

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

\emptyset

else

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

fi

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

$\text{dameUno}(ls)$

else

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

$\text{dameUno}(ls)$

else

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

fi

fi

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

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

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

```

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, *catDLink*puntero(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 : estrLLI \rightarrow bool
Rep(e) \equiv true \iff

1. $(\forall x: \text{link}) (\text{def?}(x, e.\text{accesosXLink})) \leftrightarrow (*\text{obtener}(x, e.\text{accesosXLink})).\text{catDLink} \exists \text{todasLasCategorias}(e.\text{arbolCategorias}.\text{cate})$
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: estrLLI e \rightarrow linkLinkIT
Abs(e) =_{obs} s: linkLinkIT |

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

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: lli) \rightarrow res: ac

res \leftarrow s.arbolCategorias //O(1)

Complejidad: O(1)

Algoritmo: 2

ILINKS (in s: estrLLI) \rightarrow res: conj(link)

itLista iterador \leftarrow crearIt(s.listaLinks) //O(1)

while(haySiguiente(iterador)) //O(|s.listaLinks|)

agregar(res, (*siguiente(iterador).link)) //O(|l|)

avanzar(iterador) //O(1)

end while

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

Algoritmo: 3

ICATEGORIALINK (in s: estrLLI, in l: link) \rightarrow res: categoria

res \leftarrow *((obtener(l,s.accesosXLink))).catDLink

//O(|l|)

Complejidad: $O(|l|)$

Algoritmo: 4

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

res \leftarrow s.actual

//O(1)

Complejidad: $O(1)$

Algoritmo: 5

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

res \leftarrow 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 \leftarrow vacia()

//O(1)

res \leftarrow 0

//O(1)

accesos \leftarrow *((obtener(l,s.accesosXLink))).accesosRecientes

//O(|l|)

while(\neg esVacia?(accesos) \wedge res = 0)

//O(|accesos|)

if (ultimo(accesos).dia == f

//O(1)

then res \leftarrow (ultimo(accesos)).cantAccesos

//O(1)

else accesos \leftarrow fin(accesos) FI

//O(1)

end while

Complejidad: $O(|l|)$

Algoritmo: 7

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

```
    res.actual  $\leftarrow$  1 //O(1)
    res.arbolCategorias  $\leftarrow$  &ac //O(1)
    var c: nat //O(1)
    c  $\leftarrow$  1 //O(1)
    res.arrayCantLinks  $\leftarrow$  crearArreglo(#categorias(ac)) //O(1)
    res.listaLinks  $\leftarrow$  vacia() //O(1)
    res.accesosXLink  $\leftarrow$  vacio() //O(1)
    while (c  $\leq$  #categorias(ac)) //O(#categorias(ac))
        linksFamilia llist  $\leftarrow$  vacia() //O(1)
        res.arrayCatLinks[c]  $\leftarrow$  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  $\leftarrow$  obtener(c,s.arbolCategorias) //O(|c|)
    lista(acceso) accesoDeNuevoLink  $\leftarrow$  vacia() //O(1)
    datosLink nuevoLink  $\leftarrow$  <l,cat,accesoDeNuevoLink> //O(|l|)
    puntero(datosLink) puntLink  $\leftarrow$  nuevoLink //O(1)
    definir(l,puntLink,s.accesosXLink) //O(|l|)
    agregarAtras(s.listaLinks,puntLink) //O(1)
    while(cat  $\neq$  puntRaiz(s.arbolCategorias)) //O(h)
        agregarAtras(s.arrayCatLinks[(cat).id],puntLink) //O(1)
        cat  $\leftarrow$  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 ← s.actual //O(1)
    else s.actual ← f FI //O(1)
    var puntero(datosLink) puntLink ← 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 ← menorReciente(s,l) ≤ f ∧ f ≤ fechaUltimoAcceso(s,l) //O(|l|)
```

Complejidad: $O(|l|)$

Algoritmo: 11

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

```
    res ← sumarAccesosRecientes(s, l, diasRecientesParaCategoria(s, c) ∩ diasRecientes(s, l)) //O(|l|)
```

Complejidad: $O(|l|)$

Algoritmo: 12

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

