

Algoritmos y Estructura de Datos II

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

Trabajo Práctico Diseño

Grupo 1

Integrante	LU	Correo electrónico
Almansi, Emilio	123/12	lalala@lalala.com
Chapresto, Matías	123/12	lalala@lalala.com
Vileriño, Silvio	123/12	lalala@lalala.com
Alguno más si dios quiere	123/12	lalala@lalala.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Índice

1. Módulo Temporada	3
1.1. Interfaz	3
1.2. Representación	4
1.3. Algoritmos	4
1.4. Analisis de complejidades	4
2. Módulo ArbolCategorias	4
2.1. Interfaz	4
2.2. Representación	6
2.2.1. Invariante de Representación	6
2.2.1.1. El Invariante Informalmente	6
2.2.1.2. El Invariante Formalmente	7
2.2.2. Función de Abstracción	8
2.2.2.1. Funciones auxiliares	8
2.3. Algoritmos	8
2.4. Analisis de complejidades	11
2.5. Iterador de Categorias	13
2.5.1. Interfaz	13
2.5.2. Representación	14
2.5.3. Invariante de Representación	14
2.5.3.1. El Invariante Formalmente	14
2.5.4. Función de Abstracción	14
2.5.5. Algoritmos	15
2.5.6. Analisis de complejidades	15

1. Módulo Temporada

1.1. Interfaz

parámetros formales

géneros `acat`

Operaciones

se explica con: Temporada

géneros: temporada

INICIAR() \rightarrow *res*: **temporada**

Pre $\equiv \{true\}$

Post $\equiv \{posc.\}$

Complejidad: $O(1)$

Aliasing: Aliasing iniciar

ABRIRPARITARIA()

Pre $\equiv \{true\}$

Post $\equiv \{posc.\}$

Complejidad: $O(1)$

Aliasing: Aliasing abrirParitaria

CERRARACUERDO()

Pre $\equiv \{true\}$

Post $\equiv \{posc.\}$

Complejidad: $O(1)$

Aliasing: Aliasing cerrarAcuerdo

REABRIR()

Pre $\equiv \{true\}$

Post $\equiv \{posc.\}$

Complejidad: $O(1)$

Aliasing: Aliasing reabrir

GREMIOS()

Pre $\equiv \{true\}$

Post $\equiv \{posc.\}$

Complejidad: $O(1)$

Aliasing: Aliasing gremios

ENPARITARIAS()

Pre $\equiv \{true\}$

Post $\equiv \{posc.\}$

Complejidad: $O(1)$

Aliasing: Aliasing enParitarias

GREMIOSNEGOCIANDO()

Pre $\equiv \{true\}$

Post $\equiv \{posc.\}$

Complejidad: $O(1)$

Aliasing: Aliasing gremiosNegociando

EMPRESASNEGOCIANDO()

Pre $\equiv \{true\}$

Post $\equiv \{posc.\}$

Complejidad: $O(1)$

Aliasing: Aliasing empresasNegociando

TRABAJADORESNEGOCIANDO()

Pre $\equiv \{true\}$

Post $\equiv \{posc.\}$

Complejidad: $O(1)$

Aliasing: Aliasing trabajadoresNegociando

GREMIOCONFLICTIVO()

Pre $\equiv \{true\}$

Post $\equiv \{posc.\}$

Complejidad: $O(1)$

Aliasing: Aliasing gremioConflictivo

fin interfaz

1.2. Representación

1.3. Algoritmos

1.4. Analisis de complejidades

2. Módulo ArbolCategorias

2.1. Interfaz

parámetros formales

géneros *acat*

se explica con: *ArbolDeCategorias*

Operaciones

CATEGORIASAC(**in** *ac*: *acat*) \rightarrow *res*: **itCategorias**

Pre $\equiv \{true\}$

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

Complejidad: $O(1)$

Aliasing: No se debe modificar nada de lo iterado por *res*.

