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** accesoReciente-Dia(s,l',f') **else** 0 **fi**

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

linksOrdenadosPorAccesos(s, c) $\equiv$ linksOrdernadosPorAccesosAux(s, c, linksCategoriaOHijos(s, c))

linksOrdenadosPorAccesosAux(s,c,ls) $\equiv$ **if** $\emptyset$?(ls) **then**
$\emptyset$
else
linkConMasAccesos(s, c, ls) $\bullet$ linksOrdernadosPorAccesosAux(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

β(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$=_{\text{obs}}$ 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$=_{\text{obs}}$ 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$=_{\text{obs}}$ categoriaLink(s,l)
Complejidad : O(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$=_{\text{obs}}$ 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$=_{\text{obs}}$ 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$=_{\text{obs}}$ accesosRecientesDia(s,l,f)
Complejidad : O(#accesosRecientesDia(s,l,f))
Descripción : Devuelve la cantidad de accesos a un link un cierto dia
Aliasing:ALGO

inicar (**in** ac: estrAC) $\longrightarrow$ res: lli

Pre $\equiv$ true
Post $\equiv$ res$=_{\text{obs}}$ 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) $\wedge$ $s_0 =_{\text{obs}} s$
Post $\equiv$ s$=_{\text{obs}}$ nuevoLink(s_0,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) $\wedge$ $s_0 =_{\text{obs}} s$
Post $\equiv$ s$=_{\text{obs}}$ acceso(s_0,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$=_{\text{obs}}$ 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 ≡ c ∈ categorias(s) ∧ l ∈ links(s)
Post ≡ res$=_{obs}$ 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) ⟶ res: secu(link)

Pre ≡ c ∈ categorias(s)
Post ≡ res$=_{obs}$ 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) ⟶ res: nat

Pre ≡ c ∈ categorias(s)
Post ≡ res$=_{obs}$ 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) ⟶ res: fecha

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

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

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

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

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

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

Pre ≡ c ∈ categorias(s)
Post ≡ res$=_{obs}$ diasRecientesParaCategorias(s,c)
Complejidad : O(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)) ⟶ res: link

Pre ≡ c ∈ categorias(s) ∧ esVacia??(ls) ∧ ls ⊆ linksCategoriasOHijos(s,c)
Post ≡ res$=_{obs}$ linkConUltimoAcceso(s,c,ls)
Complejidad : O(#ls??)

Descripción : Devuelve el link que se accedio 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 $\in$ links(s) $\wedge$ fs $\subseteq$ diasRecientes(s,l)
Post $\equiv$ res$=_{\text{obs}}$ 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 $\in$ categorias(s) $\wedge$ ls $\subseteq$ linksCategoriasOHijos(s,c)
Post $\equiv$ res$=_{\text{obs}}$ 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)>

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*.
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 $\Longleftrightarrow$

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

1.1.3. Función de Abstraccion

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

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

Auxiliares

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

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

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 $\leftarrow$ ultimo(*((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($\neg$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) $\longrightarrow$ res: estrLLI

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

res.arbolCategorias $\leftarrow$ &ac //O(1)

var c: nat //O(1)

c $\leftarrow$ 1 //O(1)

res.arrayCantLinks $\leftarrow$ crearArreglo(#categorias(ac)) //O(1)

res.listaLinks $\leftarrow$ vacia() //O(1)

res.accesosXLink $\leftarrow$ vacio() //O(1)

while (c $\leq$ #categorias(ac)) //O(#categorias(ac))

linksFamilia llist $\leftarrow$ vacia() //O(1)

res.arrayCatLinks[c] $\leftarrow$ llist //O(1)

c ++ //O(1)

end while //O(1)

Complejidad: (#categorias(ac))

Algoritmo: 8

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

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

lista(acceso) accesoDeNuevoLink $\leftarrow$ vacia() //O(1)

datosLink nuevoLink $\leftarrow$ $<$l,cat,accesoDeNuevoLink$>$ //O(|l|)

puntero(datosLink) puntLink $\leftarrow$ nuevoLink //O(1)

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

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

while(cat $\neq$ puntRaiz(s.arbolCategorias)) //O(h)

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

cat $\leftarrow$ cat.padre //O(1)

end while

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

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

Algoritmo: 9

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

if s.actual == f //O(1)

then s.actual ← s.actual //O(1)

else s.actual ← f FI //O(1)

var puntero(datosLink) puntLink ← obtener(l,s.accesosXLink) //O(|l|)

if (ultimo((*puntLink).accesos)).dia == f //O(1)

then (ultimo((*puntLink).accesos)).cantAccesos++ //O(1)

else agregarAtras((*puntLink).accesos), f) FI //O(1)

if longitud((*puntLink).accesos) == 4 //O(1)

then fin((*puntLink).accesos) //O(1)

fi

Complejidad: O(|l|)

