



DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2: Diseño

Primer cuatrimestre - 2016

Algoritmos y Estructuras de Datos II

Grupo XXXX

| Integrante | LU | Correo electrónico |
|-----------------------------|--------|--------------------------|
| BENZO, Mariano | 198/14 | marianobenzo@gmail.com |
| FARIAS, Mauro | 821/13 | farias.mauro@hotmail.com |
| GUTTMAN, Martin | 686/14 | haris@live.com.ar |
| MOSQUEIRA C., Edgardo Ramon | 808/13 | edgarcab666@hotmail.com |

| Instancia | Docente | Nota |
|-----------------|---------|------|
| Primera entrega | | |
| Segunda entrega | | |



Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ciudad Universitaria – Pabellón I (Planta Baja)

Intendente Güiraldes 2160 – C1428EGA

Ciudad Autónoma de Buenos Aires – Rep. Argentina

Tel/Fax: (+54 +11) 4576-3300

<http://www.exactas.uba.ar>

Índice

1 Tabla

1.1 Interfaz

se explica con TABLA

usa

géneros nat, dato, campo, tipo, registro, conjTrie, string, diccTrie(string, alfa), diccAVL

Operaciones

NOMBRE(in t : tab) $\longrightarrow res$: string

Pre $\equiv \{true\}$

Post $\equiv \{res =_{obs} nombre(t)\}$

Descripción: Devuelve el nombre de la tabla ingresada por parametro.

Complejidad: $O(1)$

Aliasing: Se retorna res por copia, por ser un tipo basico.

CLAVES(in t : tab) $\longrightarrow res$: itConjTrie(campo)

Pre $\equiv \{true\}$

Post $\equiv \{res =_{obs} claves(t)\}$

Descripción: Devuelve un conjunto de campos que son claves en la tabla ingresada por parametro.

Complejidad: $O(1)$

Aliasing: Se devuelve un iterador al conjunto claves por referencia.

INDICES(in t : tab) $\longrightarrow res$: itConjTrie(campo)

Pre $\equiv \{true\}$

Post $\equiv \{res =_{obs} indices(t)\}$

Descripción: Devuelve un conjunto de los indices de la tabla ingresada por parametro.

Complejidad: $O(calcular)$

Aliasing: Se devuelve res por referencia y no es modificable.

CAMPOS(in t : tab) $\longrightarrow res$: itConjTrie(campo)

Pre $\equiv \{true\}$

Post $\equiv \{res =_{obs} campos(t)\}$

Descripción: Devuelve un conjunto a los campos de la tabla ingresada por parametro.

Complejidad: $O(1)$

Aliasing: Se devuelve res por referencia.

TIPOCAMPO(in c : campo, in t : tab) $\longrightarrow res$: tipo

Pre $\equiv \{c \in campos(t)\}$

Post $\equiv \{res =_{obs} tipoCampo(t)\}$

Descripción: Devuelve el tipo del campo c en la tabla ingresada por parametro.

Complejidad: $O(1)$

Aliasing: Se devuelve res por referencia, no es modificable.

REGISTROS(in t : tab) $\longrightarrow res$: itConj(registro)

Pre $\equiv \{true\}$

Post $\equiv \{res =_{obs} registros(t)\}$

Descripción: Devuelve un conjunto a los registros de la tabla ingresada por parametro.

Complejidad: $O(L + \log(n))$

Aliasing: Se devuelve res referencia

CANTIDADDEACCESOS(**in** $t : \text{tab}$) $\longrightarrow res : \text{nat}$

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

Post $\equiv \{res =_{\text{obs}} \text{cantidadDeAccesos}(t)\}$

Descripción: Devuelve la cantidad de modificaciones de la tabla ingresada por parametro.

Complejidad: $O(1)$

Aliasing: Se devuelve res por copia.

NUEVATABLA(**in** $\text{nombre} : \text{string}$, $\text{in claves} : \text{conjTrie}(\text{campo})$, $\text{in columnas} : \text{registro}$) $\longrightarrow res : \text{tab}$

Pre $\equiv \{\neg \emptyset?(\text{claves}) \wedge \text{claves} \subseteq \text{campos}(\text{columnas})\}$

Post $\equiv \{res =_{\text{obs}} \text{nuevaTabla}(t)\}$

Descripción: Crea una tabla sin registros.

Complejidad: $O(\text{calcular})$

AGREGARREGISTRO(**in** $r : \text{registro}$, $\text{in } t : \text{tab}$)

Pre $\equiv \{t_0 = t \wedge \text{campos}(r) =_{\text{obs}} \text{campos}(t) \wedge \text{puedoInsertar?}(r, t)\}$

Post $\equiv \{\text{agregarRegistro}(r, t_0)\}$

Descripción: Agrega un registro a la tabla pasada por parametro.

Complejidad: $O(L + in)$

Aliasing: Agrega el registro r por referencia.

BORRARREGISTRO(**in** $\text{crit} : \text{registro}$, $\text{in } t : \text{tab}$)

Pre $\equiv \{t_0 = t \wedge \#(\text{campos}(r)) = 1 \wedge_L \text{dameUno}(\text{campos}(\text{crit})) \in \text{claves}(t)\}$

Post $\equiv \{\text{borrarRegistro}(r, t_0)\}$

Descripción: Borra los registros que cumplan el criterio pasado por parametro.

Complejidad: $O(L + in)$

INDEXAR(**in** $\text{crit} : \text{registro}$, $\text{in } t : \text{tab}$)

Pre $\equiv \{t_0 = t \wedge \text{puedeIndexar}(c, t)\}$

Post $\equiv \{\text{indexar}(c, t_0)\}$

Descripción: Borra los registros que cumplan el criterio pasado por parametro.

Complejidad: $O(L + in)$

PUEDOIINSERTAR?(**in** $r : \text{registro}$, $\text{in } t : \text{tab}$) $\longrightarrow res : \text{bool}$

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

Post $\equiv \{res =_{\text{obs}} \text{puedoInsertar?}(r, t)\}$

Descripción: Informa si el registro pasado por parametro no tiene valores repetidos con respecto a los registros existentes, para los campos clave en la tabla pasada por parametro.

Complejidad: $O(T * L + in)$

COMPATIBLE(**in** $r : \text{registro}$, $\text{in } t : \text{tab}$) $\longrightarrow res : \text{bool}$

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

Post $\equiv \{res =_{\text{obs}} \text{compatible}(r, t)\}$

Descripción: Informa si el registro pasado por parametro tiene correspondencia en los tipos de los campos de tabla pasada por parametro.

Complejidad: $O(1)$

MINIMO(**in** $c : \text{campo}$, $\text{in } t : \text{tab}$) $\longrightarrow res : \text{dato}$

Pre $\equiv \{\neg \emptyset?(\text{registro}(t)) \wedge c \in \text{indices}(t)\}$

Post $\equiv \{res =_{\text{obs}} \text{minimo}(c, t)\}$

Descripción: Retorna el minimo entre los valores de la tabla para el campo c.

Complejidad: $O(L + in)$

Aliasing: Retorna res por referencia.

MAXIMO(**in** $c : \text{campo}$, $\text{in } t : \text{tab}$) $\longrightarrow res : \text{dato}$

Pre $\equiv \{\neg \emptyset?(\text{registro}(t)) \wedge c \in \text{indices}(t)\}$

Post $\equiv \{res =_{\text{obs}} \text{maximo}(c, t)\}$

Descripción: Retorna el maximo entre los valores de la tabla para el campo c.

Complejidad: $O(L + in)$

Aliasing: Retorna res por referencia.

PUEDEINDEXAR(**in** $c : \text{campo}$, **in** $t : \text{tab}$) $\longrightarrow res : \text{bool}$

Pre $\equiv \{True\}$

Post $\equiv \{res =_{\text{obs}} \text{puedeIndexar}(c, t)\}$

Descripción: Informa si se puede crear un nuevo indice.

Complejidad: $O(L + in)$

COINCIDENCIAS(**in** $r : \text{registro}$, **in** $cj : \text{Conj}(\text{registro})$) $\longrightarrow res : \text{Conj}(\text{registro})$

Pre $\equiv \{True\}$

Post $\equiv \{res =_{\text{obs}} \text{coincidencias}(r, cj)\}$

Descripción: Compara el valor del registro con el conjunto de registros y retorna la interseccion.

Complejidad: $O(L + in)$

Aliasing: Retorna res por referencia.

HAYCOINCIDENCIA(**in** $r : \text{registro}$, **in** $cj1 : \text{ConjTrie}(\text{campo})$, **in** $cj2 : \text{Conj}(\text{registro})$) $\longrightarrow res : \text{bool}$

Pre $\equiv \{True\}$

Post $\equiv \{res =_{\text{obs}} \text{hayCoincidencia}(r, cj1, cj2)\}$

Descripción: Compara los valores del registro para los campos dados por parametro, con el conjunto de registros.

Complejidad: $O(L + in)$

COMBINARREGISTROS(**in** $c : \text{campo}$, **in** $cj1 : \text{Conj}(\text{registro})$, **in** $cj2 : \text{Conj}(\text{registro})$) $\longrightarrow res : \text{conj}(\text{registro})$

Pre $\equiv \{True\}$

Post $\equiv \{res =_{\text{obs}} \text{combinarRegistros}(c, cj1, cj2)\}$

Descripción: Combina los valores de los registros para el campo dado por parametro.

Complejidad: $O(L + in)$

Aliasing: Retorna res por copia.

DAMECOLUMNA(**in** $c : \text{campo}$, **in** $cj : \text{Conj}(\text{registro})$) $\longrightarrow res : \text{conj}(\text{dato})$

Pre $\equiv \{True\}$

Post $\equiv \{res =_{\text{obs}} \text{dameColumna}(c, cj1, cj2)\}$

Descripción: Reune en un conjunto los valores del campo pasado por parametro.

Complejidad: $O(T * L + in)$

Aliasing: Retorna res por referencia.

MISMOSTIPOS(**in** $r : \text{registro}$, **in** $t : \text{tab}$) $\longrightarrow res : \text{bool}$

Pre $\equiv \{\text{campos}(r) \subseteq \text{campos}(t)\}$

Post $\equiv \{res =_{\text{obs}} \text{mismosTipos}(r, t)\}$

Descripción: Compara los tipos correspondientes a los campos del registro y la tabla.

