Algoritmos y Estructuras de Datos II

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo práctico 1

"Álgebra relacional"

Integrante	LU	Correo electrónico
Carreira, Leandro	669/18	carreiraleandroe@gmail.com
Capello, Bruno Ignacio	623/17	bruno.icapello@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

1. TADs

TAD BaseDeDatos

géneros bdd

usa Consulta, Tabla, Registro, Bool, ...

exporta observadores, generadores

igualdad observacional

$$(\forall b_1, b_2 : \text{bdd}) \left(b_1 =_{\text{obs}} b_2 \iff \left(\begin{array}{l} (\forall \text{ nt: nombreTabla}) \\ (\text{pertenece?}(b_1, \text{nt}) \Rightarrow \text{pertenece?}(b_2, \text{nt})) \\ \wedge\ (\forall \text{ nt': nombreTabla}) \\ (\text{pertenece?}(b_2, \text{nt'}) \Rightarrow \text{pertenece?}(b_1, \text{nt'})) \\ \wedge\ \neg(\exists \text{ t,t': tabla}) \\ (\text{t} \neq \text{t'} \wedge \text{está?}(b_1,\text{t}) \wedge \text{está?}(b_1,\text{t'}) \\ \wedge\ \text{nombreTabla(t)=nombreTabla(t')}) \end{array} \right) \right)$$

observadores básicos

tablas	: bdd	$\longrightarrow$	conj(tabla)	
pertenece?	: bdd × nombreTabla	$\longrightarrow$	bool	
nombreTabla	: bdd b × tabla t	$\longrightarrow$	nombre_tabla	{está?(b, t)}

generadores

crearBDD	:	$\longrightarrow$	bdd	
agTabla	: bdd b × nombreTabla nt × tabla	$\longrightarrow$	bdd	{¬(pertenece?(b, nt))}

otras operaciones

está?	: bdd × tabla	$\longrightarrow$	bool	
eliminarTabla	: bdd b × nombreTabla nt	$\longrightarrow$	bdd	{pertenece?(b, nt)}
eliminarRegistro	: bdd b × tabla t × registro r	$\longrightarrow$	bdd	{está?(b, t) $\wedge_L$ r $\in$ registros(t))}
agregarRegistro	: bdd b × tabla t × registro r	$\longrightarrow$	bdd	{está?(b, t) $\wedge_L$ r $\notin$ registros(t)}
consultar	: conj(tabla) C × consulta q	$\longrightarrow$	conj(registro)	{tipo_consulta(q) $\neq$ FROM}

conj(registro)

from	: conj(tabla) C × nombreTabla nt	$\longrightarrow$	conj(registro)	{($\exists$ t':tabla)(t' $\in$ C $\wedge_L$ nt = nombreTabla(t')}
select	: conj(registro) R × campo$_1$ c × valor v	$\longrightarrow$	conj(registro)	{($\forall$ r:registro)(r $\in$ R $\Rightarrow_L$ c $\in$ campos(r) $\wedge_L$ ($\exists$ r':registro)(r' $\in$ R $\wedge_L$ r'[c] = v}
match	: conj(registro) R × campo$_1$ c$_1$ × campo$_2$ c$_2$	$\longrightarrow$	conj(registro)	{c$_1$ $\neq$ c$_2$ $\wedge_L$ ($\forall$ r:registro)(r $\in$ R $\Rightarrow_L$ c$_1$ $\in$ campos(r) $\wedge$ c$_2$ $\in$ campos(r)}
proj	: conj(registro) R × conj(nombreCampo) C	$\longrightarrow$	conj(registro)	{($\forall$ r:registro)(r $\in$ R $\Rightarrow_L$ C $\subseteq$ campos(r)}
rename	: conj(registro) R × campo$_1$ c$_1$ × campo$_2$ c$_2$	$\longrightarrow$	conj(registro)	{($\forall$ r: registro)(r $\in$ R $\Rightarrow_L$ c$_1$ $\in$ campos(r)}
product	: conj(registro) × conj(registro)	$\longrightarrow$	conj(registro)	
conTodos	: registro × conj(registro)	$\longrightarrow$	conj(registro)	
cambiarNombre	: registro × campo$_1$ × campo$_2$	$\longrightarrow$	registro	
eliminarCampos	: conj(nombre_campo) × conj(registro)	$\longrightarrow$	conj(registro)	
conjSinCampo	: nombre_campo × conj(registro)	$\longrightarrow$	conj(registro)	
regSinCampo	: nombre_campo × registro	$\longrightarrow$	registro	

axiomas $\forall$ b : bdd, t, t' : tabla, r : registro, cr : conj(registro), n, c : nombreCampo, cn, C: conj(nombreCampo), q : consulta, c_1 : campo$_1$, c_2 : campo$_2$, v : valor

tablas(crearBDD) ≡ ∅
tablas(agTabla(b, nt, t)) ≡ Ag(t, tablas(b))

