

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

categorias(iniciar(ac)) \equiv ac

categorias(nuevoLink(s,l,c)) \equiv categorias(ac)

categorias(acceso(s,l,f)) \equiv categorias(ac)

links(iniciar(ac)) $\equiv \emptyset$

links(nuevoLink(s,l,c)) \equiv Ag(l,links(s))

links(acceso(s,l,f)) \equiv links(s)

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

categoriaLink(acceso(s,l,f),l') \equiv categoriaLink(s,l')

fechaActual(iniciar(ac)) $\equiv 0$

fechaActual(nuevoLink(s,l,c)) \equiv fechaActual(s)

fechaActual(acceso(s,l,f)) $\equiv f$

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

fechaUltimoAcceso(acceso(s,l,f),l') \equiv fechaUltimoAcceso(s,l')

menorReciente(s,l) \equiv max(fechaUltimoAcceso(s, l) + 1, diasRecientes) - diasRecientes

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

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

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

accesosRecientes(s, c, l) \equiv sumarAccesosRecientes(s, l, diasRecientesParaCategoria(s, c) \cap diasRecientes(s, l))

linksOrdenadosPorAccesos(s, c) \equiv linksOrdenadosPorAccesosAux(s, c, linksCategoriaOHijos(s, c))

linksOrdenadosPorAccesosAux(s,c,ls) \equiv **if** $\emptyset?(ls)$ **then**

\emptyset

else

linkConMasAccesos(s, c, ls) \bullet linksOrdenadosPorAccesosAux(s, c, ls - linkConMasAccesos(s, c, ls))

fi

linkConMasAccesos(s, c, ls) \equiv **if** $\#ls == 1$ **then**

dameUno(ls)

else

if accesosRecientes(s,c,dameUno(ls)) $>$ accesosRecientes(s,c,linkConMasAccesos(s,c,sinUno(ls))) **then**
dameUno(ls)

else

linkConMasAccesos(s,c,sinUno(ls))

fi

fi

cantLinks(s, c) \equiv $\#$ linksCategoriaOHijos(s, c)

diasRecientes(s, l) \equiv diasRecientesDesde(s, l, menorReciente(s, l))

diasRecientesDesde(s, l, f) \equiv **if** esReciente?(s, l, f) **then** Ag(f, diasRecientesDesde(s, l, f+1)) **else** \emptyset **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 categoria 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 <*link*:link, *catDLink*puntero(datosCat),*accesosRecientes*:Lista(acceso),*cantAccesosRecientes*:nat >

Donde acceso es:

tupla <*dia*:nat, *cantAccesos*:nat >

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*', estan ordenados,no hay dias repetidos y la longitud de la lista es menor o igual a 3.
3. Para todo '*link*' que exista en '*accesosXLink*' su significado deberÃ¡ existir en '*listaLinks*' y viceversa.
4. Para todo '*link*' que exista en '*accesosXLink*' su significado deberÃ¡ aparecer en '*arrayCantLinks*' en la posicion igual al id de '*catDLink*' y en las posiciones de los predecesores de esa categoria y en ninguna otra.
5. No hay 2 claves que existan en '*accesosXLink*' y devuelvan el mismo significado.
6. No existen '*link*' repetidos en las tuplas de '*listaLinks*'.
7. No hay elementos repetidos en ninguna lista '*linksFamilia*'.

8. Para todo '*link*' que exista en '*accesosXLink*', '*cantAccesosRecientes*' es igual a la suma de '*cantAccesos*' de cada elemento de la lista '*accesosRecientes*'

