



DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2: Diseño

Primer cuatrimestre - 2016

Algoritmos y Estructuras de Datos II

Grupo 22

Integrante	LU	Correo electrónico
BENZO, Mariano	198/14	marianobenzo@gmail.com
FARIAS, Mauro	821/13	farias.mauro@hotmail.com
GUTTMAN, Martin	686/14	mdg_92@yahoo.com.ar

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	2
1.1. Interfaz	2
1.2. Representación	5
1.3. Algoritmos	7
1.4. Algoritmos operaciones auxiliares	16
2. Tipo es Bool	17
3. Dato(α)	17
3.1. Interfaz	17
3.2. Representación	19
3.3. Algoritmos	20
3.4. Algoritmos operaciones auxiliares	21
4. Diccionario por Naturales	21
4.1. Interfaz	21
4.2. Representación	22
4.3. Algoritmos	23
5. Modulo Diccionario Lexicografico(<i>String</i>, σ)	26
5.1. Interfaz	26
5.2. Operaciones del iterador	27
5.3. Representacion	28
5.4. Algoritmos	29
5.4.1. Algoritmos del Diccionario	29
5.4.2. Algoritmos del iterador	32
6. Modulo Campo es String	34
7. Modulo Registro	34
7.1. Interfaz	34
7.2. Representación	35
7.3. Algoritmos	37
8. NombreTabla es String	38
9. Base de Datos	38
9.1. Interfaz	38
9.2. Representación	41
9.3. Algoritmos	42

1 Tabla

1.1 Interfaz

se explica con TABLA, DICCSTRING(String, ALFHA),
DiccNat(nat, beta), Nat, String, Dato, Campo, Tipo, Registro, ConjString, ConjNat.
géneros tabla

Operaciones

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

Pre \equiv {true}

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

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 : Conj(campo)

Pre \equiv {true}

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

Descripción: Devuelve un conjunto de campos clave en la tabla ingresada por parametro.

Complejidad: $O(1)$

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

INDICES(in t : tab) \longrightarrow res : ConjString(campo)

Pre \equiv {true}

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

Descripción: Devuelve un conjunto campos con los que se crearon los indices.

Complejidad: $O(1)$

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

CAMPOS(in t : tab) \longrightarrow res : ConjString(campo)

Pre \equiv {true}

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

Descripción: Devuelve un conjunto de todos los campos de la tabla.

Complejidad: $O(1)$

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

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

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

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

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

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

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

Complejidad: $O(1)$

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** $\text{nombre} : \text{string}$, $\text{in claves} : \text{conjString}(\text{campo})$, $\text{in columns} : \text{registro}$)
 $\longrightarrow res : \text{tab}$

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

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

Descripción: Crea una tabla sin registros.

Complejidad: $O(1)$

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

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: $O(\text{Log}(n))$

Aliasing: Agrega el registro r por referencia.

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

Pre $\equiv \{t_0 = t \wedge \#(\text{campos}(r)) = 1 \wedge_{\text{L}} \text{Siguiente}(\text{CrearIt}(\text{campos}(\text{crit}))) \in \text{claves}(t)\}$

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

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

Complejidad: $O(n)$

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

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

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

Descripción: Crea un indice en base al campo de crit.

Complejidad: $O(n)$

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 respectos a los registros existentes, para los campos clave en la tabla pasada por parametro.

Complejidad: $O(n)$

Aliasing: Retorna res por referencia, no es modificable.

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: $O(1)$

Aliasing: Retorna res por referencia, no es modificable.

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: $O(L + \text{Log}(n))$

Aliasing: Retorna res por referencia.

MAXIMO(**in** $c : \text{campo}$, **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: $O(L + \text{Log}(n))$

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: $O(1)$

Aliasing: Retorna res por referencia, no es modificable.

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

Pre $\equiv \{True\}$

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

Descripción: Devuelve el conjunto de registros de la tabla, que coinciden con los valores de r .

Complejidad: $O(\text{Cardinal}(cj))$

Aliasing: Retorna res por referencia.

HAYCOINCIDENCIA(**in** $r : \text{registro}$, **in** $cjc : \text{Conj}(\text{campo})$, **in** $cjr : \text{Conj}(\text{registro})$) $\longrightarrow res : \text{bool}$

Pre $\equiv \{True\}$

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

Descripción: Retorna true si algun registro del conjunto cjr , coincide con r en todos los valores de los campos de cjc .

Complejidad: $O(\text{Cardinal}(cjr))$

Aliasing: Retorna res por referencia, no es modificable.

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: $O(\text{Cardinal}(cj1) + \text{Cardinal}(cj2))$

Aliasing: Retorna res por referencia, es modificable.

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: $O(\text{Cardinal}(cj) * \text{Log}(\text{Cardinal}(cj)))$

Aliasing: Retorna res por referencia, no es modificable.

MISMOS TIPOS(**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: $O(1)$

Aliasing: Retorna res por referencia, no es modificable.

BUSCARENTABLA(in $c : \text{campo}$, in $t : \text{tab}$) $\longrightarrow res : \text{Conj}(\text{ItConj}(\text{registro}))$

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

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

Descripción:

Complejidad: $O(O(L + \text{Log}(n)))$

Aliasing: Retorna res por referencia, es modificable.

1.2 Representación

se representa con **Tabla**

donde **tab** es $\text{tupla}(\text{Nombre} : \text{NombreTabla},$
 $\text{Registros} : \text{Conj}(\text{Registro}),$
 $\text{Campos} : \text{DiccString}(\text{Campo}, \text{Tipo}),$
 $\text{Claves} : \text{Conj}(\text{Campo}),$
 $\text{IndiceS} : \text{tupla}(\text{CampoI} : \text{campo},$
 $\text{EnUso} : \text{bool},$
 $\text{Indice} : \text{DiccString}(\text{string}, \text{Conj}(\text{ItConj}(\text{Registro}))),$
 $\text{Min} : \text{Dato},$
 $\text{Max} : \text{Dato})$
 $\text{IndiceN} : \text{tupla}(\text{CampoI} : \text{campo},$
 $\text{EnUso} : \text{bool},$
 $\text{Indice} : \text{DiccNat}(\text{nat}, \text{Conj}(\text{ItConj}(\text{Registro}))),$
 $\text{Min} : \text{Dato},$
 $\text{Max} : \text{Dato})$
 $\text{RelacionInd} : \text{tupla}(\text{CampoR} : \text{campo},$
 $\text{ConsultaN} : \text{DiccNat}(\text{nat}, \text{Acceso}),$
 $\text{ConsultaS} : \text{DiccString}(\text{string}, \text{Acceso}))$
 $\# \text{Accesos} : \text{Nat})$

donde **Acesso** es $\langle S : \text{ItConj}(\text{ItConj}(\text{Registro})), N : \text{ItConj}(\text{ItConj}(\text{Registro})) \rangle$

Invariante de representación

1. $t.\text{Claves}$ esta inclido o es igual a $t.\text{Campos}$.
2. $t.\text{Nombre}$ es un string acotado.
3. Para todo registro r de $t.\text{Registros}$, entonces $\text{Campos}(r)$ es igual al $t.\text{Campos}$.
4. Para todo registro r de $t.\text{Registros}$ y para todo campo c de $\text{Campos}(r)$, entonces $\text{Tipo}(\text{Significado}(r, c))$ es igual $\text{Significado}(t.\text{Campos}, c)$.
5. Si $t.\text{IndiceS}.\text{EnUso}$ es true y $t.\text{IndiceS}.\text{CampoI}$ pertenece a $t.\text{Campos}$, para todo string d , si $\text{Definido}(t.\text{IndiceS}.\text{Indice}, d)$ es true, entonces para todo $\text{itConj}(\text{registro})$ it que pertenece a
 $\text{el conj}(\text{ItConj}(\text{registro}))$ cj $\text{Significado}(t.\text{IndiceS}.\text{Indice}, d)$, registro $r \leftarrow \text{Siguiete}(it)$,
 r pertenece a $t.\text{Registros}$.
6. Si $t.\text{IndiceS}.\text{EnUso}$ es true y $t.\text{IndiceS}.\text{CampoI}$ pertenece a $t.\text{Campos}$, entonces para todo registro R de $t.\text{Registros}$ entonces
 $\text{Definido}(t.\text{IndiceS}.\text{Indice}, \text{Significado}(r, t.\text{IndiceS}.\text{CampoI}))$ es true,
entonces algun $\text{ItConj}(\text{registro})$ it que pertenece a
 $\text{Conj}(\text{itConj}())$ cj $\text{Significado}(t.\text{IndiceS}.\text{Indice}, \text{Significado}(r, t.\text{IndiceS}.\text{CampoI}))$ entonces $\text{Siguiete}(it) = R$.

7. Si $t.\text{IndiceN}.\text{EnUso}$ es true y $t.\text{IndiceN}.\text{CampoI}$ pertenece a $t.\text{Campos}$, para todo string d , si $\text{Definido?}(t.\text{IndiceN}.\text{Indice}, d)$ es true, entonces para todo $\text{itConj}(\text{registro})$ it que pertenece a $\text{el conj}(\text{ItConj}(\text{registro}))$ cj $\text{Significado}(t.\text{IndiceS}.\text{Indice}, d)$, registro $r \leftarrow \text{Siguiete}(it)$, r pertenece a $t.\text{Registros}$.
8. Si $t.\text{IndiceN}.\text{EnUso}$ es true y $t.\text{IndiceN}.\text{CampoI}$ pertenece a $t.\text{Campos}$, entonces para todo registro R de $t.\text{Registros}$ entonces $\text{Definido?}(t.\text{IndiceN}.\text{Indice}, \text{Significado}(r, t.\text{IndiceN}.\text{CampoI}))$ es true, entonces algun $\text{ItConj}(\text{registro})$ it que pertenece a $\text{Conj}(\text{itConj}())$ cj $\text{Significado}(t.\text{IndiceN}.\text{Indice}, \text{Significado}(r, t.\text{IndiceN}.\text{CampoI}))$ entonces $\text{Siguiete}(it)=R$.
9. El campo de $e.\text{RelacionInd}.\text{CampoR}$ pertenece a $t.\text{claves}$
10. Si $t.\text{IndiceS}.\text{EnUso}$ es true y para todo string X , $\neg \text{tipoCampo}(t.\text{campo}, e.\text{RelacionInd}.\text{CampoR})$ y $\text{Definido?}(t.\text{RelacionInd}.\text{ConsultaS}, X)$, tal que $\text{Acceso dataInd} \leftarrow \text{Significado}(t.\text{RelacionInd}.\text{ConsultaN}, Y)$, y el registro $R \leftarrow \text{Siguiete}(\text{Siguiete}(\text{dataInd.S}))$, entonces R pertenece a $t.\text{registros} \wedge_L \text{ValorString}(\text{Significado}(R, t.\text{RelacionInd}.\text{campoR}))=Y$
11. Si $t.\text{IndiceN}.\text{EnUso}$ es true y para todo nat Y , $\neg \text{tipoCampo}(t.\text{campo}, e.\text{RelacionInd}.\text{CampoR})$ y $\text{Definido?}(t.\text{RelacionInd}.\text{ConsultaN}, Y)$, tal que el $\text{Acceso dataInd} \leftarrow \text{Significado}(t.\text{RelacionInd}.\text{ConsultaN}, Y)$, y el registro $R \leftarrow \text{Siguiete}(\text{Siguiete}(\text{dataInd.N}))$, entonces R pertenece a $t.\text{registros} \wedge_L \text{ValorString}(\text{Significado}(R, t.\text{RelacionInd}.\text{campoR}))=Y$
12. Si $t.\text{IndiceS}.\text{EnUso}$ es true y para todo nat Y , $\text{tipoCampo}(t.\text{campo}, e.\text{RelacionInd}.\text{CampoR})$ y $\text{Definido?}(t.\text{RelacionInd}.\text{ConsultaN}, Y)$, tal que el $\text{Acceso dataInd} \leftarrow \text{Significado}(t.\text{RelacionInd}.\text{ConsultaN}, Y)$, y el registro $R \leftarrow \text{Siguiete}(\text{Siguiete}(\text{dataInd.S}))$, entonces R pertenece a $t.\text{registros} \wedge_L \text{ValorString}(\text{Significado}(R, t.\text{RelacionInd}.\text{campoR}))=Y$
13. Si $t.\text{IndiceN}.\text{EnUso}$ es true y para todo nat Y , $\text{tipoCampo}(t.\text{campo}, e.\text{RelacionInd}.\text{CampoR})$ y $\text{Definido?}(t.\text{RelacionInd}.\text{ConsultaN}, Y)$, tal que el $\text{Acceso dataInd} \leftarrow \text{Significado}(t.\text{RelacionInd}.\text{ConsultaN}, Y)$, y el registro $R \leftarrow \text{Siguiete}(\text{Siguiete}(\text{dataInd.N}))$, entonces R pertenece a $t.\text{registros} \wedge_L \text{ValorString}(\text{Significado}(R, t.\text{RelacionInd}.\text{campoR}))=Y$
14. El valor de $e.\#\text{Accesos}$ debe ser la cantidad de registros agregados, la cantidad de registros borrados