Complejidad: $O(1)$

1.2 Representación

se representa con *Tabla*

donde *tab* es $\text{tupla}(\text{Nombre} : \text{String},$
 $\text{Indices} : \text{DiccTrie}(\text{campo}, \text{Indice}),$
 $\text{Registros} : \text{Conj}(\text{Registro}),$
 $\text{Campos} : \text{DiccTrie}(\text{Campo}, \text{Tipo}),$
 $\# \text{Accesos} : \text{Nat})$
donde *Indice* es $\text{tupla}(\text{PorNat} : \text{DiccTrie}(\text{Dato} \text{ Conj}(\text{Registro})),$
 $\text{PorString} : \text{DiccTrie}(\text{Dato} \text{ Conj}(\text{Registro})))$

Invariante de representación

1. Para todos los registros de *r*, el tipo de los datos de las columnas de *r*, deben coincidir con los tipos de las columnas en *e.campos*.
2. Todas las columnas de *e.campos* y su tipo, deben coincidir con los campos y tipo de todos los registros de *e.registros*. Es decir no debe haber campos de mas.'
3. El nombre de la tabla que figura en *e.nombre*, es un string de longitud acotada.
4. Para todo registro *r* de *e.registros* y para todo campo *c* de *e.Indices.DiccClaves*, se debe cumplir que si tenemos $\text{valor} \leftarrow \text{Obtener}(r, c)$ y $\text{ind} \leftarrow \text{Obtener}(e.\text{Indices}, c)$. Al evaluar que $r \in \text{Obtener}(\text{ind}, \text{valor})$ y deben ser del mismo tipo.
5. Para todo campo *c*, que pertenece a *e.Indices.DiccClaves*, si tenemos que $\text{ind} \leftarrow \text{Obtener}(e.\text{Indices}, c)$, y para todo dato *d* perteneciente a *ind.DiccClaves* entonces $\text{Obtener}(\text{ind}, d)$ esta incluido o es igual a *e.registros*.
6. Para todo registro *r* perteneciente a *e.registros* *r.DiccClaves* es igual a *e.campos.DiccClaves*.
7. El valor de *e.#Accesos* debe ser la cantidad de registros agregados, la cantidad de registros borrados, mas la cantidad de indices creados.

Función de abstracción

$\text{Abs} : \widehat{\text{sistema}} s \longrightarrow \widehat{\text{CampusSeguro}} \quad \{\text{Rep}(s)\}$

$(\forall s : \widehat{\text{sistema}})$
 $\text{Abs}(s) \equiv cs : \widehat{\text{CampusSeguro}} \mid s.\text{campus} =_{\text{obs}} \text{campus}(cs) \wedge$
 $s.\text{estudiantes} =_{\text{obs}} \text{estudiantes}(cs) \wedge$
 $s.\text{hippies} =_{\text{obs}} \text{hippies}(cs) \wedge$
 $s.\text{agentes} =_{\text{obs}} \text{agentes}(cs) \wedge$
 $((\forall n : \text{nombre}) s.\text{hippies}.\text{definido}(n) \Rightarrow_L s.\text{hippies}.\text{obtener}(n) =_{\text{obs}} \text{posEstYHippie}(n, cs) \vee$
 $(\forall n : \text{nombre}) s.\text{estudiantes}.\text{definido}(n) \Rightarrow_L s.\text{estudiantes}.\text{obtener}(n) =_{\text{obs}} \text{posEstYHippie}(n, cs))$
 $(\forall pl : \text{placa}) s.\text{agentes}.\text{definido}(pl) \Rightarrow_L s.\text{estudiantes}.\text{obtener}(pl).\text{pos} =_{\text{obs}} \text{posAgente}(pl, cs))$
 $(\forall pl : \text{placa}) s.\text{agentes}.\text{definido}(pl) \Rightarrow_L s.\text{estudiantes}.\text{obtener}(pl).\text{cantSanciones} =_{\text{obs}} \text{cantSanciones}(pl, cs))$
 $(\forall pl : \text{placa}) s.\text{agentes}.\text{definido}(pl) \Rightarrow_L s.\text{estudiantes}.\text{obtener}(pl).\text{cantCapturas} =_{\text{obs}} \text{cantCapturas}(pl, cs))$

1.3 Algoritmos

| | |
|---|-----------------------|
| NOMBRE(in $t : \text{tab}$) $\longrightarrow res : \text{string}$ $res \leftarrow t.nombre$ | $O(1)$ |
| | <hr/> |
| | $O(1)$ |
| CLAVES(in $t : \text{tab}$) $\longrightarrow res : \text{ConjTrie}(\text{campo})$ $res \leftarrow t.Campos.ClavesDicc$ | $O(1)$ |
| | <hr/> |
| | $O(1)$ |
| INDICES(in $t : \text{tab}$) $\longrightarrow res : \text{ConjTrie}(\text{campo})$ $res \leftarrow t.ClavesDicc$ | $O(1)$ |
| | <hr/> |
| | $O(1)$ |
| CAMPOS(in $t : \text{tab}$) $\longrightarrow res : \text{ConjTrie}(\text{campo})$ $res \leftarrow t.Campos.ClavesDicc$ | $O(1)$ |
| | <hr/> |
| | $O(1)$ |
| TIPOCAMPO(in $c : \text{campo}$, $in t : \text{tab}$) $\longrightarrow res : \text{Tipo}$ $res \leftarrow \text{Significado}(t.Campos, c)$ | $O(1)$ |
| | <hr/> |
| | $O(1)$ |
| REGISTROS(in $t : \text{tab}$) $\longrightarrow res : \text{Conj}(\text{registro})$ $res \leftarrow t.registros$ | $\theta(L + \log(n))$ |
| | <hr/> |
| | $\theta(L + \log(n))$ |
| CANTDEACCESOS(in $t : \text{tab}$) $\longrightarrow res : \text{nat}$ $res \leftarrow t.cantDeAccesos$ | $\theta(1)$ |
| | <hr/> |
| | $\theta(1)$ |
| NUEVATABLA(in $nombre : \text{string}$, $in claves : \text{conjTrie}(\text{campo})$, $in columnas : \text{registro}$) $\longrightarrow res : \text{tab}$ $itcampos \leftarrow \text{crearItTrie}(\text{Campos}(\text{columnas}))$ $O(1)$ $res \leftarrow$ $< nombre \text{ Vacio}() \text{ Vacio}() \text{ Vacio}() \text{ } 0 >$ $O(1)$ while HaySiguiente(itcampos) do $O(1)$ Esto se debe a que # de campos a iterar es acotada. $valor \leftarrow \text{Significado}(r, \text{Siguiente}(itcampos))$ $O(1)$ DefinirRapido(res.Campos, Siguiente(itcampos), valor) $O(1)$ if Pertenece?(claves, Siguiente(itcampos)) then $O(1)$ $val \leftarrow \text{Significado}(r, \text{Siguiente}(itcampos))$ AgregarRapido(res.Campos.CamposClave, val) $O(1)$ end if Avanzar(itcampos) $O(1)$ end while | <hr/> |
| | $\theta(1)$ |
| AGREGARREGISTRO(in $r : \text{registro}$, $in t : \text{tab}$) $nuevo \leftarrow \text{AgregarRapido}(t.Registros, r)$ $\theta(1)$ $t.\#Accesos++$ $\theta(1)$ if Cardinal(t.Indices.ClavesDicc) ≥ 1 then $\theta(1)$ | |

| | |
|--|---|
| itInd \leftarrow crearItConjTrie(t.Indices.ClavesDice) | $\theta(1)$ |
| while HaySiguiente(itInd) do | $\theta(1)$ |
| indiceC \leftarrow Obtener(t.Indices, Siguiente(itInd)) | $\theta(1)$ |
| valorC \leftarrow Obtener(r, Siguiente(itInd)) | $\theta(1)$ |
| AgregarRapido(Obtener(indiceC, valorC), nuevo) | $\theta(1)$ |
| Avanzar(itInd) | $\theta(1)$ |
| end while | |
| end if | |
| <hr/> | |
| | $\theta(1)$ |
| BORRARREGISTRO(in crit : registro, in t : tab) | |
| c \leftarrow Siguiente(Campos(crit)) | $\theta(1)$ |
| valor \leftarrow Obtener(crit, c) | $\theta(1)$ |
| if Definido?(t.Indices, c) then | $\theta(1)$ |
| indiceC \leftarrow Obtener(t.Indices, c) | $\theta(1)$ |
| itcjr \leftarrow CrearItConj(Obtener(indiceC, valor)) | $\theta(1)$ |
| while HaySiguiente(itcjr) do | $\theta(\log(n))$ |
| EliminarSiguiente(Siguiente(itcjr)) | $\theta(1)$ |
| tiene sentido??? | |
| EliminarSiguiente(itcjr) | $\theta(1)$ |
| end while | |
| else | |
| cr \leftarrow Coincidencias(crit, t.registros) | $\theta(\text{Cardinal}(t.\text{registros}))$ |
| while HaySiguiente(cr) do | |
| EliminarSiguiente(Siguiente(cr)) | |
| tiene sentido??? | |
| EliminarSiguiente(cr) | |
| end while | |
| end if | |
| <hr/> | |
| | $\theta(\text{Calcular despues de consulta})$ |
| INDEXAR(in c : campo, in t : tab) | |
| if tipoCampo(c,t) then | |
| conjLog(registro) nuevo \leftarrow vacio() | |
| else | |
| conjTrie(registro) nuevo \leftarrow vacio() | |
| end if | |
| indC \leftarrow Siguiente(DefinirRapido(t.Indices, c, nuevo)) | |
| cr \leftarrow t.registros | |
| while HaySiguiente(cr) do | |
| valor \leftarrow Obtener(Siguiente(cr), c) | |
| if Definido?(indC, valor) then | |
| regviejos \leftarrow Obtener(indC, valor) | |
| AgregarRapido(regviejos, Siguiente(cr)) | |
| else | |
| DefinirRapido(indC, valor, Siguiente(cr)) | |
| end if | |
| Avanzar(cr) | |
| end while | |
| <hr/> | |
| | $\theta(1)$ |
| PUEDOINSERTAR?(in r : registro, in t : tab) \rightarrow res : bool | |

