

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

1.1. $\text{Secu}(\alpha)$

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

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 e_red

donde 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 3. 4.
- obtener(e2, obtener(e1, m.uniones)) < m.#sendas) \wedge
- (\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(α) 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*) =_{obs} 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** $r : \text{red}$, **in** $c_1 : \text{compu}$, **in** $c_2 : \text{compu}$) $\rightarrow res : \text{bool}$

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

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

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** $r : \text{red}$, **in** $c : \text{compu}$)

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** $r : \text{red}$, **in** $c_1 : \text{compu}$, **in** $i_1 : \text{interfaz}$, **in** $c_2 : \text{compu}$, **in** $i_2 : \text{interfaz}$)

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. Ciudad Robótica

Interfaz

se explica con: CIUDAD ROBÓTICA, ITERADOR UNIDIRECCIONAL(α).

géneros: ciudad, itRURs.

Operaciones básicas de Ciudad Robótica

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

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

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

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

Descripción: Devuelve el PróximoRUR de una ciudad, esto es, de añadirse un robot se le asignaría este RUR.

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

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

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

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

Descripción: Devuelve el mapa de la ciudad.

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

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

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

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

Descripción: Devuelve un iterador de los robots de la ciudad.

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

Pre $\equiv \{u \in \text{robots}(c)\}$

Post $\equiv \{res =_{\text{obs}} \text{estación}(u, c)\}$

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

Descripción: Devuelve la estación en la cual está el robot.

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

Pre $\equiv \{u \in \text{robots}(c)\}$

Post $\equiv \{res =_{\text{obs}} \text{tags}(u, c)\}$

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

Descripción: Devuelve los tags del robot.

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

Pre $\equiv \{u \in \text{robots}(c)\}$

Post $\equiv \{res =_{\text{obs}} \# \text{infracciones}(u, c)\}$

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

Descripción: Devuelve la cantidad de infracciones cometidas por el robot.

CREAR(**in** m : mapa) $\rightarrow res$: ciudad

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

Post $\equiv \{res =_{\text{obs}} \text{crear}(m)\}$

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

Descripción: Crea una ciudad con un mapa y sin robots.

ENTRAR(**in** ts : conj(tags), **in** e : estación, **in/out** c : ciudad)

Pre $\equiv \{c_0 \equiv c \wedge e \in \text{estaciones}(c_0)\}$

Post $\equiv \{c =_{\text{obs}} \text{entrar}(ts, e, c_0)\}$

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

Descripción: Añade un robot a la ciudad, le asigna el próximoRUR y sus infracciones son nulas.

MOVER(**in** u : rur, **in** e : estación, **in/out** c : ciudad)

Pre $\equiv \{(c_0 \equiv c \wedge u \in \text{robots}(c_0)) \wedge e \in \text{estaciones}(c_0)) \wedge_{\text{L}} \text{conectadas?}(\text{estación}(u, c_0), e \text{ mapa}(c_0))\}$

Post $\equiv \{c =_{\text{obs}} \text{mover}(u, e, c_0)\}$

Complejidad: $\mathcal{O}(|e| + |e_0| + \log_2 N_e + \log_2 N_{e_0})$

Descripción: Mueve un robot desde donde está a la estación indicada.

INSPECCIÓN(**in** e : estación, **in/out** c : ciudad)

Pre $\equiv \{c_0 \equiv c \wedge e \in \text{estaciones}(c_0)\}$

Post $\equiv \{c =_{\text{obs}} \text{inspeccion}(e, c_0)\}$

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

Descripción: Realiza la inspección de la estación indicada, remueve el robot con mayor cantidad de infracciones.

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

Post $\equiv \{res =_{\text{obs}} \text{Actual}(it)\}$

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

Descripción: Devuelve el actual del iterador de robots.

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

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

Post $\equiv \{res =_{\text{obs}} \text{Avanzar}(it)\}$

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

Descripción: Avanza el iterador de robots.

