

# 1. TAD DATOSCATEGORÍA

## TAD DATOSCATEGORÍA

**géneros**      `datoscat`

**exporta**      `datoscat`, `generadores`, `observadores básicos`, `idDC`, `esRaíz?`, `padreDC`, `hijosDC`

**usa**          `BOOL`, `NAT`, `CATEGORIA`, `CONJUNTO(CATEGORIA)`, `ARBOLCATEGORIAS`

**igualdad observacional**

$$(\forall dc, dc' : \text{datoscat}) \left( dc =_{\text{obs}} dc' \iff \left( \text{nombreDC}(dc) =_{\text{obs}} \text{nombreDC}(dc') \wedge \text{acatDC}(dc) =_{\text{obs}} \text{acatDC}(dc') \right) \right)$$

**generadores**

`crearDatosCat`      : `categoria c × acat ac`       $\longrightarrow$  `datoscat`       $\{c \in \text{categorias}(ac)\}$

**observadores básicos**

`nombreDC`          : `datoscat`       $\longrightarrow$  `categoria`

`acatDC`            : `datoscat`       $\longrightarrow$  `acat`

**otras operaciones**

`idDC`              : `datoscat`       $\longrightarrow$  `nat`

`esRaíz?`          : `datoscat`       $\longrightarrow$  `bool`

`padreDC`          : `datoscat dc`       $\longrightarrow$  `datoscat`       $\{\neg \text{esRaíz?}(dc)\}$

`hijosDC`          : `datoscat`       $\longrightarrow$  `conj(datoscat)`

`categoríasADatosCat` : `conj(categoria) cs × acat ac`  $\longrightarrow$  `conj(datoscat)`       $\{cs \subseteq \text{categorias}(ac)\}$

**axiomas**       $\forall c: \text{categoria}, \forall cs: \text{conj(categoria)}, \forall ac: \text{acat}$

`nombreDC(crearDatosCat(c, ac))`  $\equiv c$

`acatDC(crearDatosCat(c, ac))`  $\equiv ac$

`idDC(crearDatosCat(c, ac))`  $\equiv \text{id}(ac, c)$

`esRaíz?(crearDatosCat(c, ac))`  $\equiv c =_{\text{obs}} \text{raíz}(ac)$

`padreDC(crearDatosCat(c, ac))`  $\equiv \text{crearDatosCat}(\text{padre}(ac, c), ac)$

`hijosDC(crearDatosCat(c, ac))`  $\equiv \text{categoriasADatosCat}(\text{hijos}(ac, c), ac)$

`categoríasADatosCat(cs, ac)`  $\equiv \text{if } \emptyset?(cs) \text{ then}$

$\emptyset$

**else**

`Ag(crearDatosCat(dameUno(cs), ac), sinUno(cs))`

**fi**

**Fin TAD**

## 2. Módulo ÁrbolCategorías

### Interfaz

**se explica con:** `DATOSCATEGORÍA`, `ÁRBOLCATEGORÍAS`, `ITERADOR UNIDIRECCIONAL(CATEGORÍA)`.

**géneros:** `acat`, `datoscat`, `itercat`.

### Operaciones básicas de árbol de categorías

**CREARÁRBOL**(**in** `raiz: categoria`)  $\rightarrow$  `res: acat`

**Pre**  $\equiv \{\neg \text{vacía?}(raiz)\}$

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

**Complejidad:**  $\Theta(|raiz|)$

**Descripción:** crea un árbol nuevo cuya categoría raíz es `raiz`.

**NOMBRECATEGORÍARAÍZ**(**in** `ac: acat`)  $\rightarrow$  `res: categoria`

**Pre**  $\equiv \{\text{true}\}$

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

**Complejidad:**  $\Theta(1)$

**Descripción:** devuelve el nombre de la categoría raíz de `ac`.

**AGREGARCATEGORÍA**(**in** `hija: categoria`, **in** `padre: categoria`, **in/out** `ac: acat`)

**Pre**  $\equiv \{ac =_{\text{obs}} ac_0 \wedge \text{está?}(padre, ac) \wedge \neg \text{vacía?}(hija) \wedge \neg \text{está?}(hija, ac)\}$

