Algoritmos y Estructuras de Datos II

Trabajo Práctico 2

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

Segundo Cuatrimestre de 2012

Grupo 9

Apellido y Nombre	LU	E-mail
María Candela Capra Coarasa	234/11	canduh_27@hotmail.com
Leandro Lovisolo	645/11	leandro@leandro.me
Gastón de Orta	244/11	gaston.deorta@hotmail.com
Lautaro José Petaccio	443/11	lausuper@gmail.com

Reservado para la cátedra

Instancia	Docente que corrigió	Calificación
Primera Entrega		
Recuperatorio		

Índice

1. Módulo ÁrbolCategorías

Interfaz

se explica con: ÁrbolCategorías, Iterador Unidireccional(Categoría).

géneros: `acat`, `itcats`.

Operaciones básicas de árbol de categorías

CrearÁrbol(**in** *raiz*: `categoria`) → *res* : `acat`
Pre ≡ {¬vacía?(*raiz*)}
Post ≡ {*res* =$_{\text{obs}}$ nuevo(*raiz*)}
Complejidad: Θ(|*raiz*|)
Descripción: crea un árbol nuevo cuya categoría raíz es *raiz*.

NombreCategoríaRaíz(**in** *ac*: `acat`) → *res* : `categoria`
Pre ≡ {true}
Post ≡ {*res* =$_{\text{obs}}$ raíz(*ac*)}
Complejidad: Θ(1)
Descripción: devuelve el nombre de la categoría raíz de *ac*.
Aliasing: *res* no es modificable.

AgregarCategoría(**in** *hija*: `categoria`, **in** *padre*: `categoria`, **in/out** *ac*: `acat`)
Pre ≡ {*ac* =$_{\text{obs}}$ ac_0 ∧ está?(*padre*, *ac*) ∧ ¬vacía?(*hija*) ∧ ¬está?(*hija*, *ac*)}
Post ≡ {*ac* =$_{\text{obs}}$ agregar(ac_0, *padre*, *hija*)}
Complejidad: Θ(|*padre*| + |*hija*|)
Descripción: agrega la categoría *hija* como hija de la categoría *padre*.
Aliasing: la categoría *hija* se agrega por copia.

IdCategoríaPorNombre(**in** *c*: `categoria`, **in** *ac*: `acat`) → *res* : `nat`
Pre ≡ {está?(*c*, *ac*)}
Post ≡ {*res* =$_{\text{obs}}$ id(*ac*, *c*)}
Complejidad: Θ(|*c*|)
Descripción: devuelve el *id* de la categoría *c*.

#Categorías(**in** *ac*: `acat`) → *res* : `nat`
Pre ≡ {true}
Post ≡ {*res* =$_{\text{obs}}$ #(categorias(*ac*))}
Complejidad: Θ(1)
Descripción: devuelve la cantidad de categorías en *ac*.

Operaciones del iterador de categorías

CrearIt(**in** *padre*: `categoria`, **in** *ac*: `acat`) → *res* : `itcats`
Pre ≡ {está?(*padre*, *ac*)}
Post ≡ {*res* =$_{\text{obs}}$ CrearItUni(tuplasHijos(*padre*, *ac*)) ∧ alias(iteraLosHijos(*res*, *padre*, *ac*))}
Complejidad: Θ(|*padre*|)
Descripción: devuelve un iterador unidireccional de las categorías hijas directas de la categoría *padre*.
Aliasing: Siguientes(*res*) podrá cambiar si se agregan nuevas categorías hijas directas a la categoría *padre*.

CrearItRaíz(**in** *ac*: `acat`) → *res* : `itcats`
Pre ≡ {true}
Post ≡ {*res* =$_{\text{obs}}$ CrearItUni(tuplasHijos(raíz(*ac*), *ac*)) ∧ alias(iteraLosHijos(*res*, raíz(*ac*), *ac*))}
Complejidad: Θ(1)
Descripción: devuelve un iterador unidireccional de las categorías hijas directas de la categoría raíz de *ac*.
Aliasing: Siguientes(*res*) podrá cambiar si se agregan nuevas categorías hijas directas a la categoría raíz.

CrearItHijos(**in** *it*: `itcats`, **in** *ac*: `acat`) → *res* : `itcats`
Pre ≡ {HayMás?(*it*)}
Post ≡ {*res* =$_{\text{obs}}$ CrearItUni(tuplasHijos(Π_1(Actual(*it*)), *ac*)) ∧ alias(iteraLosHijos(*res*, Π_1(Actual(*it*)), *ac*))}
Complejidad: Θ(1)

Descripción: devuelve un iterador unidireccional de las categorías hijas directas de la categoría actual del iterador *it*.
Aliasing: Siguientes(*res*) podrá cambiar si se agregan nuevas categorías hijas directas a la categoría actual del iterador *it*.

HayMás?(**in** *it* : `itcats`) → *res* : `bool`
Pre ≡ {true}
Post ≡ {*res* =$_{\text{obs}}$ HayMás?(*it*)}
Complejidad: Θ(1)
Descripción: devuelve **true** si y sólo si en el iterador todavía quedan elementos para avanzar.

CategoríaActual(**in** *it* : `itcats`) → *res* : `categoria`
Pre ≡ {HayMás?(*it*)}
Post ≡ {*res* =$_{\text{obs}}$ Π_1(Actual(*it*))}
Complejidad: Θ(1)
Descripción: devuelve el elemento actual del iterador.
Aliasing: *res* no es modificable.

IdCategoríaActual(**in** *it* : `itcats`) → *res* : `nat`
Pre ≡ {HayMás?(*it*)}
Post ≡ {*res* =$_{\text{obs}}$ Π_2(Actual(*it*))}
Complejidad: Θ(1)
Descripción: devuelve el id del elemento actual del iterador.

Avanzar(**in/out** *it* : `itcats`)
Pre ≡ {*it* =$_{\text{obs}}$ it_0 ∧ HayMás?(it_0)}
Post ≡ {*it* =$_{\text{obs}}$ Avanzar(it_0)}
Complejidad: Θ(1)
Descripción: avanza el iterador a la posición siguiente.

Especificación de las operaciones auxiliares utilizadas en la interfaz

TAD Árbol Extendido

extiende ÁrbolCategorías

otras operaciones (no exportadas)

tuplasHijos	: categoría c × acat ac	⟶ secu(tupla(categoría, nat))		{está?(c, ac)}
categoríasATuplas	: conj(categoría) cc × acat ac	⟶ secu(tupla(categoría, nat))		{$cc \subseteq$ categorías(ac)}
iteraLosHijos	: itUni(tupla(categoría, nat)) × categoría c × acat ac	⟶ bool		{está?(c, ac)}
terminanIgual	: secu(tupla(categoría, nat)) × secu(tupla(categoría, nat))	⟶ bool		

axiomas ∀c: categoría, ∀ac: acat, ∀cc: conj(categoría), ∀sc: secu(categoría),
∀st, st': secu(tupla(categoría, nat)), ∀it: itUni(tupla(categoría, nat))

tuplasHijos(c, ac) ≡ catsATuplas(hijos(ac, c), ac)

catsATuplas(cc, ac) ≡ **if** ∅?(cc) **then** <>
else ⟨dameUno(cc), id(ac, dameUno(cc))⟩ • categoríasATuplas(sinUno(cc), ac)
fi

iteraLosHijos(it, c, ac) ≡ terminanIgual(Siguientes(it), tuplasHijos(c, ac))

terminanIgual(st, st') $\equiv$ **if** vacía?(st) $\vee$ vacía(st') **then**
true
else
ult(st) $=_{\text{obs}}$ ult(st') $\wedge$ terminanIgual(com(st), com(st'))
fi

Fin TAD

Representación

Representación del árbol de categorías

Representamos el árbol de categorías a partir de un diccionario trie de tuplas `estr_cat`, que contienen toda la información necesaria para representar cada categoría individualmente.

El diccionario nos permite acceder una categoría arbitraria c en $\Theta(|c|)$.

Además, guardamos una referencia a la categoría raíz, lo cual nos permite acceder a la misma en $\Theta(1)$.

`acat` se representa con `estr_acat`

donde `estr_acat` es `tupla`(*raíz*: `estr_cat`,
categorías: `dicctrie(estr_cat)`)
donde `estr_cat` es `tupla`(*id*: `nat`,
nombre: `categoria`,
hijos: `conj(puntero(estr_cat))`)

Invariante de representación:

1. La raíz tiene que estar en el diccionario de categorías.
2. La raíz tiene que tener id 1.
3. Para todas las categorías en el diccionario:
   a) El nombre de la categoría deber ser igual a su clave en el diccionario.
   b) El id de la categoría debe estar en rango.
   c) Dos categorías no pueden tener el mismo id.
   d) Para todos los hijos de la categoría:
      1) El hijo no puede ser nulo.
      2) El hijo tiene que estar en el diccionario de categorías.
      3) El hijo no puede estar en el conjunto de hijos de otra categoría.
      4) El hijo debe tener un id superior al de la categoría padre.

