



DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2: Diseño

Primer cuatrimestre - 2015

Algoritmos y Estructuras de Datos II

Grupo 2

Integrante	LU	Correo electrónico
Benitez, Nelson	945/13	nelson.benitez92@gmail.com
Roizman, Violeta	273/11	violeroizman@gmail.com
Vázquez, Jérica	318/13	jesis_93@hotmail.com
Zavalla, Agustín	670/13	nkm747@gmail.com

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		



Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ciudad Universitaria – Pabellón I (Planta Baja)

Intendente Güiraldes 2160 – C1428EGA

Ciudad Autónoma de Buenos Aires – Rep. Argentina

Tel/Fax: (+54 +11) 4576-3300

<http://www.exactas.uba.ar>

Índice

1. DCNet	2
1.1. Interfaz	2
1.2. Representación	3
1.3. Servicios Usados	3
2. ConjLog	4
2.1. Interfaz	4
2.2. Representación	4
2.3. Invariante de representación	5
2.4. Función de abstracción	5
2.5. Algoritmos	6
2.6. Auxiliares	9

1 DCNet

Una DCNet es

1.1 Interfaz

se explica con `DCNET`

usa `Compu`, `Paquete`, `Red`, `diccPref`, `conjLog`, `conjLogP`

géneros `dcnet`

Operaciones

`CREARSISTEMA(in r : red) → res : dcnet`

Pre $\equiv \{true\}$

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

Descripción: Crea un sistema DCNet.

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

Aliasing:

`CREARPAQUETE(in/out s : dcnet, in p : paquete)`

Pre $\equiv \{s =_{\text{obs}} s_0 \wedge \text{FALTACHOCLO}\}$

Post $\equiv \{s =_{\text{obs}} \text{crearPaquete}(s_0, p)\}$

Descripción: Crea un paquete y lo agrega a la computadora correspondiente.

Complejidad: $O(L + \log(k))$

Aliasing:

`AVANZARSEGUNDO(in/out s : dcnet)`

Pre $\equiv \{s =_{\text{obs}} s_0\}$

Post $\equiv \{s =_{\text{obs}} \text{avanzarSegundo}(s_0)\}$

Descripción: Avanza un segundo el sistema. Todas las computadoras envían su respectivo paquete y en consecuencia se actualizan los paquetes en espera de cada una de ellas.

Complejidad: $O(n \times (L + \log(n) + \log(k)))$

Aliasing:

`DAMERED(in s : dcnet) → res : red`

Pre $\equiv \{true\}$

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

Descripción: Devuelve la red de DCNet.

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

Aliasing:

`CAMINORECORRIDO(in s : dcnet, in p : paquete) → res : secu(compu)`

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

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

Descripción: Devuelve el camino recorrido hasta el momento por un paquete.

Complejidad: $O(n \times \log(\max(n, k)))$

Aliasing:

`CANTIDADENVIADOS(in s : dcnet, in c : compu) → res : nat`

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

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

Descripción: Devuelve la cantidad de paquetes enviados por una computadora.

Complejidad: $O(n)$

Aliasing:

ENESPERA(**in** $s : \text{dcnet}$, **in** $c : \text{compu}$) $\longrightarrow res : \text{puntero}(\text{conjLogP}(\text{paquete}))$

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

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

Descripción: Devuelve un iterador a los paquetes de la computadora.

Complejidad: $O(L)$

Aliasing:

LAQUEMASENVIO(**in** $s : \text{dcnet}$) $\longrightarrow res : \text{compu}$

Pre $\equiv \{true\}$

Post $\equiv \{res =_{\text{obs}} \text{laQueMasEnvio}(s, p)\}$

Descripción: Devuelve la computadora que más paquetes envió.

Complejidad: $O(1)$

Aliasing:

Las complejidades están en función de las siguientes variables:

n : la cantidad total de computadoras que hay en el sistema,

L : el hostname más largo de todas las computadoras,

k : la cola de paquetes más larga de todas las computadoras.

