

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. TAD Extendidos	3
1.1. Secu(α)	3
1.2. Mapa	3
2. Restricción	4
2.1. Representacion	4
2.2. InvRep y Abs	4
2.3. Algoritmos	5
3. Mapa	8
3.1. Representacion	9
3.2. InvRep y Abs	9
3.3. Algoritmos	10
4. Ciudad Robótica	12
4.1. Representacion	13
4.2. InvRep y Abs	13
4.3. Algoritmos	15
5. Diccionario String(α)	18
5.1. Representacion	18
5.2. InvRep y Abs	19
5.3. Algoritmos	19
6. Cola Prioritaria	21
6.1. TAD COLAPRIORITARIA	21
6.2. Representacion	22
6.3. InvRep y Abs	22
6.4. Algoritmos	23

1. Tad Extendidos

1.1. Secu(α)

otras operaciones

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

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

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. Restricción

Interfaz

se explica con: RESTRICCIÓN.

géneros: rest.

Operaciones básicas de Restricción

VERIFICA?(in c : conj(tag), in r : rest) $\rightarrow res$: bool

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

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

Complejidad: $\mathcal{O}(|r|)$

Descripción: Verifica que los tags c verifican r .

< >(in t : tag) $\rightarrow res$: rest

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

Post $\equiv \{res =_{\text{obs}} \langle t \rangle\}$

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

Descripción: Creo una restricción con t .

AND(in r_1 : rest, in r_2 : rest) $\rightarrow res$: rest

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

Post $\equiv \{res =_{\text{obs}} r_1 \text{ AND } r_2\}$

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

Descripción: Creo una restricción AND de r_1 y r_2 .

OR(in r_1 : rest, in r_2 : rest) $\rightarrow res$: rest

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

Post $\equiv \{res =_{\text{obs}} r_1 \text{ OR } r_2\}$

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

Descripción: Creo una restricción OR de r_1 y r_2 .

NOT(in r : rest) $\rightarrow res$: rest

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

Post $\equiv \{res =_{\text{obs}} \neg r\}$

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

Descripción: Creo una restricción $\neg r$.

2.1. Representacion

Representación

rest se representa con e_rest

donde e_rest es tupla(A : tag, $P1$: puntero(e_rest), $P2$: puntero(e_rest), T : tipoRest)

donde tipoRest es enum(<>, NOT, AND, OR)

2.2. InvRep y Abs

1. Para todo nodo del árbol, si ambos punteros apuntan a 'null', entonces el tipo de restricción es '<>' o 'not'. Si los dos punteros no apuntan a 'null' el tipo de restricción es 'and' o 'or'. No debe ocurrir que uno de los punteros apunte a 'null' y el otro no.

Rep : e_rest \rightarrow bool

Rep(e) \equiv cumple?(e)

```

cumple? : tupla(tag × puntero(rest) × puntero(rest) × tipoRest) → bool
cumple?(e) ≡ if π2(e) = NULL ∧ π3(e) = NULL then
    π4(e) = <> ∨ π4(e) = NOT
  else
    if (π2(e) ≠ NULL ∧ π3(e) ≠ NULL) ∧ (π4(e) = AND ∨ π4(e) = OR) then
      cumple?(*(π2(e))) ∧ cumple?(*(π3(e)))
    else
      false
  fi
fi

```

Abs : e_{rest} e → rest {Rep(e)}
 Abs(e) =_{obs} r: rest |
 (∀ c: Conj(tag))(vale?(c,e) ⇔ verifica?(c,r))

```

vale? : Conj(tag) × tuplaRes → Bool

tuplaRes es <tag, puntero(tuplaRes), puntero(tuplaRes), tipoRest>

vale?(c,e) ≡ if *(π4(e)) = <> then
    *(π1(e)) ∈ c
  else
    if *(π4(e)) = AND then
      vale?(c, *(π2(e))) ∧ vale?(c, *(π3(e)))
    else
      vale?(c, *(π2(e))) ∨ vale?(c, *(π3(e)))
    fi
  fi
fi

```

2.3. Algoritmos

Algoritmos

```

iVERIFICA?(in c: conj(tag), in r: rest) → res : bool
1: if r.T =obs ⟨⟩ then O(1)
2:   res ← Pertenece?(c, r.A) O(|r|)
3: else
4:   if r.T =obs NOT then O(1)
5:     res ← ¬Pertenece?(c, r.A) O(|r|)
6:   else
7:     if r.T =obs AND then O(1)
8:       res ← (Verifica?(c, r.P1) ∧ Verifica?(c, r.P2)) O(|r|)
9:     else
10:      res ← (Verifica?(c, r.P1) ∨ Verifica?(c, r.P2)) O(|r|)
11:    end if
12:  end if
13: end if

```

Complejidad: $\mathcal{O}(|r|)$

$\mathcal{O}(1) + \max(\mathcal{O}(|r|), \mathcal{O}(1) + \max(\mathcal{O}(|r|), \mathcal{O}(1) + \max(\mathcal{O}(|r|), \mathcal{O}(|c|)))) =$
 $\mathcal{O}(1) + \max(\mathcal{O}(|r|), \mathcal{O}(1) + \max(\mathcal{O}(|r|), \mathcal{O}(1) + \mathcal{O}(|r|))) =$
 $\mathcal{O}(1) + \max(\mathcal{O}(|r|), \mathcal{O}(1) + \mathcal{O}(1) + \mathcal{O}(|r|)) =$
 $\mathcal{O}(1) + \mathcal{O}(1) + \mathcal{O}(1) + \mathcal{O}(|r|) =$
 $3 * \mathcal{O}(1) + \mathcal{O}(|r|) = \mathcal{O}(|r|)$

