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

1.1 Interfaz

se explica con TABLA

usa

géneros

DiccString(string, alfha), DiccNat(nat, beta),Nat, String, Dato, Campo, Tipo, Registro, ConjString, ConjNat.

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 : `ConjString(campo)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} claves(t)\}$
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 $\equiv$ $\{res =_{\text{obs}} indices(t)\}$
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 $\equiv$ $\{res =_{\text{obs}} campos(t)\}$
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 $\in$ campos(t)}
Post $\equiv$ $\{res =_{\text{obs}} tipoCampo(t)\}$
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 $\equiv$ $\{res =_{\text{obs}} registros(t)\}$
Descripción: Devuelve un conjunto a los registros de la tabla.

Complejidad: O(1)
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* : `conjString(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(1)

AGREGARREGISTRO(**in** r : `registro`, *in/out* 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($Log(n)$)
Aliasing: Agrega el registro r por referencia.

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

INDEXAR(**in** *crit* : `registro`, *in/out* t : `tab`)
Pre $\equiv$ {t_0=t $\wedge$ puedeIndexar(c,t)}
Post $\equiv$ {indexar(c,t_0)}
Descripción: Crea un indice en base al campo de crit.
Complejidad: O(n)

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(n)
Aliasing: Retorna res por referencia, no es modificable.

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)
Aliasing: Retorna res por referencia, no es modificable.

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($L + Log(n)$)
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($L + Log(n)$)
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(1)
Aliasing: Retorna res por referencia, no es modificable.

COINCIDENCIAS(**in** r : `registro`, *in* cj : `Conj(registro)`) $\longrightarrow$ *res* : `Conj(ItConj(registro))`
Pre $\equiv$ {$True$}
Post $\equiv$ {$res =_{\text{obs}} coincidencias(r, cj)$}
Descripción: Devuelve el conjunto de registros de la tabla, que coinciden con los valores de r.
Complejidad: O($Cardinal(cj)$)
Aliasing: Retorna res por referencia.

HAYCOINCIDENCIA(**in** r : `registro`, **in** *cjc* : `Conj(campo)`, *in cjr* : `Conj(registro)`) $\longrightarrow$ *res* : `bool`
Pre $\equiv$ {$True$}
Post $\equiv$ {$res =_{\text{obs}} hayCoincidencia(r, cjc, cjr)$}
Descripción: Retorna true si algun registro del conjunto cjr, coincide con r en todos los valores de los campos de cjc.
Complejidad: O($Cardinal(cjr)$)
Aliasing: Retorna res por referencia, no es modificable.

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($Cardinal(cj1) + Cardinal(cj2)$)
Aliasing: Retorna res por referencia, es modificable.

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($Cardinal(cj) * Log(Cardinal(cj))$)
Aliasing: Retorna res por referencia, no es modificable.

MISMOSTIPOS(**in** r : `registro`, **in** t : `tab`) $\longrightarrow$ *res* : `bool`
Pre $\equiv$ {campos(r)$\subseteq$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)
Aliasing: Retorna res por referencia, no es modificable.

1.2 Representación

se representa con `Tabla`
donde `tab` **es** `tupla`⟨Nombre : `String`,
Registros : `Conj(Registro)`,
Campos : `DiccString(Campo,Tipo)`,
Claves : `ConjString(Campo)`,
IndiceS : `tupla`⟨CampoI : `campo`, ,
EnUso : `bool`,
Indice : `DiccString(string, Conj(ItConj(Registro)))`,
Min : `Dato`,
Max : `Dato`⟩
IndiceN : `tupla`⟨CampoI : `campo`, ,
EnUso : `bool`,
Indice : `DiccNat(nat, Conj(ItConj(Registro)))`,
Min : `Dato`,
Max : `Dato`⟩
#Accesos : `Nat`⟩

