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 $\equiv$ $\{s =_{\text{obs}} s_0 \wedge (\forall p_0 : paquete, paqueteEnTransito?(p, s))\ \neg(p_0 =_{\text{obs}} p) \wedge$
$destino(p) \in compus(red) \wedge origen(p) \in compus(red) \wedge_{\text{L}}$
$haycamino?(destino(p), origen(p), red(s))\}$
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 : `puntero(red)`
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} red(s)\}$
Descripción: Devuelve la red de DCNet.
Complejidad: O(1)
Aliasing: Devuelve un puntero a la misma red que la que se pasó como parámetro para crear el sistema

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`⟩
CompusPorPref : `diccPref(compu,tupla`⟨PorNom : `conjLog(paquete)`, ,
PorPrior : `conjLog(paquete)`⟩
CaminosMinimos : `arreglo(arreglo(arreglo(compu)))`,
LaQMasEnvio : `nat` ,
red : `red`⟩

esto se puede borrar despues: aclaracion en compus en cada indice del arreglo esta la compu correspondiente a esa numeracion

Invariante de representación

1. Todos los IP de *compus* pertenecen al conjunto de claves de *CompusPorPref* y la longitud de dicho arreglo es igual al cardinal de las claves del diccionario.
2. Los pN de las tuplas que tiene el arreglo *compus* apuntan al conjunto de paquetes(PorNom) de un significado en *CompusPorPref* cuya clave es igual al IP de esa posición en el arreglo.
3. Todos los conjuntos de los significados de *CompusPorPref* son disjuntos dos a dos.
4. Los conjuntos de los campos de la tupla PorNom, PorPrior son iguales.

5. La longitud de *CaminosMinimos* es igual a la longitud del arreglo que tiene *CaminosMinimos* en cada posición.
6. La longitud del arreglo, que tiene un arreglo de *CaminosMinimos* es menor o igual a la longitud de *CaminosMinimos*.
7. Los elementos del arreglo anteriormente mencionado son menores o iguales a la longitud de *CaminosMinimos* y no tiene repetidos.
8. La computadora que más paquetes envió es aquella cuyo índice es igual a *LaQMasEnvio*

Algoritmos

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

$res.red \leftarrow r$	
$n \leftarrow \#(\text{COMPUS}(red))$	O(#compus(red)=n)?
$i \leftarrow 0$	
$j \leftarrow 0$	O(1)
$res.Compus \leftarrow \text{CREARARREGLO}(n)$	O(n)
$res.CaminosMinimos \leftarrow \text{CREARARREGLO}(n)$	O(n)
var p : `arreglo_dimensionable de puntero(conjLog(paquete))`	
while i<n **do**	O(n)
$res.CaminosMinimos[i] \leftarrow \text{CREARARREGLO}(n)$	O(n)
$p[i] \leftarrow NULL$	O(1)
$res.Compus[i] \leftarrow < compu(r,i), p[i], 0 >$	O(1)
s: $< nat, conjLog(paquete, <_{id}), conjLog(paquete, <_p),$	
$conjLog(< paquete, indiceActual, indiceOrigen, indiceDestino >, <_{id}), conjLog(< paquete, indiceActual, in$	
$, <_p) >$ HAY que ver bien donde definir la relacion!!	
$\pi_1(s) \leftarrow compu(r,i)$	
$\pi_2(s) \leftarrow \text{NUEVO}()$	
$\pi_3(s) \leftarrow \text{NUEVO}()$	
$\pi_4(s) \leftarrow \text{NUEVO}()$	
$\pi_5(s) \leftarrow \text{NUEVO}()$	
DEFINIR(res.CompusPorPref,compu(r,i),s)	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/out** p : `paquete`)

$t_1 : < nat, conjLog(paquete, <_{id}), conjLog(paquete, <_p),$	
$conjLog(paquetePos, <_{id}), conjLog(paquetePos, <_p) >$	
$t_1 \leftarrow \text{OBTENER}(origen(p),\ s.CompusPorPref)$	O(L)
$t_2 : < nat, conjLog(paquete, <_{id}), conjLog(paquete, <_p),$	
$conjLog(paquetePos, <_{id}), conjLog(paquetePos, <_p) >$	
$t_2 \leftarrow \text{OBTENER}(destino(p),\ s.CompusPorPref)$	O(L)
p': $paquetePos$	
$indiceOrigen(p') \leftarrow \pi_1(t_1)$	O(1)