Función de abstracción

$$\begin{aligned}
\text{Abs} : \widehat{\text{tab}} s &\longrightarrow \widehat{\text{Tabla}} && \{\text{Rep}(s)\} \\
(\forall s : \widehat{\text{tab}}) & \\
\text{Abs}(s) \equiv t : \widehat{\text{Tabla}} \mid & s.\text{Nombre} =_{\text{obs}} \text{nombre}(t) \wedge s.\text{Claves} =_{\text{obs}} \text{claves}(t) \wedge \\
& s.\text{Indices} =_{\text{obs}} \text{indices}(t) \wedge s.\text{Registros} =_{\text{obs}} \text{registros}(t) \wedge \text{DiccClaves}(s.\text{Campos}) =_{\text{obs}} \text{campos}(t) \\
& \wedge s.\#\text{Accesos} =_{\text{obs}} \text{cantidadDeAccesos}(t) \wedge \\
& ((\forall c : \text{campo}) \text{Definido?}(s.\text{Campos}, c) \Rightarrow_L \text{Significado}(s.\text{Campos}, c) =_{\text{obs}} \text{tipoCampo}(c, t))
\end{aligned}$$

1.3 Algoritmos

NOMBRE(in $t : \text{tab}$) $\longrightarrow res : \text{string}$ $res \leftarrow t.Nombre$	O(1) por ref
	O(1) por ref
CLAVES(in $t : \text{tab}$) $\longrightarrow res : \text{Conj}(\text{campo})$ $res \leftarrow t.Claves$	O(1) por ref
	O(1) por ref
INDICES(in $t : \text{tab}$) $\longrightarrow res : \text{ConjString}(\text{campo})$ $\text{ConjString}(\text{campo}) \text{ } res \leftarrow \text{vacio}();$ if $t.IndiceS.EnUso$ then $\text{AgregarRapido}(res, t.IndiceS.CampoI)$ end if if $t.IndiceN.EnUso$ then $\text{AgregarRapido}(res, t.IndiceN.CampoI)$ end if	O(1) por ref O(1) por ref O(1) por ref O(1) por ref O(1) por ref
	O(1) por ref
CAMPOS(in $t : \text{tab}$) $\longrightarrow res : \text{ConjString}(\text{campo})$ $res \leftarrow \text{DiccClaves}(t.Campos)$	O(1) por ref
	O(1) por ref
TIPOCAMPO(in $c : \text{campo}$, <i>in</i> $t : \text{tab}$) $\longrightarrow res : \text{Tipo}$ $res \leftarrow \text{Significado}(t.Campos, c)$	O(1) por ref
	O(1) por ref
REGISTROS(in $t : \text{tab}$) $\longrightarrow res : \text{Conj}(\text{registro})$ $res \leftarrow t.registros$ Se retorna el conjunto por referencia	O(1) por ref
	O(1) por ref
CANTDEACCESOS(in $t : \text{tab}$) $\longrightarrow res : \text{nat}$ $res \leftarrow t.cantDeAccesos$	O(1) por ref
	O(1) por ref

NUEVATABLA(**in** *nombre* : **string**, *in* *claves* : **conj**(**campo**), *in* *columnas* : **registro**) \longrightarrow *res* :
tab

Conj(registro) Registros \leftarrow Vacio()	O(1) por ref
DiccString(campo, tipo) Campos \leftarrow Vacio()	O(1) por ref
Campo <i>c</i> \leftarrow Siguiente(CrearIt(claves))	
Dato <i>d</i> \leftarrow Obtener(columnas, Siguiente(CrearIt(claves)))	
IndiceS \leftarrow $\langle c, False, Vacio(), d, d \rangle$	O(1) por ref
IndiceN \leftarrow $\langle c, False, Vacio(), d, d \rangle$	O(1) por ref
#Acessos \leftarrow 0	O(1) por ref
RelacionInd \leftarrow $\langle c, Vacio(), Vacio() \rangle$	
<i>res</i> \leftarrow $\langle nombre, Registros, Campos, claves, IndiceS, IndiceN, RelacionInd, 0 \rangle$	O(1) por ref
itcampos \leftarrow crearIt(Campos(columnas))	O(# de campos)
while HaySiguiente(itcampos) do	O(# de campos)
Dato valor \leftarrow Significado(columnas, Siguiente(itcampos))	O(L)
Definir(<i>res</i> .Campos, Siguiente(itcampos), Tipo?(valor))	O(1) por ref
Avanzar(itcampos)	O(1) por ref

end while

Donde L es la longitud del valor string mas largo.

O((#(campos(columnas))*L)

AGREGARREGISTRO(**in** r : registro, *in/out* t : tab)

Para poder acceder al registro en el conj en $O(1)$ por ref, guardo el iterador al elemento

itConj(Registro) nuevo \leftarrow AgregarRapido(t .Registros, r) $O(1)$ por ref

t .#Accesos++ $O(1)$ por ref

Este registro debe ser indexado, si algun indice esta en uso.

if t .IndiceS.EnUso \wedge t .IndiceN.EnUso **then**

Dato valorS \leftarrow Significado(r , t .IndiceS.CampoI) $O(L)$

Dato valorN \leftarrow Significado(r , t .IndiceN.CampoI) $O(\log(n))$

Bool DefinidoS \leftarrow Definido?(t .IndiceS.Indice, ValorString(valorS)) $O(1)$

Bool DefinidoN \leftarrow Definido?(t .IndiceN.Indice, ValorNat(valorN)) $O(\log(n))$

if \neg DefinidoS \wedge \neg DefinidoN **then** $O(1)$

Ambos no estan definidos

cjS \leftarrow vacio() $O(1)$ por ref

cjN \leftarrow vacio() $O(1)$ por ref

newS \leftarrow AgregarRapido(cjS, nuevo) $O(1)$ por ref

newN \leftarrow AgregarRapido(cjN, nuevo) $O(1)$ por ref

Definir(t .IndiceS.Indice, ValorString(valorS), cjS) $O(1)$ por ref

Definir(t .IndiceN.Indice, ValorString(valorN), cjN) $O(\log(n))$ por ref

else

if DefinidoS \wedge \neg DefinidoN **then** $O(1)$

cjS \leftarrow Significado(t .IndiceS.Indice, ValorString(valorS)) $O(1)$

cjN \leftarrow vacio() $O(1)$

newS \leftarrow AgregarRapido(cjS, nuevo) $O(1)$

newN \leftarrow AgregarRapido(cjN, nuevo) $O(1)$

Definir(t .IndiceN.Indice, ValorNat(valorN), cjN) $O(\log(n))$ por ref

else

if \neg DefinidoS \wedge DefinidoN **then** $O(1)$

cjN \leftarrow Significado(t .IndiceN.Indice, ValorNat(valorN)) $O(\log(n))$

cjS \leftarrow vacio() $O(1)$

newS \leftarrow AgregarRapido(cjS, nuevo) $O(1)$

newN \leftarrow AgregarRapido(cjN, nuevo) $O(1)$

Definir(t .IndiceS.Indice, ValorString(valorS), cjS) $O(1)$ por ref

else

Caso en que esta definido en los dos indices

cjN \leftarrow Significado(t .IndiceN.Indice, ValorNat(valorN)) $O(\log(n))$

cjS \leftarrow Significado(t .IndiceS.Indice, ValorString(valorS)) $O(1)$

newS \leftarrow AgregarRapido(cjS, nuevo) $O(1)$ por ref

newN \leftarrow AgregarRapido(cjN, nuevo) $O(\log(n))$ por ref

end if

end if

end if

if t .IndiceS.Min $>$ valorS **then** $O(L)$

t .IndiceS.Min \leftarrow valorS $O(1)$ por ref

end if

if valorS $>$ t .IndiceS.Max **then** $O(L)$

```

    t.IndiceS.Max  $\leftarrow$  valorS O(1) por ref
end if
if t.IndiceN.Min > valorN then O(1) por ref
    t.IndiceN.Min  $\leftarrow$  valorN O(1) por ref
end if
if valorN > t.IndiceN.Max then O(1) por ref
    t.IndiceN.Max  $\leftarrow$  valorN O(1) por ref
end if
else
if t.IndiceS.EnUso  $\wedge$   $\neg$ t.IndiceN.EnUso then
    Obtengo de r el valor del campo con el que se creo el indice.
    Dato valor  $\leftarrow$  Significado(r, t.IndiceS.CampoI) O(L)
    Bool Def  $\leftarrow$  Definido?(t.IndiceS.Indice, ValorString(valor))
if Def then O(1) por ref
        Si esta definido, entonces hay varios registros que cumplen
        agrego el iterador al conjunto de que ya estaban.
        viejo  $\leftarrow$  Significado(t.IndiceS.Indice, ValorString(valor))
O(L)
        itConj(registro) newS  $\leftarrow$  AgregarRapido(viejo, nuevo)
O(L)
        itConj(registro) newN  $\leftarrow$  CrearIt(vacio())
else
        Conj(Registro) viejo  $\leftarrow$  Vacio() O(1) por ref
        itConj(registro) newS  $\leftarrow$  AgregarRapido(viejo, nuevo)
O(1) por ref
        itConj(registro) newN  $\leftarrow$  CrearIt(vacio())
        Definir(t.IndiceS.Indice, ValorString(valor), viejo) O(L)
        Como ingresamos un nuevo valor, actualizamos el min y max
if t.IndiceS.Min > valor then O(L)
            t.IndiceS.Min  $\leftarrow$  valor O(L)
end if
if valor > t.IndiceS.Max then O(L)
            t.IndiceS.Max  $\leftarrow$  valor O(L)
end if
end if
end if
else
if  $\neg$ t.IndiceS.EnUso  $\wedge$  t.IndiceN.EnUso then
    Obtengo de r el valor del campo con el que se creo el indice.
    Dato valor  $\leftarrow$  Significado(r, t.IndiceN.CampoI) O(L)
    Bool Def  $\leftarrow$  Definido?(t.IndiceN.Indice, ValorNat(valor))
O(log(n))
if Def then O(1) por ref
        Si esta definido, entonces hay varios registros que cumplen
        agrego el iterador al conjunto de que ya estaban.
        viejo  $\leftarrow$  Significado(t.IndiceN.Indice, ValorNat(valor))
O(L)
        itConj(registro) newN  $\leftarrow$  AgregarRapido(viejo, nuevo)
O(L)
        itConj(registro) newS  $\leftarrow$  CrearIt(vacio())
else
        Conj(Registro) viejo  $\leftarrow$  Vacio() O(1) por ref
        itConj(registro) newN  $\leftarrow$  AgregarRapido(viejo, nuevo)

```

	O(1) por ref
itConj(registro) newS ← CrearIt(vacio())	
Definir(t.IndiceN.Indice, ValorNat(valor), viejo)	
	O(L)
Como ingresamos un nuevo valor, actualizamos el min y max	
if t.IndiceN.Min > valor then	O(1) por ref
t.IndiceN.Min ← valor	O(1) por ref
end if	
if valor > t.IndiceN.Max then	O(1) por ref
t.IndiceN.Max ← valor	O(1) por ref
end if	
end if	
end if	
end if	
Bool BasadoEn ← Significado(t.campos, t.RelacionInd.CampoR)	
if BasadoEn then	
Es True entonces es un DiccNat	
Nat elem ← ValorNat(Significado(r, t.RelacionInd.CampoR))	
Definir(t.RelacionInd.ConsultaN, elem, < newS, newN >)	
else	
Es False entonces es un DiccString	
String elem ← ValorString(Significado(r, t.RelacionInd.CampoR))	
Definir(t.RelacionInd.ConsultaS, elem, < newS, newN >)	
end if	
	<hr/> O(L+Log(n))

```

INDEXAR(in c : campo, in/out t : tab)
  if tipoCampo(c,t.campos) then
    t.IndiceN.EnUso  $\leftarrow$  True
  else
    t.IndiceS.EnUso  $\leftarrow$  True
  end if
  ItConj(ItConj(registro)) newS  $\leftarrow$  CrearIt(CrearIt(Vacio()))
  ItConj(ItConj(registro)) newN  $\leftarrow$  CrearIt(CrearIt(Vacio()))
  < ItConj(ItConj(registro)), ItConj(ItConj(registro)) > dataIndices
  Buscamos en la Relacion de Indices segun su campo.
  Primero necesito saber en base a que tipo de campo fue creado.
  Tipo BasadoEn  $\leftarrow$  Significado(t.campos, t.RelacionInd.CampoR)
  itConj(registro) cr  $\leftarrow$  CrearItConj(t.registros)
  while HaySiguiente(cr) do
    if tipoCampo?(c,t) then
      Caso Naturales
      Dato valor  $\leftarrow$  Significado(Siguiente(cr), c)
      Bool Definido  $\leftarrow$  Definido?(t.IndiceN.Indice, ValorNat(valor))
      itConj(registro) itr  $\leftarrow$  Copiar(cr)
      if Definido then
        Significa que para este valor del indice hay mas de un iterador de conj a reg
        regviejos  $\leftarrow$  Significado(t.IndiceN.Indice, ValorNat(valor))
        itConj(itConj(registro)) newN  $\leftarrow$  AgregarRapido(regviejos, itr)
      else
        Conj(itConj(registro)) nuevo  $\leftarrow$  Vacio()
        newN  $\leftarrow$  AgregarRapido(nuevo, itr)
        Definir(t.IndiceN.Indice, ValorNat(valor), nuevo)
      end if
    else
      Caso Strings
      Dato valor  $\leftarrow$  Significado(Siguiente(cr), c)
      Bool Definido  $\leftarrow$  Definido?(t.IndiceS.Indice, ValorString(valor))
      itConj(registro) itr  $\leftarrow$  cr
      if Definido then
        Significa que para este valor del indice hay mas de un iterador de conj a reg
        regviejos  $\leftarrow$  Significado(t.IndiceS.Indice, ValorString(valor))
        itConj(itConj(registro)) newS  $\leftarrow$  AgregarRapido(regviejos, itr)
      else
        Conj(itConj(registro)) nuevo  $\leftarrow$  Vacio()
        newS  $\leftarrow$  AgregarRapido(nuevo, itr)
        Definir(t.IndiceS.Indice, ValorString(valor), nuevo)
      end if
    end if
  end if
  Ahora actualizo la relacion de indices
  Dato valorR  $\leftarrow$  Significado(Siguiente(cr), t.RelacionInd.CampoR)
  Bool DefinidoR
  if BasadoEn then
    La relacion de indices esta basada en un campo Natural
    DefinidoR  $\leftarrow$  Definido?(t.RelacionInd.ConsultaN, ValorNat(valorR))
  else
    La relacion de indices esta basada en un campo String
    DefinidoR  $\leftarrow$  Definido?(t.RelacionInd.ConsultaS, ValorString(valorR))
  end if