| | |
|--|---|
| $\text{res} \leftarrow \text{compatible}(r,t) \wedge \neg \text{hayCoincidencia}(r, r.\text{ClavesDicc}, \text{registros}(t))$ | $\theta(L+\log(n))$ |
| | <hr/> |
| | $\theta(L+\log(n))$ |
| $\text{COMPATIBLE}(\text{in } r : \text{registro}, \text{ in } t : \text{tab}) \longrightarrow \text{res} : \text{bool}$ | |
| $\text{res} \leftarrow \text{compatible}(r,t) \wedge_L \text{mismosTipos}(r,t)$ | $\theta(1)$ |
| | <hr/> |
| | $O(1)$ |
| $\text{PUEDEINDEXAR}(\text{in } c : \text{campo}, \text{ in } t : \text{tab}) \longrightarrow \text{res} : \text{bool}$ | |
| $\text{res} \leftarrow \text{Definido?}(t.\text{campos},c) \wedge_L \neg \text{Definido?}(t.\text{Indices},c) \wedge (\text{Cardinal}(t.\text{Indices}) \leq 1)$ | |
| | <hr/> |
| | $O(\text{calcular})$ |
| $\text{COMBINARREGISTROS}(\text{in } c : \text{campo}, \text{ in } cr1 : \text{Conj}(\text{registro}), \text{ in } cr2 : \text{Conj}(\text{registro})) \longrightarrow \text{res} : \text{Conj}(\text{registros})$ | |
| $\text{itcr1} \leftarrow \text{CrearItConjTrie}(cr1)$ | $\theta(1)$ |
| $\text{copiacr2} \leftarrow \text{Copiar}(cr2)$ | $\theta(\text{Cardinal}(cr2))$ |
| while $\text{HaySiguiente}(\text{itcr1})$ do | $\theta(\text{Cardinal}(cr1))$ |
| $\text{combinarTodos}(c, \text{Siguiente}(\text{itcr1}), \text{copiacr2})$ | $\theta(1)$ |
| $\text{Avanzar}(\text{itcr1});$ | $\theta(1)$ |
| end while | |
| $\text{res} \leftarrow \text{copiacr2};$ | $\theta(1)$ |
| | <hr/> |
| | $O(\text{Cardinal}(cr1))$ |
| $\text{HAYCOINCIDENCIA}(\text{in } r : \text{registro}, \text{ in } cc : \text{ConjTrie}(\text{campo}), \text{ in } cr : \text{Conj}(\text{registro})) \longrightarrow \text{res} : \text{bool}$ | |
| $\text{itcr} \leftarrow \text{CrearItConj}(cr);$ | $\theta(1)$ |
| $\text{res} \leftarrow \text{false};$ | $\theta(1)$ |
| while $\text{HaySiguiente}(\text{itcr})$ do | $\theta(\text{Cardinal}(cr))$ |
| $\text{res} \leftarrow \text{coincideAlguno}(r, cc, \text{Siguiente}(\text{itcr})) \vee \text{res};$ | $\theta(1)$ |
| $\text{Avanzar}(\text{itcr});$ | $\theta(1)$ |
| end while | |
| | <hr/> |
| | $O(\text{Cardinal}(cr))$ |
| $\text{COINCIDENCIAS}(\text{in } crit : \text{registro}, \text{ in } cr : \text{Conj}(\text{registro})) \longrightarrow \text{res} : \text{Conj}(\text{registro})$ | |
| $\text{res} \leftarrow \text{Vacio}();$ | $\theta(1)$ |
| $\text{itcr} \leftarrow \text{CrearItConj}(cr)$ | |
| while $\text{HaySiguiente}(cr)$ do | $\theta(\text{Cardinal}(cr))$ |
| if $\text{coincidenTodos}(crit, \text{campos}(crit), \text{Siguiente}(\text{itcr}))$ then | $\theta(1)$ |
| $\text{AgregarRapido}(\text{res}, \text{Siguiente}(\text{itcr}))$ | $\theta(1)$ |
| end if | |
| $\text{Avanzar}(\text{itcr});$ | $\theta(1)$ |
| end while | |
| | <hr/> |
| | $O(\text{Cardinal}(cr))$ |
| $\text{MINIMO}(\text{in } c : \text{campo}, \text{ in } t : \text{tab}) \longrightarrow \text{res} : \text{dato}$ | |
| $\text{res} \leftarrow \text{min}(\text{dameColumna}(c, t.\text{registros}))$ | $\theta(\text{Cardinal}(t.\text{registros}))$ |
| | <hr/> |
| | $O(\text{Cardinal}(t.\text{registros}))$ |
| $\text{MAXIMO}(\text{in } c : \text{campo}, \text{ in } t : \text{tab}) \longrightarrow \text{res} : \text{dato}$ | |
| $\text{res} \leftarrow \text{max}(\text{dameColumna}(c, t.\text{registros}))$ | $\theta(\text{Cardinal}(t.\text{registros}))$ |

| | |
|--|---|
| | <hr/> |
| | $O(\text{Cardinal}(t.\text{registros}))$ |
| DAMECOLUMNA(in $c : \text{campo}$, <i>in</i> $cr : \text{Conj}(\text{registro})$) $\longrightarrow res : \text{Conj}(\text{dato})$ | |
| $itcr \leftarrow \text{CrearItConj}(cr);$ | $\theta(1)$ |
| $res \leftarrow \text{vacio}();$ | $\theta(1)$ |
| while HaySiguiente($itcr$) do | $\theta(\text{Cardinal}(cr))$ |
| if $\neg \text{Pertenece}(res, \text{Siguiente}(itcr))$ then | $\theta(????)$ |
| $\text{AgregarRapido}(res, \text{Siguiente}(itcr))$ | |
| end if | |
| $\text{Avanzar}(itcr);$ | $\theta(1)$ |
| end while | |
| | <hr/> |
| | $O(\text{calcular})$ |
| MISMOSTIPOS(in $r : \text{registro}$, <i>in</i> $t : \text{tab}$) $\longrightarrow res : \text{bool}$ | |
| $res \leftarrow \text{True};$ | $\theta(1)$ |
| $itconjClaves \leftarrow \text{CrearItConj}(r.\text{ClavesDicc});$ | $\theta(1)$ |
| while HaySiguiente($itconjClaves$) do | $\theta(1)$ |
| $val1 \leftarrow \text{tipo?}(\text{Obtener}(r, \text{Siguiente}(itconjClaves)))$ | $\theta(\text{Cardinal}(t.\text{registros}))$ |
| $val2 \leftarrow \text{tipoCampo}(\text{Siguiente}(itconjClaves), t)$ | $\theta(1)$ |
| $res \leftarrow res \wedge val1 = val2$ | $\theta(1)$ |
| $\text{Avanzar}(cr);$ | $\theta(1)$ |
| end while | |
| | <hr/> |
| | $O(\text{calcular})$ |

1.4 Algoritmos operaciones auxiliares

2 Diccionario por Naturales

2.1 Interfaz

se explica con $\text{DICCIONARIO}(\kappa, \sigma)$

usa Bool

géneros $\text{diccNat}(\kappa, \sigma)$

Operaciones

$\text{VACIO}() \longrightarrow res : \text{diccNat}(\kappa, \sigma)$

Pre $\equiv \{true\}$

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

Descripción: Crea un nuevo diccionario

Complejidad: $O(1)$

$\text{DEFINIDO?}(\text{in } d : \text{diccNat}(\kappa, \sigma) \text{ in } n : \kappa) \longrightarrow res : \text{Bool}$

Pre $\equiv \{true\}$

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

Descripción: Indica si la clave esta definida

Complejidad: $O(m)$

DEFINIR(**in/out** $d : \text{diccNat}(\kappa, \sigma)$ *in* $n : \kappa$, *in* $s : \sigma$)

Pre $\equiv \{\neg \text{def?}(n, d) \wedge d = d_0\}$

Post $\equiv \{d =_{\text{obs}} \text{Definir}(n, s, d_0)\}$

Descripción: Se define s en el diccionario

Complejidad: $O(m)$

BORRAR(**in/out** $d : \text{diccNat}(\kappa, \sigma)$ *in* $n : \kappa$)

Pre $\equiv \{d =_{\text{obs}} d_0 \wedge \text{def?}(n, d)\}$

Post $\equiv \{d =_{\text{obs}} \text{Borrar}(n, d_0)\}$

Descripción: Elimina el elemento n

Complejidad: $O(m)$

SIGNIFICADO(**in** $d : \text{diccNat}(\kappa, \sigma)$ *in* $n : \kappa$) $\longrightarrow res : \sigma$

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

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

Descripción: Se retornan los significados

Complejidad: $O(m)$

CLAVES(**in** $d : \text{diccNat}(\kappa\sigma)$) $\longrightarrow res : \text{conjLog}(\kappa)$

Pre $\equiv \{true\}$

Post $\equiv \{res =_{\text{obs}} \text{claves}(d)\}$

Descripción: Se retornan el conjunto de claves del diccionario

Complejidad: $O(1)$

2.2 Representación

diccNat

se representa con raiz: $\text{puntero}(\text{estr}(\kappa \ \sigma))$

donde $\text{estr}(\kappa, \sigma)$ es tupla $\langle \text{clave} : \kappa,$
 $\text{significado} : \sigma,$
 $\text{hijoDer} : \text{puntero}(\text{estr}(\kappa\sigma)),$
 $\text{hijoIzq} : \text{puntero}(\text{estr}(\kappa\sigma)),$
 $\text{diccClaves} : \text{conjLog}(\kappa) \rangle$

Invariante de representación

$\text{Rep} : \widehat{\text{estr}} \longrightarrow \text{boolean}$

$(\forall e : \widehat{\text{estr}})$

$\text{Rep}(e) \equiv true \iff ((\forall n_1, n_2 : \text{estr}(\kappa, \sigma))(n_1 \in \text{arbol}(e) \wedge n_2 \in \text{arbol}(e) \Rightarrow_L$
 $(n_1.\text{clave} < n_2.\text{clave} \Rightarrow n_1 \in \text{arbol}(n_2.\text{hijoIzq})) \wedge (n_1.\text{clave} > n_2.\text{clave} \Rightarrow n_1 \in \text{arbol}(n_2.\text{hijoDer})) \wedge$
 $(\forall n_1, n_2 : \text{estr}(\kappa, \sigma))(n_1 \in \text{arbol}(e) \wedge n_2 \in \text{arbol}(e) \wedge n_1.\text{clave} \neq n_2.\text{clave} \Rightarrow_L$
 $(n_1.\text{hijoIzq} = n_2.\text{hijoIzq} \vee n_1.\text{hijoIzq} = n_2.\text{hijoDer} \Rightarrow n_1.\text{hijoIzq} = \text{NULL}) \wedge$
 $(n_1.\text{hijoDer} = n_2.\text{hijoIzq} \vee n_1.\text{hijoDer} = n_2.\text{hijoDer} \Rightarrow n_1.\text{hijoDer} = \text{NULL}) \wedge$
 $((\forall n : \text{estr}(\kappa, \sigma))(n \in \text{arbol}(e)) \Rightarrow_L \text{Pertenece?}(e.\text{diccClaves}, n.\text{clave}) \wedge$
 $(\forall k : \kappa)(k \in e.\text{diccClaves} \Rightarrow (\exists n : \text{estr}(\kappa, \sigma))(n \in \text{arbol}(e) \wedge n.\text{clave} = k)))$

1. Para todo hijoDer de un estr, si no es NULL, su clave es mayor a la clave de su padre.
2. Para todo hijoIzq de un estr, si no es NULL, su clave es menor a la clave de su padre.
3. No hay ciclos, ni nodos con dos padres.

