

Algoritmos y Estructuras de Datos II

Trabajo Práctico 2

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

Primer Cuatrimestre de 2015

Grupo 16

Apellido y Nombre	LU	E-mail
Fernando Frassia	340/13	ferfrassia@gmail.com
Rodrigo Seoane Quilne	910/11	seoane.raq@gmail.com
Sebastian Matias Giambastiani	916/12	sebastian.giambastiani@hotmail.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. Red	4
2.1. Auxiliares	5
2.2. Representacion	5
2.3. InvRep y Abs	5
2.4. Algoritmos	6
3. DCNet	9
3.1. Representacion	10
3.2. InvRep y Abs	10
3.3. Algoritmos	12
4. Diccionario String(α)	15
4.1. Representacion	15
4.2. InvRep y Abs	16
4.3. Algoritmos	16
5. DiccRapido	18
5.1. Representacion	18
5.2. InvRep y Abs	19
5.3. Algoritmos	19

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** prim(s) **else** elemDeSecu(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 restricciones(m)

restricciones(conectar(e_1, e_2, r, m)) \equiv 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 nroConexion(e_1, e_2, m)

2. Red

Interfaz

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

géneros: red, itConj(Compu).

Operaciones básicas de Red

COMPUTADORAS(in r : red) $\rightarrow res$: 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.

CONECTADAS?(in r : red, in c_1 : compu, in c_2 : compu) $\rightarrow res$: 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.

INTERFAZUSADA(in r : red, in c_1 : compu, in c_2 : compu) $\rightarrow res$: 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

INICIARRED() $\rightarrow res$: red

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

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

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

Descripción: Crea una red sin computadoras.

AGREGARCOMPUTADORA(in/out r : red, in c : 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.

CONECTAR(in/out r : red, in c_1 : compu, in i_1 : interfaz, in c_2 : compu, in i_2 : 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.

VECINOS(in r : red, in c : compu) $\rightarrow res$: itConj(compu)

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

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

Descripción: Devuelve todas las computadoras que están conectadas directamente con c

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

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

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

Descripción: Verifica que una computadora use una interfaz

CAMINOSMINIMOS(in r : red, in c_1 : compu, in c_2 : compu) $\rightarrow res$: conj(lista(compu))

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

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

Descripción: Devuelve todos los caminos minimos de conexiones entre una computadora y otra

HAYCAMINO?(in r : red, in c_1 : compu, in c_2 : compu) $\rightarrow res$: bool
Pre $\equiv \{\{c_1, c_2\} \subseteq computadoras(r)\}$
Post $\equiv \{res =_{\text{obs}} \text{hayCamino?}(r, c_1, c_2)\}$
Descripción: Verifica que haya un camino de conexiones entre una computadora y otra

2.1. Auxiliares

Operaciones auxiliares

CAMINOSMINIMOS(in r : red, in c_1 : compu, in c_2 : compu) $\rightarrow res$: conj(lista(compu))
Pre $\equiv \{\{c_1, c_2\} \subseteq computadoras(r)\}$
Post $\equiv \{res =_{\text{obs}} \text{caminosMinimos}(r, c_1, c_2)\}$
Descripción: Devuelve los caminos mínimos entre c_1 y c_2

2.2. Representacion

Representación

red se representa con e_red

donde e_red es tupla(vecinosEInterfaces: diccString(compu: string, tupla(interfaces: diccString(compu: string, interfaz: nat), computasVecinas: conj(compu))), deOrigenADestino: diccString(compu: string, diccString(compu: string, caminosMinimos: conj(lista(compu)))) , computadoras: conj(compu))

2.3. 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) $\equiv \text{true} \iff$

1.

2. 5.

3. 4.

3.

