

Algoritmos y Estructuras de Datos II

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

Trabajo Práctico de Especificación

Grupo 1

Integrante	LU	Correo electrónico
Bálsamo, Facundo	874/10	facundobalsamo@gmail.com
Lasso, Nicolás	892/10	lasso.nico@gmail.com
Rodríguez, Agustín	120/10	agustinrodriguez90@hotmail.com
Tripodi, Guido	843/10	guido.tripodi@hotmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

1. TAD LINKLINKIT

TAD LINKLINKIT

géneros **lli**

exporta generadores, categorias, links, categoriaLink, fechaActual, fechaUltimoAcceso, accesosRecientesDia, esReciente?, accesosRecientes, linksOrdenadosPorAccesos, cantLinks

usa BOOL, NAT, CONJUNTO, SECUENCIA, ARBOLCATEGORIAS

observadores básicos

categorias	: lli s	\longrightarrow acat	
links	: lli s	\longrightarrow conj(link)	
categoriaLink	: lli \times link	\longrightarrow categoria	
fechaActual	: lli	\longrightarrow fecha	
fechaUltimoAcceso	: lli $s \times$ link l	\longrightarrow fecha	$\{l \exists links(s)\}$
accesosRecientesDia	: lli $s \times$ link $l \times$ fecha f	\longrightarrow nat	

generadores

iniciar	: acat ac	\longrightarrow lli	
nuevoLink	: lli $s \times$ link $l \times$ categoria c	\longrightarrow lli	$\{\neg(l \exists links(s)) \wedge esta?(c, categorias(s))\}$
acceso	: lli $s \times$ link $l \times$ fecha f	\longrightarrow lli	$\{l \exists links(s) \wedge f \geq fechaActual(s)\}$

otras operaciones

esReciente?	: lli $s \times$ link $l \times$ fecha f	\longrightarrow bool	$\{l \exists links(s)\}$
accesosRecientes	: lli $s \times$ categoria $c \times$ link l	\longrightarrow nat	$\{esta?(c, categorias(s)) \wedge l \exists links(s) \wedge esSubCategoria(categorias(s), c, categoriaLink(s, l))\}$
linksOrdenadosPorAccesos	: lli $s \times$ categoria c	\longrightarrow secu(link)	$\{esta?(c, categorias(s))\}$
cantLinks	: lli $s \times$ categoria c	\longrightarrow nat	$\{esta?(c, categorias(s))\}$
menorReciente	: lli $s \times$ link l	\longrightarrow fecha	$\{l \exists links(s)\}$
diasRecientes	: lli $s \times$ link l	\longrightarrow fecha	$\{l \exists links(s)\}$
diasRecientesDesde	: lli $s \times$ link l	\longrightarrow fecha	$\{l \exists links(s)\}$
linksCategoriasOHijos	: lli $s \times$ categoria c	\longrightarrow conj(link)	$\{esta?(c, categorias(s))\}$
filtrarLinksCategoriaOHijos	: lli $s \times$ categoria $c \times$ conj(link) ls	\longrightarrow conj(link)	$\{esta?(c, categorias(s)) \wedge ls \subseteq links(s)\}$
diasRecientesParaCategoria	: lli $s \times$ categoria c	\longrightarrow conj(fecha)	$\{esta?(c, categorias(s))\}$
linkConUltimoAcceso	: lli $s \times$ categoria $c \times$ conj(link) ls	\longrightarrow link	$\{esta?(c, categorias(s)) \wedge \neg \emptyset?(ls) \wedge ls \subseteq linksCategoriasOHijos(s, c)\}$
sumarAccesosRecientes	: lli $s \times$ link $l \times$ conj(fecha) fs	\longrightarrow nat	$\{l \exists links(s) \wedge fs \subseteq diasRecientes(s, l)\}$
linksOrdenadosPorAccesosAux	: lli $s \times$ categoria $c \times$ conj(link) ls	\longrightarrow secu(link)	$\{esta?(c, categorias(s)) \wedge ls \subseteq linksCategoriasOHijos(s, c)\}$
linkConMasAccesos	: lli $s \times$ categoria $c \times$ conj(link) ls	\longrightarrow link	$\{esta?(c, categorias(s)) \wedge ls \subseteq linksCategoriasOHijos(s, c)\}$
β	: bool b	\longrightarrow nat	

axiomas $\forall it, it': linklinkIT$
 $\forall a: arbolDeCategorias$
 $\forall c: categoria$
 $\forall l: link$
 $\forall f: fecha$
 $\forall cc: conj(categoria)$

$\text{categorias}(\text{iniciar}(\text{ac})) \equiv \text{ac}$

$\text{categorias}(\text{nuevoLink}(\text{s}, \text{l}, \text{c})) \equiv \text{categorias}(\text{ac})$

$\text{categorias}(\text{acceso}(\text{s}, \text{l}, \text{f})) \equiv \text{categorias}(\text{ac})$

$\text{links}(\text{iniciar}(\text{ac})) \equiv \emptyset$

$\text{links}(\text{nuevoLink}(\text{s}, \text{l}, \text{c})) \equiv \text{Ag}(\text{l}, \text{links}(\text{s}))$

$\text{links}(\text{acceso}(\text{s}, \text{l}, \text{f})) \equiv \text{links}(\text{s})$

$\text{categoriaLink}(\text{nuevoLink}(\text{s}, \text{l}, \text{c}), \text{l}') \equiv \text{if } l == l' \text{ then } c \text{ else } \text{categoriaLink}(\text{s}, \text{l}') \text{ fi}$

$\text{categoriaLink}(\text{acceso}(\text{s}, \text{l}, \text{f}), \text{l}') \equiv \text{categoriaLink}(\text{s}, \text{l}')$

$\text{fechaActual}(\text{iniciar}(\text{ac})) \equiv 0$

$\text{fechaActual}(\text{nuevoLink}(\text{s}, \text{l}, \text{c})) \equiv \text{fechaActual}(\text{s})$

$\text{fechaActual}(\text{acceso}(\text{s}, \text{l}, \text{f})) \equiv \text{f}$

$\text{fechaUltimoAcceso}(\text{nuevoLink}(\text{s}, \text{l}, \text{c}), \text{l}') \equiv \text{if } l == l' \text{ then } \text{fechaActual}(\text{s}) \text{ else } \text{fechaUltimoAcceso}(\text{s}, \text{l}') \text{ fi}$

$\text{fechaUltimoAcceso}(\text{acceso}(\text{s}, \text{l}, \text{f}), \text{l}') \equiv \text{fechaUltimoAcceso}(\text{s}, \text{l}')$

$\text{menorReciente}(\text{s}, \text{l}) \equiv \max(\text{fechaUltimoAcceso}(\text{s}, \text{l}) + 1, \text{diasRecientes}) - \text{diasRecientes}$

$\text{esReciente?}(\text{s}, \text{l}, \text{f}) \equiv \text{menorReciente}(\text{s}, \text{l}) \leq \text{f} \wedge \text{f} \leq \text{fechaUltimoAcceso}(\text{s}, \text{l})$

$\text{accesoRecienteDia}(\text{nuevoLink}(\text{s}, \text{l}, \text{c}), \text{l}', \text{f}) \equiv \text{if } l == l' \text{ then } 0 \text{ else } \text{accesoRecienteDia}(\text{s}, \text{l}', \text{f}) \text{ fi}$

