

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

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

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

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

Pre \equiv true

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

Complejidad : O(1)

Descripción : Devuelve la fecha actual

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$\text{Pre} \equiv c \in \text{categorias}(s)$
 $\text{Post} \equiv \text{res} =_{\text{obs}} \text{linksCategoriasOHijos}(s, c)$
 Complejidad : $O(1)$
 Descripción : Devuelve el conjunto de links de la categoria c y sus hijos

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

$\text{Pre} \equiv c \in \text{categorias}(s) \wedge ls \subseteq \text{links}(s)$
 $\text{Post} \equiv \text{res} =_{\text{obs}} \text{filtrarLinsCategoriasOHijos}(s, c, ls)$
 Complejidad : $O(\text{no tengo idea})$

Descripción : Devuelve el conjunto de links de la categoria c y sus hijos

diasRecientesParaCategorias (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(es la cantidad de accesos recientes esto??)

Descripción : Devuelve el conjunto de fechas recientes de la categoria c

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 accedio por ultima vez del conjunto ls

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

linksOrdenadosPorAccesosAux (in s: lli, in c: categoria, in ls: conj(link)) \longrightarrow res: secu(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 la secuencia de links ordenados por accesos de mas recientes a menos recientes

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

β (in b: bool) \longrightarrow res: nat

Pre $\equiv \text{true}$

Post $\equiv \text{res} =_{\text{obs}} \beta(b)$

Complejidad : O(1)

Descripción : Devuelve 1 o 0 dependiendo el valor de verdad de b

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 $\langle \text{link}:\text{link}, \text{catDLink}\text{puntero}(\text{datosCat}), \text{accesosRecientes}:\text{Lista}(\text{acceso}) \rangle$

Donde acceso es:

tupla $\langle \text{dia}:\text{nat}, \text{cantAccesos}:\text{nat} \rangle$

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 \longrightarrow 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. $(\forall x: \text{link}) (\text{def?}(x, e.\text{accesosXLink})) \rightarrow (*\text{obtener}(x, e.\text{accesosXLink})) \exists \text{ linksDeCat}(e.\text{arrayCantLinks}[\text{id}(e.\text{arbolCategorias}, (*\text{obtener}(x, e.\text{accesosXLink})).\text{catDLink}]))$
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 |

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

Auxiliares

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

1.1.4. Algoritmos

Algoritmo: 1

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

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

Complejidad: O(1)

Algoritmo: 2

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

res \leftarrow claves(s.accesosXLink) // O($\sum_{i=1}^{\text{longitud}(s.\text{listaLinks})}$)

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

Algoritmo: 3

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

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

Complejidad: O(|l|)

Algoritmo: 4

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

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

Complejidad: O(1)

Algoritmo: 5

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

res \leftarrow prim(*((obtener(l,s.accesosXLink))).accesosRecientes).dia // O(|l|)

Complejidad: O(|l|)

Algoritmo: 6

IACCESOSRECIENTESDIA (in s: estrLLI, in l: link, in f: fecha) \longrightarrow 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 esVacía?(accesos) \wedge (prim(accesos)).dia \neq f)

```

    if (prim(accesos)).dia == f //O(1)
then res = (prim(accesos)).cantAccesos //O(1)
ELSE accesos = fin(accesos) FI //O(1) O(ALGO)
end while
Complejidad:

```

Algoritmo: 7

```

IINICIAR (in ac: acat) → res: estrLLI
    res.actual ← 1 // O(1)
    res.arbolCategorias ← ac // O(1)
    var c: nat //O(1)
    c ← 1 //O(1)
    res.arrayCantLinks ← crearArreglo(#categorias(ac)) // O(1)
    res.listaLinks ← vacia() //O(1)
    res.accesosXLink ← vacio() //O(1)
    while (c ≤ #categorias(ac))
    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> //O(1)
    puntero(datosLink) puntLink ← nuevoLink //O(1)
    definir(l,puntLink,s.accesosXLink) //O(|l|)
    agregarAtras(s.listaLinks,puntLink) //O(1)
    while(cat ≠ puntRaiz(s.arbolCategorias))
    agregarAtras(s.arrayCatLinks[(cat).id],puntLink) //O(1)
    cat ← cat.padre //O(1)
    end while
    agregarAtras(s.arrayCatLinks[(cat).id],puntLink) //O(1)
Complejidad: O(|c|+|l|+h)

