DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2: Diseño

Primer cuatrimestre - 2016 Algoritmos y Estructuras de Datos II

Grupo XXXX

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

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria – Pabellón I (Planta Baja)
Intendente Güiraldes 2160 – C1428EGA
Ciudad Autónoma de Buenos Aires – Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

Índice

1 Tabla

1.1 Interfaz

se explica con TABLA

usa

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

Operaciones

NOMBRE(**in** t : `tab`) $\longrightarrow$ res : `string`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} nombre(t)\}$
Descripción: Devuelve el nombre de la tabla ingresada por parametro.
Complejidad: O(1)
Aliasing: Se retorna res por copia, por ser un tipo basico.

CLAVES(**in** t : `tab`) $\longrightarrow$ res : `itConjTrie(campo)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} claves(t)\}$
Descripción: Devuelve un conjunto de campos que son claves en la tabla ingresada por parametro.
Complejidad: O(1)
Aliasing: Se devuelve un iterador al conjunto claves por referencia.

INDICES(**in** t : `tab`) $\longrightarrow$ res : `itConjTrie(campo)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} indices(t)\}$
Descripción: Devuelve un conjunto de los indices de la tabla ingresada por parametro.
Complejidad: O($calcular$)
Aliasing: Se devuelve res por referencia y no es modificable.

CAMPOS(**in** t : `tab`) $\longrightarrow$ res : `itConjTrie(campo)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} campos(t)\}$
Descripción: Devuelve un conjunto a los campos de la tabla ingresada por parametro.
Complejidad: O(1)
Aliasing: Se devuelve res por referencia.

TIPOCAMPO(**in** c : `campo`, **in** t : `tab`) $\longrightarrow$ res : `tipo`
Pre $\equiv$ {c $\in$ campos(t)}
Post $\equiv$ $\{res =_{\text{obs}} tipoCampo(t)\}$
Descripción: Devuelve el tipo del campo c en la tabla ingresada por parametro.
Complejidad: O(1)
Aliasing: Se devuelve res por referencia, no es modificable.

REGISTROS(**in** t : `tab`) $\longrightarrow$ res : `itConj(registro)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} registros(t)\}$
Descripción: Devuelve un conjunto a los registros de la tabla ingresada por parametro.
Complejidad: O($L + log(n)$)
Aliasing: Se devuelve res referencia

CANTIDADDEACCCESOS(**in** t : `tab`) $\longrightarrow$ *res* : `nat`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} cantidadDeAccesos(t)$}
Descripción: Devuelve la cantidad de modificaciones de la tabla ingresada por parametro.
Complejidad: O(1)
Aliasing: Se devuelve res por copia.

NUEVATABLA(**in** *nombre* : `string`, *in claves* : `conjTrie(campo)`, *in columnas* : `registro`) $\longrightarrow$ *res* : `tab`
Pre $\equiv$ {$\neg\emptyset$?(claves) $\wedge$ claves$\subseteq$campos(columnas)}
Post $\equiv$ {$res =_{\text{obs}} nuevaTabla(t)$}
Descripción: Crea una tabla sin registros.
Complejidad: O($calcular$)

AGREGARREGISTRO(**in** r : `registro`, *in* t : `tab`)
Pre $\equiv$ {t_0=t $\wedge$ campos(r)=$_{\text{obs}}$campos(t) $\wedge$ puedorInsertar?(r,t)}
Post $\equiv$ {agregarRegistro(r,t_0)}
Descripción: Agrega un registro a la tabla pasada por parametro.
Complejidad: O($L + in$)
Aliasing: Agrega el registro r por referencia.

BORRARREGISTRO(**in** *crit* : `registro`, *in* t : `tab`)
Pre $\equiv$ {t_0=t $\wedge$ #(campos(r))=1 $\wedge_{\text{L}}$ dameUno(campos(crit))$\in$claves(t)}
Post $\equiv$ {borrarRegistro(r,t_0)}
Descripción: Borra los registros que cumplan el criterio pasado por parametro.
Complejidad: O($L + in$)

INDEXAR(**in** *crit* : `registro`, *in* t : `tab`)
Pre $\equiv$ {t_0=t $\wedge$ puedeIndexar(c,t)}
Post $\equiv$ {indexar(c,t_0)}
Descripción: Borra los registros que cumplan el criterio pasado por parametro.
Complejidad: O($L + in$)