pertenece?(crearBDD,t) ≡ false
pertenece?(agTabla(b, nt', t'),t) ≡ **if** t=t' **then**
true
else
pertenece?(b,t)
fi
nombreTabla(agTabla(b, nt', t'),t) ≡ **if** nombreTabla(t)=nt' **then**
nt'
else
nombreTabla(b,t)
fi
está?(tablas(b),t) ≡ pertenece?(b,nombreTabla(b,t))
eliminarTabla(agTabla(b, nt', t'), t) ≡ **if** t=t' **then**
b
else
eliminarTabla(b,t)
fi
agregarRegistro(agTabla(b, nt', t'), t, r) ≡ **if** t'=nombreTabla(b,t) **then**
Ag(b, insertar(t, r))
else
agTabla(agregarRegistro(b, t, r), nt', t')
fi
eliminarRegistro(agTabla(b, nt', t'), t, r) ≡ **if** nt'=nombreTabla(b,t) **then**
agTabla(b, nt', borrar(t,r[clave(t)]))
else
agTabla(eliminarRegistro(b, t, r), nt', t')
fi

```
consultar(tablas(b), q) ≡ if tipo_cons(q) ∉ {INTER, UNION, PRODUCT}∧_L
                            tipo_consulta(subconsulta_1(q)) = FROM then
                              if nombre_tabla(subconsulta_1(q)) = nombre_tabla(dameUno(tablas(b))
                              then
                                if tipo_consulta(q) = SELECT then
                                  select(consultar(tablas(b), subconsulta_1(q)), campo_1(q), valor(q))
                                else
                                  if tipo_consulta(q) = MATCH then
                                    match(consultar(tablas(b), subconsulta_1(q)), campo_1(q),
                                    campo_2(q))
                                  else
                                    if tipo_consulta(q) = PROJ then
                                      proj(consultar(tablas(b), subconsulta_1(q)),
                                      conj_campos(q))
                                    else
                                      rename(consultar(tabla(b), subconsulta_1(q)), campo_1(q),
                                      campo_2(q))
                                    fi
                                  fi
                                fi
                              else
                                consultar(sinUno(tablas(b)), q)
                              fi
                            else
                              if tipo_consulta(q) = INTER then
                                consultar(tablas(b), subconsulta_1(q)) ∩ consultar(tablas(b),
                                subconsulta_2(q))
                              else
                                if tipo_consulta(q) = UNION then
                                  consultar(tablas(b), subconsulta_1(q)) ∪ consultar(tablas(b),
                                  subconsulta_2(q))
                                else
                                  product(consultar(tabla(b), subconsulta_1(q)), consultar(tabla(b),
                                  subconsulta_2(q)))
                                fi
                              fi
                            fi

from(tablas(b), t) ≡ if nombreTabla(dameUno(tablas(b))) = t then
                       registros(dameUno(tablas(b)))
                     else
                       from(sinUno(tablas(b)), t)
                     fi

select(cr, c_1, v) ≡ if vacío?(from(tablas(b),t)) then
                       {}
                     else
                       if dameUno(cr)[c1] = v then
                         Ag(dameUno(cr), select(sinUno(cr), c1 ,v)
                       else
                         select(sinUno(cr))
                       fi
                     fi

match(cr,c_1,c_2) ≡ if vacío?(cr) then
                       {}
                     else
                       if dameUno(cr)[c_1] = dameUno(cr)[c_2] then
                         Ag(dameUno(cr), match(sinUno(cr), c_1, c_2)
                       else
                         match(sinUno(cr), c_1, c_2))
                       fi
                     fi
```

```
proj(cr, C) ≡ if campos(dameUno(cr)) ⊆ C then
                  cr
              else
                  eliminarCampos(campos(dameUno(cr)) - C, cr)
              fi

eliminarCampos(cn, cr) ≡ if vacío?(cr) ∨ vacío?(cn) then
                             cr
                         else
                             conjSinCampo(dameUno(cn), cr) ∪ eliminarCampos(sinUno(cn), cr)
                         fi

conjSinCampo(n, cr) ≡ if vacío?(cr) then
                          {}
                      else
                          Ag(regSinCampo(n, dameUno(cr)), conjSinCampo(n, sinUno(cr)))
                      fi

regSinCampo(n, definir(r, c, v)) ≡ if vacío?(campos(r)) then
                                       if n = c then
                                           r
                                       else
                                           definir(r, c, v)
                                       fi
                                   else
                                       if n = c then
                                           regSinCampo(n,r)
                                       else
                                           definir(regSinCampo(n,r), c, v)
                                       fi
                                   fi

rename(cr, c1, c2) ≡ if c2 ∈ campos(dameUno(cr)) then
                         rename(cr, c1, (c2[tam(c2)+1] ← " − " ← "b" ← "i" ← "s"))
                     else
                         if c1 ∈ campos(dameUno(cr)) then
                             Ag(cambiarNombre(dameUno(cr), c1, c2), rename(sinUno(cr), c1, c2)
                         else
                             Ag(dameUno(cr), rename(sinUno(cr), c1, c2))
                         fi
                     fi

cambiarNombre(definir(r, c, v), c1, c2) ≡ if vacío?(campos(r)) then
                                              if c = c1 then
                                                  definir(r, c2, v)
                                              else
                                                  definir(r, c, v)
                                              fi
                                          else
                                              if c = c1 then
                                                  definir(r, c2, v)
                                              else
                                                  definir(cambiarNombre(r, c1, c2), c, v)
                                              fi
                                          fi


product(c1, c2) ≡ if vacío?(c2) then
                      c1
                  else
                      if vacío?(c1) then
                          {}
                      else
                          conTodos(dameUno(c1), c2)) ∪ PRODUCT(sinUno(c1), c2)
                      fi
                  fi
```

conTodos(r, c) $\equiv$ **if** vacío?(c) **then**
{}
else
Ag(concatenar(r, dameUno(c)), conTodos(r, sinUno(c)))
fi

Fin TAD

2. Extensiones

TAD Registro Extendido

extiende Registro

otras operaciones

concatenar : registro × registro $\longrightarrow$ registro

axiomas

concatenar(definir(r_1, c_1, v_1), definir(r_2, c_2, v_2)) $\equiv$ **if** vacío?(campos(definir(r_1, c_1, v_1)) **then**
definir(r_2, c_2, v_2)
else
if vacío?(campos(definir(r_2, c_2, v_2)) **then**
definir(r_1, c_1, v_1)
else
if $c_1 = c_2$ **then**
concatenar(definir(r_1, c_1, v_1), r_2)
else
definir(concatenar(definir(r_1, c_2, v_2), c_1, v_1))
fi
fi
fi

