



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, alfa), diccAVL

### Operaciones

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

Pre  $\equiv \{true\}$

Post  $\equiv \{res =_{obs} nombre(t)\}$

**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  $\equiv \{res =_{obs} claves(t)\}$

**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  $\equiv \{res =_{obs} indices(t)\}$

**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  $\equiv \{res =_{obs} campos(t)\}$

**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  $\equiv \{res =_{obs} tipoCampo(t)\}$

**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  $\equiv \{res =_{obs} registros(t)\}$

**Descripción:** Devuelve un conjunto a los registros de la tabla ingresada por parametro.

**Complejidad:**  $O(L + \log(n))$

**Aliasing:** Se devuelve res referencia

CANTIDADDEACCESOS(**in**  $t : \text{tab}$ )  $\longrightarrow res : \text{nat}$

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{cantidadDeAccesos}(t)\}$

**Descripción:** Devuelve la cantidad de modificaciones de la tabla ingresada por parametro.

**Complejidad:**  $O(1)$

**Aliasing:** Se devuelve res por copia.

NUEVATABLA(**in**  $\text{nombre} : \text{string}$ ,  $\text{in claves} : \text{conjTrie}(\text{campo})$ ,  $\text{in columnas} : \text{registro}$ )  $\longrightarrow res : \text{tab}$

**Pre**  $\equiv \{\neg \emptyset?(\text{claves}) \wedge \text{claves} \subseteq \text{campos}(\text{columnas})\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{cantidadDeAccesos}(t)\}$

**Descripción:** Crea una tabla sin registros.

**Complejidad:**  $O(\text{calcular})$

AGREGARREGISTRO(**in**  $r : \text{registro}$ ,  $\text{in } t : \text{tab}$ )

**Pre**  $\equiv \{t\_0 = t \wedge \text{campos}(r) =_{\text{obs}} \text{campos}(t) \wedge \text{puedoInsertar?}(r, t)\}$

**Post**  $\equiv \{\text{agregarRegistro}(r, t\_0)\}$

**Descripción:** Agrega un registro a la tabla pasada por parametro.

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

**Aliasing:** Agrega el registro r por referencia.

BORRARREGISTRO(**in**  $\text{crit} : \text{registro}$ ,  $\text{in } t : \text{tab}$ )

**Pre**  $\equiv \{t\_0 = t \wedge \#(\text{campos}(r)) = 1 \wedge_L \text{dameUno}(\text{campos}(\text{crit})) \in \text{claves}(t)\}$

**Post**  $\equiv \{\text{borrarRegistro}(r, t\_0)\}$

**Descripción:** Borra los registros que cumplan el criterio pasado por parametro.

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

INDEXAR(**in**  $\text{crit} : \text{registro}$ ,  $\text{in } t : \text{tab}$ )

**Pre**  $\equiv \{t\_0 = t \wedge \text{puedeIndexar}(c, t)\}$

**Post**  $\equiv \{\text{indexar}(c, t\_0)\}$

**Descripción:** Borra los registros que cumplan el criterio pasado por parametro.

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

PUEDOIINSERTAR?(**in**  $r : \text{registro}$ ,  $\text{in } t : \text{tab}$ )  $\longrightarrow res : \text{bool}$

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{puedoInsertar?}(r, t)\}$

**Descripción:** Informa si el registro pasado por parametro no tiene valores repetidos con respecto a los registros existentes, para los campos clave en la tabla pasada por parametro.

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

COMPATIBLE(**in**  $r : \text{registro}$ ,  $\text{in } t : \text{tab}$ )  $\longrightarrow res : \text{bool}$

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{compatible}(r, t)\}$

**Descripción:** Informa si el registro pasado por parametro tiene correspondencia en los tipos de los campos de tabla pasada por parametro.

**Complejidad:**  $O(1)$

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

**Pre**  $\equiv \{\neg \emptyset?(\text{registro}(t)) \wedge c \in \text{indices}(t)\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{minimo}(c, t)\}$

**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**  $c : \text{campo}$ ,  $\text{in } t : \text{tab}$ )  $\longrightarrow res : \text{dato}$

**Pre**  $\equiv \{\neg \emptyset?(\text{registro}(t)) \wedge c \in \text{indices}(t)\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{maximo}(c, t)\}$

**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**  $c : \text{campo}$ , **in**  $t : \text{tab}$ )  $\longrightarrow res : \text{bool}$

**Pre**  $\equiv \{True\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{puedeIndexar}(c, t)\}$

**Descripción:** Informa si se puede crear un nuevo indice.

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

COINCIDENCIAS(**in**  $r : \text{registro}$ , **in**  $cj : \text{Conj}(\text{registro})$ )  $\longrightarrow res : \text{Conj}(\text{registro})$

**Pre**  $\equiv \{True\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{coincidencias}(r, cj)\}$

**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**  $r : \text{registro}$ , **in**  $cj1 : \text{ConjTrie}(\text{campo})$ , **in**  $cj2 : \text{Conj}(\text{registro})$ )  $\longrightarrow res : \text{bool}$