$\neg \emptyset$

1.2 Representación

se representa con sistema

donde sistema es $\text{tupla}(\text{Compus} : \text{arreglo}(\text{tupla}(\text{IP} : \text{String}, \text{pN} : \text{puntero}(\text{conjLog}(\text{paquete})), \text{\#Paquetes} : \text{nat}, \text{Num} : \text{nat}), \text{CompusPorPref} : \text{diccPref}(\text{compu}, \text{tupla}(\text{PorNom} : \text{conjLog}(\text{paquete}), \text{PorPrior} : \text{conjLog}(\text{paquete})), \text{CaminosMinimos} : \text{arreglo}(\text{arreglo}(\text{arreglo}(\text{compu}))), \text{LaQMasEnvio} : \text{nat})$

Invariante de representación

1.

Algoritmos

ICREARSISTEMA(**in** $r : \text{red}$) $\longrightarrow res : \text{dcnet}$

$n \leftarrow \#(\text{COMPUS}(\text{red}))$ $O(\#\text{compus}(\text{red})=n)?$

$i \leftarrow 0$

$j \leftarrow 0$

$O(1)$

$res.\text{Compus} \leftarrow \text{CREARARREGLO}(n)$

$O(n)$

$res.\text{CaminosMinimos} \leftarrow \text{CREARARREGLO}(n)$

$O(n)$

var $p : \text{arreglo.dimensionable de puntero}(\text{conjLog}(\text{paquete}))$

while $i < n$ **do**

$O(n)$

$res.\text{CaminosMinimos}[i] \leftarrow \text{CREARARREGLO}(n)$

$O(n)$

$p[i] \leftarrow NULL$	$O(1)$
$res.Comp[us][i] \leftarrow \mathbf{tupla} \langle compu(r, i), p[i], 0, 0 \rangle$	
no se como se deben escribir las tuplas	$O(1)$
$\mathbf{DEFINIR}(res.Comp[us]PorPref, compu(r, i))$	$O(L)$
while $j < n$ do	$O(n)$
$res.CaminosMinimos[i][j] \leftarrow caminoMinimo(compu(r, i), compu(r, j), r)$	$O(\text{complejidad cammin}(\text{red}))$
$j++$	
end while	
$i++$	
end while	
$res.LaQM[asEnvio] \leftarrow 0$	$O(1)$
<hr/>	
$\mathbf{ICREARPAQUETE}(\mathbf{in/out } s : \mathbf{dcnet}, \mathbf{in } p : \mathbf{paquete})$	
$t : \langle conjLog(paquete), conjLog(paquete) \rangle$	
$t \leftarrow \mathbf{OBTENER}(\pi_3(p), s.Comp[us]PorPref)$	$O(L)$
$\mathbf{INSERTAR}(\pi_1(t), p)$	$O(\log(k))$
$\mathbf{INSERTAR}(\pi_2(t), p)$	$O(\log(k))$
<hr/>	
	$O(L + \log(k))$
$\mathbf{ILAQUEMASENVIO}(\mathbf{in } s : \mathbf{dcnet}) \longrightarrow res : \mathbf{compu}$	
$res \leftarrow \pi_1(s.comp[us][s.LaQM[asEnvio]])$	$O(1)$
<hr/>	
	$O(1)$
$\mathbf{IENESPERA}(\mathbf{in } s : \mathbf{dcnet}, \mathbf{in } c : \mathbf{compu}) \longrightarrow res : \mathbf{puntero}(conjLogP(paquete))$	
$t : \langle conjLog(paquete), conjLog(paquete) \rangle$	
$t \leftarrow \mathbf{OBTENER}(\pi_1(c), s.Comp[us]PorPref)$	$O(L)$
$res \leftarrow \&(\pi_2(t))$	$O(1)$
<hr/>	
	$O(L)$
$\mathbf{IAVANZARSEGUNDO}(\mathbf{in/out } s : \mathbf{dcnet})$	
var $i : \mathbf{nat}$	
$i \leftarrow 0$	$O(1)$
var $m : \mathbf{nat}$	
$m \leftarrow s.LaQM[asEnvio]$	
while $i < \mathbf{LONGITUD}(s.comp[us])$ do	$O(n)$
if $\neg \mathbf{VACIA}?$ then	
var $IP : \mathbf{String}$	
$IP \leftarrow \pi_1(s.comp[us][i])$	
$t_1 : \langle conjLog(paquete), conjLog(paquete) \rangle$	
$t_1 \leftarrow \mathbf{OBTENER}(IP, s.COMP[us]PORPREF)$	$O(L)$
var $p : \mathbf{paquete}$	
$p \leftarrow \mathbf{SACARMAX}(\pi_2(t_1))$	$O(\log(k))$
$\mathbf{BORRAR}(\pi_2(t_1), p)$	$O(\log(k))$
$\mathbf{BORRAR}(\pi_1(t_1), p)$	$O(\log(k))$
$\pi_3(s.comp[us][i]) \leftarrow \pi_3(s.comp[us][i]) + 1$	$O(1)$
$t_2 : \langle conjLog(paquete), conjLog(paquete) \rangle$	
$t_2 \leftarrow \mathbf{OBTENER}(\pi_4(P), s.Comp[us]PorPref)$	$O(L)$
$\mathbf{INSERTAR}(\pi_2(t_2), p)$	
$\mathbf{INSERTAR}(\pi_1(t_2), p)$	$O(\log(k))$
	$O(\log(k))$