Fin TAD

TAD Diccionario Extendido

extiende diccionario(clave,significado)

otras operaciones

significados : dicc(clave × significado) $\longrightarrow$ conj(significado)

axiomas

significados(vacio) $\equiv$ $\emptyset$

significados(definir(c, s, d)) $\equiv$ Ag(s, significados(d))

Fin TAD

3. Módulo BaseDeDatos

Este módulo provee todas las operaciones permitidas sobre una base de datos. Una base de datos tiene tablas unicas y dentro de las tablas hay registros. El módulo permite agregar una tabla que no estaba en la base de datos, como tambien permite eliminar una tabla de la base de datos. tambien se permite agregar o eliminar un registro cuyos campos pertenezcan a alguna tabla. Por ultimo se puede realizar consultas a la base de datos, devolviendo conjunto de registros.

Interfaz

se explica con: TAD BaseDeDatos

géneros: `bdd`.

Servicios Usados: Consulta, Tabla, Registro, DiccionarioString(α), Conjunto Lineal(α)

Operaciones básicas de BaseDeDatos

crearBDD() $\rightarrow$ *res* : `bdd`
Pre $\equiv$ {true}
Post $\equiv$ {*res* $=_{\text{obs}}$ crearBDD}
Complejidad: O (1)
Descripción: Genera una bdd vacia.

AgregarTabla(**in/out** *b*: `bdd`, **in** *nt*: `nombre_tabla`, **in** *t*: `tabla`)
Pre $\equiv$ {$b =_{\text{obs}} b_0 \ \wedge \ \neg(pertenece(b_0, nt, t))$}
Post $\equiv$ {$b =_{\text{obs}}$ agTabla(b_0, nt)}
Complejidad: O ($|nt| + copy(t)$)
Descripción: Agrega la tabla t de nombre nt a la base de datos.

EliminarTabla(**in/out** *b*: `bdd`, **in** *nt*: `nombre_tabla`)
Pre $\equiv$ {$b =_{\text{obs}} b_0 \ \wedge \ (pertenece(b_0, nt))$}
Post $\equiv$ {$b =_{\text{obs}}$ eliminarTabla(b_0, nt)}
Complejidad: O ($|nt| \bullet \#b$)
Descripción: Elimina de la base de datos la tabla de nombre nt .

AgregarRegistro(**in/out** *b*: `bdd`, **in** *t*: `tabla`, **in** *r*: `registro`)
Pre $\equiv$ {$b =_{\text{obs}} b_0 \ \wedge esta?(b_0, t) \ \wedge_L \ r \notin registros(t)$}
Post $\equiv$ {$b =_{\text{obs}}$ agregarRegistro(b_0, t, r)}
Complejidad: O ($|nt| + copy(r)$)
Descripción: Le agrega a la tabla t de la base de datos un registro r que no tenia.

EliminarRegistro(**in/out** *b*: `bdd`, **in** *t*: `tabla`, **in** *r*: `registro`)
Pre $\equiv$ {$b =_{\text{obs}} b_0 \ \wedge \ esta(b_0, t) \ \wedge_L \ r \in registros(t)$}
Post $\equiv$ {$b =_{\text{obs}}$ eliminarRegistro(b_o, t, r)}
Complejidad: O ($|nt| \bullet \#b$)
Descripción: Le elimina a la tabla t de la base de datos un registro r que contenia.

Consultar(**in** *b*: `bdd`, **in** *q*: `consulta`) $\rightarrow$ *res* : `conj(registro)`
Pre $\equiv$ {$tipo\_consulta(q) \neq \underline{FROM}$}
Post $\equiv$ {$res =_{\text{obs}}$ consultar($tablas(b), q$)}
Complejidad: O ($|t| + |q| + k(copy(r))$)
Descripción: Dada una consulta q, la base de datos devuelve el conjunto con los registros de alguna tabla que cumplen con la consulta .

Representación

Representación la Base de Datos

La Base de datos se representa con tablas que estan asociadas con un nombre unico.

`bdd` **se representa con** `estrBdd`

donde `estrBdd` es `tupla`(*columnas*: `diccString(estrTabla)`)

Rep : estrBdd $\longrightarrow$ bool
Rep(*bdd*) $\equiv$ true

Abs : estrReg r $\longrightarrow$ registro {Rep(r)}

$$\begin{aligned}\text{Abs}(r) \equiv\ & (\forall b : bdd)/\ bdd.significados = tablas(b)\ \wedge\ (\forall t : tabla)(esta?(b,t) \Rightarrow_L \\ & nombreTabla(b,t) \in t.claves)\ \wedge_L \\ & (\forall nt : nombreTabla)(pertenece?(b,nt) \Longleftrightarrow_L\ bdd.definido?(nt)\end{aligned}$$

Algoritmos

---

iCrearBDD() $\rightarrow res$: estrBdd

1: $res \leftarrow nuevoDicc$

Complejidad: O (1)
Justificacion: La operacion nuevodicc cuesta O(1).

---

iAgregarTabla(**in**/**out** b: `estrBdd`, **in** nt: `nombreTabla`, **in** t: `estrTabla`)

1: $b.definir(nt,t)$

Complejidad: O ($|nt| + copy(t)$)
Justificacion: De precondicion nt no estaba definida en la bdd.