$indiceDestino(p') \leftarrow \pi_1(t_2)$ O(1)
$indiceActual(p') \leftarrow 0$
INSERTAR($\pi_2(t), p$) O($log(k)$)
INSERTAR($\pi_3(t), p$) O($log(k)$)
INSERTAR($\pi_4(t), p'$) O($log(k)$)
INSERTAR($\pi_5(t), p'$) O($log(k)$)

O($L + log(k)$)

ILAQUEMASENVIO(**in** s : `dcnet`) $\longrightarrow$ res : `compu`

$res \leftarrow s.compus[s.LaQMasEnvio].IP$ O(1)

ACLARACION: ACA deberia devolver una compu pero es alto bardo y no puedo hacerlo en O(1) me faltan las interfaces

O(1)

IDAMERED(**in** s : `dcnet`) $\longrightarrow$ res : `puntero(red)`

$res \leftarrow \&(s.red)$ O(1)

O(1)

IENESPERA(**in** s : `dcnet`, **in** c : `compu`) $\longrightarrow$ res : `puntero(conjLogP(paquete))`

$t :< nat, conjLog(paquete, <_{id}), conjLog(paquete, <_p),$
$conjLog(paquetePos, <_{id}), conjLog(paquetePos, <_p) >$
$t \leftarrow$ OBTENER($\pi_1(c)$, $s.CompusPorPref$) O(L)
$res \leftarrow \&(\pi_3(t))$ O(1)

O(L)

IAVANZARSEGUNDO(**in/out** s : `dcnet`)

var i : `nat`
$i \leftarrow 0$ O(1)
var m : `nat`
$m \leftarrow s.Compus[LaQMasEnvio].\#PaqE$
while $i <$ LONGITUD($s.Compus$) **do** O(n)
var IP : `String`
$IP \leftarrow s.Compus[i].IP$
$t_1 :< nat, conjLog(paquete, <_{id}), conjLog(paquete, <_p),$
$conjLog(paquetePos, <_{id}), conjLog(paquetePos, <_p) >$
$t_1 \leftarrow$ OBTENER($IP, s.CompusPorPref$) O(L)
var p : `paquete`
if $\neg$VACIA?($\pi_5(t_1)$) **then**
$p' \leftarrow$ SACARMAX($\pi_5(t_1)$) O($log(k)$)
BORRAR($\pi_2(t_1)$, PAQUETE(p')) O($log(k)$)
BORRAR($\pi_3(t_1)$, PAQUETE(p')) O($log(k)$)
BORRAR($\pi_4(t_1)$, p') O($log(k)$)
BORRAR($\pi_5(t_1)$, p') O($log(k)$)
$s.Compus[i].\#PaqE \leftarrow s.compus[i].\#PaqE + 1$ O(1)
$proxima \leftarrow s.CaminosMinimos$[INDICEORIGEN($p'$)][INDICEDESTINO($p'$)][INDICEACTUAL($p'$)+1] O(1) o O(L) si se copia

ACLARACION: coherente con caminoMinimo (sup que da un arreglo de IP)

if $\neg$(DESTINO(PAQUETE(p')) $= proxima$) **then** O(L)

ACLARACION: Si no es porque ya no esta en transito entonces hay que aclarar en algún lugar si eliminamos el paquete

ACTUALIZARINDICE(p') O(1)
$t_2 :< nat, conjLog(paquete, <_{id}), conjLog(paquete, <_p),$