$\text{accesoRecienteDia}(\text{acceso}(\text{s}, \text{l}, \text{f}), \text{l}', \text{f}') \equiv \beta(l == l' \wedge f == f') + \text{if } \text{esReciente?}(\text{s}, \text{l}, \text{f}') \text{ then } \text{accesoRecienteDia}(\text{s}, \text{l}', \text{f}') \text{ else } 0 \text{ fi}$

$\text{accesosRecientes}(\text{s}, \text{c}, \text{l}) \equiv \text{sumarAccesosRecientes}(\text{s}, \text{l}, \text{diasRecientesParaCategoria}(\text{s}, \text{c}) \cap \text{diasRecientes}(\text{s}, \text{l}))$

$\text{linksOrdenadosPorAccesos}(\text{s}, \text{c}) \equiv \text{linksOrdenadosPorAccesosAux}(\text{s}, \text{c}, \text{linksCategoriaOHijos}(\text{s}, \text{c}))$

$\text{linksOrdenadosPorAccesosAux}(\text{s}, \text{c}, \text{ls}) \equiv \text{if } \emptyset?(ls) \text{ then}$

\emptyset

else

$\text{linkConMasAccesos}(\text{s}, \text{c}, \text{ls}) \bullet \text{linksOrdenadosPorAccesosAux}(\text{s}, \text{c}, \text{ls} - \text{linkConMasAccesos}(\text{s}, \text{c}, \text{ls}))$

fi

$\text{linkConMasAccesos}(\text{s}, \text{c}, \text{ls}) \equiv \text{if } \#ls == 1 \text{ then}$

$\text{dameUno}(\text{ls})$

else

$\text{if } \text{accesosRecientes}(\text{s}, \text{c}, \text{dameUno}(\text{ls})) > \text{accesosRecientes}(\text{s}, \text{c}, \text{linkConMasAccesos}(\text{s}, \text{c}, \text{sinUno}(\text{ls}))) \text{ then}$

$\text{dameUno}(\text{ls})$

else

$\text{linkConMasAccesos}(\text{s}, \text{c}, \text{sinUno}(\text{ls}))$

fi

fi

$\text{cantLinks}(\text{s}, \text{c}) \equiv \#\text{linksCategoriaOHijos}(\text{s}, \text{c})$

$\text{diasRecientes}(\text{s}, \text{l}) \equiv \text{diasRecientesDesde}(\text{s}, \text{l}, \text{menorReciente}(\text{s}, \text{l}))$

$\text{diasRecientesDesde}(\text{s}, \text{l}, \text{f}) \equiv \text{if } \text{esReciente?}(\text{s}, \text{l}, \text{f}) \text{ then } \text{Ag}(\text{f}, \text{diasRecientesDesde}(\text{s}, \text{l}, \text{f}+1)) \text{ else } \emptyset \text{ fi}$

```

linksCategoriaOHijos(s, c)  $\equiv$  filtrarLinksCategoriaOHijos(s, c, links(s))
filtrarLinksCategoriaOHijos(s, c, ls)  $\equiv$  if  $\emptyset?(ls)$  then
     $\emptyset$ 
else
    (if esSubCategoria(categorias(s),c,categoriaLink(s,dameUno(ls)))
    then
        dameUno(ls)
    else
         $\emptyset$ 
    fi)  $\cup$  filtrarLinksCategoriaOHijos(s, c, siunUno(ls))
fi
diasRecientesParaCategoria(s, c)  $\equiv$  if  $\emptyset?(linksCategoriaOHijos(s,c))$  then
     $\emptyset$ 
else
    diasRecientes(s, linkConUltimoAcceso(s, c, linksCategoriaOHijos(s,c)))
fi
sumarAccesosRecientes(s, l, fs)  $\equiv$  if  $\emptyset?(fs)$  then
    0
else
    accesosRecientesDia(s, l, dameUno(f)) + sumarAccesosRecientes(s, l,
    sinUno(fs))
fi
 $\beta(b) \equiv$  if b then 1 else 0 fi

```

Fin TAD

1.0.1. Modulo de linkLinkIT

generos: *lli*
usa: bool, nat, conjunto, secuencia, arbolCategorias
se explica con: TAD linkLinkIT
géneros: lli

1.0.2. Operaciones Básicas

categorias (in s: lli) \longrightarrow res: ac

Pre \equiv true
Post \equiv res=_{obs} categorias(s)
Complejidad : $O(\#categorias(s))$
Descripción : Devuelve el arbol de categorias con todas las categorias del sistema
Aliasing:ALGO

links (in s: estrLLI) \longrightarrow res: conj(link)

Pre \equiv true
Post \equiv res=_{obs} links(s)
Complejidad : $O(\#links(s))$
Descripción : Devuelve todos los links del sistema
Aliasing:ALGO

categoriaLink (in s: estrLLI, in l: link) \longrightarrow res: categoria

Pre \equiv true
Post \equiv res=_{obs} categoriaLink(s,l)
Complejidad : $O(\text{cuanto seria esto? todos los links?})$

Descripción : Devuelve la categoría del link ingresado

Aliasing:ALGO

fechaActual (in s: estrLLI) \longrightarrow res: fecha

Pre \equiv true

Post \equiv res=_{obs} fechaActual(s)

Complejidad : O(1)

Descripción : Devuelve la fecha actual

Aliasing:ALGO

fechaUltimoAcceso (in s: estrLLI, in l: link) \longrightarrow res: fecha

Pre \equiv l \in links(s)

Post \equiv res=_{obs} fechaUltimoAcceso(s,l)

Complejidad : O(1)

Descripción : Devuelve la fecha de ultimo acceso al link

Aliasing:ALGO

accesosRecientesDia (in s: lli, in l: link, in f: fecha) \longrightarrow res: nat

Pre \equiv l \in links(s)

Post \equiv res=_{obs} accesosRecientesDia(s,l,f)

Complejidad : O(#accesosRecientesDia(s,l,f))

Descripción : Devuelve la cantidad de accesos a un link un cierto dia

Aliasing:ALGO

iniciar (in ac: estrAC) \longrightarrow res: lli

Pre \equiv true

Post \equiv res=_{obs} iniciar(ac)

Complejidad : O(#categorias(ac))

Descripción : crea un sistema dado un arbol ac de categorias

Aliasing:ALGO

nuevoLink (in/out s: lli, in l: link , in c: categoria)

Pre \equiv c \in categorias(s) \wedge s₀ =_{obs} s

Post \equiv s=_{obs} nuevoLink(s₀,l,c)

Complejidad : O(|l|+|c|+h)

Descripción : Agregar un link al sistema

Aliasing:ALGO

acceso (in/out s: lli, in l: link , in f: fecha)

Pre \equiv l \in links(s) \wedge f \geq fechaActual(s) \wedge s₀ =_{obs} s

Post \equiv s=_{obs} acceso(s₀,l,f)

Complejidad : O(|l|)

Descripción : Acceder a un link del sistema

Aliasing:ALGO

esReciente? (in s: lli, in l: link , in f: fecha) \longrightarrow res: bool

Pre \equiv l \in links(s)

Post \equiv res=_{obs} esReciente?(s,l,f)

Complejidad : O(y esto q es??)