PUEDOINSERTAR?(**in** r : `registro`, *in* t : `tab`) $\longrightarrow$ *res* : `bool`
Pre $\equiv$ {$true$}
Post $\equiv$ {$res =_{\text{obs}} puedoInsertar?(r, t)$}
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($T * L + in$)

COMPATIBLE(**in** r : `registro`, *in* t : `tab`) $\longrightarrow$ *res* : `bool`
Pre $\equiv$ {$true$}
Post $\equiv$ {$res =_{\text{obs}} compatible(r, t)$}
Descripción: Informa si el registro pasado por parametro tiene correspondecia en los tipos de los campos de tabla pasada por parametro.
Complejidad: O(1)

MINIMO(**in** c : `campo`, *in* t : `tab`) $\longrightarrow$ *res* : `dato`
Pre $\equiv$ {$\neg\emptyset$?(registro(t)) $\wedge$ c$\in$indices(t)}
Post $\equiv$ {$res =_{\text{obs}} minimo(c, t)$}
Descripción: Retorna el minimo entre los valores de la tabla para el campo c.
Complejidad: O($L + in$)
Aliasing: Retorna res por referencia.

MAXIMO(**in** c : `campo`, *in* t : `tab`) $\longrightarrow$ *res* : `dato`
Pre $\equiv$ {$\neg\emptyset$?(registro(t)) $\wedge$ c$\in$indices(t)}

Post $\equiv \{res =_{\text{obs}} maximo(c, t)\}$
Descripción: Retorna el maximo entre los valores de la tabla para el campo c.
Complejidad: $O(L + in)$
Aliasing: Retorna res por referencia.

PUEDEINDEXAR(**in** c : `campo`, *in* t : `tab`) $\longrightarrow$ *res* : `bool`
Pre $\equiv \{True\}$
Post $\equiv \{res =_{\text{obs}} puedeIndexar(c, t)\}$
Descripción: Informa si se puede crear un nuevo indice.
Complejidad: $O(L + in)$

COINCIDENCIAS(**in** r : `registro`, *in* cj : `Conj(registro)`) $\longrightarrow$ *res* : `Conj(registro)`
Pre $\equiv \{True\}$
Post $\equiv \{res =_{\text{obs}} coincidencias(r, cj)\}$
Descripción: Compara el valor del registro con el conjunto de registros y retorna la interseccion.
Complejidad: $O(L + in)$
Aliasing: Retorna res por referencia.

HAYCOINCIDENCIA(**in** r : `registro`, **in** *cj1* : `ConjTrie(campo)`, *in cj2* : `Conj(registro)`) $\longrightarrow$ *res* : `bool`
Pre $\equiv \{True\}$
Post $\equiv \{res =_{\text{obs}} hayCoincidencia(r, cj1, cj2)\}$
Descripción: Compara los valores del registro para los campos dados por parametro, con el conjunto de registros.
Complejidad: $O(L + in)$

COMBINARREGISTROS(**in** c : `campo`, **in** *cj1* : `Conj(registro)`, *in cj2* : `Conj(registro)`) $\longrightarrow$ *res* : `conj(registro)`
Pre $\equiv \{True\}$
Post $\equiv \{res =_{\text{obs}} combinarRegistros(c, cj1, cj2)\}$
Descripción: Combina los valores de los registros para el campo dado por parametro.
Complejidad: $O(L + in)$
Aliasing: Retorna res por copia.

DAMECOLUMNA(**in** c : `campo`, **in** cj : `Conj(registro)`) $\longrightarrow$ *res* : `conj(dato)`
Pre $\equiv \{True\}$
Post $\equiv \{res =_{\text{obs}} dameColumna(c, cj1, cj2)\}$
Descripción: Reune en un conjunto los valores del campo pasado por parametro.
Complejidad: $O(T * L + in)$
Aliasing: Retorna res por referencia.