```

Algoritmo: 9

```

IACCESO (in/out s: lli, in l: link , in f: fecha)
    if s.actual == f then s.actual ← s.actual 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 then fin(*puntLink.accesos) //O(1) fi
Complejidad: O(|l|)

```

Algoritmo: 10

```

IESRECIENTE? (in s: lli, in l: link , in f: fecha) → res: bool
    res ← menorReciente(s,l) ≤ f ∧ f ≤ fechaUltimoAcceso(s,l) O(ALGO)
Complejidad: O(ALGO)

```

Algoritmo: 11**IACCESOSRECIENTES** (in s: lli, in c: categoria in l: link) \rightarrow res: natres \leftarrow sumarAccesosRecientes(s, l, diasRecientesParaCategoria(s, c) \cap diasRecientes(s, l)) //O(ALGO)**Complejidad: O(ALGO)**

Algoritmo: 12**ILINKSORDENADOSPORACCESOS** (in s: lli, in c: categoria) \rightarrow res: lista(link)nat id \leftarrow id(s.arbolCategorias,c) //O(1)lista(puntero(datosLink)) listaOrdenada \leftarrow vacia() //O(1)itLista(puntero(datosLink)) itMax \leftarrow crearIt(s.arrayCantLinks[id]) //O(1)**if** \neg listaOrdenada?(s.arrayCantLinks[id])**then****while**(haySiguiente?(s.arrayCantLinks[id]))itMax \leftarrow iBuscarMax(s.arrayCantLinks[id]) //O(n)

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

eliminarSiguiente(itMax) //O(1)

fi**end while****Complejidad: O(n²)**

Algoritmo: 13**IBUSCARMAX** (in ls: lista(puntero(datosLink))) \rightarrow res: itLista(puntero(datosLink))res \leftarrow crearIt(ls) //O(1)itLista(puntero(datosLink)) itRecorre \leftarrow crearIt(ls) //O(1)nat max \leftarrow (*siguiente(itRecorre)).cantAccesosRecientes //O(1)**while**(haySiguiente(itRecorre))**if** max < (*siguiente(itRecorre)).cantAccesosRecientes**then**max \leftarrow (*siguiente(itRecorre)).cantAccesosRecientes //O(1)res \leftarrow itRecorre //O(1)**fi**

avanzar(itRecorre)

end while**Complejidad: O(n)**

Algoritmo: 14**IESTAORDENADA** (in ls: lista(puntero(datosLink))) \rightarrow res: boolres \leftarrow true //O(1)itLista(puntero(datosLink)) itRecorre \leftarrow crearIt(ls) //O(1)nat aux \leftarrow (*siguiente(itRecorre)).cantAccesosRecientes //O(1)**while**(haySiguiente(itRecorre) \wedge res == true) avanzar(itRecorre)**if** aux < (*siguiente(itRecorre)).cantAccesosRecientes**then**res \leftarrow false //O(1)**fi**aux \leftarrow (*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 ← 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) → res: fecha
res ← max(fechaUltimoAcceso(s,l)+1,diasRecientes) - diasRecientes //O(ALGO)
Complejidad: O(ALGO)

Algoritmo: 17

IDIASRECIENTES (in s: lli, in l: link) → res: fecha
res ← diasRecientesDesde(s,l,menorReciente(s,l))
Complejidad: O(ALGO)

Algoritmo: 18

IDIASRECIENTESDESDE (in s: lli, in l: link) → res: fecha
conj ← ∅ //O(1)
fecha ← f //O(1)
while(esReciente?(s,l,fecha))
 Ag(fecha,conj)
 fecha ← fecha + 1 //O(1)
end while
res ← conj
Complejidad: O(ALGO)

Algoritmo: 19

IDIASRECIENTESPARACATEGORIAS (in s: lli, in c: categoria) → res: conj(fecha)
res ← IF ∅?(linksCategoriaOHijos(s,c)) **then**
 ∅ **ELSE** diasRecientes(s,linkConUltimoAcceso(s,c,linksCategoriaOHijos(s,c)))
FI //O(ALGO)
Complejidad: O(ALGO)