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

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

$\text{Abs} : \text{e_mapa } m \rightarrow \text{mapa}$ $\{\text{Rep}(m)\}$
 $\text{Abs}(m) =_{\text{obs}} p : \text{mapa} \mid$
 $\quad m.\text{estaciones} = \text{estaciones}(p) \wedge_L$
 $\quad (\forall e1, e2 : \text{string}) ((e1 \in \text{estaciones}(p) \wedge e2 \in \text{estaciones}(p)) \Rightarrow_L$
 $\quad (\text{conectadas?}(e1, e2, p) \iff$
 $\quad e1 \in \text{claves}(m.\text{uniones}) \wedge e2 \in \text{claves}(\text{obtener}(e2, m.\text{uniones})))) \wedge_L$
 $\quad (\forall e1, e2 : \text{string}) ((e1 \in \text{estaciones}(p) \wedge e2 \in \text{estaciones}(p)) \wedge_L$
 $\quad \text{conectadas?}(e1, e2, p) \Rightarrow_L$
 $\quad (\text{restriccion}(e1, e2, p) = m.\text{sendas}[\text{obtener}(e2, \text{obtener}(e1, m.\text{uniones})]) \wedge \text{nroConexion}(e1,$
 $\quad e2, m) = \text{obtener}(e2, \text{obtener}(e1, m.\text{uniones}))) \wedge \text{long}(\text{restricciones}(p)) = m.\#\text{sendas} \wedge_L (\forall$
 $\quad n : \text{nat}) (n < m.\#\text{sendas} \Rightarrow_L m.\text{sendas}[n] = \text{ElemDeSecu}(\text{restricciones}(p), n)))$

2.4. Algoritmos

Algoritmos

ICOMPUTADORAS(**in** $r : \text{red}$) $\rightarrow res : \text{itConj}(\text{Compu})$ $\mathcal{O}(1)$
 $1: res \leftarrow \text{CrearIt}(r.\text{computadoras})$
Complejidad: $\mathcal{O}(1)$

ICONECTADAS?(**in** $r : \text{red}$, **in** $c_1 : \text{compu}$, **in** $c_2 : \text{compu}$) $\rightarrow res : \text{bool}$ $\mathcal{O}(|c_1| + |c_2|)$
 $1: res \leftarrow \text{Definido?}(\text{Significado}(r.\text{vecinosEInterfaces}, c_1.ip).\text{interfaces}, c_2.ip)$
Complejidad: $\mathcal{O}(|c_1| + |c_2|)$

IINTERFAZUSADA(**in** $r : \text{red}$, **in** $c_1 : \text{compu}$, **in** $c_2 : \text{compu}$) $\rightarrow res : \text{interfaz}$ $\mathcal{O}(|c_1| + |c_2|)$
 $1: res \leftarrow \text{Significado}(\text{Significado}(r.\text{vecinosEInterfaces}, c_1.ip).\text{interfaces}, c_2.ip)$
Complejidad: $\mathcal{O}(|c_1| + |c_2|)$

IINICIARRED() $\rightarrow res : \text{red}$ $\mathcal{O}(1+1+1)$
 $1: res \leftarrow \text{tupla}(\text{vecinosEInterfaces: Vacío}(), \text{deOrigenADestino: Vacío}(), \text{computadoras: Vacío}())$
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** $c : \text{compu}$) $\mathcal{O}(1)$
 $1: \text{Agregar}(r.\text{computadoras}, c)$ $\mathcal{O}(|c|)$
 $2: \text{Definir}(r.\text{vecinosEInterfaces}, c.ip, \text{tupla}(\text{Vacío}(), \text{Vacío}()))$ $\mathcal{O}(|c|)$
 $3: \text{Definir}(r.\text{deOrigenADestino}, c.ip, \text{Vacío}())$ $\mathcal{O}(|c|)$
Complejidad: $\mathcal{O}(|c|)$
 $\mathcal{O}(1) + \mathcal{O}(|c|) + \mathcal{O}(|c|) =$
 $2 * \mathcal{O}(|c|) = \mathcal{O}(|c|)$