I⟨ ⟩ (in t : tag) $\rightarrow res$: rest	
1: $res \leftarrow \text{tupla}(A: \text{string}, P_1: \text{puntero}(e_rest)), P_2: \text{puntero}(e_rest), T: \text{TipoRest})$	$\mathcal{O}(1)$
2: $res.P_1 \leftarrow \text{NULL}$	$\mathcal{O}(1)$
3: $res.P_2 \leftarrow \text{NULL}$	$\mathcal{O}(1)$
4: $res.A \leftarrow t$	$\mathcal{O}(1)$
5: $res.T \leftarrow \langle \rangle$	$\mathcal{O}(1)$

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

$$\sum_{i=1}^5 \mathcal{O}(1) = 5 * \mathcal{O}(1) = \mathcal{O}(1)$$

IAND (in r_1 : rest , in r_2 : rest) $\rightarrow res$: rest	
1: $res \leftarrow \text{tupla}(A: \text{string}, P_1: \text{puntero}(e_rest)), P_2: \text{puntero}(e_rest), T: \text{TipoRest})$	$\mathcal{O}(1)$
2: $res.P_1 \leftarrow r_1$	$\mathcal{O}(1)$
3: $res.P_2 \leftarrow r_2$	$\mathcal{O}(1)$
4: $res.A \leftarrow \text{Vacía}()$	$\mathcal{O}(1)$
5: $res.T \leftarrow \text{AND}$	$\mathcal{O}(1)$

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

$$\sum_{i=1}^5 \mathcal{O}(1) = 5 * \mathcal{O}(1) = \mathcal{O}(1)$$

IOR (in r_1 : rest , in r_2 : rest) $\rightarrow res$: rest	
1: $res \leftarrow \text{tupla}(A: \text{string}, P_1: \text{puntero}(e_rest)), P_2: \text{puntero}(e_rest), T: \text{TipoRest})$	$\mathcal{O}(1)$
2: $res.P_1 \leftarrow r_1$	$\mathcal{O}(1)$
3: $res.P_2 \leftarrow r_2$	$\mathcal{O}(1)$
4: $res.A \leftarrow \text{Vacía}()$	$\mathcal{O}(1)$
5: $res.T \leftarrow \text{OR}$	$\mathcal{O}(1)$

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

$$\sum_{i=1}^5 \mathcal{O}(1) = 5 * \mathcal{O}(1) = \mathcal{O}(1)$$

INOT (in r : rest) $\rightarrow res$: rest	
1: $res \leftarrow \text{tupla}(A: \text{string}, P_1: \text{puntero}(e_rest)), P_2: \text{puntero}(e_rest), T: \text{TipoRest})$	$\mathcal{O}(1)$
2: if $r.T =_{\text{obs}} \langle \rangle$ then	$\mathcal{O}(1)$
3: $res.P_1 \leftarrow \text{NULL}$	$\mathcal{O}(1)$
4: $res.P_2 \leftarrow \text{NULL}$	$\mathcal{O}(1)$
5: $res.A \leftarrow r.A$	$\mathcal{O}(1)$
6: $res.T \leftarrow \text{NOT}$	$\mathcal{O}(1)$
7: else	
8: if $r.T =_{\text{obs}} \text{NOT}$ then	$\mathcal{O}(1)$
9: $res.P_1 \leftarrow \text{NULL}$	$\mathcal{O}(1)$
10: $res.P_2 \leftarrow \text{NULL}$	$\mathcal{O}(1)$
11: $res.A \leftarrow r.A$	$\mathcal{O}(1)$
12: $res.T \leftarrow \langle \rangle$	$\mathcal{O}(1)$
13: else	
14: if $r.T =_{\text{obs}} \text{OR}$ then	$\mathcal{O}(1)$
15: $res \leftarrow \text{AND}(\text{NOT}(*r.P_1), \text{NOT}(*r.P_2))$	$\mathcal{O}(R)$
16: else	
17: $res \leftarrow \text{OR}(\text{NOT}(*r.P_1), \text{NOT}(*r.P_2))$	$\mathcal{O}(R)$
18: end if	
19: end if	

20: **end if**

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

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

3. Mapa

Interfaz

se explica con: MAPA, ITERADOR UNIDIRECCIONAL(α).

géneros: mapa, itConj(Estacion).

Operaciones básicas de Mapa

ESTACIONES(in m : mapa) $\rightarrow res$: itConj(Estacion)

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

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

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

Descripción: Devuelve las estaciones del mapa.