4. Todos las claves de los elementos del arbol estan en diccClaves y viceversa.

Función de abstracción

$Abs : \widehat{diccNat}(\kappa, \sigma) \rightarrow \widehat{dicc}(\kappa, \sigma) \quad \{Rep(d)\}$
 $(\forall d : \widehat{diccNat}(\kappa, \sigma))$
 $Abs(d) \equiv c : \widehat{dicc}(\kappa, \sigma) \mid ((\forall k : \kappa)(k \in claves(c) \Rightarrow$
 $(\exists n : estr(\kappa, \sigma))(n \in arbol(d) \wedge n.clave = k) \wedge (k \in d.diccClaves)) \wedge$
 $((\forall k : \kappa)(k \in d.diccClaves) \Rightarrow k \in claves(c)) \wedge$
 $((\forall n : estr(\kappa, \sigma))(n \in arbol(d) \Rightarrow n.clave \in claves(c)))) \wedge_L$
 $((\forall n : estr(\kappa, \sigma))(n \in arbol(d) \Rightarrow obtener(c, n.clave) =_{obs} n.significado))$
 $arbol : puntero(estr(\kappa, \sigma)) \leftarrow conj(puntero(estr(\kappa, \sigma)))$
 $arbol(n) \equiv$
if $n.hijoIzq \neq null \wedge n.hijoDer \neq null$ **then**
 $Ag(n, arbol(n.hijoIzq) \cup arbol(n.hijoDer))$
else
if $n.hijoIzq \neq null$ **then**
 $Ag(n, arbol(n.hijoIzq))$
else
if $n.hijoDer \neq null$ **then**
 $Ag(n, arbol(n.hijoDer))$
else
 $Ag(n, \emptyset)$
end if
end if
end if

2.3 Algoritmos

| | |
|--|--|
| $IVACIO() \rightarrow res : \mathbf{estr}$ $res \leftarrow NULL$ | <hr/> $O(1)$ |
| $IDEFINIR(\mathbf{in/out} \ d : \mathbf{estr}, \ \mathbf{in} \ n : \kappa, \ \mathbf{in} \ s : \sigma)$ if $d == NULL$ then $res \leftarrow \langle n, s, NULL, NULL \rangle$ end if if $d \neq NULL \wedge n < d.clave$ then $d.hijoIzq \leftarrow iDefinir(d.hijoIzq, n, s)$ end if if $d \neq NULL \wedge n > d.clave$ then $d.hijoDer \leftarrow iDefinir(d.hijoDer, n, s)$ end if $AgregarRapido(d.diccClaves, n)$ | $O(1)$ $O(1)$ $O(\log(m))$ $O(m)$ $O(m)$ <hr/> $O(m)$ |
| $IBORRAR(\mathbf{in/out} \ d : \mathbf{estr}, \ \mathbf{in} \ n : \kappa)$ if $d == NULL$ then $FinFuncion$ | $O(1)$ |

| | |
|--|--------|
| else if $n > d.clave$ then | |
| $d.hijoDer \leftarrow iBorrar(d.hijoDer, n)$ | $O(m)$ |
| else if $n < d.clave$ then | |
| $d.hijoIzq \leftarrow iBorrar(d.hijoIzq, n)$ | $O(m)$ |
| else if $d.hijoIzq == NULL \wedge d.hijoDer == NULL$ then | |
| $Borrar(d)$ | $O(1)$ |
| $d \leftarrow NULL$ | $O(1)$ |
| else if $d.hijoIzq == NULL$ then | |
| $aux \leftarrow d.hijoDer$ | $O(1)$ |
| while $aux.hijoIzq \neq NULL$ do | $O(m)$ |
| $aux \leftarrow aux.hijoIzq$ | $O(1)$ |
| end while | |
| $d.hijoDer \leftarrow iBorrar(aux.clave, d.hijoDer)$ | |
| $d.clave \leftarrow aux.clave$ | $O(1)$ |
| $d.significado \leftarrow aux.significado$ | $O(1)$ |
| else | |
| $aux \leftarrow d.hijoIzq$ | $O(1)$ |
| while $aux.hijoDer \neq NULL$ do | $O(m)$ |
| $aux \leftarrow aux.hijoDer$ | $O(1)$ |
| end while | |
| $d.hijoIzq \leftarrow iBorrar(aux.clave, d.hijoIzq)$ | |
| $d.clave \leftarrow aux.clave$ | $O(1)$ |
| $d.significado \leftarrow aux.significado$ | $O(1)$ |
| end if | |
| $Eliminar(d.diccClaves, n)$ | $O(m)$ |
| | <hr/> |
| | $O(m)$ |
| $iSIGNIFICADO(in/out\ d : \mathbf{estr},\ in\ n : \kappa) \longrightarrow res : \sigma$ | |
| $nodoActual \leftarrow d$ | $O(1)$ |
| while $\neg(nodoActual == NULL) \wedge \neg res$ do | $O(m)$ |
| if $nodoActual.clave == n$ then | |
| $res \leftarrow nodoActual.significado$ | $O(1)$ |
| else | |
| if $c < nodoActual.clave$ then | |
| $nodoActual \leftarrow nodoActual.hijoIzq$ | $O(1)$ |
| else | |
| $nodoActual \leftarrow nodoActual.hijoDer$ | $O(1)$ |
| end if | |
| end if | |
| end while | |
| | <hr/> |
| | $O(m)$ |
| $iDEFINIDO?(in/out\ d : \mathbf{estr},\ in\ n : \kappa) \longrightarrow res : \mathbf{bool}$ | |
| $nodoActual \leftarrow d$ | $O(1)$ |
| $res \leftarrow FALSE$ | $O(1)$ |
| while $\neg(nodoActual == NULL) \wedge \neg res$ do | $O(m)$ |
| if $nodoActual.clave == n$ then | |
| $res \leftarrow TRUE$ | $O(1)$ |
| else | |
| if $c < nodoActual.clave$ then | |
| $nodoActual \leftarrow nodoActual.hijoIzq$ | $O(1)$ |
| else | |

```

        nodoActual ← nodoActual.hijoDer
    end if
end if
end while

```

$O(1)$

$O(m)$

m: En peor caso es igual a la cantidad de elementos del arbol. En promedio es $\log(\text{cantidad de elementos del arbol})$.

3 Conjunto en Promedio Logaritmico

3.1 Interfaz

se explica con $\text{CONJ}(\alpha)$

usa

géneros $\text{conjLog}(\alpha)$

Operaciones

$\text{VACIO}() \rightarrow res : \text{conjLog}(\alpha)$

Pre $\equiv \{true\}$

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

Descripción: Crea un conjunto vacio

Complejidad: $O(1)$

$\text{PERTENECE?}(\text{in } c : \text{conjLog}(\alpha), \text{ in } a : \alpha) \rightarrow res : \text{Bool}$

Pre $\equiv \{true\}$

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

Descripción: Indica si el elemento a esta en el conjunto

Complejidad: $O(m)$

$\text{AGREGAR}(\text{in/out } c : \text{conjLog}(\alpha), \text{ in } a : \alpha)$

Pre $\equiv \{c =_{\text{obs}} c_0\}$

Post $\equiv \{c =_{\text{obs}} \text{Ag}(a, c_0)\}$

Descripción: Si a no pertenece a c, entonces lo agrega

Complejidad: $O(m)$

$\text{AGREGARRAPIDO}(\text{in/out } c : \text{conjLog}(\alpha), \text{ in } a : \alpha)$

Pre $\equiv \{c =_{\text{obs}} c_0 \wedge a \notin c\}$

Post $\equiv \{c =_{\text{obs}} \text{Ag}(a, c_0)\}$

Descripción: Se agrega a en c

Complejidad: $O(m)$

$\text{ELIMINAR}(\text{in/out } c : \text{conjLog}(\alpha), \text{ in } a : \alpha)$

Pre $\equiv \{true\}$

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

Descripción: Elimina el elemento a

Complejidad: $O(m)$

3.2 Representación

conjLog

se representa con puntero($\text{estr}(\alpha)$)

donde $\text{estr}(\alpha)$ es tupla $\langle \text{dato} : \alpha,$
 $\text{hijoDer} : \text{puntero}(\text{estr}(\alpha)),$
 $\text{hijoIzq} : \text{puntero}(\text{estr}(\alpha)) \rangle$

Invariante de representación

$\text{Rep} : \widehat{\text{estr}} \rightarrow \text{boolean}$

$(\forall e : \widehat{\text{estr}})$

$\text{Rep}(e) \equiv \text{true} \iff ((\forall n_1, n_2 : \text{estr}(\alpha))(n_1 \in \text{arbol}(e) \wedge n_2 \in \text{arbol}(e) \Rightarrow_{\text{L}} (n_1.\text{dato} < n_2.\text{dato} \Rightarrow n_1 \in \text{arbol}(n_2.\text{hijoIzq})) \wedge (n_1.\text{dato} > n_2.\text{dato} \Rightarrow n_1 \in \text{arbol}(n_2.\text{hijoDer}))) \wedge (\forall n_1, n_2 : \text{estr}(\alpha))(n_1 \in \text{arbol}(e) \wedge n_2 \in \text{arbol}(e) \wedge n_1.\text{dato} \neq n_2.\text{dato} \Rightarrow_{\text{L}} (n_1.\text{hijoIzq} = n_2.\text{hijoIzq} \vee n_1.\text{hijoIzq} = n_2.\text{hijoDer} \Rightarrow n_1.\text{hijoIzq} = \text{NULL})) \wedge (n_1.\text{hijoDer} = n_2.\text{hijoIzq} \vee n_1.\text{hijoDer} = n_2.\text{hijoDer} \Rightarrow n_1.\text{hijoDer} = \text{NULL}))$

1. Para todo hijoDer de un estr , si no es NULL , su dato es mayor a el dato de su padre.
2. Para todo hijoIzq de un estr , si no es NULL , su dato es menor a el dato de su padre.
3. No hay ciclos, ni nodos con dos padres.