Invariante de representación

1. t.Claves esta inclido o es igual a t.Campos.
2. t.Nombre es un string acotado.
3. Para todo registro r de t.Registros, entonces Campos(r) es igual al t.Campos.
4. Para todo registro r de t.Registros y para todo campo c de Campos(r), entonces Tipo?(Significado(r,c)) es igual Significado(t.Campos, c).
5. Si t.IndiceS.EnUso es true y t.IndiceS.CampoI pertenece a t.Campos, entonces para todo Dato d, si Definido?(t.IndiceS.Indice, d) es true, entonces Significado(t.IndiceS.Indice, d) esta incluido o es igual a t.Registros.
6. Si t.IndiceS.EnUso es true y t.IndiceS.CampoI pertenece a t.Campos, entonces para todo registro r de t.Registros entonces Definido?(t.IndiceS.Indice, Significado(r, t.IndiceS.CampoI)) es true y r pertenece a Significado(t.IndiceS.Indice, Significado(r, t.IndiceS.CampoI)).
7. Lo anterior tambien aplica para t.IndiceN.Indice
8. El valor de e.#Accesos debe ser la cantidad de registros agregados, la cantidad de registros borrados

Función de abstracción

$\text{Abs} : \widehat{\texttt{tab}}\ s \longrightarrow \widehat{\texttt{Tabla}}$ $\{\text{Rep}(s)\}$

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

1.3 Algoritmos

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

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

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

res ← t.Claves	O(1)
	O(1)

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

ConjString(campo)res ← vacio();	
if t.IndiceS.EnUso **then**	
AgregarRapido(res, t.IndiceS.CampoI)	O(1)
end if	
if t.IndiceN.EnUso **then**	
AgregarRapido(res, t.IndiceN.CampoI)	O(1)
end if	
	O(1)

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

$res \leftarrow Claves(t.Campos)$	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)`

$res \leftarrow t.registros$	$\theta(1)$
Se retorna el conjunto por referencia	
	$\theta(1)$

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

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

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

Conj(registro) Registros $\leftarrow$ Vacio()	O(1)
DiccString(campo, tipo) Campos $\leftarrow$ Vacio()	O(1)
ConjString(campo) Claves_ $\leftarrow$ Vacio()	O(1)
Dato d $\leftarrow$ Obtener(columnas, Siguiente(CrearIt(claves)))	
IndiceS $\leftarrow <$ $Siguiente(CrearIt(claves)), False, Vacio(), d, d >$	O(1)
IndiceN $\leftarrow <$ $Siguiente(CrearIt(claves)), False, Vacio(), d, d >$	O(1)
#Acessos $\leftarrow$ 0	O(1)
res $\leftarrow <$ $nombre, Registros, Campos, Claves\_, IndiceS, IndiceN, 0 >$	O(1)
itcampos $\leftarrow$ crearItConj(Campos(columnas))	O(1)
while HaySiguiente(itcampos) **do**	O(1)
Este while es O(1) y se debe a que el cardinal de campos a iterar es acotado.	
valor $\leftarrow$ Significado(columnas, Siguiente(itcampos))	O(1)
DefinirRapido(res.Campos, Siguiente(itcampos), Tipo?(valor))	O(1)
Avanzar(itcampos)	O(1)
end while	
itclaves $\leftarrow$ crearItConj(claves)	O(1)
Paso los campos claves de un conj al conjString	
while HaySiguiente(itclaves) **do**	O(1)
Esto se debe a que # de campos a iterar es acotada.	
AgregarRapido(res.Claves, Siguiente(itclaves))	O(L)
Avanzar(itcampos)	O(1)
end while	
Donde L es la longitud de la cadena string mas larga y acotada del parametro claves.	
	$\theta(1)$