**Pre**  $\equiv \{True\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{hayCoincidencia}(r, cj1, cj2)\}$

**Descripción:** Compara los valores del registro para los campos dados por parametro, con el conjunto de registros.

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

COMBINARREGISTROS(**in**  $c : \text{campo}$ , **in**  $cj1 : \text{Conj}(\text{registro})$ , **in**  $cj2 : \text{Conj}(\text{registro})$ )  $\longrightarrow res : \text{conj}(\text{registro})$

**Pre**  $\equiv \{True\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{combinarRegistros}(c, cj1, cj2)\}$

**Descripción:** Combina los valores de los registros para el campo dado por parametro.

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

**Aliasing:** Retorna res por copia.

DAMECOLUMNA(**in**  $c : \text{campo}$ , **in**  $cj : \text{Conj}(\text{registro})$ )  $\longrightarrow res : \text{conj}(\text{dato})$

**Pre**  $\equiv \{True\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{dameColumna}(c, cj1, cj2)\}$

**Descripción:** Reune en un conjunto los valores del campo pasado por parametro.

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

**Aliasing:** Retorna res por referencia.

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

**Pre**  $\equiv \{\text{campos}(r) \subseteq \text{campos}(t)\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{mismosTipos}(r, t)\}$

**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  $tupla\langle Nombre : String,$   
 $Indices : ConjTrie(tupla\langle IndicesC : campo,$   
 $IndicesA : Arreglo(Indice)\rangle),$   
 $Registros : Conj(Registro),$   
 $Campos : DiccTrie(Campo, Tipo),$   
 $\#Accesos : Nat\rangle$   
 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

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

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

### 1.3 Algoritmos

NOMBRE( <b>in</b> $t : \text{tab}$ ) $\longrightarrow res : \text{string}$ $res \leftarrow t.nombre$	$O(1)$
	$O(1)$
CLAVES( <b>in</b> $t : \text{tab}$ ) $\longrightarrow res : \text{itConjTrie}(\text{campo})$ $res \leftarrow \text{CrearItConjTrie}(t.Campos.ClavesDicc)$	$O(1)$
	$O(1)$
INDICES( <b>in</b> $t : \text{tab}$ ) $\longrightarrow res : \text{itConjTrie}(\text{Indice})$ $res \leftarrow \text{CrearItConjTrie}(t.Indices)$	$O(1)$
	$O(1)$
CAMPOS( <b>in</b> $t : \text{tab}$ ) $\longrightarrow res : \text{itConjTrie}(\text{campo})$ $res \leftarrow \text{CrearItConjTrie}(t.Campos.ClavesDicc)$	$O(1)$
	$O(1)$
TIPOCAMPO( <b>in</b> $c : \text{campo}$ , $in t : \text{tab}$ ) $\longrightarrow res : \text{Tipo}$ $res \leftarrow \text{Significado}(t.Campos, c)$	$O(1)$
	$O(1)$
REGISTROS( <b>in</b> $t : \text{tab}$ ) $\longrightarrow res : \text{itConj}(\text{registro})$ $res \leftarrow \text{CrearItConj}(t.registros)$	$\theta(L + \log(n))$
	$\theta(L + \log(n))$
CANTDEACCESOS( <b>in</b> $t : \text{tab}$ ) $\longrightarrow res : \text{nat}$ $res \leftarrow t.cantDeAccesos$	$\theta(1)$
	$\theta(1)$
PUEDOINSERTAR?( <b>in</b> $r : \text{registro}$ , $in t : \text{tab}$ ) $\longrightarrow res : \text{bool}$ $res \leftarrow \text{compatible}(r, t) \wedge \neg \text{hayCoincidencia}(r, \text{claves}(r), \text{registros}(t))$	$\theta(L + \log(n))$
	$\theta(L + \log(n))$
COMPATIBLE( <b>in</b> $r : \text{registro}$ , $in t : \text{tab}$ ) $\longrightarrow res : \text{bool}$ $res \leftarrow \text{compatible}(r, t) \wedge_L \text{mismosTipos}(r, t)$	$\theta(1)$
	$O(1)$
PUEDEINDEXAR( <b>in</b> $c : \text{campo}$ , $in t : \text{tab}$ ) $\longrightarrow res : \text{bool}$ $res \leftarrow \text{Defnido?}(t.campos, c) \wedge_L \neg \text{Esta?}(c, t.Indices.IndicesL) \wedge$ $(\text{long}(t.Indices.IndicesL) = 0 \vee (\text{long}(t.Indices.IndicesL) < 2 \wedge$ $(\text{tipoCampo}(c, t) \neq \text{tipoCampo}(t.Indices.IndicesL.primerO, t))))$	$\theta(1)$
	$O(\text{calcular})$
COMBINARREGISTROS( <b>in</b> $c : \text{campo}$ , $in cr1 : \text{Conj}(\text{registro})$ , $in cr2 : \text{Conj}(\text{registro})$ ) $\longrightarrow$ $res : \text{Conj}(\text{registros})$ $res \leftarrow \text{vacio}();$ $itcr1 \leftarrow \text{CrearItConjTrie}(cr1)$	

```

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**  $r : \text{registro}$ ,  $in\ t : \text{tab}$ )  $\longrightarrow res : \text{bool}$