CONECTADAS?(in e_1 : estación, in e_2 : estación, in m : mapa) $\rightarrow res$: bool

Pre $\equiv \{\{e_1, e_2\} \subseteq \text{estaciones}(m)\}$

Post $\equiv \{res =_{\text{obs}} \text{conectadas?}(e_1, e_2, m)\}$

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

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

RESTRICCIÓN(in e_1 : estación, in e_2 : estación, in m : mapa) $\rightarrow res$: restricción

Pre $\equiv \{\{e_1, e_2\} \subseteq \text{estaciones}(m) \wedge_L \text{conectadas?}(e_1, e_2, m)\}$

Post $\equiv \{res =_{\text{obs}} \text{restricción}(e_1, e_2, m)\}$

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

Descripción: Devuelve la restricción entre dos estaciones.

VACIO() $\rightarrow res$: mapa

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

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

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

Descripción: Crea un mapa sin estaciones.

AGREGAR(in e : estación, in/out m : mapa)

Pre $\equiv \{m_0 =_{\text{obs}} m \wedge \neg(e \in \text{estaciones}(m))\}$

Post $\equiv \{m =_{\text{obs}} \text{agregar}(e, m_0)\}$

Complejidad: $\mathcal{O}(|e|)$

Descripción: Agrega una estación al mapa.

CONECTAR(in e_1 : estación, in e_2 : estación, in r : restricción, in/out m : mapa)

Pre $\equiv \{m_0 =_{\text{obs}} m \wedge \{e_1, e_2\} \subseteq \text{estaciones}(m) \wedge_L \neg \text{conectadas?}(e_1, e_2, m)\}$

Post $\equiv \{m =_{\text{obs}} \text{conectar}(e_1, e_2, r, \text{conectar}(e_2, e_1, r, m_0))\}$

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

Descripción: Conecta dos estaciones y les añade la restricción correspondiente.

NROCONEXION(in e_1 : estación, in e_2 : estación, in m : mapa) $\rightarrow res$: nat

Pre $\equiv \{\{e_1, e_2\} \subseteq \text{estaciones}(m) \wedge_L \text{conectadas?}(e_1, e_2, m)\}$

Post $\equiv \{res =_{\text{obs}} \text{nroConexion}(e_1, e_2, m)\}$

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

Descripción: Obtiene el Nro. de Senda entre dos estaciones.

EVALUARSENDAS(in c : conj(tag), in m : mapa) $\rightarrow res$: arreglo_dimensionable de bool

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

Post $\equiv \{(\forall i: \text{Nat}) (0 \leq i < \text{long}(\text{restricciones}(m)) \Rightarrow_L res[i] =_{\text{obs}} \text{verifica?}(c, \text{elemDeSecu}(\text{restricciones}(m), i)))\}$

Complejidad: $\mathcal{O}(S \cdot R)$

Descripción: Devuelve un arreglo con todas las sendas evaluadas con respecto al conjunto c . NOTA: La R es el costo de evaluar la restricción mas grande.

RESTRICCIONES(in m : mapa) $\rightarrow res$: arreglo_dimensionable de restricción

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

Post $\equiv \{(\forall i: \text{Nat}) (0 \leq i < \text{long}(\text{restricciones}(m)) \Rightarrow_{\text{L}} \text{res}[i] =_{\text{obs}} \text{elemDeSecu}(\text{restricciones}(m), i))\}$

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

Descripción: Devuelve un arreglo con todas las restricciones

3.1. Representacion

Representación

mapa se representa con e_mapa

donde **e_mapa** es `tupla(uniones: diccString(estacion: string, diccString(estacion: string, senda: nat)),
estaciones: conj(estacion), sendas: vector de restriccion, #sendas: nat)`

3.2. 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 sinificado 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` \longrightarrow `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
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` \longrightarrow `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` \longrightarrow `mapa` `{Rep(m)}`

Abs(*m*) $=_{\text{obs}}$ *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)))`

3.3. Algoritmos

Algoritmos

IESTACIONES(**in** *mapa*: **mapa**) \rightarrow *res* : **itConj**(**estacion**)
 1: *res* \leftarrow **CrearIt**(*m.estaciones*) $\mathcal{O}(1)$

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

ICONECTADAS?(**in** *e*₁: **estación**, **in** *e*₂: **estación**, **in** *m*: **mapa**) \rightarrow *res* : **bool**
 1: *res* \leftarrow **Definido?**(**Significado**(*m.uniones*, *e*₁), *e*₂) $\mathcal{O}(|e_1| + |e_2|)$

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

IRESTRICCIÓN(**in** *e*₁: **estación**, **in** *e*₂: **estación**, **in** *m*: **mapa**) \rightarrow *res* : **restricción**
 1: *res* \leftarrow *m.sendas*[**Significado**(**Significado**(*m.uniones*, *e*₁), *e*₂)] $\mathcal{O}(|e_1| + |e_2| + 1)$

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

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

IVACIO() \rightarrow *res* : **mapa**
 1: *res* \leftarrow **tupla**(*uniones*: **Vacío**(), *estaciones*: **Vacío**(), *sendas*: **Vacía**(), *#sendas*: 0) $\mathcal{O}(1 + 1 + 1 + 1 + 1)$

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

$\sum_{i=1}^5 \mathcal{O}(1) =$
 $5 * \mathcal{O}(1) = \mathcal{O}(1)$

IAGREGAR(**in** *e*: **estación**, **in/out** *m*: **mapa**)
 1: **Agregar**(*m.estaciones*, *e*) $\mathcal{O}(1)$
 2: **Definir**(*m.uniones*, *e*, **Vacío**()) $\mathcal{O}(|e|)$