MISMOSTIPOS(**in** r : `registro`, **in** t : `tab`) $\longrightarrow$ *res* : `bool`
Pre $\equiv \{\text{campos(r)}\subseteq\text{campos(t)}\}$
Post $\equiv \{res =_{\text{obs}} mismosTipos(r, t)\}$
Descripción: Compara los tipos correspondientes a los campos del registro y la tabla.
Complejidad: $O(1)$

1.2 Representación

se representa con `Tabla`
donde `tab` **es** `tupla`⟨Nombre : `String`,
Indices : `DiccTrie(campo, Indice)`,
Registros : `Conj(Registro)`,
Campos : `DiccTrie(Campo,Tipo)`,
#Accesos : `Nat`⟩
`donde Indice es Dicc(Dato,Conj(Registro))`

Invariante de representación

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

Función de abstracción

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

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

1.3 Algoritmos

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

$res \leftarrow t.nombre$	O(1)
	O(1)

CLAVES(**in** t : `tab`) $\longrightarrow$ res : `ConjTrie(campo)`

$res \leftarrow t.Campos.ClavesDicc$	O(1)
	O(1)

INDICES(**in** t : `tab`) $\longrightarrow$ res : `ConjTrie(campo)`

$res \leftarrow t.ClavesDicc$	O(1)
	O(1)

CAMPOS(**in** t : `tab`) $\longrightarrow$ res : `ConjTrie(campo)`

$res \leftarrow t.Campos.ClavesDicc$	O(1)
	O(1)

TIPOCAMPO(**in** c : `campo`, *in* t : `tab`) $\longrightarrow$ res : `Tipo`

$res \leftarrow Significado(t.Campos, c)$	O(1)
	O(1)

REGISTROS(**in** t : `tab`) $\longrightarrow$ res : `Conj(registro)`

$res \leftarrow t.registros$	θ(L+ log(n))
	θ(L+ log(n))

CANTDEACCESOS(**in** t : `tab`) $\longrightarrow$ res : `nat`

$res \leftarrow t.cantDeAccesos$	$\theta(1)$
	$\theta(1)$

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

itcampos $\leftarrow$ crearItTrie(Campos(columnas))	O(1)
res $\leftarrow$ $< nombre\, Vacio()\, Vacio()\, Vacio()\, 0 >$	O(1)
while HaySiguiente(itcampos) **do**	O(1)
Esto se debe a que # de campos a iterar es acotada.	
valor $\leftarrow$ Significado(r, Siguiente(itcampos))	O(1)
DefinirRapido(res.Campos, Siguiente(itcampos), valor)	O(1)
if Pertenece?(claves,Siguiente(itcampos)) **then**	O(1)
val $\leftarrow$ Significado(r, Siguiente(itcampos))	
AgregarRapido(res.Campos.CamposClave, val)	O(1)
end if	
Avanzar(itcampos)	O(1)
end while	
	$\theta(1)$

AGREGARREGISTRO(**in** r : `registro`, *in* t : `tab`)

nuevo $\leftarrow$ AgregarRapido(t.Registros,r)	$\theta(1)$
t.#Accesos++	$\theta(1)$
if Cardinal(t.Indices.ClavesDicc)$\geq$1 **then**	$\theta(1)$

itInd ← crearItConjTrie(t.Indices.ClavesDicc)	$\theta(1)$
while HaySiguiente(itInd) **do**	$\theta(1)$
indiceC ← Obtener(t.Indices, Siguiente(itInd))	$\theta(1)$
valorC ← Obtener(r, Siguiente(itInd))	$\theta(1)$
AgregarRapido(Obtener(indiceC, valorC), nuevo)	$\theta(1)$
Avanzar(itInd)	$\theta(1)$
end while	
end if	
	$\theta(1)$

BORRARREGISTRO(**in** *crit* : `registro`, *in* *t* : `tab`)

c ← Siguiente(Campos(crit))	$\theta(1)$
valor ← Obtener(crit, c)	$\theta(1)$
if Definido?(t.Indices, c) **then**	$\theta(1)$
indiceC ← Obtener(t.Indices, c)	$\theta(1)$
itcjr ← CrearItConj(Obtener(indiceC, valor))	$\theta(1)$
while HaySiguiente(itcjr) **do**	$\theta(\text{Log(n)})$
EliminarSiguiente(Siguiente(itcjr))	$\theta(1)$
tiene sentido???	
EliminarSiguiente(itcjr)	$\theta(1)$
end while	
else	
cr ← Coincidencias(crit, t.registros)	$\theta(\text{Cardinal(t.registros)})$
while HaySiguiente(cr) **do**	
EliminarSiguiente(Siguiente(cr))	
tiene sentido???	
EliminarSiguiente(cr)	
end while	
end if	
	$\theta(\text{Calcular despues de consulta})$