Rep : estr_acat $\longrightarrow$ bool
Rep(e) $\equiv$ true $\iff$

def?(e.raíz.nombre, e.categorías) $\wedge$	1.
e.raíz.id $=_{\text{obs}}$ 1 $\wedge$	2.
$(\forall c$: categoria$)$(def?(c, e.categorías) $\Rightarrow_{\text{L}}$ (	3.
cat.nombre $=_{\text{obs}}$ c $\wedge$	3. a)
$1 \leq cat$.id $\wedge$ cat.id $\leq$ #(claves(e.categorías)) $\wedge$	3. b)
$(\forall c'$: categoria$)$(def?(c', e.categorías) $\Rightarrow_{\text{L}}$ (cat.id $=_{\text{obs}}$ cat'.id $\iff$ $c =_{\text{obs}} c'$)) $\wedge$	3. c)
$(\forall h$: puntero(estr_cat)$)(h \in cat$.hijos $\Rightarrow_{\text{L}}$ (	3. d)
$\neg(h =_{\text{obs}}$ NULL$) \wedge_{\text{L}}$	3. d) 1)
def?($h\rightarrow$nombre, e.categorías) $\wedge_{\text{L}}$ $h =_{\text{obs}}$ &(obtener($h\rightarrow$nombre, e.categorías)) $\wedge$	3. d) 2)
$(\forall c'$: categoria$)$(def?(c', e.categorías) $\Rightarrow_{\text{L}}$ ($h \in cat'$.hijos $\iff$ $c =_{\text{obs}} c'$)) $\wedge$	3. d) 3)
$h\rightarrow$id $>$ cat.id	3. d) 4)
))	
)), donde cat es obtener(c, e.categorías) y cat' es obtener(c', e.categorías).	

Función de abstracción:

1. El conjunto de categorías del árbol debe ser igual al conjunto de claves del diccionario de la estructura.
2. La raíz del árbol debe ser igual a la raíz de la estructura.
3. Para todas las categorías en el árbol:
   a) El id en el árbol y la estructura deben coincidir.
   b) Los hijos de la categoría en la estructura deben tener como padre a la categoría en el árbol.

Abs : estr_acat e $\longrightarrow$ acat {Rep(e)}
Abs(e) $=_{\text{obs}}$ ac: acat | categorias(ac) $=_{\text{obs}}$ claves(e.categorías) $\wedge_{\text{L}}$ 1.
raíz(ac) $=_{\text{obs}}$ e.raíz.nombre $\wedge$ 2.
($\forall c$: categoria)($c \in$ categorías(ac) $\Rightarrow_{\text{L}}$ (3.
id(ac, c) $=_{\text{obs}}$ obtener(c, e.categorías).id $\wedge$ 3. a)
($\forall h$: puntero(estr_cat))($h \in$ obtener(c, e.categorías).hijos) $\Rightarrow_{\text{L}}$ 3. b)
padre(ac, h→nombre) $=_{\text{obs}}$ c
)
))

Representación del iterador

El iterador de categorías permite recorrer el conjunto de hijos directos de una categoría arbitraria.
Su representación es simplemente un iterador del conjunto de hijos de la categoría en cuestión, es decir, un iterador del campo *hijos* de su tupla `estr_cat`.

`itcats` **se representa con** `itConj(puntero(estr_cat))`

Invariante de representación:

1. La secuencia de elementos siguientes del iterador no puede contener punteros nulos.

Rep : itConj(puntero(estr_cat)) $\longrightarrow$ bool
Rep(it) $\equiv$ true $\iff \neg$(está?(NULL, Siguientes(it))) 1.

Función de abstracción:

1. Los elementos siguientes del iterador deben ser las tuplas (categoría, nat) que se correspondan con los elementos siguientes de la instancia de itConj(puntero(estr_cat)).

Abs : itConj(puntero(estr_cat)) $itconj$ $\longrightarrow$ itUni(tupla(categoría, nat)) {Rep($itconj$)}
Abs($itconj$) $=_{\text{obs}}$ it: itUni(tupla(categoría, nat)) | Siguientes(it) $=_{\text{obs}}$ SecuDeTuplas(Siguientes($itconj$)) 1.

SecuDeTuplas : secu(puntero(estr_cat)) sp $\longrightarrow$ secu(tupla(categoria, nat)) {$\neg$(está?(NULL, sp)}
SecuDeTuplas(sp) $\equiv$ **if** vacía?(sp) **then** $<>$ **else** $\langle$prim(sp)→nombre, prim(sp)→id$\rangle$ • SecuDeTuplas(fin(sp)) **fi**

Algoritmos

iCrearÁrbol(**in** *raiz*: `categoria`) → *res* : `estr_acat`
var *estr_raiz*: `estr_cat` ← ⟨`id`: 1, Θ(1)
`nombre`: *raiz*, Θ(1)
`hijos`: Vacío()⟩ Θ(1)
var *categorias*: `dicctrie(estr_cat)` ← CrearDiccionario() Θ(1)
Definir(*raiz*, *estr_raiz*, *categorias*) Θ(|*raiz*| + *copy*(*estr_raiz*))
res ← ⟨`raíz`: *estr_raiz*, `categorías`: *categorias*⟩ Θ(1)

Complejidad: Θ(|*raiz*|)

Θ(1) + Θ(1) + Θ(1) + Θ(1) + Θ(|*raiz*| + *copy*(*estr_raiz*)) + Θ(1) =
5 ∗ Θ(1) + Θ(|*raiz*| + *copy*(*estr_raiz*)) =
Θ(|*raiz*| + *copy*(*estr_raiz*)) =
Θ(|*raiz*|) + Θ(*copy*(*estr_raiz*)) =
Θ(|*raiz*|) + Θ(|*raiz*|) =
Θ(|*raiz*|).

iNombreCategoríaRaíz(**in** *ac*: `estr_acat`) → *res* : `categoria`
res ← *ac*.raíz.nombre Θ(1)

Complejidad: Θ(1)

iAgregarCategoría(**in** *hija*: `categoria`, **in** *padre*: `categoria`, **in/out** *ac*: `estr_acat`)
var *estr_padre*: `estr_cat` ← Obtener(*padre*, *ac*.categorías) $\Theta(|padre|)$
var *estr_hija*: `estr_cat` ← ⟨`id`: #Claves(*ac*.categorías) + 1, $\Theta(1)$
`nombre`: *hija*, $\Theta(1)$
`hijos`: Vacío()⟩ $\Theta(1)$
Definir(*hija*, *estr_hija*, *ac*) $\Theta(|hija| + copy(estr\_hija))$
estr_hija ← Obtener(*hija*, *ac*.categorías) $\Theta(|hija|)$
AgregarRápido(*estr_padre*.hijos, &(*estr_hija*)) $\Theta(1)$

Complejidad: $\Theta(|padre| + |hija|)$

$\Theta(|padre|) + \Theta(1) + \Theta(1) + \Theta(1) + \Theta(|hija| + copy(estr\_hija)) + \Theta(|hija|) + \Theta(1) =$
$\Theta(|padre|) + \Theta(|hija| + copy(estr\_hija)) + \Theta(|hija|) =$
$\Theta(|padre|) + \Theta(|hija|) + \Theta(copy(estr\_hija)) + \Theta(|hija|) =$
$\Theta(|padre|) + 2 * \Theta(|hija|) + \Theta(copy(estr\_hija)) =$
$\Theta(|padre|) + 2 * \Theta(|hija|) + \Theta(|hija|) =$
$\Theta(|padre|) + 3 * \Theta(|hija|) =$
$\Theta(|padre|) + \Theta(|hija|) =$
$\Theta(|padre| + |hija|)$.

iIdCategoríaPorNombre(**in** *c*: `categoria`, **in** *ac*: `estr_acat`) → *res* : `nat`
res ← Obtener(*c*, *ac*.categorías).id $\Theta(|c|)$

Complejidad: $\Theta(|c|)$

i#Categorías(**in** *ac*: `estr_acat`) → *res* : `nat`
res ← #Claves(*ac*.categorías) $\Theta(1)$

Complejidad: $\Theta(1)$

iCrearIt(**in** *padre*: `categoria`, **in** *ac*: `acat`) → *res* : `itConj(puntero(estr_cat))`
res ← CrearIt(Obtener(*padre*, *ac*.categorías).hijos) $\Theta(|padre|)$

Complejidad: $\Theta(|padre|)$

iCrearItRaíz(**in** *ac*: `acat`) → *res* : `itConj(puntero(estr_cat))`
res ← CrearIt(*ac*.raíz.hijos) $\Theta(1)$

Complejidad: $\Theta(1)$

iCrearItHijos(**in** *it*: `itConj(puntero(estr_cat))`) → *res* : `itConj(puntero(estr_cat))`
res ← CrearIt(Siguiente(it)→hijos) $\Theta(1)$

Complejidad: $\Theta(1)$

iHayMás?(**in** *it*: `itConj(puntero(estr_cat))`) → *res* : `bool`
res ← HaySiguiente(*it*) Θ(1)
Complejidad: Θ(1)

iCategoríaActual(**in** *it*: `itConj(puntero(estr_cat))`) → *res* : `categoria`
res ← Siguiente(*it*)→nombre Θ(1)
Complejidad: Θ(1)

iIdCategoríaActual(**in** *it*: `itConj(puntero(estr_cat))`) → *res* : `nat`
res ← Siguiente(*it*)→id Θ(1)
Complejidad: Θ(1)

iAvanzar(**in/out** *it*: `itConj(puntero(estr_cat))`)
Avanzar(*it*) Θ(1)
Complejidad: Θ(1)

2. Módulo LinkLinkIt

Interfaz

se explica con: LinkLinkIt, Iterador Unidireccional(Link).

géneros: `sistema`, `itlinks`.