**Post**  $\equiv \{ac =_{\text{obs}} \text{agregar}(ac_0, padre, hija)\}$

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

**Descripción:** agrega la categoría *hija* como hija de la categoría *padre*.

CREARITERCAT(**in** *padre*: categoria, **in** *ac*: acat)  $\rightarrow$  *res* : intercat

**Pre**  $\equiv \{está?(padre, ac)\}$

**Post**  $\equiv \{alias(esPermutacion?(SecuSuby(res), hijos(ac, padre))) \wedge vacia?(Anteriores(res))\}$

**Complejidad:**  $\Theta(|padre|)$

**Descripción:** devuelve un iterador unidireccional de las categorías hijas directas de la categoría *padre*.

DUDA: ¿Puedo tratar a *res* acá directamente como un  $itConj(\alpha)$  en la expresión  $SecuSuby(res)$ ?

DUDA: ¿Hay que extender el TAD ÁrbolCategorías como en el apunte de módulos básicos para poder especificar la operación  $esPermutacion??$

IDCATEGORÍAPORNOMBRE(**in** *c*: categoria, **in** *ac*: acat)  $\rightarrow$  *res* : nat

**Pre**  $\equiv \{está?(c, ac)\}$

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

**Complejidad:**  $\Theta(|c|)$

**Descripción:** devuelve el *id* de la categoría *c*.

## Operaciones de datos de categoría

OBTENERID(**in** *dc*: datoscat)  $\rightarrow$  *res* : nat

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} idDC(dc)\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** devuelve el *id* de la categoría asociada a *dc*.

OBTENERNOMBRE(**in** *dc*: datoscat)  $\rightarrow$  *res* : nat

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} nombreDC(dc)\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** devuelve el nombre de la categoría asociada a *dc*.

OBTENERPADRE(**in** *dc*: datoscat)  $\rightarrow$  *res* : puntero(datoscat)

**Pre**  $\equiv \{\neg esRaíz?(dc)\}$

**Post**  $\equiv \{res =_{obs} padreDC(dc)\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** devuelve un puntero a los datos de la categoría padre asociada a *dc*.

DUDA: ¿Está bien la postcondición? *res* es de tipo puntero(datoscat), mientras que padreDC devuelve datoscat.

OBTENERHIJOS(**in** *dc*: datoscat)  $\rightarrow$  *res* : conj(puntero(datoscat))

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{obs} hijosDC(dc)\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** devuelve un conjunto de punteros a los datos de las categorías hijas directas asociadas a *dc*.

DUDA: ¿Está bien la postcondición? *res* es de tipo conj(puntero(datoscat)), mientras que hijosDC devuelve conj(datoscat).

## Operaciones de iterador de categorías

HAYMÁS?(**in** *it*: intercat)  $\rightarrow$  *res* : bool

**Pre**  $\equiv \{true\}$

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

**Complejidad:**  $\Theta(1)$

**Descripción:** devuelve **true** si y sólo si en el iterador todavía quedan elementos para avanzar.

ACTUAL(**in** *it*: intercat)  $\rightarrow$  *res* : puntero(datoscat)

**Pre**  $\equiv \{HayMás?(it)\}$

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

**Complejidad:**  $\Theta(1)$

**Descripción:** devuelve el elemento actual del iterador.

AVANZAR(**in/out** *it*: intercat)

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

**Post**  $\equiv \{res =_{\text{obs}} \text{Avanzar}(it_0)\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** avanza el iterador a la posición siguiente.

## Representación

### Representación de árbol de categorías

acat se representa con `estr_acat`

donde `estr_acat` es `tupla(raíz: puntero(datoscat), categorías: dicctrie(datoscat))`

Invariante de representación:

1. raíz no puede ser nulo.
2. raíz tiene que estar en el diccionario de categorías.
3. raíz tiene que tener id 1.
4. para todas las categorías en el diccionario:
  - a) la categoría no puede ser nula.
  - b) el nombre de la categoría deber ser igual a su clave en el diccionario.
  - c) el id de la categoría debe estar en rango.
  - d) dos categorías no pueden tener el mismo id.
  - e) los hijos de la categoría tienen que estar en el diccionario de categorías.
  - f) el padre es nulo si y sólo si la categoría es la raíz.
  - g) si el padre no es nulo, tiene que estar en el diccionario de categorías.
  - h) si el padre no es nulo, la categoría está entre los hijos del padre.
  - i) si el padre no es nulo, el id de la categoría debe ser superior al del padre.

