si son  $r$  es igual a  $r'$

## 2.2. Representación

### 2.2.1. Estructura

red se representa con **estr**

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

donde **nodoRed** es  $\text{tupla}(c: \text{puntero}(\text{compu}),$   
 $\text{caminos: dicc}_{\text{Trie}}(\text{ip}, \text{conj}(\text{lista}(\text{compu}))) ,$   
 $\text{conexiones: dicc}_{\text{Lineal}}(\text{interfaz}, \text{compu}) )$

### 2.2.2. Invariante de Representación

- (I) Todas las compus deben tener IPs distintas.
- (II) Ninguna compu se conecta con si misma.
- (III) Ninguna compu se conecta a otra a través de dos interfaces distintas.



- (IV) El trie **estr.dns** apunta a un **nodoRed** por cada elemento de *compus*.
- (V) En cada **nodoRed**, *c* tiene que apuntar a un elemento de **estr.compus**.
- (VI) Para cada **nodoRed**, *caminos* tiene como claves todas las IPs de las compus de la red, y los significados corresponden a todos los caminos mínimos desde la compu *c* hacia la compu cuya IP es clave.
- (VII) **nodoRed.conexiones** contiene como claves todas las **interfaz** usaconedadas de la compu *c* (que tienen que estar en *c.interfaces*)

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

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

### 2.2.3. Función de Abstracción

$\text{Abs} : \text{estr } e \rightarrow \text{red}$

$\{\text{Rep}(e)\}$

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

### 3. Módulo Cola de mínima prioridad( $\alpha$ )

El módulo cola de mínima prioridad consiste en una cola de prioridad de elementos del tipo  $\alpha$  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( $\alpha$ )**

**igualdad observacional**

$$(\forall c, c' : \text{colaMinPrior}(\alpha)) \left( c =_{\text{obs}} c' \iff \left( \begin{array}{l} \text{vacía?}(c) =_{\text{obs}} \text{vacía?}(c') \wedge_{\text{L}} \\ (\neg \text{vacía?}(c) \Rightarrow_{\text{L}} (\text{próximo}(c) =_{\text{obs}} \text{próximo}(c') \wedge \\ \text{desencolar}(c) =_{\text{obs}} \text{desencolar}(c')) \end{array} \right) \right)$$

**parámetros formales**

**géneros**  $\alpha$

**operaciones**  $\bullet < \bullet : \alpha \times \alpha \longrightarrow \text{bool}$

Relación de orden total estricto<sup>1</sup>

**géneros**  $\text{colaMinPrior}(\alpha)$

**exporta**  $\text{colaMinPrior}(\alpha)$ , generadores, observadores

**usa** **BOOL**

**observadores básicos**

$\text{vacía?} : \text{colaMinPrior}(\alpha) \longrightarrow \text{bool}$

$\text{próximo} : \text{colaMinPrior}(\alpha) \longrightarrow \alpha$   $\{\neg \text{vacía?}(c)\}$

$\text{desencolar} : \text{colaMinPrior}(\alpha) \longrightarrow \text{colaMinPrior}(\alpha)$   $\{\neg \text{vacía?}(c)\}$

**generadores**

$\text{vacía} : \longrightarrow \text{colaMinPrior}(\alpha)$

$\text{encolar} : \alpha \times \text{colaMinPrior}(\alpha) \longrightarrow \text{colaMinPrior}(\alpha)$

**otras operaciones**

$\text{tamaño} : \text{colaMinPrior}(\alpha) \longrightarrow \text{nat}$

**axiomas**  $\forall c : \text{colaMinPrior}(\alpha), \forall e : \alpha$

$\text{vacía?}(\text{vacía}) \equiv \text{true}$

$\text{vacía?}(\text{encolar}(e, c)) \equiv \text{false}$

$\text{próximo}(\text{encolar}(e, c)) \equiv \text{if } \text{vacía?}(c) \vee_{\text{L}} \text{próximo}(c) > e \text{ then } e \text{ else } \text{próximo}(c) \text{ fi}$

$\text{desencolar}(\text{encolar}(e, c)) \equiv \text{if } \text{vacía?}(c) \vee_{\text{L}} \text{próximo}(c) > e \text{ then } c \text{ else } \text{encolar}(e, \text{desencolar}(c)) \text{ fi}$

**Fin TAD**

<sup>1</sup>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:**  $((a < b \wedge b < c) \Rightarrow a < c)$  para todo  $a, b, c : \alpha$

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

### 3.2. Interfaz

**parámetros formales**  
**géneros**       $\alpha$

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

**géneros:** colaMinPrior( $\alpha$ ).

#### 3.2.1. Operaciones básicas de Cola de mínima prioridad

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

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

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

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

### 3.3. Representación

#### 3.3.1. Representación de colaMinPrior

$\text{colaMinPrior}(\alpha)$  se representa con **estr**

donde **estr** es  $\text{dicc}_{\text{avl}}(\text{nat}, \text{nodoEncolados})$

donde **nodoEncolados** es  $\text{tupla}(\text{encolados} : \text{cola}(\alpha), \text{prioridad} : \text{nat})$

#### 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** :  $\text{estr} \rightarrow \text{bool}$

$$\text{Rep}(e) \equiv \text{true} \iff (\forall n : \text{nat}) \text{def?}(n, e) \Rightarrow_{\text{L}} ((\text{obtener}(n, e).\text{prioridad} = n) \wedge \neg \text{vacía?}(\text{obtener}(n, e).\text{encolados}))$$

### 3.3.3. Función de Abstracción

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

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

## 3.4. Algoritmos

iVacía () → res: colaMinPrior(α)

res ← Vacío ()

O(1)

**Complejidad :** O(1)

iVacía? (in c: colaMinPrior(α)) → res: bool

res ← (#Claves(c) = 0)

O(1)

**Complejidad :** O(1)

iDesencolar (in/out c: colaMinPrior(α)) → res: α

res ← Copiar(Proximo(Minimo(c).encolados))

O(copy(α))

Desencolar(Minimo(c).encolados)

O(log(tamaño(c)))

if EsVacía?(Minimo(c).encolados) then

O(1)

    Borrar(c, Minimo(c).prioridad)

O(log(tamaño(c)))

end if

**Complejidad :** O(log(tamaño(c)) + O(copy(α)))

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 ← Vacía()

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(tamaño(c)) + O(copy(α)))