---

iEliminarTabla(**in**/**out** b: `estrBdd`, **in** nt: `nombreTabla`)

1: $b.borrar(nt)$

Complejidad: O ($|nt| \bullet \#b$)
Justificacion: el costo de borrar es de O ($|C| \bullet \#c$) donde c el la clave mas larga y $\#$c es la cantidad de claves

---

iAgregarRegistro(**in**/**out** b: `estrBdd`, **in** t: `estrTabla`, **in** r: `estrReg`)

1: **for** nombreTabla nt : b.claves **do**
2: **if** b[nt].nombreTabla = t.nombreTabla **then**
3: $b[nt].registros.definir(r)$
4: **end if**
5: **end for**

Complejidad: O ($|nt| + copy(r)$)
Justificacion: Busca la tabla que se identifica con nombre unico nt y le define r (que de precondicion no la tiene).

---

iEliminarRegistro(**in**/**out** b: `estrBdd`, **in** t: `estrTabla`, **in** r: `estrReg`)

1: **for** nombreTabla nt : b.claves **do**
2: **if** b[nt].nombreTabla = t.nombreTabla **then**
3: $b[nt].registros.borrar(r)$
4: **end if**
5: **end for**

Complejidad: O ($|nt| \bullet \#b$)
Justificacion: Busca la tabla que se identifica con nombre unico nt y le define r (que de precondicion la tiene).

---

iConsultar(**in** b: `estrBdd`, **in** q: `estrConsulta`) → res : conj(estrReg)

```
1: for nombreCampo c : q.claves do
2:     for nombreTabla nt : b.claves do
3:         if b[nt].registros.definido?(c) then
4:             res.agregarRapido(q.obtener(c))
5:         end if
6:     end for
```

Complejidad: $O(|t| + |q| + k(copy(r))$
Justificacion: se identifica y se obtiene de las tablas los registros que esten definidos en la base de datos.

4. Módulo Consulta

Este módulo provee todas las distintas consultas que se pueden hacer con una base de datos. Las consultas se construyen recursivamente y el resultado de una consulta sobre una base de datos es un conjunto de registros de una tabla.

Interfaz

se explica con: TAD Consulta

géneros: `consulta`.

Servicios Usados: Registro, DiccionarioString(α), Conjunto Lineal(α)

Operaciones básicas de consulta

From(**in** nt: `nombre_tabla`) → res : `consulta`
Pre ≡ {$true$}
Post ≡ {$res =_{obs}$ FROM(nt}
Descripción: si nt es el nombre de una tabla de la base de datos, esta operacion devuelve el conjunto de los registros de esa tabla. en caso de que nt no sea el nombre de una tabla definida en la base de datos esta operacion devuelve un mensaje de error.

Proj(**in** cr: `consulta`, **in** C: `conj(campo)`) → res : `consulta`
Pre ≡ {true}
Post ≡ {$res =_{obs}$ PROJ(cr, C) }
Descripción: Si cr es una consulta y C es un conjunto de campos, PROJ(cr, C) es una nueva consulta que devuelve los mismos registros que la consulta cr, pero que incluyen solamente los campos del conjunto C.

Rename(**in** cr: `consulta`, **in** c_1: `nombre_campo`, **in** c_2: `nombre_campo`) → res : `consulta`
Pre ≡ {true}
Post ≡ {$res =_{obs}$ RENAME(cr, c_1, c_2) }
Descripción: Si cr es una consulta y c_1 y c_2 son nombres de campos, Rename(cr, c_1, c_2) renombra el campo c_1 en el resultado de la consulta q para que pase a llamarse c_2.
Si el campo c_1 no existe devuelve el mismo conjunto recibido.
Si el campo c_2 ya existe, le agrega los caracteres " $-$ *bis*" al final de c_2 hasta que no haya campos con ese nombre.

Inter(**in** cr_1: `consulta`, **in** cr_2: `consulta`) → res : `consulta`
Pre ≡ {true}
Post ≡ {$res =_{obs}$ INTER(cr_1, cr_2)}
Descripción: Si cr_1 y cr_2 son consultas, Inter(cr_1, cr_2) es una nueva consulta que devuelve la intersección entre los registros de las consultas cr_1 y cr_2.

Union(**in** cr_1: `consulta`,**in** cr_2: `consulta`) → res : `consulta`
Pre ≡ {true}
Post ≡ {$res =_{obs}$ UNION(cr_1, cr_2)}
Descripción: Si cr_1 y cr_2 son consultas, Union(cr_1, cr_2) es una nueva consulta que devuelve la unión entre los registros de las consultas cr_1 y cr_2.

select(**in** cr: `consulta`, **in** c: `nombre_campo`, **in** v: `valor`) → res : `consulta`

Pre $\equiv$ {true}
Post $\equiv$ {$res =_{obs}$ SELECT(cr, c, v)}
Descripción: Si cr es una consulta, c es el nombre de un campo y v es un valor, SELECT(cr, c, v) es una nueva consulta que devuelve los registros de la consulta cr tales que el campo c tiene valor v. Los registros que no tengan el campo c esta operacion no los devuelve.

MATCH(**in** cr : `consulta`, **in** c_1 : `nombre_campo`, **in** c_2 : `nombre_campo`) $\rightarrow$ res : `consulta`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{obs}$ MATCH(cr, c_1, c_2)}
Descripción: Si cr es una consulta y c_1 y c_2 son nombres de campos, match(cr, c_1, c_2) es una nueva consulta que devuelve los registros de la consulta q tales que los campos c_1 y c_2 tienen el mismo valor.Los registros que no tengan el campo c_1 pero si c_2 o viceversa esta operacion no los devuelve.