```

```

end if
if DefinidoR then
  Significa que el otro indice esta en uso.
  if BasadoEn then
    La relacion de indices esta basada en un campo Natural
    dataIndices  $\leftarrow$  Significado(t.RelacionInd.ConsultaN, ValorNat(valorR))
    if tipoCampo?(c,t.campos) then
      dataIndices.N  $\leftarrow$  newN
    else
      dataIndices.S  $\leftarrow$  newS
    end if
  else
    La relacion de indices esta basada en un campo String
    dataIndices  $\leftarrow$  Significado(t.RelacionInd.ConsultaS, ValorString(valorR))
    if tipoCampo?(c,t) then
      dataIndices.N  $\leftarrow$  newN
    else
      dataIndices.S  $\leftarrow$  newS
    end if
  end if
else
  Significa que el otro indice no esta en uso.
  dataIndices  $\leftarrow$  < newS, newN >
  if BasadoEn then
    Definir(t.RelacionInd.ConsultaN, ValorNat(valorR), dataIndices)
  else
    Definir(t.RelacionInd.ConsultaS, ValorString(valorR), dataIndices)
  end if
end if
  Avanzar(cr)
end while

```

O(1) por ref

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

$res \leftarrow \text{compatible}(r,t) \wedge \neg \text{hayCoincidencia}(r, \text{Campo}(r), \text{registros}(t))$

O(calcular))

O(calcular)

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

bool valor \leftarrow True

if Cardinal(campos(r))=Cardinal(DiccClaves(t.Campos)) **then**

itcampos \leftarrow CrearItString(DiccClaves(t.Campos))

while valor \wedge HaySiguiente(itcampos) **do**

Campo c \leftarrow Siguiente(itcampos)

valor \leftarrow Definido?(r, c)

end while

else

valor \leftarrow False

end if

$res \leftarrow \text{valor} \wedge_L \text{mismosTipos}(r,t)$

El costo del While es O(1) por ref ya que la cantidad de campos de la tabla es acotado

O(1)

MINIMO (in $c : \text{campo}$, $\text{in } t : \text{tab}$) $\longrightarrow res : \text{dato}$	
Si hay indice en el campo c, debe ser de complejidad O(1) por ref	
if $t.\text{IndiceS}.\text{EnUso} \wedge t.\text{IndiceS}.\text{CampoI}=c$ then	
Sabemos que hay un indice string para el campo c	
$res \leftarrow t.\text{IndiceS}.\text{Min}$	O(Cardinal(t.registros))
else	
if $t.\text{IndiceN}.\text{EnUso} \wedge t.\text{IndiceN}.\text{CampoI}=c$ then	
Sabemos que hay un indice string para el campo c	
$res \leftarrow t.\text{IndiceN}.\text{Min}$	O(Cardinal(t.registros))
end if	
end if	
	<hr/>
	O(1) por ref
MAXIMO (in $c : \text{campo}$, $\text{in } t : \text{tab}$) $\longrightarrow res : \text{dato}$	
Si hay indice en el campo c, debe ser de complejidad O(1) por ref	
if $t.\text{IndiceS}.\text{EnUso} \wedge t.\text{IndiceS}.\text{CampoI}=c$ then	
Sabemos que hay un indice string para el campo c	
$res \leftarrow t.\text{IndiceS}.\text{Max}$	O(Cardinal(t.registros))
else	
if $t.\text{IndiceN}.\text{EnUso} \wedge t.\text{IndiceN}.\text{CampoI}=c$ then	
Sabemos que hay un indice string para el campo c	
$res \leftarrow t.\text{IndiceN}.\text{Max}$	O(Cardinal(t.registros))
end if	
end if	
	<hr/>
	O(1) por ref
PUEDEINDEXAR (in $c : \text{campo}$, $\text{in } t : \text{tab}$) $\longrightarrow res : \text{bool}$	
if TipoCampo(c, t) then	
$res \leftarrow \neg(t.\text{IndiceN}.\text{EnUso})$	
else	
$res \leftarrow \neg(t.\text{IndiceS}.\text{EnUso})$	
end if	
	<hr/>
	O(1) por ref
HAYCOINCIDENCIA (in $r : \text{registro}$, $\text{in } cc : \text{ConjString}(\text{campo})$, $\text{in } cr : \text{Conj}(\text{registro})$) $\longrightarrow res : \text{bool}$	
$itcr \leftarrow \text{CrearItConj}(cr)$	O(1) por ref
$res \leftarrow \text{false}$	O(1) por ref
while HaySiguiente(itcr) do	O(Cardinal(cr))
$res \leftarrow \text{coincideAlguno}(r, cc, \text{Siguiente}(itcr)) \vee res$	O(1) por ref
Avanzar(itcr)	O(1) por ref
end while	
	<hr/>
	O(Cardinal(cr))
COINCIDENCIAS (in $\text{crit} : \text{registro}$, $\text{in } cr : \text{Conj}(\text{registro})$) $\longrightarrow res : \text{Conj}(\text{ItConj}(\text{registro}))$	
$\text{Conj}(\text{registro}) \text{ salida} \leftarrow \text{Vacio}()$	O(1) por ref
Debemos comparar todos los registros de cr.	
y agregarlos al conjunto de registros salida	
$itcr \leftarrow \text{CrearItConj}(cr)$	
while HaySiguiente?(cr) do	O(Cardinal(cr))
if coincidenTodos(crit.campos(crit), Siguiente(itcr)) then	O(1) por ref

AgregarRapido(salida, CrearIt(Siguiente(itcr)))	O(1) por ref
AgregarRapido(salida, itcr)	O(1) por ref
end if	
Avanzar(itcr)	O(1) por ref
end while	
res ← salida	O(1) por ref
	O(Cardinal(cr))
<hr/>	
COMBINARREGISTROS(in <i>c</i> : campo, <i>in cr1</i> : Conj(registro), <i>in cr2</i> : Conj(registro)) → <i>res</i> : Conj(registros)	
itcr1 ← CrearItConjString(cr1)	O(1) por ref
Lo malo es que combinarRegistros sera O(Cardinal(cr2)*Log(Cardinal(cr2)))	
if Cardinal(cr2) ≥ 1 then	O(1) por ref
Registro rtemp ← Siguiente(CrearIt(cr2))	O(1) por ref
Tipo rac ← Tipo?(Significado(rtemp, c))	O(1) por ref
if rac then	O(1) por ref
Caso Natural	
DiccNat(Nat, Conj(registro)) d ← vacio()	O(1) por ref
itcr2 ← CrearIt(cr2)	O(1) por ref
while HaySiguiente?(itcr2) do	O(1) por ref
Dato valor ← Obtener(Siguiente(itcr2), c)	O(1) por ref
if Definido?(d, ValorNat(valor)) then	O(Cardinal(cr2))
cjViejo ← Significado(d, ValorNat(valor))	O(Cardinal(cr2))
AgregarRapido(d, Siguiente(itcr2))	O(1) por ref
else	
ConjNat(registro) cjNuevo ← vacio()	O(1) por ref
AgregarRapido(d, Siguiente(itcr2))	O(1) por ref
Definir(d, ValorNat(valor), cjNuevo)	O(Cardinal(cr2))
end if	
Avanzar(itcr2)	O(1) por ref
end while	
else	
Caso String	
DiccString(String, Conj(registro)) d ← vacio()	O(1) por ref
itcr2 ← CrearIt(cr2)	O(1) por ref
while HaySiguiente?(itcr2) do	O(1) por ref
Dato valor ← Obtener(Siguiente(itcr2), c)	O(1) por ref
if Definido?(d, ValorString(valor)) then	O(Cardinal(cr2))
cjViejo ← Significado(d, ValorString(valor))	O(Cardinal(cr2))
AgregarRapido(d, Siguiente(itcr2))	O(1) por ref
else	
ConjString(registro) cjNuevo ← vacio()	O(1) por ref
AgregarRapido(d, Siguiente(itcr2))	O(1) por ref
Definir(d, ValorString(valor), cjNuevo)	O(Cardinal(cr2))
end if	
Avanzar(itcr2)	O(1) por ref
end while	
end if	
itcr1 ← CrearIt(cr1)	O(1) por ref
res ← vacio()	
while HaySiguiente(itcr1) do	O(Cardinal(cr1))
AgregarRapido(res, combinarTodos(c, Siguiente(itcr1), cr2))	O(Cardinal(cr2))

Avanzar(itcr1)	O(1) por ref
end while	
else	
res ← vacío()	
end if	
	<hr/> O(Cardinal(cr1))
DAMECOLUMNA(in <i>c</i> : campo, <i>in cr</i> : Conj(registro)) → <i>res</i> : Conj(dato)	
Conj(Dato) cj ← vacío();	O(1) por ref
Donde n es el cardinal de cr .	
	<hr/> O(n)
MISMOSTIPOS(in <i>r</i> : registro, <i>in t</i> : tab) → <i>res</i> : bool	
valor ← True	O(1) por ref
itconjClaves ← CrearIt(Campo(r))	O(1) por ref
while valor ∧ _L HaySiguiente?(itconjClaves) do	O(1) por ref
val1 ← tipo?(Significado(r, Siguiente(itconjClaves)))	O(1) por ref
val2 ← tipoCampo(Siguiente(itconjClaves), t)	O(1) por ref
valor ← (val1 = val2)	O(1) por ref
Avanzar(cr);	O(1) por ref
end while	
res ← valor	
	<hr/> O(1) por ref

1.4 Algoritmos operaciones auxiliares

BUSCARENTABLA(**in** *criterio* : registro, *in t* : tab) → *res* : Conj(ItConj(registro))

Primero busco que campos de r, estan en los campos de t y son del mismo tipo

itcampos ← DiceClaves(t.campos)

Bool Encontrado ← false

Campo EncontradoCampoInd

Conj(Campo) cj ← vacío()

while HaySiguiente?(itcampos) ∧ ¬Encontrado **do**

Campo c ← Siguiente(itcampos)

Bool Def ← Definido?(criterio, c)

if Def **then**

bool valorD ← (Tipo?(Significado(criterio, c))=Significado(t.campos, c))

if valorD **then**

El campo esta en ambos y es del mismo tipo, lo agrego al conj lineal

AgregarRapido(cj, c)

Vemos tambien si el campo c de criterio esta en los indices

Bool EncS ← (t.IndiceS.EnUso ∧_L t.IndiceS.CampoI=c)

Bool EncN ← (t.IndiceN.EnUso ∧_L t.IndiceN.CampoI=c)

Encontrado ← (EncS ∨ EncN)

if Encontrado **then**

EncontradoCampoInd ← c

end if

end if

end if

Avanzar(itcri)

end while

Si Encontrado es true entonces uso indice, sino recorro todo los registros
if Encontrado **then**
 Entonces hay un indice para EncontradoCampoInd
if t.IndiceN.EnUso \wedge_L t.IndiceN.CampoI=EncontradoCampoInd **then**
 Caso Natural
 Obtengo el valor nat del registro criterio
 Nat valor \leftarrow ValorNat(Significado(criterio, EncontradoCampoInd))
 Conj(itConj(registro)) res \leftarrow Significado(t.IndiceN.Indice, valor)
end if
if \neg Significado(t.campos, EncontradoC) **then**
 Caso String
 Obtengo el valor nat del registro criterio
 String valor \leftarrow ValorString(Significado(criterio, EncontradoCampoInd))
 Conj(itConj(registro)) res \leftarrow Significado(t.IndiceS.Indice, valor)
end if
 elseres \leftarrow Coincidencias(criterio, t.registros)
end if
 La complejidad depende de si hay indice para algun campo de criterio,
 en ese caso la complejidad es $O(\#(\text{Campos}(\text{criterio})) + L + \text{Log}(n))$
 En caso contrario es $O(\#(\text{Campos}(t)) + \#(\text{Registros}(t)))$

$O(1)$ por ref

2 Tipo es Bool

3 Dato(α)

3.1 Interfaz

se explica con DATO

usa

géneros nat, string, tipo

Operaciones

TIPO?(in d : dato) \longrightarrow res : tipo

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

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

Descripción: Devuelve el tipo del dato ingresado por parametro.