```
    nat id ← id(s.arbolCategorias,c) //O(1)
```

```
    lista(puntero(datosLink)) listaOrdenada ← vacia() //O(1)
```

```
    itLista(puntero(datosLink)) itMax ← crearIt(s.arrayCantLinks[id])
```

```

//O(1)
if ¬iestaOrdenada?(s.arrayCantLinks[id]) //O(1)
then
while(haySiguiente?(s.arrayCantLinks[id])) //O(1)
itMax ← 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(puntero(datosLink))) \rightarrow res: itLista(puntero(datosLink))

```

res ← crearIt(ls) //O(1)
itLista(puntero(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(puntero(datosLink))) \rightarrow res: bool

```

res ← true //O(1)
itLista(puntero(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(|l|)

```

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(|l|)

```

Complejidad: $O(|l|)$

Algoritmo: 18

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

```

while(esReciente?(s,l,f)) //O(|l|)

```

```

  Agregar(f,res) //O(1)

```

```

  fecha++

```

//O(1)

end while

Complejidad: $O(|l|)$

Algoritmo: 19

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

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

diasRecientes(s,linkConUltimoAcceso(s,c,links)) //O($\star|l|$)

Complejidad: $O(\star|l|)$

Algoritmo: 20

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

itConj iterador \leftarrow crearIt(fs) //O(1)

while(!haySiguiente(iterador)) //O(1)

res \leftarrow accesosRecientesDia(s,l,siguiente(iterador)) //O($|l|$)

avanzar(iterador) //O(1)

end while

Complejidad: $O(|l|)$

Algoritmo: 21

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

puntero(datosLink) max \leftarrow (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 \leftarrow (siguiente(ls)) //O(1)

fi //O(1)

end while

res \leftarrow (*max).link //O($|(*max).link|$)

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

1.2. Descripcion de Complejidades de Algoritmos

1. ICATEGORIAS:

Devuelve el arbol de categorias del sistema, esto cuesta $O(1)$.

Orden Total: $O(1)=O(1)$

2. ILINKS:

Se crea un conjunto vacio, esto tarda $O(1)$. Se crea un itLista, esto tarda $O(1)$.

Se ingresa a un ciclo preguntando si haySiguiente, esto cuesta $O(1)$, se le agrega link apuntado de cada tupla de datosLink de la lista listaLinks, esto tarda $O(|l|)$, luego se avanza el it, esto cuesta $O(1)$.

Al salir del ciclo, se devuelve el conjunto.

Orden Total: $O(1)+O(1)+O(1)+(suma\ O(|l|))+O(1)=O(suma\ O(|l|))$

3. ICATEGORIALINK:

Se utiliza la operacion obtener del diccionario accesosXLink, la cual devuelve un puntero a datosLink, se devuelve lo apuntado a catDLink, esto cuesta $O(|l|)$.

Orden Total: $O(|l|)=O(|l|)$

4. IFECHAACTUAL:

Devuelve la fecha actual del sistema, esto cuesta $O(1)$.

Orden Total: $O(1)=O(1)$

5. IFECHAULTIMOACCESO:

Se utiliza la operacion obtener del diccionario accesosXLink, la cual devuelve un puntero a datosLink,

se accede a la lista accesosRecientes dentro de la tupla, se devuelve dia del ultimo elemento, esto cuesta $O(|l|)$.

Orden Total: $O(|l|)=O(|l|)$

6. IACCESOSRECIENTESDIA:

Se crea una lista de acceso vacia, esto cuesta $O(1)$. Se le guarda a la lista, la lista de accesosRecientes, la cual se obtiene con la operacion obtener del diccionario accesosXLink consultando por el link dado, esto cuesta $O(|l|)$.

Se ingresa a un ciclo, preguntando si no es vacia la lista, esto cuesta $O(1)$.

Se pregunta si dia del primer elemento de la lista es igual a f, esto cuesta $O(1)$, en caso verdadero se devuelve cantAccesos de esa tupla, esto cuesta $O(1)$, en caso falso se modifica la lista sacando el primer elemento, esto cuesta $O(1)$. Se sale del ciclo.

