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, Jésica	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

Una DCNet es

1.1 Interfaz

se explica con DCNET

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

géneros `dcnet`

Operaciones

CREARSISTEMA(**in** r : `red`) $\longrightarrow$ res : `dcnet`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} iniciarDCNet(r)\}$
Descripción: Crea un sistema DCNet.
Complejidad: O(??????????????????????????????????)
Aliasing:

CREARPAQUETE(**in/out** s : `dcnet`, **in** p : `paquete`)
Pre $\equiv$ $\{s =_{\text{obs}} s_0 \wedge FALTACHOCLO\}$
Post $\equiv$ $\{s =_{\text{obs}} crearPaquete(s_0, p)\}$
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 $\equiv$ $\{s =_{\text{obs}} avanzarSegundo(s_0)\}$
Descripción: Avanza un segundo el sistema. Todas las computadoras envían su respectivo paquete y en consecuencia se actualizar los paquetes en espera de cada una de ellas.
Complejidad: O($n \times \ (L + log(n) + log(k))$)
Aliasing:

DAMERED(**in** s : `dcnet`) $\longrightarrow$ res : `red`
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} red(s)\}$
Descripción: Devuelve la red de DCNet.
Complejidad: O(??????????????????????????????????)
Aliasing:

CAMINORECORRIDO(**in** s : `dcnet`, **in** p : `paquete`) $\longrightarrow$ res : `secu(compu)`
Pre $\equiv$ $\{paqueteEnTransito?(s, p)\}$
Post $\equiv$ $\{res =_{\text{obs}} caminoRecorrido(s, p)\}$
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`) $\longrightarrow$ res : `nat`
Pre $\equiv$ $\{c \in computadoras(red(s))\}$
Post $\equiv$ $\{res =_{\text{obs}} cantidadEnviados(s, c)\}$
Descripción: Devuelve la cantidad de paquetes enviados por una computadora.

Complejidad: O(n)
Aliasing:

enEspera(**in** s : `dcnet`, **in** c : `compu`) $\longrightarrow$ res : `puntero(conjLogP(paquete))`
Pre $\equiv$ $\{c \in computadoras(red(s))\}$
Post $\equiv$ $\{res =_{\text{obs}} enEspera(s, c)\}$
Descripción: Devuelve un iterador a los paquetes de la computadora.
Complejidad: O(L)
Aliasing:

laQueMasEnvio(**in** s : `dcnet`) $\longrightarrow$ res : `compu`
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} laQueMasEnvio(s, p)\}$
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** `tupla`⟨Compus : `arreglo(tupla`⟨IP : `String`,),
pN : `puntero(conjLog(paquete))`,
#Paquetes : `nat`,
Num : `nat`⟩
CompusPorPref : `diccPref(compu,tupla`⟨PorNom : `conjLog(paquete)`, ,
PorPrior : `conjLog(paquete)`⟩
CaminosMinimos : `arreglo(arreglo(arreglo(compu)))`,
LaQMasEnvio : `nat` ⟩

Invariante de representación

1.

Algoritmos

icrearSistema(**in** r : `red`) $\longrightarrow$ res : `dcnet`

$n \leftarrow \#(\text{compus}(red))$	O(#compus(red)=n)?
$i \leftarrow 0$	
$j \leftarrow 0$	O(1)
$res.Compus \leftarrow$ CrearArreglo(n)	O(n)
$res.CaminosMinimos \leftarrow$ CrearArreglo(n)	O(n)
var p : `arreglo_dimensionable de puntero(conjLog(paquete))`	
while i<n **do**	O(n)
$res.CaminosMinimos[i] \leftarrow$ CrearArreglo(n)	O(n)

$p[i] \leftarrow NULL$ O(1)
$res.Compus[i] \leftarrow$ **tupla**$< compu(r,i), p[i], 0, 0 >$
no se como se deben escribir las tuplas O(1)
DEFINIR(res.CompusPorPref,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(complejidad cammin(red))
$j++$
end while
$i++$
end while
$res.LaQMasEnvio \leftarrow 0$ O(1)

O(max{$n^2 \times O(complejidadcammin(red))$),

ICREARPAQUETE(**in/out** s : dcnet, **in** p : paquete)
$t :< conjLog(paquete),\ conjLog(paquete) >$
$t \leftarrow$ OBTENER$(\pi_3(p),\ s.CompusPorPref)$ O(L)
INSERTAR$(\pi_1(t), p)$ O($log(k)$)
INSERTAR$(\pi_2(t), p)$ O($log(k)$)