if $\pi_3(s.compus[i] > max)$ then $max \leftarrow i$ end if end if $i \leftarrow i + 1$ end while $s.LaQMasEnvio \leftarrow max$	$O(1)$ $O(1)$ $O(1)$
<hr/>	
$O(n \times (L + \log(k)))$	
<hr/>	
ICANTIDADENVIADOS(in/out s : dcnet, in c : compu) \longrightarrow res : nat var $i : \text{nat}$ $i \leftarrow 0$ while $\pi_1(s.compus[i]) \neq \pi_1(c)$ do $i \leftarrow i + 1$ end while $res \leftarrow \pi_3(s.compus[i])$	 $O(1)$ $O(n)$ $O(1)$ $O(1)$
<hr/>	
$O(n)$	
<hr/>	
ICAMINORECORRIDO(in s : dcnet, in p : paquete) \longrightarrow res : secu(compu) var $i : \text{nat}$ $i \leftarrow 0$ var $b : \text{bool}$ $b \leftarrow \neg(\text{PERTENECE?}(p, \pi_3(s.compus[i])))$ while b do $i \leftarrow i + 1$ $b \leftarrow \neg(\text{PERTENECE?}(p, \pi_3(s.compus[i])))$ end while var $j : \text{nat}$ $j \leftarrow 0$ while $\pi_1(s.compus[j]) \neq \pi_3(p)$ do $i \leftarrow j + 1$ end while $res \leftarrow s.CaminosMinimos[j][\pi_4(s.compus[i])]$	 $O(1)$ $O(\log(k))$ $O(n)$ $O(1)$ $O(\log(k))$ $O(1)$ $O(n)$ $O(1)$
<hr/>	
$O(1 \text{ o } n \text{ dependiendo de si hago copia o no})$ $O(n \times \log(k))$	

1.3 Servicios Usados

2 ConjLog

2.1 Interfaz

se explica con $\text{CONJ}(\alpha)$
géneros $\text{conjLog}(\alpha)$

Operaciones

$\text{NUEVO}() \longrightarrow res : \text{conjLog}(\alpha)$

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

Post $\equiv \{res =_{\text{obs}} \emptyset\}$

Descripción: Crea un nuevo conjLog vacío

Complejidad: $O(1)$

$\text{ESTÁ}(\text{in } cl : \text{conjLog}(\alpha), \text{in } e : \alpha) \longrightarrow res : \text{bool}$

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

Post $\equiv \{res = (e \in cl)\}$

Descripción: Retorna un booleano que indica si el elemento pertenece al conjunto

Complejidad: $O(\log(\#(cl)))$

$\text{INSERTAR}(\text{in/out } cl : \text{cl}(\alpha), \text{in } e : \alpha)$

Pre $\equiv \{cl_0 =_{\text{obs}} cl \wedge \neg(e \in cl)\}$

Post $\equiv \{cl_0 =_{\text{obs}} \text{Agregar}(cl_0, e)\}$

Descripción: Inserta un nuevo elemento en el conjunto

Complejidad: $O(\log(\#(cl)))$

$\text{BORRAR}(\text{in/out } cl : \text{cl}(\alpha), \text{in } e : \alpha)$

Pre $\equiv \{cl_0 =_{\text{obs}} cl \wedge (e \in cl)\}$

Post $\equiv \{cl =_{\text{obs}} (cl_0 - \{e\})\}$

Descripción: Elimina el elemento e del conjunto cl , los iteradores que apunten a este elemento se indefinen