Orden Total: $O(1)+O(|l|)+O()=O(|l|)$

7. IINICIAR:

Se guarda en res.actual la fecha igual a 1, esto cuesta $O(1)$. Se pasa por referencia el arbol dado y se lo guarda en res.arbolCategorias, estoy cuesta $O(1)$. Se crea una variable del tipo nat, cuesta $O(1)$, se inicializa esta variable con 1, esto cuesta $O(1)$, se crea un arreglo con tamaño igual a #categorias(ac) y se lo guarda en res.arrayCatLinks, esto cuesta $O(1)$,

se inicializa res.listaLinks como vacia, esto cuesta $O(1)$, se inicializa con vacio el diccionario res.accesosXLink.

Se ingresa a un ciclo consultando si c es menor o igual a la cantidad de categorias de ac, esto cuesta $O(1)$. Se crea una lista linksFamilia inicializada con vacio, esto cuesta $O(1)$.

Se guarda en res.arrayCatLinks[c] la lista linksFamilia, esto cuesta $O(1)$, se le suma 1 a c, esto cuesta $O(1)$. Se sale del ciclo, esto cuesta $O(1)$.

Orden Total: $O(1)+O(1)+O(1)+O(1)+O(1)+O(1)+O(1)+O(1)+$

$O(\#categorias(ac)*(O(1)+O(1)+O(1)))=O(\#categorias(ac))$

8. INUEVOLINK:

Se crea un puntero a datosCat cat donde se le pasa el puntero obtenido por la operacion obtener del modulo arbolCategorias, esto cuesta $O(|c|)$. Se crea una lista de acceso inicializada vacia, que cuesta $O(1)$.

Se crea una tupla datosLink, a la cual se le pasa una tupla con el link dado, el puntero a datosCat y la lista de acceso, la cual tarda $O(|l|)$. Se crea un puntero a datosLink y se le pasa la tupla datosLink, esto cuesta $O(1)$.

Se utiliza la operacion definir del diccTrie en la cual se agrega el link dado al diccionario accesosXLink, lo cual tarda $O(|l|)$.

Se utiliza la operacion agregarAtras que agrega el puntero a datosLink a la lista listaLinks, esto demora $O(1)$.

Se ingresa a un ciclo si cat es distinto de la operacion puntRaiz de arbolCategorias, esto tarda $O(1)$. Se utiliza la operacion agregarAtras que agrega el puntero a datosLink a la lista que esta en la posicion (*cat).id del arreglo arrayCatLinks, lo cual tarda $O(1)$.

Se modifica el puntero a datosCat y se guarda cat.padre, lo cual tarda $O(1)$. Se sale del ciclo tardando $O(1)$. Se utiliza la operacion agregarAtras que agrega el puntero a datosLink a la lista que esta en la posicion (*cat).id del arreglo arrayCatLinks, lo cual tarda $O(1)$.

Orden Total: $O(|c|)+O(1)+O(|l|)+O(1)+O(1)+O(1)+O(h*(O(1)+O(1)))+O(1)=O(|l|+|c|+h)$

9. **IACCESO:**

Se pregunta si la fecha actual del sistema es igual a f, esto demora $O(1)$, en caso verdadero se deja actual como esta, en caso negativo se modifica a y se guarda f como fecha actual, esto tarda $O(1)$.

Se crea un puntero a datosLink puntLink que se le pasa un puntero obtenido por medio de la operacion obtener del diccionario accesosXLink dando el link que se quiere ingresar al sistema, esto demora $O(|l|)$.

Se pregunta si el dia de la tupla del ultimo elemento de la lista accesosRecientes de la tupla apuntada por el puntero puntLink es igual al f dado, esto cuesta $O(1)$, en caso positivo, se modifica cantAccesos de la misma tupla del elemento sumandole uno, esto demora $O(1)$

en caso negativo se utiliza la operacion agregarAtras y se agrega una tupla acceso con la fecha f y cantAccesos igual a 1 a la lista de accesosRecientes, lo cual demora $O(1)$.