Complejidad: $O(1)$

Aliasing: Se retorna res por referencia.

VALORNAT(in d : dato) \longrightarrow res : nat

Pre $\equiv \{\text{Nat?}(d)\}$

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

Descripción: Devuelve valor numerido del dato por parametro.

Complejidad: $O(1)$

Aliasing: Se devuelve res por referencia.

VALORSTRING(in d : dato) \longrightarrow res : string

Pre $\equiv \{\text{String?}(d)\}$

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

Descripción: Devuelve valor del dato por parametro.

Complejidad: $O(1)$

Aliasing: Se devuelve res por referencia.

DATONAT(**in** $n : \alpha$, **in** $\text{tipoDelDato} : \text{tipo}$) $\longrightarrow res : \text{dato}$

Pre $\equiv \{\text{tipoDelDato}\}$

Post $\equiv \{res =_{\text{obs}} \text{datoNat}(n, \text{tipoDelDato})\}$

Descripción: Crea un dato de valor numerico.

Complejidad: $O(1)$

Aliasing: Se devuelve res por referencia.

DATOSTR(**in** $n : \alpha$, **in** $\text{tipoDelDato} : \text{tipo}$) $\longrightarrow res : \text{dato}$

Pre $\equiv \{\neg \text{tipoDelDato}\}$

Post $\equiv \{res =_{\text{obs}} \text{datoString}(n, \text{tipoDelDato})\}$

Descripción: Crea un dato de valor de letras.

Complejidad: $O(1)$

Aliasing: Se devuelve res por referencia.

MISMO TIPO?(**in** $d1 : \text{dato}$, **in** $d2 : \text{dato}$) $\longrightarrow res : \text{bool}$

Pre $\equiv \{true\}$

Post $\equiv \{res =_{\text{obs}} \text{mismoTipo?}(d1, d2)\}$

Descripción: Informa si los datos pasados por parametro son del mismo tipo de valor.

Complejidad: $O(1)$

STRING?(**in** $d : \text{dato}$) $\longrightarrow res : \text{bool}$

Pre $\equiv \{true\}$

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

Descripción: Informa si el dato pasado por parametro es de tipo string.

Complejidad: $O(1)$

NAT?(**in** $d : \text{dato}$) $\longrightarrow res : \text{bool}$

Pre $\equiv \{true\}$

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

Descripción: Informa si el dato pasado por parametro es de tipo nat.

Complejidad: $O(1)$

MIN(**in** $cd : \text{Conj}(\text{dato})$) $\longrightarrow res : \text{dato}$

Pre $\equiv \{\neg \text{EsVacio?}(cd)\}$

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

Descripción: Retorna el minimo entre los valores del conjunto de datos pasado por parametro.

Complejidad: $O(\text{Cardinal}(cd))$

Aliasing: Retorna res por referencia.

MAX(**in** $cd : \text{Conj}(\text{dato})$) $\longrightarrow res : \text{dato}$

Pre $\equiv \{\neg \text{EsVacio?}(cd)\}$

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

Descripción: Retorna el maximo entre los valores del conjunto de datos pasado por parametro.

Complejidad: $O(\text{Cardinal}(cd))$

Aliasing: Retorna res por referencia.

<=(**in** $d1 : \text{dato}$, **in** $d2 : \text{dato}$) $\longrightarrow res : \text{bool}$

Pre $\equiv \{\text{mismoTipo?}(d1, d2)\}$

Post $\equiv \{res =_{\text{obs}} <= (d1, d2)\}$

Descripción: Retorna el maximo entre los valores del conjunto de datos pasado por parametro.

Complejidad: $O(\text{Cardinal}(cd))$

Aliasing: Retorna res por referencia. `oincidenTodos(crit,campos(crit),Siguiente(cr))`

3.2 Representación

se representa con `datotupla` \langle Valor : α ,
TipoValor : `bool` \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.

3.3 Algoritmos

TIPO?(in <i>a</i> : dato) \longrightarrow <i>res</i> : bool <i>res</i> \leftarrow <i>a.TipoValor</i>	O(1)
	<hr/>
	O(1)
VALORNAT(in <i>a</i> : dato) \longrightarrow <i>res</i> : nat <i>res</i> \leftarrow <i>a.Valor</i>	O(1)
	<hr/>
	O(1)
VALORSTR(in <i>a</i> : dato) \longrightarrow <i>res</i> : string <i>res</i> \leftarrow <i>a.Valor</i>	O(1)
	<hr/>
	O(1)
MISMO TIPO?(in <i>d1</i> : dato, <i>in</i> <i>d2</i> : dato) \longrightarrow <i>res</i> : bool <i>res</i> \leftarrow <i>tipo?(d1) = tipo?(d2)</i>	O(1)
	<hr/>
	O(1)
NAT?(in <i>a</i> : dato) \longrightarrow <i>res</i> : bool <i>res</i> \leftarrow <i>tipo?(a)</i>	O(1)
	<hr/>
	O(1)
STRING?(in <i>a</i> : dato) \longrightarrow <i>res</i> : bool <i>res</i> \leftarrow \neg <i>Nat?(a)</i>	O(1)
	<hr/>
	O(1)
MIN(in <i>cd</i> : Conj(dato)) \longrightarrow <i>res</i> : dato <i>itcd</i> \leftarrow CrearItConj(<i>cd</i>) <i>minimo</i> \leftarrow Siguiente(<i>itcd</i>) while HaySiguiente(<i>itcd</i>) do if Siguiente(<i>itcd</i>) _i = <i>minimo</i> then <i>minimo</i> \leftarrow Siguiente(<i>itcd</i>); end if Avanzar(<i>itcr</i>); end while	
	<hr/>
	O(Cardinal(<i>cd</i>))
MAX(in <i>cd</i> : Conj(dato)) \longrightarrow <i>res</i> : dato <i>itcd</i> \leftarrow CrearItConj(<i>cd</i>) <i>maximo</i> \leftarrow Siguiente(<i>itcd</i>) while HaySiguiente(<i>itcd</i>) do if <i>maximo</i> _j =Siguiente(<i>itcd</i>) then <i>minimo</i> \leftarrow Siguiente(<i>itcd</i>); end if	

Avanzar(itcr);	
end while	
$\leq = (\text{in } d1 : \text{dato}, \text{ in } d2 : \text{dato}) \longrightarrow res : \text{bool}$	$O(\text{Cardinal}(cd))$
if String?(d1) then res \leftarrow valorStr(d1) _i =valorStr(d2)	
else res \leftarrow valorNat(d1) _i =valorNat(d2)	
end if	$O(1)$

3.4 Algoritmos operaciones auxiliares

4 Diccionario por Naturales

4.1 Interfaz

se explica con DICCIONARIO(NAT, σ)

usa Bool

géneros diccNat(nat, σ)

Operaciones

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

Pre $\equiv \{true\}$

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

Descripción: Crea un nuevo diccionario

Complejidad: $O(1)$

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

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 : diccNat(nat, σ) in n : nat, in s : σ)

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 : diccNat(nat, σ) in n : nat)

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

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

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

Descripción: Se retornan los significados

Complejidad: $O(m)$

Aliasing: Devuelve res por referencia.

DICCCLAVES($\text{in } d : \text{diccNat}(\text{nat } \sigma)) \longrightarrow res : \text{itLista}(\text{nat})$

Pre $\equiv \{true\}$

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

Descripción: Se retorna un iterador al primer elemento de la lista de claves del diccionario

Complejidad: $O(1)$

MAXIMO($\text{in } d : \text{diccNat}(\text{nat } \sigma)) \longrightarrow res : \text{nat}$

Pre $\equiv \{true\}$

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

Descripción: Se retorna la clave maxima en forma de dato

Complejidad: $O(m)$

MINIMO($\text{in } d : \text{diccNat}(\text{nat } \sigma)) \longrightarrow res : \text{nat}$

Pre $\equiv \{true\}$

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

Descripción: Se retorna la clave minima en forma de dato

Complejidad: $O(m)$

4.2 Representación

diccNat

se representa con $\text{tupla}(\text{dicc} : \text{puntero}(\text{estr}(\text{nat } \sigma)),$
 $\text{claves} : \text{lista}(\text{nat}))$

donde $\text{estr}(\text{nat}, \sigma)$ es $\text{tupla}(\text{clave} : \text{nat},$
 $\text{significado} : \sigma,$
 $\text{hijoDer} : \text{puntero}(\text{estr}(\text{nat } \sigma)),$
 $\text{hijoIzq} : \text{puntero}(\text{estr}(\text{nat } \sigma)),$
 $\text{itClaves} : \text{itLista}(\text{nat}))$

donde claves es Lista Enlazada del apunte de modulos basicos que contiene todas las claves del diccionario.

donde itClaves es Iterador bidireccional de lista claves que apunta al elemento correspondiente con su clave.

Invariante de representación

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

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

$\text{Rep}(e) \equiv true \iff ((\forall n_1, n_2 : \text{estr}(\text{nat}, \sigma))(n_1 \in \text{arbol}(e.\text{dicc}) \wedge n_2 \in \text{arbol}(e.\text{dicc}) \Rightarrow_L$
 $(n_1.\text{clave} < n_2.\text{clave} \Rightarrow n_1 \in \text{arbol}(n_2.\text{hijoIzq})) \wedge (n_1.\text{clave} > n_2.\text{clave} \Rightarrow n_1 \in \text{arbol}(n_2.\text{hijoDer})) \wedge$
 $(\forall n_1, n_2 : \text{estr}(\text{nat}, \sigma))(n_1 \in \text{arbol}(e) \wedge n_2 \in \text{arbol}(e) \Rightarrow n_1.\text{clave} \neq n_2.\text{clave} \Rightarrow_L$

$$\begin{aligned}
& (n_1.hijoIzq = n_2.hijoIzq \vee n_1.hijoIzq = n_2.hijoDer \Rightarrow n_1.hijoIzq = NULL) \wedge \\
& (n_1.hijoDer = n_2.hijoIzq \vee n_1.hijoDer = n_2.hijoDer \Rightarrow n_1.hijoDer = NULL) \wedge \\
& ((\forall n : \text{estr}(\text{nat}, \sigma))(n \in \text{arbol}(e.dicc)) \Rightarrow_L \text{Esta?}(e.claves, n.clave)) \wedge \\
& ((\forall k : \text{nat})(k \in e.claves) \Rightarrow (\exists n : \text{estr}(\text{nat}, \sigma))(n \in \text{arbol}(e) \wedge n.clave = k)))
\end{aligned}$$

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 la lista claves y viceversa.