PRODUCT(**in** cr_1 : `consulta`, **in** cr_2 : `consulta`) $\rightarrow$ res : `consulta`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{obs}$ PRODUCT(cr_1, cr_2)}
Descripción: Si cr_1 y cr_2 son consultas, product(cr_1, cr_2) es una nueva consulta que devuelve el producto cartesiano entre los registros de las consultas cr_1 y cr_2. Si durante el producto tenemos r1 un registro de cr_1, r2 un registro de cr_2 y tienen campos en comun esta operacion descarta r2 y devuelve r1 en el conjunto de salida.

Representación

Representación de Consulta

Consulta se representa con un diccionario sobre una estructura trie donde las claves son un string *nombreTabla* y como valor un diccionario cuyos valores son de tipo *estrReg*.

`consulta` **se representa con** `estrConsulta`

donde `estrConsulta` es `tupla(`*registros*: `diccString(estrReg))`

Rep : estrConsulta $\longrightarrow$ bool
Rep(c) $\equiv$ true

Abs : estrConsulta c $\longrightarrow$ consulta {Rep(c)}
Abs(c) $\equiv$ true

Algoritmos

---

iFROM(**in** *bdd*: `estrBdD`, **in** *nt*: `nombreTabla`) $\rightarrow$ *res* : estrConsulta
1: *res.registros* $\leftarrow$ bdd.obtener(nt).registros

Complejidad: O ($|t|$)
Justificación: El diccionario de registros se devuelve por referencia =0

---

iPROJ(**in** *consulta* : `estrConsulta`, **in** *conjNC* : `conjuntoLineal(nombreCampo)` → *res* : estrConsulta

```
1: res ← consulta
   /* Itero sobre las claves de los registros */
2: for ( nombreCampo& clave : consulta.registros.claves) do
       /* Itero sobre los campos de ese registro */
3:     for ( nombreCampo& campo : consulta.registros[clave].claves) do
4:         if ¬ conjNC.pertenece?(campo) then
5:             res.registros[clave].borrar(campo)
6:         end if
7:     end for
8: end for
```

Complejidad: O ($\#registros^2$. $\#campos$. $|c|$)
Justificación: Se recorre cada campo de cada registro en busca de los no coincidentes.
Borrar en un *trie* tiene costo O ($|c|$. $\#registros$)

---

iUNION(**in** *consulta*1 : `estrConsulta`, **in** *consulta*2 : `estrConsulta`) → *res* : estrConsulta

```
1: res ← consulta1
2: for ( nombreCampo& clave : consulta2.registros.claves) do
3:     res.registros.definir(clave, consulta2.registros[clave])
4: end for
```

Complejidad: O ($\#registros * copy(r)$)
Justificación: Se recorre cada registro r y se define en el diccionario *trie* *res.registros*.

---

iSELECT(**in** *consulta* : `estrConsulta`, **in** *nc* : `nombreCampo`, **in** *v* : `valor`) → *res* : estrConsulta

```
1: if consulta.registro.definido?(nc) then
2:     res.definir(v, consulta.registros[v])
3: else
4:     for ( nombreCampo& clave : consulta.registros.claves) do
5:         if consulta.registros[clave].campos[nc] = v then
6:             res.registros.definir(clave, consulta.registros[clave])
7:         end if
8:     end for
9: end if
```

Complejidad$^{(1)}$: O ($|c| + |v|$)
Justificación: Si el campo buscado es clave, es único y se obtiene a través de *tries*.

Complejidad$^{(2)}$: O ($|c| + n|v| + k(|v| + |c|)$
Justificación: Si el campo buscado no es clave, busca y compara el valor v del campo c con todos los n registros.

---

iINTER(**in** *consulta*1 : `estrConsulta`, **in** *consulta*2 : `estrConsulta`) → *res* : estrConsulta

```
1: for ( nombreCampo& clave1 : consulta1.registros.claves ) do
2:     if consulta2.registros.definido?(clave1) then
3:         res.registros.definir(clave1, consulta1.registros[clave1])
4:     end if
5: end for
```

Complejidad: O ($n.|c|$)
Justificación: Donde n es la cantidad de registros de la consulta con más registros, y |c| la cantidad de caracteres del campo más largo.

---

iRENAME(**in** *consulta* : `estrConsulta`, **in** *campo* : `nombreCampo`, **in** *nuevo_campo* : `nombreCampo`) → *res* : estrConsulta

```
1:  res ← consulta
2:  for ( nombreCampo& clave : consulta.registros.claves ) do
3:      if res.registros[clave].columnas.definido?(campo) then
4:          while res.registros[clave].columnas.definido?(nuevo_campo) do
5:              nuevo_campo ← (s • i • b • − • nuevo_campo)
6:          end while
7:          res.registros[clave].columnas.definir(nuevo_campo, res.registros[clave].columnas[campo])
8:          res.registros[clave].columnas.borrar(campo)
9:          if  campo = consulta.registros[clave].clave  then consulta.registros[clave].clave ← nuevo_campo
10:         end if
11:     end if
12: end for
```

Complejidad: $O\ (n.|c|)$
Justificación: n es la cantidad de registros de la consulta, y |c| la cantidad de caracteres del campo más largo.

iMATCH(**in** *consulta* : `estrConsulta`, **in** *campo*1 : `nombreCampo`, **in** *campo*2 : `nombreCampo`) → *res* : estrConsulta

```
1:  for ( nombreCampo& clave : consulta.registros.claves ) do
2:      if consulta.registros[clave].columnas.definido?(campo1) ∧
3:         consulta.registros[clave].columnas.definido?(campo2) then
4:          if consulta.registros[clave].columnas[campo1] =
5:             consulta.registros[clave].columnas[campo2] then
6:              res.registros.definir(clave, consulta.registros[clave])
7:          end if
8:      end if
9:  end for
```

