

# Algoritmos y Estructuras de Datos II

## Trabajo Práctico 2

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

Segundo Cuatrimestre de 2014

### Grupo 16

Apellido y Nombre	LU	E-mail
Juan Ernesto Rinaudo	864/13	jangamesdev@hotmail.com
Mauro Cherubini	835/13	cheru.mf@gmail.com
Federico Beuter	827/13	federicobeuter@gmail.com
Fernando Frassia	340/13	ferfrassia@gmail.com

Reservado para la cátedra

Instancia	Docente que corrigió	Calificación
Primera Entrega		
Recuperatorio		

# Índice

<b>1. TAD Extendidos</b>	<b>3</b>
1.1. $\text{Secu}(\alpha)$	3
1.2. Mapa	3
<b>2. Mapa</b>	<b>4</b>
2.1. Representacion	4
2.2. InvRep y Abs	5
2.3. Algoritmos	5
<b>3. Ciudad Robótica</b>	<b>7</b>
3.1. Representacion	8
3.2. InvRep y Abs	8
3.3. Algoritmos	10
<b>4. Diccionario <math>\text{String}(\alpha)</math></b>	<b>13</b>
4.1. Representacion	13
4.2. InvRep y Abs	14
4.3. Algoritmos	14
<b>5. Cola Prioritaria</b>	<b>16</b>
5.1. TAD COLAPRIORITARIA	16
5.2. Representacion	18
5.3. InvRep y Abs	18
5.4. Algoritmos	19

## 1. Tad Extendidos

### 1.1. Secu( $\alpha$ )

#### otras operaciones

elemDeSecu : Secu( $\alpha$ )  $s \times \text{Nat } n \longrightarrow \text{RUR}$

$\{n < \text{long}(s)\}$

#### axiomas

elemDeSecu( $s, n$ )  $\equiv$  **if**  $n = 0$  **then**  $\text{prim}(s)$  **else**  $\text{elemDeSecu}(\text{fin}(s), n-1)$  **fi**

### 1.2. Mapa

#### observadores básicos

restricciones : Mapa  $m \longrightarrow \text{secu}(\text{restriccion})$

nroConexion : estacion  $e_1 \times \text{estacion } e_2 \times \text{Mapa } m \longrightarrow \text{nat}\{e_1, e_2 \subset \text{estaciones}(m) \wedge_L \text{conectadas?}(e_1, e_2, m)\}$

#### axiomas

restricciones(vacio)  $\equiv \langle \rangle$

restricciones(agregar( $e, m$ ))  $\equiv \text{restricciones}(m)$

restricciones(conectar( $e_1, e_2, r, m$ ))  $\equiv \text{restricciones}(m) \circ r$

nroConexion( $e_1, e_2, \text{conectar}(e_3, e_4, m)$ )  $\equiv$  **if**  $((e_1 = e_3 \wedge e_2 = e_4) \vee (e_1 = e_4 \wedge e_2 = e_3))$  **then**  
long(restricciones( $m$ )) - 1

**else**

nroConexion( $e_1, e_2, m$ ) - 1

**fi**

nroConexion( $e_1, e_2, \text{agregar}(e, m)$ )  $\equiv \text{nroConexion}(e_1, e_2, m)$

## 2. Mapa

### Interfaz

se explica con:  $\text{RED, ITERADOR UNIDIRECCIONAL}(\alpha)$ .

géneros:  $\text{red, itConj(Compu)}$ .

### Operaciones básicas de Red

$\text{COMPUTADORAS}(\text{in } r : \text{red}) \rightarrow res : \text{itConj(Compu)}$

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

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

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve las computadoras de red.

$\text{CONECTADAS?}(\text{in } r : \text{red, in } c_1 : \text{compu, in } c_2 : \text{compu}) \rightarrow res : \text{bool}$

**Pre**  $\equiv \{\{c_1, c_2\} \subseteq \text{computadoras}(r)\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{conectadas?}(r, c_1, c_2)\}$

**Complejidad:**  $\mathcal{O}(|c_1| + |c_2|)$

**Descripción:** Devuelve el valor de verdad indicado por la conexión o desconexión de dos computadoras.

$\text{INTERFAZUSADA}(\text{in } r : \text{red, in } c_1 : \text{compu, in } c_2 : \text{compu}) \rightarrow res : \text{interfaz}$

**Pre**  $\equiv \{\{c_1, c_2\} \subseteq \text{computadoras}(r) \wedge_L \text{conectadas?}(r, c_1, c_2)\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{interfazUsada}(r, c_1, c_2)\}$

**Complejidad:**  $\mathcal{O}(|c_1| + |c_2|)$

**Descripción:** Devuelve la interfaz que  $c_1$  usa para conectarse con  $c_2$

$\text{INICIARRED}() \rightarrow res : \text{red}$

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

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

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Crea una red sin computadoras.

$\text{AGREGARCOMPUTADORA}(\text{in/out } r : \text{red, in } c : \text{compu})$

**Pre**  $\equiv \{r_0 =_{\text{obs}} r \wedge \neg(c \in \text{computadoras}(r))\}$

**Post**  $\equiv \{r =_{\text{obs}} \text{agregarComputadora}(r_0, c)\}$

**Complejidad:**  $\mathcal{O}(|c|)$

**Descripción:** Agrega una computadora a la red.

$\text{CONECTAR}(\text{in/out } r : \text{red, in } c_1 : \text{compu, in } i_1 : \text{interfaz, in } c_2 : \text{compu, in } i_2 : \text{interfaz})$

**Pre**  $\equiv \{r_0 =_{\text{obs}} r \wedge \{c_1, c_2\} \subseteq \text{computadoras}(r) \wedge \text{ip}(c_1) \neq \text{ip}(c_2) \wedge_L \neg \text{conectadas?}(r, c_1, c_2) \wedge \neg \text{usaInterfaz?}(r, c_1, i_1) \wedge \neg \text{usaInterfaz?}(r, c_2, i_2)\}$

**Post**  $\equiv \{r =_{\text{obs}} \text{conectar}(r, c_1, i_1, c_2, i_2)\}$

**Complejidad:**  $\mathcal{O}(|c_1| + |c_2|)$

**Descripción:** Conecta dos computadoras y les añade la interfaz correspondiente.

### 2.1. Representacionrepresentacionn

### Representación

red se representa con  $\text{e\_red}$

donde  $\text{e\_red}$  es  $\text{tupla}(\text{vecinosEInterfaces: diccString(compu: string, diccString(compu: string, interfaz: nat))}, \text{deOrigenADestino: diccString(compu: string, diccString(compu: string, secu(compu): secu(string)))}, \text{computadoras: conj(compu)})$

## 2.2. InvRep y Abs

1. El conjunto de claves de "uniones" es igual al conjunto de estaciones "estaciones".
2. "#sendas" es igual a la mitad de las horas de "uniones".
3. Todo valor que se obtiene de buscar el significado del significado de cada clave de "uniones", es igual el valor hallado tras buscar en "uniones" con el significado de la clave como clave y la clave como significado de esta nueva clave, y no hay otras hojas ademas de estas dos, con el mismo valor.
4. Todas las hojas de "uniones" son mayores o iguales a cero y menores a "#sendas".
5. La longitud de "sendas" es mayor o igual a "#sendas".

Rep : e\_mapa  $\rightarrow$  bool

Rep(*m*)  $\equiv$  true  $\iff$

- m.estaciones = claves(m.uniones)  $\wedge$  1.
- m.#sendas = #sendasPorDos(m.estaciones, m.uniones) / 2  $\wedge$  m.#sendas  $\leq$  long(m.sendas)  $\wedge_L$  2. 5.
- ( $\forall$  e1, e2: string)(e1  $\in$  claves(m.uniones)  $\wedge_L$  e2  $\in$  claves(obtener(e1, m.uniones))  $\Rightarrow_L$  e2  $\in$  claves(m.uniones)  $\wedge_L$  e1  $\in$  claves(obtener(e2, m.uniones))  $\wedge_L$  obtener(e2, obtener(e1, m.uniones)) = obtener(e1, obtener(e2, m.uniones))  $\wedge$  obtener(e2, obtener(e1, m.uniones)) < m.#sendas)  $\wedge$  3. 4.
- ( $\forall$  e1, e2, e3, e4: string)((e1  $\in$  claves(m.uniones)  $\wedge_L$  e2  $\in$  claves(obtener(e1, m.uniones))  $\wedge_L$  e3  $\in$  claves(m.uniones)  $\wedge_L$  e4  $\in$  claves(obtener(e3, m.uniones)))  $\Rightarrow_L$  (obtener(e2, obtener(e1, m.uniones)) = obtener(e4, obtener(e3, m.uniones)))  $\iff$  (e1 = e3  $\wedge$  e2 = e4)  $\vee$  (e1 = e4  $\wedge$  e2 = e3)))) 3.

#sendasPorDos : conj( $\alpha$ ) c  $\times$  dicc( $\alpha \times$  dicc( $\alpha \times \beta$ )) d  $\rightarrow$  nat {c  $\subset$  claves(d)}

#sendasPorDos(c, d)  $\equiv$  **if**  $\emptyset?(c)$  **then**  
 0  
**else**  
 #claves(obtener(dameUno(c), d)) + #sendasPorDos(sinUno(c), d)  
**fi**

Abs : e\_mapa *m*  $\rightarrow$  mapa

{Rep(*m*)}

Abs(*m*) =<sub>obs</sub> p: mapa |

m.estaciones = estaciones(p)  $\wedge_L$   
 ( $\forall$  e1, e2: string)((e1  $\in$  estaciones(p)  $\wedge$  e2  $\in$  estaciones(p))  $\Rightarrow_L$   
 (conectadas?(e1, e2, p)  $\iff$   
 e1  $\in$  claves(m.uniones)  $\wedge$  e2  $\in$  claves(obtener(e2, m.uniones))))  $\wedge_L$   
 ( $\forall$  e1, e2: string)((e1  $\in$  estaciones(p)  $\wedge$  e2  $\in$  estaciones(p))  $\wedge_L$   
 conectadas?(e1, e2, p)  $\Rightarrow_L$   
 (restriccion(e1, e2, p) = m.sendas[obtener(e2, obtener(e1, m.uniones))]  $\wedge$  nroConexion(e1, e2, m) = obtener(e2, obtener(e1, m.uniones)))  $\wedge$  long(restricciones(p)) = m.#sendas  $\wedge_L$  ( $\forall$  n:nat) (n < m.#sendas  $\Rightarrow_L$  m.sendas[n] = ElemDeSecu(restricciones(p), n)))

## 2.3. Algoritmos

### Algoritmos

ICOMPUTADORAS(in *r* : red)  $\rightarrow$  res : itConj(Compu)

1: res  $\leftarrow$  CrearIt(*r.computadoras*)

$\mathcal{O}(1)$

**Complejidad:**  $\mathcal{O}(1)$

ICONECTADAS?(**in**  $r$  : red, **in**  $c_1$  : compu, **in**  $c_2$  : compu)  $\rightarrow res$  : bool

1:  $res \leftarrow \text{Definido?}(\text{Significado}(r.\text{vecinosEInterfaces}, c_1), c_2)$