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

Para poder acceder al registro, guardo el iterador al elemento	
itConj(Registro) nuevo ← AgregarRapido(t.Registros,r)	$\theta(1)$
t.#Accesos++	$\theta(1)$
Este registro debe ser indexado, si estos indices existieran.	
if t.IndiceS.EnUso **then**	
Para el indice traigo el valor del campo con el que se creo.	
valor ← Significado(r, t.IndiceS.CampoI)	$\theta(L)$
if Definido?(t.IndiceS.Indice, valor) **then**	$\theta(L)$
viejo ← Significado(t.IndiceS.Indice, valor)	$\theta(L)$
AgregarRapido(viejo, nuevo)	$\theta(L)$
else	
viejo ← Vacio()	$\theta(1)$
AgregarRapido(viejo, nuevo)	$\theta(L)$
Definir(t.IndiceS.Indice, ValorString(valor), viejo)	$\theta(L)$
Como ingresamos un nuevo valor, actualizamos el min y max	
if t.IndiceS.Min≥valor **then**	$\theta(L)$
t.IndiceS.Min ← Copiar(valor)	$\theta(L)$
end if	
if valor≥t.IndiceS.Max **then**	$\theta(L)$
t.IndiceS.Max ← Copiar(valor)	$\theta(L)$
end if	
end if	
Donde L es la longitud del valor string agregado a t.IndiceS	
end if	
if t.IndiceN.EnUso **then**	
Para el indice traigo el valor del campo con el que se creo.	
valor ← Significado(r, t.IndiceN.CampoI)	$\theta(Log(n)$
if Definido?(t.IndiceN.Indice, valor) **then**	$\theta(log(n))$
viejo ← Significado(t.IndiceN.Indice, valor)	$\theta(1)$
AgregarRapido(viejo, nuevo)	$\theta(1)$
else	
viejo ← Vacio()	$\theta(1)$
AgregarRapido(viejo, nuevo)	$\theta(1)$
Definir(t.IndiceN.Indice, valor, viejo)	$\theta(Log(n))$
Como ingresamos un nuevo valor, actualizamos el min y max	
if t.IndiceN.Min≥valor **then**	$\theta(1)$
t.IndiceS.Min ← Copiar(valor)	$\theta(1)$
end if	
if valor≥t.IndiceS.Max **then**	$\theta(1)$
t.IndiceN.Max ← Copiar(valor)	$\theta(1)$
end if	
end if	
end if	
Donde n es la cantidad de registros y L es la longitudad maxima de un valor String.	
	$\theta(Log(n))$

```
borrarRegistro(in crit : registro, in/out t : tab)
  Campo c ← Siguiente(CrearIt(Campos(crit)))                    θ(1)
  Dato valor ← Copiar(Significado(crit, c))                     θ(L)
  Sabemos que c esta incluido en claves(t)
  si hay un indice de un campo clave borrar es O(log(n)) u O(L) en peor caso
  if t.IndiceS.EnUso ∧ t.IndiceS.CampoI=c then
    String S ← ValorString(valor)
    if Definido?(t.IndiceS.Indice, S) then
      itConj(registro) itr ← Significado(t.IndiceS.Indice, S)   θ(1)
      Borro el registro del conjunto de registros
      EliminarSiguiente(itr)                                    θ(1)
      Borro el registro del indice,
      Borrar(t.IndiceS.Indice, S)                               θ(1)
      if t.IndiceS.Min=valor then
        t.IndiceS.Min← Minimo(Claves(t.IndiceS))
      end if
      if t.IndiceS.Max=valor then
        t.IndiceS.Max← Maximo(Claves(t.IndiceS))
      end if
      Dado que el crit solo tiene un campo clave, siempre elimino un registro.
      t.#Accesos++
    end if
  else
    if t.IndiceN.EnUso ∧ t.IndiceN.CampoI=c then
      N ← ValorNat(valor)
      if Definido?(t.IndiceN.Indice, valor) then
        itConj(registro) itr ← Significado(t.IndiceN.Indice, valor)
                                                                θ(1)
        EliminarSiguiente(itr)                                  θ(1)
        Borrar(t.IndiceN.Indice, valor)                         θ(1)
        if t.IndiceN.Min=valor then                             θ(1)
          t.IndiceS.Min← Minimo(Claves(t.IndiceN))              θ(Log(n))
        end if
        if t.IndiceN.Max=valor then                             θ(1)
          t.IndiceS.Max← Maximo(Claves(t.IndiceN))              θ(Log(n))
        end if
        Dado que el crit solo tiene un campo clave, siempre elimino un registro.
        t.#Accesos++
      end if
    else
      No hay indice para el campo clave de criterio pasado por parametro
      itConj(registro) cr ← CrearItConj(t.registros)            θ(1)
      while HaySiguiente(cr) do                                 θ(Cardinal(t.registros))
        valorR ← Significado(Siguiente(cr), c)                  θ(1)
        if valorR=valor then
          EliminarSiguiente(cr);                                θ(1)
          Dado que el crit solo tiene un campo clave, siempre elimino solo un registro.
          t.#Accesos++
        end if
        Avanzar(cr)                                             θ(1)
      end while
      La complejidad de la operacion borrar depende de si hay o no indices
```