Rep : $\text{estrLLI} \rightarrow \text{bool}$

$\text{Rep}(e) \equiv \text{true} \iff$

1. $(\forall x: \text{link}) (\text{def?}(x, e.\text{accesosXLink})) \rightarrow_L$
 $(\text{*obtener}(x, e.\text{accesosXLink})).\text{catDLink} \in \text{todasLasCategorias}(e.\text{arbolCategorias}.\text{categorias})$
2. $(\forall x: \text{link}) (\text{def?}(x, e.\text{accesosXLink})) \rightarrow_L$
 $\text{long}(\text{*obtener}(l, e.\text{accesosXLink})).\text{accesosRecientes} \leq 3 \wedge$
 $\text{accesoOrdenadoNoRepetido}(\text{*obtener}(l, e.\text{accesosXLink})).\text{accesosRecientes} \wedge_L$
 $\text{fechasCorrectas}(e.\text{actual}, (\text{*obtener}(l, e.\text{accesosXLink})).\text{accesosRecientes})$
3. $(\forall x: \text{link}) (\text{def?}(x, e.\text{accesosXLink})) \leftrightarrow (\text{*obtener}(x, e.\text{accesosXLink})) \in \text{todosLosLinks}(\text{listaLinks})$
4. $(\forall x: \text{link}) (\text{def?}(x, e.\text{accesosXLink})) \rightarrow_L$
 $(\forall c: \text{categoria}) c \in \text{todasLasCategorias}(e.\text{arbolCategorias}.\text{categorias}) \rightarrow_L$
 $(\text{esta?}(\text{obtener}(l, e.\text{accesosXLink}), \text{arrayCatLinks}[\text{id}(c, e.\text{arbolCategorias})]) \leftrightarrow \text{esPredecesor}(c, (\text{*obtener}(l, e.\text{accesosXLink})))$
5. $(\forall x, x': \text{link } l \neq l') \wedge (\text{def?}(x, e.\text{accesosXLink})) \wedge (\text{def?}(x', e.\text{accesosXLink})) \rightarrow_L (\text{*obtener}(x, e.\text{accesosXLink})) \neq$
 $(\text{*obtener}(x', e.\text{accesosXLink}))$
6. $(\forall i, i': \text{nat}) i < \text{long}(e.\text{listaLinks}) \wedge i' < \text{long}(e.\text{listaLinks}) \rightarrow_L e.\text{listaLinks}_i.\text{link} = e.\text{listaLinks}_{i'}.\text{link} \text{ lefttrightarrow } i = i'$
7. $(\forall i: \text{nat}) i < \text{tam}(\text{arrayCatLinks}) \rightarrow_L \text{sinRepetidos}(\text{linksFamilia}_i)$
8. $(\forall x: \text{link}) (\text{def?}(x, e.\text{accesosXLink})) \rightarrow_L (\text{*obtener}(x, e.\text{accesosXLink})).\text{cantAccesosRecientes} == \text{cantidadDeAc-}$
 $\text{cesos}(\text{*obtener}(x, e.\text{accesosXLink})).\text{accesosRecientes})$

1.1.3. Función de Abstraccion

Abs: $\text{estrLLI } e \rightarrow \text{linkLinkIT}$

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

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

Auxiliares

$\text{cantidadPorDia} : \text{fecha} \times \text{lista}(\text{acceso}) \rightarrow \text{nat}$

$\text{cantidadPorDia}(f, ls) \equiv \text{if } f == (\text{prim}(ls)).\text{dia} \text{ then } \text{cantAccesos} \text{ else } \text{cantidadPorDia}(f, \text{fin}(ls)) \text{ fi}$

$\text{listaLinks} : \text{secu}(\text{datosLink}) \rightarrow \text{conj}(\text{link})$

$\text{listaLinks}(ls) \equiv \text{Ag}((\text{prim}(ls)).\text{link}, \text{fin}(ls))$

$\text{sinRepetidos} : \text{secu}(\alpha) \rightarrow \text{bool}$

$\text{sinRepetidos}(ls) \equiv \text{if } \text{vacía?}(ls) \text{ then } \text{true}$
 else
 $\text{if } \text{hayOtro}(\text{prim}(ls), \text{fin}(ls)) \text{ then } \text{false} \text{ else } \text{sinRepetidos}(\text{fin}(ls)) \text{ fi}$