Algoritmo: 10

IESRECIENTE? (**in** s: lli, **in** l: link , **in** f: fecha) ⟶ res: bool

res ← menorReciente(s,l) ≤ f ∧ f ≤ fechaUltimoAcceso(s,l) //O(|l|)

Complejidad: O(|l|)

Algoritmo: 11

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

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

Complejidad: O(|l|)

Algoritmo: 12

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

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

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

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

//O(1)

if ¬iestaOrdenada?(s.arrayCantLinks[id]) //O(1)

then

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

itMax ← iBuscarMax(s.arrayCantLinks[id]) //O(n)

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

eliminarSiguiente(itMax) //O(1)

fi

end while

Complejidad: O($\mathbf{n^2}$)

Algoritmo: 13

IBUSCARMAX (**in** ls: lista(puntero(datosLink))) ⟶ res: itLista(puntero(datosLink))

res ← crearIt(ls) //O(1)

itLista(puntero(datosLink)) itRecorre ← crearIt(ls) //O(1)

nat max ← (*siguiente(itRecorre)).cantAccesosRecientes //O(1)

while(haySiguiente(itRecorre)) //O(1)

if max <(*siguiente(itRecorre)).cantAccesosRecientes //O(1)

then //O(1)

max ← (*siguiente(itRecorre)).cantAccesosRecientes //O(1)

res ← itRecorre //O(1)

end while

avanzar(itRecorre) //O(1)

end while //O(1)

Complejidad: O(n)

Algoritmo: 14

IESTAORDENADA (**in** ls: lista(puntero(datosLink))) ⟶ res: bool

res ← true //O(1)

itLista(puntero(datosLink)) itRecorre ← crearIt(ls) //O(1)

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

while(haySiguiente(itRecorre) ∧ res == true)

//O(1)

avanzar(itRecorre) //O(1)

if aux <(*siguiente(itRecorre)).cantAccesosRecientes //O(1)

then //O(1)

res ← false //O(1)

fi //O(1)

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

end while //O(1)

Complejidad: O(n)

Algoritmo: 15

ICANTLINKS (**in** s: lli, **in** c: categoria) ⟶ 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: fechaend while

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

Complejidad: O(1)

Algoritmo: 17

IDIASRECIENTES (**in** s: lli, **in** l: link) ⟶ res: conj(fecha)end while

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

Complejidad:O(|l|)

Algoritmo: 18

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

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

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

fecha++

//O(1)

end while

Complejidad: O(|l|)

Algoritmo: 19

IDIASRECIENTESPARACATEGORIAS (**in** s: lli, **in** c: categoria) ⟶ res: conj(fecha)end while

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: natend while

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: linkend while

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

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

avanzar(ls) //O(1)

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

then max ← (siguiente(ls)) //O(1)

fi //O(1)

end while

res ← (*max).link //O(|(*max).link|)

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

1.2. Descripcion de Complejidades de Algoritmos

1. **ICATEGORIAS**:
   Devuelve el arbol de categorias del sistema, esto cuesta O(1).
   Orden Total:O(1)=**O(1)**

2. **ILINKS**:
   Se crea un conjunto vacio, esto tarda O(1). Se crea un itLista, esto tarda O(1).
   Se ingresa a un ciclo preguntando si haySiguiente, esto cuesta O(1), se le agrega link apuntado de cada tupla de datosLink de la lista listaLinks, esto tarda O(|l|), luego se avanza el it, esto cuesta O(1).
   Al salir del ciclo, se devuelve el conjunto.
   Orden Total:O(1)+O(1)+O(1)+(suma O(|l|))+O(1)=**O(suma O(|l|))**

3. **ICATEGORIALINK**:
   Se utiliza la operacion obtener del diccionario accesosXLink, la cual devuelve un puntero a datosLink, se devuelve lo apuntado a catDLink, esto cuesta O(|l|).
   Orden Total:O(|l|)=**O(|l|)**

4. **IFECHAACTUAL**:
   Devuelve la fecha actual del sistema, esto cuesta O(1).
   Orden Total:O(1)=**O(1)**

5. **IFECHAULTIMOACCESO**:
   Se utiliza la operacion obtener del diccionario accesosXLink, la cual devuelve un puntero a datosLink, se accede a la lista accesosRecientes dentro de la tupla, se devuelve dia del ultimo elemento, esto cuesta O(|l|).
   Orden Total:O(|l|)=**O(|l|)**