Complejidad: $O(\log(\#(cl)))$

2.2 Representación

se representa con clog

donde clog es $\text{raiz} : \text{puntero}(\text{nodo})$

donde nodo es $\text{tupla}(\text{der} : \text{puntero}(\text{nodo}),$
 $\text{izq} : \text{puntero}(\text{nodo}),$
 $\text{valor} : \alpha,$
 $\text{padre} : \text{puntero}(\text{nodo}),$
 $\text{fdb} : \text{nat})$

2.3 Invariante de representación

1. Para todas las raíces, la altura del subárbol derecho menos la altura del subárbol izquierdo de esa raíz es igual al fdb.
2. El fdb de todas las raíces es 0, 1 o -1.
3. Si un nodo no es una hoja del árbol entonces los padres de los hijos derecho e izquierdo son iguales y es el nodo
4. Si un nodo es una hoja del árbol entonces los hijos derecho e izquierdo del árbol son NULL
5. Para todos los nodos n, todos los nodos del subárbol derecho son mayores que n
6. Para todos los nodos n, todos los nodos del subárbol izquierdo son menores que n
7. No hay nodos repetidos
8. El padre de la raíz es NULL

2.4 Función de abstracción

$$\text{Abs} : \widehat{\text{clog}(\alpha)} \text{ } cl \longrightarrow \widehat{\text{conj}(\alpha)} \quad \{\text{Rep}(cl)\}$$

$$(\forall cl : \widehat{\text{clog}(\alpha)})$$

$$\text{Abs}(cl) \equiv c : \widehat{\text{conj}(\alpha)} \mid$$

$$((\forall e : \alpha) e \in c \Rightarrow_L \text{esta}(cl, e)) \wedge \text{size}(cl) = \#(c)$$

2.5 Algoritmos

IBORRAR (in/out $cl : \text{conjLog}(\alpha)$, $in\ e : \alpha$)	
$clactual \leftarrow cl$	$O(1)$
if ($\neg(cl.der == NULL) \wedge \neg(cl.izq == NULL)$) then	$O(1)$
$clactual \leftarrow \text{IENCONTRARPADRE}(clactual, e)$	$O(\log(\text{size}(cl)))$
if ($clactual.der.valor == e$) then	$O(1)$
$mm \leftarrow \text{IDAMEMAYMENORES}(clactual.der.izq)$	$O(\log(\text{size}(cl)))$
$padremm \leftarrow mm.padre$	$O(1)$
if $padremm.valor == e$ then	
$padremm.fdb ++$	$O(1)$
$mm.fdb \leftarrow clactual.der.fdb$	$O(1)$
$mm.der \leftarrow clactual.der.der$	$O(1)$
$mm.padre \leftarrow clactual$	$O(1)$
$mm.padre.der \leftarrow mm$	$O(1)$
$mm.der.padre \leftarrow mm$	$O(1)$, Si no son NULL
$mm.izq.padre \leftarrow mm$	$O(1)$, Si no son NULL
$\text{IREBYRECALCFDB}(padremm, false, true)$	$O(\log(\text{size}(cl)))$
else	
$padremm.fdb --$	$O(1)$
$mm.fdb \leftarrow clactual.der.fdb$	$O(1)$
$mm.der \leftarrow clactual.der.der$	$O(1)$
$mm.izq \leftarrow clactual.der.izq$	$O(1)$
$mm.padre \leftarrow clactual$	$O(1)$
$mm.padre.der \leftarrow mm$	$O(1)$
$mm.der.padre \leftarrow mm$	$O(1)$, Si no son NULL
$mm.izq.padre \leftarrow mm$	$O(1)$, Si no son NULL
$\text{IREBYRECALCFDB}(padremm, true, true)$	$O(\log(\text{size}(cl)))$
end if	
else	
$mm \leftarrow \text{IDAMEMAYMENORES}(clactual.izq.izq)$	$O(\log(\text{size}(cl.izq.izq)))$
$padremm \leftarrow mm.padre$	$O(1)$
if $padremm.valor == e$ then	
$padremm.fdb ++$	$O(1)$
$mm.fdb \leftarrow clactual.izq.fdb$	$O(1)$
$mm.der \leftarrow clactual.izq.der$	$O(1)$
$mm.padre \leftarrow clactual$	$O(1)$
$mm.padre.izq \leftarrow mm$	$O(1)$
$mm.der.padre \leftarrow mm$	$O(1)$, Si no son NULL
$mm.izq.padre \leftarrow mm$	$O(1)$, Si no son NULL
$\text{REBYRECALCFDB}(padremm, false, true)$	$O(\log(\text{size}(cl)))$
else	
$padremm.fdb --$	$O(1)$
$mm.fdb \leftarrow clactual.der.fdb$	$O(1)$
$mm.der \leftarrow clactual.der.der$	$O(1)$
$mm.izq \leftarrow clactual.der.izq$	$O(1)$
$mm.padre \leftarrow clactual$	$O(1)$
$mm.padre.der \leftarrow mm$	$O(1)$