RAIZAC(in ac : **acat**) $\rightarrow res$: **Categoria**

Pre $\equiv \{true\}$

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

Complejidad: $O(1)$

Aliasing: El nombre de la categoría raíz se pasa por referencia, no debe ser modificado.

IDAC(in ac : **acat**, in c : **Categoria**) $\rightarrow res$: **nat**

Pre $\equiv \{true\}$

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

Complejidad: $O(|c|)$

Aliasing: No tiene.

ALTURACATAC(in ac : **acat**, in c : **Categoria**) $\rightarrow res$: **nat**

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

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

Complejidad: $O(|c|)$

Aliasing: No tiene.

HIJOSAC(in ac : **acat**, in c : **Categoria**) $\rightarrow res$: **itHijos**

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

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

Complejidad: $O(|c|)$

Aliasing: No se debe modificar nada de lo iterado por res.

PADREAC(in ac : **acat**, in c : **Categoria**) $\rightarrow res$: **Categoria**

Pre $\equiv \{true\}$

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

Complejidad: $O(|c|)$

Aliasing: El nombre de la categoría padre se pasa por referencia, no debe ser modificado.

ALTURAAC(in ac : **acat**) $\rightarrow res$: **nat**

Pre $\equiv \{true\}$

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

Complejidad: $O(1)$

Aliasing: No tiene.

PREDECESORES(in ac : **acat**, in c : **Categoria**) $\rightarrow res$: **itFamilia**

Pre $\equiv \{true\}$

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

Complejidad: $O(|c|)$

Aliasing: res itera referencias a las categorías correspondientes.

NUEVOAC(in c : **Categoria**) $\rightarrow res$: **acat**

Pre $\equiv \{\neg vacia?(c)\}$

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

Complejidad: $O(|c|)$

Aliasing: No tiene.

AGREGARAC(**in/out** *ac*: *acat*, **in** *c*: *categoria*, **in** *h*: *categoria*)

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

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

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

Aliasing: No hay alias ya que no devuelve nada.

ESTA?(**in** *c*: *categoria*, **in** *ac*: *acat*) $\rightarrow res$: **bool**

Pre $\equiv \{true\}$

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

Complejidad: $O(|c|)$

Aliasing: No tiene.

ESSUBCATEGORIA(**in** *ac*: *acat*, **in** *c*: *categoria*, **in** *h*: *categoria*) $\rightarrow res$: **bool**

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

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

Complejidad: $O(|h| + |c| + alturaAC(ac))$

Aliasing: No tiene.

fin interfaz

2.2. Representación

ArbolCategorias se representa con *estrAC*, donde *estrAC* es tupla<
 raiz: puntero(*datosCat*),
 cantidad: nat,
 alturaMax: nat,
 familia: diccTrie(*Categoria*, puntero(*datosCat*)),
 categorias: Lista(*datosCat*)>

datosCat es tupla<
 categoria: *Categoria*,
 id: nat,
 altura: nat,
 hijos: Conj(puntero(*datosCat*)),
 padre: puntero(*datosCat*)>

Arbol de Categorias guarda en su estructura una Lista de *datosCat* (*categorias*), que cada uno guarda todos los datos de una categoria.

Guardamos en un diccTrie(*familia*) para cada categoria, un puntero a su *datosCat* correspondiente de la lista *categorias* para acceder a esos datos en $O(\text{longitud de la categoria})$.

En *raiz* guardamos un puntero a *datosCat* de la categoria raiz del arbol para accederla en $O(1)$

cantidad es la cantidad de categorias que tiene el arbol y nos permite en $O(1)$ saber cual va a ser el id para una categoría que estemos agregando.

alturaMax es la altura del arbol de categorias.