para el campo del crit pasado por parametro.
En caso de que exista dicho indice, en peor caso eliminar es O(Log(n)) u O(L)
siendo n la cantidad de registros de la tabla pasada por parametro y L
el valor string mas largo definido en la tabla.
En caso de no haber tal indice es O(n).

end if

end if

O(n)

```
INDEXAR(in c : campo, in/out t : tab)
  if tipoCampo(c,t) then
     t.IndiceN.EnUso ← True
  else
     t.IndiceS.EnUso ← True
  end if
  cr ← CrearItConj(t.registros)
  if tipoCampo?(c,t) then
     while HaySiguiente(cr) do
        Dato valor ← Significado(Siguiente(cr), c)
        itConj(registro) itr ← CrearItConj(Siguiente(cr))
        if Definido?(t.IndiceN.Indice, valor) then
           regviejos ← Significado(indC, valor)
           AgregarRapido(regviejos, itr)
        else
           conj(registro) nuevo ← Vacio()
           AgregarRapido(nuevo, itr)
           DefinirRapido(t.IndiceN.Indice, valor, nuevo)
        end if
        Avanzar(cr)
     end while
  end if
  if ¬tipoCampo?(c,t) then
     while HaySiguiente(cr) do
        Dato valor ← Significado(Siguiente(cr), c)
        itConj(registro) itr ← CrearItConj(Siguiente(cr))
        if Definido?(t.IndiceS.Indice, valor) then
           regviejos ← Significado(indC, valor)
           AgregarRapido(regviejos, itr)
        else
           conj(registro) nuevo ← Vacio()
           AgregarRapido(nuevo, itr)
           DefinirRapido(t.IndiceN.Indice, valor, nuevo)
        end if
        Avanzar(cr)
     end while
  end if
                                                          ---------------------
                                                          θ(1)
```

```
PUEDOINSERTAR?(in r : registro, in t : tab) ⟶ res : bool
  res ← campatible(r,t) ∧ ¬hayCoincidencia( r, r.ClavesDicc, registros(t) )
                                                          θ(calcular))
                                                          ---------------------
                                                          θ(calcular)
```

```
COMPATIBLE(in r : registro, int t : tab) ⟶ res : bool
  bool valor ← True
  if Cardinal(campos(r))=Cardinal(t.Campos.DiccClaves) then
     itcampos ← CrearItString(t.Campos.DiccClaves)
     while valor ∧ HaySiguiente(itcampos) do              θ(1)
        Campo c ← Siguiente(itcampos)                     θ(1)
        valor ← Definido?(r, c)                           θ(1)
     end while
```

else
valor ← False — $\theta(1)$
end if
res ← valor $\wedge_{\mathrm{L}}$ mismosTipos(r,t) — $\theta(1)$
El costo del While es O(1) ya que la cantidad de campos de la tabla es acotado

---

O(1)