Función de abstracción

$\text{Abs} : \widehat{\text{conjLog}}(\alpha) \text{ } cl \rightarrow \widehat{\text{conj}}(\alpha) \quad \{\text{Rep}(cl)\}$

$(\forall cl : \widehat{\text{conjLog}}(\alpha))$

$\text{Abs}(cl) \equiv c : \widehat{\text{conj}}(\alpha) \mid ((\forall a : \alpha)(a \in c \Rightarrow \text{Pertenece?}(cl, a)) \wedge ((\forall n : \text{estr}(\alpha))(n \in \text{arbol}(cl) \Rightarrow n.\text{dato} \in c)))$

$\text{arbol} : \text{puntero}(\text{estr}(\alpha)) \leftarrow \text{conj}(\text{puntero}(\text{estr}(\alpha)))$

$\text{arbol}(n) \equiv$

```

if  $n.\text{hijoIzq} \neq \text{null} \wedge n.\text{hijoDer} \neq \text{null}$  then
   $\text{Ag}(n, \text{arbol}(n.\text{hijoIzq}) \cup \text{arbol}(n.\text{hijoDer}))$ 
else
  if  $n.\text{hijoIzq} \neq \text{null}$  then
     $\text{Ag}(n, \text{arbol}(n.\text{hijoIzq}))$ 
  else
    if  $n.\text{hijoDer} \neq \text{null}$  then
       $\text{Ag}(n, \text{arbol}(n.\text{hijoDer}))$ 
    else
       $\text{Ag}(n, \emptyset)$ 
    end if
  end if
end if

```

3.3 Algoritmos

IVACIO() $\rightarrow res : \mathbf{estr}$
 $res \leftarrow NULL$

O(1)

IAGREGAR(**in/out** $c : \mathbf{estr}$, $in\ a : \alpha$)
if $\neg iPertenece?(c, a)$ **then**
 if $c == NULL$ **then**
 $res \leftarrow \langle a, NULL, NULL \rangle$
 end if
 if $c \neq NULL \wedge a < c.dato$ **then**
 $c.hijoIzq \leftarrow iAgregarRapido(c.hijoIzq, a)$
 end if
 if $c \neq NULL \wedge a > c.dato$ **then**
 $c.hijoDer \leftarrow iAgregarRapido(c.hijoDer, a)$
 end if
end if

O(1)

O(m)

O(m)

O(m)

IAGREGARRAPIDO(**in/out** $c : \mathbf{estr}$, $in\ a : \alpha$)
if $c == NULL$ **then**
 $res \leftarrow \langle a, NULL, NULL \rangle$
end if
if $c \neq NULL \wedge a < c.dato$ **then**
 $c.hijoIzq \leftarrow iAgregarRapido(c.hijoIzq, a)$
end if
if $c \neq NULL \wedge a > c.dato$ **then**
 $c.hijoDer \leftarrow iAgregarRapido(c.hijoDer, a)$
end if

O(1)

O(m)

O(m)

O(m)

IELIMINAR(**in/out** $c : \mathbf{estr}$, $in\ a : \alpha$)
if $c == NULL$ **then**
 $FinFuncion$
else if $a > c.dato$ **then**
 $c.hijoDer \leftarrow iEliminar(c.hijoDer, a)$
else if $a < c.dato$ **then**
 $c.hijoIzq \leftarrow iEliminar(c.hijoIzq, a)$
else if $c.hijoIzq == NULL \wedge c.hijoDer == NULL$ **then**
 $Borrar(c)$
 $c \leftarrow NULL$
else if $c.hijoIzq == NULL$ **then**
 $aux \leftarrow c.hijoDer$
 while $aux.hijoIzq \neq NULL$ **do**
 $aux \leftarrow aux.hijoIzq$
 end while
 $c.hijoDer \leftarrow iEliminar(aux.dato, c.hijoDer)$
 $c.dato \leftarrow aux.dato$
else
 $aux \leftarrow c.hijoIzq$
 while $aux.hijoDer \neq NULL$ **do**

O(1)

O(m)

O(m)

O(1)

O(1)

O(1)

O(m)

O(1)

O(1)

O(1)

O(m)

| | |
|---|--------|
| $aux \leftarrow aux.hijoDer$ | $O(1)$ |
| end while | |
| $c.hijoIzq \leftarrow iEliminar(aux.dato, c.hijoIzq)$ | |
| $c.dato \leftarrow aux.dato$ | $O(1)$ |
| end if | |
| <hr/> | |
| | $O(m)$ |
| IPERTENECE? (in/out $c : \text{estr}$, <i>in</i> $a : \alpha$) $\longrightarrow res : \text{bool}$ | |
| $nodoActual \leftarrow c$ | $O(1)$ |
| $res \leftarrow FALSE$ | $O(1)$ |
| while $\neg(nodoActual == NULL) \wedge \neg res$ do | $O(m)$ |
| if $nodoActual.dato == a$ then | |
| $res \leftarrow TRUE$ | $O(1)$ |
| else | |
| if $a < nodoActual.dato$ then | |
| $nodoActual \leftarrow nodoActual.hijoIzq$ | $O(1)$ |
| else | |
| $nodoActual \leftarrow nodoActual.hijoDer$ | $O(1)$ |
| end if | |
| end if | |
| end while | |
| <hr/> | |
| | $O(m)$ |

m: En peor caso es igual a la cantidad de elementos del arbol. En promedio es $\log(\text{cantidad de elementos del arbol})$.

4 Modulo Diccionario Lexicografico($String, \sigma$)

4.1 Interfaz

se explica con DICCIONARIO($STRING, \sigma$)
usa Bool, Nat
géneros $diccString(String, \sigma)$

4.2 Operaciones basicas de diccionario

$VACIO() \longrightarrow res : diccString(String, \sigma)$
Pre $\equiv \{true\}$
Post $\equiv \{res =_{\text{obs}} vacio()\}$
Descripción: Crea un nuevo diccionario
Complejidad: $O(1)$

$DEFINIDO?(in\ d : diccString(String, \sigma)\ in\ n : String) \longrightarrow res : Bool$
Pre $\equiv \{true\}$
Post $\equiv \{res =_{\text{obs}} def?(n, d)\}$
Descripción: Indica si la clave esta definida
Complejidad: $O(Longitud(n))$

$DEFINIR(in/out\ d : diccString(String, \sigma)\ in\ n : String, in\ s : \sigma)$
Pre $\equiv \{\neg def?(n, d) \wedge d = d_0\}$

Post $\equiv \{d =_{\text{obs}} \text{Definir}(n, s, d_0)\}$
Descripción: Se define s en el diccionario
Complejidad: $O(\text{Longitud}(n) + \text{copy}(s))$
Aliasing: s se define por copia

BORRAR(in/out $d : \text{diccString}(\text{String}, \sigma)$ in $n : \text{String}$)
Pre $\equiv \{d =_{\text{obs}} d_0 \wedge \text{def?}(n, d)\}$
Post $\equiv \{d =_{\text{obs}} \text{Borrar}(n, d_0)\}$
Descripción: Elimina el elemento n
Complejidad: $O(\text{Longitud}(n))$

SIGNIFICADO(in $d : \text{diccString}(\text{String}, \sigma)$ in $n : \text{String}$) $\longrightarrow res : \sigma$
Pre $\equiv \{\text{def?}(n, d)\}$
Post $\equiv \{res =_{\text{obs}} \text{obtener}(n, d)\}$
Descripción: Se retorna el significado de n
Complejidad: $O(\text{Longitud}(n))$

4.3 Operaciones del iterador

El iterador que presentamos permite modificar el diccionario recorrido, eliminando elementos. Sin embargo, cuando el diccionario es no modificable, no se pueden utilizar las funciones de eliminacion. Ademas, las claves de los elementos iterados no pueden modificarse nunca, por cuestiones de implementacion. Cuando d es modificable, decimos que it es modificable.

Para simplificar la notacion, vamos a utilizar clave y significado en lugar de Π_1 y Π_2 cuando utilizemos una tupla (String, σ) . **CREARIT**(in $d : \text{diccString}(\text{String}, \sigma)$ in $n : \text{String}$) $\longrightarrow res : \text{itDiccString}(\text{String}, \sigma)$

Pre $\equiv \{\text{true}\}$
Post $\equiv \{\text{alias}(\text{esPermutacion}(\text{SecuSuby}(res), d)) \wedge \text{vacía?}(\text{Anteriores}(res))\}$
Descripción: crea un iterador bidireccional del diccionario, que apunta al primer elemento del mismo en orden lexicografico.
Complejidad: $O(CL * \text{long}(k))$ Donde CL es la cantidad de claves de d y k la palabra mas larga de d
Aliasing: hay aliasing entre los significados en el iterador y los del diccionario

HAYSIGUIENTE(in $it : \text{itDiccString}(\text{String}, \sigma)$ in $n : \text{String}$) $\longrightarrow res : \text{bool}$
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \text{haySiguiente?}(it)\}$
Descripción: devuelve **true** si y solo si en el iterador todavia quedan elementos para avanzar.
Complejidad: $O(1)$

HAYANTERIOR(in $it : \text{itDiccString}(\text{String}, \sigma)$ in $n : \text{String}$) $\longrightarrow res : \text{bool}$
Pre $\equiv \{\text{true}\}$
Post $\equiv \{res =_{\text{obs}} \text{hayAnterior?}(it)\}$
Descripción: devuelve **true** si y solo si en el iterador todavia quedan elementos para retroceder.
Complejidad: $O(1)$

SIGUIENTE(in $it : \text{itDiccString}(\text{String}, \sigma)$ in $n : \text{String}$) $\longrightarrow res : \text{tupla}(\text{String}, \sigma)$
Pre $\equiv \{\text{HaySiguiente?}(it)\}$
Post $\equiv \{\text{alias}(res =_{\text{obs}} \text{Siguiente}(it))\}$
Descripción: devuelve el elemento siguiente del iterador.
Complejidad: $O(1)$

Aliasing: $res.\text{significado}$ es modificable si y solo si it es modificable. En cambio, $res.\text{clave}$ no es modificable.

