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.

1.2 Representación

se representa con `sistema`

donde `sistema` **es** `tupla`⟨Compus : `arreglo(tupla`⟨IP : `String`,),
pP : `itConjLogP`,
pN : `itConjLog`⟩
CompusPorPref : `diccPref(compu, tupla`⟨ItId : `ItconjLog(paquete)`, ,
ItP : `ItconjLog(paquete)`)⟩
CaminosMinimos : `arreglo(arreglo(arreglo(compu)))`,
PaquetePorId : `conjLog(paquete)`,
PaquetesPorPrioridad : `conjLog(paquete)`,
LaQMasEnvio : `puntero(compu)` ⟩

Algoritmos

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

$t \leftarrow$ Obtener($\pi_3(p)$, $s.CompusPorPref$)	O(L)
Agregar($\pi_1(t), p$)	O($log(k)$)
Agregar2($\pi_2(t), p$)	O($log(k)$)
	O($L + log(k)$)

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

$res \leftarrow$ ObtenerMaximo($s.compusPor\#Envios$)	O(1)
	O(1)

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

$t \leftarrow$ Obtener($\pi_1(c)$, $s.CompusPorPref$)	O(L)
$res \leftarrow \&(\pi_2(t))$	O(1)
	O(L)

2 ConjLog

2.1 Interfaz

se explica con CONJ(α)

géneros `conjLog(`α`)`

Operaciones

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

BUSCAR(**in** *cl* : `conjLog(`α`)`, **in** $e : \alpha$) $\longrightarrow$ *res* : α
Pre $\equiv$ {$c \in cl$}
Post $\equiv$ {$siguiente(res) = e$}
Descripción: Retorna el elemento que se esta buscando
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 $\equiv$ {$cl_0 =_{\text{obs}} Agregar(cl_0, e) \wedge siguiente(res) = e$}
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`⟩

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. No hay valores repetidos.
4. Si un nodo no es una hoja del árbol entonces los padres de los hijos derecho e izquierdo son iguales y es el nodo
5. Si un nodo es una hoja del arbol entonces los hijos derecho e izquierdo del árbol son NULL
6. El padre de la raíz es NULL

Rep : $\widehat{\texttt{restr}} \longrightarrow boolean$

$(\forall r : \widehat{\texttt{restr}})$
Rep$(r) \equiv$ **if** $(r.izq \neq$ Null$)$ **then**
if $(r.der =$ Null$)$ **then**
$*r.val =$ "NOT" $\wedge\ rep(*r.izq)$
else $*r.val \in Ag(\text{"}OR\text{"}, Ag(\text{"}AND\text{"}, vacio)) \wedge rep(*r.izq) \wedge rep(*r.der)$

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

$(\forall cl : \widehat{\texttt{clog}})$
Abs$(cl) \equiv c : \widehat{\texttt{conj}}\ |$
(s. = Null $\wedge$ *r.der = Null $\iff nil?(a)) \vee_{\text{L}}$
*r.izq $= izq(a) \wedge$ *r.der $= der(a) \wedge r.val = raiz(a)$

crearPaquete(**in/out** s : `dcnet`, p : `paquete`)
Pre $\equiv \{s =_{\text{obs}} s_0 \wedge FALTACHOCLO\}$
Post $\equiv \{enEspera(s, \pi_3(p)) =_{\text{obs}} enEspera(s_0, \pi_3(p) \cup \{p\})\}$
Descripción: *Creaunpaqueteyloagregaalacomputadoracorrespondiente.*
Complejidad: O$(L + log(k))$
Aliasing:

iDefinir(**in/out** t : `trie(`α`)`, **in** s : `string`, **in** $a : \alpha$)

if iDefinido?(t, s) **then**	O$(\|s\|)$
$n \leftarrow$ dameNodo(t, s)	O$(\|s\|)$
else	
$n \leftarrow$ CrearNodo(t, s)	O$(\|s\|)$
$iter \leftarrow$ AgregarRápido$(t.$claves$, s)$	O$(\|s\|)$
var e : `definición(`α`)`	
e.clave $\leftarrow iter$	O(1)
n.definición $\leftarrow \&e$	O(1)
end if	
$a' \leftarrow$ Copiar(a)	O$(copy(a))$
$(*n$.definición$)$.significado $\leftarrow \&a'$	O(1)
	O$(\|s\| + copy(a))$

iDefinido?(**in/out** t : `trie(`α`)`, **in** s : `string`) $\longrightarrow res$: `bool`