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

Si hay indice en el campo c, debe ser de complejidad O(1)
if t.IndiceS.EnUso ∧ t.IndiceS.CampoI=c **then**
Sabemos que hay un indice string para el campo c
res ← t.IndiceS.Min — θ(Cardinal(t.registros))
else
if t.IndiceN.EnUso ∧ t.IndiceN.CampoI=c **then**
Sabemos que hay un indice string para el campo c
res ← t.IndiceN.Min — θ(Cardinal(t.registros))
end if
end if

O(1)

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

Si hay indice en el campo c, debe ser de complejidad O(1)
if t.IndiceS.EnUso ∧ t.IndiceS.CampoI=c **then**
Sabemos que hay un indice string para el campo c
res ← t.IndiceS.Max — θ(Cardinal(t.registros))
else
if t.IndiceN.EnUso ∧ t.IndiceN.CampoI=c **then**
Sabemos que hay un indice string para el campo c
res ← t.IndiceN.Max — θ(Cardinal(t.registros))
end if
end if

O(1)

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

if TipoCampo(c, t) **then**
res ← ¬(t.IndiceN.EnUso)
else
res ← ¬(t.IndiceS.EnUso)
end if

O(1)

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

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

O(Cardinal(cr))

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

Conj(registro) salida ← Vacio()	$\theta(1)$
Debemos comparar todos los registros de cr.	
y agregarlos al conjunto de registros salida	
itcr ← CrearItConj(cr)	
while HaySiguiente(cr) **do**	θ(Cardinal(cr))
if coincidenTodos(crit,campos(crit),Siguiente(itcr)) **then**	$\theta(1)$
AgregarRapido(salida,Siguiente(itcr))	$\theta(1)$
end if	
Avanzar(itcr);	$\theta(1)$
end while	
	O(Cardinal(cr))

combinarRegistros(**in** *c* : `campo`, *in cr1* : `Conj(registro)`, *in cr2* : `Conj(registro)`) $\longrightarrow$ *res* : `Conj(registros)`

itcr1 ← CrearItConjString(cr1)	$\theta(1)$
Se me ocurre hacer un DiccString o DiccNat, dependiendo de tipo sea el campo c, de manera que combinar todos sea complejidad O(Log(Cardinal(cr2))).	
Lo malo es que combinarRegistros sera O(Cardinal(cr2)*Log(Cardinal(cr2)))	
if Cardinal(cr2)≥1 **then**	$\theta(1)$
Registro rtemp ← Siguiente(CrearIt(cr2))	$\theta(1)$
Tipo rac ← Tipo?(Significado(rtemp, c))	$\theta(1)$
if rac **then**	$\theta(1)$
Caso Natural	
DiccNat(Nat, Conj(registro)) d ← vacio()	$\theta(1)$
itcr2 ← CrearIt(cr2)	$\theta(1)$
while HaySiguiente?(itcr2) **do**	$\theta(1)$
Dato valor ← Obtener(Siguiente(itcr2), c)	O(1)
if Definido?(d, ValorNat(valor)) **then**	O(Cardinal(cr2))
cjViejo ← Significado(d, ValorNat(valor))	O(Cardinal(cr2))
AgregarRapido(d, Siguiente(itcr2))	O(1)
else	
ConjNat(registro) cjNuevo ← vacio()	O(1)
AgregarRapido(d, Siguiente(itcr2))	O(1)
Definir(d, ValorNat(valor), cjNuevo)	O(Cardinal(cr2))
end if	
Avanzar(itcr2)	O(1)
end while	
else	
Caso String	
DiccString(String, Conj(registro)) d ← vacio()	$\theta(1)$
itcr2 ← CrearIt(cr2)	$\theta(1)$
while HaySiguiente?(itcr2) **do**	$\theta(1)$
Dato valor ← Obtener(Siguiente(itcr2), c)	O(1)
if Definido?(d, ValorString(valor)) **then**	O(Cardinal(cr2))
cjViejo ← Significado(d, ValorString(valor))	O(Cardinal(cr2))
AgregarRapido(d, Siguiente(itcr2))	O(1)
else	
ConjString(registro) cjNuevo ← vacio()	O(1)
AgregarRapido(d, Siguiente(itcr2))	O(1)
Definir(d, ValorString(valor), cjNuevo)	O(Cardinal(cr2))
end if	
Avanzar(itcr2)	O(1)
end while	
end if	
itcr1 ← CrearIt(cr1)	O(1)
res ← vacio()	
while HaySiguiente(itcr1) **do**	θ(Cardinal(cr1))
AgregarRapido(res, combinarTodos(c,Siguiente(itcr1),cr2))	θ(vvv)
Avanzar(itcr1)	O(1)
end while	
else	
res ← vacio()	
end if	

