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
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`,
Registros : `Conj(Registro)`,
Campos : `DiccTrie(Campo,Tipo)`,
Claves : `ConjTrie(Campo)`,
IndiceS : `tupla`⟨CampoI : `campo`, ,
EnUso : `bool`,
Indice : `DiccTrie(string, Conj(ItConj(Registro)))`⟩
IndiceN : `tupla`⟨CampoI : `campo`, ,
EnUso : `bool`,
Indice : `DiccNat(nat, Conj(ItConj(Registro)))`⟩
#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?(Obtener(r,c)) es igual Obtener(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 Obtener(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, Obtener(r, t.IndiceS.CampoI)) es true y r pertenece a Obtener(t.IndiceS.Indice, Obtener(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, mas la cantidad de indices creados.

Función de abstracción

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

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

1.3 Algoritmos

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

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

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

$res \leftarrow t.Claves.DiccClaves$	O(1)
	O(1)

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

res $\leftarrow$ 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 : `ConjTrie(campo)`

$res \leftarrow t.Campos.DiccClaves$	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)`

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

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)
DiccTrie(campo, tipo) Campos $\leftarrow$ Vacio()	O(1)
ConjTrie(campo) Claves_ $\leftarrow$ Vacio()	O(1)
IndiceS $\leftarrow <$ $DameUno(claves), False, Vacio() >$	O(1)
IndiceN $\leftarrow <$ $DameUno(claves), False, Vacio() >$	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(r, 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)
while HaySiguiente(itclaves) **do**	O(1)
Esto se debe a que # de campos a iterar es acotada.	
AgregarRapido(re.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* t : `tab`)	
itConj(Registro) nuevo ← AgregarRapido(t.Registros,r)	$\theta(1)$
t.#Accesos++	$\theta(1)$
if t.IndiceS.EnUso **then**	
valor ← Obtener(r, t.IndiceS.CampoI)	$\theta(1)$
if Definido?(t.IndiceS.Indice, valor) **then**	$\theta(1)$
viejo ← Obtener(t.IndiceS.Indice, valor)	$\theta(1)$
AgregarRapido(viejo, nuevo)	$\theta(1)$
else	
viejo ← Vacio()	$\theta(1)$
AgregarRapido(viejo, nuevo)	$\theta(1)$
Definir(t.IndiceS.Indice, valor, viejo)	$\theta(1)$
end if	
end if	
if t.IndiceN.EnUso **then**	
valor ← Obtener(r, t.IndiceN.CampoI)	θ(Log(n)
if Definido?(t.IndiceN.Indice, valor) **then**	θ(log(n))
viejo ← Obtener(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)	θ(Log(n))
end if	
end if	
Donde n es la cantidad de cantidad de valores distintos definidos en t.IndiceN	
	θ(Log(n))
BORRARREGISTRO(**in** *crit* : `registro`, **in** t : `tab`)	
c ← Siguiente(Campos(crit))	$\theta(1)$
valor ← Obtener(crit, c)	$\theta(1)$
if t.IndiceS.EnUso ∧ t.IndiceS.CampoI=c **then**	
if Definido?(t.IndiceS.Indice, valor) **then**	
itConj(registro) itr ← Obtener(t.IndiceS.Indice, valor)	$\theta(1)$
EliminarSiguiente(itr)	$\theta(1)$
Borrar(t.IndiceS.Indice, valor)	$\theta(1)$
end if	
end if	
if t.IndiceN.EnUso ∧ t.IndiceN.CampoI=c **then**	
if Definido?(t.IndiceN.Indice, valor) **then**	
itConj(registro) itr ← Obtener(t.IndiceN.Indice, valor)	$\theta(1)$
EliminarSiguiente(itr)	$\theta(1)$
Borrar(t.IndiceN.Indice, valor)	$\theta(1)$
end if	
end if	
itConj(registro) cr ← CrearItConj(t.registros)	$\theta(1)$
Dato valorR ← Obtener(Siguiente(cr), c)	$\theta(1)$
while HaySiguiente(cr) **do**	θ(Cardinal(t.registros))
valorR ← Obtener(Siguiente(cr), c)	$\theta(1)$
if valorR=valor **then**	
EliminarSiguiente(cr);	$\theta(1)$

end if
Avanzar(cr) $\theta(1)$
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))
siendo n la cantidad de registros de la tabla pasada por parametro.
En caso contrario borrar es O(n).

INDEXAR(**in** c : `campo`, *in* 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 ← Obtener(Siguiente(cr), c)
itConj(registro) itr ← CrearItConj(Siguiente(cr))
if Definido?(t.IndiceN.Indice, valor) **then**
regviejos ← Obtener(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 ← Obtener(Siguiente(cr), c)
itConj(registro) itr ← CrearItConj(Siguiente(cr))
if Definido?(t.IndiceS.Indice, valor) **then**
regviejos ← Obtener(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

$\theta(1)$

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

θ(calcular)

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

bool valor ← True	
if Cardinal(campos(r))=Cardinal(t.Campos.DiccClaves) **then**	
itcampos ← CrearItTrie(t.Campos.DiccClaves)	
while valor ∧ HaySiguiente(itcampos) **do**	$\theta(1)$
Campo c ← Siguiente(itcampos)	$\theta(1)$
valor ← Definido?(r, c)	$\theta(1)$
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)

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

if TipoCampo(c, t) **then**	
res ← ¬(t.IndiceN.EnUso)	
else	
res ← ¬(t.IndiceS.EnUso)	
end if	
	O(1)