```

    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**  $campus : \text{campusSeguro}$ ,  $in\ pos : \text{pos}$ ,  $in\ nombre : \text{nombre}$ )

```

    campus.campus[pos.x][pos.y].hayEst  $\leftarrow$  True
    campus.campus[pos.x][pos.y].estudiante  $\leftarrow$  definir(campus.estudiantes,nombre,pos)

```

O(1)

O(long(nombre))

---

O(long(nombre))

AGREGARHIPPIE(**in/out**  $campus : \text{campusSeguro}$ ,  $in\ pos : \text{pos}$ ,  $in\ nombre : \text{nombre}$ )

```

    campus.campus[pos.x][pos.y].hayHippie  $\leftarrow$  True
    campus.campus[pos.x][pos.y].hippie  $\leftarrow$  definir(campus.hippies,nombre,pos)

```

O(1)

O(long(nombre))

---

O(long(nombre))

SANCIONARAGENTESVECINOS(**in/out**  $campus : \text{campusSeguro}$ ,  $in\ pos : \text{pos}$ )

```

    vecinos  $\leftarrow$  campus.campusEstatico.vecinos(pos)
    if campus.atrapadoPorAgente?(pos) then
        while i < vecinos.tamano() do
            if campus.campus[vecinos[i].x][vecinos[i].y].hayAgente? then
                campus.sancionarAgente(vecinos[i].agente)
            end if
            i ++
        end while
    end if

```

O(1)

O(1)

O(1)

---

O(1)

SANCIONARAGENTESENCERRANDOESTVECINOS(**in/out**  $campus : \text{campusSeguro}$ ,  $in\ pos : \text{pos}$ )

```

    vecinos  $\leftarrow$  campus.campusEstatico.vecinos(pos)
    i  $\leftarrow$  0
    while i < vecinos.tamano do
        if campus.campus[vecinos[i].x][vecinos[i].y].hayEst  $\wedge$  atrapadoPorAgente?(campus,pos)
        then

```

O(1)

O(1)

O(1)

[illegible]
$$O(1)$$

```

SANCIONARAGENTE(in/out campus : campusSeguro, in/out agente : itDiccRapido)
    campus.conK Sanciones.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)

$$O(1)$$