$\text{hayOtro} : \alpha \times \text{secu}(\alpha) \rightarrow \text{bool}$

$\text{hayOtro}(x, ls) \equiv \text{if } \text{vacía?}(ls) \text{ then } \text{false} \text{ else if } x == \text{prim}(ls) \text{ then } \text{true} \text{ else } \text{hayOtro}(x, \text{fin}(ls)) \text{ fi fi}$

$\text{fechasCorrectas} : \text{nat} \times \text{secu}(\text{acceso}) \rightarrow \text{bool}$


```

fechasCorrectas(x,ls)  $\equiv$  if vacia?(ls) then
    true
    else
        if prim(ls).dia > f then false else fechasCorrectas(x,fin(ls)) fi
    fi
accesoOrdenadoNoRepetido : secu(acceso)  $\rightarrow$  bool
accesoOrdenadoNoRepetido(ls)  $\equiv$  if long(ls)  $\leq$  1 then
    true
    else
        if prim(ls).dia  $\geq$  prim(fin(ls)).dia then
            false
        else
            accesoOrdenadoNoRepetido(fin(ls))
        fi
    fi
cantidadDeAccesos : secu(acceso)  $\rightarrow$  nat
cantidad(ls)  $\equiv$  if vacia?(ls) then 0 else (prim(ls)).cantAccesos + fin(ls) fi

```

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

IFECHA ACTUAL (**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

IINICIAR (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 ≤ #categorias(ac)) //O(#categorias(ac))

linksFamilia llist ← vacia() //O(1)

res.arrayCatLinks[c] ← llist //O(1)

c ++ //O(1)

end while

```

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,0> //O(|l|)

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(h)

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

cat ← cat.abuelo //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

(*puntLink).cantAccesosRecientes++ //O(1)

```

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(|l|)

```

Complejidad: $O(|l|)$

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(|l|)

```

Complejidad: $O(|l|)$

Algoritmo: 12

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

```

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

```

```

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(haySiguiete?(s.arrayCantLinks[id])) //O(n)

```

```

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

```

```

agregarAtras(listaOrdenada,siguiente(itMax)) //O(1)

```

```

eliminarSiguiete(itMax) //O(1)

```

```

end while

```

```

res  $\leftarrow$  crearIt(listaOrdenada) //O(1)

```

```

s.arrayCatLinks[id]  $\leftarrow$  listaOrdenada //O(1)

```

```

else

```

```

res ← crearIt(s.arrayCantLinks[id]) //O(1)

fi

```

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(n)

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

```

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(n)

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

Complejidad: $O(n)$

Algoritmo: 15

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

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

res \leftarrow longitud(arrayCantLinks[(cat).id]) //O(1)

Complejidad: $O(|c|)$

Algoritmo: 16

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

res \leftarrow 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)

res \leftarrow 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)

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)

```

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

diasRecientes(s,linkConUltimoAcceso(s,c,links)) //O(★|l|)

```

Complejidad: $O(★|l|)$

Algoritmo: 20

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

```

itConj iterador ← crearIt(fs) //O(1)

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

res ← 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)) → res: link

```

while(haySiguiente(ls)) //O(|ls|)

if s.actual == (ultimo((*siguiente(ls)).accesosRecientes)).dia //O(1)

then res ← (siguiente(ls)) //O(1)

fi //O(1)

avanzar(ls) //O(1)

end while

```

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 $O(1)$.

Luego de recorrer toda la lista se sale del ciclo habiendo demorado finalmente $O(|lista|)$, 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 $O(1)$. Una vez recorrida toda la lista se sale del ciclo demorando $O(|lista|)$

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 $O(1)$. Una vez que no se cumple la condicion del ciclo se sale del mismo habiendo demorado finalmente $O((\#categorias(ac)))$.

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 $O(1)$. Una vez que no se cumple la condicion del ciclo se del mismo habiendo tardado $O(h)$. 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 $O(1)$.

