DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2: Diseño

Primer cuatrimestre - 2016 Algoritmos y Estructuras de Datos II

Grupo XXXX

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

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 $\equiv$ $\{res =_{\text{obs}} cantidadDeAccesos(t)\}$
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($T * L + in$)

PUEDOINSERTAR?(**in** r : `registro`, *in* t : `tab`) $\longrightarrow$ *res* : `bool`
Pre $\equiv$ {$true$}
Post $\equiv$ $\{res =_{\text{obs}} puedoInsertar?(r, t)\}$
Descripción: Informa si el registro pasado por parametro no tiene valores repetidos con respectos a los registros existentes, para los campos clave en la tabla pasada por parametro.
Complejidad: O($T * L + in$)

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

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

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

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

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

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

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

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

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

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

1.2 Representación

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

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

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

1.3 Algoritmos

NOMBRE(**in** t : `tab`) $\longrightarrow$ res : `string`
$res \leftarrow t.nombre$ — O(1)
O(1)

CLAVES(**in** t : `tab`) $\longrightarrow$ res : `itConjTrie(campo)`
$res \leftarrow CrearItConjTrie(t.Campos.ClavesDicc)$ — O(1)
O(1)

INDICES(**in** t : `tab`) $\longrightarrow$ res : `itConjTrie(Indice)`
$res \leftarrow CrearItConjTrie(t.Indices)$ — O(1)
O(1)

CAMPOS(**in** t : `tab`) $\longrightarrow$ res : `itConjTrie(campo)`
$res \leftarrow CrearItConjTrie(t.Campos.ClavesDicc)$ — O(1)
O(1)

TIPOCAMPO(**in** c : `campo`, *in* t : `tab`) $\longrightarrow$ res : `Tipo`
$res \leftarrow Significado(t.Campos, c)$ — O(1)
O(1)

REGISTROS(**in** t : `tab`) $\longrightarrow$ res : `itConj(registro)`
$res \leftarrow CrearItConj(t.registros)$ — θ(L+ log(n))
θ(L+ log(n))

CANTDEACCESOS(**in** t : `tab`) $\longrightarrow$ res : `nat`
$res \leftarrow t.cantDeAccesos$ — $\theta(1)$
$\theta(1)$

PUEDOINSERTAR?(**in** r : `registro`, *in* t : `tab`) $\longrightarrow$ res : `bool`
res $\leftarrow$ campatible(r,t) $\wedge$ $\neg$hayCoincidencia(r,claves(r),registros(t)) — θ(L+log(n))
θ(L+log(n))

COMPATIBLE(**in** r : `registro`, *int* t : `tab`) $\longrightarrow$ res : `bool`
res $\leftarrow$ campatible(r,t) $\wedge_{\mathrm{L}}$ mismosTipos(r,t) — $\theta(1)$
O(1)

PUEDEINDEXAR(**in** c : `campo`, *in* t : `tab`) $\longrightarrow$ res : `bool`
res $\leftarrow$ Definido?(t.campos,c) $\wedge_{\mathrm{L}}\neg$ Esta?(c,t.Indices.IndicesL) $\wedge$
(long(t.Indices.IndicesL)=0 $\vee$ (long(t.Indices.IndicesL)<2 $\wedge$
(tipoCampo(c,t)!=tipoCampo(t.Indices.IndicesL.primero,t))))
$\theta(1)$
O(calcular)

COMBINARREGISTROS(**in** c : `campo`, *in cr1* : `Conj(registro)`, *in cr2* : `Conj(registro)`) $\longrightarrow$ res : `Conj(registros)`
res $\leftarrow$ vacio();
itcr1 $\leftarrow$ CrearItConjTrie(cr1)

while HaySiguiente(itcr1) **do**
AgregarRapido(res, combinarTodos(c,Siguiente(itcr1),cr2));
Avanzar(itcr1);

end while

O(calcular)

HAYCOINCIDENCIA(**in** *r* : `registro`, *in cc* : `ConjTrie(campo)`, *in cr* : `Conj(registro)`) $\longrightarrow$ *res* : `bool`
itcr $\leftarrow$ CrearItConj(cr);
res $\leftarrow$ false;

while HaySiguiente(itcr) **do**
res $\leftarrow$ coincideAlguno(r,cc,Siguiente(itcr)) $\vee$res;
Avanzar(itcr);

end while

O(calcular)

COINCIDENCIAS(**in** *crit* : `registro`, *in cr* : `Conj(registro)`) $\longrightarrow$ *res* : `itConj(registro)`
res $\leftarrow$ CrearItConj(vacio());

while HaySiguiente(cr) **do**

if coincidenTodos(crit,campos(crit),Siguiente(cr)) **then**
AgregarAtras(res,Siguiente)

end if
Avanzar(cr);

end while

O(calcular)

MINIMO(**in** *c* : `campo`, *in t* : `tab`) $\longrightarrow$ *res* : `dato`
res $\leftarrow$ min(dameColumna(c, t.registros);

O(calcular)