Complejidad: $\mathcal{O}(|e|)$

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

ICONECTAR(**in** *e*₁: **estación**, **in** *e*₂: **estación**, **in** *r*: **restricción**, **in/out** *m*: **mapa**)
 1: **Definir**(**Significado**(*m.uniones*, *e*₁), *e*₂, *m.#senda*) $\mathcal{O}(|e_1| + |e_2|)$
 2: **Definir**(**Significado**(*m.uniones*, *e*₂), *e*₁, *m.#senda*) $\mathcal{O}(|e_2| + |e_1|)$
 3: **Agregar**(*m.sendas*, *m.#sendas*, *r*) $\mathcal{O}(1)$
 4: *m.#sendas* ++ $\mathcal{O}(1)$

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

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

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

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

INROCONEXION(**in** *e*₁: **estación**, **in** *e*₂: **estación**, **in** *m*: **mapa**) \rightarrow *res* : **nat**
 1: *res* \leftarrow **Significado**(**Significado**(*m.uniones*, *e*₁), *e*₂) $\mathcal{O}(|e_1| + |e_2|)$

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

IEVALUARSENDAS(**in** $c: \text{conj}(\text{tag})$, **in** $m: \text{mapa}$) $\rightarrow res: \text{arreglo_dimesionable de bool}$

1: $res \leftarrow \text{arreglo}[m.\#sendas]$ de bool	$\mathcal{O}(1)$
2: var $i: \text{nat} \leftarrow 0$	$\mathcal{O}(1)$
3: while $i < m.\#sendas$ do	$\mathcal{O}(1)$
4: $res[i] \leftarrow \text{Verifica?}(c, m.sendas[i])$	$\mathcal{O}(R)$
5: $i++$	$\mathcal{O}(1)$
6: end while	

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

$$\begin{aligned} \mathcal{O}(1) + \mathcal{O}(1) + \sum_{i=1}^S (\mathcal{O}(R) + \mathcal{O}(1)) &= \\ 2 * \mathcal{O}(1) + S * (\mathcal{O}(R) + \mathcal{O}(1)) &= \\ 2 * \mathcal{O}(1) + S * \mathcal{O}(1) + S * \mathcal{O}(R) &= \\ \mathcal{O}(S) + S * \mathcal{O}(R) &= \\ \mathcal{O}(S + S * R) &= \mathcal{O}(S * R) \end{aligned}$$

S es $|m.sendas|$ y R es la longitud de la restriccion mas grande

IRESTRICCIONES(**in** $m: \text{mapa}$) $\rightarrow res: \text{arreglo_dimesionable de restriccion}$

1: $res \leftarrow \text{arreglo_dimensionable}[m.\#sendas]$	$\mathcal{O}(1)$
2: var $i: \text{nat} \leftarrow 0$	$\mathcal{O}(1)$
3: while $i < m.\#sendas$ do	$\mathcal{O}(1)$
4: $res[i] \leftarrow m.sendas[i]$	$\mathcal{O}(1)$
5: $i++$	$\mathcal{O}(1)$
6: end while	

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

$$\begin{aligned} \mathcal{O}(1) + \mathcal{O}(1) + \sum_{i=1}^S (\mathcal{O}(1) + \mathcal{O}(1)) &= \\ 2 * \mathcal{O}(1) + S * 2 * \mathcal{O}(1) &= \\ 2 * \mathcal{O}(1) + 2 * \mathcal{O}(S) &= \\ 2 * \mathcal{O}(S) &= \mathcal{O}(S) \end{aligned}$$

4. 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 $\equiv \{(c_0 \equiv c \wedge u \in \text{robots}(c_0)) \wedge e \in \text{estaciones}(c_0)\} \wedge_L \text{conectadas?}(\text{estación}(u, c_0), e \text{ mapa}(c_0))\}$

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.

4.1. Representacion

Representación

ciudad se representa con e_cr

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

4.2. 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.RUREnEst) = 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.RUREnEst) ∧L estaEnColaP?(obtener(e1, c.RUREnEst), 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.RUREnEst) ∧ e2 ∈ claves(c.RUREnEst) ∧ e1 ≠ e2 ⇒L 4
    (∀ n:Nat)(estaEnColaP?(obtener(e1, c.RUREnEst), n) ⇒ ¬ estaEnColaP?(obtener(e2, c.RUREnEst), 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 $tupla(i: nat, maxI: nat, ciudad: puntero(ciudad))$

$Rep : e_it \rightarrow bool$

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

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

$\{Rep(u)\}$

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

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

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

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

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

4.3. 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: var it : itConj(Estacion) \leftarrow 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)

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

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: var rur : nat \leftarrow Desencolar(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: $itRURS \leftarrow \text{tupla}(i : 0, \text{maxI} : c.\#RURHistoricos, \text{ciudad} : \&c)$

$\mathcal{O}(1)$

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

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

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

$\mathcal{O}(1)$

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: $res \leftarrow (it.i < it.\text{maxI})$

$\mathcal{O}(1)$

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

5. 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.