Por ultimo, se consulta por la longitud de la lista accesosRecientes, consultando si la nueva longitud es igual a 4, esto demora $O(1)$, en caso positivo se modificara la lista sacando el primer elemento de la misma. Esto demora $O(1)$.

Orden Total: $O(1)+O(1)+O(1)+O(|l|)+O(1)+O(1)+O(1)+O(1)+O(1)=O(|l|)$

10. **IESRECIENTE:**

Devuelve un bool dependiendo si el f pasado es mayor o igual a la fecha obtenida por la operacion menorReciente(s,l) la cual tarda $O(|l|)$ y si es menor o igual a la fechaUltimoAcceso(s,l) la cual tambien tarda $O(1)$. Tanto menorReciente como fechaUltimoAcceso son operaciones del modulo LinkLinkIT y se les pasa el sistema y un link.

Orden Total: $O(|l|)+O(|l|)=O(|l|)$

11. **IACCESOSRECIENTES:**

Devuelve un nat, el cual proviene de la operacion sumarAccesosRecientes que se le pasa el sistema, el link y la interseccion que demora $O(1)$ de la operacion diasRecientesParaCategoria(s,c), que demora $O(|\star l|)$ con la operacion diasRecientes(s,l) que demora $O(|l|)$.

Aclaracion: $\star l$ es el link obtenido de la operacion linkConMasAccesos el cual pertenece al conjunto de links de la categoria c.

Orden Total: $O(|l|)+(O(1)*(O(|l|)+O(|l|)))=O(|l|)$

12. **ILINKSORDENADOSPORACCESOS:**

Orden Total: $O(|l|)+O(|l|)+O(|l|)+O(1)=O(|l|)$

13. **IBUSCARMAX:**

14. **IESTAORDENADA:**

Se inicializa res con true, esto demanda $O(1)$. Se crea un itLista de puntero a datosLink itRecorre al cual se lo inicializa con una lista ls que pasan como parametro, esto cuesta $O(1)$.

Se crea un nat aux el cual es inicializado con el valor de cantAccesosRecientes apuntado en la posicion actual del iterador. Esto cuesta $O(1)$. Se ingresa con la condicion de que haySiguiente del iterador sea verdadera y que res sea igual a true, esto cuesta $O(1)$.

Se avanza el iterador, lo que cuesta $O(1)$. Se pregunta si el valor de aux es menor a el valor de cantAccesosRecientes apuntado en la posicion actual del iterador, esto demanda $O(1)$, en caso afirmativo se modifica res por false, tardando $O(1)$.

Se modifica aux pasandole el valor de cantAccesosRecientes apuntado en la posicion actual del iterador. Luego de las iteraciones correspondientes, se sale del ciclo. **Orden Total:** $O(1)+O(1)+O(1)+[O(|ls|)*O(1)+O(1)+O(1)+O(1)]=O(|ls|)$

15. **ICANTLINKS:**

Se crea un puntero a datosCat cat al cual se le guarda el puntero obtenido por la operacion obtener del modulo arbolCategorias, lo cual tarda $O(|c|)$.

Se devuelve la longitud de la lista del arreglo arrayCantLinks[(\star cat).id], lo que demora $O(1)$

Orden Total: $O(|c|)+O(1)=O(|c|)$

16. **IMENORRECIENTE:**

Se devuelve la resta la cual demora $O(1)$, del maximo que tarda $O(1)$, de la operacion fechaUltimoAcceso(s,l) que demora $O(|l|) + 1$, con el valor constante diasRecientes.

Orden Total: $O(|l|)+O(1)+O(1)=O(|l|)$

17. **IDIASRECIENTES:**

Devuelve un conjunto de fechas dado por la operacion diasRecientesDesde que demora $O(|l|)$, a la cual se le pasa el sistema, un link y la operacion menorReciente que tambien demora $O(|l|)$.