INDEXAR(**in** *c* : `campo`, *in* *t* : `tab`)

if tipoCampo(c,t) **then**	
conjLog(registro) nuevo ← vacio()	
else	
conjTrie(registro) nuevo ← vacio()	
end if	
indC ← Siguiente(DefinirRapido(t.Indices, c, nuevo))	
cr ← t.registros	
while HaySiguiente(cr) **do**	
valor ← Obtener(Siguiente(cr), c)	
if Definido?(indC, valor) **then**	
regviejos ← Obtener(indC, valor)	
AgregarRapido(regviejos, Siguiente(cr))	
else	
DefinirRapido(indC, valor, Siguiente(cr))	
end if	
Avanzar(cr)	
end while	
	$\theta(1)$

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

res ← campatible(r,t) ∧ ¬hayCoincidencia(r, r.ClavesDicc, registros(t)) — θ(L+log(n))

θ(L+log(n))

COMPATIBLE(**in** *r* : `registro`, *int t* : `tab`) ⟶ *res* : `bool`

res ← campatible(r,t) $\wedge_{L}$ mismosTipos(r,t) — θ(1)

O(1)

PUEDEINDEXAR(**in** *c* : `campo`, *in t* : `tab`) ⟶ *res* : `bool`

res ← Definido?(t.campos,c) $\wedge_{L}\neg$ Definido?(t.Indices,c) ∧ (Cardinal(t.Indices)≤1

O(calcular)

COMBINARREGISTROS(**in** *c* : `campo`, *in cr1* : `Conj(registro)`, *in cr2* : `Conj(registro)`) ⟶ *res* : `Conj(registros)`

res ← vacio();	θ(1)
itcr1 ← CrearItConjTrie(cr1)	θ(1)
while HaySiguiente(itcr1) **do**	θ(Cardinal(cr1))
AgregarRapido(res, combinarTodos(c,Siguiente(itcr1),cr2))	θ(1)
Avanzar(itcr1);	θ(1)
end while	

O(Cardinal(cr1))

HAYCOINCIDENCIA(**in** *r* : `registro`, *in cc* : `ConjTrie(campo)`, *in cr* : `Conj(registro)`) ⟶ *res* : `bool`

itcr ← CrearItConj(cr);	θ(1)
res ← false;	θ(1)
while HaySiguiente(itcr) **do**	θ(Cardinal(cr))
res ← coincideAlguno(r,cc,Siguiente(itcr)) ∨res;	θ(1)
Avanzar(itcr);	θ(1)
end while	

O(Cardinal(cr))

COINCIDENCIAS(**in** *crit* : `registro`, *in cr* : `Conj(registro)`) ⟶ *res* : `Conj(registro)`

res ← Vacio();	θ(1)
itcr ← CrearItConj(cr)	
while HaySiguiente(cr) **do**	θ(Cardinal(cr))
if coincidenTodos(crit,campos(crit),Siguiente(itcr)) **then**	θ(1)
AgregarRapido(res,Siguiente(itcr))	θ(1)
end if	
Avanzar(itcr);	θ(1)
end while	

O(Cardinal(cr))

MINIMO(**in** *c* : `campo`, *in t* : `tab`) ⟶ *res* : `dato`

res ← min(dameColumna(c, t.registros)) — θ(Cardinal(t.registros))

O(Cardinal(t.registros))

MAXIMO(**in** *c* : `campo`, *in t* : `tab`) ⟶ *res* : `dato`

res ← max(dameColumna(c, t.registros)) — θ(Cardinal(t.registros))

	O(Cardinal(t.registros))

DAMECOLUMNA(**in** c : `campo`, *in* cr : `Conj(registro)`) $\longrightarrow$ *res* : `Conj(dato)`

itcr ← CrearItConj(cr);	$\theta(1)$
res ← vacio();	$\theta(1)$
while HaySiguiente(itcr) **do**	θ(Cardinal(cr))
if ¬Pertenece(res, Siguiente(itcr)) **then**	θ(?????)
AgregarRapido(res, Siguiente(itcr))	
end if	
Avanzar(itcr);	$\theta(1)$
end while	
	O(calcular)