5.1. 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))`

5.2. 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

5.3. 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: $|clv|$

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

6. Cola Prioritaria

6.1. TAD COLAPRIORITARIA

TAD COLAPRIORITARIA

igualdad observacional

$$(\forall c1, c2 : \text{colaPr}) (c1 =_{\text{obs}} c2 \iff ((\forall a, b : \text{nat})(\text{esta?}(a, b, c1) \iff \text{esta?}(a, b, c2))))$$

géneros colaPr

exporta colaPr, generadores, observadores

usa BOOL, NAT

observadores básicos

vacía? : colaPr $c \longrightarrow \text{bool}$

esta? : nat $a \times \text{nat } b \times \text{colaPr } c \longrightarrow \text{bool}$

borrar : nat $a \times \text{nat } b \times \text{colaPr } c \longrightarrow \text{colaPr}$

$\{\text{esta?}(a, b, c)\}$

proximo : colaPr $c \longrightarrow \text{tupla}(\text{nat}, \text{nat})$

$\{\neg \text{vacía?}(c)\}$

generadores

vacía : $\longrightarrow \text{colaPr}$

encolar : nat $a \times \text{nat } b \times \text{colaPr } c \longrightarrow \text{colaPr}$

$\{\neg \text{esta?}(a, b, c)\}$

otras operaciones

desencolar : colaPr $c \longrightarrow \text{colaPr}$

$\{\neg \text{vacía?}(c)\}$

axiomas $\forall c : \text{colaPr } \forall a, b, a1, b1 : \text{nat}$

vacía?(vacía) $\equiv \text{true}$

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

esta?($a1, b1, \text{vacía}$) $\equiv \text{false}$

esta?($a1, b1, \text{encolar}(a, b, c)$) $\equiv \text{if } a = a1 \wedge b = b1 \text{ then true else esta?}(a1, b1, c) \text{ fi}$

borrar($a1, b1, \text{encolar}(a, b, c)$) $\equiv \text{if } a = a1 \wedge b = b1 \text{ then } c \text{ else encolar}(a, b, \text{borrar}(a1, b1, c)) \text{ fi}$

proximo(encolar(a, b, c)) $\equiv \text{if vacía?}(c) \vee_L \pi_1(\text{proximo}(c)) \leq a \text{ then}$

$\text{if } \pi_1(\text{proximo}(c)) = a \text{ then}$

$\text{if } \pi_1(\text{proximo}(c)) < b \text{ then } \langle a, b \rangle \text{ else proximo}(C) \text{ fi}$

else

$\langle a, b \rangle$

fi

else

proximo(C)

fi

desencolar(c)

$\equiv \text{borrar}(\text{proximo}(c))$

Fin TAD

Interfaz

se explica con: COLAPRIORITARIA.

géneros: colaP.

Operaciones básicas de Restricción

VACÍO() $\rightarrow res : \text{colaP}$

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

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

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

Descripción: Crea una nueva cola.

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

Pre $\equiv \{c_t =_{\text{obs}} c\}$

Post $\equiv \{c =_{\text{obs}} \text{encolar}(a, b, c_t)\}$

Complejidad: $\mathcal{O}(\log_2 N)$ con N siendo la cantidad total de elementos en la cola.

Descripción: Agrega dos nuevos numeros a la cola.

VACIA?(in c : colaP) $\rightarrow res$: bool

Pre $\equiv \{true\}$

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

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

Descripción: Revisa si la cola contiene por lo menos algun elemento.

DESENCOLAR(in/out c : colaP) $\rightarrow res$: tupla(a: nat, b: nat)

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

Post $\equiv \{res =_{\text{obs}} \text{proximo}(c_t) \wedge c =_{\text{obs}} \text{desencolar}(c_t)\}$

Complejidad: $\mathcal{O}(\log_2 N)$ con N siendo la cantidad total de elementos en la cola.

Descripción: Devuelve la tupla de numeros mas grande y la quita de la cola.

TAMAÑO(in/out c : colaP) $\rightarrow res$: nat

Pre $\equiv \{true\}$

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

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

Descripción: Devuelve la cantidad total de elementos.

ELIMINAR(in/out c : colaP, in a : nat, in b : nat)

Pre $\equiv \{c = c_0\}$

Post $\equiv \{c =_{\text{obs}} \text{borrar}(c_0, k)\}$

Complejidad: $\mathcal{O}(\log_2 N)$ con N siendo la cantidad total de elementos en la cola.

Descripción: Si a y b se encuentran en la cola, los quita de la misma.

6.2. Representacion

Representación

Para representar la cola elegimos hacerla sobre un arbol AVL.

colaP se representa con eCola

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

donde nodo es tupla(pri: nat, seg: nat, padre: puntero(nodo), izq: puntero(nodo), der: puntero(nodo), alt: nat)

6.3. InvRep y Abs

InvRep en lenguaje coloquial:

1. La componente "tam" de eCola 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:

$Abs : colaP\ c \longrightarrow colaPr$ $\{Rep(c)\}$
 $Abs(c) =_{obs} p : colaPr \mid mismosProximos(c, p)$
 $mismosProximos : \langle puntero(nodo) \times nat \rangle \longrightarrow bool$
 $mismosProximos(c, p) \equiv$ **if** $\pi_1(c) = NULL \wedge vacia?(p)$ **then**
 true
 else
 if $(\pi_1(c) = NULL \wedge \neg vacia?(p)) \vee (\pi_1(c) \neq NULL \wedge vacia?(p))$ **then**
 false
 else
 if $maxElem(*(\pi_1(c))) = proximo(p)$ **then**
 $mismosProximos(borrarMax(*(\pi_1(c))), borrar(\pi_1(proximo(p)), \pi_2(proximo(p)), p)$
 else
 false
 fi
 fi
 fi