$n \leftarrow$ dameNodo(t, s)	O$(\|s\|)$
$res \leftarrow n \neq$ Null	O(1)
	O$(\|s\|)$

iSignificado(**in/out** t : `trie(`α`)`, **in** s : `string`) $\longrightarrow res : \alpha$

$n \leftarrow$ DAMENODO(t, s)	O($\|s\|$)
$res \leftarrow (*n.\text{definición}).\text{significado}$	O(1)
	O($\|s\|$)

iBorrar(**in/out** t : `trie(`α`)`, **in** s : `string`)

$n \leftarrow$ dameNodo(t, s)	O($\|s\|$)
EliminarSiguiente($(*n$.definición$)$.clave)	O(1)
n.definición $\leftarrow$ Null	O(1)
	O($\|s\|$)

iClaves(**in** t : `trie(`α`)`) $\longrightarrow$ res : `conj(string)`

$res \leftarrow t$.claves	O(1)
	O(1)

Auxiliares

dameNodo(**in** t : `trie(`α`)`, **in** s : `string`)
$\longrightarrow$ res : `puntero(definicion(`α`))`
Pre $\equiv$ {Rep(t) $\wedge_{\text{L}}$ $d_0 =_{\text{obs}}$ Abs(t)}
Post $\equiv$ {($res =_{\text{obs}}$ Null $\iff \neg$def?$(s, d_0)) \wedge$
($res \neq_{\text{obs}}$ Null $\Rightarrow_{\text{L}}$ Siguiente$((*res.definicion)$.clave$) =_{\text{obs}} s \wedge_{\text{L}}$
$(*res.definicion)$.signficado $=_{\text{obs}}$ obtener$(s, d_0))$}

var i : `nat`, n : `nodo(`α`)`	
$i \leftarrow 0$	O(1)
$n \leftarrow t$.raíz	O(1)
while $i < \|s\| \wedge$ Definido?$(n$.hijos$, s[i])$ **do**	O($\|s\|$)$\times$
$n \leftarrow$ Significado$(n$.hijos$, s[i])$	O(1)
$i \leftarrow i + 1$	O(1)
end while	
if $i = \|s\|$ **then**	O(1)
$res \leftarrow n$	O(1)
else	
$res \leftarrow$ Null	O(1)
end if	
	O($\|s\|$)

CrearNodo(**in/out** t : `trie(`α`)`, *in* s : `string`)
$\longrightarrow$ res : `nodo(`α`)`
Pre $\equiv$ {Rep(t)}
Post $\equiv$ {existeNodo$(t, s) \wedge_{\text{L}}$ $res =_{\text{obs}}$ obtenerNodo(t, s)}

var i : `nat`, n : `nodo(`α`)`, *iter* : `itDicc(char, nodo(`α`))`	
$i \leftarrow 0$	O(1)
$res \leftarrow t$.raíz	O(1)
while $i < \|s\| \wedge$ Definido?$(res$.hijos$, s[i])$ **do**	O($\|s\|$)$\times$
$res \leftarrow$ Significado$(res$.hijos$, s[i])$	O(1)
$i \leftarrow i + 1$	O(1)
end while	
n.hijos $\leftarrow$ Vacío()	O(1)
n.definición $\leftarrow$ Null	O(1)
while $i < \|s\|$ **do**	O($\|s\|$)$\times$
iter $\leftarrow$ DefinirRápido$(res$.hijos$, s[i], n)$	O(1)
$res \leftarrow$ SiguienteSignificado(*iter*)	O(1)
end while	
	O($\|s\|$)