Complejidad: $O\ (n.(|v| + |c1| + |c2|))$
Justificación: Para cada uno de los n registros con *clave* String de longitud |v|, se buscan los campos $campo1, campo2$ que coinciden con los pasados por parámetro. Se devuelve una nueva consulta donde solo quedan los registros con campos 1 y 2 iguales.

iPRODUCT(**in** *consulta*1 : `estrConsulta`, **in** *consulta*2 : `estrConsulta`) → *res* : estrConsulta

```
1:  if #consulta1.registros <#consulta2.registros then
2:      estrConsulta& min_consulta ← consulta1.registros
3:      estrConsulta& max_consulta ← consulta2.registros
4:  else
5:      estrConsulta& min_consulta ← consulta2.registros
6:      estrConsulta& max_consulta ← consulta1.registros
7:  end if
8:  for ( nombreCampo& clave1, estrReg& registro1 : min_consulta ) do
9:      for ( nombreCampo& clave2, estrReg& registro2 : max_consulta ) do
10:         res.registros.definir(clave1, registro1.concatenar(registro2))
11:     end for
12: end for
```

Complejidad: $O\ (n1.n2.|c|)$
Justificación: Se recorren por referencia las claves y registros $registro1, registro2$ de las consultas $consulta1, consulta2$ de entrada, de tamaño $n1, n2$ respectivamente. Por cada uno, se define un nuevo registro como la concatenación de campos de $registro1$ y $registro2$.

5. Módulo Tabla

Este modulo provee una tabla en la que se puede saber cuales son sus claves, registros y campos. Además se puede crear una tabla vacia a la que a su vez le podemos insertar registros, como tambien podemos borrarlos y borrar valores en sus campos.

Interfaz

se explica con: TAD TABLA

géneros: `tabla`.

Servicios Usados: Registro, DiccionarioString(α), Conjunto Lineal(α)

Operaciones básicas de tabla

NUEVATABLA(**in** *cs*: `conj(nombre_campo)`, **in** *c*: `nombre_campo`) $\rightarrow$ *res* : `tabla`
Pre $\equiv$ {*true*}
Post $\equiv$ {*res* $=_{\text{obs}}$ nueva(cs,c)}
Complejidad: O (1)
Descripción: Genera una tabla vacía, cuya clave es el campo c.

INSERTAR(**in/out** *t*: `tabla`, **in** *r*: `registro`)
Pre $\equiv$ {$t =_{\text{obs}} t_0 \ \wedge campos(t_0) =_{\text{obs}}$ campos(r) }
Post $\equiv$ {$t =_{\text{obs}}$ insertar(t_0, r)}
Complejidad: O ($|c| + copy(r)$)
Descripción: Inserta el registro r en la tabla t.

BORRAR(**in/out** *t*: `tabla`, **in** *v*: `valor`)
Pre $\equiv$ {$t =_{\text{obs}} t_0$}
Post $\equiv$ {$t =_{\text{obs}}$ borrar(t_0, v)}
Complejidad: O ($|t| \bullet \#c$)
Descripción: Borra el valor v de la tabla t.

CAMPOST(**in** *t*: `tabla`) $\rightarrow$ *res* : `conj(nombre_campo)`
Pre $\equiv$ {$True$}
Post $\equiv$ {*res* $=_{\text{obs}}$ campos(t)}
Complejidad: O (1)
Descripción: dado una tabla t devuelve un conjunto *res* de sus nombres de campo.

CLAVE(**in** *t*: `tabla`) $\rightarrow$ *res* : `nombre_campo`
Pre $\equiv$ {$True$}
Post $\equiv$ {*res* $=_{\text{obs}}$ clave(t)}
Complejidad: O (1)
Descripción: Dada una tabla t no devuelve cual es el nombre del campo que identifica sus claves.
Aliasing: *res* no es modificable

REGISTROS(**in** *t*: `tabla`) $\rightarrow$ *res* : `conj(registro)`
Pre $\equiv$ {$True$}
Post $\equiv$ {*res* $=_{\text{obs}}$ registros(t)}
Complejidad: O (1)
Descripción: Dada una tabla t devuelve un conjunto *res* con sus registros.

INSERTARREGISTRO(**in** *c*: `nombre_campo`, **in** *r*: `registro`, **in/out** *rs*: `conj(registro)`)
Pre $\equiv$ {rs $=_{\text{obs}} rs_0 \ \wedge \ c \in campos(r) \ \wedge (\forall r' : registro)(r' \in rs_0 \rightarrow c \in campos(r'))$}
Post $\equiv$ {$rs =_{\text{obs}}$ insertarRegistro(c, r, rs_0)}
Complejidad: O ($copy(r)$)
Descripción: Dado un registro r que contenga el campo c lo inserta en los registros de la tabla.

BORRARREGISTRO(**in** *c*: `nombre_campo`, **in** *v*: `valor`, **in/out** *rs*: `conj(registro)`)
Pre $\equiv$ {$rs =_{\text{obs}} rs_0 \ (\forall r : registro)(r \in rs_0 \rightarrow c \in campos(r))$}
Post $\equiv$ {$rs =_{\text{obs}}$ borrarRegistro(c, v, rs_0)}
Complejidad: O ($|c| + copy(c) + copy(v)$)

Descripción: Dado un registro r que contenga el valor v, borra uno de los registros de t.

Representación

Representación de la Tabla

La tabla se representa con su nombre unico con el que se le identifica .