Función de abstracción

$\text{Abs} : \widehat{\text{diccNat}(\text{nat}, \sigma)} d \longrightarrow \widehat{\text{dicc}(\text{nat}, \sigma)} \quad \{\text{Rep}(d)\}$

$$\begin{aligned}
& (\forall d : \widehat{\text{diccNat}(\text{nat}, \sigma)}) \\
& \text{Abs}(d) \equiv c : \widehat{\text{dicc}(\text{nat}, \sigma)} \mid ((\forall k : \text{nat})(k \in \text{claves}(c) \Rightarrow \\
& (\exists n : \text{estr}(\text{nat}, \sigma))(n \in \text{arbol}(d.dicc) \wedge n.clave = k) \wedge (k \in d.Claves)) \wedge \\
& ((\forall k : \text{nat})(k \in d.Claves) \Rightarrow k \in \text{claves}(c)) \wedge \\
& ((\forall n : \text{estr}(\text{nat}, \sigma))(n \in \text{arbol}(d.dicc) \Rightarrow n.clave \in \text{claves}(c)))) \wedge_L \\
& ((\forall n : \text{estr}(\text{nat}, \sigma))(n \in \text{arbol}(d.dicc) \Rightarrow \text{obtener}(c, n.clave) =_{\text{obs}} n.significado))
\end{aligned}$$

$\text{arbol} : \text{puntero}(\text{estr}(\text{nat}, \sigma)) \leftarrow \text{conj}(\text{puntero}(\text{estr}(\text{nat}, \sigma)))$

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

```

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

```

4.3 Algoritmos

$\text{IVACIO}() \longrightarrow \text{res} : \text{estr}$
 $\text{res} \leftarrow \text{NULL}$

O(1)

$\text{IDEFINIR}(\text{in/out } d : \text{diccNat}, \text{ in } n : \text{nat}, \text{ in } s : \sigma)$
 $\text{auxDefinir}(\pi_1(d), \pi_2(d), n, s)$

O(m)

O(m)

$\text{AUXDEFINIR}(\text{in/out } d : \text{estr}, \text{ in } l : \text{lista}(\text{nat}), \text{ in } n : \text{nat}, \text{ in } s : \sigma)$
if $d == \text{NULL}$ **then**
 $\text{res} \leftarrow \langle n, s, \text{NULL}, \text{NULL}, \text{AgregarAtras}(l, n) \rangle$

O(1)

end if	
if $d \neq NULL \wedge n < d.clave$ then	
$d.hijoIzq \leftarrow iDefinir(d.hijoIzq, n, s)$	$O(m)$
end if	
if $d \neq NULL \wedge n > d.clave$ then	
$d.hijoDer \leftarrow iDefinir(d.hijoDer, n, s)$	$O(m)$
end if	
	<hr/>
	$O(m)$
IBORRAR (in/out $d : diccNat$, <i>in</i> $n : nat$)	
$auxBorrar(\pi_1(d), \pi_2(d), n, s)$	$O(m)$
	<hr/>
	$O(m)$
AUXBORRAR (in/out $d : estr$, <i>in</i> $n : nat$)	
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	
$d.itClaves.Anterior.Siguiente \leftarrow d.itClaves.Siguiente$	$O(1)$
$d.itClaves.Siguiente.Anterior \leftarrow d.itClaves.Anterior$	$O(1)$
$Borrar(d)$	$O(1)$
$d \leftarrow NULL$	$O(1)$
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	
$d.hijoDer \leftarrow iBorrar(aux.clave, d.hijoDer)$	
$d.clave \leftarrow aux.clave$	$O(1)$
$d.significado \leftarrow aux.significado$	$O(1)$
$d.itClaves \leftarrow aux.itClaves$	$O(1)$
else	
$aux \leftarrow d.hijoIzq$	$O(1)$
while $aux.hijoDer \neq NULL$ do	$O(m)$
$aux \leftarrow aux.hijoDer$	$O(1)$
end while	
$d.hijoIzq \leftarrow iBorrar(aux.clave, d.hijoIzq)$	
$d.clave \leftarrow aux.clave$	$O(1)$
$d.significado \leftarrow aux.significado$	$O(1)$
$d.itClaves \leftarrow aux.itClaves$	$O(1)$
end if	
	<hr/>
	$O(m)$
ISIGNIFICADO (in/out $d : diccNat$, <i>in</i> $n : nat$) $\longrightarrow res : \sigma$	
$res \leftarrow auxSignificado(\pi_1(d), n)$	$O(m)$
	<hr/>
	$O(m)$
AUXSIGNIFICADO (in/out $d : estr$, <i>in</i> $n : nat$) $\longrightarrow res : \sigma$	
$nodoActual \leftarrow d$	$O(1)$

while $\neg(\text{nodoActual} == \text{NULL}) \wedge \neg \text{res}$ do	$O(m)$
if $\text{nodoActual.clave} == n$ then	
$\text{res} \leftarrow \text{nodoActual.significado}$	$O(1)$
else	
if $c < \text{nodoActual.clave}$ then	
$\text{nodoActual} \leftarrow \text{nodoActual.hijoIzq}$	$O(1)$
else	
$\text{nodoActual} \leftarrow \text{nodoActual.hijoDer}$	$O(1)$
end if	
end if	
end while	
<hr/>	
	$O(m)$
$\text{IDEFINIDO?}(\text{in/out } d : \text{diccNat}, \text{ in } n : \text{nat}) \longrightarrow \text{res} : \text{bool}$	
$\text{res} \leftarrow \text{auxDefinido?}(\pi_1(d), n)$	$O(m)$
<hr/>	
	$O(m)$
$\text{AUXDEFINIDO?}(\text{in/out } d : \text{estr}, \text{ in } n : \text{nat}) \longrightarrow \text{res} : \text{bool}$	
$\text{nodoActual} \leftarrow d$	$O(1)$
$\text{res} \leftarrow \text{FALSE}$	$O(1)$
while $\neg(\text{nodoActual} == \text{NULL}) \wedge \neg \text{res}$ do	$O(m)$
if $\text{nodoActual.clave} == n$ then	
$\text{res} \leftarrow \text{TRUE}$	$O(1)$
else	
if $c < \text{nodoActual.clave}$ then	
$\text{nodoActual} \leftarrow \text{nodoActual.hijoIzq}$	$O(1)$
else	
$\text{nodoActual} \leftarrow \text{nodoActual.hijoDer}$	$O(1)$
end if	
end if	
end while	
<hr/>	
	$O(m)$
$\text{IDICCClaves}(\text{in/out } d : \text{diccNat}, \text{ in } n : \text{nat}) \longrightarrow \text{res} : \text{itLista}(\text{nat})$	
$\text{res} \leftarrow \text{CrearIt}(\pi_2(d))$	
<hr/>	
	$O(m)$
$\text{IMAXIMO}(\text{in/out } d : \text{diccNat}) \longrightarrow \text{res} : \text{nat}$	
if $\pi_1(d).hijoDer \neq \text{NULL}$ then	
$\text{aux} \leftarrow \pi_1(d).hijoDer$	$O(1)$
while $\text{aux.hijoDer} \neq \text{NULL}$ do	$O(m)$
$\text{aux} \leftarrow \text{aux.hijoDer}$	$O(1)$
end while	
$\text{res} \leftarrow \text{aux.clave}$	$O(1)$
else	
$\text{res} \leftarrow \pi_1(d).clave$	$O(1)$
end if	
<hr/>	
	$O(m)$
$\text{IMINIMO}(\text{in/out } d : \text{diccNat}) \longrightarrow \text{res} : \text{nat}$	
if $\pi_1(d).hijoIzq \neq \text{NULL}$ then	

$aux \leftarrow \pi_1(d).hijoIzq$	$O(1)$
while $aux.hijoIzq \neq NULL$ do	$O(m)$
$aux \leftarrow aux.hijoIzq$	$O(1)$
end while	
$res \leftarrow aux.clave$	$O(1)$
else	
$res \leftarrow \pi_1(d).clave$	$O(1)$
end if	
<hr/>	
	$O(m)$

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

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

5.1 Interfaz

se explica con DICCIONARIO($STRING, \sigma$) ITERADOR BIDIRECCIONAL ($TUPLA(STRING, \sigma)$)
géneros $diccString(String, \sigma)$, $itDiccString(String, \sigma)$, $itClavesString$

Operaciones del 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 $\equiv \{d = d_0\}$

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

Descripción: Se define s en el diccionario

Complejidad: $O(Longitud(n))$

Aliasing: s se define por referencia. En caso de ya estar definido n, pisa la definición anterior

BORRAR(**in/out** $d : diccString(String, \sigma)$ **in** $n : String$)

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

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

Descripción: Elimina el elemento n

Complejidad: $O(Longitud(n))$

SIGNIFICADO(**in** $d : diccString(String, \sigma)$ **in** $n : String$) $\longrightarrow res : \sigma$

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

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

Descripción: Se retorna el significado de n

Complejidad: $O(Longitud(n))$

DICCClaves(**in** $d : \text{diccString}(\text{String } \sigma) \longrightarrow res : \text{conj}(\text{string})$)

Pre $\equiv \{TRUE\}$

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

Descripción: Se retorna el conjunto de claves del diccionario

Complejidad: $O(1)$

Aliasing: Res un conjunto devuelto por referencia no modificable.

MAXIMO(**in** $d : \text{diccString}(\text{String } \sigma) \longrightarrow res : \text{String}$)

Pre $\equiv \{true\}$

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

Descripción: Se retorna la clave maxima en orden lexicográfico

Complejidad: $O(\text{longitud}(k))$ donde k es la aplabra más larga del diccionario

MINIMO(**in** $d : \text{diccString}(\text{String } \sigma) \longrightarrow res : \text{String}$)

Pre $\equiv \{true\}$

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

Descripción: Se retorna la clave minima en orden lexicográfico

Complejidad: $O(\text{longitud}(k))$ donde k es la aplabra más larga del diccionario

5.2 Operaciones del iterador

El iterador que presentamos permite modificar el diccionario recorrido. 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). $\text{CREARIT}(\text{in } d : \text{diccString}(\text{String}, \sigma) \text{ in } n : \text{String}) \longrightarrow res : \text{itDiccString}(\text{String}, \sigma)$

Pre $\equiv \{true\}$

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

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

Pre $\equiv \{true\}$

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

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

Complejidad: $O(1)$

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

Pre $\equiv \{true\}$

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

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

Complejidad: $O(1)$

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

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, por aliasing. En cambio, $res.\text{clave}$ no es modificable.

SIGUIENTECLAVE(**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, por aliasing.

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, por aliasing. 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, por aliasing.

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: $O(1)$

5.3 Representacion

`diccString`

se representa con `dLex`)

donde dLex es tupla⟨raiz : puntero(nodo),
claves : conj(string)⟩

nodo

se representa con enodo)

donde enodo es tupla⟨dato : σ ,
esSig? : Bool,
claveEnConj : itConj(String),
continuaciones : puntero(char) \square 256 \square ⟩

1. conj(string) es el Conjunto Lineal de los modulos Básicos. itConj(string) es el iterador del mismo

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.
3. En el conjunto claves sólo se encuentran definidas las claves del diccionario y se encuentran todas ellas

Función de abstracción

Abs : diccString(string) $d \longrightarrow \text{dicc}(String\sigma) \quad \{\text{Rep}(d)\}$

Abs(d) \equiv AbsAux(d d.claves)

AbsAux : diccString(String) $d \text{conj}(String)/c \longrightarrow \text{dicc}(String\sigma)\{\text{Rep}(d) \wedge c \subseteq d.claves\}$

AbsAux(d,c) \equiv **if** $\emptyset?(c)$ **then**
 \emptyset
else
 $efinir(dameUno(c), significado(dameUno(c), d), AbsAux(d, sinUno(c)))$
fi

Representación del iterador

El iterador del diccionario lo recorre en orden lexicográfico. Los significados están por referencia. Se explica con el iterador bidireccional no modificable. itDiccString(*String*, σ)

se representa con itdLex)