Operaciones básicas del sistema

CrearSistema(**in** *ac*: `acat`) → *res* : `sistema`
Pre ≡ {true}
Post ≡ {*res* =$_{\text{obs}}$ iniciar(*ac*)}
Complejidad: Θ(#(categorías(*ac*)))
Descripción: crea un sistema cuyo árbol de categorías es *ac*.
Aliasing: *res* podrá invalidarse si se modifica *ac* luego de ejecutarse esta operación.

AgregarLink(**in** *l*: `link`, **in** *c*: `categoria`, **in/out** *s*: `sistema`)
Pre ≡ {$s =_{\text{obs}} s_0 \land \neg(l \in \text{links}(s)) \land$ está?(c, categorías(s))}
Post ≡ {$s =_{\text{obs}}$ nuevoLink(s_0, l, c)}
Complejidad: Θ($|l| + |c| + h$), donde h representa altura(categorías(s)).
Descripción: agrega al sistema el link l con categoría c.
Aliasing: el link l se agrega por copia.

AccederLink(**in** *l*: `link`, **in** *f*: `fecha`, **in/out** *s*: `sistema`)
Pre ≡ {$s =_{\text{obs}} s_0 \land l \in \text{links}(s) \land f \geq \text{fechaActual}(s)$}
Post ≡ {$s =_{\text{obs}}$ acceso(s_0, l, f)}
Complejidad: Θ($|l| + h$), donde h representa altura(categorías(s)).
Descripción: registra un acceso al link l en la fecha f.

#Links(**in** *c*: `categoría`, **in** *s*: `sistema`) → *res* : `nat`
Pre ≡ {está?(c, categorías(s))}

Post $\equiv \{res =_{\text{obs}} \text{cantLinks}(s, c)\}$
Complejidad: $\Theta(|c|)$
Descripción: devuelve la cantidad de links bajo la categoría c y todas sus subcategorías.

Operaciones de iterador de links

CrearIt(**in** c: `categoría`, **in** s: `sistema`) $\rightarrow$ res : `itlinks`
Pre $\equiv \{\text{está?}(c, \text{categorías}(s))\}$
Post $\equiv \{res =_{\text{obs}} \text{CrearItUni}(\text{tuplasLinks}(c, s)) \wedge \text{alias}(\text{iteraLinksCorrectos}(res, c, s))\}$
Complejidad: $\Theta(|c| + n^2)$, donde n representa long(linksOrdenadosPorAccesos(s, c)).
Descripción: devuelve un iterador unidireccional de los links de la categoría c y todas sus subcategorías ordenados de mayor a menor cantidad de accesos recientes.
Aliasing: el iterador podrá invalidarse si se agregan links a la categoría c o alguna subcategoría y/o si se registran accesos a links de dichas categorías.

HayMás?(**in** it: `itlinks`) $\rightarrow$ res : `bool`
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \text{HayMás?}(it)\}$
Complejidad: $\Theta(1)$
Descripción: devuelve **true** si y sólo si en el iterador todavía quedan elementos para avanzar.

LinkActual(**in** it: `itlinks`) $\rightarrow$ res : `link`
Pre $\equiv \{\text{HayMás?}(it)\}$
Post $\equiv \{res =_{\text{obs}} \Pi_1(\text{Actual}(it))\}$
Complejidad: $\Theta(1)$
Descripción: devuelve el link actual del iterador.
Aliasing: res no es modificable.

CategoríaLinkActual(**in** it: `itlinks`) $\rightarrow$ res : `categoria`
Pre $\equiv \{\text{HayMás?}(it)\}$
Post $\equiv \{res =_{\text{obs}} \Pi_2(\text{Actual}(it))\}$
Complejidad: $\Theta(1)$
Descripción: devuelve la categoría del link actual del iterador.
Aliasing: res no es modificable.

AccesosRecientesLinkActual(**in** it: `itlinks`) $\rightarrow$ res : `nat`
Pre $\equiv \{\text{HayMás?}(it)\}$
Post $\equiv \{res =_{\text{obs}} \Pi_3(\text{Actual}(it))\}$
Complejidad: $\Theta(1)$
Descripción: devuelve la cantidad de accesos del link actual del iterador durante los días de la intersección entre los tres días "recientes" del link l y los tres días "recientes" del link que tuvo último acceso entre los links de la categoría c con la que se creó este iterador, y los links de todas sus subcategorías.

Avanzar(**in/out** it: `itlinks`)
Pre $\equiv \{it =_{\text{obs}} it_0 \wedge \text{HayMás?}(it)\}$
Post $\equiv \{it =_{\text{obs}} \text{Avanzar}(it_0)\}$
Complejidad: $\Theta(1)$
Descripción: avanza el iterador a la posición siguiente.

Especificación de las operaciones auxiliares utilizadas en la interfaz

TAD Sistema Extendido

extiende LinkLinkIt

otras operaciones (no exportadas)

tuplasLinks : categoría c × lli s $\longrightarrow$ secu(tupla(link, categoría, nat)) {está?(c, categorías(s))}

linksATuplas : secu(link) sl × categoría c × lli s $\longrightarrow$ secu(tupla(link, categoría, nat)) {estánEnSistema?(sl, s) ∧ está?(c, categorías(s))}

estánEnSistema? : secu(link) × link $\longrightarrow$ bool

iteraLinksCorrectos : itUni(tupla(link, categoría, nat)) × categoría c × lli s $\longrightarrow$ bool

$\{\text{está?}(c, \text{categorías}(s))\}$

terminanIgual : secu(tupla(link, categoría, nat)) × secu(tupla(link, categoría, nat)) $\longrightarrow$ bool

axiomas $\forall c$: categoría, $\forall s$: lli, $\forall sl$: secu(link), $\forall it$: itUni(tupla(link, categoría, nat)), $\forall st, st'$: secu(tupla(link, categoría, nat))

tuplasLinks(c, s) $\equiv$ linksATuplas(linksOrdenadosPorAccesos(s, c), c, s)

linksATuplas(sl, c, s) $\equiv$ **if** vacía?(sl) **then**
<>
else
⟨prim(sl), categoríaLink(s, prim(sl)), accesosRecientes(s, c, prim(sl))⟩ • linksATuplas(fin(sl), c, s)
fi

estánEnSistema?(sl, s) $\equiv$ **if** vacía(sl) **then**
true
else
prim(sl) $\in$ links(s) $\wedge$ estánEnSistema?(fin(sl), s)
fi

iteraLinksCorrectos(it, c, s) $\equiv$ terminanIgual(Siguientes(it), tuplasLinks(c, s))

terminanIgual(st, st') $\equiv$ **if** vacía?(st) $\vee$ vacía?(st') **then**
true
else
ult(st) $=_{\text{obs}}$ ult(st') $\wedge$ terminanIgual(com(st), com(st'))
fi

Fin TAD

Representación

Representación del sistema

Representamos el sistema por medio de la tupla `estr_sistema`, en la que se redunda información sobre las categorías para poder cumplir con las complejidades pedidas. La tupla contiene:

- una referencia al árbol de categorías recibido al construir el sistema,
- un diccionario trie *links* de tuplas `estr_links`, que permite acceder a la información de cualquier link l en $\Theta(|l|)$,
- un arreglo *linksPorCatId* de tuplas `estr_linksPorCatId`, de las siguientes características:
  - tamaño igual a la cantidad de categorías en el sistema,
  - la tupla en la posición i se corresponde con la categoría de id $i + 1$,
  - cada tupla contiene el nombre de la categoría, el id del padre y una lista de links en ésa categoría y todas sus subcategorías, entre otras cosas;
- la fecha actual del sistema, que coincide con la fecha en la que se accedió a algún link del sistema por última vez, o vale cero si aún no se registraron accesos.