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

```

1: var  $tupSig1$ :tupla  $\leftarrow$  Significado( $r.vecinosEInterfaces$ ,  $c_1.ip$ )
2: Definir( $tupSig1.interfaces$ ,  $c_2.ip$ ,  $i_1$ )  $\mathcal{O}(|c_1| + |c_2| + 1)$ 
3: Agregar( $tupSig1.compusVecinas$ ,  $c_2$ )  $\mathcal{O}(1)$ 
4: var  $tupSig2$ :tupla  $\leftarrow$  Significado( $r.vecinosEInterfaces$ ,  $c_2.ip$ )
5: Definir( $tupSig2.interfaces$ ,  $c_1.ip$ ,  $i_2$ )  $\mathcal{O}(|c_1| + |c_2| + 1)$ 
6: Agregar( $tupSig2.compusVecinas$ ,  $c_1$ )  $\mathcal{O}(1)$ 
7: Definir(Significado( $r.deOrigenADestino$ ,  $c_1.ip$ ),  $c_2.ip$ , ArmarCaminosMinimos( $r$ ,  $c_1$ ,  $c_2$ ))  $\mathcal{O}(1)$  Definir(Significado( $r.deOrigenADestino$ ,  $c_2.ip$ ),  $c_1.ip$ , ArmarCaminosMinimos( $r$ ,  $c_2$ ,  $c_1$ ))  $\mathcal{O}(1)$ 

```

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

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

IVECINOS(in r : red, in c : compu) $\rightarrow res$: itConj(compu)

1: $res \leftarrow$ crearIt((Significado($r.vecinosEInterfaces$, $c.ip$)). $compusVecinas$)

Complejidad:

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

```

1: var  $tupVecinos$ :tupla  $\leftarrow$  Significado( $r.vecinosEInterfaces$ ,  $c.ip$ )  $\mathcal{O}(1)$ 
2: var  $itCompusVecinas$ :itConj(compu)  $\leftarrow$  CrearIt( $tupVecinos.compusVecinas$ )  $\mathcal{O}(1)$ 
3:  $res$ :bool  $\leftarrow$  false  $\mathcal{O}(1)$ 
4: while HaySiguiete( $itCompusVecinas$ ) AND  $\neg res$  do  $\mathcal{O}(1)$ 
5:   if Significado( $tupVecinos.interfaces$ , Siguiete( $itCompusVecinas$ ). $ip$ ) ==  $i$  then  $\mathcal{O}(1)$ 
6:      $res \leftarrow$  true  $\mathcal{O}(1)$ 
7:   end if
8:   Avanzar( $it$ )  $\mathcal{O}(1)$ 
9: end while

```

Complejidad:

ICAMINOSMINIMOS(in r : red, in c_1 : compu, in c_2 : compu) $\rightarrow res$: conj(lista(compu))

1: $res \leftarrow$ Significado(Significado($r.deOrigenADestino$, $c_1.ip$), $c_2.ip$) $\mathcal{O}(|c_1| + |c_2|)$

Complejidad:

IHAYCAMINO(in r : red, in c_1 : compu, in c_2 : compu) $\rightarrow res$: bool

```

1: var  $conjCaminosMinimos$   $\leftarrow$  CaminosMinimos( $r$ ,  $c_1$ ,  $c_2$ )  $\mathcal{O}(1)$ 
2:  $res \leftarrow$  EsVacio?( $conjCaminosMinimos$ )  $\mathcal{O}(1)$ 

```

Complejidad:

IARMARCAMINOS(in r : red, in c_1 : compu, in c_2 : compu) $\rightarrow res$: conj(lista)

```

1: var  $parcial$ :lista  $\leftarrow$  Vacía()  $\mathcal{O}(1)$ 
2: AgregarAtras( $parcial$ ,  $c_1$ )  $\mathcal{O}(1)$ 
3:  $res \leftarrow$  TODOSLOS CAMINOS( $r$ ,  $c_1$ ,  $c_2$ ,  $parcial$ )  $\mathcal{O}(1)$ 