Descripción : Chequea si el acceso fue reciente

Aliasing:ALGO

accesosRecientes (in s: lli, in c: categoria in l: link) \longrightarrow res: nat

Pre $\equiv c \in \text{categorias}(s) \wedge l \in \text{links}(s)$
Post $\equiv \text{res} =_{\text{obs}} \text{accesosRecientes}(s, c, l)$
Complejidad : $O(1)$
Descripción : Devuelve la cantidad de accesos recientes del link ingresado
Aliasing:ALGO

linksOrdenadosPorAccesos (**in** s: lli, **in** c: categoria) \longrightarrow res: secu(link)

Pre $\equiv c \in \text{categorias}(s)$
Post $\equiv \text{res} =_{\text{obs}} \text{linksOrdenadosPorAccesos}(s, c)$
Complejidad : $O(n^2)$
Descripción : Devuelve la cantidad de accesos recientes del link ingresado
Aliasing:ALGO

cantlinks (**in** s: lli, **in** c: categoria) \longrightarrow res: nat

Pre $\equiv c \in \text{categorias}(s)$
Post $\equiv \text{res} =_{\text{obs}} \text{cantlinks}(s, c)$
Complejidad : $O(|c|)$
Descripción : Devuelve la cantidad de links de la categoria c
Aliasing:ALGO

menorReciente (**in** s: lli, **in** l: link) \longrightarrow res: fecha

Pre $\equiv l \in \text{links}(s)$
Post $\equiv \text{res} =_{\text{obs}} \text{menorReciente}(s, l)$
Complejidad : $O(\text{no tengo idea})$
Descripción : Devuelve la fecha menor mas reciente
Aliasing:ALGO

diasRecientes (**in** s: lli, **in** l: link) \longrightarrow res: fecha

Pre $\equiv l \in \text{links}(s)$
Post $\equiv \text{res} =_{\text{obs}} \text{diasRecientes}(s, l)$
Complejidad : $O(1)$
Descripción : Devuelve la fecha reciente del link
Aliasing:ALGO

diasRecientesDesde (**in** s: lli, **in** l: link) \longrightarrow res: fecha

Pre $\equiv l \in \text{links}(s)$
Post $\equiv \text{res} =_{\text{obs}} \text{diasRecientesDesde}(s, l)$
Complejidad : $O(1)$
Descripción : Devuelve la fecha reciente del link
Aliasing:ALGO

diasRecientesParestrACegorias (**in** s: lli, **in** c: categoria) \longrightarrow res: conj(fecha)

Pre $\equiv c \in \text{categorias}(s)$
Post $\equiv \text{res} =_{\text{obs}} \text{diasRecientesParaCategorias}(s, c)$
Complejidad : $O(\text{es la cantidad de accesos recientes esto??})$
Descripción : Devuelve el conjunto de fechas recientes de la categoria c
Aliasing:ALGO

linkConUltimoAcceso (**in** s: lli, **in** c: categoria, **in** ls: conj(link)) \longrightarrow res: link

Pre $\equiv c \in \text{categorias}(s) \wedge \text{esVacia??}(ls) \wedge ls \subseteq \text{linksCategoriasOHijos}(s, c)$
Post $\equiv \text{res} =_{\text{obs}} \text{linkConUltimoAcceso}(s, c, ls)$
Complejidad : $O(\#ls??)$

Descripción : Devuelve el link que se accedió por ultima vez del conjunto ls

Aliasing:ALGO

sumarAccesosRecientes (in s: lli, in l: link,in fs: conj(fecha)) \longrightarrow res: nat

Pre $\equiv l \in \text{links}(s) \wedge fs \subseteq \text{diasRecientes}(s,l)$

Post $\equiv \text{res} =_{\text{obs}} \text{sumarAccesosRecientes}(s,l,fs)$

Complejidad : $O(1?)$

Descripción : Devuelve la suma de todos los accesos recientes del link l

Aliasing:ALGO

linkConMasAccesos (in s: lli, in c: categoria,in ls: conj(link)) \longrightarrow res: link

Pre $\equiv c \in \text{categorias}(s) \wedge ls \subseteq \text{linksCategoriasOHijos}(s,c)$

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

Complejidad : $O(1?)$

Descripción : Devuelve al link con mas accesos

Aliasing:ALGO

1.1. Pautas de Implementación

1.1.1. Estructura de Representación

linkLinkIT se representa con estrILL donde estrILL es:

tupla (
 arbolCategorias: acat,
 actual:nat,
 accesosXLink: diccTrie(link:string,puntero(datosLink)),
 listaLinks:Lista(datosLink), *arrayCatLinks*:arreglo-dimen(linksFamilia))

Donde datosLink es:

tupla \langle link:link, catDLinkpuntero(datosCat),*accesosRecientes*:Lista(acceso) \rangle

Donde acceso es:

tupla \langle dia:nat, cantAccesos:nat \rangle

Donde linksFamilia es:

lista (puntero(datosLink))

1.1.2. Invariante de Representación

1. Para todo '*link*' que exista en '*accesosXLink*' la '*catDLink*' de la tupla apuntada en el significado debera existir en '*arbolCategorias*'.
2. Para todo '*link*' que exista en '*accesosXLink*', todos los '*dia*' de la lista '*accesosRecientes*' deberan ser menor o igual a *actual*.
3. '*actual*' serã igual a la fecha mas grande de *accesosRecientes* de todas las claves *accesosXLink*.
4. Para todo '*link*' que exista en '*accesosXLink*' su significado deberã existir en '*listaLinks*'
5. Para todo '*link*' que exista en '*accesosXLink*' su significado deberã aparecer en '*arrayCantLinks*' en la posicion igual al id de '*catDLink*' y en todas las posiciones menores a esta.
6. Para todo '*link*' que exista en '*accesosXLink*', la '*accesosRecientes*' apuntada en el significado debera tener una longitud menor o igual a 3.
- 7.

Rep : $\text{estrLLI} \rightarrow \text{bool}$
 $\text{Rep}(e) \equiv \text{true} \iff$

1. $(\forall x: \text{link}) (\text{def?}(x, e.\text{accesosXLink})) \leftrightarrow (*\text{obtener}(x, e.\text{accesosXLink})).\text{catDLink} \exists \text{todasLasCategorias}(e.\text{arbolCategorias}, \text{categorias})$
2. $(\forall x: \text{link}) (\text{def?}(x, e.\text{accesosXLink})) \rightarrow (\text{ultimo}((*\text{obtener}(x, e.\text{accesosXLink})).\text{accesosRecientes})).\text{dia} \leq e.\text{actual}$
- 3.
4. $(\forall x: \text{link}) (\text{def?}(x, e.\text{accesosXLink})) \rightarrow (*\text{obtener}(x, e.\text{accesosXLink})) \exists \text{todosLosLinks}(\text{listaLinks})$
5. $(\forall x: \text{link}) (\text{def?}(x, e.\text{accesosXLink})) \rightarrow (*\text{obtener}(x, e.\text{accesosXLink})) \exists \text{linksDeCat}(e.\text{arrayCantLinks}[\text{id}(e.\text{arbolCategorias}, x)])$
6. $(\forall x: \text{link}) (\text{def?}(x, e.\text{accesosXLink})) \rightarrow \text{longitud}((*\text{obtener}(x, e.\text{accesosXLink})).\text{accesosRecientes}) \leq 3$