O(Cardinal(cr1))

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

Conj(Dato) cj ← vacio();	$\theta(1)$
La idea es no agregar el mismo dato dos veces, para eso uso un conj del tipo de dato de la columna para hacer consulta.	
if Cardinal(cr)≥1 **then**	$\theta(1)$
Tvalor ← Tipo?(Significado(Siguiente(CrearIt(cr))), c)	$\theta(1)$
if Tvalor **then**	
ConjLog(nat) cj ← Vacio()	$\theta(1)$
else	
ConjString(string) cj ← Vacio()	$\theta(1)$
end if	
itcr ← CrearItConj(cr)	$\theta(1)$
cjd ← Vacio()	
while HaySiguiente(itcr) **do**	θ(Cardinal(cr))
Dato data ← Significado(Siguiente(itcr), c)	
if Tvalor **then**	
if ¬Pertenece?(cj, valorNat(data)) **then**	θ(Log(n))
AgregarRapido(cjd, data)	$\theta(1)$
AgregarRapido(cj, valorNat(data))	$\theta(1)$
end if	
else	
if ¬Pertenece?(cj, valorString(data)) **then**	$\theta(1)$
AgregarRapido(cjd, data)	$\theta(1)$
AgregarRapido(cj, valorString(data))	$\theta(1)$
end if	
end if	
Avanzar(itcr);	$\theta(1)$
end while	
end if	
res ← cjd	
Si la columna es de tipo String, la complejidad es O(n), en caso de ser de tipo Nat la complejidad es O(nlog(n)).	
Donde n es el cardinal de cr .	

O(nlog(n))

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

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

O(1)

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* : `ItConjString(NombreTabla)`
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 ≡ {*b*_0=b ∧ t∈tablas(b) ∧$_{\text{L}}$ puedoInsertar?(dameTabla(t) reg)}
Post ≡ {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 ≡ {*b*_0=b ∧ t∈tablas(b) ∧ #(cr.DiccClaves)}
Post ≡ {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 ≡ {b_0=b ∧ t1⊐t2 ∧ {t1 t2}⊆tablas(b) ∧$_{\text{L}}$ (c∈dameTabla(t1 b).diccClaves ∧ c∈dameTabla(t2 b).diccCla
Post ≡ {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 ≡ {*b*_0=b ∧ hayJoin?(t1 t2 b)}
Post ≡ {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`) ⟶ *res* : `conj(registro)`
Pre ≡ {t∈tablas(b)}
Post ≡ {$res =_{\text{obs}} registros(t\,b)$}
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`) ⟶ *res* : `conj(registro)`
Pre ≡ {{t1 t2}⊆tablas(b) ∧ hayJoin?(t1 t2 b)}
Post ≡ {$res =_{\text{obs}} vistaJoin(t1\,t2\,b)$}
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`) ⟶ *res* : `nat`
Pre ≡ {t∈tablas(b)}
Post ≡ {$res =_{\text{obs}} cantidadDeAccesos(t\,b)$}
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`) ⟶ *res* : `string`
Pre ≡ {¬∅?(tablas(b))}
Post ≡ {$res =_{\text{obs}} tablaMaxima(t\,b)$}
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`) ⟶ *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`⟨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 : `ConjTrie(registro)`,
ValoresS : `ConjTrie(String)`,
JoinN : `ConjNat(registro)`,
ValoresN : `ConjNat(Nat)`⟩
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 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