$mm.der.padre \leftarrow mm$	$O(1)$, Si no son NULL
$mm.izq.padre \leftarrow mm$	$O(1)$, Si no son NULL
$I\text{REBYRECALCFDB}(padremm, true, true)$	$O(\log(size(cl)))$
end if	
end if	
else	
if $cl.der == NULL \wedge cl.izq == NULL$ then	$O(1)$
$res \leftarrow NULL$	$O(1)$
else	
if $cl.der == NULL$ then	$O(1)$
if $cl.izq.valor == e$ then	$O(1)$
$cl.izq \leftarrow NULL$	$O(1)$
else	
$cl.valor \leftarrow cl.izq.valor$	$O(1)$
$cl.izq \leftarrow NULL$	$O(1)$
end if	
else	
if $cl.der.valor == e$ then	$O(1)$
$cl.der \leftarrow NULL$	$O(1)$
else	
$cl.valor \leftarrow cl.der.valor$	$O(1)$
$cl.der \leftarrow NULL$	$O(1)$
end if	
end if	
end if	
end if	
	<hr/>
	$O(\log(size(cl)))$
$I\text{INSERTAR}(\text{in/out } cl : \text{conjLog}(\alpha), \text{ in } e : \alpha)$	
if $\neg(cl.der == NULL) \wedge \neg(cl.izq == NULL)$ then	$O(1)$
$clactual \leftarrow I\text{ENCONTRARPADRE}(clactual, e)$	$O(\log(size(cl)))$
if $clactual.valor < e$ then	
$clactual.der \leftarrow \text{tupla}\langle der : NULL,$	$O(1)$
$izq : NULL,$	
$valor : e,$	
$padre : clactual,$	
$fdb : 0 \rangle$	
$I\text{REBYRECALCFDB}(clactual, true, false)$	
else	
$clactual.izq \leftarrow \text{tupla}\langle der : NULL,$	$O(1)$
$izq : NULL,$	
$valor : e,$	
$padre : clactual,$	
$fdb : 0 \rangle$	
$I\text{REBYRECALCFDB}(clactual, false, false)$	
end if	
else	

if $cl.der == NULL \wedge cl.izq == NULL$ then	$O(1)$
$cl \leftarrow \text{tupla}(\text{der} : NULL,$	$O(1)$
$izq : NULL,$	
$valor : e,$	
$padre : clactual,$	
$fdb : 0)$	
else	
if $cl.der \neq NULL$ then	
$cl.izq \leftarrow \text{tupla}(\text{der} : NULL,$	$O(1)$
$izq : NULL,$	
$valor : e,$	
$padre : cl,$	
$fdb : 0)$	
else	
$cl.der \leftarrow \text{tupla}(\text{der} : NULL,$	$O(1)$
$izq : NULL,$	
$valor : e,$	
$padre : cl,$	
$fdb : 0)$	
end if	
end if	
end if	
<hr/>	
$I\acute{E}ST\acute{A}(\text{in/out } cl : \text{conjLog}(\alpha), \text{ in } e : \alpha) \longrightarrow res : \text{bool}$	$O(\log(\text{size}(cl)))$
$encontrado? \leftarrow false$	$O(1)$
$clactual \leftarrow cl$	$O(1)$
while $(clactual \neq NULL) \wedge \neg(encontrado?)$ do	$O(1)$
if $e > clactual.valor$ then	$O(1)$
$clactual \leftarrow clactual.der$	$O(1)$
else	
if $e < clactual.valor$ then	$O(1)$
$clactual \leftarrow clactual.izq$	$O(1)$
else	
$encontrado? \leftarrow true$	$O(1)$
end if	
end if	
end while	
$clactual \leftarrow NULL$	$O(1)$
$res \leftarrow encontrado?$	$O(1)$