MAXIMO(**in** *c* : `campo`, *in t* : `tab`) $\longrightarrow$ *res* : `dato`
res $\leftarrow$ max(dameColumna(c, t.registros);

O(calcular)

DAMECOLUMNA(**in** *c* : `campo`, *in cr* : `Conj(registro)`) $\longrightarrow$ *res* : `itConj(dato)`
itcr $\leftarrow$ CrearItConj(cr);
res $\leftarrow$ CrearItConj(vacio());

while HaySiguiente(itcr) **do**

if Definido?(Siguiente(itcr),c) **then**
AgregarAtras(res,Obtener(Siguiente(itcr), c)) ;

end if
Avanzar(itcr);

end while

O(calcular)

MISMOSTIPOS(**in** r : `registro`, *in* t : `tab`) $\longrightarrow$ *res* : `bool`
res $\leftarrow$ True;
itconjClaves $\leftarrow$ CrearItConj(claves(r));

while HaySiguiente(itconjClaves) **do**
res $\leftarrow$ res$\wedge$(tipo?(Obtener(r,Siguiente(itconjClaves))) = tipoCampo(Siguiente(itconjClaves)),t)
;
Avanzar(cr);

end while

O(calcular)

1.4 Algoritmos operaciones auxiliares

AGREGARESTUDIANTE(**in/out** *campus* : `campusSeguro`, *in pos* : `pos`, *in nombre* : `nombre`)
$campus.campus[pos.x][pos.y].hayEst \leftarrow True$ O(1)
$campus.campus[pos.x][pos.y].estudiante \leftarrow definir(campus.estudiantes, nombre, pos)$ O(long(nombre))

O(long(nombre))

AGREGARHIPPIE(**in/out** *campus* : `campusSeguro`, *in pos* : `pos`, *in nombre* : `nombre`)
$campus.campus[pos.x][pos.y].hayHippie \leftarrow True$ O(1)
$campus.campus[pos.x][pos.y].hippie \leftarrow definir(campus.hippies, nombre, pos)$ O(long(nombre))

O(long(nombre))

SANCIONARAGENTESVECINOS(**in/out** *campus* : `campusSeguro`, *in pos* : `pos`)
$vecinos \leftarrow campus.campusEstatico.vecinos(pos)$ O(1)
if $campus.atrapadoPorAgente?(pos)$ **then**
while $i < vecinos.tamanio()$ **do** O(1)
if $campus.campus[vecinos[i].x][vecinos[i].y].hayAgente?$ **then**
$campus.sancionarAgente(vecinos[i].agente)$ O(1)
end if
$i++$
end while
end if

O(1)

SANCIONARAGENTESENCERRANDOESTVECINOS(**in/out** *campus* : `campusSeguro`, *in pos* : `pos`)
$vecinos \leftarrow campus.campusEstatico.vecinos(pos)$ O(1)
$i \leftarrow 0$
while $i < vecinos.tamanio$ **do** O(1)
if $campus.campus[vecinos[i].x][vecinos[i].y].hayEst \wedge atrapadoPorAgente?(campus, pos)$
then O(1)

sancionarAgentesVecinos(*campus*, *pos*) O(1)
end if
$i++$
end while

O(1)

SANCIONARAGENTE(**in/out** *campus* : `campusSeguro`, *in/out agente* : `itDiccRapido`)
campus.conKSanciones.ocurrioSancion ← *True* O(1)
agente.siguiente.cantSanciones + 1 O(1)
agente.siguiente.miUbicacion.eliminarSiguiente() O(1)
// El iterador mismas apunta a la posicion correspondiente del agente dentro de la lista ordenada por cantSanciones
// Como la lista en el peor caso puede contener a todos los agentes con igual cant de sanciones
// la mayor cantidad posible de iteraciones del ciclo es 4
while *agente.siguiente.mismcampus.haySiguiente*()
∧ *agente.siguiente.mismas.siguiente.cantSanciones* < *agente.siguiente.cantSanciones* **do**
agente.siguiente.mismas.avanzar() O(1)
end while
// Si no hay siguiente o si la cantidad de sanciones del siguiente es menor que la del agente, entonces,
// creo un conMismasBucket, lo inserto como siguiente y me guardo el iterador en miUbicacion
// Sino, agrego el agente al conj de agentes del siguiente y me guardo el iterador en miUbicacion
if ¬(*agente.siguiente.mismas.haySiguiente*) ∨
(*agente.siguiente.mismas.haySiguiente*∧
agente.siguiente.cantSanciones = *agente.siguiente.mismas.cantSanciones*) **then**
O(1)
nConMismasB ← *nuevaTupla*(*CrearNuevoDiccLineal*(), *agente.siguiente.cantSanciones*)
agente.siguiente.mismas ← *agente.siguiente.mismas.agregarComoSiguiente*(*nConMismasB*)
O(1)
agente.siguiente.miUbicacion ←
agente.siguiente.mismas.siguiente.agentes.agregarComoSiguiente(*agente.siguiente.pl*)
O(1)
else
agente.siguiente.mismas.siguiente.agentes.agregarComoSiguiente(*agente.siguiente.pl*)
O(1)
end if