Orden Total: $O(|l|)+O(|l|)=O(|l|)$

18. **IDIASRECIENTESDESDE:**

Se ingresa a un ciclo consultando por la operacion esReciente chequeando si la fecha es reciente, esta operacion tarda $O(|l|)$, dentro del ciclo, se utiliza la operacion Agregar que agrega por copia la fecha al conjunto, esto demora $O(1)$.

Se modifica f y se le suma uno, esto demora tambien $O(1)$. Se sale del ciclo.

Orden Total: $O(|l|) + O(1) + O(1) = O(|l|)$

19. **IDIASRECIENTESPARACATEGORIAS:**

Se crea un iterador itLista de puntero a datosLink links el cual se inicializa con arrayCatLinks[id(s.arbolCategorias,c)], o sea la lista listaLinks de la posicion id(s.arbolCategorias,c) del arreglo arrayCatLinks, esto demora $O(1)$.

Se devuelve un conjunto de fechas dado por la operacion diasRecientes que demora $O(\star|l|)$ a la cual se le pasan el sistema, y la operacion linkConMasAccesos que demora $O(\star|l|)$ a la cual se le pasan, el sistema, la categoria c y el itLista links.

Aclaracion: $\star l$ es el link obtenido de la operacion linkConMasAccesos el cual pertenece al conjunto de links de la categoria c.

Orden Total: $O(1) + O(\star|l|) + O(\star|l|) = O(\star|l|)$

20. **ISUMARACCESOSRECIENTES:**

Se crea un itConj iterador al cual se le pasa un conjunto de fechas, lo que demora $O(1)$. Se ingresa a un ciclo consultando por si haySiguiente del itConj, esto demora $O(1)$.

Se modifica res sumandole, la cual demora $O(1)$, al valor anterior que tenia, el valor de la operacion accesosRecientesDia que demora $O(|l|)$, pasandole el sistema, el link, y el valor de la posicion actual del iterador.

Luego se avanza el iterador que demora $O(1)$. Se sale del ciclo

Orden Total: $O(|l|) + O(1) + O(1) = O(|l|)$

21. **ILINKCONULTIMOACCESO:**

Se crea un puntero a datosLink nombrado max que se le pasa la posicion actual el itLista que se ingresa, esto demora $O(1)$, se ingresa a un ciclo consultando si hay siguiente del itLista, lo que demora $O(1)$.

Se avanza el iterador, lo que cuesta $O(1)$. Se pregunta si el dia de la tupla del ultimo elemento de la lista de acceso de accesosRecientes de la tupla apuntada en el puntero max es menor a el dia de la tupla del ultimo elemento de la lista de acceso de accesosRecientes de la tupla apuntada en la posicion actual del iterador, esto demora $O(1)$.

En caso afirmativo, se modifica max guardando la posicion actual del iterador, esto demora $O(1)$. Se sale del ciclo. Se devuelve el link de la tupla apuntada por max. Esto demora $O(|l|)$.

Orden Total: $O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(|l|) = O(|l|)$

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 $ac \rightarrow \text{conj}(\text{categoria})$

raíz : acat $ac \rightarrow \text{categoria}$

padre : acat $ac \times \text{categoria } h \rightarrow \text{categoria}$ $\{esta?(h, ac) \wedge raiz(ac) \neq h\}$

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

generadores

nuevo : categoria $c \rightarrow \text{acat}$ $\{\neg vacia?(c)\}$

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

otras operaciones

altura : acat $ac \rightarrow \text{nat}$

esta? : categoria $c \times \text{acat } ac \rightarrow \text{bool}$

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

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

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

axiomas $\forall a: \text{arbolDeCategorias}$
 $\forall c: \text{categoria}$
 $\forall ca: \text{conj}(\text{arbolDeCategorias})$
 $\forall cc: \text{conj}(\text{categoria})$

categorias(nuevo(c)) $\equiv c$

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

raiz(nuevo(c)) $\equiv c$

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

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

id(nuevo(c), c') $\equiv 1$

id(agregar(ac,c,h), h') $\equiv \text{if } h == h' \text{ then } \# \text{categorias}(ac) + 1 \text{ else } \text{id}(ac,h2) \text{ fi}$

altura(nuevo(c)) $\equiv \text{alturaCategoria}(\text{nuevo}(c), c)$

altura(agregar(ac,c,h)) $\equiv \max(\text{altura}(ac), \text{alturaCategoria}(\text{agregar}(ac,c,h), h))$