1.1.3. Función de Abstraccion

Abs: $\text{estrLLI } e \rightarrow \text{linkLinkIT}$
 $\text{Abs}(e) =_{\text{obs}} s: \text{linkLinkIT} \mid$

$$\begin{aligned} & \text{categorias}(s) = e.\text{arbolCategorias} \wedge \\ & \text{links}(s) = \text{todosLosLinks}(s.\text{listaLinks}) \wedge \\ & \forall l: \text{link } \text{categoriaLink}(s, l) = *((\text{obtener}(l, e.\text{accesosXLink})).\text{catDLink} \wedge \\ & \quad \text{fechaActual}(s) = e.\text{actual} \wedge \\ & \quad \forall l: \text{link } l \in \text{links}(l) \wedge_L \text{fechaUltimoAcceso}(s, l) = \text{ultimo}((*(\text{obtener}(s, e.\text{accesosXLink})).\text{accesos}).\text{dia}) \wedge \\ & \quad \forall l: \text{link } \forall f: \text{nat } \text{accesoRecienteDia}(s, l, f) = \text{cantidadPorDia}(f, *(\text{obtener}(s, e.\text{accesosXLink})).\text{accesos}) \end{aligned}$$

Auxiliares

$\text{cantidadPorDia} : \text{fecha} \times \text{lista}(\text{acceso}) \rightarrow \text{nat}$
 $\text{cantidadPorDia}(f, ls) \equiv \text{if } f == (\text{prim}(ls)).\text{dia} \text{ then } \text{cantAccesos} \text{ else } \text{cantidadPorDia}(f, \text{fin}(ls)) \text{ fi}$
 $\text{listaLinks} : \text{secu}(\text{datosLink}) \rightarrow \text{conj}(\text{link})$
 $\text{listaLinks}(ls) \equiv \text{Ag}((\text{prim}(ls)).\text{link}, \text{fin}(ls))$

1.1.4. Algoritmos

Algoritmo: 1

ICATEGORIAS (**in** $s: \text{lli}$) $\rightarrow \text{res: ac}$

$\text{res} \leftarrow s.\text{arbolCategorias}$ //O(1)

Complejidad: O(1)

Algoritmo: 2

ILINKS (**in** $s: \text{estrLLI}$) $\rightarrow \text{res: conj}(\text{link})$

$\text{res} \leftarrow \text{claves}(s.\text{accesosXLink})$ //O(1)

Complejidad: O($\sum_{i=1}^{\text{longitud}(s.\text{listaLinks})}$)

Algoritmo: 3

ICATEGORIALINK (**in** $s: \text{estrLLI}$, **in** $l: \text{link}$) $\rightarrow \text{res: categoria}$


```
res ← *((obtener(l,s.accesosXLink))).catDLink //O(|l|)
```

Complejidad: $O(|l|)$

Algoritmo: 4

IFECHAACTUAL (in s: estrLLI) \rightarrow res: fecha

```
res ← s.actual //O(1)
```

Complejidad: $O(1)$

Algoritmo: 5

IFECHAULTIMOACCESO (in s: estrLLI, in l: link) \rightarrow res: fecha

```
res ← ultimo*((obtener(l,s.accesosXLink))).accesosRecientes.dia //O(|l|)
```

Complejidad: $O(|l|)$

Algoritmo: 6

IACCESOSRECIENTESDIA (in s: estrLLI, in l: link, in f: fecha) \rightarrow res: nat

```
lista(acceso) accesos ← vacia() //O(1)
```

```
res ← 0 //O(1)
```

```
accesos ← *((obtener(l,s.accesosXLink))).accesosRecientes //O(|l|)
```

```
while(¬esVacia?(accesos) ∧ (prim(accesos)).dia ≠ f) //O(1)
```

```
if (prim(accesos)).dia == f //O(1)
```

```
then res = (prim(accesos)).cantAccesos //O(1)
```

```
else accesos = fin(accesos) FI //O(1) //O(ALGO)
```

```
end while //O(1)
```

Complejidad:

Algoritmo: 7

IINICIAR (in ac: acat) \rightarrow res: estrLLI

```
res.actual ← 1
```

```

//O(1)

res.arbolCategorias ← ac //O(1)

var c: nat //O(1)

c ← 1 //O(1)

res.arrayCantLinks ← crearArreglo(#categorias(ac)) //O(1)

res.listaLinks ← vacia() //O(1)

res.accesosXLink ← vacio() //O(1)

while (c ≤ #categorias(ac)) //O(1)

linksFamilia llist ← vacia() //O(1)

res.arrayCatLinks[c] ← llist //O(1)

c ++ //O(1)

end while //O(1)

```

Complejidad: ($\#categorias(ac)$)

Algoritmo: 8

INUEVOLINK (in/out s: lli, in l: link , in c: categoria)

```

puntero(datosCat) cat ← obtener(c,s.arbolCategorias) //O(|c|)

lista(acceso) accesoDeNuevoLink ← vacia() //O(1)

datosLink nuevoLink ← <l,cat,accesoDeNuevoLink> //O(1)

puntero(datosLink) puntLink ← nuevoLink //O(1)

definir(l,puntLink,s.accesosXLink) //O(|l|)

agregarAtras(s.listaLinks,puntLink) //O(1)

while(cat ≠ puntRaiz(s.arbolCategorias)) //O(1)

agregarAtras(s.arrayCatLinks[(cat).id],puntLink) //O(1)

cat ← cat.padre //O(1)

end while

agregarAtras(s.arrayCatLinks[(cat).id],puntLink) //O(1)

```

Complejidad: $O(|c|+|l|+h)$

Algoritmo: 9

IACCESO (in/out s: lli, in l: link , in f: fecha)

```

if s.actual == f

```

```

//O(1)
then s.actual  $\leftarrow$  s.actual //O(1)
else s.actual  $\leftarrow$  f FI //O(1)
var puntero(datosLink) puntLink  $\leftarrow$  obtener(l,s.accesosXLink) //O(|l|)
if (ultimo((*puntLink).accesos)).dia == f //O(1)
then (ultimo((*puntLink).accesos)).cantAccesos++ //O(1)
else agregarAtras((*puntLink).accesos), f) FI //O(1)
if longitud((*puntLink).accesos) == 4 //O(1)
then fin((*puntLink).accesos) //O(1)
fi

```

Complejidad: $O(|l|)$

Algoritmo: 10

IESRECIENTE? (in s: lli, in l: link , in f: fecha) \rightarrow res: bool

```

res  $\leftarrow$  menorReciente(s,l)  $\leq$  f  $\wedge$  f  $\leq$  fechaUltimoAcceso(s,l) //O(ALGO)

```

Complejidad: $O(ALGO)$

Algoritmo: 11

IACCESOSRECIENTES (in s: lli, in c: categoria in l: link) \rightarrow res: nat

```

res  $\leftarrow$  sumarAccesosRecientes(s, l, diasRecientesParaCategoria(s, c)  $\cap$  diasRecientes(s, l)) //O(ALGO)