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

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

Post $\equiv \{res =_{\text{obs}} \text{HayMas?}(it)\}$

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 $\text{tupla}(\text{mapa: mapa}, \text{RUREnEst: diccString}(\text{estacion: string}, \text{robs: colaP}(\text{id: nat}, \text{inf: nat})),$
 $\text{RURs: vector de tupla}(\text{id: nat}, \text{esta?: bool}, \text{e: string}, \text{inf: nat}, \text{carac: conj(string)},$
 $\text{sendEv: arreglo_dimensionable de bool}, \text{\#RURHistoricos: nat})$

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 → bool
Rep(c) ≡ true ⇔ claves(c.RUEnEst) = estaciones(c.mapa) ∧ 1
    #RURHistoricos ≤ Long(c.RURs) ∧L (∀ i:Nat, t:<id:Nat, esta?:Bool, e:String, 2
    inf:Nat, carac:Conj(Tag), sendEv: ad(Bool)>)
    (i<#RURHistoricos ∧L ElemDeSecu(c.RURs, i) = t ⇒L (t.e ∈ estaciones(c.mapa) 3
    ∧ t.id = i ∧ tam(t.sendEv) = long(Restricciones(c.mapa)) ∧
    (t.inf > 0 ⇒ (∃ j:Nat) (j < tam(t.sendEv) ∧L ¬ (t.sendEv[j]))) ∧
    (t.esta? ⇔ (∃ e1: String) (e1 ∈ claves(c.RUEnEst) ∧L estaEnColaP?(obtener(e1, c.RUEnEst), t.id)))
    ∧ (∀ h : Nat) (h < tam(t.sendEv) ⇒L
    t.sendEv[h] = verifica?(t.carac, ElemDeSecu(Restricciones(c.mapa), h)))) ∧L
    (∀ e1, e2: String)(e1 ∈ claves(c.RUEnEst) ∧ e2 ∈ claves(c.RUEnEst) ∧ e1 ≠ e2 ⇒L 4
    (∀ n:Nat)(estaEnColaP?(obtener(e1, c.RUEnEst), n) ⇒ ¬ estaEnColaP?(obtener(e2, c.RUEnEst), n)
    ∧ n < #RURHistoricos ∧L ElemDeSecu(c.RURs, n).e = e1))

```

estaEnColaP? : ColaPri × Nat → Bool

```

estaEnColaP?(cp, n) ≡ if vacia?(cp) then
    false
else
    if desencolar(cp) = n then
        true
    else
        estaEnColaP?(Eliminar(cp, desencolar(cp)), n)
fi

```

```

Abs : e_cr c → ciudad {Rep(c)}
Abs(c) =obs u: ciudad |
    c.#RURHistoricos = ProximoRUR(U) ∧ c.mapa = mapa(u) ∧L
    robots(u) = RURQueEstan(c.RURs) ∧L
    (∀ n:Nat) (n ∈ robots(u) ⇒L estacion(n,u) = c.RURs[n].e ∧
    tags(n,u) = c.RURs[n].carac ∧ #infracciones(n,u) = c.RURs[n].inf)

```

RURQueEstan : secu(tupla) → Conj(RUR)

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