O(1)

ATRAPADOPORAGENTE?(**in** *campus* : `campusSeguro`, *in pos* : `pos`) ⟶ *res* : `bool`
vecinos ← *campus.campusEstatico.vecinos*(*pos*)
*alMenos*1*Agente* ← *False* O(1)
$i \leftarrow 0$
if ¬(*encerrado*?(*pos*, *campus.campusEstatico.vecinos*(*pos*))) **then**
return false
end if
// Veo si hay algun agente alrededor
while $i <$ *vecinos.tamanio*() **do** O(1)
if *as.campus*[*vecinos*[*i*].*x*][*vecinos*[*i*].*y*].*hayAgente*? **then**
return true
end if
$i++$ O(1)
end while

	O(1)

HIPPIFICARESTUDIANTESVECINOS(**in/out** *campus* : campusSeguro, *in pos* : pos)

$vecinos \leftarrow campus.campusEstatico.vecinos(pos)$	O(1)
$i \leftarrow 0$	O(1)
while $i < vecinos.tamanio()$ **do**	O(long(nombre))
if $estAHippie?(campus, vecinos[i])$ **then**	
$hippificar(campus, vecinos[i])$	O(long(nombre))
end if	
$i++$	O(1)
end while	
	O(long(nombre))

HIPPIFICAR(**in/out** *campus* : campusSeguro, *in pos* : pos)

// PRE: La posicion esta en el tablero y hay estudiante en la posicion	
$as.campus[pos.x][pos.y].hayHippie \leftarrow True$	O(1)
$as.campus[pos.x][pos.y].hippie.agregarComoSiguiente(nombre, pos)$	O(long(nombreEstudiante))
$as.campus[pos.x][pos.y].hayEst \leftarrow False$	O(1)
$as.campus[pos.x][pos.y].estudiante.eliminarSiguiente()$	O(long(nombreEstudiante))
	O(long(nombre))

ESTAHIPPIE?(**in** *campus* : campusSeguro, *in pos* : pos) $\longrightarrow$ *res* : bool

if $\neg(encerrado?(pos, vecinos))$ **then**	
$return\ false$	O(1)
end if	
$i \leftarrow 0$	O(1)
$cantHippies \leftarrow 0$	O(1)
$vecinos \leftarrow campus.campusEstatico.vecinos(pos)$	O(1)
while $i < vecinos.tamanio()$ **do**	
if $campus[vecinos[i].x][vecinos[i].y].hayHippie$ **then**	
$cantHippies++$	O(1)
end if	
$i++$	
end while	
$return\ cantHippies \geq 2$	O(1)
	O(1)

HIPPIEAEST?(**in** *campus* : campusSeguro, *in pos* : pos) $\longrightarrow$ *res* : bool

$i \leftarrow 0$	O(1)
$vecinos \leftarrow campus.campusEstatico.vecinos(pos)$	O(1)
while $i < vecinos.tamanio()$ **do**	O(1)
if $\neg(as.campus[vecinos[i].x][vecinos[i].y].hayEst?)$ **then**	
$return\ False$	O(1)
end if	
end while	
$return\ True$	
	O(1)

ENCERRADO?(**in** *campus* : campusSeguro, *in pos* : pos)

$vecinos \leftarrow vecinos(as.campusEstatico, pos)$	O(1)

```
i ← vecinos.tamanio()                                              O(1)
while i < vecinos.tamanio() do                                     O(1)
    if ¬(campus.campus[vecinos[i].x][vecinos[i].y].hayAgente? ∨
campus.campus[vecinos[i].x][vecinos[i].y].hayEst? ∨
campus.campus[vecinos[i].x][vecinos[i].y].hayHippie? ∨
campus.campus[vecinos[i].x][vecinos[i].y].hayObst?) then           O(1)
        return false                                               O(1)
    end if
    i++                                                            O(1)
end while
returntrue
                                                                   ---
                                                                   O(1)
```

APLICARHIPPIESVECINOS(**in/out** *campus* : `campusSeguro`, *in pos* : `pos`)

```
vecinos ← campus.campusEstatico.vecinos(pos)                       O(1)
i ← 0                                                              O(1)
while i < vecinos.tamanio() do                                     O(long(nombre))
    aplicarHippie(campus, pos)                                     O(long(nombre))
end while
                                                                   ---
                                                                   O(long(nombre))
```

