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: El nombre de la tabla se retorna por referencia, no modificable.

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, hay aliasing "no modificable".

INDICES(**in** t : `tab`) $\longrightarrow$ res : `itConjTrie( itConj ALGO (dato) )`
Pre $\equiv$ {true}
Post $\equiv$ $\{res =_{\text{obs}} indices(t)\}$
Descripción: Devuelve un conjunto a los indices de la tabla ingresada por parametro.
Complejidad: O($calcular$)
Aliasing: Se devuelve un iterador al conjunto, que contiene los iteradores de los indices, hay aliasing.

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 un iterador al conjunto, hay aliasing "no modificable".

TIPOCAMPO(**in** c : `campo`, **in** t : `tab`) $\longrightarrow$ res : `tipo`
Pre $\equiv$ {c $\in$ campos(t)}
Post $\equiv$ $\{res =_{\text{obs}} tipoCampo(t)\}$
Descripción: Devuelve el tipo del campo c en la tabla ingresada por parametro.
Complejidad: O(1)
Aliasing: Se devuelve el tipo del campo, hay aliasing "no 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 un iterador al conjunto de registros, hay aliasing.

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 la cantidad de cambios, hay aliasing "no modificable".

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($T * L + in$)

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($T * L + in$)

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 de nombre y tipo de 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....(completar)
Complejidad: O($T * L + in$)

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....(completar)

Complejidad: O($T * L + in$)

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($T * L + in$)

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($T * L + in$)

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($T * L + in$)

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($T * L + in$)

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($T * L + in$)

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

1.2 Representación

se representa con `Tabla`
donde `tab` **es** `tupla`⟨Nombre : `String`,
Indices : `tupla`⟨IndicesL : `Lista(campo)`, ,
IndicesA : `Arreglo(Indice)`⟩
Registros : `Lista(Registro)`,
Campos : `DiccTrie(Campo,Bool)`,
#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

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

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

1.3 Algoritmos

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

CLAVES(**in** t : `tab`) $\longrightarrow$ res : `itDiccTrie(campo)`
$res \leftarrow CrearItRapido(t.campos.clavesDeTabla)$ — O(1)
O(1)

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

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

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

REGISTROS(**in** t : `tab`) $\longrightarrow$ res : `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* : `itConj(registro)`, *in/out cr2* : `itConj(registro)`)

while HaySiguiente(cr1) **do**
combinarTodos(c,Siguiente(cr1),cr2);

end while

O(calcular)

HAYCOINCIDENCIA(**in** *r* : `registro`, *in cc* : `itConjTrie(campo)`, *in cr* : `itConj(registro)`) $\longrightarrow$ *res* : `bool`

res $\leftarrow$ false;

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

end while

O(calcular)

COINCIDENCIAS(**in** *crit* : `registro`, *in cr* : `itConj(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, CrearItConj(t.registros)));

O(calcular)

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

res $\leftarrow$ max(dameColumna(c, CrearItConj(t.registros)));

O(calcular)

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

res $\leftarrow$ CrearItConj(vacio());

while HaySiguiente(cr) **do**

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

end if
Avanzar(cr);

end while

O(calcular)

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

res ← True;
itconjClaves ← CrearItConj(claves(r));

while HaySiguiente(itconjClaves) **do**
res ← res∧(tipo?(Obtener(r,Siguiente(itconjClaves)))=tipoCampo(Siguiente(itconjClaves)),t)
Avanzar(cr);

end while

O(calcular)