```

RURQueEstan(s) ≡ if vacia?(s) then
    ∅
else
    if Π2(prim(fin(s))) then
        Π1(prim(fin(s))) ∪ RURQueEstan(fin(s))
    else
        RURQueEstan(fin(s))
fi

```

it se representa con e_it

donde e_it es $tupla(i: nat, maxI: nat, ciudad: puntero(ciudad))$

$Rep : e_it \rightarrow bool$

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

$Abs : e_it\ u \rightarrow itUni(\alpha)$

$\{Rep(u)\}$

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

$Siguientes : itUniu \times conj(RURs)cr \rightarrow conj(RURs)$

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

$VSiguientes : ciudadc \times Nati \times conj(RURs)cr \rightarrow conj(RURs)$

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

3.3. Algoritmos

Algoritmos

IPRÓXIMORUR(in $c: ciudad$) $\rightarrow res: rur$
1: $res \leftarrow (c.\#RURHistoricos)$

$\mathcal{O}(1)$

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

IMAPA(in $c: ciudad$) $\rightarrow res: mapa$
1: $res \leftarrow c.mapa$

$\mathcal{O}(1)$

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

IROBOTS(in $c: ciudad$) $\rightarrow res: itRobots$
1: $res \leftarrow CrearIt(c.RURs)$

$\mathcal{O}(1)$

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

IESTACIÓN(in $u: rur$, in $c: ciudad$) $\rightarrow res: estación$
1: $res \leftarrow (c.RURs[u]).estacion$

$\mathcal{O}(1)$

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

ITAGS(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: $res \leftarrow \text{tupla}(\text{mapa}: m, RUREnEst: \text{Vacío}(), RURs: \text{Vacía}(), \#RURHistoricos: 0)$

$\mathcal{O}(1)$

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($c.RURs$, $c.\#RURHistoricos$, $\text{tupla}(\text{id}: c.\#RURHistoricos, \text{esta?}: \text{true}, \text{estacion}: e, \text{inf}: 0, \text{carac}: ts, \text{sendEv}: \text{EvaluarSendas}(ts, c.mapa))$)

$\mathcal{O}(1 + S * R)$

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)

$\mathcal{O}(|e| + \log_2 N_{e0})$

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

$\mathcal{O}(|e| + \log_2 N_{e0}) + \mathcal{O}(|e| + \log_2 N_e) + \mathcal{O}(|e_0|, |e|) + \max(\mathcal{O}(1), \mathcal{O}(0)) + \mathcal{O}(1) =$

$\mathcal{O}(2 * |e| + \log_2 N_e + \log_2 N_{e0}) + \mathcal{O}(|e_0| + |e|) + 2 * \mathcal{O}(1) =$

$\mathcal{O}(|e| + \log_2 N_e + \log_2 N_{e0}) + \mathcal{O}(|e_0| + |e|) =$

$\mathcal{O}(2 * |e| + |e_0| + \log_2 N_e + \log_2 N_{e0}) = \mathcal{O}(|e| + |e_0| + \log_2 N_e + \log_2 N_{e0})$ 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** $c : \text{ciudad}$) $\rightarrow res : \text{itRURs}$

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(α)

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 α 's

se explica con: DICCIONARIO(STRING, α).

géneros: diccString(α).

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(α)` se representa con `e_nodo`

donde `e_nodo` es `tupla(definicion: puntero(α), 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"

$\text{Abs} : e_nodo\ d \longrightarrow \text{diccString} \quad \{\text{Rep}(d)\}$
 $\text{Abs}(d) =_{\text{obs}} n : \text{diccString} \mid$
 $(\forall n : e_nodo) \text{Abs}(n) =_{\text{obs}} d : \text{diccString} \mid (\forall s : \text{string}) (\text{def?}(s, d) \Rightarrow_{\text{L}} ((\text{obtenerDelArbol}(s, n) \neq \text{NULL} \wedge_{\text{L}} *(\text{obtenerDelArbol}(s, n) = \text{obtener}(s, d)))) \wedge_{\text{L}}$

$\text{obtenerDelArbol} : \text{strings} \times e_nodo \longrightarrow \text{puntero}(\alpha)$

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

4.3. Algoritmos

Algoritmos

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

1: $res \leftarrow \text{iNodoVacío}()$

$\mathcal{O}(1)$

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

$\text{INODOVACÍO}() \rightarrow res : e_nodo$

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

$\mathcal{O}(1)$

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

$\mathcal{O}(1)$

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

$\mathcal{O}(1)$

4: **end for**

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

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

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

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

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

1: var $actual : \text{puntero}(e_nodo) \leftarrow \&(d)$

$\mathcal{O}(1)$

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

$\mathcal{O}(1)$

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

$\mathcal{O}(1)$

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

$\mathcal{O}(1)$

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

Complejidad: $|clv|$

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

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

Complejidad: $|clv|$

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

ISIGNIFICADO (in $d: diccString(\alpha)$, in $clv: string$) $\rightarrow res: diccString(\alpha)$	
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: $actual \leftarrow (actual \rightarrow hijos[ord(clv[i])])$	$\mathcal{O}(1)$
4: end for	
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(α), 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))	≡ if vacía?(cp) then p else if prioridad(p) < prioridad(próximo(cp)) then p else próximo(cp) fi fi
próximo(agCompu(p, c, cp))	≡ próximo(cp)
desencolar(encolar(p, cp))	≡ if vacía?(cp) then cp else if prioridad(p) < prioridad(próximo(cp)) then cp else encolar(p, desencolar(cp)) fi fi
desencolar(agCompu(p, c, cp))	≡ desencolar(cp)
suCamino(p ₁ , encolar(p ₂ , cp))	≡ if p ₁ = p ₂ then <> else suCamino(p ₁ , cp) fi
suCamino(p ₁ , agCompu(p ₂ , c, cp))	≡ if p ₁ = p ₂ then suCamino(p ₁ , cp) o c else suCamino(p ₁ , cp) fi


```

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.

géneros: colaP(Paquete).