SIGUIENTEC clave(in $it : \text{itDiccString}(\text{String}, \sigma)$ in $n : \text{String}$) $\longrightarrow res : \text{String}$

Pre $\equiv \{\text{HaySiguiente?}(it)\}$

Post $\equiv \{\text{alias}(res =_{\text{obs}} \text{Siguiente}(it).\text{clave})\}$

Descripción: devuelve la clave del elemento siguiente del iterador.

Complejidad: $O(1)$

Aliasing: res no es modificable.

SIGUIENTESIGNIFICADO(**in** $it : \text{itDiccString}(\text{String}, \sigma)$ **in** $n : \text{String}$) $\longrightarrow res : \sigma$

Pre $\equiv \{\text{HaySiguiente?}(it)\}$

Post $\equiv \{\text{alias}(res =_{\text{obs}} \text{Siguiente}(it).\text{significado})\}$

Descripción: devuelve el significado del elemento siguiente del iterador.

Complejidad: $O(1)$

Aliasing: res es modificable si y solo si it es modificable.

ANTERIOR(**in** $it : \text{itDiccString}(\text{String}, \sigma)$ **in** $n : \text{String}$) $\longrightarrow \text{tupla}(\text{clave} : \text{String}, \text{significado} : \sigma)$

Pre $\equiv \{\text{HayAnterior?}(it)\}$

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

Descripción: devuelve el elemento anterior del iterador.

Complejidad: $O(1)$

Aliasing: $res.\text{significado}$ es modificable si y solo si it es modificable. En cambio, $res.\text{clave}$ no es modificable.

ANTERIORCLAVE(**in** $it : \text{itDiccString}(\text{String}, \sigma)$ **in** $n : \text{String}$) $\longrightarrow res : \text{String}$

Pre $\equiv \{\text{HayAnterior?}(it)\}$

Post $\equiv \{\text{alias}(res =_{\text{obs}} \text{Anterior}(it).\text{clave})\}$

Descripción: devuelve la clave del elemento anterior del iterador.

Complejidad: $O(1)$

Aliasing: res no es modificable.

ANTERIORSIGNIFICADO(**in** $it : \text{itDiccString}(\text{String}, \sigma)$ **in** $n : \text{String}$) $\longrightarrow res : \sigma$

Pre $\equiv \{\text{HayAnterior?}(it)\}$

Post $\equiv \{\text{alias}(res =_{\text{obs}} \text{Anterior}(it).\text{significado})\}$

Descripción: devuelve el significado del elemento anterior del iterador.

Complejidad: $O(1)$

Aliasing: res es modificable si y solo si it es modificable.

AVANZAR(**inout** $it : \text{itDiccString}(\text{String}, \sigma)$ **in** $n : \text{String}$)

Pre $\equiv \{it = it_0 \wedge \text{HaySiguiente?}(it)\}$

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

Descripción: avanza a la posición siguiente del iterador.

Complejidad: $O(1)$

RETROCEDER(**inout** $it : \text{itDiccString}(\text{String}, \sigma)$ **in** $n : \text{String}$)

Pre $\equiv \{it = it_0 \wedge \text{HayAnterior?}(it)\}$

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

Descripción: retrocede a la posición anterior del iterador.

Complejidad: $O(1)$

4.4 Representacion

`diccString`

se representa con `dLex`)

donde `dLex` es `tupla⟨raiz : puntero(nodo)⟩`

nodo

se representa con enodo)

donde enodo es tupla $\langle \text{dato} : \sigma, \text{esSig?} : \text{Bool}, \text{continuaciones} : \text{puntero}(\text{char}) \sqsubset 256 \sqsupset \rangle$

Invariante de representación

1. Todo Nodo, si sus continuaciones son todos punteros a null, es porque es significado.
2. No hay ciclos, ni nodos con dos padres.

Función de abstracción

Respresentación del iterador

El iterador del diccionario lo recorre en orden lexicográfico. Los significados están por referencia.
 $\text{itDiccString}(\text{String}, \sigma)$

se representa con itdLex)

donde dLex es tupla $\langle \text{claves} : \text{Lista}(\text{String}), \text{significados} : \text{Lista}(\sigma) \rangle$

4.5 Algoritmos

$\text{IVACIO}() \rightarrow \text{res} : \text{diccString}$

$\text{res.raiz} \leftarrow \text{NULL}$

$O(1)$

$O(1)$

$\text{IDEFINIR}(\text{in/out } d : \text{diccString}, \text{ in } n : \text{String}, \text{ in } s : \sigma)$

$\text{aux} \leftarrow d.\text{raiz}$

$O(1)$

$i \leftarrow 0$

$O(1)$

while $i < \text{Longitud}(n)$ **do**

$O(\text{Longitud}(n))$

if $\text{aux} == \text{NULL}$ **then**

$n\text{Nodo}.\text{esSig?} \leftarrow \text{false}$

$O(1)$

$j \leftarrow 0$

while $j < 256$ **do**

$O(256) = O(1)$

$n\text{Nodo}.\text{continuaciones}[j] \leftarrow \text{NULL}$

$O(1)$

end while

$\text{aux} \leftarrow \&n\text{Nodo}$

$O(1)$

end if

$\text{aux} \leftarrow \text{aux}.\text{continuaciones}[\text{ord}(n[i])]$

$O(1)$

end while

$\text{aux} * .\text{esSig?} \leftarrow \text{True}$

$O(1)$

$\text{aux} * .\text{dato} \leftarrow s$

$O(\text{copy}(s))$ el costo de la copia de σ

$O(\text{Longitud}(n) + \text{copy}(s))$

$\text{IDEFINIDO?}(\text{in/out } d : \text{diccString}, \text{ in } n : \text{String}) \rightarrow \text{res} : \text{bool}$

| | |
|--|----------------------------|
| $aux \leftarrow d.raiz$ | $O(1)$ |
| $i \leftarrow 0$ | $O(1)$ |
| while $i < (Longitud(n) - 1) \wedge aux \neq NULL$ do | $O(Longitud(n))$ |
| $aux \leftarrow aux.continuaciones[ord(n[i])]$ | $O(1)$ |
| end while | |
| if $aux \neq NULL$ then | |
| $res \leftarrow aux.esSig?$ | $O(1)$ |
| else | |
| $res \leftarrow false$ | $O(1)$ |
| end if | |
| <hr/> | |
| | $O(Longitud(n))$ |
| ISIGNIFICADO (in $d : diccString$, in $n : String$) $\longrightarrow res : \sigma$ | |
| $aux \leftarrow d.raiz$ | $O(1)$ |
| $i \leftarrow 0$ | $O(1)$ |
| while $i < Longitud(n)$ do | $O(Longitud(n))$ |
| $aux \leftarrow aux.continuaciones[ord(n[i])]$ | $O(1)$ |
| $i \leftarrow i + 1$ | |
| end while | |
| $res \leftarrow aux.dato$ | $O(1)$ (es una referencia) |
| <hr/> | |
| | $O(Longitud(n))$ |
| IBORRAR (in/out $d : diccString$, in $n : String$) | |
| $aux \leftarrow d.raiz$ | $O(1)$ |
| $i \leftarrow 0$ | $O(1)$ |
| $pila(puntero(nodo))p \leftarrow Vacia()$ | $O(1)$ |
| while $i < Longitud(n)$ do | $O(Longitud(n))$ |
| $Apilar(p, aux)$ | $O(1)$ |
| $aux \leftarrow aux.continuaciones[ord(n[i])]$ | $O(1)$ |
| $i \leftarrow i + 1$ | |
| end while | |
| $aux.esSig? \leftarrow false$ | |
| $i \leftarrow i - 1$ | |
| $j \leftarrow 0$ | |
| while $aux.continuaciones[j] = NULL \wedge j < 256$ do | $O(256)=O(1)$ |
| $j \leftarrow j + 1$ | |
| end while | |
| if $j < 256$ then | |
| $p \leftarrow Vacia()$ | |
| end if | |
| while $\neg EsVacía?(p)$ do | |
| $j \leftarrow 0$ | |
| $Tope(p).continuaciones[ord(n[i])] \leftarrow NULL$ | |
| while $Tope(p).continuaciones[j] = NULL \wedge j < 256$ do | $O(256)=O(1)$ |
| $j \leftarrow j + 1$ | |
| end while | |
| if $j < 256$ then | |
| $p \leftarrow Vacía()$ | |
| else | |
| $Desapilar(p)$ | |
| $i \leftarrow i - 1$ | |
| end if | |

end while

$O(\text{Longitud}(n))$

ICREARIT(in/out $d : \text{diccString}$, in $n : \text{String}$) $\longrightarrow res : \text{itDiccString}$

$lista(\text{String})cs \leftarrow \text{Vacía}()$

$lista(\sigma)ss \leftarrow \text{Vacía}()$

$\text{String}n \leftarrow \text{Vacío}()$

$auxXrIt(cs, ss, n, d.raiz)$

$res.claves \leftarrow cs$

$res.significados \leftarrow ss$

$O(\text{Longitud}(k) * \text{tam}(d))$

$O(\text{Longitud}(k) * \text{tam}(d))$ Donde k es la palabra

5 Modulo Conjunto Lexicografico

5.1 Interfaz

se explica con $\text{CONJUNTO}(\text{STRING})$

usa Bool, Nat, String

géneros $\quad \quad \text{conjString}$

Operaciones básicas

$\text{VACIO}() \longrightarrow res : \text{diccString}(\text{String}, \sigma)$

Pre $\equiv \{true\}$

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

Descripción: Crea un nuevo conjunto

Complejidad: $O(1)$

$\text{PERTENECE?}(\text{in } d : \text{conjString} \text{ in } n : \text{String}) \longrightarrow res : \text{Bool}$

Pre $\equiv \{true\}$

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

Descripción: Indica si el elemento pertenece al conjunto

Complejidad: $O(\text{Longitud}(n))$

$\text{CARDINAL}(\text{in } d : \text{conjString}) \longrightarrow res : \text{Nat}$

Pre $\equiv \{true\}$

Post $\equiv \{res =_{\text{obs}} \#d\}$

Descripción: Indica el cardinal del conjunto

Complejidad: $O(\text{Longitud}(n))$

$\text{AGREGAR}(\text{in/out } d : \text{conjString} \text{ in } n : \text{String})$

Pre $\equiv \{d = d_0\}$

Post $\equiv \{d =_{\text{obs}} \text{Ag}(n, d_0)\}$

Descripción: Agrega el elemento n al conjunto

Complejidad: $O(\text{Longitud}(n))$

Aliasing: s se define por copia

$\text{BORRAR}(\text{in/out } d : \text{conjString} \text{ in } n : \text{String})$

Pre $\equiv \{d =_{\text{obs}} d_0 \wedge n \in d\}$

Post $\equiv \{d =_{\text{obs}} d_0 - \{n\}\}$

Descripción: Elimina el elemento n

Complejidad: $O(\text{Longitud}(n))$

Trabajo Práctico 2: Diseño Operaciones del iterador

El iterador que presentamos permite modificar el diccionario recorrido.