APLICARHIPPIE(**in/out** *campus* : `campusSeguro`, *in pos* : `pos`)

```
// PRE: pos valida y hayHippie en campus.campus[pos.x][pos.y]
if campus.campus[pos.x][pos.y].hayHippie then
    if as.hippieAEst(pos) then                                     O(1)
        campus : campusSeguro.campus[pos.x][pos.y].hayHippie ← False
                                                                   O(1)
        campus : campusSeguro.campus[pos.x][pos.y].hayEst ← True
                                                                   O(1)
        as.campus[pos.x][pos.y].estudiante ← CrearIt(campus.hippies)
                                                                   O(1)
        campus.campus[pos.x][pos.y].estudiante.
        agregarComoSiguiente(campus.campus[pos.x][pos.y].estudiante.nombre)
                                                                   O(long(nombre))
        campus.campus[pos.x][pos.y].hippie.eliminarSiguiente()
                                                                   O(long(nombre))
    else
        if campus.campus[pos.x][pos.y].hayHippie? ∧ atrapadoPorAgente(pos) then
            vecinos ← campus.campusSeguro.vecinos(pos)             O(1)
            i ← 0                                                  O(1)
            while i < vecinos.tamanio() do                         O(1)
                posAct ← vecinos[pos.x][pos.y]                     O(1)
                info ← campus.campus[vecinos[i].x][vecinos[i].y]
                                                                   O(1)
                if posAct.hayAgente then
                    info.agente.siguiente.cantCapturas++           O(1)
                    // Actualizar mas vigilante
                    if campus.masVigilante.siguienteSignificado().cantCapturas <
info.agente.siguienteSignificado().cantCapturas then
                        campus.masVigilante ← info.agente          O(1)
                    else
                        if campus.masVigilante.siguienteSignificado().cantCapturas =
```

$info.agente.siguienteSignificado().cantCapturas$
$\wedge campus.masVigilante.siguienteClave() < info.agente.siguienteClave()$ **then**
O(1)

$campus.masVigilante \leftarrow info.agente$
O(1)

end if

end if

end if

$i++$

end while

$campus.campus[pos.x][pos.y].hayHippie? = False$
O(1)

$campus.campus[pos.x][pos.y].hippie.eliminarSiguiente()$
O(long(nombre))

end if

end if

end if

O(long(nombre))

PROXPOSHIPPIE(**in/out** *campus* : `campusSeguro`, *in nombre* : `string`) $\longrightarrow$ *res* : `pos`

// PRE: El nombre es un hippie y el hippie no esta encerrado

$posHippie \leftarrow campus.hippies.obtener(nombre)$ O(long(nombre))

if $campus.estudiantes.tamanio() > 0$ **then**

// Retorna de las posiciones mas cercanas, la que esta mas cerca del (0,0)

$proxPos \leftarrow aPosMasCercana(campus.campusEstatico, posHippie, campus.estudiantes.significados)$
O(N_e)

else

// Retorna el ingreso mas cercan, en caso de empate, el de abajo

$proxPos \leftarrow aIngresoMasCercano(campus.campusEstatico, posHippie)$
$O(1)$

end if

$res \leftarrow proxPos$ O(1)

O(N_e)

PROXPOSAGENTE(**in/out** *campus* : `campusSeguro`, *in posAgente* : `pos`) $\longrightarrow$ *res* : `pos`

// PRE: En la posicion hay un agente que se puede mover

if $campus.hippies.tamanio() > 0$ **then**

// Retorna de las posiciones mas cercanas, la que esta mas cerca del (0,0)

$proxPos \leftarrow aPosMasCercana(campus.campusEstatico, posAgente, campus.hippies.significados)$
O(N_h)

else

// Retorna el ingreso mas cercano, en caso de empate, el de abajo

$proxPos \leftarrow aIngresoMasCercano(campus.campusEstatico, posAgente)$
$O(1)$

end if

$res \leftarrow proxPos$ O(1)

O(N_h)

AINGRESOMASCERCANO(**in** p : `pos`, cs : `campusSeguro`) $\longrightarrow$ *res* : `pos`

if $p.Y \leq c.alto/2$ **then**

if $PosValida(cs.campus, < p.X, p.Y - 1 >) \wedge \neg HayAlgo(cs, < p.X, p.Y - 1 >)$ **then**

$res \leftarrow < p.X, p.Y - 1 >$

```
    else
        if PosValida/c, < p.X + 1, p.Y >) ∧ ¬HayAlgo(c, < p.X + 1, p.Y >) then
            res ←< p.X + 1, p.Y >
        else
            if PosValida/c, < p.X − 1, p.Y >) ∧ ¬HayAlgo(c, < p.X − 1, p.Y >) then
                res ←< p.X − 1, p.Y >
            else
                res ←< p.X, p.Y + 1 >
            end if
        end if
    end if
else
    if PosValida(cs.campus, < p.X, p.Y + 1 >) ∧ ¬HayAlgo(cs, < p.X, p.Y + 1 >) then
        res ←< p.X, p.Y − 1 >
    else
        if PosValida/c, < p.X + 1, p.Y >) ∧ ¬HayAlgo(c, < p.X + 1, p.Y >) then
            res ←< p.X + 1, p.Y >
        else
            if PosValida/c, < p.X − 1, p.Y >) ∧ ¬HayAlgo(c, < p.X − 1, p.Y >) then
                res ←< p.X − 1, p.Y >
            else
                res ←< p.X, p.Y − 1 >
            end if
        end if
    end if
end if
```

O(1)