$\mathcal{O}(|c_1| + |c_2|)$

**Complejidad:**  $\mathcal{O}(|c_1| + |c_2|)$

IINTERFAZUSADA(**in**  $r$  : red, **in**  $c_1$  : compu, **in**  $c_2$  : compu)  $\rightarrow res$  : interfaz

1:  $res \leftarrow \text{Significado}(\text{Significado}(r.\text{vecinosEInterfaces}, c_1), c_2)$

$\mathcal{O}(|c_1| + |c_2|)$

**Complejidad:**  $\mathcal{O}(|c_1| + |c_2|)$

INICIARRED()  $\rightarrow res$  : red

1:  $res \leftarrow \text{tupla}(\text{vecinosEInterfaces: Vacío}(), \text{deOrigenADestino: Vacío}(), \text{computadoras: Vacío}())$   $\mathcal{O}(1+1+1)$

**Complejidad:**  $\mathcal{O}(1)$

$\mathcal{O}(1) + \mathcal{O}(1) + \mathcal{O}(1) =$

$3 * \mathcal{O}(1) = \mathcal{O}(1)$

IAGREGARCOMPUTADORA(**in/out**  $r$  : red, **in**  $c$  : compu)

1:  $\text{Agregar}(r.\text{computadoras}, c)$

$\mathcal{O}(1)$

2:  $\text{Definir}(r.\text{vecinosEInterfaces}, c, \text{Vacío}())$

$\mathcal{O}(|c|)$

3:  $\text{Definir}(r.\text{deOrigenADestino}, c, \text{Vacío}())$

$\mathcal{O}(|c|)$

**Complejidad:**  $\mathcal{O}(|c|)$

$\mathcal{O}(1) + \mathcal{O}(|c|) + \mathcal{O}(|c|) =$

$2 * \mathcal{O}(|c|) = \mathcal{O}(|c|)$

ICONECTAR(**in/out**  $r$  : red, **in**  $c_1$  : compu, **in**  $i_1$  : interfaz, **in**  $c_2$  : compu, **in**  $i_2$  : interfaz)

1:  $\text{Definir}(\text{Significado}(r.\text{vecinosEInterfaces}, c_1), c_2, i_1)$

$\mathcal{O}(|c_1| + |c_2| + 1)$

2:  $\text{Definir}(\text{Significado}(r.\text{vecinosEInterfaces}, c_2), c_1, i_2)$

$\mathcal{O}(|c_2| + |c_1| + 1)$

3:

**Complejidad:**  $\mathcal{O}(|e_1| + |e_2|)$

$\mathcal{O}(|e_1| + |e_2|) + \mathcal{O}(|e_1| + |e_2|) + \mathcal{O}(1) + \mathcal{O}(1) =$

$2 * \mathcal{O}(1) + 2 * \mathcal{O}(|e_1| + |e_2|) =$

$2 * \mathcal{O}(|e_1| + |e_2|) = \mathcal{O}(|e_1| + |e_2|)$

### 3. DCNet

#### Interfaz

se explica con: DCNET, ITERADOR UNIDIRECCIONAL( $\alpha$ ).

géneros: dcnet.

#### Operaciones básicas de DCNet

RED(in  $d$ : dcnet)  $\rightarrow res$  : red

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

Post  $\equiv \{res =_{\text{obs}} \text{red}(d)\}$

Complejidad:  $\mathcal{O}(1)$

Descripción: Devuelve la red del dcnet.

CAMINORECORRIDO(in  $d$ : dcnet, in  $p$ : paquete)  $\rightarrow res$  : secu(compu)

Pre  $\equiv \{p \in \text{paqueteEnTransito?}(d, p)\}$

Post  $\equiv \{res =_{\text{obs}} \text{caminoRecorrido}(d, p)\}$

Complejidad:  $\mathcal{O}(n * \log_2(K))$

Descripción: Devuelve una secuencia con las computadoras por las que paso el paquete.

CANTIDADENVIADOS(in  $d$ : dcnet, in  $c$ : compu)  $\rightarrow res$  : nat

Pre  $\equiv \{c \in \text{computadoras}(\text{red}(d))\}$

Post  $\equiv \{res =_{\text{obs}} \text{cantidadEnviados}(d, c)\}$

Complejidad:  $\mathcal{O}(1)$

Descripción: Devuelve la cantidad de paquetes que fueron enviados desde la computadora.

ENESPERA(in  $d$ : dcnet, in  $c$ : compu)  $\rightarrow res$  : conj(paquete)

Pre  $\equiv \{c \in \text{computadoras}(\text{red}(d))\}$

Post  $\equiv \{res =_{\text{obs}} \text{enEspera}(d, c)\}$

Complejidad:  $\mathcal{O}(1)$

Descripción: Devuelve los paquetes que se encuentran en ese momento en la computadora.

INICIARDCNET(in  $r$ : red)  $\rightarrow res$  : dcnet

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

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

Complejidad:  $\mathcal{O}(1)$

Descripción: Inicia un dcnet con la red y sin paquetes.

CREARPAQUETE(in  $p$ : paquete, in/out  $d$ : dcnet)

Pre  $\equiv \{d_0 \equiv d \wedge \neg ((\exists p_1: \text{paquete})(\text{paqueteEnTransito}(s, p_1) \wedge \text{id}(p_1) = \text{id}(p)) \wedge \text{origen}(p) \in \text{computadoras}(\text{red}(d)) \wedge_{\text{L}} \text{destino}(p) \in \text{computadoras}(\text{red}(d)) \wedge_{\text{L}} \text{hayCamino?}(\text{red}(d, \text{origen}(p), \text{destino}(p))) \}$

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

Complejidad:  $\mathcal{O}()$

Descripción: Agrega el paquete al dcnet.

AVANZARSEGUNDO(in/out  $d$ : dcnet)

Pre  $\equiv \{d_0 \equiv d\}$

Post  $\equiv \{d =_{\text{obs}} \text{avanzarSegundo}(c_0)\}$

Complejidad:  $\mathcal{O}()$

Descripción: El paquete de mayor prioridad de cada computadora avanza a su proxima computadora siendo esta la del camino mas corto.

#### Operaciones del iterador

CREARIT(in  $c$ : ciudad)  $\rightarrow res$  : itRURs

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

Post  $\equiv \{res =_{\text{obs}} \text{CrearItUni}(\text{robots}(c))\}$

Complejidad:  $\mathcal{O}(1)$

**Descripción:** Crea el iterador de robots.

**ACTUAL**(*in it*: itRURs)  $\rightarrow$  *res* : rur

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

**Post**  $\equiv$  {*res* =<sub>obs</sub> Actual(*it*)}

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Devuelve el actual del iterador de robots.

**AVANZAR**(*in it*: itRURs)  $\rightarrow$  *res* : itRURs

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

**Post**  $\equiv$  {*res* =<sub>obs</sub> Avanzar(*it*)}

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Avanza el iterador de robots.

**HAYMAS?**(*in it*: itRURs)  $\rightarrow$  *res* : bool

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

**Post**  $\equiv$  {*res* =<sub>obs</sub> HayMas?(*it*)}

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Se fija si hay mas elementos en el iterador de robots.

### 3.1. Representacion

## Representación

ciudad se representa con **e\_cr**

donde **e\_cr** es `tupla(mapa: mapa, RUREnEst: diccString(estacion: string, robs: colaP(id: nat, inf: nat)),  
RURs: vector de tupla(id: nat, esta?: bool, e: string, inf: nat, carac: conj(string),  
sendEv: arreglo_dimensionable de bool), #RURHistoricos: nat)`

### 3.2. InvRep y Abs

1. El conjunto de estaciones de 'mapa' es igual al conjunto con todas las claves de 'RUREnEst'.
2. La longitud de 'RURs' es mayor o igual a '#RURHistoricos'.
3. Todos los elementos de 'RURs' cumplen que su primer componente ('id') corresponde con su posicion en 'RURs'. Su Componente 'e' es una de las estaciones de 'mapa', su componente 'esta?' es true si y solo si hay estaciones tales que su valor asignado en 'uniones' es igual a su indice en 'RURs'. Su Componente 'inf' puede ser mayor a cero solamente si hay algun elemento en 'sendEv' tal que sea false. Cada elemento de 'sendEv' es igual a verificar 'carac' con la estriccion obtenida al buscar el elemento con la misma posicion en la secuencia de restricciones de 'mapa'.
4. Cada valor contenido en la cola del significado de cada estacion de las claves de 'uniones' pertenecen unicamente a la cola asociada a dicha estacion y a ninguna otra de las colas asociadas a otras estaciones. Y cada uno de estos valores es menor a '#RURHistoricos' y mayor o igual a cero. Ademas la componente 'e' del elemento de la posicion igual a cada valor de las colas asociadas a cada estacion, es igual a la estacion asociada a la cola a la que pertenece el valor.

**Rep** : **e\_cr**  $\longrightarrow$  bool



$\text{Rep}(c) \equiv \text{true} \iff \text{claves}(\text{c.RURenEst}) = \text{estaciones}(\text{c.mapa}) \wedge$  1  
 $\# \text{RURHistoricos} \leq \text{Long}(\text{c.RURs}) \wedge_L (\forall i:\text{Nat}, t:<\text{id}:\text{Nat}, \text{esta?}:\text{Bool}, e:\text{String},$  2  
 $\text{inf}:\text{Nat}, \text{carac}:\text{Conj}(\text{Tag}), \text{sendEv}:\text{ad}(\text{Bool})>)$   
 $(i < \# \text{RURHistoricos} \wedge_L \text{ElemDeSecu}(\text{c.RURs}, i) = t \Rightarrow_L (t.e \in \text{estaciones}(\text{c.mapa})$  3  
 $\wedge t.\text{id} = i \wedge \text{tam}(\text{t.sendEv}) = \text{long}(\text{Restricciones}(\text{c.mapa})) \wedge$   
 $(t.\text{inf} > 0 \Rightarrow (\exists j:\text{Nat}) (j < \text{tam}(\text{t.sendEv}) \wedge_L \neg (t.\text{sendEv}[j]))) \wedge$   
 $(t.\text{esta?} \Leftrightarrow (\exists e1:\text{String}) (e1 \in \text{claves}(\text{c.RURenEst}) \wedge_L \text{estaEnColaP?}(\text{obtener}(e1, \text{c.RURenEst}), t.\text{id})))$   
 $\wedge (\forall h:\text{Nat}) (h < \text{tam}(\text{t.sendEv}) \Rightarrow_L$   
 $t.\text{sendEv}[h] = \text{verifica?}(t.\text{carac}, \text{ElemDeSecu}(\text{Restricciones}(\text{c.mapa}), h)))) \wedge_L$   
 $(\forall e1, e2:\text{String}) (e1 \in \text{claves}(\text{c.RURenEst}) \wedge e2 \in \text{claves}(\text{c.RURenEst}) \wedge e1 \neq e2 \Rightarrow_L$  4  
 $(\forall n:\text{Nat}) (\text{estaEnColaP?}(\text{obtener}(e1, \text{c.RURenEst}), n) \Rightarrow \neg \text{estaEnColaP?}(\text{obtener}(e2, \text{c.RURenEst}), n))$   
 $\wedge n < \# \text{RURHistoricos} \wedge_L \text{ElemDeSecu}(\text{c.RURs}, n).e = e1))$