```

Complejidad: $O(ALGO)$

Algoritmo: 12

ILINKSORDENADOSPORACCESOS (in s: lli, in c: categoria) \rightarrow res: lista(link)

```

nat id  $\leftarrow$  id(s.arbolCategorias,c) //O(1)

```

```

lista(puntero(datosLink)) listaOrdenada  $\leftarrow$  vacia() //O(1)

```

```

itLista(puntero(datosLink)) itMax  $\leftarrow$  crearIt(s.arrayCantLinks[id]) //O(1)

```

```

if  $\neg$ iestaOrdenada?(s.arrayCantLinks[id]) //O(1)

```

```

then

```

```

while(haySiguiente?(s.arrayCantLinks[id])) //O(1)

```

```

itMax  $\leftarrow$  iBuscarMax(s.arrayCantLinks[id]) //O(n)

```

```

agregarAtras(listaOrdenada,siguiente(itMax))

```

```

//O(1)
eliminarSiguiente(itMax) //O(1)
fi
end while

```

Complejidad: $O(n^2)$

Algoritmo: 13

```

IBUSCARMAX (in ls: lista(punero(datosLink))) → res: itLista(punero(datosLink))

res ← crearIt(ls) //O(1)
itLista(punero(datosLink)) itRecorre ← crearIt(ls) //O(1)
nat max ← (*siguiente(itRecorre)).cantAccesosRecientes //O(1)
while(haySiguiente(itRecorre)) //O(1)
if max < (*siguiente(itRecorre)).cantAccesosRecientes //O(1)
then //O(1)
max ← (*siguiente(itRecorre)).cantAccesosRecientes //O(1)
res ← itRecorre //O(1)
end while
avanzar(itRecorre) //O(1)
end while //O(1)

```

Complejidad: $O(n)$

Algoritmo: 14

```

IESTAORDENADA (in ls: lista(punero(datosLink))) → res: bool

res ← true //O(1)
itLista(punero(datosLink)) itRecorre ← crearIt(ls) //O(1)
nat aux ← (*siguiente(itRecorre)).cantAccesosRecientes //O(1)
while(haySiguiente(itRecorre) ∧ res == true) //O(1)
avanzar(itRecorre) //O(1)
if aux < (*siguiente(itRecorre)).cantAccesosRecientes //O(1)
then //O(1)
res ← false //O(1)
fi

```

```

//O(1)
aux ← (*siguiente(itRecorre)).cantAccesosRecientes //O(1)
end while //O(1)

```

Complejidad: $O(n)$

Algoritmo: 15

ICANTLINKS (in s: lli, in c: categoria) \rightarrow res: nat

```

puntero(datosCat) cat ← obtener(c,s.arbolCategorias) //O(|c|)
res = longitud(arrayCantLinks[(cat).id]) //O(1)

```

Complejidad: $O(|c|)$

Algoritmo: 16

IMENORRECIENTE (in s: lli, in l: link) \rightarrow res: fechaend while

```

res ← max(fechaUltimoAcceso(s,l)+1,diasRecientes) - diasRecientes //O(1)

```

Complejidad: $O(1)$

Algoritmo: 17

IDIASRECIENTES (in s: lli, in l: link) \rightarrow res: conj(fecha)end while

```

res ← diasRecientesDesde(s,l,menorReciente(s,l)) //O(ALGO)

```

Complejidad: $O(ALGO)$

Algoritmo: 18

IDIASRECIENTESDESDE (in s: lli, in l: link, in f: fecha) \rightarrow res: conj(fecha)end while

```

while(esReciente?(s,l,f)) //O(1)
  Agregar(f,res) //O(1)
  fecha++ //O(1)
end while

```

Complejidad: $O(\star 1)$

Algoritmo: 19

IDIASRECIENTESPARACATEGORIAS (in s: lli, in c: categoria) \rightarrow res: conj(fecha)end while

```
itLista(puntero(datosLink)) links ← crearIt(arrayCatLinks[id(s.arbolCategorias,c)] //O(1)
```

```
diasRecientes(s,linkConUltimoAcceso(s,c,links)) //O(VER COMO ESCRIBIR O DE LA LONGITUD DE UN LINK)
```

Complejidad: O(ALGO)

Algoritmo: 20

ISUMARACCESOSRECIENTES (in s: lli, in l: link,in fs: conj(fecha)) → res: natend while

```
conjFecha ← fs //O(ALGO)
```

```
while(!∅?(conjFecha)) //O(1)
```

```
res ← accesosRecientesDia(s,l,dameUno(conjFecha)) //O(ALGO)
```

```
conjFecha ← sinUno(conjFecha) //O(ALGO)
```

```
end while
```

Complejidad: O(ALGO)

Algoritmo: 21

ILINKCONULTIMOACCESO (in s: lli, in c: categoria,in ls: itLista(puntero(datosLink)) → res: linkend while

```
puntero(datosLink) max ← (siguiente(ls)) //O(1)
```

```
while(!haySiguiente(ls)) //O(1)
```

```
avanzar(ls) //O(1)
```

```
if (ultimo((*max).accesosRecientes)).dia < (ultimo((*siguiente(ls)).accesosRecientes)).dia //O(1)
```

```
then max ← (siguiente(ls)) //O(1)
```

```
fi //O(1)
```

```
end while
```

```
res ← (*max).link //O(|(*max).link|)
```

Complejidad: O(|(*max).link|)

1.2. Descripcion de Complejidades de Algoritmos

- 1.
- 2.
- 3.

- 4.
- 5.
- 6.
- 7.
- 8.
- 9.
- 10.
- 11.
- 12.
- 13.
- 14.
- 15.
- 16.
- 17.
- 18.
- 19.
- 20.
- 21.

2. TAD ARBOLDECATEGORIAS

TAD ARBOLDECATEGORIAS

géneros `acat`

exporta generadores, categorias, raíz, padre, id, altura, esta?, esSubCategoria, alturaCategoria, hijos

usa `BOOL`, `NAT`, `CONJUNTO`

observadores básicos

`categorias` : `acat ac` \longrightarrow `conj(categoria)`

`raíz` : `acat ac` \longrightarrow `categoria`

`padre` : `acat ac` \times `categoria h` \longrightarrow `categoria` $\{esta?(h, ac) \wedge raíz(ac) \neq h\}$

`id` : `acat ac` \times `categoria c` \longrightarrow `nat` $\{esta?(c, ac)\}$

generadores

`nuevo` : `categoria c` \longrightarrow `acat` $\{\neg vacia?(c)\}$

`agregar` : `acat ac` \times `categoria c` \times `categoria h` \longrightarrow `acat` $\{esta?(c, ac) \wedge \neg vacia?(h) \wedge \neg esta?(h, ac)\}$

otras operaciones

`altura` : `acat ac` \longrightarrow `nat`

`esta?` : `categoria c` \times `acat ac` \longrightarrow `bool`

`esSubCategoria` : `acat ac` \times `categoria c` \times `categoria h` \longrightarrow `bool` $\{esta?(c, ac) \wedge esta?(h, ac)\}$

`alturaCategoria` : `acat ac` \times `categoria c` \longrightarrow `nat` $\{esta?(c, ac)\}$

`hijos` : `acat ac` \times `categoria c` \longrightarrow `conj(categoria)` $\{esta?(c, ac)\}$

axiomas $\forall a: \text{arbolDeCategorias}$
 $\forall c: \text{categoria}$

$\forall ca: \text{conj}(\text{arbolDeCategoria})$
 $\forall cc: \text{conj}(\text{categoria})$