2.6 Auxiliares

IREBYRECALCFDB(**in/out** $cl : \text{conjLog}(\alpha)$, *in borroInsertoHijoDerecho?* : bool, *in estoyBorrando?* : bool)

```

while  $cl! = \text{NULL} \wedge_L cl.fdb! = 2 \wedge cl.fdb! = -2$  do                                O(1)

    if estoyBorrando? then                                                        O(1)

        if borroInsertoHijoDerecho? then                                          O(1)
             $cl.fdb --$                                                             O(1)
        else
             $cl.fdb ++$                                                             O(1)
        end if
         $borroInsertoHijoDerecho? \leftarrow (cl.padre.der! = \text{NULL} \wedge_L cl.padre.der.valor == cl.valor)$ 
                                                                                   O(1)
         $cl \leftarrow cl.padre$                                                     O(1)
    else

        if borroInsertoHijoDerecho? then                                          O(1)
             $cl.fdb ++$                                                             O(1)
        else
             $cl.fdb --$                                                             O(1)
        end if
         $borroInsertoHijoDerecho? \leftarrow (cl.padre.izq! = \text{NULL} \wedge_L cl.padre.izq.valor == cl.valor)$ 
                                                                                   O(1)
         $cl \leftarrow cl.padre$                                                     O(1)
    end if
end while

if  $cl! = \text{NULL}$  then                                                            O(1)
    IROTAR( $cl$ )                                                                    O(1)
end if

```

IROTAR(**in/out** $cl : \text{conjLog}(\alpha)$)

```

if  $cl.fdb == +2$  then                                                            O(1)

    if  $cl.der.fdb == +1$  then                                                    O(1)
         $q \leftarrow cl.der$ 
         $izqp \leftarrow cl.izq$ 
         $cl.der \leftarrow q.der$ 
         $q.der.padre \leftarrow cl$ 
         $cl.izq \leftarrow \text{tupla}(\text{der} : \text{NULL},$ 
                                 $izq : \text{NULL},$ 
                                 $valor : cl.valor,$ 
                                 $padre : cl.izq,$ 
                                 $fdb : 0)$ 
         $cl.izq.der \leftarrow q.izq$ 
         $cl.izq.izq \leftarrow izqp$ 
         $cl.izq.der.padre = cl.izq$ 
         $cl.izq.izq.padre = cl.izq$ 
    end if

```

```

    cl.valor ← q.valor
    cl.fdb ← 0
    cl.der.fdb ← 0
else
    if cl.der.fdb == 0 then                                O(1)
        q ← cl.der
        izqp ← cl.izq
        cl.der ← q.der
        q.der.padre ← cl
        cl.izq ← tupla(der : NULL,
                        izq : NULL,
                        valor : cl.valor,
                        padre : cl.izq,
                        fdb : 0)
        cl.izq.der ← q.izq
        cl.izq.izq ← izqp
        cl.izq.der.padre = cl.izq
        cl.izq.izq.padre = cl.izq
        cl.valor ← q.valor
        cl.fdb ← -1
        cl.izq.fdb ← +1
    else
        if cl.der.fdb == -1 then                            O(1)
            r ← cl.der.izq                                O(1)
            cl.der.izq ← r.der
            cl.der.izq.padre ← cl.der
            clizq ← cl.izq                                O(1)
            cl.izq ← tupla(der : NULL,                      O(1)
                            izq : NULL,
                            valor : cl.valor,
                            padre : cl,
                            fdb : 0)
            cl.izq.der ← r.izq                            O(1)
            cl.izq.izq ← clizq                            O(1)
            cl.izq.izq.padre ← cl.izq                     O(1)
            cl.izq.der.padre ← cl.izq                     O(1)
            cl.valor ← r.valor                            O(1)

            if r.fdb == -1 then                            O(1)
                cl.fdb ← 0                                O(1)
                cl.izq.fdb ← 0                            O(1)
                cl.der.fdb ← +1                            O(1)
            else
                cl.fdb ← 0                                O(1)
                cl.der.fdb ← 0                            O(1)
                cl.izq.fdb ← -1                            O(1)
            end if
            r ← NULL                                        O(1)
        end if
    end if
end if