Operaciones básicas de COLA PRIORITARIA

VACÍA?(in cp : colaP(Paquete)) $\rightarrow res$: Bool

Pre $\equiv \{true\}$

Post $\equiv \{res =_{\text{obs}} vacía?(cp)\}$

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

Descripción: Verifica si una cola esta vacía

PRÓXIMO(in cp : colaP(Paquete)) $\rightarrow res$: Paquete

Pre $\equiv \{\neg vacía?(cp)\}$

Post $\equiv \{res =_{\text{obs}} próximo(cp)\}$

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

Descripción: Devuelve el próximo paquete a desencolar

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

Pre $\equiv \{cp =_{\text{obs}} cp_0 \wedge \neg(vacía?(cp_0))\}$

Post $\equiv \{cp =_{\text{obs}} desencolar(cp_0)\}$

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

Descripción: Elimina el próximo paquete a desencolar

SUCAMINO(in p : Paquete, in cp : colaP(Paquete)) $\rightarrow res$: Secu(Compu)

Pre $\equiv \{está(p, cp)\}$

Post $\equiv \{res =_{\text{obs}} suCamino(p, cp)\}$

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

Descripción: Devuelve la secuencia de computadoras asociadas al paquete

VACÍA() $\rightarrow res$: colaP(Paquete)

Pre $\equiv \{true\}$

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

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

Descripción: Crea una nueva cola.

ENCOLAR(in p : Paquete, in/out cp : colaP(Paquete))

Pre $\equiv \{cp =_{\text{obs}} cp_0 \wedge \neg(está(p, cp_0))\}$

Post $\equiv \{cp =_{\text{obs}} encolar(p, cp_0)\}$

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

Descripción: Agrega el paquete a la cola

AGCOMPU(in p : Paquete, in c : Compu, in/out cp : colaP(Paquete))

Pre $\equiv \{cp =_{\text{obs}} cp_0 \wedge está(p, cp_0)\}$

Post $\equiv \{cp =_{\text{obs}} agCompu(p, c, cp_0)\}$

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*) =_{obs} *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:	while iTam > 0 do	$\mathcal{O}(1)$
9:	if iAux % 2 == 0 then	$\mathcal{O}(1)$
10:	aCoordenadas[iTam-1] = false	$\mathcal{O}(1)$
11:	else	
12:	aCoordenadas[iTam-1] = true	$\mathcal{O}(1)$
13:	end if	
14:	iAux \leftarrow $\lfloor iAux/2 \rfloor$	$\mathcal{O}(1)$
15:	iTam \leftarrow iTam - 1	
16:	end while	
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:	while seguir == true do	$\mathcal{O}(1)$
24:	if a \geq *(pNodo).pri then	$\mathcal{O}(1)$
25:	if a == *(pNodo).pri then	$\mathcal{O}(1)$
26:	if b > *(pNodo).seg then	$\mathcal{O}(1)$
27:	if *(pNodo).der != NULL then	$\mathcal{O}(1)$
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:	else	
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:	end if	
37:	else	
38:	if *(pNodo).izq != NULL then	$\mathcal{O}(1)$
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:	else	
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:	end if	
48:	end if	
49:	else	
50:	if *(pNodo).der != NULL then	$\mathcal{O}(1)$
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:	else	
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:	end if	
60:	end if	
61:	else	

62:	if $*(pNodo).izq \neq \text{NULL}$ then	$\mathcal{O}(1)$
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:	else	
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:	end if	
72:	end if	
73:	end while	
74:	$c.tam \leftarrow c.tam + 1$	$\mathcal{O}(1)$
75:	$seguir \leftarrow \text{true}$	$\mathcal{O}(1)$
76:	while $nroCamino \geq 0 \wedge seguir == \text{true}$ do	$\mathcal{O}(\lfloor \log_2 N \rfloor + 1)$
77:	$pNodo \leftarrow camino[nroCamino]$	$\mathcal{O}(1)$
78:	$*(pNodo).alt \leftarrow \text{ALTURA}(pNodo) \vee$	
79:	if $ \text{FACTORDESBALANCE}(camino[nroCamino]) > 1$ then	$\mathcal{O}(1)$
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:	end if	
86:	$nroCamino \leftarrow nroCamino - 1$	$\mathcal{O}(1)$
87:	end while	
88:	end if	

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

Debido a la longitud del pseudocódigo, vamos a ignorar todas las condicionales y las asignaciones en la justificación, ya que se realizan en tiempo constante. Solo nos vamos a centrar en dos puntos, la creación del arreglo "camino" y el último ciclo.