O($L + log(k)$)

ILAQUEMASENVIO(**in** s : dcnet) $\longrightarrow$ res : compu
$res \leftarrow \pi_1(s.compus[s.LaQMasEnvio])$ O(1)

O(1)

IENESPERA(**in** s : dcnet, **in** c : compu) $\longrightarrow$ res : puntero(conjLogP(paquete))
$t :< conjLog(paquete),\ conjLog(paquete) >$
$t \leftarrow$ OBTENER$(\pi_1(c),\ s.CompusPorPref)$ O(L)
$res \leftarrow \&(\pi_2(t))$ O(1)

O(L)

IAVANZARSEGUNDO(**in/out** s : dcnet)
var i : nat
$i \leftarrow 0$ O(1)
var m : nat
$m \leftarrow s.LaQMasEnvio$
while $i <$ LONGITUD$(s.compus)$ **do** O(n)
if ¬VACIA? **then**
var IP : String
$IP \leftarrow \pi_1(s.compu[i])$
$t_1 :< conjLog(paquete),\ conjLog(paquete) >$
$t_1 \leftarrow$ OBTENER(IP, S.COMPUSPORPREF) O(L)
var p : paquete
$p \leftarrow$ SACARMAX$(\pi_2(t_1))$ O($log(k)$)
BORRAR$(\pi_2(t_1),\ p)$ O($log(k)$)
BORRAR$(\pi_1(t_1),\ p)$ O($log(k)$)
$\pi_3(s.compus[i]) \leftarrow \pi_3(s.compus[i]) + 1$ O(1)
$t_2 :< conjLog(paquete),\ conjLog(paquete) >$
$t_2 \leftarrow$ OBTENER$(\pi_4(P), s.CompusPorPref)$ O(L)
INSERTAR$(\pi_2(t_2),\ p)$
O($log(k)$)
INSERTAR$(\pi_1(t_2),\ p)$ O($log(k)$)

if $\pi_3(s.compus[i] > max)$ **then**	O(1)
$max \leftarrow i$	O(1)
end if	
end if	
$i \leftarrow i + 1$	
end while	
$s.LaQMasEnvio \leftarrow max$	O(1)
	O($n \times (L + log(k))$)

ICANTIDADENVIADOS(**in/out** s : `dcnet`, **in** c : `compu`) $\longrightarrow$ res : `nat`

var i : `nat`	
$i \leftarrow 0$	O(1)
while $\pi_1(s.compus[i]) \neq \pi_1(c)$ **do**	O(n)
$i \leftarrow i + 1$	O(1)
end while	
$res \leftarrow \pi_3(s.compus[i])$	O(1)
	O(n)

ICAMINORECORRIDO(**in** s : `dcnet`, **in** p : `paquete`) $\longrightarrow$ res : `secu(compu)`

var i : `nat`	
$i \leftarrow 0$	O(1)
var b : `bool`	
$b \leftarrow \neg$(PERTENECE?$(p, \pi_3(s.compus[i])))$	O($log(k)$)
while b **do**	O(n)
$i \leftarrow i + 1$	O(1)
$b \leftarrow \neg$(PERTENECE?$(p, \pi_3(s.compus[i])))$	O($log(k)$)
end while	
var j : `nat`	
$j \leftarrow 0$	O(1)
while $\pi_1(s.compus[j]) \neq \pi_3(p)$ **do**	O(n)
$i \leftarrow j + 1$	O(1)
end while	
$res \leftarrow s.CaminosMinimos[j][\pi_4(s.compus[i])]$	O(1 o n dependiendo de si hago copia o no)
	O($n \times log(k)$)

1.3 Servicios Usados

2 ConjLog

2.1 Interfaz

se explica con CONJ(α)

géneros `conjLog(`α`)`

Operaciones

NUEVO() $\longrightarrow$ *res* : `conjLog(`α`)`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} \emptyset$}
Descripción: Crea un nuevo conjLog vacio
Complejidad: O(1)

VACÍO?(**in** *cl* : `conjLog(`α`)`) $\longrightarrow$ *res* : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res = (\emptyset?(cl))$}
Descripción: Indica si el conjunto tiene tamaño cero
Complejidad: $O(log(\#(cl)))$

SIZE(**in** *cl* : `conjLog(`α`)`) $\longrightarrow$ *res* : `nat`
Pre $\equiv$ {true}
Post $\equiv$ {$res = \#(cl)$}
Descripción: Indica la cantidad de elementos del conjunto
Complejidad: $O(\#(cl))$