IBUSQUEDABINARIAPORSANCIONES(**in** ar : `arreglo(val:nat otr: α>)`, *in* $sanc$: `nat`) $\longrightarrow$ res : $< \alpha, bool >$)

```
res.π2 ← false
min ← 0
max ← |ar|
while max − min > 1 do                                   O(log(|ar|))
    med ← (max − min)/2
    if ar[med].val ≤ sanc then
        min ← med
    else
        max ← med
    end if
end while
if ar[min].val = sanc then
    res ←< ar[min].otr, true >
end if
```

O(log(|ar|))

2 Tipo es Bool

3 Dato(α)

3.1 Interfaz

se explica con DATO

usa

géneros `nat`, `string`, `tipo`

Operaciones

TIPO?(**in** d : `dato`) $\longrightarrow$ res : `tipo`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} tipo?(d)\}$
Descripción: Devuelve el tipo del dato ingresado por parametro.
Complejidad: O(1)
Aliasing: Se retorna res por referencia.

VALORNAT(**in** d : `dato`) $\longrightarrow$ res : `nat`
Pre $\equiv$ {Nat?(d)}
Post $\equiv$ $\{res =_{\text{obs}} valorNat(t)\}$
Descripción: Devuelve valor numerido del dato por parametro.
Complejidad: O(1)
Aliasing: Se devuelve res por referencia.

VALORSTRING(**in** d : `dato`) $\longrightarrow$ res : `string`
Pre $\equiv$ {String?(d)}
Post $\equiv$ $\{res =_{\text{obs}} valorString(t)\}$
Descripción: Devuelve valor del dato por parametro.
Complejidad: O(1)
Aliasing: Se devuelve res por referencia.

DATONAT(**in** n : α, **in** $tipoDelDato$: `tipo`) $\longrightarrow$ res : `dato`
Pre $\equiv$ $\{tipoDelDato\}$
Post $\equiv$ $\{res =_{\text{obs}} datoNat(n, tipoDelDato)\}$
Descripción: Crea un dato de valor numerico.
Complejidad: O(1)
Aliasing: Se devuelve res por referencia.

DATOSTR(**in** n : α, **in** $tipoDelDato$: `tipo`) $\longrightarrow$ res : `dato`
Pre $\equiv$ $\{\neg tipoDelDato\}$
Post $\equiv$ $\{res =_{\text{obs}} datoString(n, tipoDelDato)\}$
Descripción: Crea un dato de valor de letras.
Complejidad: O(1)
Aliasing: Se devuelve res por referencia.

MISMOTIPO?(**in** $d1$: `dato`, *in* $d2$: `dato`) $\longrightarrow$ res : `bool`
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} mismoTipo?(d1, d2)\}$
Descripción: Informa si los datos pasados por parametro son del mismo tipo de valor.
Complejidad: O(1)

STRING?(**in** d : `dato`) $\longrightarrow$ res : `bool`
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} String?(d)\}$
Descripción: Informa si el dato pasado por parametro es de tipo string.
Complejidad: O(1)

NAT?(**in** d : `dato`) $\longrightarrow$ res : `bool`
Pre $\equiv$ $\{true\}$
Post $\equiv$ $\{res =_{\text{obs}} Nat?(d)\}$
Descripción: Informa si el dato pasado por parametro es de tipo nat.
Complejidad: O(1)

MIN(**in** cd : `Conj(dato)`) $\longrightarrow$ res : `dato`
Pre $\equiv$ {¬EsVacio?(cd)}
Post $\equiv$ $\{res =_{\text{obs}} min(cd)\}$
Descripción: Retorna el minimo entre los valores del conjunto de datos pasado por parametro.
Complejidad: O($Cardinal(cd)$)
Aliasing: Retorna res por referencia.

MAX(**in** cd : `Conj(dato)`) $\longrightarrow$ res : `dato`
Pre $\equiv$ {¬EsVacio?(cd)}
Post $\equiv$ $\{res =_{\text{obs}} max(cd)\}$
Descripción: Retorna el maximo entre los valores del conjunto de datos pasado por parametro.
Complejidad: O($Cardinal(cd)$)
Aliasing: Retorna res por referencia.

<=(**in** *d1* : `dato`, *in d2* : `dato`) $\longrightarrow$ res : `bool`
Pre $\equiv$ {mismoTipo?(d1,d2)}
Post $\equiv$ $\{res =_{\text{obs}} <= (d1, d2)\}$
Descripción: Retorna el maximo entre los valores del conjunto de datos pasado por parametro.
Complejidad: O($Cardinal(cd)$)
Aliasing: Retorna res por referencia. oincidenTodos(crit,campos(crit),Siguiente(cr))

3.2 Representación

se representa con `datotupla`⟨Valor : α,
TipoValor : `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

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

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

3.3 Algoritmos

TIPO?(**in** a : `dato`) $\longrightarrow$ res : `bool`
$res \leftarrow a.TipoValor$ — O(1)

O(1)

VALORNAT(**in** a : `dato`) $\longrightarrow$ res : `nat`
$res \leftarrow a.Valor$ — O(1)

O(1)