Algoritmo: 20

ISUMARACCESOSRECIENTES (in s: lli, in l: link, in fs: conj(fecha)) → res: nat
conjFecha ← fs
while(!∅?(conjFecha))
 res ← accesosRecientesDia(s,l,dameUno(conjFecha))
 conjFecha ← sinUno(conjFecha)
end while
Complejidad: O(ALGO)

2. TAD ARBOLDeCATEGORIAS

TAD ARBOLDeCATEGORIAS

géneros *acat*

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

usa *BOOL, NAT, CONJUNTO*

observadores básicos

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

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) $\equiv c \in categorias(ac)$

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

$\text{hijos}(\text{nuevo}(\text{c1}), \text{c2}) \equiv \emptyset$

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

Fin TAD

2.0.5. Modulo de Arbol de Categorías

generos: *acat*

usa: bool, nat, conjunto

se explica con: TAD ArbolDeCategorías

géneros: *acat*

2.0.6. Operaciones Básicas

categorias (**in** *ac*: *acat*) \longrightarrow *res*: conj(*categoria*)

Pre \equiv true

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

Complejidad : $O(\#\text{categorias}(\text{ac}))$

Descripción : Devuelve el conjunto de categorias de un *ac*

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

Pre \equiv true

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

Complejidad : $O(1)$

Descripción : Devuelve la raiz del arbol *ac*

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

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

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

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

Descripción : Devuelve el padre de una categoria

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

Pre $\equiv \text{h} \in \text{ac}$

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

Complejidad : $O(|c|)$

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

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

Pre $\equiv \neg \text{vacía?}(c)$

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

Complejidad : $O(|c|)$

Descripción : Crea un arbol

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

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

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

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

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

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

Complejidad : $O(\text{no tengo idea})$

Descripción : Devuelve si c es descendiente de h

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

Pre $\equiv esta?(c, ac)$

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

Complejidad : $O(\text{no tengo idea})$

Descripción : Devuelve la altura de la categoria c

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

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:v>

```

2.1.2. Invariante de Representación

1. Para cada '*padre*' obtener el significado devolvera un puntero(datosCat) donde '*categoria*' es igual a la clave
2. Para toda clave '*padre*' que exista en '*familia*' debera ser o raiz o pertenecer a algun conjunto de punteros de '*hijos*' de alguna clave '*padre*'
3. Todos los elementos de '*hijos*' de una clave '*padre*', cada uno de estos hijos tendran como '*abuelo*' a ese '*padre*' cuando sean clave.
4. '*cantidad*' sera igual a la longitud de la lista '*categorias*'.
5. Cuando la clave es igual a '*raiz*' la '*altura*' es 1.
6. La '*altura*' del puntero a datosCat de cada clave es menor o igual a '*alturaMax*'.
7. Existe una clave en la cual, la '*altura*' del significado de esta es igual a '*alturaMax*'.
8. Los '*hijos*' de una clave tienen '*altura*' igual a $1 + \text{'altura de la clave'}$.
9. Todos los '*id*' de significado de cada clave deberan ser menor o igual a '*cant*'.
10. No hay '*id*' repetidos en el '*familia*'.
11. Todos los '*id*' son consecutivos.

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

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

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

2.1.3. Función de Abstraccion

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

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

Auxiliares

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

2.1.4. Algoritmos

Algoritmo: 1

ICATEGORIAS (**in** $\text{ac} : \text{estrAC}$) $\rightarrow \text{res} : \text{conj}(\text{categoria})$
 $\text{res} \leftarrow \text{claves}(\text{ac.familia}) \text{ // } O(\text{ALGO})$

Complejidad:

Algoritmo: 2

IRAIZ (**in** $\text{ac} : \text{estrAC}$) $\rightarrow \text{res} : \text{categoria}$
 $\text{res} \leftarrow (*\text{ac.raiz}).\text{categoria} \text{ // } O(1)$

Complejidad: O(1)

Algoritmo: 3

IPADRE (**in** $\text{ac} : \text{estrAC}$, **in** $\text{h} : \text{categoria}$) $\rightarrow \text{res} : \text{puntero}(\text{categoria})$
 $\text{res} \leftarrow (*(*\text{obtener}(\text{h}, \text{ac.familia}))).\text{abuelo}.\text{categoria} \text{ // } O(\text{ALGO})$

Complejidad:

Algoritmo: 4

IID (**in** $\text{ac} : \text{estrAC}$, **in** $\text{c} : \text{categoria}$) $\rightarrow \text{res} : \text{puntero}(\text{nat})$
 $\text{res} \leftarrow (*\text{obtener}(\text{c}, \text{ac.familia})).\text{id} \text{ // } O(|c|)$

Complejidad: O(|c|)

Algoritmo: 5

INUEVO (**in** $\text{c} : \text{categoria}$) $\rightarrow \text{res} : \text{estrAC}$
 $\text{res.cantidad} \leftarrow 1 \text{ // } O(1)$
 $\text{res.raiz} = \text{c} \text{ // } O(1)$
 $\text{res.alturaMax} = 1 \text{ // } O(1)$
 $\text{var tuplaA} : \text{datosCat}$
 $\text{var punt} : \text{puntero}(\text{datosCat})$
 $\text{tuplaA} \leftarrow (\text{c}, 1, 1, \text{esVacia?}, \text{punt})$
 $\text{punt} \leftarrow \text{puntero}(\text{tuplaA})$
 $\text{res.familia} = \text{definir}(\text{padre}, \text{punt}, \text{res.familia}) \text{ // } O(|c|)$
 $\text{res.categorias} \leftarrow \text{agregarAtras}(\text{tuplaA}, \text{res.categorias})$

Complejidad: O(|c|)

Algoritmo: 6

IAGREGAR (**in/out** $\text{ac} : \text{estrAC}$, **in** $\text{c} : \text{categoria}$, **in** $\text{h} : \text{categoria}$)
 $\text{var puntPadre} : \text{puntero}(\text{datosCat}) \text{ // } O(1)$