La tupla `estr_link` contiene el link representado, el id de la categoría a la que pertenece, la fecha en la que se accedió por última vez y un arreglo de tres naturales, que contiene la cantidad de accesos en la fecha de último acceso y en los dos días anteriores.

La tupla `estr_linksPorCatId` contiene la categoría representada, el id de su padre, una lista con los links que pertenecen a esa categoría y todas sus subcategorías, la fecha en la que se accedió a algún link de esa lista por última vez, y un booleano que indica si la lista está ordenada según $linksOrdenadosPorAccesos(s, c)$, donde c es la categoría representada por esta tupla.

Las estructuras elegidas nos permiten satisfacer la complejidad de la creación de un iterador de links según $linksOrdenadosPorAccesos(s, c)$ en $\Theta(|c| + n^2)$: primero consultamos el id de c en el árbol del sistema, lo cual tiene costo $\Theta(|c|)$, luego utilizamos el id como índice en el arreglo s.linksPorCatId para acceder a la lista de links de c y sus subcategorías con costo $\Theta(1)$; finalmente ordenamos el arreglo según $linksOrdenadosPorAcceso(s, c)$ con costo $\Theta(n^2)$, sólo si no estaba previamente ordenado. El costo total es $\Theta(|c| + n^2)$, o $\Theta(|c|)$ si ya se respetaba el orden.

Agregar un link l bajo una categoría c en un sistema s tiene costo $\Theta(|l| + |c| + h)$: primero agregamos el link al diccionario trie s.links con costo $\Theta(|l|)$, luego consultamos el id de la categoría c con costo $\Theta(|c|)$, luego usamos el id de la categoría como índice en el arreglo s.linksPorCatId para agregar el link a la lista de links de su categoría y obtener el id de la categoría padre en $\Theta(1)$, y finalmente utilizamos el id de la categoría padre como índice en el arreglo s.linksPorCatId para agregar el link a su lista de links, repitiendo el proceso hasta llegar a la categoría raíz, con costo

$\Theta(h)$. De esta manera logramos que cada categoría conozca sus links, y que todos los ancestros conozcan los links de sus categorías descendientes. Notar que cada vez que se agrega un link a alguna tupla `estr_linksPorCatId` se setea en falso el booleano que indica si se cumple el orden.

De manera similar, acceder un link se logra con costo $\Theta(|l| + h)$: primero consultamos la tupla `estr_link` en el diccionario trie *links* con costo $\Theta(|l|)$, y luego recorremos la jerarquía de estructuras `estr_linksPorCatId` para setear el booleano de orden a falso y actualizar la fecha de último acceso de cada categoría de ser necesario, con costo $\Theta(h)$.

`sistema` **se representa con** `estr_sistema`

donde `estr_sistema` es `tupla`(*categorías*: `acat`,
links: `dicctrie(estr_link)`,
linksPorCatId: `arreglo_dimensionable de estr_linksPorCatId`,
fechaActual: `fecha`)

donde `estr_link` es `tupla`(*l*: `link`,
cid: `nat`,
últimoAcceso: `fecha`,
as: `arreglo_estático[3] de fecha`)

donde `estr_linksPorCatId` es `tupla`(*cat*: `categoria`,
idPadre: `nat`,
links: `lista(puntero(estr_link))`,
últimoAcceso: `fecha`,
ordenado?: `bool`)

Invariante de representación:

1. Para todos los links en el diccionario de links:
   a) El nombre del link debe coincidir con la clave.
   b) El id de su categoría debe corresponderse con el id de una categoría en el árbol.
   c) La fecha del último acceso debe ser menor o igual a la fecha actual del sistema.
2. *linksPorCatId* debe tener tantos elementos como categorías hayan en el árbol.
3. Para cada elemento en *linksPorCatId*:
   a) La categoría debe estar en el árbol.
   b) El índice del elemento debe ser igual al id de la categoría en el árbol − 1.
   c) *idPadre* debe ser igual al id del padre de la categoría en el árbol, o cero si la categoría es la raíz.
   d) La lista de punteros de estr_link no puede tener elementos repetidos.
   e) La lista de punteros de estr_link debe contener punteros a las estructuras estr_link de todos los links de la categoría del elemento.
   f) Para cada elemento en la lista de punteros de estr_link:
      1) El elemento no puede ser un puntero nulo.
      2) El elemento tiene que apuntar a un link en el diccionario de links.
      3) El elemento tiene que estar en la lista de links del elemento en la posición *idPadre* − 1 del arreglo *linksPorCatId*, a menos que *idPadre* sea cero.
   g) La fecha del último acceso debe ser menor o igual a la fecha actual del sistema.
4. Al menos un link en el diccionario de links tiene como fecha de último acceso la fecha actual del sistema.

Rep : estr_sistema $\longrightarrow$ bool

Rep(s) $\equiv$ true $\Longleftrightarrow$
- ($\forall l$: link)(def?(l, s.links) $\Rightarrow_{\text{L}}$ (— 1.
  - $estr\_l$.l $=_{\text{obs}}$ l $\wedge$ — 1. a)
  - ($\exists c$: categoría)(está?(c, s.categorías) $\wedge_{\text{L}}$ $estr\_l$.cid $=_{\text{obs}}$ id(s.categorías. c)) $\wedge$ — 1. b)
  - $estr\_l$.últimoAcceso $\leq$ s.fechaActual — 1. c)

  )) $\wedge_{\text{L}}$
- tam(s.linksPorCatId) $=_{\text{obs}}$ #(categorias(s.categorías)) $\wedge$ — 2.
- ($\forall i$: nat)(i < tam(s.linksPorCatId) $\Rightarrow_{\text{L}}$ (— 3.
  - está?($estr\_c$.cat, s.categorías) $\wedge_{\text{L}}$ — 3. a)
  - i $=_{\text{obs}}$ id(s.categorías, $estr\_c$.cat) $-$ 1 $\wedge$ — 3. b)
  - $estr\_c$.idPadre $=_{\text{obs}}$ — 3. c)
    - (**if** $estr\_c$.cat $=_{\text{obs}}$ raíz(s.categorías)
      then 0
      else id(s.categorías, padre(s.categorías, $estr\_c$.cat))
      fi) $\wedge$
  - SinRepetidos($estr\_c$.links) $\wedge$ — 3. d)
  - ($\forall l$: link)(def?(l, s.links) $\Rightarrow_{\text{L}}$ (está?($estr\_l$, s.linksPorCatId[$estr\_l$.cid - 1].links))) $\wedge$ — 3. e)
  - ($\forall l$: puntero(estr_link))(está?(l, $estr\_c$.links) $\Rightarrow_{\text{L}}$ (— 3. f)
    - $\neg$(l $=_{\text{obs}}$ NULL) $\wedge_{\text{L}}$ — 3. f) 1)
    - def?(l→l, s.links) $\wedge_{\text{L}}$ l $=_{\text{obs}}$ &(obtener(l→l, s.links)) $\wedge$ — 3. f) 2)
    - $\neg$($estr\_c$.idPadre $=_{\text{obs}}$ 0) $\Rightarrow_{\text{L}}$ está?(l, s.linksPorCatId[$estr\_c$.idPadre $-$ 1].links) — 3. f) 3)

    )) $\wedge$
  - $estr\_c$.últimoAcceso $\leq$ s.fechaActual — 3. g)

  )) $\wedge$
- ($\exists l$: link)(def?(l, s.links) $\wedge_{\text{L}}$ obtener(l, s.links).últimoAcceso $=_{\text{obs}}$ s.fechaActual), — 4.

donde $estr\_l$ es obtener(l, s.links) y $estr\_c$ es s.linksPorCatId[i].

SinRepetidos : secu(puntero(estr_link)) $\longrightarrow$ bool
SinRepetidos(sp) $\equiv$ **if** vacía?(sp) **then** **true** **else** $\neg$(está?(prim(sp), fin(sp))) $\wedge$ SinRepetidos(fin(sp)) **fi**

Función de abstracción:

1. El árbol de categorías del sistema debe ser igual al de la estructura.
2. El conjunto de links del sistema debe ser igual al conjunto de clave del diccionario de links de la estructura.
3. La fecha actual del sistema debe ser igual a la fecha actual de la estructura.
4. Para cada link en el sistema:
   - *a*) La categoría del link en el sistema debe ser igual a la categoría del link en el diccionario de la estructura.
   - *b*) La fecha de último acceso del link en el sistema debe ser igual a la fecha de último acceso del link en el diccionario de la estructura.
   - *c*) Los accesos recientes por día del link en el sistema deben coincidir con los elementos del arreglo de accesos recientes del link en el diccionario de la estructura.