$conjLog(paquetePos, <_{id}), conjLog(paquetePos, <_p) >$	
$t_2 \leftarrow$ OBTENER$(proxima, s.CompusPorPref)$	O(L)
INSERTAR$(\pi_2(t_2),$ PAQUETE$(p'))$	O$(log(k))$
INSERTAR$(\pi_3(t_2),$ PAQUETE$(p'))$	O$(log(k))$
INSERTAR$(\pi_4(t_2),\ p')$	O$(log(k))$
INSERTAR$(\pi_5(t_2),\ p')$	O$(log(k))$
end if	
if $s.Compus[i].\#PaqE > max)$ **then**	O(1)
$max \leftarrow i$	O(1)
end if	
end if	
$i \leftarrow i + 1$	O(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 $s.compus[i].IP \neq \pi_1(c)$ **do**	O(n)
$i \leftarrow i + 1$	O(1)
end while	
$res \leftarrow s.compus[i].\#PaqE$	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?$(p,*(s.compus[i].pN)))$ ESTA BIEN ESTE PERTENECE? Y CREO que la estructura deberia apuntar a colalog con paquetespos tambien	
	O$(log(k))$
while b **do**	O(n)
$i \leftarrow i + 1$	O(1)
$b \leftarrow \neg$(PERTENECE?$(p,*(s.compus[i].pN)))$	O$(log(k))$
end while	
$res \leftarrow s.CaminosMinimos[$INDICEORIGEN$(p')][i]$	
	O$(n \times log(k))$

1.3 Servicios Usados

2 ConjLog

2.1 Interfaz($\alpha, =_\alpha, <_\alpha$)

2.1.1 parámetros formales

géneros α

operaciones

- $=_\alpha \bullet : \alpha \times \alpha \to bool$ Relación de equivalencia
- $<_\alpha \bullet : \alpha \times \alpha \to bool$ Relación de orden

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

PERTENECE(**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: $O(log(\#(cl)))$

BUSCAR(**in** *cl* : `conjLog(`α`)`, **in** $e : \alpha$) $\longrightarrow$ *res* : α
Pre $\equiv \{e \in cl\}$
Post $\equiv \{res == e\}$
Descripción: Devuelve el elemento que se está buscando
Complejidad: $O(log(\#(cl)))$
Aliasing: El elemento se devuelve por referencia, hay aliasing entre el elemento buscado y el del conjunto
Alternativa, me parece que un poco mas limpia, es retornar el elemento por copia y exportar otra operacion que sea modificar, si se modifica el elemento buscado, luego se lo modifica en el conjunto

MENOR(**in** *cl* : `conjLog(`α`)`, *in* $e : \alpha$) $\longrightarrow$ *res* : α
Pre $\equiv \{e \in cl\}$
Post $\equiv \{res == max(cl)\}$
Descripción: Devuelve el menor elemento del 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`⟨der : `puntero(nodo)`,
izq : `puntero(nodo)`,
valor : α,
padre : `puntero(nodo)`,
fdb : `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 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`	
$res \leftarrow cl == NULL$	O(1)
	O(1)

iBorrar(**in/out** cl : `conjLog(`α`)`, *in* $e : \alpha$)	
var *variandoHijoDerecho?* : *bool* $\leftarrow$ *true*	
var $clactual : conjLog(\alpha) \leftarrow cl$	O(1)
var $aBorrar : conjLog(\alpha)$	
if $(\neg(cl.der == NULL) \wedge \neg(cl.izq == NULL))$ **then**	O(1)
$clactual \leftarrow$ iEncontrarPadre$(clactual, e)$	O$(log(size(cl)))$
if $cactual.der! = NULL \wedge_{\mathrm{L}} cactual.der.valor == e$ **then**	
$aBorrar \leftarrow cactual.der$	O(1)
else	
$aBorrar \leftarrow cactual.izq$	O(1)
end if	
var $mm : conjLog(\alpha) \leftarrow$ iDameMayorMenores$(clactual)$	O$(log(size(cl)))$
if $mm.valor == e$ **then**	O(1)
if $mm.padre.der! = NULL \wedge_{\mathrm{L}} mm.padre.der.valor == mm.valor$ **then**	
$variandoHijoDerecho? \leftarrow true$	O(1)
$mm.padre.der = NULL$	
$mm.padre.fdb--$	
else	
$variandoHijoDerecho? \leftarrow false$	O(1)
$mm.padre.izq = NULL$	
$mm.padre.fdb++$	
end if	
else	
var $mmValor : \alpha \leftarrow mm.valor$	O(1)
$aBorrar.valor \leftarrow mmValor$	O(1)
if $mm.izq! = NULL$ **then**	O(1)
$mm.valor \leftarrow mm.izq.valor$	O(1)
$mm.izq \leftarrow NULL$	O(1)
$mm.fdb++$	O(1)
if $mm.padre.valor == e$ **then**	
$variandoHijoDerecho? \leftarrow false$	
else	
$variandoHijoDerecho? \leftarrow true$	
end if	
else	

```
            if mm.padre.valor == e then                                   
                mm.padre.izq = NULL                                       
                variandoHijoDerecho? ← false                              
            else                                                          
                mm.padre.der = NULL                                       
                variandoHijoDerecho? ← true                               
            end if                                                        
        end if                                                            
        iRebYRecalcFDB(mm.padre, variandoHijoDerecho?, estoyBorrando?)    
                                                              O(log(size(cl)))
    else                                                                  

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

    var clactual : conjLog(α) ← cl                            O(1)        

    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($log(size(cl))$)

iPertenece(**in/out** cl : conjLog(α), *in* $e : \alpha$) $\longrightarrow$ res : bool