2.3 Algoritmos

TABLAS(**in** b : `estr`) $\longrightarrow$ *res* : `ConjTrie(string)`
  $res \leftarrow b.tablas.DiccClaves$ — O(1)
  O(1)

HAYJOIN?(**in** *t1* : `string`, *in t2* : `string`, *in b* : `estr`) $\longrightarrow$ *res* : `bool`
  res ←Definido?(Obtener(b, t1).Joins,t2) ← Definido?(Obtener(b, t2).Joins,t1) — O(1)

CAMPOJOIN(**in** *t1* : `string`, *in t2* : `string`, *in b* : `estr`) $\longrightarrow$ *res* : `campo`
  res← Obtener(Obtener(b, t1).Joins, t2).campoJ — O(1)

NUEVADB() $\longrightarrow$ *res* : `estr)`
  String s
  Nat n ← 0
  res ← $\langle\langle s, 0\rangle, vacio()\rangle$ — O(1)
  O(1)

AGREGARTABLA(**in** t : `tabla`, *in/out b* : `estr`)
  info_tabla ← $\langle t.cantidadDeAccesos, t, vacio()\rangle$ — O(1)
  Definir(b.tablas, nombre(t), info_tabla) — O(1)
  O(1)

INSERTARENTRADA(**in** *reg* : `registro`, *in t* : `string`, *in/out b* : `estr`)
  Obtenemos la tabla es O(1) porque su nombre esta acotado.
  info_tabla infoT ← Obtener(b.tablas, t).TActual — O(1) Por referencia
  Agrego el registro a la tabla. — O(1)
  Tabla T ← infoT.TActual — O(1)
  agregarRegistro(reg, T) — O(1)
  Ahora si hay Joins actualizo la informacion temporal de cada Join.
  if ¬∅?(infoT.Joins) **then** — O(1)
    ItConjString(String) itNomTab ← CrearIt(Claves(infoT.Joins)) — O(1)
    while HaySiguiente?(itNomTab) **do** — O(1)
      info_join infoJ ← Obtener(infoT.Joins, Siguiente(NomTab)) — O(1)
      Encolar(infoJ.Rcambios, $\langle reg, Siguiente(NomTab), true\rangle$) — O(L)
      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) — O(1)
    end while
  end if
  if CantidadDeAccesos(T) b.TablaMaxima.#Modif **then**
    b.TablaMaxima.NomTabla ← Copiar(Nombre(T)) — O(L)
    b.TablaMaxima.#Modif ← Copiar(CantidadDeAccesos(T)) — O(1)
  end if
  O(T*L + Log(n)) siendo n la cantidad de r