`Rep` : `estr_acat`  $\longrightarrow$  `bool`

`Rep(e)`  $\equiv$  `true`  $\iff$

- (1)  $\neg(e.\text{raíz} =_{\text{obs}} \text{NULL}) \wedge_L$
- (2)  $\text{def?}(e.\text{raíz} \rightarrow \text{nombre}, e.\text{categorías}) \wedge_L \text{obtener}(e.\text{raíz} \rightarrow \text{nombre}, e.\text{categorías}) =_{\text{obs}} e.\text{raíz} \wedge$
- (3)  $e.\text{raíz} \rightarrow \text{id} =_{\text{obs}} 1 \wedge$
- (4)  $(\forall c: \text{categoria})(\text{def?}(c, e.\text{categorías}) \Rightarrow_L ($ 
  - (4a)  $\neg(\text{obtener}(c, e.\text{categorías}) =_{\text{obs}} \text{NULL}) \wedge_L$
  - (4b)  $\text{obtener}(c, e.\text{categorías}) \rightarrow \text{nombre} =_{\text{obs}} c \wedge_L$
  - (4c)  $1 \leq \text{obtener}(c, e.\text{categorías}) \rightarrow \text{id} \wedge \text{obtener}(c, e.\text{categorías}) \rightarrow \text{id} \leq \#(\text{claves}(e.\text{categorías})) \wedge$
  - (4d)  $(\forall c': \text{categoria})(\text{obtener}(c, e.\text{categorías}) \rightarrow \text{id} =_{\text{obs}} \text{obtener}(c', e.\text{categorías}) \rightarrow \text{id} \iff c =_{\text{obs}} c') \wedge$
  - (4e)  $(\forall h: \text{puntero}(\text{datoscat}))(h \in \text{obtener}(c, e.\text{categorías}) \rightarrow \text{hijos} \Rightarrow_L$ 
 $\text{def?}(h \rightarrow \text{nombre}, e.\text{categorías})) \wedge_L$
  - (4f)  $\text{obtener}(c, e.\text{categorías}) \rightarrow \text{padre} =_{\text{obs}} \text{NULL} \iff c =_{\text{obs}} e.\text{raíz} \rightarrow \text{nombre} \wedge_L$
  - (4g)  $\neg(\text{obtener}(c, e.\text{categorías}) \rightarrow \text{padre} =_{\text{obs}} \text{NULL}) \Rightarrow_L$ 
 $\text{def?}(\text{obtener}(c, e.\text{categorías}) \rightarrow \text{padre} \rightarrow \text{nombre}, e.\text{categorías}) \wedge_L$
  - (4h)  $\neg(\text{obtener}(c, e.\text{categorías}) \rightarrow \text{padre} =_{\text{obs}} \text{NULL}) \Rightarrow_L$ 
 $\text{obtener}(c, e.\text{categorías}) \in \text{obtener}(c, e.\text{categorías}) \rightarrow \text{padre} \rightarrow \text{hijos}) \wedge$
  - (4i)  $\neg(\text{obtener}(c, e.\text{categorías}) \rightarrow \text{padre} =_{\text{obs}} \text{NULL}) \Rightarrow_L$ 
 $\text{obtener}(c, e.\text{categorías}) \rightarrow \text{padre} \rightarrow \text{id} < \text{obtener}(c, e.\text{categorías}) \rightarrow \text{id}))$

DUDA: ¿Está bien acceder a los campos de `datoscat` como si fuera una tupla? Notar que se usa el género `datoscat` en vez de su estructura de representación `estr_datoscat`.

DUDA: Para acortar el `Rep`, ¿puedo declarar variables dentro del mismo? Ejemplo:

$$x =_{\text{obs}} \text{obtener}(c, e.\text{categorías}) \wedge \neg(x \rightarrow \text{padre} =_{\text{obs}} \text{NULL}) \Rightarrow_L$$

$$(\text{def?}(x \rightarrow \text{padre} \rightarrow \text{nombre}, e.\text{categorías}) \wedge_L x \in x \rightarrow \text{padre} \rightarrow \text{hijos})$$

`Abs` : `estr_acat e`  $\longrightarrow$  `acat`

$\{\text{Rep}(e)\}$

`Abs(e)`  $=_{\text{obs}}$  `ac`: `acat` |  $\text{categorias}(ac) =_{\text{obs}} \text{claves}(e.\text{categorías}) \wedge_L$