Aclaracion h es igual a la altura de la categoria c. **Orden Total:** $O(|c|)+O(1)+O(|l|)+O(1)+O(1)+O(1)+O(h*(O(1)+O(1)))$

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:**

Se crea un nat id al cual se le pasa el id de la categoria que ingresan por medio de la operacion del modulo de arbolCategorias, lo cual demora $O(|c|)$. Se crea una lista de puntero a datosLink llamada listaOrdenada la cual se la inicializa en vacio, esto cuesta $O(1)$.

Se crea un itLista de puntero a datosLink nombrado itMax al cual se le pasa por referencia la lista del arreglo arrayCatLinks en la posicion del id de la categoria, esto demora $O(1)$. Se pregunta por si la lista del arreglo arrayCatLinks en la posicion del id de la categoria no esta ordenada, esto cuesta $O(n)$. En caso verdadero se ingresa a un ciclo, con la condicion de que haySiguiente? arrayCatLinks[id] sea verdadero, esto demora $O(1)$. Se le pasa a itMax un iterador de la operacion BuscarMax a la cual se le pasa arrayCatLinks[id], esto demora $O(n)$. Luego se utiliza la operacion agregarAtras demorando $O(1)$, la cual agrega la posicion actual del iterador itMax en la lista listaOrdenada.

Se usa la operacion eliminarSiguiente con la cual se elimina la posicion actual del iterador itMax, demorando $O(1)$. Una vez recorrido todo el itLista de puntero a datosLink y que haySiguiente? sea false se sale del ciclo tardando $O(n^2)$. Se le pasa a res un iterador unidireccional con la listaOrdenada en $O(1)$.

Se modifica arrayCatLinks[id] pasandole la listaOrdenada, esto demora $O(1)$. Luego en la parte falsa del IF en el caso de que si este ordenada la arrayCatLinks[id] se le pasa a res un iterador unidireccional con arrayCatLinks[id] demorando $O(1)$.

Aclaracion: n es igual a la cantidad de elementos de la lista. **Orden Total:** $O(|c|)+O(1)+O(1)+O(n)+[n*(O(n)+O(1)+O(1))]$

13. **IBUSCARMAX:**

Se inicializa res pasandole un itLista de puntero a datosLink demorando $O(1)$. Se crea un itLista de puntero a datosLink llamad itRecorre pasandole la lista que nos ingresa, demorando $O(1)$.

Se crea un nat llamado max el cual es inicializado pasandole el valor de cantAccesosRecientes de la tupla apuntada por la posicion actual del itRecorre, esto tarda $O(1)$. Se ingresa a un ciclo con la condicion de que haySiguiente de itRecorre sea true, esto demora $O(1)$. Se pregunta si max es menor al valor cantAccesosrecientes de la tupla apuntada por la posicion actual del itRecorre, esto demora $O(1)$. En caso verdadero se guarda en max el valor cantAccesosrecientes de la tupla apuntada por la posicion actual del itRecorre, esto demora $O(1)$.

Se guarda en res el iterador itRecorre demorando $O(1)$. No hay parte falsa. Se ultiza la operacion avanzar a la cual se le pasa itRecorre demorando $O(1)$. Una vez recorrido toda la lista, y haySiguiente?(itRecorre) sea falso, se sale del ciclo habiendo demorado $O(n)$. Aclaracion: n es igual a la cantidad de elementos de la lista. **Orden Total:** $O(1)+O(1)+O(1)+[n*(O(1)+O(1)+O(1)+O(1))]=O(n)$

14. **IENTAORDENADA:**

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 habiendo demorado en el peor de los casos $O(n)$. Aclaracion: n es igual a la cantidad de elementos de la lista. **Orden Total:** $O(1)+O(1)+O(1)+[n*(O(1)+O(1)+O(1)+O(1))]=O(n)$

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 $O(*|l|)$ a la cual se le pasan el sistema, y la operacion linkConMasAccesos que demora $O(*|l|)$ a la cual se le pasan, el sistema, la categoria c y el itLista links.

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

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

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 $O(1)$. Una vez que la condicion del ciclo es falsa, se sale del ciclo habiendo demorado $O(|fs|)$.

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

21. **ILINKCONULTIMOACCESO:**

Se ingresa a un ciclo consultando si hay siguiente del itLista, lo que demora $O(1)$.