Las funciones "maxElem" y "borrarMax" no han sido axiomatizadas. Ya que estamos trabajando con Arboles Binarios de Busqueda (en nuestro caso AVL) la logica de ambas funciones es la misma que esta expresada en el pseudocodigo del modulo. En particular "maxElem" se limita a buscar el nodo mas a la derecha del arbol, mientras que "borrarMax" una vez encontrado el maximo, procede a eliminarlo y reordenar el arbol.

6.4. Algoritmos

Algoritmos

IVACIO() $\rightarrow res : colaP$ 1: var res: colaP \leftarrow tupla(NULL, 0)	$\mathcal{O}(1)$
---	------------------

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

IAGREGAR(in/out $c : colaP$, in $a : nat$, in $b : nat$)	
1: if $c.raiz == NULL$ then	
2: $c.raiz \leftarrow \&(tupla(a, b, NULL, NULL, NULL, 1))$	$\mathcal{O}(1)$
3: $c.tam \leftarrow 1$	$\mathcal{O}(1)$
4: else	
5: var seguir: bool \leftarrow true	$\mathcal{O}(1)$
6: var pNodo: puntero(nodo) $\leftarrow c.raiz$	$\mathcal{O}(1)$
7: var camino: arreglo[$\lfloor \log_2(c.tam) \rfloor + 1$] de puntero(nodo)	$\mathcal{O}(\lfloor \log_2(c.tam) \rfloor + 1)$
8: var nroCamino: nat	
9: camino[0] $\leftarrow pNodo$	$\mathcal{O}(1)$
10: nroCamino $\leftarrow 0$	$\mathcal{O}(1)$
11: while seguir == true do	$\mathcal{O}(1)$
12: if $a \geq *(pNodo).pri$ then	$\mathcal{O}(1)$
13: if $a == *(pNodo).pri$ then	$\mathcal{O}(1)$
14: if $b > *(pNodo).seg$ then	$\mathcal{O}(1)$
15: if $*(pNodo).der \neq NULL$ then	$\mathcal{O}(1)$
16: $pNodo \leftarrow *(pNodo).der$	$\mathcal{O}(1)$

17:	nroCamino \leftarrow nroCamino + 1	$\mathcal{O}(1)$
18:	camino[nroCamino] \leftarrow pNodo	$\mathcal{O}(1)$
19:	else	
20:	*(pNodo).der \leftarrow &(tupla(a, b, pNodo, NULL, NULL, 1))	$\mathcal{O}(1)$
21:	nroCamino \leftarrow nroCamino + 1	$\mathcal{O}(1)$
22:	camino[nroCamino] \leftarrow *(pNodo).der	$\mathcal{O}(1)$
23:	seguir \leftarrow false	$\mathcal{O}(1)$
24:	end if	
25:	else	
26:	if *(pNodo).izq \neq NULL then	$\mathcal{O}(1)$
27:	pNodo \leftarrow *(pNodo).izq	$\mathcal{O}(1)$
28:	nroCamino \leftarrow nroCamino + 1	$\mathcal{O}(1)$
29:	camino[nroCamino] \leftarrow pNodo	$\mathcal{O}(1)$
30:	else	
31:	*(pNodo).izq \leftarrow &(tupla(a, b, pNodo, NULL, NULL, 1))	$\mathcal{O}(1)$
32:	nroCamino \leftarrow nroCamino + 1	$\mathcal{O}(1)$
33:	camino[nroCamino] \leftarrow *(pNodo).izq	$\mathcal{O}(1)$
34:	seguir \leftarrow false	$\mathcal{O}(1)$
35:	end if	
36:	end if	
37:	else	
38:	if *(pNodo).der \neq NULL then	$\mathcal{O}(1)$
39:	pNodo \leftarrow *(pNodo).der	$\mathcal{O}(1)$
40:	nroCamino \leftarrow nroCamino + 1	$\mathcal{O}(1)$
41:	camino[nroCamino] \leftarrow pNodo	$\mathcal{O}(1)$
42:	else	
43:	*(pNodo).der \leftarrow &(tupla(a, b, pNodo, NULL, NULL, 1))	$\mathcal{O}(1)$
44:	nroCamino \leftarrow nroCamino + 1	$\mathcal{O}(1)$
45:	camino[nroCamino] \leftarrow *(pNodo).der	$\mathcal{O}(1)$
46:	seguir \leftarrow false	$\mathcal{O}(1)$
47:	end if	
48:	end if	
49:	else	
50:	if *(pNodo).izq \neq NULL then	$\mathcal{O}(1)$
51:	pNodo \leftarrow *(pNodo).izq	$\mathcal{O}(1)$
52:	nroCamino \leftarrow nroCamino + 1	$\mathcal{O}(1)$
53:	camino[nroCamino] \leftarrow pNodo	$\mathcal{O}(1)$
54:	else	
55:	*(pNodo).izq \leftarrow &(tupla(a, b, pNodo, NULL, NULL, 1))	$\mathcal{O}(1)$
56:	nroCamino \leftarrow nroCamino + 1	$\mathcal{O}(1)$
57:	camino[nroCamino] \leftarrow *(pNodo).izq	$\mathcal{O}(1)$
58:	seguir \leftarrow false	$\mathcal{O}(1)$
59:	end if	
60:	end if	
61:	end while	
62:	c.tam \leftarrow c.tam + 1	$\mathcal{O}(1)$
63:	seguir \leftarrow true	$\mathcal{O}(1)$
64:	while nroCamino \geq 0 \wedge seguir == true do	$\mathcal{O}(\lfloor \log_2 N \rfloor + 1)$
65:	pNodo \leftarrow camino[nroCamino]	$\mathcal{O}(1)$
66:	*(pNodo).alt \leftarrow ALTURA(pNodo) v	
67:	if FACTORDESBALANCE(camino[nroCamino]) $>$ 1 then	$\mathcal{O}(1)$
68:	pNodo \leftarrow ROTAR(HIJOMASALTO (HIJOMASALTO(pNodo)), HijoMasAlto(pNodo), pNodo)	$\mathcal{O}(1)$
69:	*(*(pNodo).izq).alt \leftarrow ALTURA(*(pNodo).izq)	$\mathcal{O}(1)$
70:	*(*(pNodo).der).alt \leftarrow ALTURA(*(pNodo).der)	$\mathcal{O}(1)$
71:	*(pNodo).alt \leftarrow ALTURA(*(pNodo))	$\mathcal{O}(1)$
72:	seguir \leftarrow false	$\mathcal{O}(1)$
73:	end if	