Abs : estr_sistema e $\longrightarrow$ lli — {Rep(e)}
Abs(e) $=_{\text{obs}}$ s: lli | — 1.
- categorias(s) $=_{\text{obs}}$ e.categorías $\wedge$ — 1.
- links(s) $=_{\text{obs}}$ claves(e.links) $\wedge$ — 2.
- fechaActual(s) $=_{\text{obs}}$ e.fechaActual $\wedge$ — 3.
- ($\forall l$: link)(l $\in$ links(s) $\Rightarrow_{\text{L}}$ (— 4.
  - categoriaLink(s, l) $=_{\text{obs}}$ s.linksPorCatId[obtener(l, e.links).cid $-$ 1].cat $\wedge$ — 4. a)
  - fechaUltimoAcceso(s, l) $=_{\text{obs}}$ obtener(l, e.links).últimoAcceso $\wedge$ — 4. b)
  - MismosAccesos(l, s, e) — 4. c)

  ))

MismosAccesos : link l × lli s × estr_sistema e $\longrightarrow$ bool — {l $\in$ links(s) $\wedge$ def?(l, e.links)}
MismosAccesos(l, s, e) $\equiv$ ($\forall n$: nat)(($n \leq 2 \wedge$ fechaUltimoAcceso(s, l) $\geq n$) $\Rightarrow_{\text{L}}$
accesosRecientesDia(s, l, fechaUltimoAcceso(s, l) $-$ n) $=_{\text{obs}}$
obtener(l, e.links).as[n])

Representación del iterador

El iterador de links permite recorrer la secuencia de links según $linksOrdenadosPorAccesos(s, c)$, dado un sistema s y una categoría c. Lo representamos con una tupla que contiene una referencia al sistema, el id de la categoría c y un iterador de la lista de links de la estructura `estr_linksPorCatId` correspondiente a c, la cual se asumirá ordenada según $linksOrdenadosPorAccesos(s, c)$.

El id de la categoría nos permitirá proyectar el nombre de la misma y la cantidad de accesos recientes del link actual, utilizando el id como índice en el arreglo s.linksPorCatId para obtener en $\Theta(1)$ el nombre y la fecha en la que se accedió por última vez algún link de la categoría, siendo este último un dato necesario para calcular la cantidad de accesos recientes del link actual.

`itlinks` se representa con `estr_iter`

donde `estr_iter` es `tupla`(s: `estr_sistema`, cid: `nat`, it: `itLista(puntero(estr_link))`)

Invariante de representación:

1. s representa un sistema válido.
2. cid debe ser el id de una categoría en el sistema.
3. La secuencia subyacente del iterador debe ser la secuencia de links en la categoría con id cid y sus subcategorías.

Rep : estr_iter $\longrightarrow$ bool
Rep(e) $\equiv$ true $\Longleftrightarrow$
- Rep(e.s) $\wedge$ — 1.
- $1 \leq e.\text{cid} \wedge e.\text{cid} \leq \#(\text{categorias}(e.\text{s.categorías})) \wedge$ — 2.
- SecuSuby(e.it) $=_{\text{obs}}$ e.s.linksPorCatId[e.cid $-$ 1].links — 3.

Función de abstracción:

1. Los elementos siguientes del iterador deben ser las tuplas (link, categoría, nat) que se correspondan con los elementos siguientes del iterador de la estructura.

Abs : estr_iter e $\longrightarrow$ itUni(tupla(link, categoría, nat)) {Rep(e)}
Abs(e) $=_{\text{obs}}$ it: itUni(tupla(link, categoría, nat)) | Siguientes(it) $=_{\text{obs}}$ SecuDeTuplas(— 1.
Abs(e.s),
e.s.linksPorCatId[e.cid $-$ 1].cat,
SecuDeLinks(Siguientes(e.it)))

SecuDeTuplas : lli s $\times$ categoria c $\times$ secu(link) sl $\longrightarrow$ secu(tupla(link, categoria, nat))
{está?(c, categorias(s)) $\wedge$ EstánEnConj?(sl, links(s))}
SecuDeTuplas(s, c, sl) $\equiv$ **if** vacía?(sl) **then**
<>
else
⟨prim(sl),
categoriaLink(s, prim(sl)),
accesosRecientes(s, c, prim(sl))⟩ • SecuDeTuplas(s, c, fin(sl))
fi

EstánEnConj? : secu(link) $\times$ conj(link) $\longrightarrow$ bool
EstánEnConj?(sl, cl) $\equiv$ **if** vacía?(sl) **then** **true** **else** prim(sl) $\in$ cl $\wedge$ EstánEnConj?(fin(sl), cl) **fi**

SecuDeLinks : secu(puntero(estr_link)) sp $\longrightarrow$ secu(link) {¬(está?(NULL, sp))}
SecuDeLinks(sp) $\equiv$ **if** vacía?(sp) **then** <> **else** prim(sp)→l • SecuDeLinks(fin(sp)) **fi**

Algoritmos

```
iCrearSistema(in ac: estr_acat) → res : estr_sistema
    res ← ⟨categorías: ac,                                          Θ(1)
           links: CrearDiccionario(),                               Θ(1)
           linksPorCatId: CrearArreglo(#Categorías(ac)),            Θ(#categorias(ac))
           fechaActual: 0⟩                                          Θ(1)
```

res.linksPorCatId[0] $\leftarrow$ ⟨`cat`: NombreCategoríaRaíz(ac), $\Theta(1)$
`idPadre`: 0, $\Theta(1)$
`links`: Vacía(), $\Theta(1)$
`últimoAcceso`: 0, $\Theta(1)$
`ordenado?`: **false**⟩ $\Theta(1)$
var it: `itcats` $\leftarrow$ CrearItRaíz(ac) $\Theta(1)$
iAgregarALinksPorCatId(it, 1, res) $\Theta(\sum_{i=1}^{i=long(Siguientes(it))} \#subcategorias(Siguientes(it)_i))$

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

$9 * \Theta(1) + \Theta(\#categorias(ac)) + \Theta(\sum_{i=1}^{i=long(Siguientes(it))} \#subcategorias(Siguientes(it)_i)) =$
$\Theta(\#categorias(ac)) + \Theta(\sum_{i=1}^{i=long(Siguientes(it))} \#subcategorias(Siguientes(it)_i)) =$
$\Theta(\#categorias(ac)) + \Theta(\#categorias(ac) - 1) =$
$\Theta(\#categorias(ac))$.

Notar que $\sum_{i=1}^{i=long(Siguientes(it))} \#subcategorias(Siguientes(it)_i) = \#categorias(ac) - 1$.

iAgregarALinksPorCatId(**in/out** it: `itcats`, **in** $idPadre$: `nat`, **in/out** s: `estr_sistema`)
while HayMás?(it) **do** $\Theta(1)$
s.linksPorCatId[IdCategoríaActual(it) - 1] $\leftarrow$ ⟨`cat`: CategoríaActual(it), $\Theta(1)$
`idPadre`: $idPadre$, $\Theta(1)$
`links`: Vacía(), $\Theta(1)$
`últimoAcceso`: 0, $\Theta(1)$
`ordenado?`: **false**⟩ $\Theta(1)$
var it': `itcats` $\leftarrow$ CrearItHijos(it) $\Theta(1)$
iAgregarALinksPorCatId($\Theta(\sum_{i=1}^{i=long(Siguientes(it'))} \#subcategorias(Siguientes(it')_i))$
$ithijos$, $\Theta(1)$
IdCategoríaActual(it), $\Theta(1)$
s) $\Theta(1)$
Avanzar(it) $\Theta(1)$
end while

Complejidad: $\Theta(\sum_{i=1}^{i=long(Siguientes(it))} \#subcategorias(Siguientes(it)_i))$, donde:
- $\#subcategorias(x)$ es la cantidad de nodos del subárbol de categorías en el cual x es el nodo raíz, y
- $Siguientes(it)_i$ es el i-ésimo elemento de la secuencia de elementos siguientes del iterador it.

El algoritmo recibe un iterador de categorías, y para cada una de ellas guarda información en la posición correspondiente en el arreglo s.linksPorCatId, y luego crea un iterador de sus categorías hijas que utiliza para llamarse recursivamente.

iAgregarLink(**in** l: `link`, **in** c: `categoria`, **in/out** s: `estr_sistema`)
var $estr\_l$: `estr_link` $\leftarrow$ ⟨`l`: l, $\Theta(1)$
`cid`: IdCategoríaPorNombre(c, s.categorías), $\Theta(|c|)$
`últimoAcceso`: s.fechaActual, $\Theta(1)$
`as`: iCrearArregloDe3Nats(0, 0, 0)⟩ $\Theta(1)$
Definir(l, $estr\_l$, s.links) $\Theta(|l| + copy(estr\_l))$
$estr\_l \leftarrow$ Obtener(l, s.links) $\Theta(|l|)$

var cid: `nat` $\leftarrow$ $estr\_l$.cid $\Theta(1)$
while $\neg(cid =_{\text{obs}} 0)$ **do** $\Theta(1)$
AgregarAtrás(s.linksPorCatId[cid - 1].links, &($estr\_l$)) $\Theta(1)$
s.linksPorCatId[cid - 1].últimoAcceso $\leftarrow$ s.fechaActual $\Theta(1)$
s.linksPorCatId[cid - 1].ordenado? $\leftarrow$ **false** $\Theta(1)$
$cid \leftarrow$ s.linksPorCatId[cid - 1].idPadre $\Theta(1)$
end while

Complejidad: $\Theta(|c| + |l| + h)$, donde h es la altura del árbol de categorías.