VALORSTR(**in** a : `dato`) $\longrightarrow$ res : `string`
$res \leftarrow a.Valor$ — O(1)

O(1)

MISMOTIPO?(**in** $d1$: `dato`, in $d2$: `dato`) $\longrightarrow$ res : `bool`
$res \leftarrow tipo?(d1) = tipo?(d2)$ — O(1)

O(1)

NAT?(**in** a : `dato`) $\longrightarrow$ res : `bool`
$res \leftarrow tipo?(a)$ — O(1)

O(1)

STRING?(**in** a : `dato`) $\longrightarrow$ res : `bool`
$res \leftarrow \neg Nat?(a)$ — O(1)

O(1)

MIN(**in** cd : `Conj(dato)`) $\longrightarrow$ res : `dato`

```
itcd ← CrearItConj(cd)
minimo ← Siguiente(itcd)

while HaySiguiente(itcd) do

    if Siguiente(itcd)¡=minimo then
        minimo ← Siguiente(itcd);

    end if
    Avanzar(itcr);

end while
```

O(Cardinal(cd))

MAX(**in** cd : `Conj(dato)`) $\longrightarrow$ res : `dato`

```
itcd ← CrearItConj(cd)
maximo ← Siguiente(itcd)

while HaySiguiente(itcd) do

    if maximo¡=Siguiente(itcd) then
        minimo ← Siguiente(itcd);

    end if
```

Avanzar(itcr);

end while

O(Cardinal(cd))

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

if String?(d1) **then**
res $\leftarrow$ valorStr(d1)¡=valorStr(d2)

else
res $\leftarrow$ valorNat(d1)¡=valorNat(d2)

end if

O(1)

3.4 Algoritmos operaciones auxiliares

4 Diccionario por Naturales

4.1 Interfaz

se explica con DICCIONARIO(κ, σ)

usa Bool, Nat

géneros `diccNat(`κ, σ`)`

Operaciones

VACIO() $\longrightarrow$ *res* : `diccNat(`κ, σ`)`
Pre $\equiv \{true\}$
Post $\equiv \{res =_{\text{obs}} vacio()\}$
Descripción: Crea un nuevo diccionario
Complejidad: $O(1)$

DEFINIDO?(**in** d : `diccNat(`κ, σ`)` *in* $n : \kappa$) $\longrightarrow$ *res* : `Bool`
Pre $\equiv \{true\}$
Post $\equiv \{res =_{\text{obs}} def?(n, d)\}$
Descripción: Indica si la clave esta definida
Complejidad: $O(m)$

DEFINIR(**in/out** d : `diccNat(`κ, σ`)` *in* $n : \kappa$, *in* $s : \sigma$)
Pre $\equiv \{\neg def?(n, d) \land d = d_0\}$
Post $\equiv \{d =_{\text{obs}} Definir(n, s, d_0)\}$
Descripción: Se define s en el diccionario
Complejidad: $O(m)$

BORRAR(**in/out** d : `diccNat(`κ, σ`)` *in* $n : \kappa$)
Pre $\equiv \{d =_{\text{obs}} d_0 \land def?(n, d)\}$
Post $\equiv \{d =_{\text{obs}} Borrar(n, d_0)\}$

Descripción: Elimina el elemento n
Complejidad: $O(m)$

SIGNIFICADO(**in** d : `diccNat(`κ, σ`)` *in* $n : \kappa$) $\longrightarrow$ $res : \sigma$
Pre $\equiv \{def?(n, d)\}$
Post $\equiv \{res =_{\text{obs}} obtener(n, d)\}$
Descripción: Se retornan los significados
Complejidad: $O(m)$

4.2 Representación

diccNat

se representa con `puntero(estr(`κ, σ`))`

donde `estr(`κ, σ`)` **es** `tupla`$\langle$clave : κ,
significado : σ,
hijoDer : `puntero(estr(`$\kappa\sigma$`))`,
hijoIzq : `puntero(estr(`$\kappa\sigma$`))`$\rangle$

Invariante de representación

Rep : $\widehat{\texttt{estr}} \longrightarrow$ *boolean*

($\forall e : \widehat{\texttt{estr}}$)
Rep(e) $\equiv$

1. Para todo hijoDer de un estr, si no es NULL, su clave es mayor a la clave de su padre.
2. Para todo hijoIzq de un estr, si no es NULL, su clave es menor a la clave de su padre.
3. No hay ciclos, ni nodos con dos padres.