```

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.tamano() do O(1)
        if as.campus[vecinos[i].x][vecinos[i].y].hayAgente? then
            return true
        end if
        i ++ O(1)
    end while

```

 $O(1)$ 
$$O(1)$$

	O(1)
HIPPIFICARESTUDIANTESVECINOS( <b>in/out</b> <i>campus</i> : <b>campusSeguro</b> , <i>in pos</i> : <b>pos</b> )	
<i>vecinos</i> $\leftarrow$ <i>campus.campusEstatico.vecinos(pos)</i>	O(1)
<i>i</i> $\leftarrow$ 0	O(1)
<b>while</b> <i>i</i> < <i>vecinos.tamano()</i> <b>do</b>	O(long(nombre))
<b>if</b> <i>estAHippie?(campus, vecinos[i])</i> <b>then</b>	
<i>hippificar(campus, vecinos[i])</i>	O(long(nombre))
<b>end if</b>	
<i>i</i> ++	O(1)
<b>end while</b>	
	O(long(nombre))
HIPPIFICAR( <b>in/out</b> <i>campus</i> : <b>campusSeguro</b> , <i>in pos</i> : <b>pos</b> )	
// PRE: La posicion esta en el tablero y hay estudiante en la posicion	
<i>as.campus[pos.x][pos.y].hayHippie</i> $\leftarrow$ <i>True</i>	O(1)
<i>as.campus[pos.x][pos.y].hippie.agregarComoSiguiente(nombre, pos)</i>	O(long(nombreEstudiante))
<i>as.campus[pos.x][pos.y].hayEst</i> $\leftarrow$ <i>False</i>	O(1)
<i>as.campus[pos.x][pos.y].estudiante.eliminarSiguiente()</i>	O(long(nombreEstudiante))
	O(long(nombre))
ESTAHIPPIE?( <b>in</b> <i>campus</i> : <b>campusSeguro</b> , <i>in pos</i> : <b>pos</b> ) $\rightarrow$ <i>res</i> : <b>bool</b>	
<b>if</b> $\neg$ ( <i>encerrado?(pos, vecinos)</i> ) <b>then</b>	
<i>return false</i>	O(1)
<b>end if</b>	
<i>i</i> $\leftarrow$ 0	O(1)
<i>cantHippies</i> $\leftarrow$ 0	O(1)
<i>vecinos</i> $\leftarrow$ <i>campus.campusEstatico.vecinos(pos)</i>	O(1)
<b>while</b> <i>i</i> < <i>vecinos.tamano()</i> <b>do</b>	
<b>if</b> <i>campus[vecinos[i].x][vecinos[i].y].hayHippie</i> <b>then</b>	
<i>cantHippies</i> ++	O(1)
<b>end if</b>	
<i>i</i> ++	
<b>end while</b>	
<i>return cantHippies</i> $\geq$ 2	O(1)
	O(1)
HIPPIEAEST?( <b>in</b> <i>campus</i> : <b>campusSeguro</b> , <i>in pos</i> : <b>pos</b> ) $\rightarrow$ <i>res</i> : <b>bool</b>	
<i>i</i> $\leftarrow$ 0	O(1)
<i>vecinos</i> $\leftarrow$ <i>campus.campusEstatico.vecinos(pos)</i>	O(1)
<b>while</b> <i>i</i> < <i>vecinos.tamano()</i> <b>do</b>	O(1)
<b>if</b> $\neg$ ( <i>as.campus[vecinos[i].x][vecinos[i].y].hayEst?</i> ) <b>then</b>	
<i>return False</i>	O(1)
<b>end if</b>	
<b>end while</b>	
<i>return True</i>	
	O(1)
ENCERRADO?( <b>in</b> <i>campus</i> : <b>campusSeguro</b> , <i>in pos</i> : <b>pos</b> )	
<i>vecinos</i> $\leftarrow$ <i>vecinos(as.campusEstatico, pos)</i>	O(1)

$i \leftarrow \text{vecinos.tamano}()$	$O(1)$
<b>while</b> $i < \text{vecinos.tamano}()$ <b>do</b>	$O(1)$
<b>if</b> $\neg(\text{campus.campus}[\text{vecinos}[i].x][\text{vecinos}[i].y].\text{hayAgente?} \vee$ $\text{campus.campus}[\text{vecinos}[i].x][\text{vecinos}[i].y].\text{hayEst?} \vee$ $\text{campus.campus}[\text{vecinos}[i].x][\text{vecinos}[i].y].\text{hayHippie?} \vee$ $\text{campus.campus}[\text{vecinos}[i].x][\text{vecinos}[i].y].\text{hayObst?})$ <b>then</b>	$O(1)$
$\text{return false}$	$O(1)$
<b>end if</b>	
$i++$	$O(1)$
<b>end while</b>	
$\text{return true}$	
<hr/>	
	$O(1)$
<hr/>	
APLICARHIPPIESVECINOS( <b>in/out</b> $\text{campus} : \text{campusSeguro}$ , $\text{in pos} : \text{pos}$ )	
$\text{vecinos} \leftarrow \text{campus.campusEstatico.vecinos}(\text{pos})$	$O(1)$
$i \leftarrow 0$	$O(1)$
<b>while</b> $i < \text{vecinos.tamano}()$ <b>do</b>	$O(\text{long}(\text{nombre}))$
$\text{aplicarHippie}(\text{campus}, \text{pos})$	$O(\text{long}(\text{nombre}))$
<b>end while</b>	
<hr/>	
	$O(\text{long}(\text{nombre}))$
<hr/>	
APLICARHIPPIE( <b>in/out</b> $\text{campus} : \text{campusSeguro}$ , $\text{in pos} : \text{pos}$ )	
// PRE: pos valida y hayHippie en $\text{campus.campus}[\text{pos}.x][\text{pos}.y]$	
<b>if</b> $\text{campus.campus}[\text{pos}.x][\text{pos}.y].\text{hayHippie}$ <b>then</b>	
<b>if</b> $\text{as.hippieAEst}(\text{pos})$ <b>then</b>	$O(1)$
$\text{campus} : \text{campusSeguro.campus}[\text{pos}.x][\text{pos}.y].\text{hayHippie} \leftarrow \text{False}$	$O(1)$
$\text{campus} : \text{campusSeguro.campus}[\text{pos}.x][\text{pos}.y].\text{hayEst} \leftarrow \text{True}$	$O(1)$
$\text{as.campus}[\text{pos}.x][\text{pos}.y].\text{estudiante} \leftarrow \text{CrearIt}(\text{campus.hippies})$	$O(1)$
$\text{campus.campus}[\text{pos}.x][\text{pos}.y].\text{estudiante}.$ $\text{agregarComoSiguiete}(\text{campus.campus}[\text{pos}.x][\text{pos}.y].\text{estudiante.nombre})$	$O(\text{long}(\text{nombre}))$
$\text{campus.campus}[\text{pos}.x][\text{pos}.y].\text{hippie.eliminarSiguiete}()$	$O(\text{long}(\text{nombre}))$
<b>else</b>	
<b>if</b> $\text{campus.campus}[\text{pos}.x][\text{pos}.y].\text{hayHippie?} \wedge \text{atrapadoPorAgente}(\text{pos})$ <b>then</b>	
$\text{vecinos} \leftarrow \text{campus.campusSeguro.vecinos}(\text{pos})$	$O(1)$
$i \leftarrow 0$	$O(1)$
<b>while</b> $i < \text{vecinos.tamano}()$ <b>do</b>	$O(1)$
$\text{posAct} \leftarrow \text{vecinos}[\text{pos}.x][\text{pos}.y]$	$O(1)$
$\text{info} \leftarrow \text{campus.campus}[\text{vecinos}[i].x][\text{vecinos}[i].y]$	$O(1)$
<b>if</b> $\text{posAct.hayAgente}$ <b>then</b>	
$\text{info.agente.siguiete.cantCapturas}++$	$O(1)$
// Actualizar mas vigilante	
<b>if</b> $\text{campus.masVigilante.siguieteSignificado().cantCapturas} <$ $\text{info.agente.siguieteSignificado().cantCapturas}$ <b>then</b>	
$\text{campus.masVigilante} \leftarrow \text{info.agente}$	$O(1)$
<b>else</b>	
<b>if</b> $\text{campus.masVigilante.siguieteSignificado().cantCapturas} =$	