$\Theta(1) + \Theta(|c|) + \Theta(1) + \Theta(1) + \Theta(|l| + copy(estr\_l)) + \Theta(|l|) + \Theta(1)$ + ciclo =
$4 * \Theta(1) + \Theta(|c|) + \Theta(|l| + copy(estr\_l))$ + ciclo =
$\Theta(|c|) + \Theta(|l| + copy(estr\_l))$ + ciclo =
$\Theta(|c|) + \Theta(|l|) + \Theta(copy(estr\_l))$ + ciclo =
$\Theta(|c|) + \Theta(|l|) + \Theta(|c|)$ + ciclo =
$2 * \Theta(|c|) + \Theta(|l|)$ + ciclo =
$\Theta(|c|) + \Theta(|l|)$ + ciclo =
$\Theta(|c|) + \Theta(|l|) + h * (\Theta(1) + \Theta(1) + \Theta(1) + \Theta(1) + \Theta(1)) + \Theta(1) =$
$\Theta(|c|) + \Theta(|l|) + h * (5 * \Theta(1)) + \Theta(1) =$
$\Theta(|c|) + \Theta(|l|) + h * \Theta(1) + \Theta(1) =$
$\Theta(|c|) + \Theta(|l|) + h * \Theta(1) =$
$\Theta(|c|) + \Theta(|l|) + \Theta(h) =$
$\Theta(|c| + |l| + h)$.

---

iAccederLink(**in** l : `link`, **in** f : `fecha`, **in/out** s : `estr_sistema`)

var $estr\_l$: `estr_link` ← Obtener(l, s.links)	$\Theta(\|l\|)$
if $f =_{obs} estr\_l$.últimoAcceso **then**	$\Theta(1)$
$estr\_l$.as[0] ← $estr\_l$.as[0] + 1	$\Theta(1)$
else if $f =_{obs} estr\_l$.últimoAcceso + 1 **then**	$\Theta(1)$
$estr\_l$.as ← iCrearArregloDe3Nats(1, $estr\_l$.as[0], $estr\_l$.as[1])	$\Theta(1)$
else if $f =_{obs} estr\_l$.últimoAcceso + 2 **then**	$\Theta(1)$
$estr\_l$.as ← iCrearArregloDe3Nats(1, 0, $estr\_l$.as[0])	$\Theta(1)$
else	
$estr\_l$.as ← iCrearArregloDe3Nats(1, 0, 0)	$\Theta(1)$
end if	
s.fechaActual ← f	$\Theta(1)$
var cid: `nat` ← $estr\_l$.cid	$\Theta(1)$
while $\neg(cid =_{obs} 0)$ **do**	$\Theta(1)$
s.linksPorCatId[cid - 1].últimoAcceso ← s.fechaActual	$\Theta(1)$
s.linksPorCatId[cid - 1].ordenado? ← **false**	$\Theta(1)$
$idPadre$ ← s.linksPorCatId[cid - 1].idPadre	$\Theta(1)$
end while	

Complejidad: $\Theta(|l| + h)$, donde h es la altura del árbol de categorías.

$\Theta(|l|)$ + condicionales + $\Theta(1) + \Theta(1)$ + ciclo =
$\Theta(|l|)$ + condicionales + ciclo =
$\Theta(|l|) + \Theta(1) + (\Theta(1)$ ó $(\Theta(1) + (\Theta(1)$ ó $(\Theta(1) + (\Theta(1)$ ó $\Theta(1))))))$ + ciclo =
$\Theta(|l|) + \Theta(1)$ + ciclo =
$\Theta(|l|)$ + ciclo =
$\Theta(|l|) + h * (\Theta(1) + \Theta(1) + \Theta(1) + \Theta(1)) + \Theta(1) =$
$\Theta(|l|) + h * (4 * \Theta(1)) + \Theta(1) =$
$\Theta(|l|) + h * \Theta(1) + \Theta(1) =$
$\Theta(|l|) + h * \Theta(1) =$
$\Theta(|l|) + \Theta(h) =$
$\Theta(|l| + h)$.

---

iCrearArregloDe3Nats(**in** a: `nat`, **in** b: `nat`, **in** c: `nat`) → res : `arreglo_estático[3] de nat`

res ← CrearArreglo()	$\Theta(1)$
res[0] ← a	$\Theta(1)$

res[1] ← b $\Theta(1)$
res[2] ← c $\Theta(1)$

Complejidad: $\Theta(1)$

---

iLinks(**in** c: `categoría`, **in** s: `estr_sistema`) → res : `nat`
res ← Longitud(s.linksPorCatId[IdCategoríaPorNombre(c, s.categorías) - 1].links) $\Theta(|c|)$

Complejidad: $\Theta(|c|)$

---

iCrearIt(**in** c: `categoría`, **in** s: `estr_sistema`) → res : `estr_iter`
var $estr\_c$: `estr_linksPorCatId` ← s.linksPorCatId[$\Theta(1)$
IdCategoríaPorNombre(c, s.categorías) - 1] $\Theta(|c|)$
if ¬($estr\_c$.ordenado?) **then** $\Theta(1)$
iOrdenarLinks($estr\_c$) $\Theta(|estr\_c.\text{links}|^2)$
end if
res ← ⟨`s`: s, $\Theta(1)$
`cid`: IdCategoríaPorNombre(c, s.categorías), $\Theta(|c|)$
`it`: CrearIt($estr\_c$.links)⟩ $\Theta(1)$

Complejidad: $\Theta(|c| + n^2)$ ó $\Theta(|c|)$, donde n es la cantidad de links de c y todas sus subcategorías.

$\Theta(1) + \Theta(|c|) + \Theta(1) + (\Theta(|estr\_c.\text{links}|^2)$ ó $0) + \Theta(1) + \Theta(|c|) + \Theta(1) =$
$4 * \Theta(1) + \Theta(|c|) + (\Theta(|estr\_c.\text{links}|^2)$ ó $0) + \Theta(|c|) =$
$\Theta(|c|) + (\Theta(|estr\_c.\text{links}|^2)$ ó $0) + \Theta(|c|) =$
$2 * \Theta(|c|) + (\Theta(|estr\_c.\text{links}|^2)$ ó $0) =$
$\Theta(|c|) + (\Theta(|estr\_c.\text{links}|^2)$ ó $0) =$
$\Theta(|c|) + (\Theta(n^2)$ ó $0) =$
$(\Theta(|c|) + \Theta(n^2))$ ó $\Theta(|c|) =$
$\Theta(|c| + n^2)$ ó $\Theta(|c|)$.

Notar que $n =$ estr_c.*links*.

---

iOrdenarLinks(**in** $estr\_c$: `estr_linksPorCatId`)
var it: `itLista(puntero(estr_link))` ← CrearIt($estr\_c$.links) $\Theta(1)$
while HaySiguiente(it) **do** $\Theta(1)$
var it': `itLista(puntero(estr_link))` ← Copiar(it) $\Theta(1)$
var $itMax$: `itLista(puntero(estr_link))` ← Copiar(it') $\Theta(1)$
while HaySiguiente(it') **do** $\Theta(1)$
var $arActual$: `nat` ← iAccesosRecientes(*Siguiente(it'), $estr\_c$) $\Theta(1)$
var $arMax$: `nat` ← iAccesosRecientes(*Siguiente($itMax$), $estr\_c$) $\Theta(1)$
if $arActual > arMax$ **then** $\Theta(1)$
$itMax$ ← Copiar(it') $\Theta(1)$
end if
Avanzar(it') $\Theta(1)$
end while
iIntercamabiar(it, $itMax$) $\Theta(1)$
Avanzar(it) $\Theta(1)$
end while
$estr\_c$.ordenado? ← **true** $\Theta(1)$

Complejidad: $\Theta(n^2)$, donde n es $|estr\_c.\text{links}|$.