```

puntPadre ← (obtener(c,ac.familia)) //O(|c|)
if (*puntPadre).altura == ac.alturaMax //O(1)
then ac.alturaMax = ac.alturaMax + 1 //O(1)
ELSE ac.alturaMax = ac.alturaMax FI //O(1)
var tuplaA : datosCat //O(1)
var punt : puntero(datosCat) //O(1)
tuplaA ← (h,ac.cantidad +1,(*puntPadre).altura +1,esVacia?,puntPadre) //O(|h|)
punt ← puntero(tuplaA) //O(1)
Agregar((*puntPadre).hijos,punt) // O(1)
definir(h,punt,ac.familia) //O(|h|)
ac.cantidad ++ //O(1)
agregarAtras(tuplaA,res.categorias) //O(1)
Complejidad: O(|c|+|h|)

```

Algoritmo: 7

```

IALTURA (in ac: estrAC) → 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
    var puntPadre : puntero(datosCat) //O(1)
    puntPadre ← (obtener(c,ac.familia)) //O(|c|)
    res ← false // O(1)
    if c == ac.raiz O(|c|)
    then res ← true // O(1)
    ELSE actual ← h // O(1)
    while(res ≠ true ∧ actual ≠ ac.raiz)
    if actual ∈ (*puntPadre).hijos
    then res ← true
    ELSE actual ← (*(obtener(actual,ac.familia)) ).abuelo FI FI
Complejidad:

```

Algoritmo: 10

```

IALTURACATEGORIA (in ac: estrAC, in c: categoria) → res:nat
    res ← (*(obtener(c,ac.familia)).altura // O(|c|)
Complejidad: O(|c|)

```

Algoritmo: 11

```

IIHIJOS (in ac: estrAC, in c: categoria) → res:conj(categoria)
    res ← (*obtener(c,ac.familia)).hijos // O(ALGO) PREGUNTAR!!! EN ESTE LA COMPLEJIDAD ES EL ITER-
    ADOR DEVOLVEMOS EL PUNTERO? Complejidad:

```

Algoritmo 12

```

IOBTENER (in c: categoria, in ac: estrAC) → res:puntero(datosCat)
    res ← 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)

3. Renombres

TAD CATEGORIA

es String

Fin TAD

TAD LINK

es String

Fin TAD

TAD FECHA

es Nat

Fin TAD