donde dLex es tupla⟨claves : Lista(String),
significados : Lista(σ)⟩

5.4 Algoritmos

5.4.1 Algoritmos del Diccionario

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

$res.raiz \leftarrow NULL$	$O(1)$
$res.claves \leftarrow Vacio$	$O(1)$
<hr/>	
	$O(1)$
IDEFINIR (in/out $d : diccString$, $in\ n : String$, $in\ s : \sigma$)	
$aux \leftarrow d.raiz$	$O(1)$
$i \leftarrow 0$	$O(1)$
while $i < Longitud(n)$ do	$O(Longitud(n))$
if $aux == NULL$ then	
$nNodo.esSig? \leftarrow false$	$O(1)$
$j \leftarrow 0$	
while $j < 256$ do	$O(256)=O(1)$
$nNodo.continuaciones[j] \leftarrow NULL$	$O(1)$
end while	
$aux \leftarrow \&nNodo$	$O(1)$
end if	
$aux \leftarrow aux.continuaciones[ord(n[i])]$	$O(1)$
end while	
$aux.esSig? \leftarrow True$	$O(1)$
if $aux.esSig? \neq True$ then	
$aux.claveEnConj \leftarrow AgregarRapido(d.claves, n)$	$O(long(n))$
end if	
$aux.esSig? \leftarrow True$	$O(1)$
$aux.dato \leftarrow s$	$O(1)$
el último paso se realiza por referencia	
<hr/>	
	$O(Longitud(n))$
IDEFINIDO? (in/out $d : diccString$, $in\ n : String$) $\rightarrow res : bool$	
$aux \leftarrow d.raiz$	$O(1)$
$i \leftarrow 0$	$O(1)$
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 $d : diccString$, $in\ n : String$) $\rightarrow res : \sigma$	
$aux \leftarrow d.raiz$	$O(1)$
$i \leftarrow 0$	$O(1)$
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 ← 0                                O(1)
pila(puntero(nodo))p ← Vacia()      O(1)
while i < Longitud(n) do            O(Longitud(n))
    Apilar(p,aux)                     O(1)
    aux ← aux.continuaciones[ord(n[i])] O(1)
    i ← i + 1
end while
aux * .esSig? ← false
BorrarSiguiente(aux * .claveEnConj)
aux * esSig? ← false
i ← i − 1
j ← 0
while aux * .continuaciones[j] = NULL ∧ j < 256 do    O(256)=O(1)
    j ← j + 1
end while
if j < 256 then
    p ← Vacia()
end if
while ¬EsVacia?(p) do
    j ← 0
    Tope(p) * .continuaciones[ord(n[i])] ← NULL
    while Tope(p) * .continuaciones[j] = NULL ∧ j < 256 do    O(256)=O(1)
        j ← j + 1
    end while
    if j < 256 then
        p ← Vacia()
    else
        Desapilar(p)
        i ← i − 1
    end if
end while

```

O(*Longitud*(*n*))

IDICCClaves(**in/out** *d* : diccString(String *σ*)) → *res* : conj(String)
res ← *d.claves*

O(1), *d.claves* se pasa por referencia

MAXIMO(**in/out** *d* : diccString(String *σ*)) → *res* : String
aux ← *d.raiz* O(1)
res ← *Vacia*()
Boolf ← *True*
while *f* **do**
nati ← 256
while *aux* * *.continuaciones*[*i* − 1] = *NULL* ∧ *i* > 0 **do**
i ← *i* + 1
end while
if *i* = 0 **then**
f ← *false*
else
AgregarAtras(*res*, *ord*^{−1}(*i* − 1))
i ← 0
end if

end while

O(1) d.claves se pasa por referencia

MINIMO(in/out $d : \text{diccString}(\text{String } \sigma) \rightarrow res : \text{String}$

$aux \leftarrow d.raiz$

$res \leftarrow \text{Vacía}()$

$Bool f \leftarrow \text{True}$

while f **do**

$nati \leftarrow 0$

while $aux * .continuciones[i] = \text{NULL} \wedge i < 256$ **do**

$i \leftarrow i + 1$

end while

if $i = 256$ **then**

$f \leftarrow \text{false}$

else

$\text{AgregarAtras}(res, ord^{-1}(i))$

$i \leftarrow 0$

end if

end while

O(1) d.claves se pasa por referencia

5.4.2 Algoritmos del iterador

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

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

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

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

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

$res.claves \leftarrow cs$

$res.significados \leftarrow ss$

O(Longitud(k)*tam(d))

O(Longitud(k)*tam(d))

1. k es la palabra más larga definida en d y tam(d) es la cantidad de palabras definidas que hay¹.

AUXCRIT(in/out $cs : \text{Lista}(\text{string})$ in/out $ss : \text{Lista}(\sigma)$ in/out $n : \text{String}$ in $p : \text{puntero}(\text{nodo})$)

if $p \neq \text{NULL}$ **then**

if $p * .esSig?$ **then**

$\text{AgregarAtras}(cs, n)$

$\text{AgregarAtras}(ss, p * .dato)$

end if

$i \leftarrow 0$

while $i < 256$ **do**

$\text{AgregarAtras}(n, ord^{-1}(i))$

$auxCrIt(cs, ss, n, p * .continuciones[i])$

$\text{TirarUltimos}(n, 1)$

end while

O(1)

¹Si bien no nos fue posible realizar el calculo de complejidad correspondiente, el algoritmo recursivo recorre una vez cada nodo, y el total de nodos esta acotado por la sumatoria del largo de cada palabra, que a su vez está acotada por la cantidad de palabras multiplicada por la longitud de la palabra más larga, por lo que la cota propuesta a la complejidad es razonable

end if	<hr/>	Tn desconocido
IHAYSIGUIENTE (in/out <i>it</i> : itDiccString, <i>in n</i> : String) \longrightarrow <i>res</i> : bool <i>res</i> \leftarrow <i>it.claves.siguiente</i> \neq NULL	<hr/>	O(1)
IHAYANTERIOR (in/out <i>it</i> : itDiccString, <i>in n</i> : String) \longrightarrow <i>res</i> : bool <i>res</i> \leftarrow <i>it.claves.anterior</i> \neq NULL	<hr/>	O(1)
ISIGUIENTE (in/out <i>it</i> : itDiccString, <i>in n</i> : String) \longrightarrow <i>res</i> : tupla(string, σ) <i>res.clave</i> \leftarrow <i>it.claves.siguiente</i> * .dato <i>res.significado</i> \leftarrow <i>it.significados.siguiente</i> * .dato	<hr/>	O(1)(es una referencia) O(1)(es una referencia)
	<hr/>	O(1)
ISIGUIENTECLAVE (in/out <i>it</i> : itDiccString, <i>in n</i> : String) \longrightarrow <i>res</i> : String <i>res</i> \leftarrow <i>it.claves.siguiente</i> * .dato	<hr/>	O(1)(es una referencia)
	<hr/>	O(1)
ISIGUIENTESIGNIFICADO (in/out <i>it</i> : itDiccString, <i>in n</i> : String) \longrightarrow <i>res</i> : σ <i>res</i> \leftarrow <i>it.significados.siguiente</i> * .dato	<hr/>	O(1)(es una referencia)
	<hr/>	O(1)
IANTERIOR (in/out <i>it</i> : itDiccString, <i>in n</i> : String) \longrightarrow <i>res</i> : tupla(string, σ) <i>res.clave</i> \leftarrow <i>it.claves.anterior</i> * .dato <i>res.significado</i> \leftarrow <i>it.significados.anterior</i> * .dato	<hr/>	O(1)(es una referencia) O(1)(es una referencia)
	<hr/>	O(1)
IANTERIORCLAVE (in/out <i>it</i> : itDiccString, <i>in n</i> : String) \longrightarrow <i>res</i> : String <i>res</i> \leftarrow <i>it.claves.anterior</i> * .dato	<hr/>	O(1)(es una referencia)
	<hr/>	O(1)
IANTERIORESIGNIFICADO (in/out <i>it</i> : itDiccString, <i>in n</i> : String) \longrightarrow <i>res</i> : σ <i>res</i> \leftarrow <i>it.significados.anterior</i> * .dato	<hr/>	O(1)(es una referencia)
	<hr/>	O(1)
IAVANZAR (in/out <i>it</i> : itDiccString, <i>in n</i> : String) <i>it.claves</i> \leftarrow <i>it.claves.siguiente</i> <i>it.significados</i> \leftarrow <i>it.significados.siguiente</i>	<hr/>	O(1)(es una referencia) O(1)(es una referencia)
	<hr/>	O(1)
IRETROCEDER (in/out <i>it</i> : itDiccString, <i>in n</i> : String) <i>it.claves</i> \leftarrow <i>it.claves.anterior</i> <i>it.significados</i> \leftarrow <i>it.significados.anterior</i>	<hr/>	O(1)(es una referencia) O(1)(es una referencia)
	<hr/>	O(1)

6 Modulo Campo es String

7 Modulo Registro

7.1 Interfaz

se explica con REGISTRO

géneros reg

Operaciones

Cuando se utiliza conj se está utilizando el conjunto lineal provisto por la cátedra. En las complejidades C denota el campo de mayor longitud de un conjunto o registro al que acompaña.

$\text{NREG}() \longrightarrow res : \text{reg}$

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

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

Descripción: Crea un registro nuevo, vacío

Complejidad: $O(1)$

$\text{DEFINIDO?}(\text{in } r : \text{reg in } c : \text{campo}) \longrightarrow res : \text{Bool}$

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

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

Descripción: Indica si el campo está definido

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

$\text{DEFINIR}(\text{in/out } r : \text{reg in } c : \text{campo, in } d : \text{dato})$

Pre $\equiv \{r = r_0\}$

Post $\equiv \{d =_{\text{obs}} \text{Definir}(c, d, r_0)\}$

Descripción: Se define s en el diccionario

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

Aliasing: d se define por referencia

$\text{BORRAR}(\text{in/out } r : \text{reg in } c : \text{campo})$

Pre $\equiv \{r =_{\text{obs}} r_0 \wedge \text{def?}(c, r)\}$

Post $\equiv \{r =_{\text{obs}} \text{Borrar}(c, r_0)\}$

Descripción: Elimina el campo c

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

$\text{SIGNIFICADO}(\text{in } r : \text{reg in } c : \text{campo}) \longrightarrow res : \text{dato}$

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

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

Descripción: Se retorna el significado de c

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

$\text{CAMPOS}(\text{in } r : \text{reg}) \longrightarrow res : \text{conj}(\text{campo})$

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

Post $\equiv \{res =_{\text{obs}} \text{alias}(res, \text{claves}(r))\}$

Descripción: Devuelve un conjunto de campos que son claves del registro ingresado por parámetro

Complejidad: $O(1)$

Aliasing: Se devuelve el conjunto por referencia, hay Aliasing

$\text{BORRAR?}(\text{in } crit : \text{reg, in } r : \text{reg}) \longrightarrow res : \text{bool}$

Pre $\equiv \{\#campos(crit) = 1\}$

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

Descripción: Devuelve true si y solo si todos los campos de crit pertenecen a campos de r.

Complejidad: $O(\text{Longitud}(\text{dameUno}(\text{campos}(crit))))$

COINCIDEALGUNO(**in** $r_1 : \text{reg}$, **in** $cc : \text{conj}(\text{campo})$, **in** $r_2 : \text{reg}$) $\longrightarrow res : \text{bool}$

Pre $\equiv \{cc \subseteq \text{campos}(r_1) \cap \text{campos}(r_2)\}$

Post $\equiv \{res =_{\text{obs}} \text{coincideAlguno}(r_1, r_2)\}$

Descripción: Devuelve true si y solo si alguno de los campos(dato) de cc pertenece a r1 y r2

Complejidad: $O(\#cc * C)$

COINCIDENTODOS(**in** $r_1 : \text{reg}$, **in** $cc : \text{conj}(\text{campo})$, **in** $r_2 : \text{reg}$) $\longrightarrow res : \text{bool}$

Pre $\equiv \{cc \subseteq \text{campos}(r_1) \cap \text{campos}(r_2)\}$

Post $\equiv \{res =_{\text{obs}} \text{coincidenTodos}(r_1, cc, r_2)\}$

Descripción: Devuelve true si y solo si todos los campos(dato) de cc pertenecen a r1 y r2

Complejidad: $O(\#cc * C)$

ENTODOS(**in** $c : \text{campo}$, **in** $cr : \text{conj}(\text{registro})$) $\longrightarrow res : \text{bool}$

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

Post $\equiv \{res =_{\text{obs}} \text{enTodos}(c, cr)\}$

Descripción: Devuelve true si y solo si campo c pertenece a los campos de cada uno de los registros cr

Complejidad: $O(\#(cr))$

UNIRREGISTROS(**in** $c : \text{campo}$, **in** $r_1 : \text{reg}$, **in** $r_2 : \text{reg}$) $\longrightarrow res : \text{registro}$

Pre $\equiv \{c \in \text{campos}(r_1) \wedge c \in (\text{campos}(r_2))\}$

Post $\equiv \{res =_{\text{obs}} \text{combinarTodos}(c, r_1, \text{ag}(\emptyset, r_2))\}$

Descripción: Devuelve el registro que combina los valores de r_1 y r_2

Complejidad: $O(\#\text{campos}(r_1) * C)$

Aliasing: r_1 y r_2 son tomados por referencia

COMBINARTODOS(**in** $c : \text{campo}$, **in** $r : \text{reg}$, **in** $cr : \text{conj}(\text{reg})$) $\longrightarrow res : \text{registro}$

Pre $\equiv \{c \in \text{campos}(r_1) \wedge \text{enTodos}(c, cr)\}$

Post $\equiv \{res =_{\text{obs}} \text{combinarTodos}(c, r_1, cr)\}$

Descripción: Devuelve el registro que combina los valores de r_1 y un registro apropiado de cr

Complejidad: $O(\#\text{campos}(r_1) * C * \#(cr))$

Aliasing: r_1 y r_2 son tomados por referencia

7.2 Representación

se representa con dlex

donde dlex es subdic: $\text{diccString}(\text{campo}, \text{dato})$

Invariante de representación

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

$(\forall d : \widehat{\text{Dicc}})$

$\text{Rep}(d) \equiv \text{true}$

Función de abstracción

$\text{Abs} : \widehat{\text{reg}} d \longrightarrow \widehat{\text{reg}}$

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

$$(\forall d : \widehat{\mathbf{reg}})$$

$$\text{Abs}(d) \equiv r : \widehat{\mathbf{reg}} \mid \text{Abs}(\text{dlex})$$

7.3 Algoritmos

IDEFINIR(in $r : \text{reg}$, $in\ c : \text{campo}$, $in\ d : \text{dato}$) $Definir(r, c, d)$	$O(\text{Longitud}(c))$
	<hr/>
	$O(\text{Longitud}(c))$
IDEFINIDO?(in $r : \text{reg}$, $in\ c : \text{campo}$) $\rightarrow res : \text{bool}$ $res \leftarrow Definido?(r, c)$	$O(\text{Longitud}(c))$
	<hr/>
	$O(\text{Longitud}(c))$
ISIGNIFICADO(in $r : \text{reg}$, $in\ c : \text{campo}$) $\rightarrow res : \text{dato}$ $res \leftarrow significado(c, r.subdic)$	$O(\text{Longitud}(c))$
	<hr/>
	$O(\text{Longitud}(c))$
IBORRAR(in/out $r : \text{reg}$, $in\ c : \text{campo}$) $Borrar(r.subdic)$	$O(\text{Longitud}(c))$
	<hr/>
	$O(\text{Longitud}(c))$
CAMPOS(in $r : \text{reg}$) $\rightarrow res : \text{Conj}(\text{campo})$ $res \leftarrow DiccClaves(r.subdic)$	$O(1)$
	<hr/>
	$O(1)$
BORRAR?(in $crit : \text{reg}$ in $r : \text{reg}$) $\rightarrow res : \text{bool}$ $res \leftarrow coincidenTodos(crit, campos(crit), r)$	$O(\text{Longitud}(\text{dameUno}(\text{campos}(crit))))$
	<hr/>
	$O(\text{Longitud}(\text{dameUno}(\text{campos}(crit))))$
ICOINCIDEALGUNO(in $r1 : \text{reg}$ in $cc : \text{conj}(\text{campo})$ in $r2 : \text{reg}$) $\rightarrow res : \text{bool}$ $it \leftarrow CrearIt(cc)$ $res \leftarrow false$ while ($\neg res$) $\wedge HaySiguierte(it)$ do $res \leftarrow Significado(r1, Siguierte(it)) = Significado(r2, Siguierte(it))$ end while	$O(1)$ $O(1)$ $O(\#cc)$ $O(\text{longitud}(\text{siguierte}(it)))=O(C)$
	<hr/>
	$O(\#cc * C)$
ICOINCIDENTODOS(in $r1 : \text{reg}$ in $cc : \text{conj}(\text{campo})$ in $r2 : \text{reg}$) $\rightarrow res : \text{bool}$ $it \leftarrow CrearIt(cc)$ $res \leftarrow true$ while $res \wedge HaySiguierte(it)$ do $res \leftarrow Significado(r1, Siguierte(it)) = Significado(r2, Siguierte(it))$ end while	$O(1)$ $O(1)$ $O(\#cc)$ $O(\text{longitud}(\text{siguierte}(it)))=O(C)$
	<hr/>
	$O(\#cc * C)$

Donde C es la longitud del campo más largo en cc

IENTODOS(in $c : \text{campo}$ in $cr : \text{conj}(\text{reg})$) $\rightarrow res : \text{bool}$ $res \leftarrow True$ $it \leftarrow CrearIt(cr)$ while $haySiguierte(it) \wedge res$ do $res \leftarrow Definido?(c, siguierte(it))$	$O(1)$ $O(1)$ $O(\#(cr))$ $O(1)$
---	---

Avanzar(it)	
end while	
	<hr/>
	$O(\#(cr))$
I Combinar Todos(in $c : \text{campo}$ in $r_1 : \text{reg}$ in $cr : \text{conj}(\text{reg})$) $\longrightarrow res : \text{reg}$	
it \leftarrow CrearIt(cr)	$O(1)$
Bool f \leftarrow true	
while haySiguiente(it) \wedge f do	$O(\#(cr))$
if Significado(r,c)=Significado(Siguiente(it),c) then	$O(1)$
res \leftarrow unirRegistros(c,r, siguiente(it))	$O(\#campos(r_1) * C)$
f \leftarrow false	$O(1)$
end if	
Avanzar(it)	
end while	
	<hr/>
	$O(\#campos(r_1) * C * \#(cr))$
I Unir Registros(in $c : \text{campo}$ in $r_1 : \text{reg}$ in $r_2 : \text{reg}$) $\longrightarrow res : \text{reg}$	
res \leftarrow r ₂	
it \leftarrow CrearIt(campos(r ₁))	
while HaySiguiente(it) do	$O(\#campos(r_1))$
Definir(res, Siguiente(it), Significado(r ₁ , Siguiente(it)))	$O(\text{longitud}(\text{siguiente}(it)))=O(C)$
end while	
	<hr/>
	$O(\#campos(r_1) * C)$

8 NombreTabla es String

9 Base de Datos

9.1 Interfaz

se explica con BASEDEDATOS, NAT, STRING,

tabla, registro, campo, dato, DiccString(String, alfa), DiccNat(Nat, beta)

géneros **base**

Operaciones

TABLAS(in $b : \text{base}$) $\longrightarrow res : \text{ItConjString}(\text{NombreTabla})$

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

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

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 $t : \text{string}$, in $b : \text{base}$) $\longrightarrow res : \text{tabla}$

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

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

Descripción: Devuelve la tabla correspondiente al nombre ingresado por parametro.

Complejidad: $O(1)$

Aliasing: Se devuelve la tabla correspondiente 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 indices de la tabla ingresada por parametro.

Complejidad: $O(1)$

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: $O(1)$

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: $O(1)$

Aliasing: Agrega tabla por referencia.

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

Pre $\equiv \{b_0=b \wedge t \in \text{tablas}(b) \wedge \text{puedoInsertar}?(dameTabla(t)\ reg)\}$

Post $\equiv \{\text{insertarEntrada}(rt\ b_0)\}$

Descripción: Inserta el registro a la tabla t .

Complejidad: $O(T)$

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

Pre $\equiv \{b_0=b \wedge t \in \text{tablas}(b) \wedge \#(\text{DiccClaves}(cr))=1\}$

Post $\equiv \{\text{borrar}(cr\ t\ b_0)\}$

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

Complejidad: $O(T + \text{Log}(n))$

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

Pre $\equiv \{b_0=b \wedge t1 \sqsubseteq t2 \wedge \{t1, t2\} \subseteq \text{tablas}(b) \wedge$

$\text{Pertenece}?(Campos(dameTabla(t1, b)), c) \wedge \text{Pertenece}?(Campos(dameTabla(t1, b)), c) \wedge$
 $\neg \text{hayJoin}(t1, t2, b)\}$

Post $\equiv \{\text{generarVistaJoin}(cr, t, b_0)\}$

Descripción: Genera el Join, entre las tablas pasadas por parametro.

Complejidad: $O((n + m) * (L + \text{Log}((n + m))))$

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 el join entre tablas, pasadas por parametro.

Complejidad: $O(1)$

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

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

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

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

Complejidad: $O(1)$ por ref

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: $O(R * (L + \text{Log}(n * m)))$

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

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

Complejidad: $O(1)$ por ref

Aliasing: Se retorna res por referencia.

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

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

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

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

Complejidad: $O(1)$ por ref

Aliasing: Se retorna el nombre de la tabla por referencia, no es modificable.

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 $\equiv \{res =_{\text{obs}} \text{tablaMaxima}(tb)\}$

Descripción: Retorna el nombre de alguna de las tablas con mayor cantidad de accesos.

Complejidad: $O(1)$

Aliasing: Se retorna el nombre de la tabla por referencia, no es modificable.

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

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

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

Descripción: Retorna el conjunto de los iteradores de registro al conjunto de registros de la tabla.

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

Aliasing: Se retorna por referencia, es modificable.

9.2 Representación

se representa con Base