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

itcr1 ← CrearItConjTrie(cr1)	$\theta(1)$
copiacr2 ← Copiar(cr2)	θ(Cardinal(cr2))
while HaySiguiente(itcr1) **do**	θ(Cardinal(cr1))
combinarTodos(c,Siguiente(itcr1),copiacr2)	$\theta(1)$
Avanzar(itcr1)	$\theta(1)$
end while	
res ← copiacr2	$\theta(1)$
	O(Cardinal(cr1))

HAYCOINCIDENCIA(**in** r : `registro`, *in cc* : `ConjTrie(campo)`, *in cr* : `Conj(registro)`) $\longrightarrow$ *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)`) $\longrightarrow$ *res* : `Conj(registro)`

Conj(registro) salida ← Vacio()	
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))                    θ(1)
      end if
      Avanzar(itcr);                                             θ(1)
  end while
                                                                 ---------
                                                                 O(Cardinal(cr))
```

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

```
  res ← min( dameColumna( c, t.registros ))                      θ(Cardinal(t.registros))
                                                                 ---------
                                                                 O(Cardinal(t.registros))
```

MAXIMO(**in** c : `campo`, *in* t : `tab`) $\longrightarrow$ *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)`

```
  Conj(Dato) cj ← vacio();                                       θ(1)
  if Cardinal(cr)≥1 then                                         θ(1)
    Tvalor ← Tipo?(Obtener(DameUno(cr), c))                      θ(1)
    if Tvalor then
      ConjLog(nat) cj ← Vacio()                                  θ(1)
    else
      ConjTrie(string) cj ← Vacio()                              θ(1)
    end if
    itcr ← CrearItConj(cr)                                       θ(1)
    cjd ← Vacio()
    while HaySiguiente(itcr) do                                  θ(Cardinal(cr))
      Dato data ← Obtener(Siguiente(itcr), c)
      if Tvalor then
        if ¬Pertenece?(cj, valorNat(data)) then                  θ(Log(n))
          AgregarRapido(cjd, data)                               θ(1)
        else
          AgregarRapido(cj, valorNat(data))                      θ(1)
        end if
      else
        if ¬Pertenece?(cj, valorString(data)) then               θ(1)
          AgregarRapido(cjd, data)                               θ(1)
        else
          AgregarRapido(cj, valorString(data))                   θ(1)
        end if
      end if
      Avanzar(itcr);                                             θ(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)).
  El cardinal de res es la cantidad de datos distintos.
                                                                 ---------
                                                                 O(nlog(n))
```

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

```
  valor ← True                                                   θ(1)
  itconjClaves ← CrearItConj(r.ClavesDicc)                       θ(1)
```

while valor ∧ HaySiguiente(itconjClaves) **do**	$\theta(1)$
val1← tipo?(Obtener(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 : `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** `DiccTrie(string, info_tabla)`
`donde info_tabla es tupla`⟨#Accesos : `nat`,
TActual : `tabla`,
Joins : `DiccTrie(string; info_join)`⟩
`donde info_join es tupla`⟨R : `nat`,
Rcambios : `Lista(registro)`,
campoJ : `campo`,
campoT : `tipo`,
JoinS : `ConjTrie(registro)`,
JoinN : `ConjNat(registro)`⟩

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)`
  res ← vacio() — O(1)

O(1)

AGREGARTABLA(**in** t : `tabla`, *in 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 b* : `estr`)
  T_actual ← Obtener(b.tablas, t).TActual — O(1)
  agregarRegistro(reg, T_actual) — O(1)
  itClaves ← CrearItConjTrie(T_actual.Joins.DiccClaves) — O(1)

  while HaySiguiente(itClaves) **do** — O(1)
    t2 ← Obtener(b, Siguiente(itClaves)) — O(1)
    nuevor ← combinarRegistros(c, cj1, t2.registros)
    AgregarRapido()
    Avanzar(itClaves)

  end while

O(1)

BORRAR(**in** *cr* : `registro`, *in t* : `string`, *in b* : `estr`)
  T_actual ← Obtener(b.tablas, t).TActual — O(1)
  borrarRegistro(r, T_actual) — O(1)
  Falta hacer algo?

O(1)

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 Definido?(T_actual1.Indices, c) $\wedge$ Definido?(T_actual2.Indices, c) **then**
    ind1 ← Obtener(T_actual1.Indices, c)
    ind2 ← Obtener(T_actual2.Indices, c)
    if tipoCampo(T_actual1, c) **then**

```
        itvalores ← CrearItConjNat(ind1.PorNat.DiccClaves)
        while HaySiguiente(itvalores) do
            r1 ← Obtener(ind1.PorNat, Siguiente(itvalores))
            r2 ← Obtener(ind2.PorNat, Siguiente(itvalores))
            cj1 ← AgregarRapido(vacio(), r1)
            cj2 ← AgregarRapido(vacio(), r2)
            nuevor ← combinarRegistros(c, cj1, cj2)
            AgregarRapido(Join,DameUno(nuevor))
            Avanzar(itvalores)
        end while
    else
        itvalores ← CrearItConjString(ind1.PorString.DiccClaves)
        while HaySiguiente(itvalores) do
            r1 ← Obtener(ind1.PorString, Siguiente(itvalores))
            r2 ← Obtener(ind2.PorString, Siguiente(itvalores))
            cj1 ← AgregarRapido(vacio(), r1)
            cj2 ← AgregarRapido(vacio(), r2)
            nuevor ← combinarRegistros(c, cj1, cj2)
            AgregarRapido(Join, DameUno(nuevor))
            Avanzar(itvalores)
        end while
    end if
else
    cjr1 ← T_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)