$\text{estaEnColaP?} : \text{ColaPri} \times \text{Nat} \longrightarrow \text{Bool}$

$\text{estaEnColaP?}(\text{cp}, n) \equiv \text{if vacia?}(\text{cp}) \text{ then}$   
 $\quad \text{false}$   
 $\quad \text{else}$   
 $\quad \quad \text{if desencolar}(\text{cp}) = n \text{ then}$   
 $\quad \quad \quad \text{true}$   
 $\quad \quad \text{else}$   
 $\quad \quad \quad \text{estaEnColaP?}(\text{Eliminar}(\text{cp}, \text{desencolar}(\text{cp})), n)$   
 $\quad \text{fi}$   
 $\text{fi}$

$\text{Abs} : e\_cr\ c \longrightarrow \text{ciudad}$  {Rep(c)}  
 $\text{Abs}(c) =_{\text{obs}} u: \text{ciudad} \mid$   
 $\quad c.\# \text{RURHistoricos} = \text{ProximoRUR}(U) \wedge c.\text{mapa} = \text{mapa}(u) \wedge_L$   
 $\quad \text{robots}(u) = \text{RURQueEstan}(\text{c.RURs}) \wedge_L$   
 $\quad (\forall n:\text{Nat}) (n \in \text{robots}(u) \Rightarrow_L \text{estacion}(n, u) = \text{c.RURs}[n].e \wedge$   
 $\quad \text{tags}(n, u) = \text{c.RURs}[n].\text{carac} \wedge \# \text{infracciones}(n, u) = \text{c.RURs}[n].\text{inf})$

$\text{RURQueEstan} : \text{secu}(\text{tupla}) \longrightarrow \text{Conj}(\text{RUR})$

$\text{tupla es } <\text{id}:\text{Nat}, \text{esta?}:\text{Bool}, \text{inf}:\text{Nat}, \text{carac}:\text{Conj}(\text{tag}), \text{sendEv}:\text{arreglo dimensionable}(\text{bool})>$

$\text{RURQueEstan}(s) \equiv \text{if vacia?}(s) \text{ then}$   
 $\quad \emptyset$   
 $\quad \text{else}$   
 $\quad \quad \text{if } \Pi_2(\text{prim}(\text{fin}(s))) \text{ then}$   
 $\quad \quad \quad \Pi_1(\text{prim}(\text{fin}(s))) \cup \text{RURQueEstan}(\text{fin}(s))$   
 $\quad \quad \text{else}$   
 $\quad \quad \quad \text{RURQueEstan}(\text{fin}(s))$   
 $\quad \text{fi}$   
 $\text{fi}$

**it se representa con e\_it**

donde  $e\_it$  es  $\text{tupla}(i: \text{nat}, \text{maxI}: \text{nat}, \text{ciudad}: \text{puntero}(\text{ciudad}))$

$\text{Rep} : e\_it \longrightarrow \text{bool}$

$\text{Rep}(it) \equiv \text{true} \iff it.i \leq it.\text{maxI} \wedge \text{maxI} = \text{ciudad}.\# \text{RURHistoricos}$

$\text{Abs} : e\_it\ u \longrightarrow \text{itUni}(\alpha)$

{Rep(u)}

$Abs(u) =_{obs} it: itUni(\alpha) \mid (HayMas?(u) \wedge_L Actual(u) = ciudad.RURs[it.i] \wedge Siguietes(u, \emptyset) = VSiguietes(ciudad, it.i++, \emptyset) \vee (\neg HayMas?(u)))$

$Siguietes : itUni \times conj(RURs)cr \longrightarrow conj(RURs)$

$Siguietes(u, cr) \equiv \text{if } HayMas(u)? \text{ then } Ag(Actual(Avanzar(u)), Siguietes(Avanzar(u), cr)) \text{ else } Ag(\emptyset, cr) \text{ fi}$

$VSiguietes : ciudadc \times Nati \times conj(RURs)cr \longrightarrow conj(RURs)$

$VSiguietes(u, i, cr) \equiv \text{if } i < c.\#RURHistoricos \text{ then } Ag(c.RURs[i], VSiguietes(u, i++, cr)) \text{ else } Ag(\emptyset, cr) \text{ fi}$

### 3.3. Algoritmos

#### Algoritmos

**I**PRÓXIMORUR(**in**  $c: ciudad$ )  $\rightarrow res : rur$

1:  $res \leftarrow (c.\#RURHistoricos)$

$\mathcal{O}(1)$

**Complejidad:**  $\mathcal{O}(1)$

**I**MAPA(**in**  $c: ciudad$ )  $\rightarrow res : mapa$

1:  $res \leftarrow c.mapa$

$\mathcal{O}(1)$

**Complejidad:**  $\mathcal{O}(1)$

**I**ROBOTS(**in**  $c: ciudad$ )  $\rightarrow res : itRobots$

1:  $res \leftarrow CrearIt(c.RURs)$

$\mathcal{O}(1)$

**Complejidad:**  $\mathcal{O}(1)$

**I**ESTACIÓN(**in**  $u: rur$ , **in**  $c: ciudad$ )  $\rightarrow res : estación$

1:  $res \leftarrow (c.RURs[u]).estacion$

$\mathcal{O}(1)$

**Complejidad:**  $\mathcal{O}(1)$

**I**TAGS(**in**  $u: rur$ , **in**  $c: ciudad$ )  $\rightarrow res : conj(tags)$

1:  $res \leftarrow (c.RURs[u]).carac$

$\mathcal{O}(1)$

**Complejidad:**  $\mathcal{O}(1)$

**I**#INFRACCIONES(**in**  $u: rur$ , **in**  $c: ciudad$ )  $\rightarrow res : nat$

1:  $res \leftarrow (c.RURs[u]).inf$

$\mathcal{O}(1)$

**Complejidad:**  $\mathcal{O}(1)$

ICREAR(in  $m$ : mapa)  $\rightarrow res$  : ciudad

```

1:  $res \leftarrow \text{tupla}(mapa: m, RUREnEst: \text{Vacío}(), RURs: \text{Vacía}(), \#RURHistoricos: 0)$   $\mathcal{O}(1)$ 
2:  $\text{var } it: \text{itConj}(\text{Estacion}) \leftarrow \text{Estaciones}(m)$   $\mathcal{O}(1)$ 
3: while HaySiguiente( $it$ ) do  $\mathcal{O}(1)$ 
4:   Definir( $res.RUREnEst$ , Siguiente( $it$ ), Vacío())  $\mathcal{O}(|e_m|)$ 
5:   Avanzar( $it$ )  $\mathcal{O}(1)$ 
6: end while

```

**Complejidad:**  $\mathcal{O}(\text{Cardinal}(\text{Estaciones}(m)) * |e_m|)$

$\mathcal{O}(1) + \mathcal{O}(1) + \sum_{i=1}^{\text{Cardinal}(\text{Estaciones}(m))} (\mathcal{O}(|e_m|) + \mathcal{O}(1)) =$   
 $2 * \mathcal{O}(1) + \text{Cardinal}(\text{Estaciones}(m)) * (\mathcal{O}(|e_m|) + \mathcal{O}(1)) =$   
 $\text{Cardinal}(\text{Estaciones}(m)) * (\mathcal{O}(|e_m|))$

IENTRAR(in  $ts$ : conj(tags), in  $e$ : string, in/out  $c$ : ciudad)

```

1: Agregar(Significado( $c.RUREnEst$ ,  $e$ ), 0,  $c.\#RURHistoricos$ )  $\mathcal{O}(\log_2 n + |e|)$ 
2: Agregar( $c.RURs$ ,  $c.\#RURHistoricos$ ,  $\text{tupla}(id: c.\#RURHistoricos, esta?: true, estacion: e, inf: 0, carac: ts, sendEv: \text{EvaluarSendas}(ts, c.mapa))$ )  $\mathcal{O}(1 + S * R)$ 
3:  $c.\#RURHistoricos++$   $\mathcal{O}(1)$ 

```

**Complejidad:**  $\mathcal{O}(\log_2 n + |e| + S * R)$

$\mathcal{O}(\log_2 n + |e|) + \mathcal{O}(1 + S * R) + \mathcal{O}(1) = \mathcal{O}(\log_2 n + |e| + S * R)$

IMOVER(in  $u$ : rur, in  $e$ : estación, in/out  $c$ : ciudad)

```

1: Eliminar(Significado( $c.RUREnEst$ ,  $c.RURs[u].estacion$ ),  $c.RURs[u].inf$ ,  $u$ )  $\mathcal{O}(|e| + \log_2 N_{e_0})$ 
2: Agregar(Significado( $c.RUREnEst$ ,  $e$ ),  $c.RURs[u].inf$ ,  $u$ )  $\mathcal{O}(|e| + \log_2 N_e)$ 
3: if  $\neg(c.RURs[u].sendEv[\text{NroConexion}(c.RURs[u].estacion, e, c.mapa)])$  then  $\mathcal{O}(|e_0| + |e|)$ 
4:    $c.RURs[u].inf++$   $\mathcal{O}(1)$ 
5: end if
6:  $c.RURs[u].estacion \leftarrow e$   $\mathcal{O}(1)$ 

```

**Complejidad:**  $\mathcal{O}(|e| + \log_2 N_e)$

$\mathcal{O}(|e| + \log_2 N_{e_0}) + \mathcal{O}(|e| + \log_2 N_e) + \mathcal{O}(|e_0|, |e|) + \max(\mathcal{O}(1), \mathcal{O}(0)) + \mathcal{O}(1) =$   
 $\mathcal{O}(2 * |e| + \log_2 N_e + \log_2 N_{e_0}) + \mathcal{O}(|e_0| + |e|) + 2 * \mathcal{O}(1) =$   
 $\mathcal{O}(|e| + \log_2 N_e + \log_2 N_{e_0}) + \mathcal{O}(|e_0| + |e|) =$   
 $\mathcal{O}(2 * |e| + |e_0| + \log_2 N_e + \log_2 N_{e_0}) = \mathcal{O}(|e| + |e_0| + \log_2 N_e + \log_2 N_{e_0})$  Donde  $e_0$  es  $c.RURs[u].estacion$  antes de modificar el valor

IINSPECCIÓN(in  $e$ : estación, in/out  $c$ : ciudad)

```

1:  $\text{var } rur: \text{nat} \leftarrow \text{Desencolar}(\text{Significado}(c.RUREnEst, e))$   $\mathcal{O}(\log_2 N)$ 
2:  $c.RURs[rur].esta? \leftarrow false$   $\mathcal{O}(1)$ 

```

**Complejidad:**  $\mathcal{O}(\log_2 N)$

$\mathcal{O}(\log_2 N) + \mathcal{O}(1) = \mathcal{O}(\log_2 N)$

ICREARIT(in  $c$ : ciudad)  $\rightarrow res$  : itRURs

```

1:  $itRURS \leftarrow \text{tupla}(i: 0, maxI: c.\#RURHistoricos, ciudad: \&c)$   $\mathcal{O}(1)$ 

```

**Complejidad:**  $\mathcal{O}(1)$

**IACTUAL**(**in**  $it : \text{itRURs}$ )  $\rightarrow res : \text{rur}$

1:  $res \leftarrow (it.ciudad \rightarrow RURs)[it.i]$

$\mathcal{O}(1)$

**Complejidad:**  $\mathcal{O}(1)$

**IAVANZAR**(**in**  $it : \text{itRURs}$ )  $\rightarrow res : \text{itRURs}$

1:  $it.i++$

$\mathcal{O}(1)$

**Complejidad:**  $\mathcal{O}(1)$

**IHAYMAS?**(**in**  $it : \text{itRURs}$ )  $\rightarrow res : \text{bool}$

1:  $res \leftarrow (it.i < it.maxI)$

$\mathcal{O}(1)$

**Complejidad:**  $\mathcal{O}(1)$

## 4. Diccionario String( $\alpha$ )

### Interfaz

**parámetros formales**

**géneros**

**función** COPIA(**in**  $d : \alpha$ )  $\rightarrow res : \alpha$   
**Pre**  $\equiv \{\text{true}\}$   
**Post**  $\equiv \{res =_{\text{obs}} a\}$   
**Complejidad:**  $\Theta(\text{copy}(a))$   
**Descripción:** función de copia de  $\alpha$ 's

**se explica con:** DICCIONARIO(String,  $\alpha$ ).

**géneros:** diccString( $\alpha$ ).

### Operaciones básicas de Restricción

VACÍO()  $\rightarrow res : \text{diccString}(\alpha)$

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

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

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Crea nuevo diccionario vacío.