```
var encontrado? : bool ← false                                           O(1)

var clactual : conjLog(α) ← 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

$clactual \leftarrow NULL$	O(1)
$res \leftarrow encontrado?$	O(1)
	$O(log(size(cl)))$

iMenor(**in** cl : conjLog(α)) $\longrightarrow$ res : α

var $clactual$: conjLog(α) $\leftarrow$ cl	O(1)
$clactual \leftarrow iMenorNodo(clactual)$	O($log(size(cl))$)
$res \leftarrow clactual.valor$	O(1)
	$O(log(size(cl)))$

iBuscar(**in** cl : conjLog(α), e : α) $\longrightarrow$ res : α

var $padre$: conjLog(α) $\leftarrow iEncontrarPadre(cl, e)$	O($log(size(cl))$)
if $padre.der! = NULL \wedge_{\mathrm{L}} padre.der.valor == e$ **then**	O(1)
$res \leftarrow padre.der.valor$	
else	
$res \leftarrow padre.izq.valor$	
end if	
	O($log(size(cl))$)

2.6 Auxiliares

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

```
var clactual : conjLog(α) ← cl                                   O(1)

var termino? : bool ← false                                      O(1)

if estoyBorrando? then

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

        if variandoHijoDerecho? then

            if clactual.fdb == −1 then                           O(1)

                var fdbIzq : nat                                 O(1)

                if cl.izq! = NULL then
                    fdbIzq ← cl.izq                              O(1)
                end if

                if cl.izq! = NULL ∧_L cl.izq.fdb == 1 then       O(1)
                    iRotarLR(cl.izq)                             O(1)
                end if
                iRotarLL(cl)                                     O(1)

                if cl.izq! = NULL ∧_L fdbIzq == 0 then           O(1)
                    termino? ← true                              O(1)
                end if
            else

                if cl.fdb == +1 then                             O(1)
                    cl.fdb ← 0                                   O(1)
                    termino? ← true                              O(1)
                else
                    cl.fdb ← −1                                  O(1)
                end if
            end if
        else

            if clactual.fdb == −1 then                           O(1)

                var fdbDer : nat                                 O(1)

                if cl.der! = NULL then                           O(1)
                    fdbDer ← cl.der.fdb                          O(1)
                end if

                if cl.der! = NULL ∧_L fdbDer == 1 then           O(1)
                    iRotarRL(cl.der)                             O(1)
```

```
            end if
            iRotarRR(cl)                                              O(1)

            if fdbDer == 0 then                                       O(1)
                termino? ← true                                       O(1)
            end if
        else

            if cl.fdb == −1 then                                      O(1)
                cl.fdb ← 0                                            O(1)
                termino? ← true                                       O(1)
            else
                cl.fdb ← +1                                           O(1)
            end if
        end if
    end if
    variandoHijoDerecho ← (cl.padre! = NULL ∧L cl.padre.der.valor == cl.valor)
                                                                      O(1)
    clactual ← clactual.padre                                         O(1)
  end while
else    // No hubo borrado, entonces hubo una inserción

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

    if variandoHijoDerecho? then

      if clactual.fdb == +1 then                                      O(1)

        var fdbDer : nat                                              O(1)

        if cl.der! = NULL then
            fdbDer ← cl.der.fdb                                       O(1)
        end if

        if cl.der! = NULL ∧L fdbDer == −1 then                        O(1)
            iRotarRL(cl.der)                                          O(1)
        end if
        iRotarRR(cl)                                                  O(1)
        termino? ← true
      else

        if clactual.fdb == −1 then                                    O(1)
            clactual.fdb ← 0                                          O(1)
            termino? ← true                                           O(1)
        else
            clactual.fdb ← 1                                          O(1)
        end if
      end if
    else

      if clactual.fdb == −1 then                                      O(1)
        fdbIzq : nat                                                  O(1)
```

```
            if cl.izq! = NULL then
                fdbIzq ← cl.izq.fdb                                    O(1)
            end if

            if cl.izq! = NULL ∧L fdbIzq == +1 then                     O(1)
                iRotarLR(cl.izq)                                       O(1)
            end if
            iRotarLL(cl)                                               O(1)
            termino? ← true                                            O(1)
        else

            if clactual.fdb == +1 then                                 O(1)
                clactual.fdb ← 0                                       O(1)
                termino? ← true                                        O(1)
            else
                clactual.fdb ← −1                                      O(1)
            end if
        end if
    end if
    variandoHijoDerecho ← (cl.padre! = NULL ∧L cl.padre.der.valor == cl.valor)
                                                                       O(1)
    clactual ← clactual.padre                                          O(1)
  end while
end if
                                                                       ---
                                                                       O(log(size(cl)))
```