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

res ← True;	$\theta(1)$
itconjClaves ← CrearItConj(r.ClavesDicc);	$\theta(1)$
while HaySiguiente(itconjClaves) **do**	$\theta(1)$
val1← tipo?(Obtener(r,Siguiente(itconjClaves)))	θ(Cardinal(t.registros))
val2← tipoCampo(Siguiente(itconjClaves)),t)	$\theta(1)$
res ← res ∧ val1 = val2	$\theta(1)$
Avanzar(cr);	$\theta(1)$
end while	
	O(calcular)

1.4 Algoritmos operaciones auxiliares

2 Base de Datos

2.1 Interfaz

se explica con BASE

usa

géneros `nat`, `string`, `tabla`, `regisro`, `campo`, `dato`

Operaciones

TABLAS(**in** b : `base`) $\longrightarrow$ *res* : `conj(string)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} nombre(t)\}$
Descripción: Devuelve el nombre de la tabla ingresada por parametro.
Complejidad: O(1)
Aliasing: Se retorna res por copia, por ser un tipo basico.

DAMETABLA(**in** b : `base`) $\longrightarrow$ *res* : `tabla`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} claves(t)\}$
Descripción: Devuelve un conjunto de campos que son claves en la tabla ingresada por parametro.
Complejidad: O(1)
Aliasing: Se devuelve un iterador al conjunto claves por referencia.

HAYJOIN?(**in** *t1* : `string`, *in t2* : `string`, *in t* : `base`) $\longrightarrow$ *res* : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} indices(t)$}
Descripción: Devuelve un conjunto de los indices de la tabla ingresada por parametro.
Complejidad: O(*calcular*)
Aliasing: Se devuelve res por referencia y no es modificable.

CAMPOJOIN(**in** *t1* : `string`, *in t2* : `string`, *in t* : `base`) $\longrightarrow$ *res* : `itConjTrie(campo)`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} campos(t)$}
Descripción: Devuelve un conjunto a los campos de la tabla ingresada por parametro.
Complejidad: O(1)
Aliasing: Se devuelve res por referencia.

NUEVADB() $\longrightarrow$ *res* : `base`
Pre $\equiv$ {$True$}
Post $\equiv$ {$res =_{\text{obs}} nuevaDB()$}
Descripción: Crea una base sin tablas.
Complejidad: O(*calcular*)

AGREGARTABLA(**in** *t* : `tabla`, *in b* : `base`)
Pre $\equiv$ {b_0=b $\wedge$ nombre(t)$\notin$ tablas(b) $\wedge$ Vacio?(t.registros)}
Post $\equiv$ {agregarTabla(t b_0)}
Descripción: Agrega una tabla a la base de datos.
Complejidad: O(*calcular*)
Aliasing: Agrega tabla por referencia.

INSERTARENTRADA(**in** *reg* : `registro`, *in t* : `string`, *in b* : `base`)
Pre $\equiv$ {b_0=b $\wedge$ t$\in$tablas(b) $\wedge_{\text{L}}$ puedoInsertar?(dameTabla(t) reg)}
Post $\equiv$ {insertarEntrada(r t b_0)}
Descripción: Inserta el registro a la tabla que corresponde al string pasado por parametro.
Complejidad: O(*calcular*)

BORRAR(**in** *cr* : `registro`, *in t* : `string`, *in b* : `base`)
Pre $\equiv$ {b_0=b $\wedge$ t$\in$tablas(b) $\wedge$ #(cr.DiccClaves)}
Post $\equiv$ {borrar(cr t b_0)}
Descripción: Borra los registros que cumplan el criterio cr pasado por parametro.
Complejidad: O(*calcular*)

GENERARVISTAJOIN(**in** *t1* : `string`, *in t2* : `string`, *in c* : `campo`, *in b* : `base`)
Pre $\equiv$ {b_0=b $\wedge$ t1$\sqsupset$t2 $\wedge$ {t1 t2}$\subseteq$tablas(b) $\wedge_{\text{L}}$ (c$\in$dameTabla(t1 b).diccClaves $\wedge$ c$\in$dameTabla(t2 b).diccCla
Post $\equiv$ {generarVistaJoin(cr, t, b_0)}
Descripción: Borra los registros que cumplan el criterio cr pasado por parametro.
Complejidad: O(*calcular*)