2.2.1. Invariante de Representación

2.2.1.1. El Invariante Informalmente

1. Para cada clave de '*familia*' obtener el significado devolvera un puntero(*datosCat*) donde '*categoria*' es igual a la clave.
2. Toda clave de '*familia*' debera ser raiz o pertenecer a algun conjunto de punteros de '*hijos*' de alguna otra clave.
3. Todos los significados de '*familia*' apuntan a un nodo de '*categorias*' y cada nodo de '*categorias*' es significado de alguna clave de '*familia*'.
4. Todos los elementos de '*hijos*' de una clave de '*familia*', tendrá como '*padre*' a esa clave.
5. '*cantidad*' sera igual a la longitud de la lista '*categorias*'.
6. Cuando la clave es igual a '*raiz*' su '*altura*' e '*id*' es 1.
7. La '*altura*' de cada clave es menor o igual a '*alturaMax*' del sistema.
8. Existe una clave en la cual '*altura*' es igual a '*alturaMax*'.
9. Los '*hijos*' de una clave tienen '*altura*' igual a $1 + \text{'altura de la clave'}$.
10. Los '*id*' de cada clave deberan ser menor o igual a '*cant*'.
11. No hay '*id*' repetidos en '*familia*'.

2.2.1.2. El Invariante Formalmente

Rep : estrAC \rightarrow boolean

$(\forall ac: \text{estrAC}) \text{Rep}(ac) \equiv \text{true} \iff$

1. $(\forall c: \text{Categoria})(\text{def?}(c, e.familia)) \iff (*\text{obtener}(c, e.familia)).categoria = c \wedge_L$
2. $(\forall c_1: \text{Categoria})(\text{def?}(c_1, e.familia)) \iff ((c_1 == e.raiz) \vee$
 $((\exists c_2: \text{Categoria})(\text{def?}(c_2, e.familia)) \wedge_L c_1 \in (*\text{obtener}(c_2, e.familia)).hijos)) \wedge_L$
3. $(\forall c: \text{Categoria})(\text{def?}(c, e.familia)) \iff$
 $((\exists d: \text{datosCat})\text{esta?}(d, e.categorias) \wedge d.categoria == c) \wedge_L d == \text{obtener}(c, e.familia))$
 \wedge_L
4. $(\forall c_1, c_2: \text{Categoria})(\text{def?}(c_1, e.familia)) \wedge (\text{def?}(c_2, e.familia)) \Rightarrow_L$
 $c_2 \in (*(\text{obtener}(c_1, e.familia))).hijos \iff$
 $(*(\text{obtener}(c_2, e.familia)).padre).categoria = c_1 \wedge_L$
5. $e.cantidad = \text{longitud}(e.categorias) \wedge_L$
6. $(\forall c: \text{categoria})(\text{def?}(c, e.familia)) \wedge c = e.raiz \Rightarrow_L$
 $(*(\text{obtener}(c, e.familia))).altura = 1 \wedge (*(\text{obtener}(c, e.familia))).id = 1 \wedge_L$
7. $(\forall c: \text{Categoria})(\text{def?}(c, e.familia)) \Rightarrow_L (*\text{obtener}(c, e.familia)).altura \leq e.alturaMax$
 \wedge_L
8. $(\exists c: \text{Categoria})(\text{def?}(c, e.familia)) \wedge_L (*(\text{obtener}(c, e.familia))).altura = e.alturaMax$
 \wedge_L
9. $(\forall c_1, c_2: \text{Categoria})(\text{def?}(c_1, e.familia)) \wedge (\text{def?}(c_2, e.familia)) \wedge_L$
 $((\exists d: \text{datosCat})d \in (*(\text{obtener}(c_1, e.familia))).hijos \wedge d.categoria == c_2) \Rightarrow_L$
 $(*(\text{obtener}(c_2, e.familia))).altura = 1 + (*(\text{obtener}(c_1, e.familia))).altura \wedge_L$
10. $(\forall c: \text{Categoria})(\text{def?}(c, e.familia)) \Rightarrow_L (*(\text{obtener}(c, e.familia))).id \leq e.cant \wedge_L$
11. $(\forall c_1, c_2: \text{Categoria})(\text{def?}(c_1, e.familia)) \wedge (\text{def?}(c_2, e.familia)) \wedge c_1 \neq c_2 \Rightarrow_L$
 $(*(\text{obtener}(c_1, e.familia))).id \neq (*(\text{obtener}(c_2, e.familia))).id$