`Tabla` **se representa con** `estrTabla`

donde `estrTabla` es `tupla`(*nombre*: `nombreTabla`, *clave*: `nombreCampo`, *registros*: `diccString(estrReg)`, *campos*: `conj(nombreCampo)`)

Rep : estrTabla $\longrightarrow$ bool

Rep(t) $\equiv$ true $\iff$
t.clave $\in$ $t.campos$ $\wedge$ $(\forall c : string)(def?(c, t.registros) \Rightarrow_L claves(obtener(c, t.registros)) =_{\text{obs}} t.campos)$

Abs : estrTabla t $\longrightarrow$ tabla {Rep(t)}

Abs(t) $\equiv$ $(\forall t' : tabla)/\ campos(t') = t.campos \wedge clave(t') = t.clave \wedge registros(t') = significados(t.registros)$

Algoritmos

---

iNuevaTabla(**in** cs : `nombreCampo`, **in** c : `nombreCampo`) $\rightarrow$ res : estrTabla

1: $res.nombre \leftarrow$ ""
2: $res.clave \leftarrow c$
3: $res.registros \leftarrow nuevodicc$
4: $res.campos \leftarrow cs$

Complejidad: O (1)
Justificación: O (1) + O (1) + O (1) + O (1) *es* O (1)

---

iInsertar(**in/out** t : `estrTabla`, **in** r : `estrReg`)

1: $t.registros.definir(r)$

Complejidad: O ($|c| + copy(r)$)
Justificación: De precondicon r no estaba definido en el diccionario, definir cuesta O ($|C| + Copy(r)$)

---

iBorrar(**in/out** t : `estrTabla`, **in** v : `valor`)

1: **for** string& c : t.registros.claves **do**
2: $t.registros.borrar(v)$
3: **end for**

Complejidad: O ($|t|\bullet\#c$)
Justificación: la operacion borrar cuesta O ($|s|\bullet\#c$) donde s es la clave mas larga y $\#c$ la cantidad de claves.

---

iCamposT(**in** t : `estrTabla`) $\rightarrow$ res : conj(nombreCampo)

1: $res \leftarrow t.campos$

Complejidad: O (1)
Justificación: la operacion campos de un diccionario cuesta O (1).

---

iclave(**in** t : `estrTabla`) → res : nombreCampo

1: $res \leftarrow t.clave$

Complejidad: O (1)
Justificación: El campo clave es parte de la representacion.

iRegistros(**in** t : `estrTabla`) → res : conj(estrReg)

1: $res \leftarrow t.registros.claves$

Complejidad: O (1)
Justificación: la operacion claves costaba O (1)

iInsertarRegistro(**in** c : `nombreCampo`, **in** r : `estrReg`, **in/out** rs : `conj(estrReg)`)

1: $rs.agregarRapido(r)$

Complejidad: O ($copy(r)$)

iBorrarRegistro(**in** c : `nombreCampo`, **in** v : `valor`, **in/out** rs : `conj(estrReg)`) → res : estrTabla

1: $r \leftarrow nuevoRegistro$
2: $r.definir(c, v)$
3: $rs.eliminar(r)$

Complejidad: O ($|c| + copy(c) + copy(v)$)
Justificacion: el costo de definir es de O ($|c| + copy(c) + copy(v)$)

6. Módulo Registro

Este modulo provee operaciones sobre registros.
Se puede definir y obtener valores.
Se puede crear un registro vacío.
Se puede concatenar registros y obtener nombres de sus campos.

Interfaz

se explica con: TAD Registro

géneros: `registro`.

servicios usados: diccionarioString(α)

Operaciones básicas de registro

NuevoRegistro() → res : `registro`
Pre ≡ {true}
Post ≡ {$res =_{obs}$ nuevo}
Complejidad: O (1)
Descripción: Genera un registro vacío.

Definir(**in/out** r : `registro`, **in** c : `nombre_campo`, **in** v : `valor`)
Pre ≡ {$r =_{obs} r_0$}
Post ≡ {$r =_{obs}$ definir(r_0, c, v)}
Complejidad: O $|c|$
Descripción: Define el registro r en el campo c con valor v.
Aliasing: los elementos c y v se definen por copia.

ObtenerValor(**in** r : `registro`, **in** c : `nombre_campo`) → res : `valor`
Pre ≡ {$c \in campos(r)$}

Post $\equiv \{res =_{\text{obs}} \text{r[c]}\}$
Complejidad: O $|c|$
Descripción: Obtiene el valor res del campo c en el registro r.
Aliasing: res es modificable si el registro r era modificable

CAMPOS(**in** r : `registro`) → res : `conj(nombre_campo)`
Pre $\equiv \{True\}$
Post $\equiv \{res =_{\text{obs}} \text{campos(r)}\}$
Complejidad: O (1)
Descripción: Dado un registro r devuelve un conjunto res de sus nombres de campo.
Aliasing: res no es modificable.

CONCATENAR(**in** r_1 : `registro`,**in** r_2 : `registro`) → res : `registro`
Pre $\equiv \{True\}$
Post $\equiv \{res =_{\text{obs}} \text{concatenar}(\text{r}_1, \text{r}_2)\}$
Complejidad: O $(|r_1 + \#r_2|)$
Descripción: Toma dos registros r_1, r_2 y los concatena en un único registro res. Si hay campos compartidos, se usarán los de r_1 y descartarán los de r_2.

Representación

Representación del registro

El registro se representa con su campo clave y un diccionario string con sus valores. El campo clave debe pertenecer a las claves del diccionario.

`registro` **se representa con** `estrReg`

donde `estrReg` es `tupla(`*clave*`: nombreCampo)`, *columnas*`: diccString(valor))`

Rep : estrReg $\longrightarrow$ bool
Rep(r) $\equiv$ true $\iff$
r.clave $\in$ *claves*(r.*columnas*)

Abs : estrReg r $\longrightarrow$ registro $\{\text{Rep}(r)\}$
Abs(r) $\equiv$ $(\forall r' : registro)/\ r.claves \in campos(r') \ \wedge \ claves(r.columnas) = campos(r') \ \wedge$
$(\forall c : nombreCampo)(c \in campos(r') \Rightarrow_L r'[c] =_{\text{obs}} obtener(c, r.columnas))$

Algoritmos

---

iNuevoRegistro() → res : estrReg
1: $res.clave \leftarrow$ ""
2: $res.registro \leftarrow nuevoDicc$

Complejidad: O (1)
Justificación: Se genera la representacion del registro vacio, que al ser parte de la representacion cuesta O (1).