```

```

    end if
else
    if cl.izq.fdb == +1 then
        q ← cl.izq
        derp ← cl.der
        cl.izq ← q.der
        q.der.padre ← cl
        cl.der ← tuple{der : NULL,
                        izq : NULL,
                        valor : cl.valor,
                        padre : derp,
                        fdb : 0}
        cl.der.der ← derp.der
        cl.der.izq ← q.izq
        cl.der.der.padre = cl.der
        cl.der.izq.padre = cl.der
        cl.valor ← q.valor
        cl.fdb ← 0
        cl.der.fdb ← 0
        derp ← NULL
    else
        q ← cl.izq

        if q.fdb == 0 then
            derp ← cl.der
            cl.izq ← q.der
            q.izq.padre ← cl
            cl.der ← tuple{der : NULL,
                            izq : NULL,
                            valor : cl.valor,
                            padre : derp,
                            fdb : 0}
            cl.der.der ← derp
            cl.der.izq ← q.izq
            cl.der.der.padre = cl.der
            cl.der.izq.padre = cl.der
            cl.valor ← q.valor
            cl.fdb ← +1
            cl.der.fdb ← -1
        else
            if q.fdb == -1 then
                r ← q.izq
                q.izq ← r.der
                q.izq.padre ← q
                derp ← cl.der
                cl.der ← tuple{der : NULL,
                                izq : NULL,
                                valor : cl.valor,
                                padre : cl,
                                fdb : 0}
            end if
        end if
    end if
end if

```

<i>cl.der.der</i> \leftarrow <i>r.izq</i>	O(1)
<i>cl.der.izq</i> \leftarrow <i>derp</i>	O(1)
<i>cl.der.izq.padre</i> \leftarrow <i>cl.der</i>	O(1)
<i>cl.der.der.padre</i> \leftarrow <i>cl.der</i>	O(1)
<i>cl.valor</i> \leftarrow <i>r.valor</i>	O(1)
if <i>r.fdb</i> == -1 then	O(1)
<i>cl.fdb</i> \leftarrow 0	O(1)
<i>cl.der.fdb</i> \leftarrow 0	O(1)
<i>cl.izq.fdb</i> \leftarrow +1	O(1)
else	
<i>cl.fdb</i> \leftarrow 0	O(1)
<i>cl.izq.fdb</i> \leftarrow 0	O(1)
<i>cl.der.fdb</i> \leftarrow -1	O(1)
end if	
<i>r</i> \leftarrow <i>NULL</i>	O(1)
end if	
end if	
end if	
<hr/>	
	O(1)

3 Diccionario por Prefijos

3.1 Interfaz

parámetros formales

géneros β

se explica con $\text{DICCIONARIO}(\text{SECU}(\text{CHAR}), \beta)$

géneros $\text{diccPref}(\text{secu}(\text{char}), \beta)$

Operaciones

$\text{NUEVO}() \longrightarrow res : \text{diccPref}(\text{secu}(\text{char}), \beta)$

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

Post $\equiv \{(\forall p:\text{secu}(\text{char})) \neg(\text{def?}(p, res))\}$

Descripción: Crea un nuevo diccionario vacío

Complejidad: $O(1)$

$\text{VACIO?}(\text{in } dp : \text{diccPref}(\text{secu}(\text{char}), \beta)) \longrightarrow res : \text{bool}$

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

Post $\equiv \{res = (\forall c:\text{secu}(\text{char})) \neg \text{def?}(c, dp)\}$

Descripción: Devuelve true o false si el diccionario es o no vacío

Complejidad: $O(1)$

4 Diccionario por Prefijos

4.1 Interfaz

parámetros formales

géneros β

se explica con $\text{DICCIONARIO}(\text{SECU}(\text{CHAR}), \beta)$

géneros $\text{diccPref}(\text{secu}(\text{char}), \beta)$

Operaciones

$\text{NUEVO}() \longrightarrow res : \text{diccPref}(\text{secu}(\text{char}), \beta)$

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

Post $\equiv \{(\forall p:\text{secu}(\text{char})) \neg(\text{def?}(p, res))\}$

Descripción: Crea un nuevo diccionario vacío

Complejidad: $O(1)$

$\text{DEF?}(\text{in } dp : \text{diccPref}(\text{secu}(\text{char}), \beta), \text{ in } p : \text{secu}(\text{char})) \longrightarrow res : \text{bool}$

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

Post $\equiv \{res = p \in \text{claves}(dp)\}$

Descripción: Devuelve true o false según si la clave está o no definida

Complejidad: $O(L)$

CLAVES(**in** $dp : \text{diccPref}(\text{secu}(\text{char}), \beta) \longrightarrow res : \text{conj}(\text{secu}(\text{char}))$)

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

Post $\equiv \{(\forall c : \text{secu}(\text{char})) c \in \text{claves}(dp) \iff \text{def?}(c, dp)\}$

Descripción: Devuelve un conjunto de las claves del diccionario

Complejidad: $O(L)$?