2.2.2. Función de Abstracción

$Abs : e : \text{estrAC} \rightarrow \text{acat} \quad \{\text{Rep}(e)\}$

$(\forall e : \text{estrAC}) \text{ Abs}(e) =_{\text{obs}} \text{ac} : \text{acat} \mid$

1. $\text{categorias}(\text{ac}) =_{\text{obs}} \text{todasLasCategorias}(e.\text{categorias}) \wedge_L$
2. $\text{raiz}(\text{ac}) =_{\text{obs}} (*e.\text{raiz}).\text{categoria} \wedge_L$
3. $(\forall c : \text{Categoria}) \text{esta?}(c, \text{ac}) \wedge c \neq \text{raiz}(\text{ac}) \Rightarrow_L$
 $\text{padre}(\text{ac}, c) = (*(*(\text{obtener}(c, e.\text{familia}))).\text{padre}).\text{categoria} \wedge_L$
4. $(\forall c : \text{Categoria}) \text{esta?}(c, \text{ac}) \Rightarrow_L \text{id}(\text{ac}, c) = (*(\text{obtener}(c, e.\text{familia}))).\text{id}$

2.2.2.1. Funciones auxiliares

$\text{todasLasCategorias} : \text{secu}(\text{datosCat}) \rightarrow \text{conj}(\text{categoria})$

$\text{todasLasCategorias}(\text{cs}) \equiv \text{if } \text{vacía?}(\text{cs}) \text{ then}$
 $\quad \emptyset$
 else
 $\quad \text{Ag}((\text{prim}(\text{cs})).\text{categoria}, \text{todasLasCategorias}(\text{fin}(\text{cs})))$
 fi

$\text{predecesores} : \text{arbolCategoriasac} \times \text{Categoriac} \rightarrow \text{Conj}(\text{categoria}) \quad \{c \in \text{categorias}(\text{ac})\}$

$\text{predecesores}(\text{ac}, c) \equiv \text{predecesoresAux}(\text{ac}, \text{categorias}(\text{ac}), c)$

$\text{predecesoresAux} : \text{arbolCategoriasac} \times \text{Conj}(\text{Categoria})_{\text{cc}} \times \text{Categoriac} \rightarrow \text{Conj}(\text{categoria})$
 $\quad \{c \in \text{categorias}(\text{ac}) \wedge \text{cc} \subseteq \text{categorias}(\text{ac})\}$

$\text{predecesoresAux}(\text{ac}, \text{cc}, c) \equiv \text{if } \emptyset? \text{cc} \text{ then}$
 $\quad \emptyset$
 else
 $\quad \text{if } \text{esSubCategoria}(\text{ac}, \text{dameUno}(\text{cc}), c) \text{ then}$
 $\quad \quad \text{Ag}(\text{dameUno}(\text{cc}), \text{predecesoresAux}(\text{ac}, \text{sinUno}(\text{cc}), c))$
 $\quad \text{else}$
 $\quad \quad \text{predecesoresAux}(\text{ac}, \text{sinUno}(\text{cc}), c)$
 $\quad \text{fi}$
 fi

2.3. Algoritmos

Algoritmo 1 iCategoriasAC

1: **function** ICATEGORIASAC(**in** $\text{ac} : \text{estrAC}$) $\rightarrow \text{res} : \text{itCategorias}$

2: $\text{res} \leftarrow \text{crearItCategorias}(\text{ac})$ //O(1)

