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

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

generadores

crearDatosCat	: categoria c × acat ac	⟶	datoscat	{c ∈ categorias(ac)}

observadores básicos

nombreDC	: datoscat	⟶	categoria
acatDC	: datoscat	⟶	acat

otras operaciones

idDC	: datoscat	⟶	nat	
esRaíz?	: datoscat	⟶	bool	
padreDC	: datoscat dc	⟶	datoscat	{¬ esRaíz?(dc)}
hijosDC	: datoscat	⟶	conj(datoscat)	
categoríasADatosCat	: conj(categoria) cs × acat ac	⟶	conj(datoscat)	{cs ⊆ categorias(ac)}

axiomas ∀c: categoria, ∀cs: conj(categoria), ∀ac: acat

nombreDC(crearDatosCat(c, ac)) ≡ c
acatDC(crearDatosCat(c, ac)) ≡ ac
idDC(crearDatosCat(c, ac)) ≡ id(ac, c)
esRaíz?(crearDatosCat(c, ac)) ≡ $c =_{\text{obs}}$ raíz(ac)
padreDC(crearDatosCat(c, ac)) ≡ crearDatosCat(padre(ac, c), ac)
hijosDC(crearDatosCat(c, ac)) ≡ categoriasADatosCat(hijos(ac, c), ac)
categoríasADatosCat(cs, ac) ≡ **if** ∅?(cs) **then**
    ∅
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`) → res : `acat`
Pre ≡ {¬vacía?($raiz$)}
Post ≡ {$res =_{\text{obs}}$ nuevo($raiz$)}
Complejidad: $\Theta(|raiz|)$
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: $\Theta(1)$
Descripción: devuelve el nombre de la categoría raíz de ac.

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: $\Theta(|padre| + |hija|)$
Descripción: agrega la categoría $hija$ como hija de la categoría $padre$.

CrearIterCat(**in** $padre$: `categoria`, **in** ac: `acat`) $\rightarrow$ res : `itercat`
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(α) 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 $\equiv$ {$res =_{\mathrm{obs}}$ 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 $\equiv$ {$res =_{\mathrm{obs}}$ 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 $\equiv$ {$res =_{\mathrm{obs}}$ 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 $\equiv$ {$res =_{\mathrm{obs}}$ 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 $\equiv$ {$res =_{\mathrm{obs}}$ 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: `itercat`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\mathrm{obs}}$ HayMás?(it)}
Complejidad: $\Theta(1)$
Descripción: devuelve **true** si y sólo si en el iterador todavía quedan elementos para avanzar.

Actual(**in** it: `itercat`) $\rightarrow$ res : `puntero(datoscat)`
Pre $\equiv$ {HayMás?(it)}
Post $\equiv$ {$res =_{\mathrm{obs}}$ Actual(it)}
Complejidad: $\Theta(1)$
Descripción: devuelve el elemento actual del iterador.

Avanzar(**in/out** it: `itercat`)

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

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:
$x =_{\text{obs}} \text{obtener}(c, e.\text{categorías}) \wedge \neg(x\to\text{padre} =_{\text{obs}} \text{NULL}) \Rightarrow_{\text{L}}$
$(\text{def?}(x\to\text{padre}\to\text{nombre}, e.\text{categorías}) \wedge_{\text{L}} x \in x\to\text{padre}\to\text{hijos})$

Abs : estr_acat e $\longrightarrow$ acat {Rep(e)}
Abs(e) $=_{\text{obs}}$ ac: acat | $\text{categorias}(ac) =_{\text{obs}} \text{claves}(e.\text{categorías}) \wedge_{\text{L}}$
$\text{raíz}(ac) =_{\text{obs}} e.\text{raíz}\to\text{nombre} \wedge$
$(\forall c\text{: categoria})(c \in \text{claves}(e.\text{categorías}) \Rightarrow_{\text{L}} ($
$\text{padre}(ac, c) =_{\text{obs}} \text{obtener}(c, e.\text{categorías})\to\text{padre}\to\text{nombre} \wedge$
$\text{id}(ac, c) =_{\text{obs}} \text{obtener}(c, e.\text{categorías})\to\text{id}))$

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(α), suena razonable pensar en usar el Rep y Abs de este iterador definidos en el módulo Conjunto Lineal(α).

Algoritmos

iCrearÁrbol(**in** *raiz* : `categoria`) $\rightarrow$ *res* : `estr_acat`
var *raiz_estr*: `estr_datoscat`
var *categorias*: `dicctrie(estr_datoscat)`

raiz_estr $\leftarrow$ ⟨`id`: 1, `nombre`: *raiz*, `padre`: NULL, `hijos`: Vacío()⟩
categorias $\leftarrow$ CrearDiccionario()

Definir(*raiz*, &(*raiz_estr*), *categorias*)

res $\leftarrow$ ⟨`raiz`: &(*raiz_estr*), `categorias`: *categorias*⟩

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 $\leftarrow$ ⟨`id`: #Claves(*ac.categorias*) + 1, `nombre`: *hija*,
`padre`: Obtener(*padre*, *ac.categorias*), `hijos`: Vacío() ⟩
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 $\equiv$ {$res =_{\text{obs}}$ iniciar(ac)}
Complejidad: $\Theta(\#(\text{categorías}(ac))$
Descripción: crea un sistema cuyo árbol de categorías es *ac*.

AgregarLink(**in** *l*: `link`, **in** *c*: `categoria`, **in/out** *s*: `sistema`)
Pre $\equiv$ {$s =_{\text{obs}} s_0 \wedge \neg(l \in \text{links}(s)) \wedge \text{está?}(c, \text{categorías}(s))$}
Post $\equiv$ {$s =_{\text{obs}} \text{nuevoLink}(s_0, l, c)$}
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 $\equiv$ $\{s =_{\text{obs}} s_0 \land l \in \text{links}(s) \land f \geq \text{fechaActual}(s)\}$
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 $\equiv$ {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.