ESTÁ(**in** *cl* : `conjLog(`α`)`, **in** $e : \alpha$) $\longrightarrow$ *res* : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res = (e \in cl)$}
Descripción: Retorna un booleano que indica si el elemento pertenece al conjunto
Complejidad: $O(log(\#(cl)))$

INSERTAR(**in/out** *cl* : `cl(`α`)`, *in* $e : \alpha$)
Pre $\equiv$ {$cl_0 =_{\text{obs}} cl \wedge \neg(e \in cl)$}
Post $\equiv$ {$cl_0 =_{\text{obs}} Agregar(cl_0, e)$}
Descripción: Inserta un nuevo elemento en el conjunto
Complejidad: O($log(\#(cl))$)

BORRAR(**in/out** *cl* : `cl(`α`)`, *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** `raiz : puntero(nodo)`
donde `nodo` **es** `tupla`$\langle$der : `puntero(nodo)`,
izq : `puntero(nodo)`,
valor : α,
padre : `puntero(nodo)`,
fdb : `nat`$\rangle$

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 arbol 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

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

$(\forall cl : \widehat{\texttt{clog(}\alpha\texttt{)}})$
$\text{Abs}(cl) \equiv c : \widehat{\texttt{conj(}\alpha\texttt{)}} \mid$
$((\forall e : \alpha)e \in c \ \Rightarrow_{\text{L}} \ esta(cl, e)) \wedge size(cl) = \#(c)$

2.5 Algoritmos

iVacío?(**in** cl : `conjLog(`α`)`) $\longrightarrow$ res : `bool`	
$res \leftarrow size(cl) == 0$	O(1)
	O(1)
iBorrar(**in/out** cl : `conjLog(`α`)`, *in* $e : \alpha$)	
$clactual \leftarrow cl$	O(1)
if $(\neg(cl.der == NULL) \wedge \neg(cl.izq = NULL))$ **then**	O(1)
$clactual \leftarrow$ iEncontrarPadre$(clactual, e)$	O($log(size(cl))$)
if $(clactual.der.valor == e)$ **then**	O(1)
$mm \leftarrow$ iDameMayMenores$(clactual.der.izq)$	O($log(size(cl))$)
$padremm \leftarrow mm.padre$	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
iRebYRecalcFDB$(padremm, false, true)$	O($log(size(cl))$)
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
iRebYRecalcFDB$(padremm, true, true)$	O($log(size(cl))$)
end if	
else	
$mm \leftarrow$ iDameMayMenores$(clactual.izq.izq)$	O($log(size(cl.izq.izq))$)
$padremm \leftarrow mm.padre$	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
rebYRecalcFDB$(padremm, false, true)$	O($log(size(cl))$)
else	
$padremm.fdb--$	O(1)
$mm.fdb \leftarrow clactual.der.fdb$	O(1)

```
            mm.der ← clactual.der.der                                   O(1)
            mm.izq ← clactual.der.izq                                   O(1)
            mm.padre ← clactual                                         O(1)
            mm.padre.der ← mm                                           O(1)
            mm.der.padre ← mm                                           O(1), Si no son NULL
            mm.izq.padre ← mm                                           O(1), Si no son NULL
            iRebYRecalcFDB(padremm, true, true)                         O(log(size(cl)))
        end if
    end if
else

    if cl.der == NULL ∧ cl.izq == NULL then                             O(1)
        res ← NULL                                                      O(1)
    else

        if cl.der == NULL then                                          O(1)

            if cl.izq.valor == e then                                   O(1)
                cl.izq ← NULL                                           O(1)
            else
                cl.valor ← cl.izq.valor                                 O(1)
                cl.izq ← NULL                                           O(1)
            end if
        else

            if cl.der.valor == e then                                   O(1)
                cl.der ← NULL                                           O(1)
            else
                cl.valor ← cl.der.valor                                 O(1)
                cl.der ← NULL                                           O(1)
            end if
        end if
    end if
end if
                                                                        ---------
                                                                        O(log(size(cl)))

iInsertar(in/out cl : conjLog(α),  in e : α)

if ¬(cl.der == NULL) ∧ ¬(cl.izq == NULL) then                           O(1)
    clactual ← iEncontrarPadre(clactual, e)                             O(log(size(cl)))

    if clactual.valor < e then
        clactual.der ←tupla⟨der : NULL,                                 O(1)
                            izq : NULL,
                            valor : e,
                            padre : clactual,
                            fdb : 0⟩
        iRebYRecalcFDB(clactual, true, false)
    else
```

```
        clactual.izq ←tupla⟨der : NULL,                                  O(1)
                            izq : NULL,
                            valor : e,
                            padre : clactual,
                            fdb : 0⟩
        iRebYRecalcFDB(clactual, false, false)
    end if
else

    if cl.der == NULL ∧ cl.izq == NULL then                              O(1)
        cl ←tupla⟨der : NULL,                                            O(1)
                  izq : NULL,
                  valor : e,
                  padre : clactual,
                  fdb : 0⟩
    else

        if cl.der! = NULL then
            cl.izq ←tupla⟨der : NULL,                                    O(1)
                        izq : NULL,
                        valor : e,
                        padre : cl,
                        fdb : 0⟩
        else
            cl.der ←tupla⟨der : NULL,                                    O(1)
                        izq : NULL,
                        valor : e,
                        padre : cl,
                        fdb : 0⟩
        end if
    end if
end if
                                                                         ---------
                                                                         O(log(size(cl)))

iEstá(in/out cl : conjLog(α), in e : α) ⟶ res : bool
  encontrado? ← false                                                    O(1)
  clactual ← cl                                                          O(1)

  while (clactual! = NULL) ∧ ¬(encontrado?) do                           O(1)

    if e > clactual.valor then                                           O(1)
        clactual ← clactual.der                                          O(1)
    else

        if ce < clactual.valor then                                      O(1)
            clactual ← clactual.izq                                      O(1)
        else
            encontrado? ← true                                           O(1)
        end if
    end if
  end while
  clactual ← NULL                                                        O(1)
  res ← encontrado?                                                      O(1)
```