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

```
info_tabla infoT ← Obtener(b.tablas, t).TActual                    O(1) por ref
Tabla T ← infoT.TActual                                            O(1) por ref
NombreTabla NomTab ← NombreTabla(T)
La eliminacion en primera etapa depende de si hay
joins con la tabla pasada por parametro
if ¬∅?(Claves(infoT.Joins)) then                                   O(1)
    Creo el iterador, para navegar los nombres de tablas con los que tiene Join
    itNom ← CrearIt(Claves(infoT.Joins))                           O(1)
    while HaySiguiente?(itNom) do                                  O(Cant de tablas)
        info_join infoJ ← Siguiente(itNom)                         O(1)
        Verifico si el Join esta creado en base al
        campo del cr pasado por parametro.
        if infoJ.campoJ=DameUno(Campos(cr)) then                   O(1)
            Entonces solo actualizo la cola temporal del join
            Registro reg ← Buscar(cr, T)                           O(L + Log(n))
            Encolar(infoJ.Rcambios, ⟨reg, Siguiente(itNom), False⟩)
                                                                   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 ← Buscar(cr, T)                     O(L + Log(n))
            ItConj(Registro) itReg ← CrearIt(cjc)
            Luego agrego estos registros a la cola temporal de cambios del join
            while HaySiguiente?(itReg) do
                Encolar(infoJ.Rcambios, ⟨Siguiente(itReg), NomTab, False⟩)
                                                                   O(L)
                Avanzar(itReg)                                     O(1)
            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 ← Copiar(Nombre(T))                     O(L)
    b.TablaMaxima.#Modif ← Copiar(CantidadDeAccesos(T))
                                                                   O(1)
end if                                                             ---------
                                                                   O(T*L + n)
```

```
generarVistaJoin(in t1 : string, in t2 : string, in c : campo, in/out b : estr)
    Join ← vacio()
    T_actual1 ← Obtener(b.tablas, t1).Tactual                      O(1)
    T_actual2 ← Obtener(b.tablas, t2).Tactual                      O(1)
    if Pertenece?(Indices(T_actual1), c) ∧ Pertenece?(Indices(T_actual1), c) then
                                                                   O(1)
        ind1 ← Obtener(T_actual1.Indices, c)                       O(calcular)
```

```
    if tipoCampo(T_actual1, c) then
        ConjNat(Nat) cjNat ← vacio()
        itvalores ← CrearItConjNat(ind1.PorNat.DiccClaves)
        while HaySiguiente(itvalores) do
            Registro r ← Obtener(ind1.PorNat, Siguiente(itvalores))
            Nat n ← ValorNat(Obtener(r, c))
            if ¬ Pertenece?(cjNat, n) then
                AgregarRapido(cjNat, r)
            end if
            Avanzar(itvalores)
        end while
        ind2 ← Obtener(T_actual2.Indices, c)
        itvalores ← CrearItConjNat(ind2.PorNat.DiccClaves)
        while HaySiguiente(itvalores) do
            Registro r ← Obtener(ind2.PorNat, Siguiente(itvalores))
            Nat n ← ValorNat(Obtener(r, c))
            if ¬ Pertenece?(cjNat, n) then
                AgregarRapido(cjNat, r)
            end if
            Avanzar(itvalores)
        end while
    else
        itvalores ← CrearItConjString(ind1.PorString.DiccClaves)
        while HaySiguiente(itvalores) do
            r1 ← Obtener(ind1.PorString, Siguiente(itvalores))
            r2 ← Obtener(ind2.PorString, Siguiente(itvalores))
            cj1 ← AgregarRapido(vacio(), r1)
            cj2 ← AgregarRapido(vacio(), r2)
            nuevor ← combinarRegistros(c, cj1, cj2)
            AgregarRapido(Join, DameUno(nuevor))
            Avanzar(itvalores)
        end while
    end if
else
    cjr1 ← T_actual1.registros
    cjr2 ← T_actual1.registros
    Join←combinarRegistros(c, cjr1,cjr2)
end if
info_join ← ⟨0, vacio(), vacio(), c, tipoCampo(T_actual1, c), Join⟩
Definir(b.Joins, ⟨t1, t2⟩ , info_join)
```

O(1)

BORRARJOIN(**in** *t1* : `string`, *in t2* : `string`, *in/out b* : `estr`)

```
if Pertenece?(b.Joins, ⟨t1, t2⟩ ) then
    Borrar(b.Joins, ⟨t1, t2⟩ )
else
    if Pertenece?(b.Joins, ⟨t2, t1⟩ ) then
        Borrar(b.Joins, ⟨t2, t1⟩)
    end if
end if
```

O(1)

BUSCAR(**in** *criterio* : `registro`, *in t* : `string`, *in b* : `base`) ⟶ *res* : `conj(ItConj(registro))`

```
if Pertenece?(b.Joins, ⟨t1, t2⟩ ) then
    Borrar(b.Joins, ⟨t1, t2⟩ )
else
    if Pertenece?(b.Joins, ⟨t2, t1⟩ ) then
        Borrar(b.Joins, ⟨t2, t1⟩)
    end if
end if
```

O(1)