MOVERHIPPIE(**in/out** *campus* : `campusSeguro`, *in nombre* : `string`)
if ¬(*encerrado*?(*campus*, *campus.hippies.obtener*(*nombre*))) **then**
O(long(nombre))
// Obtener pos siguiente y actualizar posicion de hippie
$posVieja \leftarrow campus.hippies.obtener(nombre)$ O(long(nombre))
$posNueva \leftarrow proxPosHippie(campus, nombre)$ O(long(nombre) + N_e)
$campus.campus[posVieja.x][posVieja.y].hayHippie \leftarrow False$
O(1)
$campus.campus[posNueva.x][posNueva.y].hayHippie \leftarrow True$
O(1)
$itHippie \leftarrow campus.campus[posVieja.x][posVieja.y].hippie$
O(1)
$campus.campus[posVieja.x][posVieja.y].hippie \leftarrow NULL$
O(1)
$campus.campus[posNueva.x][posNueva.y].hippie \leftarrow itHippie$
O(1)
// Sancionar agentes que rodean a los estudiantes que encierro
sancionarAgentesEncerrandoEstVecinos(campus,posNueva)
O(1)
// Capturar hippies encerrados
aplicarHippiesVecinos(campus,posNueva) O(long(nombre))
// Hippificar estudiantes
hippificarEstudiantesVecinos(campus,posNueva) O(long(nombre))
end if

O(long(nombre) + N_e)

MOVERAGENTE(**in/out** *campus* : `campusSeguro`, *in placa* : `placa`)
// Obtener pos siguiente y actualizar pos de agente
$posVieja \leftarrow busquedaBinariaPorPlaca(campus.agentesPorPlaca, placa).pos$
O(log(N_a))
if ¬(*encerrado*?(*campus*, $posVieja$))
∧
$campus.campus[posVieja.x][posVieja.y].agente.siguienteSignificado().cantSanciones < 3$ **then**
O(1)
$proxPos \leftarrow campus.proxPosAgente(posVieja)$ O(N_h)
$campus.campus[posVieja.x][posVieja.y].hayAgente \leftarrow False$
O(1)
$campus.campus[proxPos.x][proxPos.y].hayAgente \leftarrow True$
O(1)
$itAgente \leftarrow campus.campus[posVieja.x][posVieja.y].agente$
O(1)
$campus.campus[proxPos.x][proxPos.y].agente \leftarrow itAgente$
O(1)

$campus.campus[posVieja.x][posVieja.y].agente \leftarrow NULL$ — O(1)

$\pi_2(busquedaBinaria(campus.agentesPorPlaca, placa)).pos \leftarrow proxPos$ — O(log(N_a))

$sancionarAgentesEncerrandoEstVecinos(campus, proxPos)$ — O(1)

$aplicarHippiesVecinos(campus, proxPos)$ — O(long(nombre))

end if

O($N_h + log(N_a) + long(nombre)$)

---

conMismasSanciones(**in** *campus* : `campusSeguro`, *in placa* : `placa`) $\longrightarrow$ *res* : `Conj(agente)`

$posAgente \leftarrow campus.agentes.obtener(placa)$ — O(θ(1))

$res \leftarrow campus.campus[posAgente.x][posAgente.y].agente.siguienteSignificado().mismcampus.agentes$ — O(1)

O(θ(1))

---

conKSanciones(**in** *campus* : `campusSeguro`, *in k* : `nat`) $\longrightarrow$ *res* : `Conj(agente)`

$res \leftarrow \emptyset$ — O(1)

if $campus.conKSanciones.ocurrioSancion$ **then**

// 'Copio' la lista de porSanciones a un vector, asi luego puedo hacer bus binaria sobre el

$campus.conKSanciones \leftarrow CrearArreglo(campus.porSanciones.tamanio())$ — O(1)

$it \leftarrow CrearItLista(campus.porSanciones)$ — O(1)

$i \leftarrow 0$ — O(1)

while $it.haySiguiente$ **do** — $O(N_a)$

$conKSanciones.arreglo[i].cantSanciones \leftarrow it.siguienteSignificado().cantSanciones$ — O(1)

// Por referencia

$conKSanciones.arreglo[i].conKSanciones \leftarrow it.siguienteSignificado().agentes$ — O(1)

if $it.siguienteSignificado().cantSanciones = k$ **then**

$res \leftarrow it.siguienteSignificado().agentes$ — O(1)

end if

$i++$ — O(1)

$it.avanzar()$ — O(1)

end while

$return\ res$ — O(1)

else

$bb \leftarrow busquedaBinaria(conKSanciones.arreglo, k)$ — $O(log(N_a))$

if $\pi_1(bb)$ **then**

$return\ res \leftarrow \pi_2(bb)$ — $O(1)$

else

$return\ res \leftarrow \emptyset$ — $O(1)$

end if

end if

O(N_a) $\vee$ O($log(N_a)$)