Función de abstracción

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

($\forall d : \widehat{\texttt{diccNat(}\kappa\texttt{, }\sigma\texttt{)}}$)
Abs(d) $\equiv c : \widehat{\texttt{dicc(}\kappa\texttt{, }\sigma\texttt{)}} \mid ((\forall k : \kappa)(k \in claves(c) \Rightarrow (\exists n : estr(\kappa, \sigma))(n \in arbol(d) \wedge n.clave = k)) \wedge ((\forall n : \text{estr}(\kappa, \sigma))(n \in arbol(d) \Rightarrow n.clave \in claves(c))))\wedge_{\text{L}}$
$((\forall n : \text{estr}(\kappa, \sigma))(n \in arbol(d) \Rightarrow obtener(c, n.clave) =_{\text{obs}} n.significado)$

$arbol : puntero(estr(\kappa, \sigma)) \leftarrow conj(puntero(estr(\kappa, \sigma)))$

$arbol(n) \equiv$
if $n.hijoIzq \neq null \wedge n.hijoDer \neq null$ **then**
  $Ag(n, arbol(n.hijoIzq) \cup arbol(n.hijoDer))$
else
  if $n.hijoIzq \neq null$ **then**
    $Ag(n, arbol(n.hijoIzq))$
  else
    if $n.hijoDer \neq null$ **then**
      $Ag(n, arbol(n.hijoDer))$
    else

$Ag(n, \emptyset)$
end if
end if
end if

4.3 Algoritmos

iVacio() $\longrightarrow$ res : `estr`
$res \leftarrow NULL$ — O(1)

iDefinir(**in/out** d : `estr`, $in\ n : \kappa$, $in\ s : \sigma$)
if $d == NULL$ **then**
$res \leftarrow < n, s, NULL, NULL >$ — O(1)
end if
if $d \neq NULL \wedge n < d.clave$ **then**
$d.hijoIzq \leftarrow iDefinir(d.hijoIzq, n, s)$ — O(log(m))
end if
if $d \neq NULL \wedge n > d.clave$ **then**
$d.hijoDer \leftarrow iDefinir(d.hijoDer, n, s)$ — O(m)
end if
— O(m)

iEliminar(**in/out** d : `estr`, $in\ n : \kappa$)
if $d == NULL$ **then**
$FinFuncion$ — O(1)
else if $n > d.clave$ **then**
$d.hijoDer \leftarrow iBorrar(d.hijoDer, n)$ — O(m)
else if $n < d.clave$ **then**
$d.hijoIzq \leftarrow iBorrar(d.hijoIzq, n)$ — O(m)
else if $d.hijoIzq == NULL \wedge d.hijoDer == NULL$ **then**
$Borrar(d)$ — O(1)
$d \leftarrow NULL$ — O(1)
else if $d.hijoIzq == NULL$ **then**
$aux \leftarrow d.hijoDer$ — O(1)
while $aux.hijoIzq \neq NULL$ **do** — O(m)
$aux \leftarrow aux.hijoIzq$ — O(1)
end while
$d.hijoDer \leftarrow iBorrar(aux.clave, d.hijoDer)$
$d.clave \leftarrow aux.clave$ — O(1)
$d.significado \leftarrow aux.significado$ — O(1)
else
$aux \leftarrow d.hijoIzq$ — O(1)
while $aux.hijoDer \neq NULL$ **do** — O(m)
$aux \leftarrow aux.hijoDer$ — O(1)
end while
$d.hijoIzq \leftarrow iBorrar(aux.clave, d.hijoIzq)$
$d.clave \leftarrow aux.clave$ — O(1)
$d.significado \leftarrow aux.significado$ — O(1)
end if
— O(m)

iSignificado(**in/out** d : `estr`, $in\ n : \kappa$) $\longrightarrow$ $res : \sigma$

$nodoActual \leftarrow d$	O(1)
while $\neg(nodoActual == NULL) \wedge \neg res$ **do**	O(m)
if $nodoActual.clave == n$ **then**	
$res \leftarrow nodoActual.significado$	O(1)
else	
if $c < nodoActual.clave$ **then**	
$nodoActual \leftarrow nodoActual.hijoIzq$	O(1)
else	
$nodoActual \leftarrow nodoActual.hijoDer$	O(1)
end if	
end if	
end while	
	O(m)

iDefinido?(**in/out** d : `estr`, $in\ n : \kappa$) $\longrightarrow$ res : `bool`

$nodoActual \leftarrow d$	O(1)
$res \leftarrow FALSE$	O(1)
while $\neg(nodoActual == NULL) \wedge \neg res$ **do**	O(m)
if $nodoActual.clave == n$ **then**	
$res \leftarrow TRUE$	O(1)
else	
if $c < nodoActual.clave$ **then**	
$nodoActual \leftarrow nodoActual.hijoIzq$	O(1)
else	
$nodoActual \leftarrow nodoActual.hijoDer$	O(1)
end if	
end if	
end while	
	O(m)

m: En peor caso es igual a la cantidad de elementos del arbol. En promedio es log(cantidad de elementos del arbol).