Para simplificar la notación, vamos a utilizar clave y significado en lugar de Π_1 y Π_2 cuando utilicemos una tupla $(String, \sigma)$. $\text{CREARIT}(\text{in } d : \text{conjString in } n : \text{String}) \longrightarrow res : \text{itDiccString}(String, \sigma)$

Pre $\equiv \{true\}$

Post $\equiv \{\text{alias}(\text{esPermutacion}(\text{SecuSuby}(res), d)) \wedge \text{vacía?}(\text{Anteriores}(res))\}$

Descripción: crea un iterador bidireccional del diccionario, que apunta al primer elemento del mismo en orden lexicográfico.

Complejidad: $O(CL * \text{long}(k))$ Donde CL es la cantidad de claves de d y k la palabra mas larga de d

Aliasing: hay aliasing entre los significados en el iterador y los del diccionario

$\text{HAYSIGUIENTE}(\text{in } it : \text{itConjString in } n : \text{String}) \longrightarrow res : \text{bool}$

Pre $\equiv \{true\}$

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

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

Complejidad: $O(1)$

$\text{HAYANTERIOR}(\text{in } it : \text{itConjString in } n : \text{String}) \longrightarrow res : \text{bool}$

Pre $\equiv \{true\}$

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

Descripción: devuelve **true** si y solo si en el iterador todavía quedan elementos para retroceder.

Complejidad: $O(1)$

$\text{SIGUIENTE}(\text{in } it : \text{itConjString in } n : \text{String}) \longrightarrow res : \text{tupla}(\text{String}, \sigma)$

Pre $\equiv \{\text{HaySiguiente?}(it)\}$

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

Descripción: devuelve el elemento siguiente del iterador.

Complejidad: $O(1)$

Aliasing: $res.\text{significado}$ es modificable si y solo si it es modificable. En cambio, $res.\text{clave}$ no es modificable.

$\text{ANTERIOR}(\text{in } it : \text{itConjString in } n : \text{String}) \longrightarrow \text{tupla}(\text{clave} : \text{String}, \text{significado} : \sigma)$

Pre $\equiv \{\text{HayAnterior?}(it)\}$

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

Descripción: devuelve el elemento anterior del iterador.

Complejidad: $O(1)$

Aliasing: $res.\text{significado}$ es modificable si y solo si it es modificable. En cambio, $res.\text{clave}$ no es modificable.

$\text{AVANZAR}(\text{inout } it : \text{itConjString in } n : \text{String})$

Pre $\equiv \{it = it_0 \wedge \text{HaySiguiente?}(it)\}$

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

Descripción: avanza a la posición siguiente del iterador.

Complejidad: $O(1)$

$\text{RETROCEDER}(\text{inout } it : \text{itConjString in } n : \text{String})$

Pre $\equiv \{it = it_0 \wedge \text{HayAnterior?}(it)\}$

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

Descripción: retrocede a la posición anterior del iterador.

Complejidad: $O(1)$

5.2 Representacion

diccString

se representa con cLex)

donde cLex es tupla(dicL : diccString(String, Respresentation del iterador

Trabajo Práctico 2: DiseñoRepresentacion del iterador

El iterador del conjunto lo recorre en orden lexicográfico. Los significados están por referencia.

itConjString

se representa con itcLex)

donde itcLex es itDiccString(String, Bool)

5.3 Algoritmos

iVACIO() \longrightarrow res : diccString

res.dicL \leftarrow Vacio()

res.cardinal \leftarrow 0

O(1)

O(1)

O(1)

6 Base de Datos

6.1 Interfaz

se explica con BASE

usa

géneros nat, string, tabla, registro, campo, dato

Operaciones

TABLAS(in b : base) \longrightarrow res : conj(string)

Pre \equiv {true}

Post \equiv {res =_{obs} nombre(t)}

Descripción: Devuelve el nombre de la tabla ingresada por parametro.

Complejidad: O(1)

Aliasing: Se retorna res por copia, por ser un tipo basico.

DAMETABLA(in b : base) \longrightarrow res : tabla

Pre \equiv {true}

Post \equiv {res =_{obs} claves(t)}

Descripción: Devuelve un conjunto de campos que son claves en la tabla ingresada por parametro.

Complejidad: O(1)

Aliasing: Se devuelve un iterador al conjunto claves por referencia.

HAYJOIN?(**in** $t1 : \text{string}$, $in\ t2 : \text{string}$, $in\ t : \text{base}$) $\longrightarrow res : \text{bool}$

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

Post $\equiv \{res =_{\text{obs}} \text{indices}(t)\}$

Descripción: Devuelve un conjunto de los índices de la tabla ingresada por parametro.

Complejidad: $O(\text{calcular})$

Aliasing: Se devuelve res por referencia y no es modificable.

CAMPOJOIN(**in** $t1 : \text{string}$, $in\ t2 : \text{string}$, $in\ t : \text{base}$) $\longrightarrow res : \text{itConjTrie}(\text{campo})$

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

Post $\equiv \{res =_{\text{obs}} \text{campos}(t)\}$

Descripción: Devuelve un conjunto a los campos de la tabla ingresada por parametro.

Complejidad: $O(1)$

Aliasing: Se devuelve res por referencia.

NUEVADB() $\longrightarrow res : \text{base}$

Pre $\equiv \{\text{True}\}$

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

Descripción: Crea una base sin tablas.

Complejidad: $O(\text{calcular})$

AGREGARTABLA(**in** $t : \text{tabla}$, $in\ b : \text{base}$)

Pre $\equiv \{b.0=b \wedge \text{nombre}(t) \notin \text{tablas}(b) \wedge \text{Vacio?}(t.\text{registros})\}$

Post $\equiv \{\text{agregarTabla}(t\ b.0)\}$

Descripción: Agrega una tabla a la base de datos.

Complejidad: $O(\text{calcular})$

Aliasing: Agrega tabla por referencia.

INSERTARENTRADA(**in** $reg : \text{registro}$, $in\ t : \text{string}$, $in\ b : \text{base}$)

Pre $\equiv \{b.0=b \wedge t \in \text{tablas}(b) \wedge_L \text{puedoInsertar?}(\text{dameTabla}(t)\ reg)\}$

Post $\equiv \{\text{insertarEntrada}(rt\ b.0)\}$

Descripción: Inserta el registro a la tabla que corresponde al string pasado por parametro.

Complejidad: $O(\text{calcular})$

BORRAR(**in** $cr : \text{registro}$, $in\ t : \text{string}$, $in\ b : \text{base}$)

Pre $\equiv \{b.0=b \wedge t \in \text{tablas}(b) \wedge \#(cr.\text{DiccClaves})\}$

Post $\equiv \{\text{borrar}(cr\ t\ b.0)\}$

Descripción: Borra los registros que cumplan el criterio cr pasado por parametro.

Complejidad: $O(\text{calcular})$

GENERARVISTAJOIN(**in** $t1 : \text{string}$, $in\ t2 : \text{string}$, $in\ c : \text{campo}$, $in\ b : \text{base}$)

Pre $\equiv \{b.0=b \wedge t1 \sqsubseteq t2 \wedge \{t1\ t2\} \subseteq \text{tablas}(b) \wedge_L (c \in \text{dameTabla}(t1\ b).\text{diccClaves} \wedge c \in \text{dameTabla}(t2\ b).\text{diccClaves})\}$

Post $\equiv \{\text{generarVistaJoin}(cr, t, b.0)\}$

Descripción: Borra los registros que cumplan el criterio cr pasado por parametro.

Complejidad: $O(\text{calcular})$

BORRARJOIN(**in** $t1 : \text{string}$, $in\ t2 : \text{string}$, $in\ b : \text{base}$)

Pre $\equiv \{b.0=b \wedge \text{hayJoin?}(t1\ t2\ b)\}$

Post $\equiv \{\text{borrarJoin}(t1\ t2\ b.0)\}$

Descripción: Borra correspondiente a los nombres de tablas, pasados por parametro.

Complejidad: $O(\text{calcular})$

REGISTROS(**in** $t : \text{string}$, $in\ b : \text{base}$) $\longrightarrow res : \text{conj}(\text{registro})$

Pre $\equiv \{t \in \text{tablas}(b)\}$

Post $\equiv \{res =_{\text{obs}} \text{registros}(t\ b)\}$

Descripción: Retorna el conjunto de registros correspondientes al nombre de tabla pasado por parametro

Complejidad: $O(\text{calcular})$

Aliasing: Se retorna el conjunto de registros por referencia.

VISTAJOIN(**in** $t1 : \text{string}$, **in** $t2 : \text{string}$, **in** $b : \text{base}$) $\longrightarrow res : \text{conj}(\text{registro})$

Pre $\equiv \{\{t1 \ t2\} \subseteq \text{tablas}(b) \wedge \text{hayJoin?}(t1 \ t2 \ b)\}$

Post $\equiv \{res =_{\text{obs}} \text{vistaJoin}(t1 \ t2 \ b)\}$

Descripción: Retorna el conjunto de registros correspondientes al nombre de tabla pasado por parametro

Complejidad: $O(\text{calcular})$

Aliasing: Se retorna el conjunto de registros por referencia.

CANTIDADDEACCESOS(**in** $t : \text{string}$, **in** $b : \text{base}$) $\longrightarrow res : \text{nat}$

Pre $\equiv \{t \in \text{tablas}(b)\}$

Post $\equiv \{res =_{\text{obs}} \text{cantidadDeAccesos}(t \ b)\}$

Descripción: Retorna la cantidad de modificaciones correspondientes al nombre de tabla pasado por parametro.

Complejidad: $O(\text{calcular})$

Aliasing: Se retorna res por referencia.