```

Complejidad:

iTodosLosCaminos(in r : red, in c_1 : compu, in c_2 : compu, in $pacial$: lista(compu)) $\rightarrow res$: conj(lista)	
1: $res \leftarrow \text{Vacio}()$	$\mathcal{O}(1)$
2: if Pertenece?(Vecinos(r , c_1), c_2) then	$\mathcal{O}(1)$
3: AgregarAtras($pacial$, c_2)	$\mathcal{O}(1)$
4: Agregar(res , $pacial$)	$\mathcal{O}(1)$
5: else	
6: var $itVecinos$:itConj \leftarrow CrearIt(Vecinos(r , c_1))	$\mathcal{O}(1)$
7: while HaySiguiente?($itVecinos$) do	$\mathcal{O}(1)$
8: if \neg Pertenece?($pacial$, Siguiente($itVecinos$)) then	$\mathcal{O}(1)$
9: var $auxParcial$:lista \leftarrow $pacial$	$\mathcal{O}(1)$
10: AgregarAtras($auxParcial$, Siguiente($itVecinos$))	$\mathcal{O}(1)$
11: Unir(res , TodosLosCaminos(r , Siguiente($itVecinos$), c_2 , $auxParcial$))	$\mathcal{O}(1)$
12: end if	
13: Avanzar($itVecinos$)	$\mathcal{O}(1)$
14: end while	
15: end if	

Complejidad:

3. DCNet

Interfaz

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

géneros: dcnet.

Operaciones básicas de DCNet

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

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

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

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

Descripción: Devuelve la red del dcnet.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Complejidad: $\mathcal{O}()$

Descripción: Agrega el paquete al dcnet.

AVANZARSEGUNDO(**in/out** d : dcnet)

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

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

Complejidad: $\mathcal{O}()$

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

Operaciones del iterador

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

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

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

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

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

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

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

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

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

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

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

it se representa con e_it

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

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

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

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

{Rep(u)}

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

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

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

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

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

3.3. Algoritmos

Algoritmos

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

1: $res \leftarrow (d.red)$

$\mathcal{O}(1)$

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

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

1: var $it \leftarrow computadoras(d.red)$

$\mathcal{O}(1)$

2: **while** HaySiguiete(it) **do**

$\mathcal{O}(1)$

3: **if** definido?($p.id, significado(Siguiete(it), CompusYPaquetes.d).PaquetesYCamino$) **then**

4: $res \leftarrow significado(p.id, significado(Siguiete(it), CompusYPaquetes.d).PaquetesYCamino).CaminoRecorrido$

5: **end if**

6: $Avanzar(it)$

$\mathcal{O}(1)$

7: **end while**

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

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

1: $res \leftarrow Significado(c, d.CompusYPaquetes).Enviados$

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

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

EN ESPERA(in $d: dcnet$, in $c: compu$) $\rightarrow res : itPaquete$

1: $res \leftarrow claves(Significado(c, d.CompusYPaquetes).PaquetesYCamino)$

$\mathcal{O}(1)$

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

INICIAR DCNET(in $r: red$, in/out $d: dcnet$)

1:

$\mathcal{O}(1)$

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

ICREARPAQUETE(in p : rur, in/out d : dcnet)

1:

$\mathcal{O}(1)$

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

IAVANZARSEGUNDO(in/out d : dcnet)

1:

$\mathcal{O}(1)$

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 $it: itConj(\text{Estacion}) \leftarrow \text{Estaciones}(m)$

$\mathcal{O}(1)$

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

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

2: $c.RURs[rur].esta? \leftarrow false$

$\mathcal{O}(1)$

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

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

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

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

$\mathcal{O}(1)$

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

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

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

$\mathcal{O}(1)$

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

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

1: $it.i++$

$\mathcal{O}(1)$

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

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

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

$\mathcal{O}(1)$

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"

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

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

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

4.3. Algoritmos

Algoritmos

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

1: $res \leftarrow iNodoVacío()$

$\mathcal{O}(1)$

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

$iNODOVACÍO() \rightarrow res : e_nodo$

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

$\mathcal{O}(1)$

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

$\mathcal{O}(1)$

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

$\mathcal{O}(1)$

4: **end for**

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

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

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

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

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

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

$\mathcal{O}(1)$

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

$\mathcal{O}(1)$

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

$\mathcal{O}(1)$

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

$\mathcal{O}(1)$

5: 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. DiccRapido

Interfaz

se explica con: DICCIONARIO(CLAVE, SIGNIFICADO).

géneros: diccRapido.