CrearIterLinks(**in** c: `categoría`, **in** s: `sistema`) $\rightarrow$ res : `iterlinks`
Pre $\equiv$ {está?(c, categorías(s))}
Post $\equiv$ {alias(SecuSuby(res) $=_{\text{obs}}$ linksOrdenadosPorAccesos(ac, $padre$)) $\land$ vacia?(Anteriores(res))}
Complejidad: $\Theta(|c| + n \cdot \log(n))$, 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.

Operaciones de iterador de links

HayMás?(**in** it: `iterlinks`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}}$ 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 $\equiv$ {HayMás?(it)}
Post $\equiv$ $\{res =_{\text{obs}}$ Actual(it)$\}$
Complejidad: $\Theta(1)$
Descripción: devuelve el link actual del iterador.

CategoríaLinkActual(**in** it: `iterlinks`) $\rightarrow$ res : `categoria`
Pre $\equiv$ {HayMás?(it)}
Post $\equiv$ $\{res =_{\text{obs}}$ categoríaLink(???, 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 $\equiv$ {HayMás?(it)}
Post $\equiv$ $\{res =_{\text{obs}}$ accesosRecientes(???, ???, 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 $\equiv$ $\{it =_{\text{obs}} it_0 \land$ HayMás?(it)$\}$
Post $\equiv$ $\{res =_{\text{obs}}$ 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` , *últimoAcceso*: `fecha` , *as*: `arreglo_estático[3] de fecha`)
donde `estr_linkscat` es `tupla`(*c*: `puntero(datoscat)` , *ls*: `lista(puntero(estr_link))` , *últimoAcceso*: `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(α)

Interfaz

parámetros formales
géneros α
función COPIAR(**in** $a: \alpha$) $\rightarrow res : \alpha$
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} a\}$
Complejidad: $\Theta(copy(a))$
Descripción: función de copia de α's

se explica con: DICCIONARIO(STRING, α), ITERADOR UNIDIRECCIONAL(α).

géneros: `dicctrie`(α), `iterdicctrie`(α).

Operaciones básicas de diccionario trie

CREARDICCIONARIO() $\rightarrow res$: `dicctrie`(α)
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}}$ vacío()$\}$
Complejidad: $\Theta(1)$
Descripción: crea un nuevo diccionario vacío.

DEFINIDO?(**in** c: `string`, **in** d: `dicctrie`(α)) $\rightarrow res$: `bool`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}}$ 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`(α))
Pre $\equiv$ $\{d =_{\text{obs}} d_0\}$
Post $\equiv$ $\{d =_{\text{obs}}$ definir$(d_0, c, s)\}$
Complejidad: $\Theta(|c| + copy(s))$
Descripción: define la clave c con el significado s en el diccionario.

OBTENER(**in** c: `string`, **in** d: `dicctrie`(α)) $\rightarrow res : \alpha$
Pre $\equiv$ {def?(c, d)}
Post $\equiv$ $\{res =_{\text{obs}}$ obtener$(c, d)\}$
Complejidad: $\Theta(|c|)$
Descripción: devuelve el significado de la clave c en d.

$\#$Claves(**in** d: `dicctrie`(α)) $\to res$: `nat`
Pre $\equiv$ {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`(α) **se representa con** `estr_dicctrie`

donde `estr_dicctrie` es `tupla`(*nodos*: `arreglo_estático[256] de puntero(estr_dicctrie)` , *significado*: `puntero`(α))

Representación del iterador

Pendiente.

Algoritmos

iCrearDiccionario() $\to res$: `estr_dicctrie`
    for i=0 to $|res$.nodos$|$ do:
        res.nodos[i] $\leftarrow$ false
    endFor
    res.significado $\leftarrow$ Null

iObtener(**in** c: `string`, **in** e: `estr_dicctrie`) $\to res$: α
    var *nodoActual*: `puntero(estr_dicctrie)` $\leftarrow$ &(e)
    for i=0 to $|c|$ do:
        nodoActual $\leftarrow$ (**nodoActual*).nodos[ord($c[i]$)]
    endFor
    res $\leftarrow$ (**nodoActual*).(*$significado$)

iDefinir(**in** c: `string`, **in** s: α, **in** e: `estr_dicctrie`)
    var *nodoActual*: `puntero(estr_dicctrie)` $\leftarrow$ &(e)
    for i=0 to $|c|$ do:
        if (**nodoActual*).nodos[ord($c[i]$)] = NULL then:
            var $nuevoNodo$: `estr_dicctrie`
            for j=0 to $|nuevoNodo.nodos|$ do:
                $nuevoNodo$.nodos[i] $\leftarrow$ NULL
            endFor
            $nuevoNodo$.significiado $\leftarrow$ NULL
            (**nodoActual*).nodos[ord($c[i]$)] $\leftarrow$ &($nuevoNodo$)
        endIf
        nodoActual $\leftarrow$ (**nodoActual*).nodos[ord($c[i]$)]
    endFor
    (**nodoActual*).significado $\leftarrow$ &(Copia(s))

iDefinido?(**in** c: `string`, **in** e: `estr_dicctrie`) $\to res$: `bool`
    var *nodoActual*: `puntero(estr_dicctrie)` $\leftarrow$ &(e)
    while $i <|c|$ $\wedge$ (**nodoActual*).nodos[ord($c[i]$)] != NULL do:
        nodoActual $\leftarrow$ (**nodoActual*).nodos[ord($c[i]$)]
    endWhile
    res $\leftarrow$ (i = $|c|$-1 $\wedge$ (**nodoActual*).siginficado != NULL)