DEFINIR(**in/out**  $d : \text{diccString}(\alpha)$ , **in**  $clv : \text{string}$ , **in**  $def : \alpha$ )

**Pre**  $\equiv \{d_0 =_{\text{obs}} d\}$

**Post**  $\equiv \{d =_{\text{obs}} \text{definir}(clv, def, d)\}$

**Complejidad:**  $\mathcal{O}(|clv|)$

**Descripción:** Agrega una nueva definición.

DEFINIDO?(**in**  $d : \text{diccString}(\alpha)$ , **in**  $clv : \text{string}$ )  $\rightarrow res : \text{bool}$

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

**Post**  $\equiv \{res =_{\text{obs}} \text{def?}(clv, d)\}$

**Complejidad:**  $\mathcal{O}(|clv|)$

**Descripción:** Revisa si la clave ingresada se encuentra definida en el Diccionario.

SIGNIFICADO(**in**  $d : \text{diccString}(\alpha)$ , **in**  $clv : \text{string}$ )  $\rightarrow res : \text{diccString}(\alpha)$

**Pre**  $\equiv \{\text{def?}(d, clv)\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{obtener}(clv, d)\}$

**Complejidad:**  $\mathcal{O}(|clv|)$

**Descripción:** Devuelve la definición correspondiente a la clave.

#### 4.1. Representación

### Representación

Esta no es la versión posta de la descripción, es solo un boceto.

Para representar el diccionario de Trie vamos a utilizar una estructura que contiene el primer Nodo y la cantidad de Claves en el diccionario. Para los nodos se utilizó una estructura formada por una tupla, el primer elemento es el significado de la clave y el segundo es un arreglo de 256 elementos que contiene punteros a los hijos del nodo (por todos los posibles caracteres ASCII).

Para conseguir el número de orden de un char tengo las funciones ord.

`diccString( $\alpha$ )` se representa con `e_nodo`

donde `e_nodo` es `tupla(definicion: puntero( $\alpha$ ), hijos: arreglo[256] de puntero(e_nodo))`

## 4.2. InvRep y Abs

1. Para cada nodo del arbol, cada uno de sus hijos que apunta a otro nodo no nulo, apunta a un nodo diferente de los apuntados por sus hermanos
2. A donde apunta el significado de cada nodo es distinto de a donde apunta el significado del resto de los nodos, con la excepcion que el significado apunta a "null"
3. No pueden haber ciclos, es decir, que todos los nodos son apuntados por un unico nodo del arbol, con la excepcion de la raiz, este no es apuntado por ninguno de los nodos del arbol
4. Debe existir aunque sea un nodo en el ultimo nivel, tal que su significado no apunta a "null"

$Abs : e\_nodo \ d \longrightarrow diccString \quad \{Rep(d)\}$   
 $Abs(d) =_{obs} n : diccString \mid$   
 $(\forall n : e\_nodo) \ Abs(n) =_{obs} d : diccString \mid (\forall s : string) \ (def?(s, d) \Rightarrow_L ((obtenerDelArbol(s, n) \neq NULL \wedge_L *(obtenerDelArbol(s, n) = obtener(s, d)))) \wedge_L$

$obtenerDelArbol : strings \times e\_nodo \longrightarrow puntero(\alpha)$

$obtenerDelArbol(s, n) \equiv$  **if** Vacía?(s) **then**  
 $\quad n.significado$   
**else**  
 $\quad$  **if** n.hijos[ord(prim(s)) = NULL **then**  
 $\quad \quad NULL$   
 $\quad$  **else**  
 $\quad \quad obtenerDelArbol(fin(s), n.hijos[ord(prim(s))])$   
 $\quad$  **fi**  
**fi**

## 4.3. Algoritmos

### Algoritmos

$iVACÍO() \rightarrow res : diccString(\alpha)$

1:  $res \leftarrow iNodoVacío()$

$\mathcal{O}(1)$

**Complejidad:**  $\mathcal{O}(1)$

$iNODOVACÍO() \rightarrow res : e\_nodo$

1:  $res \leftarrow tupla(definición : NULL, hijos : arreglo[256] \text{ de } puntero(e\_nodo))$

$\mathcal{O}(1)$

2: **for** var  $i : nat \leftarrow 0$  to 255 **do**

$\mathcal{O}(1)$

3:  $res.hijos[i] \leftarrow NULL;$

$\mathcal{O}(1)$

4: **end for**

**Complejidad:**  $\mathcal{O}(1)$

$\mathcal{O}(1) + \sum_{i=1}^{255} * \mathcal{O}(1) =$

$\mathcal{O}(1) + 255 * \mathcal{O}(1) =$

$256 * \mathcal{O}(1) = \mathcal{O}(1)$

$iDEFINIR(in/out \ d : diccString(\alpha), in \ clv : string, in \ def : \alpha)$

1: var  $actual : puntero(e\_nodo) \leftarrow \&(d)$

$\mathcal{O}(1)$

2: **for** var  $i : nat \leftarrow 0$  to  $LONGITUD(clv)$  **do**

$\mathcal{O}(1)$

3: **if**  $actual \rightarrow hijos[ord(clv[i])] =_{obs} NULL$  **then**

$\mathcal{O}(1)$

4:  $actual \rightarrow (hijos[ord(clv[i])] \leftarrow \&(iNodoVacío()))$

$\mathcal{O}(1)$

5: <b>end if</b>	$\mathcal{O}(1)$
6: $actual \leftarrow (actual \rightarrow hijos[ord(clv[i])])$	$\mathcal{O}(1)$
7: <b>end for</b>	
8: $(actual \rightarrow definicion) \leftarrow \&(Copiar(def))$	$\mathcal{O}(1)$

**Complejidad:**  $|clv|$

$$\begin{aligned}
&\mathcal{O}(1) + \sum_{i=1}^{|clv|} \max(\sum_{i=1}^2 \mathcal{O}(1), \sum_{i=1}^3 \mathcal{O}(1)) + \mathcal{O}(1) = \\
&2 * \mathcal{O}(1) + |clv| * \max(2 * \mathcal{O}(1), 3 * \mathcal{O}(1)) = \\
&2 * \mathcal{O}(1) + |clv| * 3 * \mathcal{O}(1) = \\
&2 * \mathcal{O}(1) + 3 * \mathcal{O}(|clv|) = \\
&3 * \mathcal{O}(|clv|) = \mathcal{O}(|clv|)
\end{aligned}$$

<b>IDEFINIDO?</b> ( <b>in</b> $d: diccString(\alpha)$ , <b>in</b> $def: \alpha \rightarrow res: bool$ )	
1: var $actual: puntero(e\_nodo) \leftarrow \&(d)$	$\mathcal{O}(1)$
2: var $i: nat \leftarrow 0$	$\mathcal{O}(1)$
3: $res \leftarrow true$	$\mathcal{O}(1)$
4: <b>while</b> $i < LONGITUD(clv) \wedge res =_{obs} true$ <b>do</b>	$\mathcal{O}(1)$
5: <b>if</b> $actual \rightarrow hijos[ord(clv[i])] =_{obs} NULL$ <b>then</b>	$\mathcal{O}(1)$
6: $res \leftarrow false$	$\mathcal{O}(1)$
7: <b>else</b> $actual \leftarrow (actual \rightarrow hijos[ord(clv[i])])$	$\mathcal{O}(1)$
8: <b>end if</b>	
9: <b>end while</b>	
10: <b>if</b> $actual \rightarrow definicion =_{obs} NULL$ <b>then</b>	$\mathcal{O}(1)$
11: $res \leftarrow false$	$\mathcal{O}(1)$
12: <b>end if</b>	

**Complejidad:**  $|clv|$

$$\begin{aligned}
&\mathcal{O}(1) + \mathcal{O}(1) + \mathcal{O}(1) + \sum_{i=1}^{|clv|} (\mathcal{O}(1) + \max(\mathcal{O}(1), \mathcal{O}(1))) + \mathcal{O}(1) + \max(\mathcal{O}(1), 0) = \\
&4 * \mathcal{O}(1) + \sum_{i=1}^{|clv|} (\mathcal{O}(1) + \mathcal{O}(1)) + \mathcal{O}(1) = \\
&5 * \mathcal{O}(1) + |clv| * 2 * \mathcal{O}(1) = \\
&5 * \mathcal{O}(1) + 2 * \mathcal{O}(|clv|) = \\
&2 * \mathcal{O}(|clv|) = \mathcal{O}(|clv|)
\end{aligned}$$

<b>ISIGNIFICADO</b> ( <b>in</b> $d: diccString(\alpha)$ , <b>in</b> $clv: string$ ) $\rightarrow res: diccString(\alpha)$	
1: var $actual: puntero(e\_nodo) \leftarrow \&(d)$	$\mathcal{O}(1)$
2: <b>for</b> var $i: nat \leftarrow 0$ to $LONGITUD(clv)$ <b>do</b>	$\mathcal{O}(1)$
3: $actual \leftarrow (actual \rightarrow hijos[ord(clv[i])])$	$\mathcal{O}(1)$
4: <b>end for</b>	
5: $res \leftarrow (actual \rightarrow definicion)$	$\mathcal{O}(1)$

**Complejidad:**  $|clv|$

$$\begin{aligned}
&\mathcal{O}(1) + \mathcal{O}(1) + \sum_{i=1}^{|clv|} \mathcal{O}(1) + \mathcal{O}(1) = \\
&3 * \mathcal{O}(1) + |clv| * \mathcal{O}(1) = \\
&3 * \mathcal{O}(1) + \mathcal{O}(|clv|) = \mathcal{O}(|clv|)
\end{aligned}$$