3: **end function**

Complejidad: O(1)

Algoritmo 2 iRaizAC

1: **function** IRAIZ(**in** $\text{ac} : \text{estrAC}$) $\rightarrow \text{res} : \text{Categoria}$

2: $\text{res} \leftarrow (*\text{ac.raiz}).\text{categoria}$ //O(1)

3: **end function**

Complejidad: O(1)

Algoritmo 3 iDameCantidad

```
1: function IDAMECANTIDAD(in ac: estrAC) → res: nat  
2:   res ← ac.cantidad //O(1)  
3: end function
```

Complejidad: $O(1)$

Algoritmo 4 iIdAC

```
1: function IID(in ac: estrAC, in c: Categoria) → res: nat  
2:   res ← ((*obtener(c, ac.familia)).id //O(|c|)  
3: end function
```

Complejidad: $O(|c|)$

Algoritmo 5 iAlturaCatAC

```
1: function IALTURACATAC(in ac: estrAC, in c: Categoria) → res: nat  
2:   res ← (*obtener(c, ac.familia)).altura //O(|c|)  
3: end function
```

Complejidad: $O(|c|)$

Algoritmo 6 iHijosAC

```
1: function IHIJOSAC(in ac: estrAC, in c: Categoria) → res: itHijos  
2:   res ← crearItHijos(ac, c) //O(|c|)  
3: end function
```

Complejidad: $O(|c|)$

Algoritmo 7 iPadreAC

```
1: function IPADREAC(in ac: estrAC, in c: Categoria) → res: Categoria  
2:   res ← ((*obtener(c, ac.familia)).padre).categoria //O(|c|)  
3: end function
```

Complejidad: $O(|c|)$

Algoritmo 8 iAlturaAC

```
1: function IALTURAAC(in ac: estrAC) → res: nat  
2:   res ← ac.alturaMax //O(1)  
3: end function
```

Complejidad: $O(1)$

Algoritmo 9 iPredecesores

```
1: function IPREDECESORES(in ac: estrAC, in c: Categoria) → res: itFamilia  
2:   res ← crearItFamilia(ac, c) //O(|c|)  
3: end function
```

Complejidad: $O(|c|)$

Algoritmo 10 iNuevoAC

```
1: function INUEVOAC(in  $c$ : Categoria)  $\rightarrow res$ : estrAC
2:   res.cantidad  $\leftarrow$  1 //O(1)
3:   datosCat tuplaA //O(1)
4:   tuplaA  $\leftarrow$  tupla( $c$ ,1,1,vacio()),Null //O(|c|)
5:   puntero(datosCat) punt  $\leftarrow$  &tuplaA //O(1)
6:   res.raiz  $\leftarrow$  punt //O(1)
7:   res.alturaMax  $\leftarrow$  1 //O(1)
8:   definir( $c$ , punt, res.familia) //O(|c|)
9:   agregarAtras(tuplaA, res.categorias) //O(1)
10: end function
```

Complejidad: $O(|c|)$

Algoritmo 11 iAgregarAC

```
1: function IAGREGARAC(in/out  $ac$ : estrAC, in  $c$ : Categoria, in  $h$ : Categoria)
2:   puntero(datosCat) puntPadre  $\leftarrow$  obtener( $c$ , $ac$ .familia) //O(|c|)
3:   if (*puntPadre).altura ==  $ac$ .alturaMax then //O(1)
4:      $ac$ .alturaMax++ //O(1)
5:   end if
6:   datosCat tuplaA  $\leftarrow$  ( $h$ , $ac$ .cantidad+1,(*puntPadre).altura+1,vacio()),puntPadre //O(|h|)
7:   puntero(datosCat) punt  $\leftarrow$  &tuplaA //O(1)
8:   Agregar((*puntPadre).hijos,punt) //O(1)
9:   definir( $h$ ,punt, $ac$ .familia) //O(|h|)
10:   $ac$ .cantidad++ //O(1)
11:  agregarAtras(tuplaA, $ac$ .categorias) //O(1)
12: end function
```