Operaciones básicas de DICC RAPIDO

DEF?(in c : clave, in d : diccRapido) $\rightarrow res$: bool
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \text{def?}(c, d)\}$
Complejidad: $\mathcal{O}(\log_2 n)$, siendo n la cantidad de claves
Descripción: Verifica si una clave está definida.

OBTENER(in c : clave, in d : diccRapido) $\rightarrow res$: significado
Pre $\equiv \{\text{def?}(c, d)\}$
Post $\equiv \{res =_{\text{obs}} \text{obtener}(c, d)\}$
Complejidad: $\mathcal{O}(\log_2 n)$, siendo n la cantidad de claves
Descripción: Devuelve el significado asociado a una clave

VACÍO() $\rightarrow res$: diccRapido
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \text{vacío}()\}$
Complejidad: $\mathcal{O}(1)$
Descripción: Crea un nuevo diccionario vacío

DEFINIR(in c : clave, in s : significado, in/out d : diccRapido)
Pre $\equiv \{d =_{\text{obs}} d_0\}$
Post $\equiv \{d =_{\text{obs}} \text{definir}(c, s, d_0)\}$
Complejidad: $\mathcal{O}(\log_2 n)$, siendo n la cantidad de claves
Descripción: Define la clave, asociando su significado, al diccionario

BORRAR(in c : clave, in/out d : diccRapido)
Pre $\equiv \{d =_{\text{obs}} d_0 \wedge \text{def?}(c, d_0)\}$
Post $\equiv \{d =_{\text{obs}} \text{borrar}(c, d_0)\}$
Complejidad: $\mathcal{O}(\log_2 n)$, siendo n la cantidad de claves
Descripción: Borra la clave del diccionario

CLAVES(in d : diccRapido) $\rightarrow res$: itPaquete
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \text{claves}(d)\}$
Complejidad: $\mathcal{O}(1)$
Descripción: Devuelve un iterador de paquete

5.1. Representacion

Representación

Para representar el diccionario, elegimos hacerlo sobre AVL. Sabiendo que la cantidad de claves no está acotada, este AVL estará representado con nodos y punteros. Cabe destacar, que las claves del diccionario deben contener una relación de orden. Las claves y los significados se pasan por referencia.

diccRapido se representa con estr

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

donde **nodo** es tupla(*clave*: clave, *significado*: significado, *padre*: puntero(nodo), *izq*: puntero(nodo), *der*: puntero(nodo), *alt*: nat)

5.2. InvRep y Abs

5.3. Algoritmos

Algoritmos

```
IDEF?(in c: clave, in d: diccRapido) → res : bool
1: var pNodo: puntero(nodo) ← d.raiz                                 $\mathcal{O}(1)$ 
2: while *(pNodo) != NULL do                                        $\mathcal{O}(\log_2 n)$ 
3:   if *(pNodo).clave == c then                                     $\mathcal{O}(1)$ 
4:     res ← true                                                   $\mathcal{O}(1)$ 
5:     return res                                                   $\mathcal{O}(1)$ 
6:   else
7:     if c > *(pNodo).clave then                                     $\mathcal{O}(1)$ 
8:       pNodo ← *(pNodo).der                                        $\mathcal{O}(1)$ 
9:     else
10:      pNodo ← *(pNodo).izq                                        $\mathcal{O}(1)$ 
11:    end if
12:  end if
13: end while
14: res ← false                                                     $\mathcal{O}(1)$ 

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