## 5. Cola Prioritaria

### 5.1. TAD COLAPRIORITARIA

**TAD COLAPRIORITARIA**

**igualdad observacional**

$$(\forall c_1, c_2 : \text{colaP}(\text{Paquete})) \left( c_1 =_{\text{obs}} c_2 \iff \left( \begin{array}{l} \text{vacía?}(c_1) =_{\text{obs}} \text{vacía?}(c_2) \wedge_L \\ (\neg \text{vacía}(c_1) \Rightarrow_L \\ (\text{próximo}(c_1) =_{\text{obs}} \text{próximo}(c_2) \wedge \\ \text{suCamino}(\text{próximo}(c_1)) \\ \text{suCamino}(\text{próximo}(c_2)) \wedge \\ \text{desencolar}(c_1) =_{\text{obs}} \text{desencolar}(c_2))) \end{array} \right) =_{\text{obs}} \right)$$

**géneros** colaP(Paquete)

**exporta** colaP(Paquete), generadores, observadores

**usa** BOOL, NAT, PAQUETE, SECU( $\alpha$ ), COMPU

**observadores básicos**

vacía?	: colaP(Paquete)	→ Bool	
próximo	: colaP(Paquete) cp	→ Paquete	{¬vacía?(cp)}
desencolar	: colaP(Paquete) cp	→ colaP(Paquete)	{¬vacía?(cp)}
suCamino	: Paquete p × colaP(Paquete) cp	→ Secu(Compu)	{está?(p, cp)}

**generadores**

vacía	:	→ colaP(Paquete)	
encolar	: Paquete p × colaP(Paquete) cp	→ colaP(Paquete)	{¬está?(p, cp)}
agCompu	Paquete p × Compu c × colaP(Paquete) cp	→ colaP(Paquete)	{está?(p, cp)}

**otras operaciones**

está?	: Paquete p × colaP(Paquete) cp	→ Bool
-------	---------------------------------	--------

**axiomas**  $\forall p, p_1, p_2$ : Paquete,  $\forall c$ : Compu,  $\forall cp$ : colaP(Paquete)

vacía?(vacía)	≡ true
vacía?(encolar(p, cp))	≡ false
vacía?(agCompu(p, c, cp))	≡ false
próximo(encolar(p, cp))	≡ <b>if</b> vacía?(cp) <b>then</b> p <b>else</b> if prioridad(p) < prioridad(próximo(cp)) <b>then</b> p <b>else</b> próximo(cp) <b>fi</b> <b>fi</b>
próximo(agCompu(p, c, cp))	≡ próximo(cp)
desencolar(encolar(p, cp))	≡ <b>if</b> vacía?(cp) <b>then</b> cp <b>else</b> if prioridad(p) < prioridad(próximo(cp)) <b>then</b> cp <b>else</b> encolar(p, desencolar(cp)) <b>fi</b> <b>fi</b>
desencolar(agCompu(p, c, cp))	≡ desencolar(cp)
suCamino(p <sub>1</sub> , encolar(p <sub>2</sub> , cp))	≡ <b>if</b> p <sub>1</sub> = p <sub>2</sub> <b>then</b> <> <b>else</b> suCamino(p <sub>1</sub> , cp) <b>fi</b>
suCamino(p <sub>1</sub> , agCompu(p <sub>2</sub> , c, cp))	≡ <b>if</b> p <sub>1</sub> = p <sub>2</sub> <b>then</b> suCamino(p <sub>1</sub> , cp) o c <b>else</b> suCamino(p <sub>1</sub> , cp) <b>fi</b>



```

está(p, cp)      ≡  if vacía?(cp) then
                    false
                  else
                    if p = próximo(cp) then true else está?(p, desencolar(cp)) fi
                  fi

```

**Fin TAD**

# Interfaz

se explica con: COLAPRIORITARIA.

gêneros: colaP(Paquete).

## Operaciones básicas de COLA PRIORITARIA

$$\text{VACÍA?}(\text{in } cp : \text{colaP}(\text{Paquete})) \rightarrow res : \text{Bool}$$
$$\mathbf{Pre} \equiv \{\text{true}\}$$
$$\mathbf{Post} \equiv \{res =_{\text{obs}} \text{vacía?}(cp)\}$$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Verifica si una cola esta vacía

$$\text{PRÓXIMO}(\text{in } cp : \text{colaP}(\text{Paquete})) \rightarrow res : \text{Paquete}$$
$$\mathbf{Pre} \equiv \{\neg \text{vacía?}(cp)\}$$
$$\mathbf{Post} \equiv \{res =_{\text{obs}} \text{pr\u00f3ximo}(cp)\}$$

**Complejidad:**  $\mathcal{O}(\log_2 k)$

**Descripción:** Devuelve el próximo paquete a desencolar

DESENCOLAR(in/out *cp*: colaP(Paquete))
$$\mathbf{Pre} \equiv \{cp =_{\text{obs}} cp_0 \wedge \neg(\text{vacia?}(cp_0))\}$$
$$\mathbf{Post} \equiv \{cp =_{\text{obs}} \text{desencolar}(cp_0)\}$$

**Complejidad:**  $\mathcal{O}(\log_2 k)$

**Descripción:** Elimina el próximo paquete a desencolar

$$\text{SUCAMINO}(\text{in } p: \text{Paquete}, \text{in } cp: \text{colaP}(\text{Paquete})) \rightarrow res : \text{Secu}(\text{Compu})$$
$$\mathbf{Pre} \equiv \{\text{está}(p, cp)\}$$
$$\mathbf{Post} \equiv \{res =_{\text{obs}} \text{suCamino}(p, cp)\}$$

**Complejidad:**  $\mathcal{O}(\log_2 k)$

**Descripción:** Devuelve la secuencia de computadoras asociadas al paquete

$$\text{VACÍA}() \rightarrow res : \text{colaP}(\text{Paquete})$$
$$\mathbf{Pre} \equiv \{\text{true}\}$$
$$\text{Post} \equiv \{res =_{\text{obs}} \text{vacía}()\}$$

**Complejidad:**  $\mathcal{O}(1)$

**Descripción:** Crea una nueva cola.

ENCOLAR(in *p*: Paquete, in/out *cp*: colaP(Paquete))
$$\mathbf{Pre} \equiv \{cp =_{\text{obs}} cp_0 \wedge \neg \text{está}(p, cp_0)\}$$
$$\mathbf{Post} \equiv \{cp =_{\text{obs}} \text{encolar}(p, cp_0)\}$$

**Complejidad:**  $\mathcal{O}(\log_2 k)$

**Descripción:** Agrega el paquete a la cola

$$\text{AGCOMPU}(\text{in } p: \text{Paquete}, \text{in } c: \text{Compu}, \text{in/out } cp: \text{colaP}(\text{Paquete}))$$
$$\mathbf{Pre} \equiv \{cp =_{\text{obs}} cp_0 \wedge \text{está}(p, cp_0)\}$$
$$\text{Post} \equiv \{cp =_{\text{obs}} \text{agCompu}(p, c, cp_0)\}$$

**Complejidad:**  $\mathcal{O}(\log_2 k)$

**Descripción:** Agrega la computadora al paquete

## 5.2. Representacion

### Representación

Para representar la cola de prioridad, elegimos hacerla sobre un AVL. Sabiendo que la cantidad de paquetes no está acotada, este AVL estará representado con nodos y punteros.

**colaP(Paquete) se representa con estr**

donde **estr** es **tupla**(*raiz*: puntero(nodo), *tam*: nat)

donde **nodo** es **tupla**(*paquete*: Paquete, *caminoRecorrido*: Secu(Compu), *padre*: puntero(nodo), *hijoIzq*: puntero(nodo), *hijoDer*: puntero(nodo))

## 5.3. InvRep y Abs

**InvRep en lenguaje coloquial:**

1. La componente "tam" de *e\_cola* es igual a la cantidad de nodos en el arbol.
2. Todo nodo en el arbol tiene un unico padre, con excepcion de la raiz, que no tiene padre.
3. La relacion de orden es total.
4. Un nodo es mayor a otro si la componente "pri" del primero es mayor que la del segundo.
5. Un nodo es menor a otro si la componente "pri" del primero es menor que la del segundo.
6. No pueden haber dos nodos en el arbol que tengan el mismo numero en la componente "seg".
7. Si dos nodos tienen el mismo numero en la componente "pri", se procede a verificar la componente "seg" de ambos. El que tiene el mayor numero en dicha componente es el mayor, mientras que el otro es el menor.
8. Para cada nodo, todos los elementos del subarbol que se encuentra a la derecha de la raiz son mayores que la misma.
9. Para cada nodo, todos los elementos del subarbol que se encuentra a la izquierda de la raiz son menores que la misma.
10. La componente "alt" de cada nodo es igual a la cantidad de niveles que hay que recorrer para llegar a la hoja mas lejana.
11. Para cada nodo, la diferencia en modulo de la altura entre los dos subarboles del mismo no puede diferir en mas de 1.

**Abs:**

**Abs** : **colaP**(Paquete) *c*  $\longrightarrow$  **colaP**(Paquete) {Rep(*c*)}

**Abs**(*c*) =<sub>obs</sub> *p*: **colaP**(Paquete) | **mismosProximos**(*c*, *p*)

**mismosProximos** :  $\langle \text{puntero}(\text{nodo}) \times \text{nat} \rangle \longrightarrow \text{bool}$

**mismosProximos**(*c*, *p*)  $\equiv$  **if**  $\pi_1(c) = \text{NULL} \wedge \text{vacía?}(p)$  **then**  
    true  
**else**  
    **if**  $(\pi_1(c) = \text{NULL} \wedge \neg \text{vacía?}(p)) \vee (\pi_1(c) \neq \text{NULL} \wedge \text{vacía?}(p))$  **then**  
        false  
    **else**  
        **if**  $\text{maxElem}(*(\pi_1(c))) = \text{proximo}(p)$  **then**  
            **mismosProximos**(**borrarMax**( $*(\pi_1(c))$ ), **borrar**( $\pi_1(\text{proximo}(p))$ ),  $\pi_2(\text{proximo}(p))$ , *p*)  
        **else**  
            false  
        **fi**  
    **fi**  
**fi**

Las funciones "maxElem" y "borrarMax" no han sido axiomatizadas. Ya que estamos trabajando con Árboles Binarios de Búsqueda (en nuestro caso AVL), la lógica de ambas funciones es la misma que está expresada en el pseudocódigo del módulo. En particular "maxElem" se limita a buscar el nodo más a la derecha del árbol, mientras que "borrarMax" (una vez encontrado el máximo) procede a eliminarlo y reordenar el árbol.