$\Theta(1) + (\sum_{i=1}^{i=n} \Theta(1) + \sum_{j=i}^{j=n} \Theta(1)) =$
$\sum_{i=1}^{i=n} \sum_{j=i}^{j=n} \Theta(1) =$
$\Theta(1) * (\sum_{i=1}^{i=n} \sum_{j=i}^{j=n} 1) =$
$\Theta(1) * (\sum_{i=1}^{i=n} n - i) =$
$\Theta(1) * (n^2 + (\sum_{i=1}^{i=n} -i)) =$
$\Theta(1) * (n^2 - (\sum_{i=1}^{i=n} i)) =$
$\Theta(1) * (n^2 - \frac{1}{2} * (n * (n-1))) =$
$\Theta(1) * (n^2 - \frac{1}{2} * (n^2 - n)) =$
$\Theta(n^2 - \frac{1}{2} * (n^2 - n)) =$
$\Theta(n^2 - \frac{1}{2} * n^2 - \frac{1}{2} * n)) =$
$\Theta(n^2) - \frac{1}{2} * \Theta(n^2) - \frac{1}{2} * \Theta(n) =$
$\frac{1}{2} * \Theta(n^2) - \frac{1}{2} * \Theta(n) =$
$\Theta(n^2)$.

iAccesosRecientes(**in** *estr_l*: `estr_link`, **in** *estr_c*: `estr_linksPorCatId`) → *res* : `nat`

if *estr_c*.últimoAcceso $=_{\text{obs}}$ *estr_l*.últimoAcceso **then**	$\Theta(1)$
res ← *estr_l*.as[0] + *estr_l*.as[1] + *estr_l*.as[2]	$\Theta(1)$
else if *estr_c*.últimoAcceso $=_{\text{obs}}$ *estr_l*.últimoAcceso + 1 **then**	$\Theta(1)$
res ← *estr_l*.as[0] + *estr_l*.as[1]	$\Theta(1)$
else if *estr_c*.últimoAcceso $=_{\text{obs}}$ *estr_l*.últimoAcceso + 2 **then**	$\Theta(1)$
res ← *estr_l*.as[0]	$\Theta(1)$
else	
res ← 0	$\Theta(1)$
end if	

Complejidad: $\Theta(1)$

iIntercambiar(**in/out** *it*: `itLista(puntero(estr_link))`, **in/out** *it'*: `itLista(puntero(estr_link))`)

var *ptr*: `puntero(estr_link)` ← Siguiente(*it'*)	$\Theta(1)$
EliminarSiguiente(*it'*)	$\Theta(1)$
AgregarComoSiguiente(*it'*, Siguiente(*it*))	$\Theta(copy(Siguiente(it)))$
EliminarSiguiente(*it*)	$\Theta(1)$
AgregarComoSiguiente(*it*, *ptr*)	$\Theta(copy(ptr))$

Complejidad: $\Theta(1)$

$\Theta(1) + \Theta(1) + \Theta(copy(Siguiente(it))) + \Theta(1) + \Theta(copy(ptr)) =$
$3 * \Theta(1) + \Theta(copy(Siguiente(it))) + \Theta(copy(ptr)) =$
$3 * \Theta(1) + \Theta(1) + \Theta(copy(ptr)) =$
$3 * \Theta(1) + \Theta(1) + \Theta(1) =$
$5 * \Theta(1) =$
$\Theta(1)$.

iHayMás?(**in** *it*: `estr_iter`) → *res* : `bool`

res ← HaySiguiente(*it*.it)	$\Theta(1)$

Complejidad: $\Theta(1)$

iLinkActual(**in** *it* : `estr_iter`) → *res* : `link`
res ← Siguiente(*it*.it)→l Θ(1)
Complejidad: Θ(1)

iCategoríaLinkActual(**in** *it* : `estr_iter`) → *res* : `categoria`
res ← *it*.s.linksPorCatId[Siguiente(*it*.it)→cid - 1].cat Θ(1)
Complejidad: Θ(1)

iAccesosRecientesLinkActual(**in** *it* : `estr_iter`) → *res* : `nat`
res ← iAccesosRecientes(*Siguiente(*it*.iter), *it*.s.linksPorCatId[*it*.cid - 1]) Θ(1)
Complejidad: Θ(1)

iAvanzar(**in/out** *it* : `estr_iter`)
Avanzar(*it*.it) Θ(1)
Complejidad: Θ(1)

3. Módulo Diccionario Trie(α)

Interfaz

parámetros formales
géneros α
función Copiar(**in** *a*: α) → *res* : α
Pre ≡ {true}
Post ≡ {*res* =$_{\text{obs}}$ *a*}
Complejidad: Θ(*copy*(*a*))
Descripción: función de copia de α's

se explica con: Diccionario(String, α).

géneros: `dicctrie`(α).

Operaciones básicas de diccionario trie

CrearDiccionario() → *res* : `dicctrie`(α)
Pre ≡ {true}
Post ≡ {*res* =$_{\text{obs}}$ vacío()}
Complejidad: Θ(1)
Descripción: crea un nuevo diccionario vacío.

Definir(**in** *c*: `string`, **in** *s*: α, **in/out** *d*: `dicctrie`(α))
Pre ≡ {*d* =$_{\text{obs}}$ d_0}
Post ≡ {*d* =$_{\text{obs}}$ definir(d_0, *c*, *s*)}
Complejidad: Θ(|*c*| + *copy*(*s*))
Descripción: define la clave *c* con el significado *s* en el diccionario.
Aliasing: el significado *s* se agrega por copia.

Obtener(**in** *c*: `string`, **in** *d*: `dicctrie`(α)) → *res* : α
Pre ≡ {def?(*c*, *d*)}

Post $\equiv \{res =_{\text{obs}} \text{obtener}(c, d)\}$
Complejidad: $\Theta(|c|)$
Descripción: devuelve el significado de la clave c en d.
Aliasing: res es modificable si y sólo si d es modificable.

#CLAVES(**in** d: `dicctrie`(α)) $\rightarrow$ res : `nat`
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \#(\text{claves}(d))\}$
Complejidad: $\Theta(1)$
Descripción: devuelve la cantidad de claves del diccionario.

Representación

Representamos cada nodo del árbol con una tupla que contiene un puntero a su significado, que podrá ser NULL si no hay un significado asociado a ese nodo, y un arreglo de 256 punteros a nodos hijos, en el que elemento de índice i representa al nodo hijo correspondiente al Char de valor $ord^{-1}(i)$, el cual será NULL si no se definió ninguna clave que comience con la traza de ese nodo seguida del Char $ord^{-1}(i)$.

El diccionario se representa con una tupla que mantiene una referencia al nodo raíz, y un natural que cuenta la cantidad total de claves definidas en el diccionario.

`dicctrie`(α) **se representa con** `estr_dicctrie`

donde `estr_dicctrie` es `tupla`(*raíz*: `estr_nodo`,
#Claves: `nat`)
donde `estr_nodo` es `tupla`(*significado*: `puntero`(α),
hijos: `arreglo_estático[256] de puntero(estr_nodo)`)

Invariante de representación:

1. El árbol no debe tener ciclos.
2. El árbol no debe tener nodos repetidos.
3. Los nodos terminales tienen significado no nulo.
4. La cantidad de claves debe coincidir con la cantidad de significados.

$$\begin{aligned}
&\text{Rep} : \text{estr_dicctrie} \longrightarrow \text{bool} \\
&\text{Rep}(e) \equiv \text{true} \iff \\
&\quad \text{NoHayCiclos}(e.\text{raíz}, \emptyset) \wedge_{\text{L}} && 1. \\
&\quad \text{SinRepetidos}(\text{Aplanar}(e.\text{raíz})) \wedge && 2. \\
&\quad \text{NodoTerminalTieneSignificado}(e.\text{raíz}) \wedge && 3. \\
&\quad e.\#\text{Claves} =_{\text{obs}} \text{ContarSignificados}(e.\text{raíz}) && 4.
\end{aligned}$$

$$\begin{aligned}
&\text{NoHayCiclos} : \text{estr_nodo} \times \text{conj}(\text{estr_nodo}) \longrightarrow \text{bool} \\
&\text{NoHayCiclos}(e, \mathit{ancestros}) \equiv \neg(e \in \mathit{ancestros}) \wedge_{\text{L}} \text{NoHayCiclosEnLosHijos}(e, \text{Ag}(e, \mathit{ancestros}), 0)
\end{aligned}$$

$$\begin{aligned}
&\text{NoHayCiclosEnLosHijos} : \text{estr_nodo} \times \text{conj}(\text{estr_nodo}) \times \text{nat} \longrightarrow \text{bool} \\
&\text{NoHayCiclosEnLosHijos}(e, \mathit{ancestros}, n) \equiv (\neg(e.\text{hijos}[n] =_{\text{obs}} \text{NULL}) \Rightarrow_{\text{L}} \\
&\qquad \text{NoHayCiclos}(*(e.\text{hijos}[n]), \mathit{ancestros})) \wedge_{\text{L}} \\
&\qquad (n < 255 \Rightarrow_{\text{L}} \text{NoHayCiclosEnLosHijos}(e, \mathit{ancestros}, n + 1))
\end{aligned}$$