```

info.agente.siguienteSignificado().cantCapturas
^campus.masVigilante.siguienteClave() < info.agente.siguienteClave() then
                                                                    O(1)
    campus.masVigilante ← 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) → res : pos
// PRE: El nombre es un hippie y el hippie no esta encerrado
posHippie ← campus.hippies.obtener(nombre)
                                                                    O(long(nombre))
if campus.estudiantes.tamano() > 0 then
    // Retorna de las posiciones mas cercanas, la que esta mas cerca del (0,0)
    proxPos ← aPosMasCercana(campus.campusEstatico, posHippie, campus.estudiantes.significados)
                                                                    O(Ne)
else
    // Retorna el ingreso mas cercan, en caso de empate, el de abajo
    proxPos ← aIngresoMasCercano(campus.campusEstatico, posHippie)
                                                                    O(1)
end if
res ← proxPos
                                                                    O(1)
                                                                    O(Ne)
PROXPOSAGENTE(in/out campus : campusSeguro, in posAgente : pos) → res : pos
// PRE: En la posicion hay un agente que se puede mover
if campus.hippies.tamano() > 0 then
    // Retorna de las posiciones mas cercanas, la que esta mas cerca del (0,0)
    proxPos ← aPosMasCercana(campus.campusEstatico, posAgente, campus.hippies.significados)
                                                                    O(Nh)
else
    // Retorna el ingreso mas cercano, en caso de empate, el de abajo
    proxPos ← aIngresoMasCercano(campus.campusEstatico, posAgente)
                                                                    O(1)
end if
res ← proxPos
                                                                    O(1)
                                                                    O(Nh)
AINGRESOMASCERCANO(in p : pos, cs : campusSeguro) → res : pos
if p.Y ≤ c.alto/2 then
    if PosValida(cs.campus, < p.X, p.Y - 1 >) ∧ ¬HayAlgo(cs, < p.X, p.Y - 1 >) then
        res ← < p.X, p.Y - 1 >
    end if
end if

```

```

else
  if  $PosValida/c, \langle p.X + 1, p.Y \rangle \wedge \neg HayAlgo(c, \langle p.X + 1, p.Y \rangle)$  then
     $res \leftarrow \langle p.X + 1, p.Y \rangle$ 
  else
    if  $PosValida/c, \langle p.X - 1, p.Y \rangle \wedge \neg HayAlgo(c, \langle p.X - 1, p.Y \rangle)$  then
       $res \leftarrow \langle p.X - 1, p.Y \rangle$ 
    else
       $res \leftarrow \langle p.X, p.Y + 1 \rangle$ 
    end if
  end if
end if
else
  if  $PosValida(cs.campus, \langle p.X, p.Y + 1 \rangle) \wedge \neg HayAlgo(cs, \langle p.X, p.Y + 1 \rangle)$  then
     $res \leftarrow \langle p.X, p.Y - 1 \rangle$ 
  else
    if  $PosValida/c, \langle p.X + 1, p.Y \rangle \wedge \neg HayAlgo(c, \langle p.X + 1, p.Y \rangle)$  then
       $res \leftarrow \langle p.X + 1, p.Y \rangle$ 
    else
      if  $PosValida/c, \langle p.X - 1, p.Y \rangle \wedge \neg HayAlgo(c, \langle p.X - 1, p.Y \rangle)$  then
         $res \leftarrow \langle p.X - 1, p.Y \rangle$ 
      else
         $res \leftarrow \langle p.X, p.Y - 1 \rangle$ 
      end if
    end if
  end if
end if
end if

```

---

$O(1)$

$IBUSQUEDABINARIAPOR Sanciones(\text{in } ar : \text{arreglo}(\text{val} : \text{nat } otr : \alpha), \text{ in } sanc : \text{nat}) \longrightarrow res : \langle \alpha, \text{bool} \rangle$