## 5.4. Algoritmos

### Algoritmos

IVACÍA?(in cp: colaP(Paquete)) → res : Bool

1: res ← cp.raiz == NULL

$\mathcal{O}(1)$

**Complejidad:**  $\mathcal{O}(1)$

IPRÓXIMO(in cp: colaP(Paquete)) → res : Paquete

1: var pNodo: puntero(nodo) ← cp.raiz

$\mathcal{O}(1)$

2: **while** pNodo.der != NULL **do**

$\mathcal{O}(\log_2 k)$

3:   pNodo ← pNodo.der

$\mathcal{O}(1)$

4: **end while**

5: res ← pNodo.paquete

$\mathcal{O}(1)$

**Complejidad:**  $\mathcal{O}(\log_2 k)$

$\mathcal{O}(1) + \mathcal{O}(\log_2 k) * \mathcal{O}(1) + \mathcal{O}(1) =$

$\mathcal{O}(\log_2 k)$

IDESENCOLAR(in/out cp: colaP(Paquete))

1: var pNodo: puntero(nodo) ← cp.raiz

$\mathcal{O}(1)$

2: **while** pNodo.der != NULL **do**

$\mathcal{O}(\log_2 k)$

3:   pNodo ← pNodo.der

$\mathcal{O}(1)$

4: **end while**

5: ELIMINAR(cp, \*(pNodo).pri, \*(pNodo).seg)

$\mathcal{O}(\log_2 k)$

**Complejidad:**  $\mathcal{O}(\log_2 k)$

$\mathcal{O}(1) + \mathcal{O}(\log_2 k) * \mathcal{O}(1) + \mathcal{O}(\log_2 k) =$

$2 * \mathcal{O}(\log_2 k) = \mathcal{O}(\log_2 k)$

ISUCAMINO(in p: Paquete, in cp: colaP(Paquete)) → res : Secu(Compu)

1:

**Complejidad:**  $\mathcal{O}()$

IVACIA() → res : colaP(Paquete)

1: var res: colaP(Paquete) ← tupla(NULL, 0)

$\mathcal{O}(1)$

**Complejidad:**  $\mathcal{O}(1)$

IAGREGAR(in/out c: colaP(Paquete), in p: Paquete)

1: **if** c.raiz == NULL **then**

2:   c.raiz ← &(tupla(p, <>, NULL, NULL, NULL))

$\mathcal{O}(1)$

3:   c.tam ← 1

$\mathcal{O}(1)$

4: **else**

5:	var iTam: int $\leftarrow$ c.tam	$\mathcal{O}(1)$
6:	var iAux: int $\leftarrow$ c.tam	$\mathcal{O}(1)$
7:	var aCoordenadas: arreglo[ $\lfloor \log_2(c.tam) \rfloor$ ] de bool	$\mathcal{O}(\lfloor \log_2(c.tam) \rfloor)$
8:	<b>while</b> iTam > 0 <b>do</b>	$\mathcal{O}(1)$
9:	<b>if</b> iAux % 2 == 0 <b>then</b>	$\mathcal{O}(1)$
10:	aCoordenadas[iTam-1] = false	$\mathcal{O}(1)$
11:	<b>else</b>	
12:	aCoordenadas[iTam-1] = true	$\mathcal{O}(1)$
13:	<b>end if</b>	
14:	iAux $\leftarrow$ $\lfloor iAux/2 \rfloor$	$\mathcal{O}(1)$
15:	iTam $\leftarrow$ iTam - 1	
16:	<b>end while</b>	
17:	var seguir: bool $\leftarrow$ true	$\mathcal{O}(1)$
18:	var pNodo: puntero(nodo) $\leftarrow$ c.raiz	$\mathcal{O}(1)$
19:	var camino: arreglo[ $\lfloor \log_2(c.tam) \rfloor + 1$ ] de puntero(nodo)	$\mathcal{O}(\lfloor \log_2(c.tam) \rfloor + 1)$
20:	var nroCamino: nat	
21:	camino[0] $\leftarrow$ pNodo	$\mathcal{O}(1)$
22:	nroCamino $\leftarrow$ 0	$\mathcal{O}(1)$
23:	<b>while</b> seguir == true <b>do</b>	$\mathcal{O}(1)$
24:	<b>if</b> a $\geq$ *(pNodo).pri <b>then</b>	$\mathcal{O}(1)$
25:	<b>if</b> a == *(pNodo).pri <b>then</b>	$\mathcal{O}(1)$
26:	<b>if</b> b > *(pNodo).seg <b>then</b>	$\mathcal{O}(1)$
27:	<b>if</b> *(pNodo).der != NULL <b>then</b>	$\mathcal{O}(1)$
28:	pNodo $\leftarrow$ *(pNodo).der	$\mathcal{O}(1)$
29:	nroCamino $\leftarrow$ nroCamino + 1	$\mathcal{O}(1)$
30:	camino[nroCamino] $\leftarrow$ pNodo	$\mathcal{O}(1)$
31:	<b>else</b>	
32:	*(pNodo).der $\leftarrow$ &(tupla(a, b, pNodo, NULL, NULL, 1))	$\mathcal{O}(1)$
33:	nroCamino $\leftarrow$ nroCamino + 1	$\mathcal{O}(1)$
34:	camino[nroCamino] $\leftarrow$ *(pNodo).der	$\mathcal{O}(1)$
35:	seguir $\leftarrow$ false	$\mathcal{O}(1)$
36:	<b>end if</b>	
37:	<b>else</b>	
38:	<b>if</b> *(pNodo).izq != NULL <b>then</b>	$\mathcal{O}(1)$
39:	pNodo $\leftarrow$ *(pNodo).izq	$\mathcal{O}(1)$
40:	nroCamino $\leftarrow$ nroCamino + 1	$\mathcal{O}(1)$
41:	camino[nroCamino] $\leftarrow$ pNodo	$\mathcal{O}(1)$
42:	<b>else</b>	
43:	*(pNodo).izq $\leftarrow$ &(tupla(a, b, pNodo, NULL, NULL, 1))	$\mathcal{O}(1)$
44:	nroCamino $\leftarrow$ nroCamino + 1	$\mathcal{O}(1)$
45:	camino[nroCamino] $\leftarrow$ *(pNodo).izq	$\mathcal{O}(1)$
46:	seguir $\leftarrow$ false	$\mathcal{O}(1)$
47:	<b>end if</b>	
48:	<b>end if</b>	
49:	<b>else</b>	
50:	<b>if</b> *(pNodo).der != NULL <b>then</b>	$\mathcal{O}(1)$
51:	pNodo $\leftarrow$ *(pNodo).der	$\mathcal{O}(1)$
52:	nroCamino $\leftarrow$ nroCamino + 1	$\mathcal{O}(1)$
53:	camino[nroCamino] $\leftarrow$ pNodo	$\mathcal{O}(1)$
54:	<b>else</b>	
55:	*(pNodo).der $\leftarrow$ &(tupla(a, b, pNodo, NULL, NULL, 1))	$\mathcal{O}(1)$
56:	nroCamino $\leftarrow$ nroCamino + 1	$\mathcal{O}(1)$
57:	camino[nroCamino] $\leftarrow$ *(pNodo).der	$\mathcal{O}(1)$
58:	seguir $\leftarrow$ false	$\mathcal{O}(1)$
59:	<b>end if</b>	
60:	<b>end if</b>	
61:	<b>else</b>	

62:	<b>if</b> $*(pNodo).izq \neq \text{NULL}$ <b>then</b>	$\mathcal{O}(1)$
63:	$pNodo \leftarrow *(pNodo).izq$	$\mathcal{O}(1)$
64:	$nroCamino \leftarrow nroCamino + 1$	$\mathcal{O}(1)$
65:	$camino[nroCamino] \leftarrow pNodo$	$\mathcal{O}(1)$
66:	<b>else</b>	
67:	$*(pNodo).izq \leftarrow \&(tupla(a, b, pNodo, \text{NULL}, \text{NULL}, 1))$	$\mathcal{O}(1)$
68:	$nroCamino \leftarrow nroCamino + 1$	$\mathcal{O}(1)$
69:	$camino[nroCamino] \leftarrow *(pNodo).izq$	$\mathcal{O}(1)$
70:	$seguir \leftarrow \text{false}$	$\mathcal{O}(1)$
71:	<b>end if</b>	
72:	<b>end if</b>	
73:	<b>end while</b>	
74:	$c.tam \leftarrow c.tam + 1$	$\mathcal{O}(1)$
75:	$seguir \leftarrow \text{true}$	$\mathcal{O}(1)$
76:	<b>while</b> $nroCamino \geq 0 \wedge seguir == \text{true}$ <b>do</b>	$\mathcal{O}(\lfloor \log_2 N \rfloor + 1)$
77:	$pNodo \leftarrow camino[nroCamino]$	$\mathcal{O}(1)$
78:	$*(pNodo).alt \leftarrow \text{ALTURA}(pNodo) \vee$	
79:	<b>if</b> $ \text{FACTORDESBALANCE}(camino[nroCamino])  > 1$ <b>then</b>	$\mathcal{O}(1)$
80:	$pNodo \leftarrow \text{ROTAR}(\text{HIJOMASALTO}(\text{HIJOMASALTO}(pNodo)), \text{HijoMasAlto}(pNodo), pNodo)$	$\mathcal{O}(1)$
81:	$*(pNodo).izq.alt \leftarrow \text{ALTURA}(*(pNodo).izq)$	$\mathcal{O}(1)$
82:	$*(pNodo).der.alt \leftarrow \text{ALTURA}(*(pNodo).der)$	$\mathcal{O}(1)$
83:	$*(pNodo).alt \leftarrow \text{ALTURA}(*(pNodo))$	$\mathcal{O}(1)$
84:	$seguir \leftarrow \text{false}$	$\mathcal{O}(1)$
85:	<b>end if</b>	
86:	$nroCamino \leftarrow nroCamino - 1$	$\mathcal{O}(1)$
87:	<b>end while</b>	
88:	<b>end if</b>	

**Complejidad:**  $\mathcal{O}(1)$

Debido a la longitud del pseudocodigo, vamos a ignorar todas los condicionales y las asignaciones en la justificacion, ya que se realizan en tiempo constante. Solo nos vamos a centrar en dos puntos, la creacion del arreglo "camino" y el ultimo ciclo.

Para el arreglo asignamos esa cantidad de nodos ya que contamos con un Arbol balanceado, el cual como mucho puede necesitar de  $\lfloor \log_2 N \rfloor + 1$  niveles para almacenar  $N$  nodos. Esta misma logica la utilizamos en el ultimo ciclo, en el cual para restaurar el balance del Arbol recorremos el mismo desde el ultimo nodo agregado (el cual es una hoja) hasta la raiz en el peor caso, corrigiendo cualquier desbalance en el camino.