$\text{raíz}(ac) =_{\text{obs}} e.\text{raíz} \rightarrow \text{nombre} \wedge$

$(\forall c: \text{categoria})(c \in \text{claves}(e.\text{categorías}) \Rightarrow_L ($

$\text{padre}(ac, c) =_{\text{obs}} \text{obtener}(c, e.\text{categorías}) \rightarrow \text{padre} \rightarrow \text{nombre} \wedge$

$\text{id}(ac, c) =_{\text{obs}} \text{obtener}(c, e.\text{categorías}) \rightarrow \text{id}))$

## Representación de datos de categoría

**datoscat se representa con *estr\_datoscat***

donde *estr\_datoscat* es tupla(*id*: nat , *nombre*: categoria , *padre*: puntero(datoscat) , *hijos*: conj(puntero(datoscat)) )

DUDA: ¿Hace falta crear un TAD nuevo para poder expresar el Rep y Abs del género datoscat? De ser así, ¿tengo que agregar dicho TAD en la lista 'se explica con'?

## Representación de iterador de categorías

**itercat se representa con *itConj(puntero(datoscat))***

DUDA: ¿Está bien la estructura de representación elegida?

DUDA: ¿Qué usamos para expresar el Rep y el Abs en este caso? Dado que se trata de un *itConj( $\alpha$ )*, suena razonable pensar en usar el Rep y Abs de este iterador definidos en el módulo Conjunto Lineal( $\alpha$ ).

## Algoritmos

**ICREARÁRBOL(in *raiz*: categoria)  $\rightarrow$  *res* : *estr\_acat***  
var *raiz\_estr*: *estr\_datoscat*  
var *categorias*: dicctrie(*estr\_datoscat*)

*raiz\_estr*  $\leftarrow$   $\langle$ id: 1, nombre: *raiz*, padre: NULL, hijos: VACÍO() $\rangle$   
*categorias*  $\leftarrow$  CREARDICCIONARIO()

DEFINIR(*raiz*,  $\&$ (*raiz\_estr*), *categorias*)

*res*  $\leftarrow$   $\langle$ raiz:  $\&$ (*raiz\_estr*), *categorias*: *categorias* $\rangle$

**INOMBRECATEGORÍARAÍZ(in *ac*: *estr\_acat*)  $\rightarrow$  *res* : categoria**  
*res*  $\leftarrow$  *ac.raiz*  $\rightarrow$  *nombre*

**IAGREGARCATEGORÍA(in *hija*: categoria, in *padre*: categoria, in/out *ac*: *estr\_acat*)**  
var *estr\_hija*: *estr\_datoscat*  
*estr\_hija*  $\leftarrow$   $\langle$ id: #CLAVES(*ac.categorias*) + 1, nombre: *hija*,  
padre: OBTENER(*padre*, *ac.categorias*), hijos: VACÍO()  $\rangle$   
DEFINIR(*hija*, *estr\_hija*, *ac*)

## 3. Módulo LinkLinkIt

### Interfaz

**se explica con:** LINKLINKIT, ITERADOR UNIDIRECCIONAL(LINK).

**géneros:** sistema, iterlinks.

### Operaciones básicas del sistema

**CREARSISTEMA(in *ac*: *acat*)  $\rightarrow$  *res* : sistema**  
**Pre**  $\equiv$  {true}  
**Post**  $\equiv$  {*res* =<sub>obs</sub> iniciar(*ac*)}  
**Complejidad:**  $\Theta(\#(\text{categorias}(ac)))$   
**Descripción:** crea un sistema cuyo árbol de categorías es *ac*.

**AGREGARLINK(in *l*: link, in *c*: categoria, in/out *s*: sistema)**  
**Pre**  $\equiv$  {*s* =<sub>obs</sub> *s*<sub>0</sub>  $\wedge$   $\neg(l \in \text{links}(s)) \wedge$  está?(*c*, categorías(*s*))}  
**Post**  $\equiv$  {*s* =<sub>obs</sub> nuevoLink(*s*<sub>0</sub>, *l*, *c*)}  
**Complejidad:**  $\Theta(|l| + |c| + h)$ , donde *h* representa altura(categorías(*s*)).  
**Descripción:** agrega al sistema el link *l* con categoría *c*.