Se pregunta si el dia de la tupla del ultimo elemento de la lista de acceso de accesosRecientes de la tupla apuntada en la posicion del it es igual a la del sistema, esto demora $O(1)$.

En caso afirmativo, se modifica res guardando la posicion actual del iterador, esto demora $O(1)$. Se avanza el iterador, lo que cuesta $O(1)$. Una vez recorrida toda la lista se sale del ciclo habiendo demorado $O(|ls|)$. Se devuelve el link de la tupla apuntada por max. Esto demora $O(|l|)$.

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

2. TAD ARBOLDECATEGORIAS

TAD ARBOLDECATEGORIAS

géneros acat

exporta generadores, categorias, raÃz, padre, id, altura, estÃ?, esSubCategoria, alturaCategoria, hijos

usa BOOL, NAT, CONJUNTO

observadores básicos

categorias : acat ac \longrightarrow conj(categoria)

raiz : acat ac \longrightarrow categoria

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

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 $\forall a$: arbolDeCategorias
 $\forall c$: categoria
 $\forall ca$: conj(arbolDeCategoria)
 $\forall cc$: conj(categoria)

categorias(nuevo(c)) \equiv c
 categorias(agregar(ac,c,h)) \equiv Ag(h, categorias(ac))

raiz(nuevo(c)) \equiv c
 raiz(agregar(ac,c,h)) \equiv raiz(ac)

padre(agregar(ac,c,h),h') \equiv **if** h == h' **then** c **else** padre(ac,c,h') **fi**

id(nuevo(c), c') \equiv 1
 id(agregar(ac,c,h), h') \equiv **if** h==h' **then** #categorias(ac) + 1 **else** id(ac,h2) **fi**

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

esta?(c,ac) \equiv c \exists categorias(ac)

esSubCategoria(ac,c,h) \equiv c == h \vee L (h = raiz(ac) \wedge L esSubCategoria(ac, c, padre(ac, h)))

hijos(nuevo(c1), c2) \equiv \emptyset
 hijos(agregar(ac,c,h), c') \equiv **if** h == c' **then** \emptyset **else** (**if** c==c' **then** h **else** \emptyset **fi**) \cup hijos(ac,c,c') **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 res: conj(categoria)

Pre \equiv true

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

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

Descripción : Devuelve el conjunto de categorias 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 raíz del arbol ac

Aliasing:ALGO

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

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

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

Complejidad : $O(ni\ idea)$

Descripción : Devuelve el padre de una categoria

Aliasing:ALGO

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

Pre $\equiv h \in ac$

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 $\equiv \neg vacia?(c)$

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

Post $\equiv ac =_{obs} agregar(ac_0, c, h)$

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

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. $e.cantidad = \text{longitud}(e.categorias)$
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. $(\forall x: \text{string}) (\text{def?}(x, e.familia)) \Rightarrow_L (*\text{obtener}(x, e.familia)).\text{altura} \leq e.alturaMax$
7. $(\exists x: \text{string}) (\text{def?}(x, e.familia)) \wedge_L (*(\text{obtener}(x, e.familia))).\text{altura} = e.alturaMax$
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. $(\forall x: \text{string}) (\text{def?}(x, e.familia)) \Rightarrow_L (*(\text{obtener}(x, e.familia))).\text{id} \leq e.cant$
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. $(\forall x: \text{string}) (\text{def?}(x, e.familia)) (\exists y: \text{string}) (\text{def?}(y, e.familia)) \Leftrightarrow (*(\text{obtener}(y, e.familia))).\text{id} \leq e.cantidad \wedge (*(\text{obtener}(x, e.familia))).\text{id} < e.cantidad \wedge_L (*(\text{obtener}(y, e.familia))).\text{id} = 1 + (*(\text{obtener}(x, e.familia))).\text{id}$