Esto resulta en la siguiente suma:

$$\mathcal{O}(\lfloor \log_2 N \rfloor + 1) + \mathcal{O}(\lfloor \log_2 N \rfloor + 1) =$$

$$2 * \mathcal{O}(\lfloor \log_2 N \rfloor + 1) =$$

$$\mathcal{O}(\lfloor \log_2 N \rfloor + 1) =$$

$$\mathcal{O}(\lfloor \log_2 N \rfloor) = \mathcal{O}(\log_2 N)$$

**IELIMINAR**(**in/out**  $c: \text{colaP}(\text{Paquete})$ , **in**  $a: \text{nat}$ , **in**  $b: \text{nat}$ )

1:	<b>var</b> $pNodo: \text{puntero}(\text{nodo}) \leftarrow c.raiz$	$\mathcal{O}(1)$
2:	<b>ver</b> $seguir: \text{bool} \leftarrow \text{true}$	$\mathcal{O}(1)$
3:	<b>while</b> $pNodo \neq \text{NULL} \wedge seguir == \text{true}$ <b>do</b>	$\mathcal{O}(\lfloor \log_2 N \rfloor + 1)$
4:	<b>if</b> $*(pNodo).pri == a \wedge *(pNodo).seg == b$ <b>then</b>	$\mathcal{O}(1)$
5:	$seguir \leftarrow \text{false}$	$\mathcal{O}(1)$
6:	<b>else</b>	
7:	<b>if</b> $a \geq *(pNodo).pri$ <b>then</b>	$\mathcal{O}(1)$
8:	<b>if</b> $a == *(pNodo).pri$ <b>then</b>	$\mathcal{O}(1)$
9:	<b>if</b> $b > *(pNodo).seg$ <b>then</b>	$\mathcal{O}(1)$
10:	$pNodo \leftarrow *(pNodo).der$	$\mathcal{O}(1)$
11:	<b>else</b>	
12:	$pNodo \leftarrow *(pNodo).izq$	$\mathcal{O}(1)$
13:	<b>end if</b>	

14:	<b>else</b>	
15:	pNodo $\leftarrow$ *(pNodo).der	$\mathcal{O}(1)$
16:	<b>end if</b>	
17:	<b>else</b>	
18:	pNodo $\leftarrow$ *(pNodo).izq	$\mathcal{O}(1)$
19:	<b>end if</b>	
20:	<b>end if</b>	
21:	<b>end while</b>	
22:	<b>if</b> pNodo != NULL <b>then</b>	$\mathcal{O}(1)$
23:	bNodo: puntero(nodo) $\leftarrow$ NULL	$\mathcal{O}(1)$
24:	<b>if</b> pNodo == c.raiz <b>then</b>	$\mathcal{O}(1)$
25:	<b>if</b> *(pNodo).izq == NULL $\wedge$ *(pNodo).der == NULL <b>then</b>	$\mathcal{O}(1)$
26:	c.raiz $\leftarrow$ NULL	$\mathcal{O}(1)$
27:	delete pNodo	$\mathcal{O}(1)$
28:	<b>end if</b>	
29:	<b>if</b> *(pNodo).izq != NULL $\wedge$ *(pNodo).der == NULL <b>then</b>	$\mathcal{O}(1)$
30:	c.raiz $\leftarrow$ *(pNodo).izq	$\mathcal{O}(1)$
31:	*(*(pNodo).izq).padre $\leftarrow$ NULL	$\mathcal{O}(1)$
32:	delete pNodo	$\mathcal{O}(1)$
33:	<b>end if</b>	
34:	<b>if</b> *(pNodo).izq == NULL $\wedge$ *(pNodo).der != NULL <b>then</b>	$\mathcal{O}(1)$
35:	c.raiz $\leftarrow$ *(pNodo).der	$\mathcal{O}(1)$
36:	*(*(pNodo).der).padre $\leftarrow$ NULL	$\mathcal{O}(1)$
37:	delete pNodo	$\mathcal{O}(1)$
38:	<b>end if</b>	
39:	<b>if</b> *(pNodo).izq != NULL $\wedge$ *(pNodo).der != NULL <b>then</b>	$\mathcal{O}(1)$
40:	var tNodo: puntero(nodo) $\leftarrow$ pNodo.der	$\mathcal{O}(1)$
41:	<b>while</b> *(tNodo).izq != NULL <b>do</b>	$\mathcal{O}(\lfloor \log_2 N \rfloor + 1)$
42:	tNodo $\leftarrow$ tNodo.izq	$\mathcal{O}(1)$
43:	<b>end while</b>	
44:	bNodo $\leftarrow$ *(tNodo).padre	$\mathcal{O}(1)$
45:	<b>if</b> *(tNodo).der != NULL <b>then</b>	$\mathcal{O}(1)$
46:	*(*(tNodo).der).padre $\leftarrow$ *(tNodo).padre	$\mathcal{O}(1)$
47:	<b>end if</b>	
48:	<b>if</b> *(*(tNodo).padre).izq == tNodo <b>then</b>	$\mathcal{O}(1)$
49:	*(*(tNodo).padre).izq $\leftarrow$ *(tNodo).der	$\mathcal{O}(1)$
50:	<b>else</b>	
51:	*(*(tNodo).padre).der $\leftarrow$ *(tNodo).der	$\mathcal{O}(1)$
52:	<b>end if</b>	
53:	*(tNodo).padre $\leftarrow$ *(pNodo).padre	$\mathcal{O}(1)$
54:	*(tNodo).izq $\leftarrow$ *(pNodo).izq	$\mathcal{O}(1)$
55:	*(tNodo).der $\leftarrow$ *(pNodo).der	$\mathcal{O}(1)$
56:	*(*(pNodo).izq).padre $\leftarrow$ tNodo	$\mathcal{O}(1)$
57:	*(*(pNodo).der).padre $\leftarrow$ tNodo	$\mathcal{O}(1)$
58:	delete pNodo	$\mathcal{O}(1)$
59:	<b>end if</b>	
60:	<b>else</b>	
61:	<b>if</b> *(pNodo).izq == NULL $\wedge$ *(pNodo).der == NULL <b>then</b>	$\mathcal{O}(1)$
62:	<b>if</b> *(*(pNodo).padre).izq == pNodo <b>then</b>	$\mathcal{O}(1)$
63:	*(*(pNodo).padre).izq $\leftarrow$ NULL	$\mathcal{O}(1)$
64:	bNodo $\leftarrow$ *(pNodo).padre	$\mathcal{O}(1)$
65:	delete pNodo	$\mathcal{O}(1)$
66:	<b>else</b>	
67:	*(*(pNodo).padre).der $\leftarrow$ NULL	$\mathcal{O}(1)$
68:	bNodo $\leftarrow$ *(pNodo).padre	$\mathcal{O}(1)$
69:	delete pNodo	$\mathcal{O}(1)$
70:	<b>end if</b>	

71:	<b>end if</b>	
72:	<b>if</b> $*(pNodo).izq \neq \text{NULL} \wedge *(pNodo).der == \text{NULL}$ <b>then</b>	$\mathcal{O}(1)$
73:	<b>if</b> $*(*(pNodo).padre).izq == pNodo$ <b>then</b>	$\mathcal{O}(1)$
74:	$*(*(pNodo).padre).izq \leftarrow *(pNodo).izq$	$\mathcal{O}(1)$
75:	$*(*(pNodo).izq).padre \leftarrow *(pNodo).padre$	$\mathcal{O}(1)$
76:	$bNodo \leftarrow *(pNodo).padre$	$\mathcal{O}(1)$
77:	delete pNodo	$\mathcal{O}(1)$
78:	<b>else</b>	
79:	$*(*(pNodo).padre).der \leftarrow *(pNodo).izq$	$\mathcal{O}(1)$
80:	$*(*(pNodo).izq).padre \leftarrow *(pNodo).padre$	$\mathcal{O}(1)$
81:	$bNodo \leftarrow *(pNodo).padre$	$\mathcal{O}(1)$
82:	delete pNodo	$\mathcal{O}(1)$
83:	<b>end if</b>	
84:	<b>end if</b>	
85:	<b>if</b> $*(pNodo).izq == \text{NULL} \wedge *(pNodo).der \neq \text{NULL}$ <b>then</b>	$\mathcal{O}(1)$
86:	<b>if</b> $*(*(pNodo).padre).izq == pNodo$ <b>then</b>	$\mathcal{O}(1)$
87:	$*(*(pNodo).padre).izq \leftarrow *(pNodo).der$	$\mathcal{O}(1)$
88:	$*(*(pNodo).der).padre \leftarrow *(pNodo).padre$	$\mathcal{O}(1)$
89:	$bNodo \leftarrow *(pNodo).padre$	$\mathcal{O}(1)$
90:	delete pNodo	$\mathcal{O}(1)$
91:	<b>else</b>	
92:	$*(*(pNodo).padre).der \leftarrow *(pNodo).der$	$\mathcal{O}(1)$
93:	$*(*(pNodo).der).padre \leftarrow *(pNodo).padre$	$\mathcal{O}(1)$
94:	$bNodo \leftarrow *(pNodo).padre$	$\mathcal{O}(1)$
95:	delete pNodo	$\mathcal{O}(1)$
96:	<b>end if</b>	
97:	<b>end if</b>	
98:	<b>if</b> $*(pNodo).izq \neq \text{NULL} \wedge *(pNodo).der \neq \text{NULL}$ <b>then</b>	$\mathcal{O}(1)$
99:	var tNodo: puntero(nodo) $\leftarrow pNodo.der$	$\mathcal{O}(1)$
100:	<b>while</b> $*(tNodo).izq \neq \text{NULL}$ <b>do</b>	$\mathcal{O}(\lfloor \log_2 N \rfloor + 1)$
101:	$tNodo \leftarrow tNodo.izq$	$\mathcal{O}(1)$
102:	<b>end while</b>	
103:	$bNodo \leftarrow *(tNodo).padre$	$\mathcal{O}(1)$
104:	<b>if</b> $*(tNodo).der \neq \text{NULL}$ <b>then</b>	$\mathcal{O}(1)$
105:	$*(*(tNodo).der).padre \leftarrow *(tNodo).padre$	$\mathcal{O}(1)$
106:	<b>end if</b>	
107:	<b>if</b> $*(*(tNodo).padre).izq == tNodo$ <b>then</b>	$\mathcal{O}(1)$
108:	$*(*(tNodo).padre).izq \leftarrow *(tNodo).der$	$\mathcal{O}(1)$
109:	<b>else</b>	
110:	$*(*(tNodo).padre).der \leftarrow *(tNodo).der$	$\mathcal{O}(1)$
111:	<b>end if</b>	
112:	<b>if</b> $*(*(pNodo).padre).izq == pNodo$ <b>then</b>	$\mathcal{O}(1)$
113:	$*(*(tNodo).padre).izq \leftarrow tNodo$	$\mathcal{O}(1)$
114:	<b>else</b>	
115:	$*(*(tNodo).padre).der \leftarrow pNodo$	$\mathcal{O}(1)$
116:	<b>end if</b>	
117:	$*(tNodo).padre \leftarrow *(pNodo).padre$	$\mathcal{O}(1)$
118:	$*(tNodo).izq \leftarrow *(pNodo).izq$	$\mathcal{O}(1)$
119:	$*(tNodo).der \leftarrow *(pNodo).der$	$\mathcal{O}(1)$
120:	$*(*(pNodo).izq).padre \leftarrow tNodo$	$\mathcal{O}(1)$
121:	$*(*(pNodo).der).padre \leftarrow tNodo$	$\mathcal{O}(1)$
122:	delete pNodo	$\mathcal{O}(1)$
123:	<b>end if</b>	
124:	c.tam $\leftarrow$ c.tam + 1	
125:	<b>while</b> b $\neq \text{NULL}$ <b>do</b>	$\mathcal{O}(\lfloor \log_2 N \rfloor + 1)$
126:	$*(b).alt \leftarrow \text{SETALTURA}(b)$	$\mathcal{O}(1)$
127:	<b>if</b> $ \text{FACTORDEDESBALANCE}(b)  > 1$ <b>then</b>	$\mathcal{O}(1)$

128:	*(b).alt ← ROTAR(HIJOMASALTO (HIJOMASALTO(b)), HIJOMASALTO(b), b)	$\mathcal{O}(1)$
129:	*(b).izq.alt ← SETALTURA(*(b).izq)	$\mathcal{O}(1)$
130:	*(b).der.alt ← SETALTURA(*(b).der)	$\mathcal{O}(1)$
131:	*(b).alt ← SETALTURA(*(b))	$\mathcal{O}(1)$
132:	<b>end if</b>	
133:	b ← *(b).padre	$\mathcal{O}(1)$
134:	<b>end while</b>	
135:	<b>end if</b>	
136:	<b>end if</b>	

**Complejidad:**  $\mathcal{O}(\log_2 N)$

Para la complejidad de este algoritmo nos vamos a remitir al mismo proceso que en el caso anterior, vamos a ignorar los condicionales y las asignaciones ya que estas se realizan en tiempo constante para centrarnos unicamente en los ciclos cuya complejidad depende de algun parametro.