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

Algoritmo 12 iEsta?

```
1: function IESTA?(in  $ac$ : estrAC, in  $c$ : Categoria)  $\rightarrow res$ : bool
2:    $res \leftarrow$  def?( $c$ , $ac$ .familia) //O(|c|)
3: end function
```

Complejidad: $O(|c|)$

Algoritmo 13 iEsSubCategoria

```
1: function IESSUBCATEGORIA(in ac: estrAC, in c: Categoria, in h: Categoria)  $\rightarrow$  res: bool
2:   res  $\leftarrow$  false //O(1)
3:   if h == c then //O(|h|)
4:     res  $\leftarrow$  true //O(1)
5:   else
6:     if h == raizAC(ac) then //O(|h|)
7:       res  $\leftarrow$  false //O(1)
8:     else
9:       puntero(datosCat) actual  $\leftarrow$  (*obtener(h,ac.familia)).padre //O(|h|)
10:      puntero(datosCat) puntC  $\leftarrow$  (*obtener(c,ac.familia)) //O(|c|)
11:      while res == false  $\wedge$  actual  $\neq$  NULL do //O(alturaAC(ac))
12:        if puntC.Id == actual.Id then //O(1)
13:          res  $\leftarrow$  true //O(1)
14:        else
15:          actual  $\leftarrow$  (*actual).padre //O(1)
16:        end if
17:      end while
18:    end if
19:  end if
20: end function
```

Complejidad: $O(|h| + |c| + \text{alturaAC}(\text{ac}))$

2.4. Analisis de complejidades

1. iCategoriasAC

Se devuelve un iterador de la lista **categorias** del arbol de categorias en $O(1)$. El iterador muestra sólo los nombres de las categorías.

Orden Total: $O(1)$

2. iRaiz

Se devuelve una referencia al nombre de la categoria raiz del arbol de categorias en $O(1)$.

Orden Total: $O(1)$

3. idameCantidad

Se devuelve en $O(1)$ el natural almacenado en el campo cantidad del arbol de categorias.

Orden Total: $O(1)$

4. iIdAC

Dada la categoria *c*, se obtiene en $O(|c|)$ el *datosCat* de dicha categoría y en $O(1)$ se devuelve el id que tiene el *datosCat* obtenido.

Orden Total: $O(|c|)$

5. iAlturaCatAC

Dada la categoria *c*, se obtiene en $O(|c|)$ el *datosCat* de dicha categoría y en $O(1)$ se devuelve la altura que tiene el *datosCat* obtenido.

Orden Total: $O(|c|)$

6. iHijosAC

Dada la categoria c , se obtiene en $O(|c|)$ el `datosCat` de dicha categoría y en $O(1)$ se devuelve un iterador al conjunto **hijos** del `datosCat` obtenido.

Orden Total: $O(|c|)$

7. iPadreAC

Dada la categoria c , se obtiene en $O(|c|)$ el `datosCat` de dicha categoría y en $O(1)$ se devuelve por referencia en $O(1)$ el nombre de la categoria del puntero **padre** que tiene el `datosCat` obtenido.

Orden Total: $O(|c|)$

8. iAlturaAC

Devuelve en $O(1)$ la **alturaMax** del arbol de categorias.

Orden Total: $O(1)$

9. iPredecesores

Devuelve un iterador a los predecesores de la categoría, incluyendola, en $O(|c|)$ que es lo que le cuesta conseguir el iterador.

Orden Total: $O(|c|)$

10. iNuevoAC

A `res.cantidad` le asignamos 1, que tarda $O(1)$. Creamos una nueva variable `tuplaA`, que es `datosCat`. Esto tarda $O(1)$.