ACCEDERLINK(**in**  $l$ : link, **in**  $f$ : fecha, **in/out**  $s$ : sistema)

**Pre**  $\equiv \{s =_{\text{obs}} s_0 \wedge l \in \text{links}(s) \wedge f \geq \text{fechaActual}(s)\}$

**Post**  $\equiv \{s =_{\text{obs}} \text{acceso}(s_0, l, f)\}$

**Complejidad:**  $\Theta(|l| + h)$ , donde  $h$  representa altura(categorías( $s$ )).

**Descripción:** registra un acceso al link  $l$  en la fecha  $f$ .

#LINKS(**in**  $c$ : categoría, **in**  $s$ : sistema)  $\rightarrow res$ : nat

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

**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.

CREARITERLINKS(**in**  $c$ : categoría, **in**  $s$ : sistema)  $\rightarrow res$ : iterlinks

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

**Post**  $\equiv \{\text{alias}(\text{SecuSuby}(res) =_{\text{obs}} \text{linksOrdenadosPorAccesos}(ac, \text{padre})) \wedge \text{vacía?}(\text{Anteriores}(res))\}$

**Complejidad:**  $\Theta(|c| + n \cdot \log(n))$ , donde  $n$  representa  $\text{long}(\text{linksOrdenadosPorAccesos}(s, c))$ .

**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.

## Operaciones de iterador de links

HAYMÁS?(**in**  $it$ : iterlinks)  $\rightarrow res$ : 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**  $it$ : iterlinks)  $\rightarrow res$ : link

**Pre**  $\equiv \{\text{HayMás?}(it)\}$

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

**Complejidad:**  $\Theta(1)$

**Descripción:** devuelve el link actual del iterador.

CATEGORÍALINKACTUAL(**in**  $it$ : iterlinks)  $\rightarrow res$ : categoría

**Pre**  $\equiv \{\text{HayMás?}(it)\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{categoríaLink}(???, \text{Actual}(it))\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** devuelve la categoría del link actual del iterador.

DUDA: En la postcondición no dispongo de un sistema para la llamada a categoríaLink, pues la interfaz de la función no recibe un sistema como parámetro ya que toda la información que necesita el algoritmo está contenida en el iterador. ¿Cómo hago para especificar la postcondición si no tengo un sistema?

ACCESOSRECIENTESLINKACTUAL(**in**  $it$ : iterlinks)  $\rightarrow res$ : nat

**Pre**  $\equiv \{\text{HayMás?}(it)\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{accesosRecientes}(???, ???, \text{Actual}(it))\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** devuelve la cantidad de accesos del link actual del iterador durante los días de la intersección de los tres días “recientes” del link  $l$  y de 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.

DUDA: Misma duda que en la función anterior: ¿cómo hago para especificar la postcondición si no dispongo ni de la categoría ni del sistema para la llamada a accesosRecientes?

AVANZAR(**in/out**  $it$ : iterlinks)

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

**Post**  $\equiv \{res =_{\text{obs}} \text{Avanzar}(it_0)\}$

**Complejidad:**  $\Theta(1)$

**Descripción:** avanza el iterador a la posición siguiente.

# Representación

## Representación del sistema

sistema se representa con `estr_sistema`

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

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

donde `estr_linkscat` es tupla(*c*: `puntero(datoscat)` , *ls*: `lista(puntero(estr_link))` , *ultimoAcceso*:  
`fecha , ordenado?: bool` )

## Representación de iterador de links

`iterlinks` se representa con `estr_iter`

donde `estr_iter` es tupla(*s*: `estr_sistema` , *c*: `categoria` , *it*: `itLista(puntero(estr_link))` )

## Algoritmos

Pendiente.