5 Registro

5.1 Interfaz

se explica con REGISTRO

usa

géneros `dicc(campo dato)`, `itDicc(campo dato)`

Operaciones

CAMPOS(**in** r : `dicc(campo dato)`) $\longrightarrow$ res : `itConj(campo)`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} claves(r)\}$
Descripción: Devuelve un conjunto de campos que son claves del registro ingresado por parametro

Complejidad: O($calc$)
Aliasing: Se devuelve un iterador al conjunto, hay aliasing

BORRAR?(**in** *crit* : `dicc(campo dato)`, *in r* : `dicc(campo dato)`) $\longrightarrow$ *res* : `bool`
Pre $\equiv$ {#campos(crit) = 1}
Post $\equiv$ {$res =_{\text{obs}} borrar?(crit, r)$}
Descripción: Devuelve true si y solo si todos los campos de crit pertenecen a campos de r.
Complejidad: O($calc$)

AGCAMPOS(**in/out** *r1* : `dicc(campo dato) in r2: dicc(campo dato)`) $\longrightarrow$ *res* : `reg`
Pre $\equiv$ {$r1 =_{\text{obs}} r1\_0$}
Post $\equiv$ {$r1 =_{\text{obs}} agregarCampos(r1\_0, r2)$}
Descripción: agrega los datos de los campos faltantes de r2 a los campos de r1
Complejidad: O($calcular$)

COPIARCAMPOS(**in/out** *r_1* : `dicc(campo dato)`, *in cc* : `conj(campo)`, *in r2* : `reg`)
Pre $\equiv$ {$r1 =_{\text{obs}} r1\_0 \wedge cc \in campos(r2)$}
Post $\equiv$ {$r1 =_{\text{obs}} copiarCampos(cc, r1\_0, r2)$}
Descripción: copia los datos que se encuentran en los campos de r2 a r1
Complejidad: O($calc$)

COINCIDEALGUNO(**in** *r1* : `dicc(campo dato)`, *in cc* : `conj(campo)`, *in r2* : `dicc(campo dato)`)
$\longrightarrow$ *res* : `bool`
Pre $\equiv$ {cc $\subseteq$ $campos(r1) \cap$ campos(r2)}
Post $\equiv$ {$res =_{\text{obs}} coincideAlguno(r1, r2)$}
Descripción: Devuelve true si y solo si alguno de los campos(dato) de cc pertenece a r1 y r2
Complejidad: O($calc$)

COINCIDENTODOS(**in** *r1* : `dicc(campo dato)`, *in cc* : `conj(campo)`, *in r2* : `dicc(campo dato)`)
$\longrightarrow$ *res* : `bool`
Pre $\equiv$ {cc $\subseteq$ campos(r1)$\cap$ campos(r2)}
Post $\equiv$ {$res =_{\text{obs}}$}
Descripción: Devuelve ture si y solo si todos los campos(dato) de cc pertenecen a r1 y r2
Complejidad: O($calc$)

ENTODOS(**in** *c* : `campo in cr: conj(registro)`) $\longrightarrow$ *res* : `bool`
Pre $\equiv$ {true}
Post $\equiv$ {$res =_{\text{obs}} enTodos(c, cr)$}
Descripción: Devuelve true si y solo si campo c pertenece a los campos de cada uno de los registros cr
Complejidad: O($calc$)

COMBINARTODOS(**in** *c* : `campo in r1:dicc(campo dato)`, *in cr* : `conj(dicc(campo dato))`)
$\longrightarrow$ *res* : `conj(itDicc(campo dato))`
Pre $\equiv$ {c $\in$ campos(r1) $\wedge$ enTodos(c,cr)}
Post $\equiv$ {$res =_{\text{obs}} cantidadDeAccesos(t)$}
Descripción: Devuelve.
Complejidad: O(1)

5.2 Representación

se representa con `Registro`
donde `dicc(campo dato)` **es**

5.3 Algoritmos

CAMPOS(**in** r : `dicc(campo dato)`) $\longrightarrow$ *res* : `conj(campo)`
  $res \leftarrow CrearItConjTrie(r.ClavesDicc)$ — O(1)

O(1)

AGREGARCAMPOS(**in** *r1* : `dicc(campo dato) in r2: reg`) $\longrightarrow$ *res* : `reg`
  res ← CrearItConj(vacio());

O(calcular)

COPIARCAMPOS(**in** *cc* : `conj(campo) in/out r1: dicc(campo dato) in r2: reg`)
  res ← min(dameColumna(c, CrearItConj(t.registros)));

O(calcular)

COINCIDEALGUNO(**in** *r1* : `dicc(campo dato) in cc: conj(campo)`, *in r2* : `reg`) $\longrightarrow$ *res* : `bool`
  res ←

O(calcular)

COINCIDENTODOS(**in** *r1* : `dicc(campo dato) in cc: conj(campo)`, *in r2* : `reg`) $\longrightarrow$ *res* : `bool`
  res ←

  while do

    if
      then

    end if
    Avanzar(cr);

  end while

O(calcular)

ENTODOS(**in** c : `campo in cr: conj(registro)`) $\longrightarrow$ *res* : `bool`
  res ← True;
  ;

  while
    do

  end while

O(calcular)

COMBINARTODOS(**in** c : `campo in r1: dicc(campo dato) in cr: conj(registro)`) $\longrightarrow$ *res* : `itConj(reg)`
  if $\neg(encerrado?(campus, campus.hippies.obtener(nombre)))$ **then**
    O(long(nombre))

    O(long(nombre))
  end if

O(long(nombre) + N_e)