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

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

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

hijos(nuevo(c1), c2) $\equiv \emptyset$

hijos(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 ArbolDeCategorias

géneros: *acat*

2.0.2. Operaciones Básicas

categorias (in ac: acat) $\longrightarrow \text{res: conj}(\text{categoria})$

Pre $\equiv \text{true}$

Post $\equiv \text{res}=\text{obs}$ categorias(ac)
Complejidad : $O(\# \text{categorias}(\text{ac}))$
Descripción : Devuelve el conjunto de categorias de un ac
Aliasing:ALGO

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

Pre $\equiv \text{true}$
Post $\equiv \text{res}=\text{obs}$ 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 $\equiv h \in \text{ac} \wedge \text{raiz}(\text{ac}) \neq h$
Post $\equiv \text{res}=\text{obs}$ padre(ac,h)
Complejidad : $O(\text{ni idea})$
Descripción : Devuelve el padre de una categoria
Aliasing:ALGO

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

Pre $\equiv h \in \text{ac}$
Post $\equiv \text{res}=\text{obs}$ 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 $\equiv \text{res}=\text{obs}$ nuevo(c)
Complejidad : $O(|c|)$
Descripción : Crea un arbol
Aliasing:ALGO

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

Pre $\equiv c \in \text{ac} \wedge \neg \text{vacía?}(h) \wedge \text{ac}_0 =_{\text{obs}} \text{ac}$
Post $\equiv \text{ac}=\text{obs}$ 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 $\equiv \text{true}$
Post $\equiv \text{res}=\text{obs}$ altura(ac)
Complejidad : $O(|\text{ac}|)$
Descripción : Devuelve la altura del arbol ac
Aliasing:ALGO

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

Pre $\equiv \text{true}$
Post $\equiv \text{res}=\text{obs}$ esta?(c,ac)
Complejidad : $O(|\text{ac}|)$
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(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 esta?(c,ac)

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

Complejidad : O(no tengo idea)

Descripción : Devuelve la altura de la categoria c

Aliasing:ALGO

hijos (in ac: estrAC, in c: categoria) \longrightarrow res:conj(categoria)

Pre \equiv esta?(c,ac)

Post \equiv res=_{obs} 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: diccTrie(*padre*:string,puntero(datosCat)),
 categorias: Lista(datosCat)>
Donde datosCat es:
tupla <
 categoria:string,
 id:nat,
 altura:nat,
 hijos:conj(puntero(datosCat)),
 abuelo:puntero(datosCat)>

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. $(\forall x, y: \text{string}) (\text{def?}(x, e.familia)) \wedge (\text{def?}(y, e.familia)) \Rightarrow_L y \in (*(\text{obtener}(x, e.familia))).\text{hijos} \Leftrightarrow (*(*(\text{obtener}(y, e.familia))).\text{abuelo}).\text{categoria} = x$
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. $(\forall x: \text{string}) (\text{def?}(x, e.familia)) (\exists y: \text{string}) (\text{def?}(y, e.familia)) \Leftrightarrow (*(\text{obtener}(y, e.familia))).\text{id} \leq e.\text{cantidad} \wedge (*(\text{obtener}(x, e.familia))).\text{id} < e.\text{cantidad} \wedge_L (*(\text{obtener}(y, e.familia))).\text{id} = 1 + (*(\text{obtener}(x, e.familia))).\text{id}$

2.1.3. Función de Abstraccion

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

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

Auxiliares

$\text{todasLasCategorias} : \text{secu}(\text{datosCat}) \rightarrow \text{conj}(\text{categoria})$
 $\text{Ag}((\text{prim}(\text{cs})).\text{categoria}, \text{fin}(\text{cs})) \equiv$

2.1.4. Algoritmos

Algoritmo: 1

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

$\text{res} \leftarrow \text{vacío}()$

//O(1)