```
donde estr es tupla(TablaMaxima : Tmax,  
                    Tablas : DiccTrie(NombreTabla; info_tabla))  
donde info_tabla es tupla(TActual : tabla,  
                          Joins : DiccTrie(NombreTabla; info_join))  
donde info_join es tupla(Rcambios : Cola(DatoCambio),  
                        campoJ : campo,  
                        campoT : tipo,  
                        JoinS : DiccTrie(string; itConj(registro)),  
                        JoinN : DiccNat(nat; itConj(registro)),  
                        JoinC : Conj(registro))  
donde Tmax es tupla(NomTabla : NombreTabla,  
                    #Modif : Nat)  
donde DatoCambio es tupla(Reg : Registro,  
                          NomOrigen : NombreTabla,  
                          Accion : Bool)
```

Invariante de representación

1. El Nombre de las tablas es un String acotado.
2. Índices es un arreglo de tamaño 2, que aloja el Índice correspondiente según el orden de creación.
3. Para toda Dato que es clave en Índice, su significado llamémoslo sign está incluido en Registros.
4. El nombre de la TablaMaxima está definido en Tablas.
5. El nombre de la TablaMaxima corresponde a una del conjunto de tablas más modificadas.
6. En Tmax #Modif es igual a la cantidad de modificaciones de la tabla NomTabla.
7. #Modif de TablaMaxima es mayor o igual a todas las cantidades de modificaciones de las tablas en el conjunto de claves de Tablas.
8. Cada NombreTabla definido en Tablas es igual al nombre de TActual en info_tabla de su significado.
9. Todas las claves de Joins, de todos los significados de las tablas definidas en Tablas, pertenecen al conjunto de claves de Tablas.
10. Para todo info_tabla, el nombre de TActual no está definido en Joins.
11. para todo info_tabla, el nombre de TActual pertenece al conjunto de claves del Joins de las claves de su Joins.
12. Para todo info_join, el tipo de campoJ es igual a campoT.
13. Para todo DatoCambio, NomOrigen es igual a TActual o a la clave que le corresponde en Joins.
14. Si campoT es true JoinS esta vacío, y si es false JoinN esta vacío.
15. En DatoCambio, si Accion es true entonces Reg pertenece al conjunto de registros de NomOrigen.

16. En DatoCambio, si Accion es false entonces Reg no pertenece al conjunto de registros de NomOrigen.

Función de abstracción

$$\text{Abs} : \widehat{\text{Base}} \, b \longrightarrow \widehat{\text{BaseDeDatos}} \quad \{\text{Rep}(b)\}$$

$$(\forall b : \widehat{\text{Base}})$$

$$\text{Abs}(b) \equiv cb : \widehat{\text{BaseDeDatos}} \mid$$

$$\text{Tablas}(b) =_{\text{obs}} \text{tablas}(cb) \wedge$$

$$((\forall n : \text{NombreTabla}) \text{Pertenece?}(n, \text{Tablas}(b)) \Rightarrow_{\text{L}} \text{DameTabla}(n, b) =_{\text{obs}} \text{dameTabla}(n, cb))$$

$$((\forall n1, n2 : \text{NombreTabla}) (\text{Definido?}(n1, b.\text{tablas}) \wedge \text{Definido?}(n2, b.\text{tablas})) \Rightarrow_{\text{L}} \text{HayJoin?}(n, b) =_{\text{obs}}$$

$$\text{HayJoin?}(n, cs) \wedge_{\text{L}} \text{campoJoin}(n1, n2, b) = \text{campoJoin}(n1, n2, cb))$$

9.3 Algoritmos

TABLAS(in $b : \text{estr}$) $\longrightarrow res : \text{ConjTrie}(\text{string})$ $res \leftarrow \text{DiccClaves}(b.tablas)$	O(1) por ref
	O(1) por ref
DAMETABLA(in $t : \text{string}$, in $b : \text{estr}$) $\longrightarrow res : \text{tabla}$ $info \leftarrow \text{Significado}(b.tablas, t)$ $res \leftarrow info.TActual$	O(1) por ref O(1) por ref
	O(1) por ref
HAYJOIN?(in $t1 : \text{string}$, in $t2 : \text{string}$, in $b : \text{estr}$) $\longrightarrow res : \text{bool}$ $res \leftarrow (\text{Definido?}(\text{Obtener}(b, t1).Joins, t2) \leftarrow \text{Definido?}(\text{Obtener}(b, t2).Joins, t1))$	O(1) por ref
	O(1) por ref
CAMPOJOIN(in $t1 : \text{string}$, in $t2 : \text{string}$, in $b : \text{estr}$) $\longrightarrow res : \text{campo}$ $res \leftarrow \text{Obtener}(\text{Obtener}(b, t1).Joins, t2).campoJ$	O(1) por ref
	O(1) por ref
NUEVADB() $\longrightarrow res : \text{estr}$ String s Nat $n \leftarrow 0$ TablaMaxima $\leftarrow \langle s, 0 \rangle$ $res \leftarrow \langle \text{TablaMaxima}, \text{vacio}() \rangle$	O(1) O(1) O(1) por ref O(1) por ref
	O(1) por ref
AGREGARTABLA(in $t : \text{tabla}$, in/out $b : \text{estr}$) $info_tabla \leftarrow \langle \text{cantidadDeAccesos}(t), t, \text{vacio}() \rangle$ Definir($b.tablas$, nombre(t), $info_tabla$)	O(1) por ref O(1) por ref
	O(1) por ref
INSERTARENTRADA(in $reg : \text{registro}$, in $t : \text{string}$, in/out $b : \text{estr}$) Obtenemos la tabla es O(1) por ref porque su nombre esta acotado. $info_tabla \text{ infoT} \leftarrow \text{Obtener}(b.tablas, t).TActual$ Agrego el registro a la tabla. $Tabla \text{ T} \leftarrow infoT.TActual$ agregarRegistro(reg , T) Ahora si hay Joins actualizo la informacion temporal de cada Join. if $\neg \text{Vacio?}(infoT.Joins)$ then $ItConjString(\text{String}) \text{ itNomTab} \leftarrow \text{CrearIt}(\text{Claves}(infoT.Joins))$ while HaySiguiente?($itNomTab$) do $info_join \text{ infoJ} \leftarrow \text{Obtener}(infoT.Joins, \text{Siguiente}(\text{NomTab}))$ Encolar($infoJ.Rcambios$, $\langle reg, \text{Siguiente}(\text{NomTab}), true \rangle$) Encolar es O(L) porque se copia un registro con su cantidad de campos acotada y los valores string copiarlos tiene costo O(L), siendo L el valor string mas largo. Avanzar($itClaves$) end while	O(1) por referencia O(1) por ref O(1) por ref O(L+Log(n)) O(1) por ref O(1) por ref O(W) O(1) por ref O(1) por ref O(1) por ref

end if

if CantidadDeAccesos(T) b.TablaMaxima.#Modif **then**

b.TablaMaxima.NomTabla \leftarrow Copiar(Nombre(T)) O(L)

b.TablaMaxima.#Modif \leftarrow Copiar(CantidadDeAccesos(T)) O(1) por ref

end if

W la cant de tablas en la base

O(W)

BORRAR(**in** cr : registro, **in** t : string, **in/out** b : estr)

info_tabla infoT \leftarrow Obtener(b.tablas, t).TActual O(1) por ref

Tabla T \leftarrow infoT.TActual O(1) por ref

NombreTabla NomTab \leftarrow NombreTabla(T)

La eliminacion en primera etapa depende de si hay
joins con la tabla pasada por parametro

if $\neg \emptyset?$ (Claves(infoT.Joins)) **then** O(1) por ref

Creo el iterador, para navegar los nombres de tablas con los que tiene Join

itNom \leftarrow CrearIt(Claves(infoT.Joins)) O(1) por ref

while HaySiguiente?(itNom) **do** O(Cant de tablas)

info_join infoJ \leftarrow Siguiente(itNom) O(1) por ref

Verifico si el Join esta creado en base al
campo del cr pasado por parametro.

if infoJ.campoJ=DameUno(Campos(cr)) **then** O(1) por ref

Entonces solo actualizo la cola temporal del join

Registro reg \leftarrow Buscar(cr, T) O(L + Log(n))

Encolar(infoJ.Rcambios, \langle reg, Siguiente(itNom), False \rangle)

O(L)

else

Si el campo del criterio de borrado, es distinto que el campo del Join.

Primero busco que registros coinciden con el criterio

en el peor caso con complejidad O(cantidad de registros de t) sin indices.

Conj(Registro) cjc \leftarrow Buscar(cr, T) O(L + Log(n))

ItConj(Registro) itReg \leftarrow CrearIt(cjc)

Luego agrego estos registros a la cola temporal de cambios del join

while HaySiguiente?(itReg) **do**

Encolar(infoJ.Rcambios, \langle Siguiente(itReg), NomTab, False \rangle)

O(L)

Avanzar(itReg) O(1) por ref

end while

end if

end while

end if

Habiendo actualizado las colas temporales de los joins con los
registros que cumplen con el criterio de borrado
de los joins correspondientes.

Elimino del conjunto de registros, aquellos que cumplen el criterio de borrado.

Siendo n la cantidad de registros de t y T la cantidad de tablas en la base

En peor caso con costo O(n)

borrarRegistro(r, T_actual) O(T*L + n)

if CantidadDeAccesos(T) b.TablaMaxima.#Modif **then**

b.TablaMaxima.NomTabla \leftarrow Copiar(Nombre(T)) O(L)

b.TablaMaxima.#Modif \leftarrow Copiar(CantidadDeAccesos(T)) O(1) por ref

end if

$O(T + \text{Log}(n))$

GENERARVISTAJOIN(**in** $t1$: string, **in** $t2$: string, **in** c : campo, **in/out** b : estr)

Cola(DatoCambio) Rcambio *vacía*()

Campo campoJ \leftarrow

Tipo campoT \leftarrow tipoCampo(c , Ta1)

DiccTrie(string; itConj(registro)) JoinS \leftarrow Vacio()

DiccNat(nat; itConj(registro)) JoinN \leftarrow Vacio()

Conj(Registro) JoinC \leftarrow vacio()

$O(1)$ por ref

Tabla Ta1 \leftarrow Obtener(b .tablas, $t1$).Tactual

$O(1)$ por ref

Tabla Ta2 \leftarrow Obtener(b .tablas, $t2$).Tactual

$O(1)$ por ref

Nat $n \leftarrow$ Cardinal(Registros(Ta1))

$O(1)$ por ref

Nat $m \leftarrow$ Cardinal(Registros(Ta2))

$O(1)$ por ref

Conj(dato) $cjd1 \leftarrow$ dameColumna(c , Registros(Ta1))

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

Conj(dato) $cjd2 \leftarrow$ dameColumna(c , Registros(Ta2))

$O(m \text{Log}(m))$

Ahora busco la interseccion de estos datos entre los conjuntos $cjd1$ y $cjd2$

ItConj(dato) itcjd1 \leftarrow CrearIt($cjd1$)

$O(1)$ por ref

DiccString(String, bool) $ds \leftarrow$ vacio()

$O(1)$ por ref

DiccNat(Nat, bool) $dn \leftarrow$ vacio()

$O(1)$ por ref

Conj(Dato) $cjD \leftarrow$ vacio()

Cargo los valores de $t1$ en un dicc que depende del tipo del campo c

if \neg campoT **then**

$O(1)$ por ref

El tipo del campo es string

while HaySiguiente?(itcjd1) **do**

$O(n)$

Sabemos que el dameColumna no tiene valores repetidos

Definir(ds , ValorString(Siguiente(itcjd1)), true)

$O(1)$ por ref

Avanzar(itcjd1)

$O(1)$ por ref

end while

ItConj(dato) itcjd2 \leftarrow CrearIt($cjd2$)

$O(1)$ por ref

Ahora buscamos la interseccion

while HaySiguiente?(itcjd2) **do**

$O(m)$

Sabemos que el dameColumna no tiene valores repetidos

Bool Def \leftarrow Definido?(ds , ValorString(Siguiente(itcjd2)))

$O(1)$ por ref

if Def **then** AgregarRapido(cjD , Siguiente(itcjd2))

$O(1)$ por ref

end if

Avanzar(itcjd1)

$O(1)$ por ref

end while

else

El tipo del campo es nat

while HaySiguiente?(itcjd1) **do**

$O(n * \text{log}(n))$

Sabemos que el dameColumna no tiene valores repetidos

Definir(dn , ValorNat(Siguiente(itcjd1)), true)

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

Avanzar(itcjd1)

$O(1)$ por ref

end while

ItConj(dato) itcjd2 \leftarrow CrearIt($cjd2$)

$O(1)$ por ref

Ahora buscamos la interseccion

while HaySiguiente?(itcjd2) **do**

$O(m * \text{Log}(n))$

Sabemos que el dameColumna no tiene valores repetidos

Bool Def \leftarrow Definido?(dn , ValorString(Siguiente(itcjd2)))

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

if Def **then** AgregarRapido(cjD , Siguiente(itcjd2))

$O(1)$ por ref

end if	
Avanzar(itcjd2)	$O(1)$ por ref
end while	
end if	
Si hay algun valor en la interseccion, con esos valores hago el join.	
if \neg Vacio?(cjD) then	$O(1)$ por ref
Caso Strings	
its \leftarrow CrearIt(cjD)	$O(1)$ por ref
while HaySiguiente?(its) do	$O((n+m)*(L+\text{Log}(n*m)))$
Registro regModelo \leftarrow Definir(vacio(), c, Siguiente(its))	
Como el campo es clave en ambas tablas	
Si en ambas tablas hay indice en base a c y la complejidad es	
$O(L+\text{Log}(n*m))$ sino es $O(\text{cardinal de registros de la tabla})$	
Asumimos que es el mejor caso para la complejidad.	
Registro r1 Siguiente(cj1) \leftarrow BuscarEnTabla(regModelo, ta1)	
	$O(L+\text{Log}(n*m))$
Registro r2 Siguiente(cj1) \leftarrow BuscarEnTabla(regModelo, ta1)	
	$O(L+\text{Log}(n*m))$
if \neg campoT then	
itConj(Registro) itnuevo AgregarRapido(JoinC, UnirRegistros(r1, r2))	$O(1)$ por ref
Definir(JoinS, ValorString(Siguiente(its)), itnuevo)	$O(1)$ por ref
else	
itConj(Registro) itnuevo AgregarRapido(JoinC, UnirRegistros(r1, r2))	$O(1)$ por ref
Definir(JoinN, ValorNat(Siguiente(its)), itnuevo)	$O(\text{Log}(n+m))$
end if	
Avanzar(its)	$O(1)$ por ref
end while	
end if	
info_join \leftarrow \langle Rcambios, campoJ, campoT, JoinS, JoinN, JoinC \rangle	$O(1)$ por ref
Definir(Ta1.Joins, t2, info_join)	$O(1)$ por ref
Definir(Ta2.Joins, t1, info_join)	$O(1)$ por ref
	<hr/>
	$O(1)$ por ref
BORRARJOIN (in t1 : string, in t2 : string, in/out b : estr)	
Tabla tab1 \leftarrow DameTabla(t1, b)	
Tabla tab2 \leftarrow DameTabla(t2, b)	
Borrar(tab1.Joins, t2)	
Borrar(tab2.Joins, t1)	
	<hr/>
	$O(1)$ por ref
BUSCAR (in criterio : registro, in t : string, in b : base) \longrightarrow res : conj(ItConj(registro))	
Busco los datos de la tabla.	
info_tabla infot \leftarrow Significado(b.tablas, t)	$O(1)$ por ref
Tabla tab \leftarrow infot.TActual	$O(1)$ por ref
res \leftarrow BuscarEnTabla(criterio, tab)	
	<hr/>
	$O(1)$ por ref
REGISTROS (in t : string, in b : base) \longrightarrow res : Conj(registros)	
info \leftarrow Significado(b.tablas, t)	$O(1)$ por ref

$tab \leftarrow info.TActual$	$O(1)$ por ref
$res \leftarrow registros(tab)$	$O(1)$ por ref
	<hr/>
	$O(1)$ por ref