$$\begin{aligned}
&\text{Aplanar} : \text{estr_nodo } e \longrightarrow \text{secu}(\text{estr_nodo}) && \{\text{NoHayCiclos}(e, \emptyset)\} \\
&\text{Aplanar}(e) \equiv e \bullet <> \ \& \ \text{AplanarHijos}(e, 0)
\end{aligned}$$

$$\begin{aligned}
&\text{AplanarHijos} : \text{estr_nodo } e \times \text{nat} \longrightarrow \text{secu}(\text{estr_nodo}) && \{\text{NoHayCiclos}(e, \emptyset)\} \\
&\text{AplanarHijos}(e, n) \equiv (\textbf{if } \text{e.hijos}[n] =_{\text{obs}} \text{NULL } \textbf{then } <> \textbf{ else } \text{Aplanar}(*(\text{e.hijos}[n])) \textbf{ fi}) \ \& \\
&\qquad (\textbf{if } n < 255 \textbf{ then } \text{AplanarHijos}(e, n + 1) \textbf{ else } <> \textbf{ fi})
\end{aligned}$$

$$\begin{aligned}
&\text{SinRepetidos} : \text{secu}(\text{estr_nodo}) \longrightarrow \text{bool} \\
&\text{SinRepetidos}(sn) \equiv \textbf{if } \text{vacía?}(sn) \textbf{ then } \textbf{true } \textbf{else } \neg(\text{está?}(\text{prim}(sn), \text{fin}(sn))) \wedge \text{SinRepetidos}(\text{fin}(sn)) \textbf{ fi}
\end{aligned}$$

$$\begin{aligned}
&\text{NodoTerminalTieneSignificado} : \text{estr_nodo } e \longrightarrow \text{bool} && \{\text{NoHayCiclos}(e, \emptyset)\}
\end{aligned}$$

NodoTerminalTieneSignificado(e) $\equiv$ **if** EsTerminal?(e, 0) **then**
¬(significado(e) $=_{\text{obs}}$ NULL)
else
NodoHijoTerminalTieneSignificado(e, 0)
fi

EsTerminal? : estr_nodo $e \times$ nat $\longrightarrow$ bool
EsTerminal?(e, n) $\equiv$ e.hijos[n] $=_{\text{obs}}$ NULL $\wedge_{\text{L}}$ ($n < 255 \Rightarrow_{\text{L}}$ EsTerminal?(e, $n + 1$))

NodoHijoTerminalTieneSignificado : estr_nodo $e \times$ nat $\longrightarrow$ bool {NoHayCiclos(e, $\emptyset$)}
NodoHijoTerminalTieneSignificado(e, n) $\equiv$ (¬(e.hijos[n] $=_{\text{obs}}$ NULL) $\Rightarrow_{\text{L}}$
NodoTerminalTieneSignificado(*(e.hijos[n]))) $\wedge_{\text{L}}$
($n < 255 \Rightarrow_{\text{L}}$ NodoHijoTerminalTieneSignificado(e, $n + 1$))

ContarSignificados : estr_nodo $\longrightarrow$ nat
ContarSignificados(e) $\equiv$ (**if** e.significado $=_{\text{obs}}$ NULL **then** 0 **else** 1 **fi**) + ContarSignificadosDeLosHijos(e, 0)

ContarSignificadosDeLosHijos : estr_nodo $\times$ nat $\longrightarrow$ nat
ContarSignificadosDeLosHijos(e, n) $\equiv$ (**if** e.hijos[n] $=_{\text{obs}}$ NULL **then**
0
else
ContarSignificados(*(e.hijos[n]))
fi) +
(**if** $n < 255$ **then**
ContarSignificadosDeLosHijos(e, $n + 1$)
else
0
fi)

Función de abstracción:

1. El diccionario tiene la misma cantidad de claves que la estructura.
2. Para cada clave del diccionario:
   a) La clave está definida en la estructura.
   b) El significado de la clave en el diccionario es el mismo que en la estructura.

Abs : estr_dicctrie e $\longrightarrow$ dicc(string, α) {Rep(e)}
Abs(e) $=_{\text{obs}}$ d: dicc(string, α) | #(claves(d)) $=_{\text{obs}}$ e.#Claves $\wedge_{\text{L}}$ 1.
(∀c: string)(def?(c, d) $\Rightarrow_{\text{L}}$ (2.
¬(ObtenerDeLaEstructura(c, e.raíz) $=_{\text{obs}}$ NULL) $\wedge_{\text{L}}$ 2. a)
obtener(c, d) $=_{\text{obs}}$ *(ObtenerDeLaEstructura(c, e.raíz)) 2. b)
))

ObtenerDeLaEstructura : string $\times$ estr_nodo $\longrightarrow$ puntero(α)
ObtenerDeLaEstructura(c, n) $\equiv$ **if** vacía?(c) **then**
n.significado
else
if n.hijos[Ord(prim(c))] $=_{\text{obs}}$ NULL **then**
NULL
else
ObtenerDeLaEstructura(fin(c), n.hijos[Ord(prim(c))])
fi
fi

Algoritmos

iCrearDiccionario() → *res* : `estr_dicctrie`
res ← ⟨`raíz`: iNuevoNodo(), Θ(1)
`#Claves`: 0⟩ Θ(1)

Complejidad: Θ(1)

$\Theta(1) + \Theta(1) =$
$2 * \Theta(1) =$
$\Theta(1)$.

iNuevoNodo() $\rightarrow$ *res* : `estr_nodo`
res $\leftarrow$ ⟨`significado`: NULL, $\Theta(1)$
`hijos`: CrearArreglo()⟩ $\Theta(256)$
for var *i*: `nat` $\leftarrow$ 0 to 255 **do** $\Theta(1)$
res.hijos[*i*] $\leftarrow$ NULL $\Theta(1)$
end for

Complejidad: $\Theta(1)$

$\Theta(1) + 256 * \Theta(1) + 256 * (\Theta(1) + \Theta(1)) =$
$\Theta(1) + 256 * \Theta(1) + 256 * \Theta(1) + 256 * \Theta(1) =$
$769 * \Theta(1) =$
$\Theta(1)$.

iDefinir(**in** *c*: `string`, **in** *s*: α, **in/out** *d*: `estr_dicctrie`)
var *actual*: `puntero(estr_nodo)` $\leftarrow$ &(*d*.raíz) $\Theta(1)$
for var *i*: `nat` $\leftarrow$ 0 to Longitud(*c*) **do** $\Theta(1)$
if *actual*$\rightarrow$hijos[Ord(*c*[*i*])] $=_{\text{obs}}$ NULL **then** $\Theta(1)$
actual$\rightarrow$hijos[Ord(*c*[*i*])] $\leftarrow$ &(iNuevoNodo()) $\Theta(1)$
end if
actual $\leftarrow$ *actual*$\rightarrow$hijos[Ord(*c*[*i*])] $\Theta(1)$
end for
actual$\rightarrow$significado $\leftarrow$ &(Copiar(*s*)) $\Theta(\text{copy}(s))$
d.#Claves $\leftarrow$ *d*.#Claves + 1 $\Theta(1)$

Complejidad: $\Theta(|c| + copy(s))$

$\Theta(1) + |c| * (\Theta(1) + \Theta(1) + (\Theta(1)$ ó $0) + \Theta(1)) + \Theta(copy(s)) + \Theta(1) =$
$\Theta(1) + |c| * \Theta(1) + \Theta(copy(s)) + \Theta(1) =$
$|c| * \Theta(1) + \Theta(copy(s)) =$
$\Theta(|c|) + \Theta(copy(s)) =$
$\Theta(|c| + copy(s))$.

iObtener(**in** *c*: `string`, **in** *d*: `estr_dicctrie`) $\rightarrow$ *res* : $\Theta(|c|)$
var *actual*: `puntero(estr_nodo)` $\leftarrow$ &(*d*.raíz) $\Theta(1)$
for var *i*: `nat` $\leftarrow$ 0 to Longitud(*c*) **do** $\Theta(1)$
actual $\leftarrow$ *actual*$\rightarrow$hijos[Ord(*c*[*i*])] $\Theta(1)$
end for
res $\leftarrow$ *(*actual*$\rightarrow$*significado*) $\Theta(1)$

Complejidad: $\Theta(|c|)$

$\Theta(1) + |c| * (\Theta(1) + \Theta(1)) + \Theta(1) =$
$|c| * \Theta(1) =$
$\Theta(|c|)$.

I#Claves(**in** d: `estr_dicctrie`) $\rightarrow$ res : `nat`
$res \leftarrow d.\#\text{Claves}$ $\Theta(1)$

Complejidad: $\Theta(1)$