Para el arreglo asignamos esa cantidad de nodos ya que contamos con un Árbol balanceado, el cual como mucho puede necesitar de $\lfloor \log_2 N \rfloor + 1$ niveles para almacenar N nodos. Esta misma lógica la utilizamos en el último ciclo, en el cual para restaurar el balance del Árbol recorreremos el mismo desde el último nodo agregado (el cual es una hoja) hasta la raíz 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:	var $pNodo: \text{puntero}(\text{nodo}) \leftarrow c.raiz$	$\mathcal{O}(1)$
2:	ver $seguir: \text{bool} \leftarrow \text{true}$	$\mathcal{O}(1)$
3:	while $pNodo \neq \text{NULL} \wedge seguir == \text{true}$ do	$\mathcal{O}(\lfloor \log_2 N \rfloor + 1)$
4:	if $*(pNodo).pri == a \wedge *(pNodo).seg == b$ then	$\mathcal{O}(1)$
5:	$seguir \leftarrow \text{false}$	$\mathcal{O}(1)$
6:	else	
7:	if $a \geq *(pNodo).pri$ then	$\mathcal{O}(1)$
8:	if $a == *(pNodo).pri$ then	$\mathcal{O}(1)$
9:	if $b > *(pNodo).seg$ then	$\mathcal{O}(1)$
10:	$pNodo \leftarrow *(pNodo).der$	$\mathcal{O}(1)$
11:	else	
12:	$pNodo \leftarrow *(pNodo).izq$	$\mathcal{O}(1)$
13:	end if	

14:	else	
15:	pNodo \leftarrow *(pNodo).der	$\mathcal{O}(1)$
16:	end if	
17:	else	
18:	pNodo \leftarrow *(pNodo).izq	$\mathcal{O}(1)$
19:	end if	
20:	end if	
21:	end while	
22:	if pNodo \neq NULL then	$\mathcal{O}(1)$
23:	bNodo: puntero(nodo) \leftarrow NULL	$\mathcal{O}(1)$
24:	if pNodo == c.raiz then	$\mathcal{O}(1)$
25:	if *(pNodo).izq == NULL \wedge *(pNodo).der == NULL then	$\mathcal{O}(1)$
26:	c.raiz \leftarrow NULL	$\mathcal{O}(1)$
27:	delete pNodo	$\mathcal{O}(1)$
28:	end if	
29:	if *(pNodo).izq \neq NULL \wedge *(pNodo).der == NULL then	$\mathcal{O}(1)$
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:	end if	
34:	if *(pNodo).izq == NULL \wedge *(pNodo).der \neq NULL then	$\mathcal{O}(1)$
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:	end if	
39:	if *(pNodo).izq \neq NULL \wedge *(pNodo).der \neq NULL then	$\mathcal{O}(1)$
40:	var tNodo: puntero(nodo) \leftarrow pNodo.der	$\mathcal{O}(1)$
41:	while *(tNodo).izq \neq NULL do	$\mathcal{O}(\lfloor \log_2 N \rfloor + 1)$
42:	tNodo \leftarrow tNodo.izq	$\mathcal{O}(1)$
43:	end while	
44:	bNodo \leftarrow *(tNodo).padre	$\mathcal{O}(1)$
45:	if *(tNodo).der \neq NULL then	$\mathcal{O}(1)$
46:	*(*(tNodo).der).padre \leftarrow *(tNodo).padre	$\mathcal{O}(1)$
47:	end if	
48:	if *(*(tNodo).padre).izq == tNodo then	$\mathcal{O}(1)$
49:	*(*(tNodo).padre).izq \leftarrow *(tNodo).der	$\mathcal{O}(1)$
50:	else	
51:	*(*(tNodo).padre).der \leftarrow *(tNodo).der	$\mathcal{O}(1)$
52:	end if	
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:	end if	
60:	else	
61:	if *(pNodo).izq == NULL \wedge *(pNodo).der == NULL then	$\mathcal{O}(1)$
62:	if *(*(pNodo).padre).izq == pNodo then	$\mathcal{O}(1)$
63:	*(*(pNodo).padre).izq \leftarrow NULL	$\mathcal{O}(1)$
64:	bNodo \leftarrow *(pNodo).padre	$\mathcal{O}(1)$
65:	delete pNodo	$\mathcal{O}(1)$
66:	else	
67:	*(*(pNodo).padre).der \leftarrow NULL	$\mathcal{O}(1)$
68:	bNodo \leftarrow *(pNodo).padre	$\mathcal{O}(1)$
69:	delete pNodo	$\mathcal{O}(1)$
70:	end if	