2.6 Auxiliares

iRebYRecalcFDB(**in/out** cl : `conjLog(`α`)`, *in borroInsertoHijoDerecho?* : `bool`, *in estoyBorrando?* : `bool`)

while $cl! = NULL \wedge_{\mathrm{L}} cl.fdb! = 2 \wedge cl.fdb! = -2$ **do**	O(1)
if *estoyBorrando?* **then**	O(1)
if *borroInsertoHijoDerecho?* **then**	O(1)
$cl.fdb--$	O(1)
else	
$cl.fdb++$	O(1)
end if	
$borroInsertoHijoDerecho? \leftarrow (cl.padre.der! = NULL \wedge_{\mathrm{L}} cl.padre.der.valor == cl.valor)$	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	
$borroInsertoHijoDerecho? \leftarrow (cl.padre.izq! = NULL \wedge_{\mathrm{L}} cl.padre.izq.valor == cl.valor)$	O(1)
$cl \leftarrow cl.padre$	O(1)
end if	
end while	
if $cl! = NULL$ **then**	O(1)
iRotar(cl)	O(1)
end if	
	O($log(size(cl))$)

```
iRotar(in/out cl : conjLog(α))

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

    if cl.der.fdb == +1 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 ← 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
else

    if cl.izq.fdb == +1 then                                        O(1)
        q ← cl.izq
        derp ← cl.der
        cl.izq ← q.der
        q.der.padre ← cl
        cl.der ←tupla⟨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                                          O(1)
            derp ← cl.der
            cl.izq ← q.der
            q.izq.padre ← cl
            cl.der ←tupla⟨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                                         O(1)
                r ← q.izq                                               O(1)
                q.izq ← r.der                                           
                q.izq.padre ← q                                         
                derp ← cl.der                                           O(1)
                cl.der ←tupla⟨der : NULL,                               O(1)
                             izq : NULL,                                
                             valor : cl.valor,                          
                             padre : cl,                                
                             fdb : 0⟩                                   
                cl.der.der ← r.izq                                      O(1)
                cl.der.izq ← derp                                       O(1)
                cl.der.izq.padre ← cl.der                               O(1)
                cl.der.der.padre ← cl.der                               O(1)
                cl.valor ← r.valor                                      O(1)

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

3 Diccionario por Prefijos

3.1 Interfaz

parámetros formales

géneros β

se explica con DICCIONARIO(SECU(CHAR), β)

géneros `diccPref(secu(char)`, β`)`

Operaciones

NUEVO() $\longrightarrow$ *res* : `diccPref(secu(char),`β`)`
Pre $\equiv$ {true}
Post $\equiv$ {($\forall$ p:secu(char)) $\neg$(def?(p,*res*))}
Descripción: Crea un nuevo diccionario vacío
Complejidad: O(1)

VACIO?(**in** *dp* : `diccPref(secu(char),`β`)`) $\longrightarrow$ *res* : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {res=($\forall$ c: secu(char))$\neg$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 DICCIONARIO(SECU(CHAR), β)