BORRARJOIN(**in** *t1* : `string`, *in t2* : `string`, *in b* : `base`)
Pre $\equiv$ {b_0=b $\wedge$ hayJoin?(t1 t2 b)}
Post $\equiv$ {borrarJoin(t1 t2 b_0)}
Descripción: Borra correspondiente a los nombres de tablas, pasados por parametro.
Complejidad: O(*calcular*)

REGISTROS(**in** *t* : `string`, *in b* : `base`) $\longrightarrow$ *res* : `conj(registro)`
Pre $\equiv$ {t$\in$tablas(b)}
Post $\equiv$ {$res =_{\text{obs}} registros(t\,b)$}
Descripción: Retorna el conjunto de registros correspondientes al nombre de tabla pasado por parametro

Complejidad: O(*calcular*)
Aliasing: Se retorna el conjunto de registros por referencia.

VISTAJOIN(**in** *t1* : `string`, *in t2* : `string`, *in b* : `base`) $\longrightarrow$ *res* : `conj(registro)`
Pre $\equiv$ {{t1 t2}$\subseteq$tablas(b) $\wedge$ hayJoin?(t1 t2 b)}
Post $\equiv$ {$res =_{\text{obs}} vistaJoin(t1\,t2\,b)$}
Descripción: Retorna el conjunto de registros correspondientes al nombre de tabla pasado por parametro
Complejidad: O(*calcular*)
Aliasing: Se retorna el conjunto de registros por referencia.

CANTIDADDEACCESOS(**in** *t* : `string`, *in b* : `base`) $\longrightarrow$ *res* : `nat`
Pre $\equiv$ {t$\in$tablas(b)}
Post $\equiv$ {$res =_{\text{obs}} cantidadDeAccesos(t\,b)$}
Descripción: Retorna la cantidad de modificaciones correspondientes al nombre de tabla pasado por parametro.
Complejidad: O(*calcular*)
Aliasing: Se retorna res por referencia.

TABLAMAXIMA(**in** *b* : `base`) $\longrightarrow$ *res* : `string`
Pre $\equiv$ {$\neg\emptyset$?(tablas(b))}
Post $\equiv$ {$res =_{\text{obs}} tablaMaxima(t\,b)$}
Descripción: Retorna el nombre de la tabla con la mayor cantidad de modificaciones.
Complejidad: O(*calcular*)
Aliasing: Se retorna el nombre de la tabla por referencia.

ENCONTRARMAXIMO(**in** *t* : `string`, *in ct* : `conj(string)`, *in b* : `base`) $\longrightarrow$ *res* : `string`
Pre $\equiv$ {{t}$\cup$ct $\subseteq$tablas(b)}
Post $\equiv$ {$res =_{\text{obs}} tablaMaxima(t\,b)$}
Descripción: Retorna ...
Complejidad: O(*calcular*)
Aliasing: Se retorna el nombre de la tabla por referencia.

BUSCAR(**in** *criterio* : `registro`, *in t* : `string`, *in b* : `base`) $\longrightarrow$ *res* : `conj(registro)`
Pre $\equiv$ {t$\in$tablas(b)}
Post $\equiv$ {$res =_{\text{obs}} tablaMaxima(t\,b)$}
Descripción: Retorna ...
Complejidad: O(*calcular*)
Aliasing: Se retorna el nombre de la tabla por referencia.

2.2 Representación

se representa con `Base`
donde `estr` **es** `tupla`⟨Tablas : `DiccTrie(Campo,info_tabla)`,
Joins : `DiccTrie(Campo,info_join)`⟩
`donde info_tabla es tupla`⟨#Accesos : `nat`,
TActual : `tabla`⟩
`donde info_join es tupla`⟨R : `nat`,
Rborrados : `Conj(registro)`,
Ragregados : `Conj(registro)`,
campoJ : `campo`,
campoT : `tipo`⟩

Invariante de representación

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

Función de abstracción

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

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

2.3 Algoritmos