```

 $res.\pi_2 \leftarrow false$ 
 $min \leftarrow 0$ 
 $max \leftarrow |ar|$ 
while  $max - min > 1$  do
   $med \leftarrow (max + min) / 2$ 
  if  $ar[med].val \leq sanc$  then
     $min \leftarrow med$ 
  else
     $max \leftarrow med$ 
  end if
end while
if  $ar[min].val = sanc$  then
   $res \leftarrow \langle ar[min].otr, true \rangle$ 
end if

```

$O(\log(|ar|))$

---

$O(\log(|ar|))$

## 2 Tipo es Bool

## 3 Dato( $\alpha$ )

### 3.1 Interfaz

se explica con DATO

usa

géneros nat, string, tipo

#### Operaciones

TIPO?(in  $d : \text{dato}$ )  $\longrightarrow res : \text{tipo}$

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{tipo?}(d)\}$

**Descripción:** Devuelve el tipo del dato ingresado por parametro.

**Complejidad:**  $O(1)$

**Aliasing:** Se retorna res por referencia.

VALORNAT(in  $d : \text{dato}$ )  $\longrightarrow res : \text{nat}$

**Pre**  $\equiv \{\text{Nat?}(d)\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{valorNat}(t)\}$

**Descripción:** Devuelve valor numerido del dato por parametro.

**Complejidad:**  $O(1)$

**Aliasing:** Se devuelve res por referencia.

VALORSTRING(in  $d : \text{dato}$ )  $\longrightarrow res : \text{string}$

**Pre**  $\equiv \{\text{String?}(d)\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{valorString}(t)\}$

**Descripción:** Devuelve valor del dato por parametro.

**Complejidad:**  $O(1)$

**Aliasing:** Se devuelve res por referencia.

DATONAT(in  $n : \alpha$ , in  $\text{tipoDelDato} : \text{tipo}$ )  $\longrightarrow res : \text{dato}$

**Pre**  $\equiv \{\text{tipoDelDato}\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{datoNat}(n, \text{tipoDelDato})\}$

**Descripción:** Crea un dato de valor numerico.

**Complejidad:**  $O(1)$

**Aliasing:** Se devuelve res por referencia.

DATOSTR(in  $n : \alpha$ , in  $\text{tipoDelDato} : \text{tipo}$ )  $\longrightarrow res : \text{dato}$

**Pre**  $\equiv \{\neg \text{tipoDelDato}\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{datoString}(n, \text{tipoDelDato})\}$

**Descripción:** Crea un dato de valor de letras.

**Complejidad:**  $O(1)$

**Aliasing:** Se devuelve res por referencia.

MISMO TIPO?(in  $d1 : \text{dato}$ , in  $d2 : \text{dato}$ )  $\longrightarrow res : \text{bool}$

**Pre**  $\equiv \{\text{true}\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{mismoTipo?}(d1, d2)\}$

**Descripción:** Informa si los datos pasados por parametro son del mismo tipo de valor.

**Complejidad:**  $O(1)$

**STRING?**(**in**  $d : \text{dato}$ )  $\longrightarrow res : \text{bool}$

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{String?}(d)\}$

**Descripción:** Informa si el dato pasado por parametro es de tipo string.

**Complejidad:**  $O(1)$

**NAT?**(**in**  $d : \text{dato}$ )  $\longrightarrow res : \text{bool}$

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{Nat?}(d)\}$

**Descripción:** Informa si el dato pasado por parametro es de tipo nat.

**Complejidad:**  $O(1)$

**MIN**(**in**  $cd : \text{Conj}(\text{dato})$ )  $\longrightarrow res : \text{dato}$

**Pre**  $\equiv \{\neg \text{EsVacio?}(cd)\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{min}(cd)\}$

**Descripción:** Retorna el minimo entre los valores del conjunto de datos pasado por parametro.

**Complejidad:**  $O(\text{Cardinal}(cd))$

**Aliasing:** Retorna res por referencia.

**MAX**(**in**  $cd : \text{Conj}(\text{dato})$ )  $\longrightarrow res : \text{dato}$

**Pre**  $\equiv \{\neg \text{EsVacio?}(cd)\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{max}(cd)\}$

**Descripción:** Retorna el maximo entre los valores del conjunto de datos pasado por parametro.

**Complejidad:**  $O(\text{Cardinal}(cd))$

**Aliasing:** Retorna res por referencia.

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

**Pre**  $\equiv \{\text{mismoTipo?}(d1, d2)\}$

**Post**  $\equiv \{res =_{\text{obs}} <= (d1, d2)\}$

**Descripción:** Retorna el maximo entre los valores del conjunto de datos pasado por parametro.

**Complejidad:**  $O(\text{Cardinal}(cd))$

**Aliasing:** Retorna res por referencia. oincidenTodos(crit, campos(crit), Siguiente(cr))

## 3.2 Representación

se representa con  $\text{datotupla} \langle \text{Valor} : \alpha, \text{TipoValor} : \text{bool} \rangle$

### 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

$\text{Abs} : \widehat{\text{sistema}} s \longrightarrow \widehat{\text{CampusSeguro}}$

$\{\text{Rep}(s)\}$



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

### 3.3 Algoritmos

TIPO?( <b>in</b> <i>a</i> : dato) $\longrightarrow$ <i>res</i> : bool <i>res</i> $\leftarrow$ <i>a.TipoValor</i>	O(1)
