

1. Módulo ÁrbolCategorías

Interfaz

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

géneros: `acat, datoscat, intercat`.

Operaciones básicas de árbol de categorías

CREARÁRBOL(*in raiz: categoria*) $\rightarrow res : acat$
Pre $\equiv \{\neg vacía?(raiz)\}$
Post $\equiv \{res =_{obs} nuevo(raiz)\}$
Complejidad: $\Theta(|raiz|)$
Descripción: crea un árbol nuevo cuya categoría raíz es *raiz*.

NOMBRECATEGORÍARAÍZ(*in ac: acat*) $\rightarrow res : categoria$
Pre $\equiv \{true\}$
Post $\equiv \{res =_{obs} raíz(ac)\}$
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 $\equiv \{ac =_{obs} ac_0 \wedge está?(padre, ac) \wedge \neg vacía?(hija) \wedge \neg está?(hija, ac)\}$
Post $\equiv \{ac =_{obs} agregar(ac_0, padre, hija)\}$
Complejidad: $\Theta(|padre| + |hija|)$
Descripción: agrega la categoría *hija* como hija de la categoría *padre*.

CREARITERCAT(*in padre: categoria, in ac: acat*) $\rightarrow res : intercat$
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(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 $\equiv \{res =_{obs} id(ac, c)\}$
Complejidad: $\Theta(|c|)$
Descripción: devuelve el *id* de la categoría *c*.

Operaciones de datos de categoría

DUDA: ¿Hace falta crear un TAD para el género `datoscat` para poder especificar las pre y postcondiciones de las funciones a continuación?

OBTENERID(*in dc: datoscat*) $\rightarrow res : nat$
Pre $\equiv \{???\}$
Post $\equiv \{???\}$
Complejidad: $\Theta(1)$
Descripción: devuelve el *id* de la categoría asociada a *dc*.

OBTENERNOMBRE(*in dc: datoscat*) $\rightarrow res : nat$
Pre $\equiv \{???\}$
Post $\equiv \{???\}$
Complejidad: $\Theta(1)$
Descripción: devuelve el nombre de la categoría asociada a *dc*.

OBTENERPADRE(*in dc: datoscat*) $\rightarrow res : puntero(datoscat)$
Pre $\equiv \{???\}$
Post $\equiv \{???\}$
Complejidad: $\Theta(1)$

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

$\text{OBTENERHIJOS}(\text{in } dc : \text{datoscat}) \rightarrow res : \text{conj}(\text{puntero}(\text{datoscat}))$

Pre $\equiv \{???\}$

Post $\equiv \{???\}$

Complejidad: $\Theta(1)$

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

Operaciones de iterador de categorías

$\text{HAYMÁS?}(\text{in } it : \text{itercat}) \rightarrow res : \text{bool}$

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.

$\text{ACTUAL}(\text{in } it : \text{itercat}) \rightarrow res : \text{puntero}(\text{datoscat})$

Pre $\equiv \{\text{HayMás?}(it)\}$

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

Complejidad: $\Theta(1)$

Descripción: devuelve el elemento actual del iterador.

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

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 $\text{tupla}(\text{raíz} : \text{puntero}(\text{datoscat}), \text{categorías} : \text{dicctrie}(\text{datoscat}))$

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.

$\text{Rep} : \text{estr_acat} \rightarrow \text{bool}$

$\text{Rep}(e) \equiv \text{true} \iff$

- (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:

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

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

$\text{Abs} : \text{estr_acat } e \longrightarrow \text{acat}$

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

$\text{Abs}(e) =_{\text{obs}} \text{ac: acat} \mid \text{categorias}(\text{ac}) =_{\text{obs}} \text{claves}(e.\text{categorías}) \wedge_L$
 $\text{raíz}(\text{ac}) =_{\text{obs}} e.\text{raíz} \rightarrow \text{nombre} \wedge$
 $(\forall c: \text{categoria})(c \in \text{claves}(e.\text{categorías}) \Rightarrow_L ($
 $\text{padre}(\text{ac}, c) =_{\text{obs}} \text{obtener}(c, e.\text{categorías}) \rightarrow \text{padre} \rightarrow \text{nombre} \wedge$
 $\text{id}(\text{ac}, c) =_{\text{obs}} \text{obtener}(c, e.\text{categorías}) \rightarrow \text{id}))$

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

Algoritmos

Pendiente.

2. 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* =_{obs} iniciar(*ac*)}

Complejidad: $\Theta(\#(\text{categorías}(\text{ac})))$

Descripción: crea un sistema cuyo árbol de categorías es *ac*.

AGREGARLINK(**in** *l*: link, **in** *c*: categoría, **in/out** *s*: sistema)

Pre \equiv {*s* =_{obs} *s*₀ \wedge $\neg(l \in \text{links}(s)) \wedge \text{está?}(c, \text{categorías}(s))$ }

Post \equiv {*s* =_{obs} 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* =_{obs} *s*₀ $\wedge l \in \text{links}(s) \wedge f \geq \text{fechaActual}(s)$ }

Post \equiv {*s* =_{obs} 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* =_{obs} 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*) =_{obs} linksOrdenadosPorAccesos(*ac*, *padre*)) \wedge 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* =_{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* =_{obs} Actual(*it*)}

Complejidad: $\Theta(1)$

Descripción: devuelve el link actual del iterador.

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

Pre \equiv {HayMás?(*it*)}

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

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` , `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.

3. Módulo Diccionario Trie(α)

Interfaz

parámetros formales

géneros α

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

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

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

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

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

Pendiente.

Operaciones del iterador

Pendiente.

Representación

Representación de diccionario trie

Pendiente.

Representación del iterador

Pendiente.

Algoritmos

Pendiente.