iRotarRR(**in/out** cl : `conjLog(`α`)`)

var $nietoRR$: `conjLog(`α`)` $\leftarrow cl.der.der$	O(1)
var $hijoDer$: `conjLog(`α`)` $\leftarrow cl.der$	O(1)
var $hijoIzq$: `conjLog(`α`)` $\leftarrow cl.izq$	O(1)
$cl.der \leftarrow NULL$	O(1)
$cl.izq =$`tupla`⟨der : `hijoDer.der`, izq : `cl.izq`, valor : `cl.valor`, padre : `cl`, fdb : `0`⟩	O(1)
$cl.izq.izq.padre \leftarrow cl.izq$	O(1)
$cl.izq.der.padre \leftarrow cl.izq$	O(1)
$cl.valor = hijoDer.valor$	O(1)
$cl.der = nietoRR$	O(1)
$cl.der.padre \leftarrow cl$	O(1)
	O(1)

iRotarRL(**in/out** cl : `conjLog(`α`)`)

var $nietoRR$: `conjLog(`α`)` $\leftarrow cl.der.der$	O(1)
var $nietoRL$: `conjLog(`α`)` $\leftarrow cl.der.izq$	O(1)
var $valorDer$: $\alpha \leftarrow cl.der.valor$	O(1)
$cl.der.valor \leftarrow nietoRL.valor$	O(1)
$nietoRR.izq \leftarrow nietoRL.der$	O(1)
$cl.der.der \leftarrow nietoRR$	O(1)
$cl.der.izq \leftarrow nietoRL.izq$	O(1)
$cl.der.der.padre \leftarrow cl.der$	O(1)
$cl.der.izq.padre \leftarrow cl.der$	O(1)
$cl.izq.fdb \leftarrow +1$	
	$O(1)$

iRotarLL(**in/out** cl : `conjLog(`α`)`)

var $nietoLL$: `conjLog(`α`)` $\leftarrow cl.izq.izq$	O(1)
var $hijoIzq$: `conjLog(`α`)` $\leftarrow cl.izq$	O(1)
var $hijoDer$: `conjLog(`α`)` $\leftarrow cl.der$	O(1)
$cl.izq \leftarrow NULL$	O(1)
$cl.der =$`tupla`⟨der : `cl.der`, izq : `hijoIzq.der`, valor : `cl.valor`, padre : `cl`, fdb : `0`⟩	O(1)
$cl.der.izq.padre \leftarrow cl.der$	O(1)
$cl.der.der.padre \leftarrow cl.der$	O(1)
$cl.valor = hijoIzq.valor$	O(1)