```
IOBTENER(in c: clave, in d: diccRapido) → res : significado
1: var pNodo: puntero(nodo) ← d.raiz                                 $\mathcal{O}(1)$ 
2: while *(pNodo).clave != c do                                        $\mathcal{O}(\log_2 n)$ 
3:   if c > *(pNodo).clave then                                     $\mathcal{O}(1)$ 
4:     pNodo ← *(pNodo).der                                        $\mathcal{O}(1)$ 
5:   else
6:     pNodo ← *(pNodo).izq                                        $\mathcal{O}(1)$ 
7:   end if
8: end while
9: res ← *(pNodo).significado                                        $\mathcal{O}(1)$ 

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

```
IVACÍO() → res : diccRapido
1: var res: diccRapido ← tupla(NULL, 0)                                 $\mathcal{O}(1)$ 

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

```
IDEFINIR(in c: clave, in s: significado, in/out d: diccRapido)
1:
2: if d.raiz == NULL then                                             $\mathcal{O}(1)$ 
3:   d.raiz ← &tupla(c, s, NULL, NULL, NULL, 1)                     $\mathcal{O}(1)$ 
4:   d.tam ← 1                                                        $\mathcal{O}(1)$ 
5: else
6:   if Def?(c, d) then                                              $\mathcal{O}(\log_2 n)$ 
7:     var pNodo: puntero(nodo) ← d.raiz                             $\mathcal{O}(1)$ 
8:     while *(pNodo).clave != c do                                   $\mathcal{O}(\log_2 n)$ 
9:       if c > *(pNodo).clave then                                   $\mathcal{O}(1)$ 
10:        pNodo ← *(pNodo).der                                        $\mathcal{O}(1)$ 
```

```

11:         else
12:             pNodo ← *(pNodo).izq  $\mathcal{O}(1)$ 
13:         end if
14:     end while
15:     *(pNodo).significado ← s  $\mathcal{O}(1)$ 
16: else
17:     var seguir: bool ← true  $\mathcal{O}(1)$ 
18:     var pNodo: puntero(nodo) ← d.raiz  $\mathcal{O}(1)$ 
19:     var camino: arreglo[ $\lfloor \log_2 (d.tam) \rfloor + 1$ ] de puntero(nodo)  $\mathcal{O}(\lfloor \log_2 (d.tam) \rfloor + 1)$ 
20:     var nroCamino: nat ← 0  $\mathcal{O}(1)$ 
21:     camino[0] ← pNodo  $\mathcal{O}(1)$ 
22:     while seguir == true do
23:         if c > *(pNodo).clave ∧ *(pNodo).der == NULL then  $\mathcal{O}(1)$ 
24:             if *(pNodo).izq == NULL then
25:                 *(pNodo).alt ← 2  $\mathcal{O}(1)$ 
26:             else
27:                 end if
28:                 *(pNodo).der ← &tupla(c, s, pNodo, NULL, NULL, 1)  $\mathcal{O}(1)$ 
29:                 nroCamino ← nroCamino + 1  $\mathcal{O}(1)$ 
30:                 camino[nroCamino] ← *(pNodo).der  $\mathcal{O}(1)$ 
31:                 seguir ← false  $\mathcal{O}(1)$ 
32:             else
33:                 if c > *(pNodo).clave ∧ *(pNodo).der != NULL then  $\mathcal{O}(1)$ 
34:                     if *(pNodo).izq == NULL then  $\mathcal{O}(1)$ 
35:                         *(pNodo).alt ← *(pNodo).alt + 1  $\mathcal{O}(1)$ 
36:                     else
37:                         *(pNodo).alt ← max(*(pNodo).izq).alt, *(pNodo).der).alt + 1)  $\mathcal{O}(1)$ 
38:                     end if
39:                     pNodo ← *(pNodo).der  $\mathcal{O}(1)$ 
40:                     nroCamino ← nroCamino + 1  $\mathcal{O}(1)$ 
41:                     camino[nroCamino] ← *(pNodo).der  $\mathcal{O}(1)$ 
42:                 else
43:                     end if
44:                 end if
45:             end while
46:         end if
47:     end if

```

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

IROTARSIMPLEIZQ()

1:

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

IROTARSIMPLEDER()

1:

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

IROTARDOBLEIZQ()

1:

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

IRotarDobleDer()

1:

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