6. **IACCESOSRECIENTESDIA**:
   Se crea una lista de acceso vacia, esto cuesta O(1). Se le guarda a la lista, la lista de accesosRecientes, la cual se obtiene con la operacion obtener del diccionario accesosXLink consultando por el link dado, esto cuesta O(|l|).
   Se ingresa a un ciclo, preguntando si no es vacia la lista, esto cuesta O(1).
   Se pregunta si dia del primer elemento de la lista es igual a f, esto cuesta O(1), en caso verdadero se devuelve cantAccesos de esa tupla, esto cuesta O(1), en caso falso se modifica la lista sacando el primer elemento, esto cuesta O(1). Se sale del ciclo.
   Orden Total:O(1)+O(|l|)+O()=**O(|l|)**

7. **IINICIAR**:
   Se guarda en res.actual la fecha igual a 1, esto cuesta O(1). Se pasa por referencia el arbol dado y se lo guarda en res.arbolCategorias, estoy cuesta O(1). Se crea una variable del tipo nat, cuesta O(1) , se inicializa esta variable con 1, esto cuesta O(1), se crea un arreglo con tamaño igual a #categorias(ac) y se lo guarda en res.arrayCatLinks, esto cuesta O(1),
   se inicializa res.listaLinks como vacia, esto cuesta O(1), se inicializa con vacio el diccionario res.accesosXLink.
   Se ingresa a un ciclo consultando si c es menor o igual a la cantidad de categorias de ac, esto cuesta O(1). Se crea una lista linksFamilia inicializada con vacio, esto cuesta O(1).
   Se guarda en res.arrayCatLinks[c] la lista linksFamilia, esto cuesta O(1), se le suma 1 a c, esto cuesta O(1). Se sale del ciclo, esto cuesta O(1).
   Orden Total:O(1)+O(1)+O(1)+O(1)+O(1)+O(1)+O(1)+O(1)+
   O(#categorias(ac)*(O(1)+O(1)+O(1)))=**O(#categorias(ac))**

8. **INUEVOLINK**:
   Se crea un puntero a datosCat cat donde se le pasa el puntero obtenido por la operacion obtener del modulo arbolCategorias, esto cuesta O(|c|). Se crea una lista de acceso inicializada vacia, que cuesta O(1).
   Se crea una tupla datosLink, a la cual se le pasa una tupla con el link dado, el puntero a datosCat y la lista de acceso, la cual tarda O(|l|). Se crea un puntero a datosLink y se le pasa la tupla datosLink, esto cuesta O(1).
   Se utiliza la operacion definir del diccTrie en la cual se agrega el link dado al diccionario accesosXLink, lo cual tarda O(|l|).
   Se utiliza la operacion agregarAtras que agrega el puntero a datosLink a la lista listaLinks, esto demora O(1).
   Se ingresa a un ciclo si cat es distinto de la operacion puntRaiz de arbolCategorias, esto tarda O(1). Se utiliza la operacion agregarAtras que agrega el puntero a datosLink a la lista que esta en la posicion (*cat).id del arreglo arrayCatLinks, lo cual tarda O(1).
   Se modifica el puntero a datosCat y se guarda cat.padre, lo cual tarda O(1) . Se sale del ciclo tardando O(1). Se utiliza la operacion agregarAtras que agrega el puntero a datosLink a la lista que esta en la posicion (*cat).id del arreglo arrayCatLinks, lo cual tarda O(1).
   Orden Total:O(|c|)+O(1)+O(|l|)+O(1)+O(1)+O(1)+O(h*(O(1)+O(1)))+O(1)=**O(|l|+|c|+h)**

9.

10.

11.

12.

13.

14.

15.

16.

17.

18.

19.

20.

21.

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 Categorias

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$=_{\text{obs}}$ 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$=_{\text{obs}}$ raiz(ac)
Complejidad : O(1)
Descripción : Devuelve la raiz del arbol ac
Aliasing:ALGO

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

Pre $\equiv$ h $\in$ ac $\wedge$ raiz(ac) $\neq$ h
Post $\equiv$ res$=_{\text{obs}}$ 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$=_{\text{obs}}$ id(ac,c)
Complejidad : O(|c|)
Descripción : Devuelve el id de una categoria c en el arbol ac
Aliasing:ALGO

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

Pre $\equiv$ $\neg$vacia?(c)
Post $\equiv$ res$=_{\text{obs}}$ nuevo(c)
Complejidad : O(|c|)
Descripción : Crea un arbol
Aliasing:ALGO

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

Pre $\equiv$ c $\in$ ac $\wedge$ $\neg$vacia?(h) $\wedge$ $ac_0 =_{\text{obs}}$ ac
Post $\equiv$ ac$=_{\text{obs}}$ 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$=_{\text{obs}}$ 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$=_{\text{obs}}$ 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$=_{\text{obs}}$ 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$=_{\text{obs}}$ 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$=_{\text{obs}}$ 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: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 + '*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 : estrAC $\longrightarrow$ bool
Rep(e) $\equiv$ true $\Longleftrightarrow$