74: nroCamino \leftarrow nroCamino - 1	$\mathcal{O}(1)$
75: end while	
76: end if	

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:

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

IELIMINAR(in/out c: colaP, in a: nat, in b: nat)

1: var pNodo: puntero(nodo) \leftarrow c.raiz	$\mathcal{O}(1)$
2: ver seguir: bool \leftarrow true	$\mathcal{O}(1)$
3: while pNodo != NULL \wedge seguir == 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 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 != 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 != 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 != 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 $*(pNodo).izq \neq \text{NULL} \wedge *(pNodo).der \neq \text{NULL}$ then	$\mathcal{O}(1)$
40:	var tNodo: puntero(nodo) $\leftarrow pNodo.der$	$\mathcal{O}(1)$
41:	while $*(tNodo).izq \neq \text{NULL}$ 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 $*(tNodo).der \neq \text{NULL}$ 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 $*(pNodo).izq == \text{NULL} \wedge *(pNodo).der == \text{NULL}$ then	$\mathcal{O}(1)$
62:	if $*(*(pNodo).padre).izq == pNodo$ then	$\mathcal{O}(1)$
63:	$*(*(pNodo).padre).izq \leftarrow \text{NULL}$	$\mathcal{O}(1)$
64:	bNodo $\leftarrow *(pNodo).padre$	$\mathcal{O}(1)$
65:	delete pNodo	$\mathcal{O}(1)$
66:	else	
67:	$*(*(pNodo).padre).der \leftarrow \text{NULL}$	$\mathcal{O}(1)$
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 != NULL \wedge *(pNodo).der != NULL then	$\mathcal{O}(1)$
99:	var tNodo: puntero(nodo) \leftarrow pNodo.der	$\mathcal{O}(1)$
100:	while *(tNodo).izq != 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 != 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 != NULL do	$\mathcal{O}(\lfloor \log_2 N \rfloor + 1)$
126:	*(b).alt \leftarrow SETALTURA(b)	$\mathcal{O}(1)$
127:	if FACTORDEDESBALANCE(b) > 1 then	$\mathcal{O}(1)$
128:	*(b).alt \leftarrow ROTAR(HIJOMASALTO (HIJOMASALTO(b)), HIJOMASALTO(b), b)	$\mathcal{O}(1)$
129:	*(*(b).izq).alt \leftarrow SETALTURA(*(b).izq)	$\mathcal{O}(1)$
130:	*(*(b).der).alt \leftarrow SETALTURA(*(b).der)	$\mathcal{O}(1)$
131:	*(b).alt \leftarrow SETALTURA(*(b))	$\mathcal{O}(1)$
132:	end if	
133:	b \leftarrow *(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, in p1: puntero(nodo), in p2: puntero(nodo), in p3: puntero(nodo)) \rightarrow res : puntero(nodo)