$\text{categorias}(\text{nuevo}(c)) \equiv c$
 $\text{categorias}(\text{agregar}(ac, c, h)) \equiv \text{Ag}(h, \text{categorias}(ac))$

$\text{raiz}(\text{nuevo}(c)) \equiv c$
 $\text{raiz}(\text{agregar}(ac, c, h)) \equiv \text{raiz}(ac)$

$\text{padre}(\text{agregar}(ac, c, h), h') \equiv \text{if } h == h' \text{ then } c \text{ else } \text{padre}(ac, c, h') \text{ fi}$

$\text{id}(\text{nuevo}(c), c') \equiv 1$
 $\text{id}(\text{agregar}(ac, c, h), h') \equiv \text{if } h == h' \text{ then } \# \text{categorias}(ac) + 1 \text{ else } \text{id}(ac, h') \text{ fi}$

$\text{altura}(\text{nuevo}(c)) \equiv \text{alturaCategoria}(\text{nuevo}(c), c)$
 $\text{altura}(\text{agregar}(ac, c, h)) \equiv \max(\text{altura}(ac), \text{alturaCategoria}(\text{agregar}(ac, c, h), h))$
 $\text{alturaCategoria}(ac, c) \equiv \text{if } c == \text{raiz}(ac) \text{ then } 1 \text{ else } 1 + \text{alturaCategoria}(ac, \text{padre}(ac, c)) \text{ fi}$

$\text{esta?}(c, ac) \equiv c \in \text{categorias}(ac)$

$\text{esSubCategoria}(ac, c, h) \equiv c == h \vee L(h = \text{raiz}(ac) \wedge L \text{ esSubCategoria}(ac, c, \text{padre}(ac, h)))$

$\text{hijos}(\text{nuevo}(c1), c2) \equiv \emptyset$
 $\text{hijos}(\text{agregar}(ac, c, h), c') \equiv \text{if } h == c' \text{ then } \emptyset \text{ else } (\text{if } c == c' \text{ then } h \text{ else } \emptyset \text{ fi}) \cup \text{hijos}(ac, c, c') \text{ fi}$

Fin TAD

2.0.1. Modulo de Arbol de Categorías

generos: *acat*
usa: bool, nat, conjunto
se explica con: TAD ArbolDeCategorías
géneros: *acat*

2.0.2. Operaciones Básicas

categorias (in $ac: \text{acat}$) \longrightarrow res: $\text{conj}(\text{categoria})$
Pre \equiv true
Post $\equiv \text{res} =_{\text{obs}} \text{categorias}(ac)$
Complejidad : $O(\# \text{categorias}(ac))$
Descripción : Devuelve el conjunto de categorías de un ac
Aliasing: ALGO

raiz (in ac: acat) \longrightarrow res: categoria

Pre \equiv true

Post \equiv res=_{obs} raiz(ac)

Complejidad : $O(1)$

Descripción : Devuelve la raiz del arbol ac

Aliasing:ALGO

padre (in ac: estrAC, in h: categoria) \longrightarrow res: categoria

Pre \equiv $h \in ac \wedge \text{raiz}(ac) \neq h$

Post \equiv res=_{obs} padre(ac,h)

Complejidad : $O(ni \text{ idea})$

Descripción : Devuelve el padre de una categoria

Aliasing:ALGO

id (in ac: estrAC, in c: categoria) \longrightarrow res:nat

Pre \equiv $h \in ac$

Post \equiv res=_{obs} id(ac,c)

Complejidad : $O(|c|)$

Descripción : Devuelve el id de una categoria c en el arbol ac

Aliasing:ALGO

nuevo (in c: categoria) \longrightarrow res:estrAC

Pre \equiv $\neg \text{vacía?}(c)$

Post \equiv res=_{obs} nuevo(c)

Complejidad : $O(|c|)$

Descripción : Crea un arbol

Aliasing:ALGO

agregar (in/out ac: estrAC, in c: categoria, in h: categoria)

Pre \equiv $c \in ac \wedge \neg \text{vacía?}(h) \wedge ac_0 =_{\text{obs}} ac$

Post \equiv ac=_{obs} agregar(ac₀,c,h)

Complejidad : $O(|c|+|h|)$

Descripción : Agrega una categoria hija a una padre

Aliasing:ALGO

altura (in ac: estrAC) \longrightarrow res:nat

Pre \equiv true

Post \equiv res=_{obs} altura(ac)

Complejidad : $O(|ac|)$

Descripción : Devuelve la altura del arbol ac

Aliasing:ALGO

esta? (in c: categoria, in ac: estrAC) \longrightarrow res:bool

Pre \equiv true

Post \equiv res=_{obs} esta?(c,ac)

Complejidad : $O(|ac|)$

Descripción : Devuelve si esta o no en el arbol la categoria c

Aliasing:ALGO

esSubCategoria (in ac: estrAC, in c: categoria, in h: categoria) \longrightarrow res:bool

Pre \equiv esta?(c,ac) \wedge esta?(h,ac)

Post \equiv res=_{obs} esSubCategoria(ac,c,h)

Complejidad : $O(\text{no tengo idea})$
Descripción : Devuelve si c es descendiente de h
Aliasing:ALGO

alturaCategoria (**in** ac : estrAC , **in** c : categoria) \longrightarrow res:nat

Pre $\equiv \text{esta?}(c, ac)$
Post $\equiv \text{res} =_{\text{obs}} \text{alturaCategoria}(ac, c)$
Complejidad : $O(\text{no tengo idea})$
Descripción : Devuelve la altura de la categoria c
Aliasing:ALGO

hijos (**in** ac : estrAC , **in** c : categoria) \longrightarrow $\text{res:conj}(\text{categoria})$

Pre $\equiv \text{esta?}(c, ac)$
Post $\equiv \text{res} =_{\text{obs}} \text{hijos}(ac, c)$
Complejidad : $O(|c|)$
Descripción : Devuelve el conjunto de categorias hijos de c
Aliasing:ALGO

2.1. Pautas de Implementación

2.1.1. Estructura de Representación

arbolDeCategorias **se representa con** estrAC **donde** estrAC **es:**
 tupla <
 raiz: puntero(datosCat),
 cantidad: nat,
 alturaMax: nat,
 familia: $\text{diccTrie}(\text{padre: string, puntero}(\text{datosCat}))$,
 categorias: Lista(datosCat)>
Donde datosCat **es:**
 tupla <
 categoria: string,
 id: nat,
 altura: nat,
 hijos: $\text{conj}(\text{puntero}(\text{datosCat}))$,
 abuelo: v>

2.1.2. Invariante de Representación

1. Para cada '*padre*' obtener el significado devolvera un puntero(datosCat) donde '*categoria*' es igual a la clave
2. Para toda clave '*padre*' que exista en '*familia*' debera ser o raiz o pertenecer a algun conjunto de punteros de '*hijos*' de alguna clave '*padre*'
3. Todos los elementos de '*hijos*' de una clave '*padre*', cada uno de estos hijos tendran como '*abuelo*' a ese '*padre*' cuando sean clave.
4. '*cantidad*' sera igual a la longitud de la lista '*categorias*'.
5. Cuando la clave es igual a '*raiz*' la '*altura*' es 1.
6. La '*altura*' del puntero a datosCat de cada clave es menor o igual a '*alturaMax*'.
7. Existe una clave en la cual, la '*altura*' del significado de esta es igual a '*alturaMax*'.
8. Los '*hijos*' de una clave tienen '*altura*' igual a $1 + \text{'altura de la clave'}$.
9. Todos los '*id*' de significado de cada clave deberan ser menor o igual a '*cant*'.