71:	end if	
72:	if $*(pNodo).izq \neq \text{NULL} \wedge *(pNodo).der == \text{NULL}$ then	$\mathcal{O}(1)$
73:	if $*(*(pNodo).padre).izq == pNodo$ then	$\mathcal{O}(1)$
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:	else	
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:	end if	
84:	end if	
85:	if $*(pNodo).izq == \text{NULL} \wedge *(pNodo).der \neq \text{NULL}$ then	$\mathcal{O}(1)$
86:	if $*(*(pNodo).padre).izq == pNodo$ then	$\mathcal{O}(1)$
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:	else	
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:	end if	
97:	end if	
98:	if $*(pNodo).izq \neq \text{NULL} \wedge *(pNodo).der \neq \text{NULL}$ then	$\mathcal{O}(1)$
99:	var tNodo: puntero(nodo) $\leftarrow pNodo.der$	$\mathcal{O}(1)$
100:	while $*(tNodo).izq \neq \text{NULL}$ do	$\mathcal{O}(\lfloor \log_2 N \rfloor + 1)$
101:	$tNodo \leftarrow tNodo.izq$	$\mathcal{O}(1)$
102:	end while	
103:	$bNodo \leftarrow *(tNodo).padre$	$\mathcal{O}(1)$
104:	if $*(tNodo).der \neq \text{NULL}$ then	$\mathcal{O}(1)$
105:	$*(*(tNodo).der).padre \leftarrow *(tNodo).padre$	$\mathcal{O}(1)$
106:	end if	
107:	if $*(*(tNodo).padre).izq == tNodo$ then	$\mathcal{O}(1)$
108:	$*(*(tNodo).padre).izq \leftarrow *(tNodo).der$	$\mathcal{O}(1)$
109:	else	
110:	$*(*(tNodo).padre).der \leftarrow *(tNodo).der$	$\mathcal{O}(1)$
111:	end if	
112:	if $*(*(pNodo).padre).izq == pNodo$ then	$\mathcal{O}(1)$
113:	$*(*(tNodo).padre).izq \leftarrow tNodo$	$\mathcal{O}(1)$
114:	else	
115:	$*(*(tNodo).padre).der \leftarrow pNodo$	$\mathcal{O}(1)$
116:	end if	
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:	end if	
124:	c.tam \leftarrow c.tam + 1	
125:	while b $\neq \text{NULL}$ do	$\mathcal{O}(\lfloor \log_2 N \rfloor + 1)$
126:	$*(b).alt \leftarrow \text{SETALTURA}(b)$	$\mathcal{O}(1)$
127:	if $ \text{FACTORDEDESBALANCE}(b) > 1$ then	$\mathcal{O}(1)$

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:	end if	
133:	b ← *(b).padre	$\mathcal{O}(1)$
134:	end while	
135:	end if	
136:	end if	

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:

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

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

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

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

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 $\mathcal{O}(1)$.

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

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

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

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

$\mathcal{O}(1)$

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

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

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

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

$\mathcal{O}(1)$

ISSETALTURA(in a: puntero(nodo)) → res : nat	
<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 > (*(a).der).alt then 12: res ← 1 + (*(a).izq).alt 13: else 14: res ← 1 + (*(a).der).alt 15: end if </pre>	<p>$\mathcal{O}(1)$</p> <p>$\mathcal{O}(1)$</p> <p>$\mathcal{O}(1)$</p> <p>$\mathcal{O}(1)$</p> <p>$\mathcal{O}(1)$</p> <p>$\mathcal{O}(1)$</p> <p>$\mathcal{O}(1)$</p> <p>$\mathcal{O}(1)$</p> <p>$\mathcal{O}(1)$</p> <p>$\mathcal{O}(1)$</p> <p>$\mathcal{O}(1)$</p>

```

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: $\mathcal{O}(1)$