$cl.izq = nietoLL$	O(1)
$cl.izq.padre \leftarrow cl$	O(1)
	$O(1)$

iRotarLR(**in/out** cl : conjLog(α))

var $nietoLL$: conjLog(α) $\leftarrow cl.izq.izq$	O(1)
var $nietoLR$: conjLog(α) $\leftarrow cl.izq.der$	O(1)
var $valorIzq : \alpha \leftarrow cl.izq.valor$	O(1)
$cl.izq.valor \leftarrow nietoRL.valor$	O(1)
$nietoLL.izq \leftarrow nietoLR.izq$	O(1)
$cl.izq.izq \leftarrow nietoLL$	O(1)
$cl.izq.der \leftarrow nietoLR.der$	O(1)
$cl.izq.izq.padre \leftarrow cl.izq$	O(1)
$cl.izq.der.padre \leftarrow cl.izq$	O(1)
$cl.izq.fdb \leftarrow -1$	O(1)
	$O(1)$

iEncontrarPadre(**in** cl : conjLog(α), $e : \alpha$) $\longrightarrow$ res : conjLog(α)

var $clactual$: conjLog(α)	O(1)
var $encontrado?$: bool $\leftarrow (clactual.der! = NULL \wedge_{\mathrm{L}} clactual.der.valor == e) \vee (clactual.izq! = NULL \wedge_{\mathrm{L}} clactual.izq.valor == e)$	
while $\neg encontrado?$ **do**	
if $e > clactual.valor$ **then**	
$clactual \leftarrow clactual.der$	$O(1)$
else	
$clactual \leftarrow clactual.izq$	$O(1)$
end if	
$encontrado? \leftarrow (clactual.der! = NULL \wedge_{\mathrm{L}} clactual.der.valor == e) \vee (clactual.izq! = NULL \wedge_{\mathrm{L}} clactual.izq.valor == e)$	
end while	
$res \leftarrow clactual$	$O(1)$
	$O(size(cl))$

iDameMayorMenores(**in** cl : conjLog(α), $e : \alpha$) $\longrightarrow$ res : conjLog(α)

var $clactual$: conjLog(α) $\leftarrow cl$	O(1)
if $clactual.izq! = NULL$ **then**	O(1)
$clactual \leftarrow iMayorNodo(clactual)$	$O(log(size(cl)))$
end if	
$res \leftarrow clactual$	O(1)

iMenorNodo(**in** cl : conjLog(α)) $\longrightarrow$ res : $conjLog(\alpha)$

var $clactual$: conjLog(α) $\leftarrow cl$	O(1)
while $clactual.izq! = NULL$ **do**	
$clactual \leftarrow clactual.izq$	
end while	
$res \leftarrow clactual$	O(1)
	$O(log(size(cl)))$

iMayorNodo(**in** cl : conjLog(α)) $\longrightarrow$ res : $conjLog(\alpha)$

var $clactual$: conjLog(α) $\leftarrow cl$	O(1)
while $clactual.der! = NULL$ **do**	
$clactual \leftarrow clactual.der$	
end while	
$res \leftarrow clactual$	O(1)
	$O(log(size(cl)))$

size(**in** cl : conjLog(α)) $\longrightarrow$ res : nat

if $cl == NULL$ **then**
  $res \leftarrow 0$
else
  $res \leftarrow 1 + iSize(cl.der) + iSize(cl.izq)$
end if

2.7 Operaciones auxiliares de conj(α)

$menor : conj(\alpha)\ c \rightarrow \alpha \qquad \{\#(c) > 0\}$

$menor(c) =$
  if $(\#(c) = 1)$ **then**
    $dameUno(c)$
  else
    if $(dameUno(c) < menor(sinUno(c))$ **then**
      $dameUno(c)$
    else
      $menor(sinUno(c))$
    fi
  fi

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)

5 Paquete

Un Paquete es

5.1 Interfaz

se explica con PAQUETE

géneros `paquete`

Operaciones

CREARPAQUETE(**in** id : `nat`, **in** o : `compu`, **in** d : `compu`, **in** pr : `nat`) $\longrightarrow$ res : `paquete`
Pre $\equiv$ {true}
Post $\equiv$ $\{pi_1(res) = id \wedge pi_2(res) = pr \wedge pi_3(res) = o \wedge pi_4(res) = d\}$
Descripción: Crea un paquete
Complejidad: O(1)

$\bullet <_p \bullet$(**in** p_1 : `paquete`, *in* p_2 : `paquete`) $\longrightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {res=true $\iff$ $(\pi_2(p_1) = \pi_2(p_2) \wedge pi_1(p_1) < pi_1(p_2) \vee (\pi_1(p_1) < \pi_1(p_2)))$}
Descripción: Define un orden en paquete según la prioridad
Complejidad: O(1)