10. No hay '*id*' repetidos en el '*familia*'.
11. Todos los '*id*' son consecutivos.

Rep : $\text{estrAC} \rightarrow \text{bool}$
 $\text{Rep}(e) \equiv \text{true} \iff$

1. $(\forall x: \text{string}) (\text{def?}(x, e.familia)) \leftrightarrow (*\text{obtener}(x, e.familia)).\text{categoria} = x$
2. $(\forall x, y: \text{string}) (\text{def?}(x, e.familia)) \leftrightarrow (x == e.raiz \vee (\text{def?}(y, e.familia)) \wedge_L x \in \text{hijosDe}(*(\text{obtener}(y, e.familia))).\text{hijos})$
3. $(\forall x, y: \text{string}) (\text{def?}(x, e.familia)) \wedge (\text{def?}(y, e.familia)) \Rightarrow_L y \in (*(\text{obtener}(x, e.familia))).\text{hijos} \iff$
 $(*(\text{obtener}(y, e.familia))).\text{abuelo}.\text{categoria} = x$
4. $e.\text{cantidad} = \text{longitud}(e.\text{categorias})$
5. $(\forall x: \text{string}) (\text{def?}(x, e.familia)) \wedge x = e.raiz \Rightarrow_L (*(\text{obtener}(x, e.familia))).\text{altura} = 1$
6. $(\forall x: \text{string}) (\text{def?}(x, e.familia)) \Rightarrow_L (*\text{obtener}(x, e.familia)).\text{altura} \leq e.\text{alturaMax}$
7. $(\exists x: \text{string}) (\text{def?}(x, e.familia)) \wedge_L (*(\text{obtener}(x, e.familia))).\text{altura} = e.\text{alturaMax}$
8. $(\forall x, y: \text{string}) (\text{def?}(x, e.familia)) \wedge (\text{def?}(y, e.familia)) \wedge_L y \in \text{hijosDe}(*(\text{obtener}(x, e.familia))).\text{hijos} \Rightarrow$
 $(*(\text{obtener}(y, e.familia))).\text{altura} = 1 + (*(\text{obtener}(x, e.familia))).\text{altura}$
9. $(\forall x: \text{string}) (\text{def?}(x, e.familia)) \Rightarrow_L (*(\text{obtener}(x, e.familia))).\text{id} \leq e.\text{cant}$
10. $(\forall x, y: \text{string}) (\text{def?}(x, e.familia)) \wedge (\text{def?}(y, e.familia)) \Rightarrow_L (*(\text{obtener}(x, e.familia))).\text{id} \neq (*(\text{obtener}(y, e.familia))).\text{id}$
11. $(\forall x: \text{string}) (\text{def?}(x, e.familia)) (\exists y: \text{string}) (\text{def?}(y, e.familia)) \iff$
 $(*(\text{obtener}(y, e.familia))).\text{id} \leq e.\text{cantidad} \wedge (*(\text{obtener}(x, e.familia))).\text{id} < e.\text{cantidad} \wedge_L$
 $(*(\text{obtener}(y, e.familia))).\text{id} = 1 + (*(\text{obtener}(x, e.familia))).\text{id}$

2.1.3. Función de Abstracción

Abs: $\text{estr } e \rightarrow \text{arbolDeCategorias}$
 $\text{Abs}(e) =_{\text{obs}} ac: \text{arbolDeCategorias} \mid$

$$\begin{aligned} \text{categorias}(ac) &= \text{todasLasCategorias}(e.\text{categorias}) \wedge_L \\ \text{raiz}(ac) &= (*e.\text{raiz}).\text{categoria} \wedge_L \\ (\forall c: \text{categoria}) \text{esta?}(c, ac) \wedge c \neq \text{raiz}(ac) &\Rightarrow_L \text{padre}(ac, c) = (*(*(\text{obtener}(c, e.familia))).\text{abuelo}).\text{categoria} \wedge_L \\ (\forall c: \text{categoria}) \text{esta?}(c, ac) &\Rightarrow_L \text{id}(ac, c) = (*(\text{obtener}(c, e.familia))).\text{id} \end{aligned}$$

Auxiliares

$\text{todasLasCategorias} : \text{secu}(\text{datosCat}) \rightarrow \text{conj}(\text{categoria})$

$\text{Ag}((\text{prim}(cs)).\text{categoria}, \text{fn}(cs)) \equiv$

2.1.4. Algoritmos

Algoritmo: 1

ICATEGORIAS (**in** $ac: \text{estrAC}$) \rightarrow $\text{res}: \text{conj}(\text{categoria})$ **end while**

$\text{res} \leftarrow \text{claves}(ac.familia)$

$//O(\text{ALGO})$

Complejidad:

Algoritmo: 2

IRAIZ (**in** $ac: \text{estrAC}$) \rightarrow $\text{res}: \text{categoria}$ **end while**

```
res ← (*ac.raiz).categoria //O(1)
```

Complejidad: O(1)

Algoritmo: 3

IPADRE (in ac: estrAC, in h: categoria) → res: puntero(categoria)end while

```
res ← (*(obtener(h,ac.familia)).abuelo).categoria //O(ALGO)
```

Complejidad:

Algoritmo: 4

IID (in ac: estrAC, in c: categoria) → res:natend while

```
res ← *(obtener(c,ac.familia)).id //O(|c|)
```

Complejidad: O(|c|)

Algoritmo: 5

INUEVO (in c: categoria) → res:estrACend while

```
res.cantidad ← 1 //O(1)
```

```
res.raiz = c //O(1)
```

```
res.alturaMax = 1 //O(1)
```

```
var tuplaA : datosCat //O(1)
```

```
var punt : puntero(datosCat) //O(1)
```

```
tuplaA ← (c,1,1,esVacia?,punt) //O(1)
```

```
punt ← puntero(tuplaA) //O(1)
```

```
res.familia = definir(padre, punt, res.familia) //O(|c|)
```

```
res.categorias ← agregarAtras(tuplaA,res.categorias) //O(1)
```

Complejidad: O(|c|)

Algoritmo: 6

IAGREGAR (in/out ac: estrAC,in c: categoria, in h: categoria)end while

```
var puntPadre : puntero(datosCat) //O(1)
```

```
puntPadre ← (obtener(c,ac.familia))
```

```

//O(|c|)

if (*puntPadre).altura == ac.alturaMax //O(1)
then ac.alturaMax = ac.alturaMax + 1 //O(1)
ELSE ac.alturaMax = ac.alturaMax FI //O(1)

var tuplaA : datosCat //O(1)

var punt : puntero(datosCat) //O(1)

tuplaA ← (h,ac.cantidad +1,(*puntPadre).altura +1,esVacia?,puntPadre) //O(|h|)

punt ← puntero(tuplaA) //O(1)

Agregar((*puntPadre).hijos,punt) //O(1)

definir(h,punt,ac.familia) //O(|h|)

ac.cantidad ++ //O(1)

agregarAtras(tuplaA,res.categorias) //O(1)