---

iDefinir(**in/out** r : `estrReg`, **in** c : `nombreCampo`, **in** v : `valor`)
1: **if** r.registro.definido?(c) **then**
2: $r.registro[c] = v$
3: **else**
4: $r.registro.definir(c, v)$
5: **end if**

Complejidad: O $(|C| + copy(c) + copy(v))$
Justificaciön se debe revisar si la clave estä. Esto toma O $(|C|)$, luego se reemplaza en significado anterior por v. Si la clave no estä se debe definir junto con v.

---

iObtenerValor(**in** r : `estrReg`, **in** c : `nombreCampo`) → res : valor

1: $res \leftarrow r.registro.obtener(c)$

Complejidad: O ($|c|$)
Justificación: De precondicion sabemos que c pertenece y obtener un valor de un diccionario cuesta O ($|c|$) porque lo busca en el diccionario.

iCampos(**in** r : `estrReg`) → res : conj(nombreCampo)

1: $res \leftarrow r.registro.claves$

Complejidad: O (1)
justificación: obtener el conjunto de las claves de un diccionario cuesta O (1).

iConcatenar(**in** $r1$: `estrReg`, **in** $r2$: `estrReg`) → res : estrReg

1: $res \leftarrow r1$
2: **for** (nombreCampo& c1 : r2.registro.claves) **do**
3: **if** ¬ res.registro.definido?(c1) **then**
4: res.registro.definir(c1)
5: **end if**
6: **end for**

Complejidad: O ($|r_1| + \#r_2$)
justificación: copiamos r1 que cuesta $|r_1|$, al descartar los campos compartidos de r2 en el peor caso tenemos que definir $\#r_2$ elementos al diccionario.

7. Módulo DiccionarioString(α)

El módulo provee un diccionario que se representa con un trie, que permite lectura, inserción y modificación en O (|clave|) donde clave es string y es la clave a consultar o modificar. Las claves se guardan en un conjunto lineal y como podemos saber de antemano si pertenecia o no en el trie usa inserción rapida. Tambien permite borrar un elemento con costo O ($|s_{\max}|\bullet\#$c).
Asumimos que comparar 2 strings ∈ O ($|s_{\max}|$). Notacion: copy(e) es el costo de copiar el elemento e ∈ α ,$|s_{\max}|$es la longitud de la clave mas larga y #c es la cantidad de claves

Interfaz

parámetros formales
géneros α
función COPIAR(**in** e : α) → res : α
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}} e$}
Complejidad: Θ($copy(e)$)
Descripción: función de copia de α's

se explica con: DICCIONARIO(κ, σ)

géneros: `diccString`(α).

Operaciones básicas de diccionarioString(α)

NUEVODICC() → res : `diccString`(α)
Pre ≡ {true}
Post ≡ {$res =_{\text{obs}}$ vacio}
Complejidad: O (1)
Descripción: genera un diccionario vacío.

DEFINIR(**in/out** d : `diccString(`α`)`, **in** s : `string`, **in** e : α)
Pre ≡ {$d =_{\text{obs}} d_0$}

Post $\equiv \{d =_{\text{obs}} \text{definir}(s, e, d_0)\}$
Complejidad: O ($|s_{\max}|+$ copy(e))
Descripción: define la clave s con el significado e en el diccionario.
Aliasing: los elementos s y e se definen por copia.

DEFINIDO?(**in** d: `diccString(`α`)`, **in** s: `string`) $\rightarrow$ res : `bool`
Pre $\equiv$ {true}
Post $\equiv \{res =_{\text{obs}} \text{def?}(s, d)\}$
Complejidad: O ($|s_{\max}|$)
Descripción: devuelve `true` si y sólo s está definido en el diccionario.

OBTENER(**in** d: `diccString(`α`)`, **in** s: *string*) $\rightarrow$ res : α
Pre $\equiv \{\text{def?}(s, d)\}$
Post $\equiv \{res =_{\text{obs}} \text{obtener}(s, d))\}$
Complejidad: O ($|s_{\max}|$)
Descripción: devuelve el significado de la clave s en d.
Aliasing: res es modificable si y sólo si d es modificable.

BORRAR(**in/out** d: `diccString(`α`)`, **in** s: `string`)
Pre $\equiv \{d =_{\text{obs}} d_0 \wedge \text{def?}(s, d_0)\}$
Post $\equiv \{d =_{\text{obs}} \text{borrar}(d_0, k)\}$
Complejidad: O ($|s_{\max}|\bullet\#\text{c}$)
Descripción: elimina la clave s y su significado de d.

CLAVES(**in** d: `diccString(`α`)`) $\rightarrow$ res : `conj(string)`
Pre $\equiv$ {true}
Post $\equiv \{res =_{\text{obs}} \text{claves}(d)\}$
Complejidad: O (1)
Descripción: devuelve el conjunto de las claves del diccionario.
Aliasing: res no es modificable

SIGNIFICADOS(**in** d: `diccString(`α`)`) $\rightarrow$ res : `conj(`α`)`
Pre $\equiv$ {true}
Post $\equiv \{res =_{\text{obs}} \text{significados}(d)\}$
Complejidad: O (1)
Descripción: devuelve el conjunto de significados del diccionario.