## 4. Módulo Diccionario Trie( $\alpha$ )

### Interfaz

**parámetros formales**

**géneros**  $\alpha$

**función** `COPIAR(in a:  $\alpha$ )  $\rightarrow$  res :  $\alpha$`

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{res =_{\text{obs}} a\}$

**Complejidad**:  $\Theta(\text{copy}(a))$

**Descripción**: función de copia de  $\alpha$ 's

**se explica con**: `DICCIONARIO(String,  $\alpha$ ), ITERADOR UNIDIRECCIONAL( $\alpha$ ).`

**géneros**: `dicctrie( $\alpha$ ), iterdicctrie( $\alpha$ ).`

### Operaciones básicas de diccionario trie

`CREARDICCIONARIO()  $\rightarrow$  res : dicctrie( $\alpha$ )`

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{vacío}()\}$

**Complejidad**:  $\Theta(1)$

**Descripción**: crea un nuevo diccionario vacío.

`DEFINIDO?(in c: string, in d: dicctrie( $\alpha$ ))  $\rightarrow$  res : bool`

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{def?}(c, d)\}$

**Complejidad**:  $\Theta(|c|)$

**Descripción**: devuelve **true** si y sólo si *c* está definido en el diccionario.

`DEFINIR(in c: string, in s:  $\alpha$ , in/out d: dicctrie( $\alpha$ ))`

**Pre**  $\equiv \{d =_{\text{obs}} d_0\}$

**Post**  $\equiv \{d =_{\text{obs}} \text{definir}(d_0, c, s)\}$

**Complejidad**:  $\Theta(|c| + \text{copy}(s))$

**Descripción**: define la clave *c* con el significado *s* en el diccionario.

`OBTENER(in c: string, in d: dicctrie( $\alpha$ ))  $\rightarrow$  res :  $\alpha$`

**Pre**  $\equiv \{\text{def?}(c, d)\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{obtener}(c, d)\}$

**Complejidad**:  $\Theta(|c|)$

**Descripción**: devuelve el significado de la clave *c* en *d*.

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

## Operaciones del iterador

Pendiente.

## Representación

### Representación de diccionario trie

dicctrie( $\alpha$ ) se representa con estr\_dicctrie  
 donde estr\_dicctrie es tupla(*nodos*: arreglo\_estático[255] de puntero(estr\_dicctrie) , *significado*: puntero( $\alpha$ ) )

### Representación del iterador

Pendiente.

## Algoritmos

```

ICREARDiccionario()  $\rightarrow res$  : estr_dicctrie
  for  $i=0$  to  $|res.nodos|$  do:
     $res.nodos[i] \leftarrow \text{false}$ 
  endFor
   $res.significado \leftarrow \text{Null}$ 

IOBTENER(in  $c$ : string, in  $e$ : estr_dicctrie)  $\rightarrow res$  :  $\alpha$ 
  var  $nodoActual$ : puntero(estr_dicctrie)  $\leftarrow \&(e)$ 
  for  $i=0$  to  $|c|$  do:
     $nodoActual \leftarrow (*nodoActual).nodos[\text{ORD}(c[i])]$ 
  endFor
   $res \leftarrow (*nodoActual).(*significado)$ 

IDEFINIR(in  $c$ : string, in  $s$ :  $\alpha$ , in  $e$ : estr_dicctrie)
  var  $nodoActual$ : puntero(estr_dicctrie)  $\leftarrow \&(e)$ 
  for  $i=0$  to  $|c|$  do:
    if  $(*nodoActual).nodos[\text{ORD}(c[i])] = \text{NULL}$  then:
      var  $nuevoNodo$ : estr_dicctrie
      for  $j=0$  to  $|nuevoNodo.nodos|$  do:
         $nuevoNodo.nodos[j] \leftarrow \text{NULL}$ 
      endFor
       $nuevoNodo.significado \leftarrow \text{NULL}$ 
       $(*nodoActual).nodos[\text{ORD}(c[i])] \leftarrow \&(nuevoNodo)$ 
    endIf
     $nodoActual \leftarrow (*nodoActual).nodos[\text{ORD}(c[i])]$ 
  endFor
   $(*nodoActual).significado \leftarrow \&(\text{COPIA}(s))$ 

IDEFINIDO?(in  $c$ : string, in  $e$ : estr_dicctrie)  $\rightarrow res$  : bool
  var  $nodoActual$ : puntero(estr_dicctrie)  $\leftarrow \&(e)$ 
  while  $i < |c| \wedge (*nodoActual).nodos[\text{ORD}(c[i])] \neq \text{NULL}$  do:
     $nodoActual \leftarrow (*nodoActual).nodos[\text{ORD}(c[i])]$ 
  endWhile
   $res \leftarrow (i = |c|-1 \wedge (*nodoActual).significado \neq \text{NULL})$ 
  
```