En este algoritmo contamos con 4 ciclos que dependen de alguna variable, ninguno esta anidado con ningun otro ciclo, y en el peor caso solo recorreremos 3 de ellos.

Esto nos da la siguiente suma:

$$3 * \mathcal{O}(\lfloor \log_2 N \rfloor + 1) =$$

$$\mathcal{O}(\lfloor \log_2 N \rfloor + 1) =$$

$$\mathcal{O}(\lfloor \log_2 N \rfloor) = \mathcal{O}(\log_2 N)$$

<b>IROTAR(in/out c: colaP(Paquete), in p1: puntero(nodo), in p2: puntero(nodo), in p3: puntero(nodo)) →</b>		
<b>res : puntero(nodo)</b>		
1:	var t1: puntero(nodo) ← NULL	$\mathcal{O}(1)$
2:	var t2: puntero(nodo) ← NULL	$\mathcal{O}(1)$
3:	var t2: puntero(nodo) ← NULL v	
4:	<b>if</b> (*(p3).pri ≤ *(p1).pri ∧ *(p3).seg < *(p1).seg) ∧	
5:	(*(p1).pri ≤ *(p2).pri ∧ *(p1).pri < *(p2).pri) <b>then</b>	$\mathcal{O}(1)$
6:	t1 ← p3	$\mathcal{O}(1)$
7:	t2 ← p1	$\mathcal{O}(1)$
8:	t3 ← p2	$\mathcal{O}(1)$
9:	<b>end if</b>	
10:	<b>if</b> (*(p3).pri ≥ *(p1).pri ∧ *(p3).seg > *(p1).seg) ∧	
11:	(*(p1).pri ≥ *(p2).pri ∧ *(p1).pri > *(p2).pri) <b>then</b>	$\mathcal{O}(1)$
12:	t1 ← p2	$\mathcal{O}(1)$
13:	t2 ← p1	$\mathcal{O}(1)$
14:	t3 ← p3	$\mathcal{O}(1)$
15:	<b>end if</b>	
16:	<b>if</b> (*(p3).pri ≤ *(p2).pri ∧ *(p3).seg < *(p2).seg) ∧	
17:	(*(p2).pri ≥ *(p1).pri ∧ *(p2).pri < *(p1).pri) <b>then</b>	$\mathcal{O}(1)$
18:	t1 ← p3	$\mathcal{O}(1)$
19:	t2 ← p2	$\mathcal{O}(1)$
20:	t3 ← p1	$\mathcal{O}(1)$
21:	<b>end if</b>	
22:	<b>if</b> (*(p3).pri ≥ *(p2).pri ∧ *(p3).seg > *(p2).seg) ∧	
23:	(*(p2).pri ≥ *(p3).pri ∧ *(p2).pri > *(p3).pri) <b>then</b>	$\mathcal{O}(1)$
24:	t1 ← p1	$\mathcal{O}(1)$
25:	t2 ← p2	$\mathcal{O}(1)$
26:	t3 ← p3	$\mathcal{O}(1)$
27:	<b>end if</b>	
28:	<b>if</b> c.raiz == p3 <b>then</b>	
29:	c.raiz ← p3	$\mathcal{O}(1)$
30:	*(p3).padre ← NULL	$\mathcal{O}(1)$
31:	<b>else</b>	
32:	<b>if</b> (*(p3).padre).izq = p3 <b>then</b>	$\mathcal{O}(1)$
33:	CIZQ(*(p3).padre, t2)	$\mathcal{O}(1)$



<pre> 34:   else 35:       CDER(*(p3).padre, t2) 36:   end if 37: end if 38: if *(t2).izq != p1 ∧ *(t2).izq != p2 ∧ *(t2).izq != p3 then 39:     CDER(t1, *(t2).izq) 40: end if 41: if *(t2).der != p1 ∧ *(t2).der != p2 ∧ *(t2).der != p3 then 42:     CDER(t3, *(t2).der) 43: end if 44: CIZQ(t2, t1) 45: CDER(t2, t3) 46: res ← t2 </pre>	$\mathcal{O}(1)$          $\mathcal{O}(1)$  $\mathcal{O}(1)$  $\mathcal{O}(1)$  $\mathcal{O}(1)$
--	--

**Complejidad:**  $\mathcal{O}(1)$

Al igual que en los dos casos anteriores, debido a la longitud del pseudocódigo, vamos a ignorar los condicionales y las asignaciones ya que se realizan en tiempo constante.

Como todas las ejecuciones del código se efectúan en tiempo constante, podemos ver de manera trivial que la complejidad es  $\mathcal{O}(1)$ .

<b>ICIzQ(in a: puntero(nodo), in b: puntero(nodo))</b>	
<pre> 1: *(a).izq = b 2: *(b).padre = a </pre>	$\mathcal{O}(1)$ $\mathcal{O}(1)$

**Complejidad:**  $\mathcal{O}(1)$

$\mathcal{O}(1) + \mathcal{O}(1) =$

$2 * \mathcal{O}(1) =$

$\mathcal{O}(1)$

<b>ICDER(in a: puntero(nodo), in b: puntero(nodo))</b>	
<pre> 1: *(a).der = b 2: *(b).padre = a </pre>	$\mathcal{O}(1)$ $\mathcal{O}(1)$

**Complejidad:**  $\mathcal{O}(1)$

$\mathcal{O}(1) + \mathcal{O}(1) =$

$2 * \mathcal{O}(1) =$

$\mathcal{O}(1)$

<b>ISSETALTURA(in a: puntero(nodo)) → res : nat</b>	
<pre> 1: if *(a).izq == NULL then 2:     if *(a).der == NULL then 3:         res ← 1 4:     else 5:         res ← 1 + (*(a).der).alt 6:     end if 7: else 8:     if *(a).der == NULL then 9:         res ← 1 + (*(a).izq).alt 10:    else 11:        if (*(a).izq).alt &gt; (*(a).der).alt then 12:            res ← 1 + (*(a).izq).alt 13:        else 14:            res ← 1 + (*(a).der).alt 15:        end if </pre>	$\mathcal{O}(1)$ $\mathcal{O}(1)$ $\mathcal{O}(1)$  $\mathcal{O}(1)$   $\mathcal{O}(1)$ $\mathcal{O}(1)$  $\mathcal{O}(1)$ $\mathcal{O}(1)$  $\mathcal{O}(1)$

```

16:   end if
17: end if

```

**Complejidad:**  $\mathcal{O}(1)$

$$\begin{aligned}
&\mathcal{O}(1) + \max(\mathcal{O}(1) + \max(\mathcal{O}(1), \mathcal{O}(1)), \mathcal{O}(1) + \max(\mathcal{O}(1), \mathcal{O}(1) + \max(\mathcal{O}(1), \mathcal{O}(1)))) = \\
&\mathcal{O}(1) + \max(\mathcal{O}(1) + \max(\mathcal{O}(1), \mathcal{O}(1)), \mathcal{O}(1) + \max(\mathcal{O}(1), \mathcal{O}(1) + \mathcal{O}(1))) = \\
&\mathcal{O}(1) + \max(\mathcal{O}(1) + \mathcal{O}(1), \mathcal{O}(1) + \max(\mathcal{O}(1), 2 * \mathcal{O}(1))) = \\
&\mathcal{O}(1) + \max(2 * \mathcal{O}(1), 3 * \mathcal{O}(1)) = \\
&\mathcal{O}(1) + 3 * \mathcal{O}(1) = 4 * \mathcal{O}(1) = \mathcal{O}(1)
\end{aligned}$$

IFACTORDESBALANCE(in a : puntero(nodo)) → res : int

```

1: if *(a).izq == NULL then                                 $\mathcal{O}(1)$ 
2:   if *(a).der == NULL then                               $\mathcal{O}(1)$ 
3:     res ← 0                                               $\mathcal{O}(1)$ 
4:   else
5:     res ← -(*(a).der).alt                                 $\mathcal{O}(1)$ 
6:   end if
7: else
8:   if *(a).der == NULL then                               $\mathcal{O}(1)$ 
9:     res ← *(a).izq).alt                                 $\mathcal{O}(1)$ 
10:  else
11:    res ← *(a).izq).alt - *(a).der).alt                   $\mathcal{O}(1)$ 
12:  end if
13: end if

```

**Complejidad:**  $\mathcal{O}(1)$

$$\begin{aligned}
&\mathcal{O}(1) + \max(\mathcal{O}(1) + (\max(\mathcal{O}(1)), \max(\mathcal{O}(1))), \mathcal{O}(1) + (\max(\mathcal{O}(1)), \max(\mathcal{O}(1)))) = \\
&\mathcal{O}(1) + \max(\mathcal{O}(1) + \mathcal{O}(1), \mathcal{O}(1) + \mathcal{O}(1)) = \\
&\mathcal{O}(1) + \max(2 * \mathcal{O}(1), 2 * \mathcal{O}(1)) = \\
&\mathcal{O}(1) + 2 * \mathcal{O}(1) = \\
&3 * \mathcal{O}(1) + \mathcal{O}(1)
\end{aligned}$$

IHIJOMASALTO(in a : puntero(nodo)) → res : puntero(nodo)

```

1: if (*(a).izq).alt > *(a).der).alt then                   $\mathcal{O}(1)$ 
2:   res ← *(a).der                                          $\mathcal{O}(1)$ 
3: else
4:   res ← *(a).izq                                          $\mathcal{O}(1)$ 
5: end if

```

**Complejidad:**  $\mathcal{O}(1)$

$$\begin{aligned}
&\mathcal{O}(1) + \max(\mathcal{O}(1), \mathcal{O}(1)) = \\
&\mathcal{O}(1) + \mathcal{O}(1) = \\
&2 * \mathcal{O}(1) = \mathcal{O}(1)
\end{aligned}$$

ITAMAÑO(in/out c : colaP(Paquete)) → res : nat

```

1: res ← c.tam                                               $\mathcal{O}(1)$ 

```

**Complejidad:**  $\mathcal{O}(1)$