```

VISTAJOIN(in t1 : string, in t2 : string, in/out b : estr)
  info_tabla infot1 ← Significado(b.Tablas, t1)
  Tabla tab1 ← infot1.TActual
  info_tabla infot2 ← Significado(b.Tablas, t1)
  Tabla tab2 ← infot2.TActual
  info_join infoj ← Significado(infoj1.Joins, t2)
  Campo c ← infoj.campoJ
  if ¬EsVacia?(infoj.Rcambios) then
    Hubo cambios desde la generacion del join o del ultimo vistaJoin
    while ¬EsVacia?(infoj.Rcambios) do
      DatoCambio data ← Proximo(infoj.Rcambios)
      Desencolar(infoj.Rcambios)
      Registro r ← data.Reg
      if data.Accion then
        La accion es agregar un registro al join
        Para hacer esto necesito saber a que tabla se agrego el registro
        NombreTabla NomTorigen ← data.NomOrigen
        Sabiendo la tabla de origen, necesito identificar la otra tabla para ver si
        hay un registro con el mismo valor para el campo del join.
        Armo un registro auxiliar regModelo con el campo
        Dicc(campo,dato) regModelo ← vacio()
        Definir(regModelo, c, Significado(r, c))
        Si hay indice en la tabla para el campo c y ademas es campo clave
        BuscarEnTabla(tab, regModelo) tiene complejidad  $O(L)$  u  $O(\log(n))$ 
        dependiendo del tipo del campo c.
        Si no hay indice para el campo c entonces BuscarEnTabla(tab, regModelo)
        tiene complejidad  $O(m)$  siendo m la cant de registros en tab.
        Registro rotro ← vacio()
        if NomTorigen=t1 then
          Entonces el otro registro lo tengo que buscar en t2
          Registro rotro ← Siguiente(BuscarEnTabla(tab2, regModelo))
        else
          Entonces el otro registro lo tengo que buscar en t1
          Registro rotro ← Siguiente(BuscarEnTabla(tab1, regModelo))
        end if
        if ¬Vacio?(roto) then
          Si habia un registro que cumpliera con lo pedido
          Registro rnuevo ← UnirRegistros(r, roto)
          if info_join.campoT then
            El tipo del campo es Natural.
            itConj(registro) itnew ← AgregarRapido(info_join.JoinC, rnuevo)
            Definir(info_join.JoinN, key, itnew)
          else
            El tipo del campo es String.
            itConj(registro) itnew ← AgregarRapido(info_join.JoinC, rnuevo)
            Definir(info_join.JoinS, key, itnew)
          end if
        end if
      elseCasoBorrado

```


<pre> if info_join.campoT then El tipo del campo es Natural. itConj(registro) itcjr ← Significado(info_join.JoinN, key) Borrar(info_join.JoinN, key) EliminarSiguiente(itcjr) else El tipo del campo es String. itConj(registro) itcjr ← Significado(info_join.JoinN, key) Borrar(info_join.JoinS, key) EliminarSiguiente(itcjr) end if end if end while end if res ← info_join.JoinC </pre>	
	<hr/>
	O(Tamaño(infoj.Rcambios)*)
<pre> CANTIDADDEACCESOS(in t : string, in b : base) → res : nat info ← Significado(b.tablas, t) tab ← info.TActual res ← cantidadDeAccesos(tab) </pre>	O(1) por ref O(1) por ref O(1) por ref
	<hr/>
	O(1) por ref
<pre> TABLAMAXIMA(in b : base) → res : string res ← b.TablaMaxima.NomTabla </pre>	O(1) por ref
	<hr/>
	O(1) por ref