```

Complejidad: $O(|c|+|h|)$

Algoritmo: 7

IALTURA (in ac: estrAC) \rightarrow res:natend while

```

res ← ac.alturaMax //O(1)

```

Complejidad: $O(1)$

Algoritmo: 8

UESTA? (in c: categoria,in ac: estrAC) \rightarrow res:boolend while

```

res ← def?(c,ac.familia) //O(|c|)

```

Complejidad: $O(|c|)$

Algoritmo: 9

IESSUBCATEGORIA (in ac: estrAC, in c: categoria,in h: categoria) \rightarrow res:boolend while

```

var puntPadre : puntero(datosCat) //O(1)

puntPadre ← (obtener(c,ac.familia)) //O(|c|)

res ← false //O(1)

if c == ac.raiz //O(|c|)
then res ← true //O(1)

ELSE actual ← h

```

```

//O(1)

while(res  $\neq$  true  $\wedge$  actual  $\neq$  ac.raiz) //O(1)

if actual  $\in$  (*puntPadre).hijos //O(1)

then res  $\leftarrow$  true //O(1)

ELSE actual  $\leftarrow$  (*(obtener(actual,ac.familia)) ).abuelo FI FI //O(1)

```

Complejidad:

Algoritmo: 10

IAALTURACATEGORIA (in ac: estrAC, in c: categoria) \longrightarrow res: natend while

```

res  $\leftarrow$  (*(obtener(c,ac.familia))).altura //O(|c|)

```

Complejidad: $O(|c|)$

Algoritmo: 11

IHIJOS (in ac: estrAC, in c: categoria) \longrightarrow res: conj(categoria)end while

```

res  $\leftarrow$  (*obtener(c,ac.familia)).hijos // O(ALGO) PREGUNTAR!!! EN ESTE LA COMPLEJIDAD ES EL ITER-
ADOR DEVOLVEMOS EL PUNTERO?

```

Complejidad:

Algoritmo 12

IOBTENER (in c: categoria, in ac: estrAC) \longrightarrow res: puntero(datosCat)end while

```

res  $\leftarrow$  obtener(c,ac.familia) // //O(|c|)

```

Complejidad: $O(|c|)$

Algoritmo: 13

IPUNTRAIZ (in ac: estrAC) \longrightarrow res: puntero(datosCat)end while

```

res  $\leftarrow$  ac.raiz //O(1)

```

Complejidad: $O(1)$

2.2. Descripcion de Complejidades de Algoritmos

1.

- 2.
- 3.
- 4.
- 5.
- 6.
- 7.
- 8.
- 9.
- 10.
- 11.
- 12.
- 13.
- 14.

$\text{DiccTrie}(\alpha)$ se representa con estrDT , donde estrDT es $\text{Puntero}(\text{Nodo})$

Nodo es $\text{tuplaarregloarreglo}(\text{Puntero}(\text{Nodo}))[27]$, significado $\text{Puntero}(\alpha)$

2.2.1. Invariante de Representación

El Invariante Informalmente

1. No hay repetidos en arreglo de Nodo salvo por Null . Todas las posiciones del arreglo están definidas.
2. No se puede volver al Nodo actual siguiendo alguno de los punteros hijo del actual o de alguno de los hijos de estos.
3. O bien el Nodo es una hoja, o todos sus punteros hijo no-nulos llevan a hojas siguiendo su recorrido.

El Invariante Formalmente

$\mathbf{Rep} : \text{estrAC} \longrightarrow \text{bool}$
 $\text{Rep}(e) \equiv \text{true} \iff$

- 1.
- 2.
- 3.

Funciones auxiliares

EncAEstrDTEnNMov : estrDT \times estrDT \times Nat \longrightarrow Bool

EncAEstrDTEnNMov(buscado,actual,n) \equiv **if** (n = 0) **then**
 EstaEnElArregloActual?(buscado,actual,26)
else
 RecurrenciaConLosHijos(buscado,actual, n-1,26)
fi

EstaEnElArregloActual? : estrDT \times estrDT \times nat \longrightarrow Bool

EstaEnElArregloActual?(buscado,actual,n) \equiv **if** (n=0) **then**
 ((*actual).Arreglo[0] = buscado)
else
 ((*actual).Arreglo[n] = buscado) \vee (EstaEnElArregloActual?
 (buscado,actual,n-1))
fi

RecurrenciaConLosHijos : estrDT \times estrDT \times nat \times nat \longrightarrow Bool

RecurrenciaConLosHijos(buscado,actual,n,i) \equiv **if** (i = 0) **then**
 EncAEstrDTEnNMov(buscado,(*actual).Arreglo[0],n)
else
 EncAEstrDTEnNMov(buscado, (*actual).Arreglo[i],n) \vee
 (RecurrenciaConLosHijos(buscado,actual,n,i-1))
fi

SonTodosNullOLosHijosLoSon : estrDT \longrightarrow Bool

SonTodosNullOLosHijosLoSon(e) \equiv Los27SonNull(e,26) \vee BuscarHijosNull (e, 26)

Los27SonNull : estrDT \times nat \longrightarrow Bool

Los27SonNull(e,i) \equiv **if** (i = 0) **then**
 ((*e).Arreglo[0] = null)
else
 ((*e).Arreglo[i] = null) \wedge Los27SonNull(e, i-1)
fi

BuscarHijosNull : estrDT \times nat \longrightarrow Bool

BuscarHijosNull(e,i) \equiv **if** (i = 0) **then**
 ((*e).Arreglo[0] = null) \vee SonTodosNullOLosHijosLoSon((*e).Arreglo[0])
else
 (((e).Arreglo[i] = null) \vee SonTodosNullOLosHijosLoSon((*e).Arreglo[i])) \wedge
 BuscarHijosNull(e,i-1)
fi

2.2.2. Función de Abstracción

Abs: estr e \rightarrow diccT(c, α)

$$(\forall \text{clave: } c) \text{def?}(c,d) =_{\text{obs}} \text{estaDefinido?}(c,e) \wedge_L$$

Funciones auxiliares

estaDefinido? : string \times estrDT \longrightarrow bool

estaDefinido?(c,e) \equiv **if** (e==Null) **then** false **else** NodoDef?(c,*e) **fi**

NodoDef? : string \times Nodo \longrightarrow bool

NodoDef?(c,n) \equiv **if** (vacía?(c)) **then**
 true
 else
 if (n.arreglo[numero(prim(c))] \neq Null) **then**
 NodoDef?(fin(c),*(n.arreglo[numero(prim(c))]))
 else
 false
 fi
fi

numero : char \longrightarrow nat

numero(char) \equiv char - a

ObtenerS : string \times Nodo \longrightarrow α

ObtenerS(c,n) \equiv **if** (vacía?(c)) **then** *(n.significado) **else** ObtenerS(fin(c),*(n.arreglo[numero(prim(c))])) **fi**

3. Renombres

TAD CATEGORIA

es String

Fin TAD

TAD LINK

es String

Fin TAD

TAD FECHA

es Nat

Fin TAD