géneros `diccPref(secu(char)`, β`)`

Operaciones

NUEVO() $\longrightarrow$ *res* : `diccPref(secu(char),`β`)`
Pre $\equiv$ {true}
Post $\equiv$ {($\forall$ p:secu(char)) $\neg$(def?(p,*res*))}
Descripción: Crea un nuevo diccionario vacío
Complejidad: O(1)

DEF?(**in** *dp* : `diccPref(secu(char),`β`)`, *in p* : `secu(char)`) $\longrightarrow$ *res* : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {res=p $\in$ claves(dp)}
Descripción: Devuelve true o false según si la clave está o no definida
Complejidad: O(L)

CLAVES(**in** *dp* : `diccPref(secu(char),`β`)`) $\longrightarrow$ *res* : `conj(secu(char))`
Pre $\equiv$ {true}
Post $\equiv$ {($\forall$ c: secu(char))c $\in$ claves(dp) $\iff$ def?(c,dp)}
Descripción: Devuelve un conjunto de las claves del diccionario
Complejidad: O(L)?

DEFINIR(**in/out** *dp* : `diccPref(secu(char),`β`)`, *in p* : `secu(char)`, *in s* : β)
Pre $\equiv$ {dp=dp_0 $\wedge$ $\neg$def?(p,dp)}
Post $\equiv$ {def?(p,dp) $\wedge$ obtener(p,dp)=$_{\text{obs}}$s $\wedge$ ($\forall$ c $\in$ claves(dp_0)) def?(c,dp)}
Descripción: Inserta una nueva clave con su significado en el diccionario
Complejidad: O(L)

OBTENER(**in** *dp* : `diccPref(secu(char),`β`)`, **in** *p* : `secu(char)`) $\longrightarrow$ *res* : β
Pre $\equiv$ {def?(p,dp)}
Post $\equiv$ {$res = obtener(p, dp)$}
Descripción: Retorna el significado de la clave pedida
Complejidad: O(L)
Aliasing: Devuelve res por referencia

ELIMINAR(**in/out** *dp* : `diccPref(secu(char),`β`)`, **in** *p* : `secu(char)`)
Pre $\equiv$ {dp=dp_0 $\wedge$ def?(p,dp)}
Post $\equiv$ {$\neg$def?(dp) $\wedge$($\forall c \in claves(dp_0), c \neq$ p)def?(c,dp)}
Descripción: Elimina del diccionario la clave deseada
Complejidad: O(L) DEF?(**in** *dp* : `diccPref(secu(char),`β`)`, *in p* : `secu(char)`) $\longrightarrow$ *res* : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {res=p $\in$ claves(dp)}
Descripción: Devuelve true o false según si la clave está o no definida
Complejidad: O(L)

CLAVES(**in** *dp* : `diccPref(secu(char),`β`)`) $\longrightarrow$ *res* : `conj(secu(char))`
Pre $\equiv$ {true}
Post $\equiv$ {($\forall$ c: secu(char))c $\in$ claves(dp) $\iff$ def?(c,dp)}
Descripción: Devuelve un conjunto de las claves del diccionario
Complejidad: O(L)?

DEFINIR(**in/out** *dp* : `diccPref(secu(char),`β`)`, *in p* : `secu(char)`, *in s* : β)
Pre $\equiv$ {dp=dp_0 $\wedge$ $\neg$def?(p,dp)}
Post $\equiv$ {def?(p,dp) $\wedge$ obtener(p,dp)=$_{\text{obs}}$s $\wedge$ ($\forall$ c $\in$ claves(dp_0)) def?(c,dp)}
Descripción: Inserta una nueva clave con su significado en el diccionario
Complejidad: O(L)

OBTENER(**in** *dp* : `diccPref(secu(char),`β`)`, **in** *p* : `secu(char)`) $\longrightarrow$ *res* : β
Pre $\equiv$ {def?(p,dp)}
Post $\equiv$ {$res = obtener(p, dp)$}
Descripción: Retorna el significado de la clave pedida
Complejidad: O(L)
Aliasing: Devuelve res por referencia

ELIMINAR(**in/out** *dp* : `diccPref(secu(char),`β`)`, **in** *p* : `secu(char)`)
Pre $\equiv$ {dp=dp_0 $\wedge$ def?(p,dp)}
Post $\equiv$ {$\neg$def?(dp) $\wedge$($\forall c \in claves(dp_0), c \neq$ p)def?(c,dp)}
Descripción: Elimina del diccionario la clave deseada
Complejidad: O(L)