---

masVigilante(**in** *campus* : `campusSeguro`) $\longrightarrow$ *res* : `placa`

$res \leftarrow campus.masVigilante.siguienteClave()$ — O(1)

O(1)

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)

$campus.conKSanciones.ocurrioSancion \leftarrow True$	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 $agente.siguiente.mismcampus.haySiguiente()$
$\wedge$ $agente.siguiente.mismas.siguiente.cantSanciones < agente.siguiente.cantSanciones$ **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 $\neg(agente.siguiente.mismas.haySiguiente) \vee$
$(agente.siguiente.mismas.haySiguiente \wedge$
$agente.siguiente.cantSanciones = agente.siguiente.mismas.cantSanciones)$ **then**
O(1)
$nConMismasB \leftarrow nuevaTupla(CrearNuevoDiccLineal(), agente.siguiente.cantSanciones)$
$agente.siguiente.mismas \leftarrow agente.siguiente.mismas.agregarComoSiguiente(nConMismasB)$
O(1)
$agente.siguiente.miUbicacion \leftarrow$
$agente.siguiente.mismas.siguiente.agentes.agregarComoSiguiente(agente.siguiente.pl)$
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`) $\longrightarrow$ *res* : `bool`
$vecinos \leftarrow campus.campusEstatico.vecinos(pos)$
$alMenos1Agente \leftarrow False$ O(1)
$i \leftarrow 0$
if $\neg(encerrado?(pos, campus.campusEstatico.vecinos(pos)))$ **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)
$i \leftarrow vecinos.tamanio()$	O(1)
while $i < vecinos.tamanio()$ **do**	O(1)
if $\neg(campus.campus[vecinos[i].x][vecinos[i].y].hayAgente? \vee$	
$campus.campus[vecinos[i].x][vecinos[i].y].hayEst? \vee$	
$campus.campus[vecinos[i].x][vecinos[i].y].hayHippie? \vee$	
$campus.campus[vecinos[i].x][vecinos[i].y].hayObst?)$ **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)

$vecinos \leftarrow campus.campusEstatico.vecinos(pos)$	O(1)
$i \leftarrow 0$	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)
$campus : campusSeguro.campus[pos.x][pos.y].hayHippie \leftarrow False$ O(1)
$campus : campusSeguro.campus[pos.x][pos.y].hayEst \leftarrow True$ O(1)
$as.campus[pos.x][pos.y].estudiante \leftarrow CrearIt(campus.hippies)$ 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 $campus.campus[pos.x][pos.y].hayHippie? \wedge atrapadoPorAgente(pos)$ **then**
$vecinos \leftarrow campus.campusSeguro.vecinos(pos)$ O(1)
$i \leftarrow 0$ O(1)
while $i < vecinos.tamanio()$ **do** O(1)
$posAct \leftarrow vecinos[pos.x][pos.y]$ O(1)
$info \leftarrow campus.campus[vecinos[i].x][vecinos[i].y]$ 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**
$campus.masVigilante \leftarrow info.agente$ 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 $PosValida/c, < p.X + 1, p.Y >) \wedge \neg HayAlgo(c, < p.X + 1, p.Y >)$ **then**
      $res \leftarrow < p.X + 1, p.Y >$
    else
      if $PosValida/c, < p.X - 1, p.Y >) \wedge \neg HayAlgo(c, < p.X - 1, p.Y >)$ **then**
        $res \leftarrow < p.X - 1, p.Y >$
      else
        $res \leftarrow < p.X, p.Y + 1 >$
      end if
    end if
  end if
else
  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 $PosValida/c, < p.X + 1, p.Y >) \wedge \neg HayAlgo(c, < p.X + 1, p.Y >)$ **then**
      $res \leftarrow < p.X + 1, p.Y >$
    else
      if $PosValida/c, < p.X - 1, p.Y >) \wedge \neg HayAlgo(c, < p.X - 1, p.Y >)$ **then**
        $res \leftarrow < p.X - 1, p.Y >$
      else
        $res \leftarrow < p.X, p.Y - 1 >$
      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|)