1: var t1: puntero(nodo) \leftarrow NULL	$\mathcal{O}(1)$
2: var t2: puntero(nodo) \leftarrow NULL	$\mathcal{O}(1)$
3: var t2: puntero(nodo) \leftarrow NULL v	
4: if $(*(p3).pri \leq *(p1).pri \wedge *(p3).seg < *(p1).seg) \wedge$	
5: $*(p1).pri \leq *(p2).pri \wedge *(p1).pri < *(p2).pri)$ then	$\mathcal{O}(1)$
6: t1 \leftarrow p3	$\mathcal{O}(1)$
7: t2 \leftarrow p1	$\mathcal{O}(1)$
8: t3 \leftarrow p2	$\mathcal{O}(1)$
9: end if	
10: if $(*(p3).pri \geq *(p1).pri \wedge *(p3).seg > *(p1).seg) \wedge$	
11: $*(p1).pri \geq *(p2).pri \wedge *(p1).pri > *(p2).pri)$ then	$\mathcal{O}(1)$
12: t1 \leftarrow p2	$\mathcal{O}(1)$
13: t2 \leftarrow p1	$\mathcal{O}(1)$
14: t3 \leftarrow p3	$\mathcal{O}(1)$
15: end if	
16: if $(*(p3).pri \leq *(p2).pri \wedge *(p3).seg < *(p2).seg) \wedge$	
17: $*(p2).pri \geq *(p1).pri \wedge *(p2).pri < *(p1).pri)$ then	$\mathcal{O}(1)$
18: t1 \leftarrow p3	$\mathcal{O}(1)$
19: t2 \leftarrow p2	$\mathcal{O}(1)$
20: t3 \leftarrow p1	$\mathcal{O}(1)$
21: end if	
22: if $(*(p3).pri \geq *(p2).pri \wedge *(p3).seg > *(p2).seg) \wedge$	
23: $*(p2).pri \geq *(p3).pri \wedge *(p2).pri > *(p3).pri)$ then	$\mathcal{O}(1)$
24: t1 \leftarrow p1	$\mathcal{O}(1)$
25: t2 \leftarrow p2	$\mathcal{O}(1)$
26: t3 \leftarrow p3	$\mathcal{O}(1)$
27: end if	
28: if c.raiz == p3 then	
29: c.raiz \leftarrow p3	$\mathcal{O}(1)$
30: *(p3).padre \leftarrow NULL	$\mathcal{O}(1)$
31: else	
32: if $*(p3).padre.izq = p3$ then	$\mathcal{O}(1)$
33: CIZQ(*(p3).padre, t2)	$\mathcal{O}(1)$
34: else	
35: CDER(*(p3).padre, t2)	$\mathcal{O}(1)$
36: end if	
37: end if	
38: if $*(t2).izq \neq p1 \wedge *(t2).izq \neq p2 \wedge *(t2).izq \neq p3$ then	
39: CDER(t1, *(t2).izq)	$\mathcal{O}(1)$
40: end if	
41: if $*(t2).der \neq p1 \wedge *(t2).der \neq p2 \wedge *(t2).der \neq p3$ then	
42: CDER(t3, *(t2).der)	$\mathcal{O}(1)$
43: end if	
44: CIZQ(t2, t1)	$\mathcal{O}(1)$
45: CDER(t2, t3)	$\mathcal{O}(1)$
46: res \leftarrow t2	$\mathcal{O}(1)$

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

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

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

ICIZQ(**in** a: puntero(nodo), **in** b: puntero(nodo))

1: *(a).izq = b	$\mathcal{O}(1)$
2: *(b).padre = a	$\mathcal{O}(1)$

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))

1: $*(a).der = b$ $\mathcal{O}(1)$
2: $*(b).padre = a$ $\mathcal{O}(1)$

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

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

ISEALTURA(**in** a : puntero(nodo)) $\rightarrow res$: nat

1: **if** $*(a).izq == \text{NULL}$ **then** $\mathcal{O}(1)$
2: **if** $*(a).der == \text{NULL}$ **then** $\mathcal{O}(1)$
3: $res \leftarrow 1$ $\mathcal{O}(1)$
4: **else**
5: $res \leftarrow 1 + (*(a).der).alt$ $\mathcal{O}(1)$
6: **end if**
7: **else**
8: **if** $*(a).der == \text{NULL}$ **then** $\mathcal{O}(1)$
9: $res \leftarrow 1 + (*(a).izq).alt$ $\mathcal{O}(1)$
10: **else**
11: **if** $*(*(a).izq).alt > (*(a).der).alt$ **then** $\mathcal{O}(1)$
12: $res \leftarrow 1 + (*(a).izq).alt$ $\mathcal{O}(1)$
13: **else**
14: $res \leftarrow 1 + (*(a).der).alt$ $\mathcal{O}(1)$
15: **end if**
16: **end if**
17: **end if**

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

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

IFACTORDESBALANCE(**in** a : puntero(nodo)) $\rightarrow res$: int

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

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

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

IHIJOMASALTO(**in** a : puntero(nodo)) $\rightarrow res$: puntero(nodo)

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

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

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

IVACIA?(**in** c : colaP) $\rightarrow res$: bool

1: $res \leftarrow c.raiz \neq \text{NULL}$ $\mathcal{O}(1)$

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

IDESENCOLAR(**in/out** c : colaP) $\rightarrow res$: tupla(a : nat, b : nat)

1: $\text{var } pNodo: \text{puntero(nodo)} \leftarrow c.raiz$ $\mathcal{O}(1)$
2: **while** $pNodo.der \neq \text{NULL}$ **do** $\mathcal{O}(\log_2 N)$
3: $pNodo \leftarrow pNodo.der$ $\mathcal{O}(1)$
4: **end while**
5: **ELIMINAR**($c, *(pNodo).pri, *(pNodo).seg$) $\mathcal{O}(\log_2 N)$

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

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

ITAMAÑO(**in/out** c : colaP) $\rightarrow res$: nat

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

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