Creamos la variable `punt`, que es un puntero a `datosCat` y le asignamos la referencia de `tuplaA`. Y esto tarda $O(1)$. A `tuplaA` le asignamos una nueva tupla `datosCat`, que en uno de sus componentes es la categoria c , y copiarse tarda $O(|c|)$. Los demas componentes de la tupla tardan en copiarse $O(1)$.

A `res.raiz` le asignamos `punt`, y tarda $O(1)$. A `res.alturaMax` le asignamos 1, y tarda $O(1)$. A `res.familia` le asignamos el `diccTrie` que nos da la operacion `definir`, a la cual le pasamos como clave la categoria c . Entonces `definir` tarda $O(|c|)$.

A `res.categorias` le asignamos la lista que nos da la operacion `AgregarAtras`, que tarda $O(1)$.

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

11. iAgregarAC

Obtenemos un puntero de `datosCat` de la categoria c usando la operacion `obtener` del `diccTrie` `ac.familia`, y lo asignamos a la variable `puntPadre`. Esto tarda $O(|c|)$.

Comparamos la altura de la tupla que apunta `puntPadre` con `ac.alturaMax`, y esto tarda $O(1)$. En caso que valga la guarda del `if` hacemos una suma y una asignacion, que cuesta $O(1)$.

Luego creamos y asignamos una tupla de `datosCat` `tuplaA`, que se le asigna una tupla con valores que tardan $O(1)$ en copiarse, excepto por la categoria h que es categoria. Entonces la asignacion y creacion de esa tupla tarda $O(|h|)$.

Creamos la variable `punt` que es un puntero a `datosCat`, y le asignamos la referencia de `tuplaA`. Esto tarda $O(1)$. Agregamos al conjunto de punteros `hijos` que apunta `puntPadre`, el puntero `punt`, que tarda $O(1)$. Definimos la clave h , con el significado `punt` al `diccTrie` `ac.familia`. Esto tarda $O(|h|)$.

Incrementamos `ac.cantidad`, tardando $O(1)$. Finalmente agregamos `atras tuplaA` a la lista `ac.categorias`. Esto tarda $O(1)$

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

12. iEsta?

Para ver si una categoria `c` esta en nuestro `arbolCategorias`, vemos si esta definida la clave `c` en el `diccTrie ac.familia`. Y esto tarda $O(|c|)$.

Orden Total: $O(|c|)$

13. iEsSubCategoria

Le asignamos a `res` un valor booleano igual a `false`, demorando $O(1)$. Comparamos las dos categorias si son iguales o no. Demorando $O(|h|)$. En caso afirmativo cambiamos el valor de `res` por `true`, demorando $O(1)$.

En caso negativo, consultamos si `h` es igual a `raizAC(ac)` demorando $O(|h|)$, en caso positivo le asignamos a `res` el valor `false`, tardando $O(1)$. En caso negativo: creamos un puntero a `datosCat` denominado `actual` al cual le asignamos la tupla obtenida por la operacion `obtener` del `diccTrie` pasandole la categoria `h` y pidiendo padre de la tupla obtenida por esta operacion, esto demora $O(|h|)$. Creamos un puntero a `datosCat` denominado `puntC` al cual le asignamos la tupla obtenida por la operacion `obtener` del `diccTrie` pasandole la categoria `c` y pidiendo padre de la tupla obtenida por esta operacion, esto demora $O(|c|)$. Luego, se ingresa a un ciclo con la condicion de que `res` sea igual a `false` y `actual` distinto de `NULL`. Se compara `puntC` con `actual`. En caso afirmativo se asigna a `res` el valor `true`, demorando $O(1)$, en caso negativo, se modifica `actual` asignandole el puntero a padre de la tupla a la que estaba apuntando anteriormente. Luego de realizar `alturaAC(ac)` iteraciones se sale del ciclo.