DEFINIR(**in/out** $dp : \text{diccPref}(\text{secu}(\text{char}), \beta)$, **in** $p : \text{secu}(\text{char})$, **in** $s : \beta$)

Pre $\equiv \{dp = dp_0 \wedge \neg \text{def?}(p, dp)\}$

Post $\equiv \{\text{def?}(p, dp) \wedge \text{obtener}(p, dp) =_{\text{obs}} \wedge (\forall c \in \text{claves}(dp_0)) \text{def?}(c, dp)\}$

Descripción: Inserta una nueva clave con su significado en el diccionario

Complejidad: $O(L)$

OBTENER(**in** $dp : \text{diccPref}(\text{secu}(\text{char}), \beta)$, **in** $p : \text{secu}(\text{char})$) $\longrightarrow res : \beta$

Pre $\equiv \{\text{def?}(p, dp)\}$

Post $\equiv \{res = \text{obtener}(p, dp)\}$

Descripción: Retorna el significado de la clave pedida

Complejidad: $O(L)$

Aliasing: Devuelve res por referencia

ELIMINAR(**in/out** $dp : \text{diccPref}(\text{secu}(\text{char}), \beta)$, **in** $p : \text{secu}(\text{char})$)

Pre $\equiv \{dp = dp_0 \wedge \text{def?}(p, dp)\}$

Post $\equiv \{\neg \text{def?}(dp) \wedge (\forall c \in \text{claves}(dp_0), c \neq p) \text{def?}(c, dp)\}$

Descripción: Elimina del diccionario la clave deseada

Complejidad: $O(L)$ DEF?(**in** $dp : \text{diccPref}(\text{secu}(\text{char}), \beta)$, **in** $p : \text{secu}(\text{char})$) $\longrightarrow res : \text{bool}$

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

Post $\equiv \{res = p \in \text{claves}(dp)\}$

Descripción: Devuelve true o false según si la clave está o no definida

Complejidad: $O(L)$

CLAVES(**in** $dp : \text{diccPref}(\text{secu}(\text{char}), \beta) \longrightarrow res : \text{conj}(\text{secu}(\text{char}))$)

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

Post $\equiv \{(\forall c : \text{secu}(\text{char})) c \in \text{claves}(dp) \iff \text{def?}(c, dp)\}$

Descripción: Devuelve un conjunto de las claves del diccionario

Complejidad: $O(L)$?

DEFINIR(**in/out** $dp : \text{diccPref}(\text{secu}(\text{char}), \beta)$, **in** $p : \text{secu}(\text{char})$, **in** $s : \beta$)

Pre $\equiv \{dp = dp_0 \wedge \neg \text{def?}(p, dp)\}$

Post $\equiv \{\text{def?}(p, dp) \wedge \text{obtener}(p, dp) =_{\text{obs}} \wedge (\forall c \in \text{claves}(dp_0)) \text{def?}(c, dp)\}$

Descripción: Inserta una nueva clave con su significado en el diccionario

Complejidad: $O(L)$

OBTENER(**in** $dp : \text{diccPref}(\text{secu}(\text{char}), \beta)$, **in** $p : \text{secu}(\text{char})$) $\longrightarrow res : \beta$

Pre $\equiv \{\text{def?}(p, dp)\}$

Post $\equiv \{res = \text{obtener}(p, dp)\}$

Descripción: Retorna el significado de la clave pedida

Complejidad: $O(L)$

Aliasing: Devuelve res por referencia

ELIMINAR(**in/out** $dp : \text{diccPref}(\text{secu}(\text{char}), \beta)$, **in** $p : \text{secu}(\text{char})$)

Pre $\equiv \{dp = dp_0 \wedge \text{def?}(p, dp)\}$

Post $\equiv \{\neg \text{def?}(dp) \wedge (\forall c \in \text{claves}(dp_0), c \neq p) \text{def?}(c, dp)\}$

Descripción: Elimina del diccionario la clave deseada

Complejidad: $O(L)$