TABLAMAXIMA(**in** $b : \text{base}$) $\longrightarrow res : \text{string}$

Pre $\equiv \{\neg \emptyset?(\text{tablas}(b))\}$

Post $\equiv \{res =_{\text{obs}} \text{tablaMaxima}(t \ b)\}$

Descripción: Retorna el nombre de la tabla con la mayor cantidad de modificaciones.

Complejidad: $O(\text{calcular})$

Aliasing: Se retorna el nombre de la tabla por referencia.

ENCONTRARMAXIMO(**in** $t : \text{string}$, **in** $ct : \text{conj}(\text{string})$, **in** $b : \text{base}$) $\longrightarrow res : \text{string}$

Pre $\equiv \{\{t\} \cup ct \subseteq \text{tablas}(b)\}$

Post $\equiv \{res =_{\text{obs}} \text{tablaMaxima}(t \ b)\}$

Descripción: Retorna ...

Complejidad: $O(\text{calcular})$

Aliasing: Se retorna el nombre de la tabla por referencia.

BUSCAR(**in** $\text{criterio} : \text{registro}$, **in** $t : \text{string}$, **in** $b : \text{base}$) $\longrightarrow res : \text{conj}(\text{registro})$

Pre $\equiv \{t \in \text{tablas}(b)\}$

Post $\equiv \{res =_{\text{obs}} \text{tablaMaxima}(t \ b)\}$

Descripción: Retorna ...

Complejidad: $O(\text{calcular})$

Aliasing: Se retorna el nombre de la tabla por referencia.

6.2 Representación

se representa con Base

donde estr es tupla $\langle \text{Tablas} : \text{DiccTrie}(\text{Campo}, \text{info_tabla}) \rangle$

donde info_tabla es tupla $\langle \# \text{Accesos} : \text{nat},$

$\text{TActual} : \text{tabla},$

$\text{Joins} : \text{DiccTrie}(\text{string } \text{info_join}) \rangle$

donde info_join es tupla $\langle R : \text{nat},$

$R\text{borrados} : \text{Conj}(\text{registro}),$

$R\text{agregados} : \text{Conj}(\text{registro}),$

$\text{campoJ} : \text{campo},$

$\text{campoT} : \text{tipo},$

$\text{Join} : \text{Conj}(\text{registro}) \rangle$

Invariante de representación

1. El Nombre de la tabla es un String acotado.
2. Indices es un arreglo de tamaño 2, que aloja el Indice correspondiente segun el orden de creacion.
3. Para toda Dato que es clave en Indice, su significado llamemoslo sign esta incluido en Registros.
- 4.

Función de abstracción

$$\begin{aligned}
 \text{Abs} : \widehat{\text{sistema}} s &\longrightarrow \widehat{\text{CampusSeguro}} && \{\text{Rep}(s)\} \\
 (\forall s : \widehat{\text{sistema}}) & \\
 \text{Abs}(s) \equiv cs : \widehat{\text{CampusSeguro}} & \mid s.\text{campus} =_{\text{obs}} \text{campus}(cs) \wedge \\
 s.\text{estudiantes} =_{\text{obs}} \text{estudiantes}(cs) \wedge & \\
 s.\text{hippies} =_{\text{obs}} \text{hippies}(cs) \wedge & \\
 s.\text{agentes} =_{\text{obs}} \text{agentes}(cs) \wedge & \\
 ((\forall n : \text{nombre}) s.\text{hippies}.\text{definido}(n) \Rightarrow_{\text{L}} s.\text{hippies}.\text{obtener}(n) =_{\text{obs}} \text{posEstYHippie}(n, cs) \vee & \\
 (\forall n : \text{nombre}) s.\text{estudiantes}.\text{definido}(n) \Rightarrow_{\text{L}} s.\text{estudiantes}.\text{obtener}(n) =_{\text{obs}} \text{posEstYHippie}(n, cs)) & \\
 (\forall pl : \text{placa}) s.\text{agentes}.\text{definido}(pl) \Rightarrow_{\text{L}} s.\text{estudiantes}.\text{obtener}(pl).\text{pos} =_{\text{obs}} \text{posAgente}(pl, cs)) & \\
 (\forall pl : \text{placa}) s.\text{agentes}.\text{definido}(pl) \Rightarrow_{\text{L}} s.\text{estudiantes}.\text{obtener}(pl).\text{cantSanciones} =_{\text{obs}} \text{cantSanciones}(pl, cs)) & \\
 (\forall pl : \text{placa}) s.\text{agentes}.\text{definido}(pl) \Rightarrow_{\text{L}} s.\text{estudiantes}.\text{obtener}(pl).\text{cantCapturas} =_{\text{obs}} \text{cantCapturas}(pl, cs)) &
 \end{aligned}$$

6.3 Algoritmos

| | |
|--|-------|
| TABLAS(in <i>b</i> : estr) \longrightarrow <i>res</i> : ConjTrie(string) | |
| <i>res</i> \leftarrow <i>b.tablas.DiccClaves</i> | O(1) |
| | <hr/> |
| | O(1) |
| HAYJOIN?(in <i>t1</i> : string , <i>in</i> <i>t2</i> : string , <i>in</i> <i>b</i> : estr) \longrightarrow <i>res</i> : bool | |
| <i>res</i> \leftarrow Pertenece?(<i>b</i> .Joins, $\langle t1, t2 \rangle$) \vee Pertenece?(<i>b</i> .Joins, $\langle t2, t1 \rangle$) | O(1) |
| | <hr/> |
| | O(1) |
| CAMPOJOIN(in <i>t1</i> : string , <i>in</i> <i>t2</i> : string , <i>in</i> <i>b</i> : estr) \longrightarrow <i>res</i> : campo | |
| if Pertenece?(<i>b</i> .Joins, $\langle t1, t2 \rangle$) then | O(1) |
| <i>res</i> \leftarrow Obtener(<i>e</i> .Joins, $\langle t1, t2 \rangle$).campoJ | O(1) |
| else | |
| <i>res</i> \leftarrow Obtener(<i>e</i> .Joins, $\langle t2, t1 \rangle$).campoJ | O(1) |
| end if | |
| | <hr/> |
| | O(1) |
| NUEVADB() \longrightarrow <i>res</i> : estr | |
| <i>res</i> \leftarrow $\langle vacio(), vacio() \rangle$ | O(1) |
| | <hr/> |
| | O(1) |
| AGREGARTABLA(in <i>t</i> : tabla , <i>in</i> <i>b</i> : estr) | |
| <i>info_tabla</i> \leftarrow $\langle t.cantidadDeAccesos, t \rangle$ | O(1) |
| Definir(<i>b</i> .tablas, nombre(<i>t</i>), <i>info_tabla</i>) | O(1) |
| Falta hacer algo? | |
| | <hr/> |
| | O(1) |
| INSERTARENTRADA(in <i>reg</i> : registro , <i>in</i> <i>t</i> : string , <i>in</i> <i>b</i> : estr) | |
| <i>T_actual</i> \leftarrow Obtener(<i>b</i> .tablas, <i>t</i>).TActual | O(1) |
| agregarRegistro(<i>reg</i> , <i>T_actual</i>) | O(1) |
| Falta hacer algo? | |
| | <hr/> |
| | O(1) |
| BORRAR(in <i>cr</i> : registro , <i>in</i> <i>t</i> : string , <i>in</i> <i>b</i> : estr) | |
| <i>T_actual</i> \leftarrow Obtener(<i>b</i> .tablas, <i>t</i>).TActual | O(1) |
| borrarRegistro(<i>r</i> , <i>T_actual</i>) | O(1) |
| Falta hacer algo? | |
| | <hr/> |
| | O(1) |
| GENERARVISTAJOIN(in <i>t1</i> : string , <i>in</i> <i>t2</i> : string , <i>in</i> <i>c</i> : campo , <i>in/out</i> <i>b</i> : estr) | |
| Join \leftarrow vacio() | |
| <i>T_actual1</i> \leftarrow Obtener(<i>b</i> .tablas, <i>t1</i>).Tactual | O(1) |
| <i>T_actual2</i> \leftarrow Obtener(<i>b</i> .tablas, <i>t2</i>).Tactual | O(1) |
| if Definido?(<i>T_actual1</i> .Indices, <i>c</i>) \wedge Definido?(<i>T_actual2</i> .Indices, <i>c</i>) then | |
| <i>ind1</i> \leftarrow Obtener(<i>T_actual1</i> .Indices, <i>c</i>) | |
| <i>ind2</i> \leftarrow Obtener(<i>T_actual2</i> .Indices, <i>c</i>) | |
| if tipoCampo(<i>T_actual1</i> , <i>c</i>) then | |
| <i>itvalores</i> \leftarrow CreaItConjNat(<i>ind1</i> .PorNat.DiccClaves) | |
| while HaySiguiente(<i>itvalores</i>) do | |

```

    r1 ← Obtener(ind1.PorNat, Siguiente(itvalores))
    r2 ← Obtener(ind2.PorNat, Siguiente(itvalores))
    cj1 ← AgregarRapido(vacio(), r1)
    cj2 ← AgregarRapido(vacio(), r2)
    nuevor ← combinarRegistros(c, cj1, cj2)
    AgregarRapido(Join, DameUno(nuevor))
    Avanzar(itvalores)
  end while
else
  itvalores ← CrearItConjString(ind1.PorString.DiccClaves)
  while HaySiguiente(itvalores) do
    r1 ← Obtener(ind1.PorString, Siguiente(itvalores))
    r2 ← Obtener(ind2.PorString, Siguiente(itvalores))
    cj1 ← AgregarRapido(vacio(), r1)
    cj2 ← AgregarRapido(vacio(), r2)
    nuevor ← combinarRegistros(c, cj1, cj2)
    AgregarRapido(Join, DameUno(nuevor))
    Avanzar(itvalores)
  end while
end if
else
  cjr1 ← T_actuall.registros
  cjr2 ← T_actuall.registros
  Join ← combinarRegistros(c, cjr1, cjr2)
end if
info_join ← ⟨0, vacio(), vacio(), c, tipoCampo(T_actuall, c), Join⟩
Definir(b.Joins, ⟨t1, t2⟩, info_join)

```

O(1)

```

BORRARJOIN(in t1 : string, in t2 : string, in/out b : estr)
  if Pertenece?(b.Joins, ⟨t1, t2⟩) then
    Borrar(b.Joins, ⟨t1, t2⟩)
  else
    if Pertenece?(b.Joins, ⟨t2, t1⟩) then
      Borrar(b.Joins, ⟨t2, t1⟩)
    end if
  end if
end if

```

O(1)