	<hr/>
	O(1)
VALORNAT( <b>in</b> <i>a</i> : dato) $\longrightarrow$ <i>res</i> : nat <i>res</i> $\leftarrow$ <i>a.Valor</i>	O(1)
	<hr/>
	O(1)
VALORSTR( <b>in</b> <i>a</i> : dato) $\longrightarrow$ <i>res</i> : string <i>res</i> $\leftarrow$ <i>a.Valor</i>	O(1)
	<hr/>
	O(1)
MISMO TIPO?( <b>in</b> <i>d1</i> : dato, <i>in</i> <i>d2</i> : dato) $\longrightarrow$ <i>res</i> : bool <i>res</i> $\leftarrow$ <i>tipo?(d1) = tipo?(d2)</i>	O(1)
	<hr/>
	O(1)
NAT?( <b>in</b> <i>a</i> : dato) $\longrightarrow$ <i>res</i> : bool <i>res</i> $\leftarrow$ <i>tipo?(a)</i>	O(1)
	<hr/>
	O(1)
STRING?( <b>in</b> <i>a</i> : dato) $\longrightarrow$ <i>res</i> : bool <i>res</i> $\leftarrow$ $\neg$ <i>Nat?(a)</i>	O(1)
	<hr/>
	O(1)
MIN( <b>in</b> <i>cd</i> : Conj(dato)) $\longrightarrow$ <i>res</i> : dato <i>itcd</i> $\leftarrow$ CrearItConj( <i>cd</i> ) <i>minimo</i> $\leftarrow$ Siguiente( <i>itcd</i> )  <b>while</b> HaySiguiente( <i>itcd</i> ) <b>do</b>  <b>if</b> Siguiente( <i>itcd</i> ) $\neq$ <i>minimo</i> <b>then</b> <i>minimo</i> $\leftarrow$ Siguiente( <i>itcd</i> );  <b>end if</b> Avanzar( <i>itcr</i> );  <b>end while</b>	
	<hr/>
	O(Cardinal( <i>cd</i> ))
MAX( <b>in</b> <i>cd</i> : Conj(dato)) $\longrightarrow$ <i>res</i> : dato <i>itcd</i> $\leftarrow$ CrearItConj( <i>cd</i> ) <i>maximo</i> $\leftarrow$ Siguiente( <i>itcd</i> )  <b>while</b> HaySiguiente( <i>itcd</i> ) <b>do</b>  <b>if</b> <i>maximo</i> $\neq$ Siguiente( <i>itcd</i> ) <b>then</b> <i>minimo</i> $\leftarrow$ Siguiente( <i>itcd</i> );  <b>end if</b>	

Avanzar(itcr);	
end while	
$\leq$ =(in $d1 : \text{dato}$ , in $d2 : \text{dato}$ ) $\longrightarrow res : \text{bool}$	$O(\text{Cardinal}(\text{cd}))$
if String?( $d1$ ) then $res \leftarrow \text{valorStr}(d1)_{\text{i}} = \text{valorStr}(d2)$	
else $res \leftarrow \text{valorNat}(d1)_{\text{i}} = \text{valorNat}(d2)$	
end if	$O(1)$

### 3.4 Algoritmos operaciones auxiliares

## 4 Diccionario por Naturales

### 4.1 Interfaz

se explica con  $\text{DICCIONARIO}(\kappa, \sigma)$

usa Bool, Nat

géneros  $\text{diccNat}(\kappa, \sigma)$

#### Operaciones

$\text{VACIO}() \longrightarrow res : \text{diccNat}(\kappa, \sigma)$

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{vacio}()\}$

**Descripción:** Crea un nuevo diccionario

**Complejidad:**  $O(1)$

$\text{DEFINIDO?}(\text{in } d : \text{diccNat}(\kappa, \sigma) \text{ in } n : \kappa) \longrightarrow res : \text{Bool}$

**Pre**  $\equiv \{true\}$

**Post**  $\equiv \{res =_{\text{obs}} \text{def?}(n, d)\}$

**Descripción:** Indica si la clave esta definida

**Complejidad:**  $O(m)$

$\text{DEFINIR}(\text{in/out } d : \text{diccNat}(\kappa, \sigma) \text{ in } n : \kappa, \text{ in } s : \sigma)$

**Pre**  $\equiv \{\neg \text{def?}(n, d) \wedge d = d_0\}$

**Post**  $\equiv \{d =_{\text{obs}} \text{Definir}(n, s, d_0)\}$

**Descripción:** Se define s en el diccionario

**Complejidad:**  $O(m)$

$\text{BORRAR}(\text{in/out } d : \text{diccNat}(\kappa, \sigma) \text{ in } n : \kappa)$

**Pre**  $\equiv \{d =_{\text{obs}} d_0 \wedge \text{def?}(n, d)\}$

**Post**  $\equiv \{d =_{\text{obs}} \text{Borrar}(n, d_0)\}$

**Descripción:** Elimina el elemento  $n$

**Complejidad:**  $O(m)$

SIGNIFICADO( $\text{in } d : \text{diccNat}(\kappa, \sigma) \text{ in } n : \kappa \longrightarrow res : \sigma$ )

**Pre**  $\equiv \{def?(n, d)\}$

**Post**  $\equiv \{res =_{\text{obs}} obtener(n, d)\}$

**Descripción:** Se retornan los significados

**Complejidad:**  $O(m)$

## 4.2 Representación

$\text{diccNat}$

se representa con puntero( $\text{estr}(\kappa, \sigma)$ )

donde  $\text{estr}(\kappa, \sigma)$  es tupla( $\langle$ clave :  $\kappa$ ,  
significado :  $\sigma$ ,  
hijoDer : puntero( $\text{estr}(\kappa\sigma)$ ),  
hijoIzq : puntero( $\text{estr}(\kappa\sigma)$ ) $\rangle$ )

### Invariante de representación

$\text{Rep} : \widehat{\text{estr}} \longrightarrow \text{boolean}$

( $\forall e : \widehat{\text{estr}}$ )

$\text{Rep}(e) \equiv$

1. Para todo hijoDer de un  $\text{estr}$ , si no es NULL, su clave es mayor a la clave de su padre.
2. Para todo hijoIzq de un  $\text{estr}$ , 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

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

( $\forall d : \text{diccNat}(\kappa, \sigma)$ )

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

$((\forall n : \text{estr}(\kappa, \sigma))(n \in \text{arbol}(d) \Rightarrow obtener(c, n.\text{clave}) =_{\text{obs}} n.\text{significado})$

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

$\text{arbol}(n) \equiv$