2.1.3. Función de Abstraccion

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

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

Auxiliares

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

2.1.4. Algoritmos

Algoritmo: 1

ICATEGORIAS (in ac: estrAC) \rightarrow res: conj(categoria)

```
res  $\leftarrow$  vacio() //O(1)
itLista iterador  $\leftarrow$  crearIt(ac.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 ← tupla(c,1,1,vacio(), punt) //O(|c|)

res.raiz ← punt //O(1)

res.alturaMax ← 1 //O(1)

res.familia ← 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) \rightarrow res:bool

```
puntero(datosCat) puntPadre  $\leftarrow$  (obtener(c,ac.familia)) //O(|c|)
res  $\leftarrow$  false //O(1)
puntero(datosCat) actual //O(1)
if c == ac.raiz //O(|c|)
then res  $\leftarrow$  true //O(1)
ELSE actual  $\leftarrow$  (obtener(h,ac.familia)) //O(|h|)
while(res  $\neq$  true  $\wedge$  actual  $\neq$  ac.raiz) //O( $\star$ (h))
if PERTENECE?((*puntPadre).hijos,actual) //O(1)
then res  $\leftarrow$  true //O(1)
ELSE actual  $\leftarrow$  (*(obtener(actual,ac.familia)) ).abuelo FI FI //O(1)
```

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

Algoritmo: 10

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

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

Complejidad: $O(|c|)$

Algoritmo: 11

IIHIJOS (in ac: estrAC, in c: categoria) \rightarrow res:itConj(puntero(datosCat))

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

Complejidad: $O(|c|)$

Algoritmo 12

IOBTENER (in c: categoria, in ac: estrAC) \rightarrow res:puntero(datosCat)

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

Complejidad: $O(|c|)$

Algoritmo: 13

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

res \leftarrow ac.raiz

//O(1)

Complejidad: $O(1)$

2.2. Descripción de Complejidades de Algoritmos

- 1.
- 2.
- 3.
- 4.
- 5.
- 6.
- 7.
- 8.
- 9.
- 10.
- 11.
- 12.
- 13.
- 14.

DiccTrie(α) se representa con estrDT, donde estrDT es Puntero(Nodo)

Nodo es tupla arreglo arreglo(Puntero(Nodo))[27], significadoPuntero(α)

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

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

- 1.
- 2.
- 3.

Funciones auxiliares

$\text{EncAEstrDTEnNMov} : \text{estrDT} \times \text{estrDT} \times \text{Nat} \longrightarrow \text{Bool}$

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

$\text{EstaEnElArregloActual?} : \text{estrDT} \times \text{estrDT} \times \text{nat} \longrightarrow \text{Bool}$

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

$\text{RecurrenciaConLosHijos} : \text{estrDT} \times \text{estrDT} \times \text{nat} \times \text{nat} \longrightarrow \text{Bool}$

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

$\text{SonTodosNullOLosHijosLoSon} : \text{estrDT} \longrightarrow \text{Bool}$

$\text{SonTodosNullOLosHijosLoSon}(e) \equiv \text{Los27SonNull}(e, 26) \vee \text{BuscarHijosNull}(e, 26)$

$\text{Los27SonNull} : \text{estrDT} \times \text{nat} \longrightarrow \text{Bool}$

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

$\text{BuscarHijosNull} : \text{estrDT} \times \text{nat} \longrightarrow \text{Bool}$

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

2.2.2. Función de Abstracción

Abs: $\text{estr } e \rightarrow \text{diccT}(c, \alpha)$

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

Funciones auxiliares

$\text{estaDefinido?} : \text{string} \times \text{estrDT} \rightarrow \text{bool}$

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

$\text{NodoDef?} : \text{string} \times \text{Nodo} \rightarrow \text{bool}$

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

$\text{numero} : \text{char} \rightarrow \text{nat}$

$\text{numero}(\text{char}) \equiv \text{char} - a$

$\text{ObtenerS} : \text{string} \times \text{Nodo} \rightarrow \alpha$

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

3. Renombres

TAD CATEGORIA

es String

Fin TAD

TAD LINK

es String

Fin TAD

TAD FECHA

es Nat

Fin TAD