$\bullet <_{id} \bullet$(**in** p_1 : `paquete`, *in* p_2 : `paquete`) $\longrightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {res=true $\iff$ $\mathrm{id}(\mathrm{p}_1)<\mathrm{id}(\mathrm{p}_2)$}
Descripción: Define un orden en paquete según el id
Complejidad: O(1)

ID(**in** p : `paquete`) $\longrightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ $\{res = \pi_1(paquete)\}$
Descripción: *Getterdeid*
Complejidad: O(1)

PRIORIDAD(**in** p : `paquete`) $\longrightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ $\{res = \pi_2(paquete)\}$
Descripción: *Getterdeprioridad*
Complejidad: O(1)

ORIGEN(**in** p : `paquete`) $\longrightarrow$ res : `Ip`
Pre $\equiv$ {true}
Post $\equiv$ $\{res = \pi_3(paquete)\}$
Descripción: *Getterdeorigen*
Complejidad: O(1)

DESTINO(**in** p : `paquete`) $\longrightarrow$ res : `Ip`
Pre $\equiv$ {true}
Post $\equiv$ $\{res = \pi_4(paquete)\}$
Descripción: *Getterdedestino*
Complejidad: O(1)

5.2 Representación

se representa con `paquete`

donde `paquete` **es** `tupla`⟨id : `nat`,
origen : `Ip`,
destino : `Ip`,
prioridad : `nat`⟩

6 PaquetePos

Un PaquetePos es

6.1 Interfaz

se explica con `tupla`⟨ : `Paquete`,
: `nat`,
: `nat`,
: `nat`⟩

géneros `paquetePos`

Operaciones

CREARPAQUETE(**in** id : `nat`, **in** o : `compu`, **in** d : `compu`, **in** pr : `nat`) $\longrightarrow$ res : $< paquete, nat, nat, nat >$
Pre $\equiv$ {true}
Post $\equiv$ $\{pi_1(pi_1(res)) = id \wedge pi_2(pi_1(res)) = pr \wedge pi_3(pi_1(res)) = o \wedge pi_4(pi_1(res)) = d\}$
Descripción: Crea un paquete
Complejidad: O(1)

• $<_p$ •(**in** p_1 : $< paquete, nat, nat, nat >$, **in** p_2 : $< paquete, nat, nat, nat >$) $\longrightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {res=true $\iff$ $(\pi_2(\pi_1(p_1)) = \pi_2(\pi_1(p_2)) \wedge \pi_1(\pi_1((p_1)) < \pi_1(\pi_1((p_2)) \vee (\pi_1(\pi_1((p_1)) < \pi_1(\pi_1((p_2))))$}
Descripción: Define un orden en paquete según la prioridad
Complejidad: O(1)

• $<_{id}$ •(**in** p_1 : $< paquete, nat, nat, nat >$, *in* p_2 : $< paquete, nat, nat, nat >$) $\longrightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {res=true $\iff$ $(pi_1(pi_1(p_1)) < (pi_1(pi_1(p_2))))$}
Descripción: Define un orden en paquete según el id
Complejidad: O(1)

GETPAQUETE(**in** $ppos$: $< paquete, nat, nat, nat >$) $\longrightarrow$ res : `paquete`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} \pi_1(paquete)\}$
Descripción: Getter de paquete
Complejidad: O(1)

INDICEORIGEN(**in** p : $< paquete, nat, nat, nat >$) $\longrightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ $\{res = \pi_2(p)\}$
Descripción: Getter de indiceOrigen
Complejidad: O(1)

INDICEDESTINO(**in** p : $< paquete, nat, nat, nat >$) $\longrightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ $\{res = \pi_3(p)\}$
Descripción: Getter de indiceDestino
Complejidad: O(1)

POSACTUAL(**in** p : $< paquete, nat, nat, nat >$) $\longrightarrow$ res : `nat`
Pre $\equiv$ {true}
Post $\equiv$ $\{res = \pi_4(p)\}$

Descripción: Getter de posActual
Complejidad: O(1)

6.2 Representación

se representa con `paqPos`

donde `paqPos` **es** `tupla`⟨paquete : `paquete`,
indiceOrigen : `nat`,
indiceDestino : `nat`,
posActual : `nat`⟩