$\text{itLista iterador} \leftarrow \text{crearIt}(\text{ac}.\text{categorias})$

```

                                //O(1)

while(haySiguiente(iterador))
                                //O(longitud(ac.categorias))
    agregar(res, siguiente(iterador).categoria)
                                //O(|c|)
end while
                                //O(1)

```

Complejidad: sumatoria

Algoritmo: 2

IRAIZ (in ac: estrAC) \rightarrow res: categoria

```

    res  $\leftarrow$  (*ac.raiz).categoria
                                //O(|c|)

```

Complejidad: $O(|c|)$

Algoritmo: 3

IPADRE (in ac: estrAC, in h: categoria) \rightarrow res: puntero(categoria)

```

    res  $\leftarrow$  (*(*(obtener(h,ac.familia))).abuelo).categoria
                                //O(|h| + |res|)

```

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

Algoritmo: 4

IID (in ac: estrAC, in c: categoria) \rightarrow res:nat

```

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

```

Complejidad: $O(|c|)$

Algoritmo: 5

INUEVO (in c: categoria) \rightarrow res:estrAC

```

    res.cantidad  $\leftarrow$  1
                                //O(1)

```

```

    datosCat tuplaA
                                //O(|c|)

```

```

    puntero(datosCat) punt  $\leftarrow$  &tuplaA
                                //O(1)

```

```

    tuplaA  $\leftarrow$  tupla(c,1,1,vacio(), punt)
                                //O(|c|)

```

```

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

```

```

    res.alturaMax  $\leftarrow$  1
                                //O(1)

```

```

    res.familia  $\leftarrow$  definir(c, 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)

```
puntero(datosCat) 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)
```

```
datosCat tuplaA ← (h,ac.cantidad +1,(*puntPadre).altura +1,vacio(),puntPadre) //O(|h|)
```

```
puntero(datosCat) punt ← & 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) → res:nat

```
res ← ac.alturaMax //O(1)
```

Complejidad: $O(1)$

Algoritmo: 8

UESTA? (in c: categoria,in ac: estrAC) → res:bool

```
res ← def?(c,ac.familia) //O(|c|)
```

Complejidad: $O(|c|)$

Algoritmo: 9

IESSUBCATEGORIA (in ac: estrAC, in c: categoria,in h: categoria) → res:bool

```

    puntero(datosCat) puntPadre ← (obtener(c,ac.familia)) //O(|c|)
    res ← false //O(1)
    puntero(datosCat) actual //O(1)
    if c == ac.raiz //O(|c|)
    then res ← true //O(1)
    ELSE actual ← (obtener(h,ac.familia)) //O(|h|)
    while(res ≠ true ∧ actual ≠ ac.raiz) //O(★(h))
    if PERTENECE?((*puntPadre).hijos,actual) //O(1)
    then res ← true //O(1)
    ELSE actual ← (*(obtener(actual,ac.familia)) ).abuelo FI FI //O(1)

```

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

Algoritmo: 10

IAALTURACATEGORIA (**in** ac: estrAC, **in** c: categoria) \rightarrow res:nat
 res ← (*(obtener(c,ac.familia))).altura //O(|c|)

Complejidad: $O(|c|)$

Algoritmo: 11

IHIJOS (**in** ac: estrAC, **in** c: categoria) \rightarrow res:itConj(punero(datosCat))
 res ← crearIt((*obtener(c,ac.familia)).hijos) //O(|c|)

Complejidad: $O(|c|)$

Algoritmo 12

IOBTENER (**in** c: categoria, **in** ac: estrAC) \rightarrow res:punero(datosCat)
 res ← obtener(c,ac.familia) //

Complejidad: $O(|c|)$

Algoritmo: 13

IPUNTRAIZ (**in** ac: estrAC) \rightarrow res:punero(datosCat)
 res ← 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 $\text{tupla}(\text{arreglo}(\text{Puntero}(\text{Nodo}))[27], \text{significado}(\text{Puntero}(\alpha)))$

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

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

- 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