```

if  $n.\text{hijoIzq} \neq \text{null} \wedge n.\text{hijoDer} \neq \text{null}$  then
   $\text{Ag}(n, \text{arbol}(n.\text{hijoIzq}) \cup \text{arbol}(n.\text{hijoDer}))$ 
else
  if  $n.\text{hijoIzq} \neq \text{null}$  then
     $\text{Ag}(n, \text{arbol}(n.\text{hijoIzq}))$ 
  else
    if  $n.\text{hijoDer} \neq \text{null}$  then
       $\text{Ag}(n, \text{arbol}(n.\text{hijoDer}))$ 
    else

```

```

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

```

### 4.3 Algoritmos

```

iVACIO()  $\rightarrow$  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 >
    end if
    if d  $\neq$  NULL  $\wedge$  n < d.clave then
        d.hijoIzq  $\leftarrow$  iDefinir(d.hijoIzq, n, s)
    end if
    if d  $\neq$  NULL  $\wedge$  n > d.clave then
        d.hijoDer  $\leftarrow$  iDefinir(d.hijoDer, n, s)
    end if

```

O(1)

O(log(m))

O(m)

---

O(m)

```

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

```

O(1)

O(m)

O(m)

O(1)

O(1)

O(1)

O(m)

O(1)

O(1)

O(1)

O(1)

O(m)

O(1)

O(1)

O(1)

---

O(m)

<b>ISIGNIFICADO</b> ( <b>in/out</b> $d : \mathbf{estr}$ , $in\ n : \kappa$ ) $\longrightarrow res : \sigma$	
$nodoActual \leftarrow d$	$O(1)$
<b>while</b> $\neg(nodoActual == NULL) \wedge \neg res$ <b>do</b>	$O(m)$
<b>if</b> $nodoActual.clave == n$ <b>then</b>	
$res \leftarrow nodoActual.significado$	$O(1)$
<b>else</b>	
<b>if</b> $c < nodoActual.clave$ <b>then</b>	
$nodoActual \leftarrow nodoActual.hijoIzq$	$O(1)$
<b>else</b>	
$nodoActual \leftarrow nodoActual.hijoDer$	$O(1)$
<b>end if</b>	
<b>end if</b>	
<b>end while</b>	
	<hr/>
	$O(m)$
<b>IDEFINIDO?</b> ( <b>in/out</b> $d : \mathbf{estr}$ , $in\ n : \kappa$ ) $\longrightarrow res : \mathbf{bool}$	
$nodoActual \leftarrow d$	$O(1)$
$res \leftarrow FALSE$	$O(1)$
<b>while</b> $\neg(nodoActual == NULL) \wedge \neg res$ <b>do</b>	$O(m)$
<b>if</b> $nodoActual.clave == n$ <b>then</b>	
$res \leftarrow TRUE$	$O(1)$
<b>else</b>	
<b>if</b> $c < nodoActual.clave$ <b>then</b>	
$nodoActual \leftarrow nodoActual.hijoIzq$	$O(1)$
<b>else</b>	
$nodoActual \leftarrow nodoActual.hijoDer$	$O(1)$
<b>end if</b>	
<b>end if</b>	
<b>end while</b>	
	<hr/>
	$O(m)$

m: En peor caso es igual a la cantidad de elementos del arbol. En promedio es  $\log(\text{cantidad de elementos del arbol})$ .