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

2.1.3. Función de Abstraccion

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

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

Auxiliares

todasLasCategorias : secu(datosCat) $\longrightarrow$ conj(categoria)
Ag((prim(cs)).categoria,fin(cs)) $\equiv$

2.1.4. Algoritmos

Algoritmo: 1

ICATEGORIAS (**in** ac: estrAC) $\longrightarrow$ res: conj(categoria)end while

res $\leftarrow$ claves(ac.familia) //O(ALGO)

Complejidad:

Algoritmo: 2

IRAIZ (**in** ac: estrAC) $\longrightarrow$ res: categoriaend while

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

Complejidad: O(1)

Algoritmo: 3

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

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

Complejidad:

Algoritmo: 4

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

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

Complejidad: O(|c|)

Algoritmo: 5

INUEVO (**in** c: categoria) ⟶ res:estrACend while

res.cantidad ← 1 //O(1)

res.raiz = c //O(1)

res.alturaMax = 1 //O(1)

var tuplaA : datosCat //O(1)

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

tuplaA ← (c,1,1,esVacia?,punt) //O(1)

punt ← puntero(tuplaA) //O(1)

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

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

Complejidad: O(|c|)

Algoritmo: 6

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

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

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

if (*puntPadre).altura == ac.alturaMax //O(1)

then ac.alturaMax = ac.alturaMax + 1

//O(1)

ELSE ac.alturaMax = ac.alturaMax FI //O(1)

var tuplaA : datosCat //O(1)

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

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

punt ← puntero(tuplaA) //O(1)

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

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

ac.cantidad ++ //O(1)

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

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

Algoritmo: 7

IALTURA (**in** ac: estrAC) ⟶ res:natend while

res ← ac.alturaMax //O(1)

Complejidad: O(1)

Algoritmo: 8

IESTA? (**in** c: categoria,**in** ac: estrAC) ⟶ res:boolend while

res ← def?(c,ac.familia) //O(|c|)

Complejidad: O(|c|)

Algoritmo: 9

IESSUBCATEGORIA (**in** ac: estrAC, **in** c: categoria,**in** h: categoria) ⟶ res:boolend while

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

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

res ← false //O(1)

if c == ac.raiz //O(|c|)

then res ← true //O(1)

ELSE actual ← h //O(1)

while(res ≠ true ∧ actual ≠ ac.raiz)

//O(1)

if actual ∈ (*puntPadre).hijos //O(1)

then res ← true //O(1)

ELSE actual ← (*(obtener(actual,ac.familia))).abuelo FI FI //O(1)

Complejidad:

Algoritmo: 10

IALTURACATEGORIA (**in** ac: estrAC, **in** c: categoria) ⟶ res:natend while

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

Complejidad: O(|c|)

Algoritmo: 11

IHIJOS (**in** ac: estrAC, **in** c: categoria) ⟶ res:conj(categoria)end while

res ← (*obtener(c,ac.familia)).hijos // O(ALGO) PREGUNTAR!!! EN ESTE LA COMPLEJIDAD ES EL ITERADOR DEVOLVEMOS EL PUNTERO? ——————————————————————————
Complejidad:

Algoritmo 12

IOBTENER (**in** c: categoria, **in** ac: estrAC) ⟶ res:puntero(datosCat)end while

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

Complejidad: O(|c|)

Algoritmo: 13

IPUNTRAIZ (**in** ac: estrAC) ⟶ res:puntero(datosCat)end while

res ← ac.raiz //O(1)

Complejidad: O(1)

2.2. Descripcion de Complejidades de Algoritmos

1.

2.

3.

4.

5.

6.

7.

8.

9.

10.

11.

12.

13.

14.

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

Nodo es tuplaarregloarreglo(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 estÃ¡n 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 : estrAC $\longrightarrow$ bool
Rep(e) $\equiv$ true $\Longleftrightarrow$

1.

2.

3.

Funciones auxiliares

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

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

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

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

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

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

SonTodosNullOLosHijosLoSon : estrDT $\longrightarrow$ Bool

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

Los27SonNull : estrDT $\times$ nat $\longrightarrow$ Bool

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

BuscarHijosNull : estrDT $\times$ nat $\longrightarrow$ Bool

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

2.2.2. Función de Abstracción

Abs: estr e $\rightarrow$ diccT(c,α)

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

Funciones auxiliares

estaDefinido? : string $\times$ estrDT $\longrightarrow$ bool

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

NodoDef? : string $\times$ Nodo $\longrightarrow$ bool

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

numero : char $\longrightarrow$ nat

numero(char) $\equiv$ char - a

ObtenerS : string $\times$ Nodo $\longrightarrow$ α

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

3. Renombres

TAD Categoria

es String

Fin TAD

TAD Link

es String

Fin TAD

TAD Fecha

es Nat

Fin TAD