Orden Total:

$O(1)+O(|h|)+O(1)+O(|h|)+O(1)+O(|h|)+O(|c|)+(alturaAC(ac)*(O(1)+O(1)+O(1)))=$
 $O(|h|+|c|+alturaAC(ac))$

2.5. Iterador de Categorias

2.5.1. Interfaz

parámetros formales

géneros `itCategorias`

se explica con: Iterador Unidireccional de Categoria

Operaciones

HAYSIGUIENTE?(**in** `it: itCategorias`) → `res: bool`

Pre $\equiv \{true\}$

Post $\equiv \{res =_{obs} HayMas?(it)\}$

Complejidad: $O(1)$

Aliasing: No tiene.

SIGUIENTE(**in** $it: \text{itCategorias}$) $\rightarrow res: \text{Categoria}$

Pre $\equiv \{HayMas?(it)\}$

Post $\equiv \{res =_{\text{obs}} Actual(it)\}$

Complejidad: $O(1)$

Aliasing: La categoría se pasa por referencia, no debe ser modificada.

AVANZAR(**in/out** $it: \text{itCategorias}$)

Pre $\equiv \{it =_{\text{obs}} it_0 \wedge HayMas?(it)\}$

Post $\equiv \{it =_{\text{obs}} avanzar(it_0)\}$

Complejidad: $O(1)$

Aliasing: No tiene.

fin interfaz

2.5.2. Representación

itCategorias se representa con itLista(datosCat)

datosCat es tupla<
categoria: Categoria,
id: nat,
altura: nat,
hijos: Conj(puntero(datosCat)),
padre: puntero(datosCat)>

itCategorias es un iterador de lista común. Sus complejidades nos alcanzan para iterar una Lista(datosCat).

2.5.3. Invariante de Representación

2.5.3.1. El Invariante Formalmente

$\text{Rep} : \text{estrITC} \rightarrow \text{boolean}$

$(\forall it: \text{estrITC}) \text{Rep}(it) \equiv \text{true}$

2.5.4. Función de Abstracción

$\text{Abs} : e: \text{estrITC} \rightarrow \text{itUni}(\text{Categoria}) \quad \{\text{Rep}(e)\}$

$(\forall e: \text{estrITC}) \text{Abs}(e) =_{\text{obs}} it: \text{itUni}(\text{Categoria}) \mid$

1. $siguientes(e) =_{\text{obs}} siguientes(it)$

2.5.5. Algoritmos

Algoritmo 14 iCrearItCategorias

```
1: function ICLEARITCATEGORIAS(in ac: estrAC) → res: estrITC  
2:   res ← crearIt(ac.categorias) //O(1)  
3: end function
```

Complejidad: O(1)

Algoritmo 15 iHaySiguiente?

```
1: function IHAYSIGUIENTE?(in e: estrITC) → res: bool  
2:   res ← haySiguiente(e) //O(1)  
3: end function
```

Complejidad: O(1)

Algoritmo 16 iSiguiente

```
1: function ISIGUIENTE(in e: estrITC) → res: Categoria  
2:   res ← (siguiente(e)).categoria //O(1)  
3: end function
```

Complejidad: O(1)

Algoritmo 17 iAvanzar

```
1: function IAVANZAR(in/out e: estrITC)  
2:   avanzar(e) //O(1)  
3: end function
```

Complejidad: O(1)

2.5.6. Analisis de complejidades

1. iCrearItCategorias

Crea un itCategorias con la lista del arbol de categorias que se pasa como parametro y se la asigna a res, esto demora O(1).

Orden Total: O(1)

2. iHaySiguiente?

Se llama a HaySiguiente del Iterador de Lista en O(1).

Orden Total: O(1)

3. iSiguiente

Se llama a Siguiente del Iterador de Lista en O(1). Se devuelve una referencia al valor categoria de la tupla DatosCat.

Orden Total: O(1)

4. iAvanzar

Se llama a Avanzar del Iterador de Lista en O(1).

Orden Total: O(1)